Pandas

Table of Contents

|  |  |
| --- | --- |
| Chapter - 1 : Introduction To Pandas  1. What is Pandas? 2. What are Pandas Data Frames? 3. Advantages of Pandas |  |
| Chapter - 2 : Data Structures in Pandas  1. Introduction 2. Basic Operations |  |
| [Chapter - 3 :](https://docs.google.com/document/d/18IYBEswvdZaOeQXqntmas-Pc9MjOSI_x5JhyTv6Y2rE/edit" \l "heading=h.qdoh2tpoxc5b)Reading & saving Dataframes  1. Introduction 2. Reading & Saving |  |
| Chapter - 4 - DataFrame Operations  1. Adding a row/column 2. Deleting a row/column 3. Sorting (ascending/descending) |  |
| Chapter - 5 - Null Handling  1. Finding Nulls - isna(), isnull(), notna() 2. Replacing Nulls - replace(), fillna() |  |
| Chapter - 6 : Aggregation of groups (Group By Function)  1. Introduction 2. Aggregate Functions |  |
| Chapter - 7 : Lambda Functions  1. What are lambda functions? 2. How to use lambda functions? 3. Implementation of lambda functions |  |
| Chapter - 8 : Joining Two Dataframes (Merge & Concat Functions)  1. Introduction 2. Join 3. Concat Function |  |
| Chapter - 9 : Basic Functions  1. Unique() 2. nunique() 3. value\_counts() 4. describe() 5. isin() |  |

# Introduction To Pandas

You can refer to the Jupyter Notebook attached [here](https://drive.google.com/file/d/1p9QjEuuoPokvhWch-iaysusustE63Qd-/view?usp=sharing). Make sure you create a copy of this in your drive before executing the same.

## What is Pandas?

**Pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of thepython programming language.

Pandas is quite a game changer when it comes to **analyzing data** with Python and it is one of the most preferred and widely used tools in**data munging/**[**wrangling**](https://docs.google.com/document/d/1YQlD-3l3UhFKH0MFduBPVPXOLZ_aJ9mjiVpojdFvCIE/edit#bookmark=id.ie2j9cckql3o). Pandas is an open source, free to use and it was originally written by Wes McKinney .

What’s cool about Pandas is that it takes data (like a CSV or TSV file, or a SQL database) and creates a Python object with rows and columns called data frame that looks very similar to a table in a statistical software (like Excel).

### Importing Pandas :

After the pandas have been installed into the system, you need to import the library. This module is generally imported as:

****

Here, pd is referred to as an alias to the Pandas. However, it is not necessary to import the library using the alias, it just helps in writing less code every time a method or property is called.

## What are Pandas Data Frames?

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

In general, we can say that the Pandas DataFrame consists of three main components: the data, the index, and the columns. DataFrames are extremely important going forward, as we can read & store excel sheets into DataFrames and use many manipulation techniques on them, as we’ll learn ahead.

## Advantages of Pandas

1. Fast and efficient for manipulating and analyzing data.
2. Data from different file objects can be loaded.
3. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
4. Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
5. Data set merging and joining.
6. Flexible reshaping and pivoting of data sets
7. Provides time-series functionality.
8. Powerful group by functionality for performing split-apply-combine operations on data sets.

# Data Structures in Pandas

## Introduction

### Series :

Pandas Series is a one-dimensional labeled array capable of holding any data type. So, in terms of Pandas DataStructure, A Series represents a single column in memory, which is either independent or belongs to a Pandas DataFrame.

#### 

#### **Why do we use series in pandas?**

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

### Dataframe :

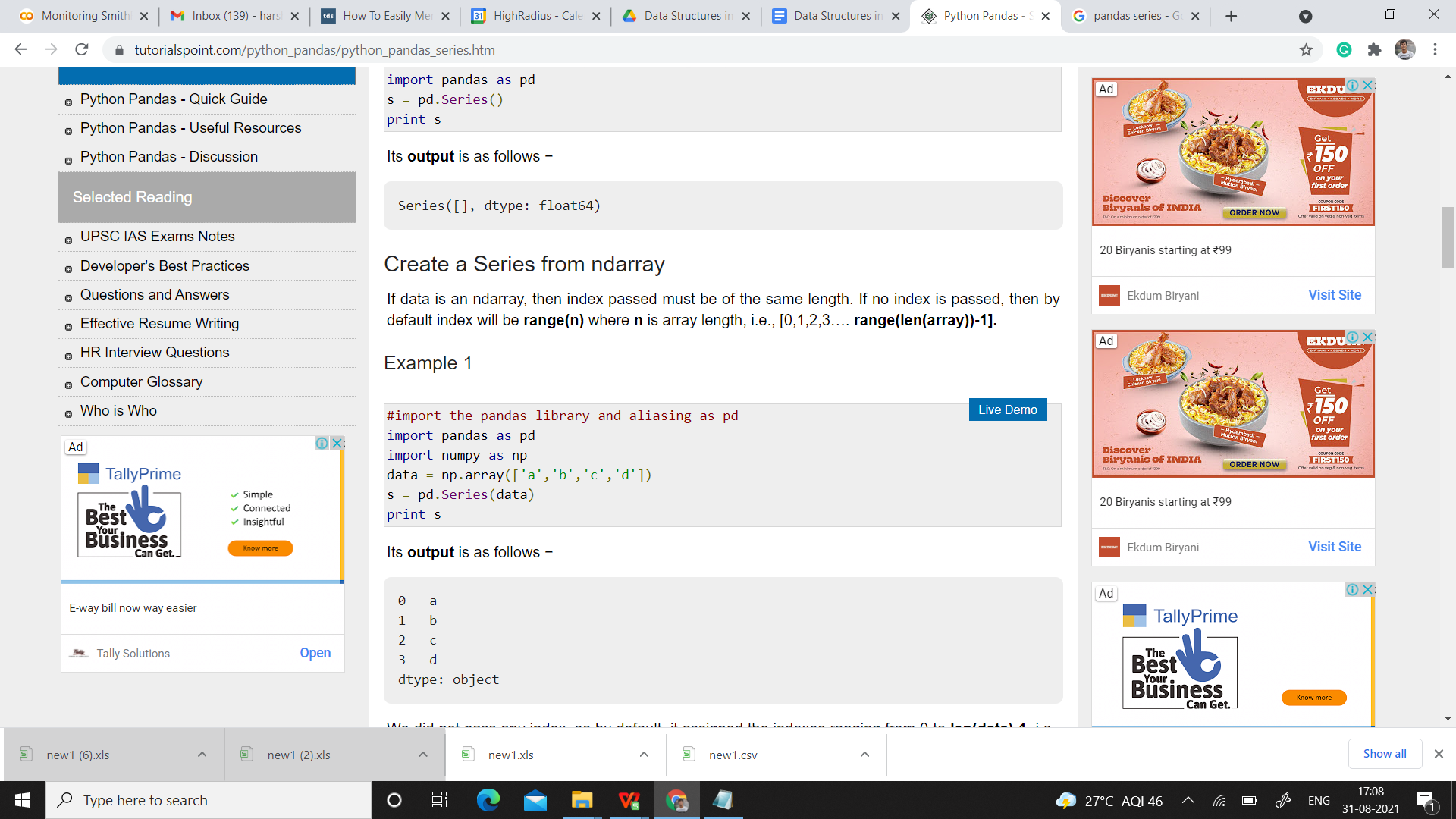
Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes known as rows and columns. a dataframe is a collection of series that can be used to analyse the data. DataFrame can be created from the lists, dictionary, and from a list of dictionaries etc.

