# Learning Objectives

In the third module of this course, you will learn how to handle different scenarios in order to better manipulate your data.

***Learning Objectives:***

1. Combining Data Frames
2. Understanding and applying merges
3. Understanding and applying joins

# Concatenate

## ### Concatenate

Join method combines two data frames on the basis of their indexes.

There are mainly five types of joins in Pandas:

- Inner join
- Left outer join
- Right outer join
- Full outer join or simply outer join
- Index join

To better understand joins we will create a Data Frame with two columns, one containing an id and the other a boolean representing whether the eye color is brown or not.

```
Eye_data = {
        'id':[1,2,3,4,5],
        'Brown': [True, False, True, False, False],
        }
df=pd.DataFrame(Eye_data, columns = ['id', 'Brown'])
print(df)
```

Lets create another data frame but this time for blue eyes

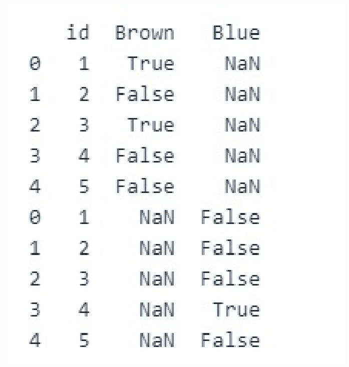
```
Eye_data2 = {
        'id':[1,2,3,4,5],
        'Blue': [False, False, False, True, False],
        }
df2=pd.DataFrame(Eye_data2, columns = ['id', 'Blue'])
print(df2)
```

We can now concatenate our two data frames!
Using `pd.concat([dataframe1,dataframe2])` we can combine the information from our data.

Feel free to delete `print(df)` and `print(df2)`. Since we already checked our previous data frames.

```
df_row = pd.concat([df, df2])

print(df_row)
```



```
   id  Brown   Blue
0   1   True    NaN
1   2  False    NaN
2   3   True    NaN
3   4  False    NaN
4   5  False    NaN
0   1    NaN  False
1   2    NaN  False
2   3    NaN  False
3   4    NaN   True
4   5    NaN  False
```

images/img3

As you can see from our example when concatenating our data frames if there is a value that is not present in both you will get `NaN` associated to where it was not present.

Let us create a third data frame with more information with users with brown eyes.

```
Eye_data3 = {
        'id':[1,2,6,7,8],
        'Brown': [True, False, True, False, False],
        }
df3=pd.DataFrame(Eye_data3, columns = ['id', 'Brown'])
```

Now let us try grouping together the two data frames with information on brown eyes.

```
Eye_data3 = {
        'id':[1,2,6,7,8],
        'Brown': [True, False, True, False, False],
        }
df3=pd.DataFrame(Eye_data3, columns = ['id', 'Brown'])

df_row2 = pd.concat([df, df3])
print(df_row2)
```

The two concatenations we did, one where we share common columns name and one where we had different ones. We can see that ‘concat’ simply adds the two pieces of information from our data frame one after the other.

# Joins

## ### Inner Join

In the examples on the previous page we had similar id names. We can use the merge function if we want to combine data. The merge function takes for the first argument a Data Frame, the second argument is a Data Frame, and then a merge column name, or a column to merge “on”.

Let’s do an example with the blue eye data first.

```
df_row3=pd.merge(df,df2,on='id')


print(df_row3)
```

We no longer have duplicate id’s but all the information about on id is presented in a single row.

Now remember, for the third Data Frame we created, we had different ids. What do you think will happen?

```
df_row4=pd.merge(df,df3,on='id')


print(df_row4)
```

The data presented is an example of an **inner join**.

**Inner Join** : The default Pandas behavior, only shows rows where the merge “on” have common characteristics. For example, on the data above, they only had two similar ids which are id 1,2.

Inner Join is the most basic, the default for the pandas merge function. `df_row4=pd.merge(df,df3,on='id')` is equal to the same thing as

```
df_row4=pd.merge(df,df3,on='id',how='inner')
print(df_row4)
```

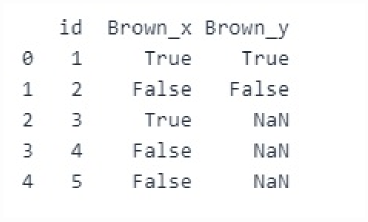
## ### Left Join

As you can see, the merge operation from above the merge operation takes in an extra option how. The how allows us users to determine which join we want to use. The default is Inner where we merge only on common characteristics of the element we merged on.

Now we are going to look over the ***left join*** operation.

**Left Join** : Only shows rows, that are present on the left (first data frame argument), and if a value is not present on the right it will replace it with NaN.

```
df_row5=pd.merge(df,df3,on='id',how='left')
print(df_row5)
```



```
   id  Brown_x Brown_y
0   1     True    True
1   2    False   False
2   3     True     NaN
3   4    False     NaN
4   5    False     NaN
```

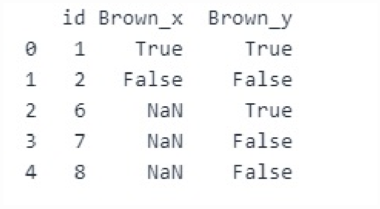
images/img4

### Right Join

Now that we know, what the left join does. We can pretty easily guess, how the right join works.

**Right Join** : Only shows rows , that are present on the right (second data frame argument), and if a value is not present on the right it will replace it with NaN.

```
df_row6=pd.merge(df,df3,on='id',how='right')
print(df_row6)
```



```
   id Brown_x  Brown_y
0   1    True     True
1   2   False    False
2   6     NaN     True
3   7     NaN    False
4   8     NaN    False
```

images/img5

---

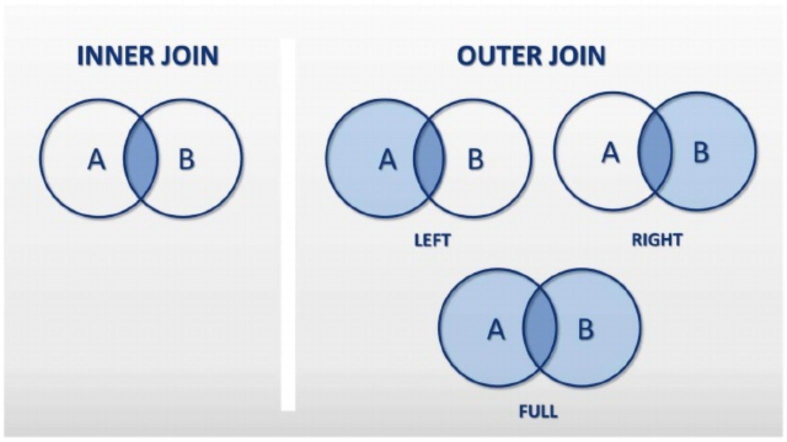
### Full Outer Join

In order for us to join all information present in both data frames, we have to perform a full outer join.

**Full Outer Join** : This operation, returns all rows that are present in both data frames, and where possible combines the information.

```
df_row7=pd.merge(df,df3,on='id',how='outer')
print(df_row7)
```

Now that we have covered the basic ways to joins. Here is a visual representation of the joins discuss above.



## Merges vs. joins

Both join and merge can be used to combine two data frames but the join method combines two data frames on the basis of their indexes whereas the merge method is more versatile and allows us to specify columns beside the index to join on for both data frames. Merge more or less does the same thing as join.

Merge requires specifying the columns as a merge key. We can specify the overlapping columns with parameter on, or can separately specify it with left_on and right_on parameters.

```
joined_df = df.join(df2, lsuffix='_')
print(joined_df)
```