**Pandas Introduction: -**

**Definition**:

* Open- source library used to work with opensource data
* Used for manipulation
* Built on NumPy
* Has high performance, productivity and is fast

**Advantages**:

* Allows working with different file format
* Fast and efficient for manipulating data
* Helps with operations like handling missing values, provides group by functions etc.
* Columns can be inserted and deleted from higher dimensional data.

**Installation of pandas:**

**Syntax:**

**! pip install package\_name**



**Importing of pandas:**

**Syntax:**

**Import package\_name as alias**



**DataStructures in pandas:**

1. **Series –**A one-dimensional data
2. **DataFrame –**2-dimensional data or multi-dimensional data

**1. Series: -**

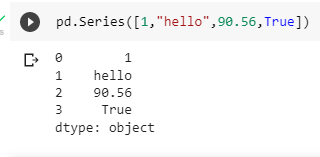
**Definition:**

* It is a one-dimensional heterogenous collection of data

**Specialty:**

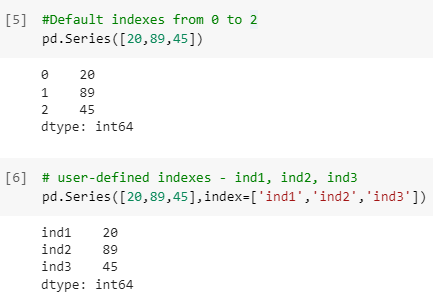
* Contains heterogenous collection of data

**Ex:**

****

* Axis labels are called as indexes.
* We have **user-defined** and default **indexes.**

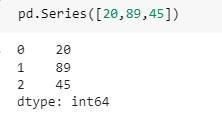
**Ex:**

****

**Creation of Series:**

**Syntax:**

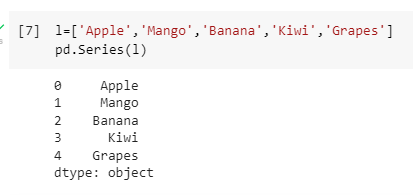
Pd.Series(data)

****

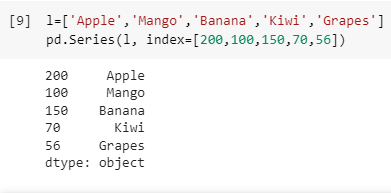
**Different Ways to Create a Series:**

1. **With list:**

* **Default indexes:**

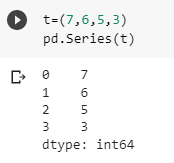
****

* **User-defined indexes:**

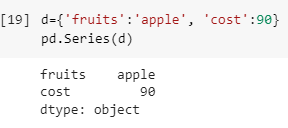
****

1. **With tuple:**

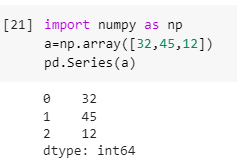
**Note:**Here the syntax for user-defined and default index will remain the same.

****

1. **With dictionary:**

****

1. **With NumPy array:**

****

1. **DataFrame: -**

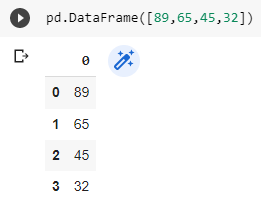
**Definition:**

* 2-dimensional heterogenous collection of data
* Here we will have rows and columns.

**Creating a DataFrame:**

**Syntax:**

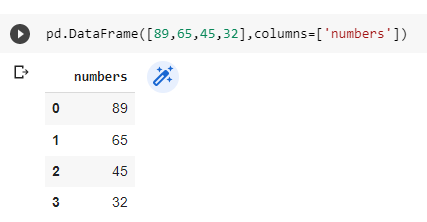
Pd.DataFrame(data)

****

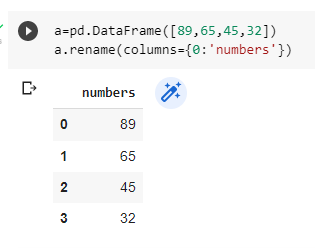
* Default column indexes are 0,1,2,3 etc.
* Default row indexes are 0,1,2,3 etc.

**Renaming columns in the DataFrame:**

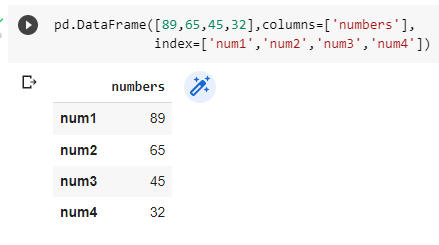
1. **With columns parameter in DataFrame function**

