**Pandas**

**Introduction to Pandas**

Data processing is important part of analyzing the data, because data is not always available in desired format. Various processing are required before analyzing the data such as cleaning, restructuring or merging etc. Numpy, Scipy, Cython and Panda are the tools available in python which can be used fast processing of the data. Further, Pandas are built on the top of Numpy.

Pandas provides rich set of functions to process various types of data. Further, working with Panda is fast, easy and more expressive than other tools. Pandas provides fast data processing as Numpy along with flexible data manipulation techniques as spreadsheets and relational databases. Lastly, pandas integrates well with matplotlib library, which makes it very handy tool for analyzing the data.



Note:

1. In chapter 1, two important data structures i.e. Series and DataFrame are discussed.
2. Chapter 2 shows the frequently used features of Pandas with example. And later chapters include various other information about Pandas.



**Data structures**

Pandas provides two very useful data structures to process the data.

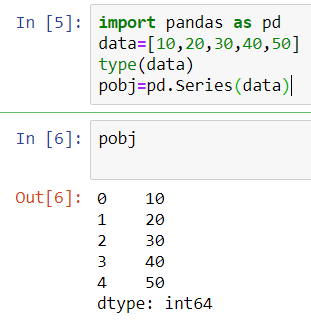
* Series
* DataFrame

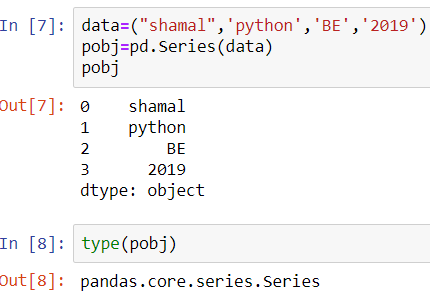
**Series**

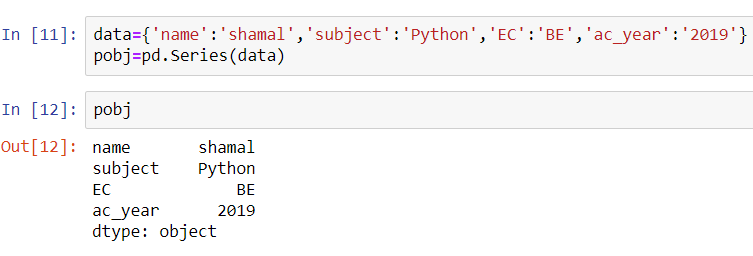
The Series is a one-dimensional array that can store various data types, including mix data types.

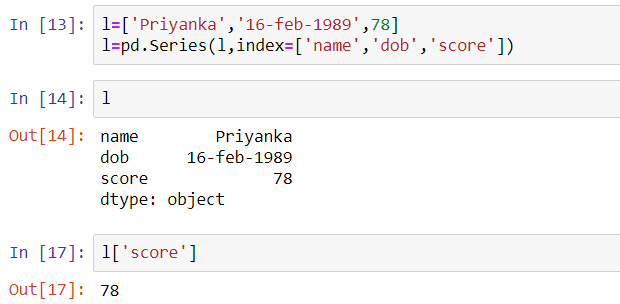
The row labels in a Series are called the index.

Any list, tuple and dictionary can be converted in to Series using ’series’ method as shown below.









**DataFrame**

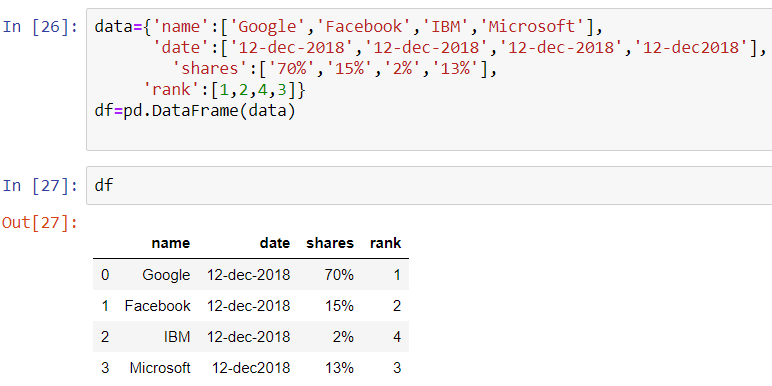
DataFrame is the widely used data structure of pandas.

Note that, Series are used to work with one dimensional array, whereas DataFrame can be used with two dimensional arrays.

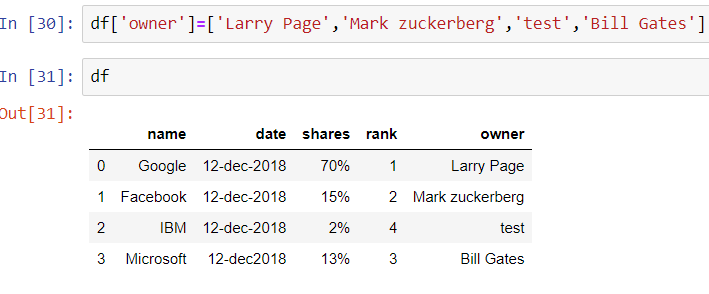
DataFrame has two different index i.e. column-index and row-index.

The most common way to create a DataFrame is by using the dictionary of equal-length list as shown below.

