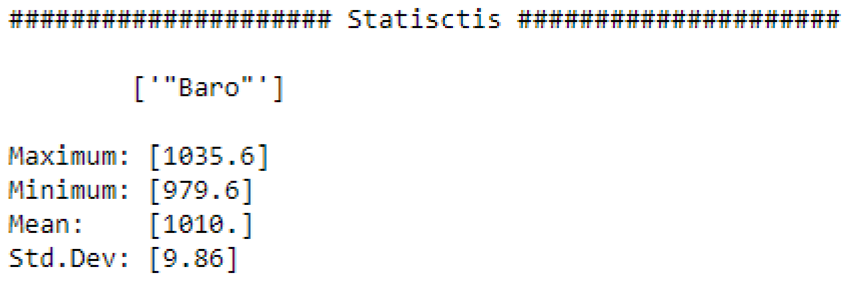
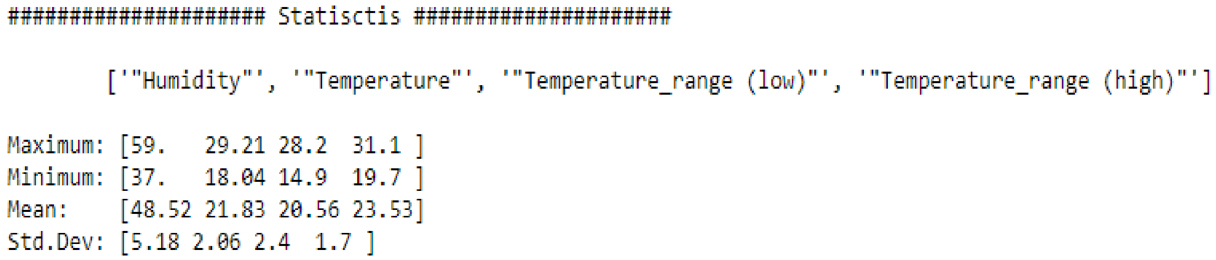
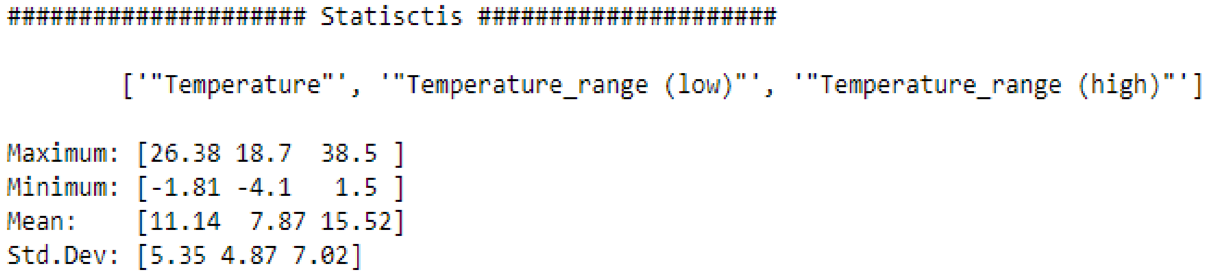
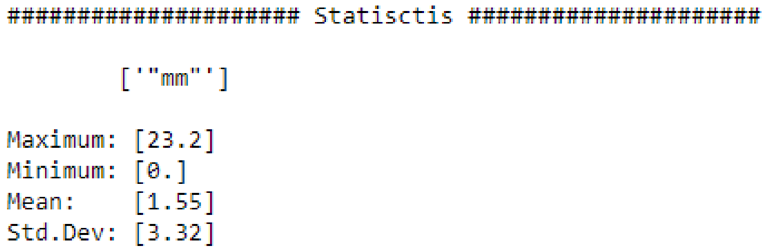
Implementation of a CSV-parser by using Basic Python File I/O

The present work shows the results I got based on my csv parser.