## Basic Operations

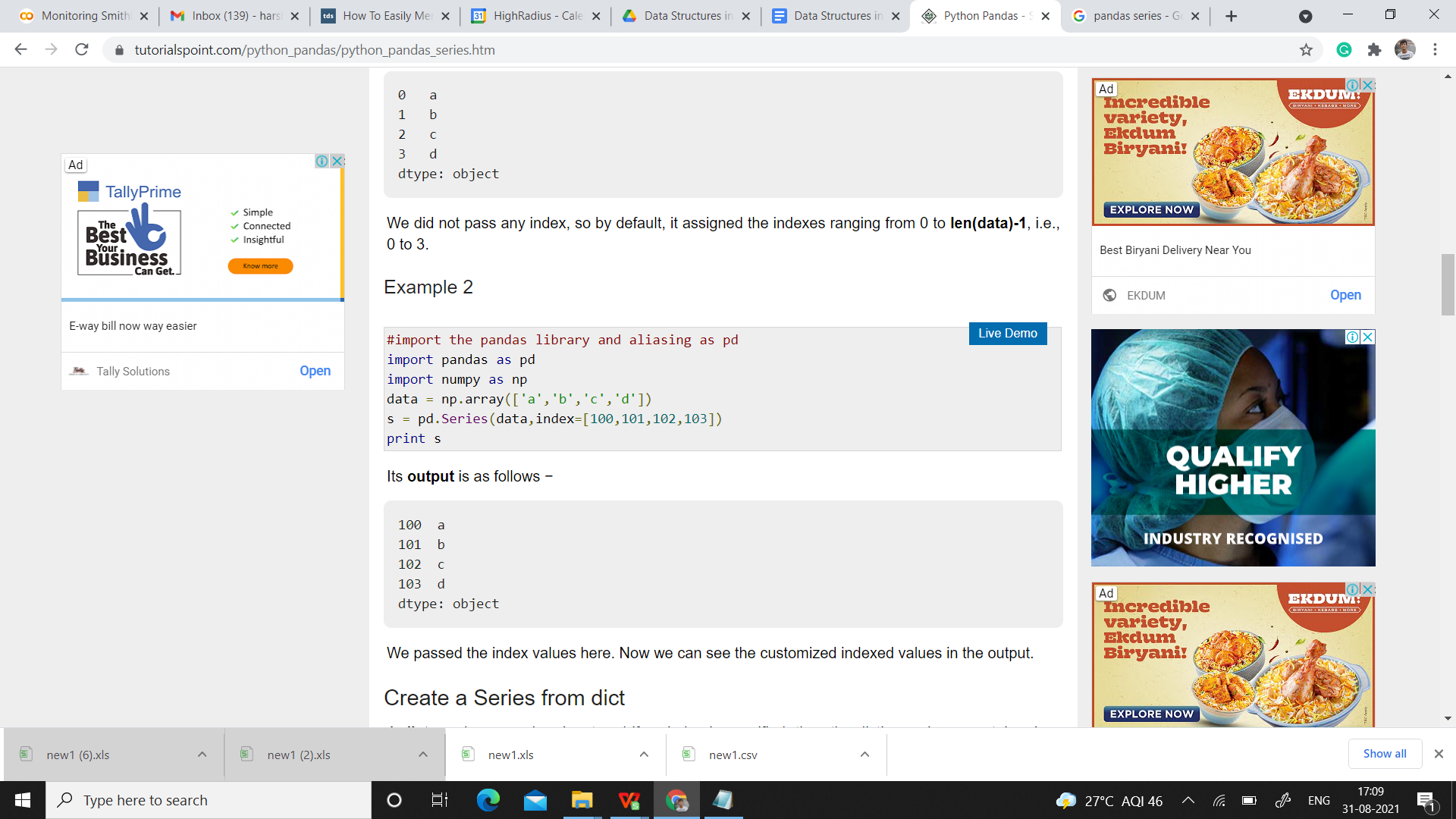
### Series -

**Creating a Series -**

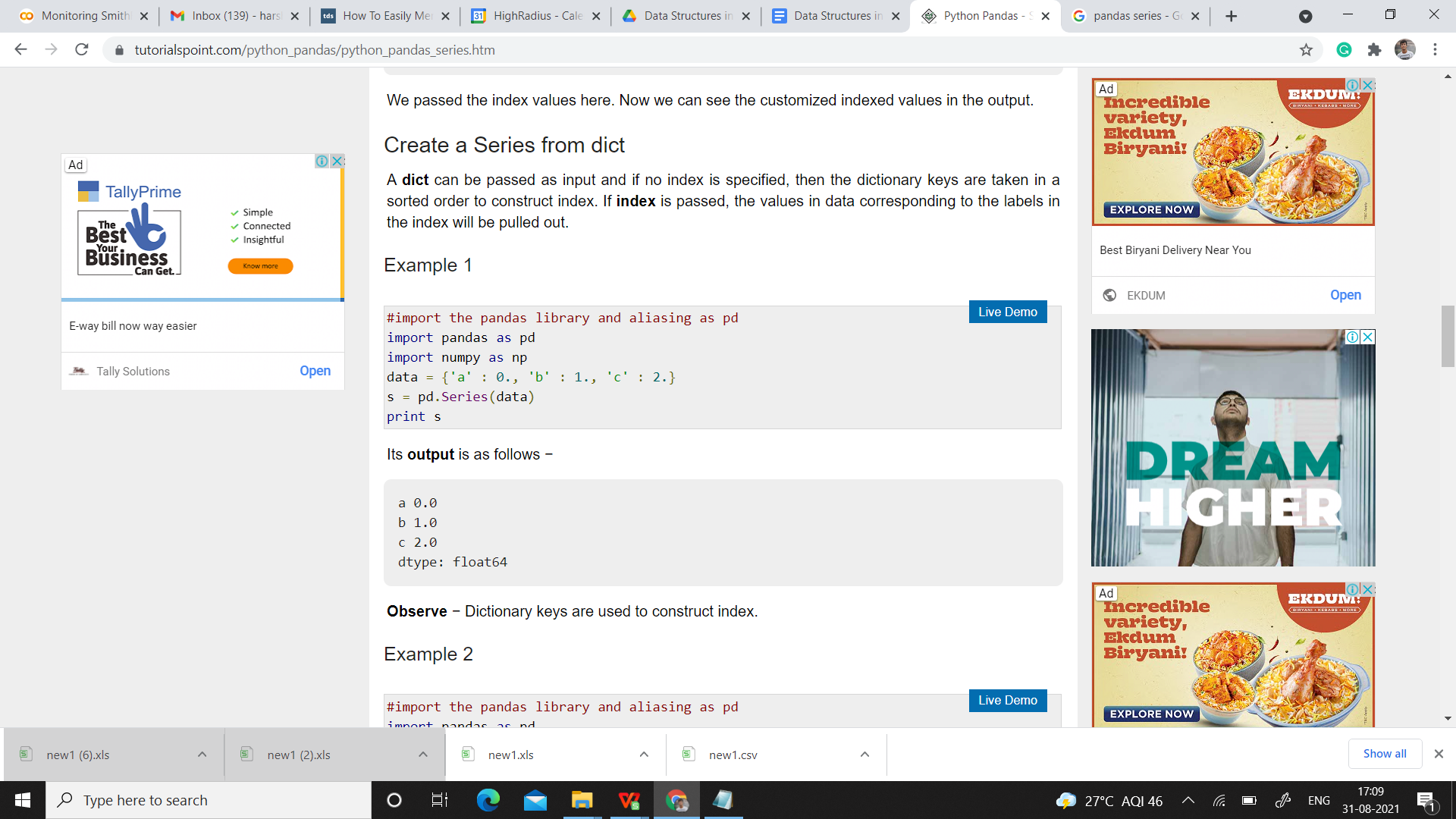
Using Default Indexing.

****

Using Custom Indexing**,**

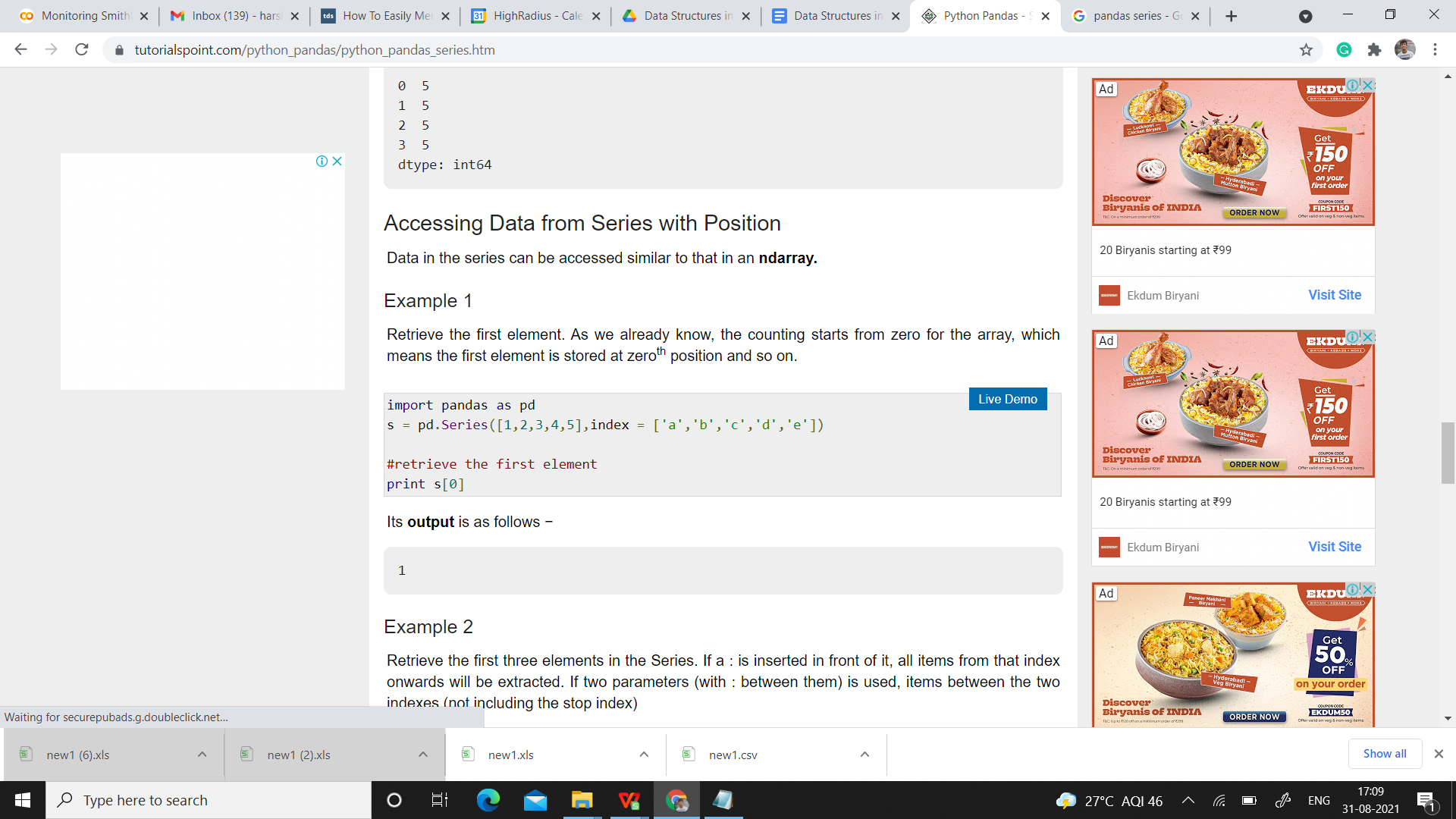
****

**Creating a Series from Dictionary -**

****

**Accessing Data from Series -**

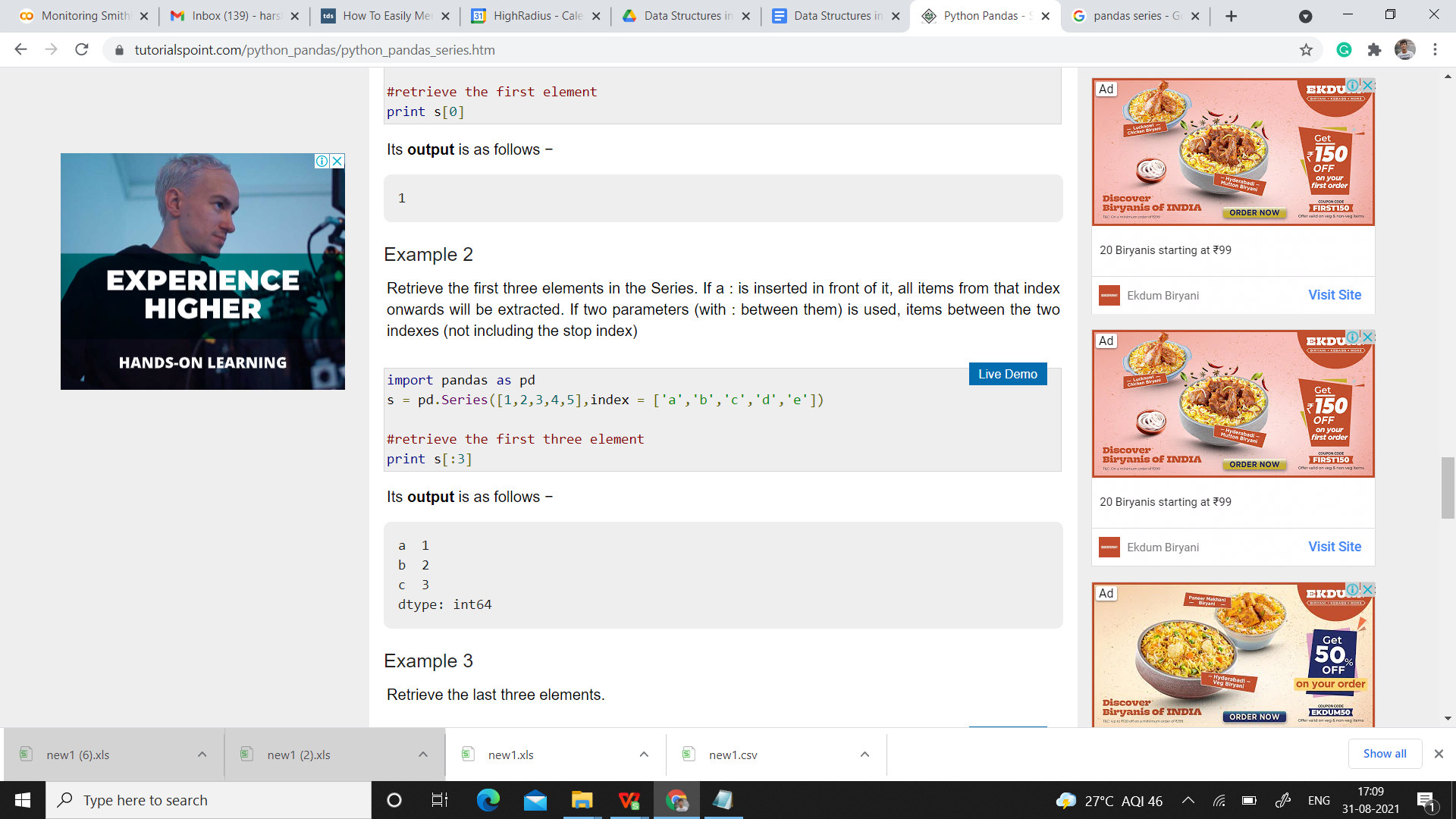
Using Position.



