PANDAS

1. What is Pandas?

Pandas is defined as an open-source library that provides high-performance data manipulation in Python. The name of Pandas is derived from the word **Panel Data**, which means **an Econometrics from Multidimensional data**. It is used for data analysis in Python and developed by **Wes McKinney** in **2008**. Data analysis requires lots of processing, such as **restructuring, cleaning** or **merging**, etc. There are different tools are available for fast data processing, such as **Numpy, Scipy, Cython**, and **Panda**. But we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools. Pandas is built on top of the **Numpy** package, means **Numpy** is required for operating the Pandas.

Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., **load, manipulate, prepare, model, and analyse**.

2. Why use Pandas?

We use Pandas for its following advantages:

- Easily handles missing data
- It uses **Series for one-dimensional data structure** and **DataFrame for multi-dimensional data structure**
- It provides an efficient way to slice the data
- It provides a flexible way to merge, concatenate or reshape the data
- It includes a powerful time series tool to work with

In a nutshell, Pandas is a useful library in data analysis. It can be used to perform data manipulation and analysis. Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

3.How to install Pandas?

To install Python Pandas, go to your command prompt and type **"pip install pandas"** or else, if you have anaconda installed in your system, just type in **"conda install pandas".** Once the installation is completed, go to your IDE (Jupyter, PyCharm etc.) and simply import it by typing: **"import pandas as pd".**

Refer video for more details

1. install pandas in python	https://youtu.be/OMMWGnWSZlQ
2. install pandas in anaconda	https://youtu.be/CO5vAGW3Ozs

3. install pandas in jupyter notebook

https://youtu.be/OxuedW1CZPU

4. Python: Pandas Data Structure

The Pandas provides two data structures for processing the data, i.e., Series and DataFrame, which are discussed below:

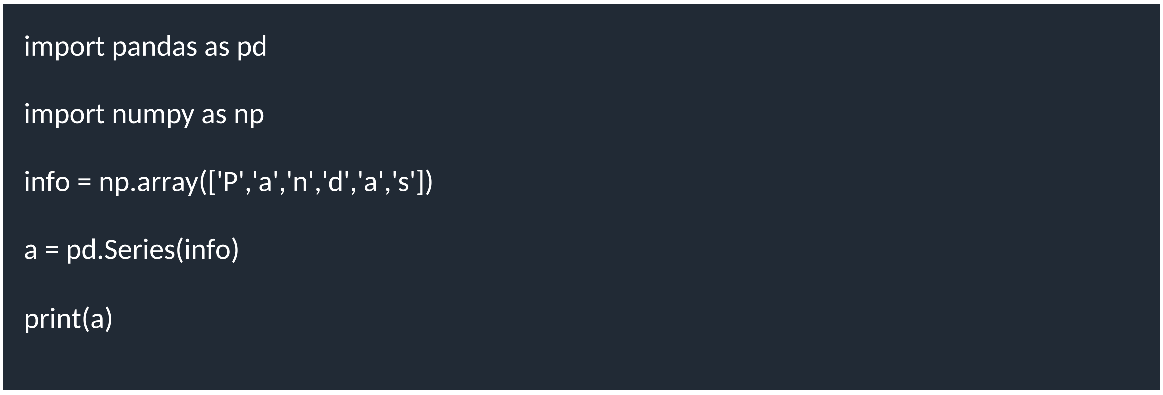
4.1. Series:

It is defined as a one-dimensional array that is capable of storing various data types. The row labels of series are called the **index**. We can easily convert the list, tuple, and dictionary into series using "series' method. A Series cannot contain multiple columns. It has one parameter:

Data: It can be any list, dictionary, or scalar value.

Creating Series from Array:

Before creating a Series, Firstly, we have to import the numpy module and then use array() function in the program.



```
import pandas as pd

import numpy as np

info = np.array(['P','a','n','d','a','s'])

a = pd.Series(info)

print(a)
```

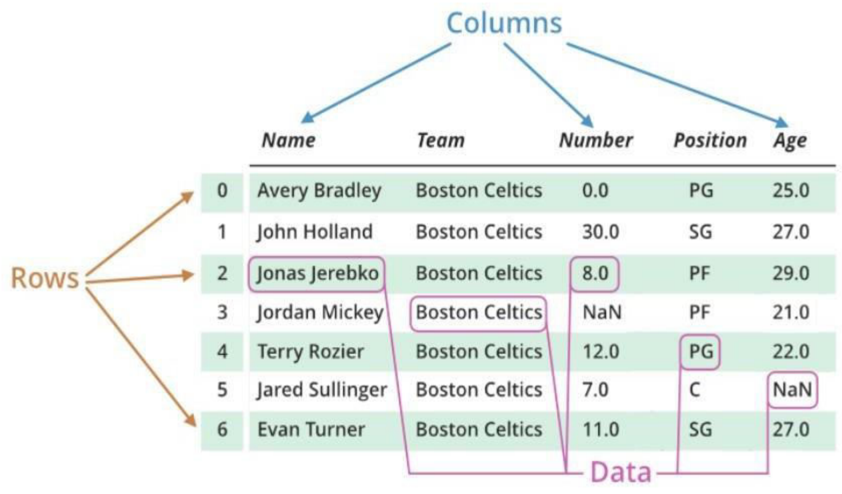
OUTPUT:

```
0   P
1   a
2   n
3   d
4   a
5   s

dtype: object
```

Explanation: In this code, firstly, we have imported the **pandas** and **numpy** library with the **pd** and **np** alias. Then, we have taken a variable named "info" that consist of an array of some values. We have called the **info** variable through a **Series** method and defined it in an "**a**" variable. The Series has printed by calling the **print(a)** method.

1.1. DataFrames:

It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

Create a DataFrame using List:

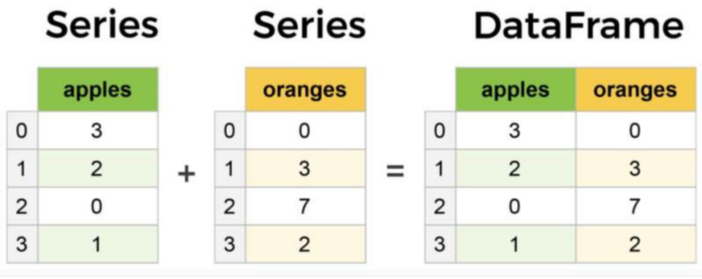
We can easily create a DataFrame in Pandas using list.

```
import pandas as pd

# list of strings

Lst = ["EST","Temperature","DewPoint","Humidity","Sea Level PressureIn","VisibilityMiles",
"WindSpeedMPH","PrecipitationIn","CloudCover","Events","WindDirDegrees" ]

# Calling DataFrame constructor on list

df = pd.DataFrame(Lst)

print(df)
```

OUTPUT:

```
                  0
0                EST
1          Temperature
2            DewPoint
3            Humidity
4   Sea Level PressureIn
5   VisibilityMiles
6          WindSpeedMPH
7   PrecipitationIn
8           CloudCover
9              Events
10       WindDirDegrees
```

Explanation: In this code, we have defined a variable named "Lst" that consist of string values. The DataFrame constructor is being called on a list to print the values. We can create DataFrame using dict also .



Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

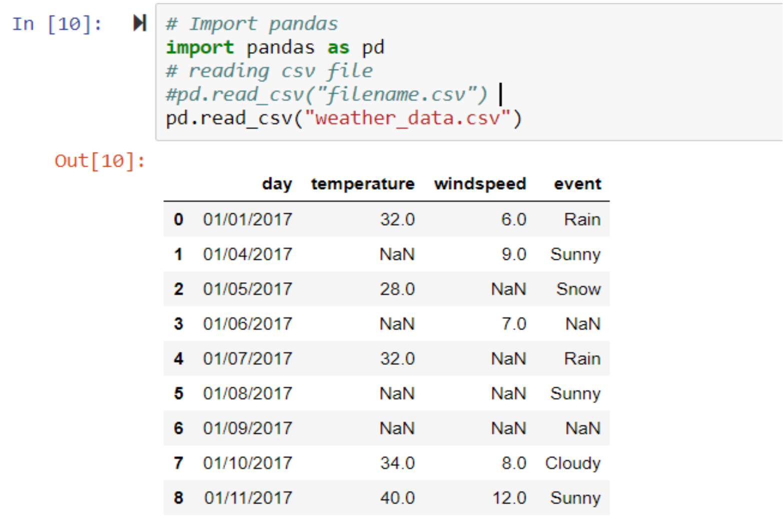
DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

5. How to read in data
   It’s quite simple to load data from various file formats into a DataFrame.

4.2.Reading data from CSVs :

With CSV files all you need is a single line to load in the data:



In [10]:

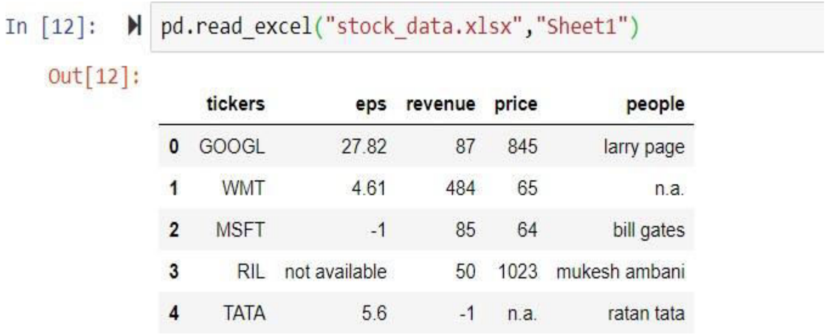
```
# Import pandas
import pandas as pd
# reading csv file
#pd.read_csv("filename.csv")
pd.read_csv("weather_data.csv")
```

Out[10]:

	day	temperature	windspeed	event
0	01/01/2017	32.0	6.0	Rain
1	01/04/2017	NaN	9.0	Sunny
2	01/05/2017	28.0	NaN	Snow
3	01/06/2017	NaN	7.0	NaN
4	01/07/2017	32.0	NaN	Rain
5	01/08/2017	NaN	NaN	Sunny
6	01/09/2017	NaN	NaN	NaN
7	01/10/2017	34.0	8.0	Cloudy
8	01/11/2017	40.0	12.0	Sunny

4.3.Reading data from excel :

With excel files all you need is a single line to load in the data and if there is various sheets in single excel file then we have to write sheet number .



In [12]:

```
pd.read_excel("stock_data.xlsx","Sheet1")
```

Out[12]:

	tickers	eps	revenue	price	people
0	GOOGL	27.82	87	845	larry page
1	WMT	4.61	484	65	n.a.
2	MSFT	-1	85	64	bill gates
3	RIL	not available	50	1023	mukesh ambani
4	TATA	5.6	-1	n.a.	ratan tata

5. Most important DataFrame operations

DataFrames possess hundreds of methods and other operations that are crucial to any analysis. As a beginner, you should know the operations that perform simple transformations of your data and those that provide fundamental statistical analysis.

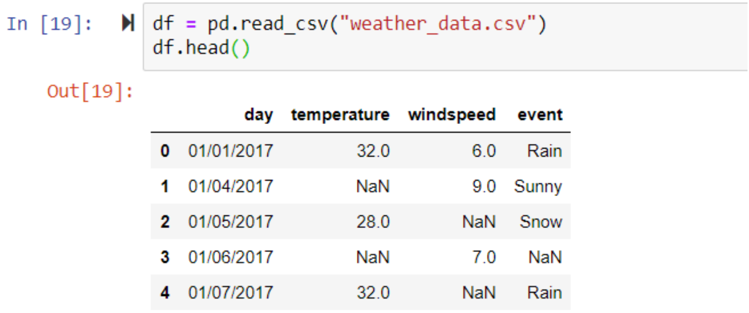
We're loading this dataset from a CSV and designating the **event** to be our index from **weather_data.csv**.

```
import pandas as pd

# making data frame from csv file and indexig of col

df = pd.read_csv("weather_data.csv", index_col ="event")

df
```

5.2.Viewing your data & Getting info about your data:

Viewing your data:

The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with .head()

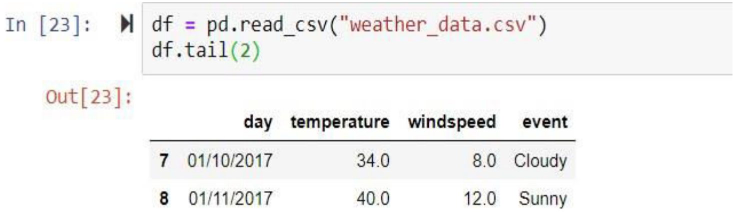


```
In [19]:  df = pd.read_csv("weather_data.csv")
          df.head()
Out[19]:
```

	day	temperature	windspeed	event
0	01/01/2017	32.0	6.0	Rain
1	01/04/2017	NaN	9.0	Sunny
2	01/05/2017	28.0	NaN	Snow
3	01/06/2017	NaN	7.0	NaN
4	01/07/2017	32.0	NaN	Rain

.head() outputs the **first** five rows of your DataFrame by default, but we could also pass a number as well: **df.head(7)** would output the top seven rows.

To see the **last** five rows use .tail() also accepts a number, and in this case we printing the bottom two rows.:

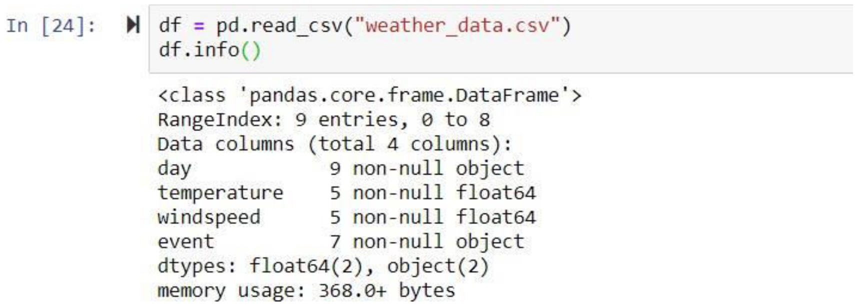


```
In [23]:  df = pd.read_csv("weather_data.csv")
          df.tail(2)
Out[23]:
```

	day	temperature	windspeed	event
7	01/10/2017	34.0	8.0	Cloudy
8	01/11/2017	40.0	12.0	Sunny

Getting info about your data:

.info() should be one of the very first commands you run after loading your data:



```
In [24]:  df = pd.read_csv("weather_data.csv")
          df.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 9 entries, 0 to 8
          Data columns (total 4 columns):
          day            9 non-null object
          temperature    5 non-null float64
          windspeed      5 non-null float64
          event          7 non-null object
          dtypes: float64(2), object(2)
          memory usage: 368.0+ bytes
```

.info() provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using. Seeing the datatype quickly is actually quite useful.

