Pandas module

* Using CSV files instead of XML, we can load and analyze them with many libraries. By converting to CSV, we can reduce the size of the dataset. Also, we can choose what we want to process and only convert those to CSV. For example, we don't want to process the body of questions in the stack overflow dataset. This helps with memory usage.
* **dusk** // is a library that can work with **large** datasets

## Site

* pandas.pydata.ort/docs/getting\_started/tutorials.html // one of the community documentation

## Jupiter notebook

* We can run python codes in the Jupyter notebook
* Vscode plugins: Jupyter, Jupyter keymap, Jupyter notebook renderers (visualizes the data)
* filename.ipynb //Create a file with extension

Panda basic commands:

* pip install pandas // in cmd
* Use the correct python version, and you may need to install pip install ipykernel.
* import pandas as pd
* Everything in pandas is a data frame. A frame is similar to a table. A table has a set of columns with headers and rows with values of any type.

## methods

* print( pd.\_\_version\_\_)

## Creating data frames:

* we can use a dictionary (or a CSV file)

// set of key/values pair. The headers will be the keys.

users\_dict = { "first\_name": [ "Derry", "Rose", "Dudu"],

"last\_name": ["Marry", "Chinto", "River"],

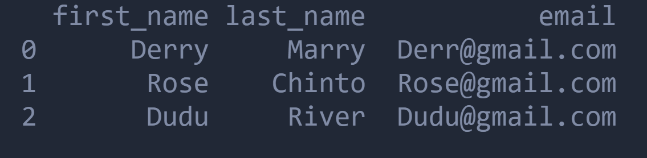
"email": ["Derr@gmail.com", "Rose@gmail.com", "Dudu@gmail.com"]

}

## Creating dataframes:

**dict**

* df\_users = pd.**DataFrame**(csv file or dict) // column by column. Pandas set the integer numbers on the left as indexes. They are unique identifiers for rows.



**list**

* [ [first\_row\_values], [second\_row\_values],…] // row by row

users\_list = [ ["Derry","Marry","Derr@gmail.com"],["Rose","Chinto"Rose@gmail.com",],["Dudu",""River","Rose@gmail.com"]]

* df\_users = pd.**DataFrame(list,columns=["first\_nam","last\_name","email"])**

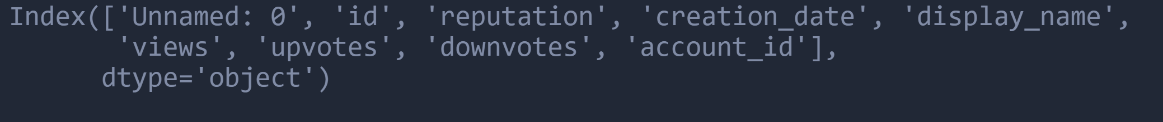
**load csv file**

* df\_users = pd.**read\_csv("file address")**

users = pd.read\_csv("C:/Users/Alma/Desktop/Courses Uni/Special Topis in SE/prof code works/stackoverflow\_bare\_code/csv\_files/users.csv")

print(pd.DataFrame(users))

* df\_users.**head(number)** // just a few entries, few rows. Number is optional
* df\_users.**columns** // The first unnamed is added when converting from xml to csv



* df\_users.**shapes[0 or o1]** // An attribute. the number of rows and columns
* df\_users.**dtypes** // the datatypes of columns. The date/string will appear as objects.
* df\_users.**info()** // More info about the data frame. How many rows, columns, non-null entries, and memory is used.
* pd.**set\_option('display.expand\_frame\_repr'.False)** // set options. This one will show the data frame in one row, don't break it into separate rows.

Data analysis

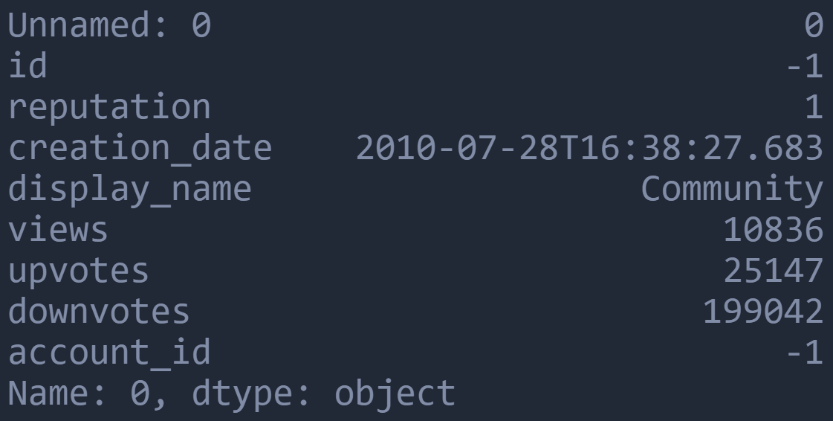
## Selection

* select row columns and attributes
* users**["columnName"]** // A column is of type of Panada Series. The data frame is made of a set of data series.

users[["id,"reputation", "creation\_date"]] // select multiple columns. The type of multiple columns here is a data frame. It returns a new object. The original object will no be affected.

