**Course-1/Week-3**

**Merging:**

Pandas has full-featured, high performance in memory join operations idiomatically very similar to relational database like SQL.

Pandas provides a single function, merge as the entry point for all standard database join operations between Dataframe objects



Here, we have used the following parameters

· **Left –** A DataFrame object

· **Right –** Another DataFrame object

· **On –** Columns to join on. Must be found in both the left and right dataframes objects.

· **Left\_on-** Columns from the left dataframe to use as keys. Can either be column names or arrays with length equal to the length of the dataframe.

· **Right\_on-** Columns from the right dataframe to use as keys. Can either be column names or arrays with length equal to the length of the dataframe.

· **Left\_index-** if true, use index from the left dataframe as its join key. In case of dataframe with a multiindex, the number of levels must match the number of join keys from the right dataframe.

· **Right\_index-** Same usage as left\_index for the right dataframe

· **How-** One of ‘left’, ’right’, ’outer’, ’inner’. Defaults to inner. Each method has been described below.

· **Sort-** sort the result dataframe by the join keys in lexicographical order. Defaults to ture, setting to false will improve the performance substantially in many cases.

**Example:**

Let us create two different dataframes and perform the merging operations on it.



Adding index values to DataFrame:



Add to new column:

Here we have to add new data to dataframe with the name of column is date.



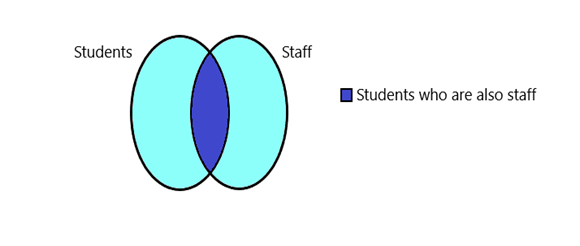
Adding same data to all columns at a time with new column i.e.,



Adding index:



Venn diagram:



Now we have to creating two DataFrames i.e., Staff and Students.

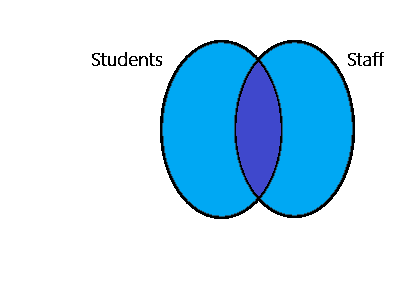
Staff DataFrame:

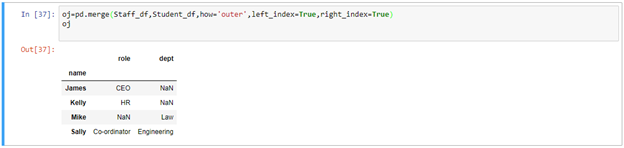


Student DataFrame:

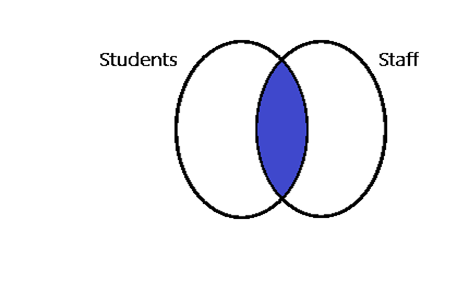


Outer join:





Inner join:

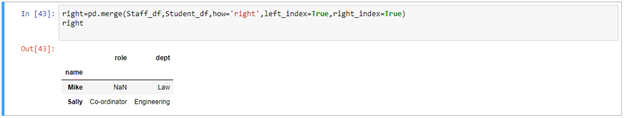




Left:



Right:



Index:

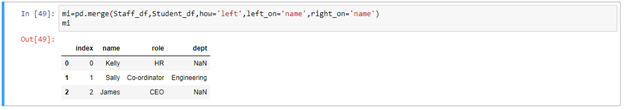
Staff\_df:



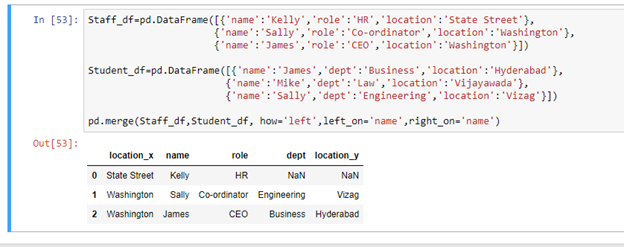
Student\_df:

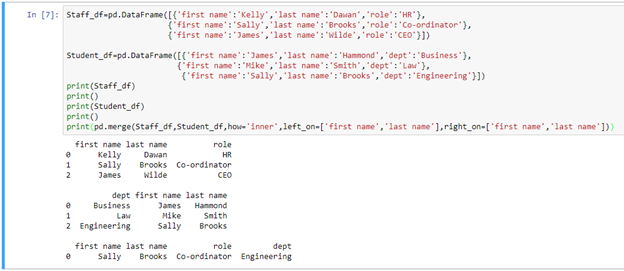


Left\_on & Right\_on:



Add Field with left join:



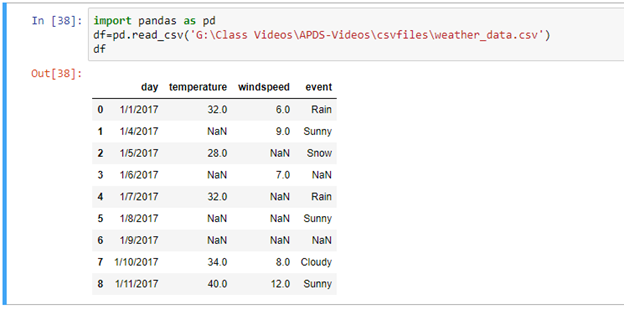


Idiomatic Pandas: Making Code Pandorable:

CSV files:

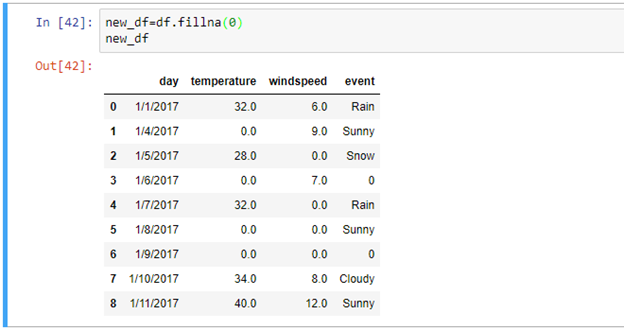
<https://drive.google.com/drive/folders/11bxcQ3ZwbsI_n8CBMe4Wnf1h3hUOkTEt?usp=sharing>

how to getting CSV file through pandas data in table format

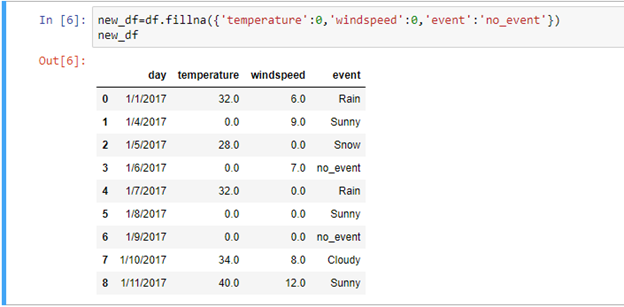


Fillna:

Pandas provides various methods for cleaning the missing values. The fillna function can fill in NA value with non-null data in a couple of ways, which we have illustrated in the following sections



Another way of fillna declarations using dictionary



Ffill-method:

“ffill”-method means forward fillna. Which is used to calculating mean value 0th index with 1st index values. This method is only applicable for null or zero values.

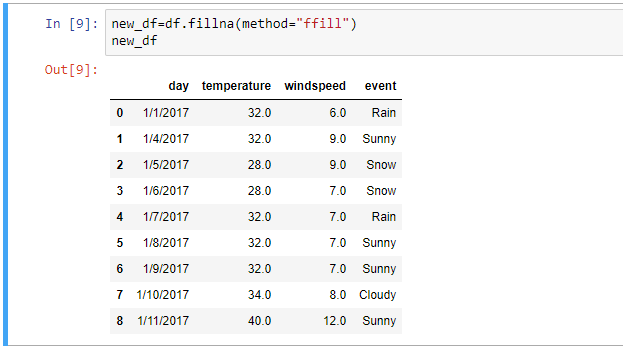
Example:

0th = 32 and 1st = 0

32+0=32

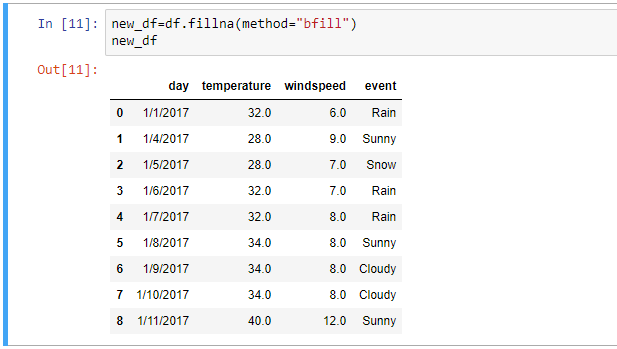
Now result is,

0th = 32 and 1st = 32



Bfill-method:

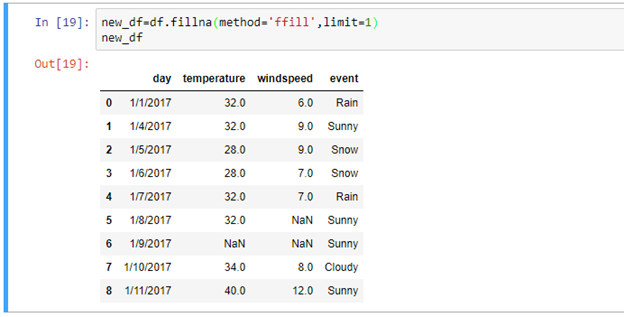
“bfill”-method means backward fillna. Which is used to calculating mean value 1st index with 0th index values. This method is only applicable for null or zero values.