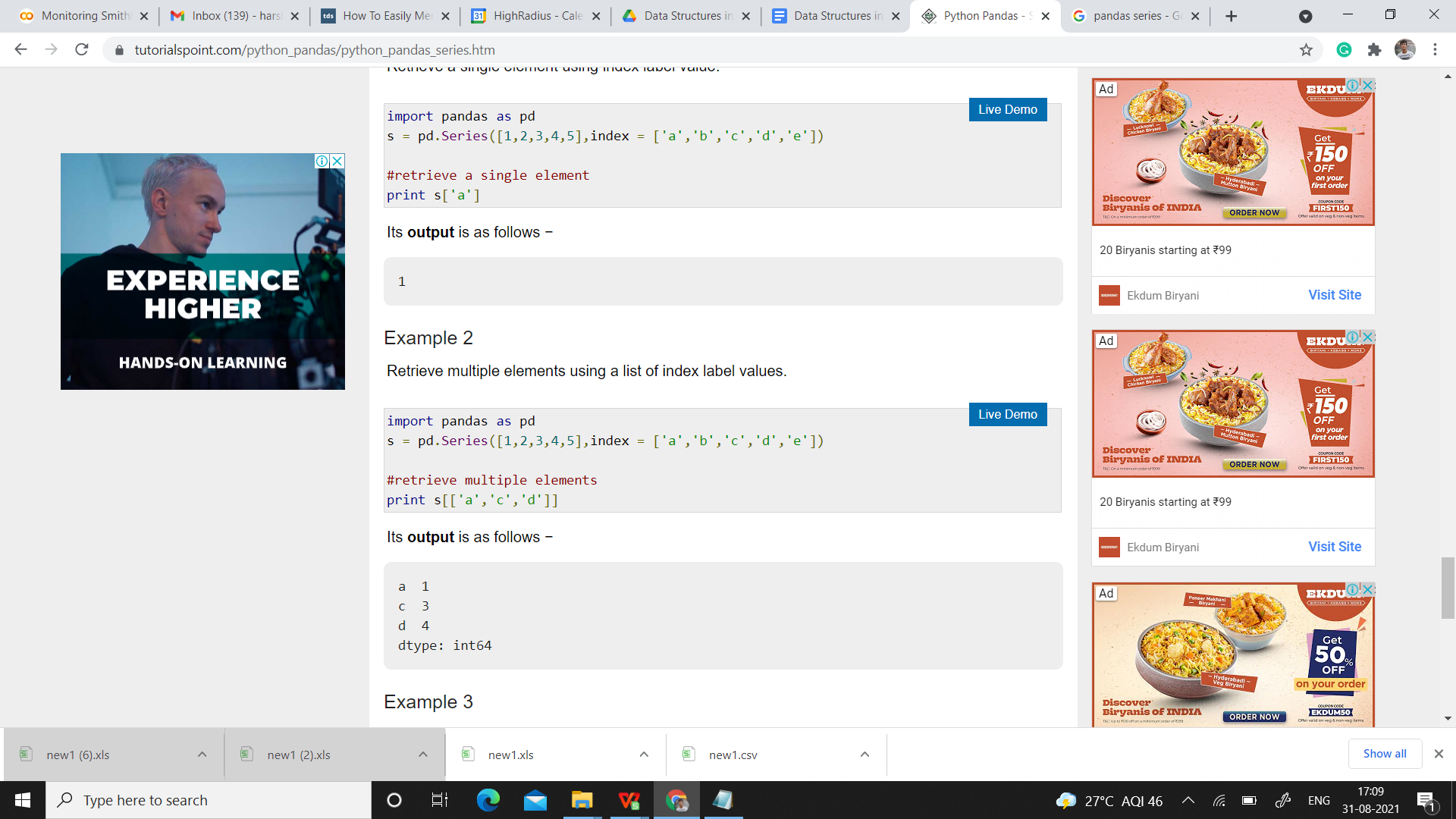
Using Index labels.



Accessing First 3 Elements:-



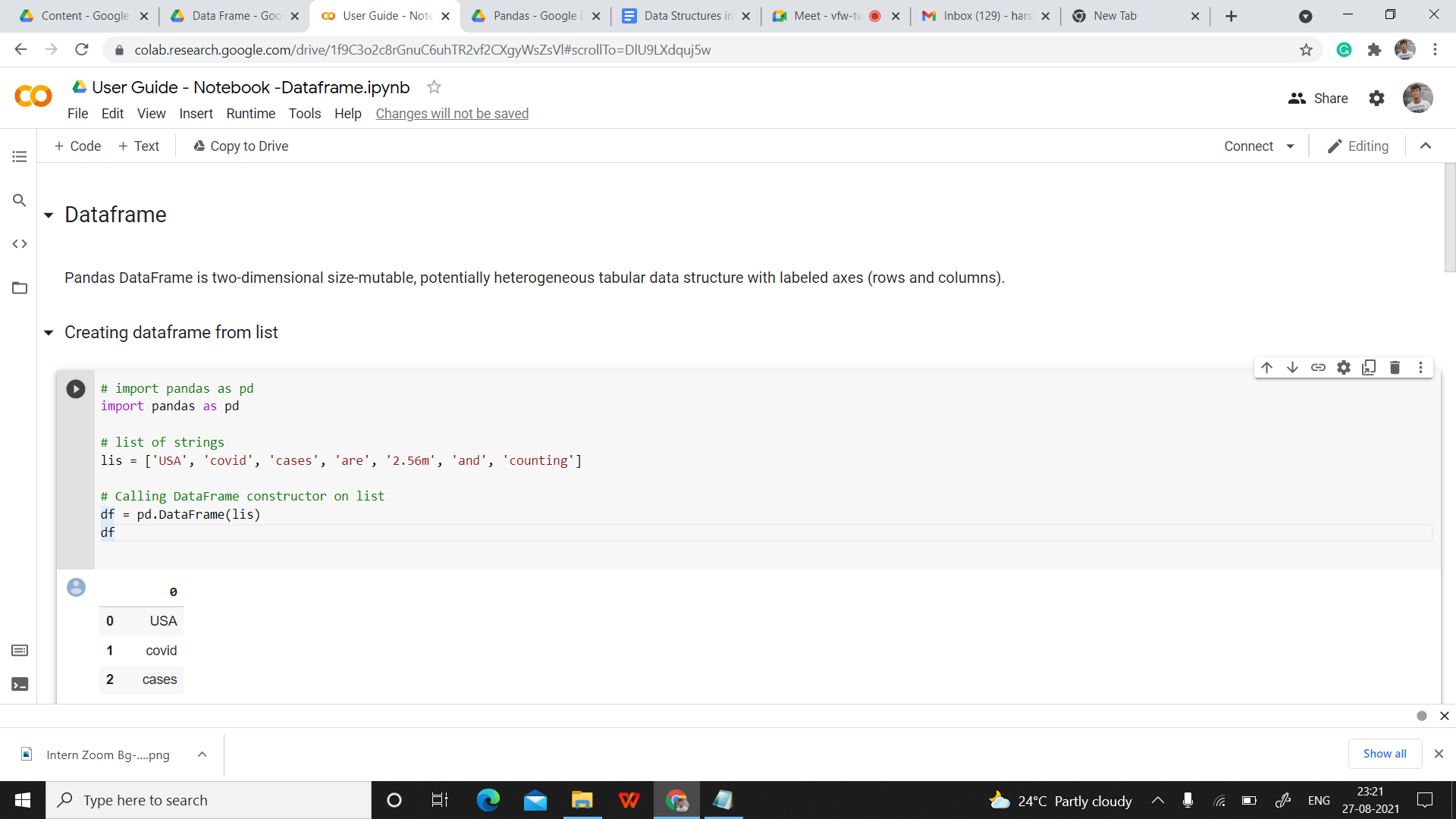
Accessing Multiple Elements using Index Label.



