Series:

--> A Pandas Series is a ***one-dimensional labeled array in Python*** that is capable of ***holding any data type***—such as integers, floats, strings, Python objects, etc.

--> A ***Series Combines*** the best features of a ***list and a dictionary***

--> A ***series maintains*** a single collection if ordered value.

***One-dimensional:***

--> Like a ***column in a table*** or a single list of values.

***Labeled index:***

--> Each item in the ***series has a label (index) by default***, starting from 0 unless specified.

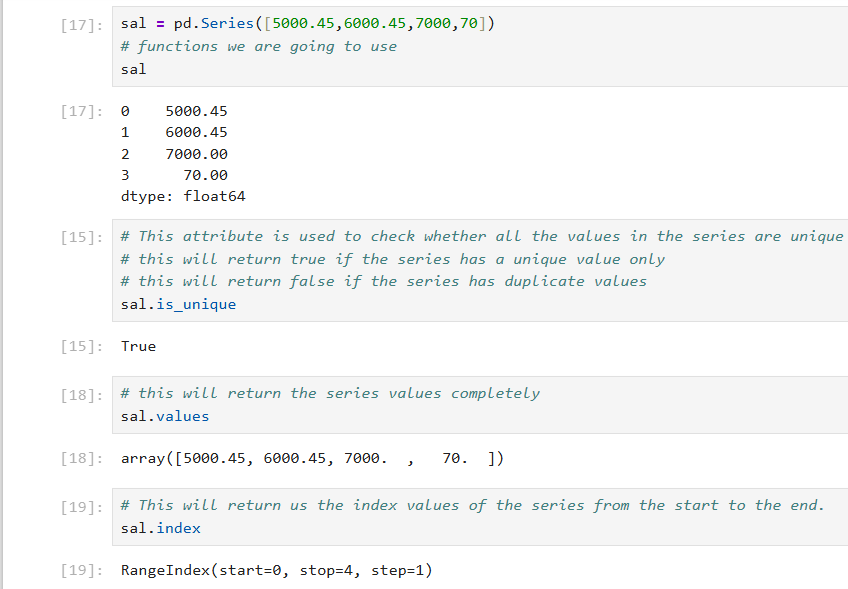
***Methods and Attributes:***

***In pandas series:***

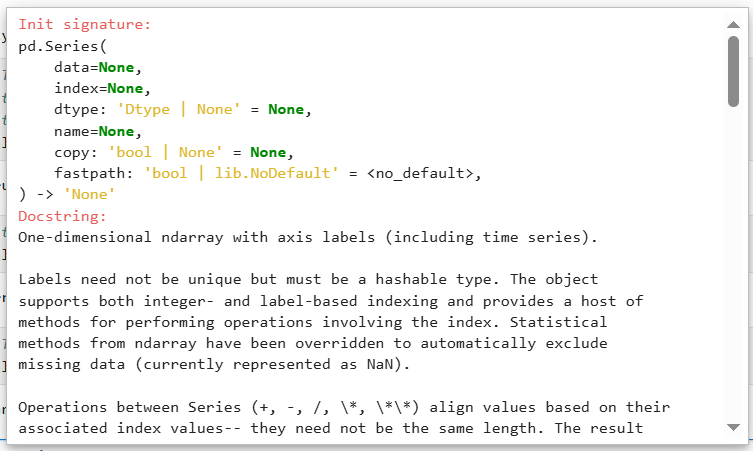
--> ***attributes*** tells about the object and does not required any paraenthesis

--> ***methods*** is a behavior or action available in the series





***Arguments in the pd.Series() syntax.***



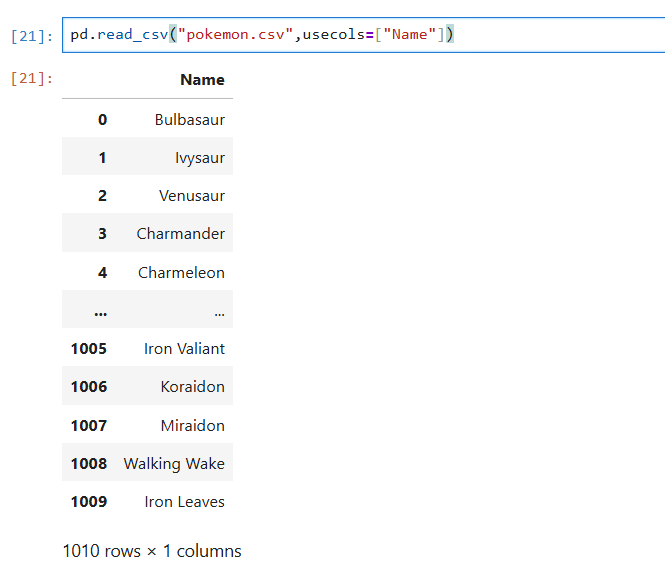
***data*** which will take the input source , ***index*** which will take custom index value which we map for the data if ***do not give index*** by default ***from 0 index position*** will get assigned to each values in the data.

Import Series with the ***pd.read\_csv*** Function

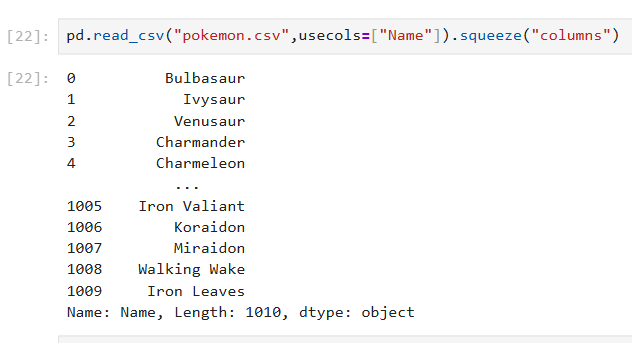
A ***CSV is a plain text file*** that uses line breaks to ***separate rows and commas to separate row values***.

1. Pandas ships with many ***different read\_ functions for different types of files***.
2. The ***read\_csv function*** accepts many different parameters. The first one specifies the file ***name/path***.
3. The ***read\_csv function*** will import the ***dataset as a DataFrame, a 2-dimensional table***.
4. The ***usecols parameter*** accepts a list of the ***column(s) to import***.
5. The ***squeeze method*** converts a ***DataFrame to a Series***.
6. By using the ***squeeze method*** we are going to convert the ***dataframe to series*** where we need to have ***only one column*** in the dataframe
7. ***Pandas Series method*** can support ***only one column*** and provide ***one dimesional array view***

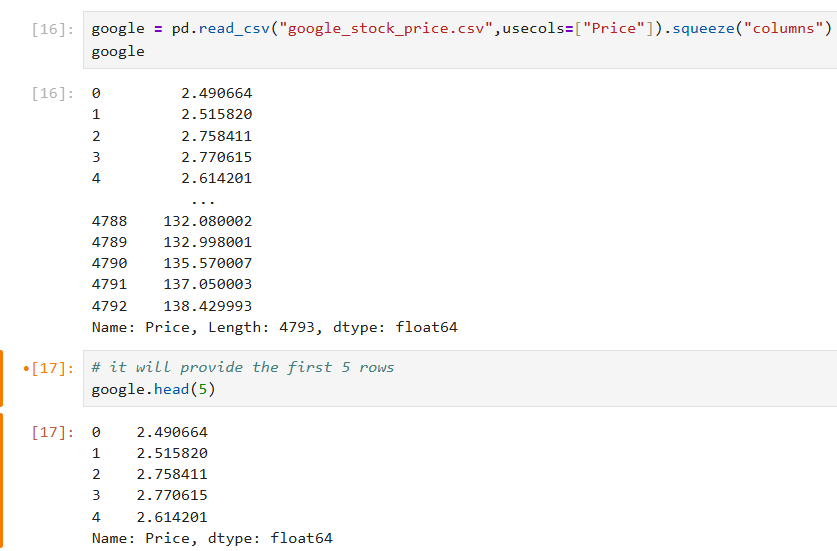
Two dimesional View in DataFrames:



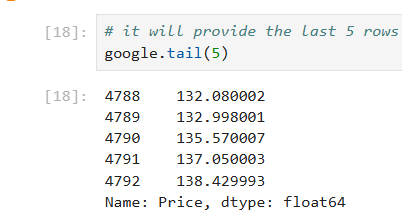
After converting the dataFrames to Series by using the squeeze method we can view the one dimesional view:



The ***head method*** returns a number of rows from the beginning of the series.



The ***tail method*** returns a number of rows from the end of the series.



***Passing Series to Python's Built-In Functions***

The ***len function*** returns the ***length of the Series***.

The ***type function*** returns the ***type of an object***.

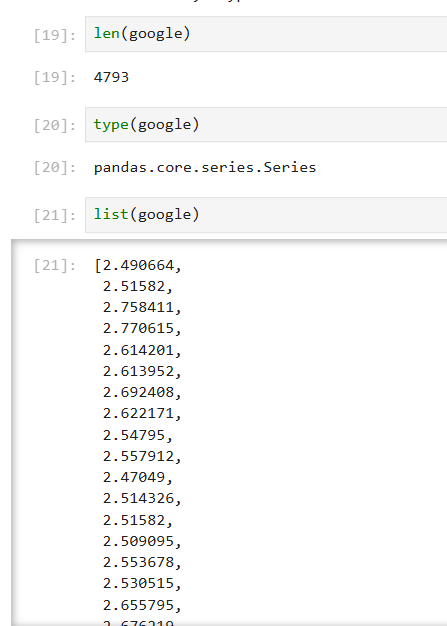
The ***list function*** converts the ***Series to a list***.

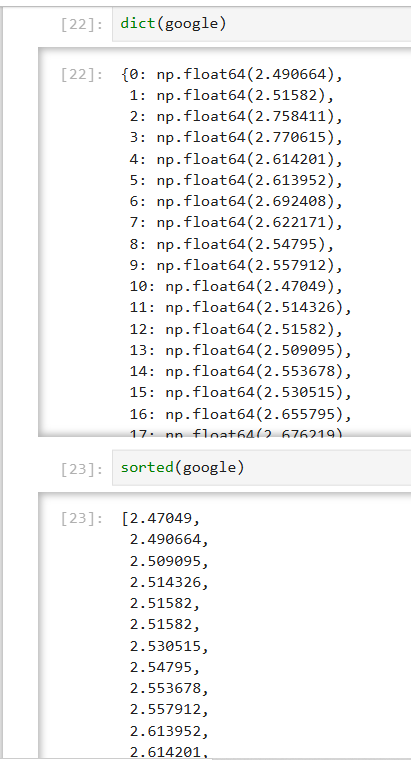
The ***dict function*** converts the ***Series to a dictionary***.

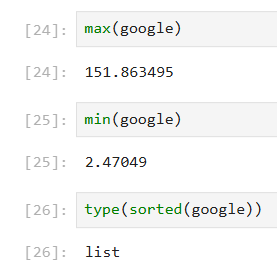
The ***sorted function*** converts the ***Series to a sorted list***.(ascending to descending order)

The ***max function*** returns the ***largest value*** in the Series.

The ***min function*** returns the ***smallest value*** in the Series.







***Check for Inclusion with Python's in Keyword***

The ***in*** keyword checks ***if a value exists within an object***.

The ***in*** keyword will look ***for a value in the Series's index***.

Use the ***index and values*** attributes to ***access "nested" objects*** within the Series.

Combine the ***in keyword*** with values to search within the ***Series's values.***

***The sort\_values Method***

The ***sort\_values*** method sorts a ***Series values in order***.

By default, pandas **applies** an ***ascending sort (smallest to largest)***.

Customize the ***sort order*** with the ***ascending parameter***.

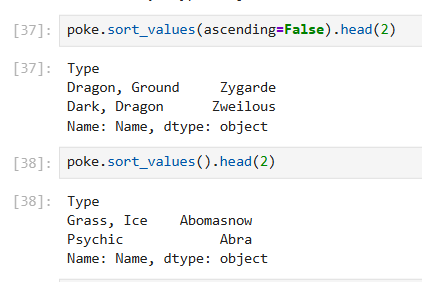
***The sort\_index Method***

The ***sort\_index*** method sorts a ***Series by its index***.

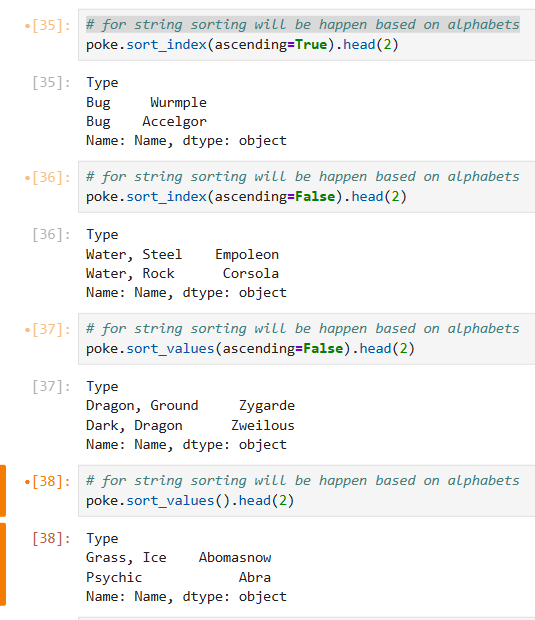
The ***sort\_index method*** also accepts an ascending parameter to ***set sort order***.

In the eaxmple ***pokemon*** we can see we have the ***index\_col as Type*** and value as ***Name***





For the ***string sorting based on the alphabtes*** only sort will be happening.



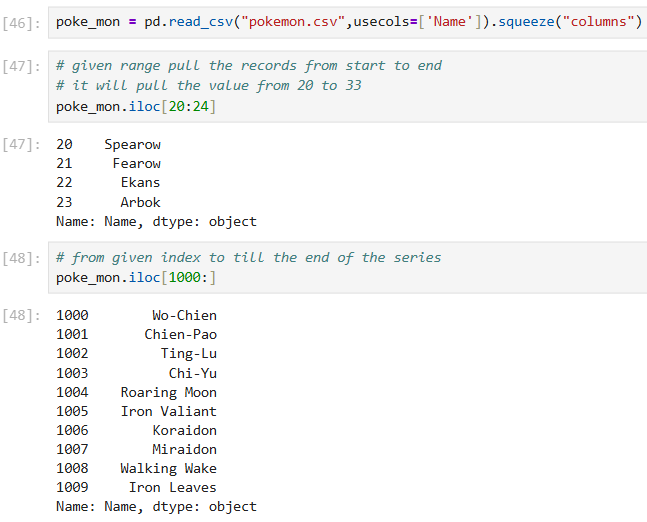
***Extract Series Value by Index Position***

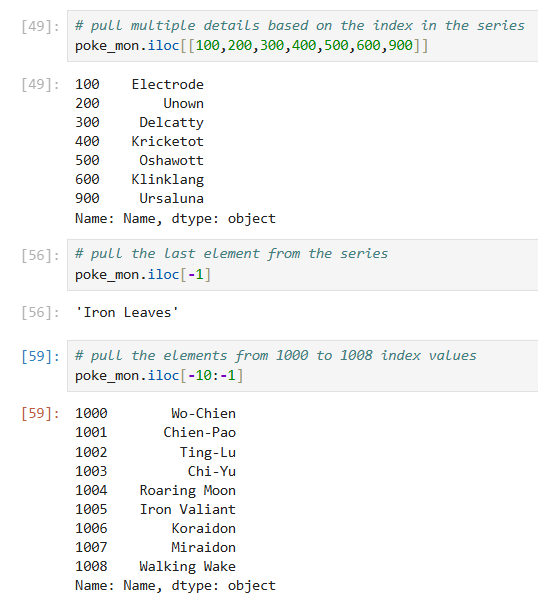
Use the ***iloc accessor*** to extract a ***Series value*** by its ***index position***.

***iloc*** is short for ***"index location"***.

***Python's list slicing*** syntaxes (slices, slices from start, slices to end, etc.) are supported with ***Series objects***.

Syntax: ***Object.iloc[]***



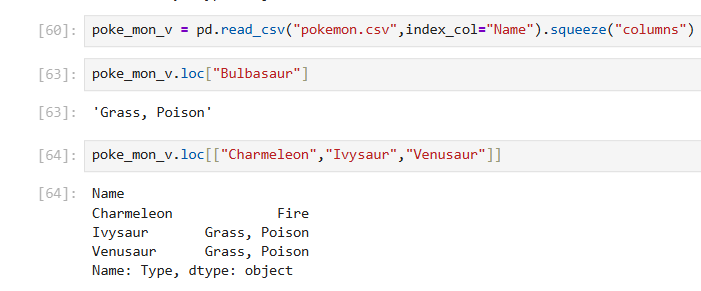


***Extract Series Value by Index Label***

Use the ***loc accessor*** to ***extract a Series value*** by its ***index label***.

