1 GBA 465 Lab 06 - Skill-Building with Pandas (Starter)

In this assignment, you will build foundational skills with Pandas by creating and inspecting Series (for one-dimensional data) and DataFrame (for two-dimensional data) objects.

Let's start by importing Pandas.

Action: Execute the following code block.

In [110]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import pandas as pd
```

1.1 Series (One-Dimensional Data) ¶

Consider the following data on some large Lego sets (by number of pieces, not necessarily physical dimensions) from 2020:

Lego Set Name	Number of Pieces
Millennium Falcon	7,541
Hogwarts Castle	6,020
Taj Mahal	5,923
Ninjago City	4,867
Star Destoyer	4,784
Ghostbusters	4,600

1.1.1 Part A - Creating a Series

Actions:

- Create a Series object and store in a variable called `s` .
- Using the Series constructor, pass in a list of integers. The integers are the number of pieces for each of the six (6) large Lego sets listed above. Don't worry about the set names for now.
- Print out the `s` object to see the default data view.
- Inspect the Series object `s` by printing out its data type.

In [111]:

```
# Your implementation:
s = pd.Series([7541,6020,5923,4867,4784,4600])
print (s)
print ()
print (type (s))
```

```
0    7541
1    6020
2    5923
3    4867
4    4784
5    4600
dtype: int64

<class 'pandas.core.series.Series'>
```

1.1.2 Part B - Custom Index Labels

If you created `s` successfully above, the index labels will be 0, 1, 2, 3, 4, and 5.

Actions:

- Re-create the Series object in a variable called `s` .
- Pass an optional argument to the constructor method: a list of custom index labels.
- The custom index labels are the names of the Lego sets (e.g., "Millennium Falcon", "Hogwarts Castle", etc.). Each label must correctly match up with the number of pieces data point.
- Print out the `s` object.

In [112]:

```
# Your implementation:
s = pd.Series([7541,6020,5923,4867,4784,4600]
              , index= [['Millennium Falcon','Hogwarts Castle','Taj Mahal','Ninja
print(s)
```

```
Millennium Falcon    7541
Hogwarts Castle      6020
Taj Mahal            5923
Ninjago City         4867
Star Destoyer        4784
Ghostbusters         4600
dtype: int64
```

Actions:

- Re-create the Series object in a variable called `s` .
- Instead of passing in custom index labels to the constructor method, set the value of the `index` attribute on the Series object.
- Print out the `s` object.

In [113]:

```
# Your implementation:
labels = [ "Millennium Falcon", "Hogwarts Castle", "Taj Mahal", "Ninjago City", '

s = pd.Series ( [ 7541, 6020, 5923, 3867, 4784, 4600 ] )

s.index = labels

print (s)
```

```
Millennium Falcon    7541
Hogwarts Castle      6020
Taj Mahal            5923
Ninjago City         3867
Star Destroyer       4784
Ghostbusters         4600
dtype: int64
```

1.1.3 Part C - Indexing a Series

Actions:

- Using a numerical index with bracket syntax on the Series object `s` , access and store the number of pieces for **Hogwarts Castle** in a variable called `numPiecesA` .
- Print the value in `numPiecesA` .

In [114]:

```
# Your implementation:
numPiecesA = s[1]

print ("A method: The number of pieces in Lego Hogwarts Castle is {:,}.".format (
```

```
A method: The number of pieces in Lego Hogwarts Castle is 6,020.
```

Actions:

- Using a string index (i.e., label) with bracket syntax on the Series object `s` , access and store the number of pieces for **Ninjago City** in a variable called `numPiecesB` .
- Print the value in `numPiecesB` .

```
In [115]: # Your implementation:

          numPiecesB = s["Ninjago City"]

          print ("B method: The number of pieces in Lego Ninjago City is {:,}.".format (nur
```

```
B method: The number of pieces in Lego Ninjago City is 3,867.
```

Actions:

- Using the `loc` attribute of the Series object `s`, access and store the number of pieces for **Taj Mahal** in a variable called `numPiecesC`.
- Print the value in `numPiecesC`.

