Working with lists

Tad Dallas

Contents

What are lists?

Lists in R are collections of objects that can be of any mix of types (or all the same type). They can be useful when dealing with multiple data.frames that each correspond to a different unit of study (note that we solved this before by considering a single data.frame with a column corresponding to country). Lists tend to be useful in my research when I'm simulating ecological dynamics, where each list item can be the results of a single simulation, or when I'm writing functions to return data that is not really structured as a data.frame. Before we go too far into use cases, lets refresh on how we form and index list objects.

```
lst <- list(runif(100), data.frame(x=runif(100), y=runif(100)), letters[1:10], 'a')

# index single list elements
lst[[1]]
```

```
##   [1] 0.042178699 0.345887301 0.139452587 0.206275186 0.452084869 0.717351019
##   [7] 0.126463111 0.640952523 0.801419775 0.703852942 0.692316750 0.661239739
##  [13] 0.602484902 0.812405247 0.552596993 0.435955625 0.534386890 0.944155622
##  [19] 0.923601537 0.015106945 0.745050451 0.073284685 0.113772524 0.449166791
##  [25] 0.995989168 0.743967811 0.220956105 0.663049563 0.898064294 0.068160296
##  [31] 0.173852696 0.506830390 0.614981351 0.033218715 0.970304654 0.991651755
##  [37] 0.115741640 0.570753892 0.315901366 0.367685139 0.573164525 0.357366330
##  [43] 0.080241389 0.858652729 0.210387068 0.499990614 0.745983964 0.127897062
##  [49] 0.806415981 0.581949470 0.912762206 0.656683683 0.221398124 0.001243219
##  [55] 0.800384199 0.872079416 0.026579988 0.528983004 0.477748008 0.930355262
##  [61] 0.428420319 0.953042187 0.905021955 0.585571432 0.338094982 0.582047309
##  [67] 0.034133181 0.349469113 0.802252213 0.363345457 0.013893382 0.417913733
##  [73] 0.996726355 0.191934285 0.059203505 0.467623840 0.828425161 0.591229726
##  [79] 0.772098457 0.340390051 0.617618100 0.077781499 0.351816075 0.481981283
##  [85] 0.323888728 0.605172041 0.894832945 0.591233902 0.443998539 0.365908439
##  [91] 0.389481444 0.618297326 0.734688197 0.896326262 0.097087294 0.725656056
##  [97] 0.600022382 0.443346858 0.393072930 0.642777560
```

```
lst[[2]]
```

```
##               x          y
## 1   0.5929958255 0.17029630
## 2   0.2273250618 0.53290314
## 3   0.9836112957 0.86924316
## 4   0.6195508826 0.12927133
## 5   0.5355325681 0.67682812
```

```
## 6   0.4026766657 0.43604307
## 7   0.6762322055 0.40647950
## 8   0.3963014511 0.65137982
## 9   0.6644938316 0.40649973
## 10  0.7806688529 0.11119371
## 11  0.7036440966 0.08355948
## 12  0.2227644119 0.09273169
## 13  0.9076592533 0.56791180
## 14  0.0635249263 0.92151644
## 15  0.3846838397 0.88746459
## 16  0.0828204176 0.34702496
## 17  0.0089157571 0.92305583
## 18  0.2628677031 0.55358793
## 19  0.2834337901 0.55690307
## 20  0.4602273584 0.22505792
## 21  0.7177088670 0.89639362
## 22  0.6354470754 0.54169483
## 23  0.8354455414 0.28061722
## 24  0.9358799148 0.10206755
## 25  0.5355347167 0.71250764
## 26  0.8747624087 0.06888842
## 27  0.2216022352 0.93237516
## 28  0.5067166621 0.27075326
## 29  0.0707920054 0.75904288
## 30  0.4882760381 0.27788979
## 31  0.3969312292 0.97731666
## 32  0.5253981669 0.40926696
## 33  0.5058445535 0.35889997
## 34  0.7965803898 0.31126656
## 35  0.0596600999 0.25050042
## 36  0.7038921718 0.76961634
## 37  0.1191473629 0.64044271
## 38  0.9839006439 0.94068814
## 39  0.4704159640 0.30293546
## 40  0.2987238981 0.19892445
## 41  0.8741614837 0.42501121
## 42  0.7907014692 0.31980800
## 43  0.9785891976 0.59483399
## 44  0.5011787198 0.41937950
## 45  0.9215252022 0.51594234
## 46  0.6987903509 0.95576815
## 47  0.4357083272 0.29526233
## 48  0.4117776004 0.38421484
## 49  0.4168347928 0.10734488
## 50  0.9232791164 0.75880276
## 51  0.9032334939 0.24470358
## 52  0.3211764796 0.65926991
## 53  0.2621534232 0.89903603
## 54  0.3911324134 0.85420419
## 55  0.5994308286 0.56347249
## 56  0.1905797804 0.45521579
## 57  0.0301213486 0.23576408
## 58  0.8193471823 0.88387164
## 59  0.3885250064 0.13804423
```

```
## 60  0.9287083931 0.30271911
## 61  0.2830678045 0.13248990
## 62  0.1789407472 0.01514116
## 63  0.3676212989 0.44306144
## 64  0.7971775078 0.41296414
## 65  0.6569309647 0.31814069
## 66  0.2762217510 0.44418724
## 67  0.4994225851 0.41114025
## 68  0.1897352901 0.78714515
## 69  0.4413122577 0.04854814
## 70  0.9033120016 0.85028695
## 71  0.4265394043 0.72077410
## 72  0.3448200265 0.80585701
## 73  0.2047726877 0.60327130
## 74  0.2204449545 0.89111093
## 75  0.6505701537 0.48149126
## 76  0.1425985745 0.80020751
## 77  0.0936325965 0.98537957
## 78  0.7073836210 0.69407463
## 79  0.5972432334 0.67179105
## 80  0.4497663204 0.09625875
## 81  0.3366073773 0.53599529
## 82  0.0005900322 0.85409393
## 83  0.0109457942 0.24297398
## 84  0.2126020296 0.43961311
## 85  0.8253891452 0.38094641
## 86  0.3196911684 0.13826310
## 87  0.2563249010 0.49484376
## 88  0.0729169154 0.11026593
## 89  0.1348894653 0.39964117
## 90  0.9543932071 0.99344686
## 91  0.7279738130 0.76181286
## 92  0.2284339434 0.11774609
## 93  0.0653524157 0.11063333
## 94  0.0506504318 0.88935758
## 95  0.6178512189 0.75469403
## 96  0.0067869988 0.79478008
## 97  0.6904353665 0.64182053
## 98  0.2003440587 0.79347836
## 99  0.6667320076 0.92626494
## 100 0.4037483367 0.04517744
```

