R package tidytable: A visualization tool for multi-way tables

by

Xiaomei Yu

A research paper presented to the
University of Waterloo
in partial fulfillment of the requirements for the degree of
Master of Mathematics
in
Statistics

Waterloo, Ontario, Canada, 2015

Contents

Abstract

Tables can be regarded as a data exploration way. A well-presented table can make it easier for the reader to see the patterns or decide the next analysis step. Ehrenberg (1977) summarized several basic table visualization rules, we're trying to apply these rules using the R package: tidytable. Furthermore, our tidytable package generalizes these basic rules to high-dimensional space, so this package can deal with multi-way tables. We also propose a new idea: location. By finding a location and extract it, we can make the original table clearer and briefer, without affecting the patterns of the original table.

keywords: Table visualization Multi-way tables Location Order Swap Table unit

1 Introduction

1.1 Background

Graphics are widely known as a data visualization tool. Indeed, they can attract the reader more easily. However, Ehrenberg (1977) mentioned that, “graphs are of little use in communicating the quantitative aspects of the data, but they can highlight quantitative results (like that something has gone up, is a curve rather than a straight line, or is smaller than large). A graph can make the points more “graphic”, and hence graphs can be very useful at the beginning or end of an analysis.”

In the discussion article *Why tables are really much better than graphs* (2011), Andrew Gelman provides a tongue-in-cheek argument about graphical methods, from the standpoint of “a hypothetical old-school analytical statistician or social scientist”. He mentioned the real problem is that “graphs are inherently a way of implying results that are often not statistically significant”. While showing the aversion to graphs, he also expresses support to the tables. He gives his preferences of table layout, such as putting the exact numbers of results summary, a minimum of four significant digits, using the variable names provided by the computer program as row and column names etc.

At the same year of Gelman’s discussion article, Michael Friendly & Ernest Kwan (2011) gave a comment about this article from a psychological perspective. This comment article emphasizes that graphs and tables are both communication modes, which should be tailored to the audience to achieve the desired goal. They also assert that there is a dimension of cognition, where people can be divided to graph people or table people. This comment also calls attention to a brief note by Karl M. Dallenbach (1963) that,

- All the evidence obtained from the reproduction of the study mentioned here indicates that the graph method is ’better’ than the tabular. Tables, since graphs are based on them, are necessary, but they are like background rocks, heavy and interesting.

Graphs, on the other hand spice the reports; clarify them, and make them interesting and palatable. (Dallenbach 1963, p.702)

With regard to Gelman's preference that the exact number of results summary should be put into the table, Friendly & Kwan referred in their comment that “in many case either the coefficients in fitted model are meaningless without graphical display or their interpretation is exceedingly difficult to understand”, and “even pure table people cannot extract any sunlight from such tabular cucumbers”. This comment also mentioned some combination way of tables and graphs such as the semi-graphic display and the tableplots.

Richard A. Feinberg & Howard Wainer (2011) surveys display formats in the *Journal of Computational and Graphical statistics* during the period 2005 - 2010, and discover that the most dominant format was table. They mentioned brothers Farquhar's comment:

- The graphical method has considerable superiority for the exposition of statistical facts over the tabular. A heavy bank of figures is grievously wearisome to the eye, and the popular mind is as incapable of drawing any useful lessons from it as of extracting sunbeams from cucumbers. (Farquhar and Farquhar, 1891, p.55).

Feinberg & Wainer referred that, in the past data storage is an important role for tables; but in the modern world table is used for human eyes and human minds, so many of the various elements critical for an accurate archive may no longer be suitable. They also proposed that we should make the table more graphical, and they recommend several steps toward tabular improvement: rounding; using summary statistics and bolding, spacing them apart and/or separating them with a ruled line; and ordering.

1.2 Table visualization

Tables are a data exploration method if they're presented in an appropriate way. In Data Reduction (1975), Ehrenberg displays how to analyze the undigested table using various steps to see the patterns. He also gives a summary of guidelines that can be regarded as the basic rules of table visualization. For instance:

- Rule 1: Reduce number of digits, usually round to two significant or efficient digits for mental arithmetic.
- Rule 2: Order rows and columns by size, rearrange rows to have large numbers appear above small numbers, rearrange columns so that averages are strictly decreasing (or increasing) from left to right.
- Rule 3: Figures are easier to compare in columns, numbers that vary the least should appear in columns.
- Rule 4: Use averages (or medians) to help focus the eye over the array.
- Rule 5: Note dramatically exceptional values and exclude them from pattern summary calculations.
- Rule 6: Avoid introducing new variables or scales (e.g. totals) whenever possible.
- Rule 7: Figures to be compared should be close together.
- Rule 8: Use memorable self-explanatory symbols and labels.
- Rule 9: Separate different types of items/groups with white space or gridlines.
- Rule 10: Summarize irregular aspects of the data statistically, e.g. by average deviations from appropriate averages.

In his paper Ehrenberg (1977), Ehrenberg gives a discussion of those objections and problems having been raised from using some of those basic rules, which makes these rules more reasonable.

The tables (Murdoch, 2014) package developed a way to provide the user with good-looking tables on R, LaTeX or other formats, to do further analysis to find the patterns in the table. However, in terms of applying these basic rules, it still needs users to analyze the table manually.

SAS proc tabulate has the similar functions as tables package does; it uses SAS to produce nice looking tables with several types of format. Except for two-way tables, where the rows are the first way and the columns are the second way, SAS proc tabulate could also make three-way tables by spanning multiple blocks of two-dimensional table. But still, all the analysis work are left to be done by users, it can not give a neat and tidy table directly.

In terms of multi-way tables, in R ftable() function in stats package could produce 'flat' contigency tables with multiple variables, it could store high-dimension information. The user could specify the variables shown on the rows using $row.vars$ parameter and specify the variables on the columns by $col.vars$.

What we have done is trying to implement those table visualization rules automatically in R using the package called tidytable. It still needs the user's judgments at some places, but generally the tidytable package tries to do the user's work as much as possible and give a tidy table for the user to see the patterns or decide what the next analysis step is, after the user inputting the initial table numbers.

Here, we also propose a new idea, namely "location", in the table analysis, based on those situations where the numbers in the table look massive, but actually most of them share a same location, which can be removed to see the patterns more clearly and won't affect the original patterns of the table.

Furthermore, we consider multi-way tables; try to generate these basic rules to higher dimension, which requires our own explorations and creations, since these rules become much more complicated to match in higher dimension. Tables we have seen before are mainly two-way tables, but there are more complex tables where more than two dimensions of information are recorded in the table. These tables actually require implementation of these basic rules to improve the data presentation even more than the ordinary two-way tables do.

2 Constructing more meaningful tables

2.1 General Structure of tidy table analysis

Consider the following fictitious two-way table:

Table 1

	Q1	Q2	Q3	Q4
North	4102097.62	4102092.24	4102100.90	4102090.39
South	4102048.29	4102042.31	4102049.98	4102039.09
East	4102075.23	4102075.16	4102100.11	4102074.23
West	4102049.69	4102057.21	4102080.19	4102051.09

As presented, it is difficult to see what patterns, if any, appear in the data. For example, all of the numbers are small variations from the relatively large number 4102000. It might make sense then to begin by removing this value from all elements in the table. The resulting table would be:

Table 2

	Q1	Q2	Q3	Q4
North	97.62	92.24	100.90	90.39
South	48.29	42.31	49.98	39.09
East	75.23	75.16	100.11	74.23
West	49.69	57.21	80.19	51.09

In this case, we can regard 4102000 as a "location", by finding and removing the location, the table has less digits, this achieves part of the purpose of Rule 1 in Introduction section: reduce number of digits. The resulting Table 2 is actually one example in Data Reduction (1975). This two-way table records the sales data in four areas (North, East, West and South) and four three-month periods (Q1, Q2, Q3 and Q4) in 1969. After removing the location, there are still too many digits in the table, so we round the table to two significant digits, as showed in Table 3.

Table 3

	Q1	Q2	Q3	Q4
North	98	92	100	90
South	48	42	50	39
East	75	75	100	74
West	50	57	80	51

Now without the distraction of too many digits, the table becomes much succincter. We also achieve the remaining part of Rule 1: round to 2 significant digits.

Then we might be interested in which of the four areas has the largest sales outcome and which has the smallest outcome. Since each area has four sales records corresponding to four quarters in 1969, we need to combine these four numbers as a summary to compare among the four areas. For example, we can use the average or the median as the summary. If we use the average as the summary of each area (row) and then order the rows based on this average, Table 3 becomes as follows:

Table 4

	Q1	Q2	Q3	Q4
North	98	92	100	90
East	75	75	100	74
West	50	57	80	51
South	48	42	50	39

So based on the average sales data in 1969, the North area has the largest sales and the South area has the smallest sales outcome. We can also order the columns to see which quarter has the largest sales, but some people also prefer to keep the chronological order, so here we don't order the columns.

We may also find that for each row, the numbers don't differ much. We try to swap the rows and columns, see Table 5. Now we can see how much easier it is to scan down the columns to see the little variability in the leading digit. Also the exceptions now stand out more easily, namely the 100 in the column East and the 80 in the column West, similarly the Q3 row overall.

Table 5

	North	East	West	South
Q1	98	75	50	48
Q2	92	75	57	42
Q3	100	100	80	50
Q4	90	74	51	39

Following the above steps, we gradually transform Table 1 to Table 5. Patterns of the originally noisy data become clearer and clearer. These are thegeneral steps of our tidytable() function, but in the function we choose to round the table to two significant digits at the last step, we use original data during the intermediate analysis for the purpose of precision. At the last step, we also try to find a unit of the table.

The general structure of our tidytable analysis can be summarized as Figure 1.

2.2 Get the location

The more digits the number has, the harder it is for the reader to absorb information effectively. Given a table with long-digit numbers, we need to reduce the number of digits to make the patterns clearer. Reducing digits can not only be achieved by rounding, subtracting a location is a more effective way, given the condition that most of the numbers in the table share a similar location. Briefly speaking, the location of a table is a number constructed by several digits shared by all the table numbers. By subtracting the location, the table will be tidier; and it won't affect patterns of the data, subtracting a same number won't affect the variability (such as standard deviation) of a sample. We can add the location back

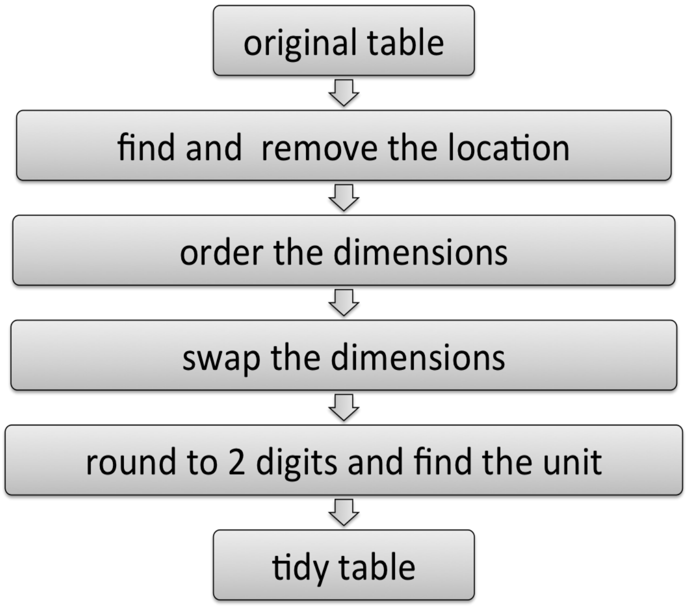

original table
find and remove the location
order the dimensions
swap the dimensions
round to 2 digits and find the unit
tidy table

