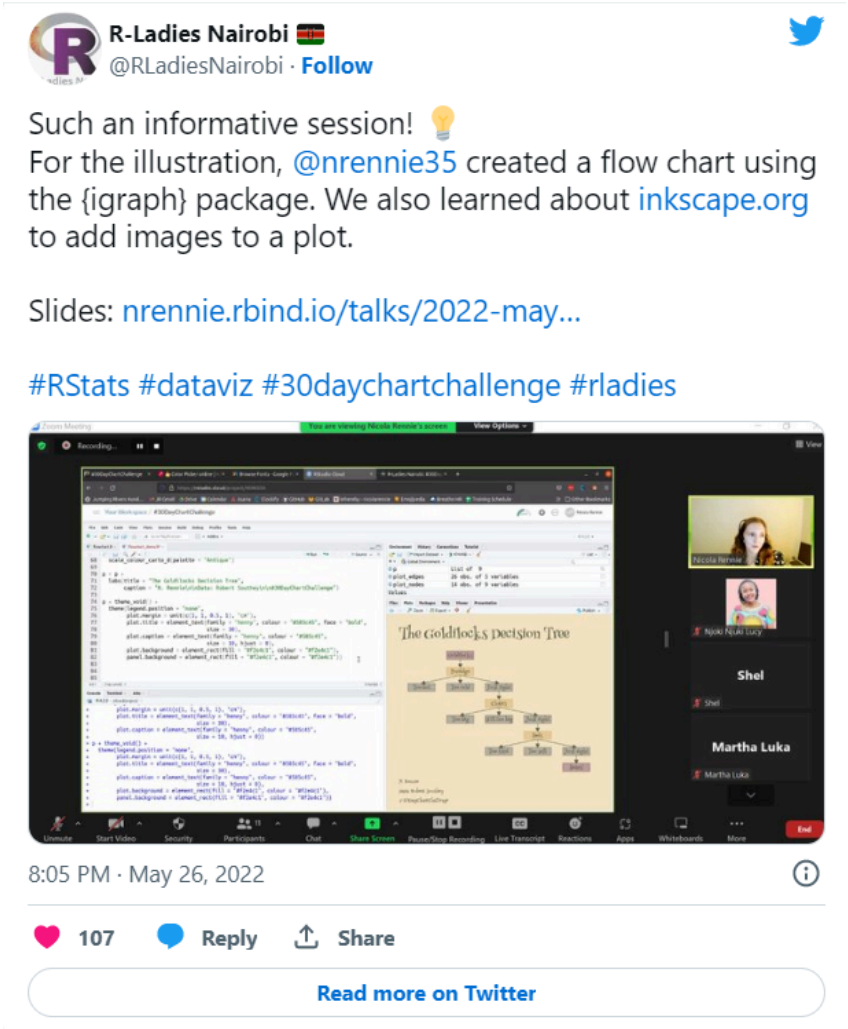
Creating flowcharts with {ggplot2}

Flowcharts can be a useful way to visualise complex processes. This tutorial blog will explain how to create one using {igraph} and {ggplot2}.

June 6, 2022

I recently gave a talk to R-Ladies Nairobi, where I discussed the #30DayChartChallenge. In the second half of my talk, I demonstrated how I created the Goldilocks Decision Tree flowchart using {igraph} and {ggplot2}. This blog post tries to capture that process in words.



Flowcharts can be a useful way to visualise complex processes. Although the example here is rather trivial and created purely for fun, nonetheless flowcharts have been a useful part of data visualistaion in my work.

Packages for making flowcharts in R

Having previously exclusively used tools like MS Visio for creating flowcharts, using R for the same thing was new to me. Before creating flowcharts from scratch using {ggplot2}, I explored a few other packages to see if they would do what I wanted.

- {grid}: for drawing simple grobs e.g., rectangles, lines
- {DiagrammeR}: interface to the DOT language
- {igraph}: package for working with graph objects
- {ggnetwork}, {ggnet2}, and {ggraph}: packages for working with and plotting network data
- {tikz}: (okay, this is actually a LaTex package for flowcharts but you *can* write LaTeX in R!)

None of these packages did *quite* what I was looking for - a programmatic way of creating highly customisable, good looking flowcharts in R. In essence, flowcharts are just rectangles, text, and arrows. And since {ggplot2} is capable of building all three of those things, so I decided to use it build a flowchart. This blog post illustrates the process of doing so.

R packages required

I used four packages in creating the flowchart (technically more, since {tidyverse} is a collection of packages!). The {showtext} package is used for fonts, and {rcartocolor} for the colour palettes. Therefore, these are not necessary packages for building a generic non-styled flowchart.

Copy

```
library(tidyverse)
library(igraph)
library(showtext)
library(rcartocolor)
```

The building blocks for flowcharts

The first step in building the flowchart was to create data frames (or tibbles) of the information I would need to construct the rectangles, text, and arrows in the chart.

Creating the initial data set

The input data required is a data frame with two columns specifying the start and end points of the arrows in my chart. I constructed it manually by writing out a tibble, but you could alternatively store this information in a .csv file, for example.

Copy

```
goldilocks <- tibble(from = c("Goldilocks",
                              "Porridge", "Porridge", "Porridge",
                              "Just right",
                              "Chairs", "Chairs", "Chairs",
                              "Just right2",
                              "Beds", "Beds", "Beds",
                              "Just right3"),
                     to = c("Porridge",
                            "Too cold", "Too hot", "Just right",
                            "Chairs",
                            "Still too big", "Too big", "Just right2",
                            "Beds",
                            "Too soft", "Too hard", "Just right3",
                            "Bears!"))
```

This is what the data should look like:

Copy

```
# A tibble: 6 × 2
  from       to
  <chr>      <chr>
1 Goldilocks Porridge
2 Porridge   Too cold
3 Porridge   Too hot
4 Porridge   Just right
```

```
8 5 Just right Chairs
9 6 Chairs     Still too big
```

One key thing to note here, is that each node in my flowchart must have a unique name. There are a couple of nodes which will have the same text labels, but the node names must be different. Hence, the variables `"Just right"`, `"Just right2"`, and `"Just right3"`.

