

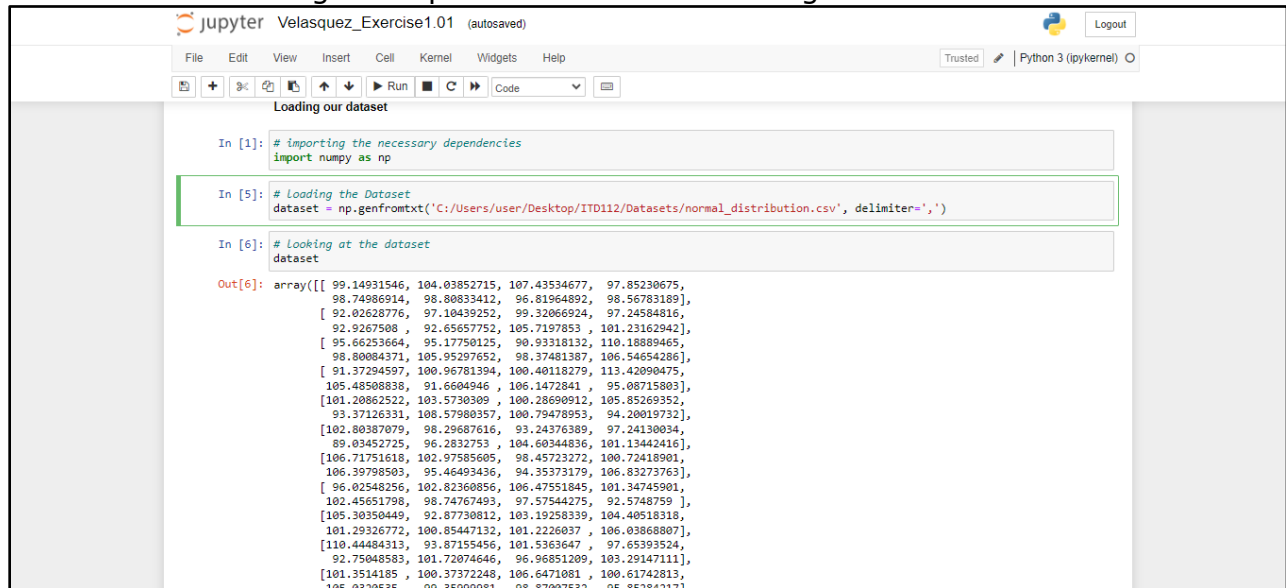
# Laboratory Exercises #1

Name Clint Joshua O. Velasquez

Perform the following exercises. Screenshot your output. Submit in pdf format.

Source Code Link: <https://github.com/kiyojiii/ITD112>

## Exercise 1.01: Loading a Sample Dataset and Calculating the Mean



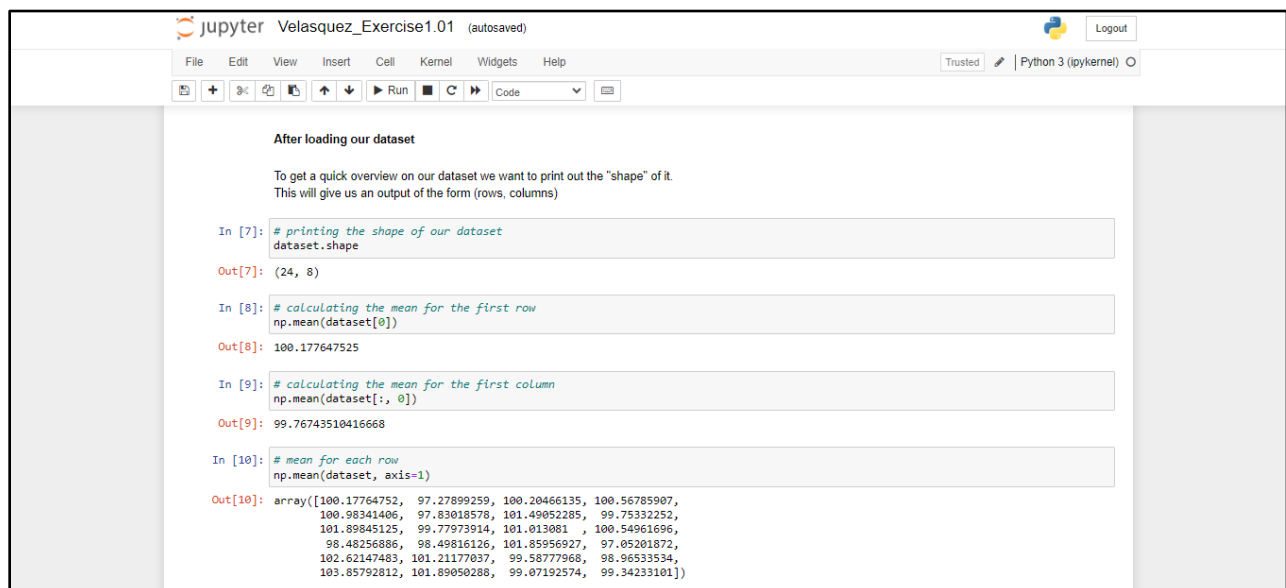
The screenshot shows a Jupyter Notebook titled 'Velasquez\_Exercise1.01'. The code in the first three cells is as follows:

```
In [1]: # importing the necessary dependencies
import numpy as np

In [5]: # Loading the Dataset
dataset = np.genfromtxt('C:/Users/user/Desktop/ITD112/Datasets/normal_distribution.csv', delimiter=',')

In [6]: # Looking at the dataset
dataset
```

The output of the third cell is a large array of 24 rows and 8 columns of numerical data.



The screenshot shows the continuation of the Jupyter Notebook. The text 'After loading our dataset' is followed by an explanation: 'To get a quick overview on our dataset we want to print out the "shape" of it. This will give us an output of the form (rows, columns)'. The code in the next three cells is as follows:

```
In [7]: # printing the shape of our dataset
dataset.shape

Out[7]: (24, 8)

In [8]: # calculating the mean for the first row
np.mean(dataset[0])

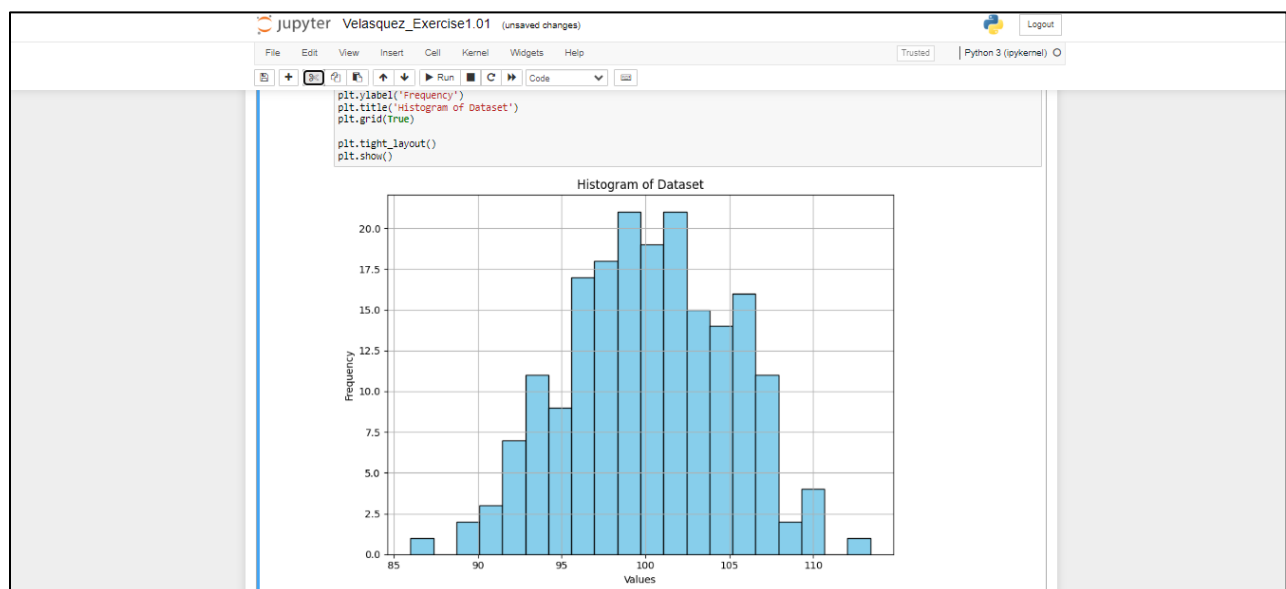
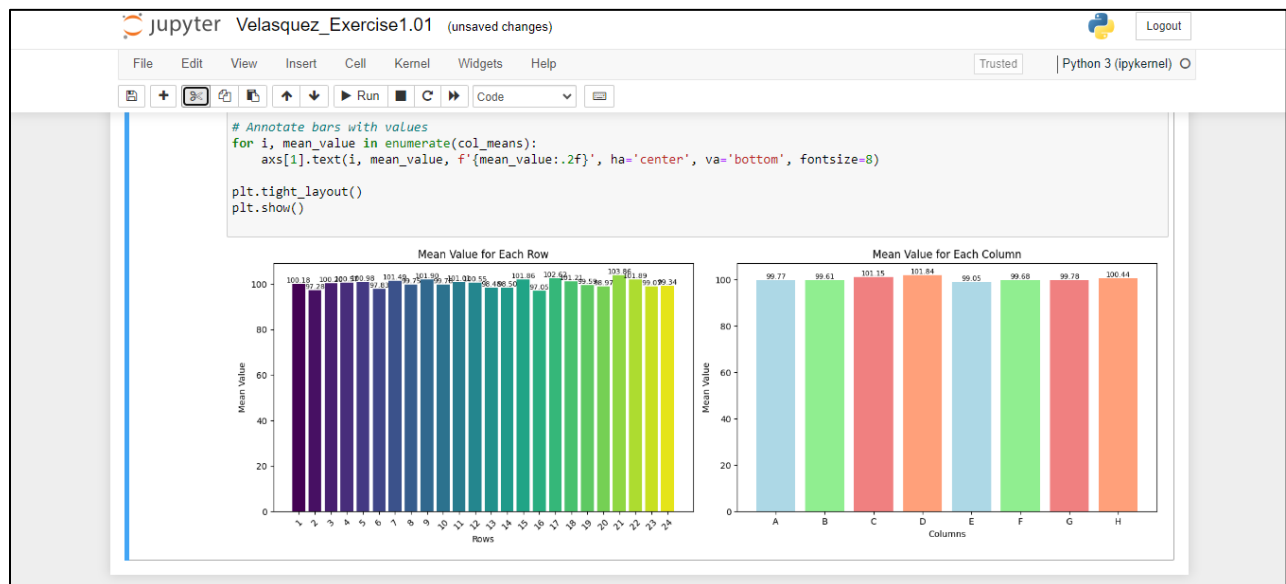
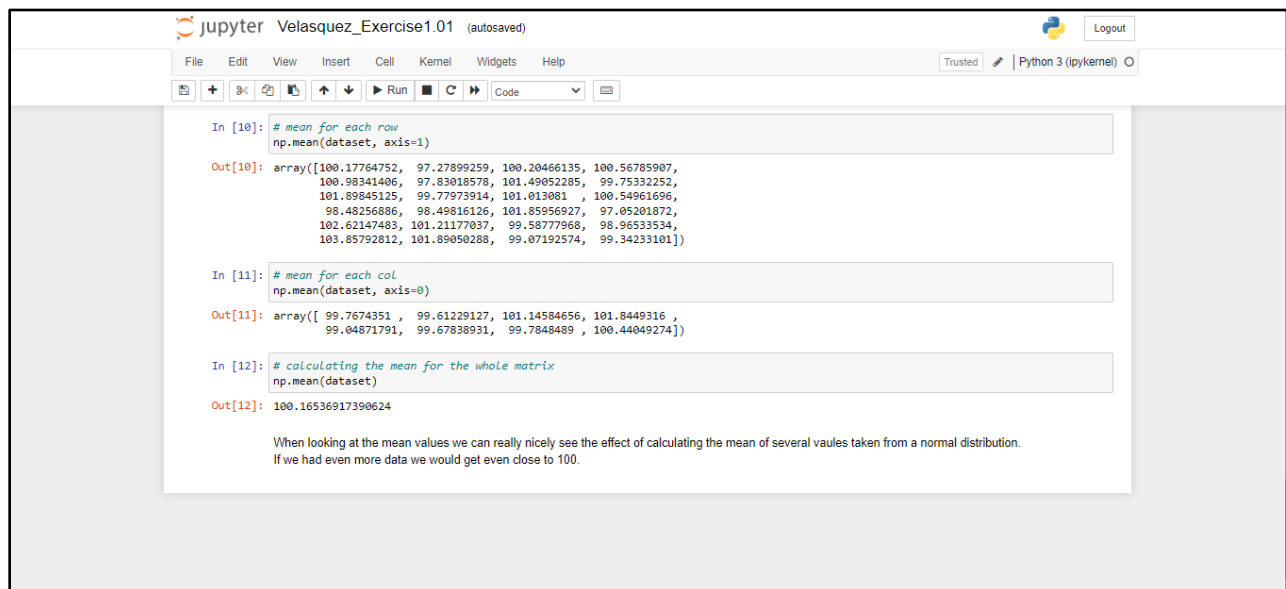
Out[8]: 100.177647525

In [9]: # calculating the mean for the first column
np.mean(dataset[:, 0])

Out[9]: 99.76743510416668

In [10]: # mean for each row
np.mean(dataset, axis=1)

Out[10]: array([100.17764752, 97.27899259, 100.20466135, 100.56785907,
100.98341406, 97.83018578, 101.49052285, 99.75332252,
101.89845125, 99.77973914, 101.013081, 100.54961696,
98.48256886, 98.49816126, 101.85956927, 97.05201872,
102.62147483, 101.21177037, 99.58777968, 98.96533534,
103.85792812, 101.89050288, 99.07192574, 99.34233101])
```



