

Exploring Data with Pandas I

**Georgia
Tech**

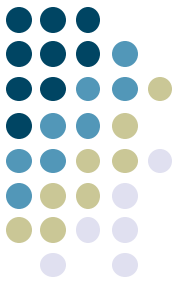


John Stasko

CS 6452

Prototyping Interactive Systems

Learning Objectives



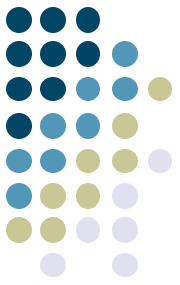
- Become familiar with Pandas library
- Learn about Pandas ability to read in and manipulate CSV files

Pandas



- Open source library providing data analysis capabilities for Python
- Designed to be high-performance with easy-to-use data structures
- Connects with other libraries to provide visualization capabilities too

Application



- Tabular data such as that from Excel spreadsheet or SQL table

- Two primary data structures
 - Series: 1-dimensional
 - DataFrame: 2-dimensional

Series



Like an array or list in Python

```
>>> s = pd.Series([12, 'Atlanta', 4.2, 'Jackets'])
>>> s
0          12
1  Atlanta
2         4.2
3  Jackets
dtype: object
```

Series



Can specify an index

```
>>> s = pd.Series([12, 'Atlanta', 4.2, 'Jackets'],
                  index=['A', 'B', 'C', 'D'])
>>> s
A      12
B  Atlanta
C      4.2
D  Jackets
dtype: object
```

Series



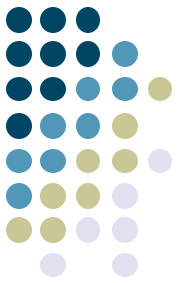
Can read in from a dictionary

```
>>> d = {'Chicago': 1000, 'New York': 1300, 'Portland': 900,
        'San Francisco': 1100, 'Austin': 450, 'Boston': None}
>>> cities = pd.Series(d)
>>> cities
```

Austin	450
Boston	NaN
Chicago	1000
New York	1300
Portland	900
San Francisco	1100
dtype: float64	

Keys become the index
NaN – not a number

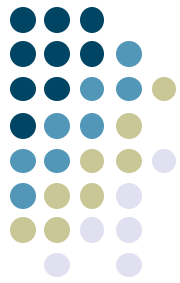
DataFrame



Like a table

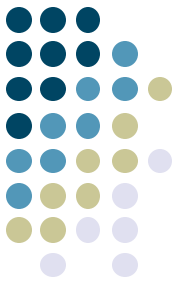
```
>>> data = {'year': [2010, 2011, 2012, 2011, 2012, 2010, 2011, 2012],
            'team': ['Bears', 'Bears', 'Bears', 'Packers',
                    'Packers', 'Lions', 'Lions', 'Lions'],
            'wins': [11, 8, 10, 15, 11, 6, 10, 4],
            'losses': [5, 8, 6, 1, 5, 10, 6, 12]}
>>> football = pd.DataFrame(data, columns=['year', 'team', 'wins',
                                           'losses'])
>>> football

# next slide...
```

	year	team	wins	losses
0	2010	Bears	11	5
1	2011	Bears	8	8
2	2012	Bears	10	6
3	2011	Packers	15	1
4	2012	Packers	11	5
5	2010	Lions	6	10
6	2011	Lions	10	6
7	2012	Lions	4	12

Play Along



- Grab `whiskeys.csv` from t-square
- Open Python from the folder that contains the file
- Do your `imports`

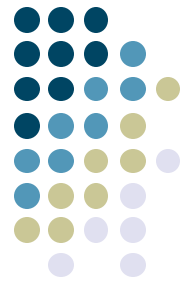
```
import pandas as pd
import matplotlib
```

Read in CSV



```
>>>import pandas as pd

>>>df = pd.read_csv("whiskeys.csv")
>>>df
```



	Name	Rating	Country
0	Tyrconnell 10 Year Old Single Malt Madeira Cas...	100	Ireland
1	Dalmore 18 Year Old Single Highland Malt Scotc...	100	Scotland
2	Powers 12 Year Old Irish Whiskey	99	Ireland
3	Suntory The Yamazaki 18 Year Old Single Malt W...	99	Japan
4	Glenmorangie 10 Year Old Single Malt Scotch Wh...	99	Scotland
5	Glenmorangie Single Malt Scotch	99	Scotland
6	Bunnahabhain 18 Year Old Single Malt Scotch	99	Scotland
7	Laphroaig 18 Year Old Single Malt Scotch	99	Scotland
8	...		
279	Glengoyne 10 Year Old Single Malt Scotch	*	Scotland
280	The Tweeddale Blend	*	Scotland
281	A.H. Hirsch Finest Reserve 20 Bourbon	*	USA
282	Elijah Craig 18 Year Bourbon	*	USA
283	Kavalan Solist Vinho Single Malt Whisky	*	Taiwan