****

1. **With rename function:**

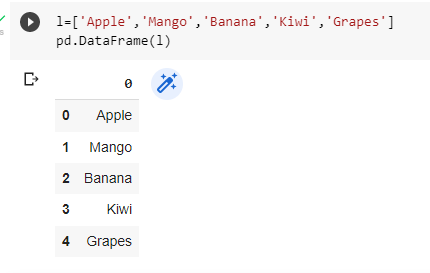
****

**Renaming indexes in the DataFrame:**

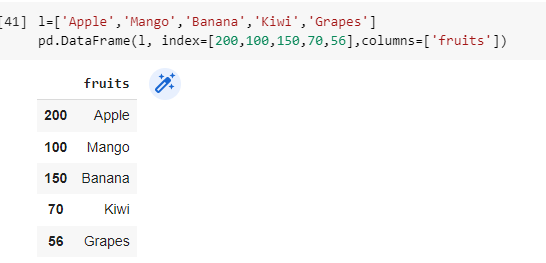


**DataFrame Creation methods:**

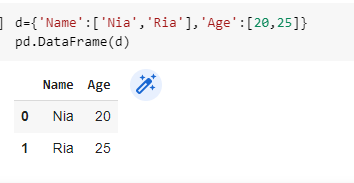
1. **With list:**



**With list + column name and indexes:**



1. **With Dictionary:**



**Note:** The creation methods shall remain same as creation of Series

**Google Colab Link: - Pandas\_Day\_1**

<https://colab.research.google.com/drive/1BC0PJrB5hBp70snJlxWf0GWSzwCDKCzn?usp=sharing>

**Indexing rows and columns: -**

**1.columns: -**

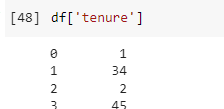
**Indexing of Columns:**

1. **Using [ ] operator:**

**Syntax:**

**Df['column']**

**Single Column:**

****

**Multiple Columns**

****

1. **Using loc:**

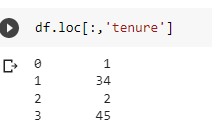
 You have to give column name but not index of column

We use loc for only if we know column name

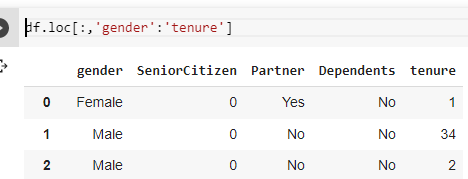
**Syntax:**

**Df.loc[s\_row:e\_row , s:col:e\_col]**

**Single Column:**

****

**Multiple Columns:**



**Using iloc:**

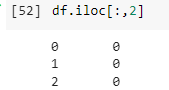
You must give column Index but not column Name.

We use iloc only if we do not know column name but only, we know column index.

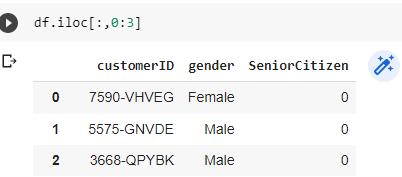
**Syntax:**

Df.loc[s\_row : e \_row +1, s : col:e\_col+1]

**Single column:**



**Multiple columns:**



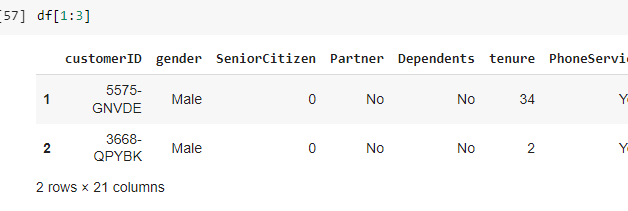
1. **Rows: -**

**Indexing with rows:**

1. **Using [ ] operator:**

**Syntax:**

**Data[ start: end+1 ]**



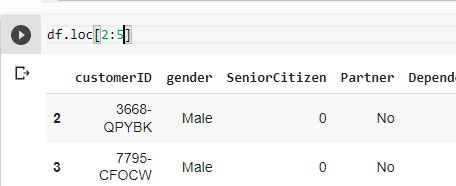
1. **Using loc:**

You have to give column name but not index of column

We use loc for only if we know column name

**Note:** The syntax will remain the same as above

**Loc with all columns and specific rows**



**Loc with specific columns and rows**

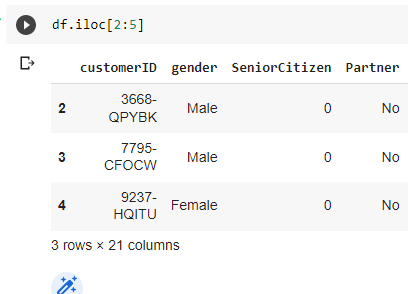


1. **Using iloc :**

You have to give column Index but not column Name

We use iloc only if we don’t know column name but only we know column index

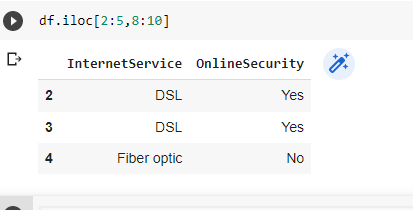
**Iloc for specific rows and all columns:**



**Iloc for specific rows and columns:**

You have to give column Index but not column Name

We use iloc only if we don’t know column name but only we know column index



**EDA: - Exploratory Data Analysis**

Used to Analyse the data and give Insite of Dataset by understanding it.

