Data Analysis with Pandas

~~Tushar B. Kute,~~

~~http://tusharkute.com~~

Pandas

- Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric Python packages.
- Pandas is one of those packages, and makes importing and analyzing data much easier.
- Pandas builds on packages like NumPy and matplotlib to give you a single, convenient, place to do most of your data analysis and visualization work.

Importing data with Pandas

- The first step we’ll take is to read the data in.
- The data is stored as a comma-separated values, or csv, file, where each row is separated by a new line, and each column by a comma (,).

Sample: movies.csv

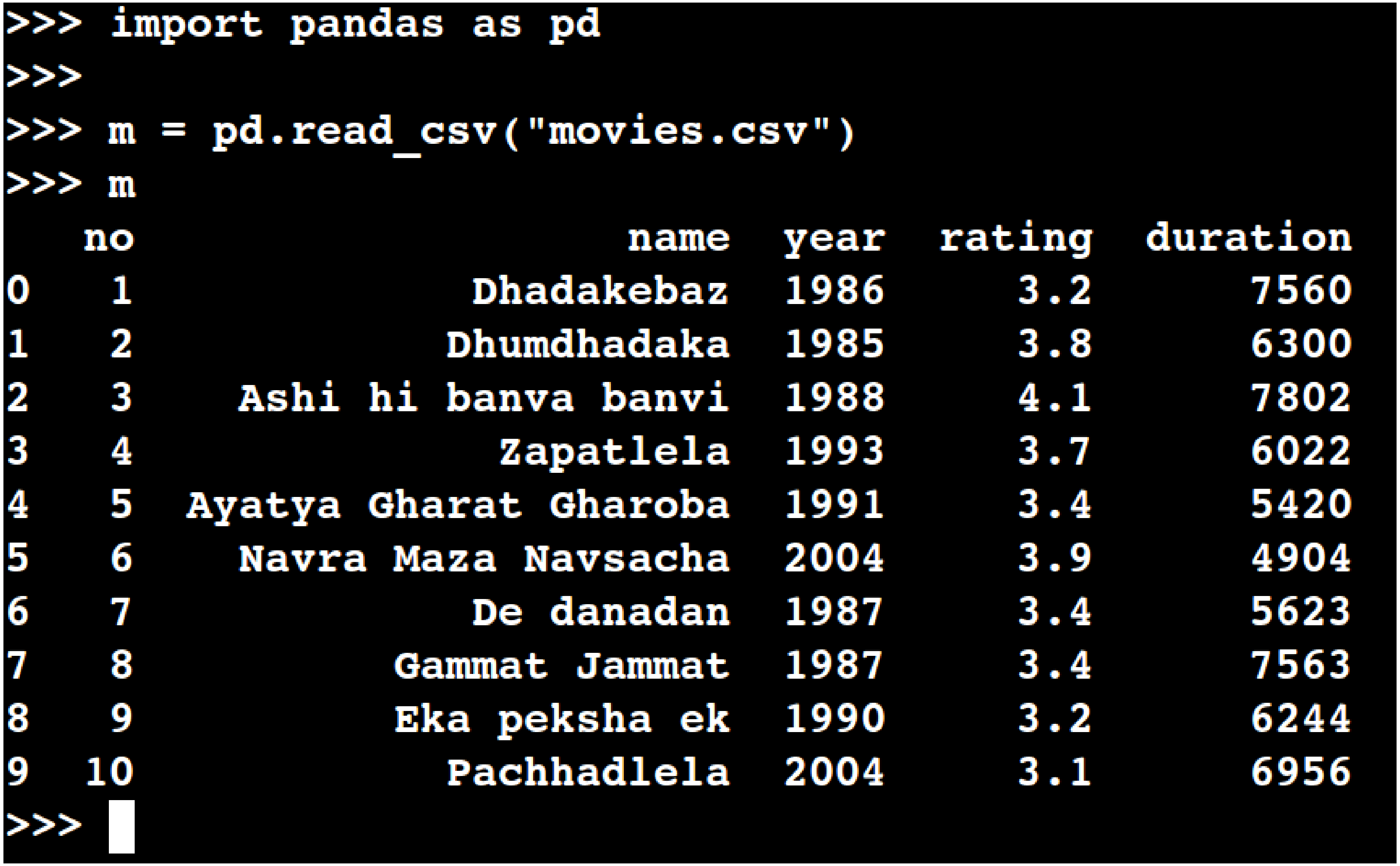
```
no,name,year,rating,duration
1,Dhadakebaz,1986,3.2,7560
2,Dhumdhadaka,1985,3.8,6300
3,Ashi hi banva banvi,1988,4.1,7802
4,Zapatlela,1993,3.7,6022
5,Ayatya Gharat Gharoba,1991,3.4,5420
6,Navra Maza Navsacha,2004,3.9,4904
7,De danadan,1987,3.4,5623
8,Gammat Jammat,1987,3.4,7563
9,Eka peksha ek,1990,3.2,6244
10,Pachhadlela,2004,3.1,6956
```

