Chapter 1

Charting, Plotting and Curving using Charter

Charting, plotting and curving data points is the basis of many kinds of analyses. To enable this, Roassal provides Charter: a builder dedicated to draw charts. We will first present the three different kinds of charts that can be built using Charter, then discuss fine-tuning the chart, and end with a word on the implementation.

1.1 Scatterplot

Simple chart

Charter offers flexible way to draw scatterplots. Consider the following example, to be evaluated in a playground:

```
methods := Collection withAllSubclasses flatCollect: #methods.

b := RTCharterBuilder new.
b extent: 200 @ 200.
b shape circle color: (Color red alpha: 0.3).
b points: methods.
b x: #numberOfLinesOfCode.
b y: [ :m | m getSource size ].

b axisX.
b axisY.
b build.
```

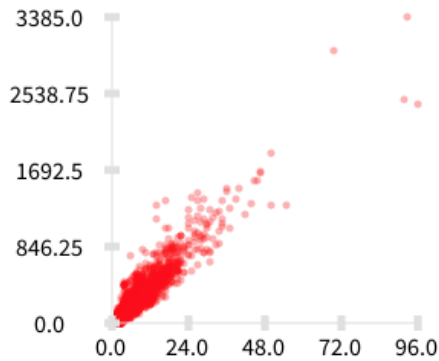

Figure 1.1: Scatterplot built by hand.

Figure 1.1 illustrates the result of the script given above. In the code, the variable methods contains all the methods of the Pharo Collection class hierarchy. Note that approximately 3,500 methods are defined in the hierarchy.

The use of extent: defines the size in pixels of the part in which charts are drawn. Each data point will be represented as a circle, using the default size of 5 pixels, and a translucent red. The collection of data points is specified using points:.

The x-value and y-value have to be computed for each data point. The method objects contained in the methods collection answer to the message numberOfLinesOfCode, returning the number of lines of code for the method definition. The block [:m | m getSource size] returns the number of characters that define the method.

Lastly, the two axes are then added, each having 4 ticks by default.

Figure 1.1 reveals an obvious correlation between the number of lines of code and the number of characters of the method. Deviation from the diagonal indicates methods with either very long or very short lines of code.

Multiple charts

Charter supports different data set to be simultaneously displayed. Consider the following example:

```
methods := Collection withAllSubclasses flatCollect: #methods.
trachelMethods := TRObject withAllSubclasses flatCollect: #methods.

b := RTCharterBuilder new.
b extent: 200 @ 200.
```

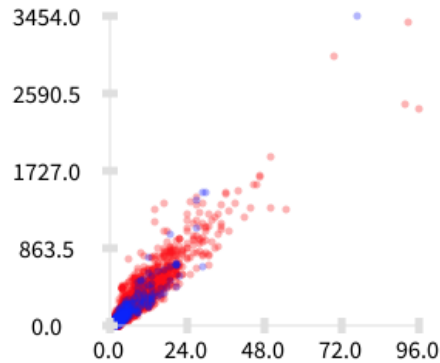

Figure 1.2: Two data set in the same chart.

```
"Data set 1"
b shape circle color: (Color red alpha: 0.3).
b points: methods.

"Data set 2"
b shape circle color: (Color blue alpha: 0.3).
b points: trachelMethods.

"The same metrics for all data points"
b allX: #numberOfLinesOfCode.
b allY: [ :m | m getSource size ].

b axisX.
b axisY.
b build.
```

Figure 1.2 shows two colored data sets. The first data set, the methods of the Collection class hierarchy (methods), is colored in red. The second data set, all the methods defined in Trachel (trachelMethods), are colored in blue.

Axis configuration

Properly defining the X and Y axes is a complex task. A great deal of parameters usually have to be taken into account. By default, both axes have four ticks, and each tick is labelled with a numerical value with a precision of one decimal point. Charter however offers a number of configuration options for the configuration of Axis. For example, in our situation, both axes have to be labelled with integer values: the number of lines and the size of method def-

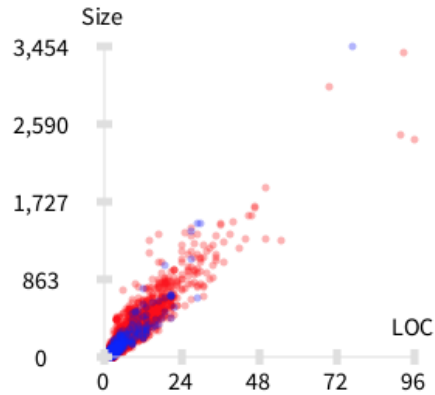

Figure 1.3: Axis configuration.

initions are integer values. Consider the example seen previously, for which only integer values with a comma as thousand separators:

```
...
"Axis configuration"
b axisConfiguration withThousandsSeparator.
b axisXTitled: 'LOC'.
b axisYTitled: 'Size'.
b build.
```

Book Todo:
there should be a table here with a list of the configuration options and an indication on how to find them in the code. (JF)

1.2 Curve

A curve is obtained by connecting data points with a line. Curves may be obtained intensionally by giving a function or extensionally by providing the data points.

