Pandas DataFrame : Columns, Rows , Index and

finally understand merge( )

```
#Create DataFrame to work with:

sales_data = pd.DataFrame({

"name":["William","Emma","Sofia","Markus","Edward","Thomas","Ethan","Olivia","Arun","Anika","Paulo"]

,"region":["East","North","East","South","West","West","South","West","West","East","South"]

,"sales":[50000,52000,90000,34000,42000,72000,49000,55000,67000,65000,67000]

,"expenses":[42000,43000,50000,44000,38000,39000,42000,60000,39000,44000,45000] })
```

DataFrame Index:

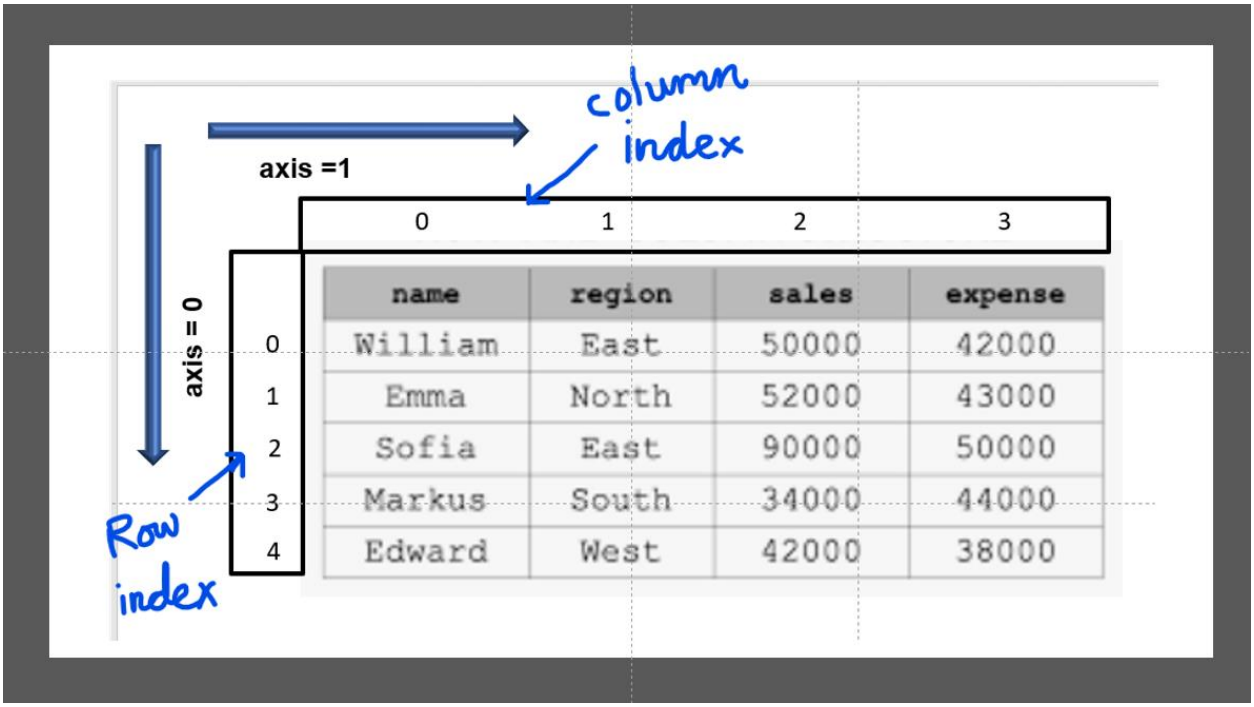
When you create a DataFrame in Pandas , the DataFrame will automatically

have certain properties. Each Row and Column will have an Integer “location” in the DataFrame. **Index starts at 0**.