Pass a list to ***extract multiple values*** by ***index label***.

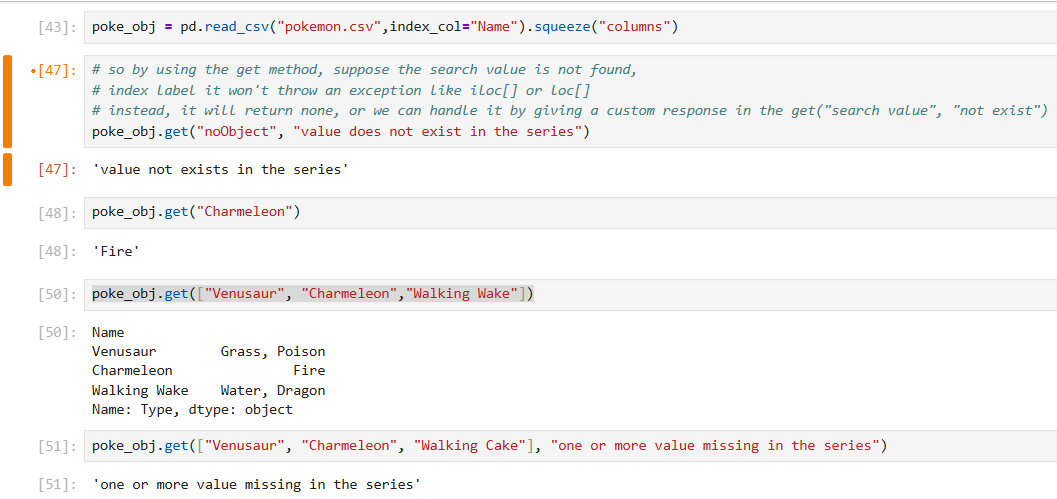
If ***one index label/position*** in the ***list does not exist***, Pandas will ***raise an error***.



***The get Method on a Series***

The ***get method extracts a Series value by index label***. It is an alternative option to square brackets.

The ***get method's second argument sets the fallback value*** to return if the label/position does not exist.



***Overwrite a Series Value:***

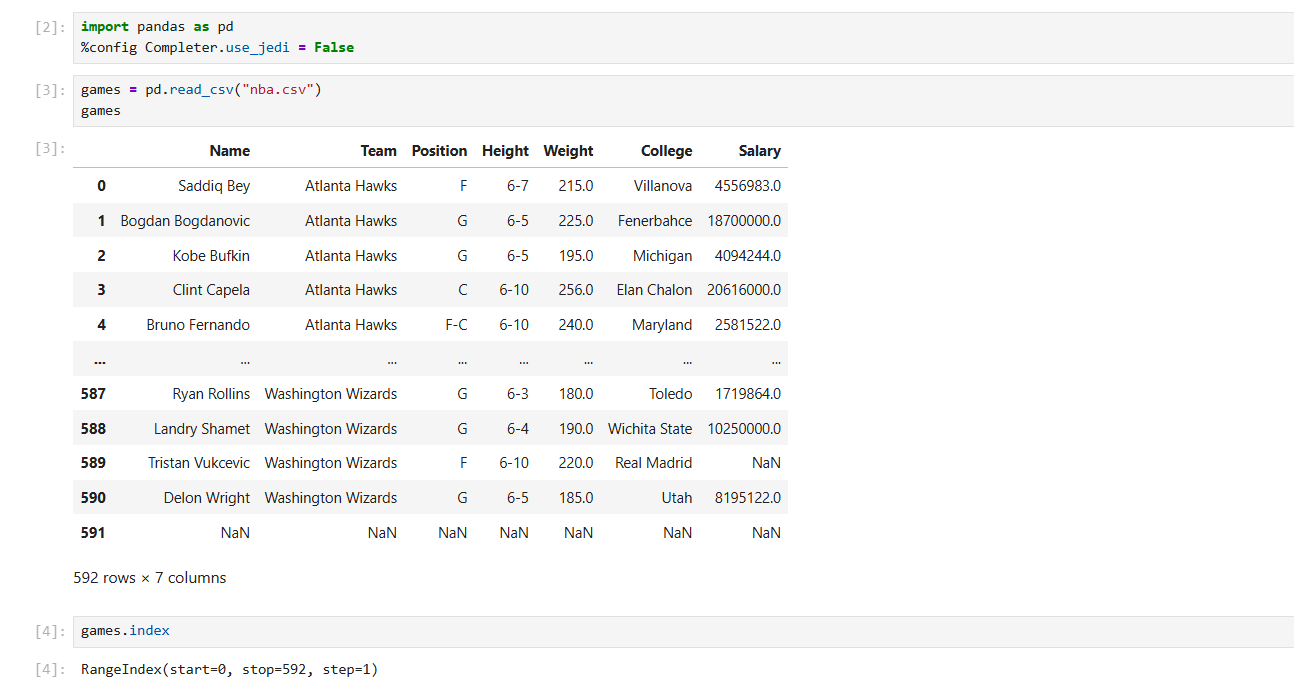
Use the ***loc/iloc accessor to target an index label/position***, then use an equal sign to provide a new value

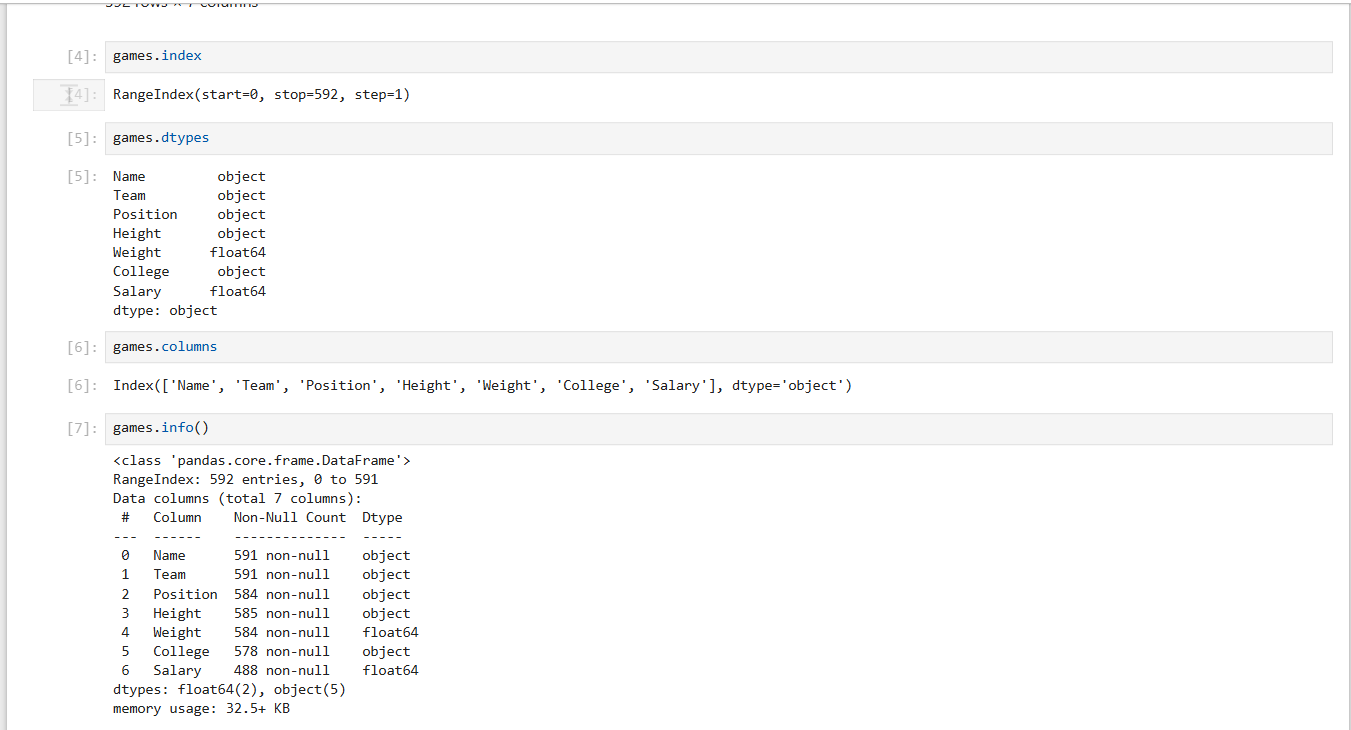




***Methods and Attributes between Series and DataFrames***

* A DataFrame is a 2-dimensional table consisting of rows and columns.
* Pandas uses a ***NaN designation*** for cells that have a missing value. It is short for "not a number". Most operations on NaN values will produce NaN values.
* Like with a Series, Pandas assigns an ***index position/label*** to each DataFrame row.
* The DataFrame and Series have common and exclusive methods/attributes.
* The ***hasnans attribute exists only a Series***. The c***olumns attribute exists only on a DataFrame***.
* Some methods/attributes will return ***different types of data***.
* The info method returns a summary of the pandas object.





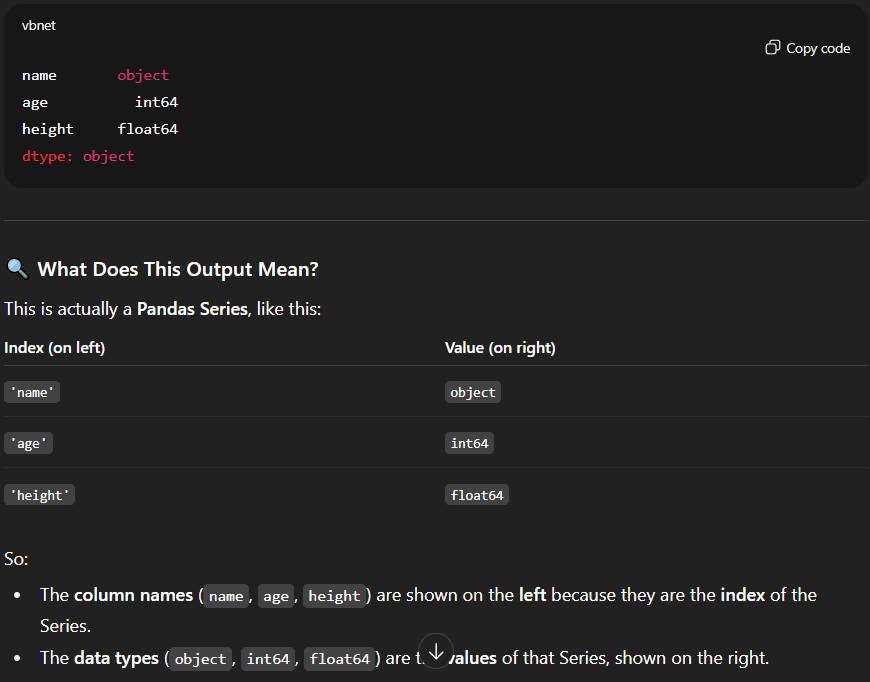
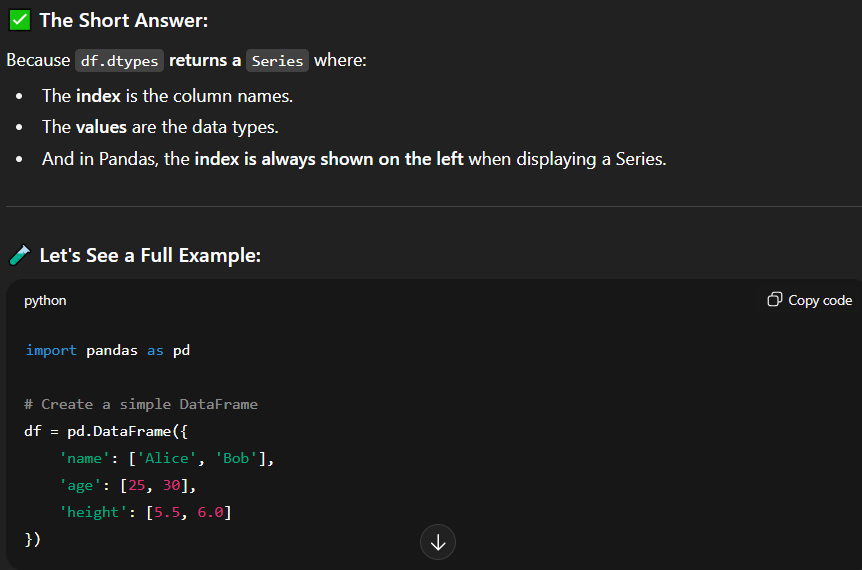
***Differences between Shared Methods***

The ***sum method adds a Series's values***.

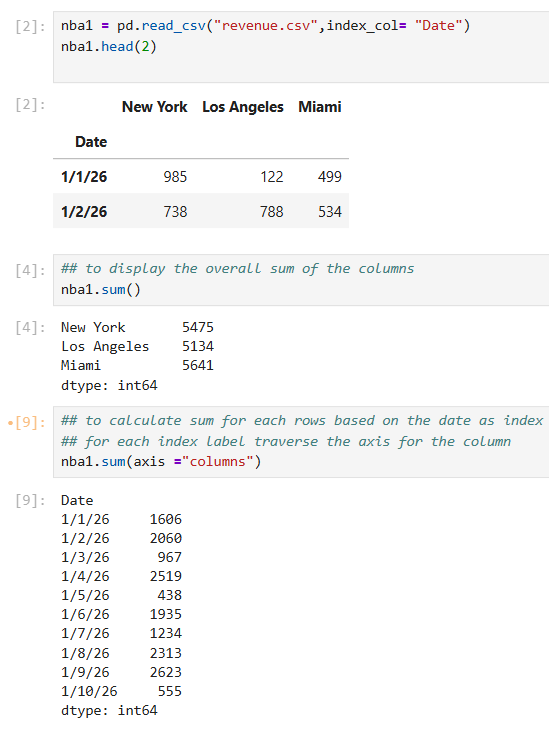
On a DataFrame, the ***sum method*** defaults to adding the values by ***traversing the index (row values)***.

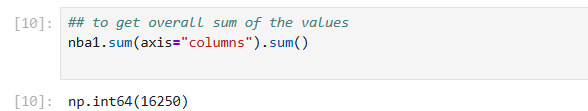
The ***axis parameter customizes*** the direction that we add across. Pass ***"columns" or 1 to add "across" the columns***.

***#Important note in the dataFrame***



Examples:





***# Important Note***

Once you ***readcsv file*** it first keep the data in the form of DataFrame

Then when you tried to select a single column or multiple column at that time it will get converted to series.

***Select One Column from a DataFrame***

We can use ***attribute syntax (df.column\_name)*** to select a column from a DataFrame. The ***syntax will not work*** if the ***column name has spaces***.

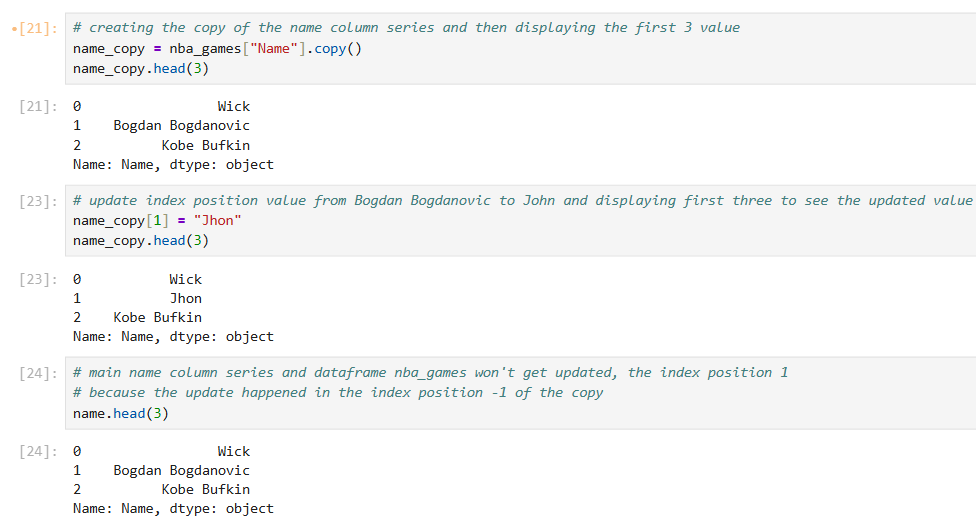
We can also use ***square bracket syntax (df ["column name"])*** which will work for any column name.

Pandas extracts a ***column from a DataFrame as a Series***.

The **Series is a view**, so changes to the Series will affect the DataFrame.

