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.

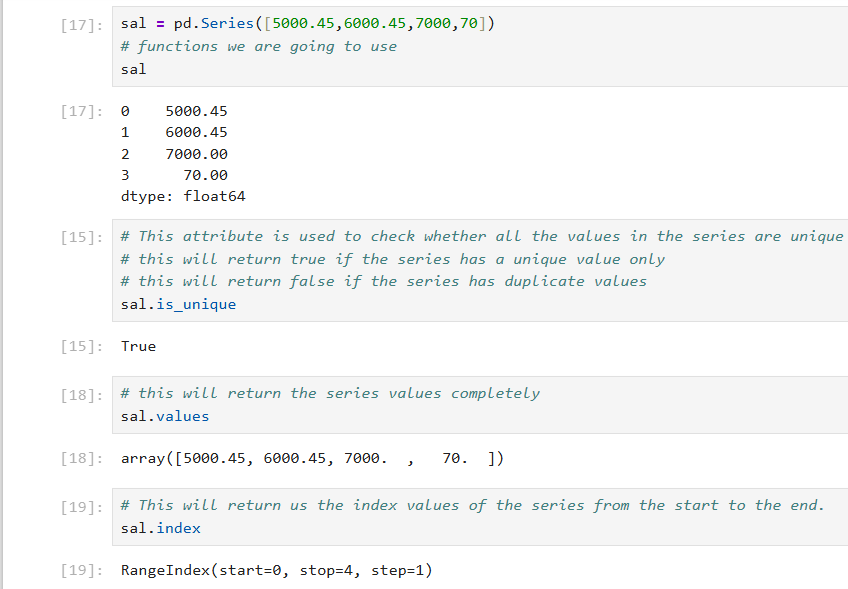
***Methods and Attributes:***

***In pandas series:***

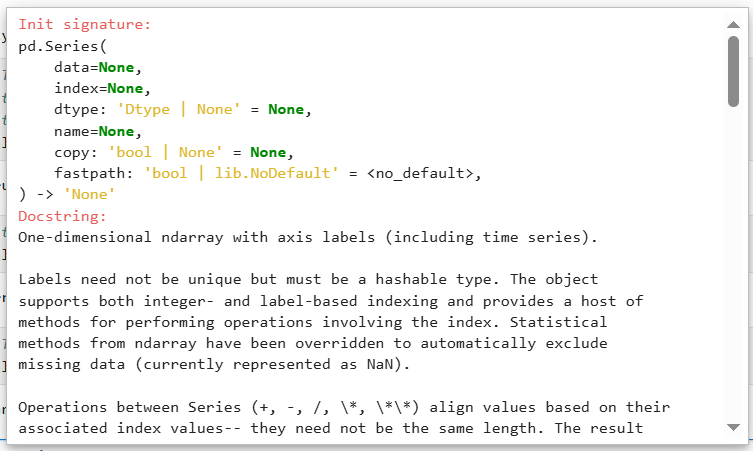
--> ***attributes*** tells about the object and does not required any paraenthesis

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





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



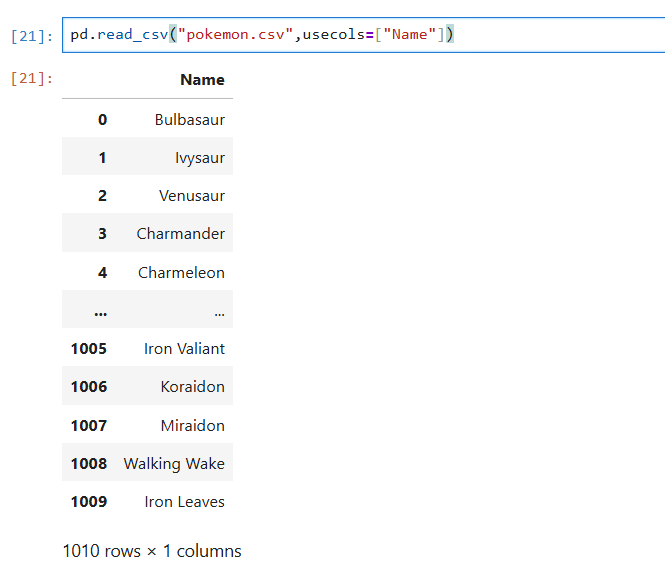
***data*** which will take the input source , ***index*** which will take custom index value which we map for the data if ***do not give index*** by default ***from 0 index position*** will get assigned to each values in the data.

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

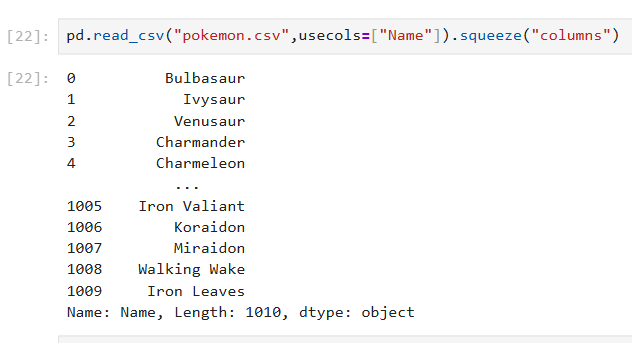
A ***CSV is a plain text file*** that uses line breaks to ***separate rows and commas to separate row values***.

