

Pandas is an open source library used for data structure and data analysis within the python environment.

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](#) 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

maria wrote:

what is useful to know about pandas: keep in mind:

```
-import pandas as pd
```

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

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

```
DataSet = zip(col2, col3)
```

```
print (DataSet)
```

with the function zip here you are taking a list as an object and returns an iterator of tuples in this case:

```
{('gene1',0.0169659034755), ('gene2',0.0178512938094) ecc..}
```

create the table with the data from the tuple you created and order in columns

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

then you can export this data set into a csv file:

```
- df.to_csv('gene_expression.csv',index=False,header=False)  
(you have to put false to make the index and the header be printed also)
```

to pull the table in the csv file:

- `df = pd.read_csv('Location of the file')`

to avoid the problem of having the first record of the csv file as an header we can write;

- `df = pd.read_csv(Location, header=None)`

and we can give the columns a name:

- `df = pd.read_csv(Location, names=['gene','express'])`

we can also Check the data type of the columns using:

- `print df.dtypes`

POSSIBLE WORKFLOW

- 1) Brief introduction on what pandas library is useful for (data analysis and plotting, R-like fashion) [\[Lussyboi\]](#)
- 2) Data Structure
 - a) DataFrames [\[Khansa\]](#)
 - i) A 2-dimensional labeled data structure, creating DataFrames
 - (1) type of input data, examples
 - (2) columns can contain different data types, NaN will be automatically inserted when needed
 - (3) index (colnames, rownames)
 - ii) DataFrames methods, examples
 - iii) DataFrames selection, boolean indexing, boolean masks, isna, examples
 - b) Series [\[Nico\]](#)
 - i) A one-dimensional labeled array capable of holding any data type: examples. Supports non-unique index values
 - ii) Accessing Series: `s[index]`, `s.get(index)` will not raise exceptions for missing labels, index in `s`
 - iii) Mathematical operations on series, operations between series automatically align the data based on labels
- 3) Filehandling [\[Marilù\]](#)
 - a) Reading from different file types, examples
 - b) Process data exploiting proper data structures, examples
 - c) Writing to different filetypes, examples
- 4) Plots
 - a) To define [\[Lussyboi\]](#)
 - b) To define

Secondo me rega qualcuno può cominciare con questo:

https://github.com/ELIXIR-IIB-training/python_course/blob/master/day4/3-PythonLibraries/pandas/pandas_tutorial.docx

Nico, is it ok for you? Yea I don't mind

Nicooooooo :-P if you are up to it, see the presentation track below, you can decide what to do!

Use <http://pandas.pydata.org/pandas-docs/stable/10min.html> to have an idea on what to do about each part (and the python course repository too

https://github.com/ELIXIR-IIB-training/python_course/tree/master/day4/3-PythonLibraries/pandas)

=====

Presentation begins:

- **Intro** [LussyBoi] [what pandas is useful for, pandas as R inside python, some data about pandas in data science]

Handling of complex set of data is easily achievable thanks to smart classes and functions; the most important object type is DataFrame:

- **DataFrames** and **Series** [Khansa] [create and show some features and methods of a dummy DataFrame] [selection by position, by label] [create and show some features and methods of a dummy Serie]

We can exploit these types of objects to represent real data:

- **filehandling I/O and operations** [Mimi] (read a real world file, manage the col and row names, do other basic important stuff, select a part of it and write the new object to a file) (read, sort, sum, stats ... , write)

Exemple to filter and process data in DataFrames:

- **Filtering of DataFrames** [Nico] [show fancy and smart ways to filter the previously loaded DataFrame using boolean masks and other things] [boolean indexing, selection, boolean masks, columns that satisfy a condition...]
- **Exemple of dummy analysis** [Nico] (perform some processing on the same data to show some useful functions) [std]

Hey, furthermore, you can also represent on a graph your fucking dataaaaaa:

- **Plotting** [Nico]

=====

```
#####
```

```
### Basic usage examples ###
```

For Khansa (et al.):

```
## =====
```

```
## Create a dummy DataFrame
```

```
import pandas as pd
```

```
genes = ["gene1", "gene2", "gene3", "gene4"]
```

```
t1 = [1.0, 2.0, 1.3, 0.5]
```

```
t2 = [2.0, 2.1, 1.0, 0.8]
```

```
t3 = [2.5, 1.9, 0.8, 1.4]
```

```
t4 = [2.7, 1.9, 0.7, 1.9]
```

```
genexpr = pd.DataFrame(data = [t1, t2, t3, t4], index = ["expr1", "expr2", "expr3", "expr4"], columns=genes)
```

```
genexpr = genexpr.T      ## transpose
```

```
'''
```

```
>>> genexpr
```

```
      expr1  expr2  expr3  expr4
gene1    1.0    2.0    2.5    2.7
gene2    2.0    2.1    1.9    1.9
gene3    1.3    1.0    0.8    0.7
gene4    0.5    0.8    1.4    1.9
```

```
'''
```

```
genexpr.columns      ## returns column names in an indexable object
```

```
genexpr.index        ## returns row names in an indexable object
```

```
genexpr.values        ## returns values in an indexable object
```

```
len(genexpr.index)    ## will be the number of rows
```

```
len(genexpr.columns)  ## will be the number of columns
```

```
## =====
```

```
## Accessing DataFrame by column
```

```
expr1_column = genexpr.expr1 = genexpr["expr1"] = genexpr[genexpr.columns[0]]  ## returns a Series
```

```
'''
```

```
>>> expr1_column
```

```
gene1    1.0
gene2    2.0
gene3    1.3
gene4    0.5
Name: expr1, dtype: float64
```

```
'''
```

```
## =====
```

```
## Accessing DataFrame by row
```

```
gene1_gene3 = genexpr[0:3:2]
```

```
'''
```

```
>>> gene1_gene3
```

```
      expr1  expr2  expr3  expr4
gene1    1.0    2.0    2.5    2.7
gene3    1.3    1.0    0.8    0.7
```

```

'''

## =====
## Sorting DataFrames

genexpr.sort_index(axis=1, ascending=False)    ## axis=1: columns; axis=0: rows
genexpr.sort_values(by="expr2")                ## sorts rows according to the value in col expr2

'''

>>> genexpr.sort_values(by="expr2")
      expr1  expr2  expr3  expr4
gene4    0.5    0.8    1.4    1.9
gene3    1.3    1.0    0.8    0.7
gene1    1.0    2.0    2.5    2.7
gene2    2.0    2.1    1.9    1.9
'''