Defining the layout

I initially toyed with the idea of writing my own code to define the layout of the nodes. However, the {igraph} package actually did what I wanted. Flowcharts are essentially tree graphs, and the `layout_as_tree()` function constructs a tree layout of an input graph.

Copy

```
1 g = graph_from_data_frame(goldilocks, directed = TRUE)
2 coords = layout_as_tree(g)
3 colnames(coords) = c("x", "y")
```

The returns a data frame of x and y coordinates for the centre points of the node locations:

Copy

```
1       x y
2 [1,]  0 7
3 [2,]  0 6
4 [3,] -1 5
5 [4,] -1 4
6 [5,] -2 3
7 [6,] -2 2
```

Adding attributes

The `coords` data will become my main data set relating to the rectangles. Currently, it contains only the x and y coordinates of the centre of the rectangle. After converting my data frame to a tibble, I add some additional information. Using the `vertex_attr()` function, I add the names of the nodes from the original `goldilocks` tibble.

I use a regex to remove the appended numbers from the names, and create the labels that will actually appear in my flowchart.

I also multiply the x-coordinates by -1. This reverse the plotting from a top-right – bottom-left direction, to become a top-left – bottom-right direction. Although I could have used something like `scale_x_reverse()` at a later stage, when I was working out how to construct the coordinates, I found it easier to think about the data without accounting for future transformations. Finally, I add a `type` variable, to classify the nodes into actions, decisions, and outcomes. I'll later colour the rectangles based on `type`.

Copy

```
1 output_df = as_tibble(coords) %>%
2   mutate(step = vertex_attr(g, "name"),
3          label = gsub("\\d+$", "", step),
4          x = x*-1,
5          type = factor(c(1, 2, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 3, 1)))
```

Copy

```
# A tibble: 6 × 5
      x     y step        type  label
  <dbl> <dbl> <chr>       <fct> <chr>
1     0     7 Goldilocks  1     Goldilocks
2     0     6 Porridge    2     Porridge
3     1     5 Just right  3     Just right
4     1     4 Chairs      2     Chairs
5     2     3 Just right2 3     Just right
6     2     2 Beds        2     Beds
```

Making the boxes

The columns in `output_df` give me the x and y coordinates of the centre of the nodes. I'm going to use `geom_rect()` from {ggplot2} to plot the rectangles, and it requires four arguments: `xmin`, `xmax`, `ymin` and `ymax` - essentially specifying the coordinates of the corners of the boxes. I use `mutate()` from {dplyr} to created new columns, specifying how far away the top, bottom, left, and right of the rectangles should be from the center. It took a little bit of trial and error to find the correct values here.

Copy

```
plot_nodes = output_df %>%
  mutate(xmin = x - 0.35,
         xmax = x + 0.35,
         ymin = y - 0.25,
         ymax = y + 0.25)
```

Now `plot_nodes` tibble looks like this:

Copy

```
# A tibble: 6 × 9
      x     y step        type  label       xmin  xmax  ymin  ymax
  <dbl> <dbl> <chr>       <fct> <chr>      <dbl> <dbl> <dbl> <dbl>
1     0     7 Goldilocks  1     Goldilocks -0.35  0.35  6.75  7.25
2     0     6 Porridge    2     Porridge   -0.35  0.35  5.75  6.25
3     1     5 Just right  3     Just right  0.65  1.35  4.75  5.25
4     1     4 Chairs      2     Chairs      0.65  1.35  3.75  4.25
5     2     3 Just right2 3     Just right  1.65  2.35  2.75  3.25
6     2     2 Beds        2     Beds        1.65  2.35  1.75  2.25
```

Making the edges

I need to adapt the original `goldilocks` tibble, to include the information on the x and y coordinates of the start and end point of the arrows. This step took a lot of experimenting before I got it right. First, I added an `id` column based on row number which later helped me keep track of which elements relate to which arrow. I use `pivot_longer()` from {tidyr} to put my data into long format - now each row relates to a single coordinate point.

The `left_join()` function from {dplyr} is then used to match up these coordinates to the rectangle the arrow will start or end at. Here, `select()` is used solely for tidying up purposes to get rid of the columns I no longer need.

The x-coordinates of my arrows will always start from the horizontal centre of the rectangle, so I can use the existing x-coordinates of the rectangles for this. The y-coordinate is a little trickier. The y-coordinate of the arrow endpoint depends if it's the "from" or the "to" part of the arrow. Arrows leaving a rectangle should leave from the bottom of the rectangle - the `"ymin"` value. Arrows arriving at a rectangle should arrive at the top of the rectangle - the `"ymax"` value. A combination of `mutate()` and `ifelse()` constructs the y-coordinates.