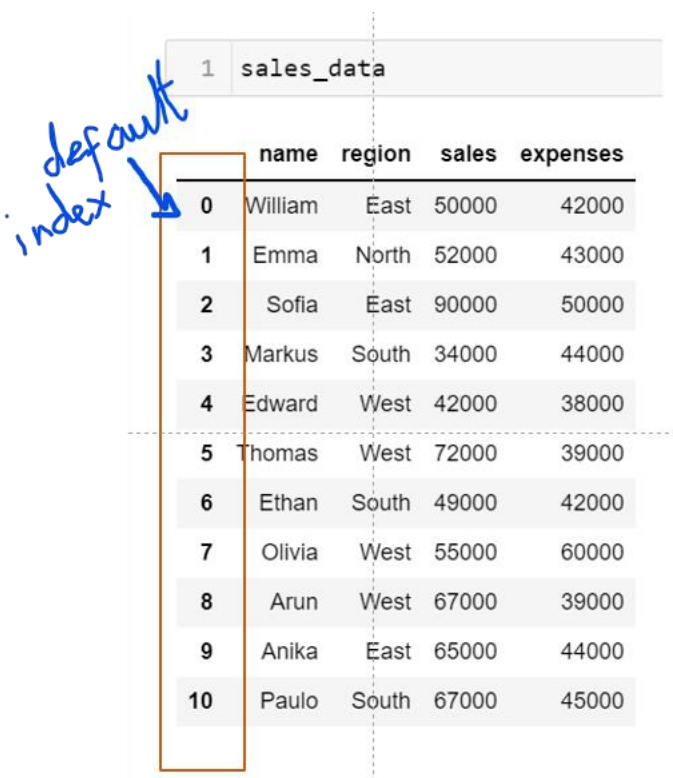
So, we can retrieve specific columns by its numeric Index or Column Names. Similarly , for Rows , a numeric Index is created by default.

So, by default , Index is the list of numbers starting at 0.

An Index is nothing but an Address or identification of a particular row or column in a DataFrame.



Look at below img for more clarity:



	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

And this is how we can print the index of sales_data :

```python
import pandas as pd
import numpy as np
```

```
In [125]: print(sales_data.index)

RangeIndex(start=0, stop=11, step=1)
```

We can go ahead and re-define the Index of the DataFrame. Meaning , we want to have some column from the DataFrame as our new Row Index. We may use one of the existing columns of the DataFrame as a new Index by using Pandas set_index( ) function. Once we set the “name” variable as Row Index , we will not be able to access “name” as column. But, we can use the values of this new index to retrieve the Rows of the DataFrame.

We set the "name" variable as the new Index:

```
In [126]: sales_data.set_index('name', inplace = True)

In [127]: sales_data

Out[127]:
```

	region	sales	expenses
name			
William	East	50000	42000
Emma	North	52000	43000
Sofia	East	90000	50000
Markus	South	34000	44000
Edward	West	42000	38000
Thomas	West	72000	39000
Ethan	South	49000	42000
Olivia	West	55000	60000
Arun	West	67000	39000

Using this new Index this is how we can access a specific Row :

```
In [133]:  1  sales_data.loc['Markus', :]

Out[133]: region      South
          sales       34000
          expenses    44000
          Name: Markus, dtype: object
```

This is how we Slice the Rows with the help of new index :

name	region	sales	expenses
William	East	50000	42000
Emma	North	52000	43000
Sofia	East	90000	50000
Markus	South	34000	44000
Edward	West	42000	38000
Thomas	West	72000	39000
Ethan	South	49000	42000
Olivia	West	55000	60000
Arun	West	67000	39000
Anika	East	65000	44000
Paulo	South	67000	45000

```
In [135]:  1  sales_data.loc['Markus':'Thomas', :]

Out[135]:
```

name	region	sales	expenses
Markus	South	34000	44000
Edward	West	42000	38000
Thomas	West	72000	39000

We can reverse the whole thing again and reset the “name” variable back to its original. We use the reset_index( ) method .

reset_index( ) turns the index back into a regular column.

```
In [136]: sales_data.reset_index(inplace = True)
```

```
In [137]: sales_data
```

Out[137]:

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000

But , what is the point of indexing ?

An Index on Pandas DataFrame gives us a way to identify rows. Identifying Rows by a label is arguably better than identifying Row by a number.