	Category	Price	ABV	Age	Brand
0	Single Malt	72	46	10	Tyrconnell
1	Highlands	165	43	18	Dalmore
2	Blended	35	40	12	Powers
3	...				
277	Speyside	69	57	18	Glenlivet
278	Blended	*	43	*	Bell's
279	Highlands	NaN	*	10	Glengoyne
280	Blended	30	46	12	Tweeddale
281	Bourbon	280	45	16	A.H. Hirsch
282	Bourbon	40	45	18	Elijah Craig
283	Single Malt	100	59.2	*	Kavalan

[284 rows x 8 columns]

Info



What fields?

```
>>>df.columns
```

```
Index(['Name', 'Rating', 'Country', 'Category', 'Price', 'ABV', 'Age',  
      'Brand'],  
      dtype='object')
```

How many items?

```
>>>df.index
```

```
RangeIndex(start=0, stop=284, step=1)
```

**Note: All columns
are objects, not
ints or numbers**

Types

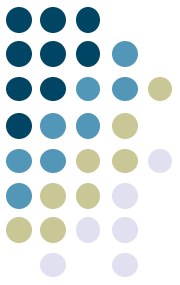


What types are in the file?

```
>>> df.types
Name          object
Rating        object
Country       object
Category      object
Price         object
ABV           object
Age           object
Brand         object
dtype: object
```

```
>>> df.ABV.dtype
dtype('O')
```

More Info



```
>>> df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 284 entries, 0 to 283  
Data columns (total 8 columns):  
Name          284 non-null object  
Rating        284 non-null object  
Country       284 non-null object  
Category      284 non-null object  
Price         281 non-null object  
ABV           282 non-null object  
Age           284 non-null object  
Brand         284 non-null object  
dtypes: object(8)  
memory usage: 8.9+ KB
```

Note



More Info



```
>>> df.describe(include=['object'])
```

```
count      Name Rating  Country Category \
unique      284     47         9         15
top  Aberfeldy 12 Year Old Single Malt Scotch      88  Scotland  Blended
freq                1     29         109         73

count  Price  ABV  Age  Brand
unique  107   48   25   153
top     70   40   *   Balvenie
freq    11   99  109         7
```


More Info



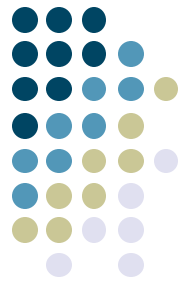
```
>>> df.describe(include=['int'])
```

```
Error
```

Why?

All columns read as strings in this file

Check First Few



```
>>> df.head()
```

```
                                Name Rating  Country \
0  Tyrconnell 10 Year Old Single Malt Madeira Cas...    100  Ireland
1  Dalmore 18 Year Old Single Highland Malt Scotc...    100  Scotland
2                Powers 12 Year Old Irish Whiskey      99  Ireland
3  Suntory The Yamazaki 18 Year Old Single Malt W...     99   Japan
4  Glenmorangie 10 Year Old Single Malt Scotch Wh...     99  Scotland
```

```
      Category  Price  ABV  Age      Brand
0  Single Malt     72   46   10  Tyrconnell
1   Highlands    165   43   18    Dalmore
2    Blended     35   40   12    Powers
3  Single Malt    120   43   18    Suntory
4   Highlands     42   40   10  Glenmorangie
```

Getting Fields



```
>>> pr_col = df["Price"]  
>>> pr_col
```

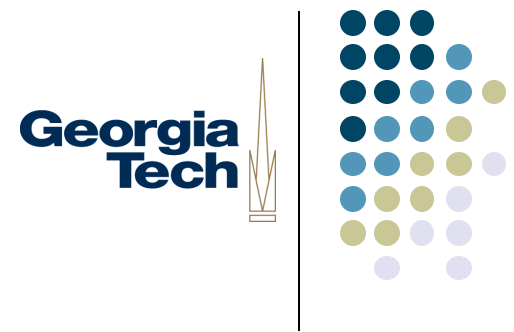
```
# results
```

```
>>> ra_col = df.Rating  
>>> ra_col
```

```
# results
```

Equivalent

Narrowing



How to get first 5 rows of a column?

```
>>> df["Country"][:5]
```

Getting count of unique values

```
>>> df["Country"].value_counts()
```

Narrowing



How to get all the Corn whiskeys?

```
>>> corns = df["Category"] == "Corn"  
>>> df[corns]
```

Getting count of unique values

```
>>> tops = df["Rating"] > 95
```

Why won't this work?