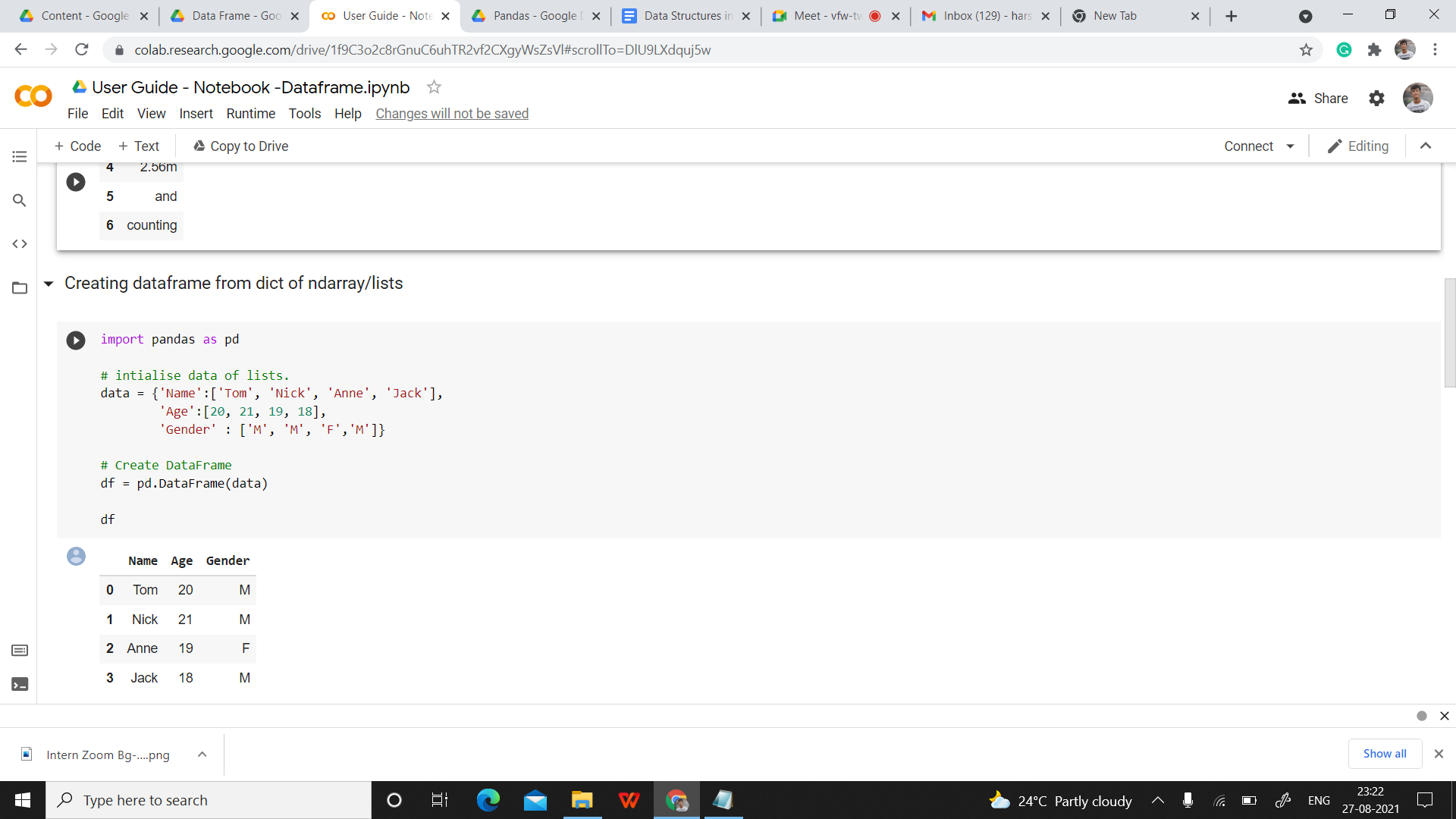
## 

### Dataframe :

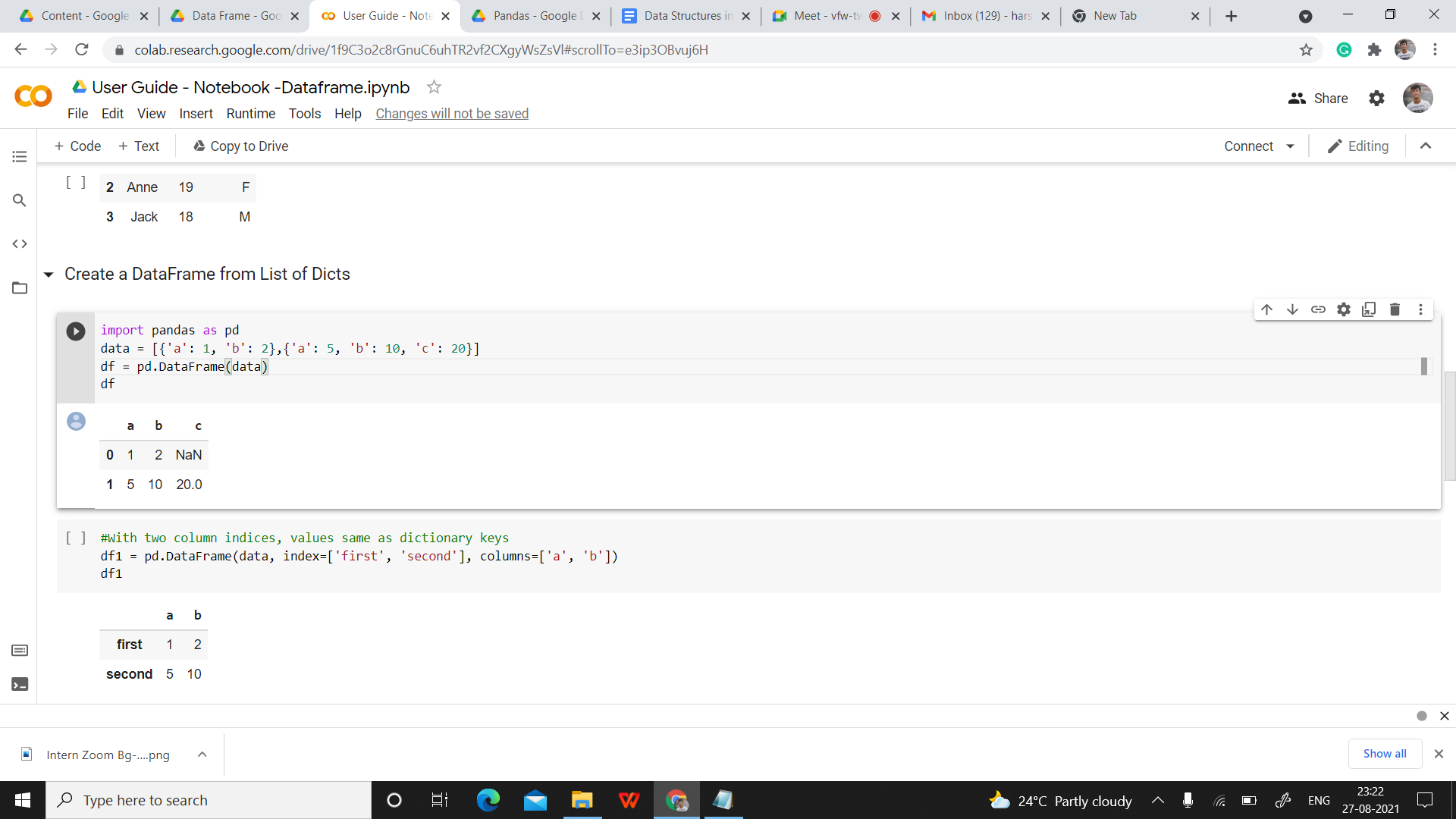
**Creating Dataframe using list -**

****

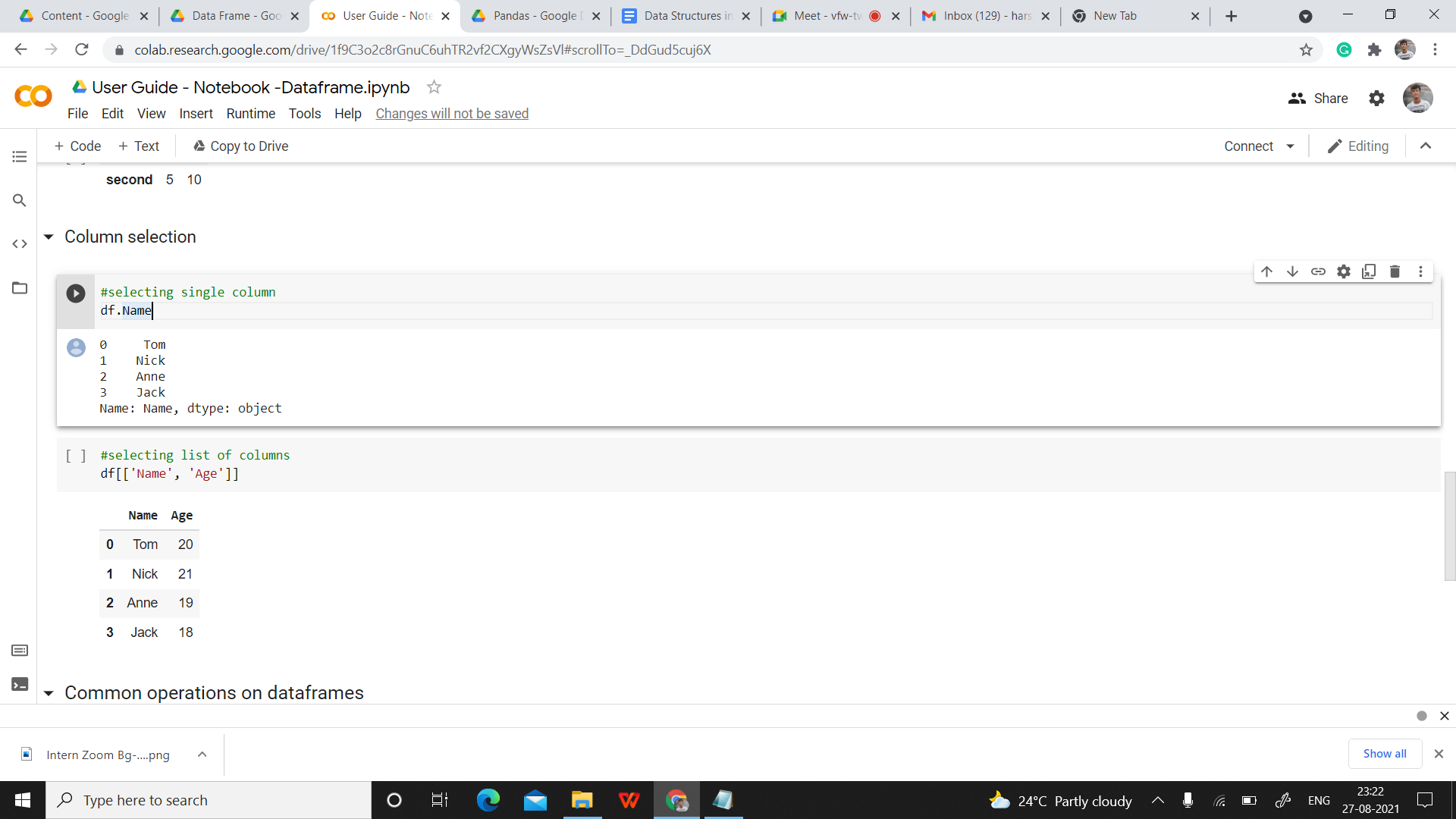
#### **Creating dataframe from dict of ndarray/lists -**



**Create a DataFrame from List of Dicts -**

****

#### **Column selection -**



Other Dataframe Operations can be found [here](https://ujjwalkarn.me/2016/05/30/common-operations-on-pandas-dataframe/).

# Introduction to Reading and Saving Dataframes :

## Introduction

## 

### CSV

A [comma-separated values (CSV)](https://en.wikipedia.org/wiki/Comma-separated_values) file is a plaintext file with a .csv extension that holds tabular data. This is one of the most popular file formats for storing large amounts of data. Each row of the CSV file represents a single table row. The values in the same row are by default separated with commas, but you could change the separator to a semicolon, tab, space, or some other character.

Eg:

****

### Excel

Excel is a popular spreadsheet program used with data like numbers and formulas, text, and drawing shapes. Excel is part of the Microsoft Office Suite of software. XLS files use a Binary Interchange File Format to store spreadsheet data and are proprietary to Microsoft.

Eg:

## How to retrieve data from one excel worksheet and extract to a new excel file in

## Reading & Saving

### 

### CSV

##### **Read a CSV File**

##### Once your data is saved in a CSV file, you’ll likely want to load and use it from time to time. You can do that with the Pandas [read\_csv()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html) function, by passing the destination of the csv file in the function.

##### **Eg:**

df = pd.read\_csv('data.csv', index\_col=0)

In this case, the Pandas read\_csv() function returns a new DataFrame with the data and labels from the file data.csv, which you specified with the first argument. This string can be any valid path, including [URLs](https://en.wikipedia.org/wiki/URL).

##### **Write a CSV File**

You can save your Pandas DataFrame as a CSV file with [.to\_csv()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html):

**Eg:**

df.to\_csv('data.csv')

### Excel

##### **Read an Excel File**

You can load data from Excel files with [read\_excel()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html):

Eg**:**

df = pd.read\_excel('data.xlsx', index\_col=0)

read\_excel() returns a new DataFrame that contains the values from data.xlsx.

##### **Write an Excel File**

Once you have those packages installed, you can save your DataFrame in an Excel file with [.to\_excel()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_excel.html)

Eg**:**

df.to\_excel('data.xlsx')

The argument 'data.xlsx' represents the target file and, optionally, its path. The above statement should create the file data.xlsx in your current working directory.

