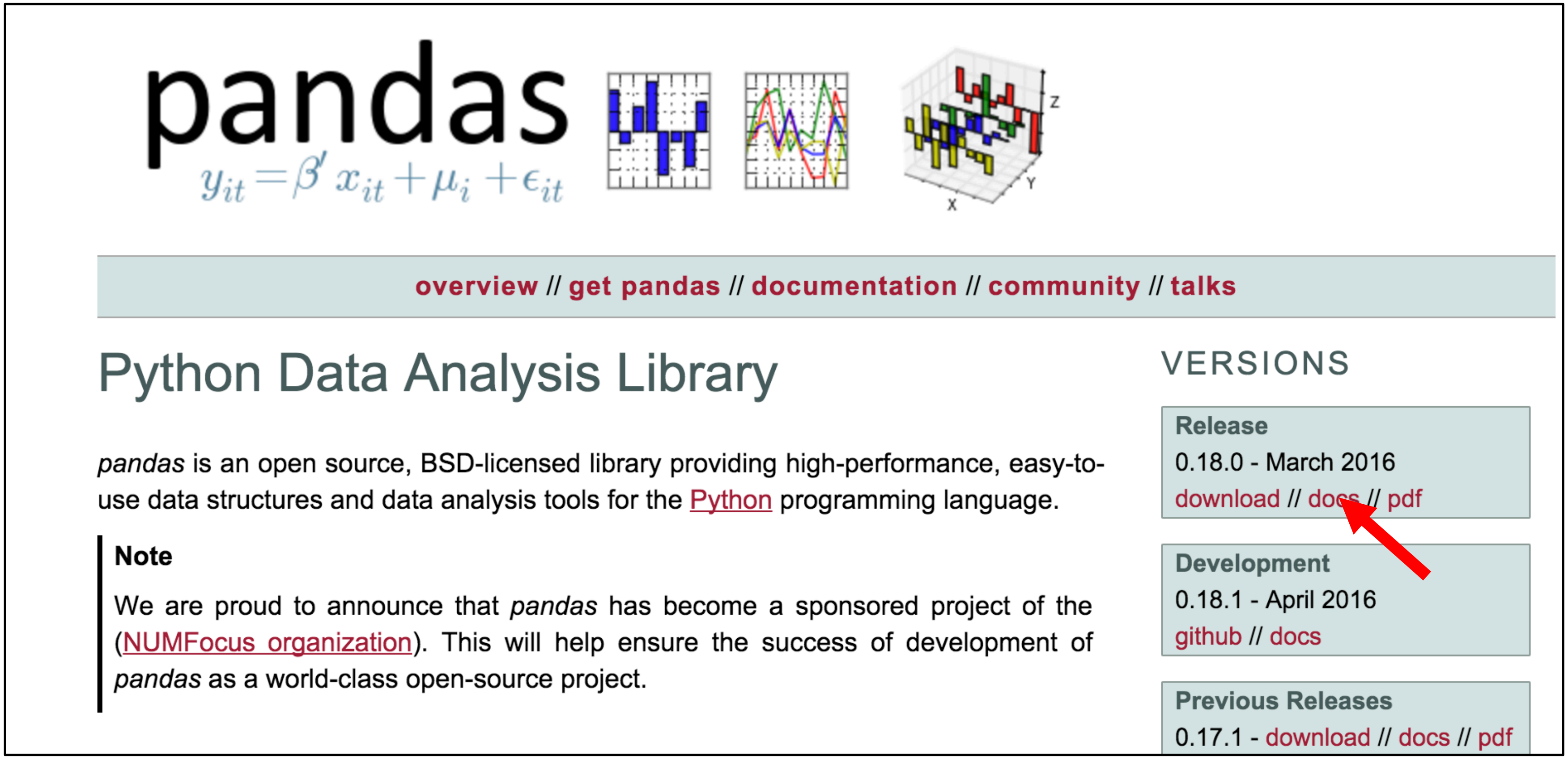
Advanced tabular data processing with pandas

Day 2

Pandas library

- Library for tabular data I/O and analysis
- Useful in stored scripts and in ipython notebooks



overview // get pandas // documentation // community // talks

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

> **Note**
>
> We are proud to announce that *pandas* has become a sponsored project of the (NUMFocus organization). This will help ensure the success of development of *pandas* as a world-class open-source project.

