###

# CST8333 2018 Final Project #

# Created by Arish Kakadiya #

# Student number: 040894137 #

# November 26 ,2018 #

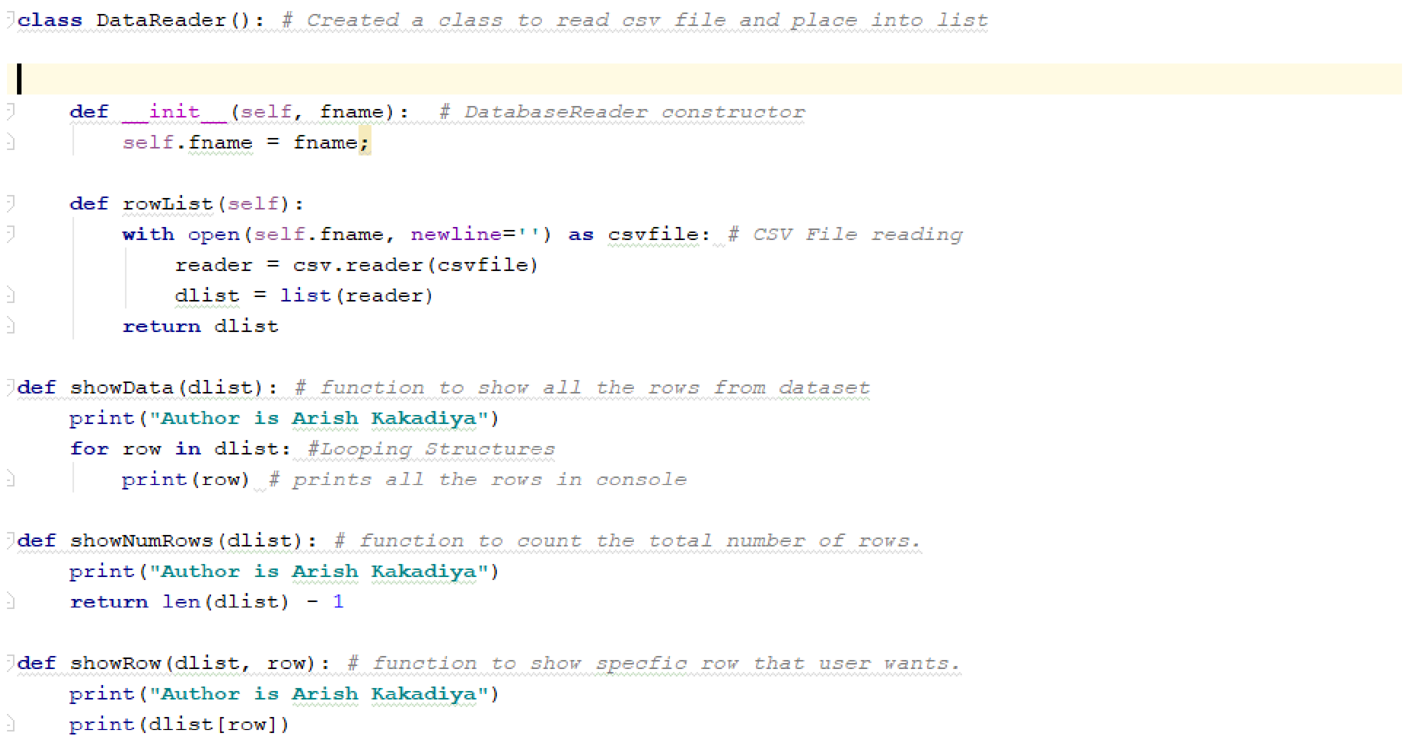
# #

###

I chose Python as the language I wanted to research, and my project is centered around an 32100054.csv food database. Which is having Food data for Canada Geo location and have Food categories, Commodity UOM, UOM_ID etc. as column data.

I have implemented Feature on the given data base are listed below:

Created a class to read csv file and place into list

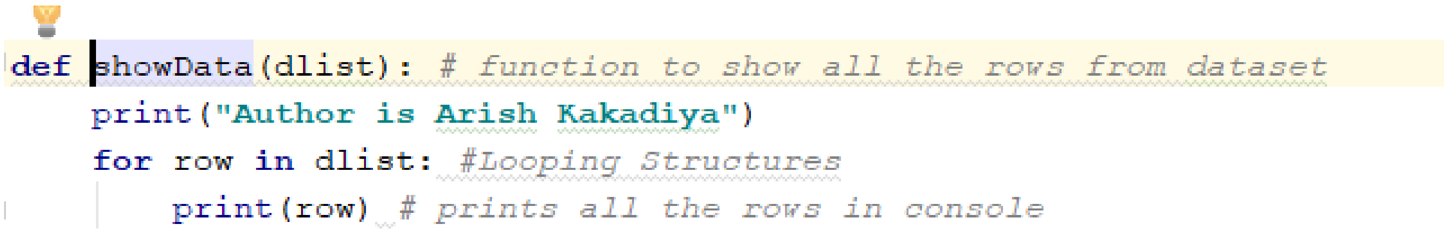


```
class DataReader(): # Created a class to read csv file and place into list


    def __init__(self, fname):  # DatabaseReader constructor
        self.fname = fname;

    def rowList(self):
        with open(self.fname, newline='') as csvfile: # CSV File reading
            reader = csv.reader(csvfile)
            dlist = list(reader)
        return dlist

def showData(dlist): # function to show all the rows from dataset
    print("Author is Arish Kakadiya")
    for row in dlist: #Looping Structures
        print(row) # prints all the rows in console

def showNumRows(dlist): # function to count the total number of rows.
    print("Author is Arish Kakadiya")
    return len(dlist) - 1

def showRow(dlist, row): # function to show specfic row that user wants.
    print("Author is Arish Kakadiya")
    print(dlist[row])
```

Feacture & Functions are listed below:

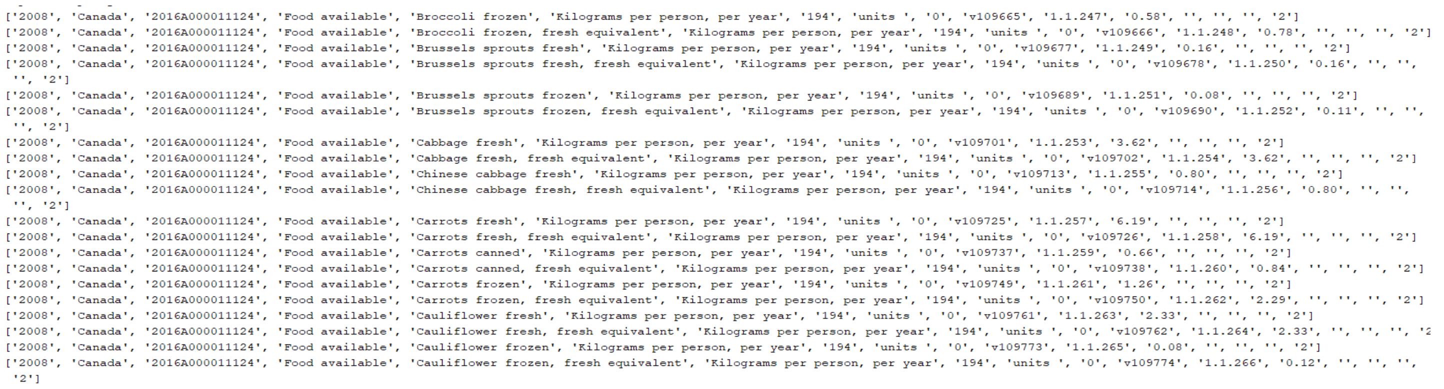
showData(dList):

To show the all the rows and column data I have used For Loop on RowList and Panda DataFrame to display output data in terminal as shown below:



```python
def showData(dlist): # function to show all the rows from dataset
    print("Author is Arish Kakadiya")
    for row in dlist:  #Looping Structures
        print(row) # prints all the rows in console
```

And output for this code is :