Figure 1

when summarizing the final patterns. In our package, getLocation() function can find the location of a sample of data. For the example Table 1 dispalyed in the beginning of Section 2, getLocation() will give 4102000 as the location. Let's look at some other examples first.

```
library(tidytable)
table1
```

```
##            C          D
## A 1111110000 1346578900
## B 1134516000 1234567890
```

```
getLocation(table1)
```

```
## [1] 1.1e+09
```

```
table1 - getLocation(table1)
```

```
##          C         D
## A 11110000 246578900
## B 34516000 134567890
```

By removing the location, we reduce the number of digits from 10 to 8.

```
table2
```

```
## [1] 0.7999 0.7998 0.7998 0.7997
```

```
getLocation(table2)
```

```
## [1] 0.799
```

```
table2 - getLocation(table2)
```

```
## [1] 0.0009 0.0008 0.0008 0.0007
```

In **table2** example, we reduce the number of digits from 4 to 1 by removing the location. Here, we extract 0.799 as the location not 0.7997. If we use 0.7997 as the location, then we will get the analyzed sample as follows:

```
table2 - 0.7997
```

```
## [1] 0.0002 0.0001 0.0001 0.0000
```

Location doesn't mean the smallest number in the table; otherwise those smallest numbers will become zeros after removing the location. We still want to keep the last digit of the numbers, because sometimes the last digits are important for visualizing the patterns. Although in some cases, the last so many digits will still become zero due to the final rounding, but here we still want to keep the last digit in case it's important in further analysis.

getLocation() can also deal with situations of negative numbers.

```
table3
```

```
## [1] -49991234 -49983454 -50013333 -49923333
```

```
getLocation(table3)
```

```
## [1] -49900000
```

```
table3 - getLocation(table3)
```

```
## [1]  -91234  -83454 -113333  -23333
```

```
table4
```

```
## [1]  4220  4232 -6332  4578
```

```
getLocation(table4)
```

```
## [1] 0
```

getLocation() can deal with multi-way tables. In summary, when we input a sample of data, getLocation() can help us find a location shared by all the numbers in the sample.

The algorithm of getLocation() involves checking the exponent and value of a digit. Here, "value" means the value of the digit, and "exponent" means the magnitude (power of 10) of the digit. For example, for 5423, the value of the first non-zero digit is 5, and the exponent of it is 4; the value of the second digit is 4, and the exponent is 3. Similarly, 0.00702, the value of the first non-zero digit is 7, and the exponent is -3; the value of second digit is 0, and the exponent is -4. Then the algorithm of getLocation() can be described as:

1. Start with the first non-zero digit of numbers in the table, if the value and exponent of the first non-zero digits are all the same, then we record the value and the exponent of it.
2. Continue to check the second digit to see if it's shared by all the numbers in the table, if it is, then record the second digit's value and exponent.
3. Keep checking the value and exponent of each digit until you find one digit that is not shared by all the numbers. Then those digits in-common recorded can be used to construct the location.

In order to decompose the value and the exponent of each digit, we need to first decompose the coefficient and the exponent of each number. For example, the scientific notation for 5423 is 5.423e+03, then the coefficient of 5423 is 5.423, and the exponent of it is 3. The scientific notation of 0.00702 is 7.02e-03, the coefficient is 7.02 and the exponent is -3. Therefore, we need a function to extract the coefficient and exponent, which are those two numbers before and after the letter "e" in the scientific notation. The SciNotation() function is written to achieve this aim.

When checking the value and exponent of each digit, it's not necessarily to be as strict as "exactly the same". We could also allow some extreme values to happen, but if most of the numbers share a same digit, that digit can also be extracted to construct the location. cSpread() function can be used to calculate the center spread of a sample of data. By setting a fraction, say 0.8, we could calculate the 80% center spread of the sample by ignoring 10% of the data in both ends. In cSpread() function, we use the range to represent the spread. In the getLocation() algorithm, we can use cSpread() to calculate the center spreads of exponents and the values of a digit, if the center spreads are less than some thresholds, we can extract the digit to construct the location. In this way, getLocation() function can be more robust by not considering extreme values.

During the construction of the function getLocation(), we encountered some problems of machine floating point system.

The paper *Why every computer scientist should know about floating-point arithmetic* (1991) illustrates this problem that, "Squeezing infinitely many real numbers into a finite number of bits requires an approximate representation. Although there are infinitely many integers, in most programs the result of integer computations can be stored in 32 bits. In construst, given any fixed number of bits, most calculations with real numbers will produce quantities that can not be exactly represented using that many bits. Therefore the result of a floating-point calculation must often be rounded in order to fit back into its finate representation." For example, the number $\frac{1}{3}$ is a repeating decimal 0.33333..., but in R,

```
print(1/3,digits=22)
```

```
## [1] 0.3333333333333333148296
```

Similarly,

```
print(0.7,digits=22)
```

```
## [1] 0.6999999999999999555911
```

So some inaccuracies might happen here. We try to avoid these inaccuracies as much as possible, by transforming those rational numbers to integers, then follow the way of dealing with integers. After the analysis, we transform the numbers back.

Location can be understood more generally. When we compare different things, removing the common part they share can make the difference more obvious. In this way, our attention can also be focused to the difference, instead of letting the difference submerge within the huge common part.

2.3 Order the dimensions

Ordering can provide a general ranking of the layers in a dimension. Like the example displayed in Section 2.1, ordering the rows can give us a rough idea about the sales ranking of four areas. As Rule 3 says, order the rows and columns by size. Here, "size" is a summary of a subsample of the data. In the table case, the table contains a whole sample of data; with regard to one dimension of the data (like the areas in sales data examples), different levels (like the four areas: North, South, East and West) divide the whole sample into several subgroups. We use a summary to represent the data in each subsample, and then rank the subsamples based on their corresponding summaries. The idea can also be described visually. Given a sample of data with two dimensions, it can be shown as a square as Figure 2.

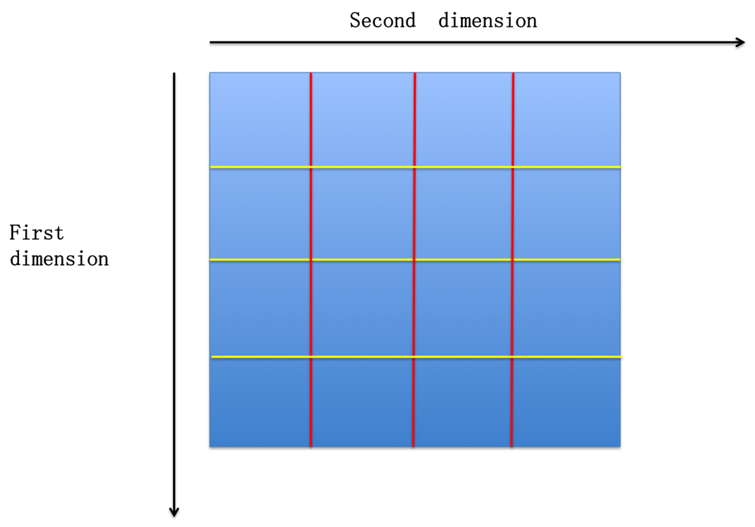


Figure 2

The first dimension of the data has four levels, and the second dimension has also four levels. This sample of data can also be regarded as a two-way table, like the sales data example.

Now if we want to order the first dimension, according to the four levels of the first dimension, the square can be divided into four layers, each layer can be denoted as

$$(i, \quad \cdot) \quad , \qquad i = 1, 2, 3, 4$$

Here, $\cdot$ represents all the data of level i in the second dimension, so it also represents all the data of level i in the whole sample since there are 2 dimensions in total in the data. For example the first layer $(1, \cdot)$ can be shown as the blue colored part in Figure 3.

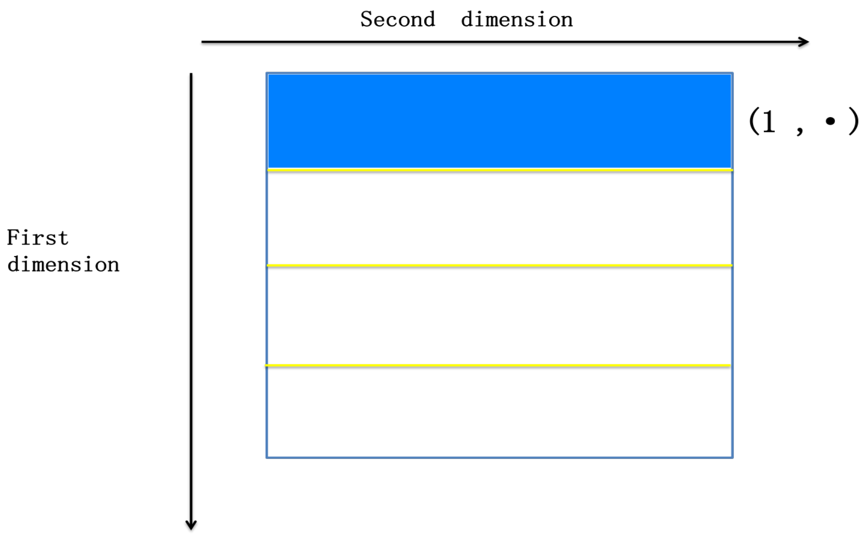


Figure 3

To order the layers in the first dimension, we need to get a summary of each layer. The layers of the first dimension (rows) can be represented as:

$$(i, \quad \cdot) \quad , \qquad i = 1, 2, 3, 4$$

For instance, let the average be the summary function, then the summary of each layer in the first dimension can be denoted as:

$$mean(i, \quad \cdot) \quad , \qquad i = 1, 2, 3, 4$$

Similarly, the layers of second dimension (columns) can be denoted as

$$(\cdot \,, \quad j) \quad , \qquad j = 1, 2, 3, 4$$

And we need to calculate the summary of each layer to order them.

Next let's think about data with three dimensions. It can be visually described as a cube, see Figure 4.

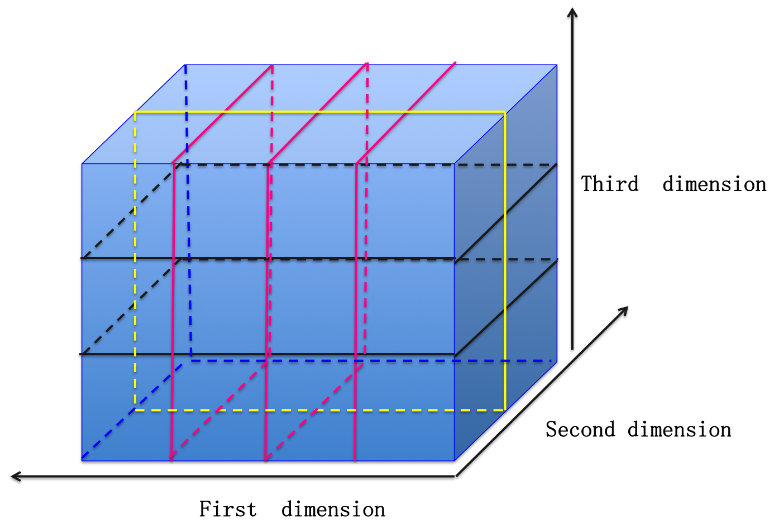


Figure 4

The number of levels corresponding to the three dimensions are 4,2,3. Now the layers of the first dimension are denoted as:

$$(i, \quad \cdot, \quad \cdot) \quad , \qquad i = 1, 2, 3, 4$$

The first $\cdot$ corresponds to all the data of level i in the second dimension, the second $\cdot$ stands for all the data of level i in the third dimension. All together, $(i, \quad \cdot, \quad \cdot)$ stands for all the data of level i in the whole sample. For example, $(1, \quad \cdot, \quad \cdot)$ can be shown as the blue colored part in Figure 5.

