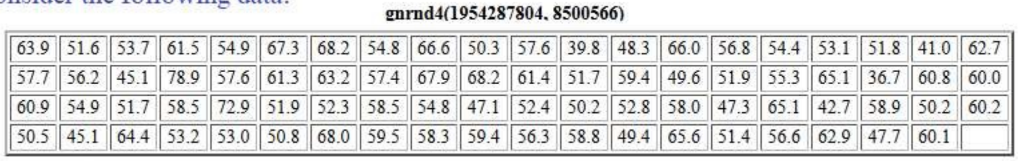
Topic 7m: Computing in R: Frequency Tables -- Grouped Values

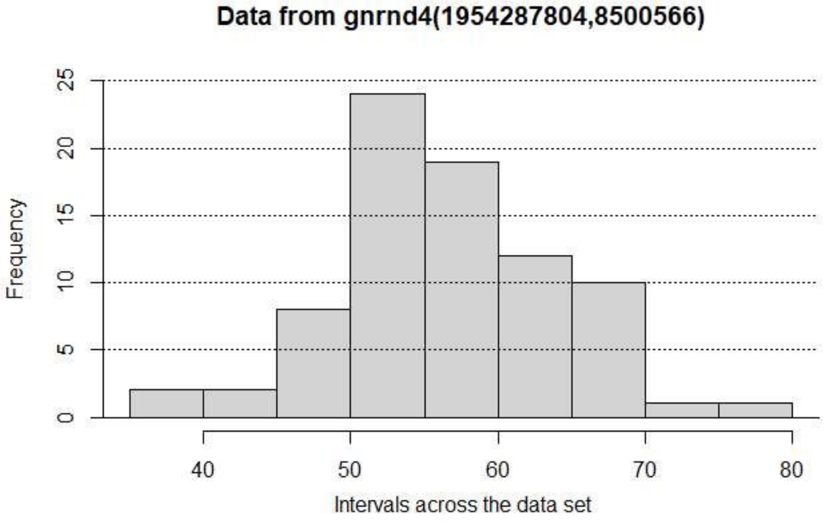
Consider the following data:



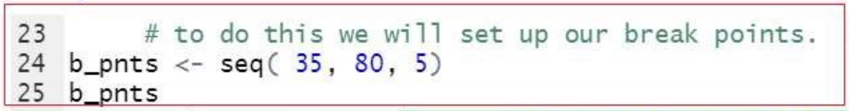
gnrnd4(1954287804, 8500566)

63.9	51.6	53.7	61.5	54.9	67.3	68.2	54.8	66.6	50.3	57.6	39.8	48.3	66.0	56.8	54.4	53.1	51.8	41.0	62.7
57.7	56.2	45.1	78.9	57.6	61.3	63.2	57.4	67.9	68.2	61.4	51.7	59.4	49.6	51.9	55.3	65.1	36.7	60.8	60.0
60.9	54.9	51.7	58.5	72.9	51.9	52.3	58.5	54.8	47.1	52.4	50.2	52.8	58.0	47.3	65.1	42.7	58.9	50.2	60.2
50.5	45.1	64.4	53.2	53.0	50.8	68.0	59.5	58.3	59.4	56.3	58.8	49.4	65.6	51.4	56.6	62.9	47.7	60.1	

Clearly this is not the sort of discrete data for which we could make a frequency chart. But we could get a histogram to show the frequency of values inside of intervals (or bins, or buckets).