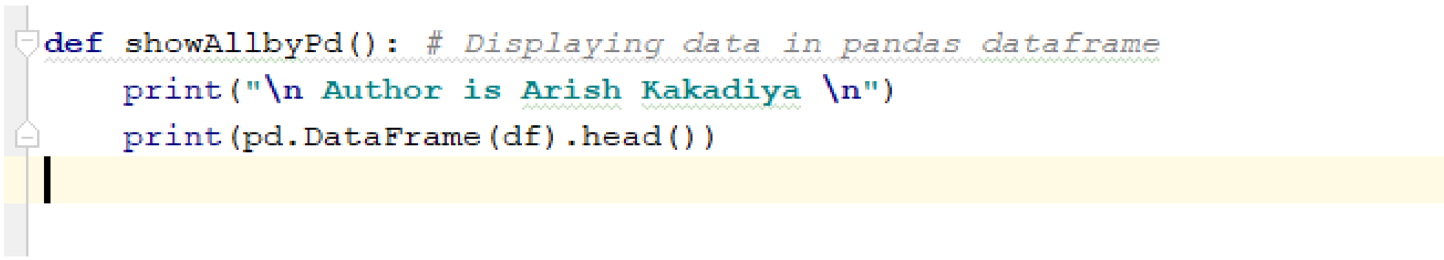


```
['2008', 'Canada', '2016A000011124', 'Food available', 'Broccoli frozen', 'Kilograms per person, per year', '194', 'units ', '0', 'v109665', '1.1.247', '0.58', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Broccoli frozen, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109666', '1.1.248', '0.78', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Brussels sprouts fresh', 'Kilograms per person, per year', '194', 'units ', '0', 'v109677', '1.1.249', '0.16', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Brussels sprouts fresh, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109678', '1.1.250', '0.16', '', '',
 '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Brussels sprouts frozen', 'Kilograms per person, per year', '194', 'units ', '0', 'v109689', '1.1.251', '0.08', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Brussels sprouts frozen, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109690', '1.1.252', '0.11', '', '',
 '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Cabbage fresh', 'Kilograms per person, per year', '194', 'units ', '0', 'v109701', '1.1.253', '3.62', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Cabbage fresh, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109702', '1.1.254', '3.62', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Chinese cabbage fresh', 'Kilograms per person, per year', '194', 'units ', '0', 'v109713', '1.1.255', '0.80', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Chinese cabbage fresh, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109714', '1.1.256', '0.80', '', '',
 '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots fresh', 'Kilograms per person, per year', '194', 'units ', '0', 'v109725', '1.1.257', '6.19', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots fresh, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109726', '1.1.258', '6.19', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots canned', 'Kilograms per person, per year', '194', 'units ', '0', 'v109737', '1.1.259', '0.66', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots canned, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109738', '1.1.260', '0.84', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots frozen', 'Kilograms per person, per year', '194', 'units ', '0', 'v109749', '1.1.261', '1.26', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Carrots frozen, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109750', '1.1.262', '2.29', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Cauliflower fresh', 'Kilograms per person, per year', '194', 'units ', '0', 'v109761', '1.1.263', '2.33', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Cauliflower fresh, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109762', '1.1.264', '2.33', '', '', '', '2
['2008', 'Canada', '2016A000011124', 'Food available', 'Cauliflower frozen', 'Kilograms per person, per year', '194', 'units ', '0', 'v109773', '1.1.265', '0.08', '', '', '', '2']
['2008', 'Canada', '2016A000011124', 'Food available', 'Cauliflower frozen, fresh equivalent', 'Kilograms per person, per year', '194', 'units ', '0', 'v109774', '1.1.266', '0.12', '', '', '',
 '2']
```

Here Data is displayed using pandas DataFrame:

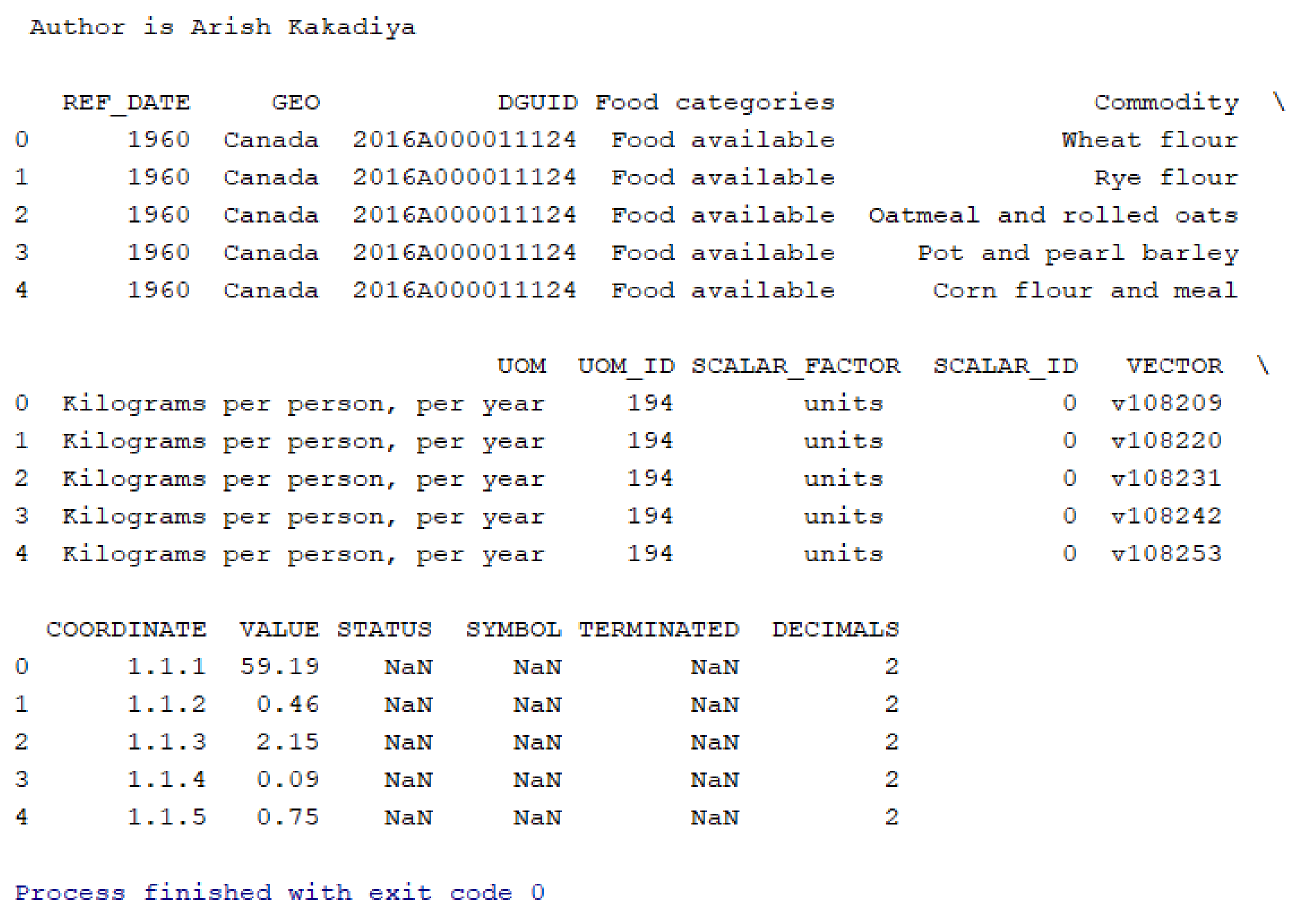
Head will print 5 rows by default

(Ref. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.head.html)



```python
def showAllbyPd(): # Displaying data in pandas dataframe
    print("\n Author is Arish Kakadiya \n")
    print(pd.DataFrame(df).head())
```

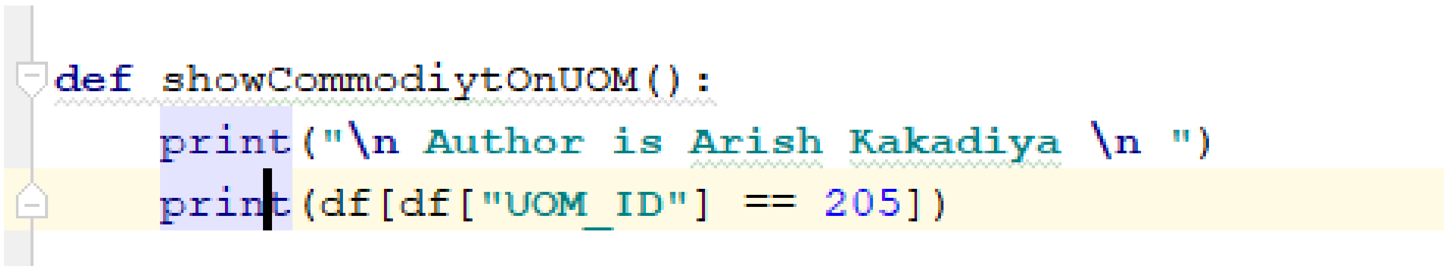
And output for this code this:



```
 Author is Arish Kakadiya

   REF_DATE     GEO            DGUID Food categories                  Commodity  \
0      1960  Canada  2016A000011124  Food available                Wheat flour
1      1960  Canada  2016A000011124  Food available                  Rye flour
2      1960  Canada  2016A000011124  Food available  Oatmeal and rolled oats
3      1960  Canada  2016A000011124  Food available      Pot and pearl barley
4      1960  Canada  2016A000011124  Food available       Corn flour and meal

                                UOM  UOM_ID SCALAR_FACTOR  SCALAR_ID   VECTOR  \
0  Kilograms per person, per year     194         units          0  v108209
1  Kilograms per person, per year     194         units          0  v108220
2  Kilograms per person, per year     194         units          0  v108231
3  Kilograms per person, per year     194         units          0  v108242
4  Kilograms per person, per year     194         units          0  v108253

  COORDINATE  VALUE  STATUS  SYMBOL  TERMINATED  DECIMALS
0      1.1.1  59.19     NaN     NaN         NaN         2
1      1.1.2   0.46     NaN     NaN         NaN         2
2      1.1.3   2.15     NaN     NaN         NaN         2
3      1.1.4   0.09     NaN     NaN         NaN         2
4      1.1.5   0.75     NaN     NaN         NaN         2

Process finished with exit code 0
```

showCommodiytOnUOM() # Function for Showing all rows Commodity based on UOM

Here I have displayed Commodity data on given UOM_ID which is done using Pandas Data Frame.



```python
def showCommodiytOnUOM():
    print("\n Author is Arish Kakadiya \n ")
    print(df[df["UOM_ID"] == 205])
```