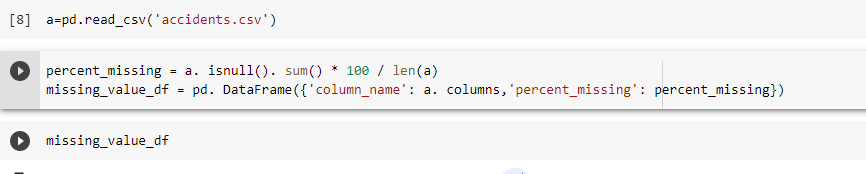
* a.head() 🡪 top 5 default and we can assign any number in this paranthasis to call.
* a.tail() 🡪 bottom 5
* a.describe() 🡪gives q1,q3, and mean, count all the details of columns
* a.info() 🡪 full details
* a.dtypes 🡪data types
* a.shape 🡪 no. of rows and columns

**Google Colab Link: -Pandas\_day\_2**

<https://colab.research.google.com/drive/1p3uCnuuVJKbtcX_PGIuddcNqIl_1yGyN?usp=sharing>

**Missing Value Analysis – Null Values**

**Finding % of null values in the dataset:**

****

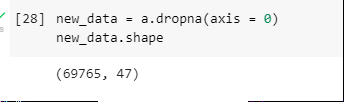
Missing\_value\_df=all columns & percentage of those columns in one table

1. **Dropna():**

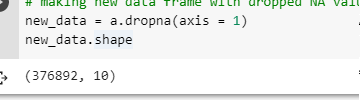
**Syntax:**

**Data.dropna()**

* **Row-wise dropna()**

****

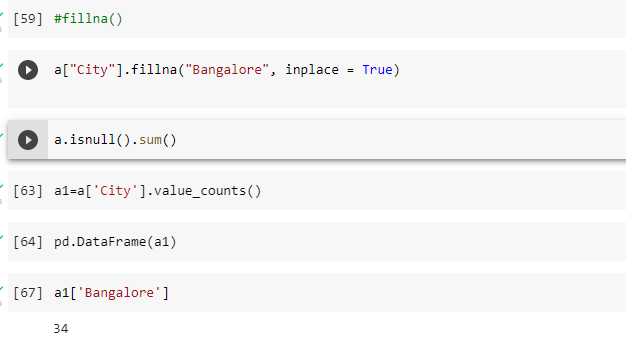
* **Column-wise dropna()**

****

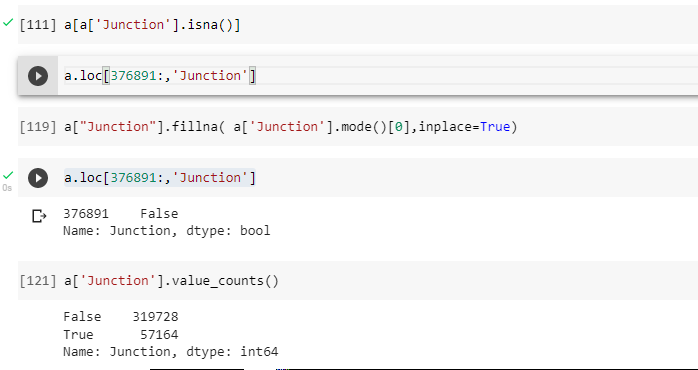
1. **Fillna():**

**Syntax:**

**Data.fillna('replacing\_value')**

****

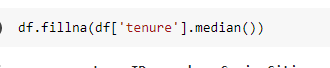
* **Fillna() with mode:**

****

* **Fillna() with mean:**

****

* **Fillna() with median:**

****

1. **Replace():**

**Syntax:**

**Data.replace(to\_replace= old\_value , value = new\_value)**

****

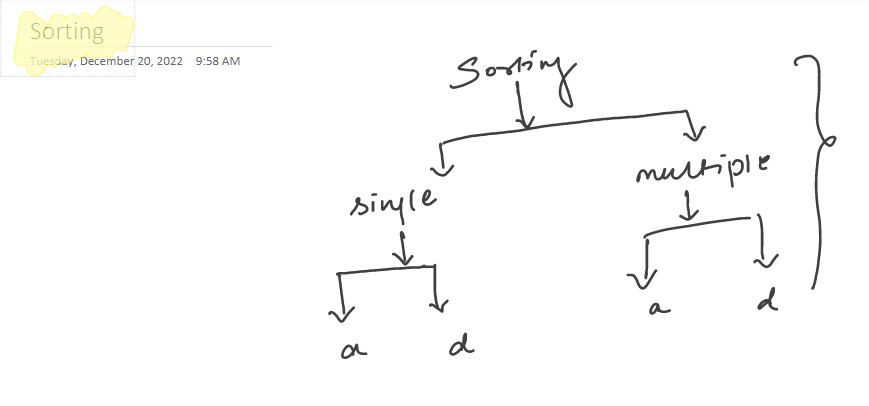
**Google Colab Link: - Pandas\_Day\_3**

<https://drive.google.com/file/d/1-VY-nChFk46qYAFPg3-6hsnfRrJCEEoN/view?usp=sharing>