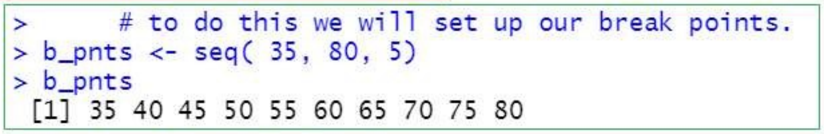
This is not a frequency table, but we could try to read the frequencies right from the histogram. However, we want to compute them. To do this in a detailed approach we could first create the breakpoints.



```
     # to do this we will set up our break points.
b_pnts <- seq( 35, 80, 5)
b_pnts
```

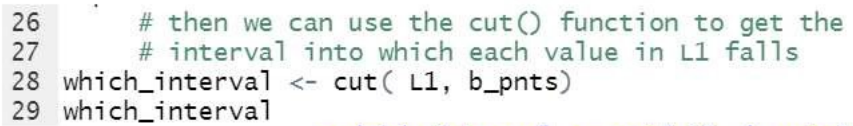
lines from the script

lines generated in the console

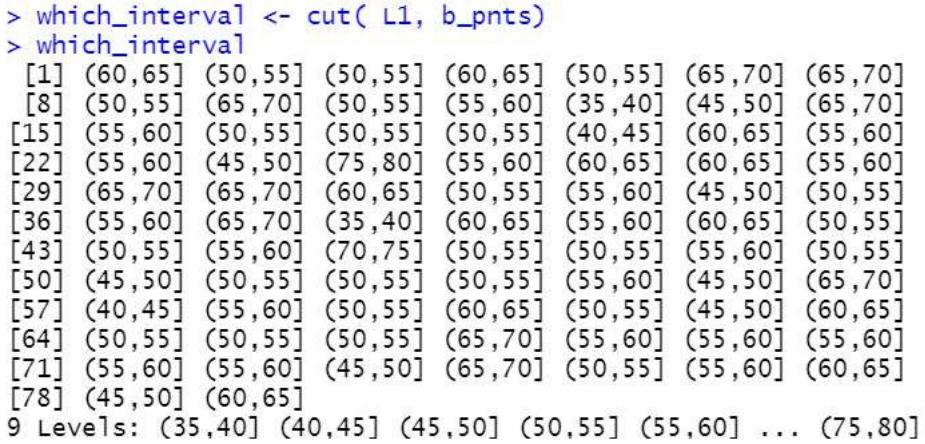


```
>      # to do this we will set up our break points.
> b_pnts <- seq( 35, 80, 5)
> b_pnts
 [1] 35 40 45 50 55 60 65 70 75 80
```

Then, get the list of intervals into which the data values will be placed.

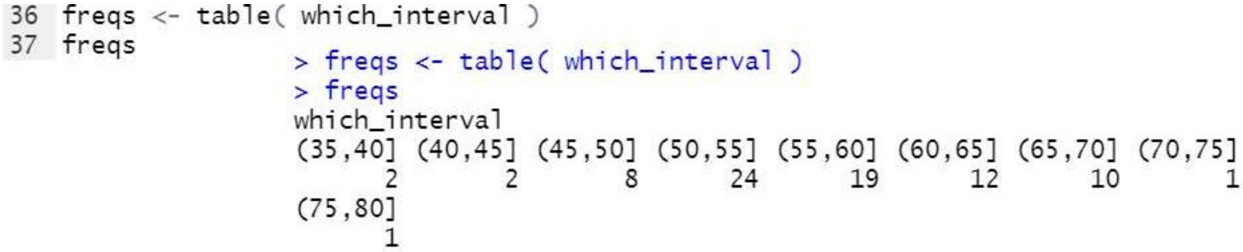


```
     # then we can use the cut() function to get the
     # interval into which each value in L1 falls
which_interval <- cut( L1, b_pnts)
which_interval
```