## Exercise 1.02: Indexing, Slicing, Splitting, and Iterating



**Jupyter Velasquez\_Exercise1.02** (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted | Python 3 (ipykernel)

**Splitting**

Our client's team only wants to use a small subset of the given dataset.  
Therefore we need to first split it into 3 equal pieces and then give them the first half of the first split.  
They sent us this drawing to show us what they need:

1, 2, 3, 4, 5, 6	=>	1, 2	3, 4	5, 6	=>	1, 2	
3, 2, 1, 5, 4, 6		3, 2	1, 5	4, 6		3, 2	=> 1, 2
5, 3, 1, 2, 4, 3		5, 3	1, 2	4, 3			3, 2
1, 2, 2, 4, 1, 5		1, 2	2, 4	1, 5		5, 3	
						1, 2	

**Note:**  
We are using a very small dataset here but imagine you have a huge amount of data and only want to look at a small subset of it to tweak your visualizations

```
In [10]: # splitting up our dataset horizontally on indices one third and two thirds
hor_splits = np.hsplit(dataset,(3))

In [11]: # splitting up our dataset vertically on index 2
ver_splits = np.vsplit(hor_splits[0],(2))

In [12]: # requested subsection of our dataset which has only half the amount of rows and only a third of the columns
print("Dataset", dataset.shape)
print("Subset", ver_splits[0].shape)

Dataset (24, 9)
Subset (12, 3)
```

**Jupyter Velasquez\_Exercise1.02** (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3 (ipykernel)

list

However, they want to also now the position in the dataset itself.

They send you this piece of code and tell you that it's not working as mentioned.  
Come up with the right solution for their needs using the `ndenumerate` method.

```
In [13]: # iterating over whole datamaiset (each value in each row)
curr_index = 0
for x in np.nditer(dataset):
    print(x, curr_index)
    curr_index += 1
```

```
99.14931546 0
104.03852715 1
107.43534677 2
97.85230675 3
98.74986914 4
98.80833412 5
96.81964892 6
98.56783189 7
101.34745901 8
92.02628776 9
97.10439252 10
99.32066924 11
97.24584816 12
92.9267508 13
92.65657752 14
105.7197853 15
101.23162942 16
93.87155456 17
95.66253664 18
```

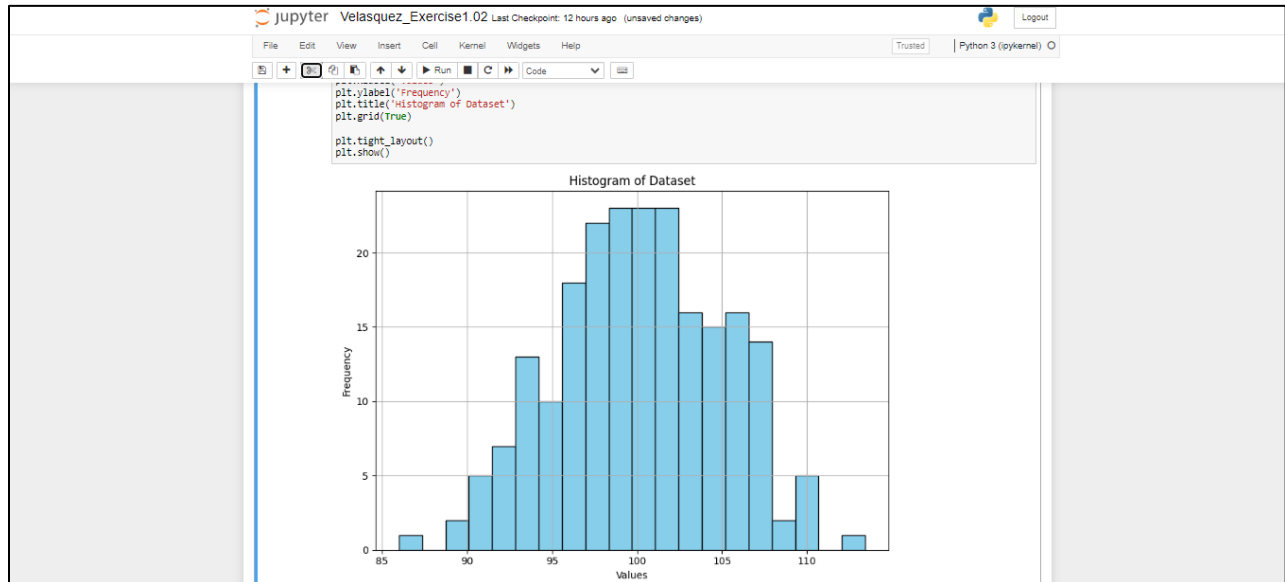
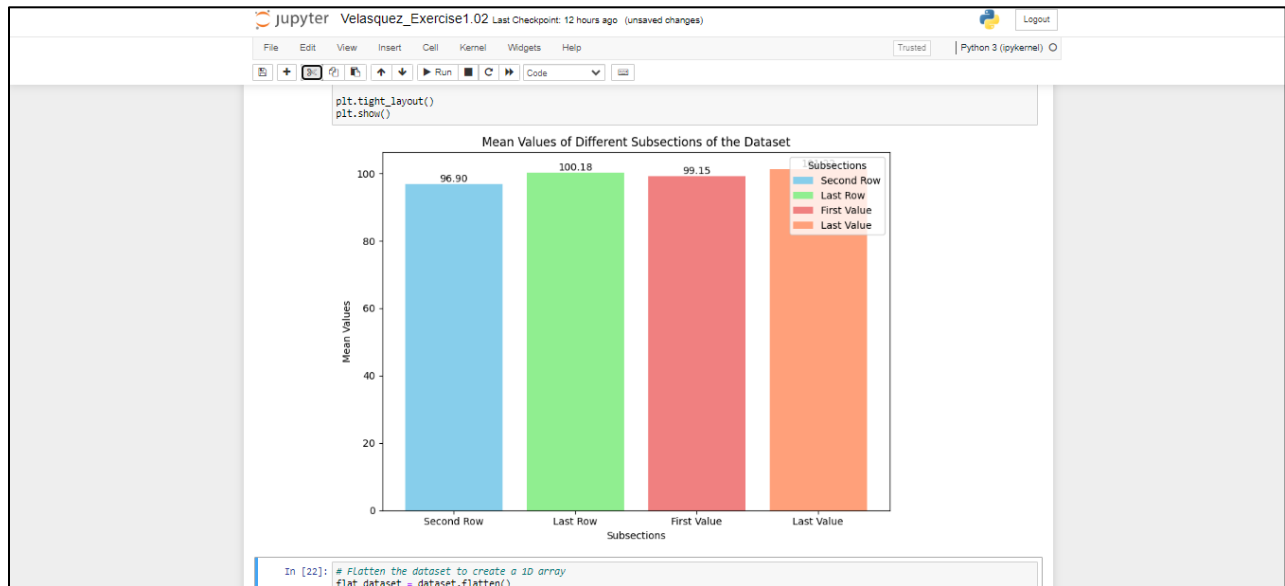
**Jupyter Velasquez\_Exercise1.02** (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3 (ipykernel)

```
107.22482426 26
91.37294597 27
100.96781394 28
100.40118279 29
113.42090475 30
105.48508838 31
```

```
In [14]: # iterating over whole dataset with indices matching the position in the dataset
for index, value in np.ndenumerate(dataset):
    print(index, value)
```

```
(0, 0) 99.14931546
(0, 1) 104.03852715
(0, 2) 107.43534677
(0, 3) 97.85230675
(0, 4) 98.74986914
(0, 5) 98.80833412
(0, 6) 96.81964892
(0, 7) 98.56783189
(0, 8) 101.34745901
(1, 0) 92.02628776
(1, 1) 97.10439252
(1, 2) 99.32066924
(1, 3) 97.24584816
(1, 4) 92.9267508
(1, 5) 92.65657752
(1, 6) 105.7197853
(1, 7) 101.23162942
(1, 8) 93.87155456
(2, 0) 95.66253664
```



### Exercise 1.03: Filtering, Sorting, Combining, and Reshaping

Jupyter Velasquez\_Exercise1.03 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

#### Exercise 1.03: Filtering, Sorting, Combining, and Reshaping

Following up on the last exercise, we are asked to deliver some more complex operations. We will, therefore, continue to work with the same dataset, our `normal_distribution.csv`.