**Google Colab Link: - Pandas\_Day\_4**

<https://drive.google.com/file/d/1epaDMb7lw3QUrR_39n-96ENwvZUgvCgV/view?usp=sharing>

**Sorting**

****

**what is sorting?**

**rearranging data in ascending/ descending order**

**how can we achieve this?**

**function --> sort\_values**

**in this function we have parameter --> ascending**

**by-default ascending --> True**

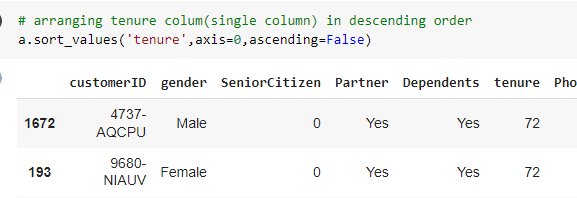
**we dont have descending=True instead we have ascending = False--> descending order**

**Syntax:**

**dataframe.sort\_values(column,axis=0,ascending=True/False)**

**Sorting Single column:**

**Descending order:**

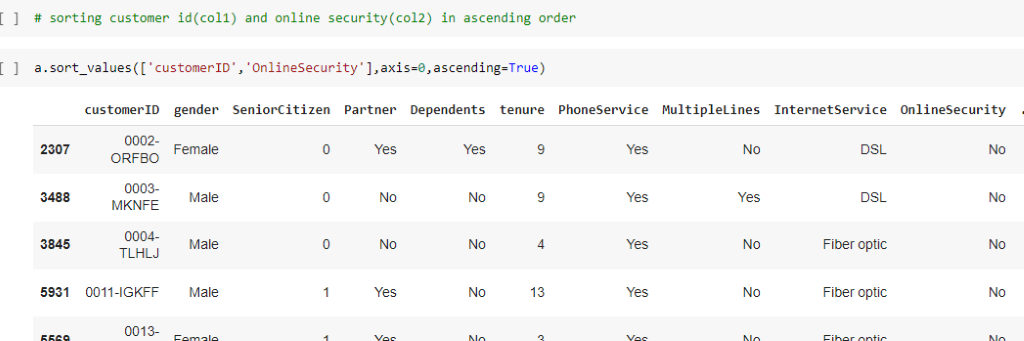


**Ascending Order:**

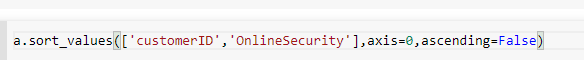


**Sorting multiple columns:**

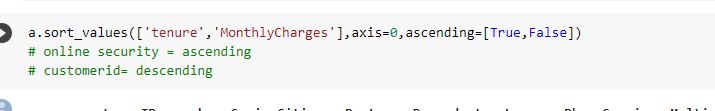
**Ascending order:**



**Descending Order:**



**Sorting Multiple columns in multiple orders:**



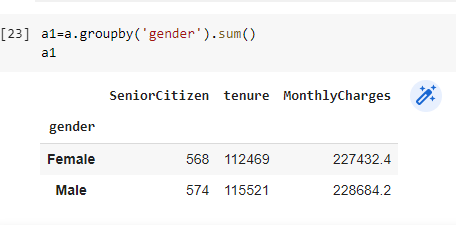
**Google Colab Link: - Pandas\_Day\_5**

<https://drive.google.com/file/d/1qnEXX0xAskSAIuWfYZtWNro9aWdL3fHt/view?usp=sharing>

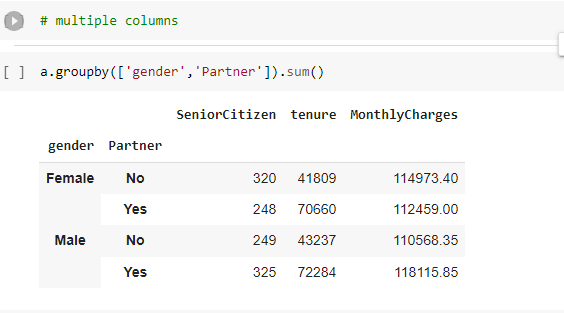
**Grouping (groupby, get\_group)**

* **We can create a grouping of categories and apply a function to the categories.**
* **In real data science projects, you’ll be dealing with large amounts of data and trying things over and over, so for efficiency, we use Groupby concept.**
* **You will get return only int value columns beside groupby column**

