vectools 0.1.1

# Supplementary Vector-Related Tools

**Abby Spurdle**

January 8, 2020

*Supports formatted nested/partitioned matrices, formatted object arrays and similar formatted data.frame(s), via coercion. These objects can be printed with plain text mark up, including their partitions and submatrices. Also, includes an SQL-like select function, grouped head functions and combined head and tail functions.*

*****note that this package is subject to change*****

## Introduction

Formatted tables are a central feature of software like Microsoft Excel.

Formatted matrices and tables are also used in reporting contexts, including Latex/amsmath and HTML documents.

However, R has limited support for such objects.

The primary goal of this package is to support formatted object arrays and formatted nested/partitioned matrices, at an interactive level. Also, it contains tools for subsetting and tabular data analysis. Other tools are likely be added in the near future.

## Preliminary Code

I will load (and attach) the vectools package:

```
> library (vectools)
```

Also, I will create three matrices:

```
> s4x4 = matrix (1:16, 4, 4)
> s10 = matrix (1:100, 10, 10)
> s20 = matrix (1:400, 20, 20)
```

# Part One
# Classes and Objects

## Classes

In this package, the root class is:

- **VectorLike** (Vector-Like Object)

This has two subclasses:

- **ObjectArray** (Object Array)
  These objects are similar to lists, including list matrices and list arrays.
- **MatrixLike** (Matrix-Like Object)

The MatrixLike class, has two subclasses:

- **NestMatrix** (Nested Matrix)
  These objects are similar to two-dimensional object arrays, and represent matrices where their elements are also matrices. Currently, there's no restrictions on the dimensions of their submatrices. Also, nested matrices may be recursively nested. i.e. A matrix within a matrix within a matrix within a matrix...
- **SectMatrix** (Sectioned Matrix)
  These objects are derived from a single matrix, with rectangular sections defining arbitrary submatrices, which may or may not, form a partition of the matrix.

Furthermore, SectMatrix class, has a single subclass:

- **PartMatrix** (Partitioned Matrix)
  These objects are also derived from a single matrix, and can be interpreted as a matrix with horizontal/vertical seperators between rows/columns, which implicitly define a (non-recursively) nested matrix, where the submatrices have matching dimensions.

There are limited subsetting operators, and dim, print, format, head and tail methods.

## Object Arrays

In R, the standard way to create object arrays is via list matrices and list arrays. That works, however, the resulting objects are difficult to format.

Here, **ObjectArray** objects, can be used to simplify the process, and support flexible formatting.

Constructors for near-trivial (S3) classes:

```
> alphabet.1 = function ()
      structure (LETTERS, class="alphabet.1")
> alphabet.2 = function ()
      structure (sample (LETTERS), class="alphabet.2")
```

A near-trivial 2x2 object array:

```
> x = ObjectArray (c (2, 2) )
> x [[1, 1]] = alphabet.1 ()
> x [[2, 1]] = alphabet.1 ()
> x [[1, 2]] = alphabet.2 ()
> x [[2, 2]] = alphabet.2 ()
```

Printed with default formatting:

```
> x
```

```
     [,1]   [,2]
[1,] <v 26> <v 26>
[2,] <v 26> <v 26>
```

To customize the formatting, we can write (S3) objtag methods for our classes:

```
> objtag.alphabet.1 = function (x)
      paste ("<A1 ", x [1], ":", x [26], ">", sep="")
> objtag.alphabet.2 = function (x)
      paste ("<A2 ", x [1], ":", x [26], ">", sep="")
```

And we get:

```
> x
```

```
     [,1]     [,2]
[1,] <A1 A:Z> <A2 Y:H>
[2,] <A1 A:Z> <A2 L:K>
```

Note that objtag methods need to return a single string.

# Nested Matrices (Simple Case)

Re-iterating, **NestMatrix** objects are similar to two-dimensional object arrays.