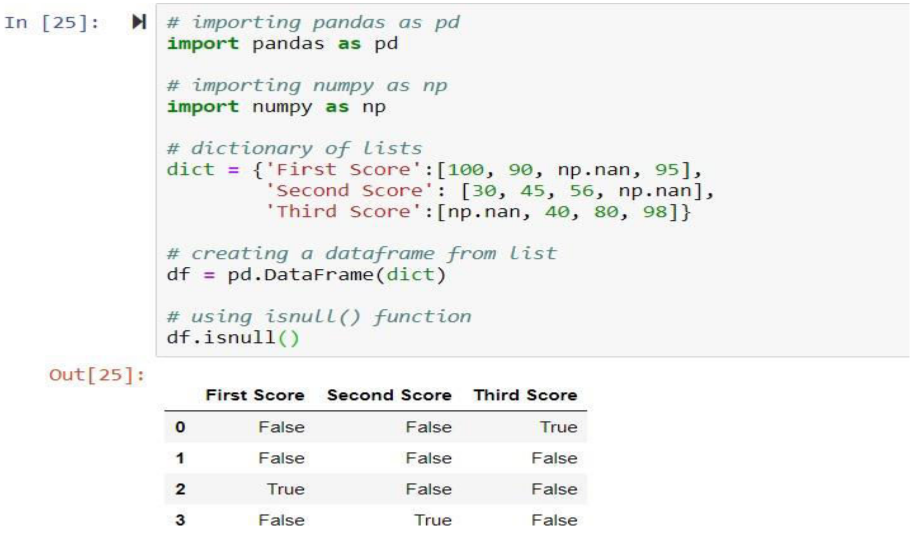
Similarly TDO, *DataFrame.append()*, *DataFrame.apply()*, *DataFrame.aggregate()*, *DataFrame.assign()*, *DataFrame.astype()*, *DataFrame.count()*, *DataFrame.cut()*, *DataFrame.describe()*, *DataFrame.drop_duplicates()*

5.3.How to work with missing values:

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA(Not Available) values in pandas.

A) Checking for missing values using isnull() and notnull() :

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.



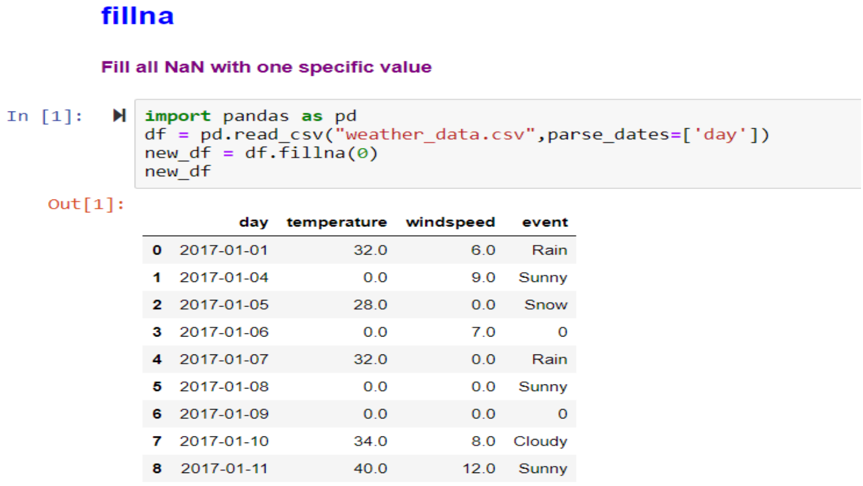
```
In [25]:  # importing pandas as pd
          import pandas as pd

          # importing numpy as np
          import numpy as np

          # dictionary of lists
          dict = {'First Score':[100, 90, np.nan, 95],
                  'Second Score': [30, 45, 56, np.nan],
                  'Third Score':[np.nan, 40, 80, 98]}

          # creating a dataframe from list
          df = pd.DataFrame(dict)

          # using isnull() function
          df.isnull()
```

Out[25]:

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

B) Filling missing values using fillna()**, and** interpolate() :

In order to fill null values in a datasets, we use fillna() and interpolate() function these function replace NaN values with some value of their own.These function help in filling a null values in datasets of a DataFrame. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the value.



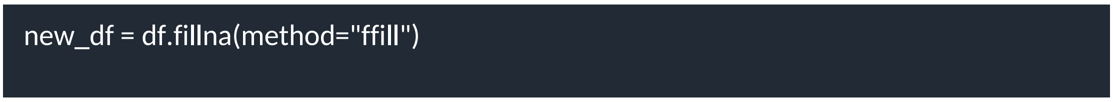
fillna

Fill all NaN with one specific value

```
In [1]:  import pandas as pd
         df = pd.read_csv("weather_data.csv",parse_dates=['day'])
         new_df = df.fillna(0)
         new_df
```

Out[1]:

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-04	0.0	9.0	Sunny
2	2017-01-05	28.0	0.0	Snow
3	2017-01-06	0.0	7.0	0
4	2017-01-07	32.0	0.0	Rain
5	2017-01-08	0.0	0.0	Sunny
6	2017-01-09	0.0	0.0	0
7	2017-01-10	34.0	8.0	Cloudy
8	2017-01-11	40.0	12.0	Sunny



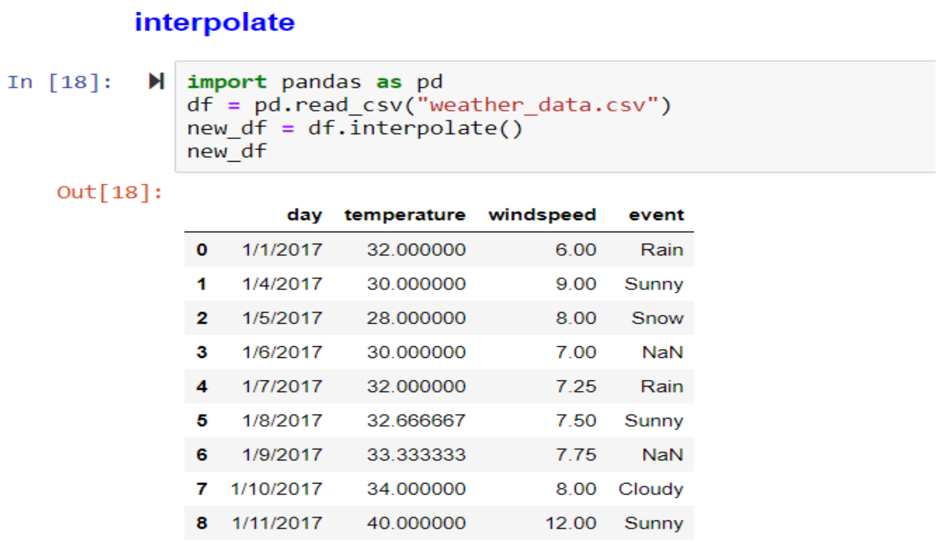
```
new_df = df.fillna(method="ffill")
```

here we are using forward filling method in fillna . where forward row's value will be copied in place of NaN in data .

```
new_df = df.fillna(method="bfill")
```

here we are using backward filling method in fillna . where backward row’s value will be copied in place of NaN in data. Similarly we can copy col’s value by simply declaring axis as

“new_df = df.fillna(method="bfill", axis="columns") # axis is either "index" or "columns" ”



interpolate

```
In [18]:  import pandas as pd
          df = pd.read_csv("weather_data.csv")
          new_df = df.interpolate()
          new_df
```

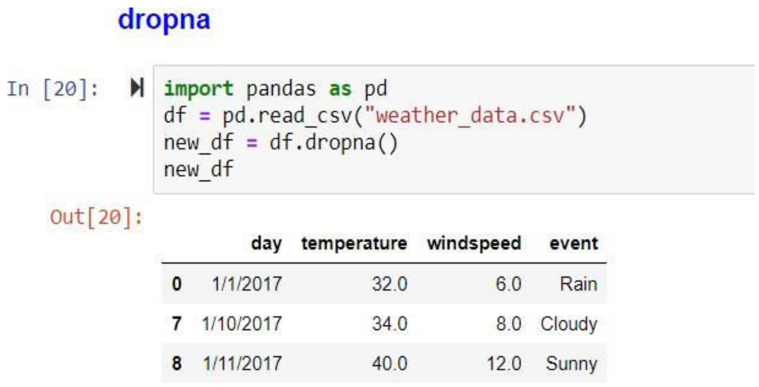
Out[18]:

	day	temperature	windspeed	event
0	1/1/2017	32.000000	6.00	Rain
1	1/4/2017	30.000000	9.00	Sunny
2	1/5/2017	28.000000	8.00	Snow
3	1/6/2017	30.000000	7.00	NaN
4	1/7/2017	32.000000	7.25	Rain
5	1/8/2017	32.666667	7.50	Sunny
6	1/9/2017	33.333333	7.75	NaN
7	1/10/2017	34.000000	8.00	Cloudy
8	1/11/2017	40.000000	12.00	Sunny

Here in interpolation method it is placing average value from upper row and lower row in-place of NaN .