```
lst[[3]]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
lst[[4]]
```

```
## [1] "a"
```

```
# index multiple list elements
lst[1:2]
```

```
## [[1]]
##   [1] 0.042178699 0.345887301 0.139452587 0.206275186 0.452084869 0.717351019
##   [7] 0.126463111 0.640952523 0.801419775 0.703852942 0.692316750 0.661239739
##  [13] 0.602484902 0.812405247 0.552596993 0.435955625 0.534386890 0.944155622
```

```
##  [19] 0.923601537 0.015106945 0.745050451 0.073284685 0.113772524 0.449166791
##  [25] 0.995989168 0.743967811 0.220956105 0.663049563 0.898064294 0.068160296
##  [31] 0.173852696 0.506830390 0.614981351 0.033218715 0.970304654 0.991651755
##  [37] 0.115741640 0.570753892 0.315901366 0.367685139 0.573164525 0.357366330
##  [43] 0.080241389 0.858652729 0.210387068 0.499990614 0.745983964 0.127897062
##  [49] 0.806415981 0.581949470 0.912762206 0.656683683 0.221398124 0.001243219
##  [55] 0.800384199 0.872079416 0.026579988 0.528983004 0.477748008 0.930355262
##  [61] 0.428420319 0.953042187 0.905021955 0.585571432 0.338094982 0.582047309
##  [67] 0.034133181 0.349469113 0.802252213 0.363345457 0.013893382 0.417913733
##  [73] 0.996726355 0.191934285 0.059203505 0.467623840 0.828425161 0.591229726
##  [79] 0.772098457 0.340390051 0.617618100 0.077781499 0.351816075 0.481981283
##  [85] 0.323888728 0.605172041 0.894832945 0.591233902 0.443998539 0.365908439
##  [91] 0.389481444 0.618297326 0.734688197 0.896326262 0.097087294 0.725656056
##  [97] 0.600022382 0.443346858 0.393072930 0.642777560
##
## [[2]]
##                x          y
## 1   0.5929958255 0.17029630
## 2   0.2273250618 0.53290314
## 3   0.9836112957 0.86924316
## 4   0.6195508826 0.12927133
## 5   0.5355325681 0.67682812
## 6   0.4026766657 0.43604307
## 7   0.6762322055 0.40647950
## 8   0.3963014511 0.65137982
## 9   0.6644938316 0.40649973
## 10  0.7806688529 0.11119371
## 11  0.7036440966 0.08355948
## 12  0.2227644119 0.09273169
## 13  0.9076592533 0.56791180
## 14  0.0635249263 0.92151644
## 15  0.3846838397 0.88746459
## 16  0.0828204176 0.34702496
## 17  0.0089157571 0.92305583
## 18  0.2628677031 0.55358793
## 19  0.2834337901 0.55690307
## 20  0.4602273584 0.22505792
## 21  0.7177088670 0.89639362
## 22  0.6354470754 0.54169483
## 23  0.8354455414 0.28061722
## 24  0.9358799148 0.10206755
## 25  0.5355347167 0.71250764
## 26  0.8747624087 0.06888842
## 27  0.2216022352 0.93237516
## 28  0.5067166621 0.27075326
## 29  0.0707920054 0.75904288
## 30  0.4882760381 0.27788979
## 31  0.3969312292 0.97731666
## 32  0.5253981669 0.40926696
## 33  0.5058445535 0.35889997
## 34  0.7965803898 0.31126656
## 35  0.0596600999 0.25050042
## 36  0.7038921718 0.76961634
## 37  0.1191473629 0.64044271
```

```
## 38  0.9839006439 0.94068814
## 39  0.4704159640 0.30293546
## 40  0.2987238981 0.19892445
## 41  0.8741614837 0.42501121
## 42  0.7907014692 0.31980800
## 43  0.9785891976 0.59483399
## 44  0.5011787198 0.41937950
## 45  0.9215252022 0.51594234
## 46  0.6987903509 0.95576815
## 47  0.4357083272 0.29526233
## 48  0.4117776004 0.38421484
## 49  0.4168347928 0.10734488
## 50  0.9232791164 0.75880276
## 51  0.9032334939 0.24470358
## 52  0.3211764796 0.65926991
## 53  0.2621534232 0.89903603
## 54  0.3911324134 0.85420419
## 55  0.5994308286 0.56347249
## 56  0.1905797804 0.45521579
## 57  0.0301213486 0.23576408
## 58  0.8193471823 0.88387164
## 59  0.3885250064 0.13804423
## 60  0.9287083931 0.30271911
## 61  0.2830678045 0.13248990
## 62  0.1789407472 0.01514116
## 63  0.3676212989 0.44306144
## 64  0.7971775078 0.41296414
## 65  0.6569309647 0.31814069
## 66  0.2762217510 0.44418724
## 67  0.4994225851 0.41114025
## 68  0.1897352901 0.78714515
## 69  0.4413122577 0.04854814
## 70  0.9033120016 0.85028695
## 71  0.4265394043 0.72077410
## 72  0.3448200265 0.80585701
## 73  0.2047726877 0.60327130
## 74  0.2204449545 0.89111093
## 75  0.6505701537 0.48149126
## 76  0.1425985745 0.80020751
## 77  0.0936325965 0.98537957
## 78  0.7073836210 0.69407463
## 79  0.5972432334 0.67179105
## 80  0.4497663204 0.09625875
## 81  0.3366073773 0.53599529
## 82  0.0005900322 0.85409393
## 83  0.0109457942 0.24297398
## 84  0.2126020296 0.43961311
## 85  0.8253891452 0.38094641
## 86  0.3196911684 0.13826310
## 87  0.2563249010 0.49484376
## 88  0.0729169154 0.11026593
## 89  0.1348894653 0.39964117
## 90  0.9543932071 0.99344686
## 91  0.7279738130 0.76181286
```

```
## 92  0.2284339434 0.11774609
## 93  0.0653524157 0.11063333
## 94  0.0506504318 0.88935758
## 95  0.6178512189 0.75469403
## 96  0.0067869988 0.79478008
## 97  0.6904353665 0.64182053
## 98  0.2003440587 0.79347836
## 99  0.6667320076 0.92626494
## 100 0.4037483367 0.04517744
```