Pandas will ***display a warning if you mutate the Series***. Use the copy method to create a duplicate.

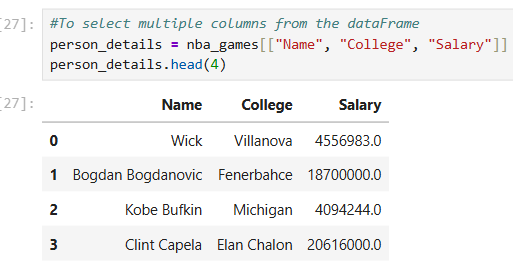




***Select Multiple Columns from a DataFrame***

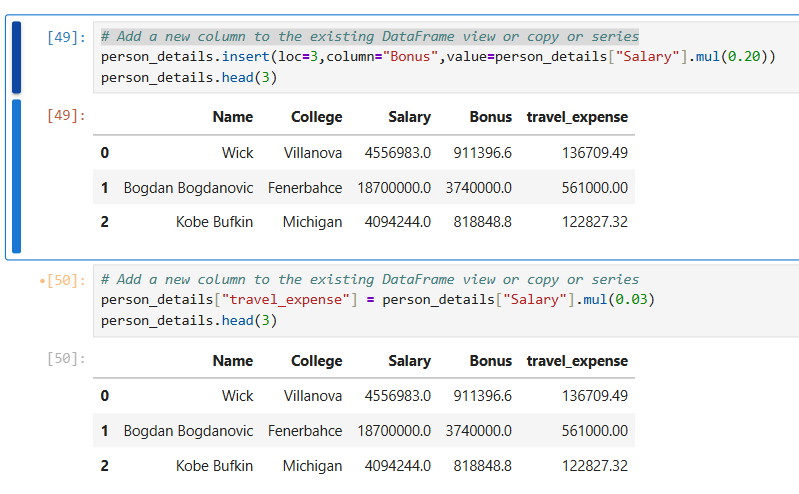
Use square brackets with a list of names to extract multiple DataFrame columns.

Pandas stores the result in a new DataFrame (a copy).



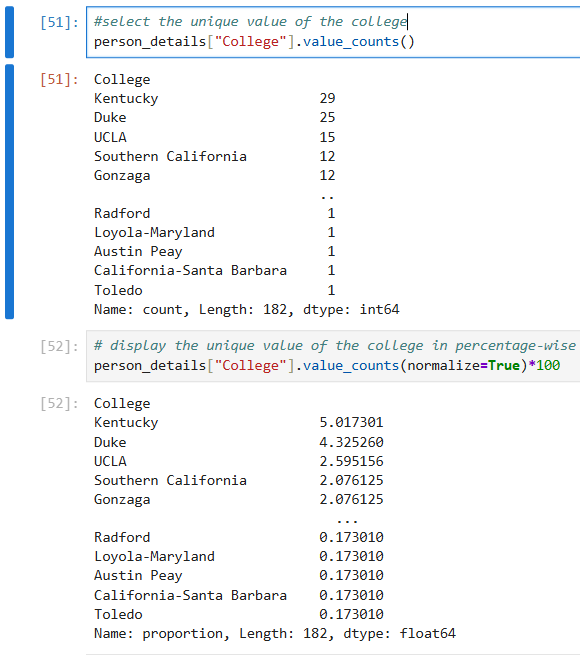
***Add New Column to DataFrame***

* Use ***square bracket extraction*** syntax with an ***equal sign to add a new Series*** to a DataFrame.
* The ***insert method*** allows us to ***insert an element*** at a specific ***column index***.
* On the ***right-hand side***, we can reference an ***existing DataFrame column*** and perform a **broadcasting operation** on it to create the new Series.



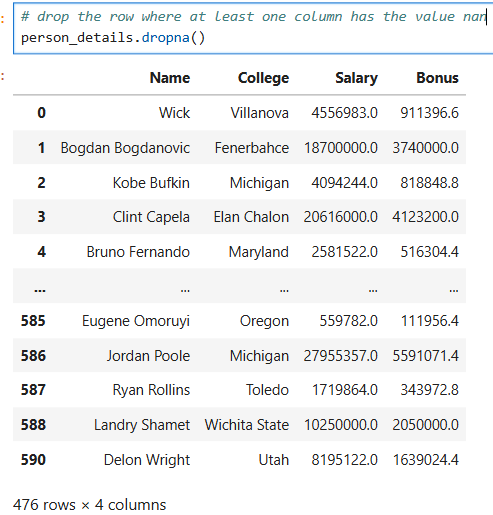
***A Review of the value\_counts Method***

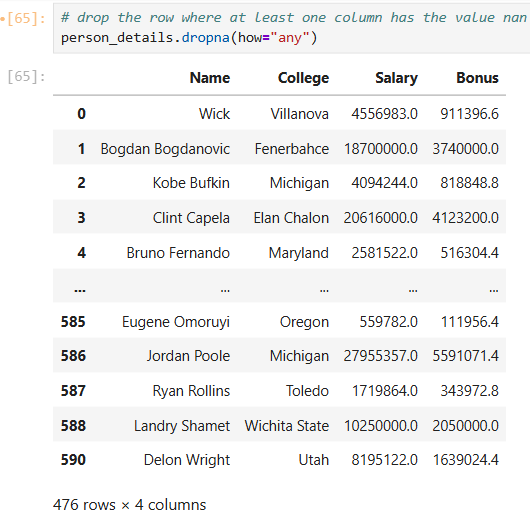
The ***value\_counts method*** counts the number of times that each unique value occurs in a Series.



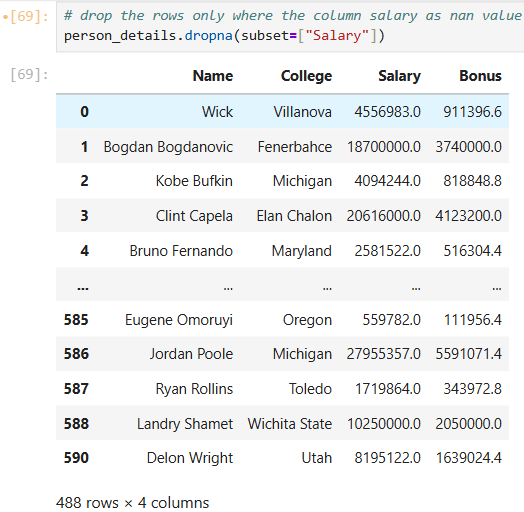
***Drop Rows with Missing Values***

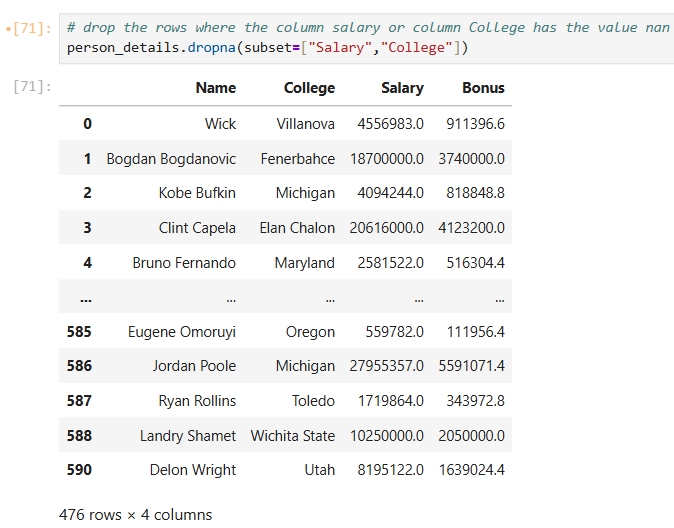
* Pandas uses a ***NaN designation*** for cells that have a missing value.
* The ***dropna method deletes rows with missing values***. Its default behavior is to ***remove a row*** if it has any missing values.
* Pass the how parameter an ***argument of "all" to delete rows*** where all the values are NaN
* The subset parameters ***customizes/limits the column***s that pandas will use to ***drop rows with missing values***.





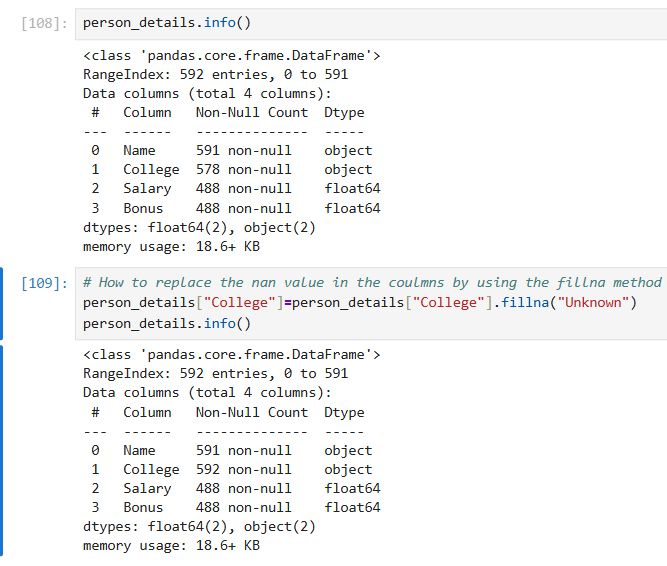






***Fill in Missing Values with the fillna Method***

* The ***fillna method replaces missing NaN values*** with its argument.
* The ***fillna method is available*** on both ***DataFrames and Series***.
* An ***extracted Series is a view*** on the ***original DataFrame***, but the fillna method returns a copy.

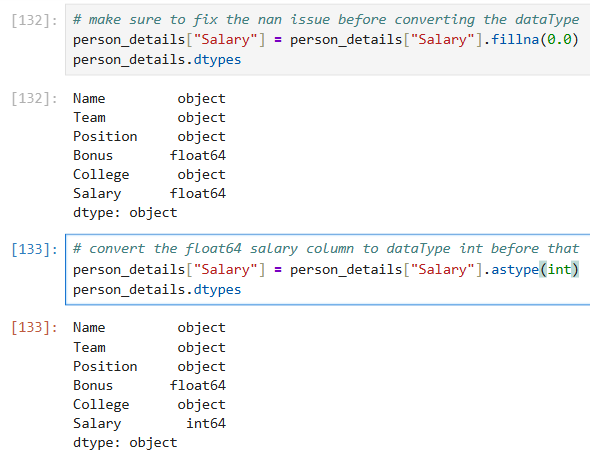


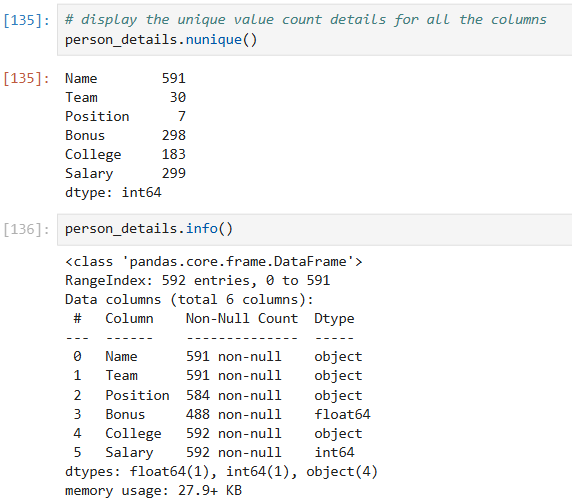
***The astype Method I***

* The ***astype method converts a Series's values*** to a specified type.
* Pass in the ***specified type as either a string*** or the core Python data type.
* Pandas ***cannot convert NaN values to numeric type***s, so we need to ***eliminate/replace them*** before we perform the conversion.
* The ***dtypes attribute returns a Series*** with the ***DataFrame's columns*** and their types.

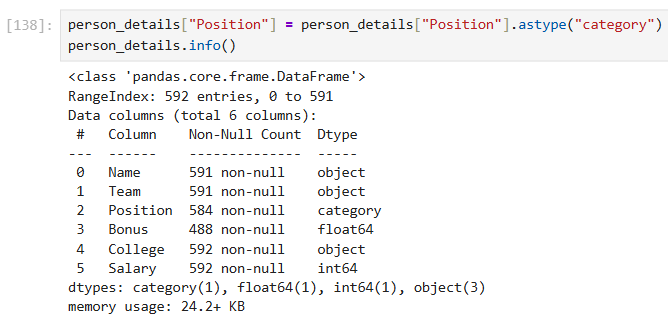
***The astype Method II***

* The ***category type is ideal for columns*** with a limited number of unique values.
* The ***nunique method will return a Series*** with the number of unique values in each column.
* With ***categories, pandas does not create a separate value in memory*** for each "cell". Rather, the cells point to a ***single copy for each unique value***.





By using the ***nunique() method*** that will return all the unique value count in the column so the column which have very less number can be changed to ***dataType as category. By doing this you can see there is memory optimization happened which helps to peform the cation much faster.***



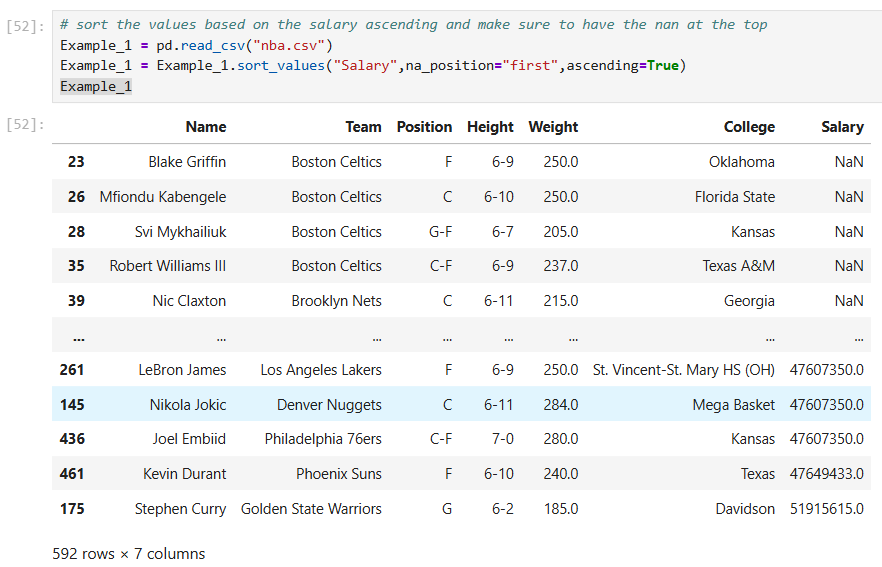
Once we changed the dataType of Position to category you can see the memory optimization is happened.

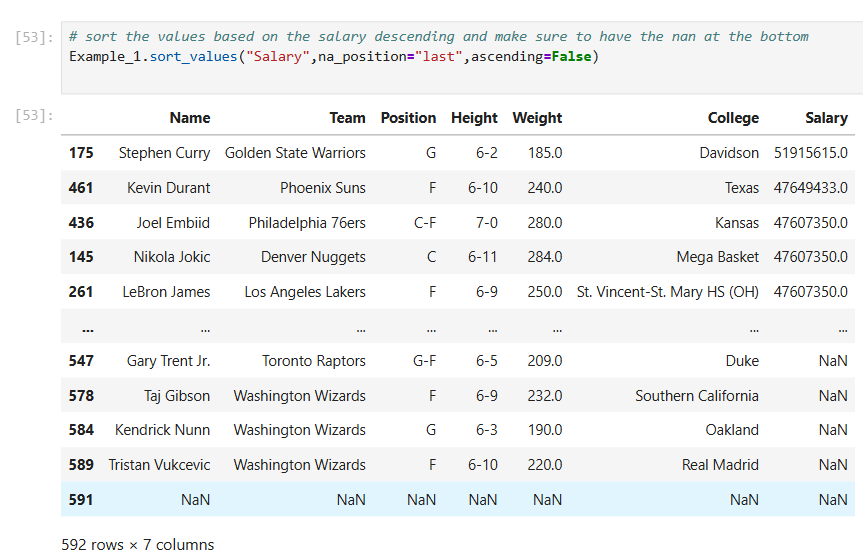
***Sort a DataFrame with the sort\_values Method I***

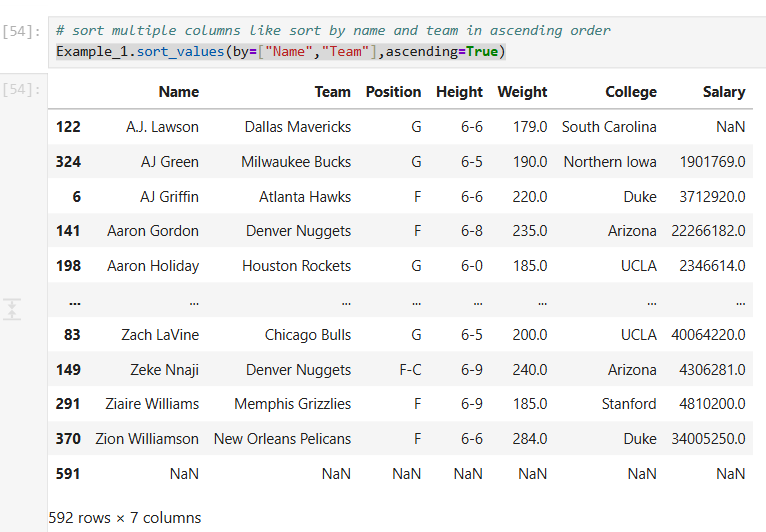
* The ***sort\_values method*** sorts a ***DataFrame by the values in one or more columns***. The default sort is an ascending one (alphabetical for strings).
* The first parameter ***(by) expects the column(s) to sort by***.
* If sorting by a ***single column, pass a string with its name***.
* The ascending parameter ***customizes the sort order***.
* The ***na\_position parameter*** customizes where ***pandas places NaN values***.