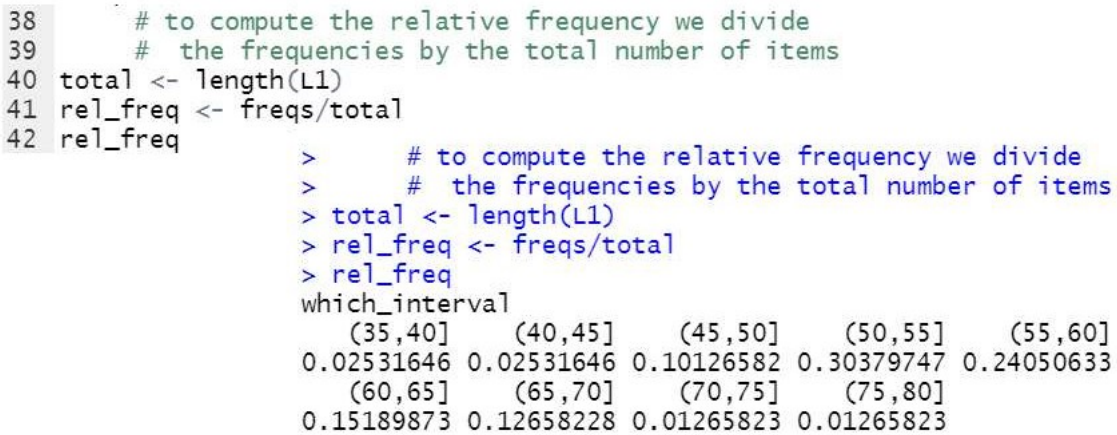
```
> which_interval <- cut( L1, b_pnts)
> which_interval
 [1] (60,65] (50,55] (50,55] (60,65] (50,55] (65,70] (65,70]
 [8] (50,55] (65,70] (50,55] (55,60] (35,40] (45,50] (65,70]
[15] (55,60] (50,55] (50,55] (50,55] (40,45] (60,65] (55,60]
[22] (55,60] (45,50] (75,80] (55,60] (60,65] (60,65] (55,60]
[29] (65,70] (65,70] (60,65] (50,55] (55,60] (45,50] (50,55]
[36] (55,60] (65,70] (35,40] (60,65] (55,60] (60,65] (50,55]
[43] (50,55] (55,60] (70,75] (50,55] (50,55] (55,60] (50,55]
[50] (45,50] (50,55] (50,55] (50,55] (55,60] (45,50] (65,70]
[57] (40,45] (55,60] (50,55] (60,65] (50,55] (45,50] (60,65]
[64] (50,55] (50,55] (50,55] (65,70] (55,60] (55,60] (55,60]
[71] (55,60] (55,60] (45,50] (65,70] (50,55] (55,60] (60,65]
[78] (45,50] (60,65]
9 Levels: (35,40] (40,45] (45,50] (50,55] (55,60] ... (75,80]
```

Now we have changed the problem into one dealing with discrete values. Proceed as before.



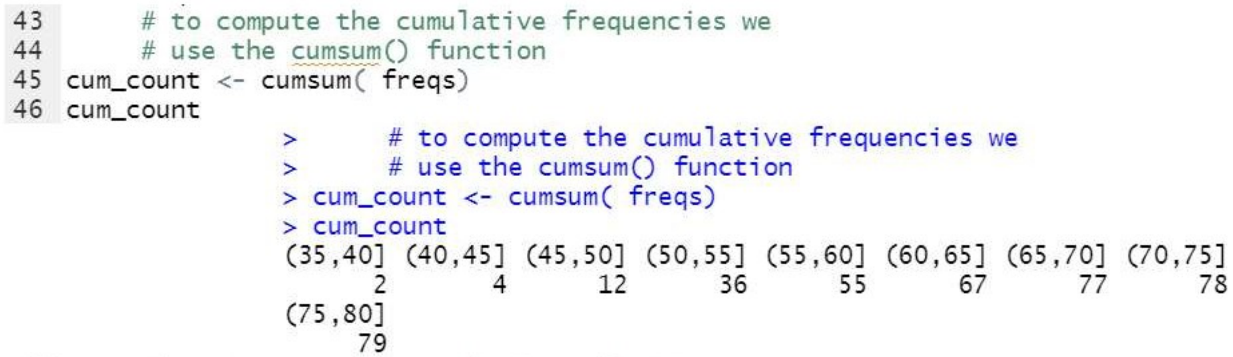
```
freqs <- table( which_interval )
freqs
```

```
> freqs <- table( which_interval )
> freqs
which_interval
(35,40] (40,45] (45,50] (50,55] (55,60] (60,65] (65,70] (70,75] 
      2       2       8      24      19      12      10       1 
(75,80] 
      1 
```