Loading the dataset

```
In [1]: # importing the necessary dependencies
import numpy as np

In [2]: # Loading the Dataset
dataset = np.genfromtxt('C:/Users/user/Desktop/ITD112/Datasets/normal_distributionSplittable.csv', delimiter=',')
dataset

Out[2]: array([[ 99.14931546, 104.03852715, 107.43534677,  97.85230675,
  98.74986914,  98.80833412,  96.81964892,  98.56783189,
  101.34745901],
 [ 92.02628776,  97.10439252,  99.32066924,  97.24584816,
  92.9267508 ,  92.65657752, 105.7197853 , 101.23162942,
  93.87155456],
 [ 95.66253664,  95.17750125,  90.93318132, 110.18889465,
  98.80084371, 105.95297652,  98.37481387, 106.54654286,
  107.22482426],
 [ 91.37294597, 100.96781394, 100.40118279, 113.42090475,
  105.48508838,  91.6604946 , 106.1472841 ,  95.08715803,
  103.40412146],
 [101.20862522, 103.5730309 , 100.28690912, 105.85269352,
```

jupyter Velasquez\_Exercise1.03 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3 (ipykernel)

Filtering

To get better insights into our dataset, we want to only look at the value that fulfills certain conditions. Our client reaches out to us and asks us to provide lists of values that fulfills these conditions, filter down our dataset to contain only:

- all values greater than 105 (>105)
- all values that are between 90 and 95 (>90 and <95)
- the indices of all values that have a delta of less than 1 to 100 ( $x-100 < 1$ )

In [3]:

# values that are greater than 105  
vals\_greater\_five = dataset[dataset > 105]  
vals\_greater\_five

Out[3]:

array([[107.43534677, 105.7197853 , 110.18889465, 105.95297652,  
106.54654286, 107.22482426, 113.42090475, 105.48508838,  
106.1472841 , 105.85269352, 108.57980357, 106.71751618,  
106.39798503, 106.83273763, 106.47551845, 105.30350449,  
106.03688807, 110.44484313, 106.6471081 , 105.0320535 ,  
107.02874163, 105.07475277, 106.57364584, 107.22482426,  
107.19119932, 108.09423367, 109.40523174, 106.11454989,  
106.57052697, 105.13668343, 105.37011896, 110.44484313,  
105.86078488, 106.89005002, 106.57364584, 107.40064604,  
106.38276709, 106.46476468, 110.43976681, 105.02389857,  
106.05042487, 106.89005002])

In [4]:

# values that are between 90 and 95  
vals\_between\_90\_95 = np.extract((dataset > 90) & (dataset < 95), dataset)  
vals\_between\_90\_95

Out[4]:

106.05042487, 106.89005002])

In [4]:

# values that are between 90 and 95  
vals\_between\_90\_95 = np.extract((dataset > 90) & (dataset < 95), dataset)  
vals\_between\_90\_95

Out[4]:

array([[92.02628776, 92.9267508 , 92.65657752, 93.87155456, 90.93318132,  
91.37294597, 91.6604946 , 93.37126331, 94.20019732, 93.24376389,  
94.35373179, 92.5748759 , 91.37294597, 92.87730812, 93.87155456,  
92.75048583, 93.97853495, 91.32093303, 92.0188226 , 93.18884302,  
93.83969256, 94.5081787 , 94.59300658, 93.04610867, 91.6779221 ,  
91.37294597, 94.76253572, 94.57421727, 94.11176915, 93.97853495])

Note:

Conditional filtering can be done either using the brackets syntax or NumPys `extract` method

In [5]:

# indices of values that have a delta of less than 1 to 100  
rows, cols = np.where(abs(dataset - 100) < 1)  
  
one\_away\_indices = [[rows[index], cols[index]] for (index, \_) in np.ndenumerate(rows)]  
one\_away\_indices

Out[5]:

[[0, 0],  
[1, 2],  
[3, 1],  
[3, 2],  
[4, 2],  
[4, 6],  
[6, 3],  
[6, 4]]

jupyter Velasquez\_Exercise1.03 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

Sorting

They also want to experiment with some more plotting techniques so they ask you to also deliver these datasets. Sort our dataset with:

- values sorted in ascending order for each row
- values sorted in ascending order for each column
- the matrix of indices indicating the position in a sorted list of each value

[3, 1, 2, 5, 4] => [1, 2, 0, 4, 3]

In [6]:

# values sorted for each row  
row\_sorted = np.sort(dataset)  
row\_sorted

Out[6]:

array([[ 96.81964892, 97.85230675, 98.56783189, 98.74986914,  
 98.80833412, 99.14931546, 101.34745901, 104.03852715,  
107.43534677],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331, 94.20019732, 96.10020311, 100.28690912,  
100.79478953, 101.20862522, 103.5730309 , 105.85269352,  
108.57980357],  
[ 92.02628776, 92.65657752, 92.9267508 , 93.87155456,  
 97.10439252, 97.24584816, 99.32066924, 101.23162942,  
105.7197853 ],  
[ 90.93318132, 95.17750125, 95.66253664, 98.37481387,  
 98.80084371, 105.95297652, 106.54654286, 107.22482426,  
110.18889465],  
[ 91.37294597, 91.6604946 , 95.08715803, 100.40118279,  
100.96781394, 103.40412146, 105.48508838, 106.1472841 ,  
113.42090475],  
[ 93.37126331,

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

In [7]: # values sorted for each column
col_sorted = np.sort(dataset, axis=0)
col_sorted

Out[7]: array([[ 91.37294597,  88.80221141,  90.93318132,  93.18884302,
  85.98839623,  91.6604946 ,  91.32093303,  92.5748759 ,
  91.37294597],
 [ 92.02628776,  91.6779221 ,  93.24376389,  94.59300658,
  89.03452725,  92.65657752,  93.04610867,  94.20019732,
  91.37294597],
 [ 94.11176915,  92.0108226 ,  93.83969256,  96.74630281,
  92.75048583,  95.19184343,  94.35373179,  94.76253572,
  93.87155456],
 [ 95.65982034,  92.87730812,  94.5081787 ,  97.24130034,
  92.9267508 ,  95.46493436,  96.50342927,  95.08715803,
  93.97853495],
 [ 95.66253664,  93.87155456,  97.75887636,  97.24584816,
  93.37126331,  95.62359311,  96.81964892,  95.85284217,
  95.19184343],
 [ 96.02548256,  94.57421727,  98.45723272,  97.62787811,
  93.97853495,  96.2832753 ,  96.89244283,  97.59572169,
  96.10020311],
 [ 96.10020311,  95.17750125,  99.32066924,  97.65393524,
  95.93799169,  96.34622848,  96.96851209,  98.00253006,
  97.10439252],
 [ 96.76814836,  96.59385486,  99.57859892,  97.85230675,
  98.29243952,  96.5937781 ,  97.57544275,  98.07122664,
  97.24130034],
 [ 96.78266211,  97.10439252, 100.28690912,  99.4889538 ,
  98.61325194,  98.65912661,  97.94046856,  98.56783189,
  97.62787811],
 [ 97.21315663,  98.29687616, 100.40118279,  99.95827854,
```

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

110.44484313]])

In [8]: # sorted indices of positions for first row
index_sorted = np.argsort(dataset[0])
dataset[0][index_sorted]

Out[8]: array([ 96.81964892,  97.85230675,  98.56783189,  98.74986914,
 98.80833412,  99.14931546, 101.34745901, 104.03852715,
107.43534677])

Combining

After finishing their visualization and doing ask you to deliver a way they can incrementally add the split parts of the dataset to make sure it works with every subset, too.
Create a combined dataset by:

• adding the second half of the first column
• adding the second column
• adding the third and last separate column

In [9]: # split up dataset from exercise02
thirds = np.hsplit(dataset, (3))
halfed_first = np.vsplit(thirds[0], (2))

# this is the part we've sent the client in exercise02
halfed_first[0]
```

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

In [9]: # split up dataset from exercise02
thirds = np.hsplit(dataset, (3))
halfed_first = np.vsplit(thirds[0], (2))

# this is the part we've sent the client in exercise02
halfed_first[0]

Out[9]: array([[ 99.14931546, 104.03852715, 107.43534677],
 [ 92.02628776,  97.10439252,  99.32066924],
 [ 95.66253664,  95.17750125,  90.93318132],
 [ 91.37294597, 100.96781394, 100.40118279],
 [101.20862522, 103.5730309 , 100.28690912],
 [102.80387079,  98.29687616,  93.24376389],
 [106.71751618, 102.97585605,  98.45723272],
 [ 96.02548256, 102.82360856, 106.47551845],
 [105.30350449,  92.87730812, 103.19258339],
 [110.44484313,  93.87155456, 101.5363647 ],
 [101.3514185 , 100.37372248, 106.6471081 ],
 [ 97.21315663, 107.02874163, 102.17642112]])

In [10]: # adding the second half of the first column to the data
first_col = np.vstack([halfed_first[0], halfed_first[1]])
first_col

Out[10]: array([[ 99.14931546, 104.03852715, 107.43534677],
 [ 92.02628776,  97.10439252,  99.32066924],
 [ 95.66253664,  95.17750125,  90.93318132],
 [ 91.37294597, 100.96781394, 100.40118279],
 [101.20862522, 103.5730309 , 100.28690912],
 [102.80387079,  98.29687616,  93.24376389],
 [106.71751618, 102.97585605,  98.45723272],
 [ 96.02548256, 102.82360856, 106.47551845],
```

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

[106.89005002, 106.57364584, 102.26648279],
[99.80873105, 101.63973121, 106.46476468],
[96.10020311, 94.57421727, 100.80409326],
[94.11176915, 99.62387832, 104.51786419]]

In [11]: # adding the second column to our combined dataset
first_second_col = np.hstack([first_col, thirds[1]])
first_second_col

Out[11]: array([[ 99.14931546, 104.03852715, 107.43534677, 97.85230675,
 98.74986914, 98.80833412],
 [ 92.02628776, 97.10439252, 99.32066924, 97.24584816,
 92.9267508 , 92.65657752],
 [ 95.66253664, 95.17750125, 90.93318132, 110.18889465,
 98.80084371, 105.95297652],
 [ 91.37294597, 100.96781394, 100.40118279, 113.42090475,
 105.48508838, 91.6604946 ],
 [101.20862522, 103.5730309 , 100.28690912, 105.85269352,
 93.37126331, 108.57980357],
 [102.80387079, 98.29687616, 93.24376389, 97.24130034,
 89.03452725, 96.2832753 ],
 [106.71751618, 102.97585605, 98.45723272, 100.72418901,
 106.39798503, 95.46493436],
 [ 96.02548256, 102.82360856, 106.47551845, 101.34745901,
 102.45651798, 98.74767493],
 [105.30350449, 92.87730812, 103.19258339, 104.40518318,
 101.29326772, 108.85447132],
 [110.44484313, 93.87155456, 101.5363647 , 97.65393524,
 92.75048583, 101.72074646],
 [101.3514185 , 100.37372248, 106.6471081 , 100.61742813,
 105.0320535 , 99.35999981],
 [ 97.21315663, 107.02874163, 102.17642112, 96.74630281,
 95.93799169, 102.62384733],
```