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***

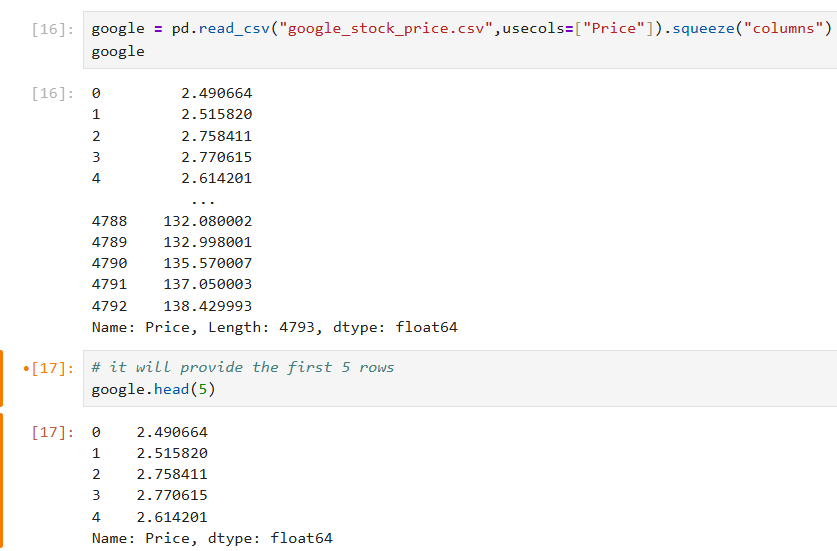
Two dimesional View in DataFrames:



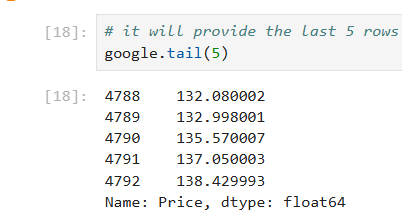
After converting the dataFrames to Series by using the squeeze method we can view the one dimesional view:



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***.

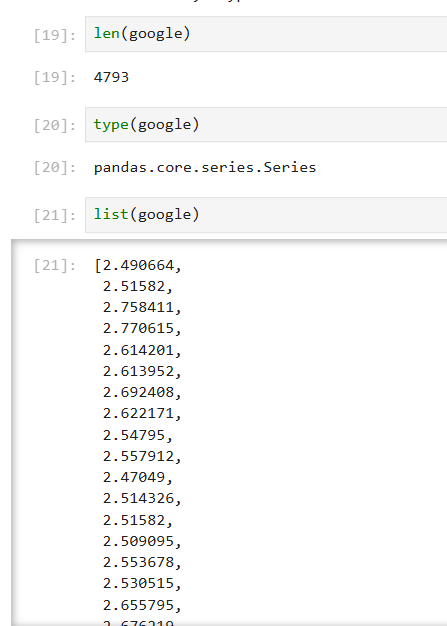
The ***list function*** converts the ***Series to a list***.

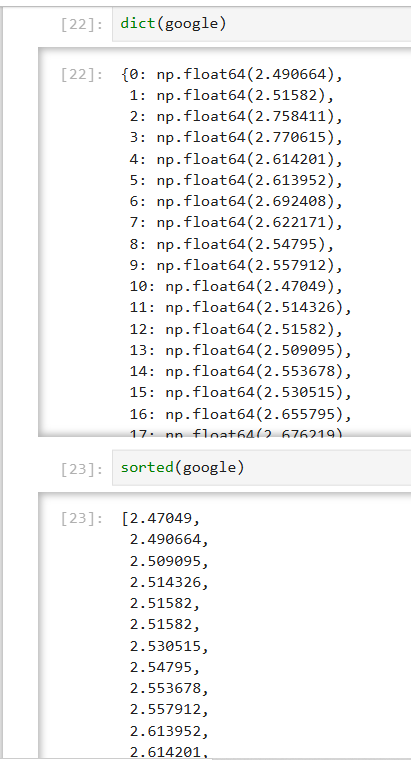
The ***dict function*** converts the ***Series to a dictionary***.

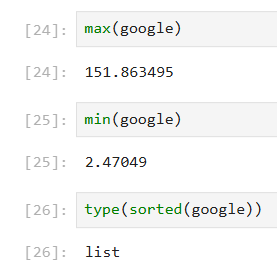
The ***sorted function*** converts the ***Series to a sorted list***.(ascending to descending order)

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***.