Specific Items



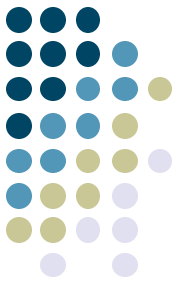
```
>>> df.iloc[3]
```

```
>>> df.loc[3]
```

```
>>> df[3]  
# Doesn't work
```

```
>>> df[0:3]
```

Accessing Items



What does this produce?

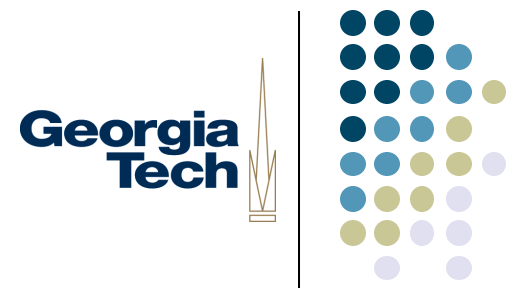
```
>>> df.Name == "Powers 12 Year Old Irish Whiskey"
```

```
>>> df[df.Name == "Powers 12 Year Old Irish Whiskey"]
```

```
      Name Rating  Country Category  Price  ABV  \
2  Powers 12 Year Old Irish Whiskey    99  Ireland  Blended    35  40
```

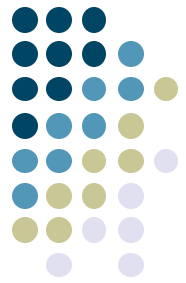
```
      Age  Brand
2    12  Powers
```

Cleaning the Data



- Frequently there are problems in the data
 - Strange values
 - Missing values
 - ...

Info on a Field

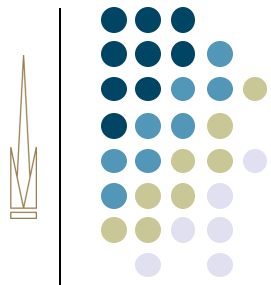


```
>>> df.Price.describe()  
count      281  
unique     107  
top         70  
freq        11  
Name: Price, dtype: object
```

3 are missing

```
>>> pr_col.count()  
281
```

Conversions



Let's change all the blank entries to *

```
>>> import numpy as np
```

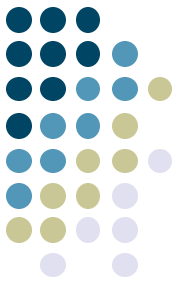
Act on the object
being called

```
>>> pr_col.replace(np.nan, '*', inplace=True)
```

```
>>> pr_col.count()
```

NumPy's NaN
(not a number)
denotes missing values

Substitutions

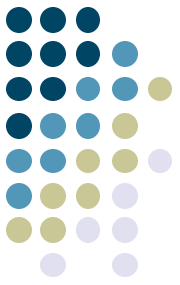


```
>>> df[df.Price.str.contains("\*")]["Name"]  
# What are the names of ones with Price '*'?  
# replace uses regex so need to escape *
```

Replace * by -1

```
>>> df.Price.replace('*', '-1', inplace=True)
```

Type Conversion



Want to change columns of strings (numbers) to just being numbers

```
>>> df["Price"] = df["Price"].astype("int")
```

Let's now focus on all whiskeys with Price > 100

```
>>> tops = df["Price"] > 100  
>>> tops  
>>> df[tops]
```

Sorting



Sort by some field

```
>>> df3 = df.sort_values(['Price', 'Name'], ascending=[0, 1])  
  
# First by Price (hi->low), then by Name (low->hi)
```

An Alternate Approach



Flexibility on reading in the file

```
>>> df = pd.read_csv("whiskeys.csv")
>>> df['Age'] = pd.to_numeric(df['Age'], errors='coerce')

# Change Age column to numeric
# coerce results in non-numeric becoming NaN

>>> df = df.dropna()

# Remove all the rows with nothing in some column

>>> print(df['Age'].max())
```

More...



Fix blanks on way in

```
>>> df.fillna(value=5)
```

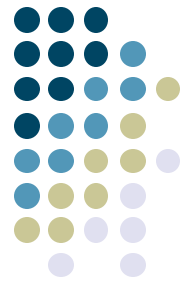
```
# Puts 5 in for all missing values
```