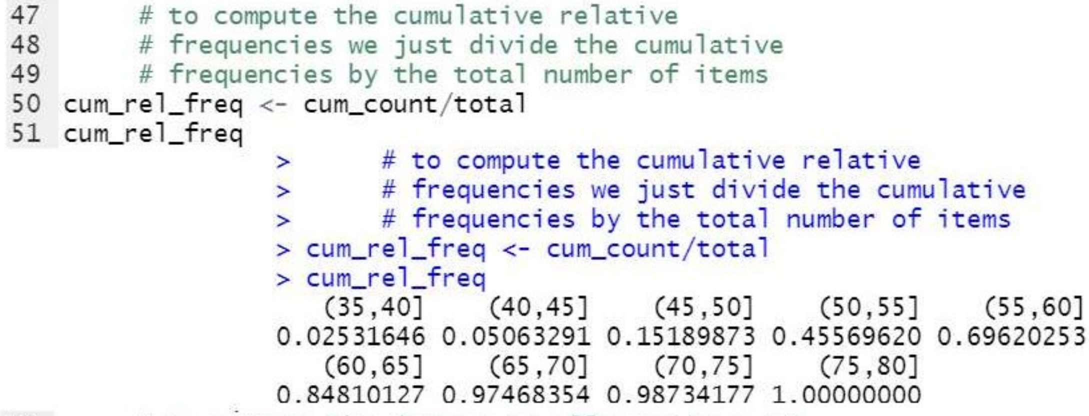
```
     # to compute the relative frequency we divide
     #  the frequencies by the total number of items
total <- length(L1)
rel_freq <- freqs/total
rel_freq
```

```
>      # to compute the relative frequency we divide
>      #  the frequencies by the total number of items
> total <- length(L1)
> rel_freq <- freqs/total
> rel_freq
which_interval
   (35,40]    (40,45]    (45,50]    (50,55]    (55,60] 
0.02531646 0.02531646 0.10126582 0.30379747 0.24050633 
   (60,65]    (65,70]    (70,75]    (75,80] 
0.15189873 0.12658228 0.01265823 0.01265823 
```



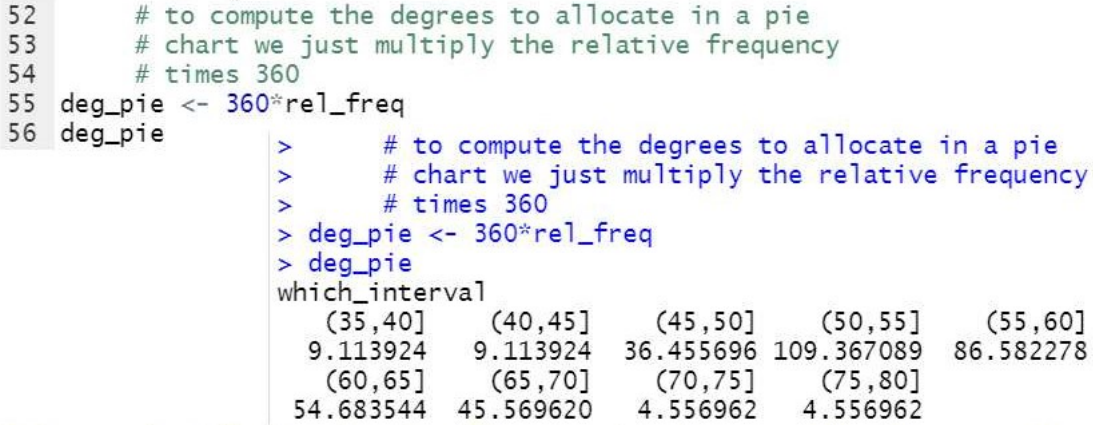
```
     # to compute the cumulative frequencies we
     # use the cumsum() function
cum_count <- cumsum( freqs)
cum_count
```

```
>      # to compute the cumulative frequencies we
>      # use the cumsum() function
> cum_count <- cumsum( freqs)
> cum_count
(35,40] (40,45] (45,50] (50,55] (55,60] (60,65] (65,70] (70,75] 
      2       4      12      36      55      67      77      78 
(75,80] 
     79 
```



```
     # to compute the cumulative relative
     # frequencies we just divide the cumulative
     # frequencies by the total number of items
cum_rel_freq <- cum_count/total
cum_rel_freq
```