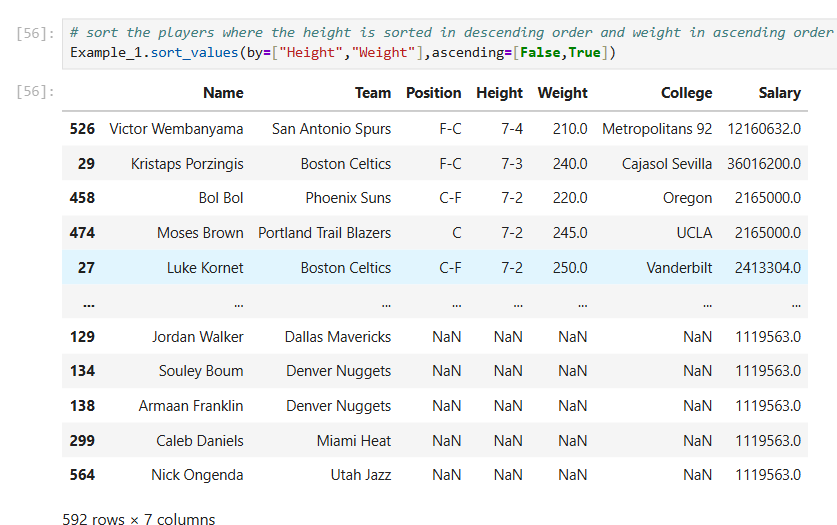
***Sort a DataFrame with the sort\_values Method II***

* To ***sort by multiple columns***, pass the by ***parameter a list of column names***. Pandas will sort in the ***specified column order*** (first to last).
* Pass the ***ascending parameter a Boolean*** to sort ***all columns in a consistent order*** (all ascending or all descending).
* ***Pass ascending a list*** to customize the sort order per column. The ascending list length must match the by list.







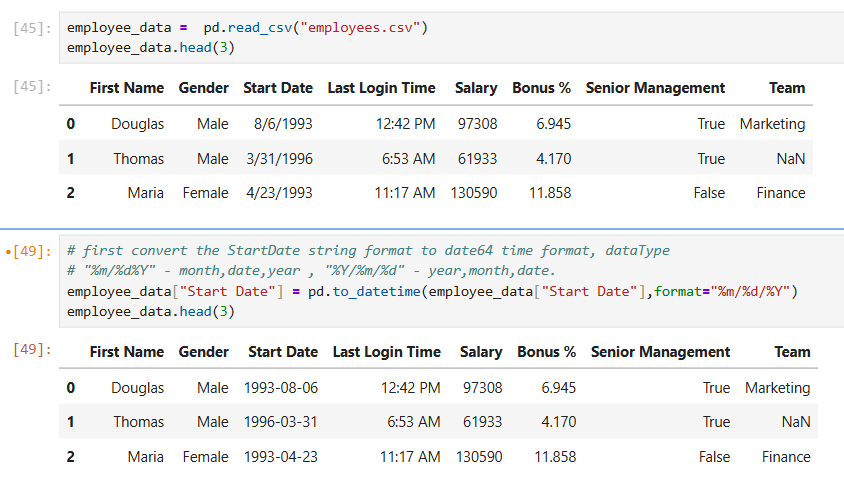


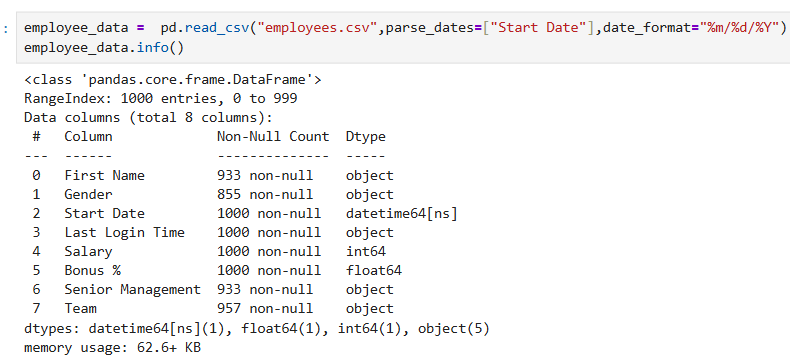
***This Module's Dataset + Memory Optimization***

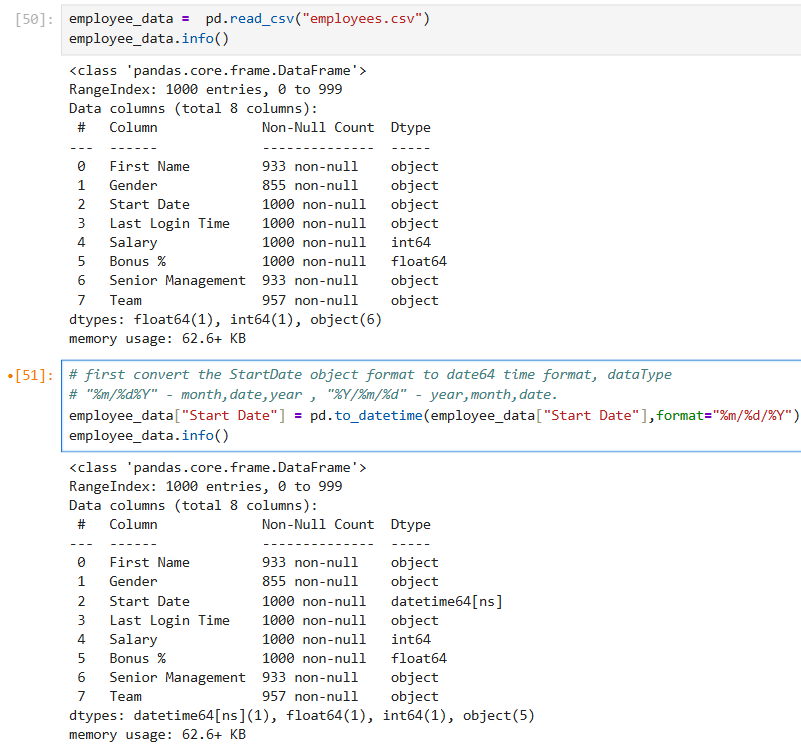
* The ***pd.to\_datetime*** method converts a ***Series to hold datetime*** values.
* The ***format parameter*** informs pandas of the format that the ***times are stored in***.
* We ***pass symbols designating*** the segments of the string. For example, ***%m means "month" and %d means day.***
* The ***dt attribute reveals*** an object with many ***datetime-related*** attributes and methods.
* The ***dt.time attribute*** extracts only the time from each value in a ***datetime Series***.
* Use the ***astype method*** to convert the values in a ***Series to another type***.
* The ***parse\_dates parameter*** of read\_csv is an ilternate way to ***parse strings as datetimes***

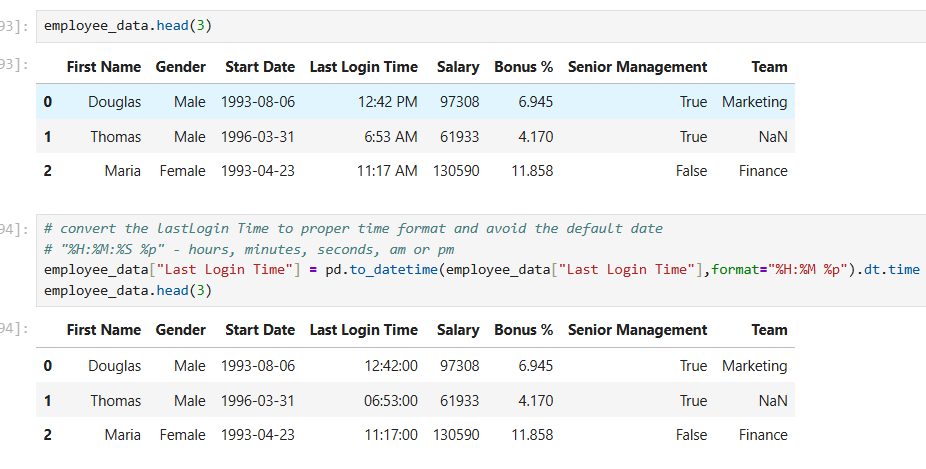
#Note: To optimize the data memory please follow the below format

1. If the column as very less number unqiue value please change the column dataType as Category
2. If you find any column only as Boolean value then convert the dataType of the column as boolean
3. If you find the date and time convert the format which is being recognized by the data frame.



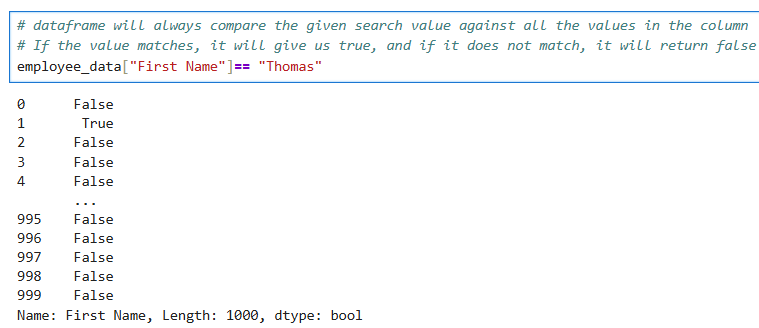


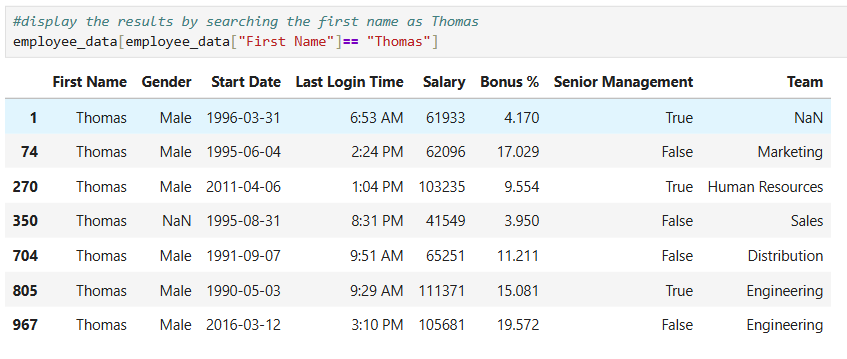


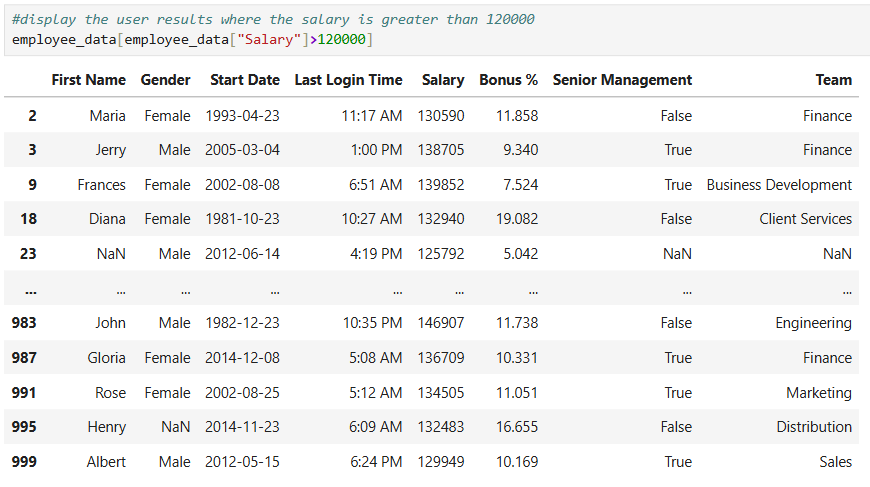


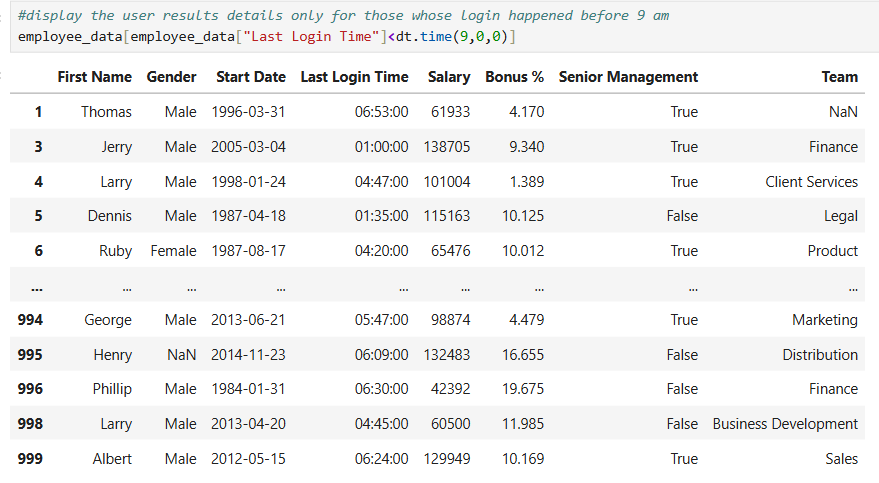
***Filter A DataFrame Based On A Condition***

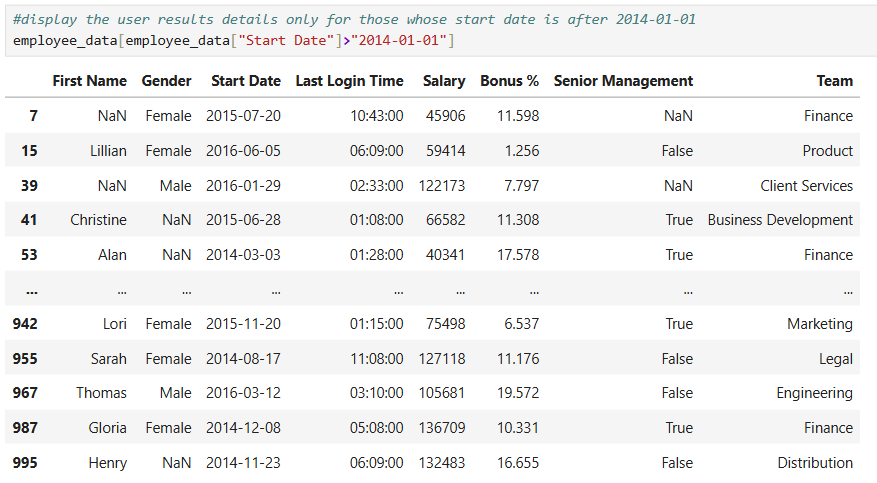
* Pandas needs a Series of Booleans to perform a filter.
* Pass the Boolean Series inside square brackets after the DataFrame.
* We can generate a Boolean Series using a wide variety of operations (equality, inequality, less than, greater than, inclusion, etc)