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

[ 96.10020311, 94.57421727, 100.80409326, 105.02389857,
 98.61325194, 95.62359311],
[ 94.11176915, 99.62387832, 104.51786419, 97.62787811,
 93.97853495, 98.75108352]]

In [12]: # adding the third column to our combined dataset
full_data = np.hstack([first_second_col, thirds[2]])
full_data

Out[12]: array([[ 99.14931546, 104.03852715, 107.43534677, 97.85230675,
 98.74986914, 98.80833412, 96.81964892, 98.56783189,
 101.34745901],
 [ 92.02628776, 97.10439252, 99.32066924, 97.24584816,
 92.9267508 , 92.65657752, 105.7197853 , 101.23162942,
 93.87155456],
 [ 95.66253664, 95.17750125, 90.93318132, 110.18889465,
 98.80084371, 105.95297652, 98.37481387, 106.54654286,
 107.22482426],
 [ 91.37294597, 100.96781394, 100.40118279, 113.42090475,
 105.48508838, 91.6604946 , 106.1472841 , 95.08715803,
 103.40412146],
 [101.20862522, 103.5730309 , 100.28690912, 105.85269352,
 93.37126331, 108.57980357, 100.79478953, 94.20019732,
 96.10020311],
 [102.80387079, 98.29687616, 93.24376389, 97.24130034,
 89.03452725, 96.2832753 , 104.60344836, 101.13442416,
 97.62787811],
 [106.71751618, 102.97585605, 98.45723272, 100.72418901,
 106.39798503, 95.46493436, 94.35373179, 106.83273763,
 100.07721494],
 [ 96.02548256, 102.82360856, 106.47551845, 101.34745901,
 102.45651798, 98.74767493, 97.57544275, 92.5748759 ,
 91.37294597],
 [105.30350449, 92.87730812, 103.19258339, 104.40518318,
```

```
jupyter Velasquez_Exercise1.03 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3 (ipykernel)

Reshaping

For their internal AI algorithms, they need the dataset in a reshaped manner that reduces the number of columns.
They asked us to deliver the whole dataset in the following shapes. Create new datasets that are:

• reshaped in a one-dimensional list with all values
• reshaped in a matrix with only 2 columns

In [13]: # reshaping to a list of values
single_list = np.reshape(dataset, (1, -1))
single_list

Out[13]: array([[ 99.14931546, 104.03852715, 107.43534677, 97.85230675,
 98.74986914, 98.80833412, 96.81964892, 98.56783189,
 101.34745901, 92.02628776, 97.10439252, 99.32066924,
 97.24584816, 92.9267508 , 92.65657752, 105.7197853 ,
 101.23162942, 93.87155456, 95.66253664, 95.17750125,
 90.93318132, 110.18889465, 98.80084371, 105.95297652,
 98.37481387, 106.54654286, 107.22482426, 91.37294597,
 100.96781394, 100.40118279, 113.42090475, 105.48508838,
 91.6604946 , 106.1472841 , 95.08715803, 103.40412146,
 101.20862522, 103.5730309 , 100.28690912, 105.85269352,
 93.37126331, 108.57980357, 100.79478953, 94.20019732,
 96.10020311, 102.80387079, 98.29687616, 93.24376389,
 97.24130034, 89.03452725, 96.2832753 , 104.60344836,
 101.13442416, 97.62787811, 106.71751618, 102.97585605,
 98.45723272, 100.72418901, 106.39798503, 95.46493436,
 94.35373179, 106.83273763, 100.07721494, 96.02548256,
 102.82360856, 106.47551845, 101.34745901, 102.45651798,
 98.74767493, 97.57544275, 92.5748759 , 91.37294597,
```

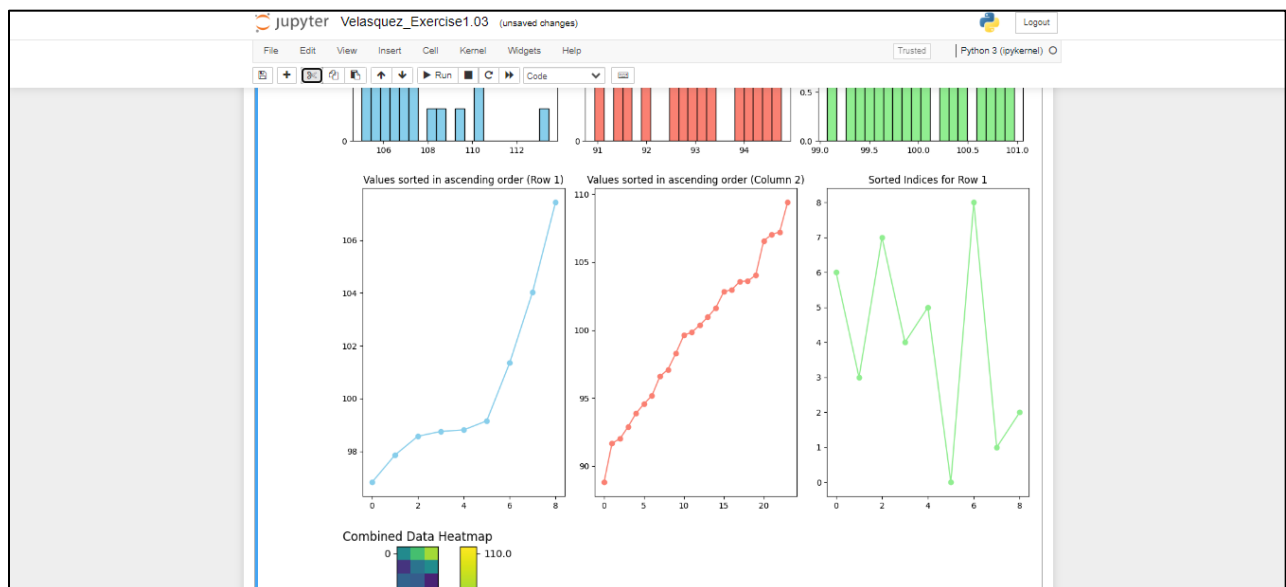
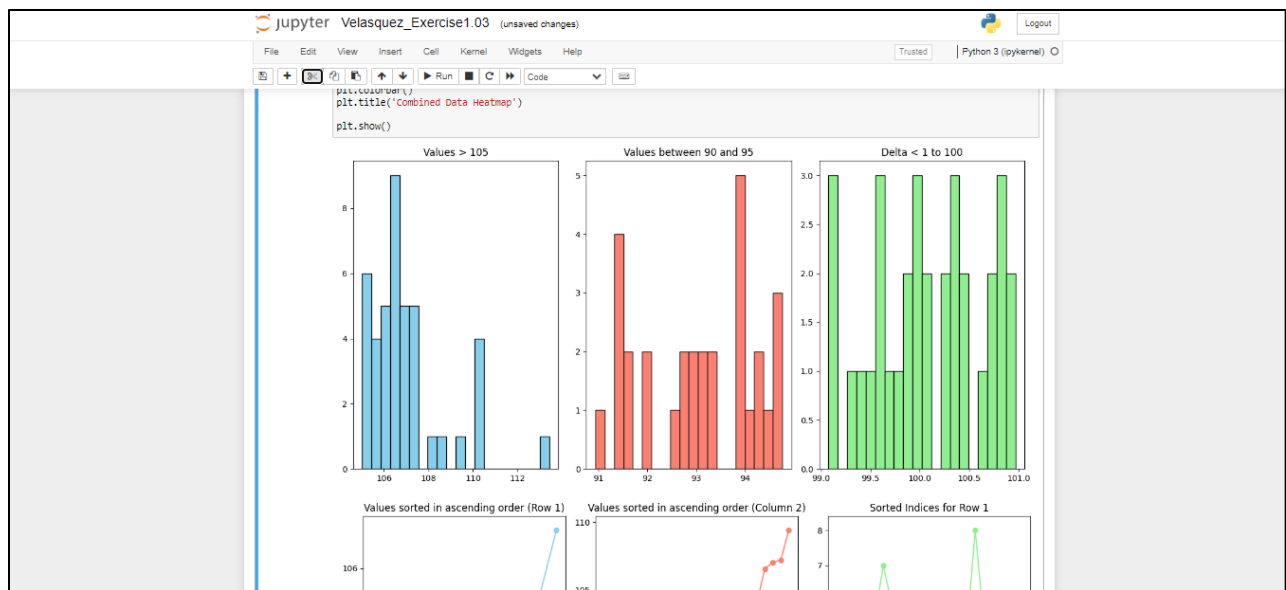


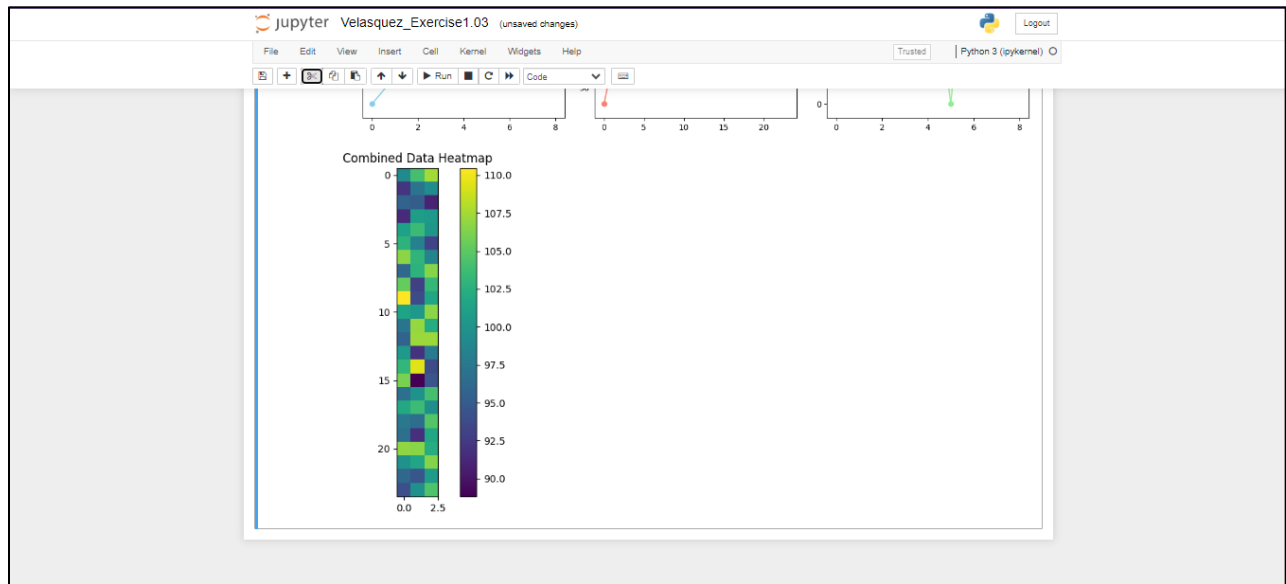
jupyter Velasquez\_Exercise1.03 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [14]: # reshaping to a matrix with two columns
two_col_dataset = dataset.reshape(-1, 2)
two_col_dataset
```

Note:  
-1 in the dimension definition means that it figures out the other dimension on its own





