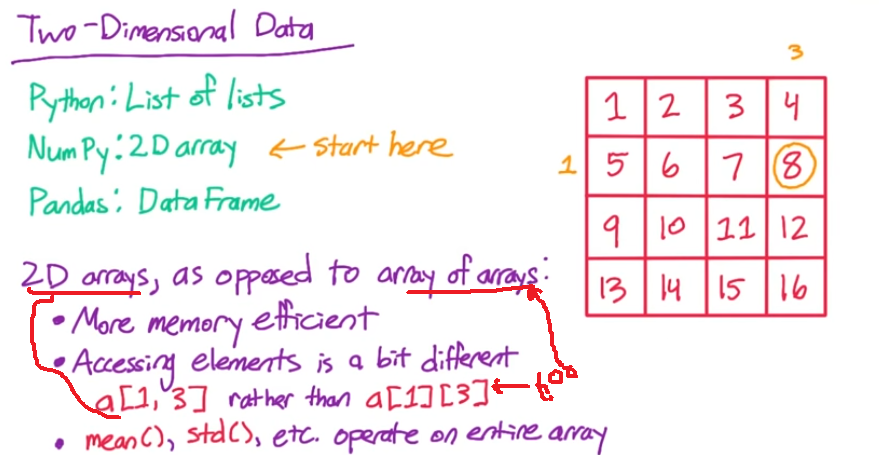
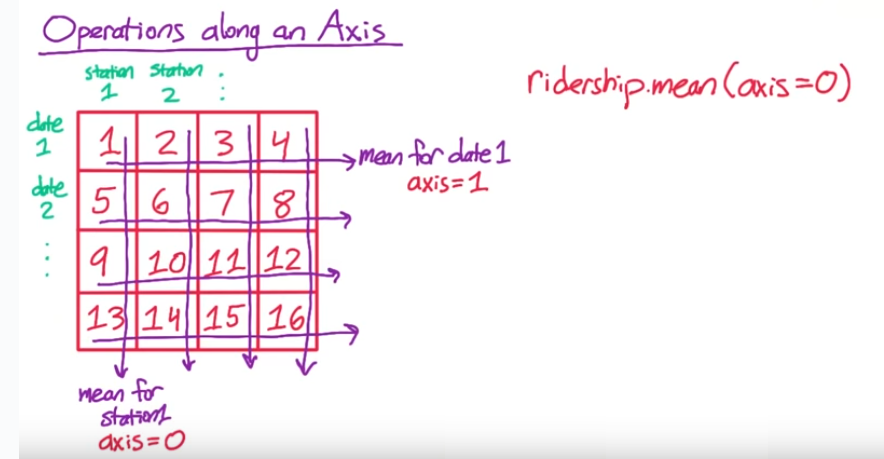
1.

In array of arrays-

Accessing elements = a[1][3]

Mean(), std() operate on each individual array and not on entire array, unlike in 2D array.

2. Axis operations

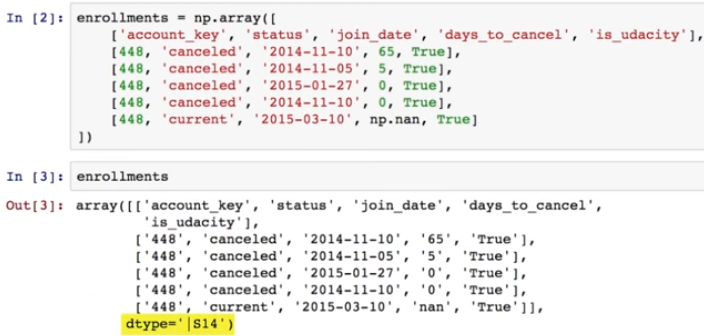


3. pandas dataframe-

-why needed? When we read a csv file, some columns are numeric and some are characters.