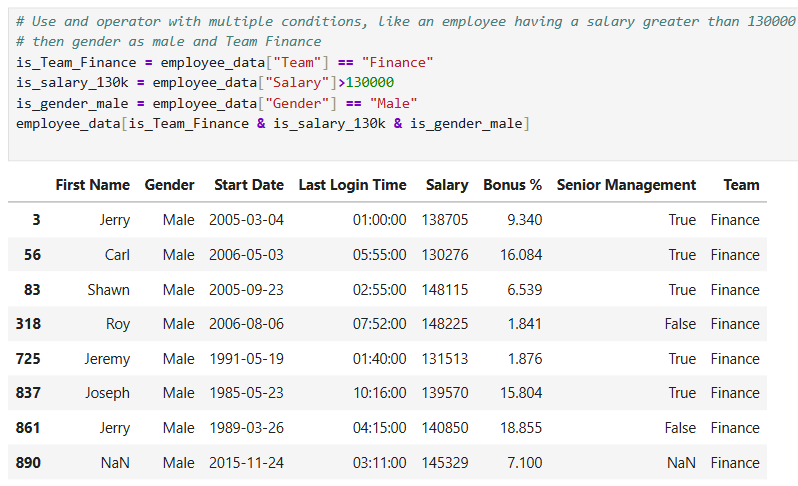


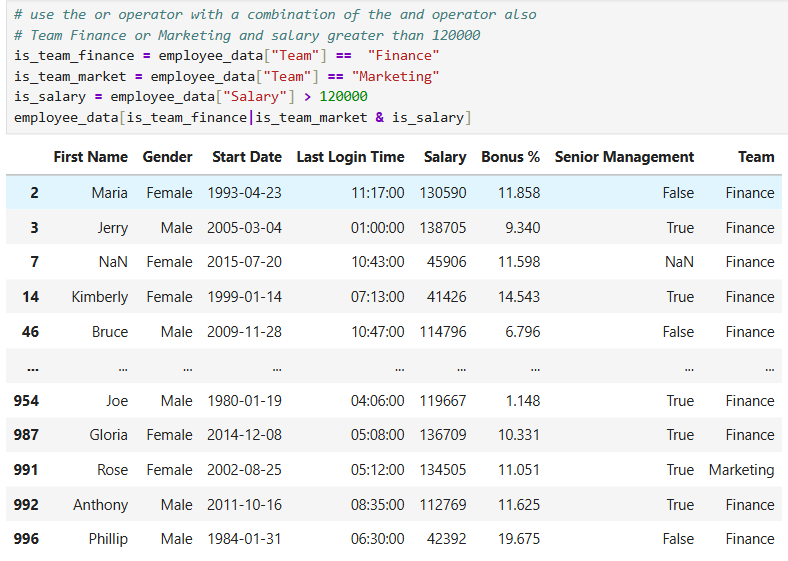






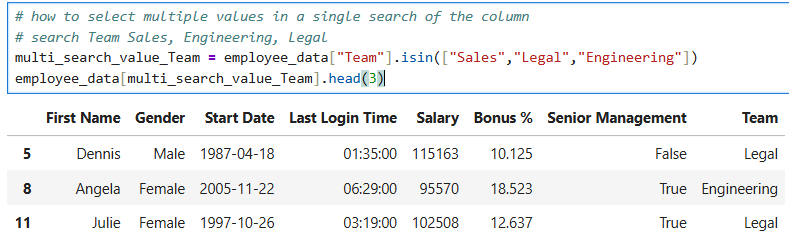
***And Operator and Or Operator UseCases.***





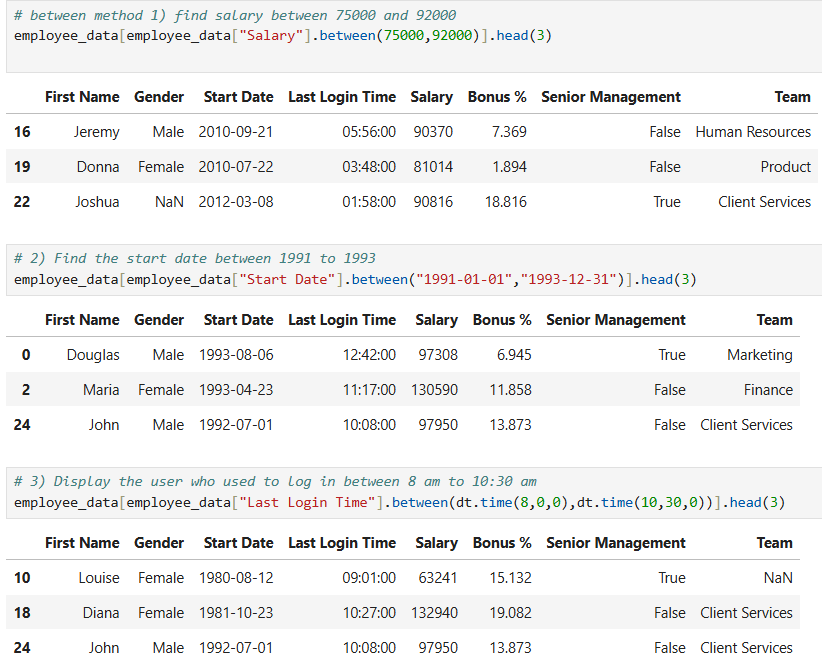
***The isin Method***

* The ***isin Series method*** accepts a collection object like a ***list, tuple, or Series***.
* The ***method returns True*** for a row if its ***value is found*** in the collection.



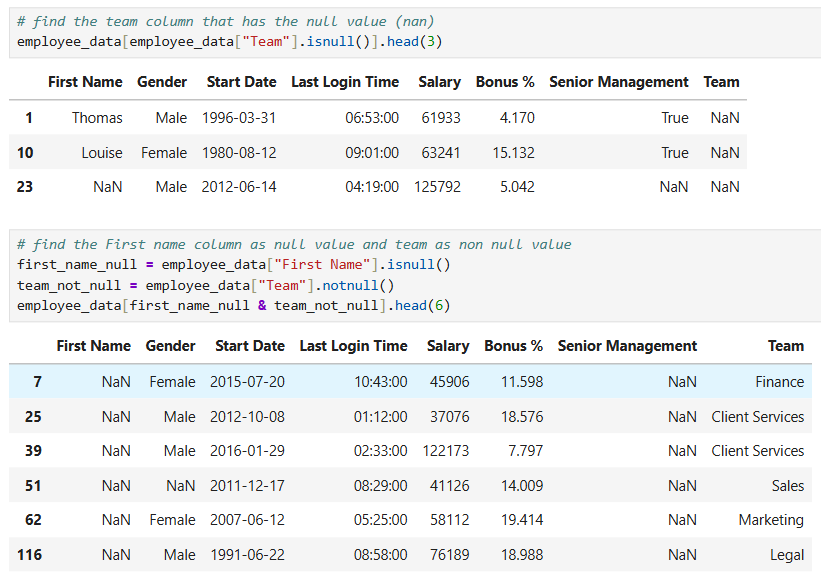
***The between Method***

The ***between method returns True*** if a Series value is found within its range.



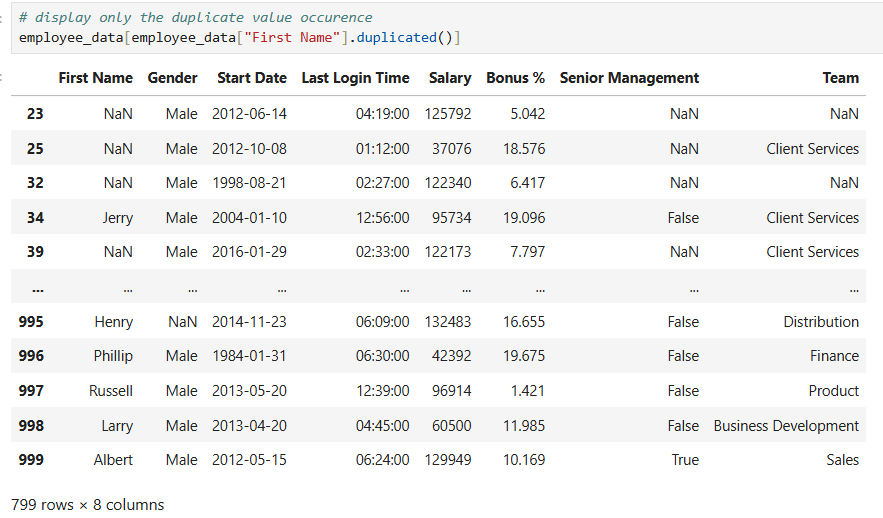
***The isnull and notnull Methods***

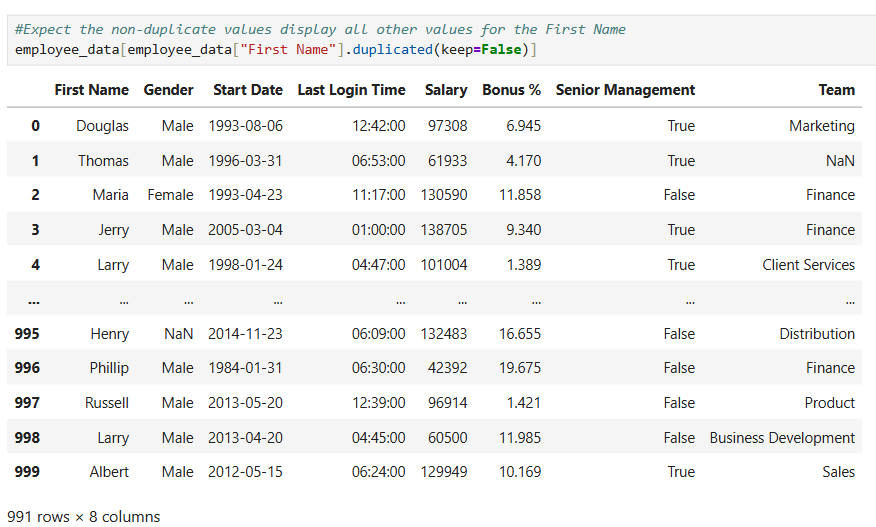
* The ***isnull method returns True for NaN*** values in a Series.
* The ***notnull method returns True*** for present values in a Series.



***The duplicated Method***

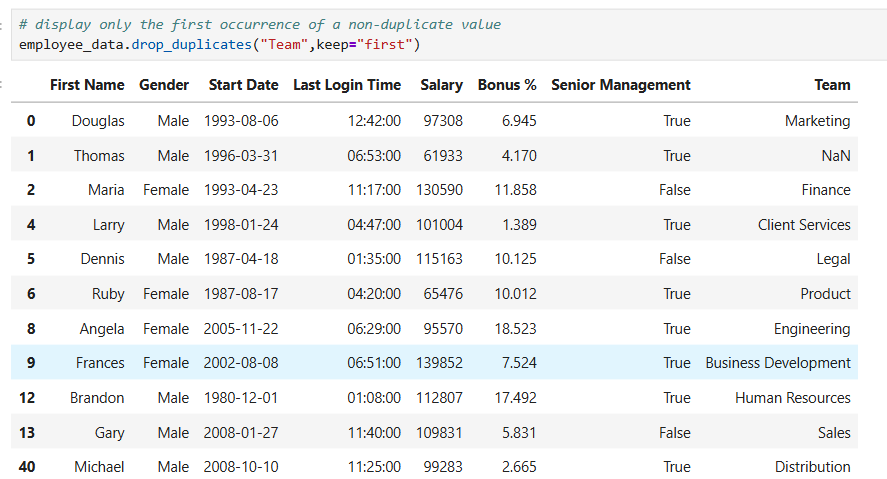
* The ***duplicated method returns True*** if a Series value is a duplicate.
* Pandas will ***mark one occurrence of a repeated value*** as a non-duplicate.
* Use the keep parameter to designate whether the ***first or last occurrence of a repeated value should be considered the "non-duplicate"***
* Pass ***False to the keep parameter*** to mark all occurrences of ***repeated values as duplicates***.
* Use the ***tilde symbol (~) to invert a Series's values***, Trues will become Falses, and Falses will become trues.

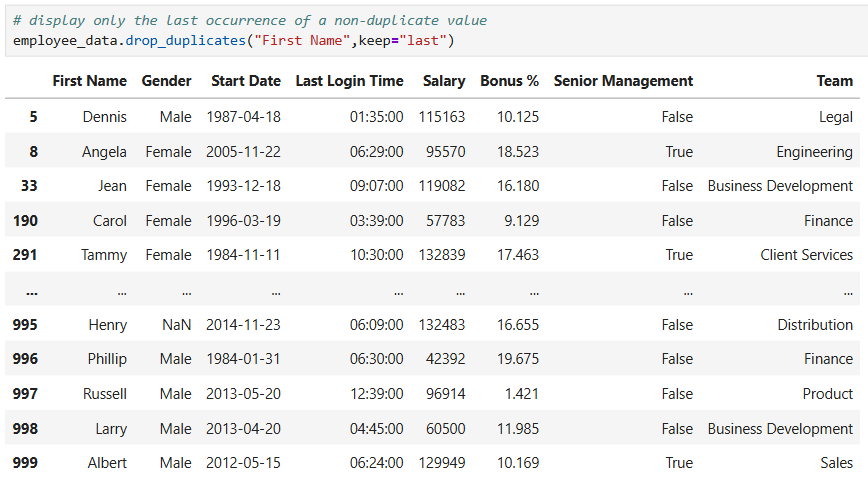


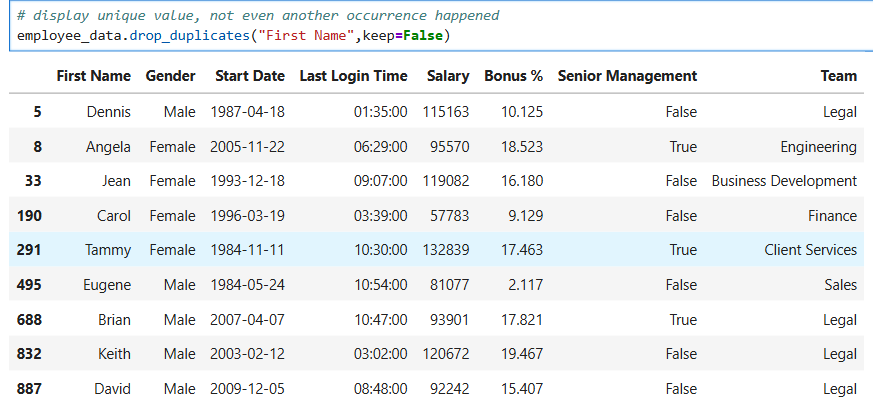


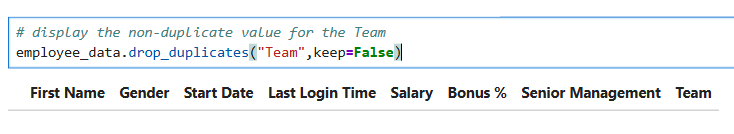
***The drop\_duplicates Method***

* The ***drop\_duplicates*** method ***deletes rows with duplicate values***.
* By default, it will ***remove a row if all of its values*** are shared with another row.
* The subset parameter ***configures the columns to look for duplicate values*** within.
* Pass a list to ***subset parameter to look for duplicates*** across multiple columns.