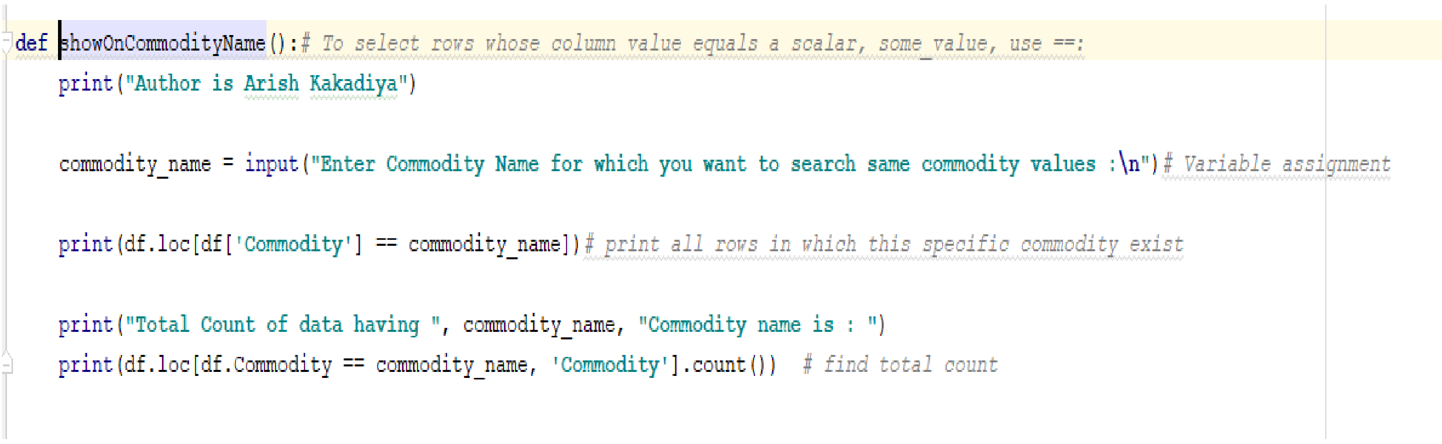
Output:

```
 Author is Arish Kakadiya

     REF_DATE     GEO           DGUID                      Food categories  \
18       1960  Canada  2016A000011124                       Food available
22       1960  Canada  2016A000011124                       Food available
23       1960  Canada  2016A000011124                       Food available
24       1960  Canada  2016A000011124                       Food available
25       1960  Canada  2016A000011124                       Food available
26       1960  Canada  2016A000011124                       Food available
27       1960  Canada  2016A000011124                       Food available
42       1960  Canada  2016A000011124                       Food available
44       1960  Canada  2016A000011124                       Food available
46       1960  Canada  2016A000011124                       Food available
48       1960  Canada  2016A000011124                       Food available
50       1960  Canada  2016A000011124                       Food available
52       1960  Canada  2016A000011124                       Food available
54       1960  Canada  2016A000011124                       Food available
56       1960  Canada  2016A000011124                       Food available
```

showOnCommodityName() *#Function for Showing all rows having specific commodity name*

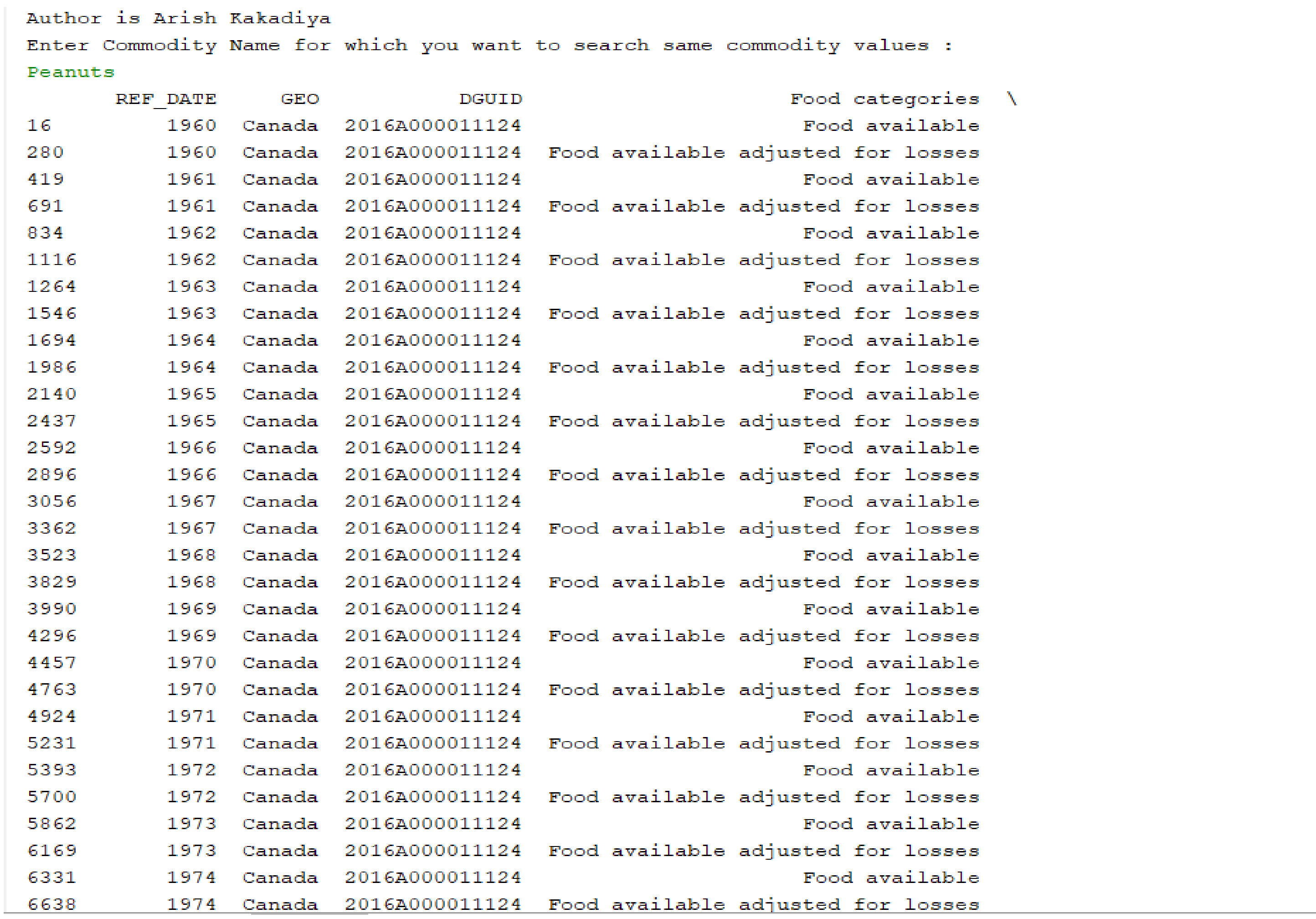
To show data Rows for given specific column values in input which was taken as input using Input function and Also have shown Total count of data rows which is having Specific commodity name which was done using the loc indexer for Pandas Dataframe is used for integer-location based indexing / selection by position.

Ref.(https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.loc.html)



```python
def showOnCommodityName():# To select rows whose column value equals a scalar, some_value, use ==:
    print("Author is Arish Kakadiya")

    commodity_name = input("Enter Commodity Name for which you want to search same commodity values :\n")# Variable assignment

    print(df.loc[df['Commodity'] == commodity_name])# print all rows in which this specific commodity exist

    print("Total Count of data having ", commodity_name, "Commodity name is : ")
    print(df.loc[df.Commodity == commodity_name, 'Commodity'].count())  # find total count
```

Output:



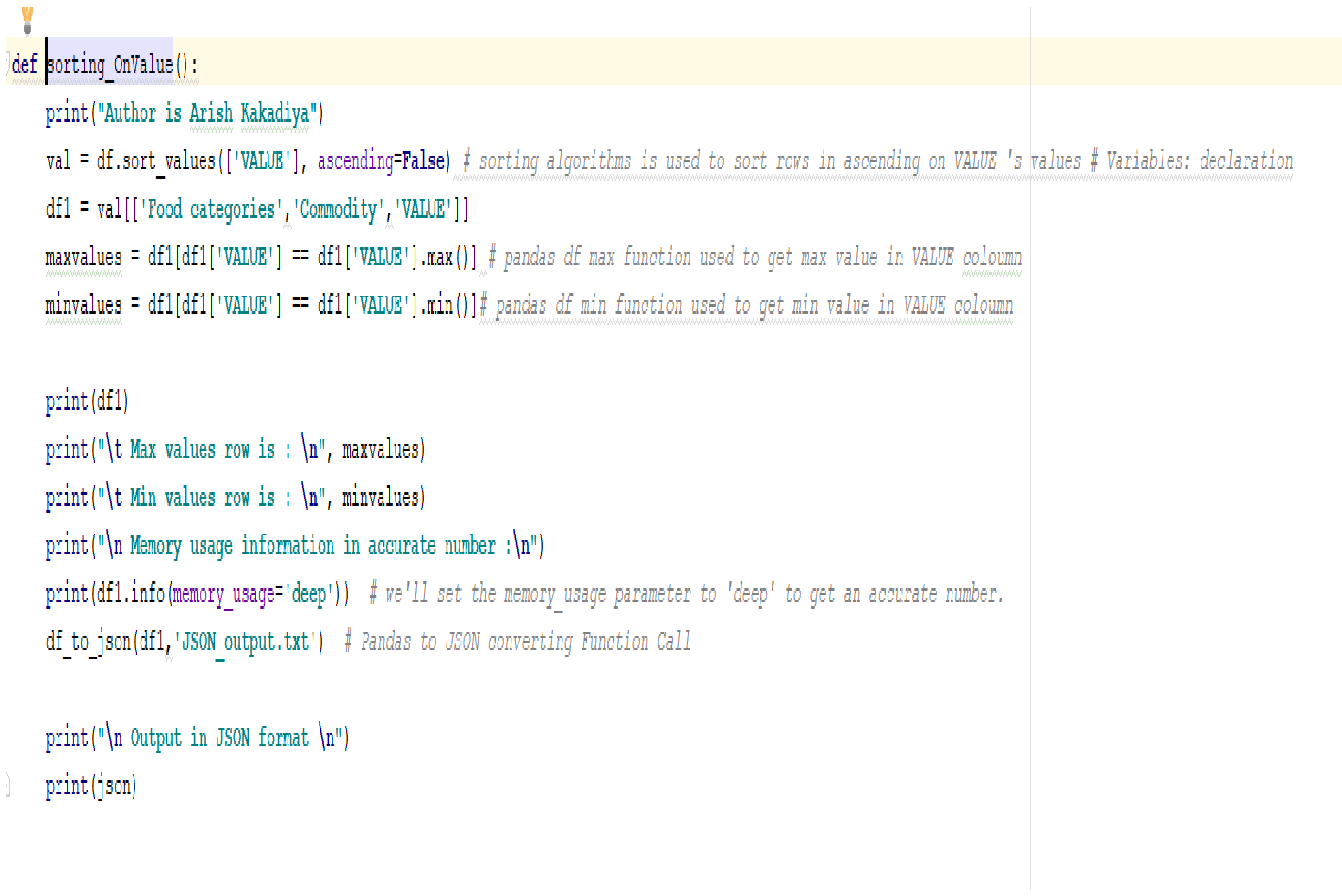
```
Author is Arish Kakadiya
Enter Commodity Name for which you want to search same commodity values :
Peanuts
       REF_DATE    GEO           DGUID                    Food categories  \
16         1960  Canada  2016A000011124                    Food available
280        1960  Canada  2016A000011124  Food available adjusted for losses
419        1961  Canada  2016A000011124                    Food available
691        1961  Canada  2016A000011124  Food available adjusted for losses
834        1962  Canada  2016A000011124                    Food available
1116       1962  Canada  2016A000011124  Food available adjusted for losses
1264       1963  Canada  2016A000011124                    Food available
1546       1963  Canada  2016A000011124  Food available adjusted for losses
1694       1964  Canada  2016A000011124                    Food available
1986       1964  Canada  2016A000011124  Food available adjusted for losses
2140       1965  Canada  2016A000011124                    Food available
2437       1965  Canada  2016A000011124  Food available adjusted for losses
2592       1966  Canada  2016A000011124                    Food available
2896       1966  Canada  2016A000011124  Food available adjusted for losses
3056       1967  Canada  2016A000011124                    Food available
3362       1967  Canada  2016A000011124  Food available adjusted for losses
3523       1968  Canada  2016A000011124                    Food available
3829       1968  Canada  2016A000011124  Food available adjusted for losses
3990       1969  Canada  2016A000011124                    Food available
4296       1969  Canada  2016A000011124  Food available adjusted for losses
4457       1970  Canada  2016A000011124                    Food available
4763       1970  Canada  2016A000011124  Food available adjusted for losses
4924       1971  Canada  2016A000011124                    Food available
5231       1971  Canada  2016A000011124  Food available adjusted for losses
5393       1972  Canada  2016A000011124                    Food available
5700       1972  Canada  2016A000011124  Food available adjusted for losses
5862       1973  Canada  2016A000011124                    Food available
6169       1973  Canada  2016A000011124  Food available adjusted for losses
6331       1974  Canada  2016A000011124                    Food available
6638       1974  Canada  2016A000011124  Food available adjusted for losses
```

```
Total Count of data having  Peanuts Commodity name is :
116
```

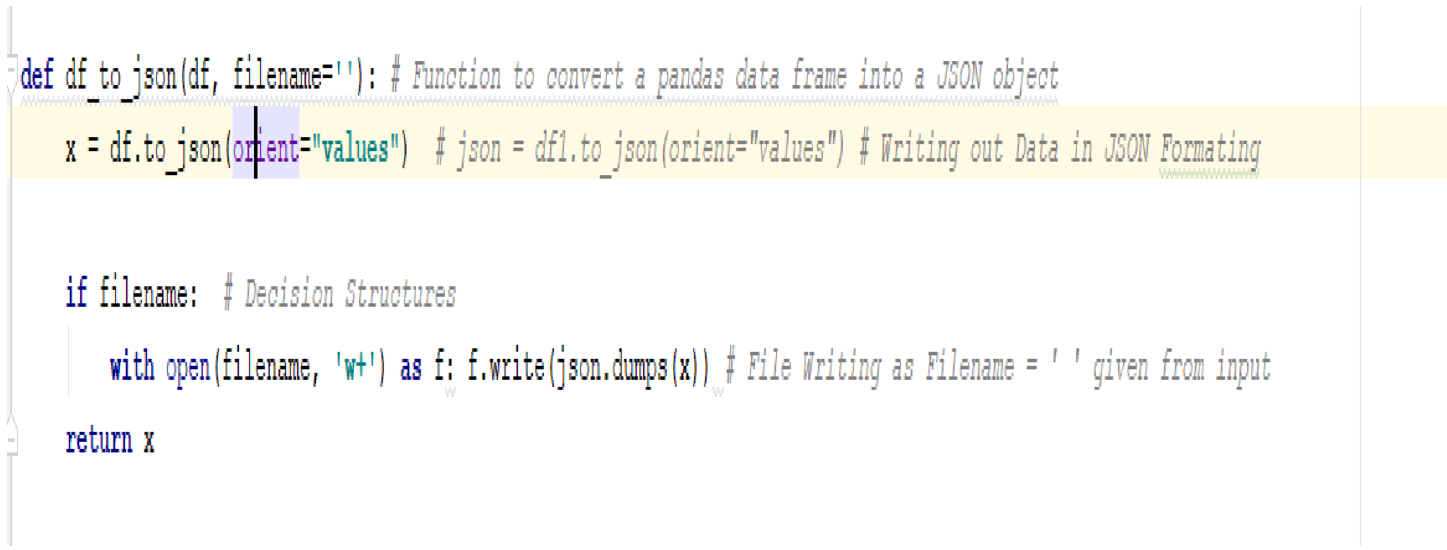
sorting_OnValue() *# function for sorting Values in ascending or descending order*

This function is used to sort the given dataset based on VALUE's data value to sort I used sort function and calculated Max and Min data Values of VALUE column using Max and Min function.
And to show Memory Usage during execution, I used **df1.info(memory_usage='deep')** which gives summary of a Data-Frame and returns None. And shown Sorted data in JSON format by converting csv to JSON.