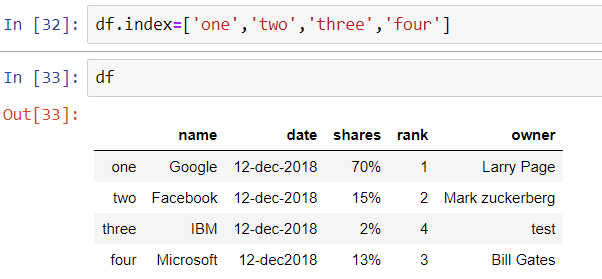
Further, all the spreadsheets and text files are read as DataFrame, therefore it is very important data structure of pandas.



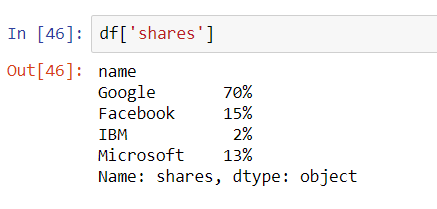
**Adding new Column**



**Adding new Index**

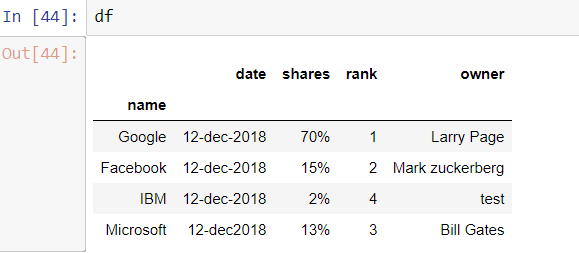


**Fetch Column by Name**



Set Index From Column which contains unique values

df.set\_index(['name'])