Function

Consider the following script:

```
b := RTCharterBuilder new.
b extent: 200 @ 200.

b shape rectangle size: 0.
b points: (0 to: 3.1415 * 5 by: 0.01).
b y: #sin.
b x: #yourself.
b connectDotColor: (Color red alpha: 0.4).

b shape rectangle size: 0.
b points: (0 to: 3.1415 * 5 by: 0.01).
b y: #cos.
b x: #yourself.
b connectDotColor: (Color blue alpha: 0.4).

b axisX.
b axisY.

b build.
```

Book Todo:
Why b y: [:v | v sin]. and not b y: #sin. ? and idem for cos. ? I have changed it to be more consistent internally. (JF)

Each data point is defined as a rectangle of size 0. Which means that each datapoint will not be drawn. Instead, they are linked to each other via a connecting line, as defined by the new keyword here connectDotColor:. A Line is drawn between the dots in the order they were provided to points:.

Stacking

Data points may be stacked, meaning that the index of the point in the collection is its X value. This is useful in case that data points are not obtained by applying a function. Consider the following example: it shows 4 data points horizontally ordered in the same order as they were provided to points:, as seen in Figure 1.5.

```
b := RTCharterBuilder new.

b shape rectangle color: Color red.
b points: #(5 1 20 5).
b stackX.
b y: #yourself.
```

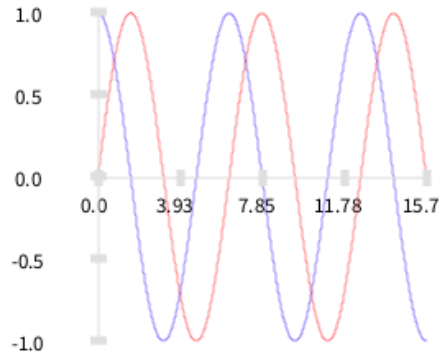

Figure 1.4: Curves defined as functions.

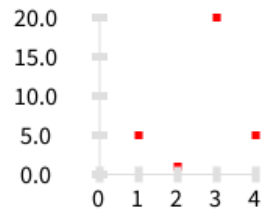

Figure 1.5: Simple stack of data points.

```
b axisY.
b axisConfiguration noDecimals.
b axisX.

b build.
```

A slightly more elaborate example is given below. Each curve is a class contained in Roassal. Each data point is a method, sorted in a reverse alphabetical order. The Y-value of a method is its size in number of lines of code.

```
classes := RTShape withAllSubclasses.

b := RTCharterBuilder new.
b extent: 400 @ 200.

normalizer := RTMultiLinearColorForIdentity new objects: classes.
classes do: [ :c |
    | data |
    data := (c methods reverseSortedAs: #numberOfLinesOfCode ).
```

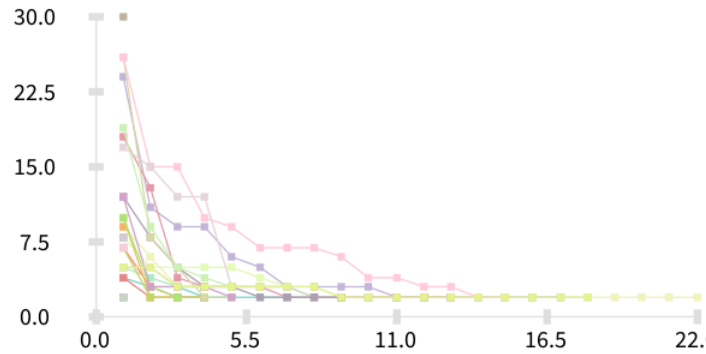

Figure 1.6: Stacking data points.

```
    b interaction popup.
    b shape rectangle color: (normalizer rtValue: c).
    b points: data.
    b connectDotColor: (normalizer rtValue: c).
].

b allY: #numberOfLinesOfCode.
b stackX.

b axisX.
b axisY.

b build.
```

Figure 1.6 shows 27 different curves, each representing a subclass of the class RTShape.

Also, a distinct color is given to each curve. This is useful to be able to differentiate classes. To achieve this, a RTMultiLinearColorForIdentity object has to be initialized with the objects that will be colored. The message objects: is used for that purpose. The expression (normalizer rtValue: c) will return a color that is specific for the argument c.

Stacking multiple curve

Book Todo:
I do not understand this section at all. What is the problem and what is the solution? (JF)

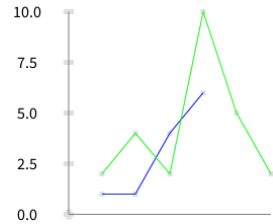

Figure 1.7: Stacking multiple curve.