## Exercise 1.04: Loading a Sample Dataset and Calculating the Mean

Jupyter Velasquez\_Exercise1.04 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [3]: `# Looking at the dataset  
dataset.head()`

Out[3]:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966 ...	2007	2008
Aruba	ABW	Population density (people per sq. km of land ...	EN.POPDNST	NaN	307.972222	312.366667	314.983333	316.827778	318.666667	320.622222 ...	562.322222	563.011111 563
Andorra	AND	Population density (people per sq. km of land ...	EN.POPDNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213	41.800000 ...	180.591489	182.161702 181
Afghanistan	AFG	Population density (people per sq. km of land ...	EN.POPDNST	NaN	14.038148	14.312061	14.599692	14.901579	15.218206	15.545203 ...	39.637202	40.634655 41
Angola	AGO	Population density (people per sq. km of land ...	EN.POPDNST	NaN	4.305195	4.384299	4.464433	4.544558	4.624228	4.703271 ...	15.387749	15.915819 16
Albania	ALB	Population density (people per sq. km of land ...	EN.POPDNST	NaN	60.576642	62.456898	64.329234	66.209307	68.058066	69.874927 ...	108.394781	107.566204 106

5 rows x 60 columns

Jupyter Velasquez\_Exercise1.04 (autosaved) Python 3 (ipykernel)

```
In [4]: # printing the shape of our dataset
dataset.shape

Out[4]: (264, 60)

In [5]: # calculating the mean for 1961 column
dataset["1961"].mean()

Out[5]: 176.91514132840555

In [6]: # calculating the mean for 2015 column
dataset["2015"].mean()

Out[6]: 368.70660104001837
```

**Note:**  
Only by comparing the overall mean of the two years, 1961 and 2015, we can already see that the mean population density more than doubled in this time range.

```
In [10]: # Assuming 'dataset' is your DataFrame, select only valid numeric columns
numeric_columns = dataset.select_dtypes(include='number')

# Calculate the mean of selected numeric columns
mean_values = numeric_columns.mean(axis=1)

# Display the mean values for the first 10 rows
print(mean_values.head(10))
```

Country Name	mean
Aruba	413.944949
Andorra	106.838839
Afghanistan	25.373379
Angola	9.649583
Albania	99.159197
Arab World	16.118586
United Arab Emirates	31.321721
Argentina	11.634028
Armenia	103.415539
American Samoa	211.855636

Jupyter Velasquez\_Exercise1.04 (autosaved) Python 3 (ipykernel)

```
In [10]: # Assuming 'dataset' is your DataFrame, select only valid numeric columns
numeric_columns = dataset.select_dtypes(include='number')

# Calculate the mean of selected numeric columns
mean_values = numeric_columns.mean(axis=1)

# Display the mean values for the first 10 rows
print(mean_values.head(10))
```

Country Name	mean
Aruba	413.944949
Andorra	106.838839
Afghanistan	25.373379
Angola	9.649583
Albania	99.159197
Arab World	16.118586
United Arab Emirates	31.321721
Argentina	11.634028
Armenia	103.415539
American Samoa	211.855636

```
In [7]: # mean for each country (row)
dataset.mean(axis=1).head(10)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2920\51040833.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

```
dataset.mean(axis=1).head(10)
```

```
Out[7]: Country Name
Aruba      413.944949
Andorra    106.838839
```

Jupyter Velasquez\_Exercise1.04 (autosaved) Python 3 (ipykernel)

```
In [13]: # Assuming 'dataset' is your DataFrame, select only valid numeric columns
numeric_columns = dataset.select_dtypes(include='number')

# Calculate the mean of selected numeric columns
mean_values = numeric_columns.mean(axis=0)

# Display the mean values for the first 10 rows
print(mean_values.tail(10))
```

	mean
2007	331.995474
2008	338.688417
2009	343.649206
2010	347.967029
2011	351.942027
2012	357.787305
2013	360.985726
2014	364.849194
2015	368.706601
2016	NaN

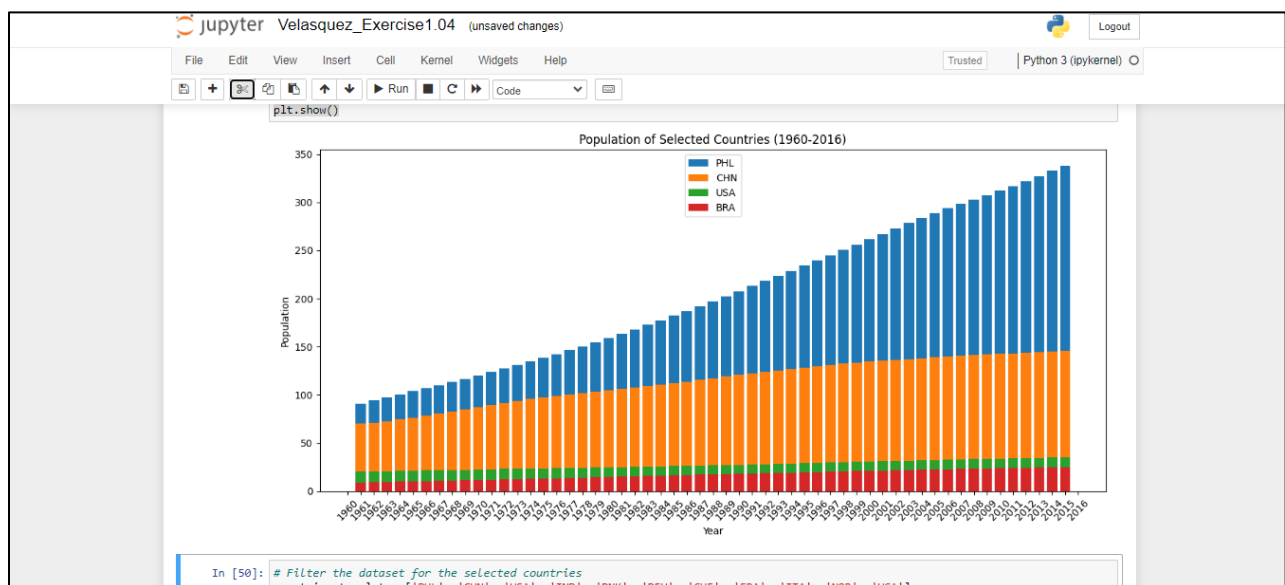
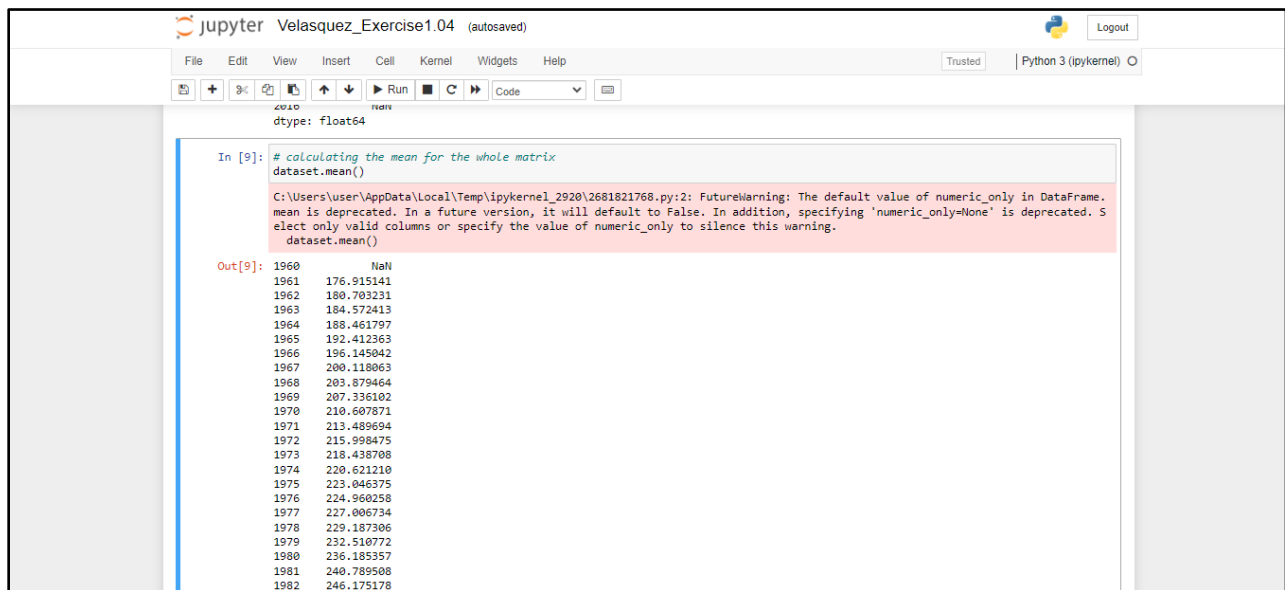
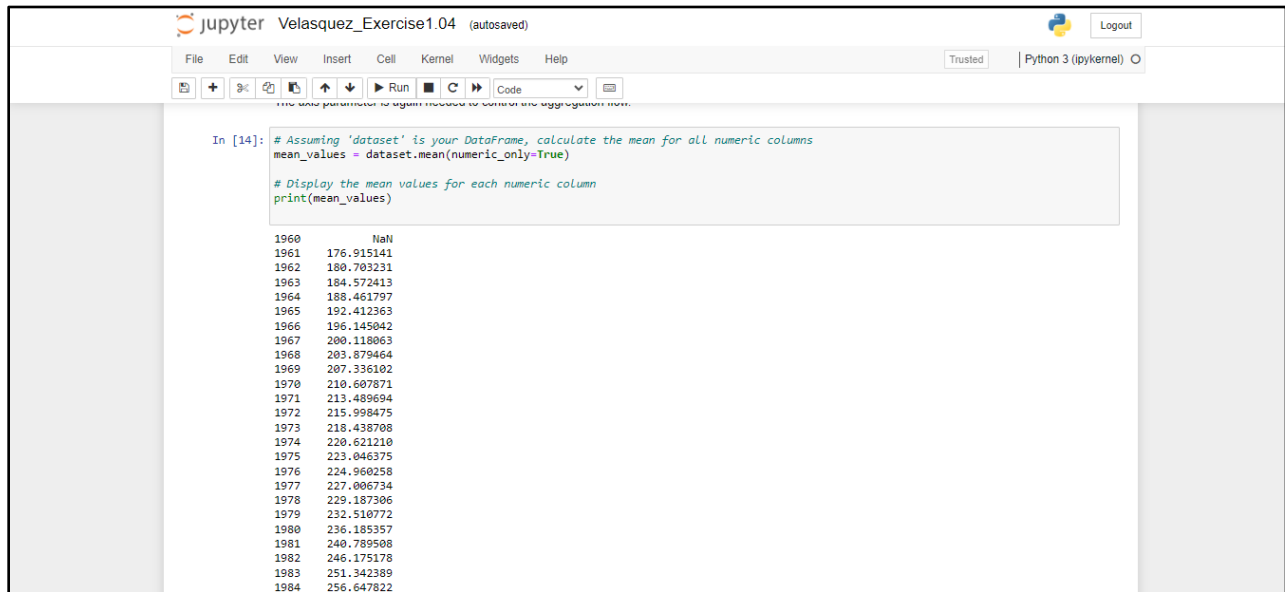
dtype: float64

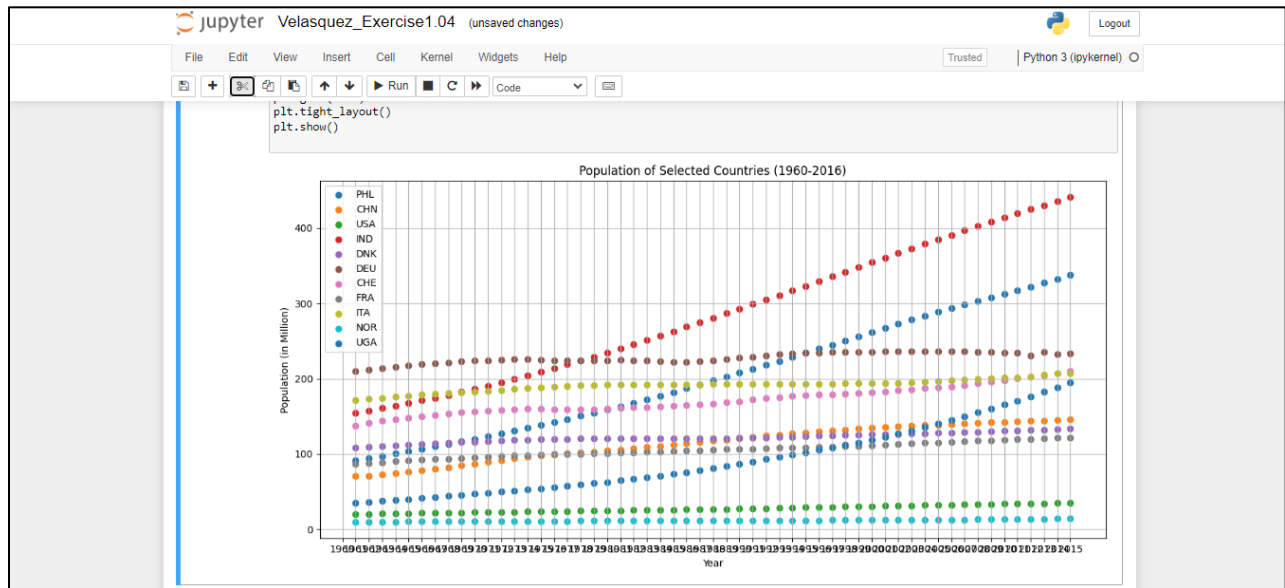
```
In [8]: # mean for each feature (col)
dataset.mean(axis=0).tail(10)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2920\2830591496.py:2: FutureWarning: The default value of numeric\_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.