Copy

```
plot_edges = goldilocks %>%
  mutate(id = row_number()) %>%
  pivot_longer(cols = c("from", "to"),
               names_to = "s_e",
               values_to = "step") %>%
  left_join(plot_nodes, by = "step") %>%
  select(-c(label, type, y, xmin, xmax)) %>%
  mutate(y = ifelse(s_e == "from", ymin, ymax)) %>%
  select(-c(ymin, ymax))
```

Copy

```
# A tibble: 6 × 5
     id s_e   step           x     y
  <int> <chr> <chr>      <dbl> <dbl>
1     1 from  Goldilocks     0  6.75
2     1 to    Porridge       0  6.25
3     2 from  Porridge       0  5.75
4     2 to    Too cold       0  5.25
5     3 from  Porridge       0  5.75
6     3 to    Too hot       -1  5.25
```

Plotting a flowchart with {ggplot2}

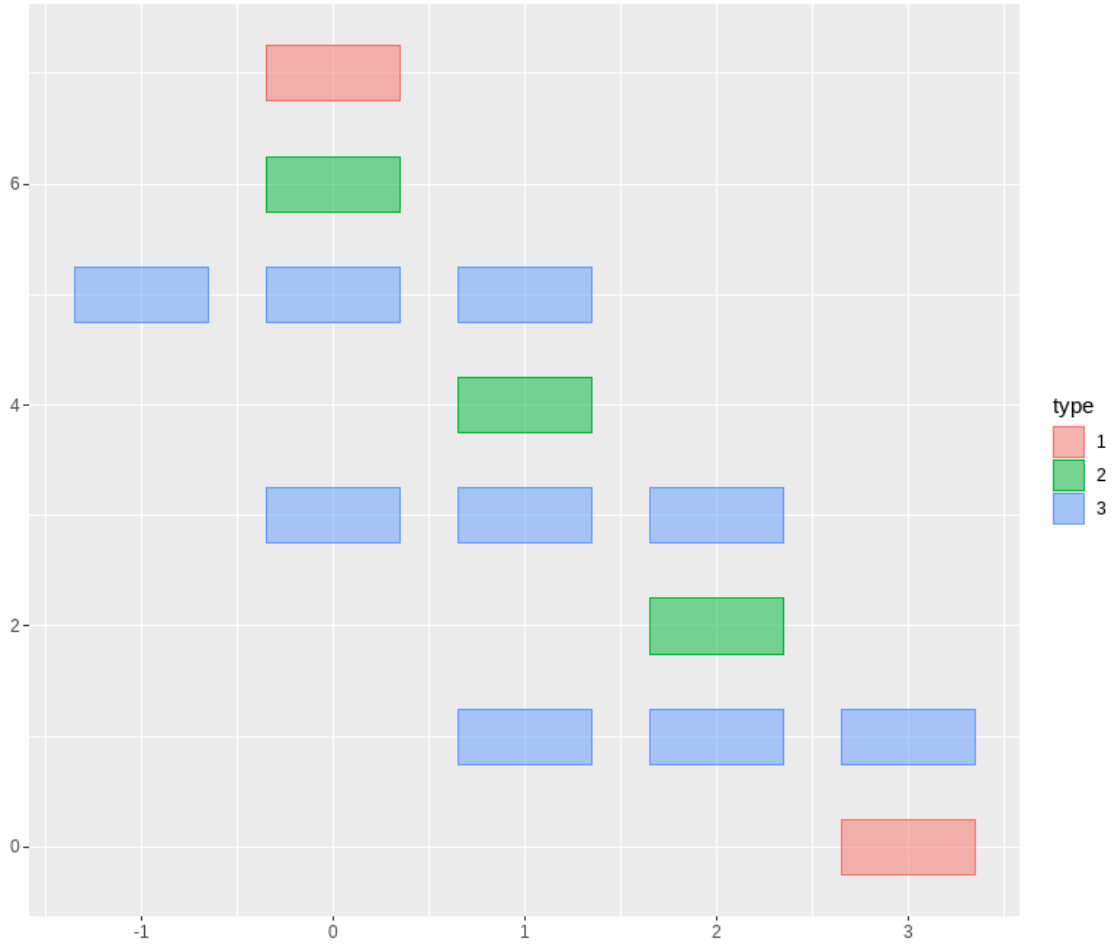
There are three main components to flowcharts: rectangles, text, and arrows. I'll add these components as different layers with {ggplot2}. First up - rectangles:

Drawing rectangles

Copy

```
p = ggplot() +
  geom_rect(data = plot_nodes,
            mapping = aes(xmin = xmin, ymin = ymin,
                          xmax = xmax, ymax = ymax,
                          fill = type, colour = type),
            alpha = 0.5)
```

I pass in the `xmin`, `xmax`, `ymin` and `ymax` values defined earlier in the `plot_nodes` tibble to `geom_rect()`, and colour the rectangles based on the `type` variable. I also make the boxes slightly transparent.



Adding labels and choosing fonts

Before I add the text labels, I need to choose what font I want to use. Although the default font would work well for simpler flowcharts, for this example I want to choose a fun font! There are a few different R packages for working with fonts (including {extrafont} and {ragg}). My preference is the {showtext} package as I've found it the easiest to use, and it works in the same way on both Linux and Windows OS. I also like the fact that it works with Google fonts. I can visually browse through these fonts at fonts.google.com, which reduces the trial and error of finding a font I like.

For this flowchart, I settled on the *Henny Penny* font from Google - it gives off fairy tale vibes to me! I load it into R using the `font_add_google()` function, giving the official name and the name I will use to refer to the font in R as arguments. Running `showtext_auto()` is an important step as it makes the loaded fonts available to R.