```

For Mimi:

```

## =====
## Reading data from a file

## GEUVADIS.expr.cuttet.txt can be downloaded from my repo
## https://github.com/alussana/python-programming-alessandro-lussana/tree/master/pandas_presentation
data = pd.read_table("GEUVADIS.expr.cuttet.txt", header=0, index_col=0, sep="\t")

## This is a resized version of a real dataset (GEUVADIS, gene expression data from EBV-lymphoblastoid cell
## lines) downloaded from
## https://www.ebi.ac.uk/arrayexpress/files/E-GEUV-1/analysis\_results/

## header: the number of the row in which col names are specified
## index_col: the number of the col(s) in which the indexes are specified (multi index is allowed)
## sep: field separator

'''

>>> data.head()

                                Gene_Symbol  chr   Coord   HG000096   HG000097   ...
TargetID
ENSG00000162408.10_6589054_6589231  ENSG00000162408.10   1  6614595  83.834815  64.014944  ...
ENSG00000162408.10_6592028_6592139  ENSG00000162408.10   1  6614595  44.003241  39.387407  ...
ENSG00000162408.10_6592523_6592820  ENSG00000162408.10   1  6614595  88.731652  62.839503  ...
ENSG00000162408.10_6593340_6593501  ENSG00000162408.10   1  6614595  36.878233  25.444402  ...
ENSG00000162408.10_6601890_6601987  ENSG00000162408.10   1  6614595  26.020655  24.026659  ...
'''

## =====
## Useful methods and functions

data.describe()                                ## returns summary statistics for columns

## =====
## Deleting a column

data = data.drop("Gene_Symbol", 1)              ## 1 for columns names; 0 for indexes (specify the axis)

```

```

## =====
## Merging

## NA20828.txt can be downloaded from my repo:
## https://github.com/alussana/python-programming-alessandro-lussana/tree/master/pandas_presentation
new_sample = pd.read_table("NA20828.txt", sep="\t", header=0, index_col=0)

'''
>>> new_sample.head()

                                NA20828
TargetID
ENSG00000162714.7_247460714_247464578    1257.080064
ENSG00000162714.7_247471777_247471890     40.075274
ENSG00000162714.7_247473001_247473108     53.058649
ENSG00000162714.7_247473626_247473758     65.187342
ENSG00000162714.7_247486000_247486107      2.676004
'''

## It is not needed to do the following prior to merge:
data = data.sort_index()
new_sample = new_sample.sort_index()
## because pd.merge will auto align the data according to a col value (see on = "TargetID")

merged_data = pd.merge(data, new_sample, on = "TargetID")

## =====
## Writing data to a file

merged_data = merged_data.drop(["Coord", "chr"], 1)
merged_data.to_csv("merged_data.csv")

```

For Nico:

```

## =====
## Boolean Indexing

## that is, using columns values to select data

## selected data comes from before (see Mimi's section)
## this filters the records that refer to chr2 and with genomic coordinate > 1000000000
selected_data = data[data.chr == 2][data.Coord > 1000000000]

## =====
## Example of dummy analysis

## computing standard deviation of expr in the population for each gene
selected_data = selected_data.drop(["chr", "Coord"], 1)  ## delete these non useful fields (just keep data)
selected_data = selected_data.T                          ## transpose
expr_std = selected_data.std()                           ## std: no loop needed -> vectorize!

'''
>>> expr_std.head()

TargetID
ENSG00000162804.9_241976637_241976770    1.631483
ENSG00000162804.9_241979492_241979605    1.524546
'''

```

```

ENSG00000162804.9_241979717_241979830    1.732765
ENSG00000162804.9_241987732_241987857    1.896867
ENSG00000162804.9_241988079_241988183    1.367311
dtype: float64
'''

## Computing the difference between quantiles
expr_quant_10 = selected_data.quantile(q=0.1, axis=1)
expr_quant_90 = selected_data.quantile(q=0.9, axis=1)
expr_quant_range = expr_quant_90 - expr_quant_10
## Some parameters:
## q = quantile
## axis = 1: vectorize on columns; axis = 0: vectorize on rows

'''
>>> expr_quant_range.head()
HG00096    195.997076
HG00097    190.084365
HG00099    145.578205
HG00100    189.320729
HG00101    173.758388
dtype: float64
'''

## =====
## Plotting the results

## having scipy installed is also needed for this
import matplotlib.pyplot as plt          ## plt is needed since pd.plot is a wrapper of plt functions
std_plot = expr_std.plot.density()       ## create the object needed to plot the density
hist_plot = expr_quant_range.plot.hist() ## create the object needed to plot the histogram
plt.show()                              ## show the graph

selected_data2 = data['Coord'].isin(['9999999'])
selected_data2.head()
selected_data2 = selected_data.drop(["chr", "Coord"], 1)
selected_data2_mean = selected_data2.mean()
selected_data2_mean.plot(kind = 'bar')

```