In general, a nested matrix is mathematically equivalent to a partitioned matrix, so we can construct them in a similar way to a partitioned matrix, discussed later.

```
> x = as.NestMatrix (s10, 5, c (2, 4, 6, 8) )
```

The top-level object:

```
> x
```

```
     [,1]    [,2]    [,3]    [,4]    [,5]
[1,] <m 5x2> <m 5x2> <m 5x2> <m 5x2> <m 5x2>
[2,] <m 5x2> <m 5x2> <m 5x2> <m 5x2> <m 5x2>
```

Expanding the first element:

```
> x [[1, 1]]
```

```
     [,1] [,2]
[1,]    1   11
[2,]    2   12
[3,]    3   13
[4,]    4   14
[5,]    5   15
```

## Nested Matrices (General Case)

For more general cases, we can use the main NestMatrix constructor or the as.NestMatrix.2 function:

```
> xsub = NestMatrix (4, 4)
> for (i in 1:4)
  {   for (j in 1:4)
          xsub [[i, j]] = s4x4
  }

> x = NestMatrix (4, 4)
> for (i in 1:4)
  {   for (j in 1:4)
          x [[i, j]] = xsub
  }
```

The top-level object (x):

```
> x

     [,1]     [,2]     [,3]     [,4]
[1,] <NM 4x4> <NM 4x4> <NM 4x4> <NM 4x4>
[2,] <NM 4x4> <NM 4x4> <NM 4x4> <NM 4x4>
[3,] <NM 4x4> <NM 4x4> <NM 4x4> <NM 4x4>
[4,] <NM 4x4> <NM 4x4> <NM 4x4> <NM 4x4>
```

Expanding the first element (one of the xsub objects):

```
> x [[1, 1]]

     [,1]    [,2]    [,3]    [,4]
[1,] <m 4x4> <m 4x4> <m 4x4> <m 4x4>
[2,] <m 4x4> <m 4x4> <m 4x4> <m 4x4>
[3,] <m 4x4> <m 4x4> <m 4x4> <m 4x4>
[4,] <m 4x4> <m 4x4> <m 4x4> <m 4x4>
```

Expanding the first element within the first element (one of the s4x4 objects):

```
> x [[1, 1]][[1, 1]]

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

## Sectioned Matrices

Here, **SectMatrix** objects contain a single matrix, with sections.

They can be created using either the SectMatrix or as.SectMatrix functions, and then using the setmap function to define what I refer to as section maps.

```
> x = as.SectMatrix (s10, 2)
> setmap (x, 1) = c (2, 2, 4, 4)
> setmap (x, 2) = c (7, 7, 9, 9)
> x
```

```
      [,1]   [,2] [,3] [,4]   [,5] [,6]   [,7] [,8] [,9]   [,10]
 [1,]    1     11   21   31     41   51     61   71   81     91
           + ---  ---  ---  +                                    
 [2,]    2  |  12   22   32  |  42   52     62   72   82     92
 [3,]    3  |  13   23   33  |  43   53     63   73   83     93
 [4,]    4  |  14   24   34  |  44   54     64   74   84     94
           + ---  ---  ---  +                                    
 [5,]    5     15   25   35     45   55     65   75   85     95
 [6,]    6     16   26   36     46   56     66   76   86     96
                                        + ---  ---  ---  +      
 [7,]    7     17   27   37     47   57  |  67   77   87  |  97
 [8,]    8     18   28   38     48   58  |  68   78   88  |  98
 [9,]    9     19   29   39     49   59  |  69   79   89  |  99
                                        + ---  ---  ---  +      
 [10,]  10     20   30   40     50   60     70   80   90    100
```

Single bracket subsetting gives submatrices and double bracket subsetting gives elements from the combined matrix:

```
> x [1]
```

```
     [,1] [,2] [,3]
[1,]   12   22   32
[2,]   13   23   33
[3,]   14   24   34
```

```
> x [1][1, 2]
```

```
[1] 22
```

```
> x [[2, 3]]
```

```
[1] 22
```

Note that it's possible for submatrices to overlap:

```
> setmap (x, 1) = c (2, 2, 7, 7)
> setmap (x, 2) = c (4, 4, 9, 9)
> x
```

```
      [,1]   [,2] [,3]     [,4] [,5] [,6] [,7]     [,8] [,9]   [,10]
 [1,]    1     11   21       31   41   51   61       71   81     91
           + ---  ---  --- ---  ---  ---  ---  +                    
 [2,]    2  |  12   22       32   42   52   62  |    72   82     92
 [3,]    3  |  13   23       33   43   53   63  |    73   83     93
           |           +   ---  ---  ---  ---  --- ---  ---  +      
 [4,]    4  |  14   24  |    34   44   54   64  |    74   84  |  94
 [5,]    5  |  15   25  |    35   45   55   65  |    75   85  |  95
 [6,]    6  |  16   26  |    36   46   56   66  |    76   86  |  96
 [7,]    7  |  17   27  |    37   47   57   67  |    77   87  |  97
           + ---  ---  |   ---  ---  ---  ---  +             |      
 [8,]    8     18   28  |    38   48   58   68       78   88  |  98
 [9,]    9     19   29  |    39   49   59   69       79   89  |  99
                       +   ---  ---  ---  ---  --- ---  ---  +      
 [10,]  10     20   30       40   50   60   70       80   90    100
```

Als note that you need to set all the section maps, before printing or formatting.

## Partitioned Matrices

Re-iterating, **PartMatrix** objects extend sectioned matrices.

They're created using either the PartMatrix or as.PartMatrix functions, and by specifying the inter-row and inter-column indicies of seperators.

Here's a partitioned matrix, with one row separator, and four column separators:

```
> x = as.PartMatrix (s10, 5, c (2, 4, 6, 8) )
> x
```

```
      [,1] [,2]   [,3] [,4]   [,5] [,6]   [,7] [,8]   [,9] [,10]
[1,]    1   11  |  21   31  |  41   51  |  61   71  |  81   91
[2,]    2   12  |  22   32  |  42   52  |  62   72  |  82   92
[3,]    3   13  |  23   33  |  43   53  |  63   73  |  83   93
[4,]    4   14  |  24   34  |  44   54  |  64   74  |  84   94
[5,]    5   15  |  25   35  |  45   55  |  65   75  |  85   95
      ---  ---  + ---  ---  + ---  ---  + ---  ---  + ---  ---
[6,]    6   16  |  26   36  |  46   56  |  66   76  |  86   96
[7,]    7   17  |  27   37  |  47   57  |  67   77  |  87   97
[8,]    8   18  |  28   38  |  48   58  |  68   78  |  88   98
[9,]    9   19  |  29   39  |  49   59  |  69   79  |  89   99
[10,]  10   20  |  30   40  |  50   60  |  70   80  |  90  100
```

In principle, subsetting is the same as sectioned matrices:

```
> x [1, 2]
```

```
     [,1] [,2]
[1,]   21   31
[2,]   22   32
[3,]   23   33
[4,]   24   34
[5,]   25   35
```

```
> x [1,2][2, 1]
```

```
[1] 22
```

```
> x [[2, 3]]
```

```
[1] 22
```

However, partitioned matrices always use two dimensional section indicies, whereas sectioned matrices can use one, two or higher dimensional indices.
i.e. The example in the previous section, used one dimensional indices.

## Nested Matrices vs Partitioned Matrices

If we limit nested matrices to the simple case, then nested matrices and (regular) partitioned matrices are mathematically equivalent. However, this package implements them differently, and uses different formatting.

Here's a comparision:

```
> nm = as.NestMatrix (s10, 5, c (2, 4, 6, 8) )
> pm = as.PartMatrix (s10, 5, c (2, 4, 6, 8) )
```

```
> nm
```

```
     [,1]    [,2]    [,3]    [,4]    [,5]
[1,] <m 5x2> <m 5x2> <m 5x2> <m 5x2> <m 5x2>
[2,] <m 5x2> <m 5x2> <m 5x2> <m 5x2> <m 5x2>
```

