Simple Chart Type Examples

Anamaria Crisan

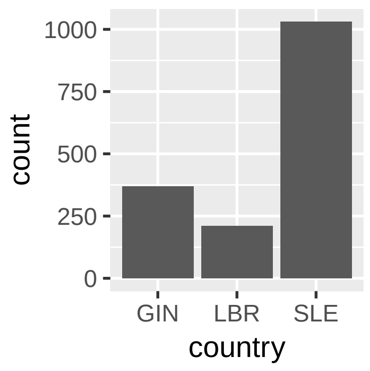
2019-01-08

This document demonstrates how to implement and plot different chart types using minCombinR. This document assumes that you have already run the "Getting started with minCombinR" and that the necessary data has already been loaded into your R workspace.

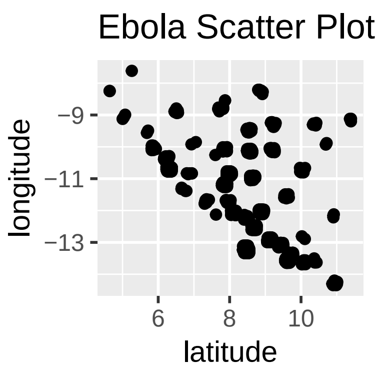
```r
devtools::load_all() # TODO: temporary once things are done
library(dplyr)
library(shiny)

# Table data
tab_dat <- input_data(file = system.file("extdata", "ebov_metadata.csv", package = "mincombinr"),
                      dataType = "table")

# Tree data
tree_dat<-input_data(file = system.file("extdata", "ebov_tree.nwk", package = "mincombinr"),
                     dataType = "tree")

# Genomic data
genomic_dat<-input_data(file = system.file("extdata", "ebov_GIN_genomic_FIXED.fasta", package = "mincomb
                        dataType = "dna")

# Shape files
# Shape files require that .shp, .shx, and .prj files at a minimun to be in the same directory.
# If you would like to add metadata to the shape file, you can also add .dbf files.
gin_shape_dat <- input_data(file = system.file("extdata", "gin_admbnda_adm1_ocha_itos.shp", package = "r
                            dataType = "spatial")

lbr_shape_dat <- input_data(file = system.file("extdata", "lbr_admbnda_adm1_ocha.shp", package = "mincor
                            dataType = "spatial")

sle_shape_dat <- input_data(file = system.file("extdata", "sle_admbnda_adm1_1m_gov_ocha_20161017.shp",
                                               package = "mincombinr"),
                            dataType = "spatial")

# Put all the individual shape files together so that the system knows to try visualize it all together
shape_dat <- join_spatial_data(gin_shape_dat, lbr_shape_dat, sle_shape_dat)
```

Common statistical charts

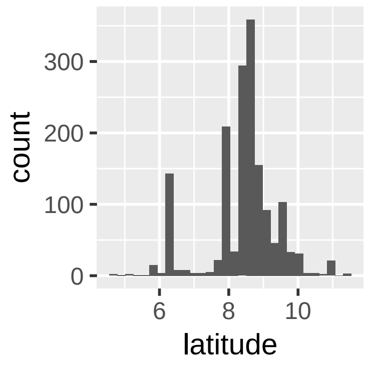
```r
#
# TODO: "swarmplot" => implemented specs for this

# Let's specify and plot some single charts.

# Bar chart:
bar_chart <- specify_single(chart_type = "bar", data = "tab_dat", x = "country")
plot(bar_chart)
```



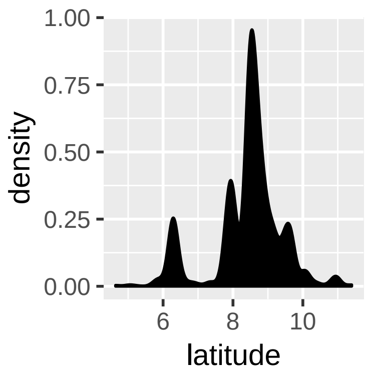
```
# Scatter plot with a title:
scatter_chart <- specify_single(chart_type = "scatter", data = "tab_dat", x = "latitude", y = "longitud
plot(scatter_chart)
```



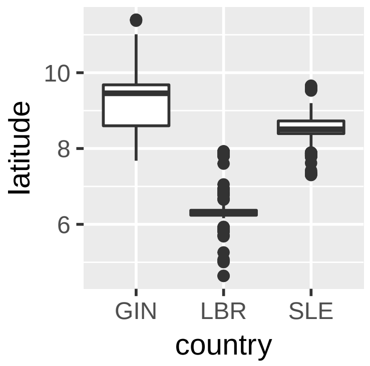
```
# Histogram:
histogram_chart <- specify_single(chart_type = "histogram", data = "tab_dat", x = "latitude")
plot(histogram_chart)
```