**Grouping single column:**



**Grouping multiple columns:**

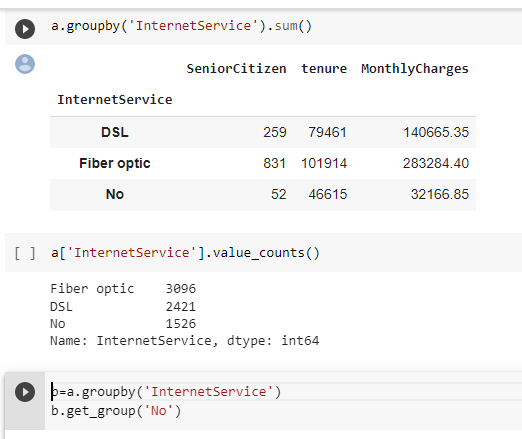


**Sorting with groupby:**

**Note:**By default, the sort is True



**Extract particular group from a grouped column:**



**Extracting multiple groups from multiple columns:**



**Google Colab Link: - Pandas\_Day\_5 🡪 bottom**

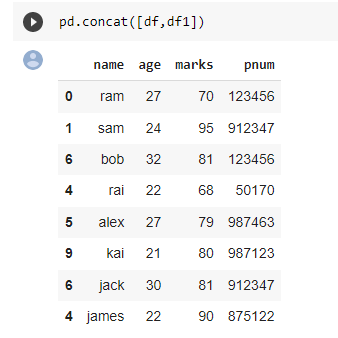
<https://drive.google.com/file/d/1qnEXX0xAskSAIuWfYZtWNro9aWdL3fHt/view?usp=sharing>

