Hillsboro Python Machine Learning Meetup

Feb/2017

Ernest Bonat, Ph.D.

Senior Software Engineer

Senior Data Scientist

PC Wi-Fi: un = ps = PSUSUMMER2014

* 6:00 – 6:40 pm: Pizza, **water only** and networking.
* 6:40 – 6:45 pm: Welcome message by Ernest Bonat, Ph.D.
* 6:45 – 8:00 pm: Presentation and open discussions.
* 8.00 pm – 9.00 pm: Coding and learning session. Bring your Python development laptop!

Why did I create this meetup?

1. Bad traffic to Portland downtown.
2. Hard to find a parking.
3. Bad Python presentation code.
4. No time at all to review the presentation and learn something after the meeting.

We need your support:

1. Need 2 Senior Python Developers for presentation and code review every month (Co-organizers, 4-6 hours a month).
2. Every meeting cost about $200. We need companies to sponsor our meetings.
3. Email Ernest at [ebonat@15itresources.com](mailto:ebonat@15itresources.com)

Our Meetup Mission:

1. *“Come, Listen, Code and Learn”.*
2. Finding and presenting best practices of Machine Learning using Python Data Stack.
3. Create great networking place for Hillsboro-Beaverton Data Scientists.

Today Presentation

“Using Python Pandas Library for Data Manipulation and Cleansing”

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

(<http://pandas.pydata.org>)

Cheat Sheet

<https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf>

PDF Documentation File

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

Two main imports:

import pandas as pd

import numpy as np

Definition

|  |  |
| --- | --- |
| df | Any pandas DataFrame object |
| ds | Any pandas (Data) Series object |

Importing Data

|  |  |
| --- | --- |
| pd.read\_csv(filename) | From a CSV file |
| pd.read\_table(filename) | From a delimited text file (like TSV) |
| pd.read\_excel(filename) | From an Excel file |
| pd.read\_sql(query, connection\_object) | Read from a SQL table/database |
| pd.read\_json(json\_string) | Read from a JSON formatted string, URL or file. |
| pd.read\_html(url) | Parses an html URL, string or file and extracts tables to a list of dataframes |
| pd.read\_clipboard() | Takes the contents of your clipboard and passes it to read\_table() |
| pd.DataFrame(dict) | From a dict, keys for columns names, values for data as lists |

Exporting Data

|  |  |
| --- | --- |
| df.to\_csv(filename) | Write to a CSV file |
| df.to\_excel(filename) | Write to an Excel file |
| df.to\_sql(table\_name, connection\_object) | Write to a SQL table |
| df.to\_json(filename) | Write to a file in JSON format |

Viewing/Inspecting Data

|  |  |
| --- | --- |
| df.head(n) | First n rows of the DataFrame |
| df.tail(n) | Last n rows of the DataFrame |
| df.shape() | Number of rows and columns |
| df.info() | Index, Datatype and Memory information |
| df.describe() | Summary statistics for numerical columns |
| ds.value\_counts(dropna=False) | View unique values and counts |
| df.apply(pd.Series.value\_counts) | Unique values and counts for all columns |

Selection

|  |  |
| --- | --- |
| df[col] | Return column with label col as Series |
| df[[col1, col2]] | Return Columns as a new DataFrame |
| ds.iloc[0] | Selection by position |
| ds.loc['index\_one'] | Selection by index |
| df.iloc[0,:] | First row |
| df.iloc[0,0] | First element of first column |

Data Cleaning

|  |  |
| --- | --- |
| df.columns = ['a','b','c'] | Rename columns |
| pd.isnull() | Checks for null Values, Returns Boolean Arrray |
| pd.notnull() | Opposite of pd.isnull() |
| df.dropna() | Drop all rows that contain null values |
| df.dropna(axis=1) | Drop all columns that contain null values |
| df.dropna(axis=1,thresh=n) | Drop all rows have have less than n non null values |
| df.fillna(x) | Replace all null values with x |
| ds.fillna(s.mean()) | Replace all null values with the mean (mean can be replaced with almost any function from the statistics section) |
| ds.astype(float) | Convert the datatype of the series to float |
| ds.replace(1,'one') | Replace all values equal to 1 with 'one' |
| ds.replace([1,3],['one','three']) | Replace all 1 with 'one' and 3 with 'three' |
| df.rename(columns=lambda x: x + 1) | Mass renaming of columns |
| df.rename(columns={'old\_name': 'new\_ name'}) | Selective renaming |
| df.set\_index('column\_one') | Change the index |
| df.rename(index=lambda x: x + 1) | Mass renaming of index |