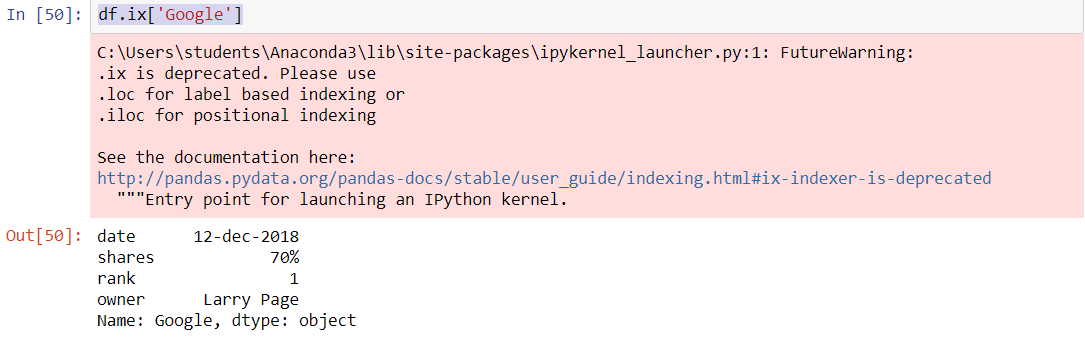
**Fetch Records From Index**

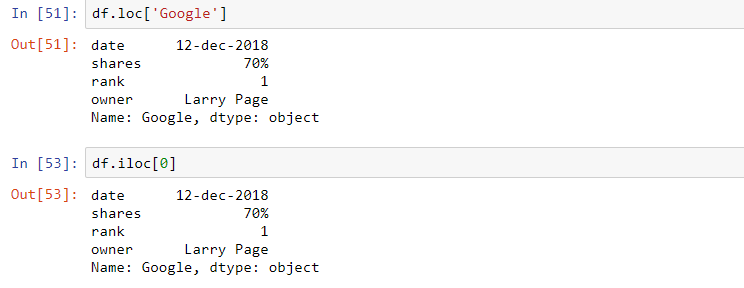
**# ix() is deprecated**

**Dataframe.ix[‘column name’]**

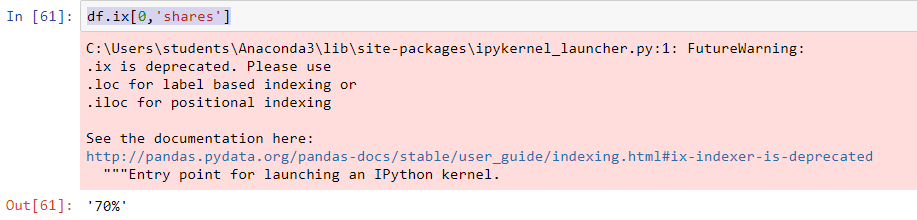
**DataFrame.loc[‘columnname’]**

**DataFrame.iloc[‘columnid’]**

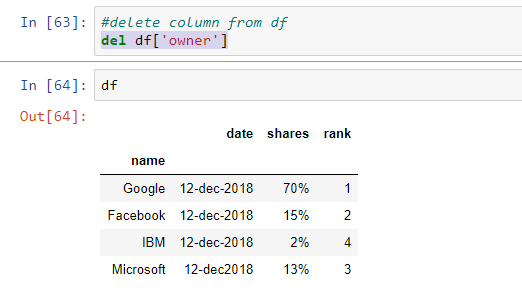


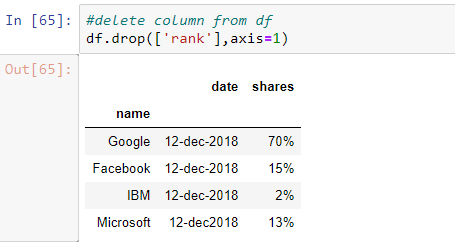


**Get Records from rows & column**

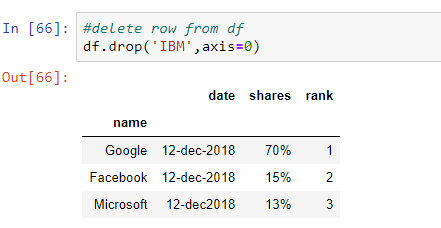


**Deleting a Column**

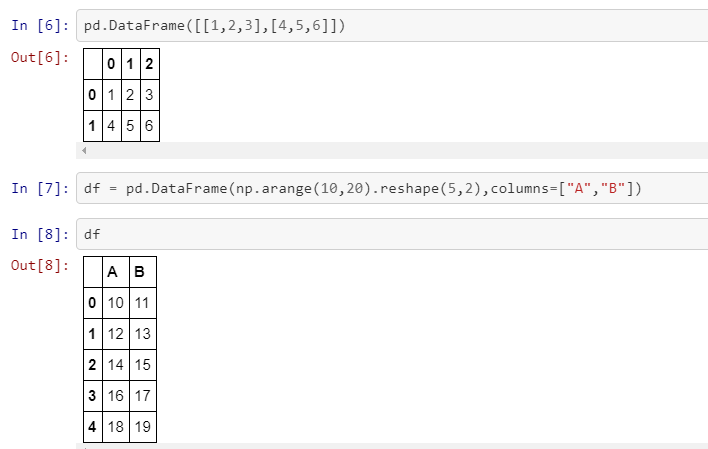




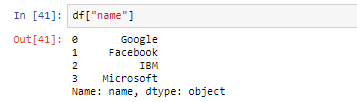
**Deleting Rows**

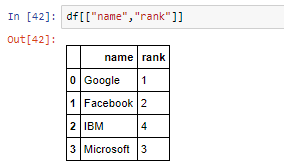


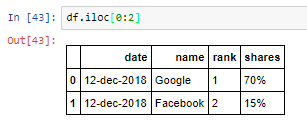
**Reshape Data**



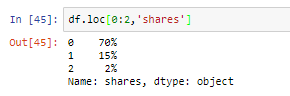
# **Indexing and Slicing**









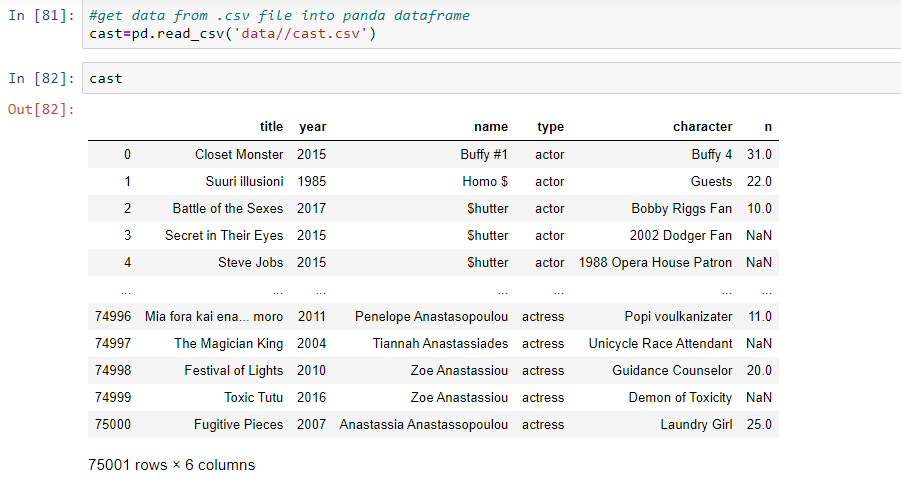




**Reading files**

In this section, two data files are used i.e. ‘titles.csv’ and ‘cast.csv’.

The ‘titles.csv’ file contains the list of movies with the releasing year; whereas ‘cast.csv’ file has five columns which store the title of movie, releasing year, star-casts, type(actor/actress), characters and ratings for actors, as shown below