Useful TRICK : If you want to have the “name” column both as an Index and as a column:

```
In [138]: sales_data['NAME'] = sales_data["name"] #creating duplicate column
```

```
In [139]: sales_data
```

Out[139]:

	name	region	sales	expenses	NAME
0	William	East	50000	42000	William
1	Emma	North	52000	43000	Emma

```
In [141]: #Rearranging  columns
```

```
In [140]: sales_data = sales_data[['name' , "NAME" , "region" , "sales", "expenses"]]
```

```
In [142]: sales_data
Out[142]:
```

	name	NAME	region	sales	expenses
0	William	William	East	50000	42000
1	Emma	Emma	North	52000	43000
2	Sofia	Sofia	East	90000	50000

```
In [143]: sales_data.set_index('name', inplace = True)  #setting the "name" column as New Index
```

```
In [144]: sales_data
Out[144]:
```

name	NAME	region	sales	expenses
William	William	East	50000	42000
Emma	Emma	North	52000	43000
Sofia	Sofia	East	90000	50000
Markus	Markus	South	34000	44000
Edward	Edward	West	42000	38000

```
In [145]: sales_data.loc[sales_data['NAME'] == 'William']
Out[145]:
```

name	NAME	region	sales	expenses
William	William	East	50000	42000

But “name” will give an error:

```
In [147]:  1  sales_data.loc[sales_data['name'] == 'William']

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, to
   2645             try:
-> 2646                 return self._engine.get_loc(key)
   2647             except KeyError:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
```

But, this will work:

```
In [152]:  1  sales_data.loc['William']

Out[152]: NAME         William
          region          East
          sales          50000
          expenses       42000
          Name: William, dtype: object
```

<u>Single Record / Row selection using iloc : **iloc[0]**</u>

The **iloc** indexer for Pandas DataFrame is used for **Integer-Location based indexing** / selection by position.

Pandas iloc enables you to select Data from a DataFrame by Numeric Index.

“iloc” in pandas is used to **select rows and columns by number**, in the order that they appear in the data frame.

```
sales_data
```

	NAME	region	sales	expenses
name				
William	William	East	50000	42000
Emma	Emma	North	52000	43000
Sofia	Sofia	East	90000	50000
Markus	Markus	South	34000	44000
Edward	Edward	West	42000	38000
Thomas	Thomas	West	72000	39000

```
sales_data.iloc[0]   #Single Record selection using iloc
```

```
NAME        William
region         East
sales         50000
expenses      42000
Name: William, dtype: object
```

Accessing Last Row of DataFrame: **iloc[-1]**

```
sales_data
```

	NAME	region	sales	expenses
name				
William	William	East	50000	42000
Emma	Emma	North	52000	43000
Sofia	Sofia	East	90000	50000
Markus	Markus	South	34000	44000
Edward	Edward	West	42000	38000
Thomas	Thomas	West	72000	39000
Ethan	Ethan	South	49000	42000
Olivia	Olivia	West	55000	60000
Arun	Arun	West	67000	39000
Anika	Anika	East	65000	44000
Paulo	Paulo	South	67000	45000

```
sales_data.iloc[-1]   #Last Record selection using iloc
```

```
NAME        Paulo
region      South
sales       67000
expenses    45000
Name: Paulo, dtype: object
```

Accessing All rows of First column:

```
sales_data
```

	NAME	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

All rows

```
sales_data.iloc[: , 0]   #Fist column of DataFrame
```

```
0     William
1        Emma
2       Sofia
3      Markus
4      Edward
5      Thomas
6       Ethan
7      Olivia
8        Arun
9       Anika
10      Paulo
Name: NAME, dtype: object
```

Accessing Multiple Rows :

Select First Five Rows of sales_data: ¶

```
sales_data.iloc[0:5]
```

```
sales_data
```

	NAME	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

Select First Five Rows of sales_data:

In [165]:
```
sales_data.iloc[0:5]
```

Out[165]:

	NAME	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000

Select First two Columns of sales_data with all Rows:

```
sales_data.iloc[:, 0:2] # first two columns of data frame with all rows
```

All rows

	NAME	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

Select First two Columns of sales_data with all Rows:

```
sales_data.iloc[:, 0:2] # first two columns of data frame with all rows
```

	NAME	region
0	William	East
1	Emma	North
2	Sofia	East
3	Markus	South
4	Edward	West
5	Thomas	West
6	Ethan	South
7	Olivia	West
8	Arun	West
9	Anika	East
10	Paulo	South

