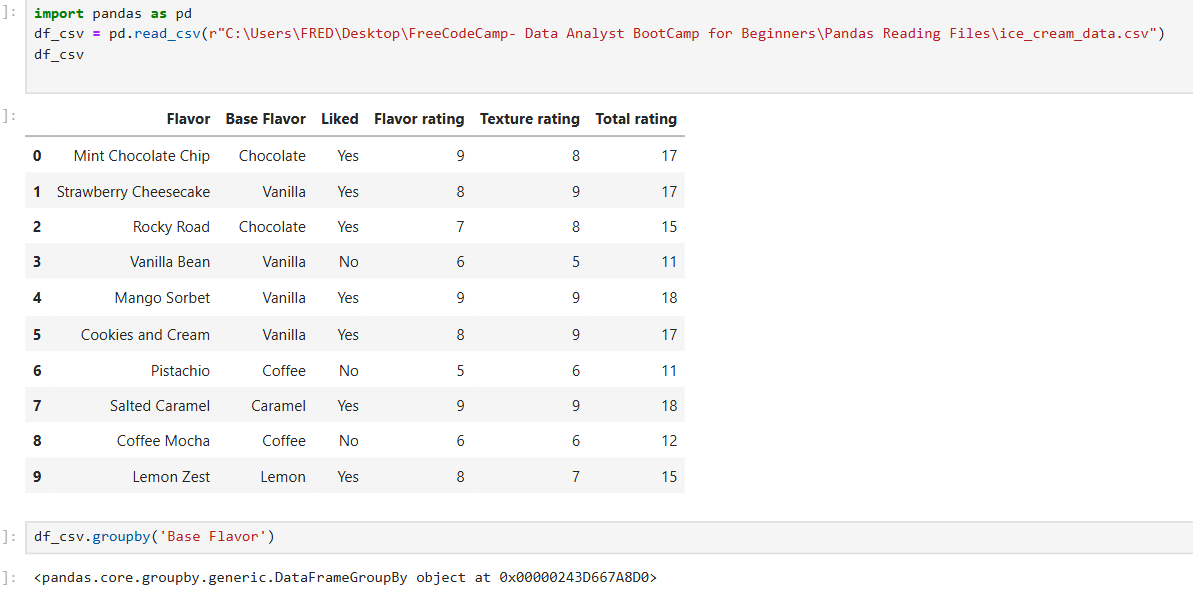
GROUPBY AND AGGREGATING

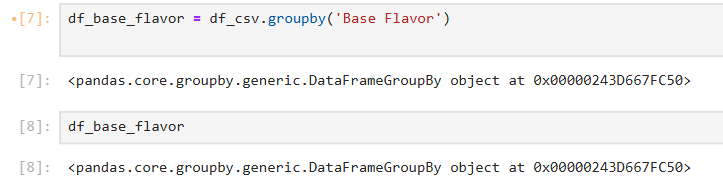
Groupby will group together values in a column and display them all in the same row to perform aggragate functions on those groupings

1. groupby can be used in column where there are similar data like ‘Base Flavor’. ‘Flavor’ column has unique datas so it cannot be used as key for groupby

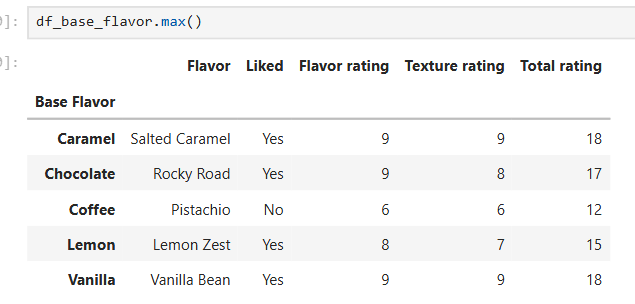
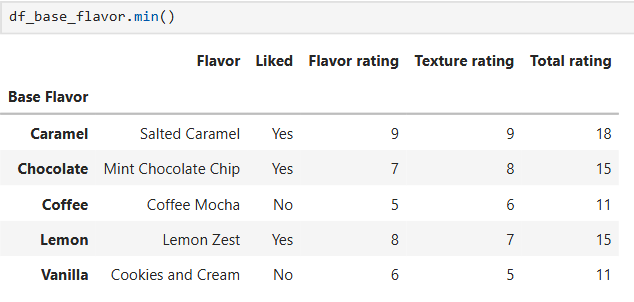
As you can see below when you use groupby, an object of ‘groupby’ was generated



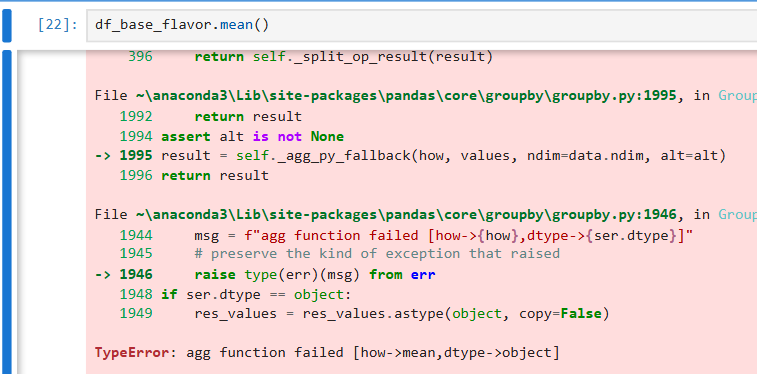
If you store the object in a variable and execute the variable, it will output the same object data