Then ordering the first dimension is to rearrange the four cuboids. Similarly, the layers of the second dimension can be regarded as two cuboids in the cube, which are denoted as

$$(\cdot \quad , j, \quad \cdot) \quad , \qquad j = 1, 2$$

For example, the first layer (cuboid) is showed in Figure 6:

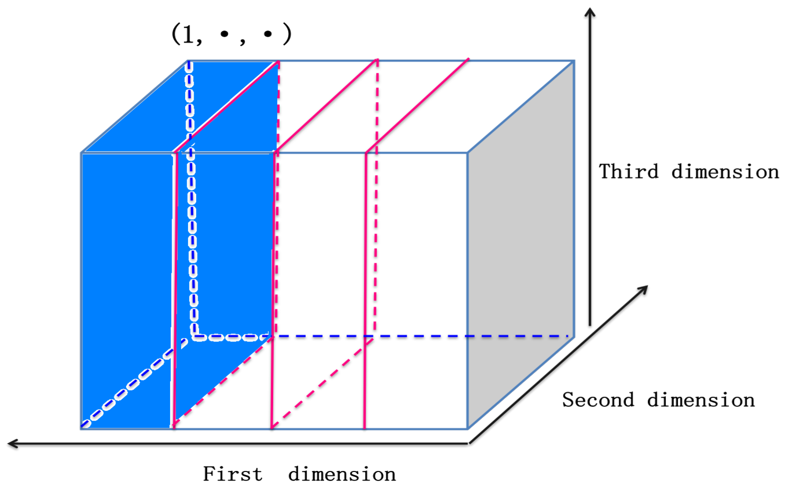


Figure 5

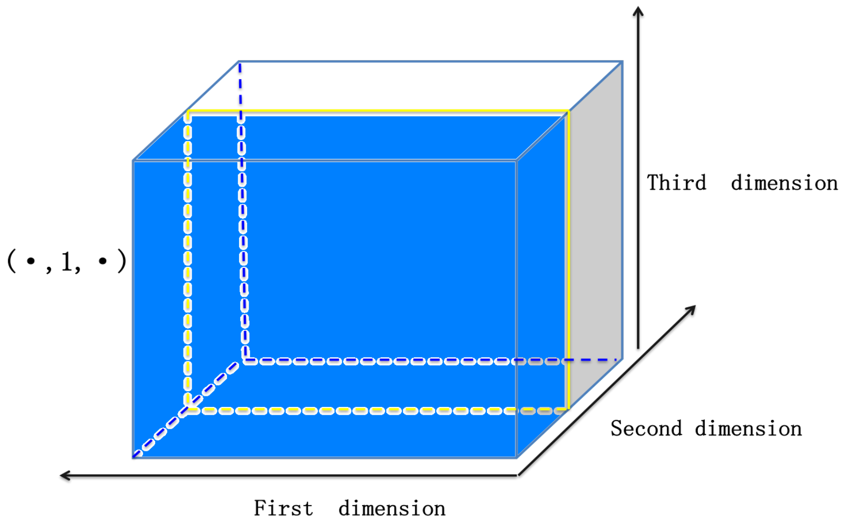


Figure 6

And the layers of the third dimension are denoted as

$$(\cdot \ ,\cdot \ , \ k) \quad , \qquad k = 1, 2, 3$$

Visually, $(\cdot \ ,\cdot \ , \ 1)$ can be shown as the blue colored part in Figure 7.

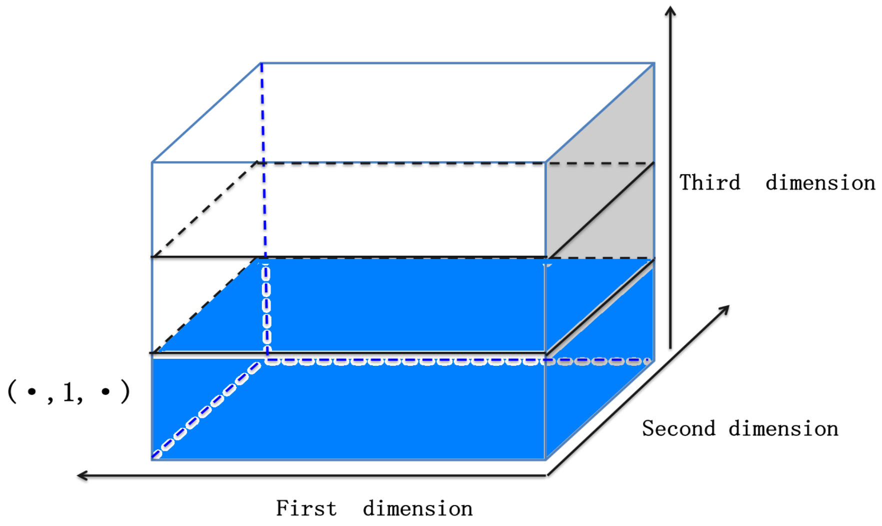


Figure 7

Spatially, ordering the layers of one dimension is ordering the cuboids in a cube.

Now let's look at an example to see what reordering the layers means in R console layout.

HairEyeColor is a three-way table that contains hair and eye color and sex information of 592 statistics students.

```
HairEyeColor
```

```
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red       10   10     7     7
##   Blond      3   30     5     8
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

The first dimension is Hair color, the second dimension is Eye color and the third is Sex. The first layer of the first dimension (Hair Color) $(1, \cdot, \cdot)$ is all the data with Hair color = Brown, which contains the first row of the first block and the first row of the second block. Similarly, the second layer $(2, \cdot, \cdot)$ is the second rows of two blocks. As for the second dimension, the first layer contains the first column of the first block and the first column of the second block. The first layer of the third dimension is the first block, and the second layer is the second block.

orderDim() function can order the layers of a specific dimension. For example, for the HairEyeColor example, ordering the first dimension will give the following result:

```
orderDim(HairEyeColor,dim=1)
```

```
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Brown    53   50    25    15
##   Blond     3   30     5     8
##   Black    32   11    10     3
##   Red      10   10     7     7
##
## , , Sex = Female
##
##         Eye
## Hair    Brown Blue Hazel Green
##   Brown    66   34    29    14
##   Blond     4   64     5     8
##   Black    36    9     5     2
##   Red      16    7     7     7
```

The rows are re-arranged; moreover, the rows of two blocks are rearranged together. So overall, people with brown hair form the largest population in the sample. Then we continue to order the second dimension,

```r
orderDim(orderDim(HairEyeColor,dim=1), dim=2)
```

```
## , , Sex = Male
##
##         Eye
## Hair    Brown Blue Hazel Green
##   Brown    53   50    25    15
##   Blond     3   30     5     8
##   Black    32   11    10     3
##   Red      10   10     7     7
```

```
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Brown    66   34    29    14
##   Blond     4   64     5     8
##   Black    36    9     5     2
##   Red      16    7     7     7
```

The columns haven't changed, so the original order is already in decreasing ranking. Lastly, we order the third dimension,

```
orderDim(orderDim(orderDim(HairEyeColor,dim=1), dim=2),dim=3)
```

```
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Brown    66   34    29    14
##   Blond     4   64     5     8
##   Black    36    9     5     2
##   Red      16    7     7     7
##
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Brown    53   50    25    15
##   Blond     3   30     5     8
##   Black    32   11    10     3
##   Red      10   10     7     7
```

The two blocks are re-arranged, so in this sample, there are more females than males.

In the orderDim() function, you can change the summary type, the examples we displayed above use summary = mean. You can also specify which dimension to order by specifying the *dim* parameter.

In the orderDim() function, the default order type (direction) is "decreasing", following Ehrenberg (1977)'s advice. He points out that "for the rows of a table, showing the larger numbers above the smaller numbers helps because we are used to doing mental subtraction that way." For example,

$$\begin{array}{c}\hline\hline 450\\ 320\\ \hline\end{array} \text{ compared to } \begin{array}{c}\hline\hline 320\\ 450\\ \hline\end{array}.$$

It's easier to mentally subtract 320 from 450 while we scanning down. With more digits, this effect is even stronger:

$$\begin{array}{c}\hline\hline 447\\ 318\\ \hline\end{array} \text{ compared to } \begin{array}{c}\hline\hline 318\\ 447\\ \hline\end{array}.$$

Facilitating such mental arithmetic is important when one is scanning large sets of data.

The user can specify the order type as increasing if the user prefers, by setting the parameter $decreasing = FALSE$.

2.4 Swap the dimensions

In tidytable package, swapping the dimensions is achieved by the function swap(). In Section 2.1, we have swapped the two dimensions in the sales data example. Let's look at another fictitious two-way table first:

```
table5
```

```
##         1       2       3        4
## A  0.7999  0.7998  0.7998   0.7997
## B  3.7999  3.7824  3.7662   3.7223
## C  0.3000  1.2000  4.9000 145.7000
## D 20.7990 20.6990 20.8990 145.6990
## E 35.3000 34.5000 33.6000  34.7000
```

We notice that in some rows, the numbers are quite similar. If we swap the dimensions, **table5** becomes