C) **Dropping missing values using** dropna() **:**

In order to drop a null values from a dataframe, we used dropna() function this fuction drop Rows/Columns of datasets with Null values in different ways. We can use this in data where dropping will not affect our main data .



dropna

```
In [20]:  import pandas as pd
          df = pd.read_csv("weather_data.csv")
          new_df = df.dropna()
          new_df
```

Out[20]:

	day	temperature	windspeed	event
0	1/1/2017	32.0	6.0	Rain
7	1/10/2017	34.0	8.0	Cloudy
8	1/11/2017	40.0	12.0	Sunny

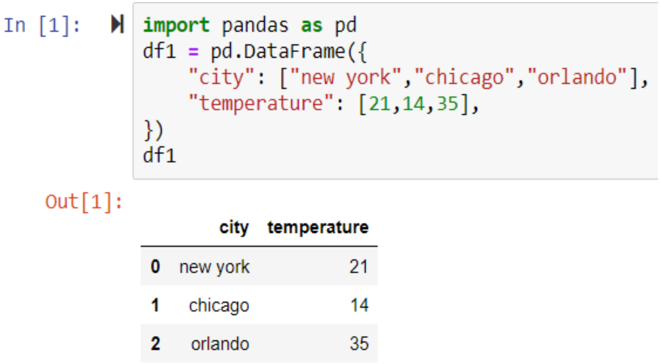
Here in our data there is only 3 rows which do not contain any NaN value .

6.3 DataFrame Merging, Joining, Concatenating , Group By & Crosstab :

A) Merging & Joining :

In merging, you can merge two data frames to form a single data frame. You can also decide which columns you want to make common. Let me implement that practically, first we will create two data frames, which has some key-value pairs and then merge the data frames together.

1st DataFrame:

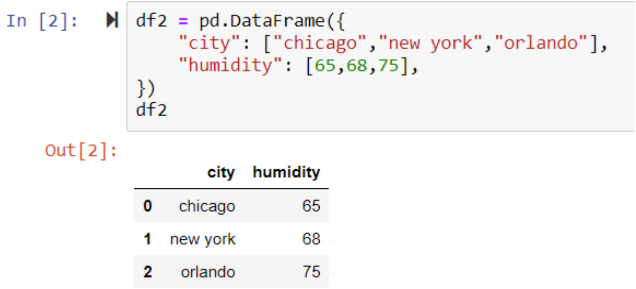


```
In [1]:  import pandas as pd
         df1 = pd.DataFrame({
             "city": ["new york","chicago","orlando"],
             "temperature": [21,14,35],
         })
         df1
```

Out[1]:

	city	temperature
0	new york	21
1	chicago	14
2	orlando	35

2^{nd} DataFrame:

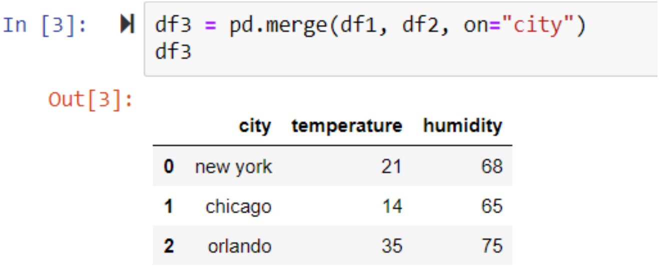


```
In [2]:  df2 = pd.DataFrame({
             "city": ["chicago","new york","orlando"],
             "humidity": [65,68,75],
         })
         df2
```

Out[2]:

	city	humidity
0	chicago	65
1	new york	68
2	orlando	75

Merging both:

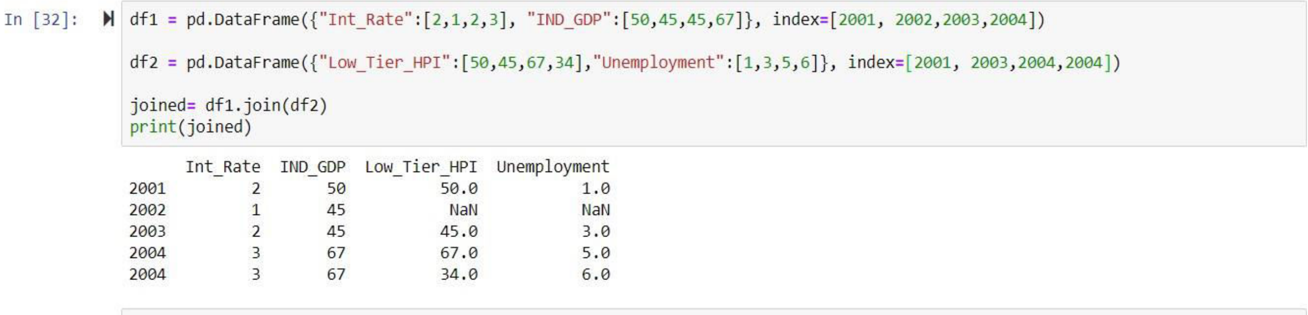


```
In [3]:  df3 = pd.merge(df1, df2, on="city")
         df3
```

Out[3]:

	city	temperature	humidity
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

joining in python pandas. It is yet another convenient method to combine two differently indexed dataframes into a single result dataframe. This is quite similar to the "merge" operation, except the joining operation will be on the "index" instead of the "columns". Let us implement it practically.

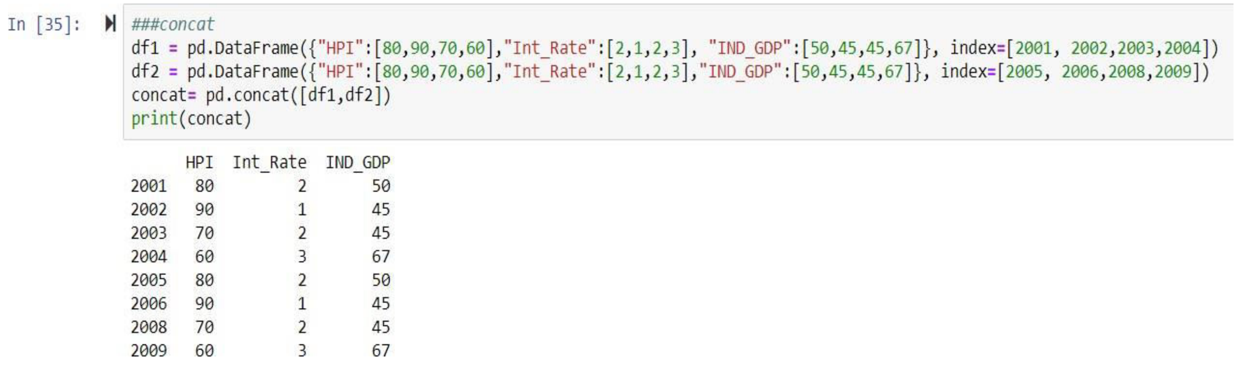


```
In [32]:  df1 = pd.DataFrame({"Int_Rate":[2,1,2,3], "IND_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])

          df2 = pd.DataFrame({"Low_Tier_HPI":[50,45,67,34],"Unemployment":[1,3,5,6]}, index=[2001, 2003,2004,2004])

          joined= df1.join(df2)
          print(joined)

      Int_Rate  IND_GDP  Low_Tier_HPI  Unemployment
2001         2       50          50.0           1.0
2002         1       45           NaN           NaN
2003         2       45          45.0           3.0
2004         3       67          67.0           5.0
2004         3       67          34.0           6.0
```

As you can notice in the above output, in year 2002(index), there is no value attached to columns "low_tier_HPI" and "unemployment", therefore it has printed NaN (Not a Number). Later in 2004, both the values are available, therefore it has printed the respective values.

B) Concatenation :

Concatenation basically glues the dataframes together. You can select the dimension on which you want to concatenate. For that, just use "pd.concat" and pass in the list of dataframes to concatenate together. Consider the below example.

