# STATS 507 Data Analysis in Python

Lecture 10: Basics of pandas

### Pandas



Open-source library of data analysis tools

Low-level ops implemented in Cython (C+Python=Cython, often faster) Database-like structures, largely similar to those available in R Optimized for most common operations

E.g., vectorized operations, operations on rows of a table

**From the documentation:** pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

# Installing pandas





Anaconda:

conda install pandas

Using pip:

pip install pandas

From binary (not recommended):

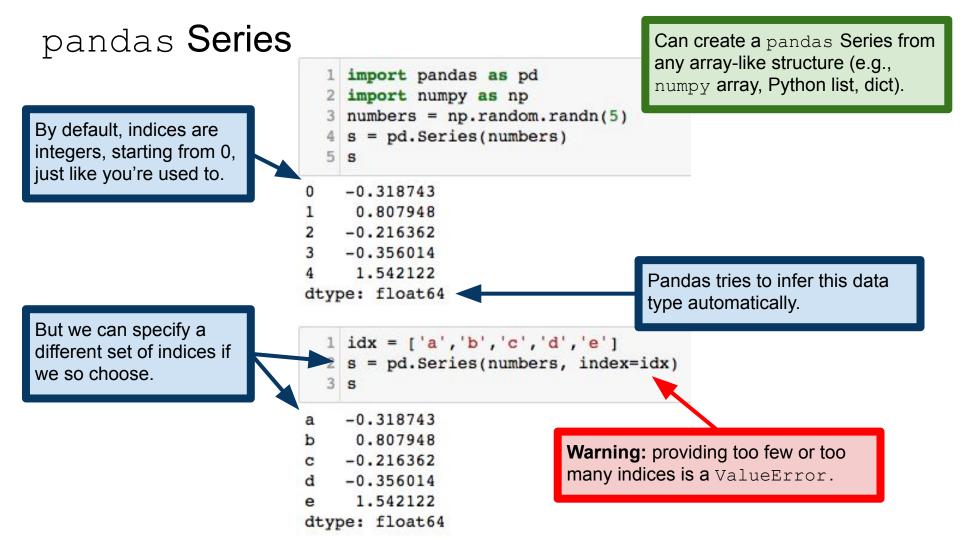
http://pypi.python.org/pypi/pandas

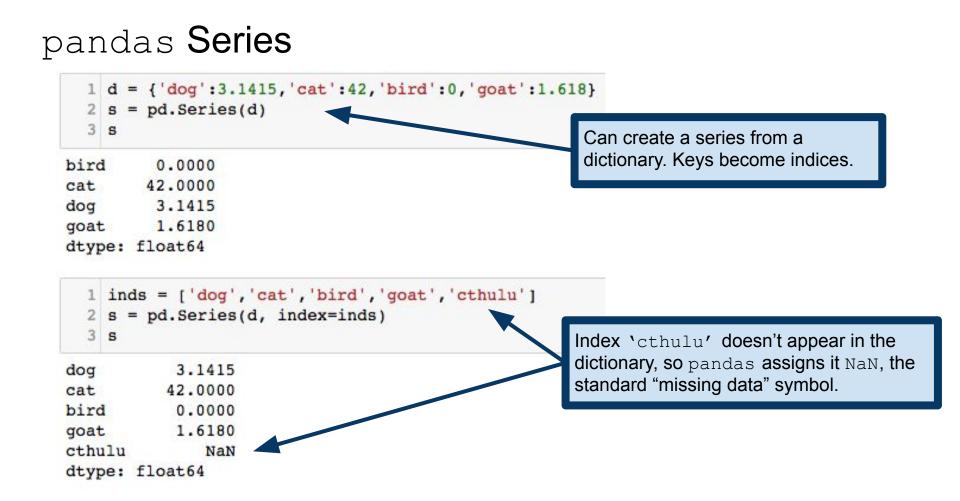
**Warning:** a few recent updates to pandas have been API-breaking changes, meaning they changed one or more functions (e.g., changed the number of arguments, their default values, or other behaviors). This shouldn't be a problem for us, but you may as well check that you have the most recent version installed.

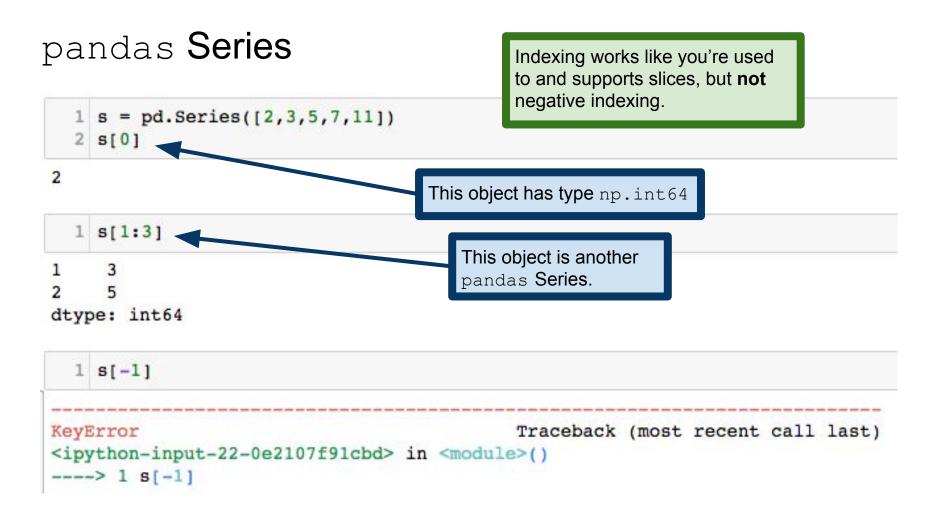
#### **Basic Data Structures**

Series: represents a one-dimensional **labeled** array Labeled just means that there is an index into the array Support vectorized operations

DataFrame: table of rows, with labeled columns Like a spreadsheet or an R data frame Support numpy ufuncs (provided data are numeric)



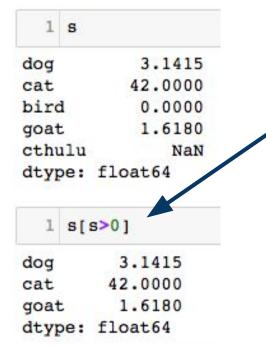




# pandas $\ensuremath{\mathsf{Series}}$

1	s = pd.Series([2,3])	3,5,7,11], index=['a','a','a','a','a'])
2	S	
a	2	
a	3	
a	5	Caution: indices need not be unique in pandas
a	7	Series. This will only cause an error if/when you
a	11	perform an operation that requires unique indices.
dty	pe: int64	
	s['a']	
1		
1 a		
1 a a	s['a'] 2 3	
a a a	s['a'] 2 3	

#### $\texttt{pandas} \; \textbf{Series}$



Series objects are like np.ndarray objects, so they support all the same kinds of slice operations, but note that the indices come along with the slices.