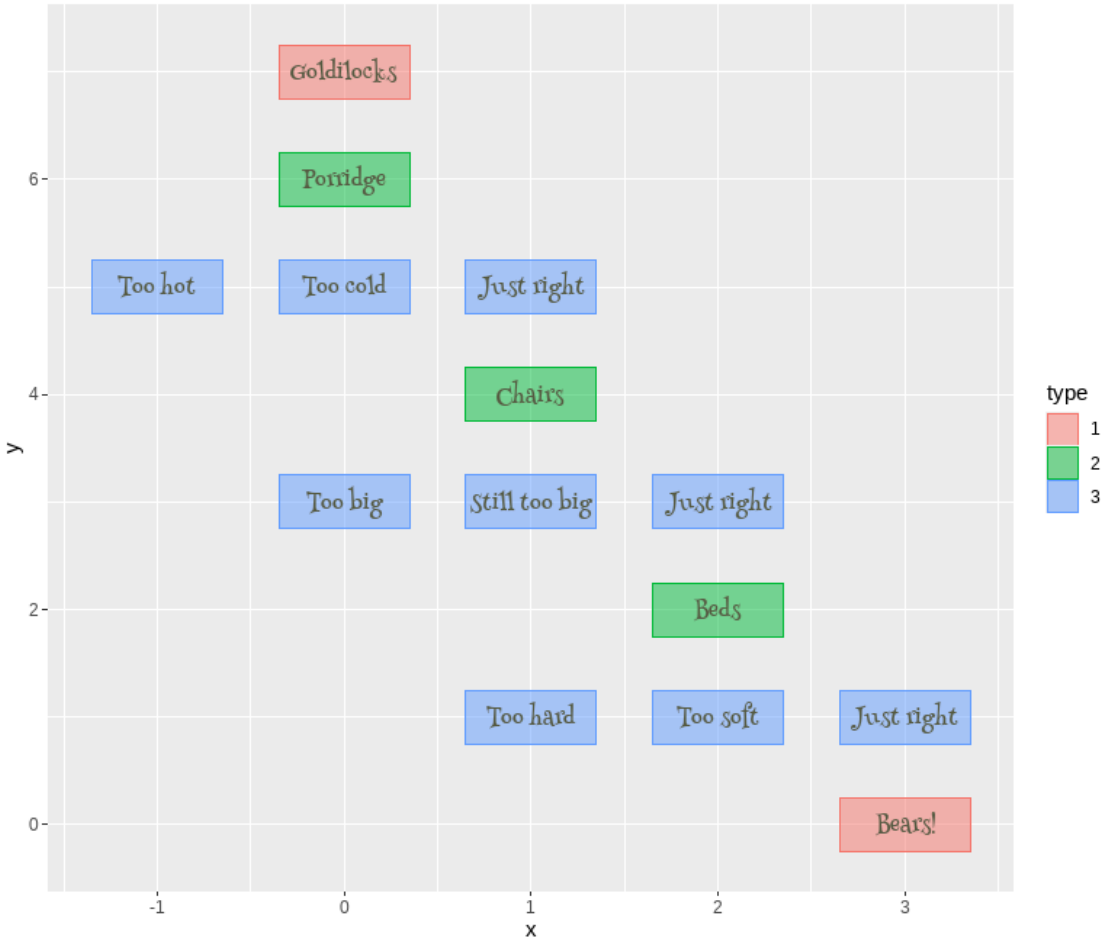
Copy

```
font_add_google(name = "Henny Penny", family = "henny")
showtext_auto()
```

I can then add text labels to my flowchart with `geom_text()`, specifying the font and colour.

Copy

```
p = p +
  geom_text(data = plot_nodes,
            mapping = aes(x = x, y = y, label = label),
            family = "henny",
            color = "#585c45")
```