Ref.(https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.info.html)



```python
def sorting_OnValue():
    print("Author is Arish Kakadiya")
    val = df.sort_values(['VALUE'], ascending=False) # sorting algorithms is used to sort rows in ascending on VALUE 's values # Variables: declaration
    df1 = val[['Food categories','Commodity','VALUE']]
    maxvalues = df1[df1['VALUE'] == df1['VALUE'].max()] # pandas df max function used to get max value in VALUE coloumn
    minvalues = df1[df1['VALUE'] == df1['VALUE'].min()]# pandas df min function used to get min value in VALUE coloumn

    print(df1)
    print("\t Max values row is : \n", maxvalues)
    print("\t Min values row is : \n", minvalues)
    print("\n Memory usage information in accurate number :\n")
    print(df1.info(memory_usage='deep'))  # we'll set the memory_usage parameter to 'deep' to get an accurate number.
    df_to_json(df1,'JSON_output.txt')  # Pandas to JSON converting Function Call

    print("\n Output in JSON format \n")
    print(json)
```



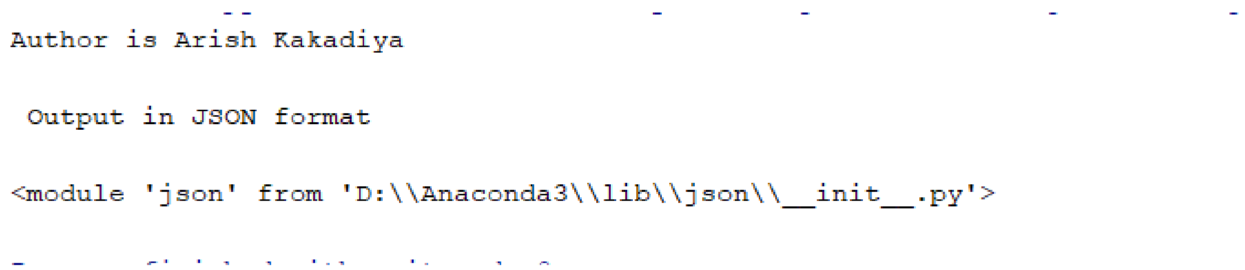
```python
def df_to_json(df, filename=''): # Function to convert a pandas data frame into a JSON object
    x = df.to_json(orient="values")  # json = df1.to_json(orient="values") # Writing out Data in JSON Formating

    if filename:  # Decision Structures
        with open(filename, 'w+') as f: f.write(json.dumps(x)) # File Writing as Filename = ' ' given from input
    return x
```

Output:

Output in sorted order (Descending order)

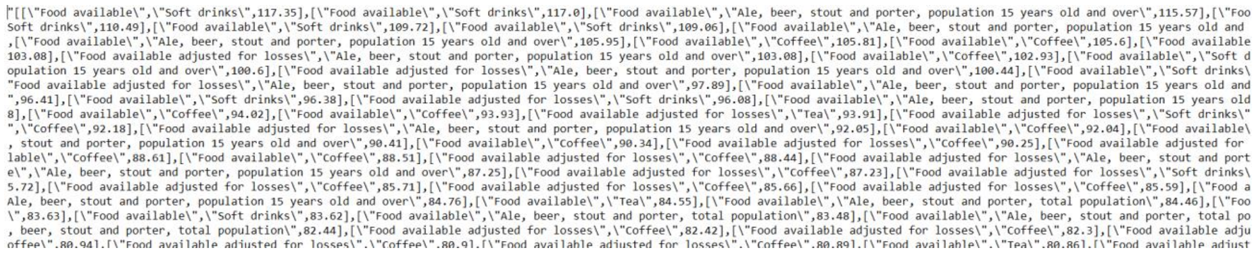
```
Author is Arish Kakadiya
      Food categories  \
19160  Food available
19729  Food available
6808   Food available
5870   Food available
6339   Food available

                                                   Commodity   VALUE
19160                                            Soft drinks  117.35
19729                                            Soft drinks  117.00
6808   Ale, beer, stout and porter, population 15 years old and over  115.57
5870   Ale, beer, stout and porter, population 15 years old and over  115.53
6339   Ale, beer, stout and porter, population 15 years old and over  115.31
```

```
 Max values row is :
      Food categories    Commodity   VALUE
19160  Food available  Soft drinks  117.35

 Min values row is :
      Food categories                                  Commodity  \
11706  Food available  Vegetables not specified frozen, fresh equivalent

      VALUE
11706  -0.21

 Memory usage information in accurate number :

<class 'pandas.core.frame.DataFrame'>
Int64Index: 30559 entries, 19160 to 30552
Data columns (total 3 columns):
Food categories    30559 non-null object
Commodity          30559 non-null object
VALUE              29926 non-null float64
dtypes: float64(1), object(2)
memory usage: 5.0 MB
None
```



```
Author is Arish Kakadiya

 Output in JSON format

<module 'json' from 'D:\\Anaconda3\\lib\\json\\__init__.py'>
```

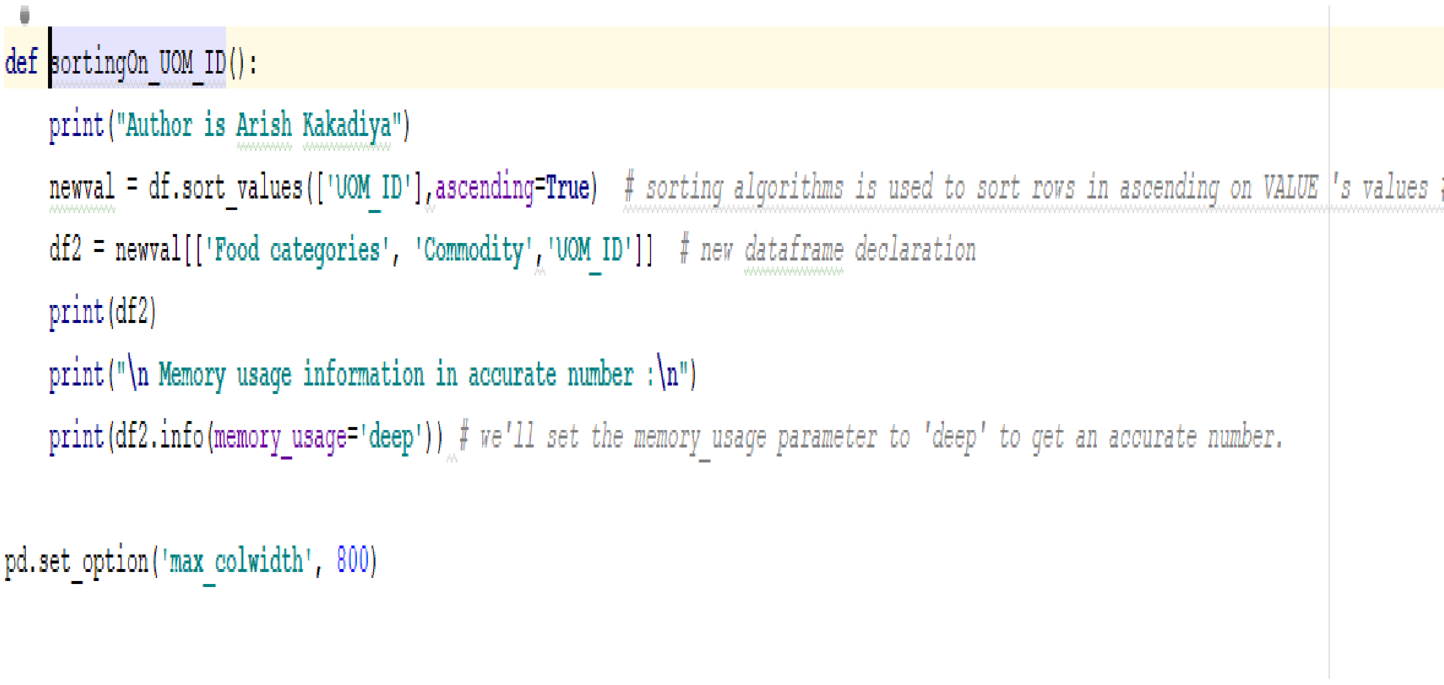
Output JSON file is created in file directory and Data In JSON format will be like :