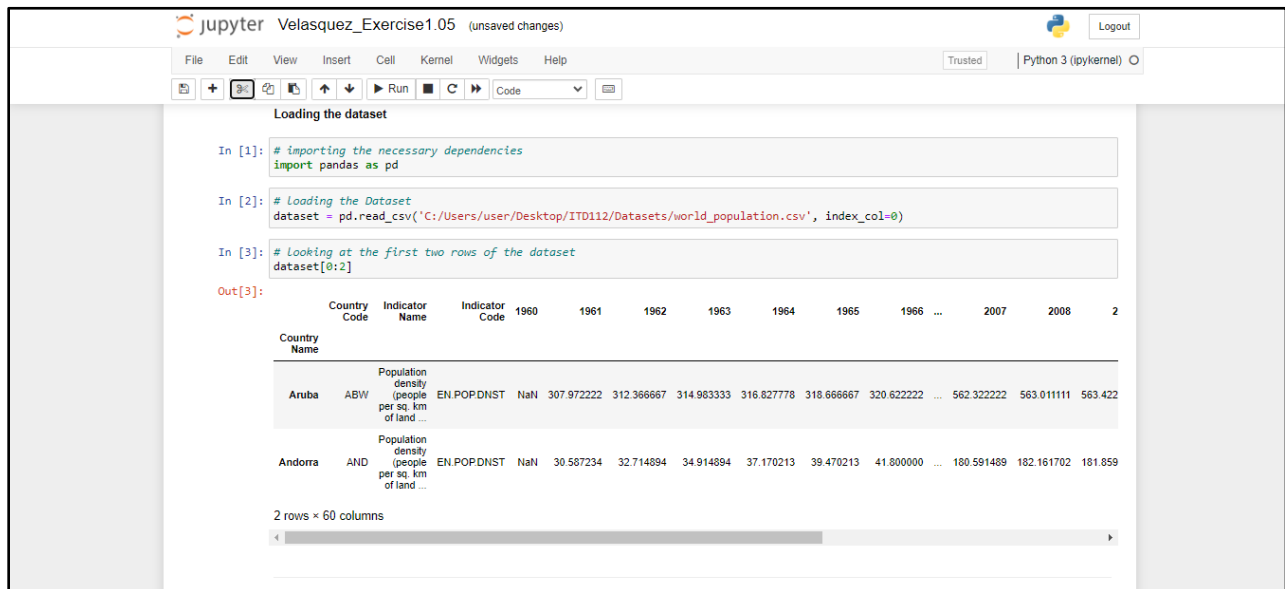
```
dataset.mean(axis=0).tail(10)
```

```
Out[8]: 2007    331.995474
2008    338.688417
2009    343.649206
```





### Exercise 1.05: Using pandas to Compute the Mean, Median, and Variance of a Dataset



Jupyter Velasquez\_Exercise1.05 (unsaved changes) Python 3 (ipykernel)

```
In [12]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
selected_columns = dataset.iloc[[2]][[column for column in dataset.columns if pd.api.types.is_numeric_dtype(dataset[column])]]

# Calculate the mean for the selected columns
row_mean = selected_columns.mean(axis=1)

# Display the mean value for the specified row
print(row_mean)
```

Country Name  
Afghanistan 25.373379  
dtype: float64

```
In [4]: # calculate the mean of the third row
dataset.iloc[[2]].mean(axis=1)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2080\1258020032.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.  
dataset.iloc[[2]].mean(axis=1)

Out[4]: Country Name  
Afghanistan 25.373379  
dtype: float64

```
In [13]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
```

Jupyter Velasquez\_Exercise1.05 (unsaved changes) Python 3 (ipykernel)

```
In [13]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
selected_columns = dataset.iloc[[-1]][[column for column in dataset.columns if pd.api.types.is_numeric_dtype(dataset[column])]]

# Calculate the mean for the selected columns
row_mean = selected_columns.mean(axis=1)

# Display the mean value for the specified row
print(row_mean)
```

Country Name  
Zimbabwe 24.520532  
dtype: float64

```
In [5]: # calculate the mean of the last row
dataset.iloc[[-1]].mean(axis=1)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2080\1844506785.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.  
dataset.iloc[[-1]].mean(axis=1)

Out[5]: Country Name  
Zimbabwe 24.520532  
dtype: float64

```
In [26]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
selected_columns = dataset.loc[["Germany"]][[column for column in dataset.columns if pd.api.types.is_numeric_dtype(dataset[column])]]

# Calculate the mean for the selected columns
```

Jupyter Velasquez\_Exercise1.05 (unsaved changes) Python 3 (ipykernel)

```
In [26]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
selected_columns = dataset.loc[["Germany"]][[column for column in dataset.columns if pd.api.types.is_numeric_dtype(dataset[column])]]

# Calculate the mean for the selected columns
row_mean = selected_columns.mean(axis=1)

# Display the mean value for the specified row
print(row_mean)
```

Country Name  
Germany 227.773688  
dtype: float64

```
In [6]: # calculate the mean of the country Germany
dataset.loc[["Germany"]].mean(axis=1)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2080\2599282623.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.  
dataset.loc[["Germany"]].mean(axis=1)

Out[6]: Country Name  
Germany 227.773688  
dtype: float64

Note:  
.iloc() and .loc() are two important methods when indexing with Pandas. They allow to make precise selections of data based on either the integer value index (.iloc) or the index column (.loc), which in our case is the country name column.

jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run

Median

```
In [29]: # Assuming 'dataset' is your DataFrame
# Select the columns you want to include in the mean calculation
selected_columns = dataset.iloc[[-1]][[column for column in dataset.columns if pd.api.types.is_numeric_dtype(dataset[column])]]

# Calculate the mean for the selected columns
row_median = selected_columns.mean(axis=1)

# Display the mean value for the specified row
print(row_median)

Country Name
Zimbabwe    25.505431
dtype: float64
```

```
In [7]: # calculate the median of the last row
dataset.iloc[[-1]].median(axis=1)

C:\Users\user\AppData\Local\Temp\ipykernel_2080\1533885436.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
  dataset.iloc[[-1]].median(axis=1)
```

```
Out[7]: Country Name
Zimbabwe    25.505431
dtype: float64
```

```
In [31]: # Select the rows you want (last 3 rows)
last_3_rows = dataset.iloc[-3:]
```

jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run

```
In [31]: # Select the rows you want (last 3 rows)
last_3_rows = dataset.iloc[-3:]

# Select the columns you want to include in the median calculation
selected_columns = last_3_rows.select_dtypes(include=['number'])

# Calculate the median for each row along columns (axis=1)
row_medians = selected_columns.median(axis=1)

# Display the medians for the last 3 rows
print(row_medians)

Country Name
Congo, Dem. Rep.    14.419050
Zambia              10.352668
Zimbabwe            25.505431
dtype: float64
```

```
In [8]: # calculate the median of the last 3 rows
dataset[-3:].median(axis=1)

C:\Users\user\AppData\Local\Temp\ipykernel_2080\1139480153.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
  dataset[-3:].median(axis=1)
```

```
Out[8]: Country Name
Congo, Dem. Rep.    14.419050
Zambia              10.352668
Zimbabwe            25.505431
dtype: float64
```

Note:

jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run

```
In [32]: # Select the columns you want to include in the median calculation
selected_columns = dataset.head(10).select_dtypes(include=['number'])

# Calculate the median for each row along columns (axis=1)
row_medians = selected_columns.median(axis=1)

# Display the medians for the first 10 rows
print(row_medians)

Country Name
Aruba              348.022222
Andorra            107.300000
Afghanistan        19.998926
Angola             8.458253
Albania            106.001058
Arab World         15.307283
United Arab Emirates 19.305072
Argentina          11.618238
Armenia            105.898033
American Samoa     220.245000
dtype: float64
```

```
In [9]: # calculate the median of the first 10 countries
dataset.head(10).median(axis=1)

C:\Users\user\AppData\Local\Temp\ipykernel_2080\2702851610.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
  dataset.head(10).median(axis=1)
```

```
Out[9]: Country Name
Aruba              348.022222
Andorra            107.300000
```

Jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help Trusted

American Samoa 220.245000  
dtype: float64

```
In [9]: # calculate the median of the first 10 countries
dataset.head(10).median(axis=1)
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2080\2702851610.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric\_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

```
dataset.head(10).median(axis=1)
```

```
Out[9]: Country Name
Aruba      348.022222
Andorra    107.300000
Afghanistan 19.998926
Angola      8.458253
Albania     106.001058
Arab World  15.307283
United Arab Emirates 19.305072
Argentina  11.618236
Armenia     105.898033
American Samoa 220.245000
dtype: float64
```

**Note:**  
When handling larger datasets, the order in which methods get executed definitely matters. Think about what `.head(10)` does for a moment, it simply takes your dataset and returns the first 10 rows of it, cutting down your input to the `.mean()` method drastically. This will definitely have an impact when using more memory intensive calculations, so keep an eye on the order.

Jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help Trusted

**Variance**

```
In [33]: # Calculate the variance for columns with numeric data only
variance = dataset.var(numeric_only=True).tail()

# Display the variance for the selected columns
print(variance)
```

```
2012    3.063475e+06
2013    3.094597e+06
2014    3.157111e+06
2015    3.220634e+06
2016         NaN
dtype: float64
```

```
In [10]: # calculate the variance of the last 5 columns
dataset.var().tail()
```

C:\Users\user\AppData\Local\Temp\ipykernel\_2080\3575222615.py:2: FutureWarning: The default value of numeric\_only in DataFrame.var is deprecated. In a future version, it will default to False. In addition, specifying 'numeric\_only=None' is deprecated. Select only valid columns or specify the value of numeric\_only to silence this warning.

```
dataset.var().tail()
```

```
Out[10]: 2012    3.063475e+06
2013    3.094597e+06
2014    3.157111e+06
2015    3.220634e+06
2016         NaN
dtype: float64
```

Jupyter Velasquez\_Exercise1.05 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help Trusted

```
dataset.var().tail()
```

```
Out[10]: 2012    3.063475e+06
2013    3.094597e+06
2014    3.157111e+06
2015    3.220634e+06
2016         NaN
dtype: float64
```

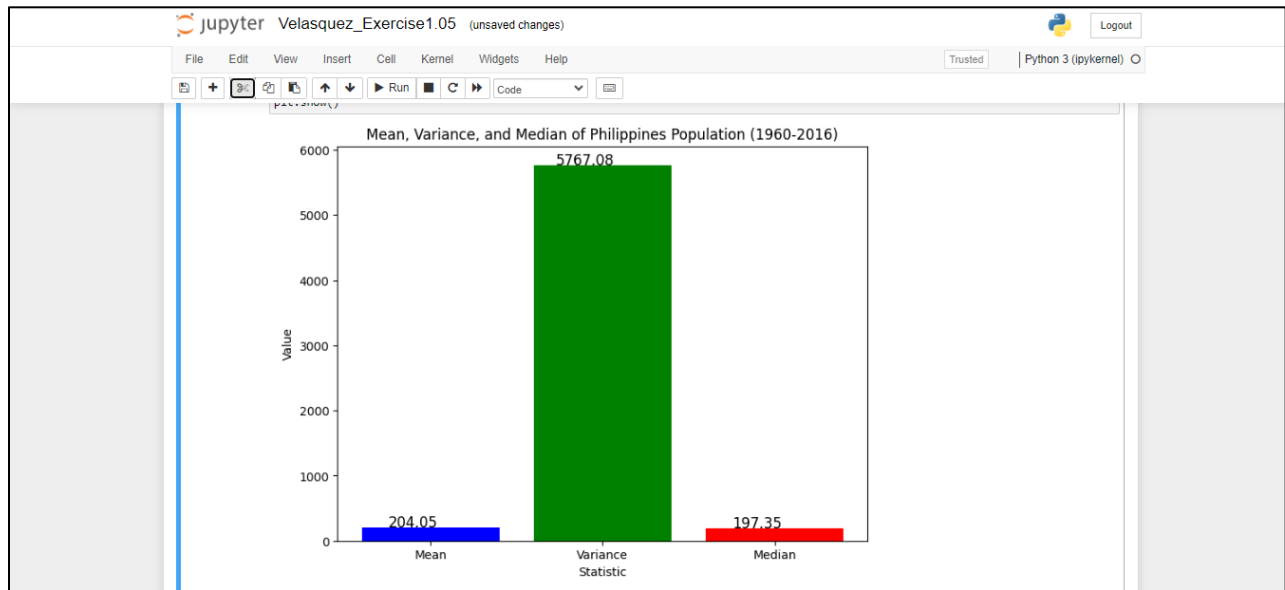
As mentioned in the introduction of Pandas, it's interoperable with several of NumPy's features. Here's an example of how to use NumPy's `mean` method with a Pandas DataFrame.

```
In [11]: # NumPy Pandas interoperability
import numpy as np

print("Pandas", dataset["2015"].mean())
print("NumPy", np.mean(dataset["2015"]))
```

```
Pandas 368.70660104001837
NumPy 368.70660104001837
```





## Exercise 1.06: Indexing, Slicing, and Iterating Using pandas

Jupyter Velasquez\_Exercise1.06 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Loading the dataset

```
In [1]: # importing the necessary dependencies
import pandas as pd

In [2]: # Loading the Dataset
dataset = pd.read_csv('C:/Users/user/Desktop/ITD112/Datasets/world_population.csv', index_col=0)

In [3]: # Looking at the first 2 elements of the dataset
dataset.head(2)
```

Out[3]:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2
Aruba	ABW	Population density (people per sq. km of land ...	EN.POP.DNST	NaN	307.972222	312.366667	314.983333	316.827778	318.666667	320.622222	...	562.322222	563.011111	563.422
Andorra	AND	Population density (people per sq. km of land ...	EN.POP.DNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213	41.800000	...	180.591489	182.161702	181.859

2 rows x 60 columns

Jupyter Velasquez\_Exercise1.06 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Indexing

Since we need several rows and columns of our dataset to complete the given task, we have to use indexing to get the right rows and columns. Use indexing to get:

- the row of the USA
- the second to last row
- the column of year 2000 as Series
- the population density for India in 2000

```
In [4]: # indexing the USA row
dataset.loc[["United States"]].head()
```

Out[4]:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009	2
United States	USA	Population density (people per sq. km of land ...	EN.POP.DNST	NaN	20.05588	20.366723	20.661953	20.950959	21.214527	21.460952	...	32.878611	33.243687	33.536399	33.817

1 rows x 60 columns

Jupyter Velasquez\_Exercise1.06 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [5]: # indexing the last second to last row by index
dataset.iloc[[-2]]
```

Out[5]:

Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009	2010
Zambia	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	4.227724	4.359305	4.496824	4.639914	4.788452	4.942343	...	17.135926	17.641587	18.170609	18.721585

1 rows x 60 columns

```
In [6]: # indexing the column of 2000 as a Series
dataset["2000"].head()
```

Out[6]:

Country Name	2000
Aruba	504.766667
Andorra	139.146809
Afghanistan	30.177894
Angola	12.078798
Albania	112.738212

Name: 2000, dtype: float64

```
In [7]: # indexing the population density of India in 2000 (Dataframe)
dataset[["2000"]].loc[["India"]]
```

Jupyter Velasquez\_Exercise1.06 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [7]: # indexing the population density of India in 2000 (Dataframe)
dataset[["2000"]].loc[["India"]]
```

Out[7]:

Country Name	2000
India	354.326858

Note:  
Using single brackets to index columns (like with NumPy) we will get a pandas Series object.  
When using double brackets to do indexing, a DataFrame will be returned. This way we can also index several elements with one query.  
When comparing the output of the DataFrame query to the Series query, we can see the difference between Series and DataFrames

```
In [8]: # indexing the population density of India in 2000 (Series)
dataset["2000"].loc["India"]
```

Out[8]: 354.326858357522

Slicing

Other than the single rows and columns and we also need to get some Subsets of the dataset.  
Use slicing for:

- the countries in row 2 to 5

Jupyter Velasquez\_Exercise1.06 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

- Germany, Singapore, United States, and India with their population density of years 1970, 1990, 2010

```
In [9]: # slicing countries of rows 2 to 5
dataset.iloc[1:5]
```

Out[9]:

Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	2009
Andorra	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	30.587234	32.714894	34.914894	37.170213	39.470213	41.800000	...	180.591489	182.161702	181.859574
Afghanistan	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	14.038148	14.312061	14.599692	14.901579	15.218206	15.545203	...	39.637202	40.634655	41.674005
Angola	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	4.305195	4.384299	4.464433	4.544558	4.624228	4.703271	...	15.387749	15.915819	16.459536
Albania	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	60.576642	62.456898	64.329234	66.209307	68.058066	69.874927	...	108.394781	107.566204	106.843756

4 rows x 60 columns

Jupyter Velasquez\_Exercise1.06 (autosaved) Python 3 (ipykernel)

```
In [10]: # slicing rows Germany, Singapore, United States, and India
dataset.loc[["Germany", "Singapore", "United States", "India"]]
```

Out[10]:

Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008
Germany	DEU	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	210.172807	212.029284	214.001527	215.731495	217.579970	219.403406	...	235.943362	235.52217
Singapore	SGP	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	2540.895522	2612.238806	2679.104478	2748.656716	2816.268657	2887.164179	...	6602.300719	6913.42285
United States	USA	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	20.055880	20.366723	20.861953	20.950959	21.214527	21.460952	...	32.878611	33.24361
India	IND	Population density (people per sq. km of land ...)	EN.POPDNST	NaN	154.275864	157.424902	160.679256	164.029246	167.470047	170.995768	...	396.774384	402.62146

4 rows x 60 columns

Jupyter Velasquez\_Exercise1.06 (autosaved) Python 3 (ipykernel)

```
In [11]: # slicing a subset of Germany, Singapore, United States, and India
# for years 1970, 1990, 2010 <
country_list = ["Germany", "Singapore", "United States", "India"]
dataset.loc[country_list][["1970", "1990", "2010"]]
```

Out[11]:

Country Name	1970	1990	2010
Germany	223.897371	227.517054	234.606908
Singapore	3096.268657	4547.958209	7231.811966
United States	22.388131	27.254514	33.817936
India	186.312757	292.817404	414.028200

Iterating

As the last task of this exercise, we want to iterate over the first three countries of our dataset and print:

- name
- country code
- years 1970, 1990, 2010

Jupyter Velasquez\_Exercise1.06 (autosaved) Python 3 (ipykernel)