```
swap(table5)
```

```
## $swappedTable
##        A      B     C       D    E
## 1 0.7999 3.7999   0.3  20.799 35.3
## 2 0.7998 3.7824   1.2  20.699 34.5
## 3 0.7998 3.7662   4.9  20.899 33.6
## 4 0.7997 3.7223 145.7 145.699 34.7
##
## $newDimOrder
## [1] 2 1
```

The "swappedTable" is the table we get after swapping the dimensions of the original table, "newDimOrder" records the re-ordered dimensions, so in this case the original second dimension becomes the first dimension of the swapped table. Now the patterns become more obvious, and the exceptions also stand out, such as the "145.699" in the fourth column and the third column overall. Ehrenberg (1977) explained “figures are easier to follow reading down a column than across a row, especially for a larger number of items.” By arranging the dimension with less variability as columns, people can scan down the columns to find the patterns more easily, in other words, as Rule 3 says, “Figures are easier to compare in columns, numbers that vary the least should appear in columns.”

In order to swap the dimensions, we need to compare the variability of different dimensions. In the two-way table case, we need to compare the variability of the rows and thecolumns, then set the dimension with less variability as the columns. We first need to figure out how to summarize the variability of one dimension. Each dimension has several layers, each layer is a subsample of data, as we explained before. So firstly we need to summarize each subsample's variability, we use $layerSummary$ to denote the summary function to calculate each layer's variability. After getting the summary of each layer, we need to combine these summaries to get the overall variability of the dimension, so here we need another summary function, denoted as dimSummary. Then the variability of a dimension can be formulated as:

$$dimSummary_m \left(\; layerSummary_n \;\; (\;\; X_{mn} \;\;) \;\; \right) \qquad\qquad m = 1, ..., M, \;\; n = 1, ..., N_m$$

Here, X_{mn} is the n^{th} element of the m^{th} layer, M is the total number of layers in the dimension, N_m is the total number of elements in the m^{th} layer.

For two-way tables case, there are only two dimensions, so we only need to compare the variability of rows and columns. For instance, the calculation of the rows' variability can be processed as in Figure 8:

Here $(i, \;\; \cdot)$ can be visually shown as the sub-rectangle of a rectangle, as showed in Figure 9, here we use the example of sales data, i.e. the 4×4 two-way table we introduced in Section 2.1. We calculate the variability of the information in each sub-rectangle using $layerSummary$, then combine these variabilities of sub-rectangles to get the variability of the whole rectangle using $dimSummary$, we can obtain the variability of the rows. This gives you a visually idea of the above process in Figure 8.

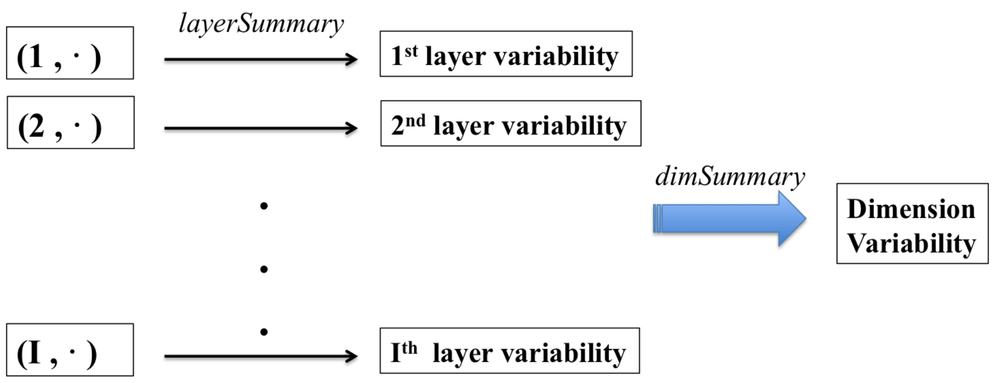


Figure 8

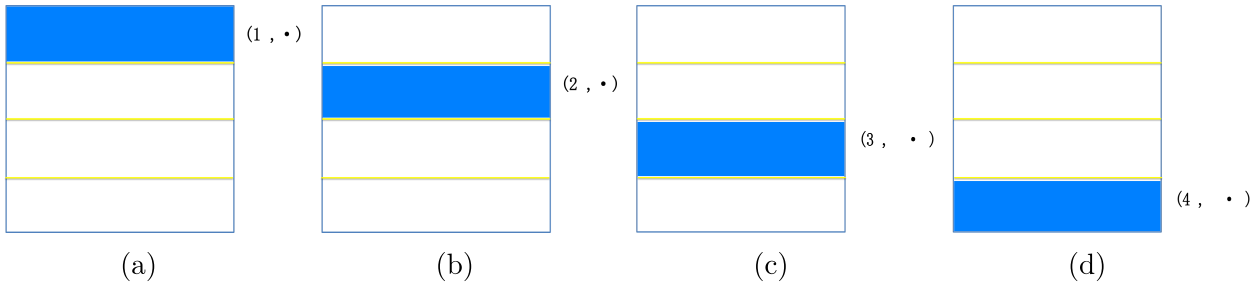


Figure 9

Similarly, we can calculate the columns' variability. Then if the rows' variability is smaller than the columns', swap rows and columns; otherwise, keep the original dimensions setting.

In statistics, there are some quantities to measure the variability, such as variance, standard deviation, range etc. Since the data in the table sometimes might include extreme values, a robust statistical quantity is needed to measure the variability. In our package, we use mad (median absolute deviation) to be the $layerSummary$ and median to be $dimSummary$. The mad is a very robust measure of variability. Let $x_1, x_2, ..., x_n$ be a dataset, then the mad is defined as the median of the absolute deviations from the data's median:

$$mad = median_i(|x_i - median_j(x_j)|)$$

The user can specify the $layerSummary$ by $valueFun$ parameter and specify the $dimSummary$ by $chooseFun$ parameter in the function swap().

In order to better illustrate the variability of a dimension, here we use the example of the sales data. (Here we use $layerSummary = mad$, $dimSummary = median$)

```
sales
```

```
##          Q1    Q2     Q3    Q4
## North 97.62 92.24 100.90 90.39
## South 48.29 42.31  49.98 39.09
## East  75.23 75.16 100.11 74.23
## West  49.69 57.21  80.19 51.09
```

First we calculate the variability of the first dimension, the mad values of each row are as follows:

```
mad(sales[1,])
```

```
## [1] 5.359599
```

```
mad(sales[2,])
```

```
## [1] 5.685771
```

```
mad(sales[3,])
```

```
## [1] 0.7413
```

```
mad(sales[4,])
```

```
## [1] 5.574576
```

The above results are variability of each layer, then we combine them together using the median:

```r
median(c(mad(sales[1,]),mad(sales[2,]),mad(sales[3,]),mad(sales[4,])))
```

```
## [1] 5.467088
```

So the variability of the first dimension is 5.467088. Similarly, we can calculate the variability of the second dimension:

```r
median(c(mad(sales[,1]),mad(sales[,2]),mad(sales[,3]),mad(sales[,4])))
```

```
## [1] 22.16116
```

The variability of the second dimension is 22.16116, which is larger than that of the first dimension, so we should swap the two dimensions to let the dimension with less variability be columns (second dimension).

2.4.1 Swap dimensions of high-dimensional data

For multi-way tables with $D(>= 3)$ dimensions, swaping the dimensions is not as simple as ordering the dimensions based on their variabilities. We need to consider the multi-way table layout on screen. For example, in the R console, we have got an initial idea about multi-way table layout from the three-way table HairEyeColor example, in Section 2.3. Let's look at the example of a four-way table, Titanic, which records the survival of passengers on the Titanic.