VERSIONS

Release
0.18.0 - March 2016
download // docs // pdf

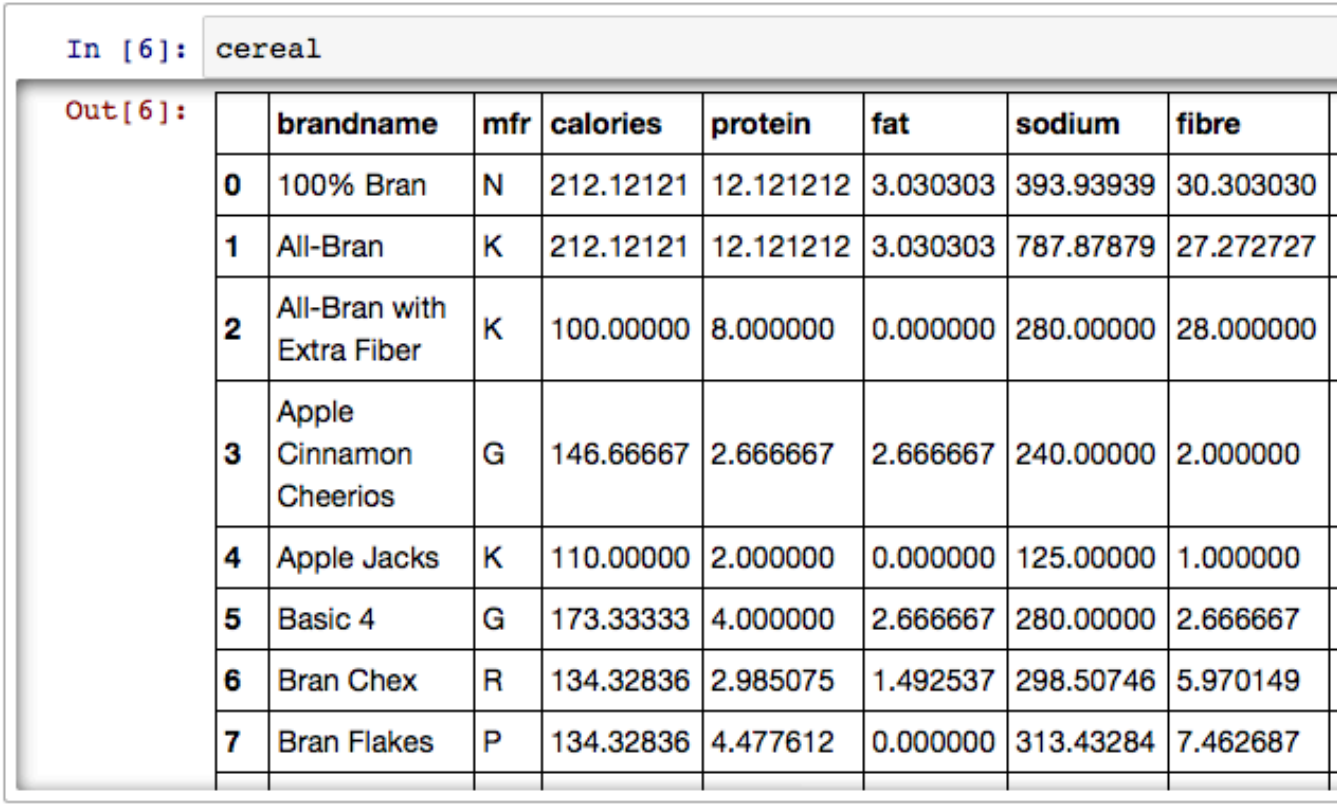
Development
0.18.1 - April 2016
github // docs

Previous Releases
0.17.1 - download // docs // pdf

http://pandas.pydata.org/

DataFrame

- Tables of 2D data = rows x columns
- Similar to "data.frame" in R
- Notebook provides "pretty print"



```
In [6]: cereal
```

Out[6]:

	brandname	mfr	calories	protein	fat	sodium	fibre
0	100% Bran	N	212.12121	12.121212	3.030303	393.93939	30.303030
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.272727
2	All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	28.000000
3	Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2.000000
4	Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1.000000
5	Basic 4	G	173.33333	4.000000	2.666667	280.00000	2.666667
6	Bran Chex	R	134.32836	2.985075	1.492537	298.50746	5.970149
7	Bran Flakes	P	134.32836	4.477612	0.000000	313.43284	7.462687

Read data frames from files

- Pandas can read data from various formats
- Most common in genomics:
- `pd.read_table` – read from comma or tab delimited file
  - http://pandas.pydata.org/pandas-docs/version/0.18.0/io.html#io-read-csv-table
  - Full docs here
- `pd.read_excel` – read from Excel spreadsheet
- http://pandas.pydata.org/pandas-docs/version/0.18.0/io.html#io-excel-reader
  - Full docs here

- Read in US Cereal stats table (source)
- What type of value does this return?

Write data frames to files

- Data can be written out in various formats too
- `df.to_csv` – write to tab/comma delimited
  - where df is a DataFrame value
  - http://pandas.pydata.org/pandas-docs/version/0.18.0/io.html#io-store-in-csv