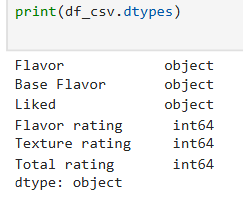
You need to use aggregate functions to display data. For these aggregate functions, it includes non-numeric columns for it aggregates each columns individually and letters are aggregated based on the alphabetical order

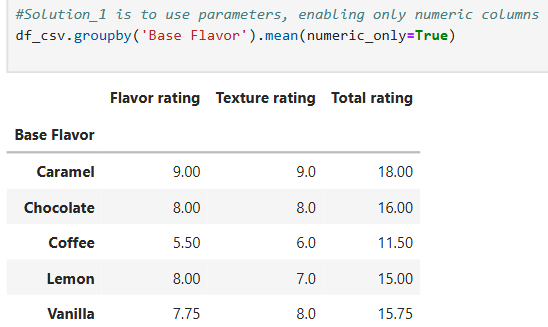
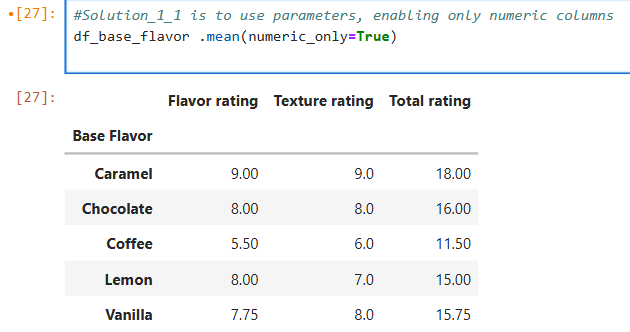
2.But when I use mean() there is an error

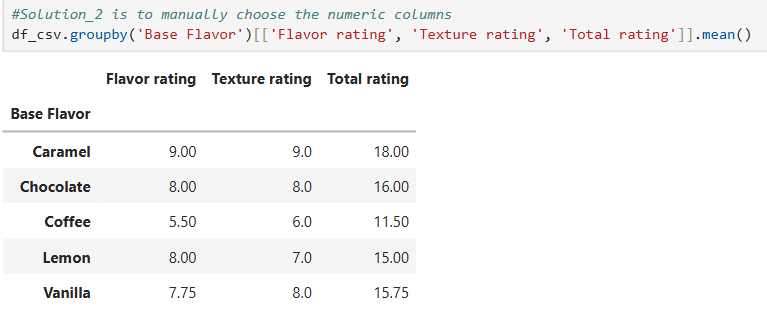


Possible reasons are you're using an **older version of pandas** that doesn't skip non-numeric data properly during aggregation or you have **non-numeric values mixed in** one of the numeric columns (e.g., a stray string, null, or whitespace). If the data type displayed is ‘int64’ then that means it has no stray string, null, or whitespace, if it does it will be an object

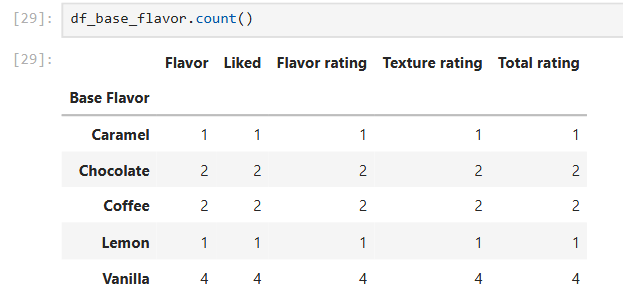


The solution: **Pandas 2.2.2 does support skipping non-numeric columns** during aggregation like .mean() — but only if you explicitly tell it to do so using the numeric\_only=True parameter.

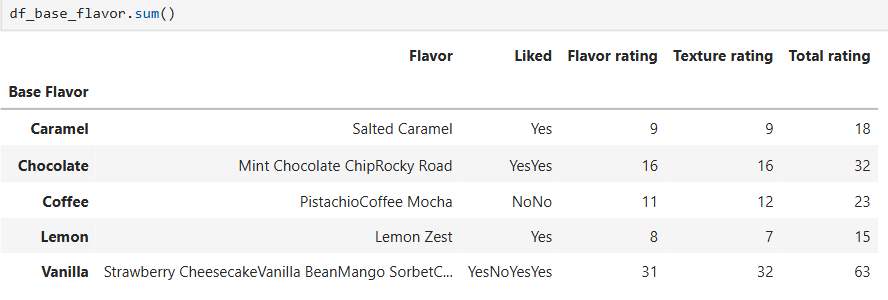
 



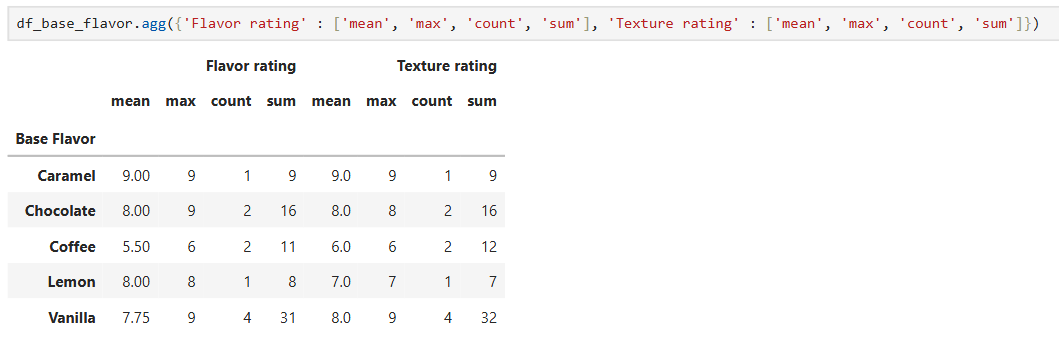
2. count()- it important to see the count of something for it will see the biased in some aggregate functions like mean(). Lower count might suggest the mean is being geared towards the highest or lowest value