Let's think about how we might use lists. For one, many functions in R output data in list format. For instance, working with network data in R through `igraph`, most things are lists. Let's explore this, both as a way to introduce lists and to talk about how to analyze network data in R.

```
#install.packages('igraph')
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

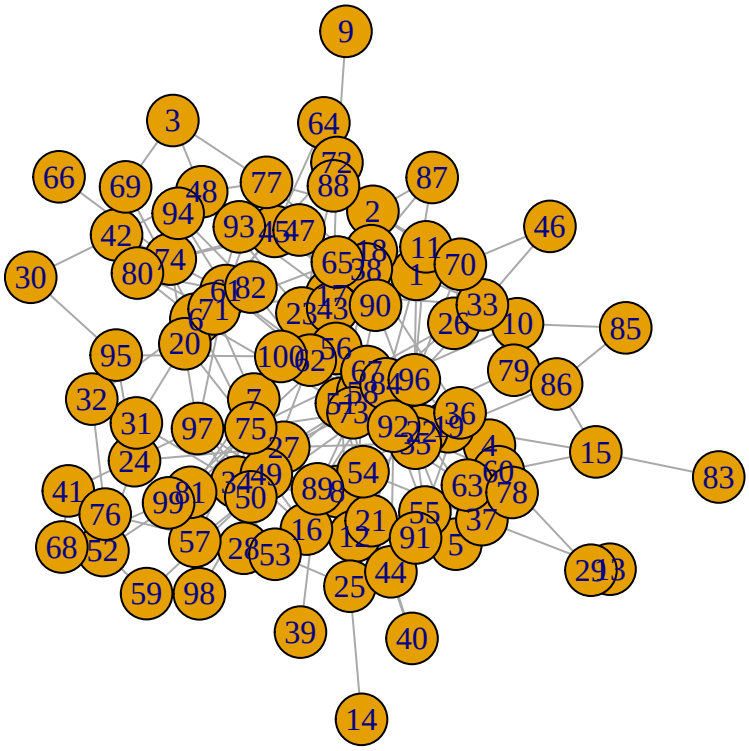
```
g <- igraph::sample_gnm(100, 200)

str(g)
```

```
## Class 'igraph'  hidden list of 10
##  $ : num 100
##  $ : logi FALSE
##  $ : num [1:200] 10 11 18 22 24 26 27 27 30 32 ...
##  $ : num [1:200] 1 4 14 6 13 23 11 20 19 0 ...
##  $ : NULL
##  $ : NULL
##  $ : NULL
##  $ : NULL
##  $ :List of 4
##   ..$ : num [1:3] 1 0 1
##   ..$ :List of 4
##   .. ..$ name : chr "Erdos-Renyi (gnm) graph"
##   .. ..$ type : chr "gnm"
##   .. ..$ loops: logi FALSE
##   .. ..$ m    : num 200
##   ..$ : list()
##   ..$ : list()
##  $ :<environment: 0x5561b6205c60>
```

Recall when we introduced the `plot` function, and I said that packages build in functionality such that some base functions will work with more complex objects (the `igraph` object above is a list). Try it here.

```
plot(g)
```

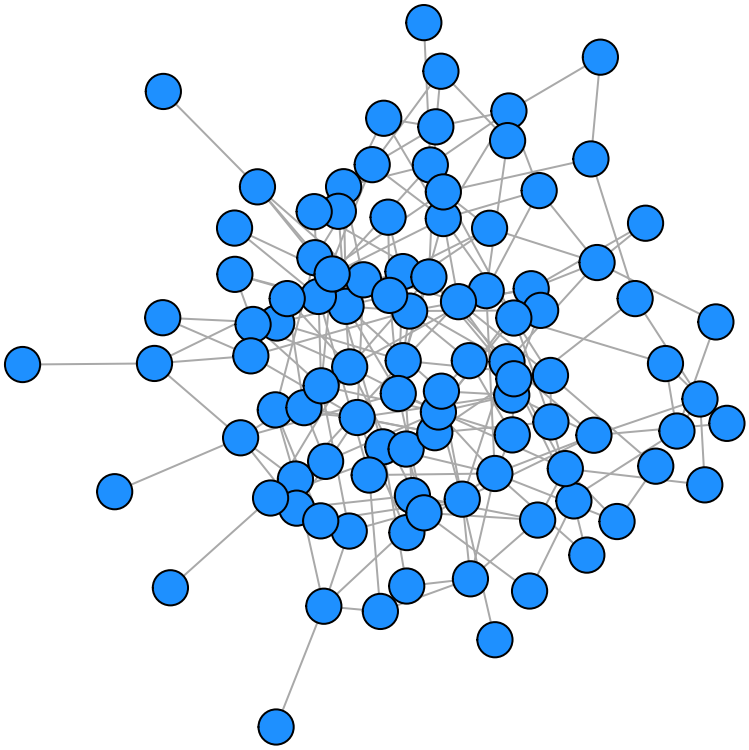


Nice. That's neat. We can also write a wrapper function (we will not go over function writing now, but will save that for the coming weeks), which can be useful across multiple projects. This is a function I routinely use for visualizing networks in a prettier way.