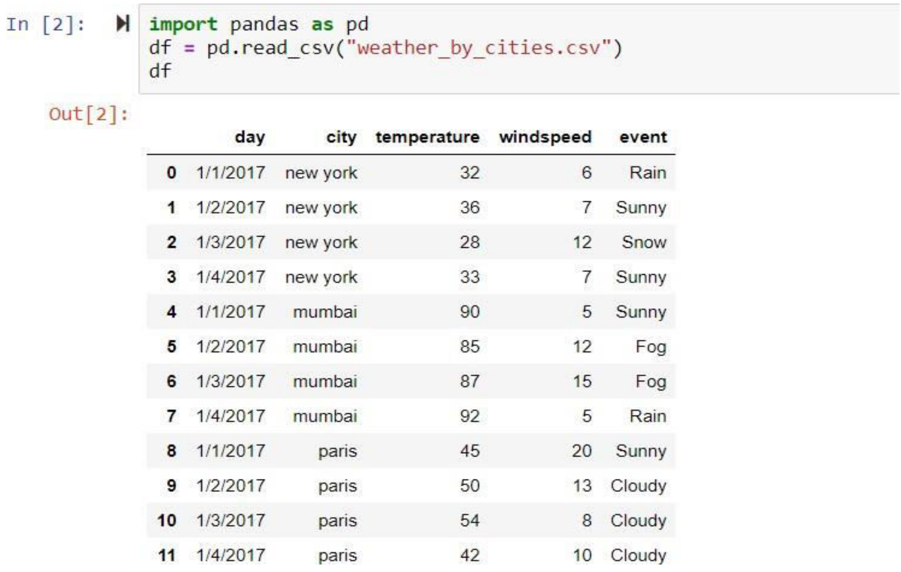


```
In [35]:  ###concat
          df1 = pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3], "IND_GDP":[50,45,45,67]}, index=[2001, 2002,2003,2004])
          df2 = pd.DataFrame({"HPI":[80,90,70,60],"Int_Rate":[2,1,2,3],"IND_GDP":[50,45,45,67]}, index=[2005, 2006,2008,2009])
          concat= pd.concat([df1,df2])
          print(concat)

                HPI  Int_Rate  IND_GDP
          2001   80         2       50
          2002   90         1       45
          2003   70         2       45
          2004   60         3       67
          2005   80         2       50
          2006   90         1       45
          2008   70         2       45
          2009   60         3       67
```

As you can see above, the two dataframes are glued together in a single dataframe, where the index starts from 2001 and ends at 2009.

C) Group By :

In Pandas, **groupby()** function allows us to rearrange the data by utilizing them on real-world data sets. Its primary task is to split the data into various groups. These groups are categorized based on some criteria. The objects can be divided from any of their axes.



```
In [2]:  import pandas as pd
          df = pd.read_csv("weather_by_cities.csv")
          df
```

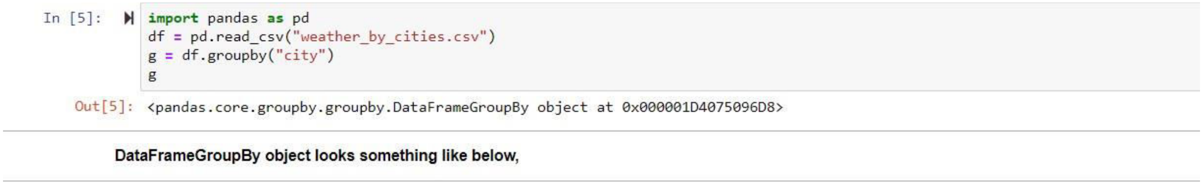
Out[2]:

	day	city	temperature	windspeed	event
0	1/1/2017	new york	32	6	Rain
1	1/2/2017	new york	36	7	Sunny
2	1/3/2017	new york	28	12	Snow
3	1/4/2017	new york	33	7	Sunny
4	1/1/2017	mumbai	90	5	Sunny
5	1/2/2017	mumbai	85	12	Fog
6	1/3/2017	mumbai	87	15	Fog
7	1/4/2017	mumbai	92	5	Rain
8	1/1/2017	paris	45	20	Sunny
9	1/2/2017	paris	50	13	Cloudy
10	1/3/2017	paris	54	8	Cloudy
11	1/4/2017	paris	42	10	Cloudy

For this dataset, get following answers,

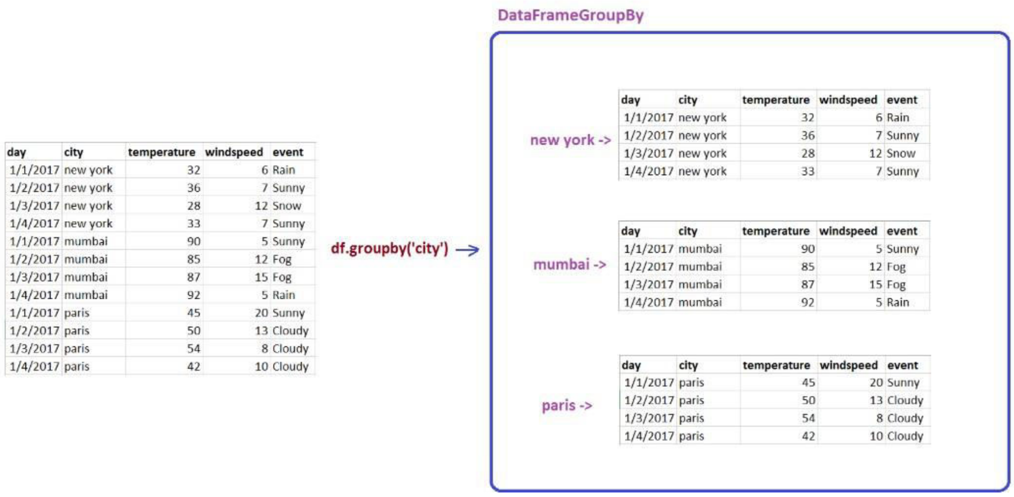
1. What was the maximum temperature in each of these 3 cities?

2. What was the average windspeed in each of these 3 cities?



```
In [5]:  import pandas as pd
         df = pd.read_csv("weather_by_cities.csv")
         g = df.groupby("city")
         g
Out[5]: <pandas.core.groupby.groupby.DataFrameGroupBy object at 0x000001D4075096D8>
```

DataFrameGroupBy object looks something like below,

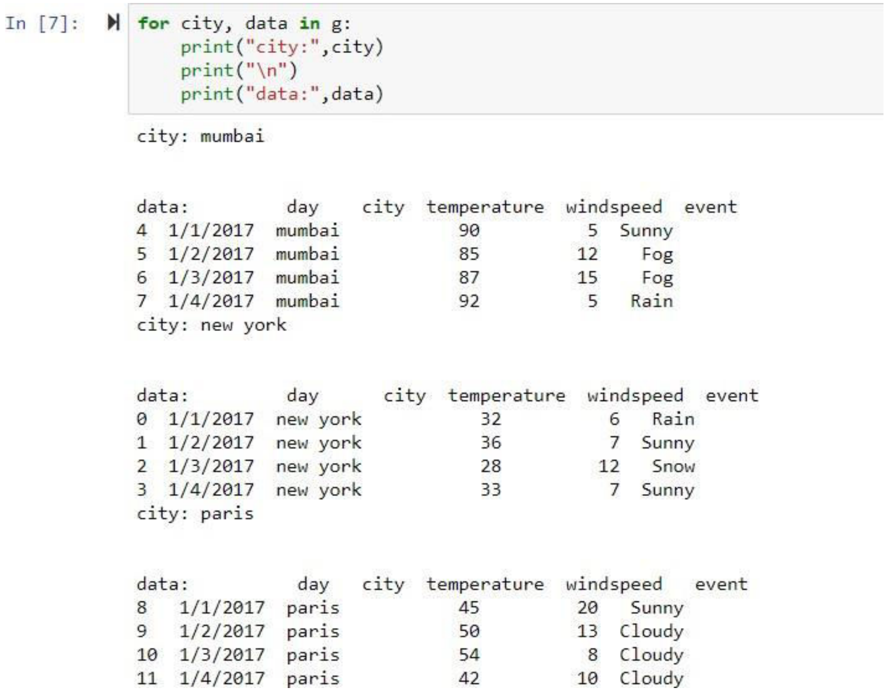


day	city	temperature	windspeed	event
1/1/2017	new york	32	6	Rain
1/2/2017	new york	36	7	Sunny
1/3/2017	new york	28	12	Snow
1/4/2017	new york	33	7	Sunny
1/1/2017	mumbai	90	5	Sunny
1/2/2017	mumbai	85	12	Fog
1/3/2017	mumbai	87	15	Fog
1/4/2017	mumbai	92	5	Rain
1/1/2017	paris	45	20	Sunny
1/2/2017	paris	50	13	Cloudy
1/3/2017	paris	54	8	Cloudy
1/4/2017	paris	42	10	Cloudy

