Data Scientist's Python Toolbox

Advanced Research Computing, Virginia Tech

Srijith Rajamohan

Course Contents

In this seminar:

- Introduction to Pandas for Data Science
- Introduction to Visualization and Plotting with Plot.ly
- Out-of-Core Computing with Dask for 'Biggish' Data

Introduction to Pandas

What is Pandas?

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

Pandas - import modules

Example

```
# General syntax to import a library but no
    functions:
>>>import pandas as pd #this is how I usually
    import pandas
>>>from pandas import DataFrame, read_csv
```

Pandas - Create a dataframe

- A DataFrame is a collection of records.
- It has a number of rows and columns.
- It can be 'indexed' for easy access.

Pandas - Create a dataframe

Example

```
>>>d = {'one' : pd.Series([1., 2., 3.], index=['a',
     'b', 'c']),
         'two' : pd.Series([1., 2., 3., 4.], index=[
            'a', 'b', 'c', 'd'])}
>>>df = pd.DataFrame(d)
>>>df
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
```

Pandas - Create a dataframe

Example

```
>>>names = ['Bob','Jessica','Mary','John','Mel']
>>>births = [968, 155, 77, 578, 973]
#To merge these two lists together we will use the
    zip function.

>>>BabyDataSet = list(zip(names,births))
>>>BabyDataSet
[('Bob', 968), ('Jessica', 155), ('Mary', 77), ('
    John', 578), ('Mel', 973)]
```

Pandas - Create a data frame and write to a csv file

Use the pandas module to create a dataset.

Example

```
>>>df = pd.DataFrame(data = BabyDataSet, columns=['
    Names', 'Births'])
>>>df.to_csv('births1880.csv',index=False,header=
    False)
```

Pandas - Read data from a file

Import data from the csv file

Example

```
>>>df = pd.read_csv(filename)
#Don't treat the first row as a header
>>>df = pd.read_csv(Location, header=None)
# Provide specific names for the columns
>>>df = pd.read_csv(Location, names=['Names','
    Births'])
```

Pandas - Get data types

Example

```
# Check data type of the columns
>>>df.dtypes
Names      object
Births      int64
dtype: object
# Check data type of Births column
>>>df.Births.dtype
dtype('int64')
```

Pandas - Take a look at the data

Example

```
>>>df.head(2)
           Names    Births
0          Bob      968
1    Jessica        155
>>>df.tail(2)
         Names    Births
3        John     578
4        Mel      973
```

Pandas - Take a look at the data

Example

```
>>>df.values
array([['Bob', 968],
       ['Jessica', 155],
       ['Mary', 77],
       ['John', 578],
       ['Mel', 973]], dtype=object)

>>>df.index
Int64Index([0, 1, 2, 3, 4], dtype='int64')
>>>df.columns
Index([u'Names', u'Births'], dtype='object')
```

Pandas - Working on the data

Example

```
>>>df['Births'].plot()
# Maximum value in the data set
>>>MaxValue = df['Births'].max()
# Name associated with the maximum value
>>>MaxName = df['Names'][df['Births'] == df['Births
    '].max()].values
```

Pandas - Describe the data

Example

```
>>>df.describe(include='all') # If you omit 'all',
    only numerical columns are considered
>>>print(df['Names'].describe())
count         5
unique        5
top        Mary
freq          1
Name: Names, dtype: object
>>>df['Names'].unique()
array(['Mary', 'Jessica', 'Bob', 'John', 'Mel'],
    dtype=object)
```

Pandas - Add a column

Example

```
>>>d = [0,1,2,3,4,5,6,7,8,9]

# Create dataframe
>>>df = pd.DataFrame(d)
#Name the column
>>>df.columns = ['Rev']
#Add another one and set the value in that column
>>>df['NewCol'] = 5
```

Pandas - Accessing and indexing the data

Example

```
#Perform operations on columns
>>>df['NewCol'] = df['NewCol'] + 1
#Delete a column
>>>del df['NewCol']
#Edit the index name
>>>i = ['a','b','c','d','e','f','g','h','i','j']
>>>df.index = i
```

Pandas - Accessing the data

Example

```
#Find rows based on index value
>>>df.loc['a']
>>>df.loc['a':'d']
#Do integer position based indexing
>>>df.iloc[0:3]
#Access a column using the column name
>>>df['Rev']
#Access multiple columns
>>>df[['Rev', 'test']]
#Subset the data using 'ix' which is a hybrid of '
   iloc' and 'loc'. Only works if index is not an
   integer. Can also subset with 'loc' and 'iloc'.
>>>df.ix[:3,['Rev', 'test']]
```

Pandas - Accessing and indexing the data

Get a specific value from the Dataframe

Example

```
#Find based on index value
>>>df.at['a','Rev']
0
>>>df.iat[0,0]
0
# Equivalent to saying
>>>df.iloc[0:1,0:1]
# or
>>>df.loc['a','Rev']
```