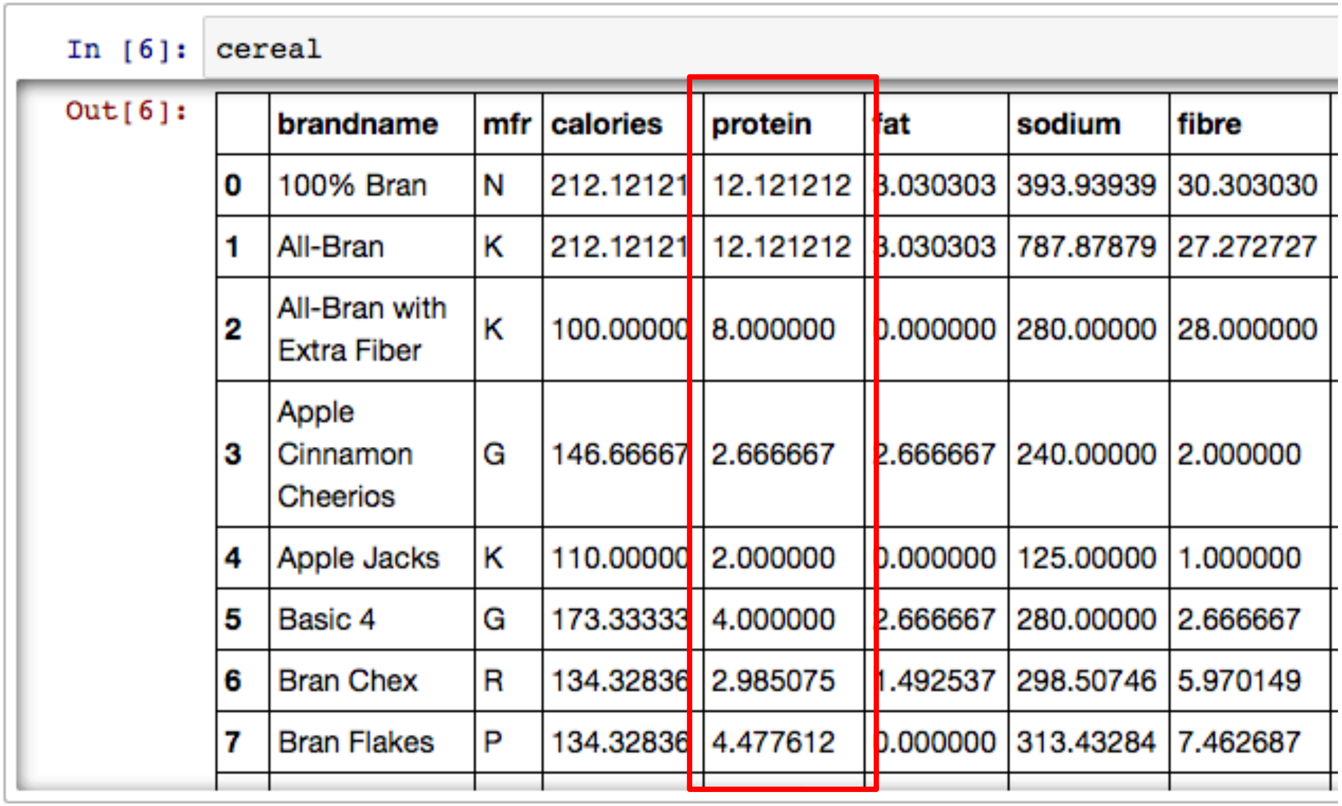
- Write US cereal stats back out to disk, using comma deliminters, to "cereals.csv".

Exploring tabular data

- `df.shape` – retrieve table dimensions as tuple
- `df.columns` – retrieve columns
  - To rename a column, set df.columns = [list of names]
- `df.dtypes` – retrieve data type of each column
- `df.head(n)` – retrieve first *n* rows
- `df.tail(n)` – retrieve last *n* rows
- `df.describe()` – retreive summary stats (for numerical columns)

Accessing by column

- To retrieve a single column, use `df[ 'protein' ]`
- Or `df[ my_col_name ]` (How do these differ?)
- This returns a 1D pandas "Series"