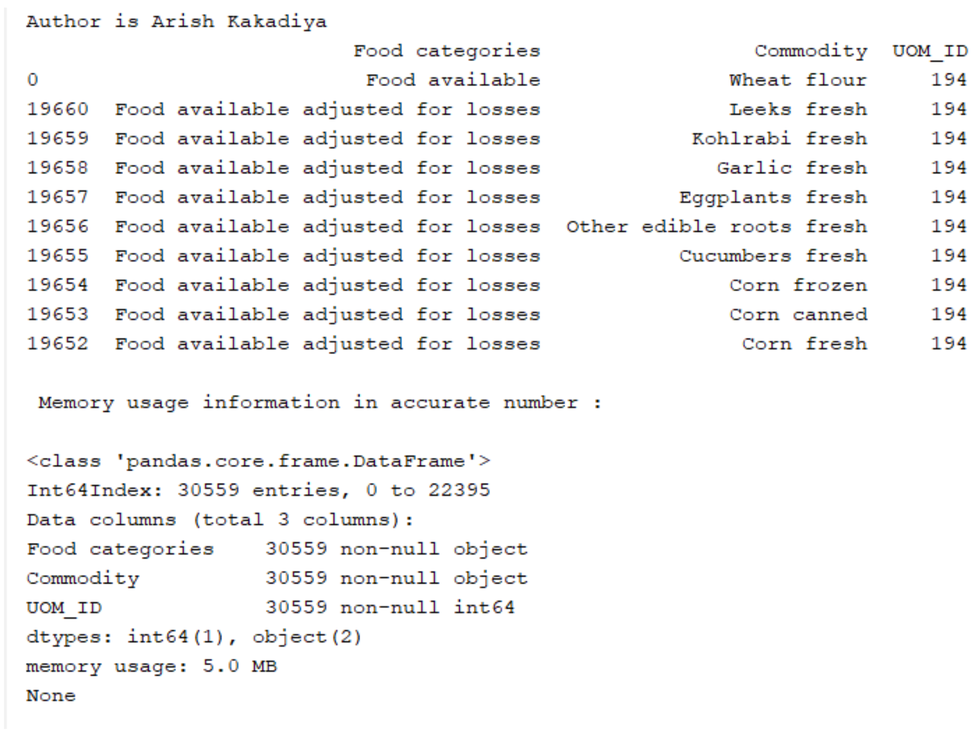
```
"[[\"Food available\",\"Soft drinks\",117.35],[\"Food available\",\"Soft drinks\",117.0],[\"Food available\",\"Ale, beer, stout and porter, population 15 years old and over\",115.57],[\"Foo
Soft drinks\",110.49],[\"Food available\",\"Soft drinks\",109.72],[\"Food available\",\"Soft drinks\",109.06],[\"Food available\",\"Ale, beer, stout and porter, population 15 years old and
,[\"Food available\",\"Ale, beer, stout and porter, population 15 years old and over\",105.95],[\"Food available\",\"Coffee\",105.81],[\"Food available\",\"Coffee\",105.6],[\"Food available
103.08],[\"Food available adjusted for losses\",\"Ale, beer, stout and porter, population 15 years old and over\",103.08],[\"Food available\",\"Coffee\",102.93],[\"Food available\",\"Soft d
opulation 15 years old and over\",100.6],[\"Food available adjusted for losses\",\"Ale, beer, stout and porter, population 15 years old and over\",100.44],[\"Food available\",\"Soft drinks\
"Food available adjusted for losses\",\"Ale, beer, stout and porter, population 15 years old and over\",97.89],[\"Food available\",\"Ale, beer, stout and porter, population 15 years old and
",96.41],[\"Food available\",\"Soft drinks\",96.38],[\"Food available adjusted for losses\",\"Soft drinks\",96.08],[\"Food available\",\"Ale, beer, stout and porter, population 15 years old
8],[\"Food available\",\"Coffee\",94.02],[\"Food available\",\"Coffee\",93.93],[\"Food available adjusted for losses\",\"Tea\",93.91],[\"Food available adjusted for losses\",\"Soft drinks\"
",\"Coffee\",92.18],[\"Food available adjusted for losses\",\"Ale, beer, stout and porter, population 15 years old and over\",92.05],[\"Food available\",\"Coffee\",92.04],[\"Food available\
, stout and porter, population 15 years old and over\",90.41],[\"Food available\",\"Coffee\",90.34],[\"Food available adjusted for losses\",\"Coffee\",90.25],[\"Food available adjusted for
lable\",\"Coffee\",88.61],[\"Food available\",\"Coffee\",88.51],[\"Food available adjusted for losses\",\"Coffee\",88.44],[\"Food available adjusted for losses\",\"Ale, beer, stout and port
e\",\"Ale, beer, stout and porter, population 15 years old and over\",87.25],[\"Food available adjusted for losses\",\"Coffee\",87.23],[\"Food available adjusted for losses\",\"Soft drinks\
5.72],[\"Food available adjusted for losses\",\"Coffee\",85.71],[\"Food available adjusted for losses\",\"Coffee\",85.66],[\"Food available adjusted for losses\",\"Coffee\",85.59],[\"Food a
Ale, beer, stout and porter, population 15 years old and over\",84.76],[\"Food available\",\"Tea\",84.55],[\"Food available\",\"Ale, beer, stout and porter, total population\",84.46],[\"Foo
\",83.63],[\"Food available\",\"Soft drinks\",83.62],[\"Food available\",\"Ale, beer, stout and porter, total population\",83.48],[\"Food available\",\"Ale, beer, stout and porter, total po
, beer, stout and porter, total population\",82.44],[\"Food available adjusted for losses\",\"Coffee\",82.42],[\"Food available adjusted for losses\",\"Coffee\",82.3],[\"Food available adju
offee\",80.94],[\"Food available adjusted for losses\",\"Coffee\",80.9],[\"Food available adjusted for losses\",\"Coffee\",80.89],[\"Food available\",\"Tea\",80.86],[\"Food available adjust
```

sortingOn_UOM_ID()# *function for sorting Values in ascending or descending order*

This function is used to sort the given dataset based on UOM_ID data value to sort I used sort function And to show Memory Usage during execution, I used **df2.info(memory_usage='deep')** which gives summary of a Data-Frame and returns None. And shown Sorted data in JSON format by converting csv to JSON.



```python
def sortingOn_UOM_ID():
    print("Author is Arish Kakadiya")
    newval = df.sort_values(['UOM_ID'],ascending=True)  # sorting algorithms is used to sort rows in ascending on VALUE 's values
    df2 = newval[['Food categories', 'Commodity','UOM_ID']]  # new dataframe declaration
    print(df2)
    print("\n Memory usage information in accurate number :\n")
    print(df2.info(memory_usage='deep')) # we'll set the memory_usage parameter to 'deep' to get an accurate number.


pd.set_option('max_colwidth', 800)
```



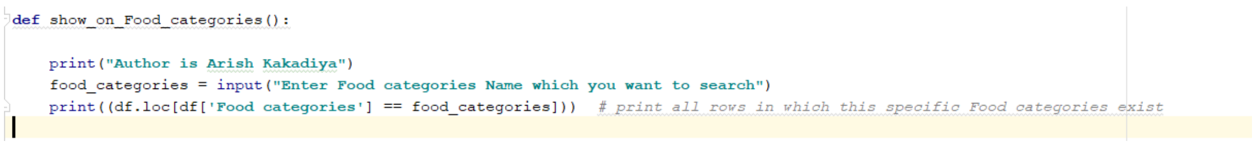
```
Author is Arish Kakadiya
                          Food categories                 Commodity  UOM_ID
0                          Food available               Wheat flour     194
19660  Food available adjusted for losses               Leeks fresh     194
19659  Food available adjusted for losses            Kohlrabi fresh     194
19658  Food available adjusted for losses              Garlic fresh     194
19657  Food available adjusted for losses           Eggplants fresh     194
19656  Food available adjusted for losses  Other edible roots fresh     194
19655  Food available adjusted for losses           Cucumbers fresh     194
19654  Food available adjusted for losses               Corn frozen     194
19653  Food available adjusted for losses               Corn canned     194
19652  Food available adjusted for losses                Corn fresh     194

 Memory usage information in accurate number :

<class 'pandas.core.frame.DataFrame'>
Int64Index: 30559 entries, 0 to 22395
Data columns (total 3 columns):
Food categories    30559 non-null object
Commodity          30559 non-null object
UOM_ID             30559 non-null int64
dtypes: int64(1), object(2)
memory usage: 5.0 MB
None
```

show_on_Food_categories()

#function for showing all rows having specific food category

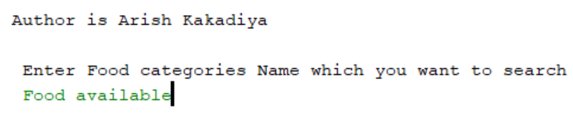
To display all the row having Specific searched Food Category , I used loc to select rows and column in Pandas Dataframe.

Ref.(https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.loc.html)

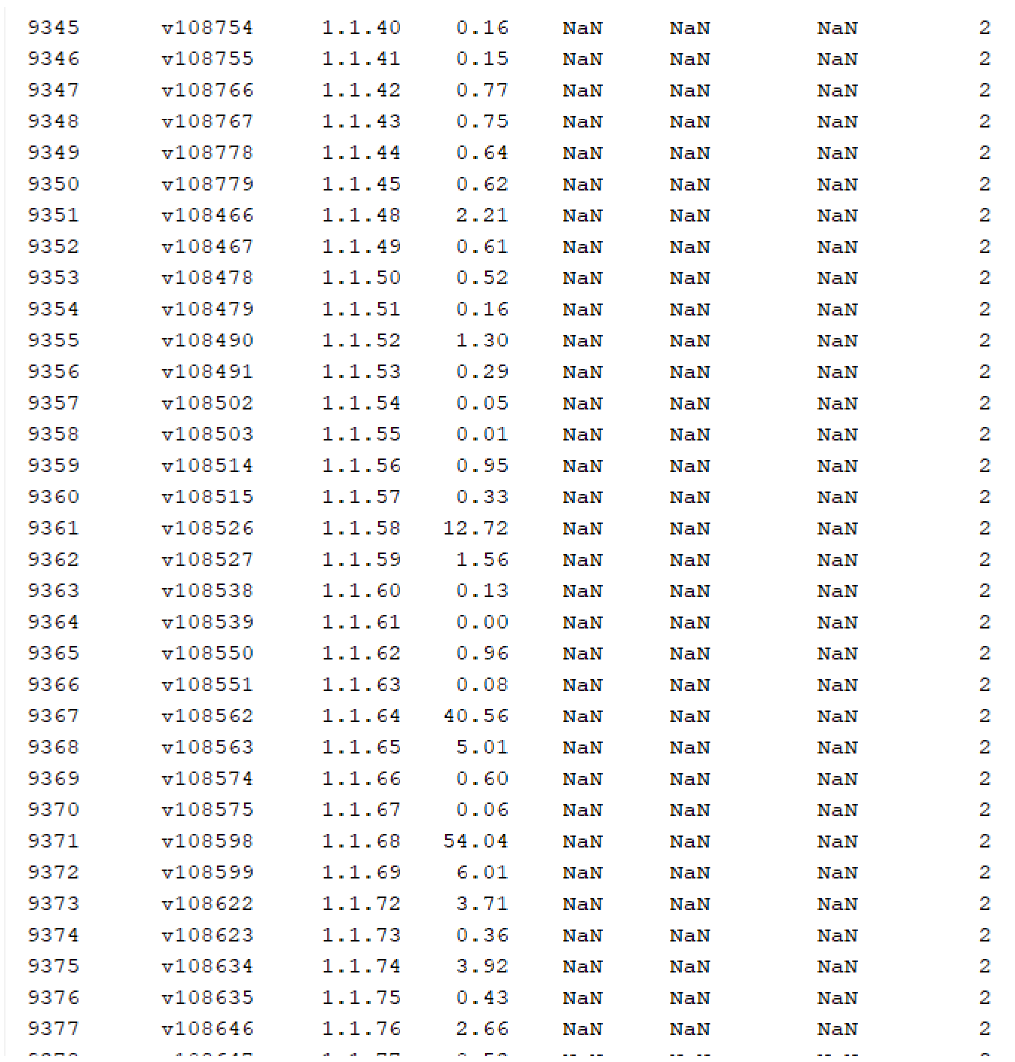


```python
def show_on_Food_categories():

    print("Author is Arish Kakadiya")
    food_categories = input("Enter Food categories Name which you want to search")
    print((df.loc[df['Food categories'] == food_categories]))  # print all rows in which this specific Food categories exist
```