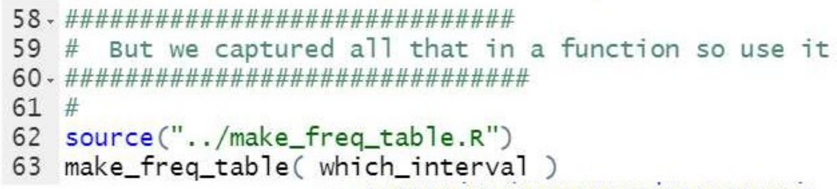
```
>      # to compute the cumulative relative
>      # frequencies we just divide the cumulative
>      # frequencies by the total number of items
> cum_rel_freq <- cum_count/total
> cum_rel_freq
   (35,40]    (40,45]    (45,50]    (50,55]    (55,60] 
0.02531646 0.05063291 0.15189873 0.45569620 0.69620253 
   (60,65]    (65,70]    (70,75]    (75,80] 
0.84810127 0.97468354 0.98734177 1.00000000 
```



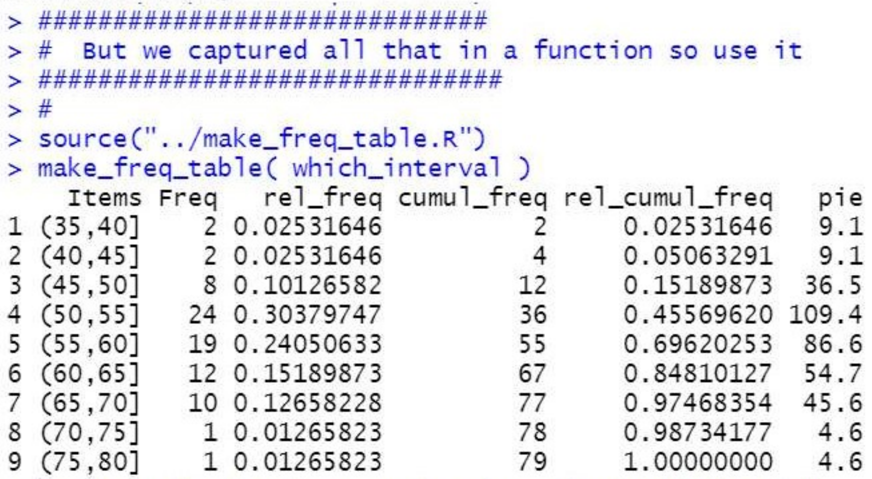
```
     # to compute the degrees to allocate in a pie
     # chart we just multiply the relative frequency
     # times 360
deg_pie <- 360*rel_freq
deg_pie
```

```
>      # to compute the degrees to allocate in a pie
>      # chart we just multiply the relative frequency
>      # times 360
> deg_pie <- 360*rel_freq
> deg_pie
which_interval
   (35,40]    (40,45]    (45,50]    (50,55]    (55,60] 
  9.113924   9.113924  36.455696 109.367089  86.582278 
   (60,65]    (65,70]    (70,75]    (75,80] 
 54.683544  45.569620   4.556962   4.556962 
```

We have produced all of the values that will have to go into a frequency table. Of course we did not have to do all of this work because, once we had the discrete values, we could have just used the make_freq_table() function and produced a nice output.

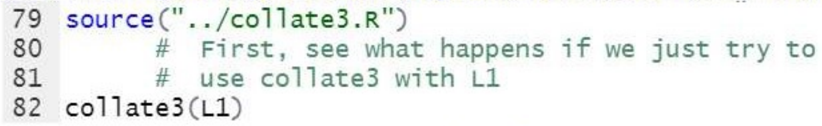


```
58 ##############################
59 #  But we captured all that in a function so use it
60 ###############################
61 #
62 source("../make_freq_table.R")
63 make_freq_table( which_interval )
```

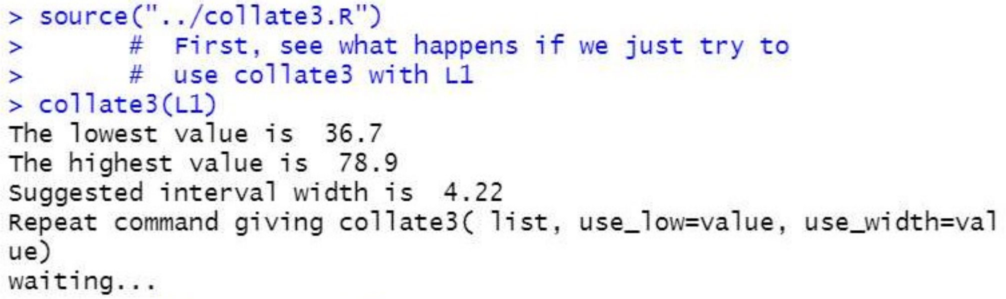


```
> ##############################
> #  But we captured all that in a function so use it
> ###############################
> #
> source("../make_freq_table.R")
> make_freq_table( which_interval )
    Items Freq   rel_freq cumul_freq rel_cumul_freq   pie
1 (35,40]    2 0.02531646          2     0.02531646   9.1
2 (40,45]    2 0.02531646          4     0.05063291   9.1
3 (45,50]    8 0.10126582         12     0.15189873  36.5
4 (50,55]   24 0.30379747         36     0.45569620 109.4
5 (55,60]   19 0.24050633         55     0.69620253  86.6
6 (60,65]   12 0.15189873         67     0.84810127  54.7
7 (65,70]   10 0.12658228         77     0.97468354  45.6
8 (70,75]    1 0.01265823         78     0.98734177   4.6
9 (75,80]    1 0.01265823         79     1.00000000   4.6
```

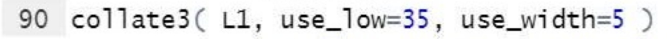
Rather than taking the steps to convert the problem to a discrete mode, we have a function that does it all.



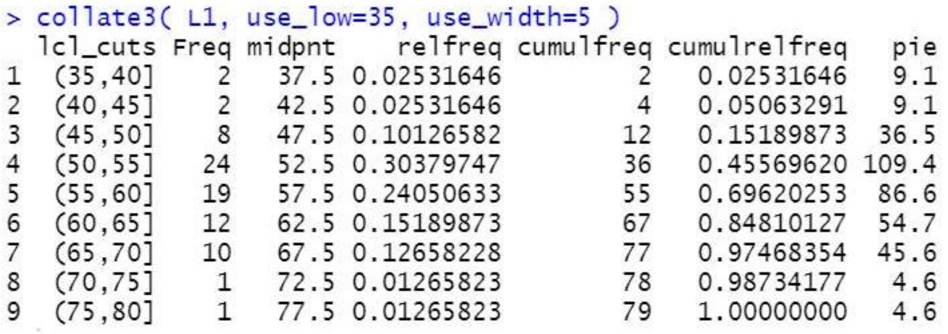
```
79 source("../collate3.R")
80       #  First, see what happens if we just try to
81       #  use collate3 with L1
82 collate3(L1)
```



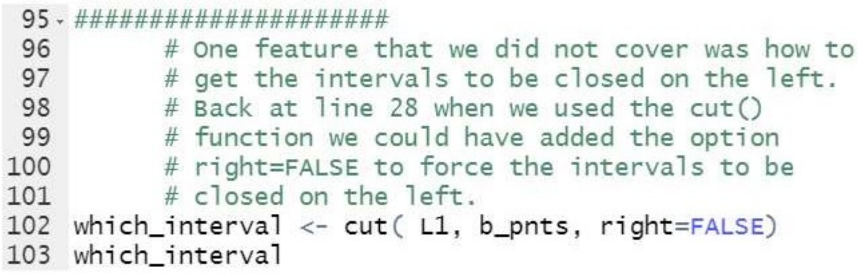
```
> source("../collate3.R")
>       #  First, see what happens if we just try to
>       #  use collate3 with L1
> collate3(L1)
The lowest value is  36.7
The highest value is  78.9
Suggested interval width is  4.22
Repeat command giving collate3( list, use_low=value, use_width=val
ue)
waiting...
```