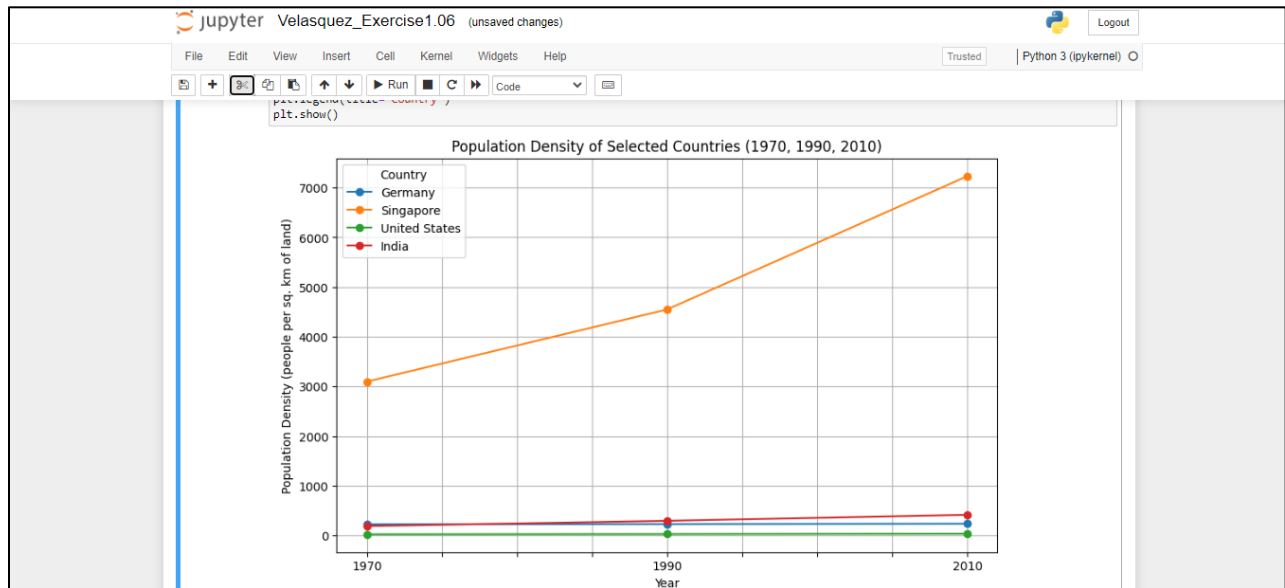
```
In [12]: # iterating over the first three countries (row by row)
for index, row in dataset.iterrows():
    # only printing the rows until Angola
    if index == 'Angola':
        break

    print(index, '\n', row[["Country Code", "1970", "1990", "2010"]], '\n')
```

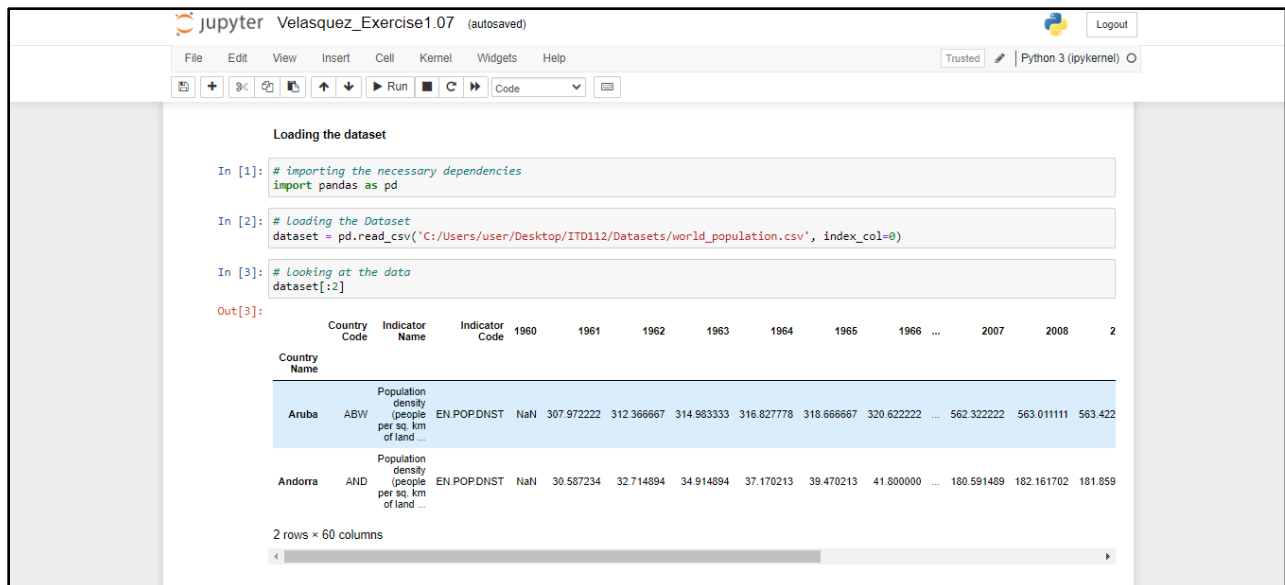
Aruba  
Country Code ABW  
1970 328.138889  
1990 345.266667  
2010 564.427778  
Name: Aruba, dtype: object

Andorra  
Country Code AND  
1970 51.657447  
1990 115.980851  
2010 179.614894  
Name: Andorra, dtype: object

Afghanistan  
Country Code AFG  
1970 17.034429  
1990 18.484162  
2010 42.830327  
Name: Afghanistan, dtype: object



## Exercise 1.07: Filtering, Sorting, and Reshaping



Jupyter Velasquez\_Exercise1.07 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [4]: # filtering columns 1961, 2000, and 2015
dataset.filter(items=["1961", "2000", "2015"]).head()
```

Out[4]:

	1961	2000	2015
Country Name			
Aruba	307.972222	504.766667	577.161111
Andorra	30.587234	139.146809	149.942553
Afghanistan	14.038148	30.177894	49.821649
Angola	4.305195	12.078798	20.070565
Albania	60.576642	112.738212	105.444051

```
In [5]: # filtering countries that had a greater population density than 500 in 2000
dataset[(dataset["2000"] > 500)][["2000"]]
```

Out[5]:

	2000
Country Name	
Aruba	504.766667
Bangladesh	1008.532988
Bahrain	939.232394
Bermuda	1236.660000
Barbados	627.530233
Channel Islands	766.823711
Gibraltar	2735.100000

Jupyter Velasquez\_Exercise1.07 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Sint Maarten (Dutch part) 89/81/64/

```
In [6]: # filtering for years 2000 and later
dataset.filter(regex="^2", axis=1).head()
```

Out[6]:

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
Country Name												
Aruba	504.766667	516.077778	527.750000	538.972222	548.566667	555.727778	560.166667	562.322222	563.011111	563.422222	564.427778	566.311111
Andorra	139.146809	144.191489	151.161702	159.112766	166.674468	172.814894	177.389362	180.591489	182.161702	181.859574	179.614894	175.161702
Afghanistan	30.177894	31.448029	32.912231	34.475030	35.995236	37.373936	38.574296	39.637202	40.634655	41.674005	42.830327	44.127634
Angola	12.078798	12.483188	12.921871	13.388462	13.873025	14.368286	14.872437	15.387749	15.915819	16.459536	17.020898	17.600302
Albania	112.738212	111.685146	111.350730	110.934891	110.472226	109.908285	109.217044	108.394781	107.566204	106.843759	106.314635	106.013689

```
In [7]: # # filtering countries that start with A
dataset.filter(regex="^A", axis=0).head()
```

Out[7]:

	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	
Country Name														
Aruba	ABW	Population density (people per sq. km of land ...	EN.POPDNST	NaN	307.972222	312.366667	314.983333	316.827778	318.666667	320.822222	...	562.322222	563.011111	563
Population														

Jupyter Velasquez\_Exercise1.07 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [8]: # filtering countries that contain the word Land
dataset.filter(like="Land", axis=0).head()
```

Out[8]:

	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	1966	...	2007	2008	
Country Name														
Switzerland	CHE	Population density (people per sq. km of land ...	EN.POPDNST	NaN	137.479609	141.009285	144.056036	146.458915	148.160089	149.716707	...	191.090115	193.533632	195.
Channel Islands	CHI	Population density (people per sq. km of land ...	EN.POPDNST	NaN	569.067010	574.551546	580.386598	586.484536	592.742268	599.103093	...	806.783505	812.304124	817.
Cayman Islands	CYM	Population density (people per sq. km of land ...	EN.POPDNST	NaN	33.441667	33.925000	34.283333	34.579167	34.879167	35.175000	...	214.500000	220.520833	226.
Finland	FIN	Population density (people per sq. km of land ...	EN.POPDNST	NaN	14.645934	14.745865	14.850484	14.933330	14.983197	15.039460	...	17.391956	17.484038	17.
Faroe Islands	FRO	Population density (people per sq. km of land ...	EN.POPDNST	NaN	24.878223	25.181232	25.465616	25.749284	26.047994	26.363897	...	34.813037	34.834527	34.

Jupyter Velasquez\_Exercise1.07 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run Code

- values sorted in ascending order by 1961
- values sorted in ascending order by 2015
- values sorted in descending order by 2015

```
In [9]: # values sorted by column 1961
dataset.sort_values(by=["1961"])[["1961"]].head(10)
```

```
Out[9]:
```

1961	
Country Name	
Greenland	0.098625
Mongolia	0.632212
Namibia	0.749775
Libya	0.843320
Mauritania	0.856916
Botswana	0.946793
United Arab Emirates	1.207955
Australia	1.364565
Iceland	1.785825
Oman	1.825186

```
In [10]: # values sorted by column 2015
dataset.sort_values(by=["2015"])[["2015"]].head(10)
```

```
Out[10]:
```

2015	
------	--

Jupyter Velasquez\_Exercise1.07 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run Code

```
In [10]: # values sorted by column 2015
dataset.sort_values(by=["2015"])[["2015"]].head(10)
```

```
Out[10]:
```

2015	
Country Name	
Greenland	0.136713
Mongolia	1.904744
Namibia	2.986590
Australia	3.095579
Iceland	3.299980
Suriname	3.480609
Libya	3.568227
Guyana	3.896800
Canada	3.942567
Mauritania	3.946409

**Note:**  
Comparisons like this are very valuable to get a good understanding not only of your dataset but also the underlying data itself. For example, here we can see that the ranking of the lowest densely populated countries changed.

```
In [11]: # values sorted by column 2015 in descending order
dataset.sort_values(by=["2015"], ascending=False)[["2015"]].head(10)
```

```
Out[11]:
```

2015	
------	--

Jupyter Velasquez\_Exercise1.07 (autosaved) Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

Run Code

**Note:**  
Comparisons like this are very valuable to get a good understanding not only of your dataset but also the underlying data itself. For example, here we can see that the ranking of the lowest densely populated countries changed.

```
In [11]: # values sorted by column 2015 in descending order
dataset.sort_values(by=["2015"], ascending=False)[["2015"]].head(10)
```

```
Out[11]:
```

2015	
Country Name	
Macao SAR, China	19392.937294
Monaco	18865.500000
Singapore	7828.857143
Hong Kong SAR, China	6957.809524
Gibraltar	3221.700000
Bahrain	1788.619481
Maldives	1363.876667
Malta	1347.915625
Bermuda	1304.700000
Bangladesh	1236.810648

Reshaping

Jupyter Velasquez\_Exercise1.07 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Reshaping

In order to create a visualization that focuses on 2015, they ask you to create a subset of your DataFrame which only contains one row that holds all the values for the year 2015 mapped to the country codes as columns.

They've sent you this scribble:

```
Country Code  ABN  AFG  AGO  ...
-----
2015         577    49    20  ...
```

They were lazy so they didn't write the digits after the comma. Make sure to keep the original values

```
In [12]: # reshaping to 2015 as row and country codes as columns
dataset_2015 = dataset[["Country Code", "2015"]]
dataset_2015.pivot(index="2015", columns="Country Code", values="2015")
```

Out[12]:

2015	2015	2015	2015	2015	2015	2015	2015	2015	2015	2015	2015
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713	0.136713
1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744	1.904744
2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590	2.986590
3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579	3.095579