Read data

```
import pandas as pd

m = pd.read_csv("movies.csv")
```

Example:



```
>>> import pandas as pd
>>>
>>> m = pd.read_csv("movies.csv")
>>> m
   no                   name  year  rating  duration
0   1             Dhadakebaz  1986     3.2      7560
1   2            Dhumdhadaka  1985     3.8      6300
2   3    Ashi hi banva banvi  1988     4.1      7802
3   4              Zapatlela  1993     3.7      6022
4   5  Ayatya Gharat Gharoba  1991     3.4      5420
5   6    Navra Maza Navsacha  2004     3.9      4904
6   7             De danadan  1987     3.4      5623
7   8          Gammat Jammat  1987     3.4      7563
8   9          Eka peksha ek  1990     3.2      6244
9  10            Pachhadlela  2004     3.1      6956
>>>
```

Head and Tail

- Once we read in a DataFrame, Pandas gives us two methods that make it fast to print out the data. These functions are:
  - pandas.DataFrame.head – prints the first N rows of a DataFrame. By default 5.
  - pandas.DataFrame.tail – prints the last N rows of a DataFrame. By default 5.
- We'll use the head method to see what's in movies:

  m.head()

Find number of rows and columns

```
>>> m.shape
(10, 5)
>>> x = m.shape
>>> type(x)
<type 'tuple'>
>>> x[0]
10
>>> x[1]
5
```

Indexing DataFrames with Pandas

- Earlier, we used the head method to print the first 5 rows of reviews. We could accomplish the same thing using the pandas.DataFrame.iloc method.
- The iloc method allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well.
- The below code will replicate m.head():

```
m.iloc[0:5,:]
```

Some indexing examples

- m.iloc[:5,:] – the first 5 rows, and all of the columns for those rows.
- m.iloc[:,:] – the entire DataFrame.
- m.iloc[5:,5:] – rows from position 5 onwards, and columns from position 5 onwards.
- m.iloc[:,0] – the first column, and all of the rows for the column.
- m.iloc[9,:] – the 10th row, and all of the columns for that row.

Some indexing examples

- Now that we know how to retrieve rows and columns by position, it's worth looking into the other major way to work with DataFrames, which is to retrieve rows and columns by label.
- A major advantage of Pandas over NumPy is that each of the columns and rows has a label. Working with column positions is possible, but it can be hard to keep track of which number corresponds to which column.
- We can work with labels using the pandas.DataFrame.loc method, which allows us to index using labels instead of positions.
- We can display the first five rows of reviews using the loc method like this:

```
reviews.loc[0:5,:]
```

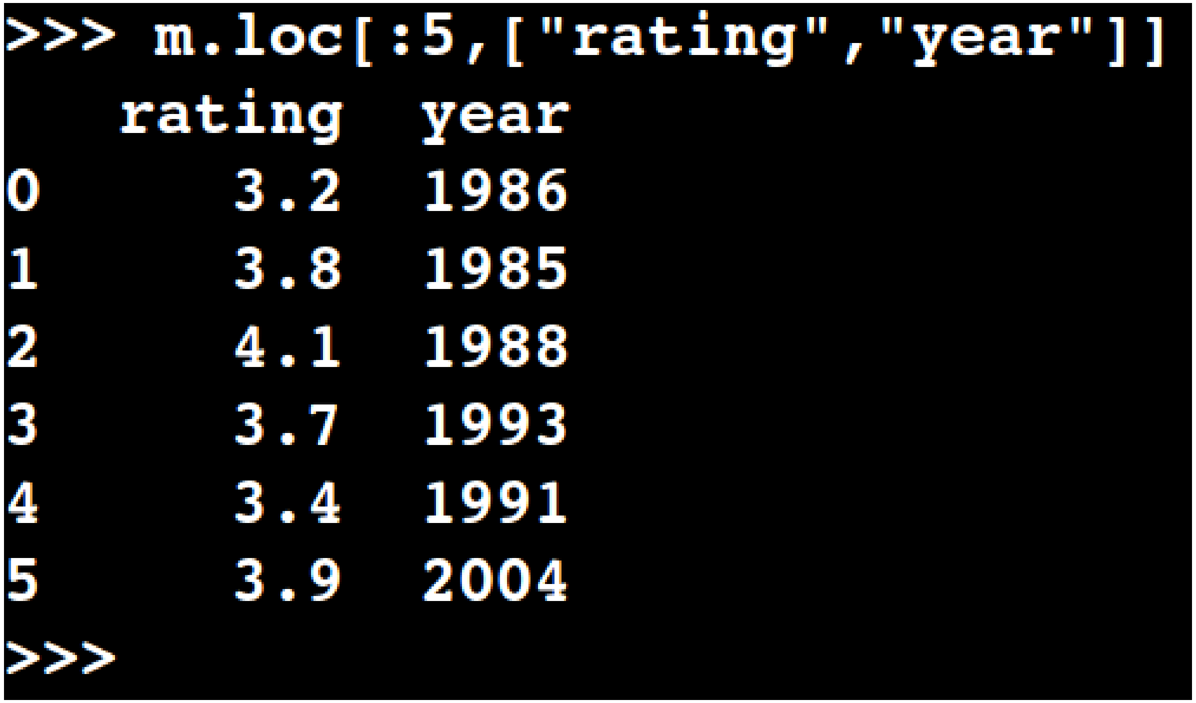
Some indexing examples

- Column labels can make life much easier when you’re working with data. We can specify column labels in the loc method to retrieve columns by label instead of by position.

```
m.loc[:5,"year"]
```