* users.**id()** // This also selects id columns with it's values.
* users.**iloc**[] // integer locations. Selects rows based on those integer indexes that was add by panda.

user.iloc[0] // first row



user.iloc[0:2,2:3] // a range. Two rows(0,1), one column (2)

user.iloc[[2,3,4],[2,3]] // the second [2,3] is for columns. Columns start from 0

user.iloc[[2,3,4]] // rows 2,3,4

## Filter

* To analyze only specific items for example only answers in stack overflow dataset.
* users[column].**max**() or **min**() // the maximum of that column
* users[users['column'] > 1000] // use a filter

filter = users['reputation'] > 1000

print(filters) // says on what row this filter is False or True

users[filter] // apply filters. Gives all users that the rep is bigger than 1000

users[filter].shapes // the count

questions = posts[posts['post\_type\_id'] == 1] // post type id 1 means a question, 2 means answer

print(questions.head())

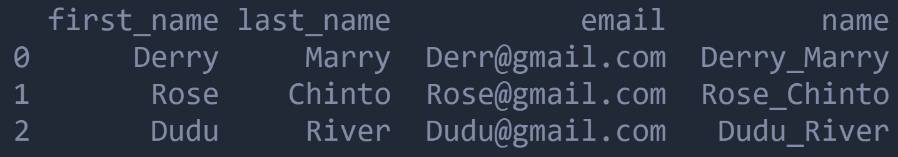
questions[questions['tags'].str.contains('<power-management>')].head() // str is to process column as string. All the questions that have the given tag

Drop and add columns

* user.**drop**(['id',reputation'], axis = 1) // axis = 0 means column axis =1 means rows
* users['new column name"] = new row value // to add rows and columns. Everything in pandas is a vectorized operation, meaning you write for one row it is applied to all and do it to all colum value that is applied. Panda fast because it parallelizes this.

// example for the data frame we made with dict

users['name'] = users['first\_name'] + "\_" users['last\_name']



## Aggregation

* dataframe.**groupby**() // The aggregation operation is performed by group by. For example, group by post\_type\_id to see how many rows each category is. Group by basically divides the entire data frame into categories(groups) which are also data frames based on a specific column.
* To know the number of questions for every type. one approach is to write a filter based on types. But it takes a lot of time.

set(users['post\_type\_id']) // to see what are the post type ids. Returns {1,2,3,4,5,6,7}

post\_groups = posts.groupby('posts\_type\_id') // returns a dict which the key is the post type id and the value a dataframe of posts with that id.

post\_groups\_dict = dict(list(post\_group))

print(post\_groups\_dict.keys()) // returns dict\_keys([1,2,3,4,5,6,7])

* post\_groups.**get\_group**(keyvalus) // getting individual groups

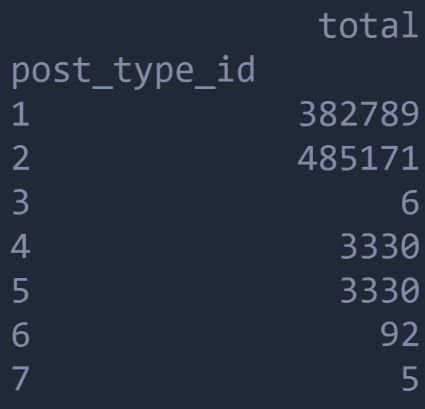
print(post\_groups.get\_group(2)) // all the answers that don't have tags

* posts.groupby(column).**agg**(nameofnewcolumn = pd.**NameAgg**('which column to perform agg', **aggfun**=len) // By grouping we can perform aggregation operations on every group. Len functions says what is the length of column (series). The example below says for each data frame associated with a post\_type\_id, it'll select post\_type\_id and applies len function which mean gives how much elements we have with that post\_type\_id.

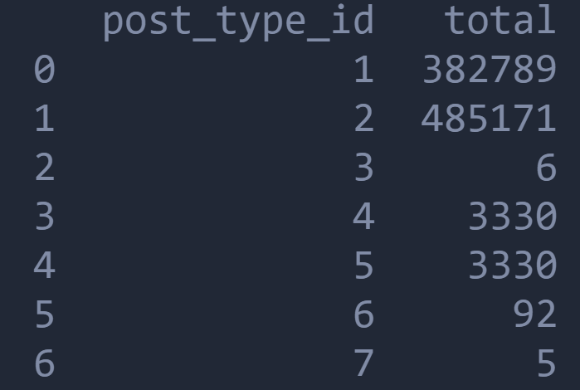
postId\_len\_result = posts.groupby('score').agg(total = pd.NamedAgg('post\_type\_id', aggfunc=len))

postId\_len\_result.reset\_index(inplace=True)

print(postId\_len\_result)



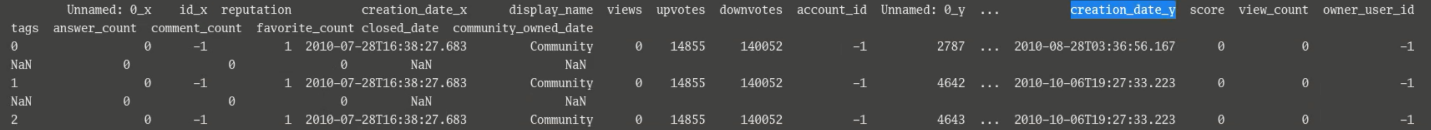
* postId\_len\_result.**reset\_index**(inplace=True) // in any pandas method there is an attribute called inpalce = True, which means it changes the result, it will not create a new data frame.



## Joining

* Join two different tables(data frames)
* For example to know what users have how many question answers
* pd.**merge**(**left**="fromTableName", **right**="toTableName", **how** ="inner", **left\_on**="fromColumn", **right\_on**=" toColumn ") // merges two tables and filter based on column. For join you should select what id represents what id in the other table and join(match) based on them.

pd.merge(left="users", right="posts", how ="inner", left\_on="id", right\_on="owner\_user\_id") // match the id of the user in user table to user id in post table.

* x means the user table, y means the post table
* now we can use aggregate to see how many posts a user have

pd.merge(left=df\_users,right=df\_posts, how="inner",left\_on='id',right\_on='owner\_user\_id').groupby('id\_x').agg(total = pd.NamedAgg('reputation',aggfunc=len))