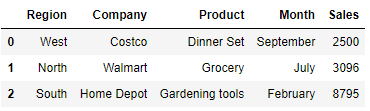
# Dataframe Operation:

## Adding a row/column

### Adding a row

You may want to add another row to a dataframe in many scenarios. Here’s how you can do it

**Eg:**



A data dictionary with the values of one Region - East that we want to enter in the above dataframe (df).

The data is basically a list with a dictionary having columns as keys and their corresponding values.

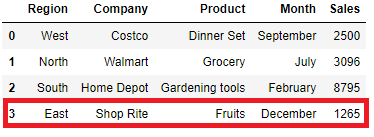
data=[{'Region':'East','Company':'Shop Rite','Product':'Fruits','Month':'December','Sales': 1265}]

##### **First Way - Using append function**

##### It basically creates a new dataframe object with the new data row at the end of the dataframe. The old dataframe will be unchanged.

**Eg:**

df.append(data,ignore\_index=True,sort=False)



##### **Second Way - Using loc:**

loc is used to access a group of rows and columns by labels or a boolean array. However with an assignment(=) operator you can also set the value of a cell or insert a new row all together at the bottom of the dataframe.

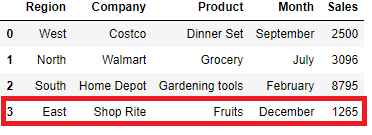
**Eg:**

>>> df.loc[3]=list(data[0].values())

Or

>>> df.loc[len(df.index)]=list(data[0].values())

>>> df



##### **Third Way - Using iloc**

##### We can replace a row with the new data as well using iloc, which is integer-location based indexing for selection by position. In our original dataframe we want to replace the data for North Region with the new data of the East Region. Update the row at index position 1 using iloc and list values of the data dictionary i.e

list(data[0].values()) = ['East','Shop Rite','Fruits','December',1265]

**Eg:** Update row at Index position 1

>>>df.iloc[1]=list(data[0].values())

>>>df

## 

### Adding columns :

Pandas allows users to add a new column by initializing on the fly. For example: the list below is the purchase value of three different regions i.e. West, North and South. We want to add this new column to our existing dataframe above

**Eg:**

>>> purchase = [3000, 4000, 3500]

>>> df.assign(Purchase=purchase)

Or

>>> df[“Purchase”] = purchase

## 

### Add Multiple Columns to a Dataframe

Lets add these three list (Date, City, Purchase) as column to the existing dataframe using assign with a dict of column names and values

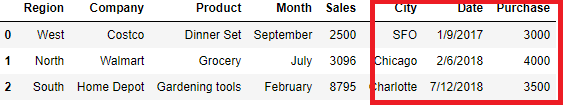
**Eg:**

>>> Date = ['1/9/2017','2/6/2018','7/12/2018']

>>> City = ['SFO', 'Chicago', 'Charlotte']

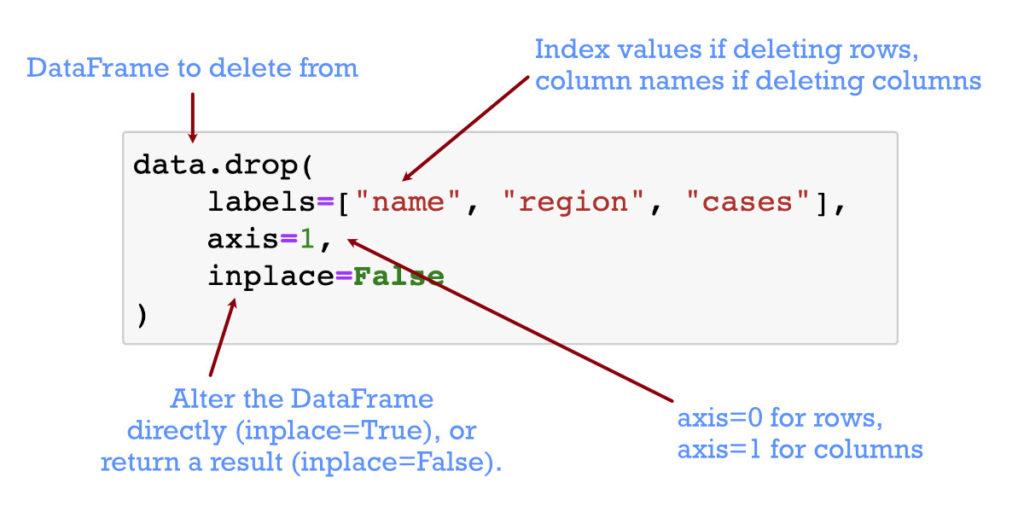
>>> Purchase = [3000, 4000, 3500]

>>> df.assign(\*\*{'City' : City, 'Date' : Date,'Purchase':Purchase})



## Deleting a row/column :

The Pandas “drop” function is used to delete columns or rows from a Pandas DataFrame.



**Eg:**

Creating dataframe df, to perform operations.

>>> df = pd.DataFrame(np.arange(12).reshape(3, 4), columns=['A', 'B', 'C', 'D'])

>>>df

A B C D

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

### Dropping columns:

>>>df.drop(['B', 'C'], axis=1)

output:

A D

0 0 3

1 4 7

2 8 11

>>> df.drop(columns=['B', 'C'])

output:

A D

0 0 3

1 4 7

2 8 11

### Drop a row by index

>>> df.drop([0, 1])

output:

A B C D

2 8 9 10 11

## Sorting a DataFrame

## To sort the DataFrame based on the values in a single column, use .sort\_values(). By default, this will return a new DataFrame sorted in ascending order. It does not modify the original DataFrame.

**Syntax:**

DataFrame.sort\_values(*by*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na\_position='last'*, *ignore\_index=False*, *key=None)*

*Axis: {0 or ‘index’, 1 or ‘columns’}, default 0.*

*Ascending :* bool or list of bool, default True.

Inplace : bool, default False . If True, perform operation in-place.

### Sorting by a Column in Ascending Order

## To use .sort\_values(), pass a single argument to the method containing the name of the column you want to sort by.

## In this example, you sort a DataFrame by the city08 column,

>>> df.sort\_values("city08")

Output:

city08 cylinders fuelType ... mpgData trany year

99 9 8 Premium ... N Automatic 4-spd 1993

1 9 12 Regular ... N Manual 5-spd 1985

80 9 8 Regular ... N Automatic 3-spd 1985

47 9 8 Regular ... N Automatic 3-spd 1985

3 10 8 Regular ... N Automatic 3-spd 1985

.. ... ... ... ... ... ... ...

9 23 4 Regular ... Y Automatic 4-spd 1993

8 23 4 Regular ... Y Manual 5-spd 1993

7 23 4 Regular ... Y Automatic 3-spd 1993

76 23 4 Regular ... Y Manual 5-spd 1993

2 23 4 Regular ... Y Manual 5-spd 1985

[100 rows x 10 columns]

This sorts your DataFrame using the column values from city08,

By default, .sort\_values() sorts your data in ascending order.

### Sorting by a Column in Descending Order

To sort the DataFrame in descending order, pass “ascending=False” as one of the parameter as shown in the example below,

**Eg:**

>>> df.sort\_values( by="city08", ascending=False)

Output:

city08 cylinders fuelType ... mpgData trany year

9 23 4 Regular ... Y Automatic 4-spd 1993

2 23 4 Regular ... Y Manual 5-spd 1985

7 23 4 Regular ... Y Automatic 3-spd 1993

8 23 4 Regular ... Y Manual 5-spd 1993

76 23 4 Regular ... Y Manual 5-spd 1993

.. ... ... ... ... ... ... ...

58 10 8 Regular ... N Automatic 3-spd 1985

80 9 8 Regular ... N Automatic 3-spd 1985

1 9 12 Regular ... N Manual 5-spd 1985

47 9 8 Regular ... N Automatic 3-spd 1985

99 9 8 Premium ... N Automatic 4-spd 1993

[100 rows x 10 columns]

# Null Handling

There are numerous ways to detect the presence of missing values in a dataset.

## isna()

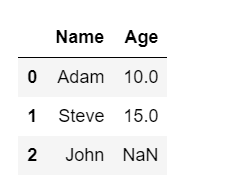
The isna() function is used to check missing (null) values. It is a boolean function that looks for the missing values and returns TRUE where it detects a missing value.

Look at the below syntax:

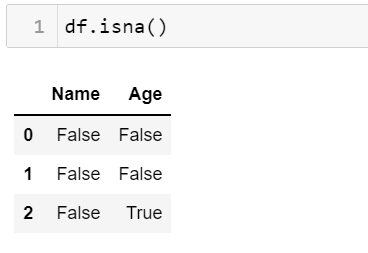
**dataframe.isna()**

In this example below, we have made use of .isna() function to check for the presence of missing values. The cell of the dataframe containing missing values only returns TRUE, otherwise it returns FALSE.

**Dataframe:**

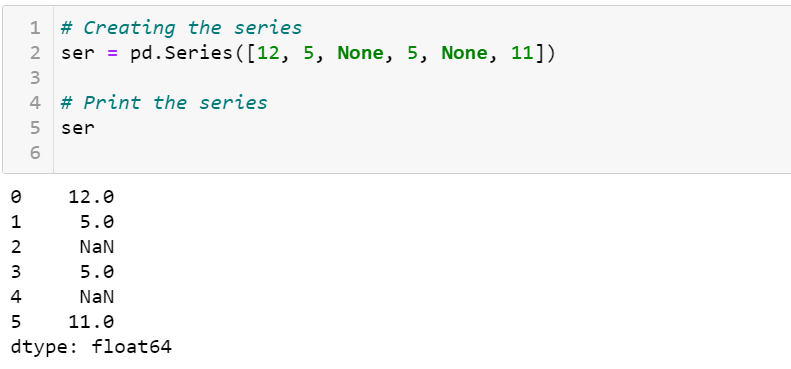


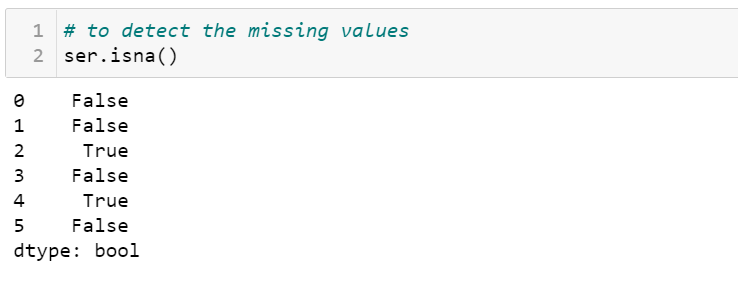
**Checking missing values using isna():**



**We can also use isna() in pandas series.**

**Example 2:** Using isna() function to detect missing values in a pandas series object.



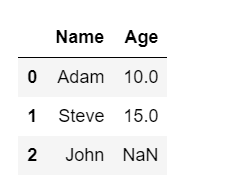


It returned True for all the missing values in the series.

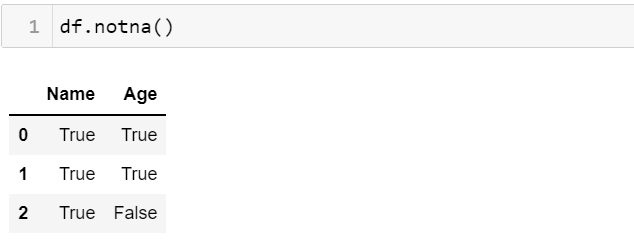
## notna()

Similarly We have notna(). With notna() function, we can easily pick out data that does not occupy missing values or NA values. The notna() function returns TRUE, if the data is free from missing values else it returns FALSE (if null values are encountered).