```
In [116]: # Your implementation:

          numPiecesC = s.loc ["Taj Mahal"]

          print ("C method: The number of pieces in Lego Taj Mahal is {:,}.".format (numPie
```

```
C method: The number of pieces in Lego Taj Mahal is 5,923.
```

1.2 DataFrame (Two-Dimensional Data)

Let's expand our data set of large Lego sets from 2020, adding more sets (rows) and data points (columns):

Lego Set Name	Number of Pieces	Set Number	Number of Minifigures	Retail Price
Millennium Falcon	7,541	75192	10	$799.99
Hogwarts Castle	6,020	71043	4	$399.99
Taj Mahal	5,923	10256	0	$369.99
Ninjago City	4,867	70620	17	$399.99
Star Destoyer	4,784	75055	6	$699.99
Ghostbusters	4,600	75827	9	$648.00
Tower Bridge	4,295	10214	0	$369.75
Big Ben	4,162	10253	0	$527.47
Roller Coaster	4,124	10261	11	$599.99
Disney Castle	4,080	71040	5	$584.70

1.2.1 Part D - Exploring the Data

Before you can create a Pandas DataFrame object to represent this data, you need to get the data into a format that Python can recognize.

Actions:

- Create a list called `setNames` which contains each of the "Set Names" data points, in the order provided (from 1 to 10).
- Create a list called `numpieces` which contains each of the "Number of Pieces" data points, in the order provided (from 1 to 10).
- Create a list called `setNumbers` which contains each of "Set Numbers" data points, in the order provided (from 1 to 10).
- Create a list called `numMinifigures` which contains each of the "Number of Minifigures" data points, in the order provided (from 1 to 10).
- Create a list called `retailPrices` which contains each of the "Retail Prices" data points, in the order provided (from 1 to 10).

In [117]:

```
# Your implementation
setNames = [ "Millenium Falcon", "Hogwarts Castle", "Taj Mahal", "Ninjago City",
numPieces = [ 7541, 6020, 5923, 4867, 4784, 4600, 4295, 4162, 4124, 4080 ]
setNumbers = [ 75192, 71043, 10256, 70620, 75055, 75827, 10214, 10253, 10261, 71
numMinifigures = [ 10, 4, 0, 17, 6, 9, 0, 0, 11, 5 ]
retailPrices = [ 799.99, 399.99, 369.99, 399.99, 699.99, 648.00, 369.75, 527.47,
```

Action:

- Print out the length of each list variable to make sure that they are all the same length.

In [119]:

```
# Your implementation:
print (len (setNames))
print (len (numPieces))
print (len (setNumbers))
print (len (numMinifigures))
print (len (retailPrices))
```

```
10
10
10
10
10
```

1.2.2 Part E - Creating a DataFrame

Actions:

- Create a Pandas DataFrame object called `df` .
- Pass the data from `setNames` , `pieces` , `setNumbers` , `minifigures` , and `retailPrices` into the DataFrame constructor, providing column labels.
- Print out the `df` object to see the default data view.
- Inspect the Pandas DataFrame object `df` by printing out its data type.

In [122]:

```
# Your implementation:
df = pd.DataFrame({
    "Set Names": setNames,
    "Num Pieces": numPieces,
    "Set Numbers": setNumbers,
    "Num Minifigures": numMinifigures,
    "Retail Prices": retailPrices})
print (df)
print ()
print (type (df))
#print(type(df))
```

```
          Set Names  Num Pieces  Set Numbers  Num Minifigures  Retail Prices
0  Millenium Falcon        7541        75192               10         799.99
1   Hogwarts Castle        6020        71043                4         399.99
2         Taj Mahal        5923        10256                0         369.99
3      Ninjago City        4867        70620               17         399.99
4    Star Destoryer        4784        75055                6         699.99
5      Ghostbusters        4600        75827                9         648.00
6      Tower Bridge        4295        10214                0         369.75
7           Big Ben        4162        10253                0         527.47
8    Roller Coaster        4124        10261               11         599.99
9     Disney Castle        4080        71040                5         584.70

<class 'pandas.core.frame.DataFrame'>
```

1.2.3 Part F - Accessing a Series

Action:

- Use bracket syntax to access the Series representing `setNames` from the DataFrame object `df` . You will need to use whatever column label you provided to the constructor to represent `setNames` (this value is a String).
- Store the Series in `s` .
- Print `s` .
- Print the data type for `s` .

In [123]:

```
# Your implementation:
s = df [ "Set Names" ]

print (s)
print (type (s))
```

```
0    Millenium Falcon
1     Hogwarts Castle
2           Taj Mahal
3        Ninjago City
4      Star Destoryer
5        Ghostbusters
6        Tower Bridge
7             Big Ben
8      Roller Coaster
9       Disney Castle
Name: Set Names, dtype: object
<class 'pandas.core.series.Series'>
```

1.2.4 Part G - Inspecting DataFrame Data Types

Action:

- Print out the `dtypes` attribute of the DataFrame object `df` .

In [124]:

```
# Your implementation:
print(df.dtypes)
```

```
Set Names           object
Num Pieces           int64
Set Numbers          int64
Num Minifigures      int64
Retail Prices      float64
dtype: object
```

1.2.5 Part H - Reading Data

Actions:

- Use Pandas to read the assignment's data file directly into a Pandas DataFrame object called `df` .
- Print the `df` object.

In [125]:

```
# Your implementation:
df = pd.read_csv("gba-465-lab-06-skill-building-with-pandas-data.csv")
print(df)
```

```
           Set Name  Pieces  Set Number  Minifigures  Retail Price
0           Big Ben    4162       10253            0        527.47
1     Disney Castle    4080       71040            5        584.70
2      Ghostbusters    4600       75827            9        648.00
3   Hogwarts Castle    6020       71043            4        399.99
4  Millenium Falcon    7541       75192           10        799.99
5      Ninjago City    4867       70620           17        399.99
6    Roller Coaster    4124       10261           11        599.99
7    Star Destroyer    4784       75055            6        699.99
8         Taj Mahal    5923       10256            0        369.99
9      Tower Bridge    4295       10214            0        369.75
```

1.2.6 Part I - Previewing Large Data Sets

Action:

- Use the `head` method to print out the top five (5) rows of the Pandas DataFrame object `df`.

In [130]:

```
# Your implementation:

print(df.head(5))
```

```
           Set Name  Pieces  Set Number  Minifigures  Retail Price
0           Big Ben    4162       10253            0        527.47
1     Disney Castle    4080       71040            5        584.70
2      Ghostbusters    4600       75827            9        648.00
3   Hogwarts Castle    6020       71043            4        399.99
4  Millenium Falcon    7541       75192           10        799.99
```

1.2.7 Part J - Inspecting the DataFrame

Action:

- Use the `info` method to print out descriptive information about `df` and its data.

In [131]:

```
# Your implementation:

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Set Name      10 non-null     object 
 1   Pieces        10 non-null     int64  
 2   Set Number    10 non-null     int64  
 3   Minifigures   10 non-null     int64  
 4   Retail Price  10 non-null     float64
dtypes: float64(1), int64(3), object(1)
memory usage: 528.0+ bytes
None
```

Action:

- Use the `shape` attribute to print the shape (rows, then columns) of `df`.

In [132]:

```
# Your implementation:
print(df.shape)
```

```
(10, 5)
```

Action:

- Print the `columns` attribute of `df` , and also print its type.

In [133]:

```
# Your implementation:
print (df.columns)
print (type (df.columns))
```

```
Index(['Set Name', 'Pieces', 'Set Number', 'Minifigures', 'Retail Price'], dtyp
e='object')
<class 'pandas.core.indexes.base.Index'>
```

Action:

- Print the `index` attribute of `df` .

In [134]:

```
# Your implementation:

print(df.index)
```

```
RangeIndex(start=0, stop=10, step=1)
```

Actions:

- Print out statistics of the data in `df` using the `describe` method.
- Print out the type of the value returned from the `describe` method.

In [135]:

```
# Your implementation:
print (df.describe ())
print ()
print (type (df.describe()))
```

```
            Pieces    Set Number  Minifigures  Retail Price
count    10.000000     10.000000    10.000000     10.000000
mean   5039.600000  47976.100000     6.200000    539.986000
std    1124.016528  32524.622155     5.613476    151.915017
min    4080.000000  10214.000000     0.000000    369.750000
25%    4195.250000  10257.250000     1.000000    399.990000
50%    4692.000000  70830.000000     5.500000    556.085000
75%    5659.000000  74052.000000     9.750000    635.997500
max    7541.000000  75827.000000    17.000000    799.990000

<class 'pandas.core.frame.DataFrame'>
```

1.2.8 Part K - Renaming Columns

Let's rename some of the columns in the `df` DataFrame to make them easier to read.

Actions:

- Rename the "Pieces" column to "Number of Pieces"
- Rename the "Minifigures" column to "Number of Minifigures"
- Print out the `df` object to see the default data view.

In [136]:
```
# Your implementation:
df = df.rename (columns = { "Pieces":"Number of Pieces", "Minifigures":"Number of
print(df)
```