```
#' @param g graph object
#' @param colz
#' @param nodeColor
#' @param nodeSize
#'
#' @return a graph plot

plotGraph <- function(g, lay=layout_nicely(g), colz='dodgerblue'){
    plot(g, layout=lay, edge.width= E(g)$weight, vertex.size=10, directed=FALSE,
        vertex.color= colz,
    vertex.label=NA)
}


plotGraph(g)
```



But let's get back to lists. We've seen that `igraph` graph objects are lists, but also many of the outputs of functions within `igraph` are lists (or even lists of lists!). I will not defend nested lists as being all that useful, but we will see them in a couple of weeks when we talk about APIs and spatial data.

So one of the functions built into `igraph` is the `sir` function. This is a function which runs a model on the network known as the Susceptible-Infected-Recovered model (or SIR for short), which aims to capture how diseases spread through populations.

$$\frac{dS}{dt} = -\beta SI \tag{1}$$

$$\frac{dI}{dt} = \beta SI - \gamma I \tag{2}$$

$$\frac{dR}{dt} = \gamma I \tag{3}$$

The default behavior of the function (`?sir`) will run 100 simulations of the SIR model on a graph object that you provide to the function, and store the output as a list.

```
sims <- igraph::sir(g, beta=0.5, gamma=0.5, no.sim=100)
typeof(sims)
```

```
## [1] "list"
```

```
class(sims)
```

```
## [1] "sir"
```

```
# explore the structure of each list element
sims[[1]]
```

```
## $times
##   [1]  0.0000000  0.1921186  0.2149632  0.3312428  0.4021254  0.4726701
```

```
##   [7]  0.5800408  0.6413701  0.7161071  0.9886130  1.0304971  1.0696435
##  [13]  1.2313899  1.2872173  1.3119527  1.4449528  1.5140874  1.5755478
##  [19]  1.6062055  1.6473703  1.7077337  1.7928525  1.9688433  1.9776394
##  [25]  1.9960763  2.0607897  2.0807966  2.1513068  2.1532876  2.1898241
##  [31]  2.2713481  2.2867670  2.3019247  2.3509631  2.3918212  2.3918680
##  [37]  2.3922000  2.5481631  2.6262734  2.6385709  2.6628643  2.7045432
##  [43]  2.7519082  2.8219201  2.8325705  2.8426960  2.8435007  2.8454576
##  [49]  2.8605168  2.8910748  2.9486736  2.9881167  3.0211511  3.0342535
##  [55]  3.0757526  3.1321501  3.1367684  3.1473258  3.1487937  3.1893742
##  [61]  3.2898005  3.3066820  3.3401556  3.3510520  3.3605245  3.3749615
##  [67]  3.4082842  3.4146831  3.4194386  3.6133908  3.6428218  3.7000315
##  [73]  3.7061213  3.7338666  3.7768862  3.8196489  3.8396501  3.8509928
##  [79]  3.9119508  4.0545137  4.0881923  4.1628582  4.2105582  4.2723094
##  [85]  4.3056992  4.3125000  4.3502725  4.3504797  4.3601547  4.3844769
##  [91]  4.4808210  4.6101286  4.6386225  4.7206608  4.7344202  4.7887437
##  [97]  4.7908863  4.8782672  4.8945931  4.9015184  4.9495701  4.9640523
## [103]  4.9918006  4.9954934  5.0405440  5.1994470  5.2130955  5.2472855
## [109]  5.2483268  5.2484064  5.2954193  5.3686232  5.3748980  5.4029873
## [115]  5.4160585  5.4601942  5.5156173  5.5354701  5.7068908  5.7766615
## [121]  5.7785157  5.9292836  6.1361620  6.1435300  6.1463360  6.2195805
## [127]  6.2809526  6.3276955  6.3846034  6.4233745  6.4539118  6.4757150
## [133]  6.5328834  6.5796885  6.6043169  6.6366575  6.6476493  6.6690822
## [139]  6.8082685  6.8798711  6.9199198  7.0031820  7.0939676  7.1745020
## [145]  7.2261478  7.2590301  7.6945963  7.7038922  7.8408254  8.0291370
## [151]  8.1667851  8.8027106  8.8190854  8.8299496  9.0961298  9.1867725
## [157]  9.2331006  9.4058848  9.4305037  9.9327765 10.7029140 10.8638923
## [163] 11.1481500 11.4511554 12.5097873 14.1424459
##
## $NS
##   [1] 99 98 97 96 96 96 95 94 93 92 91 90 89 89 88 87 86 86 85 84 83 82 82 81
##  [26] 81 80 80 79 79 78 78 77 76 75 74 73 72 72 71 70 69 69 68 67 66 65 64 64 63
##  [51] 62 62 62 61 60 59 59 59 58 57 56 55 54 53 52 52 51 51 51 51 50 49 48 48 48
##  [76] 48 47 47 47 47 47 46 46 45 45 44 44 43 43 43 42 41 40 39 38 38 37 37 37 36
## [101] 35 35 35 35 34 33 33 32 31 30 30 30 29 28 28 28 27 27 26 25 24 23 22 22 22
## [126] 22 21 21 21 20 20 19 19 19 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
## [151] 18 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
##
## $NI
##   [1]  1  2  3  4  3  2  3  4  5  6  7  8  9  8  7  8  9 10  9 10 11 12 13 12 13
##  [26] 12 13 12 13 12 13 12 13 14 15 16 17 18 17 18 19 20 19 20 21 22 23 24 23 24
##  [51] 25 24 23 24 25 26 25 24 25 26 27 28 29 30 31 30 31 30 29 28 29 30 31 30 29
##  [76] 28 29 28 27 26 25 26 25 26 25 26 25 26 25 24 25 26 27 28 29 28 29 28 27 28
## [101] 29 28 27 26 27 28 27 28 29 30 29 28 29 30 29 28 29 28 29 30 31 32 33 32 31
## [126] 30 31 30 29 30 29 30 29 28 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14
## [151] 13 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
##
## $NR
##   [1]  0  0  0  0  1  2  2  2  2  2  2  2  2  3  4  4  4  4  5  5  5  5  5  6  6
##  [26]  7  7  8  8  9  9 10 10 10 10 10 10 10 11 11 11 11 12 12 12 12 12 12 13 13
##  [51] 13 14 15 15 15 15 16 17 17 17 17 17 17 17 17 18 18 19 20 21 21 21 21 22 23
##  [76] 24 24 25 26 27 28 28 29 29 30 30 31 31 32 33 33 33 33 33 33 34 34 35 36 36
## [101] 36 37 38 39 39 39 40 40 40 40 41 42 42 42 43 44 44 45 45 45 45 45 45 46 47
## [126] 48 48 49 50 50 51 51 52 53 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
## [151] 69 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
```