Results:

The following results consider **not malformed data**, all fields were filled according to their corresponding type of data

File	Results
barometer-1617.csv	 ######################## Statisctis ######################## ['"Baro"'] Maximum: [1035.6] Minimum: [979.6] Mean: [1010.] Std.Dev: [9.86]
indoor-temperature-1617.csv	 ##################### Statisctis ##################### ['"Humidity"', '"Temperature"', '"Temperature_range (low)"', '"Temperature_range (high)"'] Maximum: [59. 29.21 28.2 31.1] Minimum: [37. 18.04 14.9 19.7] Mean: [48.52 21.83 20.56 23.53] Std.Dev: [5.18 2.06 2.4 1.7]
outside-temperature-1617.csv	 ###################### Statisctis ###################### ['"Temperature"', '"Temperature_range (low)"', '"Temperature_range (high)"'] Maximum: [26.38 18.7 38.5] Minimum: [-1.81 -4.1 1.5] Mean: [11.14 7.87 15.52] Std.Dev: [5.35 4.87 7.02]
rainfall-1617.csv	 ###################### Statisctis ###################### ['"mm"'] Maximum: [23.2] Minimum: [0.] Mean: [1.55] Std.Dev: [3.32]

In the next section, I am presenting my findings, solutions and rational when I modified the file called: ***"barometer-1617.csv"***, which is now ***"barometer-1617_2.csv"***

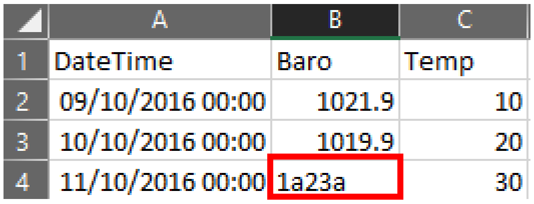
To be more practical, I only considered the first 4 rows to analyze and modify them, then I added columns, rows, etc. to see the behavior of the program.

- **Mixed Data**

  In the very first trials I mixed numbers with alphabetical characters (see red rectangle) to see how the results could change by adding a different type of data in the same column, a typical type-o is to swap the letter "O", with the zero "0".

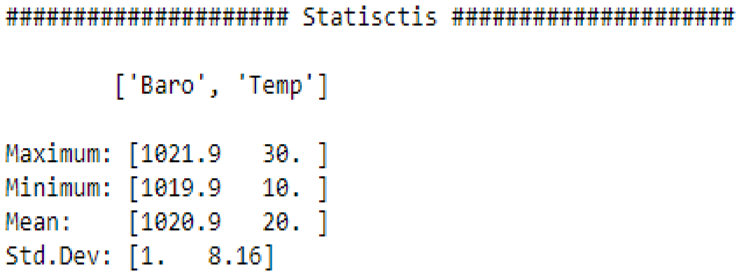
  The result raised an error when the program tried to convert that data into a number, with that in mind I created a method that convert and differentiate string data and integer / float data (def str_float), this way the program can continue with the next step which is to get the statistics, however as result of the data structure contained a non-valid data to calculate the statistics, another error brought. To solve that, the method "creating_array", creates an array and take only the data which is a number, similar to what pandas **describe()** method does.

Excel View



	A	B	C
1	DateTime	Baro	Temp
2	09/10/2016 00:00	1021.9	10
3	10/10/2016 00:00	1019.9	20
4	11/10/2016 00:00	1a23a	30