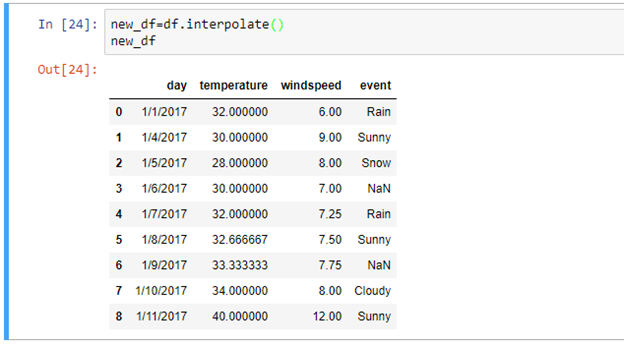
Limit:

Before using this method first we have to observer fillna modified table, in that 4th index to 6th index value 4th index value is 32.0 and 5th, 6th index value is NaN. Data will have continues with null value that cases only this method was applicable.

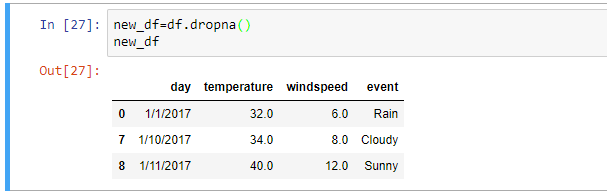
In this below example we are considering method and limit. In the method we declare ffill, and limit is 1.



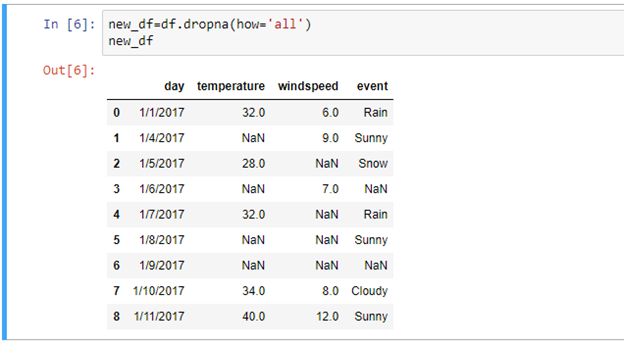
Interpolate:



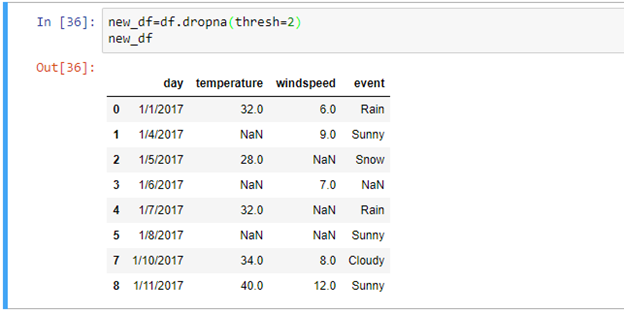
Dropna:



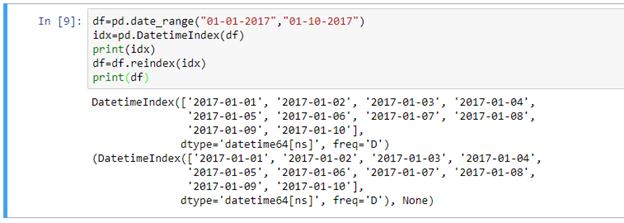
Dropna(how=’all’):



Dropna(thresh=2)



Adding date:



Group by:

Any groupby operation involves one of the following operations on the original object. They are,

· 1. Splitting the object

· 2. Applying a function

· 3. Combining the results

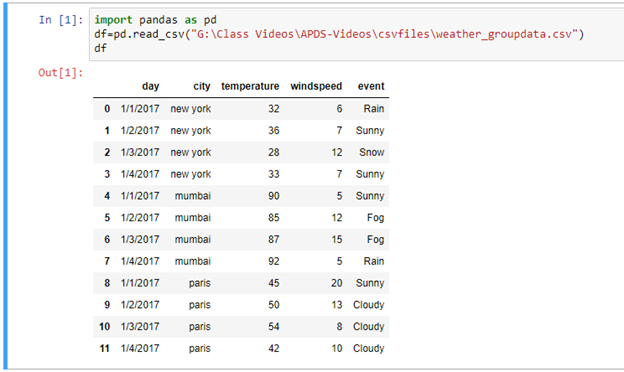
In many situations, we split the data into sets we apply some functionality on each subset. In the apply functionality, we can perform the following operations.