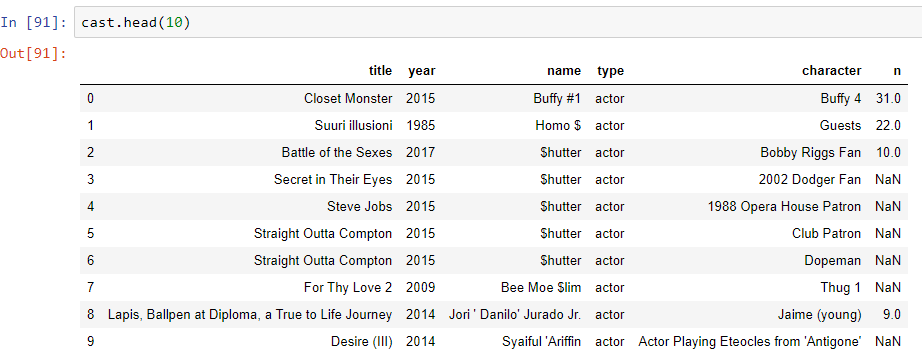


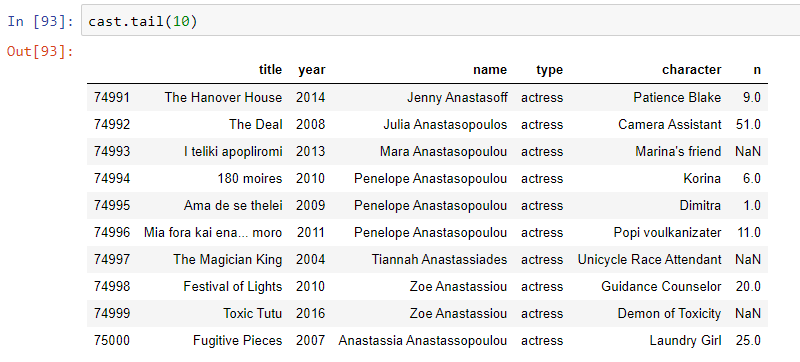
1. **read\_csv :** load the data from the csv file.
2. **index\_col** = None : there is no index i.e. first column is data
3. **head() :** show only first five elements of the DataFrame
4. **tail() :** show only last five elements of the DataFrame

If there is some error while reading the file due to encoding, then try for following option as well,

**Note**: head() and tail() commands can be used for remind ourselves about the header and contents of the file.

These two commands will show the first and last 5 lines respectively of the file. Further, we can change the total number of lines to be displayed by these commands,

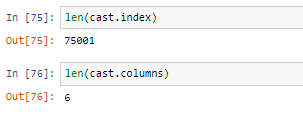




**Shape** :-

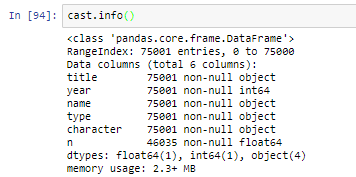
Used to check datafram shape





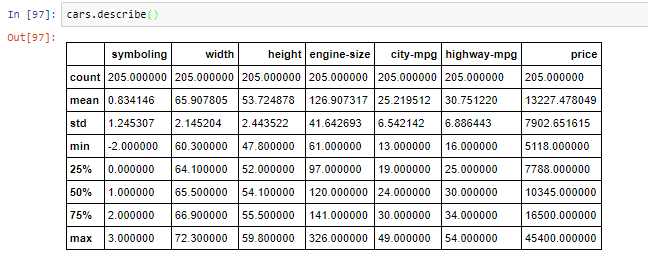
**Info:**

* gives information about data



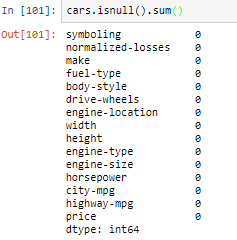
**Describe:**

* describe data in detail with
  + count
  + mean
  + sdv
  + min
  + max
  + 25%
  + 75%
  + 50%



**ISNull():**

* Isnull() check null values and sum() will perform addition of it in respective column



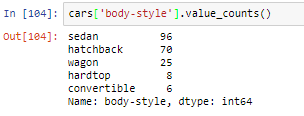
**Unique()**

Give unique value from column

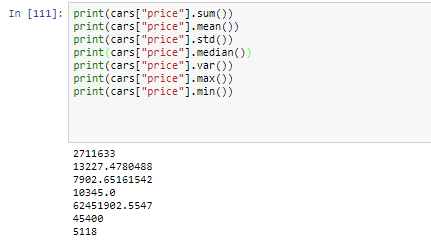


**Value\_counts()**

* Will count how many time that value is repeat in data



**Mathematical Operations:-**



**Data operations**

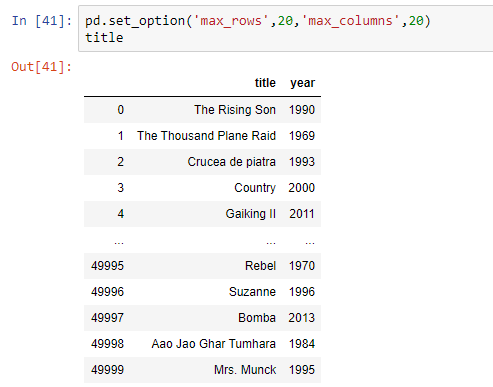
**Row and column selection**



**If we simply type the name of the DataFrame (i.e. cast in below code), then it will show the first thirty and last twenty rows of the file along with complete list of columns.**

**This can be limited using ‘set\_options’ as below.**

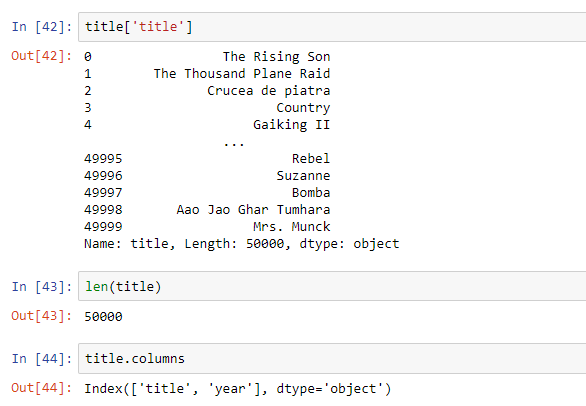
**Further, at the end of the table total number of rows and columns will be displayed.**



Access individual Column = df[“columnname”]

Len of data len() = len(df)

See list of columns by column attribute = df.columns



**Note:**

head() and tail() commands can be used for remind ourselves about the header and contents of the file.

These two commands will show the first and last 5 lines respectively of the file.