```
# each list element is another list
sims[[1]][[1]]
```

```
##   [1]  0.0000000  0.1921186  0.2149632  0.3312428  0.4021254  0.4726701
##   [7]  0.5800408  0.6413701  0.7161071  0.9886130  1.0304971  1.0696435
##  [13]  1.2313899  1.2872173  1.3119527  1.4449528  1.5140874  1.5755478
##  [19]  1.6062055  1.6473703  1.7077337  1.7928525  1.9688433  1.9776394
##  [25]  1.9960763  2.0607897  2.0807966  2.1513068  2.1532876  2.1898241
##  [31]  2.2713481  2.2867670  2.3019247  2.3509631  2.3918212  2.3918680
##  [37]  2.3922000  2.5481631  2.6262734  2.6385709  2.6628643  2.7045432
##  [43]  2.7519082  2.8219201  2.8325705  2.8426960  2.8435007  2.8454576
##  [49]  2.8605168  2.8910748  2.9486736  2.9881167  3.0211511  3.0342535
##  [55]  3.0757526  3.1321501  3.1367684  3.1473258  3.1487937  3.1893742
##  [61]  3.2898005  3.3066820  3.3401556  3.3510520  3.3605245  3.3749615
##  [67]  3.4082842  3.4146831  3.4194386  3.6133908  3.6428218  3.7000315
##  [73]  3.7061213  3.7338666  3.7768862  3.8196489  3.8396501  3.8509928
##  [79]  3.9119508  4.0545137  4.0881923  4.1628582  4.2105582  4.2723094
##  [85]  4.3056992  4.3125000  4.3502725  4.3504797  4.3601547  4.3844769
##  [91]  4.4808210  4.6101286  4.6386225  4.7206608  4.7344202  4.7887437
##  [97]  4.7908863  4.8782672  4.8945931  4.9015184  4.9495701  4.9640523
## [103]  4.9918006  4.9954934  5.0405440  5.1994470  5.2130955  5.2472855
## [109]  5.2483268  5.2484064  5.2954193  5.3686232  5.3748980  5.4029873
## [115]  5.4160585  5.4601942  5.5156173  5.5354701  5.7068908  5.7766615
## [121]  5.7785157  5.9292836  6.1361620  6.1435300  6.1463360  6.2195805
## [127]  6.2809526  6.3276955  6.3846034  6.4233745  6.4539118  6.4757150
## [133]  6.5328834  6.5796885  6.6043169  6.6366575  6.6476493  6.6690822
## [139]  6.8082685  6.8798711  6.9199198  7.0031820  7.0939676  7.1745020
## [145]  7.2261478  7.2590301  7.6945963  7.7038922  7.8408254  8.0291370
## [151]  8.1667851  8.8027106  8.8190854  8.8299496  9.0961298  9.1867725
## [157]  9.2331006  9.4058848  9.4305037  9.9327765 10.7029140 10.8638923
## [163] 11.1481500 11.4511554 12.5097873 14.1424459
```

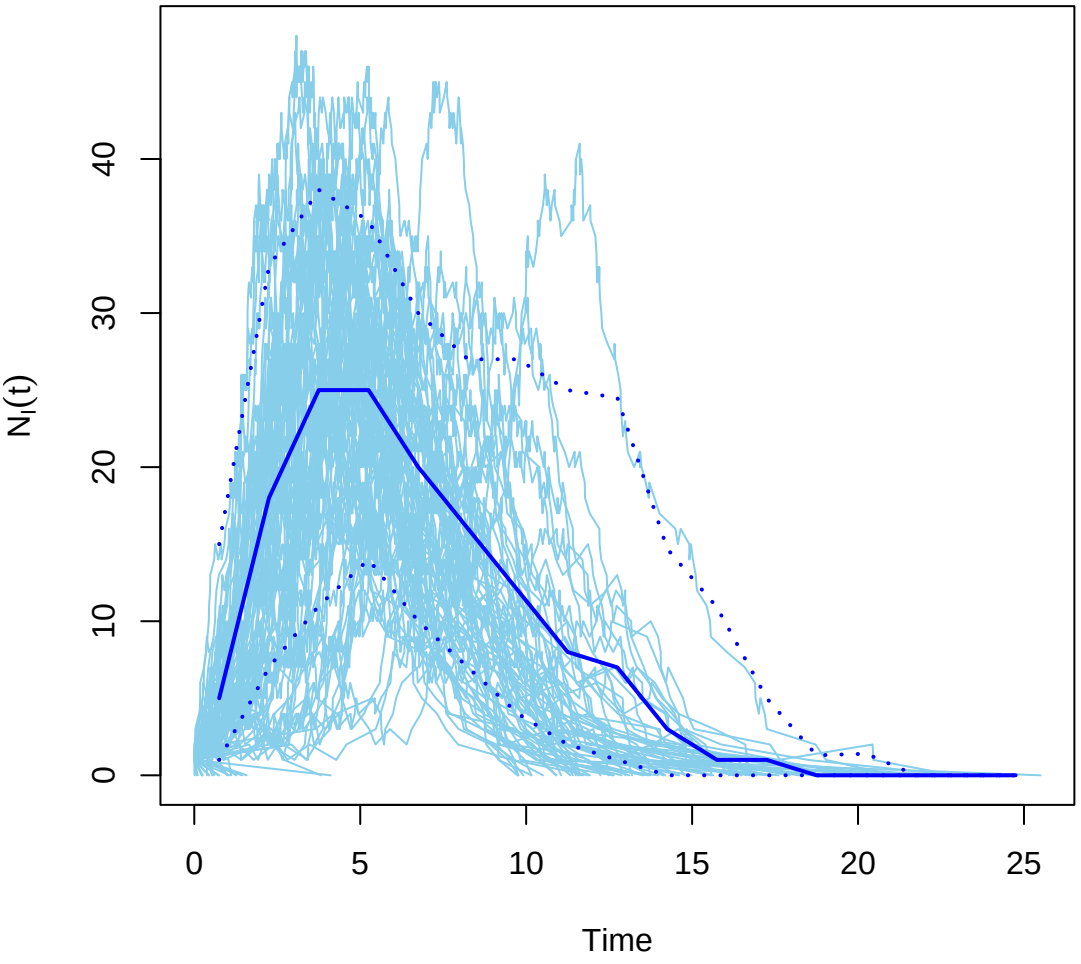
```
sims[[1]]$times
```