**Dataframe:**



**Checking missing values using notna():**



It returned False only for the Age of John (which was the only missing value in the dataframe).

## isnull()

**isnull()** function also works the same as **isna()**, detects missing values in the given series or dataframe. It returns a boolean same-sized object indicating if the values are NA (null). Missing values get mapped to True and non-missing values get mapped to False.

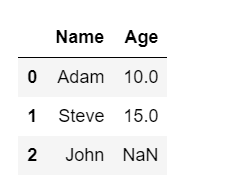
Look at the below syntax:

**Series.isnull() Or,**

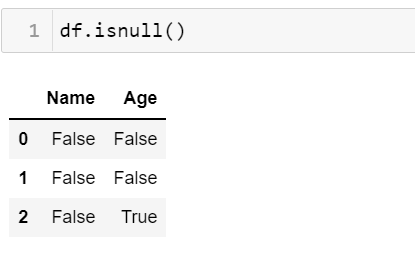
**Dataframe.isnull()**

We will take the same example for easy understanding,

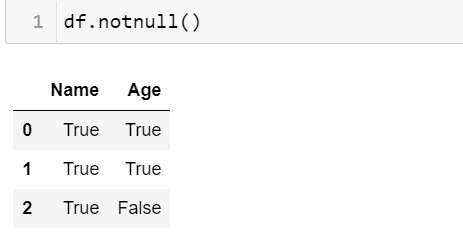
**Dataframe:**



**Checking missing values using isnull():**



Similar to notna(), there is also **notnull()** function in pandas.



## fillna()

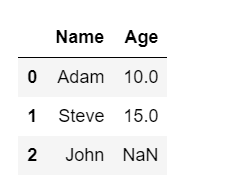
fillna() manages and lets the user replace NaN values with some value of their own.

Syntax:

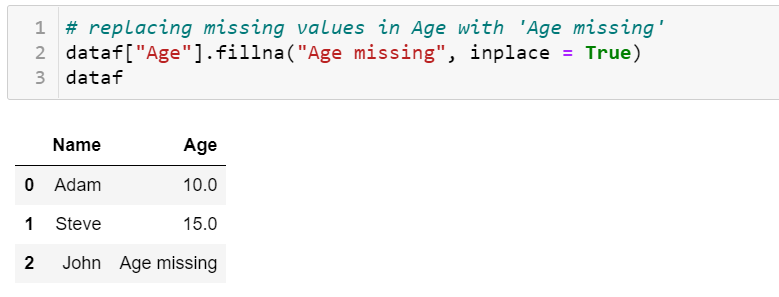
**dataframe["Column name"].fillna("Value", inplace = True)**

Eg:

**Dataframe:**



We will replace missing values in Age (Age of john) with ‘Age missing’



## replace()

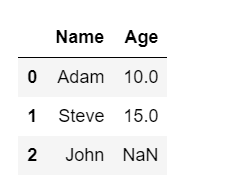
Pandas replace() is a very rich function that is used to replace a string, regex, dictionary, list, and series from the DataFrame. The values of the DataFrame can be replaced with other values dynamically. It is capable of working with the Python regex (regular expression). It differs from updating with .loc or .iloc, which requires you to specify a location where you want to update with some value.

**Syntax:**

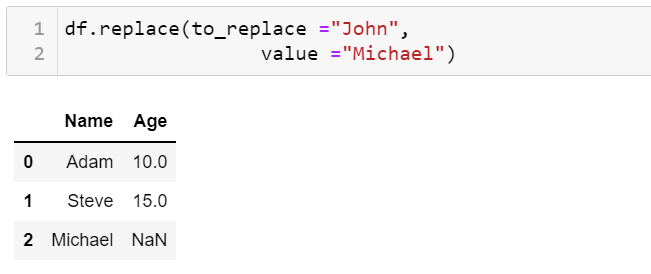
DataFrame.replace(to\_replace= “ ”, value=” ”)

Let us take an example:

**Dataframe:**

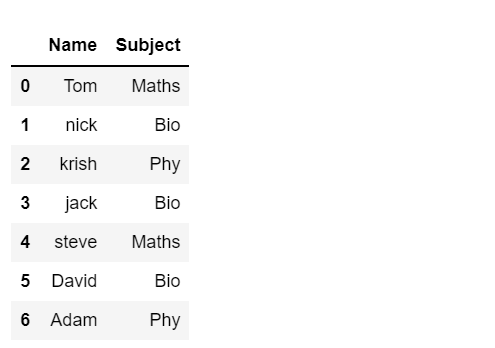


We are going to replace team “John” with “Michael” in the ‘df’ data frame.

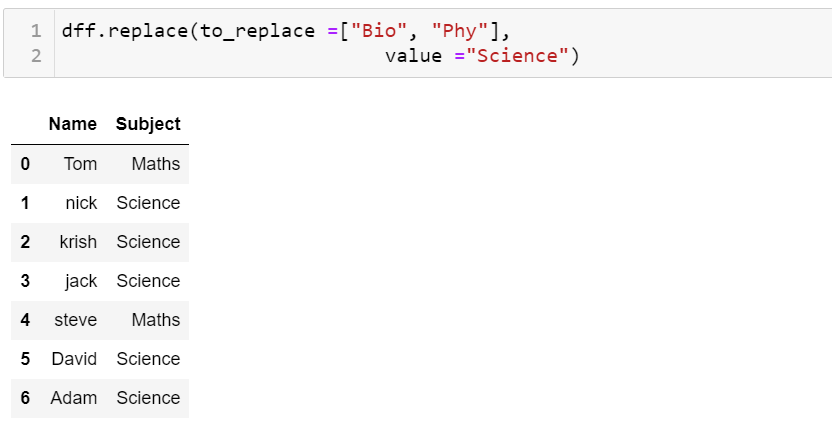


**Example 2: Replacing more than one value at a time.**

Let us take a new dataframe.



Using python list as an argument We are going to replace “Phy” and “Bio” with “Science” in the dataframe.



**Can we replace missing values using replace() function?**

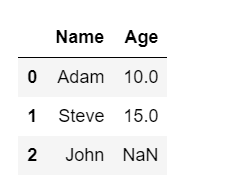
Yes, we can !!!

See how :

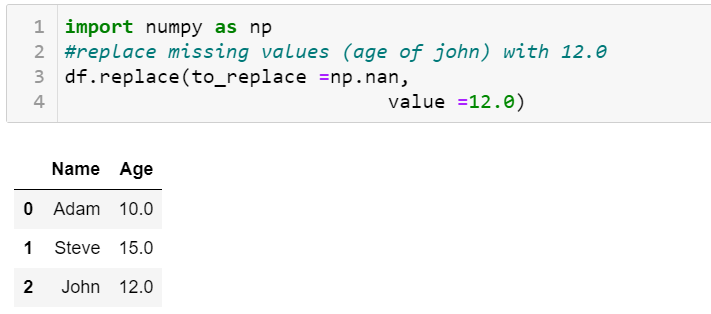
**Example 3:** Replace the Nan value in the data frame with a specific value

Let us try to replace missing values (age of john) with 12.0

**Dataframe:**



We will use numpy.nan for NaN values.



# Aggregation of Groups (Group By) :

## Introduction

Pandas group by is used for grouping the data according to the categories and apply a function to the categories. It also helps to aggregate data efficiently. Pandas dataframe.groupby() function is used to split the data into groups based on some criteria.

In this play, you will learn how to do analysis on groups(individual categories) in a pandas dataframe of a categorical feature. The concept of split-apply-combine, manipulates groups separately based on different rules.

When you are working on a dataset, be it data preparation phase, EDA phase, you will often come upon a situation where you would like to get some statistical inference from the data within each individual category.

**Syntax -**

DataFrame.groupby(by=None, axis=0, level=None, as\_index=True, sort=True, group\_keys=True, squeeze=False)

*Too much to process what I just said? Let’s think with an example.*

For example,

Let’s say you have a dataset where you have records of your friends and the duration of talk you had with that friend each time you met him/her.

|  |  |  |
| --- | --- | --- |
| S.no | Friend | Duration of Meet |
| 1. | A | 10 mins |
| 2. | A | 13 mins |
| 3. | B | 6 mins |
| 4. | B | 2 mins |
| 5. | B | 3 mins |
| 6. | C | 40 mins |
| 7. | D | 1 min |

In real life let’s assume you had about 1000s to 10,000s of such records, now someday you want to just see whom you are more in touch with, whom you talked the most etc. etc.

The result we want to generate would be something like this,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Friend | Average talk time | Frequency of meet | Longest Talk | Shortest talk |
| A | 11.5 mins | 2 | 13 mins | 10 mins |
| B | 3.67 mins | 3 | 6 mins | 2 mins |
| C | 40 mins | 1 | 40 mins | 40 mins |
| D | 1 min | 1 | 1 mins | 1 mins |

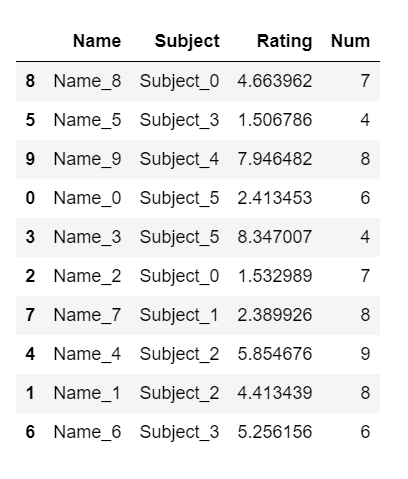
So as you can see we just checked some statistical analysis on group level (your each friend).

## Aggregate Functions

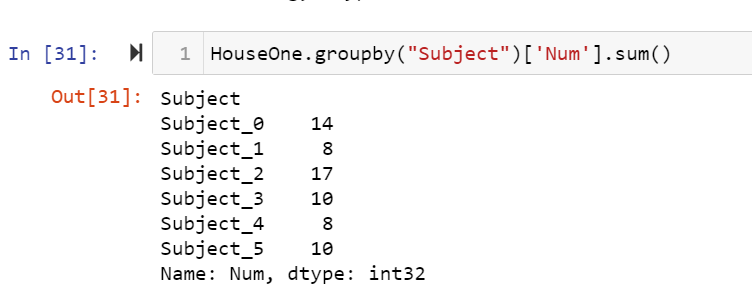
An aggregated function returns a single aggregated value for each group. Once the group by object is created, several aggregation operations can be performed on the grouped data.

**Sum ()**

groupby() function takes up the column name as argument followed by sum() function.



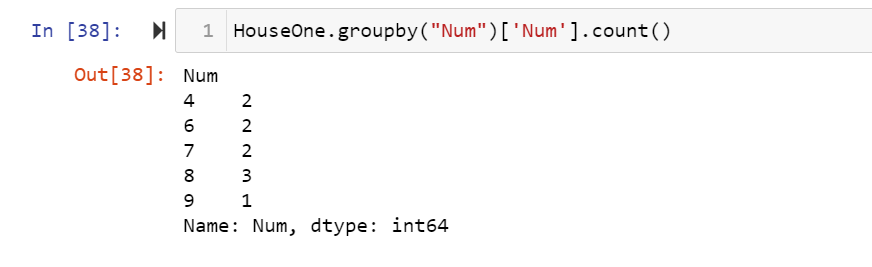
Suppose HouseOne is the name of the dataframe. We are grouping the data by Subject and also calculating the sum of the num column for each subject.