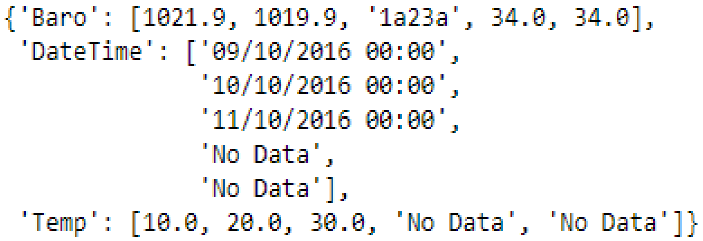
Results



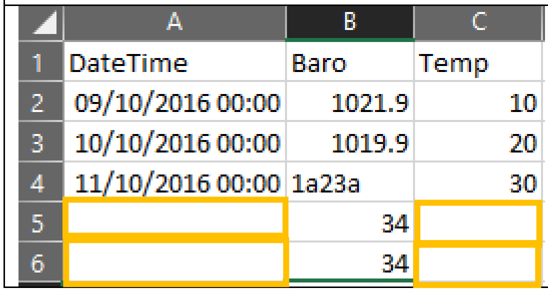
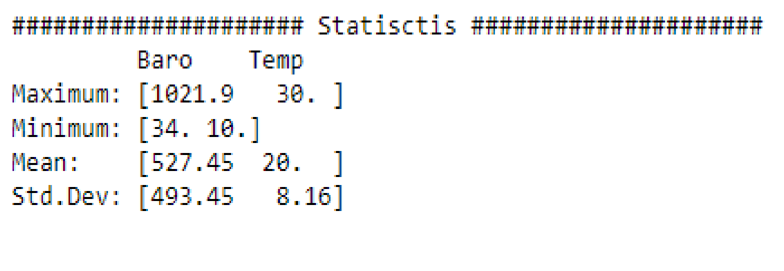
```
###################### Statisctis #######################


        ['Baro', 'Temp']

Maximum: [1021.9   30. ]
Minimum: [1019.9   10. ]
Mean:    [1020.9   20. ]
Std.Dev: [1.    8.16]
```

- **Adding extra values**

  Another point that should be noticed is that if I add more values in the same column, what the programs does is to fill all remaining fields (see yellow rectangle) by adding the string characters **"No Data"** (See picture below), which is part of the checking method that replaces empty spaces with those words and the calculation continues with no issues based on the creating_array and str_float method previously described.



```
{'Baro': [1021.9, 1019.9, '1a23a', 34.0, 34.0],
 'DateTime': ['09/10/2016 00:00',
              '10/10/2016 00:00',
              '11/10/2016 00:00',
              'No Data',
              'No Data'],
 'Temp': [10.0, 20.0, 30.0, 'No Data', 'No Data']}
```

Excel View	Results
	

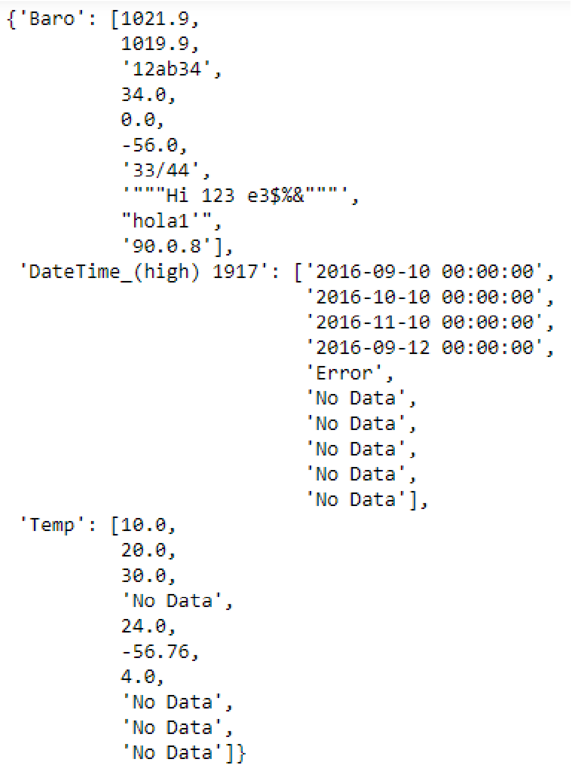
	A	B	C
1	DateTime	Baro	Temp
2	09/10/2016 00:00	1021.9	10
3	10/10/2016 00:00	1019.9	20
4	11/10/2016 00:00	1a23a	30
5		34	
6		34	

```
##################### Statisctis #####################
          Baro    Temp
Maximum: [1021.9   30. ]
Minimum: [34. 10.]
Mean:    [527.45  20.  ]
Std.Dev: [493.45   8.16]
```