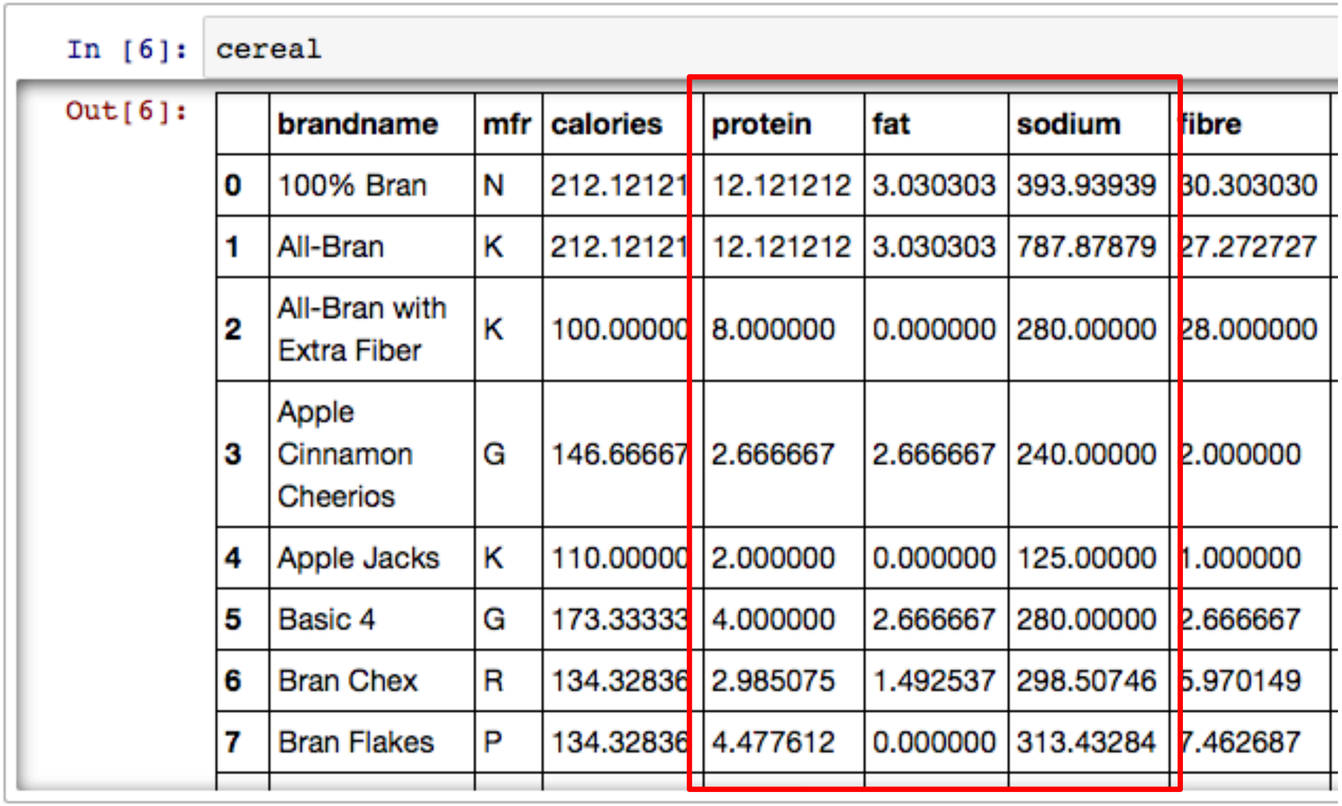
```
In [6]: cereal
```

Out[6]:

	brandname	mfr	calories	protein	fat	sodium	fibre
0	100% Bran	N	212.12121	12.121212	3.030303	393.93939	30.303030
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.272727
2	All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	28.000000
3	Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2.000000
4	Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1.000000
5	Basic 4	G	173.33333	4.000000	2.666667	280.00000	2.666667
6	Bran Chex	R	134.32836	2.985075	1.492537	298.50746	5.970149
7	Bran Flakes	P	134.32836	4.477612	0.000000	313.43284	7.462687

Accessing multiple columns

- Similar syntax, but provide a list or tuple of column names, e.g., `df[ ['protein','fat','sodium'] ]`