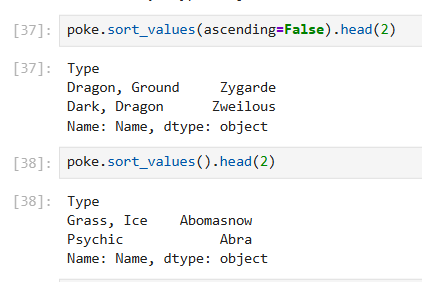
***The sort\_index Method***

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

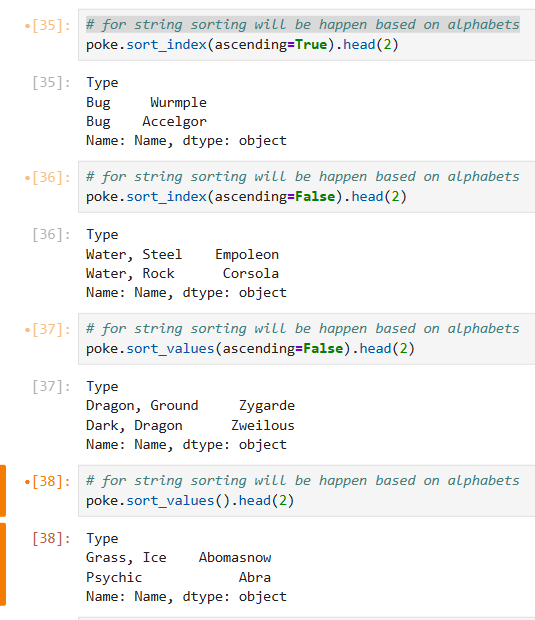
The ***sort\_index method*** also accepts an ascending parameter to ***set sort order***.

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.



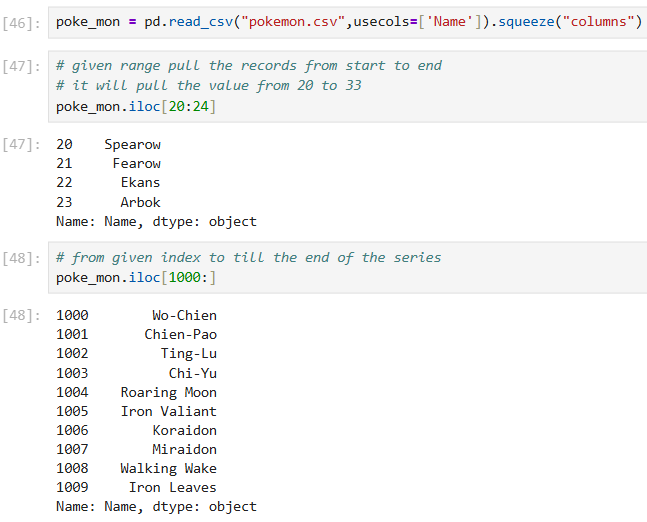
***Extract Series Value by Index Position***

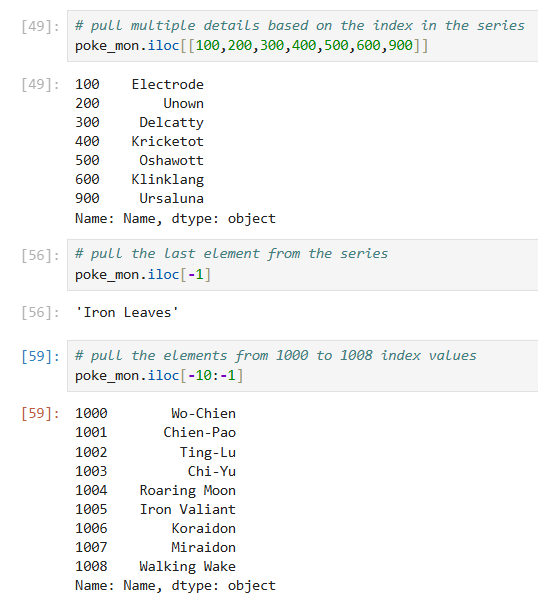
Use the ***iloc accessor*** to extract a ***Series value*** by its ***index position***.

***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[]***



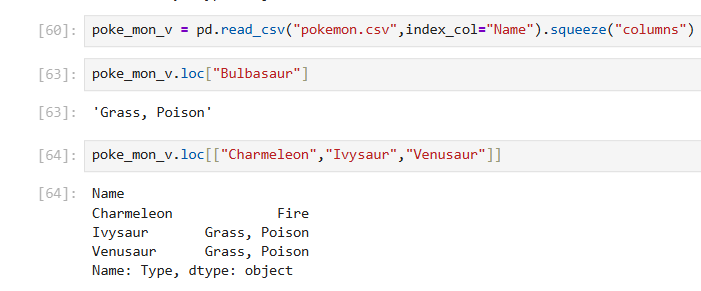


***Extract Series Value by Index Label***

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

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

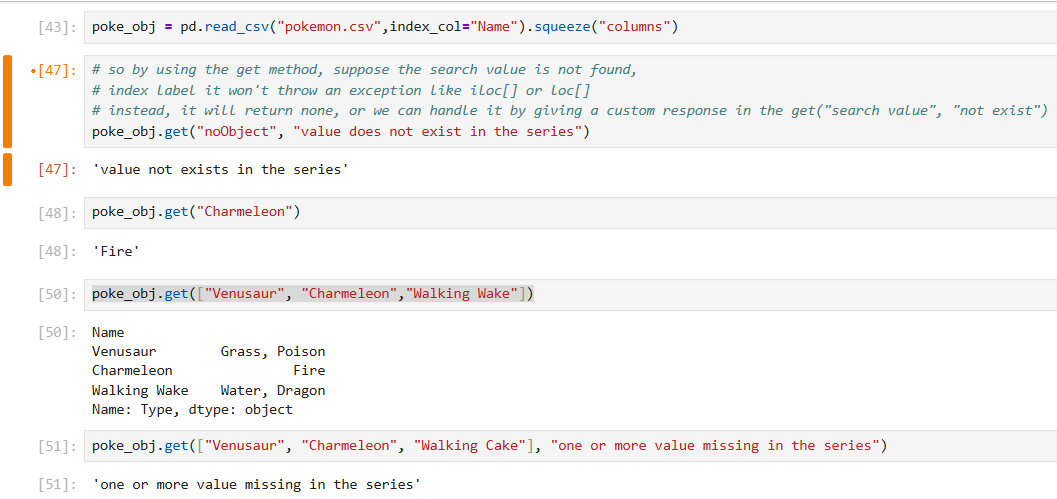
If ***one index label/position*** in the ***list does not exist***, Pandas will ***raise an error***.