Series objects even support most numpy functions that act on arrays.

- 1		-	-	-
100	S	-	-	4
_	_			_

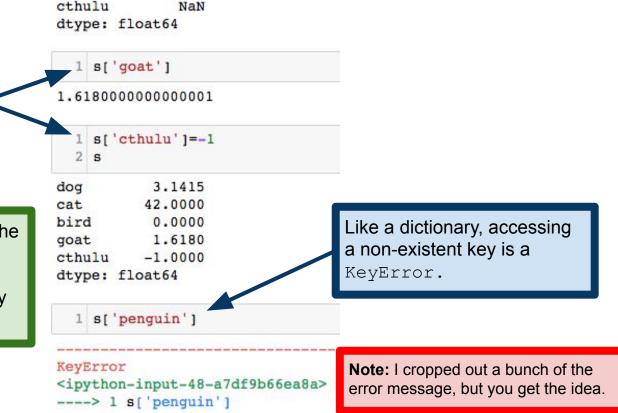
dog	9.869022
cat	1764.000000
bird	0.000000
goat	2.617924
cthulu	NaN
dtype:	float64

# $\texttt{pandas} \; \textbf{Series}$

1 s		
dog	3.1415	
cat	42.0000	
bird	0.0000	
goat	1.6180	
cthulu	NaN	
dtype:	float64	

Series objects are dict-like, in that we can access and update entries via their keys.

Not shown: Series also support the in operator: x in s checks if x appears as an index of Series s. Series also supports the dictionary get method.



# pandas **Series**

Entries of a Series can be of (almost) any type, and they may be mixed (e.g., some floats, some ints, some strings, etc), but they **can not** be sequences.

More information on indexing: https://pandas.pydata.org/pandas-d ocs/stable/indexing.html

#### 1 s dog 3.1415 cat 42.0000 bird 0.0000 goat 1.6180 cthulu -1.0000 dtype: float64

```
ValueError
<ipython-input-50-47579d9278ca>
----> 1 s['cthulu'] = (1,1)
```

/Use	ers/keith/anacon	da/lib/python2.7/site-packages/pandas
	744	# GH 6043
	745	<pre>elif _is_scalar_indexer(indexer):</pre>
>	746	<pre>values[indexer] = value</pre>
	747	
	748	# if we are an exact match (ex-broad
		and a statement of the statement of the statement of the

ValueError: setting an array element with a sequence.

#### $\texttt{pandas} \; \textbf{Series}$

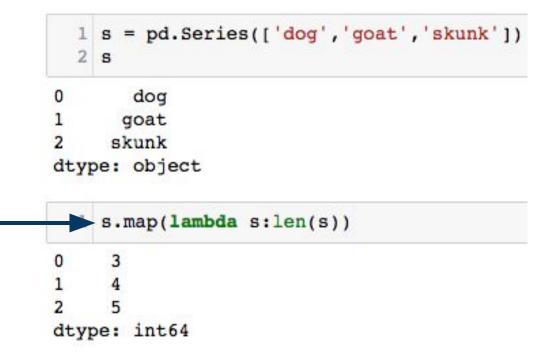
1 <b>s</b>			
cat bird	42 function 0 entries	s support universal ons, so long as all their s support operations.	1 d = {'dog':2,'cat':1.23456} 2 t = pd.Series(d) 3 t
goat 1.618 cthulu abcde dtype: object		Series operations require	cat 1.23456 dog 2.00000 dtype: float64
1 s + 2*s		that keys be shared. Missing values become NaN by default.	s+t
dog	9.4245	Nan by doldali.	bird NaN
cat bird	126		cat 43.2346
	0		cthulu NaN
goat	4.854		dog 5.1415
	bcdeabcdeabcde		goat NaN
dtype: obje	Ct		dtype: object

To reiterate, Series objects support most numpy ufuncs. For example, np.sqrt(s) is valid, so long as all entries are positive.

pandas <b>Series</b>	1 s bird 0.0000 cat 42.0000 dog 3.1415 goat 1.6180 dtype: float64	
Series have an optional name attribute.	1 s.name = 'aminals' 2 s	
After it is set, name attribute can be changed with rename method.	bird 0.0000 cat 42.0000 dog 3.1415 goat 1.6180 Name: aminals, dtype: float64	
Note: this returns a new Series. It does not change s.name.	bird 0.0000 cat 42.0000	This will become especially useful when we start talking about DataFrames, because these name attributes will be column names.

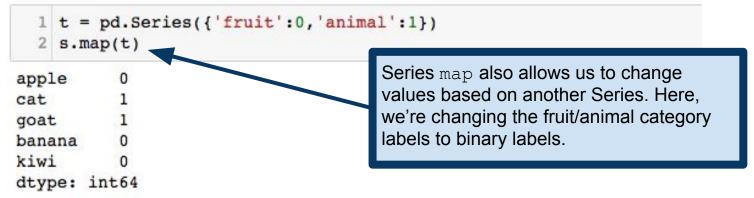
## Mapping and linking Series values

Series map method works analogously to Python's map function. Takes a function and applies it to every entry.



# Mapping and linking Series values

1 2	S	= pd.Series	<pre>(['fruit', 'animal', 'animal', 'fruit', 'fruit'], index=['apple','cat', 'goat', 'banana', 'kiwi'])</pre>
3	s		
appl	e	fruit	
cat		animal	
goat	1	animal	
bana	ina	fruit	
kiwi	8	fruit	
dtyp	e:	object	



# pandas DataFrames

Fundamental unit of pandas Analogous to R data frame

2-dimensional structure (i.e., rows and columns) Columns, of potentially different types Think: spreadsheet (or, better, database, but we haven't learned those, yet)

Can be created from many different objects Dict of {ndarrays, Python lists, dicts, Series} 2-dimensional ndarray Series

#### pandas DataFrames

Creating a DataFrame from a dictionary, the keys become the column names. Values become the columns of the dictionary.



#### A B

1.0	3.1400
2.0	2.7180
NaN	0.5772
	2.0

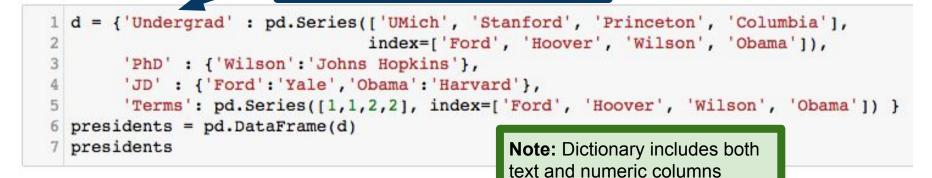
Indices that are unspecified for a given column receive NaN.

Each column may have its own indices, but the resulting DataFrame will have a row for every index (i.e., every row name) that appears.