```r
Titanic
```

```
## , , Age = Child, Survived = No
##
##       Sex
## Class  Male Female
##   1st     0      0
##   2nd     0      0
##   3rd    35     17
##   Crew    0      0
##
## , , Age = Adult, Survived = No
##
##       Sex
## Class  Male Female
##   1st   118      4
##   2nd   154     13
##   3rd   387     89
##   Crew  670      3
##
## , , Age = Child, Survived = Yes
##
##       Sex
## Class  Male Female
##   1st     5      1
##   2nd    11     13
##   3rd    13     14
##   Crew    0      0
##
## , , Age = Adult, Survived = Yes
##
##       Sex
## Class  Male Female
##   1st    57    140
##    2nd     14      80
##    3rd     75      76
##    Crew  192      20
```

There are four dimensions in this data: Class, Sex, Age, Survived, the corresponding numbers of layers are (4, 2, 2, 2). Class is the first dimension and Sex is the second dimension, so in each block (i.e. each 4×2 two-way table), the rows are Class information and the columns are the Sex information. For each layer of the first dimension, say the first layer, i.e. the 1st class, it contains all the information recorded on the first row of each block. Similarly, for the second dimension (the columns), the female survival information consists of the second columns of each block. The third dimension is Age, with two layers, child and adult; all the information of child is stored in the first and third blocks, and all information of adult is in the second and fourth blocks. The last dimension is Survived, Yes or No. The first two blocks of the table record the information of survived=Yes, and the last two blocks record the information of survived=No.

Generally, we can see that the rows and columns are two dimensions which can be compared the easiest, since they are showed on each block, we can read down the blocks naturally to compare the figures. Within the first two dimensions, again, the columns are easier to compare than the rows, as Rule 3 says. The third dimension is easier to compare than the fourth dimension, since the information of different layers in the third dimension is put closer than in the fourth dimension. By other multi-way table examples, we could arrive at a similar conclusion: the first two dimensions have the most beneficial positions to find the patterns, and columns are easier to be compared than rows; for those dimensions whose order are > 2, the smaller the order is, the more visualization convenience.

When dealing with multi-way tables, we created a series of swapping principles as follows:

- Firstly, get all the two-dimensional combinations of all D dimensions, the number of two-dimensional combinations is $\binom{D}{2}$. Then calculate the variability of each two-dimensional combination. The two-dimensional combination with the least variability should be the first two dimensions of the swapped table, we denote them as i, j. Here we don't decide which of i and j should be columns, we leave this as the last step.

- Secondly, we decide on the left $D-2$ dimensions according to the variabilities of three-way dimension combinations. These combinations is constructed by the already decided first two dimensions i, j and each of the remaining $D-2$ dimensions, so totally $\binom{D-2}{1} = D-2$ combinations. Then we calculate the variability of each of these three-dimension combinations and find the combination with the least variability. Then in this three-dimensional combination, the dimension except for i and j will be the third dimension of the swapped table, denoted as k. Similarly, we determine the fourth dimension by picking the combination with the least variability from those four-way dimension combinations constructed by the first 3 dimensions i, j ,k and each of the left $\binom{D-3}{1} = D-3$ dimensions. We follow this way to determine the order of the remaining dimensions .

- Lastly, after having decided the general order of all the dimensions, we come back to decide about the order within the first two dimensions, i.e. which one is the columns and which one is the rows. For each one of the first two dimensions i and j, we consider the combination of it with the left $D-2$ dimensions, which combination has the smallest variability, then the corresponding dimension will be the columns (i.e. the second dimension), the other one will be the rows (i.e., the first dimension).

We decide the above swapping principles also based on considerations as follows:

As we illustrated in Section 2.3, for each dimension of a table, it contains several layers. We need to combine the variability of each layer to get an overall measure of the variability of one dimension, which can be described as follows:

$$dimSummary_m \Bigg(\; layerSummary_n \; (\; X_{mn} \;) \; \Bigg) \qquad m = 1, ..., M, \;\; n = 1, ..., N_m$$

We have shown the process of calculating the variability in a two-way table case. Now in the multi-way table case, we first consider the three-way table case. The layers of the first dimension can be shown as follows, where I is the total number of layers (levels) in the first dimension.

$$(i, \quad \cdot, \quad \cdot) \quad , \qquad i = 1, 2, ..., I$$

Then the layers of the second dimension can be shown as follows, where J is the total number of layers (levels) in the second dimension.

$$(\cdot \quad , j, \quad \cdot) \quad , \qquad j = 1, 2, ..., J$$

The layers of the third dimension can be shown as follows, where K is the total number of layers (levels) in the third dimension.

$$(\cdot \quad , \cdot \quad , \quad k) \quad , \qquad k = 1, 2, ..., K$$

For instance, the variability calculation of the third dimension can be processed as Figure 10:

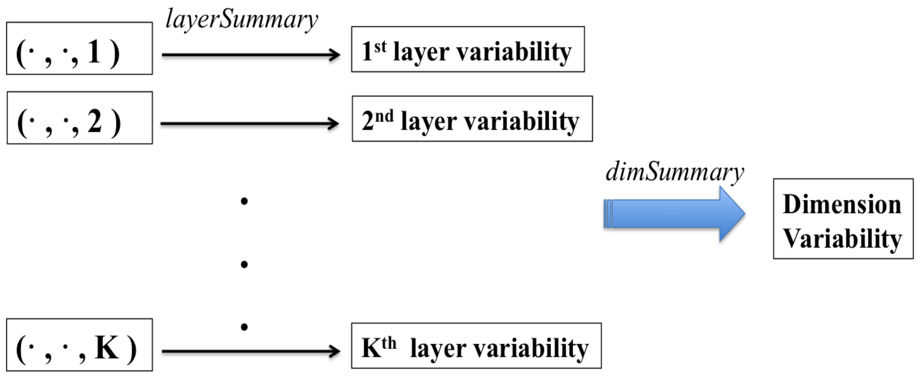


Figure 10

We use the example we shown in Section 2.3 to give you a visual idea of the above process. The three-way table has three dimensions with corresponding number of levels is (4,2,3), it can be showed as a three-dimensional space as Figure 4.

For instance, in order to calculate the variability of the third dimension in this table, we first calculate the variability of the first layer, i.e. shaded area of Figure 11(a), denoted as $layerSummary_1$; then calculate the variability of the second layer, which is the shaded area in Figure 11(b), is denoted as $layerSummary_2$; and the variability of the third layer, $layerSummary_3$, corresponding to data in shaded area of Figure 11(c). Then the variability of the third dimension is the $dimSummary$ of $layerSummary_1$, $layerSummary_2$ and $layerSummary_3$.

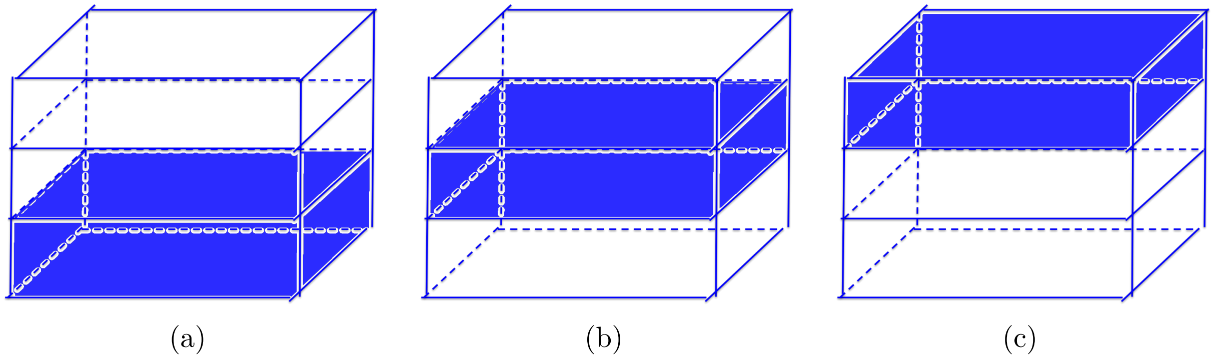


Figure 11

When there are more than 2 dimensions in the table, considering only the variability of one dimension is not enough. Since the dimensions are usually combined together to determine the final display of numbers in the table. So when we arrange the order of dimensions, we need to consider the variability of a combination of several dimensions. For example, for a four-dimensional table, if we want to calculate the variability of two dimensions' combination, say the second and the fourth dimension, the process can be described as in Figure 12. Here, there are J layers in the second dimension and L layers in the fourth dimension, so the combination of these 2 dimensions has $I * J$ layers.

Here we use a four-dimensional data example to give a visual display of how to calculate the variability of combination of dimensions. A four-way table can be displayed as a four-dimensional space in Figure 13. As last example, the first, second, third dimension has 4, 2, 3 layers. There is also the fourth dimension with 2 layers, and each layer is a cube.

Now let's consider the combination of the second and fourth dimension. Since the second dimension has 2 layers and the third dimension has 2 layers, the combination of these 2

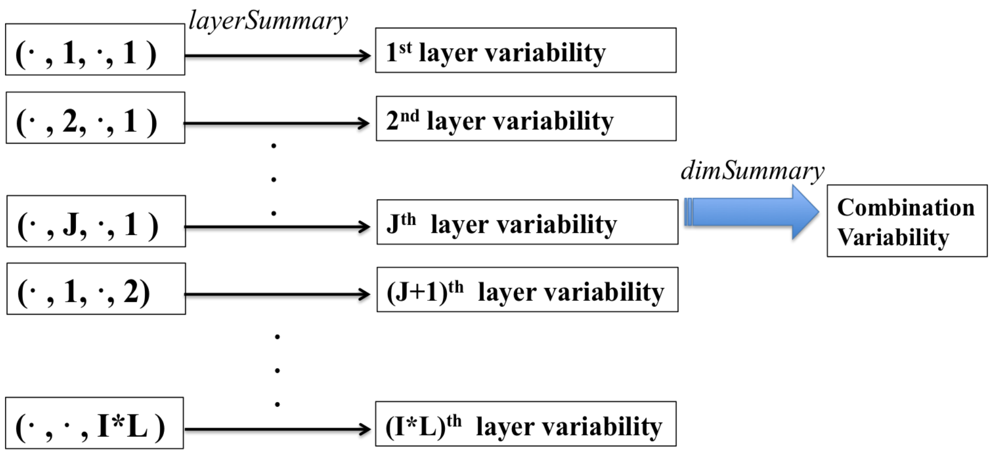


Figure 12

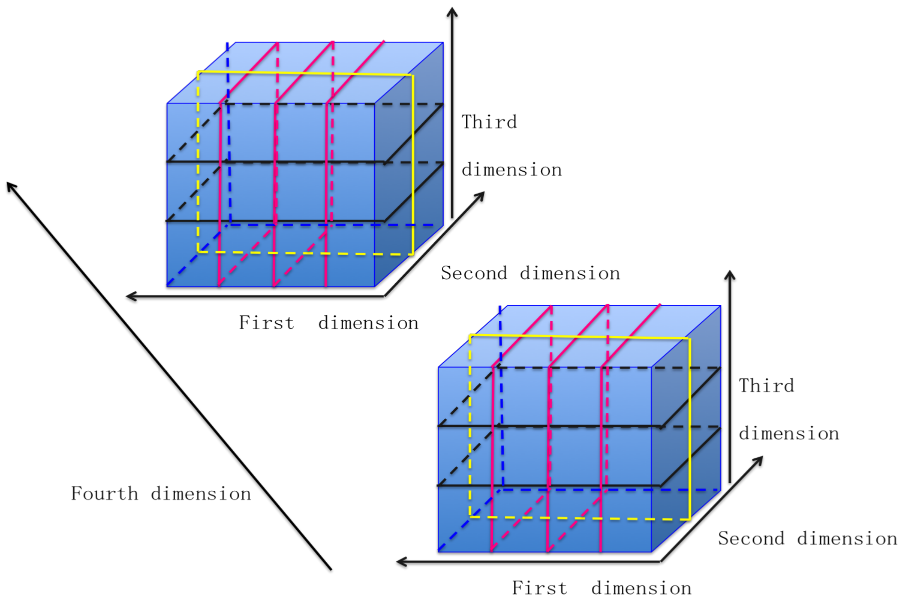


Figure 13

dimensions should have 4 layers. They are shown as (a), (b), (c) and (d) in the Figure 14, as the shaded area. Then the variability of the combination of dimension 2 and 4 should be the $dimSummary$ of the four layers' $layerSummary$ values.

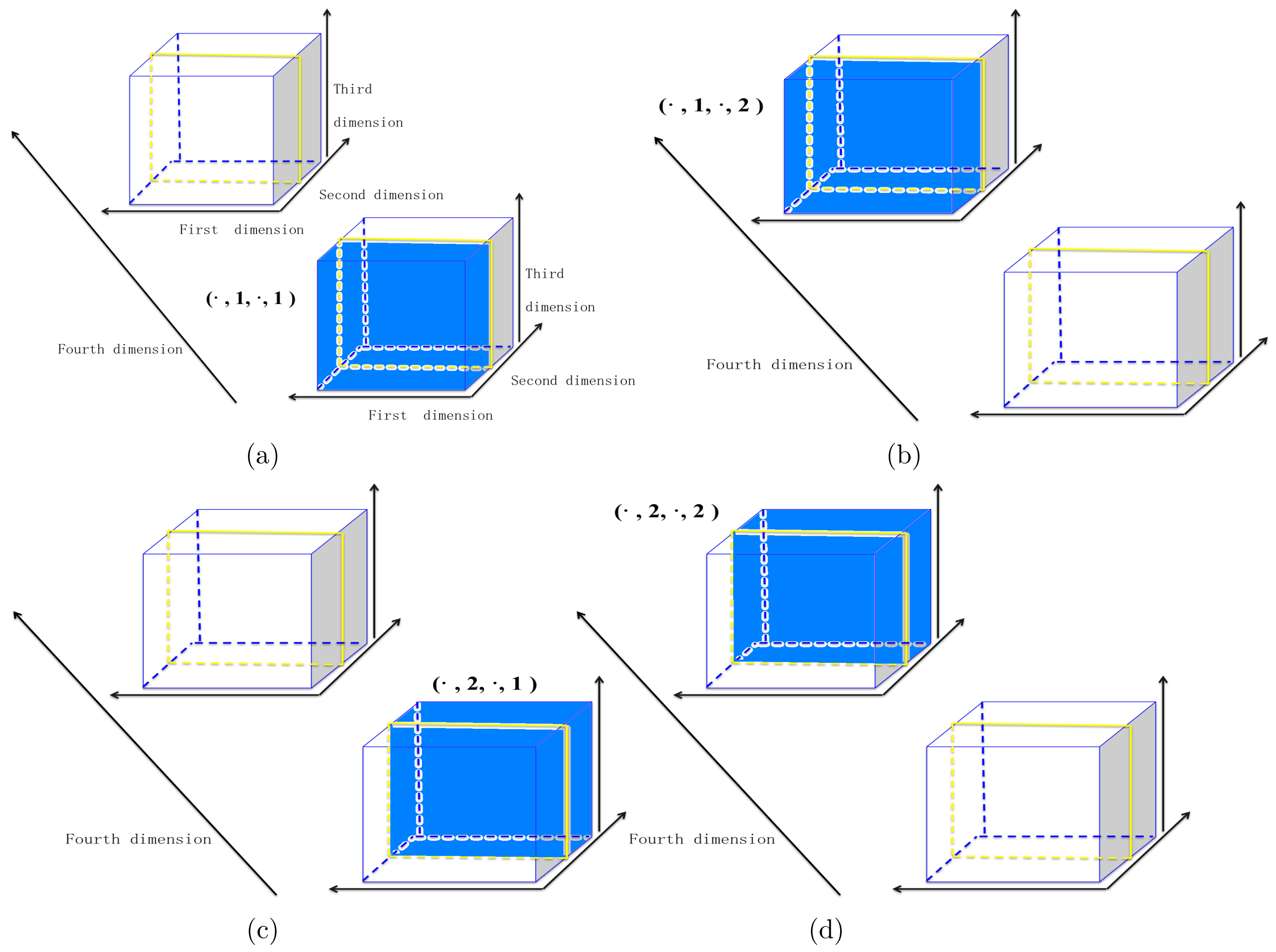


Figure 14

Generally, in a high-dimensional data set, the layers of the p^{th} dimension could be denoted as:

$$(\ldots, \quad \cdot, q, \quad \cdot, \ldots) \quad , \qquad i = 1, 2, \ldots, Q$$

Here Q is the total number of layers in the p^{th} dimension. "..." represents all the information of those dimensions before $q-1$ dimension and after $q+1$ dimension, actually "..." represents those omitted $\cdot$ s.

The variability calculation of the p^{th} dimension can be generalized as in Figure 15.

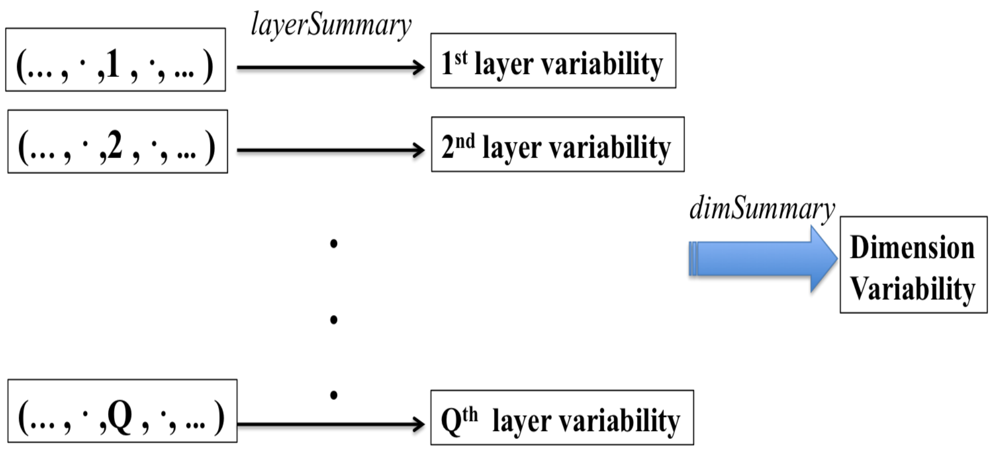


Figure 15

The variability calculation of the combination is quite similar to the variability of one dimension, but here each layer is denoted by a combination of dimension levels. For the combination of n dimensions, the number of layers of that combination is

$$\prod_{i=1}^{n} Q_i$$

where Q_i is the total number of layers of the i^{th} dimension in the combination. For example, we want to calculate the variability of three dimensions p_i, p_j and p_k, we could generalize the calculation process as Figure 16, where the Q_i, Q_j and Q_k are the corresponding total number of layers of the dimensions p_i, p_j and p_k..

In the swap() function, there is a parameter setting: $preserveFirst2dim$, if the user wants to set the first two dimensions as those two he/she prefers, then by setting this parameter as a vector of the two dimensions' numbers, the user can make sure that in the final result, each block of the table will show those 2 dimensions. For example, set $preserveFirst2dim = c(1, 3)$, then the first and the third dimensions of the original table will be the first two dimensions of the swapped table.

Now let's look at Titanic example to show how swap() works for multi-way tables.

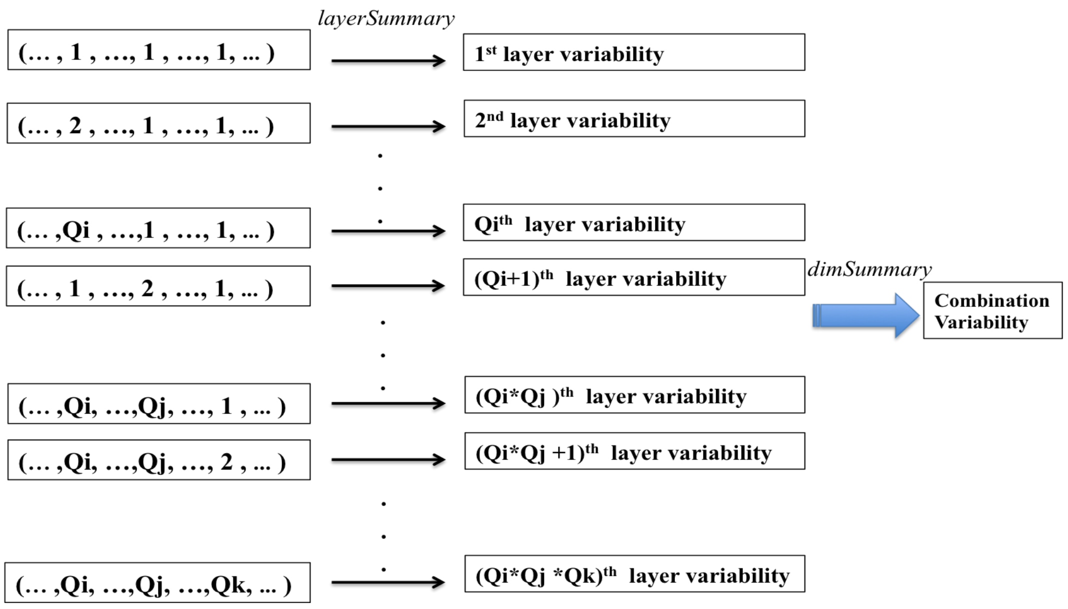

layerSummary
(… , 1 , …, 1 , …, 1, ...)
1st layer variability
(… , 2 , …, 1 , …, 1, ...)
2nd layer variability
(… ,Qi , …,1 , …, 1, ...)
Qith layer variability
(… , 1 , …, 2 , …, 1, ...)
(Qi+1)th layer variability
dimSummary
Combination Variability
(… ,Qi, …,Qj, …, 1 , ...)
(Qi*Qj)th layer variability
(… ,Qi, …,Qj, …, 2 , ...)
(Qi*Qj +1)th layer variability
(… ,Qi, …,Qj, …,Qk, ...)
(Qi*Qj *Qk)th layer variability

Figure 16