```
##   [1]  0.0000000  0.1921186  0.2149632  0.3312428  0.4021254  0.4726701
##   [7]  0.5800408  0.6413701  0.7161071  0.9886130  1.0304971  1.0696435
##  [13]  1.2313899  1.2872173  1.3119527  1.4449528  1.5140874  1.5755478
##  [19]  1.6062055  1.6473703  1.7077337  1.7928525  1.9688433  1.9776394
##  [25]  1.9960763  2.0607897  2.0807966  2.1513068  2.1532876  2.1898241
##  [31]  2.2713481  2.2867670  2.3019247  2.3509631  2.3918212  2.3918680
##  [37]  2.3922000  2.5481631  2.6262734  2.6385709  2.6628643  2.7045432
##  [43]  2.7519082  2.8219201  2.8325705  2.8426960  2.8435007  2.8454576
##  [49]  2.8605168  2.8910748  2.9486736  2.9881167  3.0211511  3.0342535
##  [55]  3.0757526  3.1321501  3.1367684  3.1473258  3.1487937  3.1893742
##  [61]  3.2898005  3.3066820  3.3401556  3.3510520  3.3605245  3.3749615
##  [67]  3.4082842  3.4146831  3.4194386  3.6133908  3.6428218  3.7000315
##  [73]  3.7061213  3.7338666  3.7768862  3.8196489  3.8396501  3.8509928
##  [79]  3.9119508  4.0545137  4.0881923  4.1628582  4.2105582  4.2723094
##  [85]  4.3056992  4.3125000  4.3502725  4.3504797  4.3601547  4.3844769
##  [91]  4.4808210  4.6101286  4.6386225  4.7206608  4.7344202  4.7887437
##  [97]  4.7908863  4.8782672  4.8945931  4.9015184  4.9495701  4.9640523
## [103]  4.9918006  4.9954934  5.0405440  5.1994470  5.2130955  5.2472855
## [109]  5.2483268  5.2484064  5.2954193  5.3686232  5.3748980  5.4029873
## [115]  5.4160585  5.4601942  5.5156173  5.5354701  5.7068908  5.7766615
## [121]  5.7785157  5.9292836  6.1361620  6.1435300  6.1463360  6.2195805
```

```
## [127]  6.2809526  6.3276955  6.3846034  6.4233745  6.4539118  6.4757150
## [133]  6.5328834  6.5796885  6.6043169  6.6366575  6.6476493  6.6690822
## [139]  6.8082685  6.8798711  6.9199198  7.0031820  7.0939676  7.1745020
## [145]  7.2261478  7.2590301  7.6945963  7.7038922  7.8408254  8.0291370
## [151]  8.1667851  8.8027106  8.8190854  8.8299496  9.0961298  9.1867725
## [157]  9.2331006  9.4058848  9.4305037  9.9327765 10.7029140 10.8638923
## [163] 11.1481500 11.4511554 12.5097873 14.1424459
```

And just to go back to plotting for a quick second, `igraph` has written functionality into the `sir` class to work well with the base R `plot` function.

```
plot(sims)
```



Neat, right?

But back to lists. Let's work through the rest of working with lists through some exercises. Given the simulations above (`sims` list) ...

Calculate the mean number of infected individuals for each simulation

Find the time associated with the maximum number of infected nodes

Calculate the fraction of all simulations in which fewer than 5 nodes are infected

apply statements

How did we approach the above questions? You almost certaintly used a `for` loop, right? This is perfectly fine, but there is another way. `apply` statements allow you to take a function and run it over all elements of a vector, columns/rows of a matrix, or a list.

`apply` statements typically have a prefix which gives information about what type of data it works well with. For instance, working with lists, we will use `lapply`. The base `apply` function is to work with matrices, where we want to apply a function to every row or column of a matrix (e.g., `apply(matrix, 2, sum)` is the same as `colSums(matrix)`). We will go over more examples of `apply` statements at some point, but for now we will focus on `lapply` and our problem of working with lists.

And here we hit an issue. `lapply` statements take a function argument, where the function needs to take the list object as an argument and then does something with it. So we'll have to learn a bare minimum of function writing now. Let's use a problem above as a motivating example, where we try to calculate the mean number of infected individuals for each simulation.