· 1. Aggregation computing a summary statistic

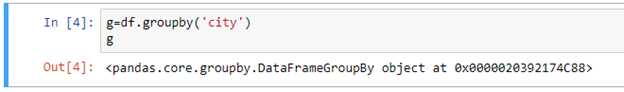
· 2. Transformation perform some group specific operation

· 3. Filtration discarding the data with some condition

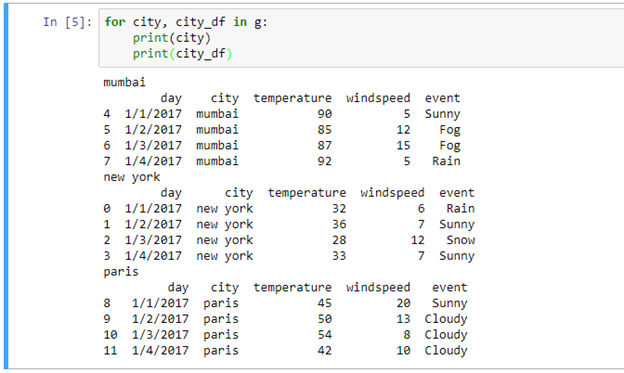
Let us one CSV file with the name of weather\_groupdata.csv and perform all the operations on it



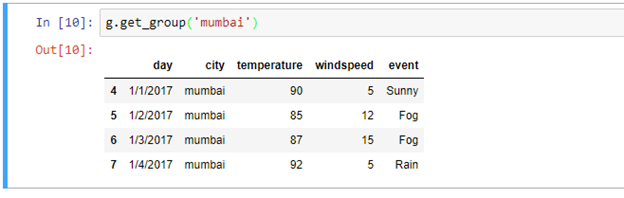
Finding particular fields from csv file



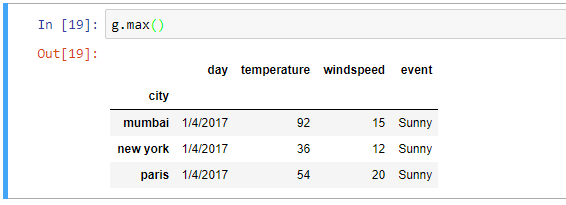
In this above output, it shows dataframe object value only but not showing number of cities from csv file. Why because we are finding group of cities in the list it will not take only single value in this case we should keep in loop then only it will displaying



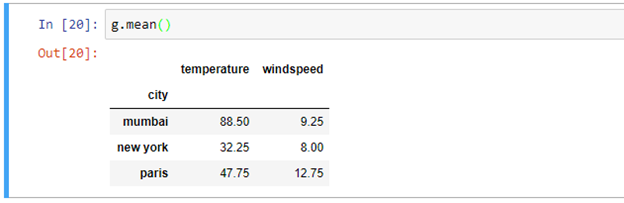
Now we want to display the specific data of particular city



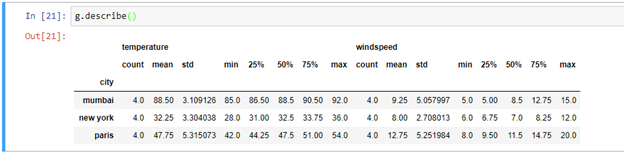
Now getting maximum temperature, wind speed and event from each city



Now we findout mean value of temperature and windspeed in each cities



Now complete description of temperature and windspeed in each cities



Making the first plot:

Now that we have the preliminaries out the way, we are ready to draw our first plot using matplotlib. In matplotlib drawing a plot involves:

· Creating a figure to draw plots into

· Creating one or more axes objects to draw the plots.

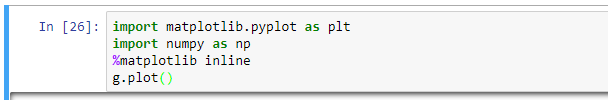
· Showing the figure and any plots inside, as an image

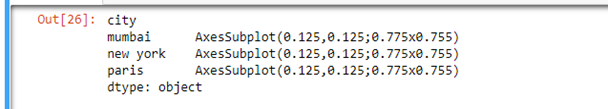
Because of its flexible structure, you can draw multiple plots into a single image in matplotlib. Each axes object represents a single plot, like a bar plot or a histogram.

This may sound complicated, but matplotlob has convince methods that do all the work of setting up a figure and axes object for us.

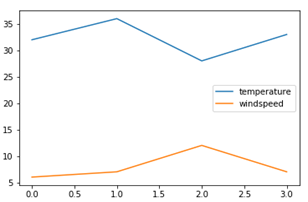
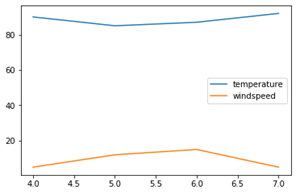
Importing matplotlib:

In order to use matplotlib, you will need to first import the library using import matplotlib.pyplot as plt. If oyu are using Jupyter notebook, you can setup matplotlib to work inside the notebook using %matplotlib inline