**Note:** in the code above, we specified the two columns differently. One was specified as a Series object, and the other as a dictionary. This is just to make the point that there is flexibility in how you construct your DataFrame. More options: <u>https://pandas.pydata.org/pandas-docs/stable/generated/pandas.</u> DataFrame.html

#### pandas DataFrames: creating DataFrames

Dictionary has 4 keys, so 4 columns.



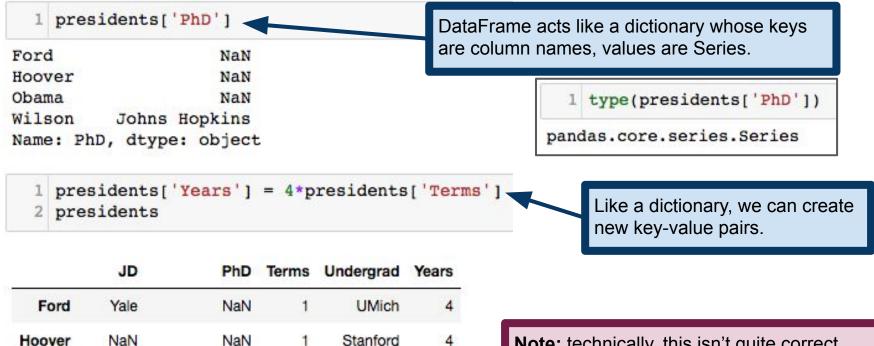
	JD	PhD	Terms	Undergrad
Ford	Yale	NaN	1	UMich
Hoover	NaN	NaN	1	Stanford
Obama	Harvard	NaN	2	Columbia
Wilson	NaN	Johns Hopkins	2	Princeton

By default, rows and columns are ordered alphabetically.

#### pandas DataFrames: row/column names

JD	PhD	Terms	Undergrad	
Yale	NaN	1	UMich	
NaN	NaN	1	Stanford	
Harvard	NaN	2	Columbia	
NaN	NaN Johns Hopkins		Princeton	Row and column names accessible as the index and column attributes, respectively, of the DataFrame.
esident	s.columns			
[u'JD',	u'PhD', u'	Terms	', u'Underg	rad'], dtype='object')
and dand	.index		>	Both are returned as pandas Index objects
	Yale NaN Harvard NaN esident	Yale NaN NaN NaN Harvard NaN NaN Johns Hopkins esidents.columns [u'JD', u'PhD', u'	Yale NaN 1 NaN NaN 1 Harvard NaN 2 NaN Johns Hopkins 2 esidents.columns [u'JD', u'PhD', u'Terms	YaleNaN1UMichNaNNaN1StanfordHarvardNaN2ColumbiaNaNJohns Hopkins2Princetonesidents.columns[u'JD', u'PhD', u'Terms', u'Undergo

#### pandas DataFrames: accessing/adding columns



8

8

Obama Harvard

NaN

Wilson

NaN

Johns Hopkins

2

2

Columbia

Princeton

**Note:** technically, this isn't quite correct, because Ford did not serve a full term. https://en.wikipedia.org/wiki/Gerald\_Ford

#### pandas DataFrames: accessing/adding columns

œ	JD	PhD	Terms	Undergrad	Years
Ford	Yale	NaN	1	UMich	4
Hoover	NaN	NaN	1	Stanford	4
Obama	Harvard	NaN	2	Columbia	8
Wilson	NaN	Johns Hopkins	2	Princeton	8

Since the row labels are ordered, we can specify a new column directly from a Python list, numpy array, etc. without having to specify indices.

	JD	PhD	Terms	Undergrad	Years	Nobels
Ford	Yale	NaN	1	UMich	4	0
Hoover	NaN	NaN	1	Stanford	4	0
Obama	Harvard	NaN	2	Columbia	8	1
Wilson	NaN	Johns Hopkins	2	Princeton	8	1

Note: by default, new column are inserted at the end. See the insert method to change this behavior: <u>https://pandas.pydata.org/pandas-d</u> <u>ocs/stable/generated/pandas.DataFr</u> <u>ame.insert.html</u>

## pandas DataFrames: accessing/adding columns

12	JD	PhD	Terms	Undergrad	Nobels	Years
Ford	Yale	NaN	1	UMich	0	4
Hoover	NaN	NaN	1	Stanford	0	4
Obama	Harvard	NaN	2	Columbia	1	8
Wilson	NaN	Johns Hopkins	2	Princeton	1	8

Scalars are broadcast across the rows.

	JD	PhD	Terms	Undergrad	Nobels	Years	Fields Medals
Ford	Yale	NaN	1	UM <mark>i</mark> ch	0	4	0
Hoover	NaN	NaN	1	Stanford	0	4	0
Obama	Harvard	NaN	2	Columbia	1	8	0
Wilson	NaN	Johns Hopkins	2	Princeton	1	8	0

# **Deleting columns**

Ford	Yale	NaN	1	UMich	0	4	0
Hoover	NaN	NaN	1	Stanford	0	4	0
Obama	Harvard	NaN	2	Columbia	1	8	0
Wilson	NaN	Johns Hopkins	2	Princeton	1	8	0

Undergrad Nobels Years Fields Medals

Terms

PhD

1 del presidents['Years']
2 presidents

JD

Delete columns identically to deleting keys from a dictionary. One can use the del keyword, or pop a key.

	JD	PhD	Terms	Undergrad	Nobels	Fields Medals
Ford	Yale	NaN	1	UMich	0	0
Hoover	NaN	NaN	1	Stanford	0	0
Obama	Harvard	NaN	2	Columbia	1	0
Wilson	NaN	Johns Hopkins	2	Princeton	1	0

	JD	ng and		Undergrad		1 presidents['JD']
Ford	Yale	NaN	1	UMich	0	Ford Yale
Hoover	NaN	NaN	1	Stanford	0	Hoover NaN Obama Harvard
Obama	Harvard	NaN	2	Columbia	1	Wilson NaN
Wilson	NaN	Johns Hopkins	2	Princeton	1	Name: JD, dtype: object
1 pre	sident	ts.loc['Obam	na']			1 presidents[1:3]
		Harvard				
hD		NaN				log selects rows by their labels
hD erms	78	NaN 2				.loc selects rows by their labels.
JD PhD Cerms Jndergra	78	NaN			$\mathbf{>}$	. loc selects rows by their labels. . iloc selects rows by their integer 1 Stanford
PhD Ferms Indergra Nob <mark>els</mark>	ad	NaN 2	ct		>	.loc selects rows by their labels.
PhD Terms Jndergra Nobels Name: Of	ad bama,	NaN 2 Columbia 1	ict			. loc selects rows by their labels. . iloc selects rows by their integer 1 Stanford
PhD Perms Undergra Nobels Name: Of	ad bama,	NaN 2 Columbia 1 dtype: obje	ict			. loc selects rows by their labels. . iloc selects rows by their integer els (starting from 0). 1 Stanford 2 Columbia
PhD Perms Indergra Iobels Iame: Ol 1 pre	ad bama,	NaN 2 Columbia 1 dtype: obje	ict			<pre>.loc selects rows by their labels. .iloc selects rows by their integer els (starting from 0).</pre>
hD erms ndergra obels ame: Ol 1 pre D hD erms	ad bama, sident	NaN 2 Columbia 1 dtype: obje ts.iloc[1] NaN NaN 1	ict			. loc selects rows by their labels. . iloc selects rows by their integer els (starting from 0). 1 Stanford 2 Columbia 1 presidents[presidents['Terms']<2 JD PhD Terms Undergrad Nobels
PhD Perms Indergra Iobels Name: Of 1 pre	ad bama, sident	NaN 2 Columbia 1 dtype: obje ts.iloc[1]	ct			<pre>.loc selects rows by their labels. .iloc selects rows by their integer els (starting from 0).</pre>