```
           Set Name  Number of Pieces  Set Number  Number of Minifigures  \
0           Big Ben              4162       10253                      0
1     Disney Castle              4080       71040                      5
2      Ghostbusters              4600       75827                      9
3   Hogwarts Castle              6020       71043                      4
4  Millenium Falcon              7541       75192                     10
5      Ninjago City              4867       70620                     17
6    Roller Coaster              4124       10261                     11
7    Star Destroyer              4784       75055                      6
8         Taj Mahal              5923       10256                      0
9      Tower Bridge              4295       10214                      0

   Retail Price
0        527.47
1        584.70
2        648.00
3        399.99
4        799.99
5        399.99
6        599.99
7        699.99
8        369.99
9        369.75
```

1.2.9 Part L - Filtering by Columns

We have too much data (too many columns). Let's filter the colums in the `df` DataFrame to focus our analysis.

Actions:

- Filter the existing DataFrame `df` so that it only includes the following columns, and in this order:
    - Set Number
    - Set Name
    - Number of Pieces
    - Number of Minifigures
- Store the new, filtered DataFrame back into a variable called `df_filtered` .
- Print the head for `df_filtered` .
- Print the number of rows in `df_filtered` .

In [137]:

```python
# Your implementation:
df_filtered = df[['Set Number','Set Name','Number of Pieces','Number of Minifigur
df_filtered
print(df_filtered.head())
print(df_filtered.shape[0])
print ("Number of rows: {}".format (df_filtered.shape [0]))
```

Out[137]:

	Set Number	Set Name	Number of Pieces	Number of Minifigures
0	10253	Big Ben	4162	0
1	71040	Disney Castle	4080	5
2	75827	Ghostbusters	4600	9
3	71043	Hogwarts Castle	6020	4
4	75192	Millenium Falcon	7541	10
5	70620	Ninjago City	4867	17
6	10261	Roller Coaster	4124	11
7	75055	Star Destroyer	4784	6
8	10256	Taj Mahal	5923	0
9	10214	Tower Bridge	4295	0

```
   Set Number          Set Name  Number of Pieces  Number of Minifigures
0       10253           Big Ben              4162                      0
1       71040     Disney Castle              4080                      5
2       75827      Ghostbusters              4600                      9
3       71043   Hogwarts Castle              6020                      4
4       75192  Millenium Falcon              7541                     10
10
Number of rows: 10
```

1.2.10 Part M - Filtering Using a Condition

We have too much data (too many rows). Let's filter the rows in the `df_filtered` DataFrame to focus our analysis.

Actions:

- Filter the existing DataFrame `df_filtered` :
  - Create a variable called `condition1` to only contain the large Lego sets which include **more than 5 minifigures**.
  - Use `condition1` to filter `df_filtered` using bracket syntax.
- Store the new, filtered DataFrame back into `df_filtered` .
- Print the head for `df_filtered` .
- Print the number of rows in `df_filtered` .

In [138]:

```python
# Your implementation:
condition1 = df_filtered['Number of Minifigures'] > 5
df_filtered = df_filtered[condition1]
print(df_filtered.head())
print(df_filtered.shape[0])
print ("Number of rows: {}".format (df_filtered.shape [0]))
```

```
   Set Number          Set Name  Number of Pieces  Number of Minifigures
2       75827      Ghostbusters              4600                      9
4       75192  Millenium Falcon              7541                     10
5       70620      Ninjago City              4867                     17
6       10261    Roller Coaster              4124                     11
7       75055    Star Destroyer              4784                      6
5
Number of rows: 5
```

Let's create a second condition filter.

Actions:

- Filter the existing DataFrame `df_filtered` :
    - Create a variable called `condition2` to only contain the large Lego sets which have **at least 5,000 pieces**.
    - Use `condition2` to filter `df_filtered` using backet syntax.
- Store the new, filtered DataFrame back into `df_filtered` .
- Print the head for `df_filtered` .
- Print the number of rows in `df_filtered` .

In [139]:

```
# Your implementation:
condition2 = df_filtered['Number of Pieces'] > 5000
df_filtered = df_filtered[condition2]
print(df_filtered.head())
print(df_filtered.shape[0])
print ("Number of rows: {}".format (df_filtered.shape [0]))
```