```
meanInfections <- function(x){
  # if we consider the mean infecteds as the mean of the infected vector
  return(mean(x$NI))
  # if we consider the mean infecteds as the mean number of nodes infected
  # return((x$NS[1]+1)-tail(x$NS,1))
}


meanInfs <- lapply(sims, meanInfections)
str(meanInfs)
```

```
## List of 100
##  $ : num 20.6
##  $ : num 18.4
##  $ : num 0.5
##  $ : num 24.7
##  $ : num 26.4
##  $ : num 24.2
##  $ : num 25.1
##  $ : num 21.4
##  $ : num 15.1
##  $ : num 20.9
##  $ : num 18.6
##  $ : num 1.5
##  $ : num 15.8
##  $ : num 18.2
##  $ : num 13.4
##  $ : num 18.1
##  $ : num 21.7
##  $ : num 0.5
##  $ : num 1
##  $ : num 27.5
##  $ : num 23.6
##  $ : num 20.2
##  $ : num 19
##  $ : num 0.5
##  $ : num 15.2
##  $ : num 1
##  $ : num 23.5
```

```
##  $ : num 12.9
##  $ : num 15.4
##  $ : num 19.6
##  $ : num 21.3
##  $ : num 14.5
##  $ : num 25.8
##  $ : num 14.2
##  $ : num 0.5
##  $ : num 21.1
##  $ : num 0.5
##  $ : num 17.5
##  $ : num 0.5
##  $ : num 21
##  $ : num 1
##  $ : num 19.7
##  $ : num 9.22
##  $ : num 16.8
##  $ : num 26.3
##  $ : num 24.9
##  $ : num 12.5
##  $ : num 12.2
##  $ : num 1
##  $ : num 0.5
##  $ : num 0.5
##  $ : num 23.3
##  $ : num 21.8
##  $ : num 21.4
##  $ : num 17.3
##  $ : num 13.4
##  $ : num 23.6
##  $ : num 8.04
##  $ : num 1
##  $ : num 18.8
##  $ : num 13.3
##  $ : num 25.4
##  $ : num 18.8
##  $ : num 27.7
##  $ : num 21.3
##  $ : num 21.7
##  $ : num 0.5
##  $ : num 0.5
##  $ : num 1
##  $ : num 20.1
##  $ : num 15.6
##  $ : num 18.9
##  $ : num 13.3
##  $ : num 1
##  $ : num 17.4
##  $ : num 24
##  $ : num 12.5
##  $ : num 16
##  $ : num 19.4
##  $ : num 16.9
##  $ : num 21.5
```

```
##  $ : num 16.9
##  $ : num 26.7
##  $ : num 24.1
##  $ : num 20.8
##  $ : num 25.6
##  $ : num 0.5
##  $ : num 0.5
##  $ : num 22.7
##  $ : num 11.3
##  $ : num 18.5
##  $ : num 14.5
##  $ : num 1.17
##  $ : num 17.3
##  $ : num 20.3
##  $ : num 0.5
##  $ : num 15.5
##  $ : num 0.5
##  $ : num 17.6
##   [list output truncated]
```

`lapply` is nice in that if we give it a list object, it gives us a list object back. This makes analytical pipelines that deal with lists pretty straightforward, but if the output is a single value, we may want this to be a vector instead of a list.

```
unlist(meanInfs)
```

```
##   [1] 20.596386 18.431507  0.500000 24.664835 26.352273 24.231707 25.077778
##   [8] 21.387640 15.104938 20.873626 18.586957  1.500000 15.757143 18.223684
##  [15] 13.414286 18.092105 21.664179  0.500000  1.000000 27.466667 23.596386
##  [22] 20.235632 19.000000  0.500000 15.220000  1.000000 23.535294 12.892857
##  [29] 15.402778 19.585366 21.290123 14.486842 25.782353 14.234177  0.500000
##  [36] 21.125000  0.500000 17.525000  0.500000 20.971264  1.000000 19.662162
##  [43]  9.224638 16.808642 26.316092 24.900000 12.511905 12.166667  1.000000
##  [50]  0.500000  0.500000 23.253086 21.783784 21.371795 17.285714 13.426471
##  [57] 23.650000  8.037037  1.000000 18.770270 13.342857 25.379121 18.842857
##  [64] 27.675824 21.347059 21.660920  0.500000  0.500000  1.000000 20.084416
##  [71] 15.589744 18.909091 13.253247  1.000000 17.371795 24.000000 12.455224
##  [78] 16.013158 19.357143 16.948276 21.512195 16.897260 26.683908 24.117284
##  [85] 20.833333 25.566667  0.500000  0.500000 22.660920 11.340000 18.486842
##  [92] 14.525316  1.166667 17.253247 20.290698  0.500000 15.545977  0.500000
##  [99] 17.608434 28.467391
```

```
#or
meanInfs2 <- sapply(sims, meanInfections)
```

`sapply` statements are essentially just `lapply` statements that simplify the result to a vector. This is useful when the output of the function is a single value, and not so useful when function returns multiple values.

A side note: some people will criticize `for` loops in R, and say "just use apply, it's faster". It's not, really. Write however you feel comfortable. For awhile, `apply` statements were super confusing to me, so I tended to use `for` loops instead. After more work in, I shifted and tend to use `apply` statements when it fits, as they are less code and are more intuitive to me for many situations.

Let's practice a bit.

Calculate the maximum number of infected inviduals at any time in the `sims` list using the `apply` approach.

What is the mean duration (the total time the epidemic took before it stopped) across all the epidemics in `sims`?

plyr apply functionality tweaks

`XYply` statements as nice wrappers to more classic `apply` statements. Here, `X` and `Y` can take values of 'a', 'l', or 'd', depending on the input or output data structure desired. For instance, if we have a list that we would like to apply over and return a data.frame, we would use `ldply`, where the `l` is claiming that the input is a list object, and the `d` is claiming that the output should be formatted as a data.frame. Other examples of this syntax would be `adply`, `ddply`, `laply`, `aaply`, etc. etc.

Below, I provide an example of the aXply syntax (e.g., adply, alply, aaply).