Further, we can change the total number of lines to be displayed by these commands,

**Filter Data**

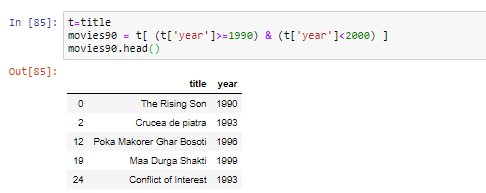
Data can be filtered by providing some boolean expression in DataFrame.

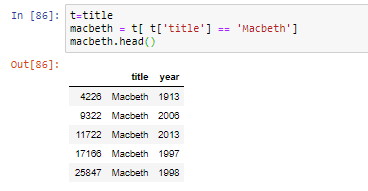
For example, in below code, movies which released after 1985 are filtered out from the DataFrame ’titles’ and stored in a new DataFrame i.e. after85.



**Note:**

* When we pass the boolean results to DataFrame, then panda will show all the results which corresponds to True (rather than displaying True and False),
* as shown in above code. Further, ‘& (and)’ and ‘| (or)’ can be used for joining two conditions as shown below,\*\*
* In below code all the movies in decade 1990 (i.e. 1900-1999) are selected. Also ’t = titles’ is used for simplicity purpose only.

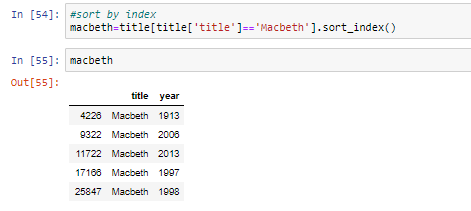


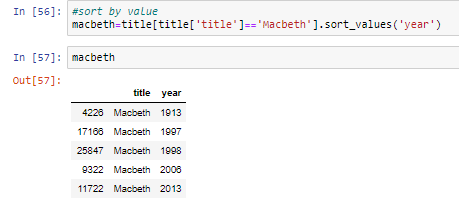


**Sorting**

Sorting can be performed using

* Df.sort\_index()
* Df.Sort\_values()



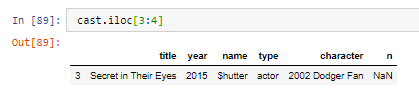


**Note**

* That in above filtering operation, the data is sorted by index and values
* By default ’sort\_index’ operation is used

**Null values**

* Note that, various columns may contains no values, which are usually filled as NaN.
* For example, rows 3-4 of casts are NaN as shown below,

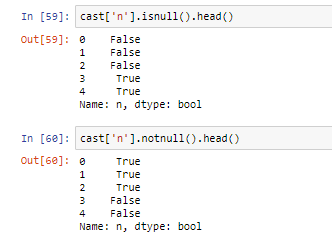


These null values can be easily selected, unselected or contents can be replaced by any other values e.g. empty strings or 0 etc.

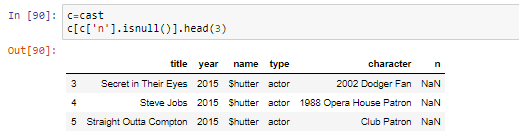
Various examples of null values are shown in this section.

* ’isnull’ command returns the true value if any row of has null values.
* ’notnull’ is opposite of isnull, it returns true for not null values,

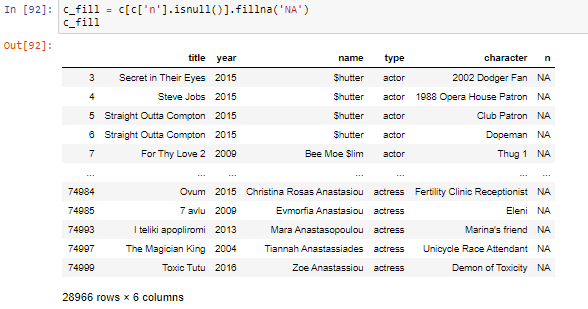
Since the rows 3-4 has NaN value, therefore, these are displayed as True.



* To display the rows with null values, the condition must be passed in the DataFrame,

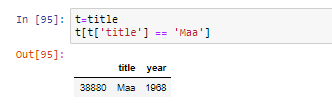


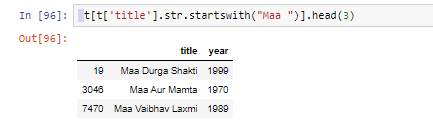
* NaN values can be fill by using fillna, ffill(forward fill), and bfill(backward fill) etc. In below code, ’NaN’ values are replace by NA. Further, example of ffill and bfill are shown in later part of the tutorial,



**String operations**

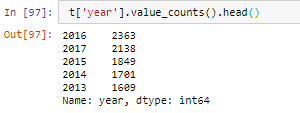
* Various string operations can be performed using ’.str.’ option. Let’s search for the movie "Maa" first





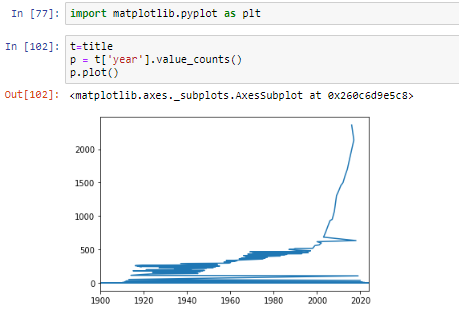
**Count Values(unsorted groupby)**

* Total number of occurrences can be counted using ’value\_counts()’ option. In following code, total number of movies are displayed base on years.