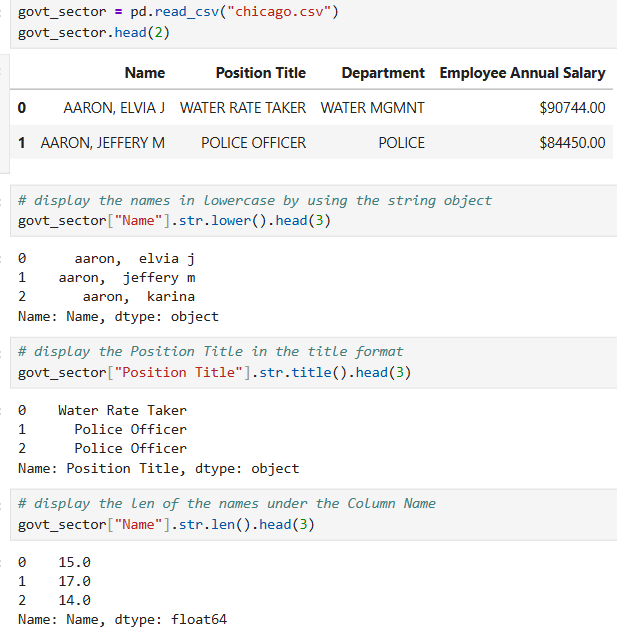


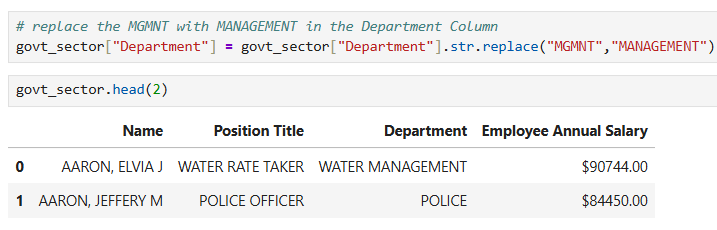


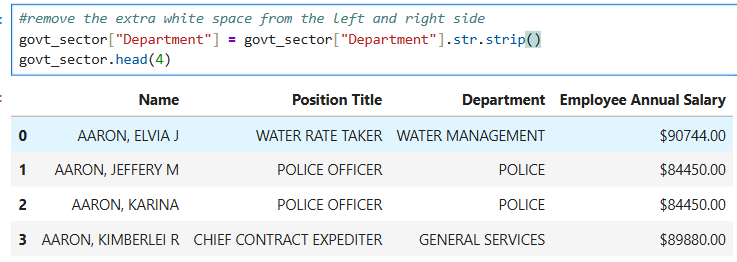


***Common String Methods***

* A Series has a ***special str attribute*** that exposes an object with ***string methods.***
* Access the ***stir attribute***, then invoke the ***string method*** on the nested object.
* Most method names will match their Python method equivalents (upper, lower, title, etc)







***str.strip()*** - this will remove all the extra white spaces from both left and right side of the word

***str.lstrip()*** - this will remove white space from the left side of the word only

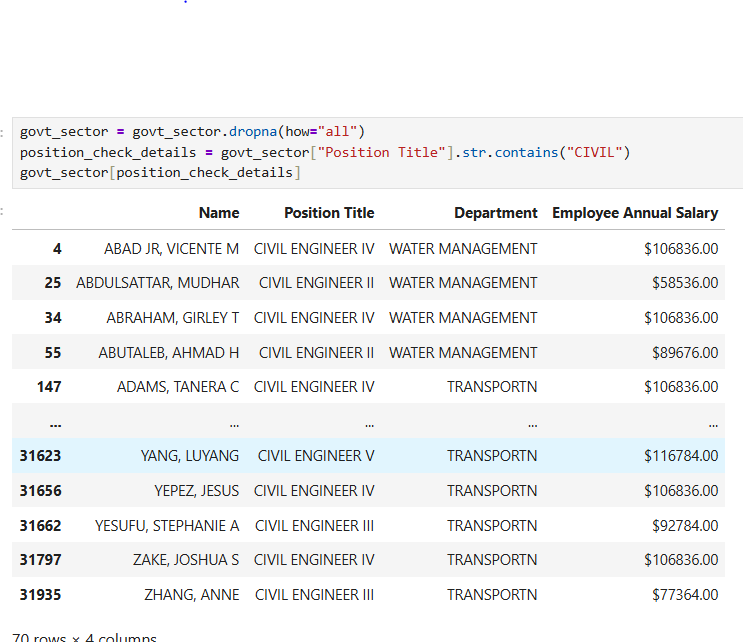
***str.rstrip()*** - this will removed white space from the right side of the word only

***Filtering with String Methods***

The ***str.contains*** method checks whether a substring exists ***anywhere*** in the string.

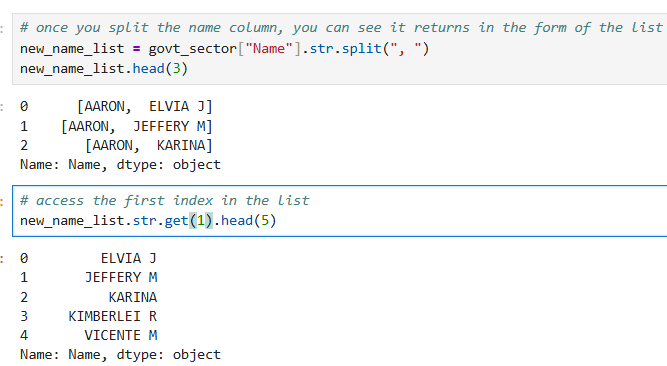
The ***str.startswith*** method checks whether a substring exists at the ***start of the string***.

The ***str.endswith*** method checks whether a substring exists at the ***end of the string***.



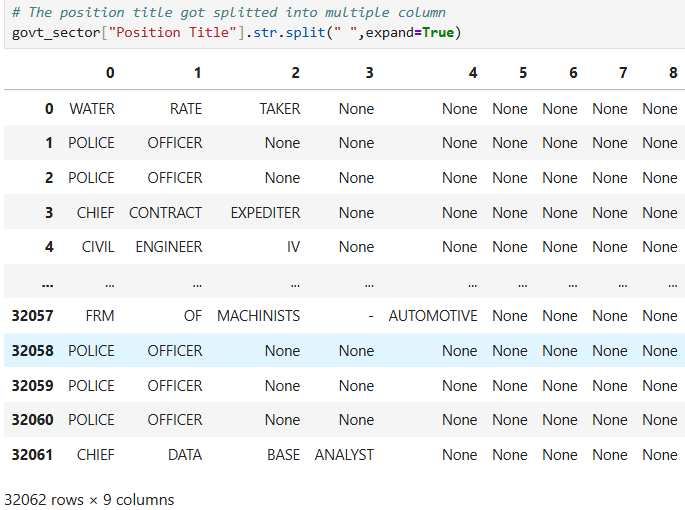
***The split Method***

* The ***str.split method splits a string*** by the occurrence of a delimiter. Pandas returns a ***Series of lists***.
* Use the ***str.get method*** to access a nested list element by its ***index position***.

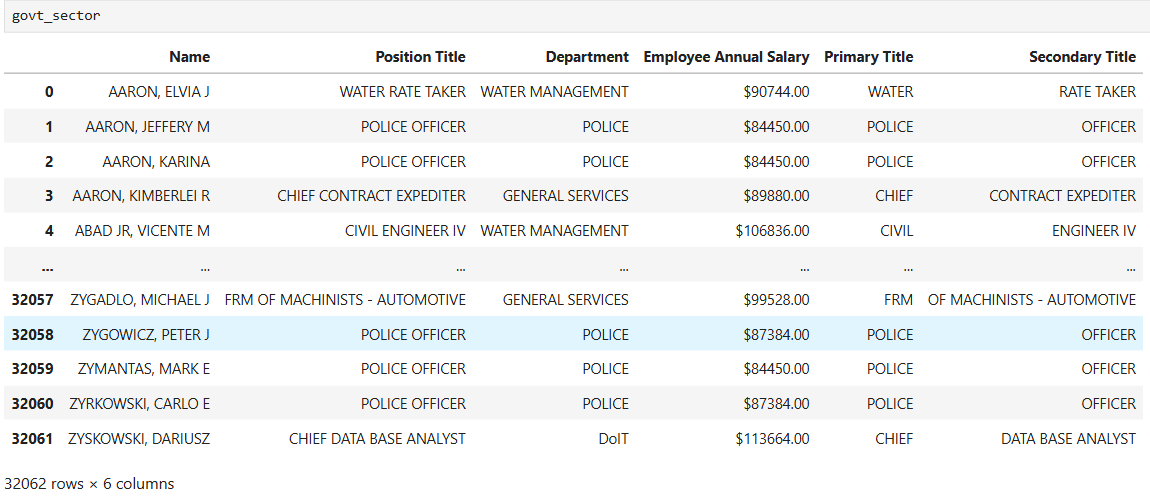


***The expand and n Parameters of the split Method***

* The ***expand parameter returns a DataFrame*** instead of a Series of lists.
* The ***n parameter limits*** the number of splits.

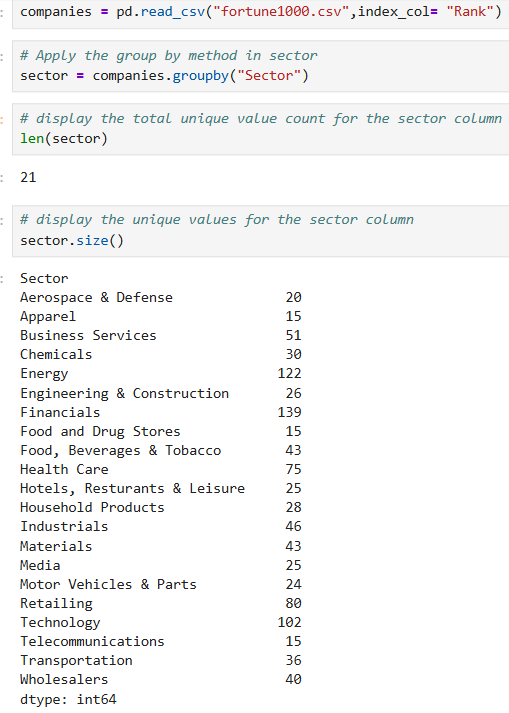


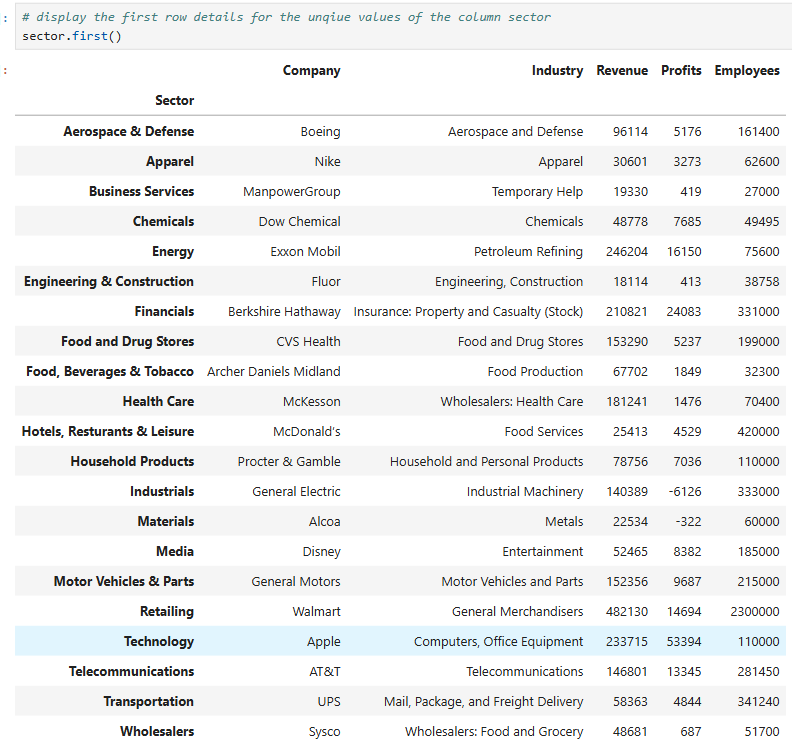


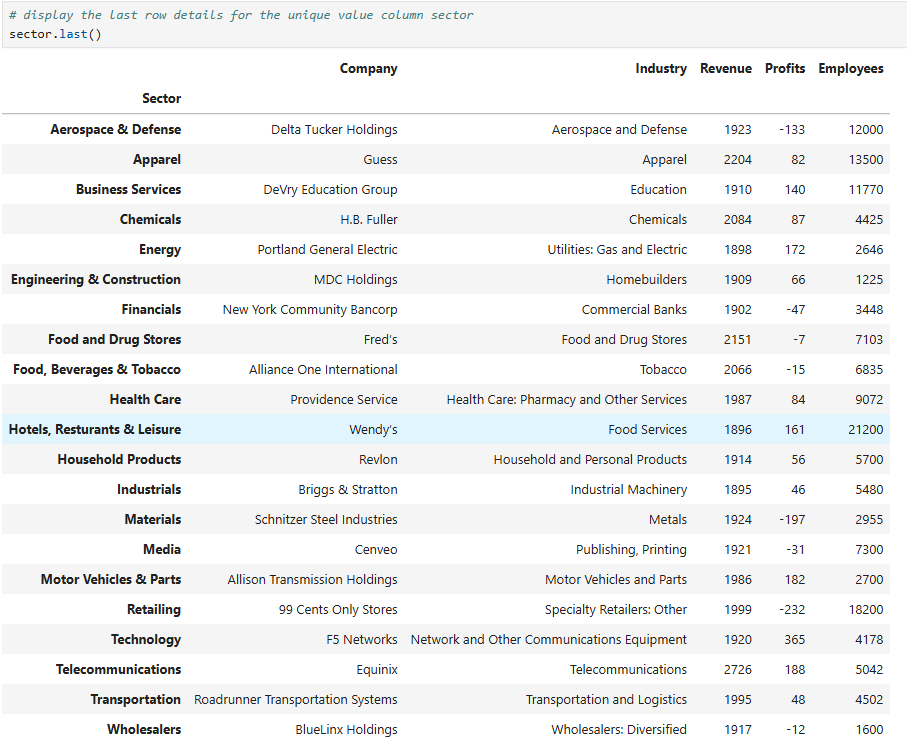


***The groupby Method***

* Grouping is a way to ***organize/categorize/group*** the data based on a column's values.
* The ***groupby method returns a DataFrameGroupBy object***. It resembles a ***group/collection of DataFrames*** in a ***dictionary-like*** structure.
* The ***DataFrameGroupBy*** object can perform ***aggregate operations on each group within it***.

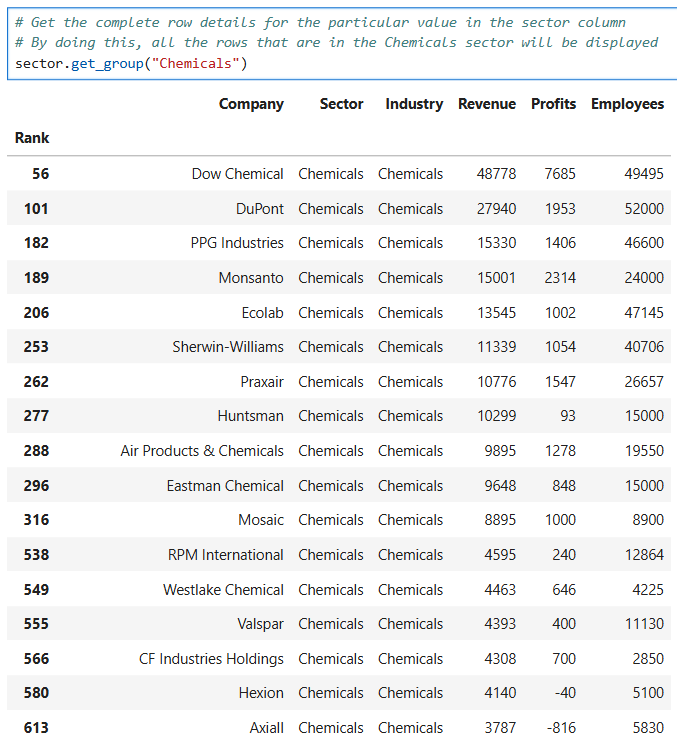






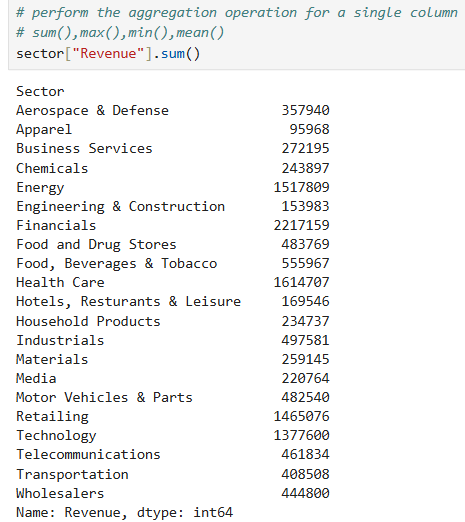
***Retrieve a Group with the get\_group Method***

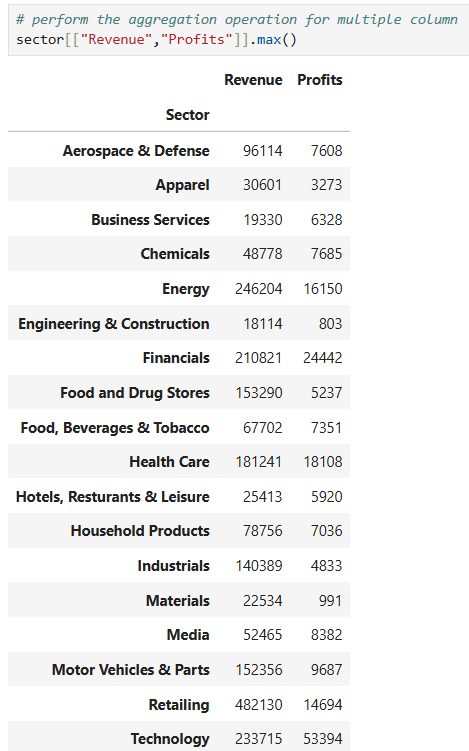
* The ***get\_group method on the DataFrameGroupBy*** object retrieves a nested DataFrame belonging to a specific group/category.