***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:***

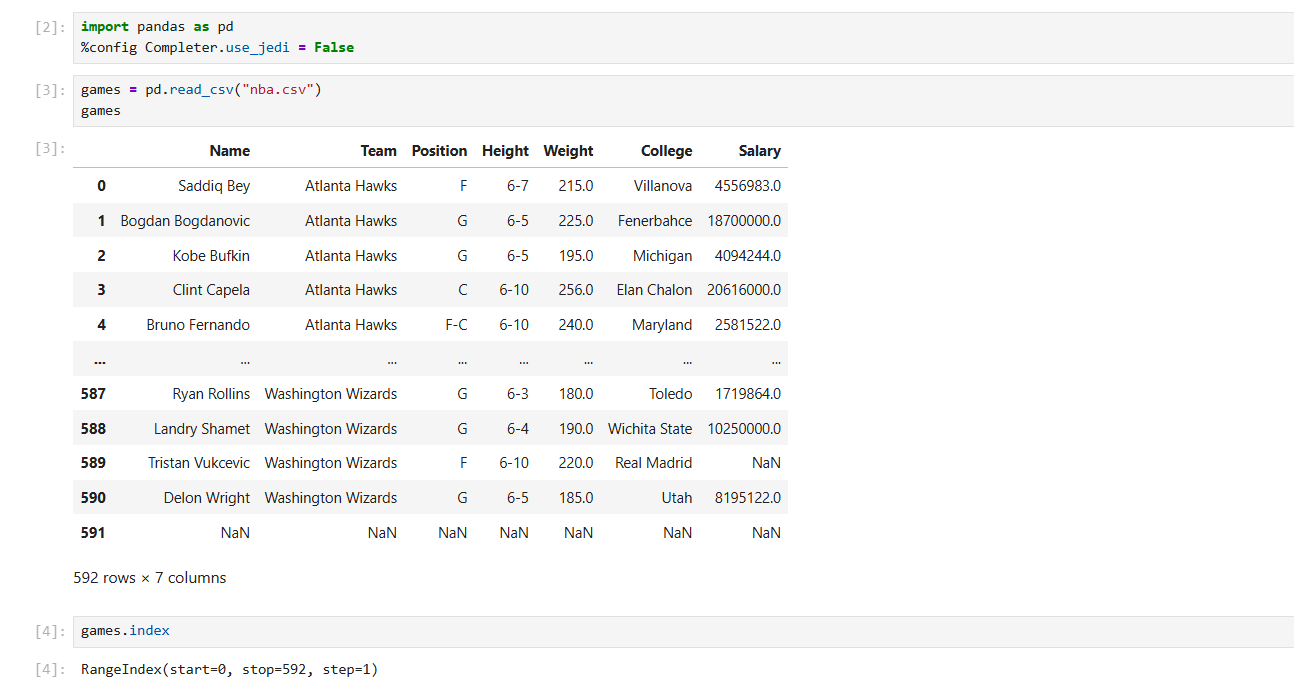
Use the ***loc/iloc accessor to target an index label/position***, then use an equal sign to provide a new value