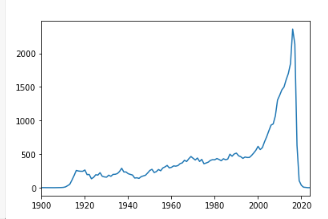


**Plots**

* Pandas supports the matplotlib library and can be used to plot the data as well.
* In previous section, the total numbers of movies/year were filtered out from the DataFrame. In the below code, those values are saved in new DataFrame and then plotted using panda.

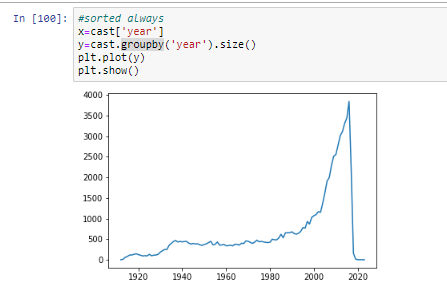




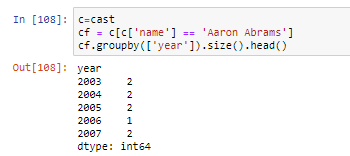


**Groupby**

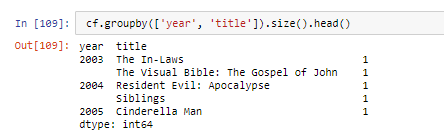
* Data can be grouped by columns-headers. Further, custom formats can be defined to group the various elements of the DataFrame.
* **Groupby with column-names**
* In Section Count Values, the value of movies/year were counted using ’count\_values()’ method.
* Same can be achieve by ’groupby’ method as well.
* The ’groupby’ command return an object, and we need to an additional functionality to it to get some results.
* For example, in below code, data is grouped by ’year’ and then size() command is used.
* The size() option counts the total number for rows for each year; therefore the result of below code is same as ’count\_values()’ command.



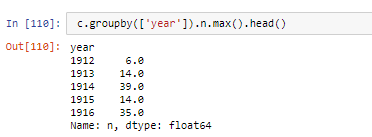
* Further, groupby option can take multiple parameters for grouping. For example, we want to group the movies of the actor ’Aaron Abrams’ based on year,



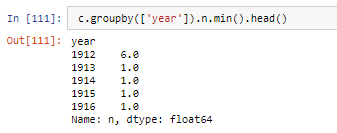
* Above list shows that year-2003 is found in two rows with name-entry as ’Aaron Abrams’. In the other word, he did 2 movies in 2003.
* Next, we want to see the list of movies as well, then we can pass two parameters in the list as shown below,



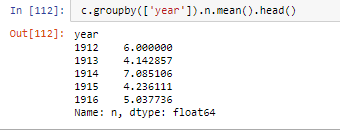
* In above code, the groupby operation is performed on the ’year’ first and then on ’title’.
* In the other word, first all the movies are grouped by year.
* After that, the result of this groupby is again grouped based on titles.
* Note that, first group command arranged the year in order i.e. 2003, 2004 and 2005 etc.;
* then next group command arranged the title in alphabetical order.
* Next, we want to do grouping based on maximum ratings in a year; i.e. we want to group the items by year and see the maximum rating in those years,



* Above results show that the maximum rating in year 1912 is 6 for Aaron Abrams. • Similarly, we can check for the minimum rating,

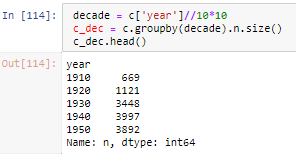


* Lastly, we want to check the mean rating each year,



**Groupby with custom field**

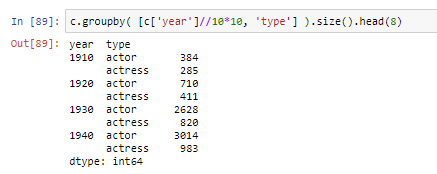
* Suppose we want to group the data based on decades, then we need to create a custom groupby field,



* Above results shows the total number of movies in each decade.

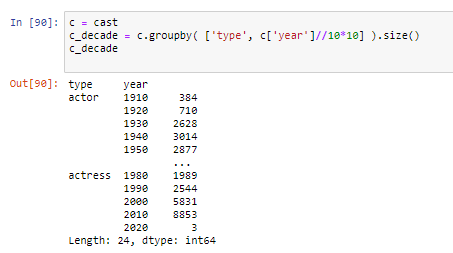
**Unstack**

* Before understanding the unstack, let’s consider one case from cast.csv file.
* In following code, the data is grouped by decade and type i.e. actor and actress.