Select 1st, 4th, 7th ROWS & 1st 3rd and 4th columns.:

	0	1	2	3
	NAME	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

Select 1st, 4th, 7th ROWS & 1st 3rd and 4th columns.:

```
sales_data.iloc[[0,3,6], [0,2,3]]
```

	NAME	sales	expenses
0	William	50000	42000
3	Markus	34000	44000
6	Ethan	49000	42000

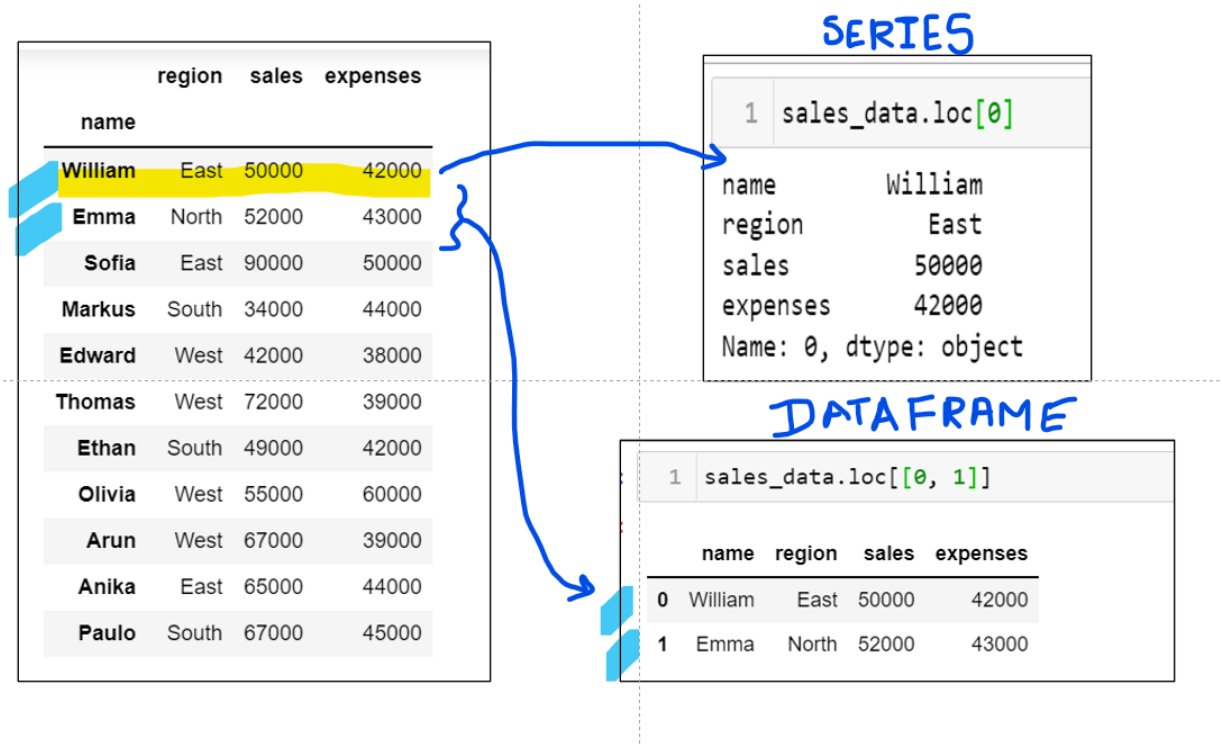
If the Output is one single row or one single column then it’s a SERIES

The Pandas loc[ ] is label based data selecting method which means that we have to pass the name of the row or column which we want to select.

a.) Selecting rows by Label / Index (which is the default label of a Row)

b.) Selecting rows with a Boolean or conditional lookup

Selecting Rows by Index or Label:



	region	sales	expenses
name			
William	East	50000	42000
Emma	North	52000	43000
Sofia	East	90000	50000
Markus	South	34000	44000
Edward	West	42000	38000
Thomas	West	72000	39000
Ethan	South	49000	42000
Olivia	West	55000	60000
Arun	West	67000	39000
Anika	East	65000	44000
Paulo	South	67000	45000

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000

It allows you to “**loc**ate” data in a DataFrame.