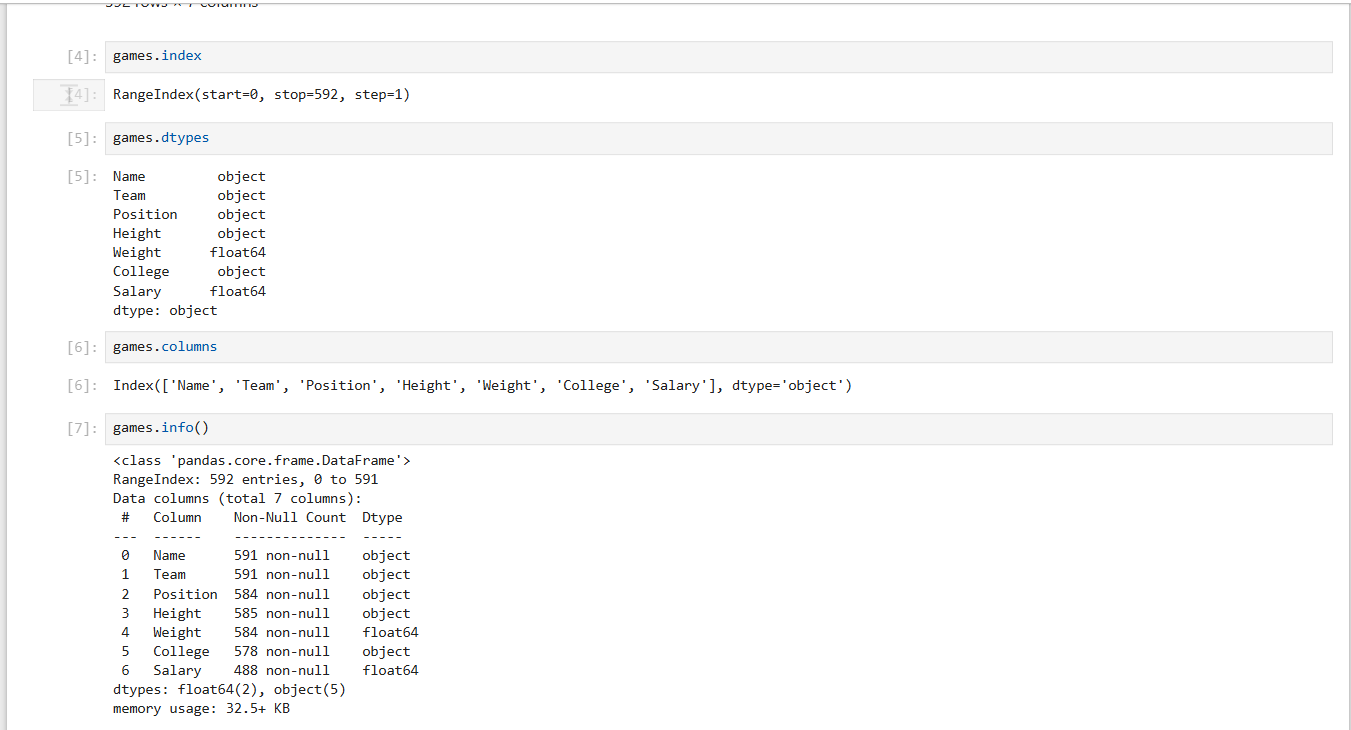




***Methods and Attributes between Series and DataFrames***

* 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.





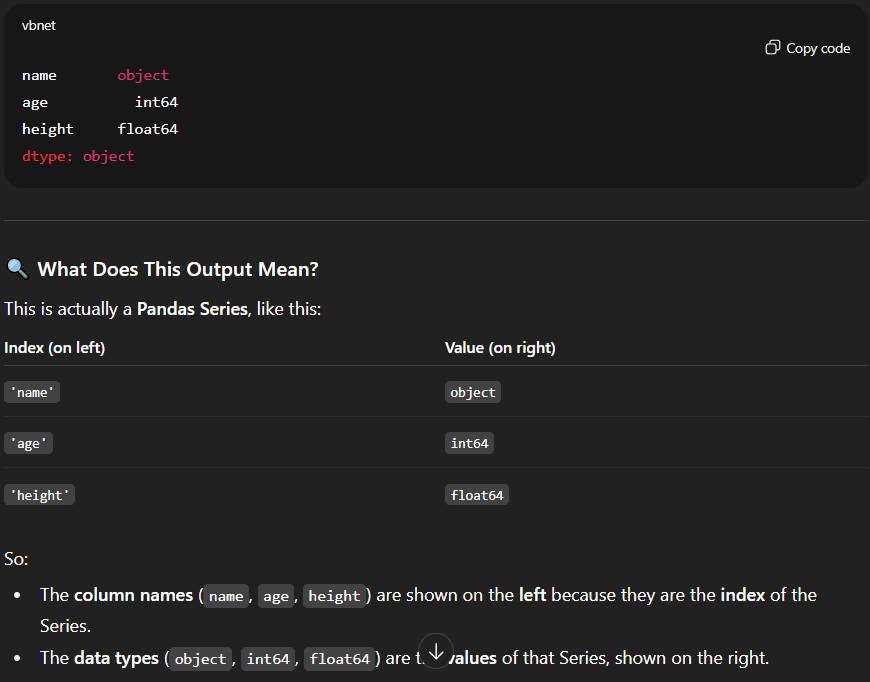
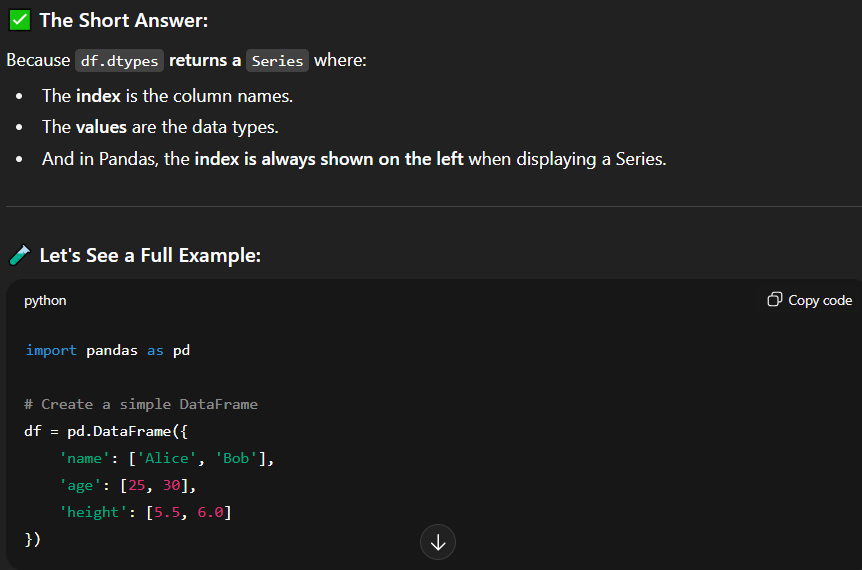
***Differences between Shared Methods***