**Concatenation**

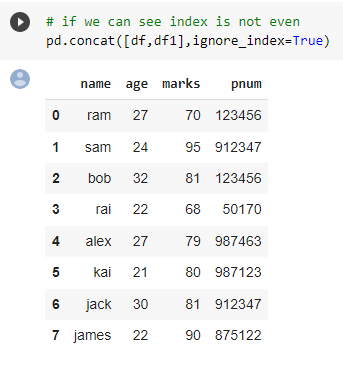
**Definition:**

**Concatenating means obtaining a new string that contains both of the original strings.**

**Concat with original indexes:**

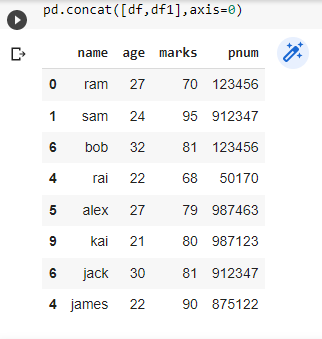
****

**Concat with continuous indexes:**

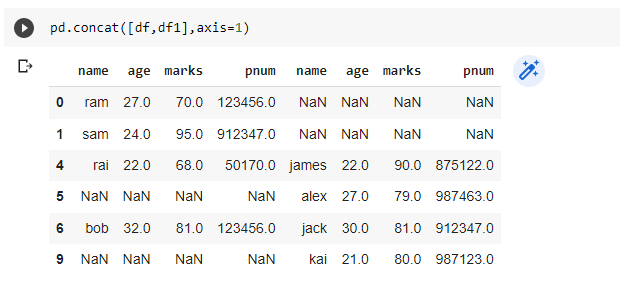
****

**Concat with respect to rows:**

**Note:**by default axis=0 like above given example.

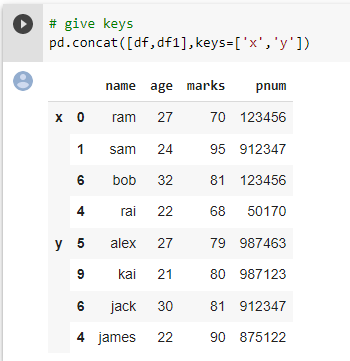
****

**Concat with respect to columns:**

****

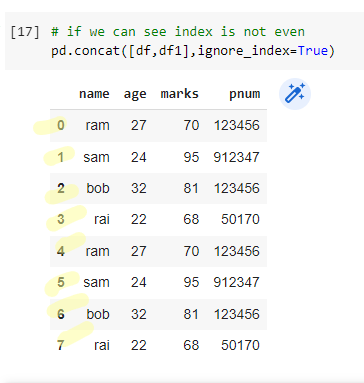
**Concat with keys:**

**Note:** this helps to differentiate b/w 2 data frames which are concatenated.

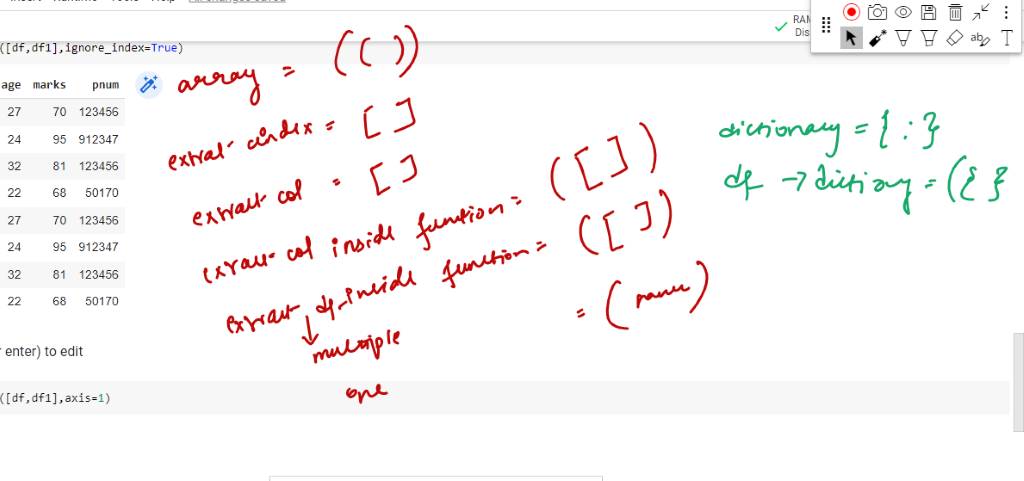
****

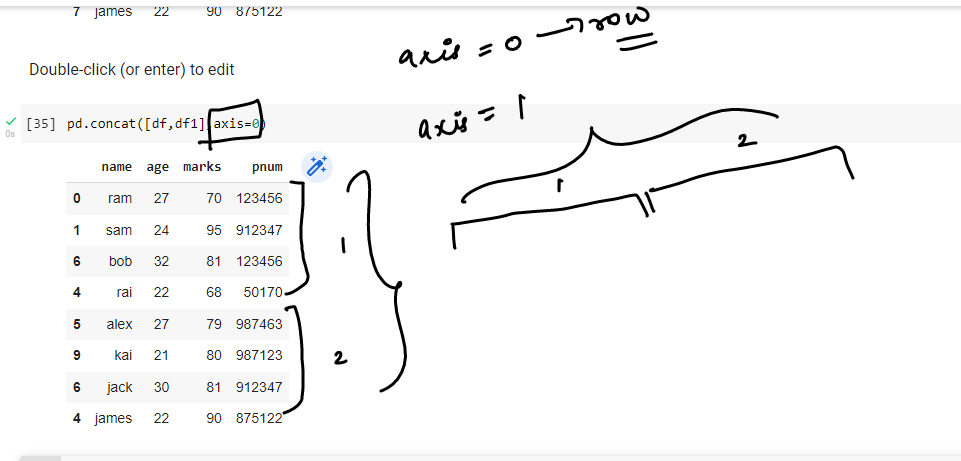
****

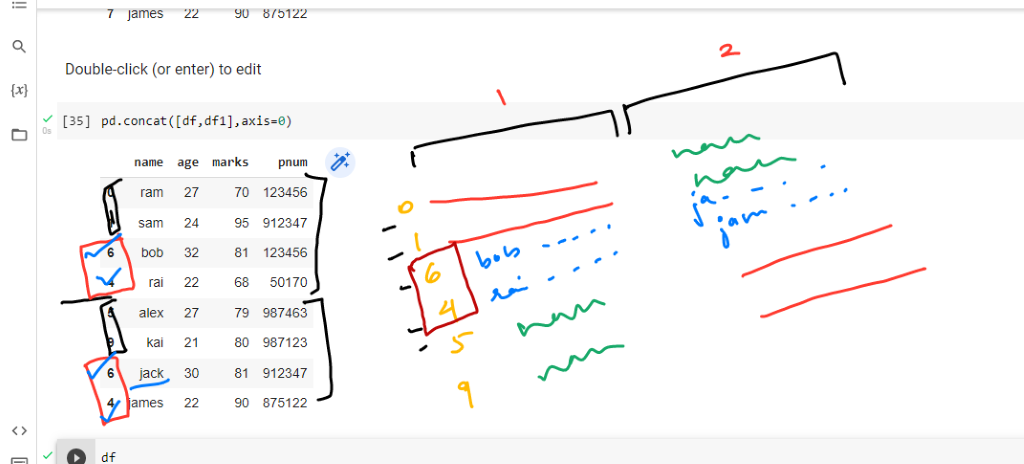
**Trying to ignore indexes:**

****

**When To Use Which Brackets**

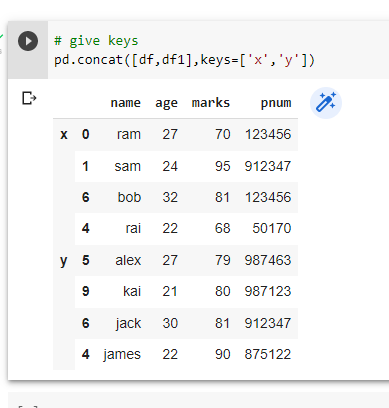


****

****

With respect to keys:

We can easily understand where the next data is starting

****

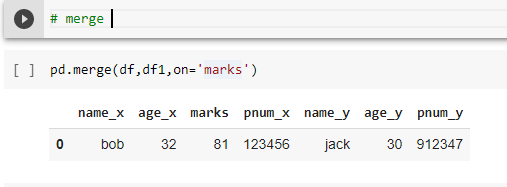
**Merge**

**Definition:**  its is similar to concatenation but column wise only.

**It is a way to join 2 dataframes. In merge we have,**

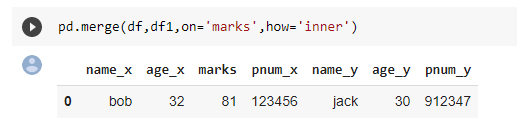
* **Outer merge**: total table is visible
* **Inner merge**: only rows visible which has all values in each column
* **Right merge**: visible right-side table
* **Left merge:** visible left-side table