```r
swap(Titanic)
```

```
## $swappedTable
## , , Survived = No, Age = Child
##
##         Class
## Sex      1st 2nd 3rd Crew
##   Male     0   0  35    0
##   Female   0   0  17    0
##
## , , Survived = Yes, Age = Child
##
##         Class
## Sex      1st 2nd 3rd Crew
##   Male     5  11  13    0
##   Female   1  13  14    0
```

```
##
## , , Survived = No, Age = Adult
##
##         Class
## Sex      1st 2nd 3rd Crew
##   Male   118 154 387  670
##   Female   4  13  89    3
##
## , , Survived = Yes, Age = Adult
##
##         Class
## Sex      1st 2nd 3rd Crew
##   Male    57  14  75  192
##   Female 140  80  76   20
##
##
## $newDimOrder
## [1] 2 1 4 3
```

After swapping, the swapped table put similar numbers closer, such as the first two blocks and the last two blocks.

2.5 Focus the table and present

After getting location, ordering and swapping, now it's time to think about how to summarize all these results onto the final table.

Recall Rule 1 in the introduction, "reduce number of digits, usually round to two significant or efficient digits for mental arithmetic". By finding and removing the location, we have partly achieved the digit reduction. We keep the digits during the intermediate analysis due to the accuracy considered, but when it comes to show the final report to the reader, numbers in the table shouldn't hold the long digits anymore. Ehrenberg (1977) explains that "Understanding any set of numbers involves relating the different numbers to each other. For example, mentally subtracting the 330.9 from 597.9 and remembering the answer is relatively difficult. Taking ratios mentally (330.9 into 597.9) is virtually impossible. Most of us can do such mental arithmetic only by first rounding the figures to one or two digits in our heads. The general rule is to round to two significant or effective digits, where 'significant' or 'effective' here means digits that vary in that kind of data. (Final 0's do not matter as the eye can readily filter them out.)" In this paper, Ehrenberg also provides a discussion about the objections of rounding to two digits and gives the proof that why two significant digits is better.

However, summarizing the data is not as simple as only rounding to two significant or efficient digits. The magnitude range needs to be considered here, and we also need to avoid showing scientific notations and non-integer numbers in the table. Let's look at several examples to illustrate the problems.

```
table6
```

```
##       [,1]  [,2]   [,3]  [,4]
## [1,] 97623 92243 100906 90397
## [2,] 48296 42317  49983 39097
## [3,] 75238 75162 100116 74235
## [4,] 49699 57212  80196 51092
```

table6 has many digits, by rounding to 2 digits, it becomes:

```
table6_2digits
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 98000 92000 1e+05 90000
## [2,] 48000 42000 5e+04 39000
## [3,] 75000 75000 1e+05 74000
## [4,] 50000 57000 8e+04 51000
```

Now the table numbers has many zeros in the tail and there are some scientific notations in the table.

For those numbers with long digits, after rounding to two significant or efficient digits, the digits in the tail will be rounded to zeros. Although as Ehrenberg (1977) says, “final 0’s do not matter as the eye can readily filter them out”, but too many zeros showed on the table will looks messy, moreover, there may be not enough space to present those interminable zeros, then scientific notations may happens here. Scientific notation is a interruption when it’s shown on the table. For example, a fictitious two-way table based on the sales data example is as follows.

Table 6

	Q1	Q2	Q3	Q4
North	98	92	100	90
South	48	42	50	39
East	75	75	$7e + 05$	74
West	50	57	80	51

The appearance of scientific notations abruptly interrupt us no matter when we scan down the column or scan across the row, or look through the numbers in the table. Because we need to stop for a while, look carefully about those two numbers before and after “e”, i.e. the coefficient and the exponent, to know what exactly the number is. Scientific notations are really neat when presenting some long digit numbers, but when people want to quickly look through a group of numbers and get a general idea about the data, it's really a obstacle. If there are more than one scientific notation in the table, the situation will be worse, just as the above table6_2digits example.

Of course, we could change the print setting in order to not show the scientific notations, for example, in R, the "scipen" parameter in options() function controls the scientific notation expression. The default of "scipen" is zero, which allows less than or equal to four zeros shown in the tail of a number. For example, 20000 can be printed as it is, but 200000 will be printed as scientific notation 2e+05. Adding one to the "scipen" parameter will allows one more zero to be printed without scientific notation.

But in this case, there are some common zeros in the tail, which can be extracted out of the table. For example, in this case, we can extract three zeros in the tail, the table6_2digits then becomes

```
table6_2digits/1000
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   98   92  100   90
## [2,]   48   42   50   39
## [3,]   75   75  100   74
## [4,]   50   57   80   51
```

Now there are no longer scientific notations in the table, and the "1000" can be recorded as the unit of the table. By this way, the real table numbers won't be affected but the layout is much tidier, the readers can focus more on the non-zero digits and mentally relate different numbers more easily, so it's easier for the reader to find patterns.

Now let's look at another example to illustrate another problem.

Recall the **table2** we have seen in Section 2.2

```
table2
```

```
## [1] 0.7999 0.7998 0.7998 0.7997
```

After getting and removing the location 0.799, **table2** becomes

```
table2 - getLocation(table2)
```

```
## [1] 0.0009 0.0008 0.0008 0.0007
```

Now there is only one significant digit in table, so we don't need to round it anymore. It seems fine, but when we try printing it, the result is showed as Table 7.

Table 7

0.000900000000000012
0.000799999999999912
0.000799999999999912
0.000699999999999923

That's the problem of machine floating point storage, which is inevitable. We have discussed this problem in Section 2.2. To deal with this problem, we try to transform the decimal numbers into integers. By extracting a unit as 10^{-4}, the table becomes:

```
(table2 - getLocation(table2))*10^4
```

```
## [1] 9 8 8 7
```

So when dealing with non-integers in the tables, we try to transform them to integers by using a unit in case those non-integers will be shown as ugly long floating points like Table 7. Now let's look at another example.