### Count ()

groupby() function takes up the column name as argument followed by count() function.

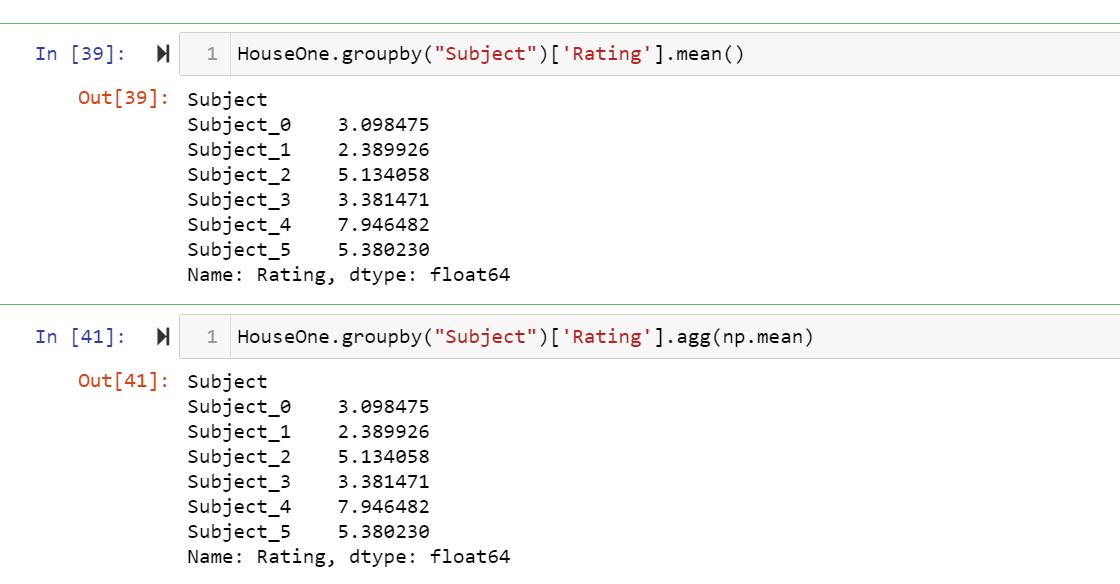
For the same dataframe, we are grouping the data by Num and also calculating the count of the same column. It will five us the frequency of each number in the num column.



### Mean, Median, Mode

groupby() function takes up the column name as an argument followed by the name of the function or by using agg() function.

For the same dataframe,we are grouping the data by Subject and also calculating the mean of the Rating column for each category in the subject.



# Lambda Functions

## What are Lambda functions?

In Python, an anonymous function is a function that is defined without a name.

While defining normal functions, we are using the def keyword in Python, but while defining anonymous functions we are using the lambda keyword.

Hence, anonymous functions also are called Lambda functions.

## How to use lambda Functions in Python?

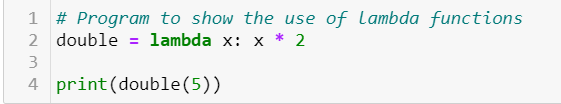
A lambda function in python has the following syntax.

**lambda argument(s): expression**

A lambda function evaluates an expression for a given argument. We give the function a value (argument) and then provide the operation (expression). The keyword lambda must come first. A full colon (:) separates the argument and the expression.

**Eg :**

**Input :**



**Output :**

****

In the above example , **lambda x: x \* 2** is the lambda function. Here **x** is the argument and **x \* 2** is the expression that gets evaluated and returned.

## Implementation of Lambda Function in Python

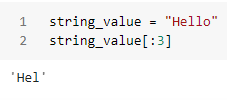
We use lambda functions when we require a nameless function for a short period of time.

While working with DataFrames, lambda functions can help us in applying any operations on individual values of a series, instead of applying the operation on the series as a whole.

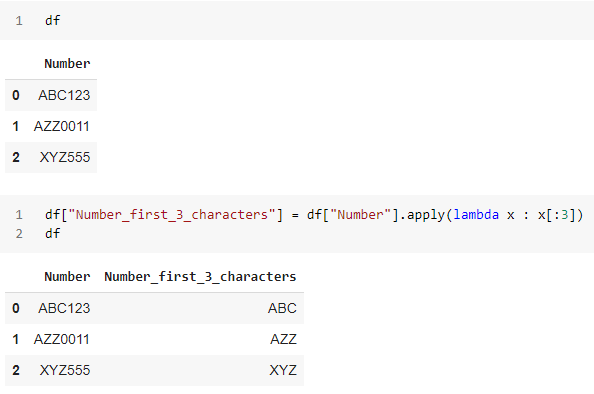
For example -

Let’s say we have a dataframe where a column called Number contains some alpha numeric values, and we want to fetch the first three characters of these values. We can do this easily using lambda functions, which would otherwise be difficult to do.

We already know we can fetch the first 3 characters of a string value like below -



We can apply this same operation on each value of the series/column Number without using loops as shown below,



Similarly, you can use lambda functions to apply any operations on individual values of a Series.

### Additional References :

### [Reference-1](https://towardsdatascience.com/lambda-functions-with-practical-examples-in-python-45934f3653a8)

[Reference-2](https://www.analyticsvidhya.com/blog/2021/06/anonymous-or-lambda-functions-in-python-a-beginners-guide/)

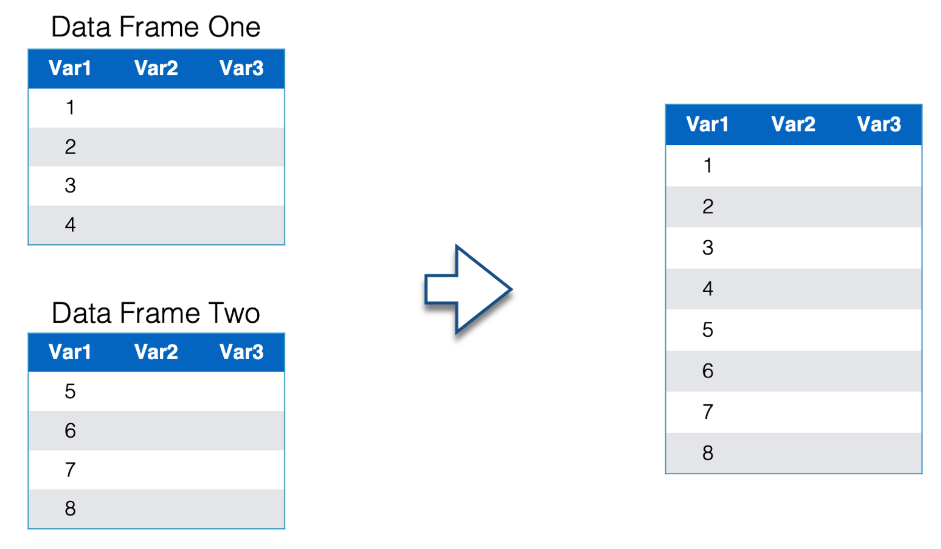
# 

# Joining DataFrames

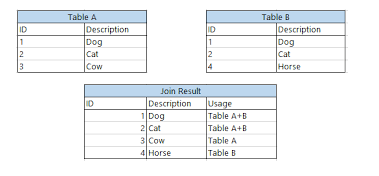
## Introduction

There are many times when you’d need to merge/join two dataframes either horizontally or vertically.

### Vertical Join (Concat Function)



### Horizontal Join (Merge Function)



## Join

Join is not only one of the most important concepts to master for data manipulation but also one of the easiest.

Let us start by understanding different types of joins,

1) Left Join

2) Right Join

3) Inner Join

4) Outer Join

Left Join, returns all records from the left table and the matched results from the right table.

In Simpler Words with an example: You have a table consisting of student names and Respective marks. In another table, you have other details of the students which can be helpful for you. The easiest way to lookup for these details is by performing a simple Left Join.

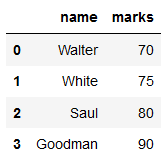


Table A(Left) : Names and their Corresponding marks.

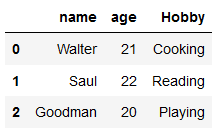
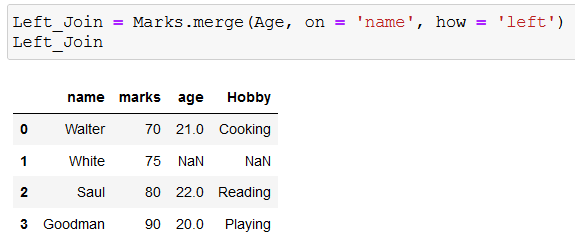
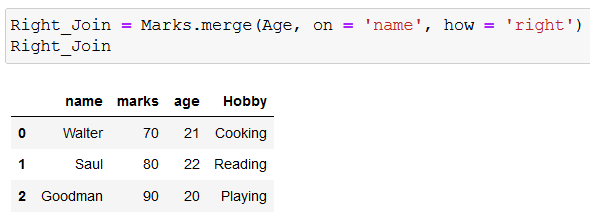


Table B(Right) : Names and the other details associated with the Students.



Performed a simple Left Join to bring all the details in the same table. This is a very useful operation when you need to look up the corresponding details associated with an id that exists in a different table. Like in this case, id is ‘name’ and ‘age’ and ‘hobby’ associated with it exists in a different table.

Right Join, Very Similar to the Left Join, but it returns all the records from the right table and the matched records from the left table.



As we can see, by using right join, we can populate the right table (Age) with marks from left table (Marks). Both Left Join and Right Join are very similar in nature.

On the Contrary, Inner Join is slightly different from both of the other joins. It returns a table which have ids (name in our case), which exists in both the tables, in technical terms, it returns the intersection of the both the table based on the id (name).

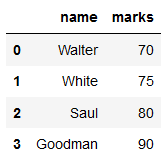


Table A (Marks) : Has an extra id of ‘white’.

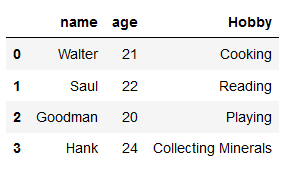
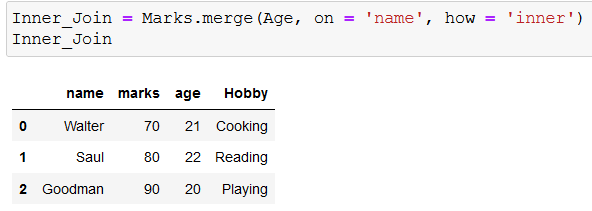


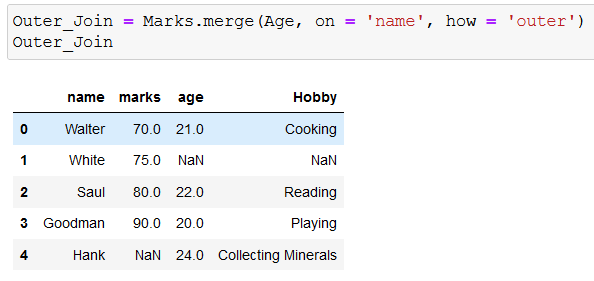
Table B (Age) : Has an extra id of ‘Hank’.