```
> pm
```

```
      [,1] [,2]   [,3] [,4]   [,5] [,6]   [,7] [,8]   [,9] [,10]
[1,]    1   11  |  21   31  |  41   51  |  61   71  |  81   91
[2,]    2   12  |  22   32  |  42   52  |  62   72  |  82   92
[3,]    3   13  |  23   33  |  43   53  |  63   73  |  83   93
[4,]    4   14  |  24   34  |  44   54  |  64   74  |  84   94
[5,]    5   15  |  25   35  |  45   55  |  65   75  |  85   95
      ---  ---  + ---  ---  + ---  ---  + ---  ---  + ---  ---
[6,]    6   16  |  26   36  |  46   56  |  66   76  |  86   96
[7,]    7   17  |  27   37  |  47   57  |  67   77  |  87   97
[8,]    8   18  |  28   38  |  48   58  |  68   78  |  88   98
[9,]    9   19  |  29   39  |  49   59  |  69   79  |  89   99
[10,]  10   20  |  30   40  |  50   60  |  70   80  |  90  100
```

Also, note the differences in dimensions and subsetting operators:

```
> dim (nm)
```

```
[1] 2 5
```

```
> dim (pm)
```

```
[1] 10 10
```

```
> nm [[1, 1]]
```

```
     [,1] [,2]
[1,]    1   11
[2,]    2   12
[3,]    3   13
[4,]    4   14
[5,]    5   15
```

```
> pm [1, 1]
```

```
     [,1] [,2]
[1,]    1   11
[2,]    2   12
[3,]    3   13
[4,]    4   14
[5,]    5   15
```

Expanding on a previous point, this package is subject to change, and in particular, the handling of dimensions and subsetting operations.

# Part Two
# SQL-Like Functions

## SQL-Like Select Function

The **select** function is a R (only) function with standard R syntax, but nonstandard evaluation. Currently, it supports a subset of SQL select functionality, and is designed for convenience only and not for high performance or large datasets. This vignette is just a demonstration, refer to the help page for more details.

I will use the mtcars data.

Trivial use:

```
> #all variables
> head (
      select (., from (mtcars) )
  )

   mpg cyl disp  hp drat    wt  qsec vs am gear carb
1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

> head (
      select (am, cyl, mpg, from (mtcars) )
  )

  am cyl  mpg
1  1   6 21.0
2  1   6 21.0
3  1   4 22.8
4  0   6 21.4
5  0   8 18.7
6  0   6 18.1
```

And for a less trivial example, using the **select** function to produce the count and mean of mpg, grouped by am and cy1:

```
> select (am, cyl,
      from (mtcars),
      group.by (am, cyl),
          count <- length (mpg),
          mean.mpg <- mean (mpg) )

  am cyl count mean.mpg
1  0   4     3 22.90000
2  0   6     4 19.12500
3  0   8    12 15.05000
4  1   4     8 28.07500
5  1   6     3 20.56667
6  1   8     2 15.40000
```

Using an SQL-like function for aggregation, has the advantage that it's relative simple and intuitive.

Optionally, we can partition and sort the data:

```
> select (am, cyl,
      from (mtcars),
      group.by (am, cyl), partition.by (am), sort.by (-am, -mean.mpg),
          count <- length (mpg),
          mean.mpg <- mean (mpg) )
```

```
  am          cyl      count    mean.mpg
  -------- + -------- -------- --------
1 1        | 4         8       28.07500
2          | 6         3       20.56667
3          | 8         2       15.40000
  -------- | -------- -------- --------
4 0        | 4         3       22.90000
5          | 6         4       19.12500
6          | 8        12       15.05000
```

The where construct is currently simple, and can be applied to variables present both before and after grouping:

```
> head (
      select (am, cyl, mpg, from (mtcars), where (mpg >= 20) )
  )
```

```
  am cyl  mpg
1  1   6 21.0
2  1   6 21.0
3  1   4 22.8
4  0   6 21.4
5  0   4 24.4
6  0   4 22.8
```

# Part Three
# Head and Tail Generalizations

## Combined Head and Tail Methods (Matrices)

The **headt** function can be used to print the head and tail simultaneously.

I will use a SectMatrix object, however, this works on several classes:

```
> x = as.SectMatrix (s20, 2)
> setmap (x, 1) = c (2, 2, 19, 19)
> setmap (x, 2) = c (3, 3, 18, 18)
```

Head and tail, with n = 6.