Pandas - Accessing and indexing for loc

- A single label, e.g. 5 or 'a', (note that 5 is interpreted as a label of the index. This use is not an integer position along the index)
- A list or array of labels ['a', 'b', 'c']
- A slice object with labels 'a':'f', (note that contrary to usual python slices, both the start and the stop are included!)
- A boolean array

Pandas - Accessing and indexing for iloc

- An integer e.g. 5
- A list or array of integers [4, 3, 0]
- A slice object with ints 1:7

Pandas - Accessing and indexing summarized

Example

```
loc:  only work on index
iloc: work on position
ix:    this is the most general and supports index
   and position based retrieval
at:   get scalar values, it's a very fast loc
iat:  get scalar values, it's a very fast iloc
[]: used to access the lower-dimensional slices
```

Pandas - Boolean indexing

Using boolean expressions to filter the data

Example

```
>>>df3 = pd.DataFrame(np.random.rand(6,4),index=
    list('abcdef'),columns=list('ABCD'))
>>>df3[df3.A > 0.5]
# Compound boolean expression
>>>df3[(df3.A > 0.5) & (df3.C > 0.5)]
```

Pandas - Access with a Callable

A callable is a function with one argument and returns valid output for indexing

Example

```
# Example of Callable
>>>df3.loc[lambda df: df.A > 0.5, ['A','C']]
# Chained indexing, slower so avoid it
>>>df3[df3.A > 0.5][['A','C']]
# Both examples above return the same result
```

Pandas - View vs Copy

Don’t assign with chained indexing, you are setting values to a copy.

Example

```
# This will throw a warning saying you are setting
    a value on a copy of a slice of a DataFrame
>>> df3[df3.C <= df3.B].loc[:,'A':'D'] = 3
# View, this is the way to perform the above
    operation
>>> df3.loc[df3.C <= df3.B,'A':'D'] = 3
```

Pandas - Copy a Dataframe

To copy a dataframe do the following

Example

```
>>>df2 = df.copy()
```

Pandas - Missing data

How do you deal with data that is missing or contains NaNs

Example

```
>>>df = pd.DataFrame(np.random.randn(5, 3), index=[
    'a', 'c', 'e', 'f', 'h'],
columns=['one', 'two', 'three'])
>>>df.loc['a','two'] = np.nan
        one       two     three
a -1.192838       NaN -0.337037
c  0.110718 -0.016733 -0.137009
e  0.153456  0.266369 -0.064127
f  1.709607 -0.424790 -0.792061
h -1.076740 -0.872088 -0.436127
```

Pandas - Missing data

How do you deal with data that is missing or contains NaNs?

Example

```
>>>df.isnull()
     one    two  three
a  False   True  False
c  False  False  False
e  False  False  False
f  False  False  False
h  False  False  False
```

Pandas - Missing data

You can fill this data in a number of ways.

Example

```
>>>df.fillna(0)
          one        two      three
a -1.192838  0.000000  -0.337037
c  0.110718 -0.016733 -0.137009
e  0.153456  0.266369 -0.064127
f  1.709607 -0.424790 -0.792061
h -1.076740 -0.872088 -0.436127
```

Pandas - Query the data

Also, use the query method where you can embed boolean expressions on columns within quotes

Example

```
>>>df.query('one > 0')
        one       two     three
c  0.110718 -0.016733 -0.137009
e  0.153456  0.266369 -0.064127
f  1.709607 -0.424790 -0.792061
>>>df.query('one > 0 & two > 0')
        one       two     three
e  0.153456  0.266369 -0.064127
```

Pandas - Apply a function

You can apply any function to the columns in a dataframe. Note that attention must be paid to the data type in the columns.

Example

```
>>>df.apply(lambda x: x.max() - x.min())
one      2.902445
two      1.138457
three    0.727934
```

Pandas - Applymap a function

You can apply any function to the element wise data in a dataframe

Example

```
# You can also use user-defined functions here
>>>df.applymap(np.sqrt)
        one       two  three
a       NaN       NaN    NaN
c  0.332742       NaN    NaN
e  0.391735  0.516109    NaN
f  1.307520       NaN    NaN
h       NaN       NaN    NaN
```

Pandas - Query data

Determine if certain values exist in the dataframe

Example

```
>>>s = pd.Series(np.arange(5), index=np.arange(5)
   [::-1], dtype='int64')
>>>s.isin([2,4,6])
4    False
3    False
2     True
1    False
0     True
```

Pandas - Query data

Use the where method

Example

```
>>>s = pd.Series(np.arange(5), index=np.arange(5)
   [::-1], dtype='int64')
>>>s.where(s>3)  # By default, positions with
   unsatisfied conditions are filled with NaNs.
   This can be changed with the `other = VALUE'
   argument to `where'
4   NaN
3   NaN
2   NaN
1   NaN
0     4
```