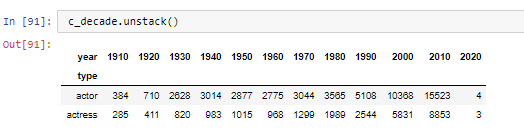


**Note**: Unstack is discussed in Section Unstack the data in detail.

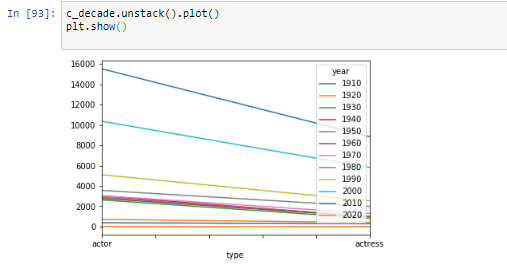
* Now we want to compare and plot the total number of actors and actresses in each decade.
* One solution to this problem is to grab even and odd rows separately and plot the data, which is quite complicated operation if types has more varieties e.g. new-actor, new-actress and teen-actors etc.
* A simple solution to such problem is the ’unstack’, which allows to create a new DataFrame based on the grouped Dataframe, as shown below.
* Since we want a plot based on actors and actress, therefore first we need to group the data based on ’type’ as below,

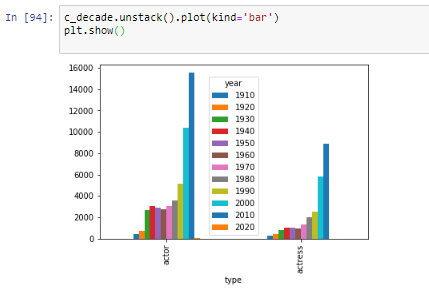


* Now we can create a new DataFrame using ’unstack’ command. The ’unstack’ command creates a new DataFrame based on index,

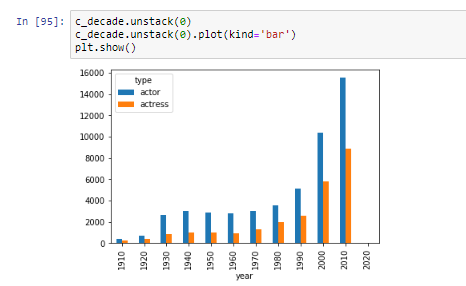


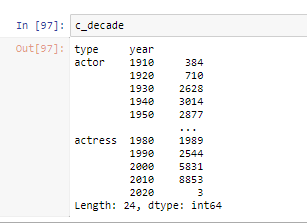
* Use following commands to plot the above data,





* Below figure will be generated from above command. Note that in the plot, actor and actress are plot separately in the groups.
* To plot the data side by side, use unstack(0) option as shown below (by default unstack(-1) is used),



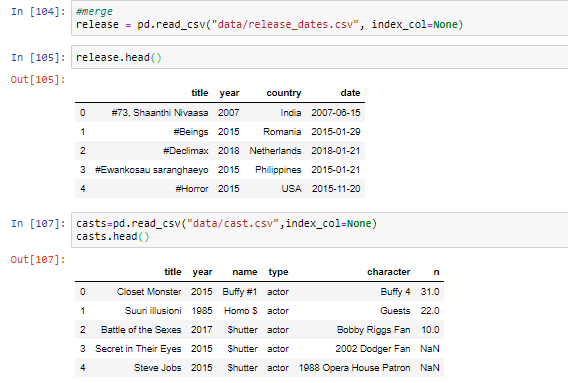


**Merge**

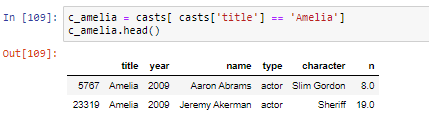
* Usually, different data of same project are available in various files.
* To get the useful information from these files, we need to combine these files.
* Also, we need to merge to different data in the same file to get some specific information.
* In this section, we will understand these two merges i.e. merge with different file and merge with same file.

**Merge with different files**

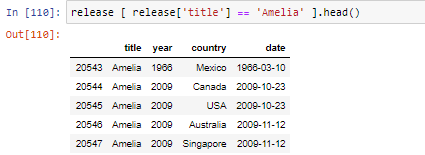
* In this section, we will merge the data of two table i.e. ’release\_dates.csv’ and ’cast.csv’.
* The ’release\_dates.csv’ file contains the release date of movies in different countries.
* First, load the ’release\_dates.csv’ file, which contains the release dates of some of the movies, listed in ’cast.csv’.
* Following are the content of ’release\_dates.csv’ file,



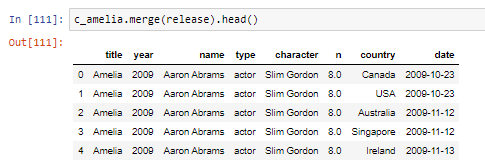
* Let’s we want to see the release date of the movie ’Amelia’. For this first, filter out the Amelia from the DataFrame ’cast’ as below. There are only two entries for the movie Amelia.



* Next, we will see the entries of movie ’Amelia’ in release dates as below. In the below result, we can see that there are two different release years for the movie i.e. 1966 and 2009.

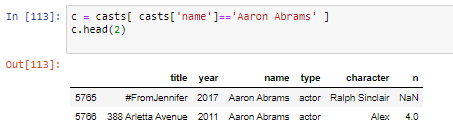


* Since there is not entry for Amelia-1966 in casts DataFrame, therefore merge command will not merge the Amelia-1966 release dates. In following results, we can see that only Amelia 2009 release dates are merges with casts DataFrame.