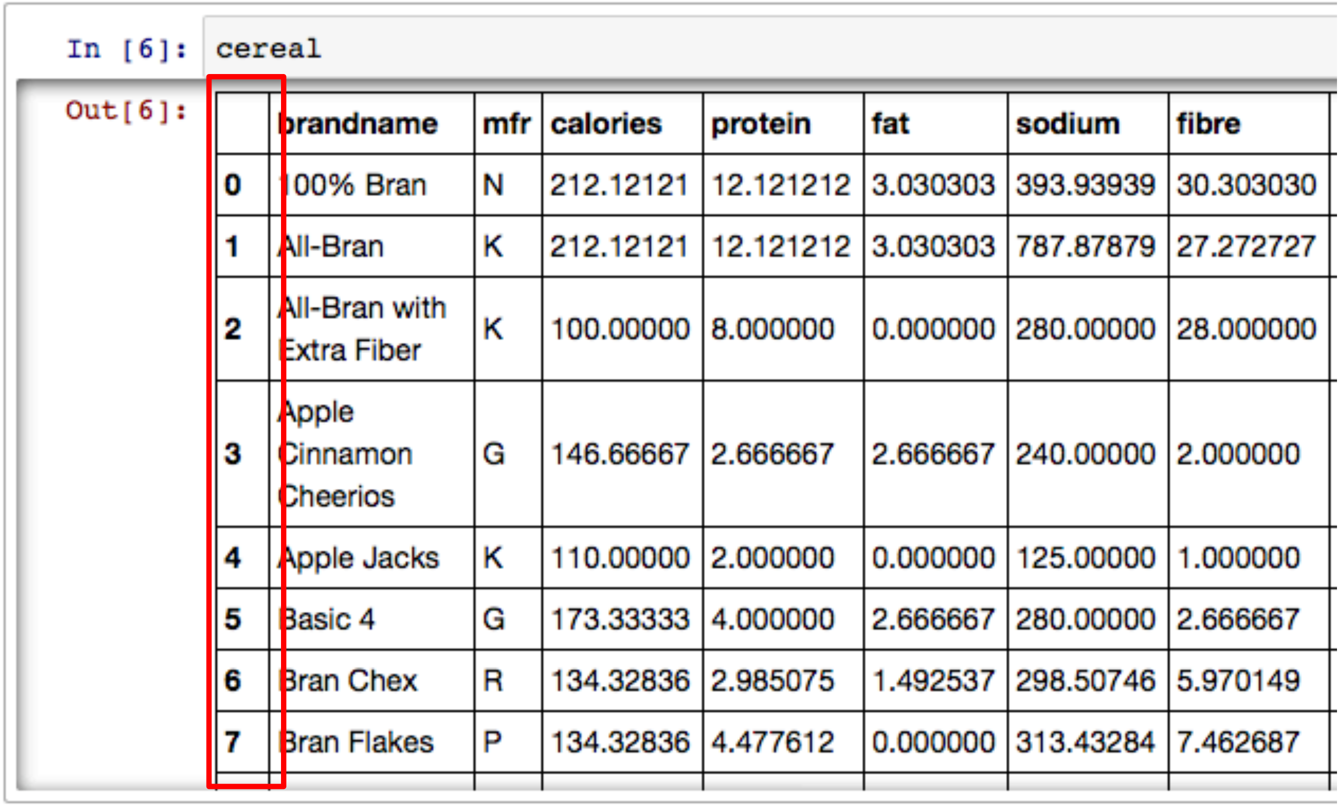
```
In [6]: cereal
```

Out[6]:

	brandname	mfr	calories	protein	fat	sodium	fibre
0	100% Bran	N	212.12121	12.121212	3.030303	393.93939	30.303030
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.272727
2	All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	28.000000
3	Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2.000000
4	Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1.000000
5	Basic 4	G	173.33333	4.000000	2.666667	280.00000	2.666667
6	Bran Chex	R	134.32836	2.985075	1.492537	298.50746	5.970149
7	Bran Flakes	P	134.32836	4.477612	0.000000	313.43284	7.462687

Accessing by row

- Each row has an index (often unique but not required)
- By default, integers 0…N-1
- `df.index` – retrieve these row indices



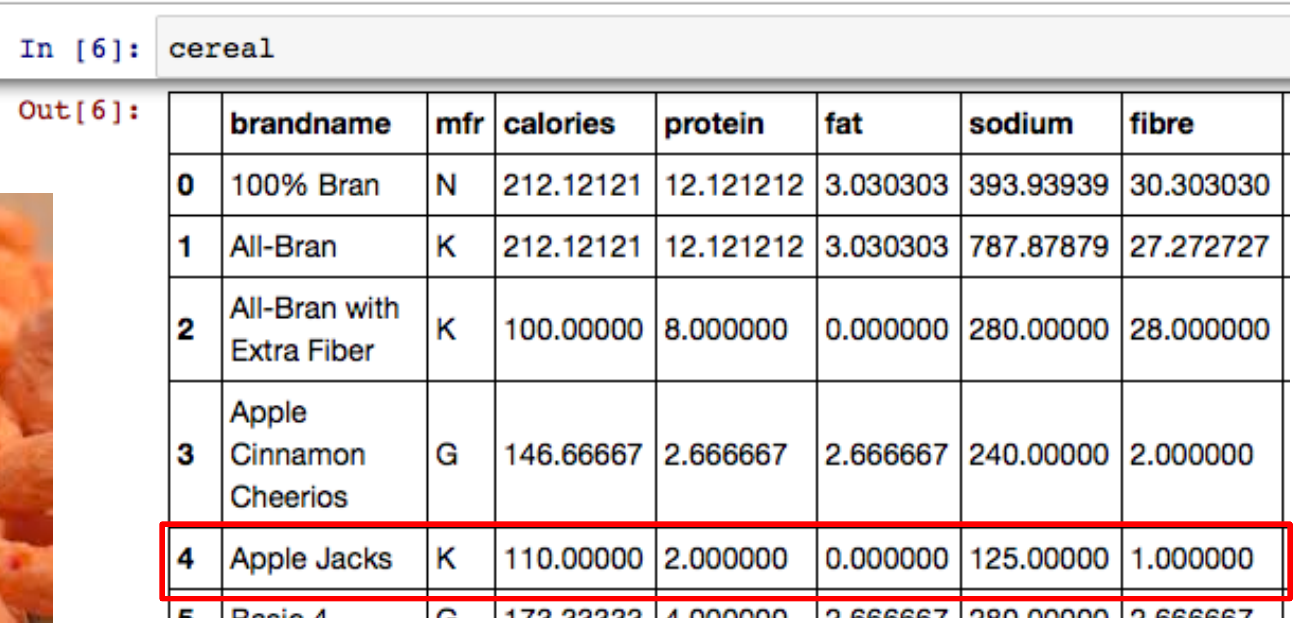
```
In [6]: cereal
```

Out[6]:

	brandname	mfr	calories	protein	fat	sodium	fibre
0	100% Bran	N	212.12121	12.121212	3.030303	393.93939	30.303030
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.272727
2	All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	28.000000
3	Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2.000000
4	Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1.000000
5	Basic 4	G	173.33333	4.000000	2.666667	280.00000	2.666667
6	Bran Chex	R	134.32836	2.985075	1.492537	298.50746	5.970149
7	Bran Flakes	P	134.32836	4.477612	0.000000	313.43284	7.462687

Accessing by rows using index

- With integer indices, selection works similarly to lists-of-lists you implemented in homework
- `df.iloc[X]` – get the row **at position #X** (0 …. L-1)
- Position is relative to the current dataframe (or portion thereof)



```
In [6]: cereal
Out[6]:
```