Descriptive Statistics

(These can all be applied to a series as well)

|  |  |
| --- | --- |
| df.describe() | Summary statistics for numerical columns |
| df.mean() | Return the mean of all columns |
| df.corr() | Finds the correlation between columns in a DataFrame |
| df.count() | Counts the number of non-null values in each DataFrame column |
| df.max() | Finds the highest value in each column |
| df.min() | Finds the lowest value in each column |
| df.median() | Finds the median of each column |
| df.std() | Finds the standard deviation of each column |

Why pandas?

* Heterogeneous data types
* Easy, fast missing data handling
* Easier to write generic code
* Labeled data (numpy mostly assumes index == label)
* Relational data

pandas Data Structures Objects

1. Series
2. DataFrame
3. Panel

Series

A one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the **index**.

ds = pd.Series(data, index=index)

Where: **data** can be: Python dictionary, ndarray (n-dimensional array or any scalar value (like 10)

Example:

ds = pd.Series(np.random.randn(5))

print(s)

Result:

0 0.3674

1 -0.8230

2 -1.0295

3 -1.0523

4 -0.8502

dtype: float64

DataFrame

A 2-dimensional labeled data structure with rows and columns of potentially different types (similar to Microsoft Excel spreadsheet or SQL database table)

df = pd.DataFrame(data, …)

**DataFrame** accepts many different kinds of input:

* Dictonary of 1D ndarrays, lists, dicts, or Series
* 2-D numpy.ndarray
* Structured or record ndarray
* A Series
* Another DataFrame

Example:

dictionary = {“one” : [1., 2., 3., 4.], “two” : [4., 3., 2., 1.]}

df = pd.DataFrame(dictionary)

Result:

one two

0 1.0 4.0

1 2.0 3.0

2 3.0 2.0

3 4.0 1.0

Indexing / Selection

The basics of indexing are as follows:

| **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 |

Panel

A 3-dimensional labeled data structure. It’s less-used today!

Missing Data

Missing Data is define as Non-available (NA), null or “not present for whatever reason”

**pandas uses “NaN” (Non-a-Number) or “nan” internally for simplicity and performance reasons**

In CSV file:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| one | two | three | four | five | timestamp |
|  | 2.1 | 3.1 | bar | 1 |  |
|  | 2.3 | 3.2 |  | 0 | 1/1/2017 |
| 1.3 |  | 3.3 | bar | 1 | 2/1/2017 |
| 1.4 | 2.4 |  | bar |  | 3/1/2017 |
| 1.5 | 2.5 | 3.5 | bar | 0 |  |

In pandas DataFrame:

==== ===== ===== ======= ====== ======

.. one two three four five timestamp

==== ===== ===== ======= ====== ======

0 **nan** 2.1 3.1 bar 1 nan

1 **nan** 2.3 3.2 **nan** 0 1/1/2017

2 1.3 **nan** 3.3 bar 1 2/1/2017

3 1.4 2.4 nan bar **nan** 3/1/2017

4 1.5 2.5 3.5 bar 0 **nan**

==== ===== ===== ======= ====== ======

Data Science Two Main Tasks:

|  |  |  |
| --- | --- | --- |
| 1 | **Data Cleansing** | **60% - 70% work** |
| 2 | Data Analytics | 40% - 30% work |

Data Cleansing very important task. Be careful with **“Garbage IN – Garbage OUT”**

Beginning Steps:

1. Organize Input and Output Data Files Path Name

2. Import Data File to pandas DataFrame

3. Get Number of Rows and Columns

4. Get Index, Datatype and Memory Information

5. Remove Duplicates Rows

6. Fill Nan Values (Mean, Median, Defaults, etc.)

7. Remove Rows by Row/Column Conditions

8. Replace Values by Row/Column Conditions