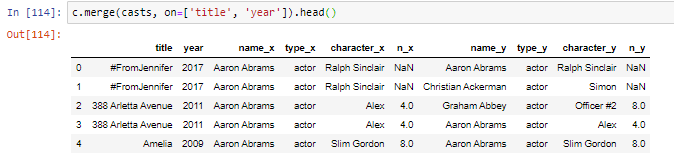


**Merge table with itself**

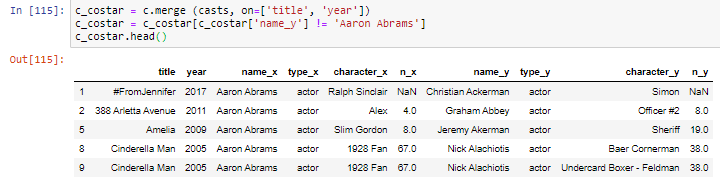
* Suppose, we want see the list of co-actors in the movies.
* For this, we need to merge the table with itself based on the title and year, as shown below.
* In the below code, co-star for actor ’Aaron Abrams’ are displayed,
* First, filter out the results for ’Aaron Abrams’

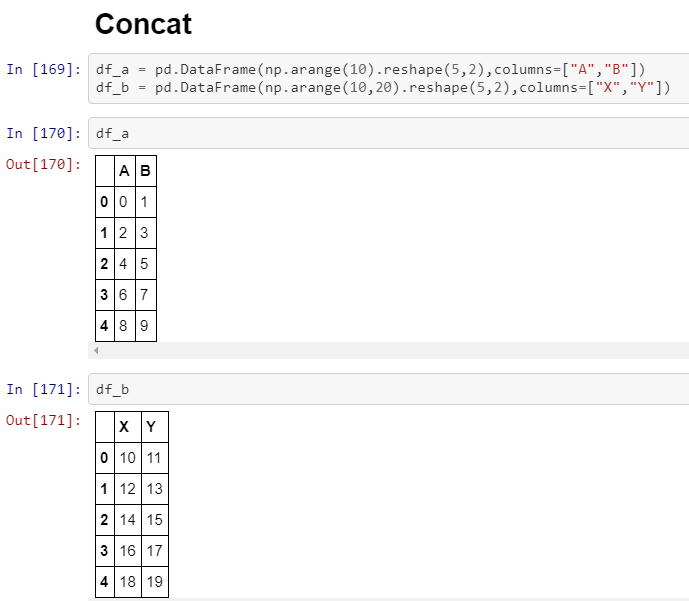


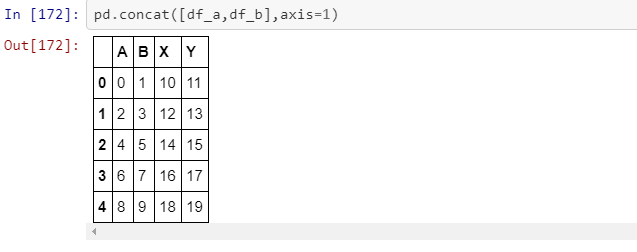
* Next, to find the co-stars, merge the DataFrame with itself based on ’title’ and ’year’ i.e. for being a co-star, the name of the movie and the year must be same
* Note that ’casts’ is used inside the bracket instead of c.



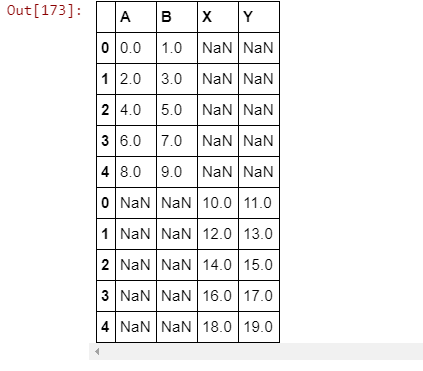
* The problem with above joining is that it displays the ’Aaron Abrams’ as his co-actor as well (see first row).
* This problem can be avoided as below





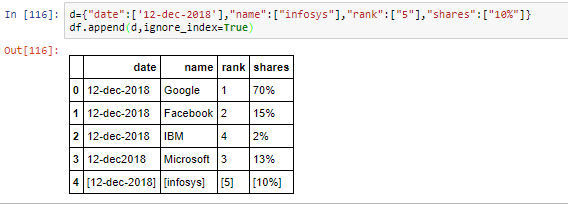






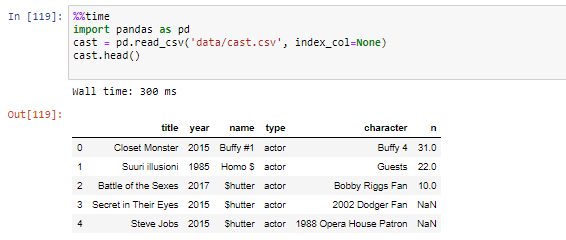
**Append**

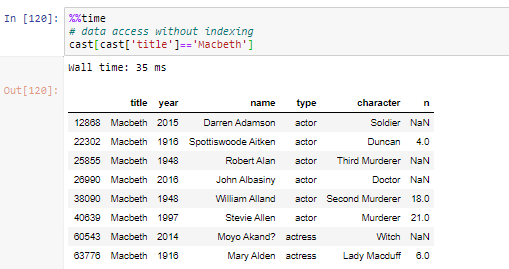
* Append data into df using dictionary



**Creating index**

* In the previous section, we saw some uses of index for sorting and plotting the data.
* In this section, index are discussed in detail. Index is very important tool in pandas.
* It is used to organize the data and to provide us fast access to data.
* In this section, time for data-access are compared for the data with and without indexing.
* For this section, Jupyter notebook is used as ’%%timeit’ is very easy to use in it to compare the time required for various access-operations





* ’%%timeit’ can be used for more precise results as it run the shell various times and display the average time; but it will not show the output of the shell,