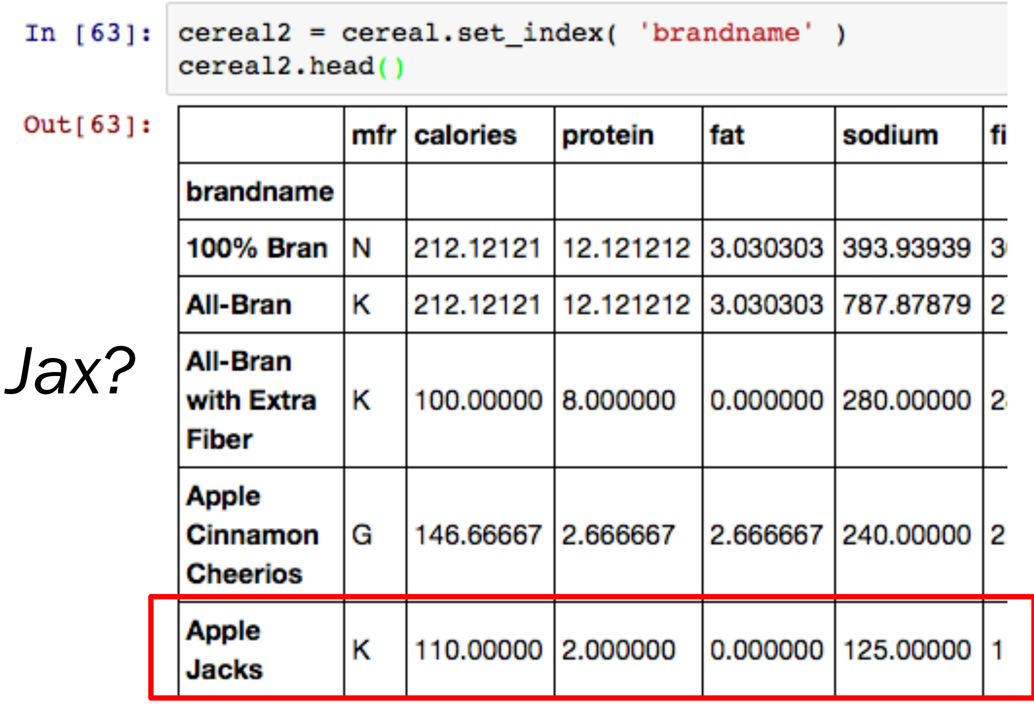
	brandname	mfr	calories	protein	fat	sodium	fibre
0	100% Bran	N	212.12121	12.121212	3.030303	393.93939	30.303030
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.272727
2	All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	28.000000
3	Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2.000000
4	Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1.000000
5	Basic 4	G	173.33333	4.000000	2.666667	280.00000	2.666667

[Pandas docs – indexing choices]

Indices don't have to be numbers

- Keeping track of item ←→ row number is cumbersome
- Indexes in pandas don't have to be numeric
- Instead they can be descriptive labels
- Use df.set_index() to index by a given column
- That column will (by default) disappear from the table and become the index
- `df.loc[X]` – get the row **with label** X

- *How to get Apple Jacks?*
- *What if we try to get Apple Jax?*
- *How would we instead get all Kellogg cereals?*

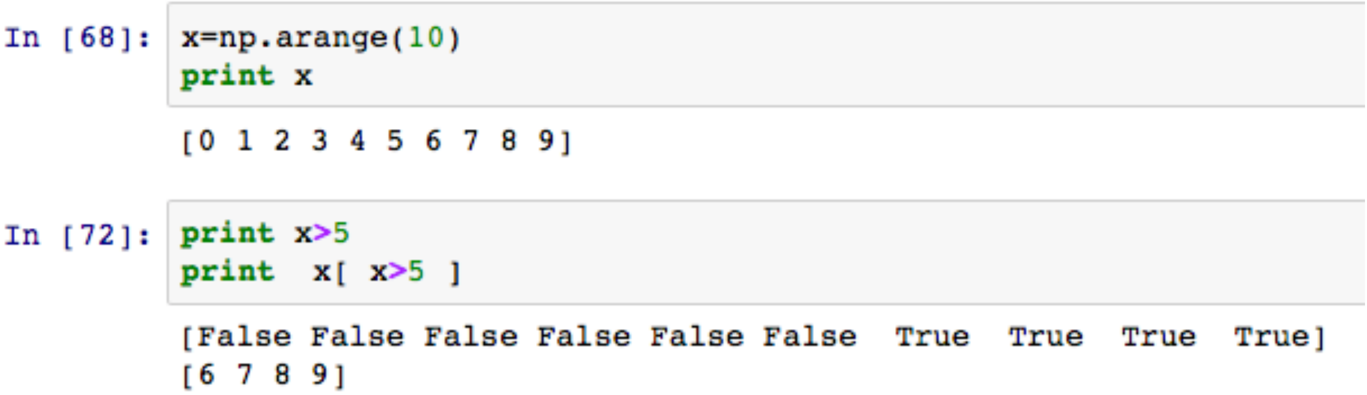


```
In [63]: cereal2 = cereal.set_index( 'brandname' )
         cereal2.head()
Out[63]:
```

	mfr	calories	protein	fat	sodium	fi
brandname						
100% Bran	N	212.12121	12.121212	3.030303	393.93939	3
All-Bran	K	212.12121	12.121212	3.030303	787.87879	2
All-Bran with Extra Fiber	K	100.00000	8.000000	0.000000	280.00000	2
Apple Cinnamon Cheerios	G	146.66667	2.666667	2.666667	240.00000	2
Apple Jacks	K	110.00000	2.000000	0.000000	125.00000	1

