Qlearsite Coding Challenge

Introduction

The purpose of this coding challenge is to build a command line tool for data-frame processing. For this exercise a "data-frame" will be a 2D array of floating point numbers (the "data") and two 1D arrays of strings (the "indices"). These indices could also be thought of as "column headers" and "row headers".

We represent these data-frames on disk as CSV files where the first row is the column index, and the first column is the row index. The end of the file *must* be indicated with a single blank line. Missing entries can be treated as `0`. For example, here is payroll data for first three months of 2016 for three fictional employees "A", "B", and "C".

```
,A,B,C,
Jan-16,2874.99,2713.23,1972.41
Feb-16,3205.4,1806.67,1625.87
Mar-16,2098.82,1693.19,3760.11
```

The command line tool must take a single argument, which is the data-frame processing operation we want the tool to perform. The CSV data for it to process will then be piped in through `stdin`. Multiple CSVs can be piped in, which is why CSVs must be terminated with a blank line. The output of the tool should always be a single data-frame, encoded as a CSV file, properly terminated with a final blank line.

Challenge parts

This challenge is split into three parts. So as not to take too much of your time, you are free to complete as many parts as you would like, but you should complete at least the first part.

Each part is another operation the command line tool can perform.

Note Since this is a programming *challenge*, the use of a data-frame processing library, such as `pandas`, `spark`, or `saddle`, is heavily discouraged. However, the use of libraries in general is encouraged.

1. The first operation your command line tool should be able to perform is adding data-frames together, chosen by providing the argument `+` to the tool. This addition should respect the row and column indices, adding values together when they have equal column headers and row headers. If a value only exists in one file, then only this value is returned, viz. nothing is added to it, but it exists in the output file. If a value exists in none of the input files, but exists in the outer-join of the file indices, then output the value `0.0`. For example, given two files:

   ```
   $ cat ex1.csv
   ,A,B
   Jan-16,100.0,200.0
   Feb-16,100.0,200.0

   $ cat ex2.csv
   ,A,C
   Feb-16,100.0,200.0
   Mar-16,,210.0
   ```

   If we call our command line tool `qs`, then we expect the following output from running it:

   ```
   $ cat ex1.csv ex2.csv | qs +
   ,"A","B","C"
   "Jan-16",100.0,200.0,0.0
   "Feb-16",200.0,200.0,200.0
   "Mar-16",0.0,0.0,210.0
   ```

   The order of rows/columns is **not** important, as long as the row and column indices are correct. The following output would also be okay:

   ```
   $ cat ex1.csv ex2.csv | qs +
   ,"C","A","B"
   ```

```
"Mar-16",210.0,0.0,0.0
"Jan-16",0.0,100.0,200.0
"Feb-16",200.0,200.0,200.0
```

Along with this challenge description you should have been provided four files: `bonuses-2016.csv`, `bonuses-2017.csv`, `payroll-2016.csv`, and `payroll-2017.csv`. These are filled with randomly generated values.

Here is the expected output (modulo row/column order) of using `qs +` on various combinations of these files:

```
$ cat payroll-2016.csv bonuses-2016.csv | qs +
,"A","B","C","D","E"
"Jan-16",2874.99,2713.23,1972.41,2312.73,0.0
"Feb-16",3205.4,1806.67,1625.87,2876.39,0.0
"Mar-16",2098.82,1693.19,3760.11,2442.27,0.0
"Apr-16",2222.22,3111.64,2610.13,2718.85,0.0
"May-16",1917.06,2862.88,3171.45,3061.55,0.0
"Jun-16",3678.95,3950.13,3463.36,1538.85,0.0
"Jul-16",1083.01,0.0,2607.69,3146.24,0.0
"Aug-16",3545.92,0.0,1831.38,2592.41,0.0
"Sep-16",2619.49,0.0,2753.77,1577.62,0.0
"Oct-16",2348.62,0.0,1591.76,1458.1,0.0
"Nov-16",2983.45,0.0,2284.48,3629.38,0.0
"Dec-16",4298.22,0.0,2242.48,5206.67,3698.07
```