```
table7
```

```
##             [,1]   [,2]        [,3]   [,4]
## [1,] 1.2345e-05     20 1.2345e-05     20
## [2,] 2.0000e-03     34 2.0000e-03     34
## [3,] 3.3000e+00  20000 3.3000e+00  20000
## [4,] 4.0000e+00 300000 4.0000e+00 300000
```

table7 includes numbers with large magnitude range, for example, 1.2345e-05 and 3000000, so there are some scientific notations in the table and the table looks very messy. By rounding to 2 significant digits, **table7** becomes:

```
table7_2digits
```

```
##         [,1]   [,2]    [,3]   [,4]
## [1,] 1.2e-05     20 1.2e-05     20
## [2,] 2.0e-03     34 2.0e-03     34
## [3,] 3.3e+00  20000 3.3e+00  20000
## [4,] 4.0e+00 300000 4.0e+00 300000
```

Now the situation becomes better but it's still messy with so many scientific notations. Notice in this case, there are both small numbers as 1.2e-05 and large numbers as 300000 shown in the table. If we want to construct the unit based on smaller numbers in the table, say 10^{-6)} as the unit, then the table becomes

```r
table7_2digits*10^6
```

```
##          [,1]    [,2]    [,3]    [,4]
## [1,]       12 2.0e+07      12 2.0e+07
## [2,]     2000 3.4e+07    2000 3.4e+07
## [3,] 3300000 2.0e+10 3300000 2.0e+10
## [4,] 4000000 3.0e+11 4000000 3.0e+11
```

Now the small numbers (non-integers) became integers, so we don't need to worry about the floating point storage problems as we illustrated above. But the large numbers will have too many zeros with the unit 10^(-6), 300000 becomes 300000000000, scientific notation as 3e+11 is inevitable. Even if we could change the "scipen" setting in options() to avoid scientific notation, there are still too many zeros.

```r
op<-options("scipen")
options("scipen"=7)
table7_2digits*10^6
```

```
##          [,1]         [,2]    [,3]         [,4]
## [1,]       12     20000000      12     20000000
## [2,]     2000     34000000    2000     34000000
## [3,] 3300000  20000000000 3300000  20000000000
## [4,] 4000000 300000000000 4000000 300000000000
```

```r
options(op)
```

Imagine an extreme example that there is a number with 20 zeros in the table, do you still want to print it by setting the "scipen" parameter?

If we choose a unit considering the large numbers, for example in this case 10^1 as the unit to avoid scientific notations without setting the "scipen" parameter, and round the table in order to avoid non-integers. The table now becomes

```r
round(table7_2digits/10^1)
```

```
##      [,1]  [,2] [,3]  [,4]
## [1,]    0     2    0     2
## [2,]    0     3    0     3
## [3,]    0  2000    0  2000
## [4,]    0 30000    0 30000
```

Then many numbers (8 of 16) are rounded to zeros, we lose too much information in this way.

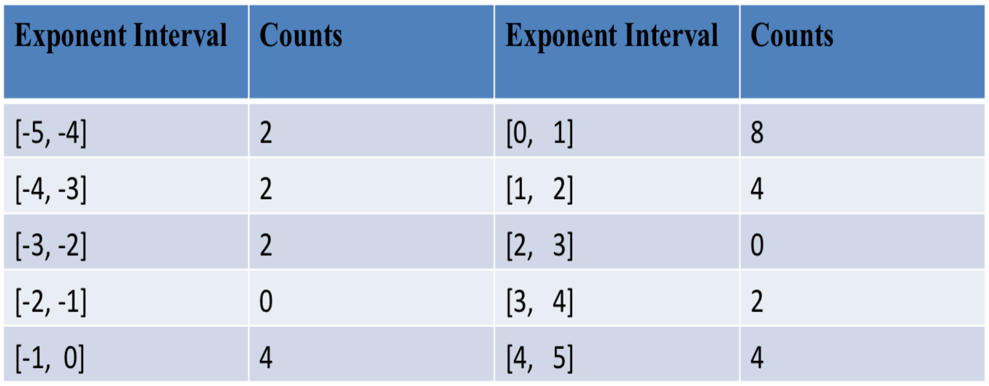
Therefore, when the magnitude range of table numbers is large, it's usually hard to consider both large numbers and small numbers. In this case, we need to sacrifice some information by focusing on the majority of the table. Here "majority"" means the majority of the numbers' magnitude. We construct the unit based on the majority of the numbers' magnitude, so most of the numbers on the level of major magnitude will be shown on the table, then the smaller numbers will be rounded to zeros and the larger number will end up with lots of zeros in the tail or the scientific notations. Then we could suggest a "scipen" setting to make those larger numbers be shown normally, without scientific notations.

Now we use table7_2digits to illustrate the above ideas. Let's first look at the scientific notation coefficients and exponents of table numbers:

```r
SciNotation(table7_2digits, ndigits=2)
```

```
## $coefficient
##      [,1] [,2] [,3] [,4]
## [1,]  1.2  2.0  1.2  2.0

## [2,]  2.0  3.4  2.0  3.4

## [3,]  3.3  2.0  3.3  2.0

## [4,]  4.0  3.0  4.0  3.0

##

## $exponent

##  [1] -5 -3  0  0  1  1  4  5 -5 -3  0  0  1  1  4  5
```

The maximum exponent is 5, and the minimum is -5, so the magnitude range is 10, which is pretty large. In order to measure the majority of the exponents, we need a threshold, denoted as *expsRange*. Then we starts from the minimum exponent, -5, and we count how many exponents are within $[-5, -5 + expsRange]$ in the table. For example, here we use $expsRange = 1$. Then there are 2 exponents in total, which are within [-5, -4], , i.e. -5 and -5. Then we continue with add 1 to both sides of the interval, i.e. [-4, -3], then there are 2 exponents within this range, i.e. -3 and -3. Continue this process until the right handside of the interval reaches the maximum exponent, i.e. 5 in this case, the results are listed as follows.



Exponent Interval	Counts	Exponent Interval	Counts
[-5, -4]	2	[0, 1]	8
[-4, -3]	2	[1, 2]	4
[-3, -2]	2	[2, 3]	0
[-2, -1]	0	[3, 4]	2
[-1, 0]	4	[4, 5]	4

Then the interval [0,1] has the maximum count (8) of table numbers' exponents, which is what we called "majority of the numbers' magnitude". Then we construct the unit based on this interval, in this case, we set the interval as 10^{-1}, so the table now becomes.

```
##          [,1]    [,2]    [,3]    [,4]
## [1,] 1.2e-04     200 1.2e-04     200
## [2,] 2.0e-02     340 2.0e-02     340
## [3,] 3.3e+01  200000 3.3e+01  200000
## [4,] 4.0e+01 3000000 4.0e+01 3000000
```

Then we round the table to avoid decimal digits, the table becomes

```
##      [,1]    [,2] [,3]    [,4]
## [1,]    0     200    0     200
## [2,]    0     340    0     340
## [3,]   33  200000   33  200000
## [4,]   40 3000000   40 3000000
```

We sacrifice some information, such as four numbers are rounded to zeros in this case, but the large numbers are shown with acceptable number of zeros in the tail.

The complete output if applying focusTable() on **table7** is as follows:

```
focusTable(table7)
```

```
## $table
##      [,1]    [,2] [,3]    [,4]
## [1,]    0     200    0     200
## [2,]    0     340    0     340
## [3,]   33  200000   33  200000
## [4,]   40 3000000   40 3000000
##
## $base
## [1] -1
##
## $scipen
## [1] 2
```

Here, \$table is the one we will print in the final tidy table report, which has been rounded to required number of significant digits by setting $nSig$ parameter in focusTable() function; \$base is the number we use to construct the unit of \$table, $unit = 10^{base}$; \$scipen is our suggested "scipen" setting in options() function, to print the table without scientific notations. In some extreme cases, after using the focusTable() function, maybe there are still some pretty large numbers with lots of zeros in the tail, we can't rule out this situation, but we have tried to avoid this by sacrificing some information as showed above.

Now let's look at some other examples of focusTable():

```
table8
```

```
##         [,1]   [,2]    [,3]   [,4]
## [1,] 0.12345   20.3 0.12345   20.3
## [2,] 2.00000   34.5 2.00000   34.5
## [3,] 3.30000  200.0 3.30000  200.0
## [4,] 4.00000 3000.0 4.00000 3000.0
```

```
focusTable(table8)
```

```
## $table
##      [,1]  [,2] [,3]  [,4]
## [1,]    1   200    1   200
## [2,]   20   350   20   350
## [3,]   33  2000   33  2000
## [4,]   40 30000   40 30000
##
## $base
## [1] -1
##
## $scipen
## [1] 0
```

```
table9
```

```
##      [,1]  [,2] [,3]  [,4]
## [1,]    1   203    1   203
## [2,]   20   345   20   345
## [3,]   33  2000   33  2000
## [4,]   40 30000   40 30000
```

```
focusTable(table9)
```

```
## $table
##      [,1]  [,2] [,3]  [,4]
## [1,]    1   200    1   200
## [2,]   20   350   20   350
## [3,]   33  2000   33  2000
## [4,]   40 30000   40 30000
##
## $base
## [1] 0
```

```
##
## $scipen
## [1] 0
```

```r
table10
```

```
## , , k1
##
##        j1    j2  j3   j4
## i1 5e+05 50000 500 5000
## i2 9e+05 90000 900 9000
## i3 2e+05 20000 200 2000
## i4 8e+05 80000 800 8000
##
## , , k2
##
##        j1  j2   j3    j4
## i1 7e+09 800 6000 4e+07
## i2 4e+09 300 8000 9e+07
## i3 2e+09 700 2000 3e+07
## i4 5e+09 100 9000 1e+07
```

```r
focusTable(table10)
```

```
## $table
## , , k1
##
##       j1  j2 j3 j4
## i1 5000 500  5 50
## i2 9000 900  9 90
## i3 2000 200  2 20
## i4 8000 800  8 80
##
## , , k2
##
##        j1 j2 j3    j4
## i1 7e+07  8 60 4e+05
## i2 4e+07  3 80 9e+05
## i3 2e+07  7 20 3e+05
## i4 5e+07  1 90 1e+05
##
##
## $base
## [1] 2
##
## $scipen
## [1] 3
```

The function focusTable() is designed to provide the elements needed when present the final table. In summary, “focus” has several meanings:

- Focus on the significant digits.
- Focus on the non-zero digits.
- Focus on the majority of the table numbers
- Focus on the integers

There are two significant digits in the focusTable() function, *nSig* and *expsRange*. *nSig* controls how many significant digits to be shown in the table, the default is 2; *expsRange* is the threshold we use to determine the majority of magnitude, the default is $nSig - 1$. The user can specify these parameters based on their preference.

3 Tidy up the table and print the tidy table

The functions we have introduced in the Section 2 are like the parts of a machine. With these parts ready, now we can assemble the machine. Recall the Figure 1 of the general structure of tidy table analysis, we just follow the order shown on Figure 1 to construct the tidytable() function. Let's first look at the example we showed in Table 1 in the Section 2.1 to see how the tidytable() works.