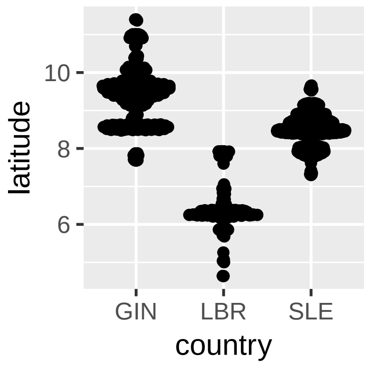
```
# Probability Density Function (PDF) plot:
pdf_chart <- specify_single(chart_type = "pdf", data = "tab_dat", x = "latitude")
plot(pdf_chart)
```



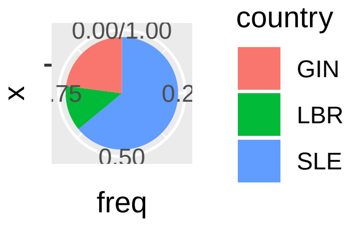
```r
# Boxplot
boxplot_chart <- specify_single(chart_type = "boxplot",data="tab_dat",x = "country", y ="latitude")
plot(boxplot_chart)
```



```r
# Swarm plot
beeswarm_chart <- specify_single(chart_type = "swarmplot",data="tab_dat",x = "country", y ="latitude")
plot(beeswarm_chart)
```

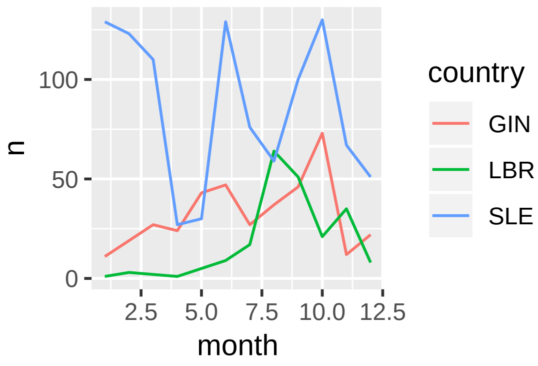


```r
# We'll even let you make a pie chart
pie_chart <- specify_single(chart_type = "pie", data = "tab_dat", x = "country")
plot(pie_chart)
```



minCombinR can also work with data frames so you can perform some analyses and go ahead and plot the data. Here's an example using a line chart :

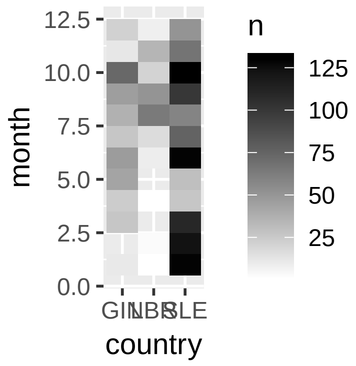
```r
# Let's use our tabular data:
ebov <- tab_dat@data[[1]] # TODO: make this nicer
ebov_case_counts <- ebov %>%
  group_by(country, month) %>%
  count()

# Now let's specify and plot a line chart:
line_chart <- specify_single(chart_type = "line", data = "ebov_case_counts", x = "month", y = "n", grou
plot(line_chart)
```



Color charts

Color charts are common statistical charts that fundamentally require color to communicate their results. By comparsion, adding color to a common statistical chart can be seen as an enhancement (a nice to have, not a need to have).

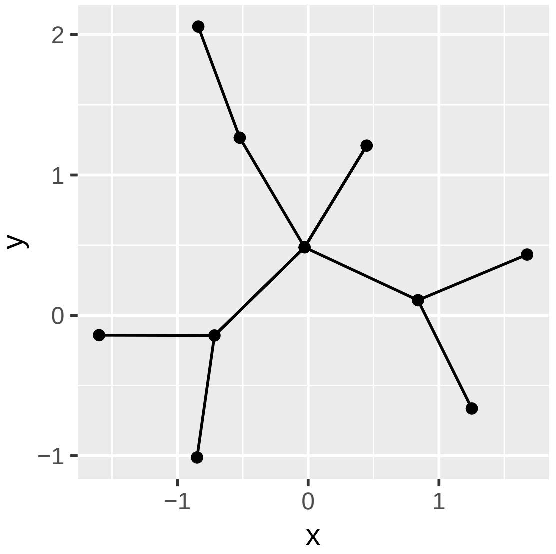
```r
# Get our data:
ebov <- tab_dat@data[[1]] # TODO: make this nicer
ebov_heat_data <- ebov %>%
  group_by(country,month) %>%
  count()

# Specify and plot a line chart:
heatmap_chart <- specify_single(chart_type = "heatmap", data = "ebov_heat_data", x = "country", y = "mo
plot(heatmap_chart)
```