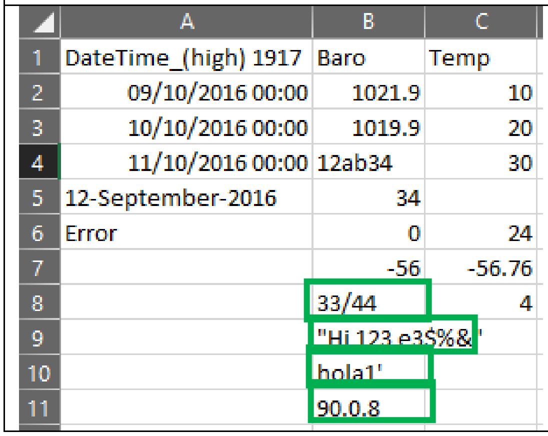
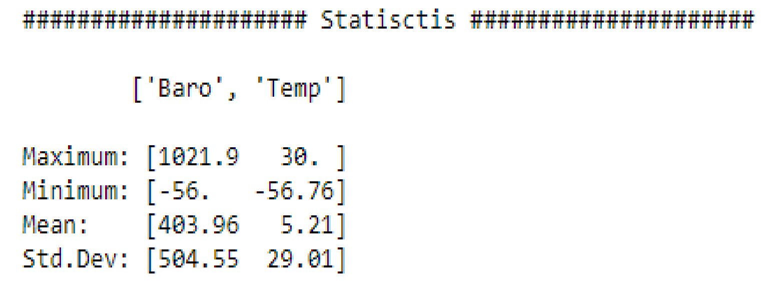
- **Random values**

Moving on to another point of interested is to check if the fields were filled with weird values / malformed data and verify if the program continues working.

Some random values I used to test this part were by using double quotations, non-alphabetical or numerical characters, fractions, etc. (see green rectangles) and again the results were positives.



```
{'Baro': [1021.9,
          1019.9,
          '12ab34',
          34.0,
          0.0,
          -56.0,
          '33/44',
          '"""Hi 123 e3$%&"""',
          "hola1'",
          '90.0.8'],
 'DateTime_(high) 1917': ['2016-09-10 00:00:00',
                          '2016-10-10 00:00:00',
                          '2016-11-10 00:00:00',
                          '2016-09-12 00:00:00',
                          'Error',
                          'No Data',
                          'No Data',
                          'No Data',
                          'No Data',
                          'No Data'],
 'Temp': [10.0,
          20.0,
          30.0,
          'No Data',
          24.0,
          -56.76,
          4.0,
          'No Data',
          'No Data',
          'No Data']}
```

Excel View	Results
	

	A	B	C
1	DateTime_(high) 1917	Baro	Temp
2	09/10/2016 00:00	1021.9	10
3	10/10/2016 00:00	1019.9	20
4	11/10/2016 00:00	12ab34	30
5	12-September-2016	34	
6	Error	0	24
7		-56	-56.76
8		33/44	4
9		"Hi 123 e3$%&"	
10		hola1'	
11		90.0.8	

```
##################### Statisctis #####################


         ['Baro', 'Temp']

Maximum: [1021.9   30. ]
Minimum: [-56.   -56.76]
Mean:    [403.96   5.21]
Std.Dev: [504.55  29.01]
```

- **Date Formats**

  In spite of the Date Time is not part of the statiscts calcuation, this is an interest topic to discuss as result of it is hard to interpret the value, and how to hanle this kind of data is a little tricky.

  Let´s take the example of this date: “2018-12-1”, this can be the January 12th, 2020 or December 1st, 2020, when no custom formatting is given, the default string format is used, so in ISO 8601 format (YYYY-MM-DDTHH:MM:SS.mmmmmm) this will be the second option I described (Dec 1st), however this is only one conevtion, but we know a variety of ways to represent this single date, based on this; the use of Regular Expressions is very powerfull and if we add the dateutil library, this task to handle Date time will be a little easier.

Conclusions

A csv parser could be a huge project as result of there are **multiple factors** that you need **to consider** and obviously how you want **to process the data**, *from what delimiter is the one that affects your file, no standardization, different process/procedures, until how data is presented* (**Date Time is a perfect candidate** to represent this last point due to there a bunch of ways we can write, represent and understand the date i.e. “16/Sept/2019”, “September 16, 2019”, “16.09.2019”, etc).

Another excellent example is: if we consider the last file presented in “Random values” section, with malformed data; we can easily **discard all those weird values** and then **calculate the statistics**, however **pandas library has another perspective,** which is *not to consider all the column as result of the malformed values inside.*

Some other one is that: my program *is not considering fractions*, so the fraction value could not be converted in its decimal representation, thus *the calculation is not taking into account that field* (**changing the results if a fraction is in a file**).

As we can see *data ingestion and wrangling is a huge world* and create a generic csv parser that can detect everything and do the correct according to the author/creator/user is a big challenge, with a lot of **points to face which are playing an important role**, so for this reason is important to **define and clarify the objective** of our csv parser by **delimiting some methods or analyses**.