```
arr <- array(1:27, c(3,3,3))
rownames(arr) = c("Curly", "Larry", "Moe")
colnames(arr) = c("Groucho", "Harpo", "Zeppo")
dimnames(arr)[[3]] = c("Bart", "Lisa", "Maggie")

arr
```

```
## , , Bart
##
##       Groucho Harpo Zeppo
## Curly       1     4     7
## Larry       2     5     8
## Moe         3     6     9
##
## , , Lisa
##
##       Groucho Harpo Zeppo
## Curly      10    13    16
## Larry      11    14    17
## Moe        12    15    18
##
## , , Maggie
##
##       Groucho Harpo Zeppo
## Curly      19    22    25
## Larry      20    23    26
## Moe        21    24    27
```

Arrays are something that we did not introduce when we talked about `R` basics, and that is because they really are not used *too* often. Think of matrix. It has two dimensions (x and y), so it can be viewed as a rectangle of data. Arrays simply add more dimensions. In the example above, there is another dimension, forming a data cube (in the rectangle analogy).

We can use `plyr` functionality to operate on this array and return different forms. For instance, `aaply` takes an array and returns a simplified array (here a vector).

```
plyr::aaply(arr, 1, sum)
```

```
## Curly Larry   Moe
##   117   126   135
```

We can change one letter and now return a data.frame containing two columns. This is also a good time to point out the flexibility of the XYply statements to different margins. Margins (denoted as `.margins` argument in `R`, asks along which axis you would like to operate on the array. If we set .margins=1, this corresponds to a row-wise operation, so we calculate the sum across the array for Curly, Larry, and Moe. If

we change this to .margins=2, we operate on columns, and will return sums for Groucho, Harpo, and Zeppo. And if we use .margins=3, we will return sums for Bart, Lisa, and Maggie.

```
plyr::adply(.data=arr, .margins=1, .fun=sum)
```

```
##      X1  V1
## 1 Curly 117
## 2 Larry 126
## 3   Moe 135
```

```
plyr::adply(.data=arr, .margins=2, .fun=sum)
```

```
##        X1  V1
## 1 Groucho  99
## 2   Harpo 126
## 3   Zeppo 153
```

```
plyr::adply(.data=arr, .margins=3, .fun=sum)
```

```
##       X1  V1
## 1   Bart  45
## 2   Lisa 126
## 3 Maggie 207
```

Finally, we can return a list object. In this use case, this is not super helpful, but in other use cases the list output is pretty helpful.

```
plyr::alply(.data=arr, .margins=1, .fun=sum)
```

```
## $`1`
## [1] 117
##
## $`2`
## [1] 126
##
## $`3`
## [1] 135
##
## attr(,"split_type")
## [1] "array"
## attr(,"split_labels")
##      X1
## 1 Curly
## 2 Larry
## 3   Moe
```

A pitch for `plyr::ldply`. I really like this function, as I often find myself with lists of similar structures that I want to operate on and get a single clean object back. I will not go into an example, but this is a pretty useful function (though all the utility is basically contained in `vapply`).

Finally, you may wonder why am I pushing apply statements so hard. It has nothing to do with speed, and only a bit to do with code clarity. The main advantage is understanding the programmatic nature of apply statements (which will be similar but less chronological than a for loop), and many parallel computing packages have their own little versions of apply statements ready to go (e.g., `parallel::mclapply`, `parallel::parLapply`, `parallel::clusterApplyLB`).

Let's do one practice problem to showcase the utility of `ldply` specifically.

Calculate the correlation between number of infections and time for each simulation, reporting the estimate,

p-value, and confidence intervals around the estimate. (you will use `cor.test` to do this, whose output is a list object as well)

A note about do.call and Reduce

While a bit opaque, these functions are pretty useful in a variety of situations. Speaking of data manipulation functions that are useful but a bit conceptually difficult, `do.call` and `Reduce` are solid base `R` functions.

`do.call` is a way of calling the same function recursively on multiple objects, and may have similar output to `Reduce`, which is also a way to recursively apply a function.

```
lst <- list(1:10, 1:10, 1:10, 1:10, 1:10)
lst
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[3]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[4]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[5]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
#this makes a single rbind call with each element of the list as an argument
str(do.call(rbind, lst))
```

```
##  int [1:5, 1:10] 1 1 1 1 1 2 2 2 2 2 ...
```

```
#this does it iteratively (so makes n-1 rbind calls)
Reduce(rbind, lst)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## init    1    2    3    4    5    6    7    8    9    10
##         1    2    3    4    5    6    7    8    9    10
##         1    2    3    4    5    6    7    8    9    10
##         1    2    3    4    5    6    7    8    9    10
##         1    2    3    4    5    6    7    8    9    10
```

sessionInfo

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.10.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
```

```
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] igraph_1.5.0.1 dplyr_1.1.2    plyr_1.8.8     DBI_1.1.3      rgbif_3.7.7
## [6] jsonlite_1.8.7 httr_1.4.6     rmarkdown_2.23
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.3        generics_0.1.3    xml2_1.3.5        RSQLite_2.3.1
##  [5] stringi_1.7.12    httpcode_0.3.0    digest_0.6.33     magrittr_2.0.3
##  [9] evaluate_0.21     grid_4.3.1        fastmap_1.1.1     blob_1.2.4
## [13] maps_3.4.1        whisker_0.4.1     crul_1.4.0        tinytex_0.45
## [17] urltools_1.7.3    purrr_1.0.1       fansi_1.0.4       scales_1.2.1
## [21] oai_0.4.0         lazyeval_0.2.2    cli_3.6.1         rlang_1.1.1
## [25] dbplyr_2.3.2      triebeard_0.4.1   bit64_4.0.5       munsell_0.5.0
## [29] withr_2.5.0       cachem_1.0.8      yaml_2.3.7        tools_4.3.1
## [33] memoise_2.0.1     colorspace_2.1-0  ggplot2_3.4.2     curl_5.0.1
## [37] vctrs_0.6.3       R6_2.5.1          lifecycle_1.0.3   stringr_1.5.0
## [41] bit_4.0.5         pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.3
## [45] data.table_1.14.8 glue_1.6.2        Rcpp_1.0.11       xfun_0.39
## [49] tibble_3.2.1      tidyselect_1.2.0  highr_0.10        knitr_1.43
## [53] htmltools_0.5.5   compiler_4.3.1
```