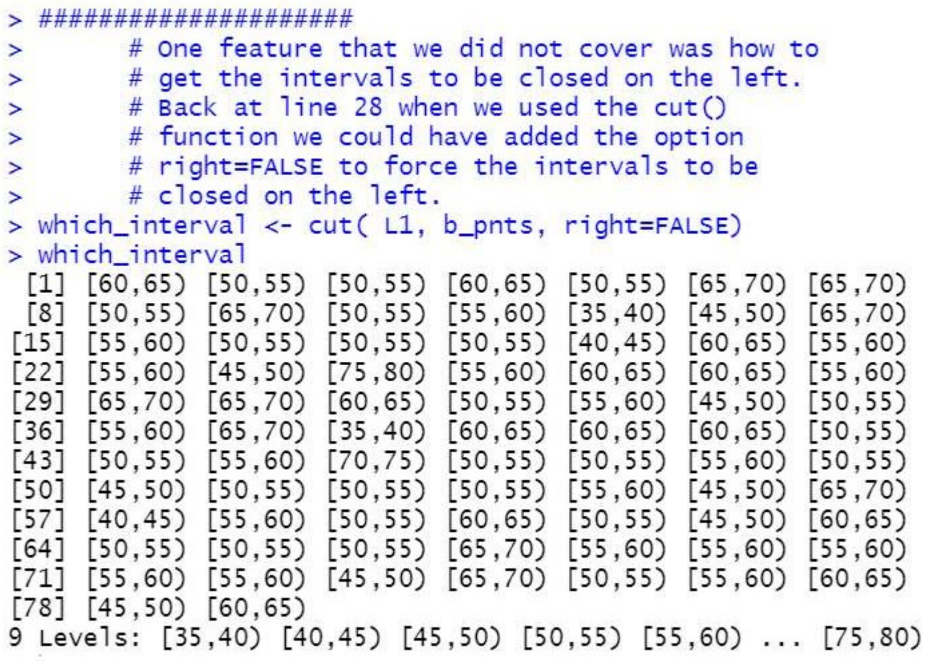
```
90 collate3( L1, use_low=35, use_width=5 )
```



```
> collate3( L1, use_low=35, use_width=5 )
  lcl_cuts Freq midpnt    relfreq cumulfreq cumulrelfreq   pie
1  (35,40]    2   37.5 0.02531646         2   0.02531646   9.1
2  (40,45]    2   42.5 0.02531646         4   0.05063291   9.1
3  (45,50]    8   47.5 0.10126582        12   0.15189873  36.5
4  (50,55]   24   52.5 0.30379747        36   0.45569620 109.4
5  (55,60]   19   57.5 0.24050633        55   0.69620253  86.6
6  (60,65]   12   62.5 0.15189873        67   0.84810127  54.7
7  (65,70]   10   67.5 0.12658228        77   0.97468354  45.6
8  (70,75]    1   72.5 0.01265823        78   0.98734177   4.6
9  (75,80]    1   77.5 0.01265823        79   1.00000000   4.6
```

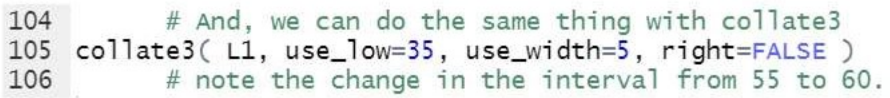


```
 95 ####################
 96       # One feature that we did not cover was how to
 97       # get the intervals to be closed on the left.
 98       # Back at line 28 when we used the cut()
 99       # function we could have added the option
100       # right=FALSE to force the intervals to be
101       # closed on the left.
102 which_interval <- cut( L1, b_pnts, right=FALSE)
103 which_interval
```