# Indexing and selection

	JD	F	hD	Terms	Undergrad	Nobels
Ford	Yale	١	NaN	1	UMich	0
Hoover	NaN	٩	VaN	1	Stanford	0
Obama	Harvard	Ν	VaN	2	Columbia	1
Wilson	NaN	Johns Hop	kins	2	Princeton	1
1 pr	esident	Select	: co	lumns	by their n	ames.
JD		Harvard	l)			
PhD		NaN	I			
Terms		2				
Underg	rad	Columbia	0			
Nobels		1	3			
Name: (	Obama,	dtype: c	bje	ect		
1 pr	esident	s.iloc[]	]			
JD		NaN	1			
PhD		NaN	I			
Terms		1				
Undergi	rad	Stanford	L			
Nobels		0	1			
Name: H	Hoover,	dtype:	obj	ect		

Ford Yale Hoover NaN Obama Harvard Wilson NaN Name: JD, dtype: object 1 presidents[1:3] JD PhD Terms Undergrad Nobels	1	pr	eside	nt	:s[']	נ' <b>ת</b>		
Obama Harvard Wilson NaN Name: JD, dtype: object 1 presidents[1:3]	Ford	1			Yale	,		
Wilson NaN Name: JD, dtype: object 1 presidents[1:3]	HOON	ver			NaN	I		
Name: JD, dtype: object 1 presidents[1:3]	Oban	na	н	ar	vard	1		
1 presidents[1:3]	Wils	on			NaN	I		
	Mama							
	Name	:	JD, d	ty	pe:	object	5	
	1	pr	eside J	nt D	s[1: PhD	:3]	Undergrad	
Hoover NaN NaN 1 Stanford 0	1	pr	eside J	nt D	s[1:	:3]		

1 presidents[presidents['Terms']<2]</pre>

	JD	PhD	Terms	Undergrad	Nobels
Ford	Yale	NaN	1	UMich	0
Hoover	NaN	NaN	1	Stanford	0

<u>.</u>	JD					1	presid	ents	'JD']		
Ford	Yale	NaN	1	UMich	0	Ford		Ya	le		
Hoover	NaN	NaN	1	Stanford	0	Hoov		N Harva	aN		
Obama Ha	arvard	NaN	2	Columbia	1	Wils	0.000		aN		
Wilson	NaN Joh	ins Hopkins	2	Princeton	1	Name	: JD, d	itype	: obje	ect	
1 pres	idents		2011			1	preside	ents (	1:3]		
ID PhD Terms	i	Select rows ndices (ag supports sl	jain 0	-indexed)					nD Terr	-	
Indergrad	d Co.	umpra				Hoo	ver N	a <mark>N N</mark>	aN	1 Stanfo	ord
Nobels	ana, u	Note: one with lists of	_			Oba	ma Harva	ard N	aN	2 Colum	oia
Name: Oba					• • •	1					
		presiden	its[[	_'JD','P	'nD′]].	1	preside	ents	presi	dents['Te	rms']<2
1 pres		presiden NaN	its[[	_ 'JD','P	'nD′]].	1	presid	ents	presi	dents['Te	erms']<2
1 pres			its[[	_ 'JD','P	'nD′]].	1	JD	PhD	presi Terms	Undergrad	
	idents <sup>I</sup>	NaN	its[[	_ 'JD','P	'nD′]].		JD				

# Indexing and selection

<i>2</i>	JD	PhD	Terms	Undergrad	Nobels
Ford	Yale	NaN	1	UMich	0
Hoover	NaN	NaN	1	Stanford	0
Obama	Harvard	NaN	2	Columbia	1
Wilson	NaN	Johns Hopkins	2	Princeton	1

1 presid	ents.loc['Obama'	1
JD	Harvard	
PhD	NaN	
Terms	2	
Undergrad	Columbia	
Nobels	1	
Name Oham	dtunge object	
Name: Obam	a, dtype: object	
	ents.iloc[1]	Select columns by
1 presid	ents.iloc[1]	Select columns by
1 presid	ents.iloc[1] NaN	Select columns by
1 presid JD PhD	ents.iloc[1] NaN	Select columns by

Name: Hoover, dtype: object

1	presi	dents['JD']	
Ford	L	Yale	
HOOV	er	NaN	
Oban	ia	Harvard	
Wils	on	NaN	
Name	: JD,	dtype: object	

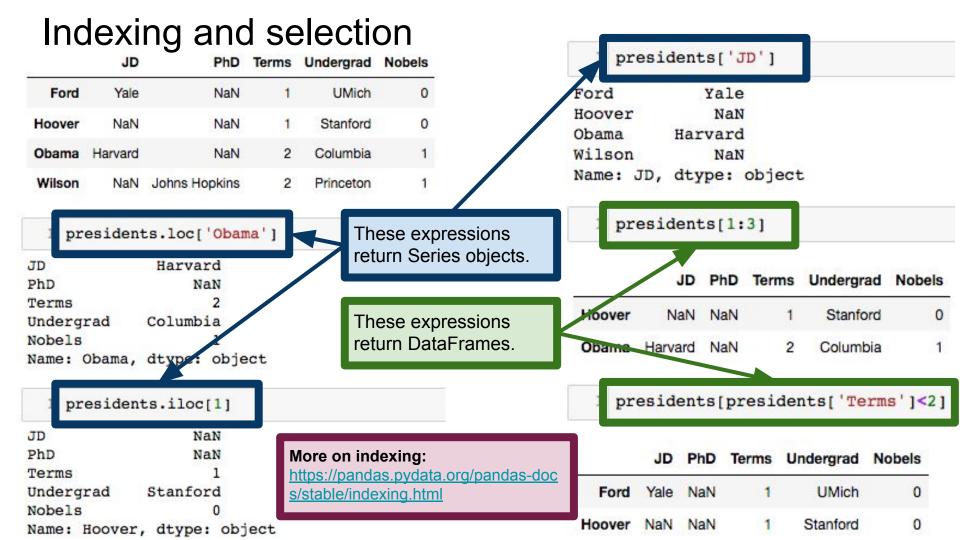
1 presidents[1:3]