day	city	temperature	windspeed	event
1/1/2017	new york	32	6	Rain
1/2/2017	new york	36	7	Sunny
1/3/2017	new york	28	12	Snow
1/4/2017	new york	33	7	Sunny

day	city	temperature	windspeed	event
1/1/2017	mumbai	90	5	Sunny
1/2/2017	mumbai	85	12	Fog
1/3/2017	mumbai	87	15	Fog
1/4/2017	mumbai	92	5	Rain

day	city	temperature	windspeed	event
1/1/2017	paris	45	20	Sunny
1/2/2017	paris	50	13	Cloudy
1/3/2017	paris	54	8	Cloudy
1/4/2017	paris	42	10	Cloudy



```
In [7]:  for city, data in g:
             print("city:",city)
             print("\n")
             print("data:",data)

         city: mumbai


         data:          day    city  temperature  windspeed  event
         4  1/1/2017  mumbai           90          5  Sunny
         5  1/2/2017  mumbai           85         12    Fog
         6  1/3/2017  mumbai           87         15    Fog
         7  1/4/2017  mumbai           92          5   Rain
         city: new york


         data:          day      city  temperature  windspeed  event
         0  1/1/2017  new york           32          6   Rain
         1  1/2/2017  new york           36          7  Sunny
         2  1/3/2017  new york           28         12   Snow
         3  1/4/2017  new york           33          7  Sunny
         city: paris


         data:           day   city  temperature  windspeed   event
         8   1/1/2017  paris           45         20   Sunny
         9   1/2/2017  paris           50         13  Cloudy
         10  1/3/2017  paris           54          8  Cloudy
         11  1/4/2017  paris           42         10  Cloudy
```

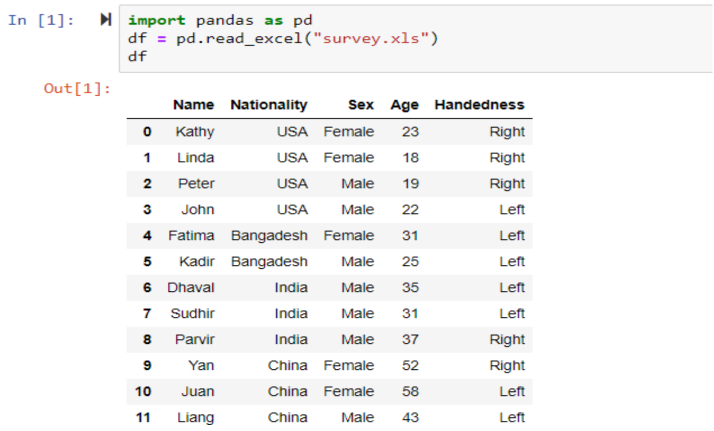
You can use this … To find max temperature , avg windspeed ,etc

```
g.max()

g.min()

g.mean()

g.describe()

g.count()

g.size()
```

D) Crosstab : (also known as contingency table or cross tabulation) is a table showing frequency distribution of one variable in rows and another on columns. **pandas crosstab** method can be used to generate these contingency tables that are extremely useful in survey and business analytics.

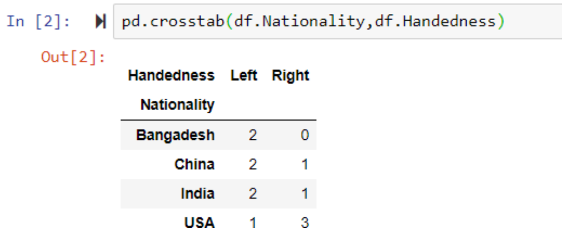
For the first example, let’s use pd.crosstab . here we have different parameters like name , nationality , sex , age & handedness to understand information in tabular form

>original data



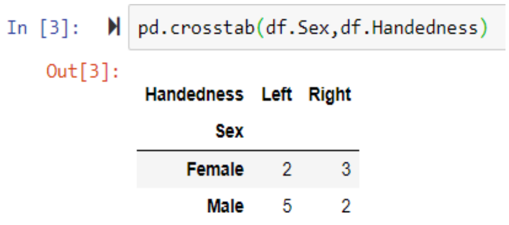
```
In [1]:  import pandas as pd
         df = pd.read_excel("survey.xls")
         df

Out[1]:
          Name  Nationality     Sex  Age  Handedness
     0   Kathy          USA  Female   23       Right
     1   Linda          USA  Female   18       Right
     2   Peter          USA    Male   19       Right
     3    John          USA    Male   22        Left
     4  Fatima    Bangadesh  Female   31        Left
     5   Kadir    Bangadesh    Male   25        Left
     6  Dhaval        India    Male   35        Left
     7  Sudhir        India    Male   31        Left
     8  Parvir        India    Male   37       Right
     9     Yan        China  Female   52       Right
    10    Juan        China  Female   58        Left
    11   Liang        China    Male   43        Left
```

>plotted in terms of nationality with handedness



```
In [2]:  pd.crosstab(df.Nationality,df.Handedness)

Out[2]:
    Handedness  Left  Right
    Nationality
     Bangadesh     2      0
         China     2      1
         India     2      1
           USA     1      3
```

>plotted in terms of sex with handedness



```
In [3]:  pd.crosstab(df.Sex,df.Handedness)
Out[3]:
   Handedness  Left  Right
          Sex
       Female     2      3
         Male     5      2
```

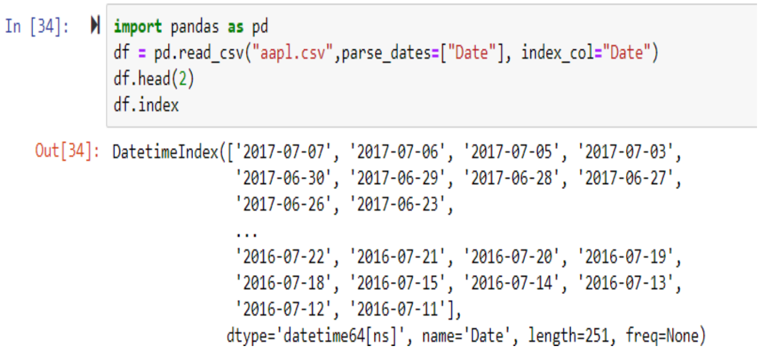
The pandas crosstab function is a useful tool for summarizing data. The functionality overlaps with some of the other pandas tools but it occupies a useful place in your data analysis toolbox.

7 Pandas Time Series

The Time series data is defined as an important source for information that provides a strategy that is used in various businesses. From a conventional finance industry to the education industry, it consist of a lot of details about the time. Time series forecasting is the machine learning modeling that deals with the Time Series data for predicting future values through Time Series modeling.

