**Questions**

You are given a dataset (attached/linked, dataLarge.gz, 727MB compressed) that consists of 25 time series.

Set A:

1. How will you analyze this data for cleanliness?
   1. Apply your suggestions to this data and describe ways in which it is not clean.
2. Describe which parts (if any) of the data you would want to find ways to clean and which parts (if any) you would abandon as being beyond cleanable and why.
3. Write a program to provide your best estimate for the mean and standard-deviation of each of the time-series.
   1. What if you had only (very) limited RAM to run your program? Can you run your program without loading the whole time series in RAM?
   2. What if you had multiple CPUs available - could you speed up your program by parallelizing it?
4. Write a program to find the relation, if any, between series 2 and series 3. Describe your reasoning for what evidence you have found for this relation.
   1. What if you had only (very) limited RAM to run your program? Can you run your program without loading the whole time series in RAM?
   2. What if you had multiple CPUs available - could you speed up your program by parallelizing it?
5. Assume this dataset was data collected over a day and this collection was to be repeated every day. Describe ways in which you will store this data over time.

Set B:

1. Write a program to compute the running PNL given a series of trades (quantity bought/sold) and a series of prices (attached/linked, trades.txt, 18.7KB uncompressed)

answers :

1. firstly, the structure of the data was checked, are there actually presence of 26 fields (including index), once that is done, Not a number values were counted across all field, and it was found out that only 4 time series suffer from this problem. The NaN values count is as follows -  
      
   [0, 0, 0, 832864, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3070569, 0, 0, 2272683, 0, 0, 0, 0, 3122569, 0, 0]  
   There are 50 lakh rows in total, for the fourth time series it will make sense to handle NaNs but for the 15th and 23rd series more than 60% of series is unclean, so this is not fit for cleaning. In fact one might even want to ignore cleaning of series 18, it also has too many Nans.  
     
   Outlier removal - cap all the entries to a mean +- k\*sigma value, this would remove the unexpected jumps which might be noise in the data.  
   referring [here](http://www.apple.com) , thus we can cap the values in u+-5\*sigma , so you will still cover 96 % of the values
2. There are many methods that can be used for figuring out the missing values (NAN) in this case - Mean imputation, mode imputation or you might even want to ignore the missing values but in this case series 4 has 1/6 th of its data as NaN thus imputing should make more sense  
   I found a manuscript describing a method for multiple imputations to time series - [Fernando TUSSEL](http://www.et.bs.ehu.es/~etptupaf/pub/papiros/mits.pdf), but haven't gone through it fully
   1. In general we can build a forecaster (predictor) and impute values in the series which reduces the forecast error. This is very difficult to do when the series has sufficiently high NaNs, series 15,18,23, on an average series 18 will have one missing values in 4.5 rows, this will make forecasting very error prone, even if it did not then using a different sample for estimating missing value will need a continuous dataset which is clean which is less probable owing to high missing values
   2. We can also try methods like polynomial interpolation or exponential smoothening to guess missing values
   3. Another way we can solve this problem is by establishing relationship between time series and then using this model to estimate the missing values of another time series

3.I don't have enough RAM on my system, so instead of storing the complete file in memory, it is read line by line (the moment next line is read, previous line is discarded), otherwise one could easily load the complete file in ram as a matrix and carry on with operations required to get mean and standard deviations.

Multiple CPUs are available hence the solution can be broken up into different threads, each thread concentrate only on one part of the data set ( as in divide data set in batches and distribute it to the threads) finally when all threads have done their execution, each of their means will be combined. In order to do this you will need a function that computes mean, square sum average and returns these quantities along with the count, so now when we have this from all the threads we only need to combine these together to calculate the population mean / standard deviation .   
 (Mean(i)\*i + Mean(n-i)\*(n-i) ) / n

(Square-mean(i)\*i + square-mean(n-i)\*(n-i) ) / n

Standard-deviation = sqrt ( square-mean - Mean^2)