**Simple Merge:**

****

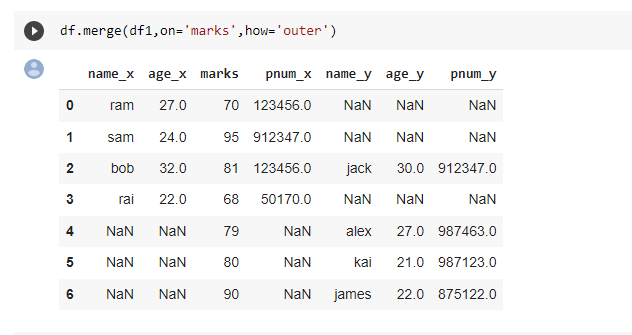
**Inner merge:**

**Here we get all the common elements from both dataframes**

****

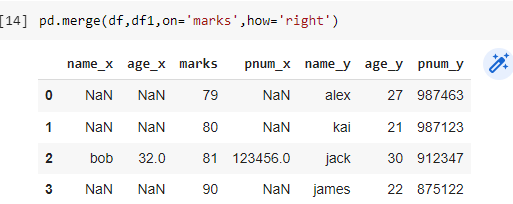
**Outer merge:**

**Here we get all the data present in both dataframes.**

****

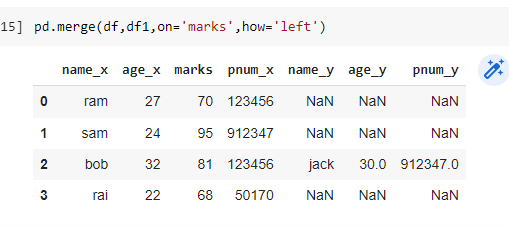
**Right merge:**

**Here we get the right table as well as the common data b/w both DF.**

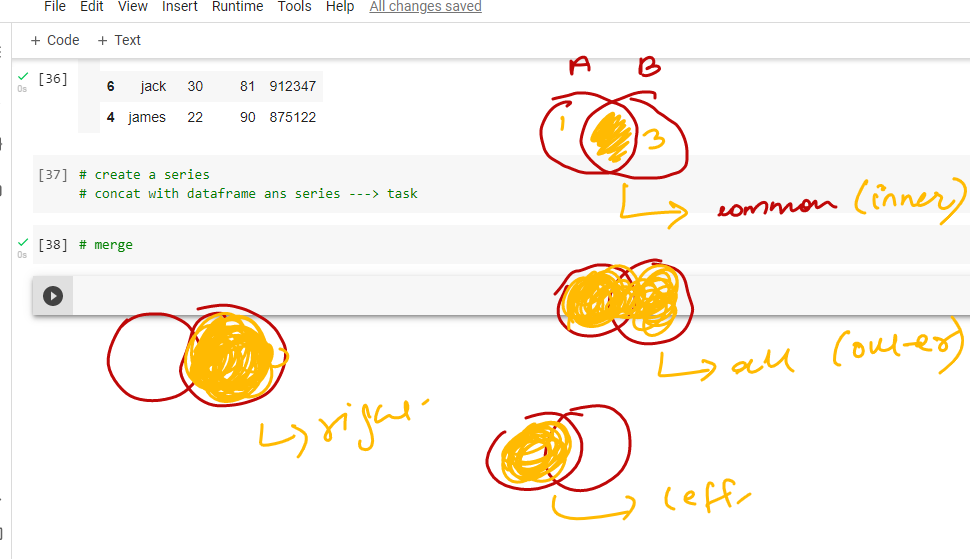
****

**Left merge:**

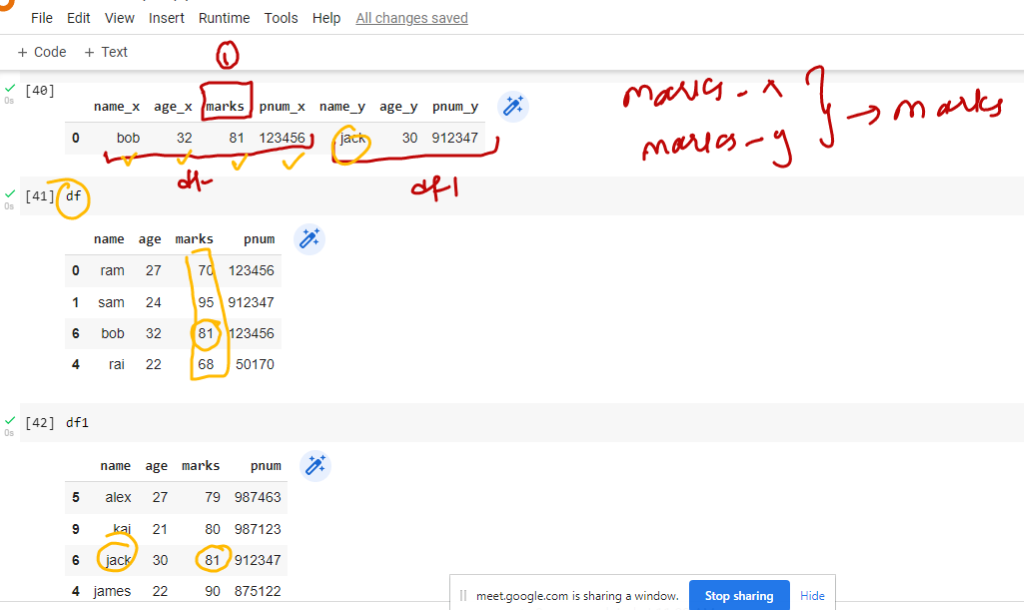
**Here we get the left table as well as the common data b/w both DF.**

