# Data Processing Brief

*Written by Ben Tristem on 5th July 2013*

The mission is to write a Python script to “bucket” the data given into according to the specification below. There will be sufficient test data available to validate the code, and we want access to well structured and well-commented code (no need to send the processed test data back). This is to execute a million rows of data on a currently available Apple MacBook Pro (with 8GB RAM) within 10 minutes.

## Input Spec

We have a few hundred thousand rows of space and comma separated text data in the format:

**RowID, a1 a2 a3 … aX , b1 b2 b3 bX**

Row332, 4 5554 -234 9945, 33 4 0 0

Row12, 0 1 1 0, -2232 334 1 -887

(This is the format we are given the data in, slightly odd I know but it’s where we’re starting from. Notice the rows are note ordered, but we want to preserver the IDs)

The number of a and b are equal in each row: aX = bX for any given row. If this isn’t the case please write error details to a text file, and continue. There are up to 20,000 columns per row: 0 < X < 20,000ish. The values of each item are whole numbers including 0, typically of the order

-100,000ish < X < 100,000ish

Test data will be supplied as text file(s). **We need to be able to change the input file** by changing a well-marked parameter in the code. A relative file path is fine.

## Transformation

This data could be represented as a scatter plot of a vs b. We want to effectively…

1. Produce a scatter plot per row, each row having it’s a and b axis fitted to the maximum and minimum in that row.
2. Overlay a grid with equal numbers of rows and columns. **We want to be able to vary this number from G=10 to G=100** (producing grids with between 10x10=100 and 100x100=10,000 squares). Again this grid side-length (G) can be a parameter in the code.
3. For each square in the grid we want to count the number of data points that lay within that square (handling those on the boundaries consistently)
4. We then want to return a grid of numbers, where each number is the number of data points in that grid square as a fraction of the total number of data points in the row (F), ie F = points in square / x
5. The F values should be floating-point number between 0.000…0 to 1.000…0.

## Output Spec

The output will be a new text file in a similar format, with a number of space separated F values equal to the squares in our grid (i.e. 100 – 10,000)

Each RowID must be preserved. F will be full-precision floating-point numbers in the range 0.000…0 (no data points in that square) to 1.000…0 (all of the row’s data points in that square). Here’s a row as an example, notice only one comma after the RowID…

**RowID, F1 F2 F3 … FY**

Row332, 0.01234 0.823978 0.312111 0.0000000 0.0000000 … 0.023432

The new grid square of F values should be traversed from “top-left” to “bottom right”. That is the square bucket for (lowest a, highest b) first across to (highest a, highest b) then down a row until the last F number is for the bucket (highest a, lowest b)

Any questions please email [kaggle@bentristem.com](mailto:kaggle@bentristem.com)

## Trivial Test Example

### Input data

Row12, -16399 31440 -36640 16722 -41250, -28126 -43687 11355 29562 -32224

### Output data

Row12, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.20etc 0 0 0 0.20etc 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.20etc 0 0 0 0 0 0.20etc 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.20etc

0.20etc means show all the floating-point accuracy in your file. In this case they are all 0.20etc because 1/5 of the points fall in each square. If two points were in one square that square would have a value of 2/5 = 0.40etc.