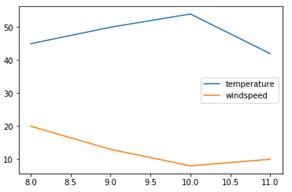




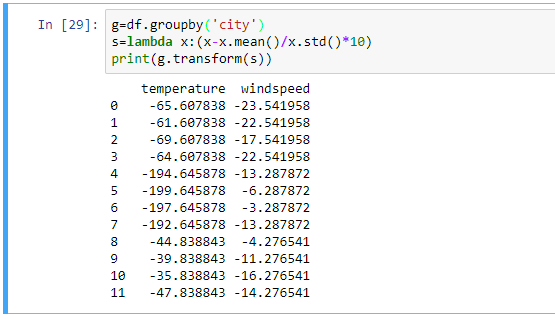
Mumbai New York



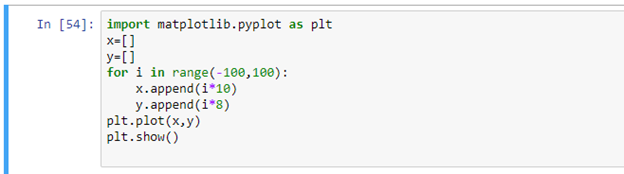
Paris

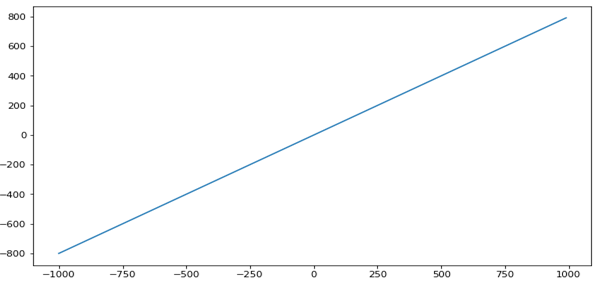


Using lambda function to find out mean and standard values



Scales:

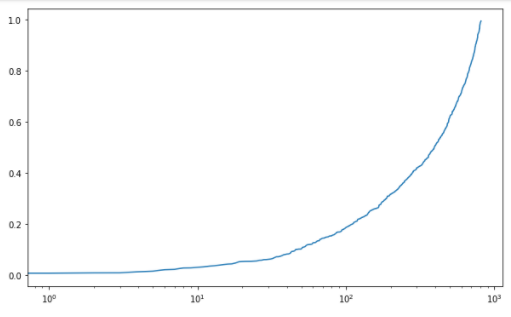




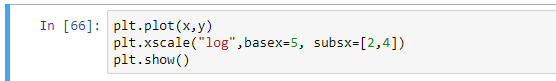
Different types of scales

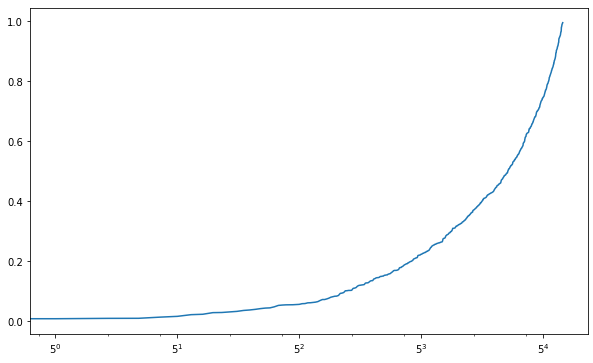
Log:



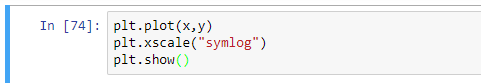


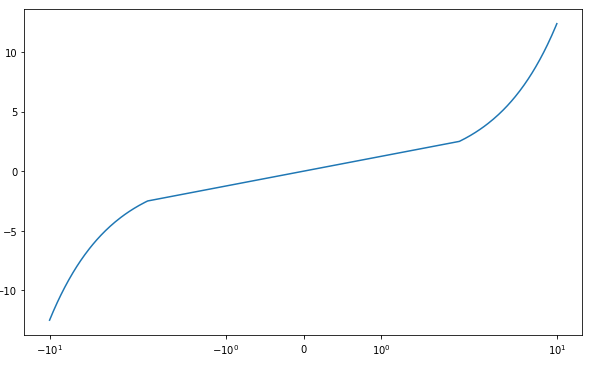
Set base value to x-axis:



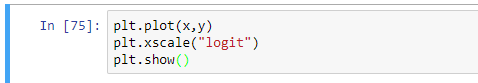


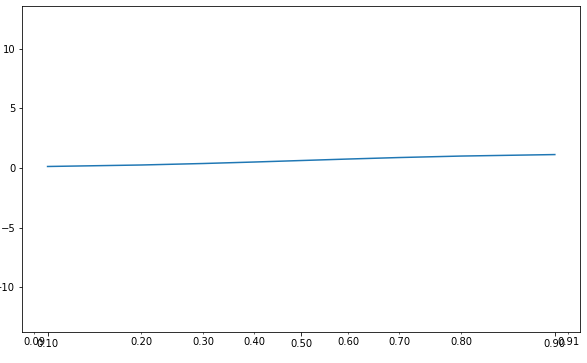
Symmetric log:





Logit:





For more go through this link:

https://matplotlib.org/tutorials/index.html

Pivot:

Most people likely have experience with pivot tables in excel. Pandas provides a similar function called pivot table. While it is exceedingly useful, I frequently find myself struggling to remember how to use the syntax to format the output for my needs. This article will focus on explaining the pandas pivot\_table function and how to use it for your data analysis.

The Data:

One of the challenges with using the panda’s pivot table is making sure you understand you data and what questions you are trying to answer with the pivot table. It is a seemingly simple function but can produce very powerful analysis very quickly.

In this scenario, I’m going to be tracking a sales pipeline. The basic problem is that some sales cycles are very long and management wants to understand it in more details throughout the year.

Typical questions include:

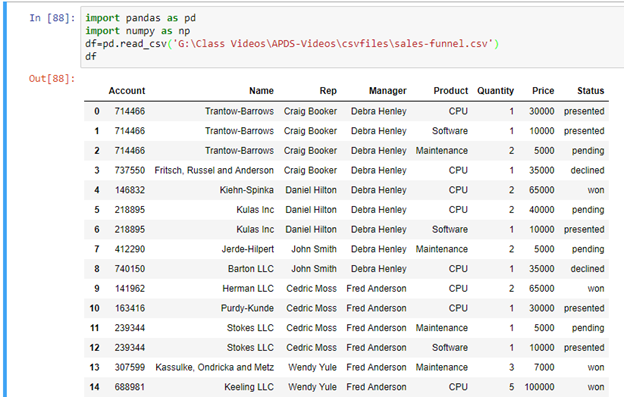
· How much revenue is in the pipeline?

· What product are in the pipeline?

· Who has what products at what stage?

· How likely are we to close deals by year end?

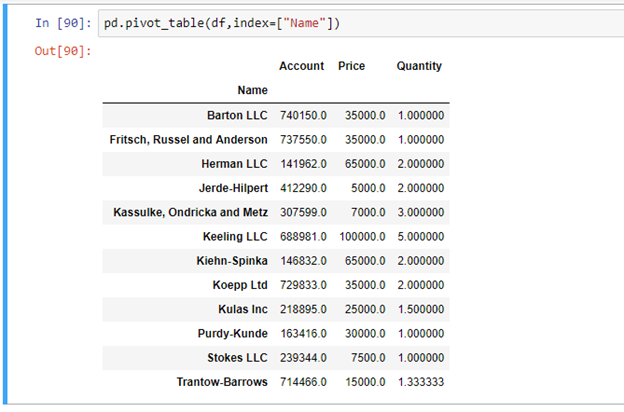
Read the data:



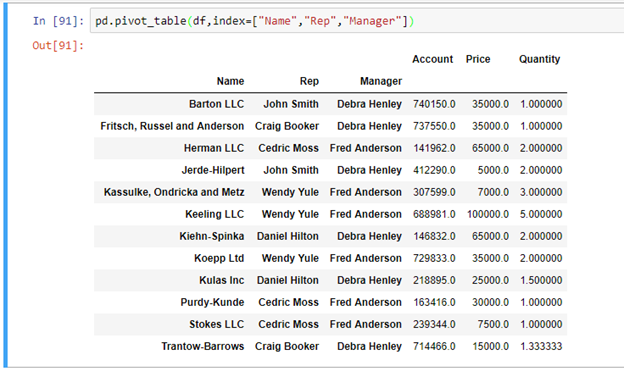
Pivot the data:

As we build up the pivot table. I think it’s easiest to take to one step at a time. Add items and check each step to verify you are getting the results you expect. Don’t be afraid to play with the order and the variables to see what the order and the variables to see what presentation makes the sense for your needs.

The simplest pivot table must have a dataframe and an index. In this case, lets use the name

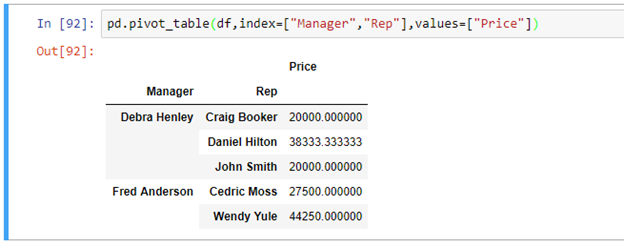


You can have multiple indexes as well. In fact, most of the pivot\_table args can take multiple values via a list.

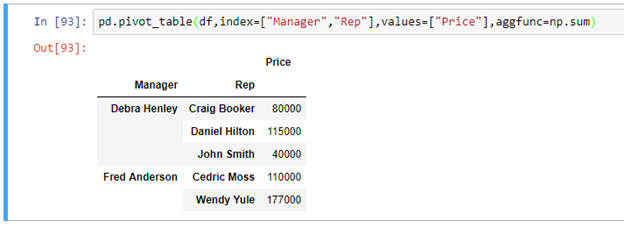


You can see that the pivot table is smart enough to start aggregating the data and summarizing it by grouping the reps with their managers. Now we start to get a glimpse of what a pivot table can do for us.