Output:



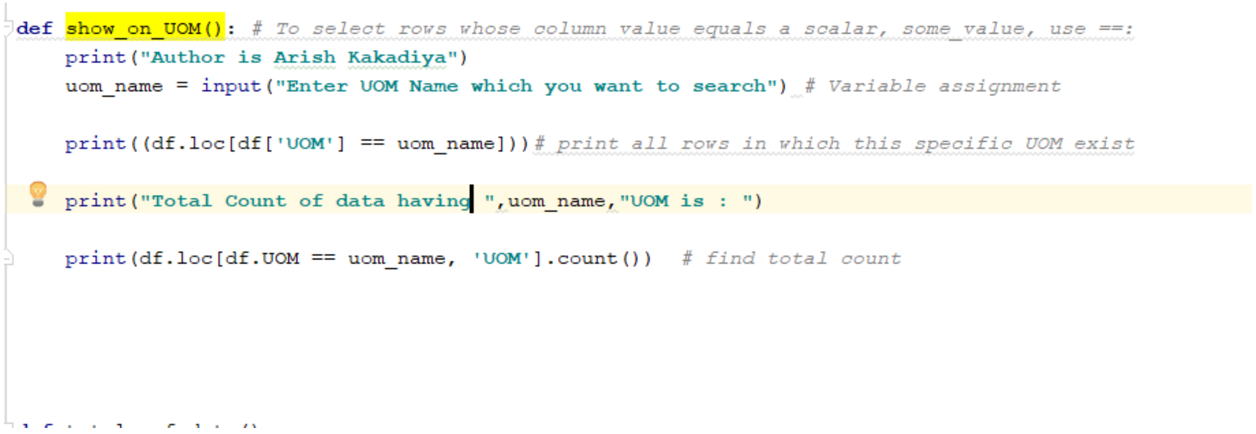
```
Author is Arish Kakadiya

 Enter Food categories Name which you want to search
 Food available
```



```
9345      v108754    1.1.40    0.16    NaN    NaN       NaN         2
9346      v108755    1.1.41    0.15    NaN    NaN       NaN         2
9347      v108766    1.1.42    0.77    NaN    NaN       NaN         2
9348      v108767    1.1.43    0.75    NaN    NaN       NaN         2
9349      v108778    1.1.44    0.64    NaN    NaN       NaN         2
9350      v108779    1.1.45    0.62    NaN    NaN       NaN         2
9351      v108466    1.1.48    2.21    NaN    NaN       NaN         2
9352      v108467    1.1.49    0.61    NaN    NaN       NaN         2
9353      v108478    1.1.50    0.52    NaN    NaN       NaN         2
9354      v108479    1.1.51    0.16    NaN    NaN       NaN         2
9355      v108490    1.1.52    1.30    NaN    NaN       NaN         2
9356      v108491    1.1.53    0.29    NaN    NaN       NaN         2
9357      v108502    1.1.54    0.05    NaN    NaN       NaN         2
9358      v108503    1.1.55    0.01    NaN    NaN       NaN         2
9359      v108514    1.1.56    0.95    NaN    NaN       NaN         2
9360      v108515    1.1.57    0.33    NaN    NaN       NaN         2
9361      v108526    1.1.58   12.72    NaN    NaN       NaN         2
9362      v108527    1.1.59    1.56    NaN    NaN       NaN         2
9363      v108538    1.1.60    0.13    NaN    NaN       NaN         2
9364      v108539    1.1.61    0.00    NaN    NaN       NaN         2
9365      v108550    1.1.62    0.96    NaN    NaN       NaN         2
9366      v108551    1.1.63    0.08    NaN    NaN       NaN         2
9367      v108562    1.1.64   40.56    NaN    NaN       NaN         2
9368      v108563    1.1.65    5.01    NaN    NaN       NaN         2
9369      v108574    1.1.66    0.60    NaN    NaN       NaN         2
9370      v108575    1.1.67    0.06    NaN    NaN       NaN         2
9371      v108598    1.1.68   54.04    NaN    NaN       NaN         2
9372      v108599    1.1.69    6.01    NaN    NaN       NaN         2
9373      v108622    1.1.72    3.71    NaN    NaN       NaN         2
9374      v108623    1.1.73    0.36    NaN    NaN       NaN         2
9375      v108634    1.1.74    3.92    NaN    NaN       NaN         2
9376      v108635    1.1.75    0.43    NaN    NaN       NaN         2
9377      v108646    1.1.76    2.66    NaN    NaN       NaN         2
```

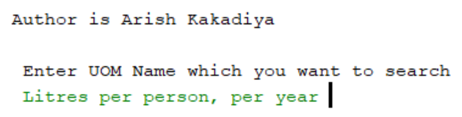
show_on_UOM()

To display data on searched UOM name and selected rows , column using .loc and counted total such rows using count function .

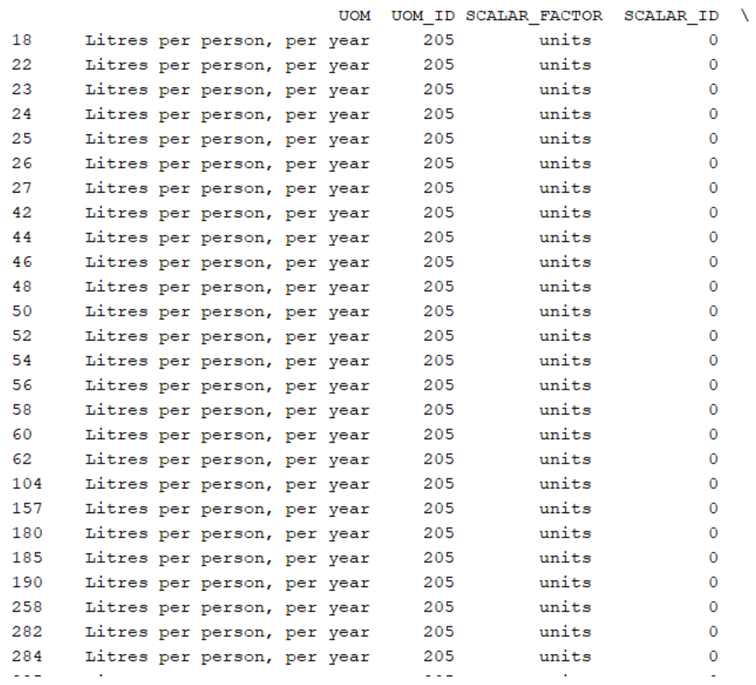


```
def show_on_UOM(): # To select rows whose column value equals a scalar, some_value, use ==:
    print("Author is Arish Kakadiya")
    uom_name = input("Enter UOM Name which you want to search") # Variable assignment

    print((df.loc[df['UOM'] == uom_name]))# print all rows in which this specific UOM exist

    print("Total Count of data having ",uom_name,"UOM is : ")

    print(df.loc[df.UOM == uom_name, 'UOM'].count())   # find total count
```

Output:



```
Author is Arish Kakadiya

 Enter UOM Name which you want to search
 Litres per person, per year
```



```
                                UOM  UOM_ID SCALAR_FACTOR  SCALAR_ID  \
18     Litres per person, per year     205         units          0
22     Litres per person, per year     205         units          0
23     Litres per person, per year     205         units          0
24     Litres per person, per year     205         units          0
25     Litres per person, per year     205         units          0
26     Litres per person, per year     205         units          0
27     Litres per person, per year     205         units          0
42     Litres per person, per year     205         units          0
44     Litres per person, per year     205         units          0
46     Litres per person, per year     205         units          0
48     Litres per person, per year     205         units          0
50     Litres per person, per year     205         units          0
52     Litres per person, per year     205         units          0
54     Litres per person, per year     205         units          0
56     Litres per person, per year     205         units          0
58     Litres per person, per year     205         units          0
60     Litres per person, per year     205         units          0
62     Litres per person, per year     205         units          0
104    Litres per person, per year     205         units          0
157    Litres per person, per year     205         units          0
180    Litres per person, per year     205         units          0
185    Litres per person, per year     205         units          0
190    Litres per person, per year     205         units          0
258    Litres per person, per year     205         units          0
282    Litres per person, per year     205         units          0
284    Litres per person, per year     205         units          0
```

# Multithreading to execute two given process

Multithreading is a way of achieving multitasking. In multithreading, the concept of threads is used.