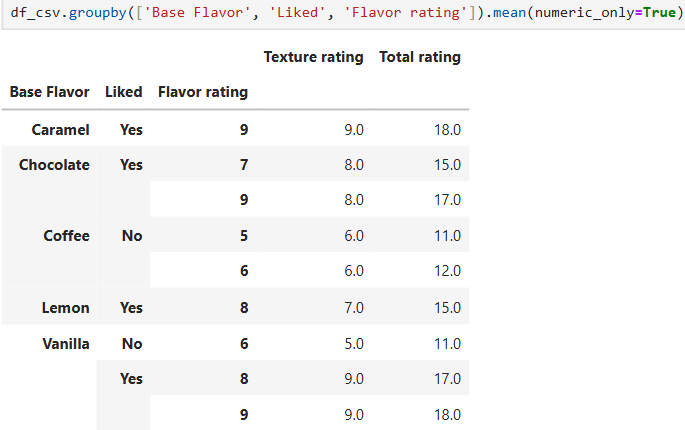
4. sum()

5. aggr()



6. groupby multiple columns

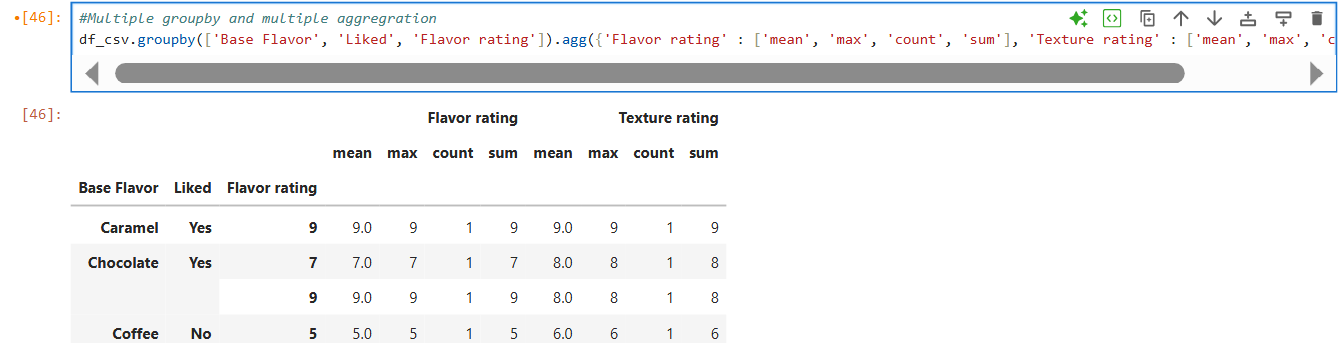


Note: you can’t use inplace=True with groupby(), because:

a. groupby() doesn't modify the original DataFrame — it returns a new GroupBy object.

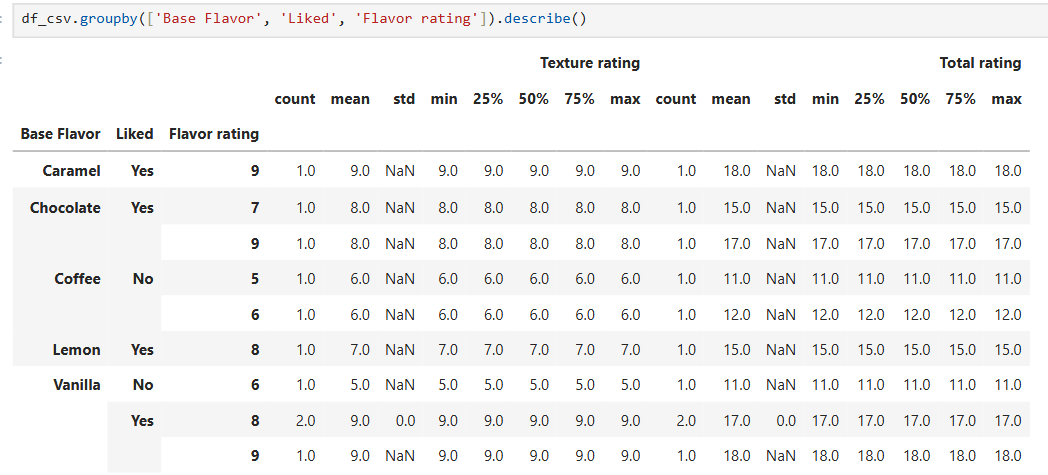
b. inplace is only available for certain methods like .drop(), .sort\_values(), .sort\_index(), .fillna(), .replace(), .rename(), .reset\_index(), .set\_index(), .update(), .astype(), .where(), .mask(), .interpolate(), .ffill(), .bfill(), and .clip(), but not for methods like groupby(), .agg(), .mean(), or most aggregation functions because those return new objects and do not modify the original DataFrame directly."