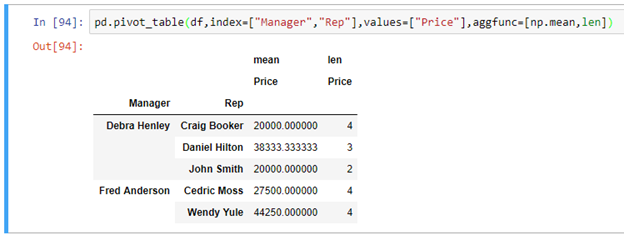
For this purpose, the account and quality columns aren’t really useful. Lets remove it by explicitly defining the columns we care about using the values field.



The price column automatically averages the data but we can do a count or a sum. Adding them is simple using “aggfunc” and “np.sum”.

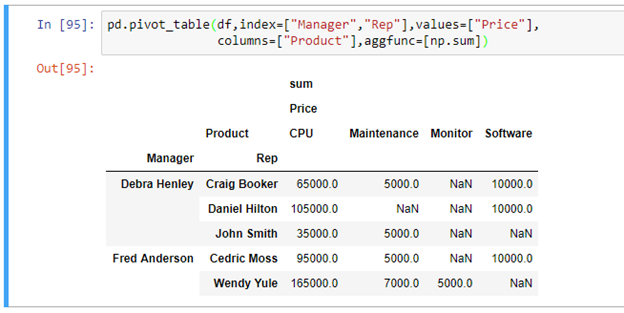


Aggfunc can take a list of functions. Lets try a mean using the numpy mean function and len to get a count.

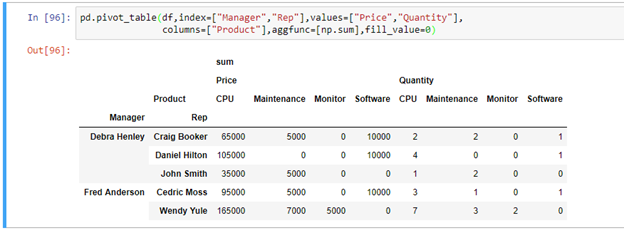


If we want to see sales broken down by the products, the columns variables allows us to define one or more columns.

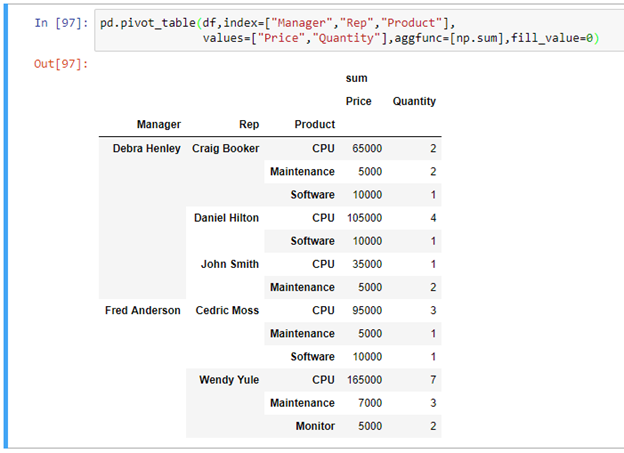
I think one of the confusing points with the pivot\_table is the use of columns and values. Remember , columns are optional they provides an additional way to segment the actual values you care about. The aggregation funvtions are applied to the values you list.



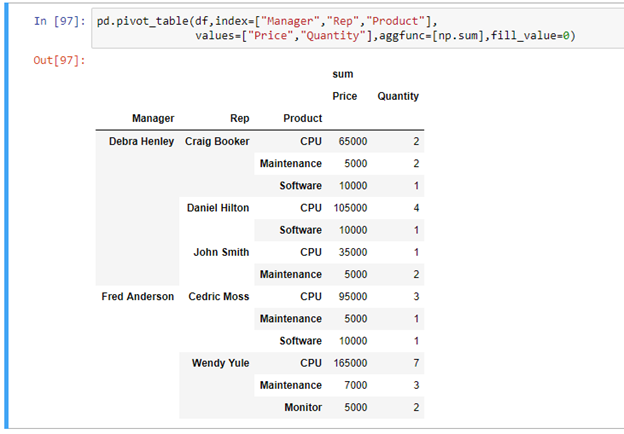
I think it would be useful to add the quantity as well. Add quantity to the values list.



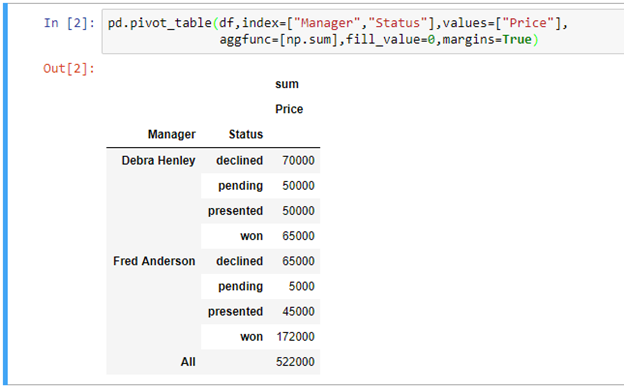
What’s interesting is that you can move items to the index to get a different visual representation. Remove product from the columns and add to the index.