***Methods on the GroupBy Object***

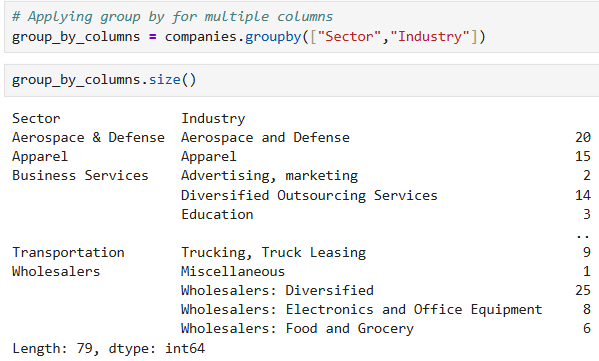
* Use ***square brackets on the DataFrameGroupBy object to "extract"*** a column from the original DataFrame.
* The resulting Series ***GroupBy object will have aggregation methods*** available on it.
* Pandas will perform the ***calculation on every group*** within the collection.
* For example, the ***sum method will sum together*** the Revenues for ***every row by group/category.***

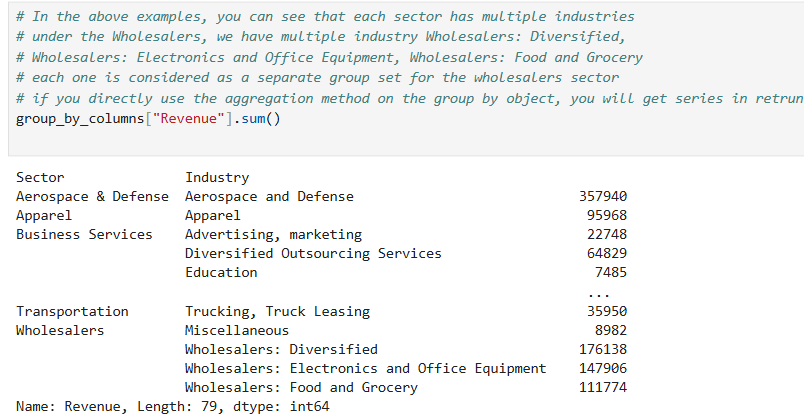




***Grouping by Multiple Columns***

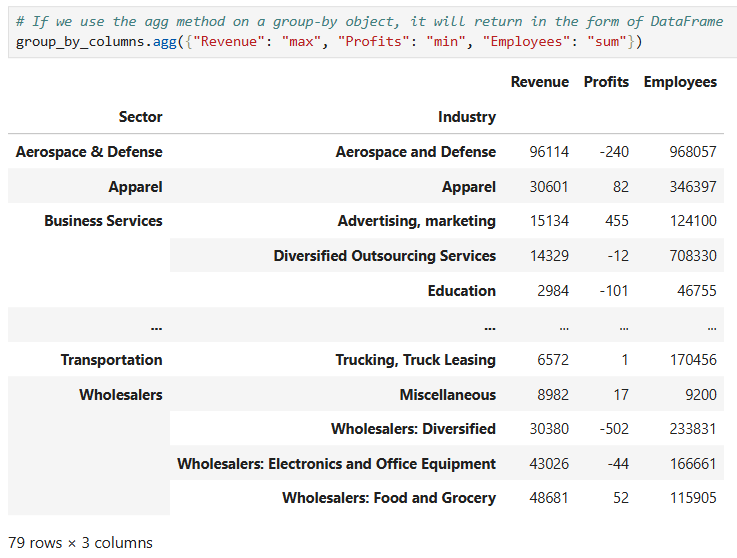
* Pass a ***list of columns to the groupby method*** to group ***by pairings of values*** across columns.
* ***Target a column to retrieve the SeriesGroupBy objec***t, then perform an aggregation with a method.
* Pandas will return a ***MultiIndex Series where the levels*** will be the original groups.





***The agg Method***

* The ***agg method*** applies different ***aggregation methods on different columns***.
* Invoke the ***agg method directly*** on the ***DataFrameGroupBy object***.
* ***Pass the method a dictionary*** where the ***keys are the columns and the values*** are the aggregation operations.



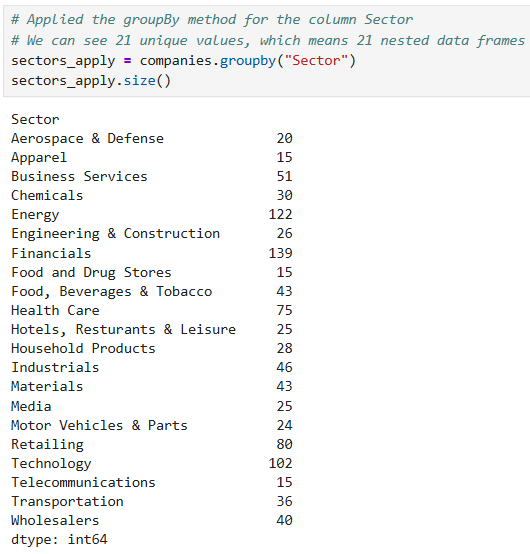
***Iterating through Groups***

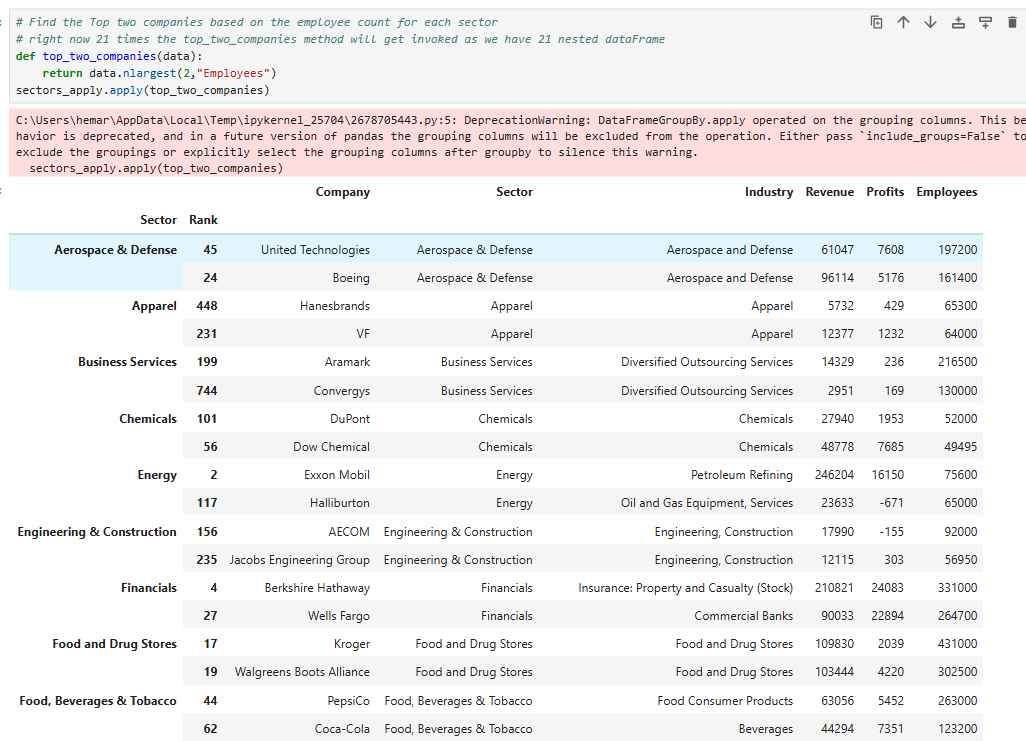
The ***DataFrameGroupBy object*** supports the ***apply method*** (just like a Series and a DataFrame do).

The ***apply method invokes*** a function on every ***nested DataFrame*** in the ***DataFrameGroupBy*** object.

It captures the ***return values of the functions and collects*** them in a new DataFrame (the return value).

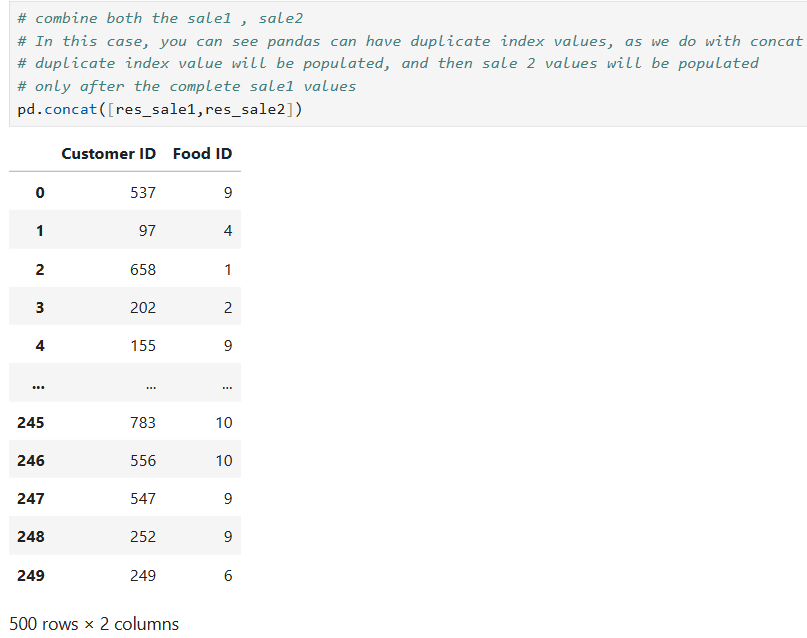
***#Note:*** Once you applied the groupBy it will have nested dataFrame

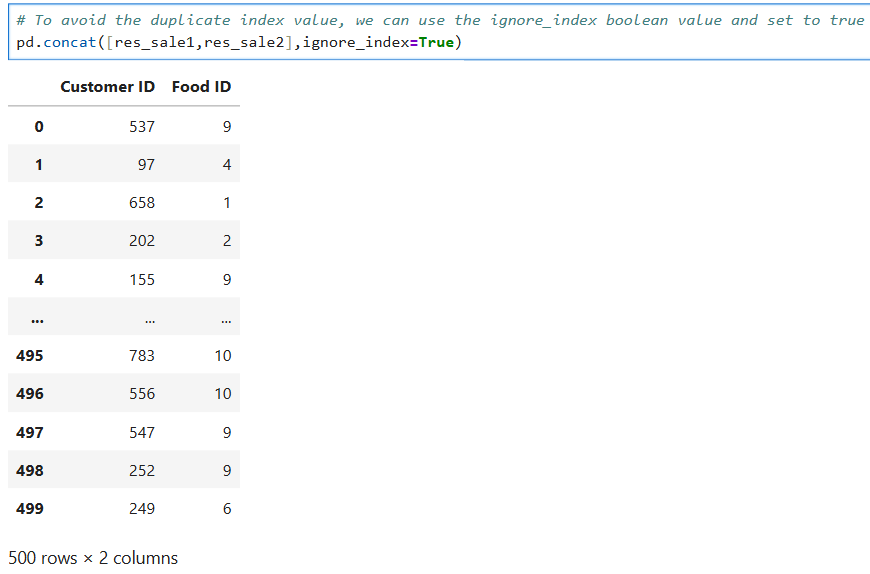


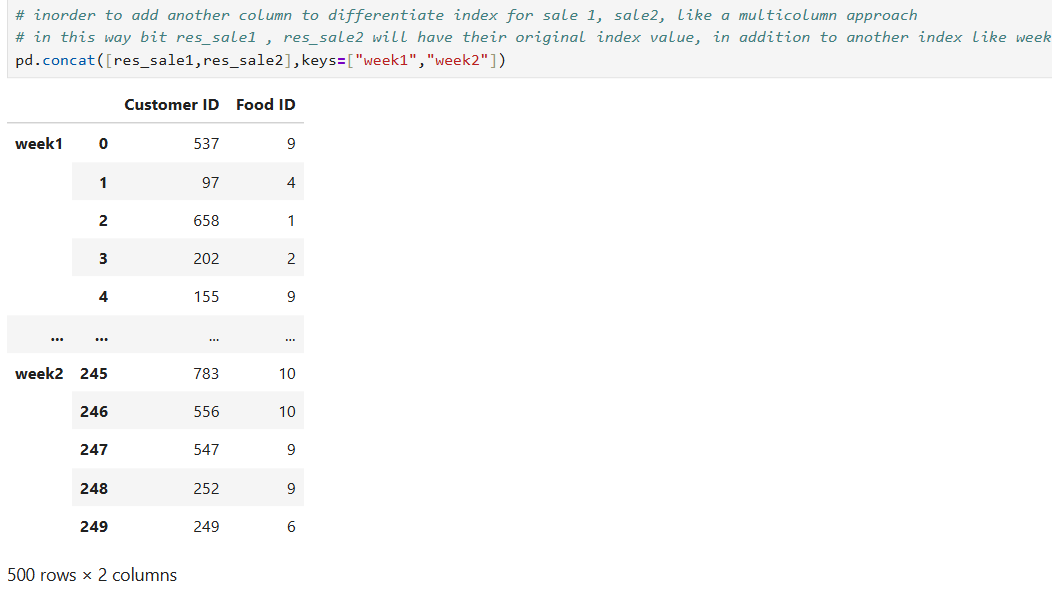


***The pd.concat Function I***

* The ***concat function concatenates one DataFrame to the end of anothe***r.
* The ***original index labels*** will be kept by ***default***. Set ***ignore\_index to True*** to generate a ***new index***.
* The keys parameter create a ***MultiIndex using the specified keys/labels***.

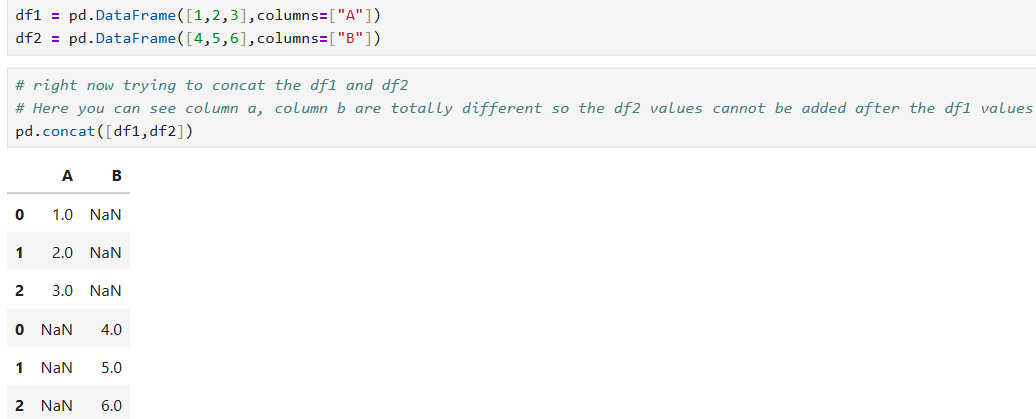


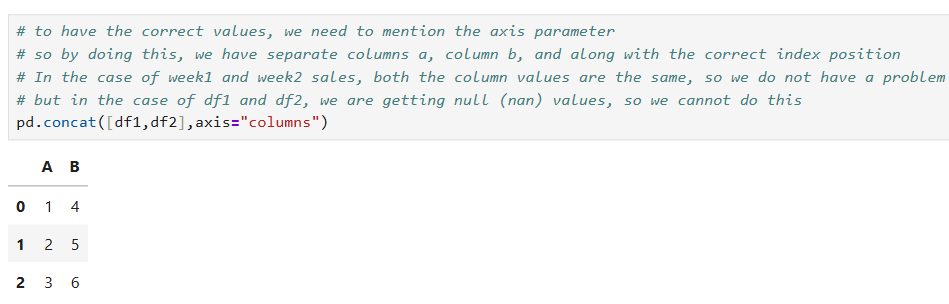




***The pd.concat Function II***

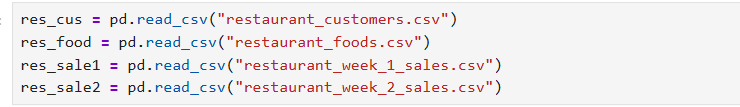
* Pandas will ***concatenate the DataFrames*** along the row/index axis.
* Pandas will include ***all columns that exist*** in either DataFrame. If there are no matching values, ***pandas will use NaN values***.
* We can pass the ***axis parameter an argument of "columns"*** to concatenate on the column axis.

