Pandas!

Pandas is a Python module for data analysis. The central feature of Pandas is a data frame object. Data frames are useful objects for analyzing and manipulating tabular data. Think of a spreadsheet on steriods! The R programming language features a build in data frame class. In fact, data frames are the fundamental thing to work with in R. Newer versions of Matlab have the `table` class, which implements some of the functionality available in the Pandas and R data frames.

From the Pandas website:

- A fast and efficient DataFrame object for data manipulation with integrated indexing;
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible reshaping and pivoting of data sets;
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets;
- Columns can be inserted and deleted from data structures for size mutability;
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets;
- High performance merging and joining of data sets;
- Hierarchical axis indexing provides an intuitive way of working with high- dimensional data in a lower-dimensional data structure;
- Time series-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly optimized for performance, with critical code paths written in Cython or C.
- Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

First we must import pandas. The conventional way to do so is shown below:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

Import data

Let's look to see what is in this directory with a "magic" Jupyter command:

```
%ls
```

Ahh, we have a `csv` file. This is very easy to open in Pandas. Pandas can also open many other data types.

See: http://pandas.pydata.org/pandas-docs/version/0.17.0/io.html

Specifically, it is very easy to load data from MS Excel files (`pd.read_excel()`).

We have some basketball data from: http://www.basketball-reference.com/

```
nba = pd.read_csv("nba-2015.csv")
```

Let's inspect the data with the `head` method:

```
nba.head(10)
```

Exercise:

Modify the call to the `head` method to show more or fewer rows.

The output above does not show all of the columns! We can inspect the set of columns in a Pandas data fram by looking at the `columns` attribute.

```
nba.columns
```

Exercise

Write a loop to nicely print out the column headers.

Column glossary

In many data sets, a code is use for column headers. It is important to know about the data you are working with. Here is what the columns in this dataset mean:

```
| Rk     |                                |
| Player |                                |
| Pos    | Position                       |
| Age    |                                |
| Tm     | Team                           |
| G      | Games                          |
| GS     | Games started                  |
| MP     | Minutes played                 |
| FG     | Field goals                    |
| FGA    | Field goals attempted          |
| FG%    | Field goal percentage          |
| 3P     | 3 pt field goals               |
| 3PA    | 3 pt field goals attempted     |
| 3P     | 3 pt field goal percentace     |
| 2P     | 2 pt field goals               |
| 2PA    | 2 pt field goals attempted     |
| 2P     | 2 pt field goals percentage    |
| eFG%   | effective field goal percentage |
| FT     | Free throws                    |
| FTA    | Free throws attempted          |
| FT%    | Free throw percentage          |
| ORB    | Offensive rebounds             |
| DRB    | Defenseive rebounds            |
| TRB    | Total rebounds                 |
| AST    | Assists                        |
| STL    | Steals                         |
| BLK    | Blocks                         |
| TOV    | Turnovers                      |
| PF     | Personal fouls                 |
| PTS    | Total points                   |
```

```
nba.info()
```

Indexing

Operation	Syntax	Result
Select column	df[col]	Series
Select row by label	df.loc[label]	Series
Select row by integer location	df.iloc[loc]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean vector	df[bool_vec]	DataFrame

- A *Series* object is a single column in the table
- A *DataFrame* object is the table

Column selection

```
# extract a single column with column index name
nba['Player'].head()
```

```
nba['PTS'].head()
```

```
# select several columns by passing a sequence of column names
# (this returns a data frame)
nba[['Player','PTS']].head()
```

Row selection

Currently all row labels in this data set are integers, so row access via `nba.loc` and `nba.iloc` are equivalent.

```
nba.loc[100]
```

```
# select multiple rows with an integer index slice
nba[200:205]
```

```
# we can compute a boolean series with python inequality operators
(nba['PTS'] >= 1000).head(10)
```

```
# we can select all rows that pass a filter
nba[nba['PTS'] >= 1500]
```

Column modifications

```
# let's create a new column
nba['ASB'] = nba['AST'] + nba['STL'] + nba['BLK']
nba.head()
```

```
# let's delete the column we just made
del nba['ASB']
nba.head()
```

Simple plotting

```
# histogram of single column
nba['PTS'].hist()
```

```
# scatter plot of two columns
plt.plot(nba['G'],nba['FGA'],'o',alpha=0.2)
plt.xlabel('games played')
plt.ylabel('field goal %')
```

```
# scatter matrix of multiple columns
pd.tools.plotting.scatter_matrix(nba[['AST','FG','TRB','PF']], alpha=0.2)
plt.tight_layout()
```

Grouping and aggregation

In this data set, a player may have multiple rows based on teams they played for during the year. Let's get the total points for all players.

```
player_pts = nba[['Player','PTS']].groupby('Player').agg({'PTS':np.sum})
# this is an example of "method chaining"
```

```
player_pts.head()
```

```
player_pts.loc['Arron Afflalo']
```

```
nba[nba['Player'] == 'Arron Afflalo']
```

Exercises

List players who have played for more than one team. (Hint: use the `count()` method on the object returned from `groupby`)

What is the max number of teams any player has played for?

What is the highest scoring team? What is the lowest scoring team?

What is highest scoring position? What is lowest scoring position? How many players in each position? What is average score per position?

More questions: * Is there a correlation between fouls and rebounds? What if we normalize by minutes played? * What player has the most points per minute played? * Plot a histogram of minutes played for players? * What team has the highest variance of points scored by individual players? * What team has the lowest variance of points scored by individual players?

References

- **Python for Data Analysis** by Wes McKinney (2012) http://proquest.safaribooksonline.com/book/programming/python/9781449323592/firstchapter
- Pandas online documentation: http://pandas.pydata.org/pandas-docs/stable