```
> headt (x, 6)
```

```
      [,1]     [,2]     [,3] [,4]   [,17] [,18]     [,19]     [,20]
[1,]     1       21       41   61  # 321   341       361       381
           +   ---  --- ---  ---  # ---   ---   --- ---   +
[2,]     2   |   22       42   62  # 322   342       362   |   382
           |       +   ---  ---  # ---   ---   +         |
[3,]     3   |   23   |   43   63  # 323   343   |   363   |   383
[4,]     4   |   24   |   44   64  # 324   344   |   364   |   384
       ###  ### ###  ### ###  ###  # ###   ###   ### ###   ### ###
[17,]   17   |   37   |   57   77  # 337   357   |   377   |   397
[18,]   18   |   38   |   58   78  # 338   358   |   378   |   398
           |       +   ---  ---  # ---   ---   +         |
[19,]   19   |   39       59   79  # 339   359       379   |   399
           +   ---  --- ---  ---  # ---   ---   --- ---   +
[20,]   20       40       60   80  # 340   360       380       400
```

We can specify rows and columns separately:

```
> headt (x, c (3, 6) )
```

```
      [,1]     [,2]     [,3] [,4]   [,17] [,18]     [,19]     [,20]
[1,]     1       21       41   61  # 321   341       361       381
           +   ---  --- ---  ---  # ---   ---   --- ---   +
[2,]     2   |   22       42   62  # 322   342       362   |   382
       ###  ### ###  ### ###  ###  # ###   ###   ### ###   ### ###
[19,]   19   |   39       59   79  # 339   359       379   |   399
           +   ---  --- ---  ---  # ---   ---   --- ---   +
[20,]   20       40       60   80  # 340   360       380       400
```

And head and tail separately:

```
> headt (x, 6, 3)
```

```
      [,1]     [,2]     [,3] [,4]   [,19]     [,20]
[1,]    1       21       41   61  # 361       381
           +   ---  --- ---  ---  # ---   +
[2,]    2  |    22       42   62  # 362   |   382
           |        +   ---  ---  #       |
[3,]    3  |    23  |    43   63  # 363   |   383
[4,]    4  |    24  |    44   64  # 364   |   384
      ###  ### ###  ### ###  ###  # ###   ### ###
[19,]  19  |    39       59   79  # 379   |   399
           +   ---  --- ---  ---  # ---   +
[20,]  20       40       60   80  # 380       400
```

Note that currently, the size arguments, include the seperators.

## Grouped Head (Tables)

The **headg** function can be used to produce head(s) for subsets:

```
> headg (iris, "Species")
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    ---------- ---------- ---------- ---------- ----------
1   5.1          3.5         1.4          0.2         setosa
2   4.9          3.0         1.4          0.2         setosa
3   4.7          3.2         1.3          0.2         setosa
    ---------- ---------- ---------- ---------- ----------
51  7.0          3.2         4.7          1.4         versicolor
52  6.4          3.2         4.5          1.5         versicolor
53  6.9          3.1         4.9          1.5         versicolor
    ---------- ---------- ---------- ---------- ----------
101 6.3          3.3         6.0          2.5         virginica
102 5.8          2.7         5.1          1.9         virginica
103 7.1          3.0         5.9          2.1         virginica
```

Currently, it only supports data.frame(s).

## Combined Head and Tail Methods (Tables)

The **headt** function works on standard matrices and data.frame(s), too.

Here's an example using the trees data:

```
> headt (trees)
```

```
   Girth Height Volume
1    8.3  70     10.3
2    8.6  65     10.3
3    8.8  63     10.2
   ####  ####   ####
29 18.0  80     51.5
30 18.0  80     51.0
31 20.6  87     77.0
```