This method includes the last element of the range passed in it, unlike iloc()

.loc[ ]

In [193]:
```
sales_data.loc[0:3]
```

Out[193]:

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000

.iloc[ ]

In [194]:
```
sales_data.iloc[0:3]
```

Out[194]:

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000

.loc method to retrieve data using row label and column label:

```
sales_data
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000

Row label

column Label

```
sales_data.loc[0 , 'name']
```

```
'William'
```

Row index

numeric index Column

```
sales_data.iloc[0 , 0]
```

```
'William'
```

.loc to retrieve data from Row label 0 (that is the default Index label 0) and all columns:

Row 0

All columns

```
In [202]:  1  sales_data.loc[0 , :]

Out[202]: name        William
          region         East
          sales         50000
          expenses      42000
          Name: 0, dtype: object
```

Access all rows of Column : “name”

```
1  sales_data
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

All rows

Column name

```
In [209]:  1  sales_data.loc[: , 'name']

Out[209]: 0     William
          1       Emma
          2      Sofia
          3     Markus
          4     Edward
          5     Thomas
          6      Ethan
          7     Olivia
          8       Arun
          9      Anika
          10     Paulo
          Name: name, dtype: object
```

Retrieve Slices of Data : “Slices” of data are basically “ranges” of data:

```
sales_data
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

```
sales_data.loc[0 :3]
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000

.loc[ ] : All Rows of Columns region to expenses:

```
sales_data
```

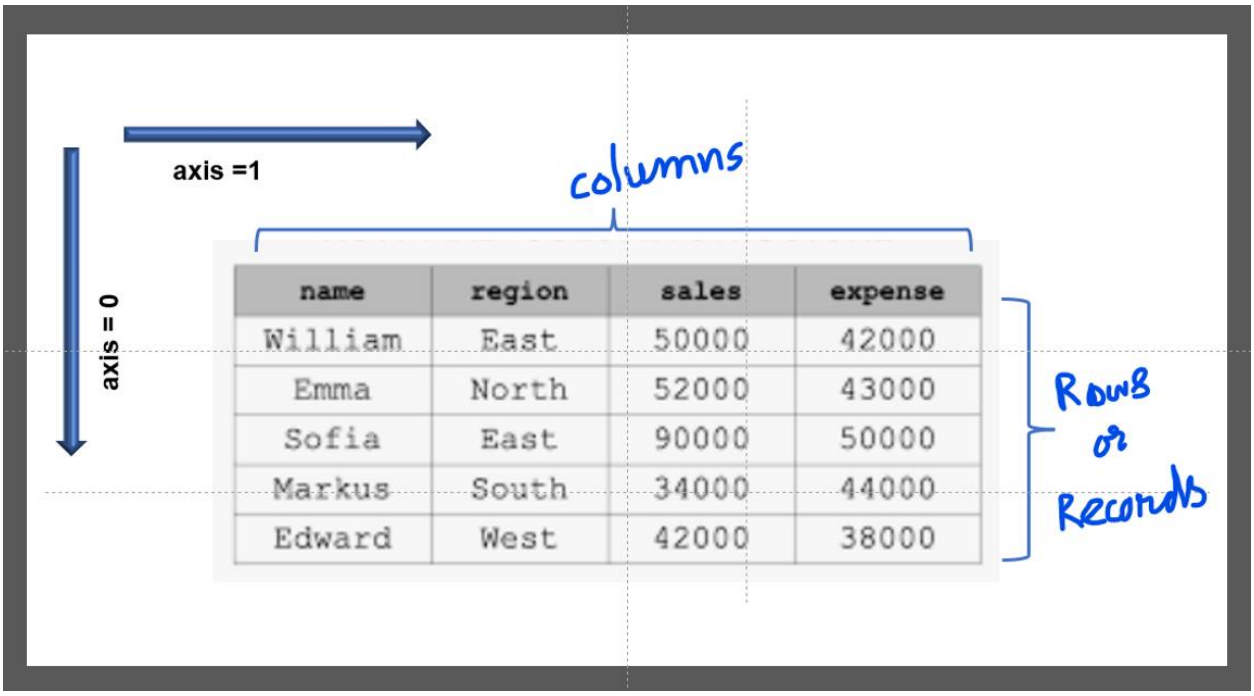
	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