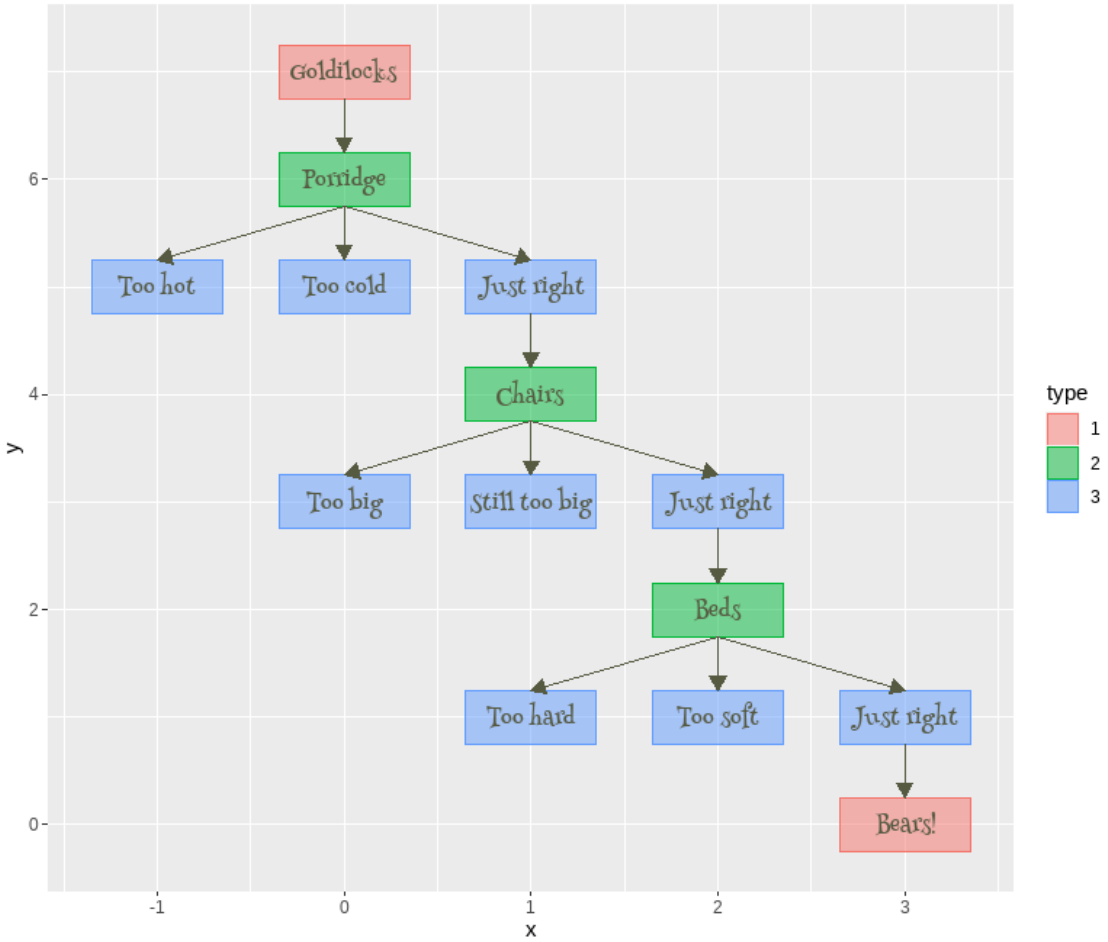
Drawing the arrows

The arrows are drawn using `geom_path()`. It's important that I use `geom_path()` instead of `geom_line()` since I don't want {ggplot2} to re-order the arrows based on their x-coordinates. I also specify the `group` variable to ensure that each arrow is only drawn between two points, instead of all connected to each other.

The arrowheads are specified using the `arrow` argument and `arrow()` function. Again, it took a little bit of trial and error to find the right size of arrowhead.

Copy

```
p = p +
  geom_path(data = plot_edges,
            mapping = aes(x = x, y = y, group = id),
            colour = "#585c45",
            arrow = arrow(length = unit(0.3, "cm"), type = "closed"))
```

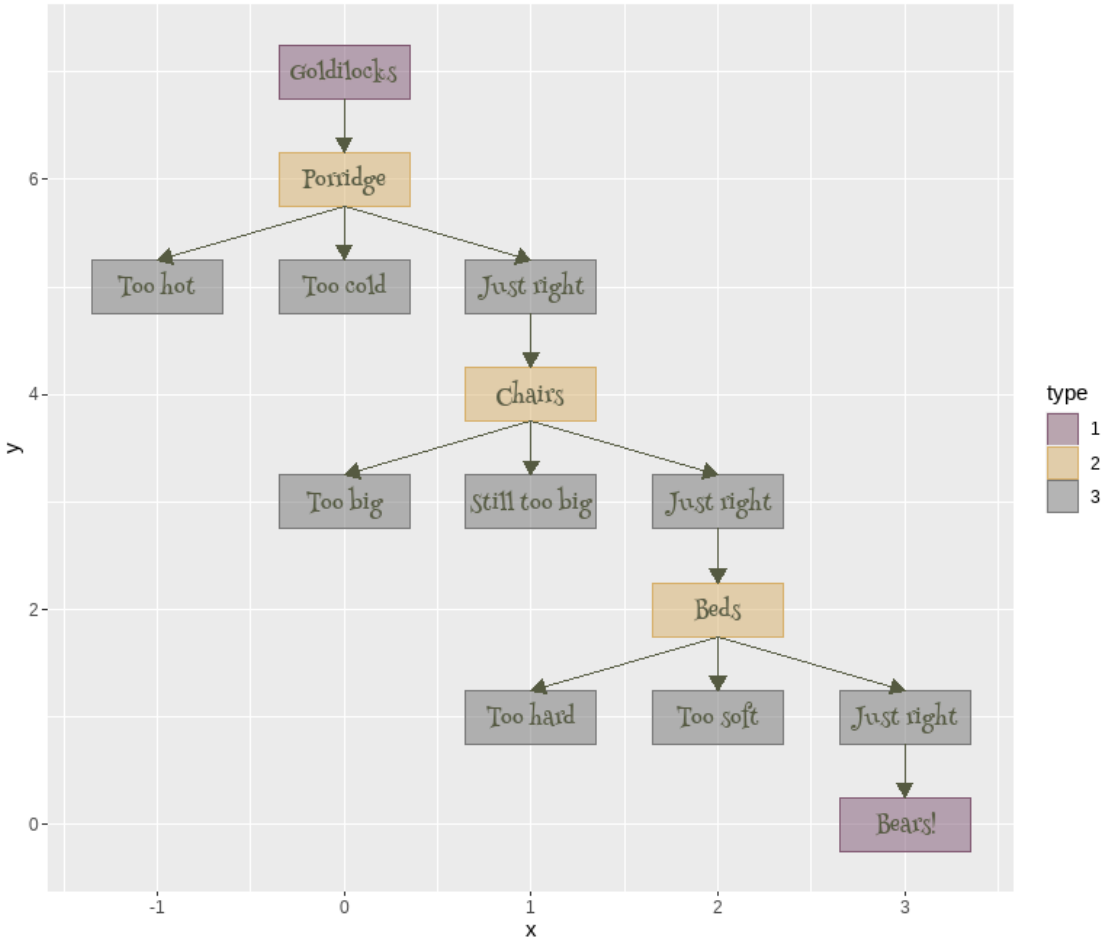


Colour schemes

We now have the basic flowchart constructed and it's time to start styling it - this is my favourite part! Instead of the default colour palette used by {ggplot2}, I'm going to use a palette from the {rcartocolor} package called `"Antique"`. You can browse the palettes in this package at jakubnowosad.com/rcartocolor. I change both the outline and inner colour of the rectangles to have the same colours.