7.1 Pandas DatetimeIndex:

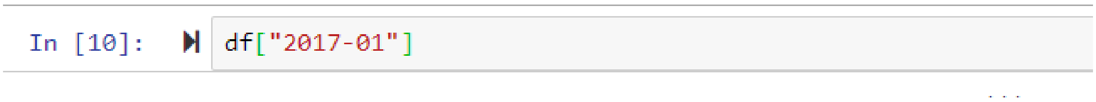
The Pandas can provide the features to work with time-series data for all domains. It also consolidates a large number of features from other Python libraries like scikits.timeseries . It provides new functionalities for manipulating the time series data. The time series tools are most useful for data science applications and deals with other packages used in Python.



```
In [34]:  import pandas as pd
          df = pd.read_csv("aapl.csv",parse_dates=["Date"], index_col="Date")
          df.head(2)
          df.index

Out[34]: DatetimeIndex(['2017-07-07', '2017-07-06', '2017-07-05', '2017-07-03',
                        '2017-06-30', '2017-06-29', '2017-06-28', '2017-06-27',
                        '2017-06-26', '2017-06-23',
                        ...
                        '2016-07-22', '2016-07-21', '2016-07-20', '2016-07-19',
                        '2016-07-18', '2016-07-15', '2016-07-14', '2016-07-13',
                        '2016-07-12', '2016-07-11'],
                       dtype='datetime64[ns]', name='Date', length=251, freq=None)
```

Here , imported pandas library , read csv file i.e. aapl.csv, parse_dates is use to typecast as type date , printed index with head (2) . Advantages of having datetimeindex:

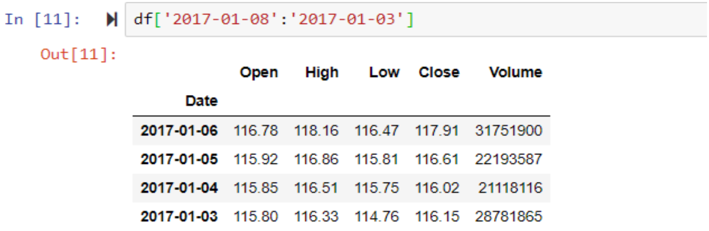
A) Partial Date Index: Select Specific Months Data



```
In [10]:  df["2017-01"]
```

Here , we have printed only specific month from specific year . if we need only MM/YY with all the dates .

B) Select Date Range

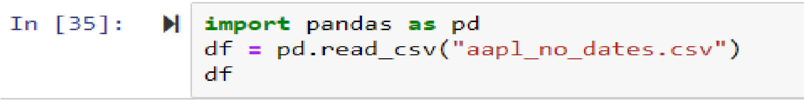


```
In [11]:  df['2017-01-08':'2017-01-03']
Out[11]:
```

Date	Open	High	Low	Close	Volume
2017-01-06	116.78	118.16	116.47	117.91	31751900
2017-01-05	115.92	116.86	115.81	116.61	22193587
2017-01-04	115.85	116.51	115.75	116.02	21118116
2017-01-03	115.80	116.33	114.76	116.15	28781865

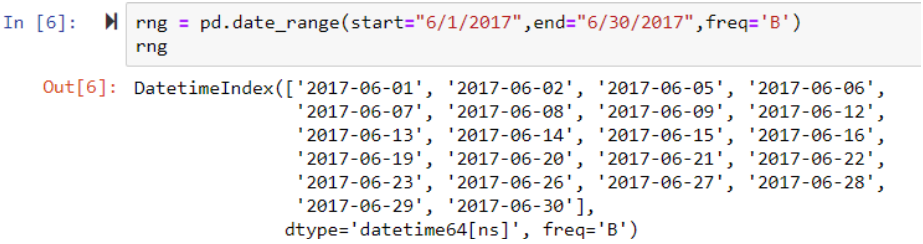
Here , we can select only specific range if we need only of this .

C) Finding missing dates from datetimeindex



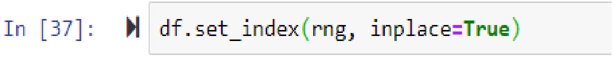
```
In [35]:  import pandas as pd
          df = pd.read_csv("aapl_no_dates.csv")
          df
```

here, we have imported lib and read csv file i.e. **“aapl_no_dates.csv”**



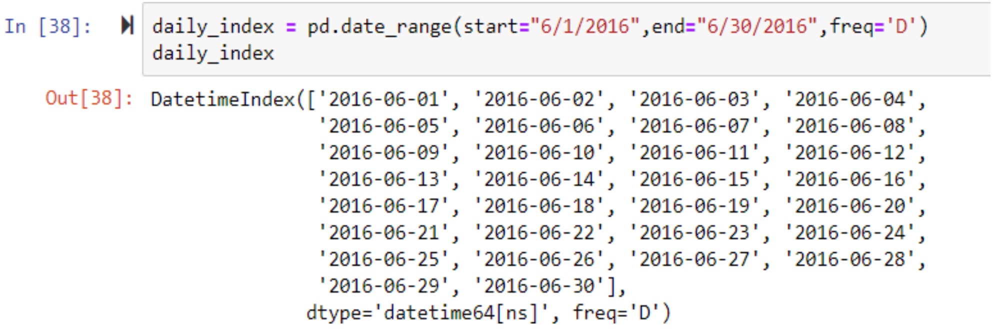
```
In [6]:  rng = pd.date_range(start="6/1/2017",end="6/30/2017",freq='B')
         rng

Out[6]: DatetimeIndex(['2017-06-01', '2017-06-02', '2017-06-05', '2017-06-06',
                       '2017-06-07', '2017-06-08', '2017-06-09', '2017-06-12',
                       '2017-06-13', '2017-06-14', '2017-06-15', '2017-06-16',
                       '2017-06-19', '2017-06-20', '2017-06-21', '2017-06-22',
                       '2017-06-23', '2017-06-26', '2017-06-27', '2017-06-28',
                       '2017-06-29', '2017-06-30'],
                      dtype='datetime64[ns]', freq='B')
```

Our data does not have dates so we inserted dates in data with date_range function . arguments which we have passed is start date end date and freq which is B means business days which will drop weekends .



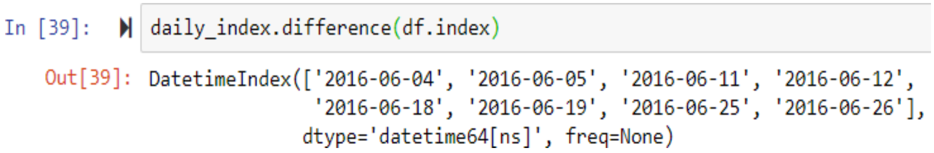
```
In [37]:  df.set_index(rng, inplace=True)
```

Set index rng i.e. dates



```
In [38]:  daily_index = pd.date_range(start="6/1/2016",end="6/30/2016",freq='D')
          daily_index

Out[38]: DatetimeIndex(['2016-06-01', '2016-06-02', '2016-06-03', '2016-06-04',
                        '2016-06-05', '2016-06-06', '2016-06-07', '2016-06-08',
                        '2016-06-09', '2016-06-10', '2016-06-11', '2016-06-12',
                        '2016-06-13', '2016-06-14', '2016-06-15', '2016-06-16',
                        '2016-06-17', '2016-06-18', '2016-06-19', '2016-06-20',
                        '2016-06-21', '2016-06-22', '2016-06-23', '2016-06-24',
                        '2016-06-25', '2016-06-26', '2016-06-27', '2016-06-28',
                        '2016-06-29', '2016-06-30'],
                       dtype='datetime64[ns]', freq='D')
```

Daily_index function contains all days with the same dates . arguments which we have passed is start date end date and freq which is D .