	JD	PhD	Terms	Undergrad	Nobels
Hoover	NaN	NaN	1	Stanford	0
Obama	Harvard	NaN	2	Columbia	1

presidents[presidents['Terms']<2]</pre>

	JD	PhD	Terms	Undergrad	Nobels
Ford	Yale	NaN	1	UMich	0
Hoover	NaN	NaN	1	Stanford	0

	JD	•	Terms	electic Undergrad			presi	dent	s['J	ניס		
Ford	Yale	NaN	1	UMich	0	Ford			Yale			
Hoover	NaN	NaN	1	Stanford	0	Hoov		Uar	NaN vard			
Obama	Harvard	NaN	2	Columbia	1	Wils		nai	NaN			
Wilson	NaN	Johns Hopkins	2	Princeton	1	Name	: JD,	dty	pe:	object	t	
nr	esident	s.loc['Obam	a'1	The	ese expressions	1	presi	dent	s[1:	3]		
0	obidom	Harvard	<b>"</b> ]		urn Series objects	S.					are at the	
nD		NaN						JD	PhD	Terms	Undergrad	Nobe
erms		2				Hee						
dona	rad	Columbia				Hoo	/er	NaN	NaN	1	Stanford	
-	rad	Columbia					1022			1		
bels		Columbia dtype: objec	ct			Oba	1022	NaN vard	NaN NaN	1	Stanford	
bels me: (	Obama,	1	ct				ma Ha	vard	NaN			
bels me: (	Obama,	dtype: objec	ct				ma Ha	vard	NaN		Columbia	
bels me: 0 pr	Obama,	dtype: objects.iloc[1]	ct				ma Ha	vard dent	NaN s[pr	eside	Columbia	ns']<
bels me: 0 1 pr D rms	Obama, esident	dtype: objects.iloc[1] NaN NaN 1	ct			Oba 1	ma Ha presi JI	vard dent D Ph	NaN s[pr D Te	eside mms U	Columbia nts['Tern ndergrad N	ns']< lobels
	Obama, esident	dtype: objects.iloc[1] NaN	ct			Oba 1	ma Ha	vard dent D Ph	NaN s[pr D Te	eside	Columbia nts['Terr	ns']•



1 dfl = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
2 df2 = pd.DataFrame(np.random.randn(5, 3), columns=['A', 'B', 'C'])

	A	в	С	D
0	0.722814	-1.889204	-1.170304	NaN
1	1.370720	-1.033425	-0.719628	NaN
2	-2.281526	0.899515	-0.298246	NaN
3	-4.276271	-2.327304	-0.444528	NaN
4	-1.418512	0.463528	0.428446	NaN
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN

3 df1+df2

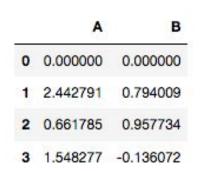
pandas tries to align the DataFrames as best it can, filling in non-alignable entries with NaN.

In this example, rows 0 through 4 and columns A through C exist in both DataFrames, so these entries can be successfully added. All other entries get NaN, because x + NaN = NaN.

1 df = pd.DataFrame(np.random.randn(4, 2), columns=['A', 'B'])
2 df

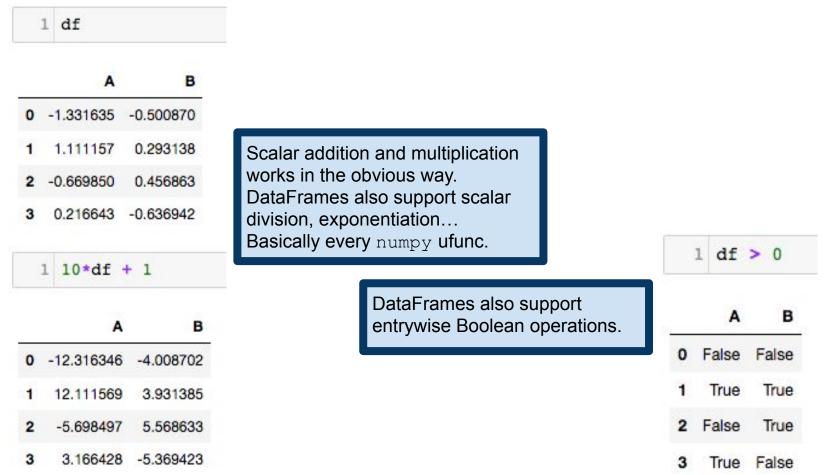
B	Α	
-0.500870	-1.331635	0
0.293138	1.111157	1
0.456863	-0.669850	2
-0.636942	0.216643	3

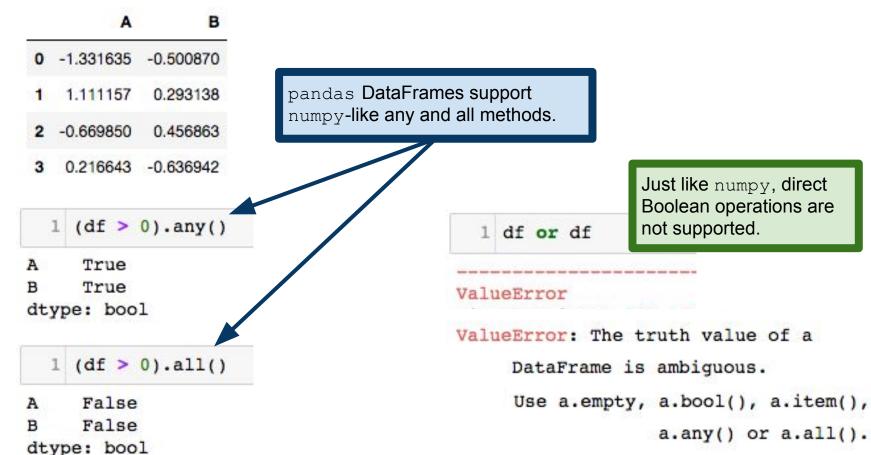
1 df - df.iloc[0] -

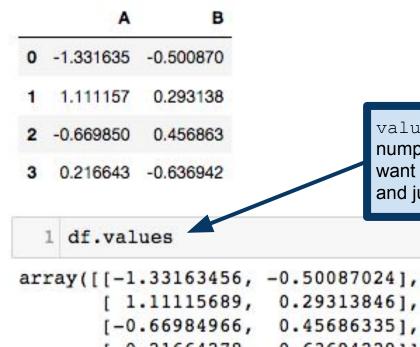


By default, Series are aligned to DataFrames via row-wise broadcasting.

df.iloc[0] is a Series representing the 0-th row of df. When we try to subtract it from df, pandas forces dimensions to agree by broadcasting the operation across all rows of df.