Copy

```
p = p +
  scale_fill_carto_d(palette = "Antique") +
  scale_colour_carto_d(palette = "Antique")
```

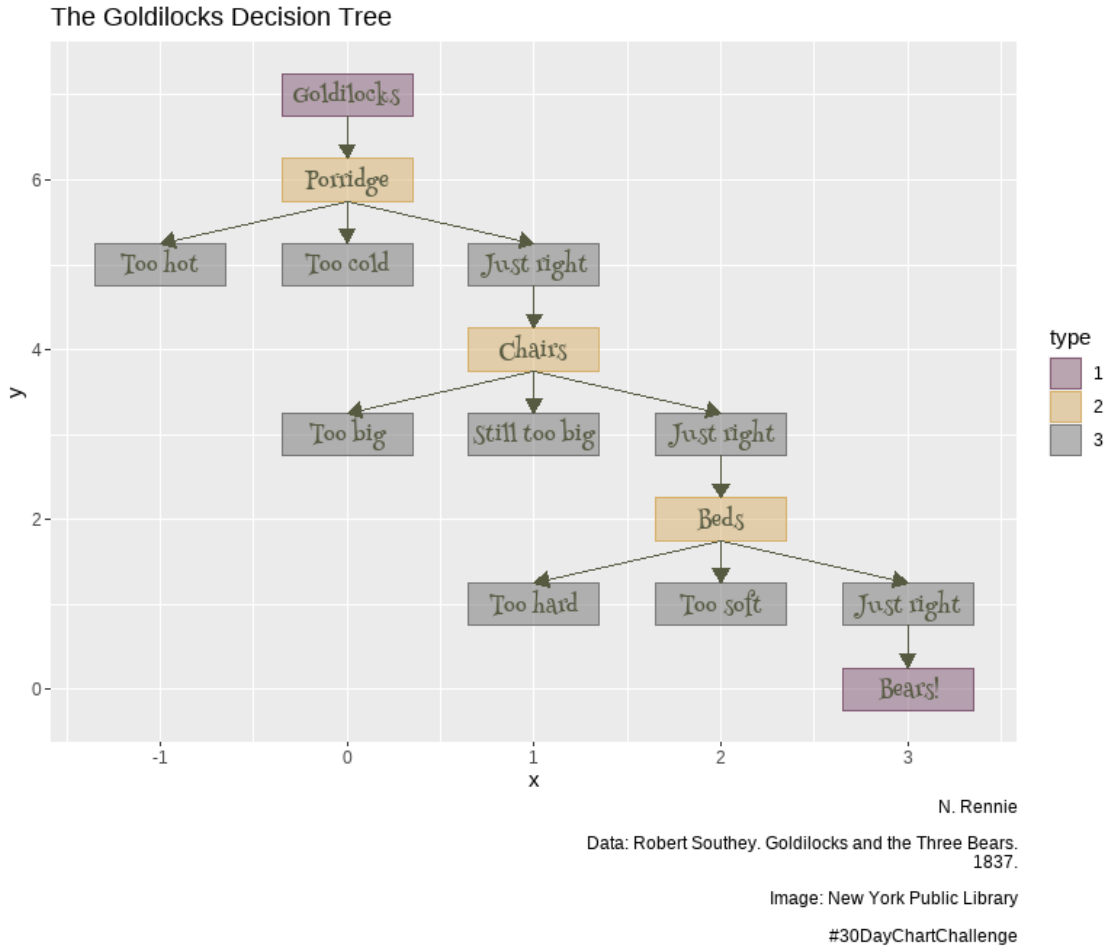


Adding text

The next step is adding a title and caption using the `labs()` function. In the caption, I usually include my name, the data source, and (in this case) the source of the image I will add later.

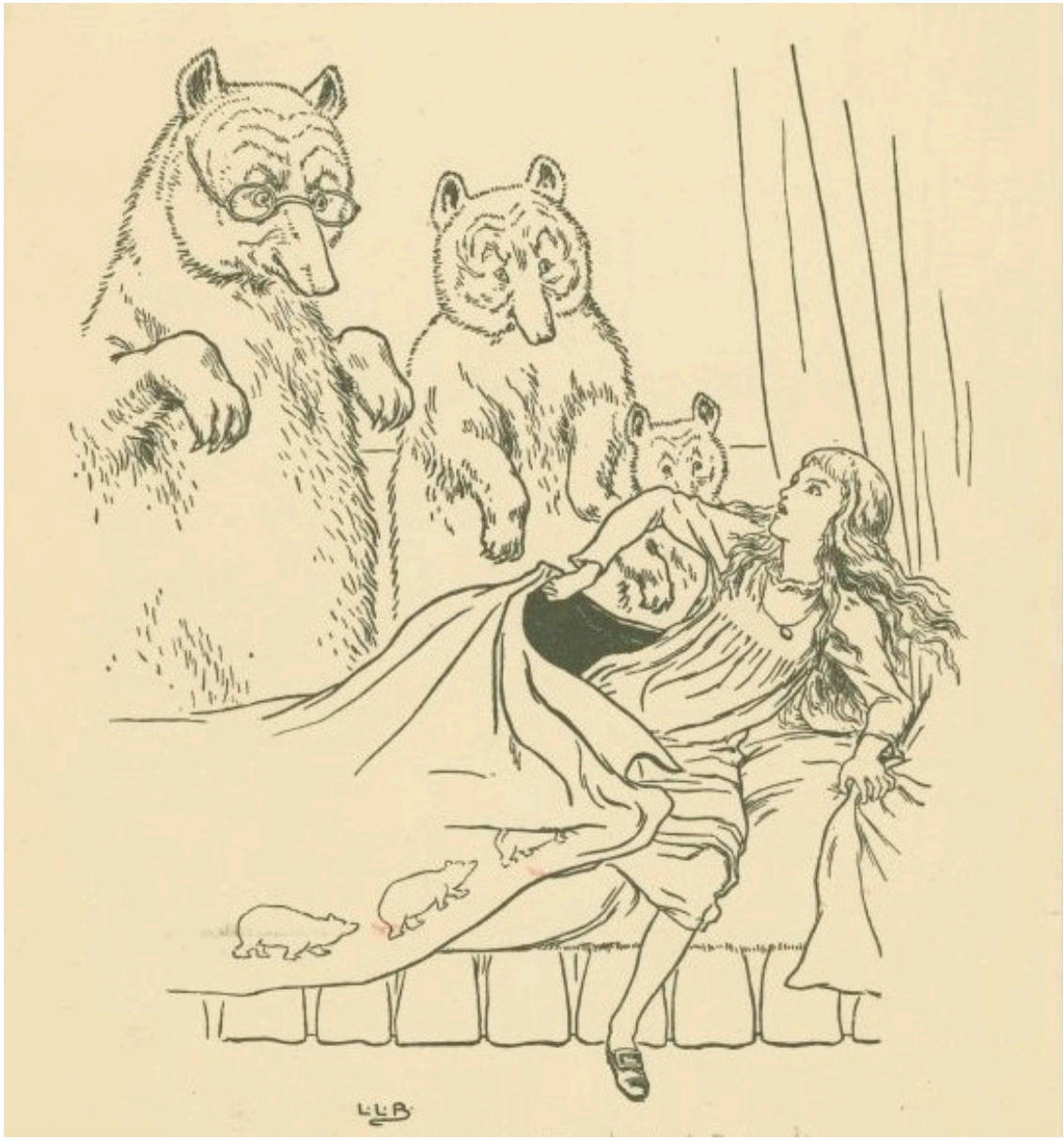
Copy

```
p = p +
  labs(title = "The Goldilocks Decision Tree",
       caption = "N. Rennie\n\nData: Robert Southey. Goldilocks and the Three Bears.
       1837.\n\nImage: New York Public Library\n\n#30DayChartChallenge")
```