```r
tidytable(Table1)
```

```
## $table
##    North East West South
## Q3   100  100   80    50
## Q1    98   75   50    48
## Q2    92   75   57    42
## Q4    90   74   51    39
##
## $units
## [1] 1
##
## $location
## [1] 4102000
##
```

```
## $nSig
## [1] 2
##
## $scipen
## [1] 0
##
## $newDimOrder
## [1] 2 1
##
## $originalTable
##            Q1      Q2      Q3      Q4
## North 4102098 4102092 4102101 4102090
## South 4102048 4102042 4102050 4102039
## East  4102075 4102075 4102100 4102074
## West  4102050 4102057 4102080 4102051
```

The first element of the output $table is the tidy table we finally get; the second element $unit is the unit of the tidy table; the third element of the output is the location of the original table; $nSig records the number of significant digits we have set in the tidytable(); $newDimOrder records the dimensions' order after swapping, we have introduced this in Section 2.4. Last but not least, the original table is also shown in the output as $originalTable, here the decimal digits of Table 1 hasn't been shown in the $originalTable, that's because the default number of digits is 7 in R, so the decimal digits have been rounded.

In the function tidytable(), you can choose to only do a part of the steps shown in Figure 1 by setting the related parameters. You can also choose to only order some of the dimensions of the original table by specifying the parameter $fixedOrderDims$. For example, the example Table 1, we want to order the first dimension, areas, but we don't want to order the second dimension, quarters, to keep a chronological order. By setting the $fixedOrderDims = c(2)$, which means we want to fix the order of the second dimension, then the tidy table is

```
tidytable(Table1, fixedOrderDims=c(2))$table
```

```
##    North East West South
## Q1    98   75   50    48
## Q2    92   75   57    42
## Q3   100  100   80    50
## Q4    90   74   51    39
```

This time the second dimension of the original table hasn't been changed, which is still "Q1, Q2, Q3, Q4".

Let's then look at the three-way table HairEyeColor we introduced before.

```
tidytable(HairEyeColor)
```

```
## $table
## , , Eye = Brown
##
##          Hair
## Sex       Brown Blond Black Red
##   Female     66     4    36  16
##   Male       53     3    32  10
##
## , , Eye = Blue
##
##          Hair
## Sex      Brown Blond Black Red
##   Female    34    64     9   7
##   Male      50    30    11  10
##
## , , Eye = Hazel
##
##         Hair
## Sex      Brown Blond Black Red
##   Female    29     5     5   7
##   Male      25     5    10   7
##
## , , Eye = Green
##
##         Hair
## Sex      Brown Blond Black Red
##   Female    14     8     2   7
##   Male      15     8     3   7
##
##
## $units
## [1] 1
##
## $location
## [1] 0
##
## $nSig
## [1] 2
```

```
##
## $scipen
## [1] 0
##
## $newDimOrder
## [1] 3 1 2
##
## $originalTable
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

Compared with the original table, the tidy table has much clearer pattern, especially from the third and fourth blocks of the tidy table, we can see the numbers on each column are pretty similar. You can continue to find other patterns if you are interested.

Next we look at the four-way table Titanic.

```r
tidytable(Titanic)
```

```
## $table
## , , Survived = No, Age = Adult
##
##         Class
## Sex      Crew 3rd 1st 2nd
##   Male    670 390 120 150
##   Female    3  89   4  13
##
## , , Survived = Yes, Age = Adult
##
##         Class
## Sex      Crew 3rd 1st 2nd
##   Male    190  75  57  14
##   Female   20  76 140  80
##
## , , Survived = No, Age = Child
##
##         Class
## Sex      Crew 3rd 1st 2nd
##   Male      0  35   0   0
##   Female    0  17   0   0
##
## , , Survived = Yes, Age = Child
```

```
##
##          Class
## Sex       Crew 3rd 1st 2nd
##   Male       0  13   5  11
##   Female     0  14   1  13
##
##
## $units
## [1] 1
##
## $location
## [1] 0
##
## $nSig
## [1] 2
##
## $scipen
## [1] 0
##
## $newDimOrder
## [1] 2 1 4 3
##
## $originalTable
## , , Age = Child, Survived = No
##
##        Sex
## Class  Male Female
##   1st     0      0
##   2nd     0      0
##   3rd    35     17
##   Crew    0      0
##
## , , Age = Adult, Survived = No
##
##       Sex
## Class  Male Female
##   1st   118      4
##   2nd   154     13
##   3rd   387     89
##   Crew  670      3
##
## , , Age = Child, Survived = Yes
##
##       Sex
## Class  Male Female
##   1st     5      1
##   2nd    11     13
##   3rd    13     14
##   Crew    0      0
##
## , , Age = Adult, Survived = Yes
##
##       Sex
## Class  Male Female
##    1st     57     140
##    2nd     14      80
##    3rd     75      76
##    Crew   192      20
```

tidytable() function mainly give us a list of results. We need to combine them together to print the tidy table. Also notice here, in terms of multi-way table (number of dimensions >= 3), we find the location, order, swap and find the unit all based on the whole table, but not on each block. For example, the tidy table of the HairEyeColor example is as shown above. The order of the first dimension Sex is Female, Male; but in the fourth block (, , Eye = Green) it's obvious that the average of the fist row (Female) is smaller than the second row (Male). So we might be interested in finding the location, ordering, swapping and finding the unit within each block of the tidy table we've already obtained.

showtidytable() gives the user the choices of *locationType*, *orderType*, *swapType* and *unitType*, by setting these four parameters as "common" or "withinEachTwoWay", we can decide if we want to find the location, order, swap, or find the unit within each block of the tidy table we have obtained. The user can also choose the *display* as "console" or "latex" to print the tidy table in R console or latex. The "latex" option will give the latex code for each separate block, and these latex codes of blocks will be transformed to a series of two-way tables in the document (e.g. PDF file). Here is the example of Titanic when applying the showtidytable().

```
showtidytable(Titanic,display="console",tableCaption="Titanic tidy table",
              locationType="withinEachTwoWay",orderType="withinEachTwoWay",
              swapType="withinEachTwoWay", unitType="withinEachTwoWay")
```

```
## Titanic tidy table
##  Location:  0
```

```
##
## , , Survived = No,Age = Adult
## Unit:  10
##        Sex
## Class  Male Female
##   Crew   67      0
##   3rd    39      9
##   2nd    15      1
##   1st    12      0
##
## , , Survived = Yes,Age = Adult
## Unit:  1
##        Sex
## Class  Male Female
##   Crew  190     20
##   1st    57    140
##   3rd    75     76
##   2nd    14     80
##
## , , Survived = No,Age = Child
## Unit:  1
##        Sex
## Class  Male Female
##   3rd    35     17
##   Crew    0      0
##   1st     0      0
##   2nd     0      0
```

```
##
## , , Survived = Yes,Age = Child
## Unit:  1
##         Class
## Sex      3rd 2nd 1st Crew
##   Male    13  11   5    0
##   Female  14  13   1    0
```

Here although we choose $locationType = "withinEachTwoWay"$, since the locations in each block are the same, which is 0, so the location information are only printed once on top. Similarly, if the units of each block are all the same, then there will only be an overall unit information to be printed on top, even if $unitType = "withinEachTwoWay"$.

By choosing any of the *orderType* and *swapType* as “withinEachTwoWay”, the tidy table finally printed is not a strictly defined multi-way table since the first dimension may not be the same among blocks. But it sill has other uniform dimensions except for the first two dimensions, because we swap/order within blocks of the tidy table from tidytable(), the general structure is the same as the tidy table, so it can be regarded as a generalized tidy table.

4 Limitations and Future work

4.1 Application of the remaining rules

Up till now, our tidytable package has almost achieved the objective of table analysis based on the Rule 1, Rule 2 and Rule 3 introduced in the Introduction part. However, other rules still need to be further applied, for example:

- Rule 4: Use averages (or medians) to help focus the eye over the array.

- Rule 5: Note dramatically exceptional values and exclude them from pattern summary calculations.
- Rule 6: Avoid introducing new variables or scales (e.g. totals) whenever possible.
- Rule 7: Figures to be compared should be close together.

  ...

As for the Rule 4, Ehrenberg (1977) explains that it "concerns the use of row and column averages to provide a visual focus and a possible summary of the data." In the paper *Extracting Sunbeams From Cucumbers* (2011), Feinberg & Wainer mentioned that "In most tables it is usually useful, and often critical, to surround the table with some kind of summary statistics. Sometimes these are sums, sometimes means, but most often medians have proven to be the summary of choice. The characteristic that makes medians especially useful is their insensitivity to unusual data points. Thus the median can represent the mass of points and when we look at deviations from medians unusual points will have large residuals." So we need to further think about and design an appropriate way to generate and display these table summaries.

As for Rule 7, spacing and layout can improve the visualization efficiency. Placing is particularly effective in making the eye read down columns. But there are also need to be deliberate gaps to guide the eye across the table. However, compared with other rules, this rule is harder to be achieved automatically. What's more, as Ehrenberg (1977) mentioned, "many typists, printers and computers are programmed differently. Double spacing in tables is common, as are columns spaced unevenly according to the width of the headings, and occasional irregular gaps between single-spaced rows because some row captions ran to two lines. One needs not only good typists or printers, but also thoughtful control of these facilities." In order to apply this rule, we need to further learn and know more about different softwares, computers' and typists' settings, create a well-designed way to achieve table visualization benefits.

In summary, we just started the analysis of multi-way tables, there are many other rules to be considered. We will continue to explore those rules and try to make the tidytable package more complete gradually.

4.2 R contingency table layout

During the analysis of our tidytable package, we mainly considered the R console layout of multi-way table, there is a ftable() function in stats package to produce multi-way 'flat' contigency tables. For example, the Titanic example can be shown as a contigency table as follows:

```r
ftable(Titanic, row.vars = 1:2)
```

```
##              Age      Child     Adult
##              Survived    No Yes    No Yes
## Class Sex
## 1st   Male                0   5   118  57
##       Female              0   1     4 140
## 2nd   Male                0  11   154  14
##       Female              0  13    13  80
## 3rd   Male               35  13   387  75
##       Female             17  14    89  76
## Crew  Male                0   0   670 192
##       Female              0   0     3  20
```

By setting the parameter *row.vars* and *col.vars*, we can choose vraibles to be shown on the rows and on the columns. This seems to give us more flexibility of the table layout. If we want to apply those table visualization rules to this type of contigency tables, how to ordering within the dimensions and swapping among dimensions need to be well-considered to adapt to the new layout way of tables.

5 Acknowledgments

I would like to thank Professor Wayne Oldford. As my supervisor, he gave me patient and inspiring instructions about how to conduct this research project. I learned a lot during the process of this research. For example, I learned to combine the ideas thinking and examples testing together to create new ideas and improvements of the research. Sometimes only using our brain cannot imagine all the situations, research projects especially developing a R package need us to try some examples and see what will happen, these may also give us some inspiration of what to do next. Another important thing I learned is that every time we are given a task, I should first try all the means to solve that by ourselves before appealing to others. I need to be able to solve a problem independently. Again, thanks very much for my supervisor's careful and considerate guidance, also the understanding about my occasional personal situations. I really found this research project and data visualization area very interesting and creative. In the future, I hope I have the opportunities to do more research about it.

6 References

A. S. C. Ehrenberg (1977). Rudiments of Numeracy. *J. R. Statist. Soc. A*, 140, part 3, 277-297.

A. S. C. Ehrenberg (1975). Data Reduction: Analysis and Interpreting Statistical Data, John Wiley & Sons Ltd.

David Goldberg (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 5-48.

Andrew Gelman (2011). Why Tables Are Really Much Better Than Graphs (with discussion). *Journal of Computational and Graphical Statistics*, 3-7.

Michael Friendly and Ernest Kwan (2011). Comment on Why Tables Are Really Much Better Than Graphs. *Journal of Computational and Graphical Statistics*, 18-27.

Richard A. Feinberg and Howard Wainer (2011). Extracting Sunbeams From Cucumbers. *Journal of Computational and Graphical Statistics*, 793-810.

Marco Demaio (2010). Answer to question: how to check if the number is integer?. http://stackoverflow.com/questions/3476782/how-to-check-if-the-number-is-integer