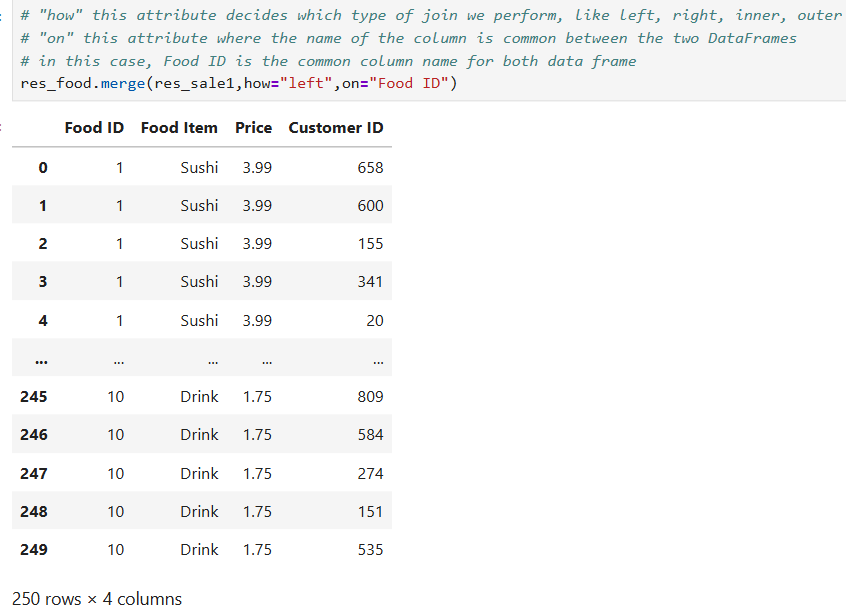




***Left Joins***

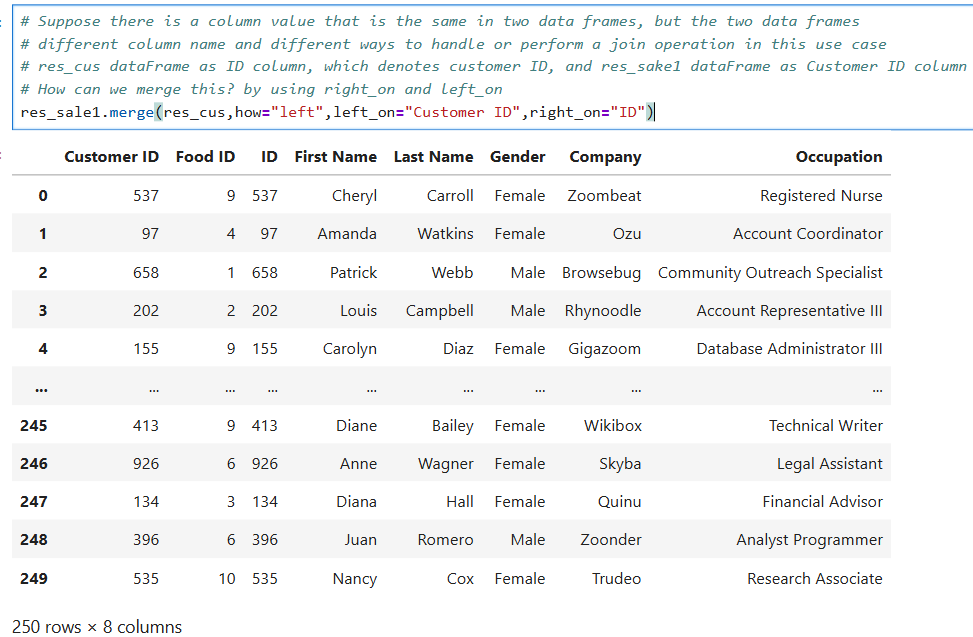
* The ***merge method*** joins ***two DataFrames together*** based on shared values in a column or an index.
* A ***left join merges*** one DataFrame into ***another based on values*** in the first one.
* The ***"left" DataFrame*** is the one we invoke the merge method on.
* If the ***left DataFrame's value is not found in the right DataFrame***, the row will hold ***NaN values***.

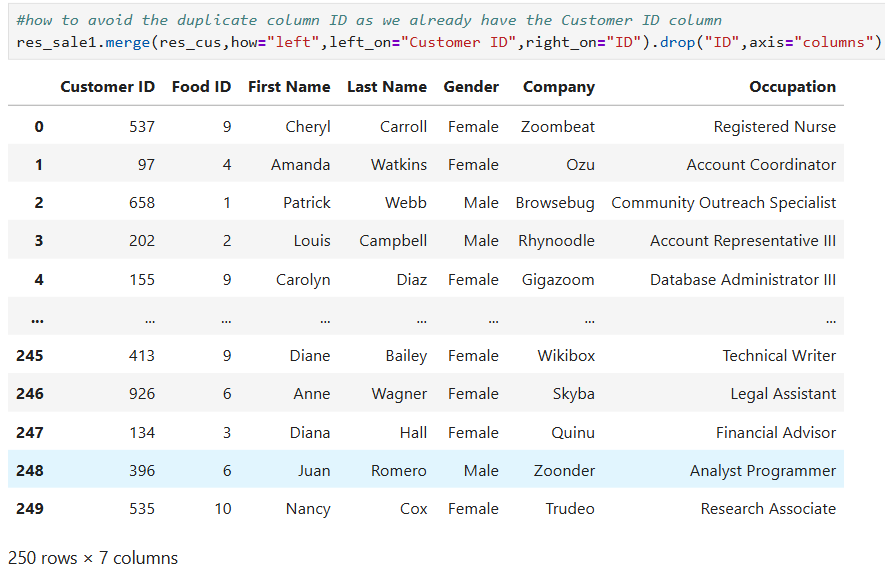




***The left\_on and right\_on Parameters***

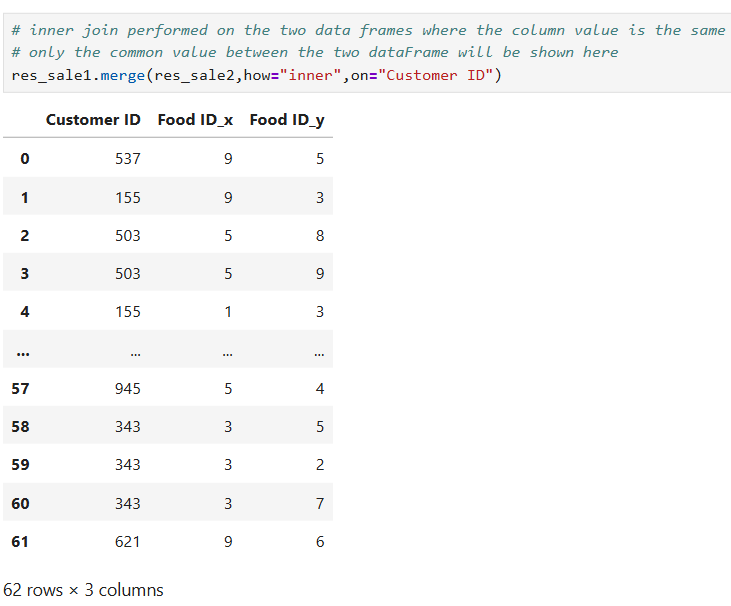
The ***left\_on and right\_on*** parameters designate the ***column names*** from each DataFrame to use in the merge.

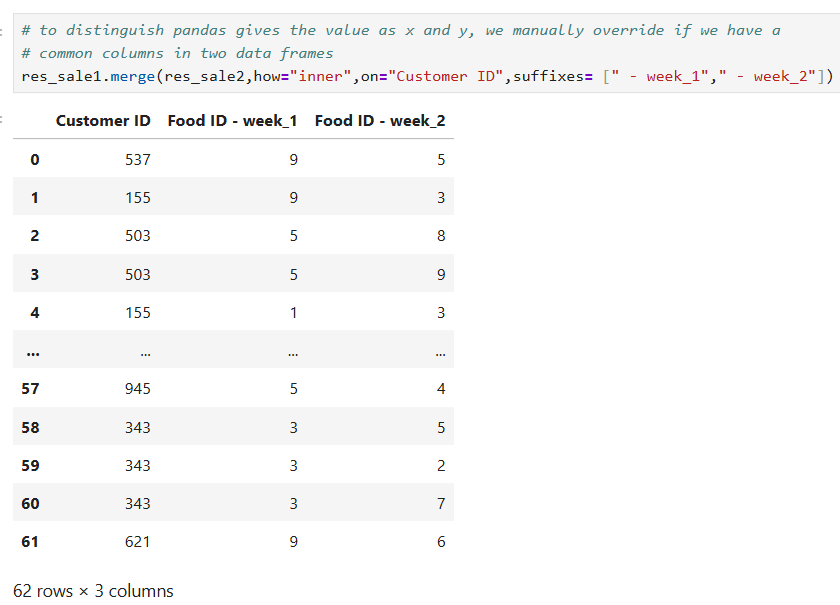




***Inner Joins I***

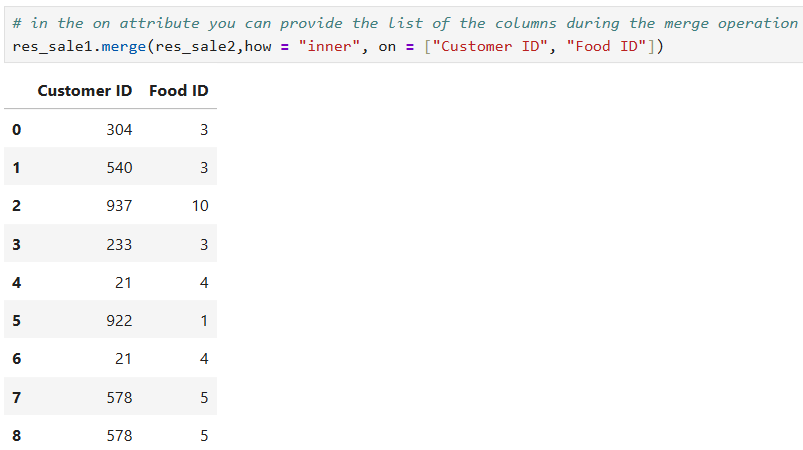
* Inner joins merge ***two tables based on shared/common values*** in columns.
* If only ***one DataFrame has a value***, pandas will exclude it from the final results set.
* If the ***same ID occurs multiple times***, pandas will store each possible combination of the values.
* The ***design of the join ensures that the results will be the same*** no matter what DataFrame the merge method is invoked upon.





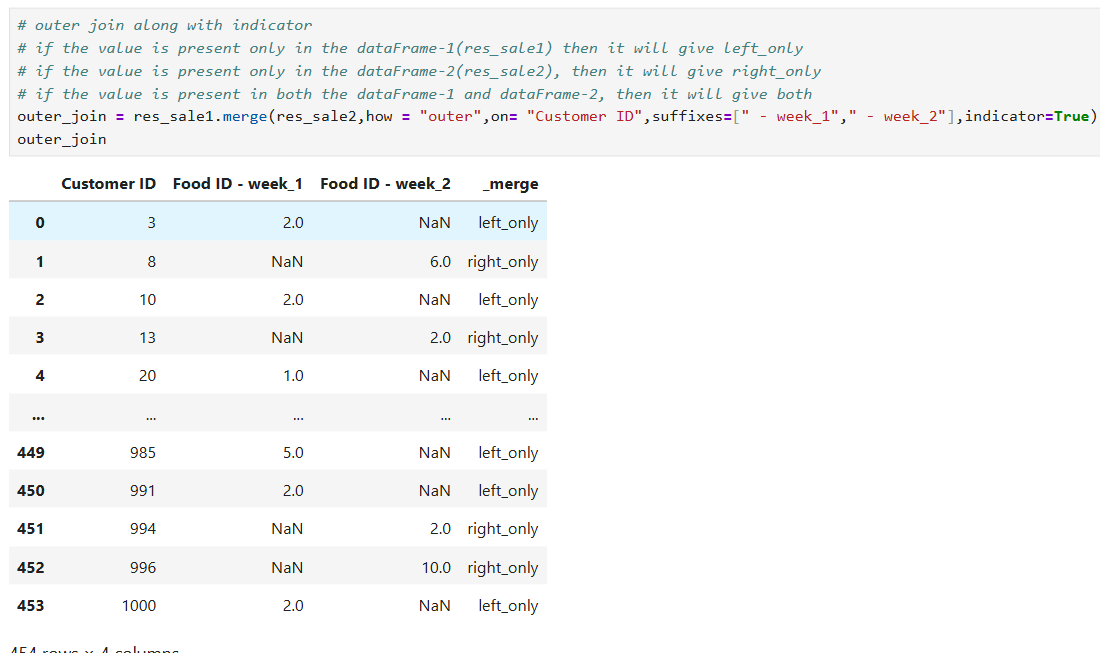
***# Note :***

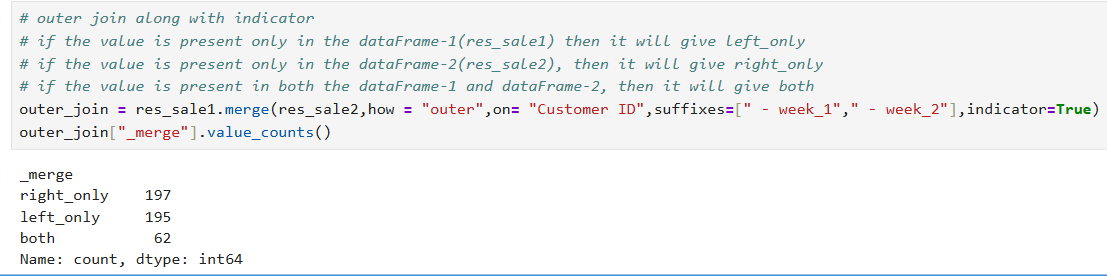
Remember that if dataFrame-1 as one value which is occurred more than thrice and if dataFrame-2 as one value occurred once still pandas during the inner join pandas will still shows the three rows for that value with all the possible combination.



***Full/Outer Join***

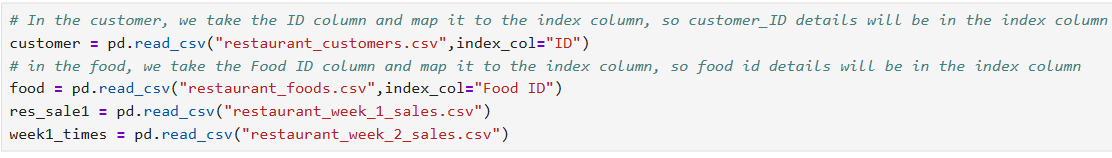
* A ***full/outer joins values*** that are found in either ***DataFrame or both DataFrames***.
* Pandas does not mind if a value exists in one DataFrame but not the other.
* If a value ***does not exist in one DataFrame***, it will have a ***NaN.***

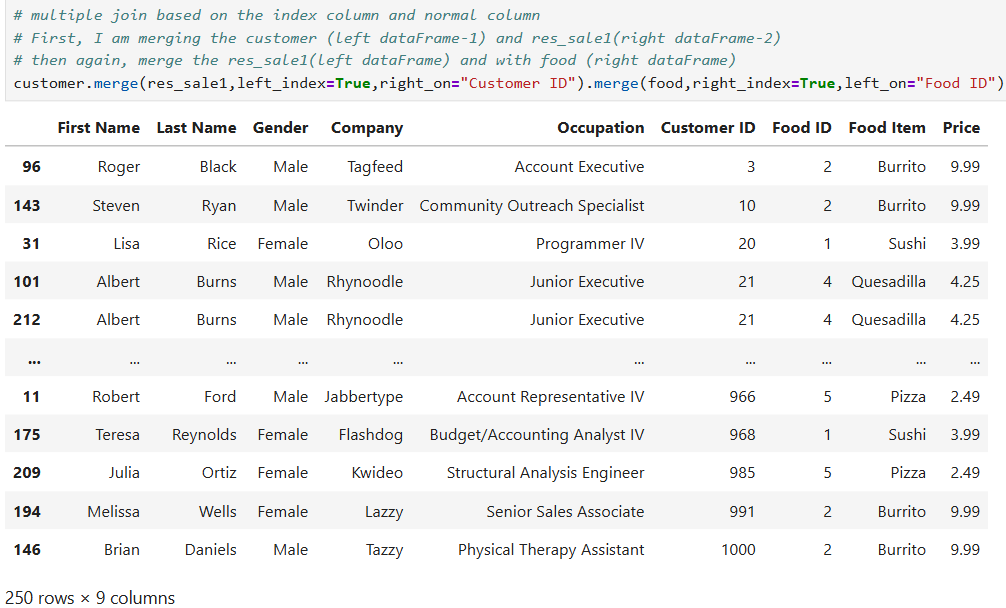




***Merging by Indexes with the left\_index and right\_index Parameters***

* Use the ***on parameter if the column(s) to be matched on have the same names*** in both DataFrames.
* Use the ***left\_on and right\_on parameters if the column(s)*** to be matched d on have ***different names*** in the two DataFrames.
* Use the ***left\_index or right\_index parameters (set to True)*** to if the values to be matched on ***are found in the index of a DataFrame***.





***The join Method***

The ***join method is a shortcut for concatenating two DataFrames*** when merging by index labels.