Relational charts

These data do not have network data, so we'll make up so data to use.

```
# Data was found in r-graph-gallery:
links_data <- data.frame(
  source = c("A","A", "A", "A", "A", "J", "B", "B", "C", "C", "D","I"),
  target = c("B","B", "C", "D", "J","A","E", "F", "G", "H", "I","I")
)

highschool_dat <- ggraph::highschool

node_link <- specify_single(chart_type = "node-link", data = "links_data")
# And plot!
plot(node_link)
```



Spatial charts

The associated data with the spatial charts is always a bit tricky because there are a lot of ways that a user could bring such data in and it's not possible for the system to catch them all. So, some extra attention needs to be paid for this chart type.

minCombinR does some work for you when you join spatial datasets, so we can use that to add information. At a bare minimum, if you don't just want to draw the polygons of the shape file, but you want to add some information to them, you need to make sure that the data you've loaded in actually contains usable information. A lot of data does not, and it's not possible for minCombinR to fill in those gaps.

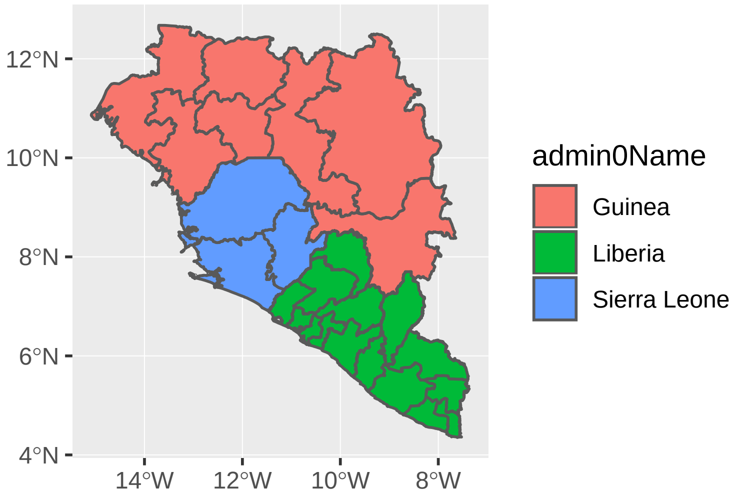
```
meta_tmp <- shape_dat@data$metadata
geo <- shape_dat@data$geometry
```

There are two ways to view a geographic map. First, working from shape files:

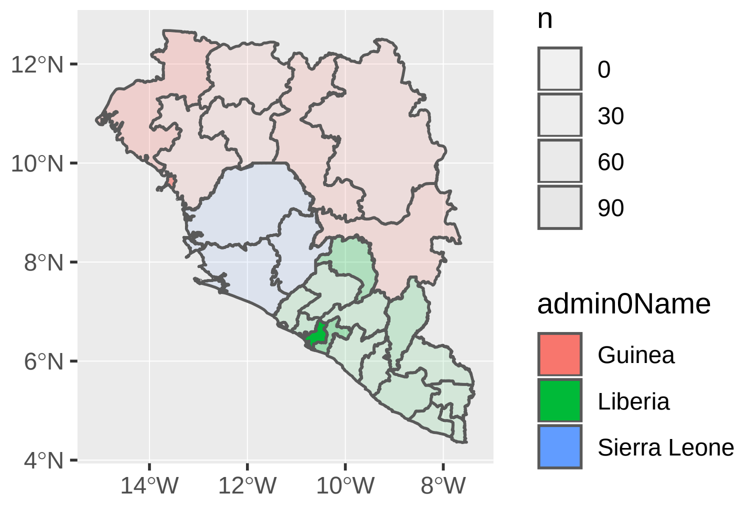
First, let's specify and plot all of the regions in the map together

```
# TODO: I do not think these comments below are relevant anymore b/c we load in the metadata above (Sha

# metadata from the shape files is incomplete
# this is something you need to know and not something that the systems resolves for you
```

```r
# so, we'll work to clean up the metadata a bit more before we # TODO: Not sure what this was trying to
spatial_chart <- specify_single(chart_type = "choropleth", data = "shape_dat", color = "admin0Name")
plot(spatial_chart)
```