values without stripping outliers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MEAN | Standard Deviation | Standard Deviation | Stripped standar dev. | NaNs |
| 2500000 | 2500000 | 1443375.384 | 1443375.384 | 0 |
| -0.01425864303 | -0.01426259634 | 33.07532163 | 33.07529801 | 0 |
| -0.01112186552 | -0.01112061076 | 26.76096167 | 26.76095525 | 0 |
| -0.00498402345 | -0.004985826737 | 26.0943026 | 26.09425314 | 832864 |
| 0.01094161323 | 0.01094093926 | 17.79882149 | 17.79879758 | 0 |
| 250.0429179 | 220.8997167 | 1265.617262 | 1079.743079 | 0 |
| 0.01383857485 | 0.01383900031 | 23.2403559 | 23.24035376 | 0 |
| 0.01094161323 | 0.01094093926 | 17.79882149 | 17.79879758 | 0 |
| -0.006197666701 | -0.006197224033 | 7.670766523 | 7.670764246 | 0 |
| -0.0009983435622 | -0.0009983435622 | 6.438240722 | 6.438240722 | 0 |
| 0.001807129592 | 0.001806763015 | 8.646852316 | 8.646850402 | 0 |
| -0.0003997471831 | -0.0003994644317 | 4.903375068 | 4.903373614 | 0 |
| -0.0008897841833 | -0.0008895252218 | 5.622452878 | 5.622449497 | 0 |
| -0.0009204917741 | -0.0009202464963 | 4.056769622 | 4.056767385 | 0 |
| 249.2302866 | 219.7440319 | 1264.08682 | 1075.187829 | 3070569 |
| 0.0002261775079 | 0.0002262451814 | 4.799221843 | 4.799220651 | 0 |
| 0.0004731815047 | 0.0004729523409 | 3.241850346 | 3.241849178 | 0 |
| -0.002287365906 | -0.00228723684 | 2.775775639 | 2.775774985 | 2272683 |
| 249.9879797 | 220.7750867 | 1266.248931 | 1079.861673 | 0 |
| 0.001564742489 | 0.001564742489 | 3.084667758 | 3.084667758 | 0 |
| 250.0277053 | 220.8586237 | 1266.427168 | 1080.363149 | 0 |
| -0.002362957324 | -0.002363152543 | 3.883462712 | 3.883461712 | 0 |
| 0.001499426256 | 0.001499204163 | 1.974737342 | 1.974736208 | 3122569 |
| -0.0003947239187 | -0.0003947660588 | 0.5346703234 | 0.5346700584 | 0 |
| 0.0002643677402 | 0.0002644119017 | 1.536784775 | 1.536784551 | 0 |

Note that in python due to GIL, multithreading might not be able to utilize the resources efficiently,as mentioned [here](https://docs.python.org/2/library/threading.html#module-threading), its better to use multiprocessing library

Execution time on 5 parallel jobs 95.821 seconds - dividing dataset into 5 partitions then aggregating  
Execution time with a single process 180.344 seconds

4. Perform a correlation analysis between the two, to get the dependence / linear relationship between the two series using pearson’s moment of correlation coefficient, CORRELATION between the SERIES 2 & SERIES 3 is 0.608203228653

rho = E[(x-x’)\*(y-y’)] / sigma(x)sigma(y) , x’,y’ are means

This also can be parallelized, when you have already calculated sigma and mean, you can distribute jobs in batches in a similar way as done in standard deviation / mean calculation of previous question. The p-value obtained from t-statistic in this case using t = rho\* sqrt( n - 2 ) / sqrt (1 - r^2) was found to be 0.

In this particular case the linear relationship is significant, its much greater than 0.

However given more time, would like to see in the factors such as cross correlation, or figuring out a non linear relationship between the series.

An alternative method to get more confidence that the correlation observed is not by chance:

Using the max and min values for series 2 and series 3 construct a histogram for each, assign k intervals to each of the series. Now populate the frequencies for these intervals for both the series, this will give us a rough probability density function, say a number which will lie in interval i will have probability pi. Calculate the mean across each of these intervals.

Now you can also create cumulative distribution (add the pi incrementally).

So we will now run experiments for the series 2 and 3, using this cumulative distribution, first we generate random no in to 1 and then choose and interval i S.T. p1 + p2 +… p(i-1) < number < p1 + p2+…… + pi , now we will use the mean of this interval as a value for series, doing this for both the series we can generate series from their approximated cdf.

Now we will calculate the correlation coefficient using pearson’s moment.

We will now run the above mentioned experiment a number of times and find the percentage of case when the pearson’s moment was greater than the calculated person’s moment (rho calculated above), if its lesser than 5% then we will say that our results are significant.

This implementation will need more time, some work came in office came up, hence I’m not coding this as of now.

5. Firstly if the number of time series is fixed, then a structured schema would suffice, so you can store this in a table, but then there might be cases that a particular time t0, values of only one of the fields has changed, thus you would end up wasting a lot of space (repeating values for each of the other series (fields) ). If thats the case a NO-SQL database like mongo db would be better option, where each of the fields will be a key in each row. Now even if you don't have values for some series at a given time t0 you will not be required to specifically write blank in them, you can skip that particular key. If indexing is done on the keys, then the average, standard deviation and other aggregation will also speed up.

Moreover if your data size increase beyond the capacity of single instance, you can have a cluster arrangement and distribute data across different nodes (this is true in case of any database - sql / no sql).

Having multiple cores running your jobs (aggregation related for example mean,std\_deviation) should solve most of the troubles of optimizing time but if your computations are beyond the limits of a single machine, a better scalable approach would be to rely on spark streaming (map-reduce type of implementation) it has decent library of Machine Learning / stats modules. The idea here is to have multiple machines taking care of your jobs, yeah may have a parallel architecture in them, thus giving an overall computing boost

A better implementation for this assignment would be to load the file in a database, otherwise repeatedly you have to read the file, even in the case you want to jump to a specific section in a file you will have to traverse the offset.

INSTRUCTIONS to run the programs.

1. The filename for setb is trades.txt, and seta is dataLarge,
2. keep both these files in the same directory as seta.py and set.py
3. settings.py contains some of the settings, you can modify the param BATCH\_SIZE to increase or decrease the number of processes, also in case you don't want to spend too much time processing all the 50 lac rows, change the MAX\_ROWS
4. The function get\_correlation\_parallel() computes the mean and standard deviation, also counts the missing values for each of the series,