Editing themes

The final aesthetic changes are done using the `theme()` function - this lets you control the look of all the non-data elements of your plot. The first thing I change is the background colour. I chose the background colour based on the image I want to overlay later. For reference, I browsed for images with a creative commons licence and found this one from the New York Public Library.



I used imagecolorpicker.com to extract the hex code of the background colour of the image and then set the plot background to be the same. There are two elements to changing the background colour: `panel.background` and `plot.background`. The `panel.background` argument changes the colour of the area behind the plotted data (grey by default). The `plot.background` argument changes the colour of the area around the plot (white by default).

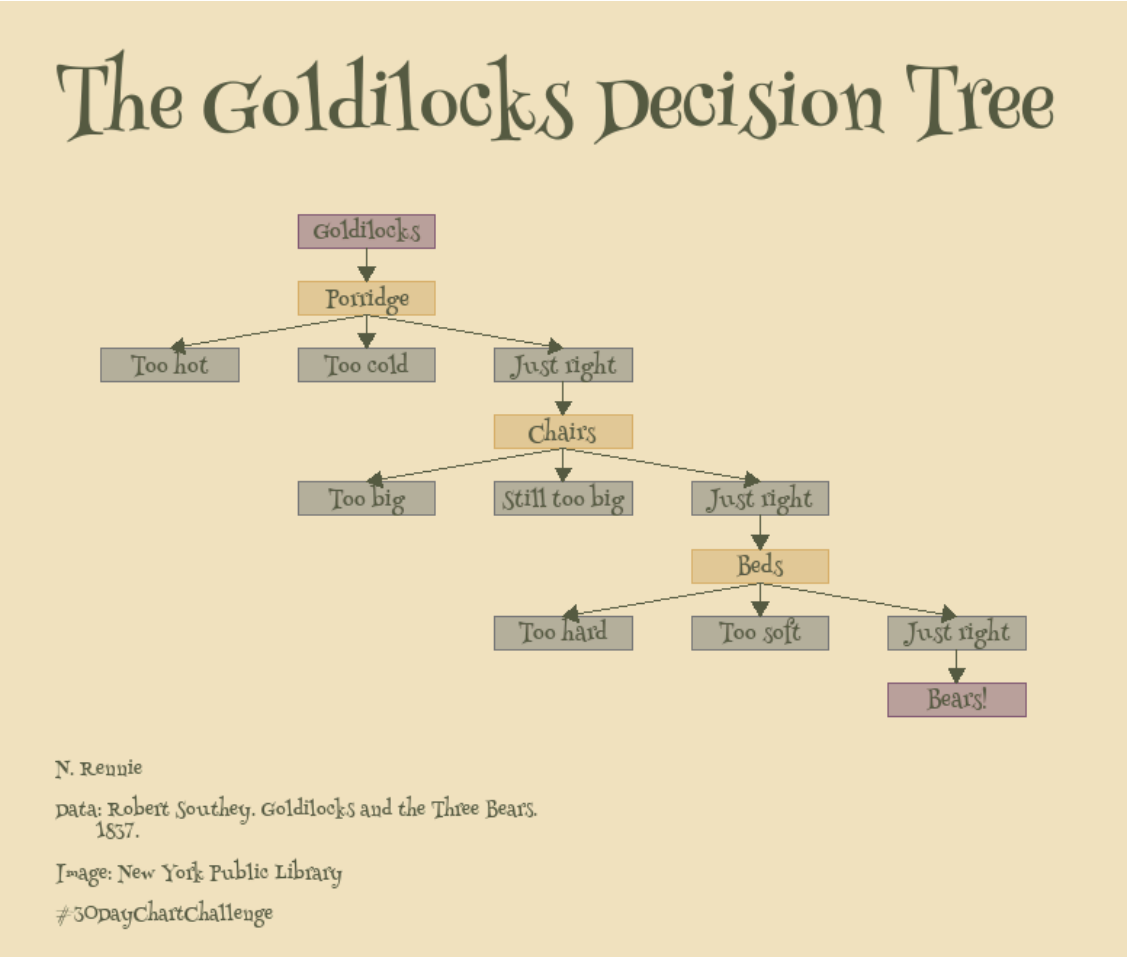
Here, I also use `theme_void()` to remove all axis labels, titles, ticks, and gridlines. Unfortunately, this also removes the space around the edge of the plot, so I add it back in using the `plot.margin` argument. I also remove the legend in this example by setting `legend.position = "none"`.