Composing stacked curves requires manually adjusting the scale of the Y-axis. The scaling is typically done using minY and maxY variables. Consider two data sets #(1 1 4 6) and #(2 4 2 10 5 2) (Figure 1.7):

```
points1 := #(1 1 4 6).
points2 := #(2 4 2 10 5 2).

minY := 0.
maxY := (points1 maxValue: #yourself) max: (points2 maxValue: #yourself).

b := RTCharterBuilder new.
b extent: 200 @ 200.
b points: points1.
b connectDotColor: Color blue.
b y: #yourself min: minY max: maxY.

b points: points2.
b connectDotColor: Color green.
b y: #yourself min: minY max: maxY.
b stackX.

b axisXNoLabel; axisY.
b build
```

Charting these two sets requires one single call to stackX. The metric function for the Y-axis has to be specified using y: (simply #yourself in our example).

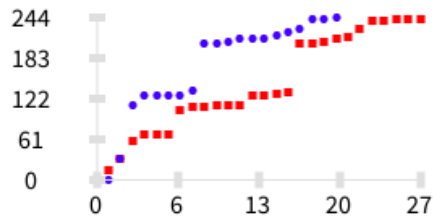

Figure 1.8: Data point aspect.

Data point aspect

Book Todo:
this section is already too big and this info is already relevant for scatterplots. Why not move it back: make the example with different colors in the Multiple charts section of scatterplot also use different shapes? (JF)

The visual aspects of data points may be customized. Consider the following example.

```
b := RTCharterBuilder new.
b extent: 200 @ 100.
b shape rectangle color: Color red.
b points: (RTShape withAllSubclasses sortedAs: #ageInDays).

b shape circle color: Color blue.
b points: (TRShape withAllSubclasses sortedAs: #ageInDays).

b stackX.
b allY: [ :c | c ageInDays ].

b axisConfiguration noDecimals.
b axisX.
b axisY.

b build.
```

Figure 1.8 represents the age of shape classes contained in Trachel (*i.e.,* subclasses of the class TRShape) and Roassal (*i.e.,* subclasses of the class RTShape). Trachel shapes are older than Roassal shapes.

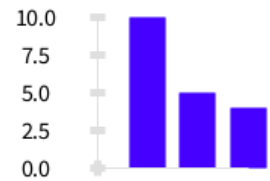

Figure 1.9: Simple bar chart.

1.3 Bar

RTCharterBuilder supports bar charts. The following example displays a simple bar chart (Figure 1.9):

```
b := RTCharterBuilder new.
b extent: 100 @ 100.
b shape rectangle size: 0.
b interaction popup.
b points: #(10 5 4).
b y: #yourself.
b stackX.
b shape rectangle color: Color blue.
b histogram.

b axisY.
b newAxisConfiguration plain.
b axisX.

b build.
```

In a bar chart, data points are typically not visible. Setting a size of 0 will make data points not apparent. The use of the interaction popup makes the bars react when the mouse is above a bar. Three numerical values are provided, and the height of a bar is given by the value itself. Bar are stacked from left to right and are colored in blue.

> **Book Todo:**
> There is too much magic here. Is the histogram method responsible for the bar-chart-nature? What's with the b shape rectangle color: Color blue. then? Do I need to do this before the histogram or can I also do it after? What are the other magic messages that make a bar chart and what is their semantics? Also, the interaction stuff should be moved to the next section. (JF)

1.4 Interaction

Interactions may be defined to get particular behavior upon user actions. Typical cases is getting a value when the mouse hovers a point or a bar.

> **Book Todo:**
> This should be clear that it's for all kinds of charts. Can you add a simple example for scatterplot and for line? (JF)

Consider the following code (Figure 1.10):

```
b := RTCharterBuilder new.
b extent: 300 @ 200.
b shape rectangle size: 0.
b points: (RTShape withAllSubclasses).
b y: [ :c | c methods size ].
b stackX.
b shape rectangle color: Color veryLightGray.

b interaction highlightColored: Color red.

b interaction popup
    named: [ :c | c name, ' methods']
    background: Color veryVeryLightGray
    group: [ :group :element |
        | s ms |
        s := RTBox new size: #numberOfLinesOfCode; color: Color red.
        ms := element model methods sortedAs: #numberOfLinesOfCode.
        group addAll: (s elementsOn: ms).
        RTGridLayout on: group ].

b histogram.

b axisY.
b newAxisConfiguration plain.
b axisX.

b build.
```

Two interactions are defined on each bar. First, the bar on which the mouse is above is red. The original bar color is restored when the mouse leaves the bar. The second interaction is a grouped popup, which has a name, a background color, and a set of method elements. The size of a method reflects the number of lines of code defining the method.