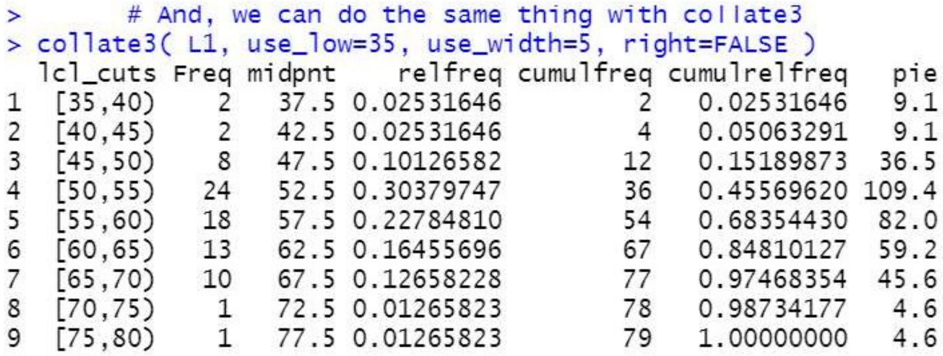


```
> ####################
>       # One feature that we did not cover was how to
>       # get the intervals to be closed on the left.
>       # Back at line 28 when we used the cut()
>       # function we could have added the option
>       # right=FALSE to force the intervals to be
>       # closed on the left.
> which_interval <- cut( L1, b_pnts, right=FALSE)
> which_interval
 [1] [60,65) [50,55) [50,55) [60,65) [50,55) [65,70) [65,70)
 [8] [50,55) [65,70) [50,55) [55,60) [35,40) [45,50) [65,70)
[15] [55,60) [50,55) [50,55) [50,55) [40,45) [60,65) [55,60)
[22] [55,60) [45,50) [75,80) [55,60) [60,65) [60,65) [55,60)
[29] [65,70) [65,70) [60,65) [50,55) [55,60) [45,50) [50,55)
[36] [55,60) [65,70) [35,40) [60,65) [60,65) [60,65) [50,55)
[43] [50,55) [55,60) [70,75) [50,55) [50,55) [55,60) [50,55)
[50] [45,50) [50,55) [50,55) [50,55) [55,60) [45,50) [65,70)
[57] [40,45) [55,60) [50,55) [60,65) [50,55) [45,50) [60,65)
[64] [50,55) [50,55) [50,55) [65,70) [55,60) [55,60) [55,60)
[71] [55,60) [55,60) [45,50) [65,70) [50,55) [55,60) [60,65)
[78] [45,50) [60,65)
9 Levels: [35,40) [40,45) [45,50) [50,55) [55,60) ... [75,80)
```

And we can do the same thing with collate3().



```
      # And, we can do the same thing with collate3
collate3( L1, use_low=35, use_width=5, right=FALSE )
      # note the change in the interval from 55 to 60.
```



```
>       # And, we can do the same thing with collate3
> collate3( L1, use_low=35, use_width=5, right=FALSE )
  lcl_cuts Freq midpnt    relfreq cumulfreq cumulrelfreq   pie
1  [35,40)    2   37.5 0.02531646         2   0.02531646   9.1
2  [40,45)    2   42.5 0.02531646         4   0.05063291   9.1
3  [45,50)    8   47.5 0.10126582        12   0.15189873  36.5
4  [50,55)   24   52.5 0.30379747        36   0.45569620 109.4
5  [55,60)   18   57.5 0.22784810        54   0.68354430  82.0
6  [60,65)   13   62.5 0.16455696        67   0.84810127  59.2
7  [65,70)   10   67.5 0.12658228        77   0.97468354  45.6
8  [70,75)    1   72.5 0.01265823        78   0.98734177   4.6
9  [75,80)    1   77.5 0.01265823        79   1.00000000   4.6
```