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

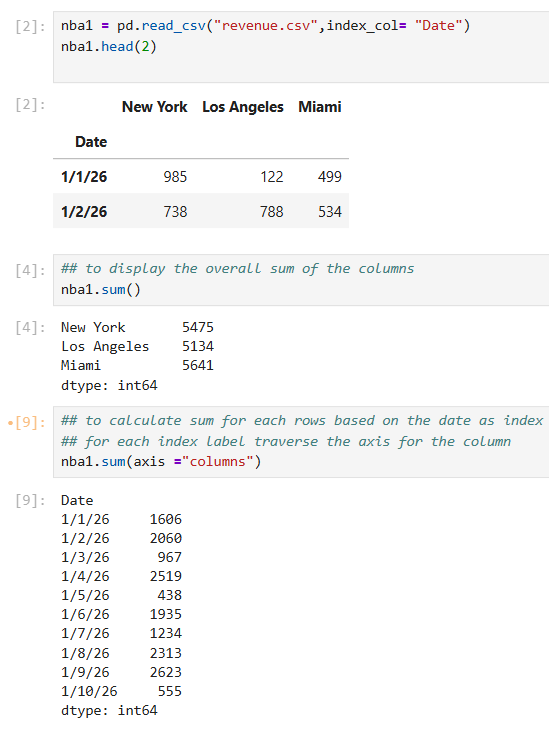
On a DataFrame, the ***sum method*** defaults to adding the values by ***traversing the index (row values)***.

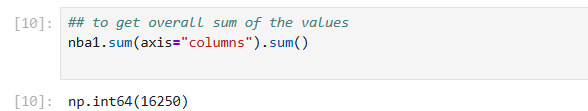
The ***axis parameter customizes*** the direction that we add across. Pass ***"columns" or 1 to add "across" the columns***.

***#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***.

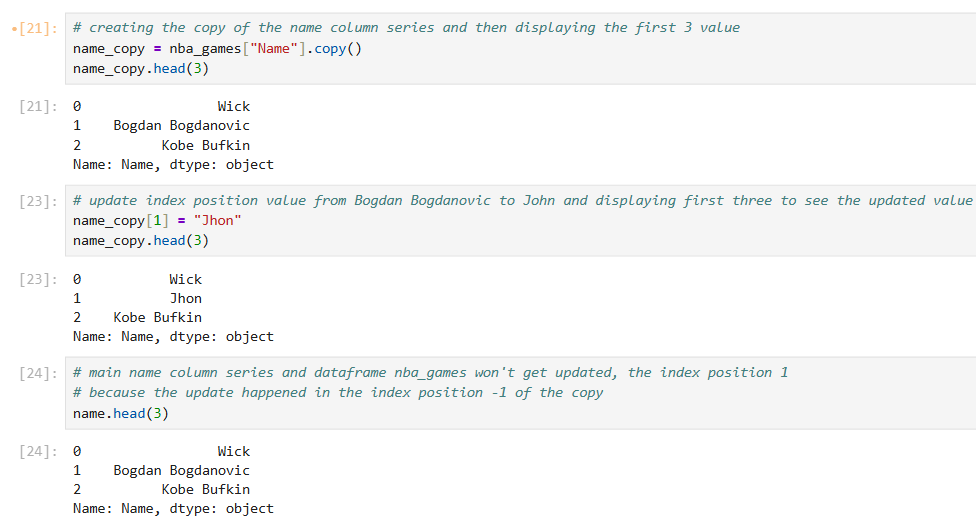
We can also use ***square bracket syntax (df ["column name"])*** which will work for any column name.

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.

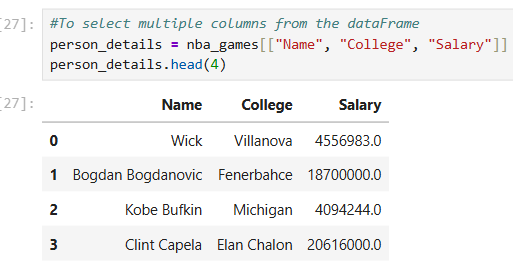




***Select Multiple Columns from a DataFrame***

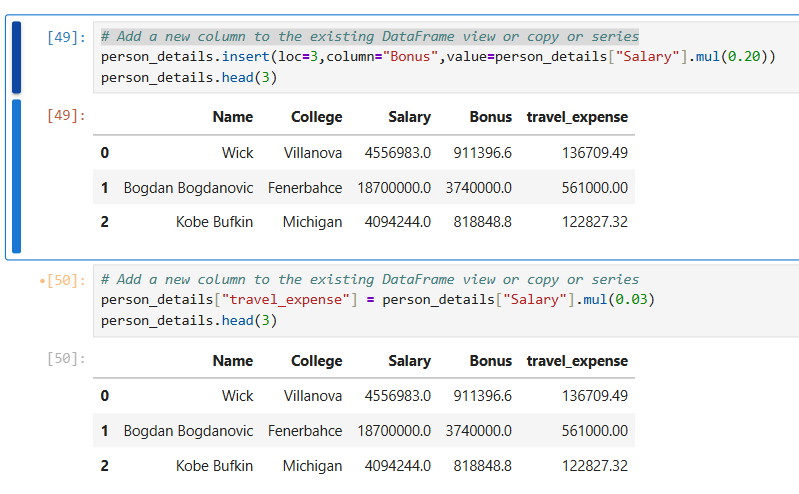
Use square brackets with a list of names to extract multiple DataFrame columns.