To create a new thread, I create an object of Thread class. It takes following arguments:

target: the function to be executed by thread
args: the arguments to be passed to the target function
I created 2 threads with different target functions

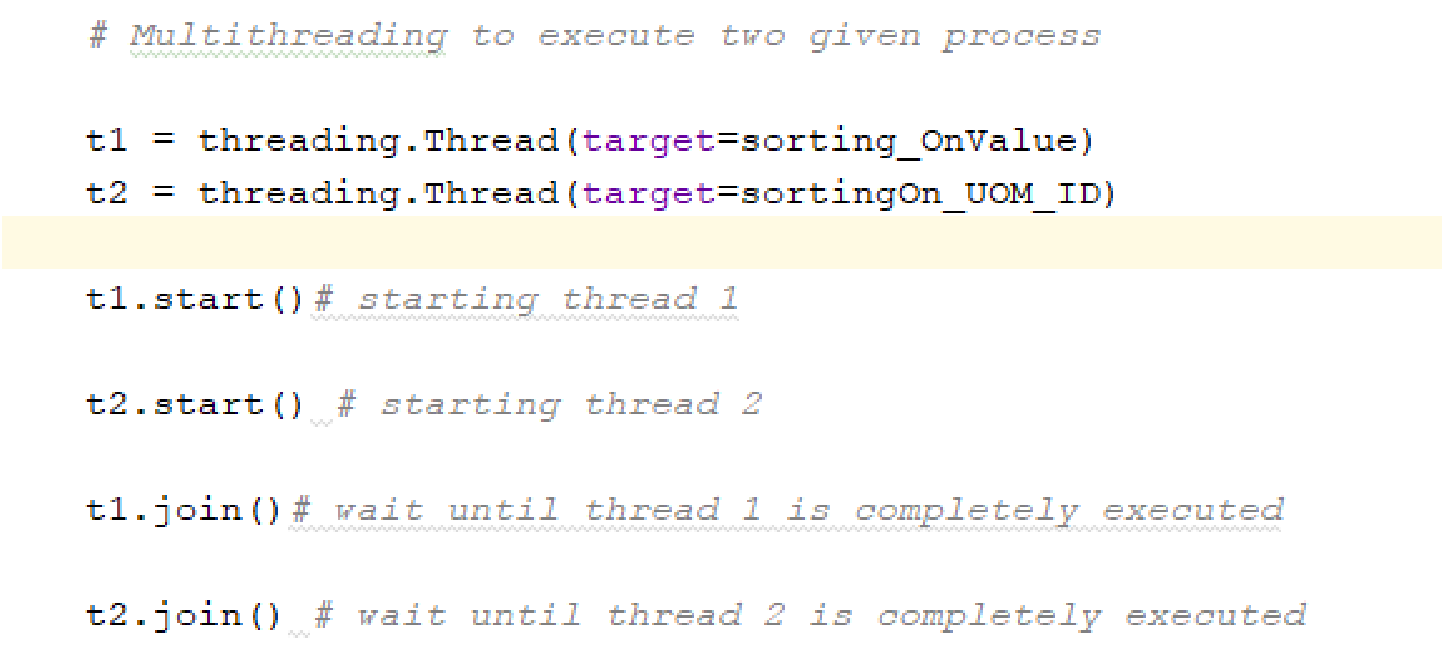
```
t1 = threading.Thread(target=sorting_OnValue)
t2 = threading.Thread(target=sortingOn_UOM_ID)
```

To start a thread, I will use start method of Thread class.

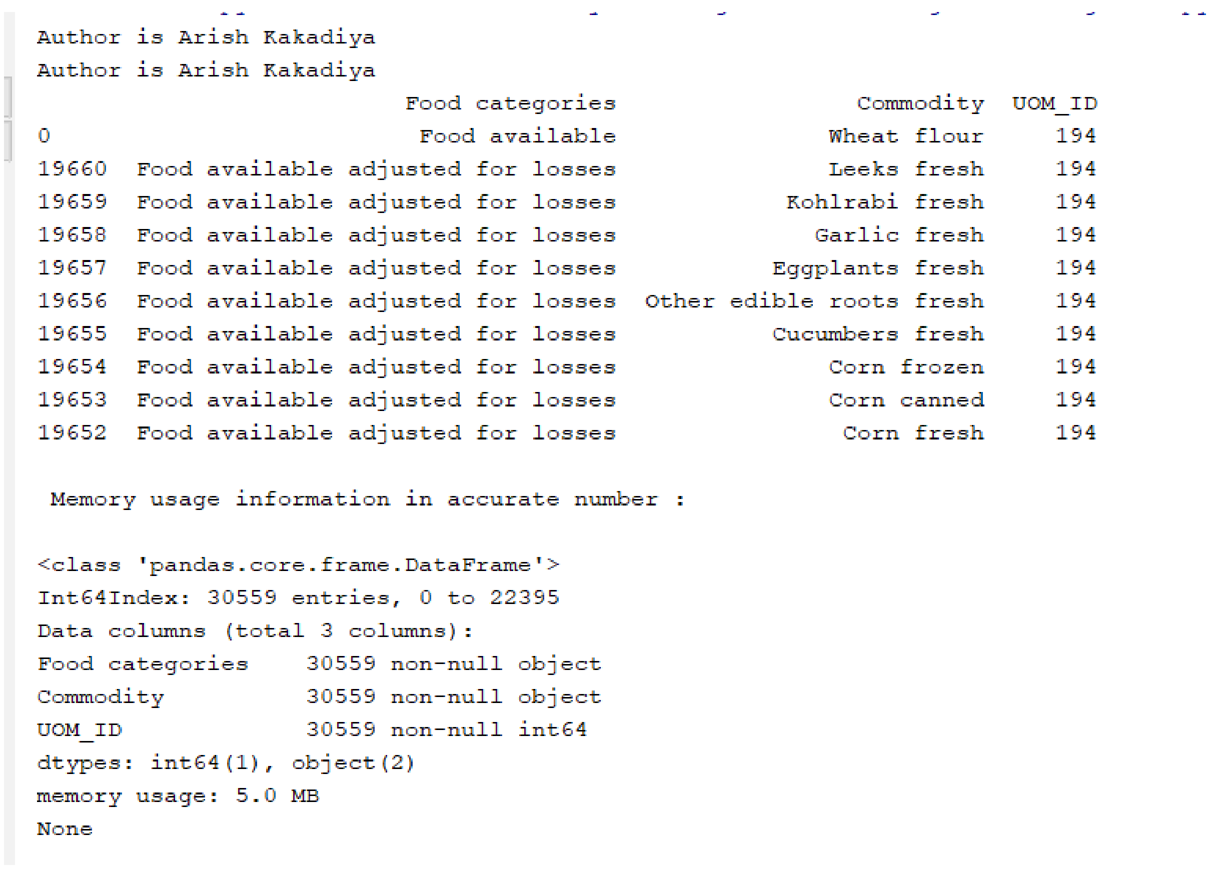
```
t1.start()# starting thread 1

t2.start() # starting thread 2
```

Once the threads start, the current program (you can think of it like a main thread) also keeps on executing. In order to stop execution of current program until a thread is complete, I use join method.

```
t1.join()
t2.join()
```



```
# Multithreading to execute two given process

t1 = threading.Thread(target=sorting_OnValue)
t2 = threading.Thread(target=sortingOn_UOM_ID)


t1.start()# starting thread 1

t2.start() # starting thread 2

t1.join()# wait until thread 1 is completely executed

t2.join() # wait until thread 2 is completely executed
```



```
Author is Arish Kakadiya
Author is Arish Kakadiya
                          Food categories                  Commodity  UOM_ID
0                          Food available                Wheat flour     194
19660  Food available adjusted for losses                Leeks fresh     194
19659  Food available adjusted for losses             Kohlrabi fresh     194
19658  Food available adjusted for losses               Garlic fresh     194
19657  Food available adjusted for losses            Eggplants fresh     194
19656  Food available adjusted for losses  Other edible roots fresh     194
19655  Food available adjusted for losses            Cucumbers fresh     194
19654  Food available adjusted for losses                Corn frozen     194
19653  Food available adjusted for losses                Corn canned     194
19652  Food available adjusted for losses                 Corn fresh     194

 Memory usage information in accurate number :

<class 'pandas.core.frame.DataFrame'>
Int64Index: 30559 entries, 0 to 22395
Data columns (total 3 columns):
Food categories    30559 non-null object
Commodity          30559 non-null object
UOM_ID             30559 non-null int64
dtypes: int64(1), object(2)
memory usage: 5.0 MB
None
```

End

###