7. Multiple groupby and multiple aggregation

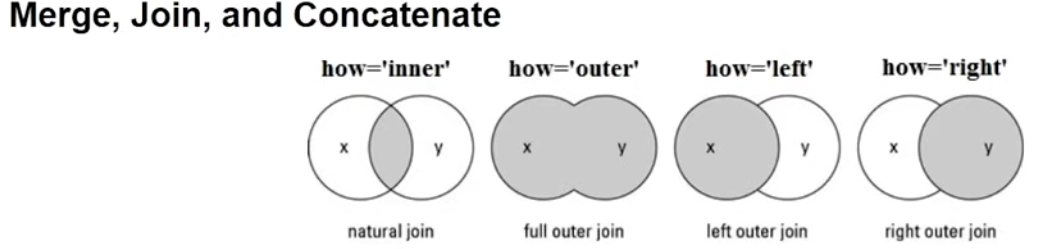


8. describe()- is a generalized structure that contains different aggregate function. ‘std’ is standard deviation.

Observation: You can see that there are 2 counts of vanilla that has liked=yes and flavor rating =8



MERGE , JOIN AND CONCATENATE in PANDAS

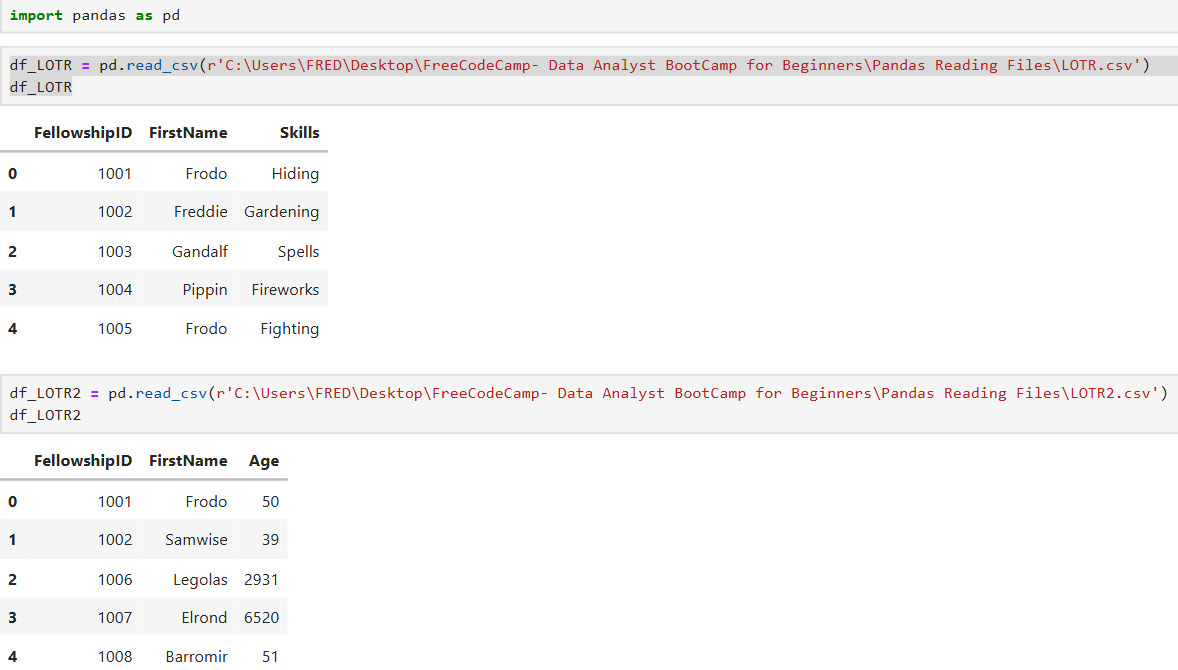


Natural join – returns everything that is similar with the 2 dataframe

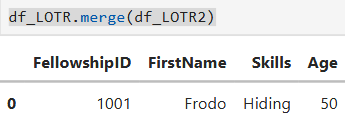
Full outer join – joins everything

Left outer join – return everything from the left dataframe and all similar data with the right

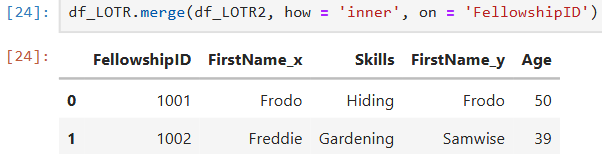
Right outer join – return everything from the right dataframe and all similar data with the left



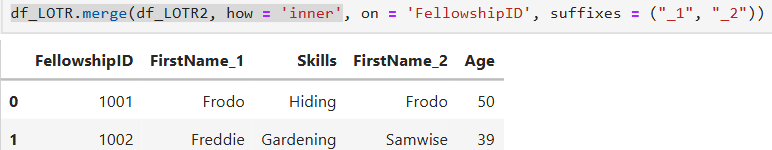
1. pd.merge() is like an inner join where only similar data on the first column will be displayed from the 2 data frames. The parameter is the right data frame. All columns are joined together.