All rows

```
sales_data.loc[ : , 'region' : 'expenses']
```

	region	sales	expenses
0	East	50000	42000
1	North	52000	43000
2	East	90000	50000
3	South	34000	44000
4	West	42000	38000
5	West	72000	39000
6	South	49000	42000
7	West	55000	60000
8	West	67000	39000
9	East	65000	44000
10	South	67000	45000

Columns and Rows of a DataFrame:



name	region	sales	expense
William	East	50000	42000
Emma	North	52000	43000
Sofia	East	90000	50000
Markus	South	34000	44000
Edward	West	42000	38000

Boolean Indexing :

Boolean is nothing but True or False

When we are not sure about the actual location of the value we are searching for, we ask the DataFrame if it exists . So the DataFrame replies back with a True or False.

To select the data using Boolean index we first create **Boolean Mask**.

Select Rows where sales < 40000

```
sales_data
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

```
#All rows where sales < 40,000

sales_data['sales'] < 40000
```

```
0     False
1     False
2     False
3      True
4     False
5     False
6     False
7     False
8     False
9     False
10    False
Name: sales, dtype: bool
```

```
#Save this boolean Mask in a variable and access the data like this:
```

```
sales_mask = sales_data['sales'] < 40000
```

```
sales_data[sales_mask]
```

	name	region	sales	expenses
3	Markus	South	34000	44000

OR

```
sales_data.loc[sales_data['sales'] < 40000]
```

	name	region	sales	expenses
3	Markus	South	34000	44000

Select fewer columns instead of selecting all columns :

```
sales_data.loc[sales_data['name']=='Markus', ['sales','expenses']]
```

	sales	expenses
3	34000	44000

Boolean Indexing on columns using multiple conditions:

```
sales_data
```

	name	region	sales	expenses
0	William	East	50000	42000
1	Emma	North	52000	43000
2	Sofia	East	90000	50000
3	Markus	South	34000	44000
4	Edward	West	42000	38000
5	Thomas	West	72000	39000
6	Ethan	South	49000	42000
7	Olivia	West	55000	60000
8	Arun	West	67000	39000
9	Anika	East	65000	44000
10	Paulo	South	67000	45000

```
sales_data.loc[(sales_data['region']=='East') & (sales_data['sales'] > 60000)]
```

	name	region	sales	expenses
2	Sofia	East	90000	50000
9	Anika	East	65000	44000

You may try other conditions and practice.

Merging DataFrames:

merge ( )

"Merging" two datasets is the process of bringing two datasets together into one, and aligning the rows from each based on common attributes or columns.

Pandas Merge will join two DataFrames together resulting in a **single, final dataset**. You have full control how your two datasets are combined.

We **can join columns from two Dataframes using the merge() function**

We bring two Datasets together via merge()

1. **What data do you want to join?** – This will be determined by your 'right' and 'left' datasets.
2. **How do you want to join them?** – You have different ways to join two datasets – left, right, full, inner. It's up to you depending on your final output requirements.
3. **What do you want to join them with?** – You also need to tell Pandas how your two datasets are related. What is the common column between the two?

```
pd.DataFrame.merge(other_dataset,
                   how='type_of_join',
                        on='columns_to_join_on')
```

Merge Parameters:

- **right:** The DataFrame you're calling `.merge()` is considered your 'left' dataset. You need to specify your other dataset in the `right` parameter. This can be another DataFrame or named Series.

- **how ('left', 'right', 'outer', 'inner', default= 'inner'):** How will determine 'how' to join your two datasets together. Do you want to keep all of your samples from your left df? Or your right? Maybe just where they have common rows.

- **on:** Sometimes called merge/join 'key'. The common column between your two datasets that you'd like to join on. You'll use `on` when the two columns have the same name. If they aren't named the same, then try left_ or right_ on

- **left_on/right_on**: If your columns to join on do not have the same name, no problem, simply pass their names into left_on (for your left dataset) and right_on (for your right dataset).

- **left_index/right_index:** Alternatively, instead of specifying a column, if you column to join on sits within a DataFrame's index, you can set left_index/right_index=True.
- **suffixes (Default=('_x', '_y')**: If you wanted to add a suffix (to help tell which columns came from which DataFrame) to the end of your newly-merged columns you can add them here.
- **indicator (Default = False)**: Helpful function that will attribute each row to a specific DataFrame or both. Simply, was this row only on the left side, right side, or was it shared by both?
- **validate**: Do a sanity check on both of your datasets before you merge. One of the 'gotchas' with merges is when there are unknown/unintended duplicates in either of your datasets. The `validate` parameter will help you check for these.
  - "**one_to_one**" or "1:1": checks if the merge column is unique in both left and right datasets.
  - "**one_to_many**" or "1:m": checks if the merge column is unique in left dataset
  - "**many_to_one**" or "m:1": checks if the merge column is unique in right dataset
  - "**many_to_many**" or "m:m": does not check duplicates in either column