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



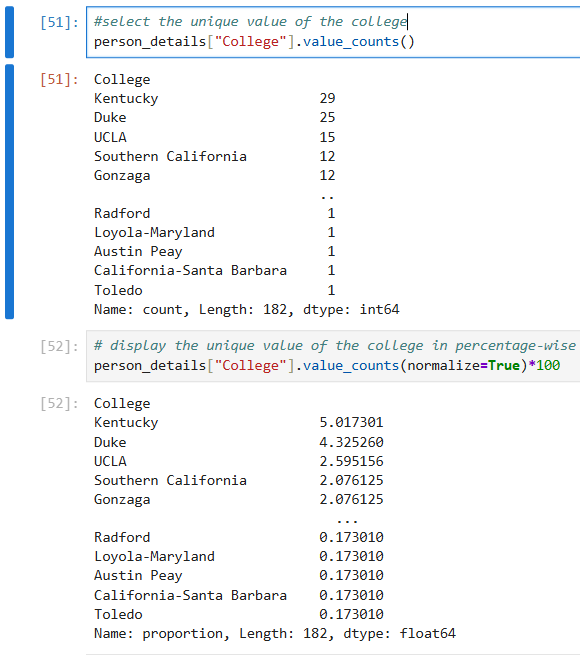
***Add New Column to DataFrame***

* 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.



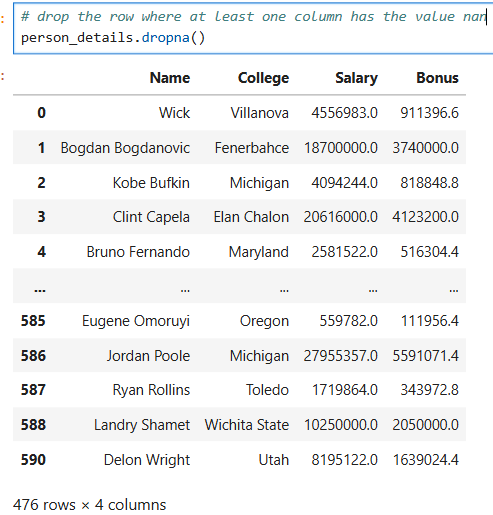
***A Review of the value\_counts Method***

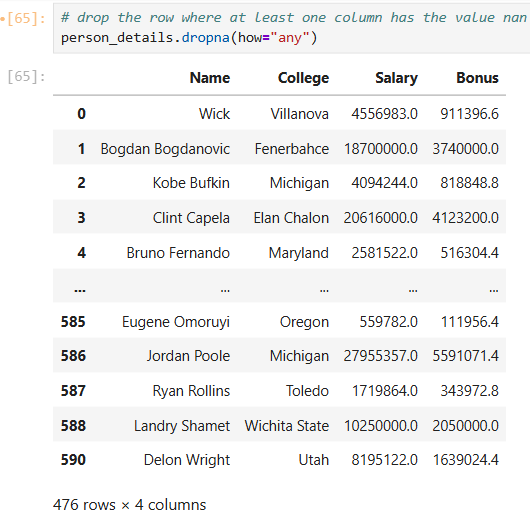
The ***value\_counts method*** counts the number of times that each unique value occurs in a Series.