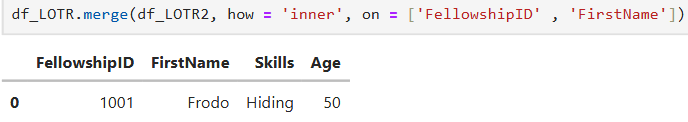
2. the ‘how’ parameter is the type of merge, the ‘on’ parameter is the column where the 2 data frame will merge. So other columns that has similar name and value will be displayed separately



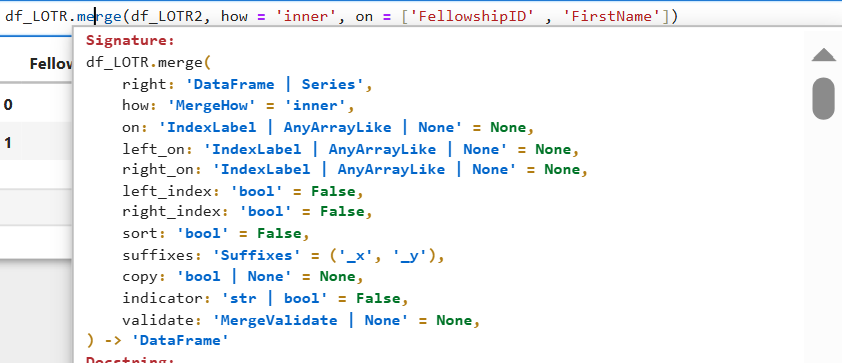
You can change the suffix on the name of the same columns



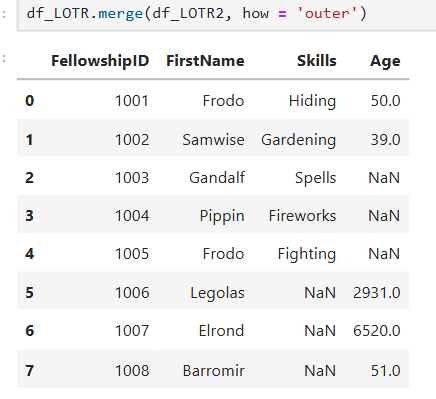
You can use multiple column join



These are all the parameters of merge (shift + tab)



3. outer(). Notice that in the outer FellowshipID ‘1002’ has not displayed .



ISSUE: The issue comes from the mismatch in the FirstName values between the two CSVs.

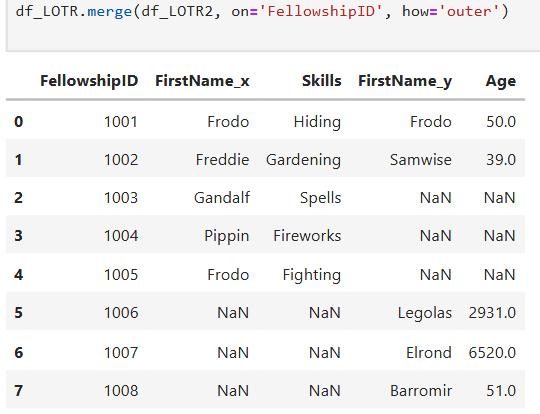
Here's what's happening:

* In df\_LOTR, the row with FellowshipID 1002 has FirstName = Freddie.
* In df\_LOTR2, that same FellowshipID 1002 has FirstName = Samwise

By default, merge() uses **all common columns** to perform the join. Since both FellowshipID and FirstName appear in both DataFrames, it tries to match on both. But because "Freddie" ≠ "Samwise", the row doesn't match and gets treated as separate entries (or possibly dropped depending on later operations).

**Fix: Merge on FellowshipID only**

To avoid this issue, you should tell merge() to use just the FellowshipID

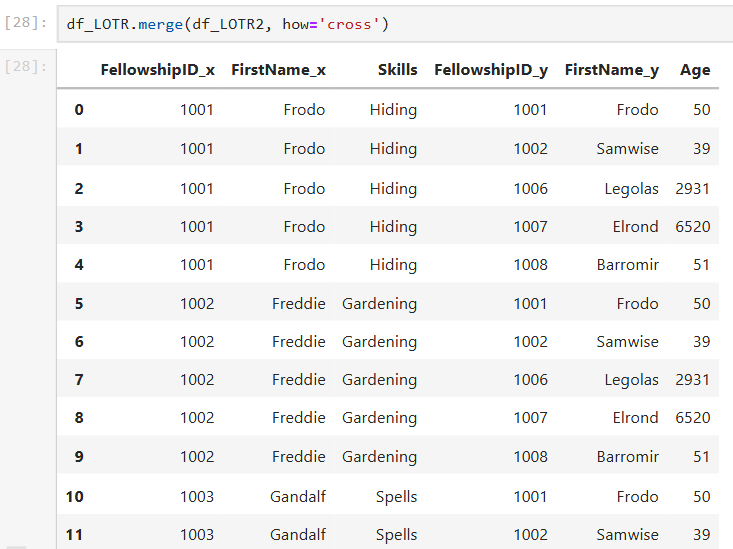


4. pd.cross() –cross join will compare each value on the left data frame to the all value on the right data frame.

The cross() method in pandas is used to compute the **Cartesian product** of two DataFrames — meaning it returns every possible combination of rows from the first DataFrame with rows from the second.

### What's the Practical Use?

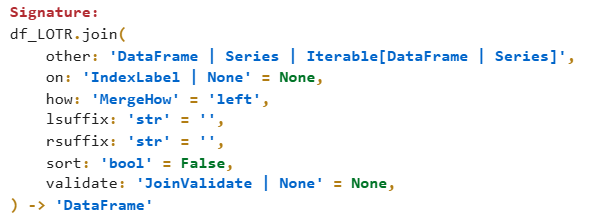
1. **Testing combinations**  
   If you're designing experiments or test cases, you might want to generate all combinations of parameters (like A/B testing, price vs. product, etc.).
2. **Scheduling or planning**  
   You could use it to create all possible combinations of:
   * Employees × Shifts
   * Teachers × Courses
   * Products × Locations
3. **Pairwise analysis**  
   In some machine learning or recommendation scenarios, you might want to pair every user with every item to score the

### Important Notes:

* The .merge(..., how='cross') option was introduced in **pandas 1.2.0+**.
* It's memory-intensive for large datasets since the number of resulting rows is len(df1) × len(df2).