Scripts

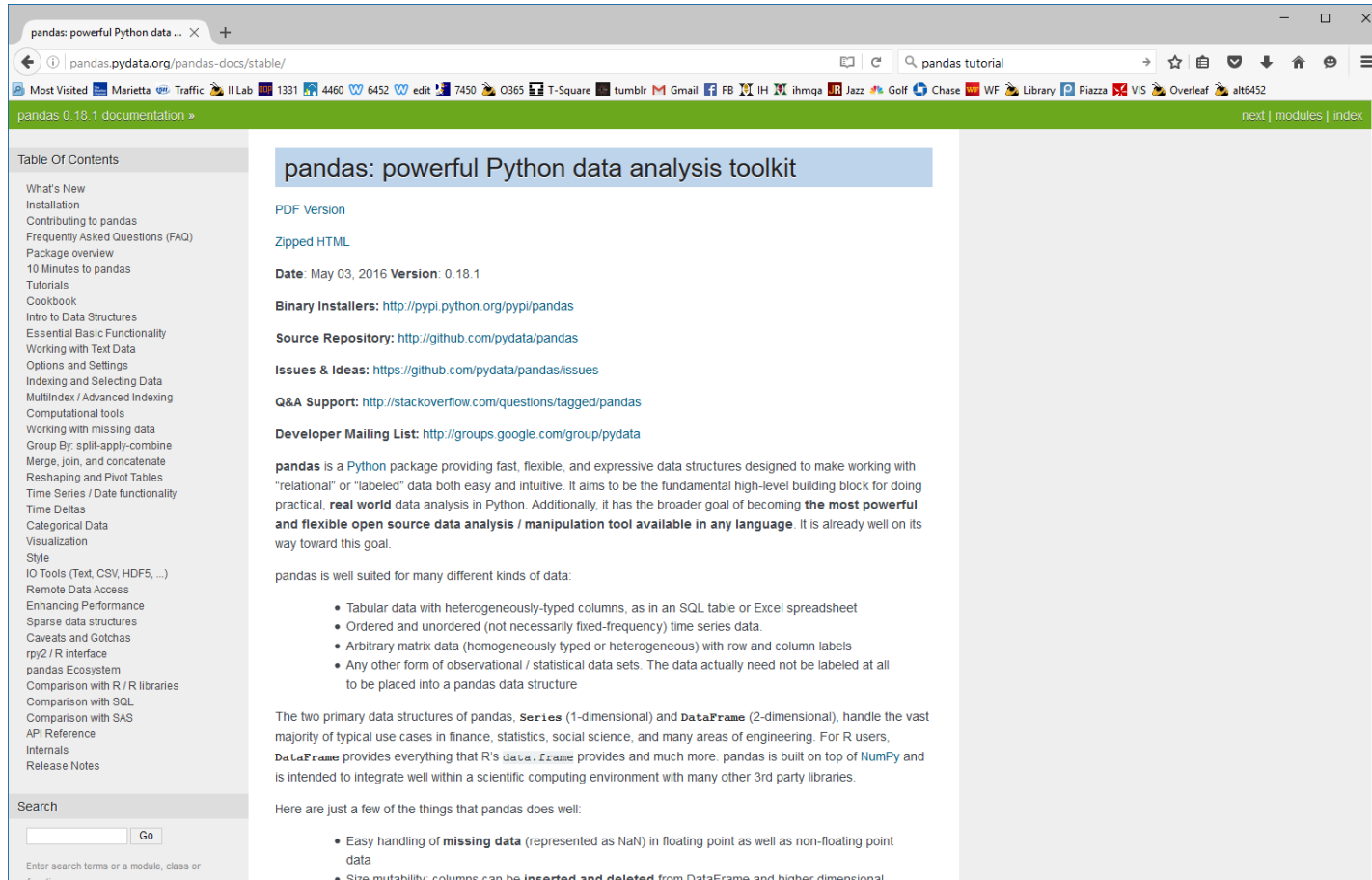


- We did this interactively
- But you can save a bunch of commands in a python file and run it like a script against different data sets

Documentation



<http://pandas.pydata.org/pandas-docs/stable/>



The screenshot shows the pandas documentation page for version 0.18.1. The page title is "pandas: powerful Python data analysis toolkit". It includes a Table of Contents on the left, a search bar, and a main content area with links to PDF, HTML, and binary installers. The main text describes pandas as a Python package for data analysis, highlighting its flexibility and performance. It lists various data structures it handles, such as tabular data, time series, and matrix data. The page also mentions that pandas is built on top of NumPy and is intended to integrate well with other scientific computing libraries.

pandas: powerful Python data analysis toolkit

PDF Version
Zipped HTML

Date: May 03, 2016 **Version:** 0.18.1

Binary Installers: <http://pypi.python.org/pypi/pandas>

Source Repository: <http://github.com/pydata/pandas>

Issues & Ideas: <https://github.com/pydata/pandas/issues>

Q&A Support: <http://stackoverflow.com/questions/tagged/pandas>

Developer Mailing List: <http://groups.google.com/group/pydata>

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. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

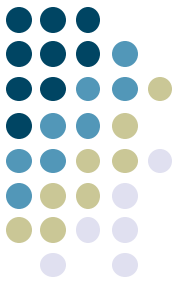
- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, **DataFrame** provides everything that R's **data.frame** provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional

Learning Objectives



- Become familiar with Pandas library
- Learn about Pandas ability to read in and manipulate CSV files

Next Time

- Using Pandas to visualize data