Finally, I style the title and caption text, and use the same font as I did for the rectangle labels.

Copy

```
p = p +
  theme_void() +
  theme(plot.margin = unit(c(1, 1, 0.5, 1), "cm"),
        legend.position = "none",
        plot.background = element_rect(colour = "#f2e4c1", fill = "#f2e4c1"),
        panel.background = element_rect(colour = "#f2e4c1", fill = "#f2e4c1"),
        plot.title = element_text(family = "henny", hjust = 0, face = "bold",
                                  size = 40, color = "#585c45",
                                  margin = margin(t = 10, r = 0, b = 10, l = 0)),
        plot.caption = element_text(family = "henny", hjust = 0,
```

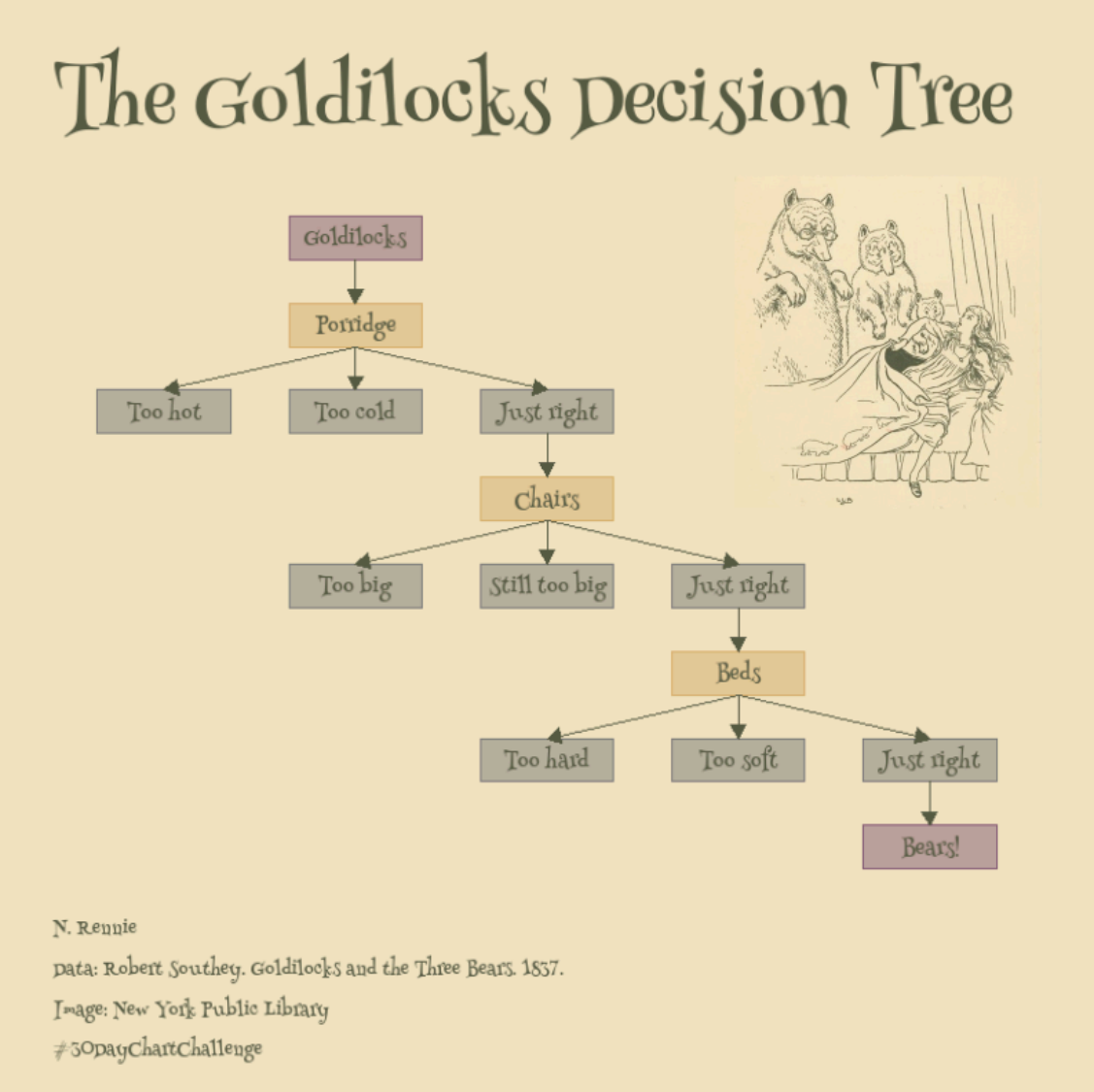
```
11                                  size = 10, color = "#585c45",
12                                  margin = margin(t = 10)))
```



Adding images

There are a few different packages in R that are capable of adding images on top of plots. I most commonly use a combination of {magick} and {cowplot}. However, in this instance, I actually used Inkscape.org, a free, open-source image editing tool, instead. The process of adding the image on top of my plot, and arranging it exactly where I wanted, was much simpler and faster using Inkscape rather than R in this case.

And that gives us the final image:



Hopefully, this tutorial blog demonstrated the process of creating a flowchart in R using {igraph} and {ggplot2}, and encourages you to create your own! You can also find the slides and recording of the talk I gave to R-Ladies Nairobi here.

For attribution, please cite this work as:

> **Creating flowcharts with {ggplot2}**.
> Nicola Rennie. June 6, 2022.
> nrennie.rbind.io/blog/creating-flowcharts-with-ggplot2

← 30 Day Chart Challenge 2022 Mapping a marathon with {rStrava} →