****

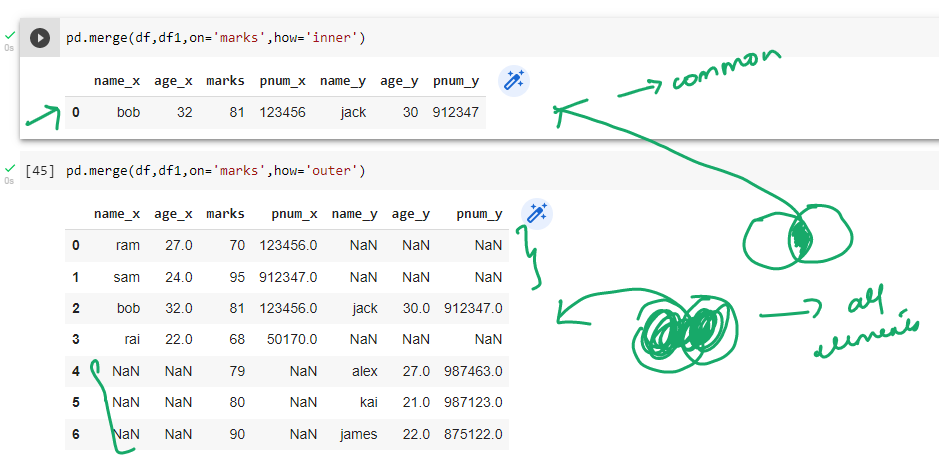
**Merge:**

****

**Working of merge:**

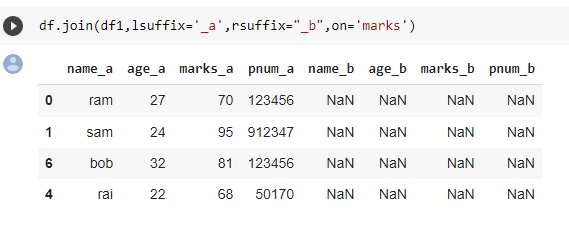
****

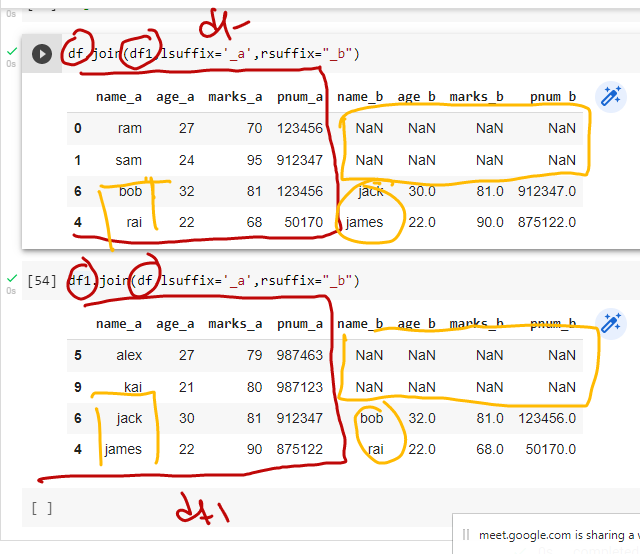
**Merge with how parameter:**

****

**Join**

**Simple joins:**





**Google Colab Link: - Pandas\_Day\_6 🡪concat,merge,joins**

<https://drive.google.com/file/d/13dlgCzBbctNgUWNsGQpu3DljWjc6nO9W/view?usp=sharing>

In Pandas, concat, merge, and join are functions used to combine data from multiple sources.

concat is used to stack data frames vertically or horizontally. It can be used to combine data frames by rows or by columns. For example, you can use pd.concat([df1, df2]) to stack df1 and df2 vertically (i.e., one below the other). You can also specify the axis parameter to stack them horizontally (i.e., side by side).

merge is used to combine data frames by matching rows from different data frames based on one or more key columns. merge can be used to combine data frames with different shapes and sizes. It's similar to a SQL JOIN statement.

join is used to combine data frames by matching index labels. It's similar to merge, but it only works on data frames with a shared index.

In summary, concat is used to stack data frames vertically or horizontally, merge is used to combine data frames by matching rows based on key columns, and join is used to combine data frames by matching index labels.

Go through this below link to know more about pandas

<https://levelup.gitconnected.com/20-pandas-functions-for-80-of-your-data-science-tasks-b610c8bfe63c>