```
import pandas as pd

df7 = pd.DataFrame({'Age': [11,13,16,8,9], 'Country': ['US', 'India', 'Africa', 'China', 'Korea']})

df8 = pd.DataFrame({'Age': [13,18,11,3,5],'Month': ['Jan','Feb','Mar','Apr','May'],'Year':
[2010,2012,2004,2005,2007]})
```

Combine two DataFrames on the column : “Age”

```
df7
```

	Age	Country	Year
0	11	US	2010
1	13	India	2012
2	16	Africa	2004
3	8	China	2005
4	9	Korea	2007

```
df8
```

	Age	Year	Birth_Month
0	11	2010	jun
1	13	2012	jul
2	10	2004	aug
3	18	2005	sep
4	19	2007	oct

```
df7.merge(df8, on='Age')
```

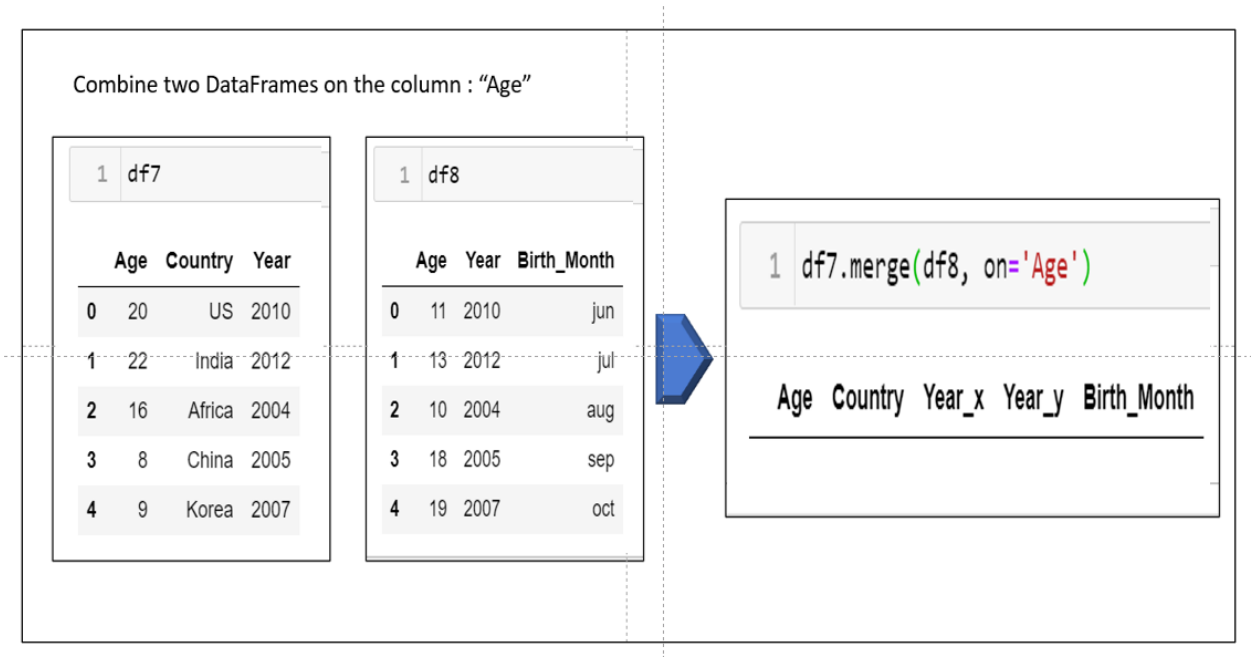
	Age	Country	Year_x	Year_y	Birth_Month
0	11	US	2010	2010	jun
1	13	India	2012	2012	jul

Result is a DataFrame with common Rows in the “Age” column

“Year_x” is the “Year” column belonging to df7

“Year_y” is the “Year” column belonging to df8

When there are no common rows between the two then ::: its an empty DataFrame.



Let's say you have DataFrames that share a common column, but unfortunately that column has a different name on either df.

You could rename the columns to be the same then perform the merge.

Alternatively, you can just specify a left_on and right_on :