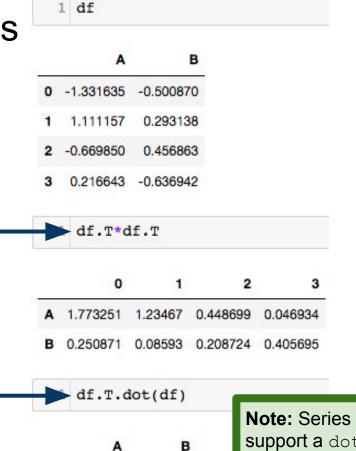
values attribute stores the entries of the table in a numpy array. This is occasionally useful when you want to stop dragging the extra information around and just work with the numbers in the table.

```
1.11115689, 0.293138461,
[-0.66984966, 0.45686335],
 0.21664278, -0.6369422911)
```

DataFrames support entrywise

transpose of the DataFrame.

multiplication. The T attribute is the



3.503553 0.548680

0.548680

0.951221

DataFrames also support matrix multiplication via the numpy-like dot method. The DataFrame dimensions must be conformal, of course.

Note: Series also support a dot method, so you can compute inner products.

### **Removing NaNs**

	А	В	С	D
0	-9.422331	1.100197	8.034010	NaN
1	-1.520140	5.655382	-1. <mark>692761</mark>	NaN
2	0.399654	10.058568	0.502007	NaN
3	-4.070947	2.237868	10.530079	NaN
4	1.603739	8.255591	1.892258	NaN
5	1.123450	3.141590	NaN	NaN

DataFrame dropna method removes rows or columns that contain NaNs.

axis argument controls whether we act on rows, columns, etc.

how='any' will remove all rows/columns that contain even one NaN. how='all' removes rows/columns that have all entries NaN.

	A	В	
0	-9.422331	1.100197	
1	-1.520140	5.655382	
2	0. <mark>39965</mark> 4	10.058568	
3	-4.070947	2.237868	
4	1.603739	8.255591	
÷	1.123450	3.141590	

df.dropna(axis=1, how='any')

	A	В	С
0	- <mark>9.42233</mark> 1	1.100197	8.034010
1	-1.520140	5.655382	-1.692761
2	0.399654	10.058568	0.502007
3	-4.070947	2.237868	10.530079
4	1.603739	8.255591	1.892258
5	1.123450	3.141590	NaN

# Reading/writing files

pandas supports read/write for a wide range of different file formats. This flexibility is a major advantage of pandas.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

https://pandas.pydata.org/pandas-docs/stable/io.html

# Reading/writing files

pandas supports read/write for a wide range of different file formats. This flexibility is a major advantage of pandas.

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
hinary	MS Excel	read_excel	to_excel
andas <b>file l</b> /	O is largely similar to R read.table	read_hdf	to_hdf
nd similar fui	nctions, so I'll leave it to you to read the	read_feather	to_feather
andas <b>docu</b>	mentation as needed.	read_parquet	to_parquet
binary	мздраск	road magnack	
binary		read_msgpack	to_msgpack
binary	Stata	read_stata	to_msgpack to_stata
binary			
	Stata	read_stata	
SQL	Stata SAS	read_stata read_sas	to_stata
	Stata SAS Python Pickle Format	read_stata read_sas read_pickle	to_stata to_pickle

Table credit: https://pandas.pydata.org/pandas-docs/stable/io.html

### Summarizing DataFrames

pd.read\_csv() reads a comma-separated file into a DataFrame.

info() method prints summary data about the DataFrame. Number of rows, column names and their types, etc.

**Note:** there is a separate to\_string() method that generates a string representing the DataFrame in tabular form, but this usually doesn't display well if you have many columns. baseball = pd.read\_csv('baseball.csv')
baseball.info()
lass 'pandas.core.frame.DataFrame'>
t64Index: 21699 entries, 4 to 89534
ba columns (total 22 columns);

			es, 4 to 89534
Data co	lumns (t	otal 22 d	columns):
		non-null	
		non-null	
		non-null	
team	21699	non-null	object
lg	21634	non-null	object
g	21699	non-null	int64
	21699	non-null	int64
r	21699	non-null	int64
h	21699	non-null	int64
X2b	21699	non-null	int64
X3b	21699	non-null	int64
hr	21699	non-null	int64
rbi	21687	non-null	float64
sb	21449	non-null	float64
CS	17174	non-null	float64
bb	21699	non-null	int64
SO	20394	non-null	float64
ibb	14171	non-null	float64
hbp	21322	non-null	float64
sh	20739	non-null	float64
sf	14309	non-null	float64
		non-null	
dtypes:	float64	(9), inte	54(10), object(3)
memory w	isage: 3	8.8+ MB	

### Summarizing DataFrames

1 baseball.head()

head() method displays just the first few rows of the DataFrame (5 by default; change this by supplying an argument). tail() displays the last few rows.

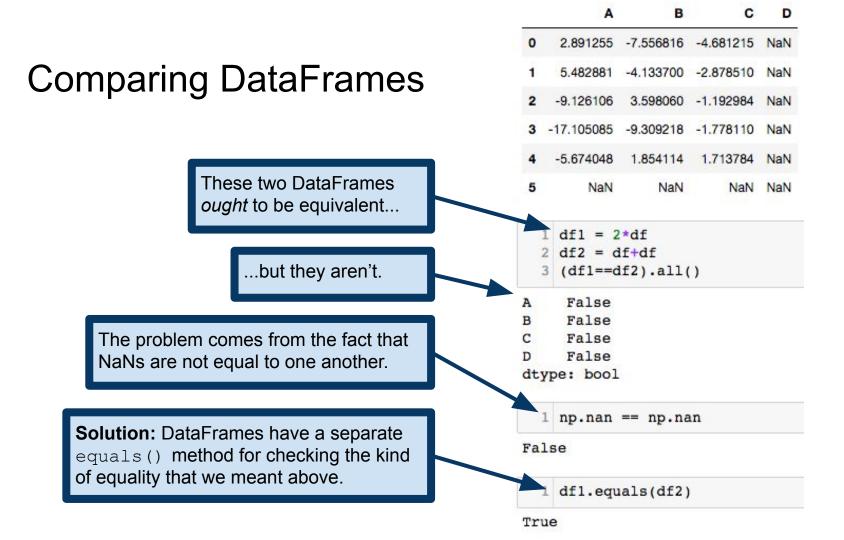
	id	year	stint	team	lg	g	ab	r	h	X2b	 rbi	sb	CS	bb	SO	ibb	hbp	sh	sf	gidp
4	ansonca01	1871	1	RC1	NaN	25	120	29	39	11	 16.0	6.0	2.0	2	1.0	NaN	NaN	NaN	NaN	NaN
44	forceda01	1871	1	WS3	NaN	32	162	45	45	9	 29.0	8.0	0.0	4	0.0	NaN	NaN	NaN	NaN	NaN
68	mathebo01	1871	1	FW1	NaN	19	89	15	24	3	 10.0	2.0	1.0	2	0.0	NaN	NaN	NaN	NaN	NaN
99	startjo01	1871	1	NY2	NaN	33	161	35	58	5	 34.0	4.0	2.0	3	0.0	NaN	NaN	NaN	NaN	NaN
102	suttoez01	1871	1	CL1	NaN	29	128	35	45	3	 23.0	3.0	1.0	1	0.0	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns

**Note:** R and pandas both supply head/tail functions, named after UNIX/Linux commands that displays the first/last lines of a file.