5. pd.join() is similar with merge but it doesn’t work well with column name as index AND it needs many parameters for the query to work



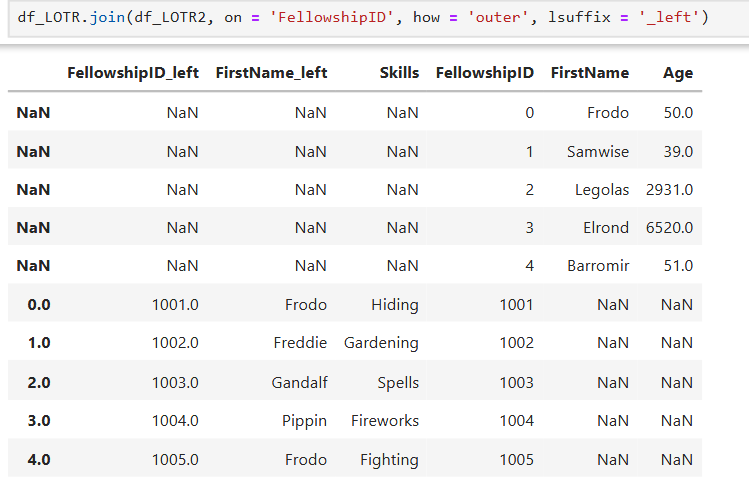
Even though there are already two parameters, these two lines will have error

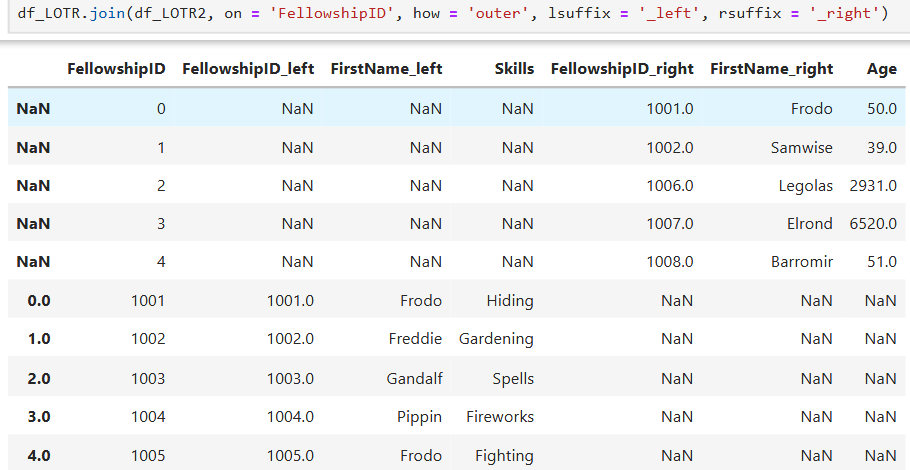


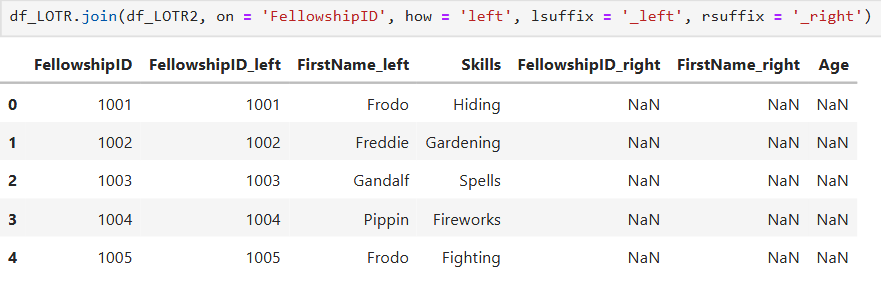


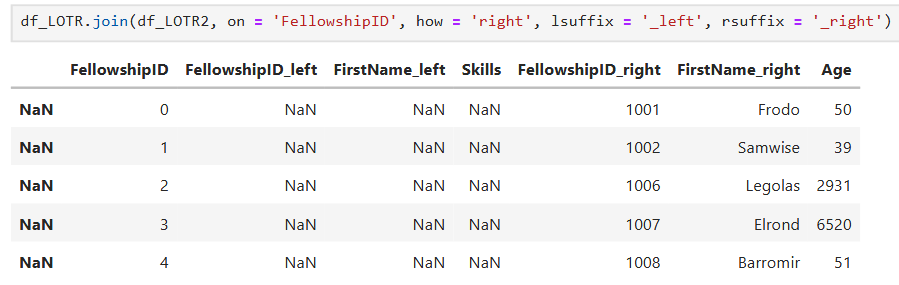


As you can see below, we were able to join the two tables using pd.join() but it is very confusing