Selecting with boolean masks

- Recall from numpy array indexing that a rapid way to select a subset of entries is by list of booleans

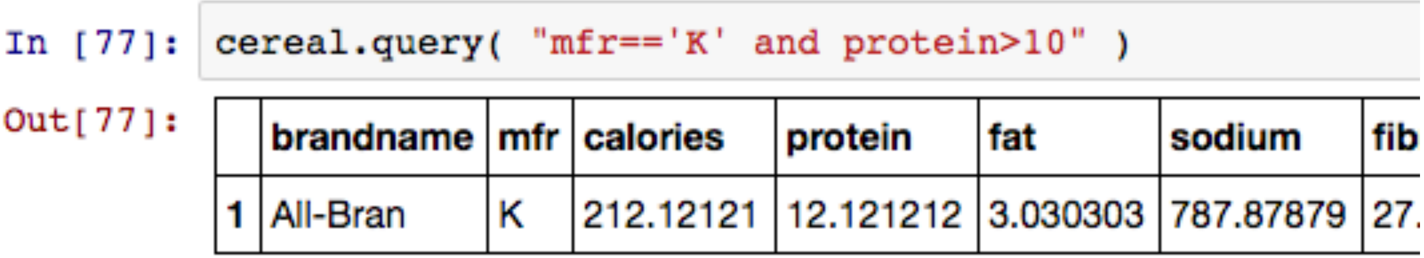


```
In [68]: x=np.arange(10)
         print x

         [0 1 2 3 4 5 6 7 8 9]

In [72]: print x>5
         print  x[ x>5 ]

         [False False False False False False  True  True  True  True]
         [6 7 8 9]
```

- Pandas supports a similar syntax. Can you retrieve all cereals made by Kellogg? Or, all with < 100 calories per serving?

Selecting with a query

- A second way to do this is to construct an expression string and pass that to df.query



```
In [77]: cereal.query( "mfr=='K' and protein>10" )
```

Out[77]:

	brandname	mfr	calories	protein	fat	sodium	fib
1	All-Bran	K	212.12121	12.121212	3.030303	787.87879	27.

Looping over all the rows

- Often we may wish to loop over rows and perform some task
- Use df.iterrows
- Note that each time through, it will return an index and the corresponding row

```
for curidx, currow in df.iterrows():
     print currow
```

Modifying/adding data

- DataFrame size is not fixed

- Can add columns to existing df:

```
cereal[ "delicious" ] = True  (repeat value for col)
cereal[ "transfat" ] = [1, 2.3, 3.4, …., 4.1 ]
```

  - This affects the dataframe in-place

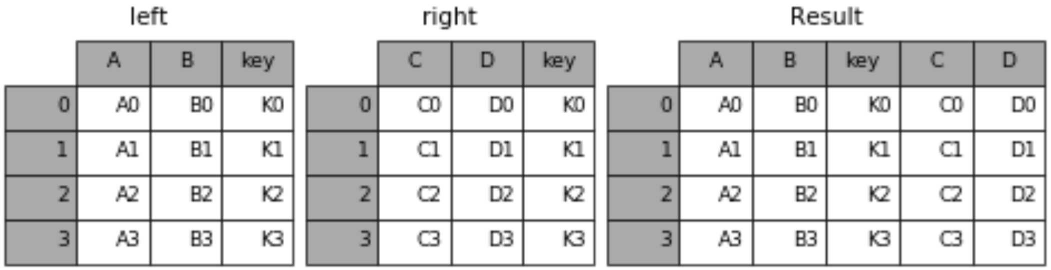
- Can append rows to an existing df

```
cereal.append( {'brandname':'oats',
  'mfr':'O', 'calories':55.5 }, ignore_index=True )
```

  - Makes a copy of the original dataframe
  - For large datasets this may be slow

Join

- Join two dataframes that share an index
- pd.merge(df_left, df_right, how)
  - How ='inner', 'left', 'right', 'outer'
  - All keys shared