		A	В	С	D
	0	2.891255	-7.556816	-4.681215	NaN
ng DataFrames	1	5.482881	-4.133700	-2.878510	NaN
ng Dului Tumoo	2	-9.126106	3.598060	-1.192984	NaN
	3	-17.105085	-9.309218	-1.778110	NaN
	4	-5.674048	1.854114	1.713784	NaN
	5	NaN	NaN	NaN	NaN
but they aren't.	A B C D	False False False False False pe: bool	lf2).all(		
	ucy		== np.na	n	
	Fal				
	1	dfl.equ	als(df2)	)	
	Tru	e			

### Compari



### **Comparing DataFrames**

There is a solid design principle behind this. If there are NaNs in our data, we want to err on the side of being overly careful about what operations we perform on them. We see similar ideas in numpy and in R.

**Solution:** DataFrames have a separate equals () method for checking the kind of equality that we meant above.

	A	В	С	D
0	2.891255	-7.556816	-4.681215	NaN
1	5.482881	-4.1337 <mark>0</mark> 0	-2.878510	NaN
2	-9.126106	3.598060	-1.192984	NaN
3	-17.105085	-9.309218	-1.778110	NaN
4	-5.674048	1.854114	1.713784	NaN
5	NaN	NaN	NaN	NaN

1	df1 = 2*df	
2	df2 = df+df	
3	(df1==df2).all(	)

7	False
3	False
:	False
)	False

dtype: bool

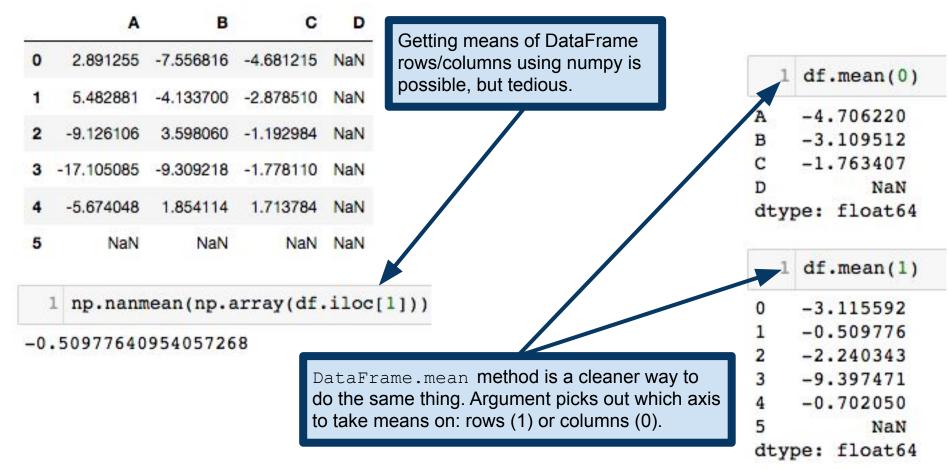
1 np.nan == np.nan

False

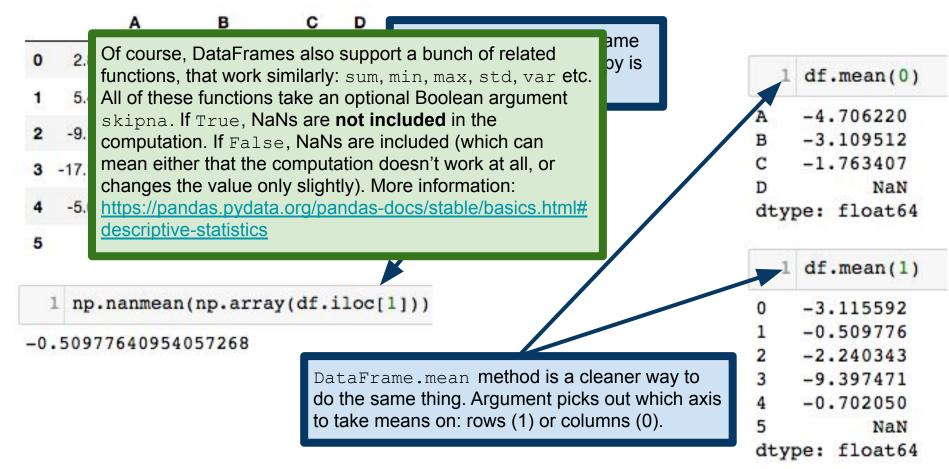
1 dfl.equals(df2)

True

### **Statistical Operations on DataFrames**



### Statistical Operations on DataFrames



## Summarizing DataFrames

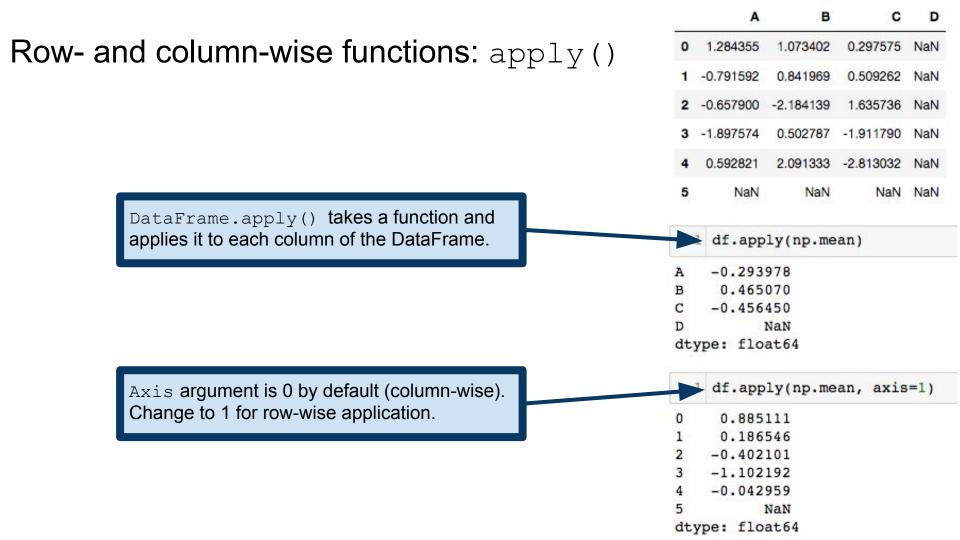
DataFrame.describe() is similar to the R summary() function. Non-numeric data will get statistics like counts, number of unique items, etc. If a DataFrame has mixed types (both numeric and non-numeric), the non-numeric data is excluded by default.