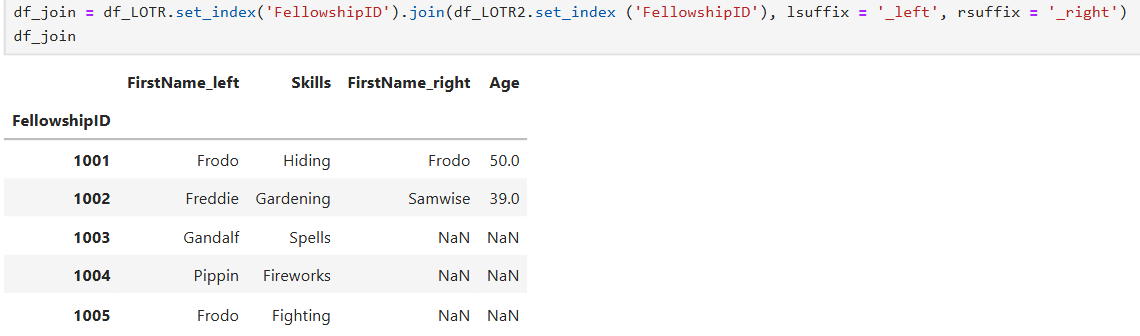




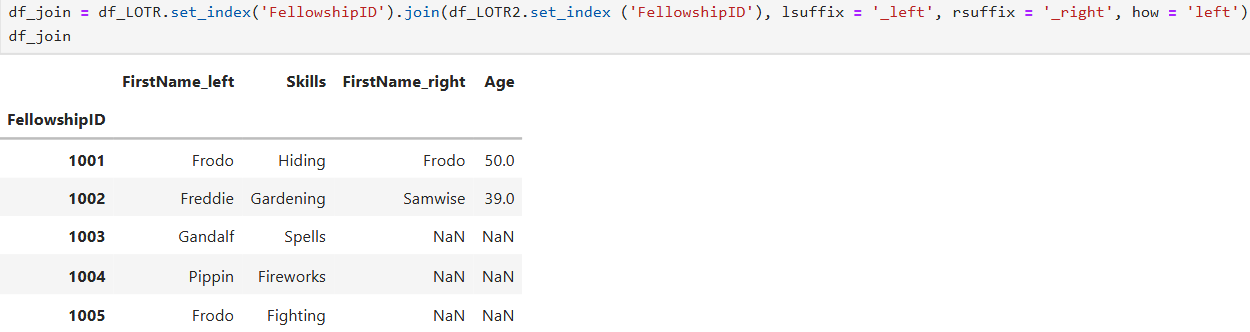


JOINS

6. pd.pd.join() works well with index

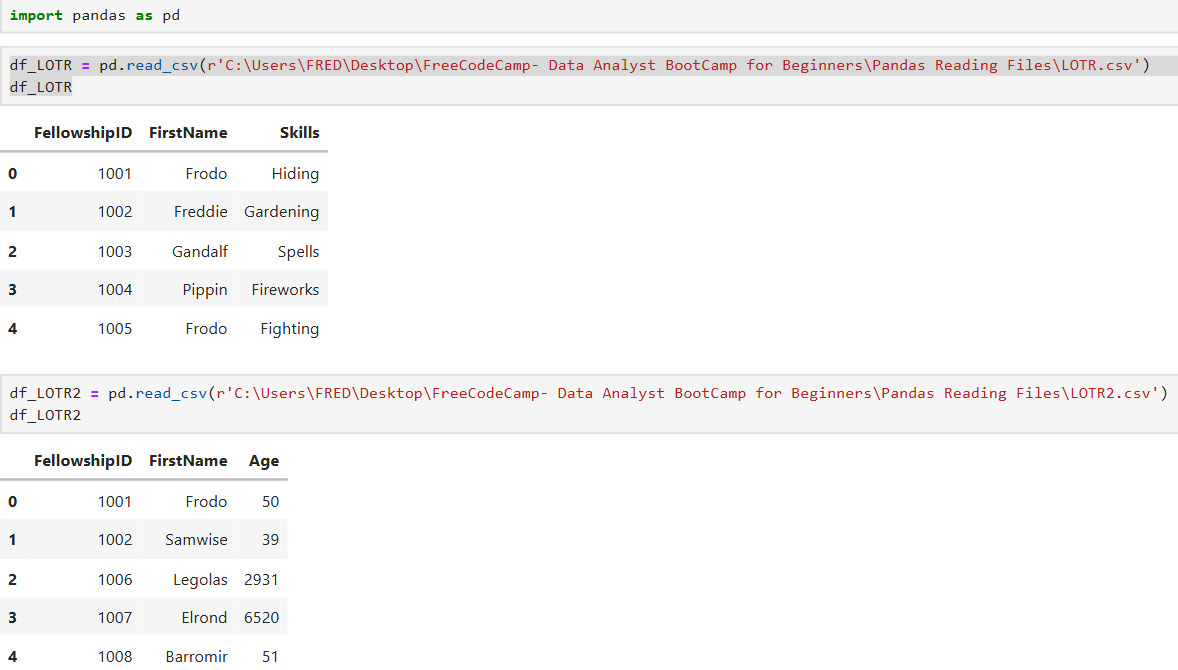
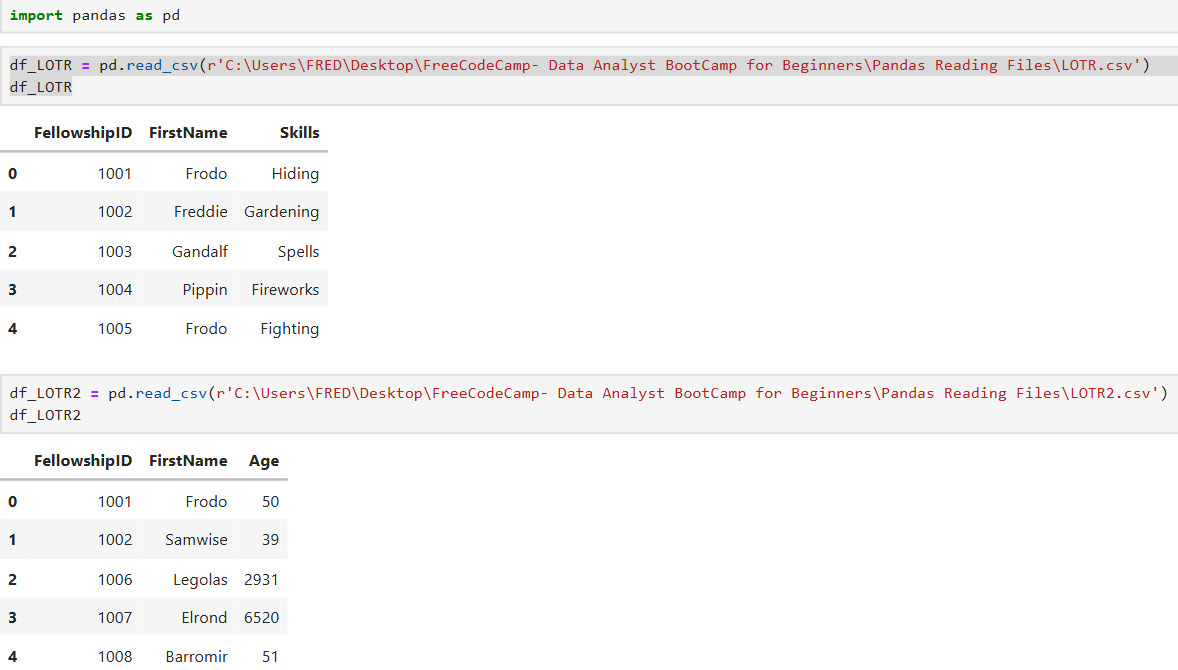