```
   Set Number          Set Name  Number of Pieces  Number of Minifigures
4       75192  Millenium Falcon              7541                     10
1
Number of rows: 1
```

1.2.11 Part N - Sorting a DataFrame

Let's practice sorting the original DataFrame `df` in a couple of different ways.

First, let's sort by creating a new index in the DataFrame based on an existing column of data.

Actions:

- Use the `set_index` method to put "Set Names" into the index.
- Store the new, sorted DataFrame into a new variable called `df_sorted` .
- Use the `sort_index` method to sort by "Set Names" in alphabetical (ascending) order.
- Store the new, sorted DataFrame back into `df_sorted` .
- Print the head for `df_sorted` .

In [97]:

```
# Your implementation:
df_sorted = df.set_index('Set Name')
df_sorted = df_sorted.sort_index(ascending= True)
print(df_sorted.head())
```

```
                  Number of Pieces  Set Number  Number of Minifigures  \
Set Name
Big Ben                       4162       10253                      0
Disney Castle                 4080       71040                      5
Ghostbusters                  4600       75827                      9
Hogwarts Castle               6020       71043                      4
Millenium Falcon              7541       75192                     10

                  Retail Price
Set Name
Big Ben                 527.47
Disney Castle           584.70
Ghostbusters            648.00
Hogwarts Castle         399.99
Millenium Falcon        799.99
```

Second, let's sort on one of the columns in the DataFrame.

Actions:

- Use the `sort_values` method to sort on "Number of Pieces".
- Use the optional `ascending` parameter in the `sort_values` method to adjust the sorting from largest to smallest (descending) order.
- Store the new, sorted DataFrame back into `df_sorted` .
- Print the head for `df_sorted` .

In [98]:
```
# Your implementation:
df_sorted = df.set_index('Number of Pieces')
df_sorted = df_sorted.sort_index(ascending= True)
print(df_sorted.head())
```

```
                        Set Name  Set Number  Number of Minifigures  \
Number of Pieces
4080               Disney Castle       71040                      5
4124              Roller Coaster       10261                     11
4162                     Big Ben       10253                      0
4295                Tower Bridge       10214                      0
4600                Ghostbusters       75827                      9

                  Retail Price
Number of Pieces
4080                    584.70
4124                    599.99
4162                    527.47
4295                    369.75
4600                    648.00
```

Third, let's sort on multiple columns in the DataFrame.

Actions:

- Use the `sort_values` method to sort first on "Number of Minifigures" (descending), then by "Number of Pieces" (descending).
- Store the new, sorted DataFrame back into `df_sorted`.
- Print the head for `df_sorted`.

In [103]:
```
# Your implementation:
df_sorted = df_sorted.sort_values(["Number of Minifigures","Number of Pieces"],as
print(df_sorted.head())
```

```
                          Set Name  Set Number  Number of Minifigures  \
Number of Pieces
4867                  Ninjago City       70620                     17
4124                Roller Coaster       10261                     11
7541              Millenium Falcon       75192                     10
4600                  Ghostbusters       75827                      9
4784                Star Destroyer       75055                      6

                  Retail Price
Number of Pieces
4867                    399.99
4124                    599.99
7541                    799.99
4600                    648.00
4784                    699.99
```

1.2.12 Part O - Statistical Operations

Let's calculate the sum and mean for some of the colums in the existing DataFrame `df`.

Actions:

- Use bracket syntax to access the following for the "Number of Pieces" column.
  - Determine the sum of all pieces. Store this value in `totalPieces`, then print.
  - Determine the mean number of pieces. Store this value in `averageNumPieces`, then print.
  - Determine the minimum number of pieces. Store this value in `minNumPieces`, then print.
  - Determine the maximum number of pieces. Store this value in `maxNumPieces`, then print.

In [140]:

```
# Your implementation:
totalPieces = df['Number of Pieces'].sum()
averageNumPieces = df['Number of Pieces'].mean()
minNumPieces = df['Number of Pieces'].min()
maxNumPieces = df['Number of Pieces'].max()
print ("Total Pieces: {:,}".format (totalPieces))
print ("Average Number of Pieces: {:,.2f}".format (averageNumPieces))
print ("Min Num Pieces: {:,}".format (minNumPieces))
print ("Max Num Pieces: {:,}".format (maxNumPieces))
```

```
Total Pieces: 50,396
Average Number of Pieces: 5,039.60
Min Num Pieces: 4,080
Max Num Pieces: 7,541
```

In [ ]: