pandas

Python Data Analysis Library

*pandas* is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the [Python](http://www.python.org/) programming language.

Python has long been great for data munging and preparation, but less so for data analysis and modeling. *pandas* helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

**Library Highlights**

* 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](http://www.cython.org/) 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.

***pandas* Tutorial**

The purpose of this tutorial is to use pandas to manage (biological) data stored in a table. It has been adapted from "Lessons for New pandas Users" (<http://pandas.pydata.org/pandas-docs/version/0.16.0/tutorials.html>) by David Rojas.

**Lesson 1**

Create Data - We begin by creating our own data set for analysis. This prevents the end user reading this tutorial from having to download any files to replicate the results below. We will export this data set to a text file so that you can get some experience pulling data from a text file.

Get Data - We will learn how to read in the text file. The data consist of genes and their expression value .

Prepare Data - Here we will simply take a look at the data and make sure it is clean. By clean I mean we will take a look inside the contents of the text file and look for any anomalities. These can include missing data, inconsistencies in the data, or any other data that seems out of place. If any are found we will then have to make decisions on what to do with these records.

Analyze Data - We will simply find the most expressed gene.

Present Data - Through tabular data and a graph, clearly show the end user what is the most expressed gene.

The pandas library is used for all the data analysis excluding a small piece of the data presentation section. The matplotlib library will only be needed for the data presentation section. Importing the libraries is the first step we will take in the lesson.

'''

# Import all libraries needed for the tutorial

# General syntax to import specific functions in a library:

##from (library) import (specific library function)

from pandas import DataFrame, read\_csv

# General syntax to import a library but no functions:

##import (library) as (give the library a nickname/alias)

import matplotlib.pyplot as plt

import pandas as pd #this is how I usually import pandas

import sys #only needed to determine Python version number

# Enable inline plotting

#%matplotlib inline

print 'Python version ' + sys.version

print 'Pandas version ' + pd.\_\_version\_\_

#CREATE DATA

#The data set will consist of a set of numbers extracted from the third column of random\_distribution.tsv.

#You can pretend these are numbers reflecting the expression levels of the genes in the first column

col2 = ['gene1', 'gene2', 'gene3', 'gene4', 'gene5']

col3 = [0.0169659034755, 0.0178512938094, 0.015126870527, 0.018630495179, 0.0142203334423]

DataSet = zip(col2, col3)

print DataSet

#We now will use the pandas library to export this data set into a csv file

#df will be a DataFrame object. You can think of this object holding the contents

#of the DataSet in a format similar to a sql table or an excel spreadsheet.

#Lets take a look below at the contents inside df.

df = pd.DataFrame(data = DataSet, columns = ['gene', 'express'])

#print df

#Export the dataframe to a csv file. The function to\_csv will be used to export the file.

#The only parameters we will use is index and header. Setting these parameters to True will

# prevent the index and header names from being exported.

#Change the values of these parameters to get a better understanding of their use.

df.to\_csv('gene\_expression.csv',index=False,header=False)

#GET DATA

#To pull in the csv file, we will use the pandas function read\_csv.

#Let us take a look at this function and what inputs it takes.

Location = r'/Users/allegra1/Documents/didattica/TrainingCourses/TGAC/April2015/Course/Day4/PythonLibraries/Pandas/gene\_expression.csv'

df = pd.read\_csv(Location)

#print df

#This brings us the our first problem of the exercise. The read\_csv function treated

#the first record in the csv file as the header names. This is obviously not correct

#since the text file did not provide us with header names.

df = pd.read\_csv(Location, header=None)

print df

#If we wanted to give the columns specific names, we would have to pass another

#paramter called names. We can also omit the header parameter.

df = pd.read\_csv(Location, names=['gene','express'])

print df

#You can think of the numbers [0,1,2,3,4] as the row numbers in an Excel file.

#In pandas these are part of the index of the dataframe. You can think of the

#index as the primary key of a sql table with the exception that an index is

#allowed to have duplicates.

#[gene, express] can be though of as column headers similar to the ones found in

#an Excel spreadsheet or sql database.

#Delete the csv file now that we are done using it.

import os

os.remove(Location)

#PREPARE DATA

'''

The gene column at this point is of no concern since it most likely is just

composed of alpha numeric strings (genes). There is a chance of bad data

in this column but we will not worry about that at this point of the analysis.

The express column should just contain floating numbers representing the expression

values. We can check if the all the data is of the data type float.

I would not worry about any possible outliers at this point

of the analysis.

'''

# Check data type of the columns

print df.dtypes

# Check data type of Births column

print df.express.dtype

#ANALYSE DATA

'''

To find the most expressed gene, we can do one of the following.

- Sort the dataframe and select the top row

- Use the max() attribute to find the maximum value

'''

# Method 1:

Sorted = df.sort(['express'], ascending=False)

print Sorted.head(1)

# Method 2:

print df['express'].max()

#PRESENT DATA

'''

Here we can plot the express column and label the graph to show the end user

the highest point on the graph. In conjunction with the table, the end user

has a clear picture that gene4 is the most expressed gene in the data set.

plot() is a convinient attribute where pandas lets you painlessly plot the data

in your dataframe. We learned how to find the maximum value of the express column

in the previous section. Now to find the actual most expressed gene looks a bit

tricky, so lets go over it.

Explain the pieces:

df['gene'] - This is the entire list of genes, the entire gene column

df['express'] - This is the entire list of expression values, the entire express column

df['express'].max() - This is the maximum value found in the express column

[df['express'] == df['express'].max()] IS EQUAL TO [Find all of the records in the express

column where it is equal to 0.018630495179]

df['gene'][df['express'] == df['express'].max()] IS EQUAL TO Select all of the records in

the gene column WHERE [The express column is equal to 0.018630495179]

An alternative way could have been to use the Sorted dataframe:

Sorted['gene'].head(1).value

The str() function simply converts an object into a string.

'''

# import the figure and savefig objects from matplotlib (pylab):

from pylab import figure, savefig

# Create graph

df['express'].plot()

# Maximum value in the data set

MaxValue = df['express'].max()

# Name associated with the maximum value

MaxName = df['gene'][df['express'] == df['express'].max()].values

# Text to display on graph

Text = str(MaxValue) + " - " + MaxName

# Add text to graph

plt.annotate(Text, xy=(1, MaxValue), xytext=(8, 0),

xycoords=('axes fraction', 'data'), textcoords='offset points')

print "The most expressed gene"

print df[df['express'] == df['express'].max()]

#Sorted.head(1) can also be used

#Show the plot

plt.show()

#Save the plot to a file

savefig('GeneExpression.png')