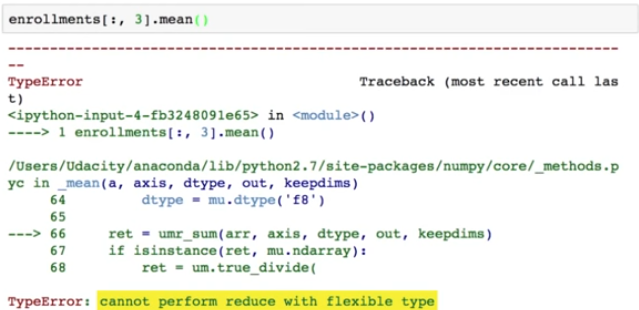
If these values are read in numpy 2d array, they all will be read as string(including columns with numeric values), and so numpy numeric operations on those columns with numeric values will give an error.

This is coz in an array, all elements are of SAME DATA TYPE.



As seen above, even Boolean and integer values are converted to STRING, and dtype = **string 14**

So taking mean of column 3(days\_to\_cancel) gives an error.



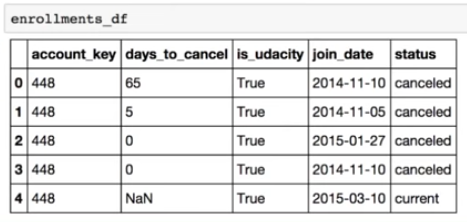
In contrast, DataFrame is also a 2D structure, but each column is assumed to be of different data type.

Similar to Series, DataFrames have indexes

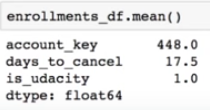
One way to create pandas dataframe is to pass dictionary, where keys are columns names, and corresponding values are passed as **list**



On displaying the contents of this dataframe, it is displayed as a nice table.



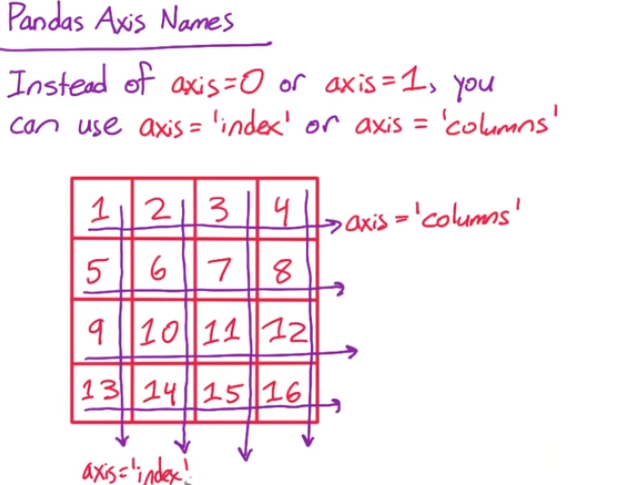
If we take mean() of dataframe, it takes mean only of numerical column and ignores the others.



**Enrollment\_df.mean(axis=1) #will let us take mean of each row**

But in above case, this won’t work as row contains non-numeric data as well

**Axis in pandas dataframe**

****

**Summary of above pic-**

**To apply fn along ->**

* **Rows: axis=1**
* **Columns : axis=0**

**Applymap() – to apply a user defined function on each element of DATAFRAME(not Series).**

**Apply() – can be applied on dataframe such that –**

* **It applies to each COLUMN, where each column is treated as individual series, and returns a Series each time, so as to modify the DATAFRAME.**
* **It can also be applied to each COLUMN of a dataframe such that it for each column, it returns a single object, and final returned value is a SERIES**

**3D data in NumPy**

NumPy arrays can have arbitrarily many dimensions. Just like you can create a 1D array from a list, and a 2D array from a list of lists, you can create a 3D array from a list of lists of lists, and so on. For example, the following code would create a 3D array:

**a** = np.array([

[['A1a', 'A1b', 'A1c'], ['A2a', 'A2b', 'A2c']],

[['B1a', 'B1b', 'B1c'], ['B2a', 'B2b', 'B2c']]

])

**3D data in Pandas**

Pandas has a data structure called a Panel, which is similar to a DataFrame or a Series, but for 3D data.