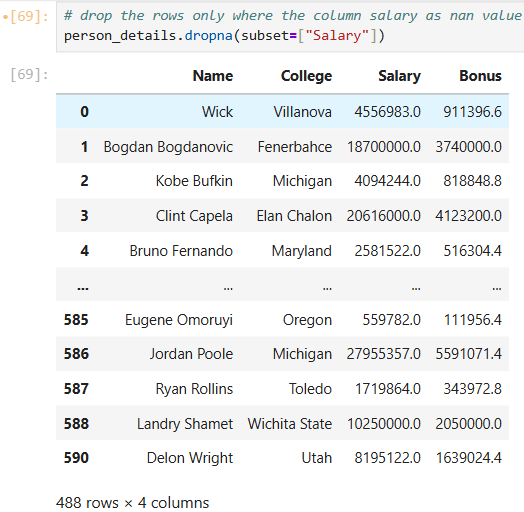
***Drop Rows with Missing Values***

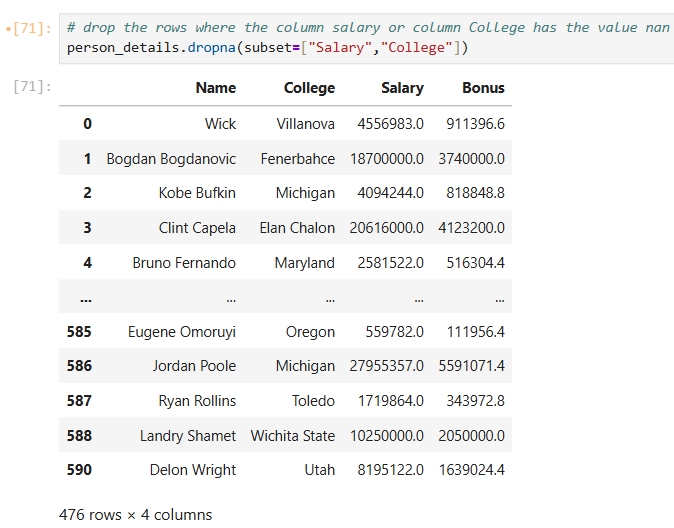
* Pandas uses a ***NaN designation*** for cells that have a missing value.
* The ***dropna method deletes rows with missing values***. Its default behavior is to ***remove a row*** if it has any missing values.
* Pass the how parameter an ***argument of "all" to delete rows*** where all the values are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





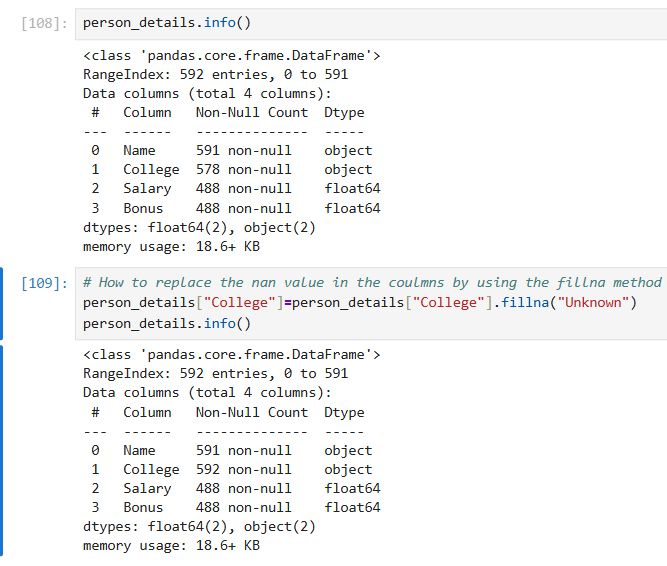






***Fill in Missing Values with the fillna Method***

* 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.

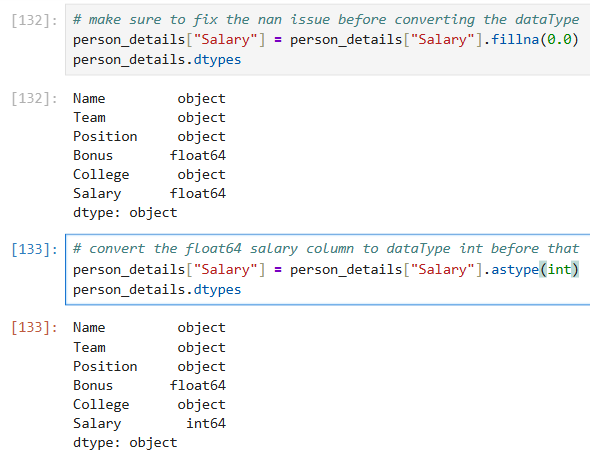


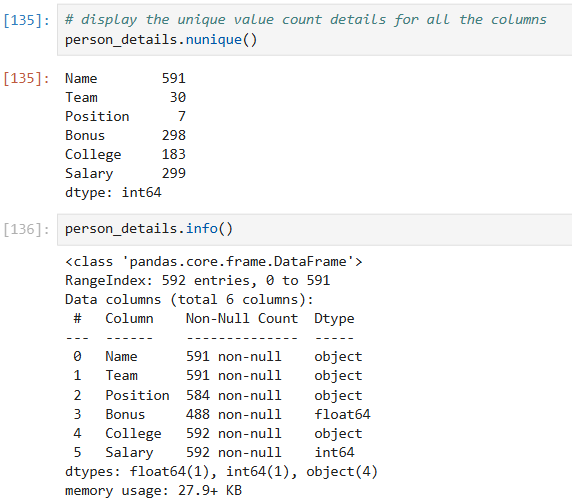
***The astype Method I***

* 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.

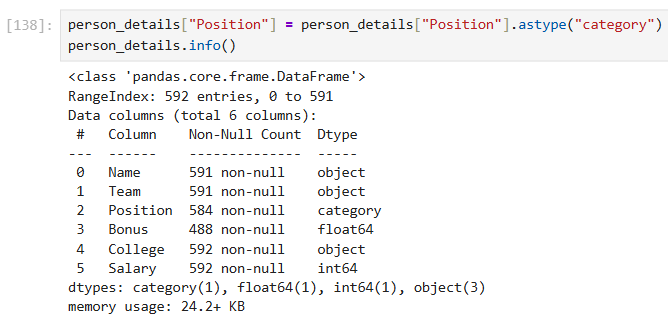
***The astype Method II***

* 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.