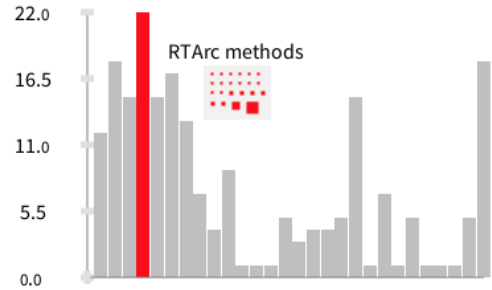

Figure 1.10: Interaction in histogram.

1.5 Distribution

1.6 Logarithmic Scale

1.7 Axis Range

1.8 Date on the axis

Dates are particular values that requires an adequate control over what is being displayed on the X-axis. This is enabled using the julianDayNumber message on a date object, converting it into a number.

Consider the following example that shows the creation of methods along time (Figure 1.11):

```
methods := RTObject withAllSubclasses flatCollect: #methods.
methods := methods reject: [ :m | m numberOfLinesOfCode > 150 ].
oldestMethod := methods minFor: #date.

b := RTCharterBuilder new.
b extent: 300 @ 200.
b shape circle size: 5; color: (Color blue alpha: 0.1).
b interaction popup.
b points: methods.
b allY: #numberOfLinesOfCode.
b allX: [ :m | m date julianDayNumber − oldestMethod date julianDayNumber ].

b axisConfiguration
   noDecimals;
   title: 'LOC'.
b axisY.
```

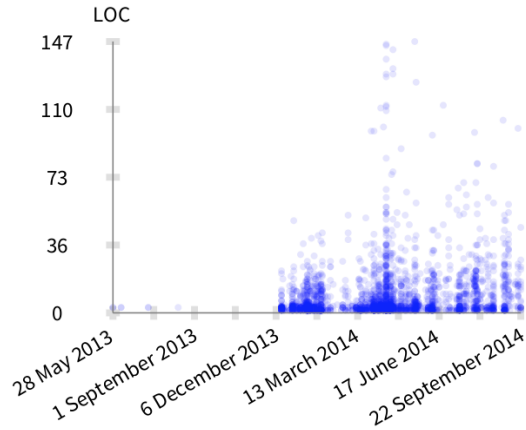

Figure 1.11: Date on the X-axis.

```
b axisConfiguration
    title: '';
    labelRotation: -30;
    numberOfTicks: 10;
    numberOfLabels: 10;
    labelConvertion: [ :v | (Date julianDayNumber: v + oldestMethod date
        julianDayNumber) ].
b axisX.

b build
```

Book Todo:
the code sais numberOfTicks: 10; numberOfLabels: 10; but this is not what's shown. (JF)

1.9 Line Shape

Book Todo:
dashed, thickness

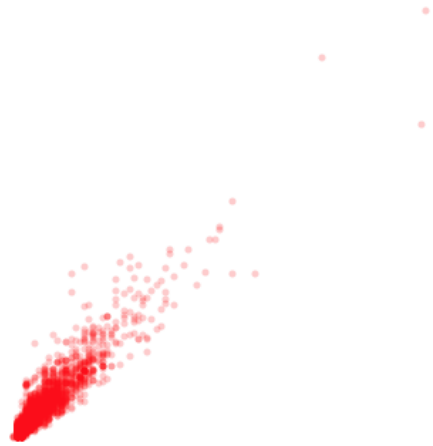

Figure 1.12: Scatterplot.

1.10 Internal of RTCharterBuilder

Drawing charts often implies normalizing elements over a well determined range of values. Typically the size of the chart. The class RTCharterBuilder is built on the principle illustrated in the following code:

```
methods := Collection withAllSubclasses flatCollect: #methods.
v := RTView new.
es := (RTEllipse new color: Color red) elementsOn: methods.

v addAll: es.
RTMetricNormalizer new
    elements: es;
    alphaColor: 0.2;
    normalizeX: #numberOfLinesOfCode min: 0 max: 300;
    normalizeY: [ :m | m getSource size ] min: 0 max: -300.
v
```

Figure 1.12 illustrates the result of the script. Each position coordinate ranges over a ordered set of values.

Axis

Each axis is described by an instance of RTAxisConfiguration.

Histogram

Rectangles are located below the data point.

Stacking or not stacking a data set

You will define an histogram as a list of values, e.g,. 4, 5, 6, 7 You will define a scatterplot as a list of points, e.g., 2 @ 3, 5 @ 6, ...

An histogram is obtained from a list of objects, and a function Y to obtain the values. A scatter plot is obtained from a list of objects, and _two_ functions X and Y to obtain the points.

When you do not need to specify an X, e.g., an histogram, then you need a RTStackedDataSet. When you need to specify an X, e.g., a scatter plot, then you need a RTDataSet.