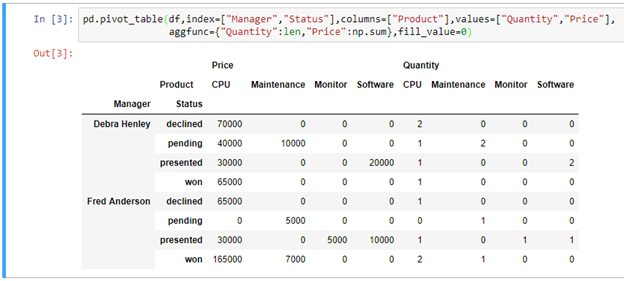
For this data set, this representation makes more sense. Now, what if I want to see some totals? Margins=True does that for us



Let’s move the analysis up a level and look at our pipeline at the manager level. Notice the status is ordered based on our earlier category definition.



A really handy feature is the ability to pass a dictionary to the aggfunc so you can perform different functions on each of the values you select. This has a side effect of making the labels a little cleaner.



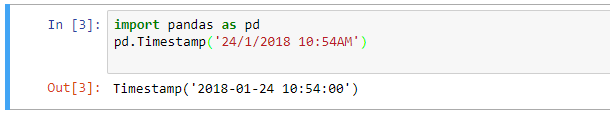
Date functionality in pandas:

Extending the time series, date functionalities play major role in financial data analysis. While working with date data, we will frequently come across the following

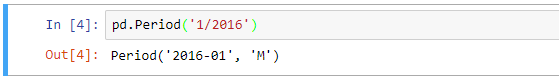
· Generating sequence of dates

· Convert the date series to different frequencies

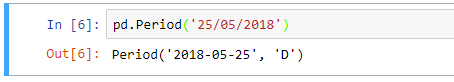
Creating dates:



Creating Month Period:



Creating Day module:



Date-Time index:



>>> t1=pd.Series([pd.Period("25/05/2018"),pd.Period("24/03/2018"),pd.Period("21/02/2018")])

>>> t1

0 2018-05-25

1 2018-03-24

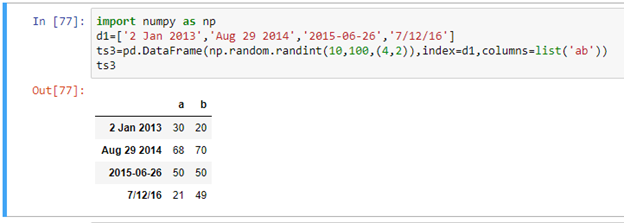
2 2018-02-21

dtype: object

Period index:

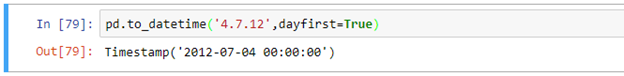


Converting to Datetime:



Datetime index:

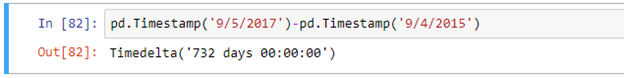


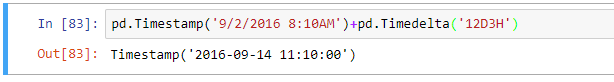


>>> pd.to\_datetime('04.08.2018')

Timestamp('2018-04-08 00:00:00')

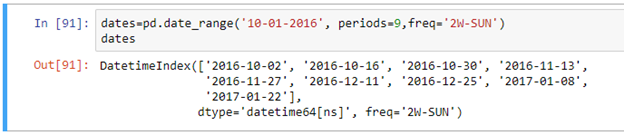
Time deltas:



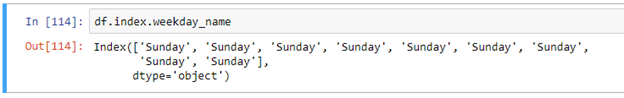


Dates range:

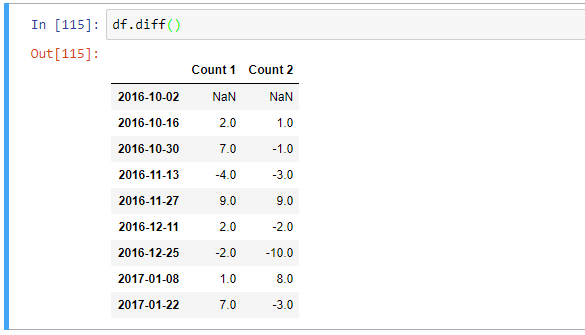
Using the date.range() function by specifying the periods and the frequency, we can create the date series. By default, the frequency of range is Days.







Difference between two dates



Calculating mean value:

