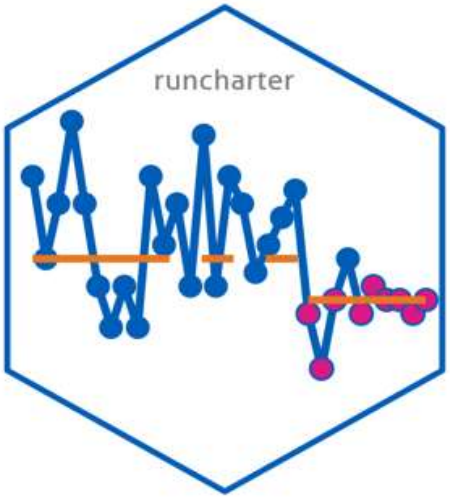
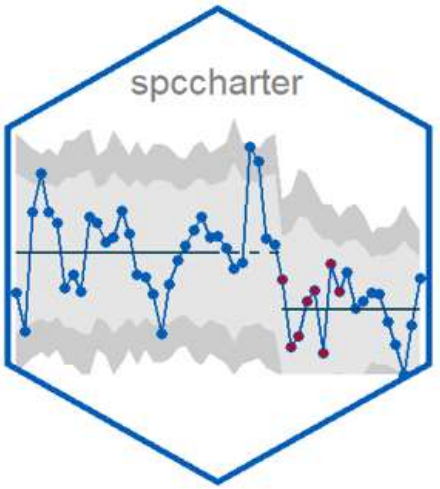
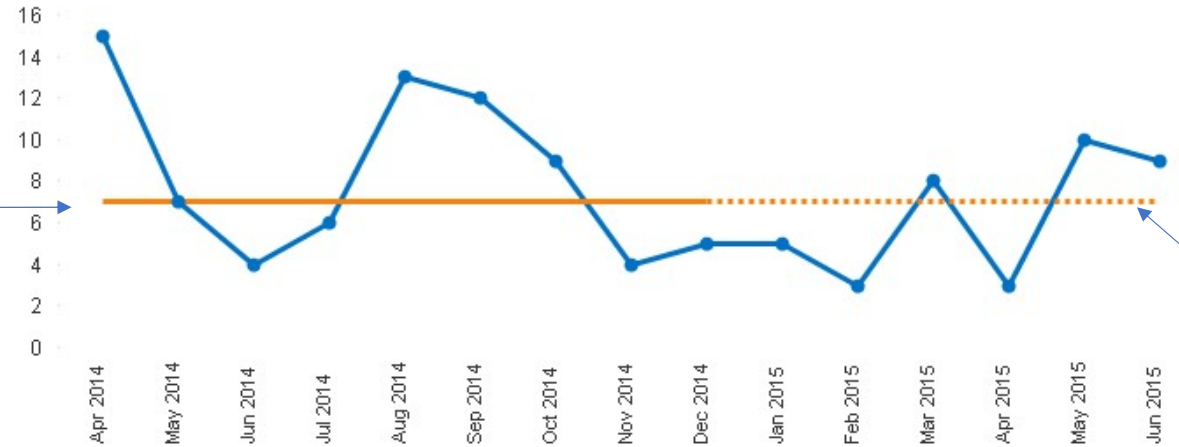
My  hat-trick





Anatomy of a run chart

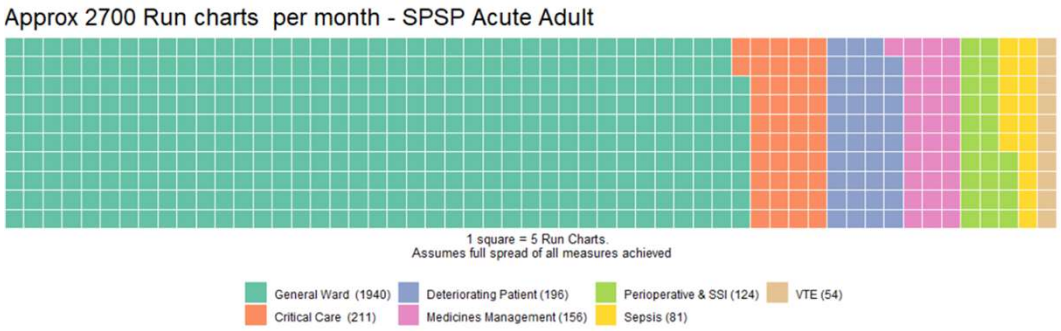


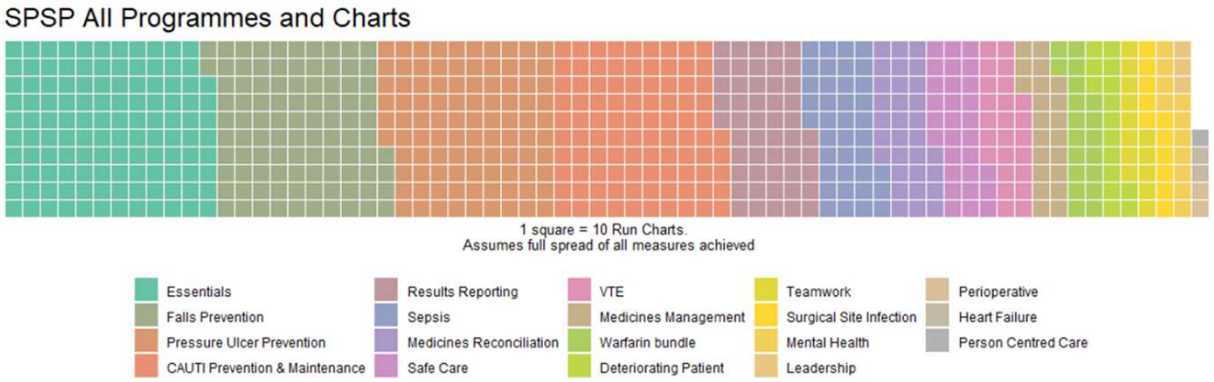
Median value

Extended median for analysis using run chart rules

A ‘run’ is 1 or more consecutive data points on the same side of the median

Scale of the challenge





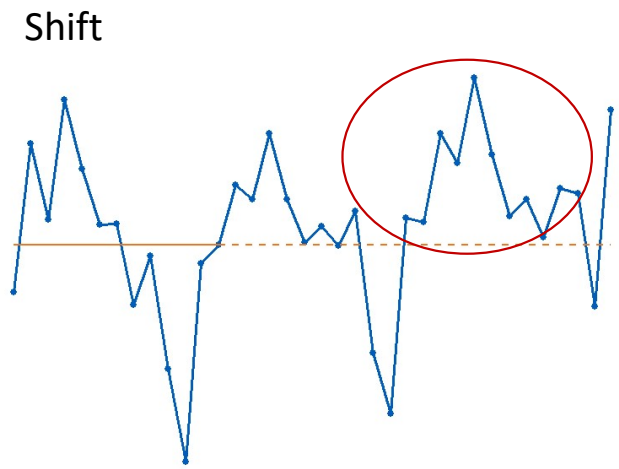
Care Bundles to Support

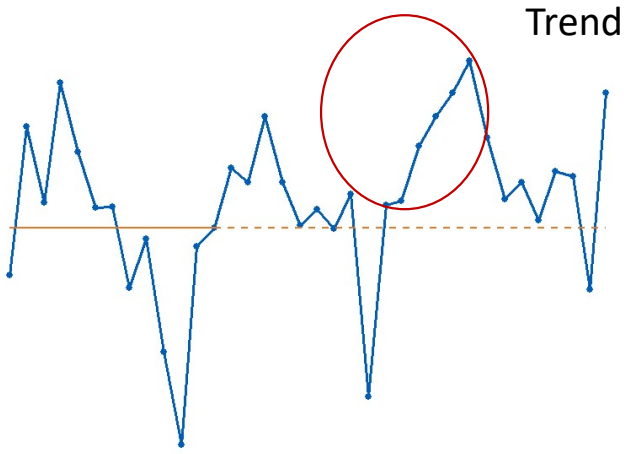
Peripheral Vascular Catheter (PVC) Bundle (HPS)

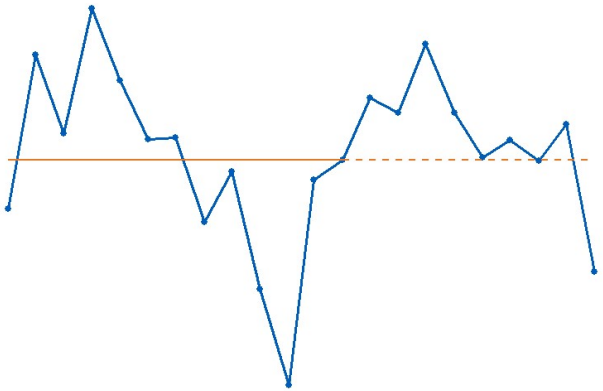
- Check to ensure the PVC *in situ* are *still required*
- Remove PVCs where there is *extravasation or inflammation*
- Check PVC *dressings are intact*
- Consider removal of PVS *in situ longer than 72 hours*
- Perform hand hygiene *before and after* all PVS procedures

- Care bundles require 1 chart for each element, plus one for overall compliance
- This bundle requires 6 charts

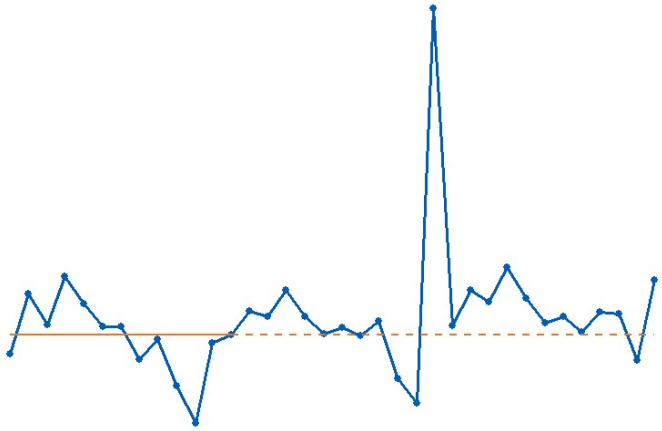
Run Chart Rules







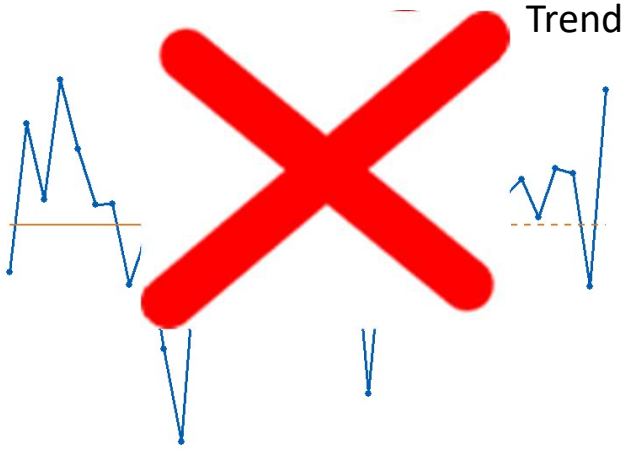
Too few /too many runs



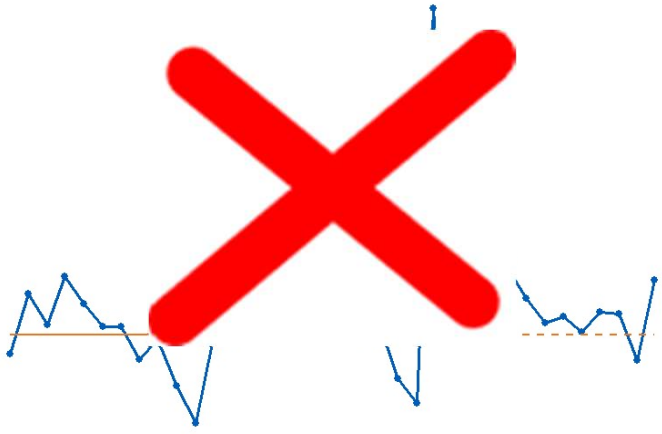
Astronomical data point

- Shift – 6 or more consecutive points on one side of the median
- Trend 5 or more consecutively increasing or decreasing points (regardless of the median)
- Tables to determine too few / too many runs

Run chart rules - critiques



Too few / too many runs



Astronomical data point

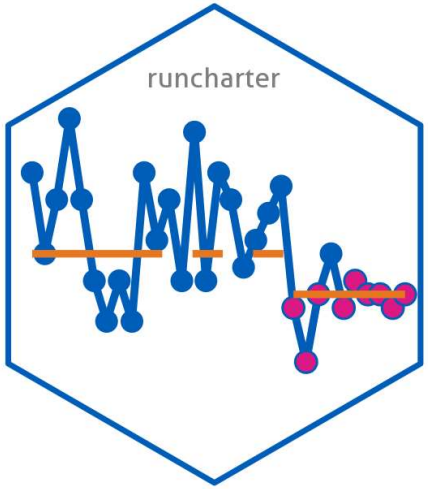
[“trend rule .. virtually useless”](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0113825)

Astronomical data point - vague

Too many runs will never be a signal of improvement

qicharts2 package detects signals based on modified ‘too few runs’ rule : https://github.com/anhoej/qicharts2

runcharter



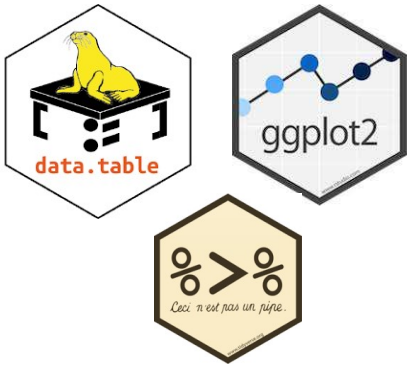
Calculates baseline medians

Finds sustained runs of improvement in desired direction

Calculates new median when a run occurs

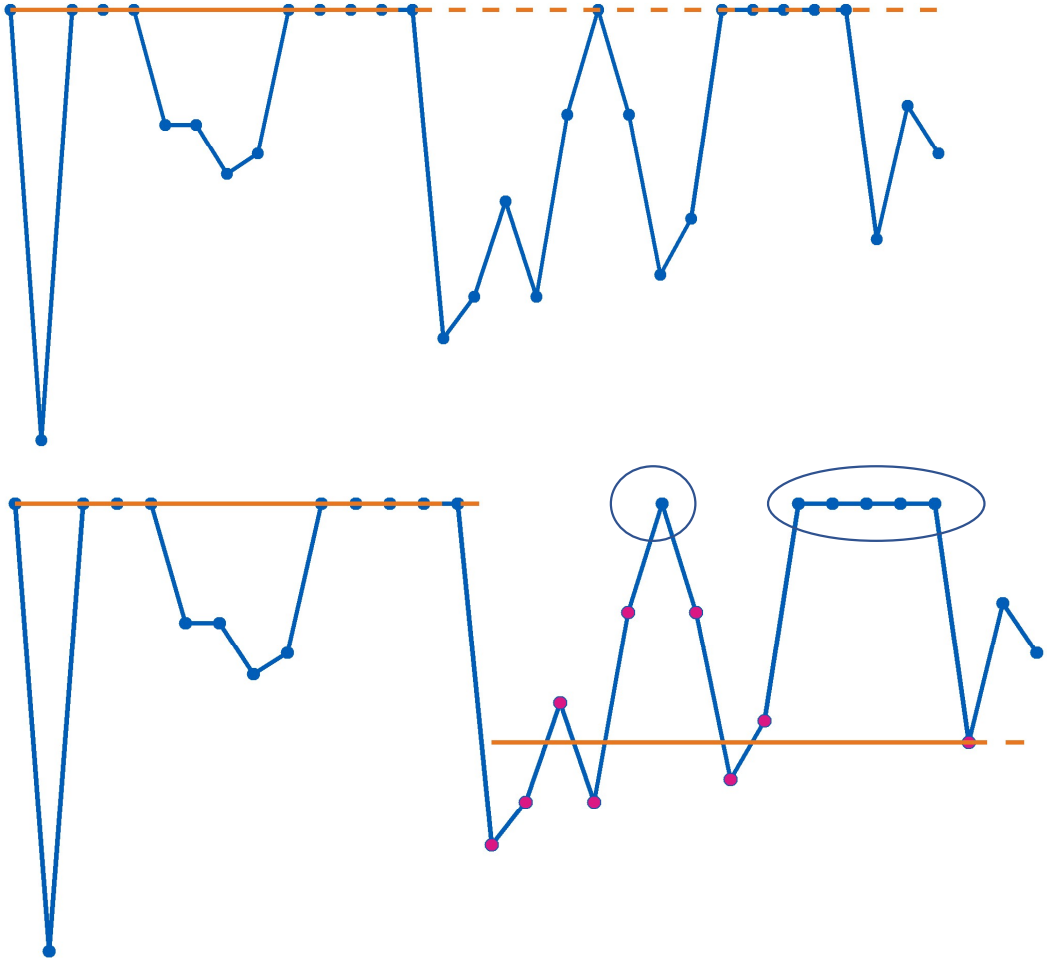
Finds additional runs based on new median

Returns plot & data



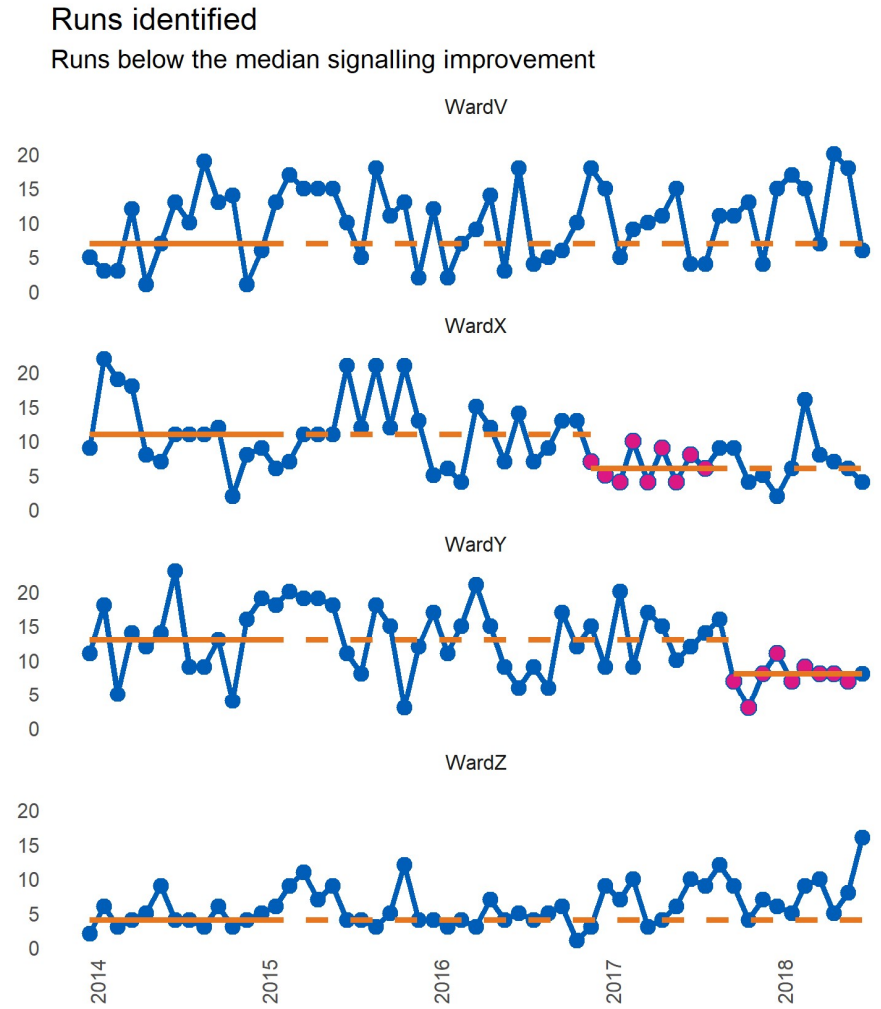
https://github.com/johnmackintosh/runcharter

Points on median do not make or break a run



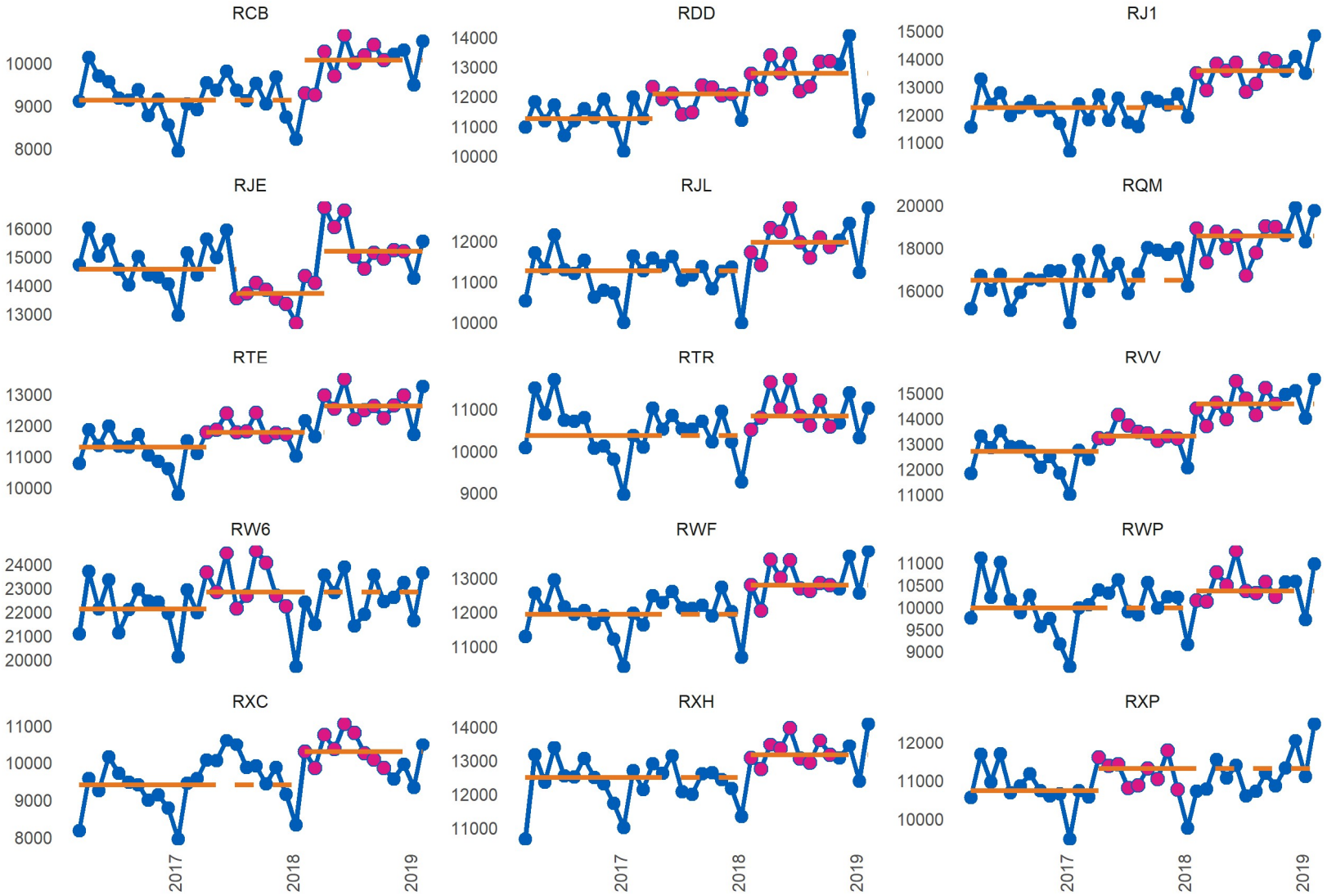
```r
library(runcharter)

runcharter(signals,
           direction = "below",
           datecol = "date",
           grpvar = "grp",
           yval = "y",
          chart_title = " Runs identified",
          chart_subtitle = "Runs below the median signalling improvement")
```

```
#> $sustained
#>      grp median start_date   end_date  extend_to  run_type
#> 1: WardV      7 2014-01-01 2015-01-01 2018-07-01  baseline
#> 2: WardX     11 2014-01-01 2015-01-01 2016-12-01  baseline
#> 3: WardY     13 2014-01-01 2015-01-01 2017-10-01  baseline
#> 4: WardZ      4 2014-01-01 2015-01-01 2018-07-01  baseline
#> 5: WardX      6 2016-12-01 2017-08-01 2018-07-01 sustained
#> 6: WardY      8 2017-10-01 2018-06-01 2018-07-01 sustained
```



```
runcharter(df,
  med_rows = 13,
  runlength = 9,
  direction = c("above", "below", "both"),
  datecol = NULL,
  grpvar = NULL,
  yval = NULL,
  facet_cols = NULL,
  facet_scales = "fixed",
  chart_title = NULL,
  chart_subtitle = NULL,
  chart_caption = NULL,
  chart_breaks = NULL,
  line_colr = "#005EB8",
  line_size = 1.1,
  point_colr = "#005EB8",
  point_size = 2.5,
  median_colr = "#E87722",
  median_line_size = 1.05,
  highlight_fill = "#DB1884",
  highlight_point_size = 2.7,
  ...
)
```

- Pipe friendly
- Length of initial median
- Length of run
- Direction – above, below or both
- Number of facets
- Fixed or free scale y axis
- Date column axis breaks
- Chart appearance – line, centre line, point and highlight size and colours


RCB
10000
9000
8000
RDD
14000
13000
12000
11000
10000
RJ1
15000
14000
13000
12000
11000
RJE
16000
15000
14000
13000
RJL
12000
11000
10000
RQM
20000
18000
16000
RTE
13000
12000
11000
10000
RTR
11000
10000
9000
RVV
15000
14000
13000
12000
11000
RW6
24000
23000
22000
21000
20000
RWF
13000
12000
11000
RWP
11000
10500
10000
9500
9000
RXC
11000
10000
9000
8000
RXH
14000
13000
12000
11000
RXP
12000
11000
10000
2017
2018
2019

Why data.table?

Some things are wrong – using data.table is not one of them

- Grouping is straightforward
- Run length analysis
- Advanced joins
- Less code, less dependencies, less to go wrong
- Very helpful error messages – proposes solutions

Help others & increase my data.table skills

Understanding DT joins and working with dates / times

How many patients were in the hospital at 10 AM yesterday?
How many were in during each 15 minute spell between 2pm and 6pm?
How many were in during the last week, by hour?

This package aims to make answering these questions easier and quicker.

No SQL? No problem!

If you have time in, time out, a unique patient identifier, and optionally, a grouping variable to track moves between departments, this package will tell you how many patients were 'IN' at any time, at whatever granularity you need.

https://github.com/johnmackintosh/patientcounter

```
> beds
   bed patient          start_time            end_time
1    A       1 2020-01-01 09:34:00 2020-01-01 10:34:00
2    A       2 2020-01-01 10:55:00 2020-01-01 11:15:24
3    A       3 2020-01-01 11:34:00 2020-01-02 17:34:00
4    A       4 2020-01-01 18:00:00 2020-01-03 00:00:00
5    B       5 2020-01-01 09:45:00 2020-01-01 14:45:00
6    B       6 2020-01-01 16:13:00 2020-01-01 21:27:24
7    B       7 2020-01-01 21:41:48 2020-01-01 22:56:12
8    B       8 2020-01-01 23:13:00 2020-01-02 00:43:00
9    C       9 2020-01-01 10:05:00 2020-01-01 10:35:00
10   D      10 2020-01-01 10:30:00                 <NA>
```

```r
interval_census(beds,
identifier = 'patient',
admit = 'start_time',
discharge = 'end_time',
group_var = 'bed',
time_unit = '1 hour',
results = "patient",
uniques = TRUE)
```

```
    bed patient          start_time            end_time  interval_beginning        interval_end  base_date base_hour
 1:   A       1 2020-01-01 09:34:00 2020-01-01 10:34:00 2020-01-01 09:00:00 2020-01-01 10:00:00 2020-01-01         9
 2:   B       5 2020-01-01 09:45:00 2020-01-01 14:45:00 2020-01-01 09:00:00 2020-01-01 10:00:00 2020-01-01         9
 3:   A       1 2020-01-01 09:34:00 2020-01-01 10:34:00 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10
 4:   B       5 2020-01-01 09:45:00 2020-01-01 14:45:00 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10
 5:   C       9 2020-01-01 10:05:00 2020-01-01 10:35:00 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10
---
116:   A       4 2020-01-01 18:00:00 2020-01-03 00:00:00 2020-01-02 21:00:00 2020-01-02 22:00:00 2020-01-02        21
117:   D      10 2020-01-01 10:30:00 2020-01-03 00:00:00 2020-01-02 22:00:00 2020-01-02 23:00:00 2020-01-02        22
118:   A       4 2020-01-01 18:00:00 2020-01-03 00:00:00 2020-01-02 22:00:00 2020-01-02 23:00:00 2020-01-02        22
119:   D      10 2020-01-01 10:30:00 2020-01-03 00:00:00 2020-01-02 23:00:00 2020-01-03 00:00:00 2020-01-02        23
120:   A       4 2020-01-01 18:00:00 2020-01-03 00:00:00 2020-01-02 23:00:00 2020-01-03 00:00:00 2020-01-02        23
```

```r
setDT(beds)

pat_d <- beds[bed =='D',]

interval_census(pat_d,
identifier = 'patient',
admit = 'start_time',
discharge = 'end_time',
group_var = 'bed',
time_unit = '1 hour',
results = "patient",
uniques = TRUE)
```

	bed	patient	start_time	end_time	interval_beginning	interval_end	base_date	base_hour
1:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-01-01 10:00:00	2020-01-01 11:00:00	2020-01-01	10
2:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-01-01 11:00:00	2020-01-01 12:00:00	2020-01-01	11
3:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-01-01 12:00:00	2020-01-01 13:00:00	2020-01-01	12
4:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-01-01 13:00:00	2020-01-01 14:00:00	2020-01-01	13
5:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-01-01 14:00:00	2020-01-01 15:00:00	2020-01-01	14

7473:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-11-07 18:00:00	2020-11-07 19:00:00	2020-11-07	18
7474:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-11-07 19:00:00	2020-11-07 20:00:00	2020-11-07	19
7475:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-11-07 20:00:00	2020-11-07 21:00:00	2020-11-07	20
7476:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-11-07 21:00:00	2020-11-07 22:00:00	2020-11-07	21
7477:	D	10	2020-01-01 10:30:00	2020-11-07 23:00:00	2020-11-07 22:00:00	2020-11-07 23:00:00	2020-11-07	22

```r
> system.time(interval_census(pat_d,
+ identifier = 'patient',
+ admit = 'start_time',
+ discharge = 'end_time',
+ group_var = 'bed',
+ time_unit = '1 hour',
+ results = "patient",
+ uniques = TRUE))
   user  system elapsed
   0.02    0.00    0.02
```

```r
grouped <- interval_census(beds,
                identifier = 'patient',
                admit = 'start_time',
                discharge = 'end_time',
                group_var= 'bed',
                time_unit =  '1 hour',
                results = 'group',
                uniques = FALSE)
```

```
> head(grouped,20)
    bed  interval_beginning        interval_end  base_date base_hour N
 1:   A 2020-01-01 09:00:00 2020-01-01 10:00:00 2020-01-01         9 1
 2:   B 2020-01-01 09:00:00 2020-01-01 10:00:00 2020-01-01         9 1
 3:   A 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10 2
 4:   B 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10 1
 5:   C 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10 1
 6:   D 2020-01-01 10:00:00 2020-01-01 11:00:00 2020-01-01        10 1
 7:   B 2020-01-01 11:00:00 2020-01-01 12:00:00 2020-01-01        11 1
 8:   A 2020-01-01 11:00:00 2020-01-01 12:00:00 2020-01-01        11 2
 9:   D 2020-01-01 11:00:00 2020-01-01 12:00:00 2020-01-01        11 1
10:   B 2020-01-01 12:00:00 2020-01-01 13:00:00 2020-01-01        12 1
11:   D 2020-01-01 12:00:00 2020-01-01 13:00:00 2020-01-01        12 1
12:   A 2020-01-01 12:00:00 2020-01-01 13:00:00 2020-01-01        12 1
13:   B 2020-01-01 13:00:00 2020-01-01 14:00:00 2020-01-01        13 1
14:   D 2020-01-01 13:00:00 2020-01-01 14:00:00 2020-01-01        13 1
15:   A 2020-01-01 13:00:00 2020-01-01 14:00:00 2020-01-01        13 1
16:   B 2020-01-01 14:00:00 2020-01-01 15:00:00 2020-01-01        14 1
17:   D 2020-01-01 14:00:00 2020-01-01 15:00:00 2020-01-01        14 1
18:   A 2020-01-01 14:00:00 2020-01-01 15:00:00 2020-01-01        14 1
19:   D 2020-01-01 15:00:00 2020-01-01 16:00:00 2020-01-01        15 1
20:   A 2020-01-01 15:00:00 2020-01-01 16:00:00 2020-01-01        15 1
> 
```

```r
overall <- interval_census(beds,
identifier = 'patient',
admit = 'start_time',
discharge = 'end_time',
group_var = 'bed',
time_unit = '1 hour',
results = "total",
uniques = TRUE)
```

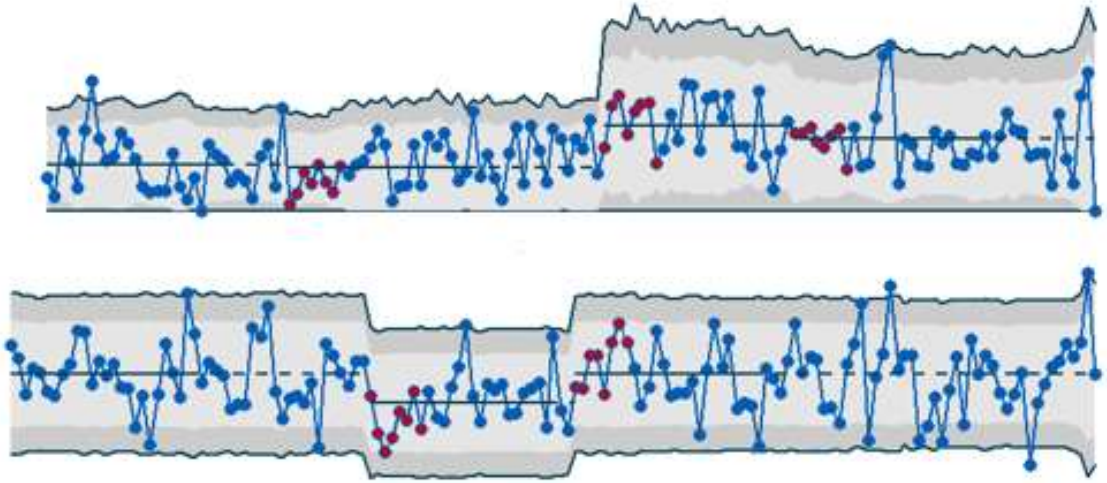
	interval_beginning	interval_end	base_date	base_hour	N
1:	2020-01-01 09:00:00	2020-01-01 10:00:00	2020-01-01	9	2
2:	2020-01-01 10:00:00	2020-01-01 11:00:00	2020-01-01	10	5
3:	2020-01-01 11:00:00	2020-01-01 12:00:00	2020-01-01	11	4
4:	2020-01-01 12:00:00	2020-01-01 13:00:00	2020-01-01	12	3
5:	2020-01-01 13:00:00	2020-01-01 14:00:00	2020-01-01	13	3
6:	2020-01-01 14:00:00	2020-01-01 15:00:00	2020-01-01	14	3
7:	2020-01-01 15:00:00	2020-01-01 16:00:00	2020-01-01	15	2
8:	2020-01-01 16:00:00	2020-01-01 17:00:00	2020-01-01	16	3
9:	2020-01-01 17:00:00	2020-01-01 18:00:00	2020-01-01	17	3
10:	2020-01-01 18:00:00	2020-01-01 19:00:00	2020-01-01	18	4
11:	2020-01-01 19:00:00	2020-01-01 20:00:00	2020-01-01	19	4
12:	2020-01-01 20:00:00	2020-01-01 21:00:00	2020-01-01	20	4
13:	2020-01-01 21:00:00	2020-01-01 22:00:00	2020-01-01	21	5
14:	2020-01-01 22:00:00	2020-01-01 23:00:00	2020-01-01	22	4
15:	2020-01-01 23:00:00	2020-01-02 00:00:00	2020-01-01	23	4
16:	2020-01-02 00:00:00	2020-01-02 01:00:00	2020-01-02	0	4
17:	2020-01-02 01:00:00	2020-01-02 02:00:00	2020-01-02	1	3
18:	2020-01-02 02:00:00	2020-01-02 03:00:00	2020-01-02	2	3
19:	2020-01-02 03:00:00	2020-01-02 04:00:00	2020-01-02	3	3
20:	2020-01-02 04:00:00	2020-01-02 05:00:00	2020-01-02	4	3
21:	2020-01-02 05:00:00	2020-01-02 06:00:00	2020-01-02	5	3
22:	2020-01-02 06:00:00	2020-01-02 07:00:00	2020-01-02	6	3
23:	2020-01-02 07:00:00	2020-01-02 08:00:00	2020-01-02	7	3
24:	2020-01-02 08:00:00	2020-01-02 09:00:00	2020-01-02	8	3
25:	2020-01-02 09:00:00	2020-01-02 10:00:00	2020-01-02	9	3
26:	2020-01-02 10:00:00	2020-01-02 11:00:00	2020-01-02	10	3
27:	2020-01-02 11:00:00	2020-01-02 12:00:00	2020-01-02	11	3
28:	2020-01-02 12:00:00	2020-01-02 13:00:00	2020-01-02	12	3
29:	2020-01-02 13:00:00	2020-01-02 14:00:00	2020-01-02	13	3
30:	2020-01-02 14:00:00	2020-01-02 15:00:00	2020-01-02	14	3
31:	2020-01-02 15:00:00	2020-01-02 16:00:00	2020-01-02	15	3
32:	2020-01-02 16:00:00	2020-01-02 17:00:00	2020-01-02	16	3

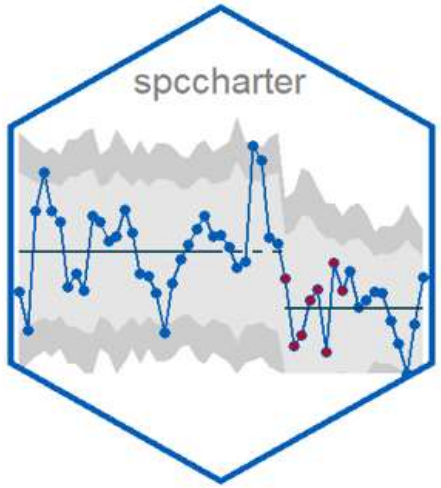
`time_unit`	Character string to denote time intervals to count by e.g. "1 hour", "15 mins".
`time_adjust_period`	Optional argument which allows the user to obtain a snapshot at a specific time of day by making slight adjustments to the specified interval. Possible values are "start_sec","start_min","end_sec", or "end_min". For example, you may specify hourly intervals, but adjust these to 1 minute past the hour with "start_min", or several seconds before the end with "end_sec".
`time_adjust_value`	Optional. An integer to adjust the startor end of each period in minutes or seconds, depending on the chosen time_adjust_period (if specified).

```r
overall <- interval_census(beds,
                           identifier = 'patient',
                           admit = 'start_time',
                           discharge = 'end_time',
                           group_var = 'bed',
                           time_unit = '1 hour',
                           time_adjust_period = 'end_min',
                           time_adjust_value = 59,
                           results = "total",
                           uniques = TRUE)
```

	interval_beginning	interval_end	base_date	base_hour	N
1:	2020-01-01 09:00:00	2020-01-01 09:01:00	2020-01-01	9	2
2:	2020-01-01 10:00:00	2020-01-01 10:01:00	2020-01-01	10	5
3:	2020-01-01 11:00:00	2020-01-01 11:01:00	2020-01-01	11	4
4:	2020-01-01 12:00:00	2020-01-01 12:01:00	2020-01-01	12	3
5:	2020-01-01 13:00:00	2020-01-01 13:01:00	2020-01-01	13	3
6:	2020-01-01 14:00:00	2020-01-01 14:01:00	2020-01-01	14	3
7:	2020-01-01 15:00:00	2020-01-01 15:01:00	2020-01-01	15	2
8:	2020-01-01 16:00:00	2020-01-01 16:01:00	2020-01-01	16	3
9:	2020-01-01 17:00:00	2020-01-01 17:01:00	2020-01-01	17	3
10:	2020-01-01 18:00:00	2020-01-01 18:01:00	2020-01-01	18	4
11:	2020-01-01 19:00:00	2020-01-01 19:01:00	2020-01-01	19	4
12:	2020-01-01 20:00:00	2020-01-01 20:01:00	2020-01-01	20	4
13:	2020-01-01 21:00:00	2020-01-01 21:01:00	2020-01-01	21	5
14:	2020-01-01 22:00:00	2020-01-01 22:01:00	2020-01-01	22	4
15:	2020-01-01 23:00:00	2020-01-01 23:01:00	2020-01-01	23	4
16:	2020-01-02 00:00:00	2020-01-02 00:01:00	2020-01-02	0	4
17:	2020-01-02 01:00:00	2020-01-02 01:01:00	2020-01-02	1	3
18:	2020-01-02 02:00:00	2020-01-02 02:01:00	2020-01-02	2	3
19:	2020-01-02 03:00:00	2020-01-02 03:01:00	2020-01-02	3	3
20:	2020-01-02 04:00:00	2020-01-02 04:01:00	2020-01-02	4	3