left

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

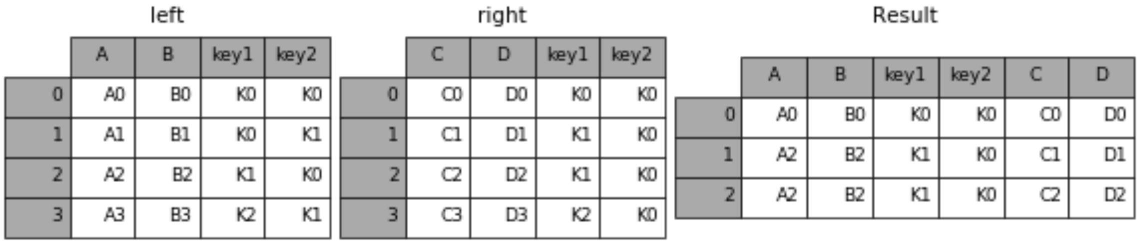
right

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

Result

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

  - Some missing: inner



left

	A	B	key1	key2
0	A0	B0	K0	K0
1	A1	B1	K0	K1
2	A2	B2	K1	K0
3	A3	B3	K2	K1

right

	C	D	key1	key2
0	C0	D0	K0	K0
1	C1	D1	K1	K0
2	C2	D2	K1	K0
3	C3	D3	K2	K0

Result

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2

http://pandas.pydata.org/pandas-docs/stable/merging.html

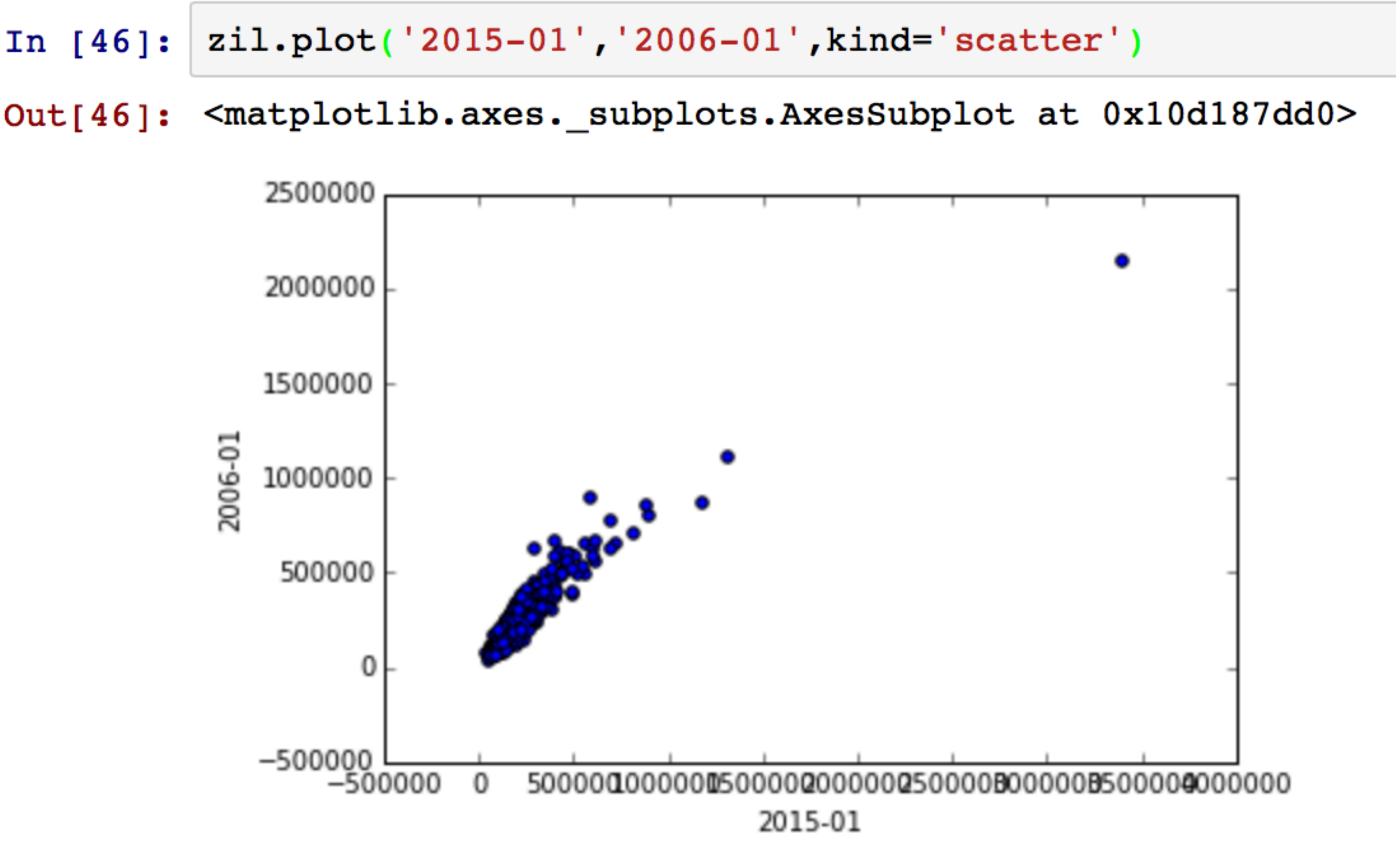
Group by

- `g = df.groupby( column )` → a grouped representation of the table
- Can iterate over the groups
- Can aggregate values *within* each group to get summary stats using `agg` function

- Try this:
  - `cereal.groupby('mfr').agg(mean)`

Pandas built-in visualization functionality

- `df.plot( x_column, y_column,  plot_name,  … )`



```
In [46]: zil.plot('2015-01','2006-01',kind='scatter')

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x10d187dd0>
```

More: http://pandas.pydata.org/pandas-docs/stable/visualization.html