The result above is similar to left join because left join is the default joins





Original data for reference

CONCATENATE

Comparison with joins and merge

### pd.merge() - Combine two DataFrames based on one or more common columns or indexes like SQL JOIN.

**Use when:**

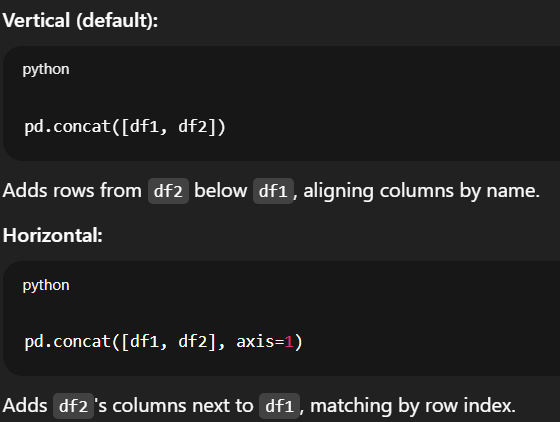
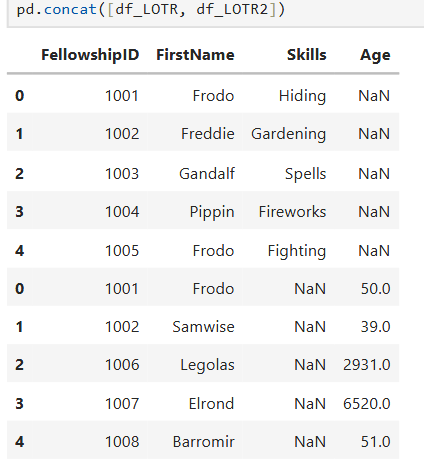
* You need to match rows using key columns.
* You're doing something relational — like linking customers to orders.

### pd.join() - A more convenient version of merge() that joins based on index by default, or on a key column.

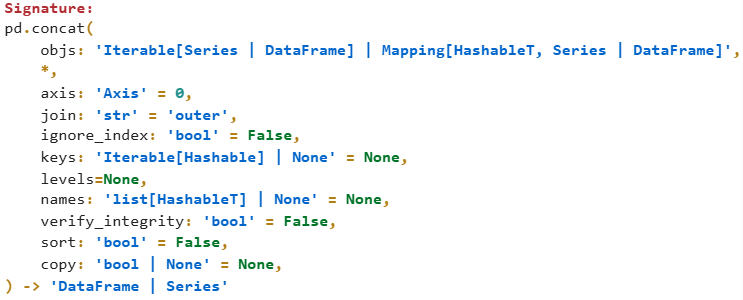
**Use when:**

* You want to join by index or a simple key.
* You're doing basic joins (not multi-key or complex matching).

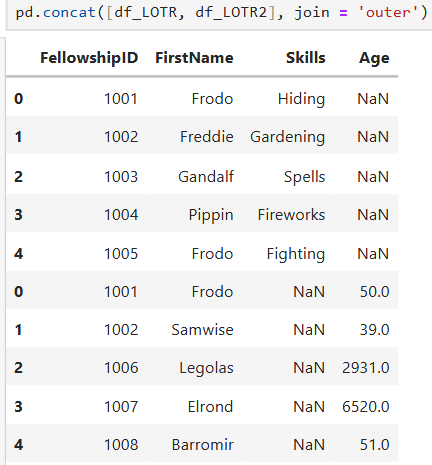
7. pd.concat() - stack DataFrames vertically (on top of each other) or horizontally (side by side).

### 



‘inner’ joins data frames on columns that are the same and put on top of each other while outer combines all column

Axis =0 is the default axis, it combines vertically. Axis=1 combines horizontally