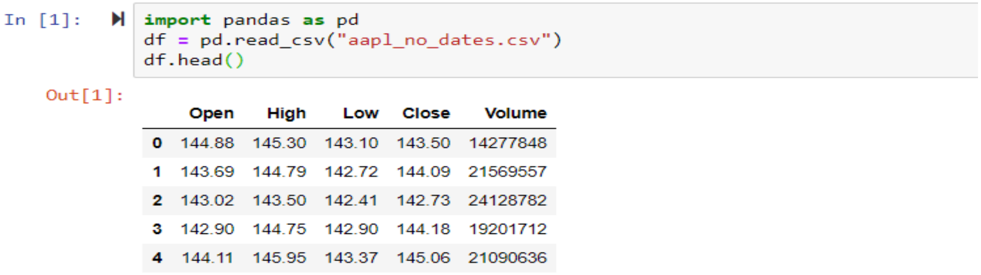


```
In [39]:  daily_index.difference(df.index)

Out[39]: DatetimeIndex(['2016-06-04', '2016-06-05', '2016-06-11', '2016-06-12',
                        '2016-06-18', '2016-06-19', '2016-06-25', '2016-06-26'],
                       dtype='datetime64[ns]', freq=None)
```

On compairing business days with all days, we will get missing dates from rng function.

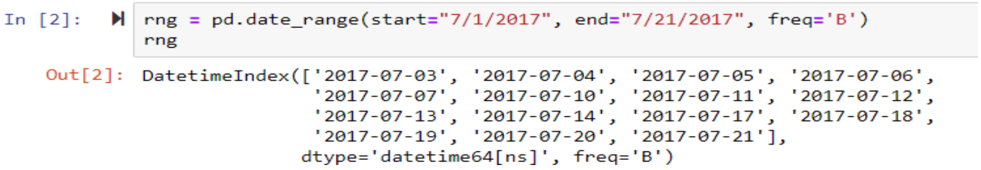
5.4. Handling Holidays:

Here , importing pandas , read csv file and on next cell our data does not have dates so we inserted dates in data with date_range function . arguments which we have passed is start date end date and freq which is B.

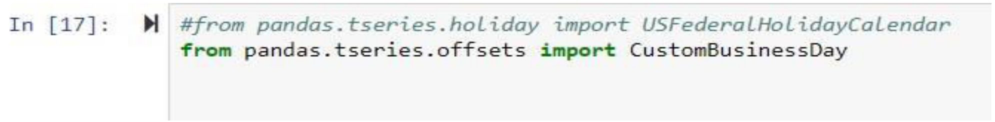


```
In [1]:  import pandas as pd
         df = pd.read_csv("aapl_no_dates.csv")
         df.head()

Out[1]:
```

	Open	High	Low	Close	Volume
0	144.88	145.30	143.10	143.50	14277848
1	143.69	144.79	142.72	144.09	21569557
2	143.02	143.50	142.41	142.73	24128782
3	142.90	144.75	142.90	144.18	19201712
4	144.11	145.95	143.37	145.06	21090636



```
In [2]:  rng = pd.date_range(start="7/1/2017", end="7/21/2017", freq='B')
         rng

Out[2]: DatetimeIndex(['2017-07-03', '2017-07-04', '2017-07-05', '2017-07-06',
                       '2017-07-07', '2017-07-10', '2017-07-11', '2017-07-12',
                       '2017-07-13', '2017-07-14', '2017-07-17', '2017-07-18',
                       '2017-07-19', '2017-07-20', '2017-07-21'],
                      dtype='datetime64[ns]', freq='B')
```

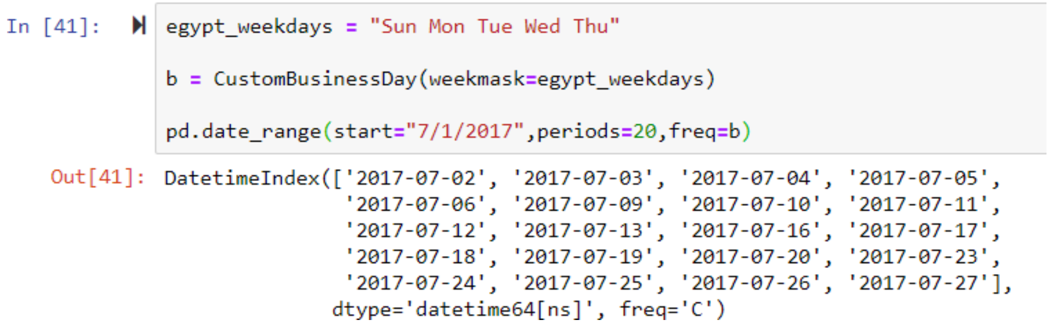


```
In [17]:  #from pandas.tseries.holiday import USFederalHolidayCalendar
           from pandas.tseries.offsets import CustomBusinessDay
```

here , we imported custom businessday.

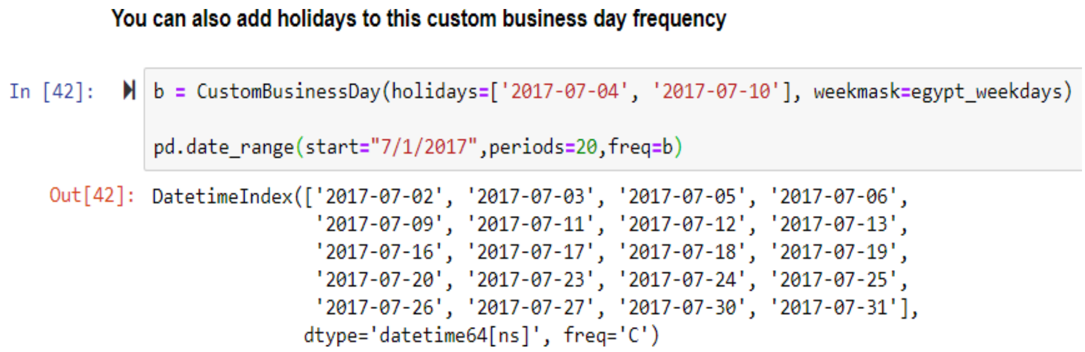
CustomBusinessDay

Weekend in egypt is Friday and Saturday. Sunday is just a normal weekday and you can handle this custom week schedule using CustomBusinessDay with weekmask as shown below



```
In [41]:  egypt_weekdays = "Sun Mon Tue Wed Thu"

          b = CustomBusinessDay(weekmask=egypt_weekdays)

          pd.date_range(start="7/1/2017",periods=20,freq=b)

Out[41]: DatetimeIndex(['2017-07-02', '2017-07-03', '2017-07-04', '2017-07-05',
                        '2017-07-06', '2017-07-09', '2017-07-10', '2017-07-11',
                        '2017-07-12', '2017-07-13', '2017-07-16', '2017-07-17',
                        '2017-07-18', '2017-07-19', '2017-07-20', '2017-07-23',
                        '2017-07-24', '2017-07-25', '2017-07-26', '2017-07-27'],
                       dtype='datetime64[ns]', freq='C')
```

here , we have defined Egypt_weekdays with starting range 7/1/2017 upto 20 periods so we got output by dropping weekends .



You can also add holidays to this custom business day frequency

```
In [42]:  b = CustomBusinessDay(holidays=['2017-07-04', '2017-07-10'], weekmask=egypt_weekdays)

          pd.date_range(start="7/1/2017",periods=20,freq=b)

Out[42]: DatetimeIndex(['2017-07-02', '2017-07-03', '2017-07-05', '2017-07-06',
                        '2017-07-09', '2017-07-11', '2017-07-12', '2017-07-13',
                        '2017-07-16', '2017-07-17', '2017-07-18', '2017-07-19',
                        '2017-07-20', '2017-07-23', '2017-07-24', '2017-07-25',
                        '2017-07-26', '2017-07-27', '2017-07-30', '2017-07-31'],
                       dtype='datetime64[ns]', freq='C')
```

here , we defined 2017-07-04 and 2017-07-10 as holidays so we got output by dropping holidays along with week ends.

… Happy Learning ! ...

References :

- https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe- python
- https://www.tutorialspoint.com/python_pandas/index.htm
- https://www.javatpoint.com/python-pandas
- https://www.edureka.co/blog/python-pandas-tutorial/

Channel Link For more detail:

- https://www.youtube.com/playlist?list=PLeo1K3hjS3uuASpe-1LjfG5f14Bnozjwy