spccharter



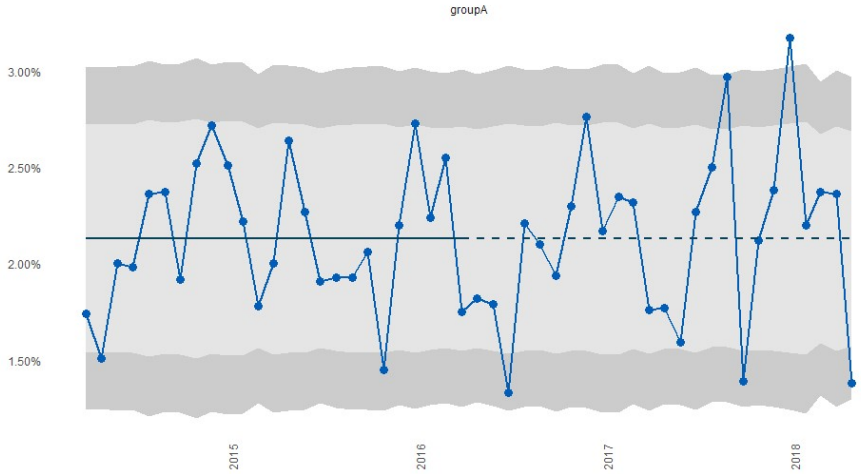


https://github.com/johnmackintosh/spccharter

“runcharter, but for SPC charts”

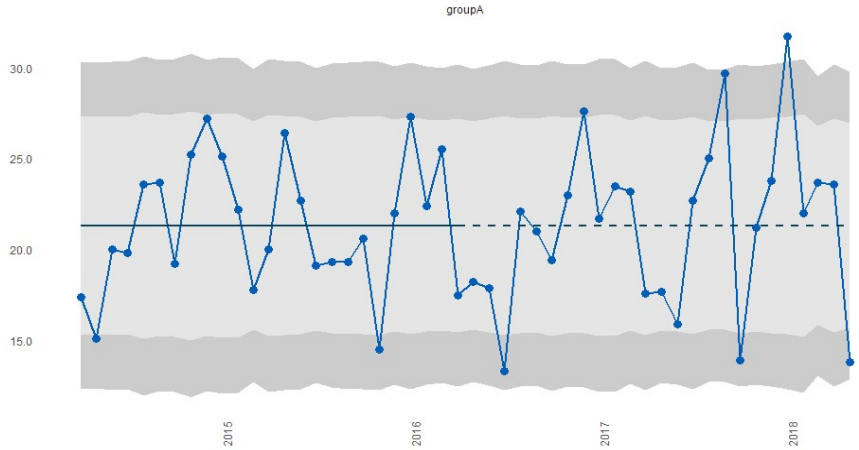
```r
# p chart
spccharter(test2,
           numerator = defects,
           denominator = possible,
           datecol = report_month,
           by = testgroup,
           plot_type = 'p',
           direction = 'both')
```

P chart - Percentages



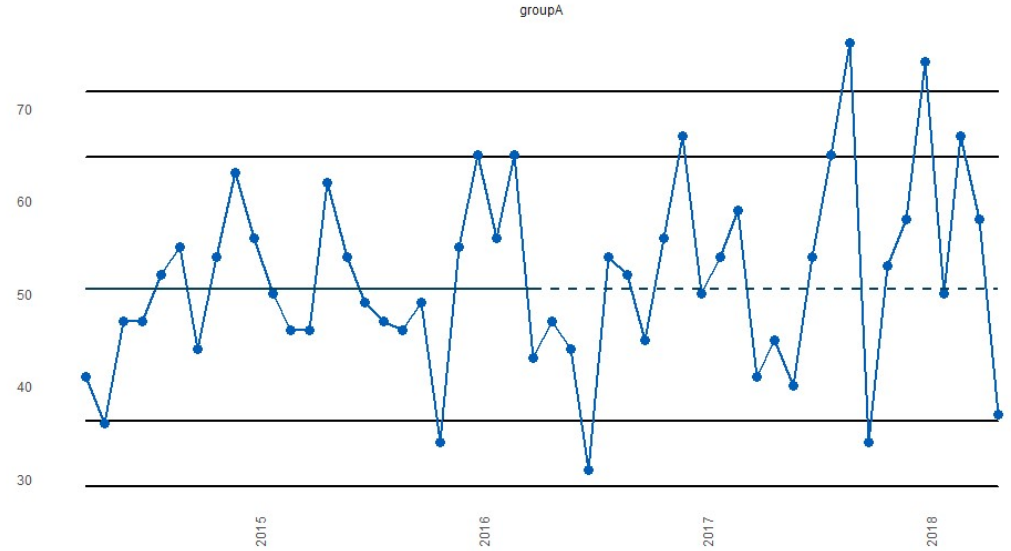

```r
# u chart
spccharter(test2,
           numerator = defects,
           denominator = possible,
           datecol = report_month,
           by = testgroup,
           plot_type = 'u',
           direction = 'both',
           multiplier = 1000)
```