### Details and optional arguments: https://pandas.pydata.org/pandas-docs/stable/basics.ht ml#summarizing-data-describe

A	В	С	D
2.891255	-7.556816	-4.681215	NaN
5.482881	-4.133700	-2.878510	NaN
-9.126106	3.598060	-1.192984	NaN
-17.105085	-9.309218	-1.778110	NaN
-5.674048	1.854114	1.713784	NaN
NaN	NaN	NaN	NaN
	5.482881 -9.126106 -17.105085 -5.674048	5.482881       -4.133700         -9.126106       3.598060         -17.105085       -9.309218         -5.674048       1.854114	-9.1261063.598060-1.192984-17.105085-9.309218-1.778110-5.6740481.8541141.713784

#### df.describe()

	Α	В	С	D
count	5.000000	5.000000	5.000000	0.0
mean	-4.706220	-3.109512	-1.7 <mark>63</mark> 407	NaN
std	9.161650	5.676551	2.354438	NaN
min	-17.105085	-9.309218	-4. <mark>6</mark> 81215	NaN
25%	-9.126106	-7.556816	-2.878510	NaN
50%	-5.674048	-4.133700	-1.778110	NaN
75%	2.891255	1.854114	-1.192984	NaN
max	5.482881	3.598060	1.713784	NaN



		A	E	s (	D
<b>Row- and column-wise functions:</b> apply()	0	1.284355	1.073402	0.29757	5 NaN
	1	-0.791592	0.841969	0.50926	2 NaN
	2	-0.657900	-2.184139	1.63573	6 NaN
	3	-1.897574	0.502787	-1.91179	0 NaN
	4	0.592821	2.091333	3 -2.81303	2 NaN
Numpy ufuncs take vectors and spit out vectors, so using df.apply() to apply a	5	NaN	NaN	I Naf	N NaN
ufunc to every row or column in effect ends up applying the ufunc to every element.		▶ df.ap	ply(np.e	xp)	
		A	В	С	D
	0	3.612337	2.925314	1.346589	NaN
	1	0.453123	2.320931	1.664062	NaN
	2	0.517938	0.112575	5.133236	NaN
	3	0.149932	1.653323	0.147816	NaN
	4	1.809085	8.095701	0.060023	NaN
	5	NaN	NaN	NaN	NaN

	А	В	С
0	0.938898	2.047553	-0.525091
1	1.066293	-0.599466	-0.195606
2	-0.939341	0.022376	1.453082
3	1.114664	-0.408026	-0.811081
4	2.257680	0.280994	0.847329

### 1 def quadratic(x, a, b, c=1): 2 return a\*x\*\*2 + b\*x + c 3 df.apply(quadratic, args=(1,2), c=5)

We can pass positional and keyword arguments into the function via df.apply. Args is a tuple of the positional arguments (in order), followed by the keyword arguments.

	A	В	С
0	7.759325	13.287581	4.225538
1	8.269566	4.160428	4.647050
2	4.003679	5.045253	10.017612
3	8. <mark>4</mark> 71805	4.350433	4.035691
4	14.612481	5.640946	7.412624

**Note:** "apply() takes an argument raw which is False by default, which converts each row or column into a Series before applying the function. When set to True, the passed function will instead receive an ndarray object, which has positive performance implications if you do not need the indexing functionality." This can be useful if your function is meant to work specifically with Series.

### Row- and column-wise functions: apply()

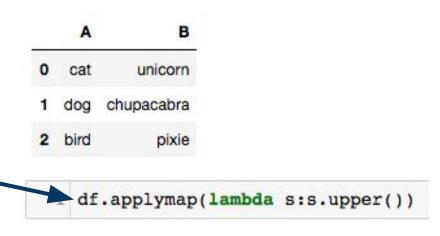
### Element-wise function application

This causes an error, because apply thinks that its argument should be applied to Series (i.e., columns), not to individual entries.

```
----> 1 df.apply(lambda s:s.upper())
```

### **Element-wise function application**

applymap works similarly to Python's map function (and the Series map method). Applies its argument function to every entry of the DataFrame.



	A	в
0	CAT	UNICORN
1	DOG	CHUPACABRA
2	BIRD	PIXIE

### **Tablewise Function Application**

Here we have a function composition applied to a DataFrame. This is perfectly valid code, but pandas supports another approach.

1	f = lambda x:x**2
2	g = lambda x:x+1
3	h = lambda x: 2*x
4	df = pd.DataFrame(np.random.randn(5, 3),
5	columns=['A', 'B', 'C'])
6	df

	A	В	C
0	-2.072339	-1.282539	-1.241128
1	-0.587874	0.517591	-0.394561
2	-0.1 <mark>64436</mark>	1.450398	-0.975424
3	-1.215576	-0.671235	0.394053
4	-0.350299	1.958805	0.467778

	A	В	С
0	10.589182	5.289812	5.080798
1	2.691193	2.535802	2.311357
2	2.054078	6.207308	3.902906
3	4.955251	2.901113	2.310556
4	2.245 <mark>41</mark> 9	9.673833	2.437633

### **Tablewise Function Application**

The DataFrame pipe method is built for a pattern called **method chaining**. The pipe method has better support for passing additional arguments around than does the function composition to the right. This pattern using pipe is also more conducive to functional programming patterns.

1 df.pipe(f).pipe(g).pipe(h)

	A	В	С
0	10.589182	5.289812	5.080798
1	2.691193	2.535802	2.311357
2	2.054078	6.207308	3.902906
3	4.955251	2.901113	2.310556
4	2.245419	9.673833	2.437633

1	f = lambda x: x**2
2	g = lambda x:x+1
3	h = lambda x: 2*x
4	df = pd.DataFrame(np.random.randn(5, 3),
5	columns=['A', 'B', 'C'])
6	df

	A	В	С
0	-2.072339	<mark>-1.282539</mark>	-1.241128
1	-0.587874	0.517591	-0.394561
2	-0.1 <mark>6443</mark> 6	1.450398	-0.975424
3	-1.215576	-0.671235	0.394053
4	-0.350299	1.958805	0.467778

#### 1 h(g(f(df)))

	A	В	C
0	10.589182	5.289812	5.080798
1	2.691193	2.535802	2.311357
2	2.054078	6.207308	3.902906
3	4.955251	2.901113	2.310556
4	2.245419	9.673833	2.437633