```
df7
```

	Age	Country_Student
0	4	US
1	20	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Country_Candidate	Year
0	Spain	2010
1	India	2012
2	Yemen	2004
3	China	2005
4	Korea	2007

```
df7.merge(df8, left_on='Country_Student', right_on='Country_Candidate')
```

	Age	Country_Student	Country_Candidate	Year
0	20	India	India	2012
1	8	China	China	2005
2	9	Korea	Korea	2007

When you don't want to merge the entire DataFrames and only want to merge a subset of columns:

```
df7
```

on

	Age	Country
0	11	US
1	13	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Age	Month	Year
0	13	Jan	2010
1	18	Feb	2012
2	11	Mar	2004
3	3	Apr	2005
4	5	May	2007

```
df7.merge(df8[['Age', 'Month']], on='Age')
```

	Age	Country	Month
0	11	US	Mar
1	13	India	Jan

How do you want to merge your DataFrames?

how = left, right, outer, inner ?

how = ‘left’

```
df7
```

	Age	Country
0	11	US
1	13	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Age	Month	Year
0	13	Jan	2010
1	18	Feb	2012
2	11	Mar	2004
3	3	Apr	2005
4	5	May	2007

```
# We want to include all columns from the LEFT DataFRame (df7)
#df7 is your left DataFrame and df8 is your right

df7.merge(df8, on='Age', how='left')
```

	Age	Country	Month	Year
0	11	US	Mar	2004.0
1	13	India	Jan	2010.0
2	16	Africa	NaN	NaN
3	8	China	NaN	NaN
4	9	Korea	NaN	NaN

how = ‘right’

```
df7
```

	Age	Country
0	11	US
1	13	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Age	Month	Year
0	13	Jan	2010
1	18	Feb	2012
2	11	Mar	2004
3	3	Apr	2005
4	5	May	2007

```
# We want to include all columns from the RIGHT DataFRame (df8)
#df7 is your left DataFrame and df8 is your right

df7.merge(df8, on='Age', how='right')
```

	Age	Country	Month	Year
0	11	US	Mar	2004
1	13	India	Jan	2010
2	18	NaN	Feb	2012
3	3	NaN	Apr	2005
4	5	NaN	May	2007

how = ‘outer’

Unmapped rows will have NaNs

```
df7
```

	Age	Country
0	11	US
1	13	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Age	Month	Year
0	13	Jan	2010
1	18	Feb	2012
2	11	Mar	2004
3	3	Apr	2005
4	5	May	2007

```
# Include all columns from both DataFrames
df7.merge(df8, on='Age', how='outer')
```

	Age	Country	Month	Year
0	11	US	Mar	2004.0
1	13	India	Jan	2010.0
2	16	Africa	NaN	NaN
3	8	China	NaN	NaN
4	9	Korea	NaN	NaN
5	18	NaN	Feb	2012.0
6	3	NaN	Apr	2005.0
7	5	NaN	May	2007.0

how = ‘inner’

```
df7
```

	Age	Country
0	11	US
1	13	India
2	16	Africa
3	8	China
4	9	Korea

```
df8
```

	Age	Month	Year
0	13	Jan	2010
1	18	Feb	2012
2	11	Mar	2004
3	3	Apr	2005
4	5	May	2007

```
# Include all items that both DataFrames share
df7.merge(df8, on='Age', how='inner')
```

	Age	Country	Month	Year
0	11	US	Mar	2004
1	13	India	Jan	2010

Thank you

- Aisha Khalid