U chart - Rates



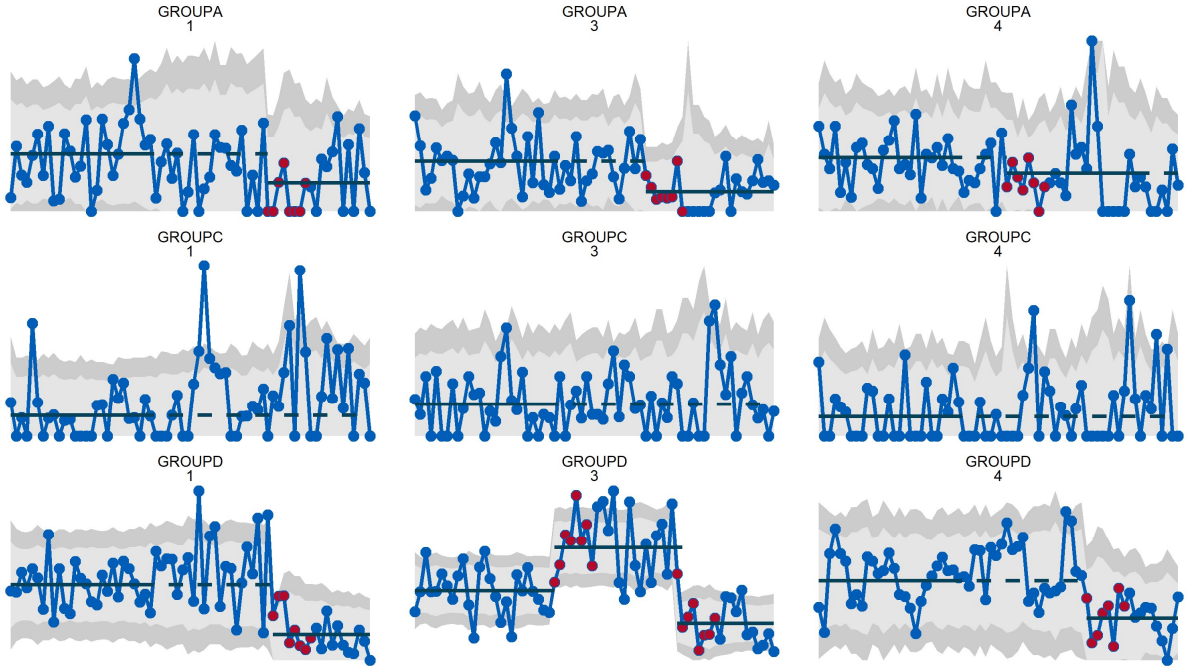
C chart - counts

```
#  c chart
spccharter(test2,
           numerator = defects,
           denominator = possible, # ignored if plot_type = 'c'
           datecol = report_month,
           by = testgroup,
           plot_type = 'c',
           direction = 'both',
           cl_fill = 'white',
           wl_fill = 'white',
           cl_colr = 'black',
           wl_colr = 'black')
```



spccharter – small multiples

```
spccharter(test_df1,
           numerator = fail,
           denominator = possible,
           datecol = report_date,
           by = c('group','subgroup'),
           plot_type = 'p',
           direction = 'both',
           facet_scales = 'free_y')
```



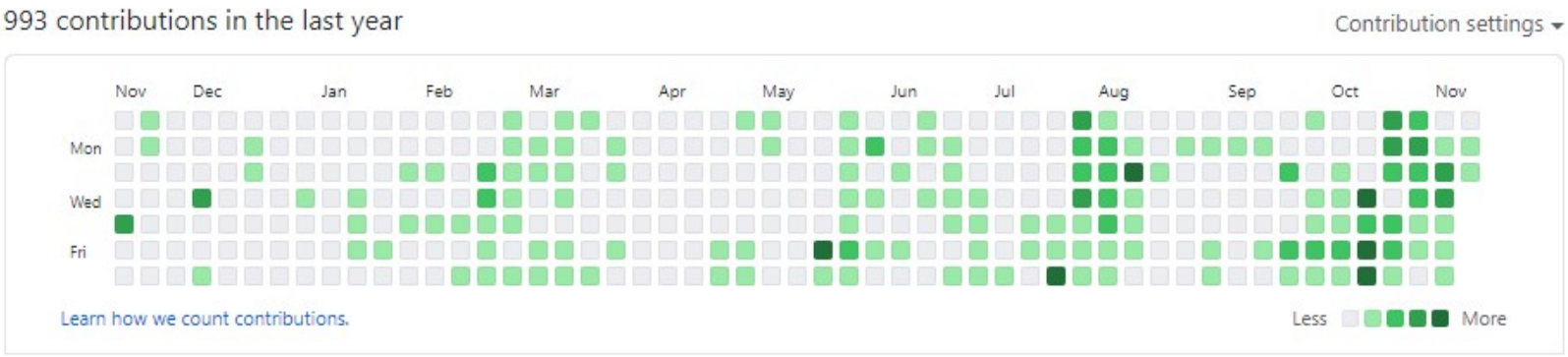
```
# spccharter
# single unquote group variable

by = group

# or 2 variables as character string
by = c('group','subgroup')


# runcharter
# only accepts one quoted group variable
# (for now)

grpvar = 'Ward'


# outputs

# spccharter

# plot only
outputs = 'plot'

# data only
outputs = 'data'

# a list with data and plot
outputs = 'both'

# runcharter
# returns a list containing data and plot
# currently no 'outputs' option to amend this
```

Main differences

Item	runcharter	spccharter
variables	quoted	unquoted (NSE)
grouping variable	1, quoted	1 unquoted or 2 as character vector
Y axis value	Supplied by user in desired format	Calculated from supplied numerator and denominator, with supplied number of decimal places
Outputs	Plot and data	Plot only, data only, plot + data

What's next?

- runcharter : accept NSE and optional additional group variable
- patientcounter : better site / docs & hex sticker
- spccharter : Prime P & U charts (for large denominators) & improve testing



Thanks!

Mohammed, Anastasiia and all @ NHS-R

Tom Jemmett and Chris Mainey for NHSRdatasets / general awesomeness

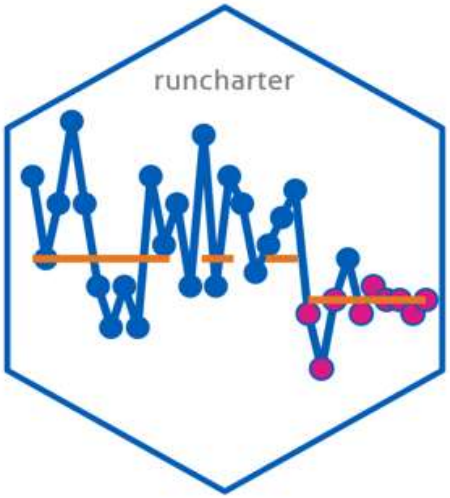
Chris Beeley - Chris Beeley

Andrew Hill – very helpful feedback for runcharter V1

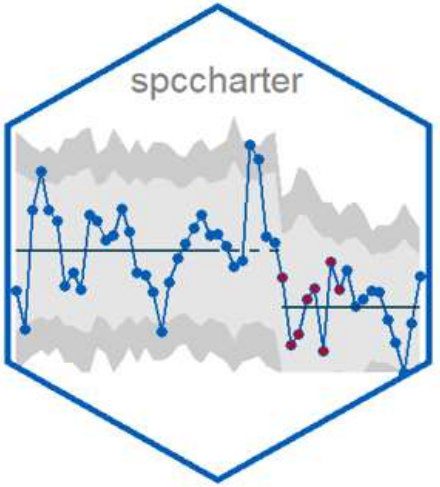
Royal Free Hospital for all the interest, support and feedback

Zoe Turner - support , making NHS-R a great place to hang out, general awesomeness

Those who have liked / starred/ used my packages so far


runcharter


patientcounter


spccharter