Above is the total amount paid to each employee on each month in 2016. Now let's calculate the total amount paid to all employees across 2016 and 2017.

```
$ ((cat payroll-2016.csv bonuses-2016.csv | qs +) ; (cat payroll-2017.csv bonuses-2017.csv | qs +)) | qs +
,"A","B","C","D","E"
"Jan-16",2874.99,2713.23,1972.41,2312.73,0.0
"Feb-16",3205.4,1806.67,1625.87,2876.39,0.0
"Mar-16",2098.82,1693.19,3760.11,2442.27,0.0
"Apr-16",2222.22,3111.64,2610.13,2718.85,0.0
"May-16",1917.06,2862.88,3171.45,3061.55,0.0
"Jun-16",3678.95,3950.13,3463.36,1538.85,0.0
"Jul-16",1083.01,0.0,2607.69,3146.24,0.0
"Aug-16",3545.92,0.0,1831.38,2592.41,0.0
"Sep-16",2619.49,0.0,2753.77,1577.62,0.0
"Oct-16",2348.62,0.0,1591.76,1458.1,0.0
"Nov-16",2983.45,0.0,2284.48,3629.38,0.0
"Dec-16",4298.22,0.0,2242.48,5206.67,3698.07
"Jan-17",2826.09,0.0,2047.75,2423.1,2595.69
"Feb-17",1949.79,0.0,2202.55,3524.77,3846.41
"Mar-17",1060.89,0.0,1723.39,3781.86,1481.81
"Apr-17",1178.64,0.0,1334.1,2602.85,2491.78
"May-17",1280.61,0.0,1378.65,3923.34,1770.97
"Jun-17",2418.24,0.0,2111.53,2803.96,2245.28
"Jul-17",3708.48,0.0,2872.52,3787.41,3198.53
"Aug-17",3625.03,0.0,1834.23,3301.23,1129.99
"Sep-17",1176.75,0.0,0.0,2246.67,1998.69
"Oct-17",1000.99,0.0,0.0,1186.52,2816.14
"Nov-17",3636.62,0.0,0.0,2837.94,3112.99
"Dec-17",4241.5,0.0,0.0,2620.08,5322.25
```

2. In part 1. we implemented outer-join point-wise addition of data-frames. Now you should implement a "reduce" operation to get the sum of a data-frame across all rows. This operation should take a single data-frame from `stdin` and output a data-frame with a single row, which is the sum of all rows of the input, per column. For example (files are the same as in the previous part):

```
$ cat ex1.csv | qs sum
,"A","C"
"sum",200.0,400.0
```

We can use this to calculate the total amount paid to each employee in 2016:

```
$ cat payroll-2016.csv bonuses-2016.csv | qs + | qs sum
,"A","B","C","D","E"
"sum",32876.15,16137.74,29914.89,32561.06,3698.07
```

3. In part 2. you created a operation for taking the sum across all rows of a data-frame. Now you should implement an operation which will allow us to sum across all columns, and to take the sum of an entire data-frame in general: `transpose`. The transpose of a data-frame is a rotation of the data about the diagonal, so that the row index becomes the column index and the column index becomes the row index. This operation should take a single data-frame from `stdin` and output its transpose, e.g.

```
$ cat ex1.csv | qs transpose
,"Jan-16","Feb-16"
"A",100.0,100.0
"B",200.0,200.0
```

We can use this, along with `sum`, to calculate the total amount paid in each month in 2016:

```
$ cat payroll-2016.csv bonuses-2016.csv | qs + | qs transpose | qs sum
,"Jan-16","Feb-16","Mar-16","Apr-16","May-16","Jun-16","Jul-16","Aug-16","Sep-16","Oct-16","Nov-16","Dec-16"
"sum",9873.36,9514.33,9994.39,10662.84,11012.94,12631.29,6836.94,7969.71,6950.88,5398.48,8897.31,15445.44
```

Finally, let's calculate the total amount paid out in both 2016 and 2017:

```
$ ((cat payroll-2016.csv bonuses-2016.csv | qs +) ; (cat payroll-2017.csv bonuses-2017.csv | qs +)) | qs + |
qs sum | qs transpose | qs sum
,"sum"
"sum",225846.52
```

Submission instructions

Please submit an archive with all your source code, unit tests, build scripts/config files, etc., which you used to solve this problem. Programming languages I am happy to accept submissions in are listed below. I've put them in order of mild preference, but please use whichever language you are most comfortable with. Choice of language does *not* factor into our assessment of the challenge.

1. Haskell
2. Scala
3. F#
4. Java
5. C#
6. Python (with type hints)
7. C++
8. Python (without type hints)
9. C