Pandas - Grouping the data

Creating a grouping organizes the data and returns a groupby object

Example

```
>>>grouped = obj.groupby(key)
>>>grouped = obj.groupby(key, axis=1)
>>>grouped = obj.groupby([key1, key2])
```

Pandas - Grouping the data

Example

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar
    ',
'foo', 'bar', 'foo', 'foo'],
'B' : ['one', 'one', 'two', 'three',
'two', 'two', 'one', 'three'],
'C' : np.random.randn(8),
'D' : np.random.randn(8)})
```

Pandas - Grouping the data

Example

```
     A      B         C         D
0  foo    one  0.469112 -0.861849
1  bar    one -0.282863 -2.104569
2  foo    two -1.509059 -0.494929
3  bar  three -1.135632  1.071804
4  foo    two  1.212112  0.721555
5  bar    two -0.173215 -0.706771
6  foo    one  0.119209 -1.039575
7  foo  three -1.044236  0.271860
```

Pandas - Grouping the data

Group by either A or B columns or both

Example

```
>>>grouped = df.groupby('A')
>>>grouped = df.groupby(['A', 'B'])
# Sorts by default, disable this for potential
    speedup
>>>grouped = df.groupby('A',sort=False)
```

Pandas - Grouping the data

Get statistics for the groups

Example

```
>>>grouped.ngroups
>>>grouped.size()
>>>grouped.describe()
>>>grouped.count()
```

Pandas - Grouping the data

Print the grouping

Example

```
>>>list(grouped)
 A      B          C         D
 1  bar    one -1.303028 -0.932565
 3  bar  three  0.135601  0.268914
 5  bar    two -0.320369  0.059366)
 0  foo    one  1.066805 -1.252834
 2  foo    two -0.180407  1.686709
 4  foo    two  0.228522 -0.457232
 6  foo    one -0.553085  0.512941
 7  foo  three -0.346510  0.434751)]
 # You can also list it as such
 >>> grouped.groups
```

Pandas - Grouping the data

Get the first and last elements of each grouping. Also, get a particular group.

Example

```
>>>grouped.first() # Similar results can be
    obtained with g.last()
A      B          C          D
bar  one -1.303028 -0.932565
foo  one  1.066805 -1.252834
# Get a particular group
>>>grouped.get_group('bar')
    B         C              D
1   one    -0.786532   1.213176
3   three   0.515047  -0.664787
5   two    -1.254082  -1.322434
```

Pandas - Grouping the data

Group aggregation, apply the 'sum' function to each column for each group

Example

```
# Built in function for grouped object
>>>grouped.sum()
 A           C          D
bar -1.487796 -0.604285
foo  0.215324  0.924336
# Apply a function that does group aggregation,
   including user-defined functions that perform
   aggregation
>>>grouped.aggregate(np.sum)
A           C          D
bar -1.487796 -0.604285
foo  0.215324  0.924336
```

Pandas - Grouping the data

Apply multiple functions to a grouped column

Example

```
# You can also apply this to an entire Dataframe
>>>grouped['C'].aggregate([np.sum, np.mean])
A         sum      mean
bar -1.487796 -0.495932
foo  0.215324  0.043065
```

Pandas - Grouping the data

Visually inspecting the grouping

Example

```
>>>w = grouped['C'].agg([np.sum, np.mean]).plot(
    kind='bar')
>>>import matplotlib.pyplot as plt
>>>plt.show()
```

Pandas - Grouping the data

Apply a transformation to the grouping

Example

```
>>>f = lambda x: x*2
>>>transformed = grouped.transform(f)
>>>print transformed
```

Pandas - Grouping the data

Apply a filter to select a group based on some criterion. Only groups that satisfy this condition will be printed. The function passed to the filter method should return a scalar value (True or False) for each group.

Example

```
>>>grouped.filter(lambda x: sum(x['C']) > 0)
```

```
     A       B         C         D
0  foo     one  1.066805 -1.252834
2  foo     two -0.180407  1.686709
4  foo     two  0.228522 -0.457232
6  foo     one -0.553085  0.512941
7  foo   three -0.346510  0.434751
```

Visualization

Visualization

We will use the Jupyter Notebook format to learn more about

- Plot.ly

Plot.ly

Learn more at `https://plot.ly`

- Open-source and commercial visualization tool for interactive charts
- Collaborative and online chart creation tool
- API for offline use
- Can be used to share online
- Part of a larger Analytics package

Out-of-Core Computing

Out-of-Core Computing

We will use the Jupyter Notebook format to learn more about

- Dask: Dask is a flexible parallel computing library for analytic computing

Dask

Learn more at `http://dask.pydata.org/en/latest/`

- Dynamic task scheduling optimized for computation
- “Big Data” collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments.
- These parallel collections run on top of the dynamic task schedulers.

Conclusion

Summary

For questions about this material please email me at:

srijithr@vt.edu

Questions?