Note : By extra it means, not common in both the tables.



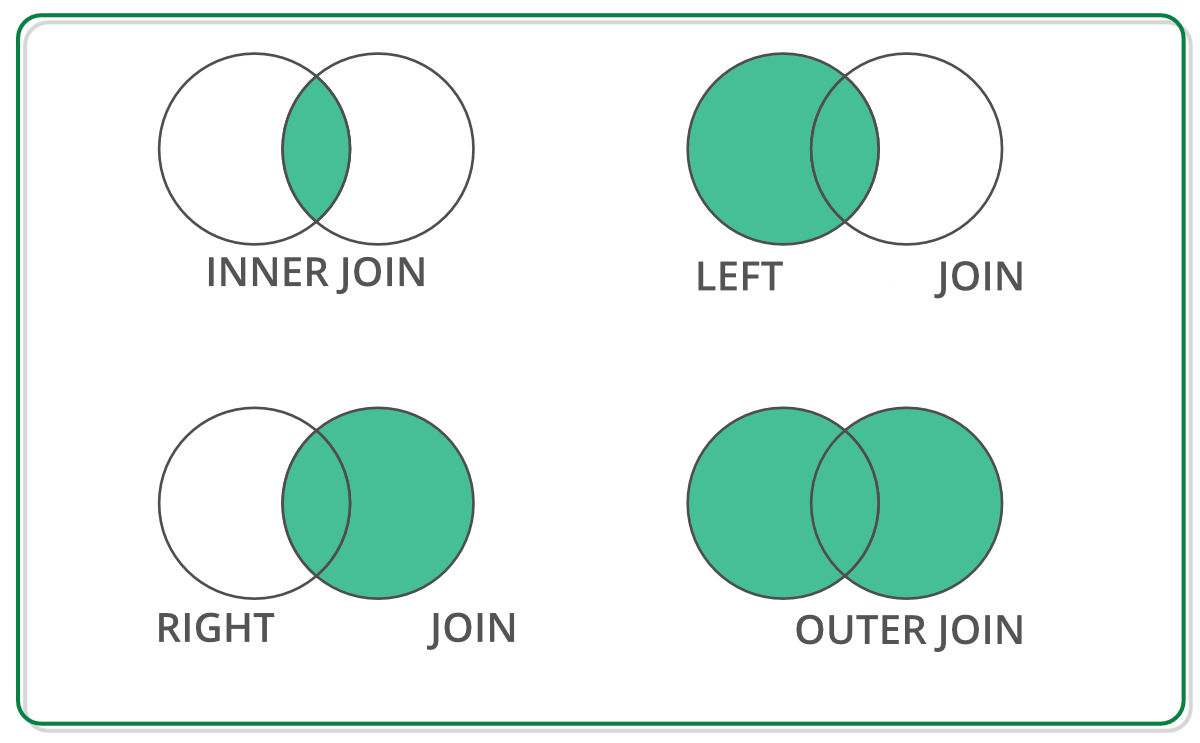
In Inner Join, we can clearly see, it returns a table that consists only of the ids (name) that is common in both the tables. (i.e Walter, Saul and Goodman.)

The Final one being Outer Join, it is just the opposite of Inner Join. In technical terms, it returns all the records from both tables (right and left) based on the id that we set. It is the Union of both the tables.



As Evident, it is returning all the records present in both the tables. ‘White’ is only present in the Marks tables, whereas ‘Hank’ is only present in the Age table, still it is returning those records as well.

Note : NaN is Null, since the marks for ‘Hank’ are not present. Similarly, the age and hobby of ‘White’ is not known.

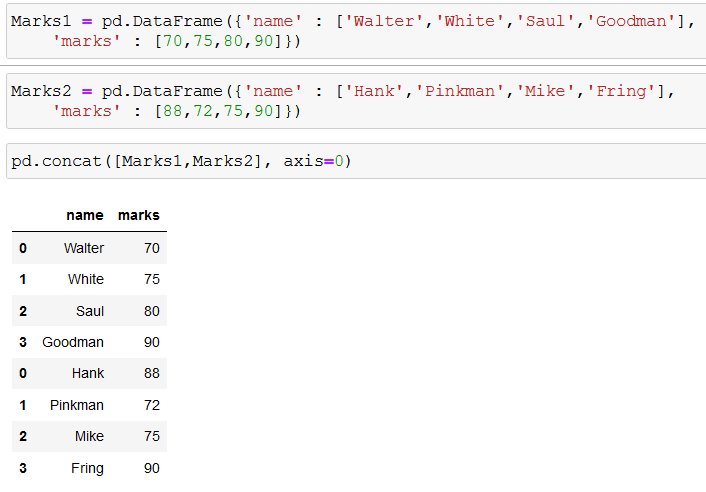


In this image, Venn Diagrams of all the joins are present for effective visualization of different types of joins.

## Concat

The **concat** function does all of the heavy lifting of performing concatenation operations along an axis.

You can vertically/horizontally concat any number of tables by using this function from pandas library.



The Above image is an example of vertical (axis = 0) concatenation of two dataframes.

Similarly you can do horizontal concatenation of two tables.

# Introduction to Some Basic Functions :

## UNIQUE function

unique() function is used to return unique values from a 1D array. This method works only on a single column and not on whole Data Frames. This method includes NULL value as a unique value.

**Syntax:** Series.unique()

Dataframe.column.unique()

**Return Type:** Numpy array of unique values in that column

# 

## NUNIQUE function

nunique() function is used to find the number of unique values in a series. This method works only on a single column and not on whole Data Frames. It counts null value as a unique value.

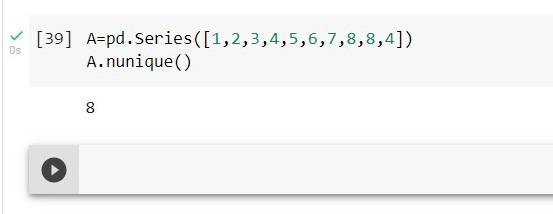
**Syntax:** Series.nunique(axis=0, dropna=True)

DataFrame.column.nunique( dropna=True)

**Parameters :**

axis : {0 or ‘index’, 1 or ‘columns’}, default 0.

dropna : Don’t include NaN in the counts.



## VALUE\_COUNTS function

value\_counts() function returns a Series containing counts of unique values. The result will be in descending order ,i.e. the first element is the most frequently-occurring element.

**Syntax:** Series.value\_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)

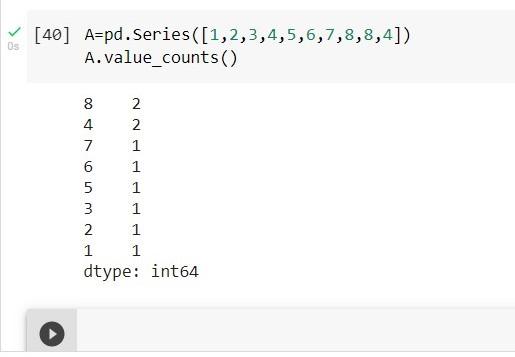
**Parameter:**

**normalize :** If True then the object returned will contain the relative frequencies of the unique values.

**sort :** Sort by values.

**ascending :** Sort in ascending order.

**dropna :** Don’t include counts of NaN.



## DESCRIBE function

The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

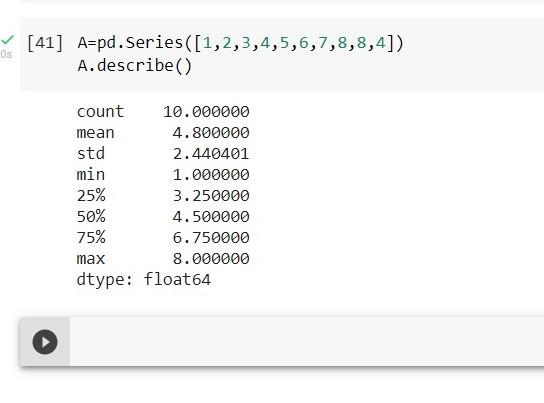
**Syntax:** Series.describe(percentiles=None, include=None, exclude=None)

**Parameters:**

**percentile:** list like data type of numbers between 0-1 to return the respective percentile

**include:** List of data types to be included while describing dataframe. Default is None

**exclude:** List of data types to be Excluded while describing dataframe. Default is None



## ISIN function

The isin() function is used to check whether each element in the DataFrame or Series is contained in values or not.

**Syntax:** DataFrame.isin(values)

**Parameters:**

**values:** iterable, Series, List, Tuple, DataFrame or dictionary to check in the caller Series/Data Frame.



# Introduction to Enumerate and Zip :

## Enumerate

Often, when dealing with iterators, we also get a need to keep a count of iterations.Python eases the programmers’ task by providing a built-in function enumerate() for this task. Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.

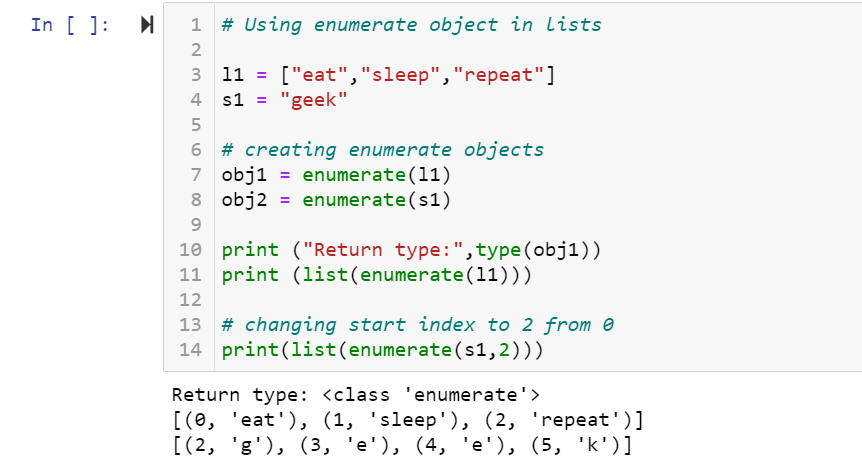
Enumerate() method adds a counter to an iterable and returns it in the form of an enumerating object. This enumerated object can then be used directly for loops or converted into a list of tuples using the list() method.

**Syntax :** enumerate(iterable, start=0)

**Parameters :**

* Iterable: any object that supports iteration.
* Start: the index value from which the counter is to be started, by default it is 0.

**Example : -**



## Zip

If you are a regular computer user, you should have used the .zip file extension. Do you know what it is? Basically, .zip is a container itself. It holds the real file inside.

Similarly,the purpose of the Python zip() method is to map the similar index of multiple containers so that they can be used just as a single entity. Python zip() takes iterable elements as input, and returns an iterator. If it gets no iterable elements, it returns an empty iterator.

**Syntax :** zip(\*iterators)

**Parameters :**

Python iterables or containers ( list, string etc )

**Return Value :**

Returns a single iterator object, having mapped values from all the containers.

**Example :-**