Multiple indexing

m.loc[:5,["rating","year"]]



```
>>> m.loc[:5,["rating","year"]]
   rating  year
0     3.2  1986
1     3.8  1985
2     4.1  1988
3     3.7  1993
4     3.4  1991
5     3.9  2004
>>>
```

Pandas series objects

- We can retrieve an individual column in Pandas a few different ways. So far, we’ve seen two types of syntax for this:

  m.iloc[:,1] – will retrieve the second column.

  m.loc[:,"year"] – will also retrieve the second column.

- There’s a third, even easier, way to retrieve a whole column. We can just specify the column name in square brackets, like with a dictionary:

  m["year"]

Data types

- When we retrieve a single column, we’re actually retrieving a Pandas Series object. A DataFrame stores tabular data, but a Series stores a single column or row of data.

- We can verify that a single column is a Series:

```
type(m["rating"])
pandas.core.series.Series
```

Series object

- We can create a Series manually to better understand how it works. To create a Series, we pass a list or NumPy array into the Series object when we instantiate it:

```
s1 = pd.Series([1,2])
s1
```

Series object

- A Series can contain any type of data, including mixed types. Here, we create a Series that contains string objects:

```
s2 = pd.Series(["Sachin Tendulkar", "Rahul Dravid"])
s2
```

Creating DataFrames

- We can create a DataFrame by passing multiple Series into the DataFrame class.
- Here, we pass in the two Series objects we just created, s1 as the first row, and s2 as the second row:

```
pd.DataFrame([s1,s2])
```

Creating DataFrames

- We can also accomplish the same thing with a list of lists. Each inner list is treated as a row in the resulting DataFrame:

```
pd.DataFrame(
    [
        [1,2],
        ["Sachin Tendulkar", "Rahul Dravid"]
    ]
)
```

Creating DataFrames

```
pd.DataFrame(
    [
        [1,2],
        ["Sachin Tendulkar", "Rahul Dravid"]
    ],
    columns = ["first","second"]
 )
```

Creating DataFrames

```
pd.DataFrame(
    [
        [1,2],
        ["Sachin Tendulkar", "Rahul Dravid"]
    ],
    columns = ["first","second"],
    index = ["row1","row2"]
)
```

DataFrame methods

- As we mentioned earlier, each column in a DataFrame is a Series object:

```
type(m["name"])
pandas.core.series.Series
```

- We can call most of the same methods on a Series object that we can on a DataFrame, including head:

```
m["name"].head()
```

DataFrame methods

- Pandas Series and DataFrames also have other methods that make calculations simpler. For example, we can use the pandas.Series.mean method to find the mean of a Series:

```
m["rating"].mean()
3.5200000000000005
```

- We can also call the similar pandas.DataFrame.mean method, which will find the mean of each numerical column in a DataFrame by default:

```
m.mean()
```

DataFrame methods

- We can modify the axis keyword argument to mean in order to compute the mean of each row or of each column.
- By default, axis is equal to 0, and will compute the mean of each column. We can also set it to 1 to compute the mean of each row. Note that this will only compute the mean of the numerical values in each row:

m.mean(axis=1)

DataFrame methods

- There are quite a few methods on Series and DataFrames that behave like mean. Here are some handy ones:
  - pandas.DataFrame.corr – finds the correlation between columns in a DataFrame.
  - pandas.DataFrame.count – counts the number of non-null values in each DataFrame column.
  - pandas.DataFrame.max – finds the highest value in each column.
  - pandas.DataFrame.min – finds the lowest value in each column.
  - pandas.DataFrame.median – finds the median of each column.
  - pandas.DataFrame.std – finds the standard deviation of each column.

DataFrame with Pandas

- We can also perform math operations on Series or DataFrame objects.
- For example, we can divide every value in the duration column by 2 to switch the scale from 0-10 to 0-5:

```
m["rating"]/2
```

- All the common mathematical operators that work in Python, like +, -, *, /, and ** will work, and will apply to each element in a DataFrame or a Series.

Boolean indexing

```
r_filter = m["rating"] > 3.7
r_filter

fm = m[r_filter]
fm
```

Multiple filtering

```
filter1 = (m["rating"] > 3.6) & (m["year"] > 1990)
filter1

m[filter1]
```

Thank you

Web Resources
http://mitu.co.in
http://tusharkute.com

Blogs
http://digitallocha.blogspot.in
http://kyamputar.blogspot.in

tushar@tusharkute.com