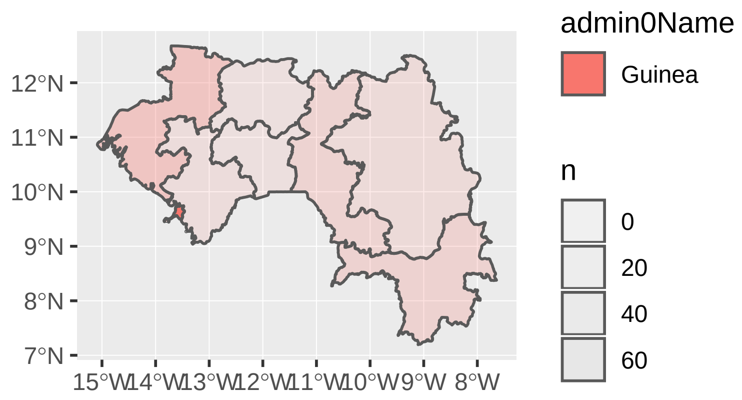
```r
#metadata frome the shape files is incomplete
#this is something you need to know and not something that the systems resolves for you
#so, we'll work to clean up the metadata a bit more before

shape_meta<-shape_dat@data$metadata

#a little wrangling to put things under the same
#column header since they have slightly different names
tmp<-shape_meta$admin1Name
tmp[is.na(tmp)]<-as.character(shape_meta$admin1name[is.na(tmp)])
shape_meta$admin1Name<-tmp

#now add this with the sample data
tab<-tab_dat@data[[1]] %>% group_by(location) %>% tally()
tab<-left_join(shape_meta,tab,by=c("admin1Name"="location"))
tab[is.na(tab$n),]$n<-0

#visualize
spatial_chart<-specify_single(chart_type="choropleth",
                              data="shape_dat",
                              metadata="tab",#has priority over metadata stored with spatial object
                              color="admin0Name",
                              alpha="n")
plot(spatial_chart)
```



Also, it possible just to see one individual map too, not just everything together

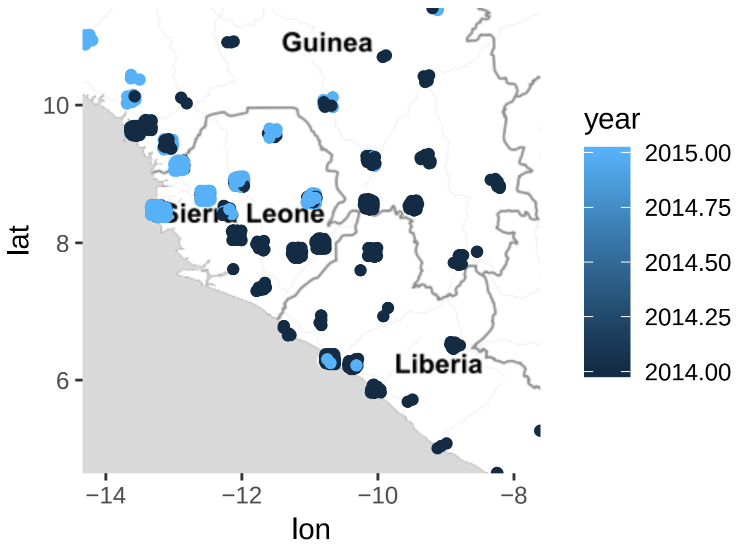
Here's an example where we compute some new data and specify and plot the "Guinea" region

```
# Let's change things up a little bit and plot the incidence of things

# You can change the metadata
gin_meta <- gin_shape_dat@data$metadata

tab<-tab_dat@data[[1]] %>% group_by(location) %>% tally()

tab<-left_join(gin_meta,tab,by=c("admin1Name"="location"))
tab[is.na(tab$n),]$n<-0

spatial_chart<-specify_single(chart_type="choropleth",data="gin_shape_dat",metadata = "tab",color="admi
plot(spatial_chart)
```



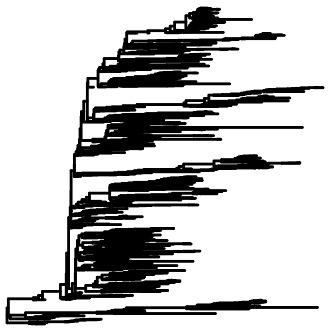
Second, is working from a co-ordinates from a tabular data file. The background raster image is supplied by the 'Openstreemaps' package (and the broader project). They do great work, consider supporting them if you like this feature too.

```
# Geographic Map
map_chart <- specify_single("geographic map", data = "tab_dat", lat = "latitude", long = "longitude", c
plot(map_chart)
```



Tree charts

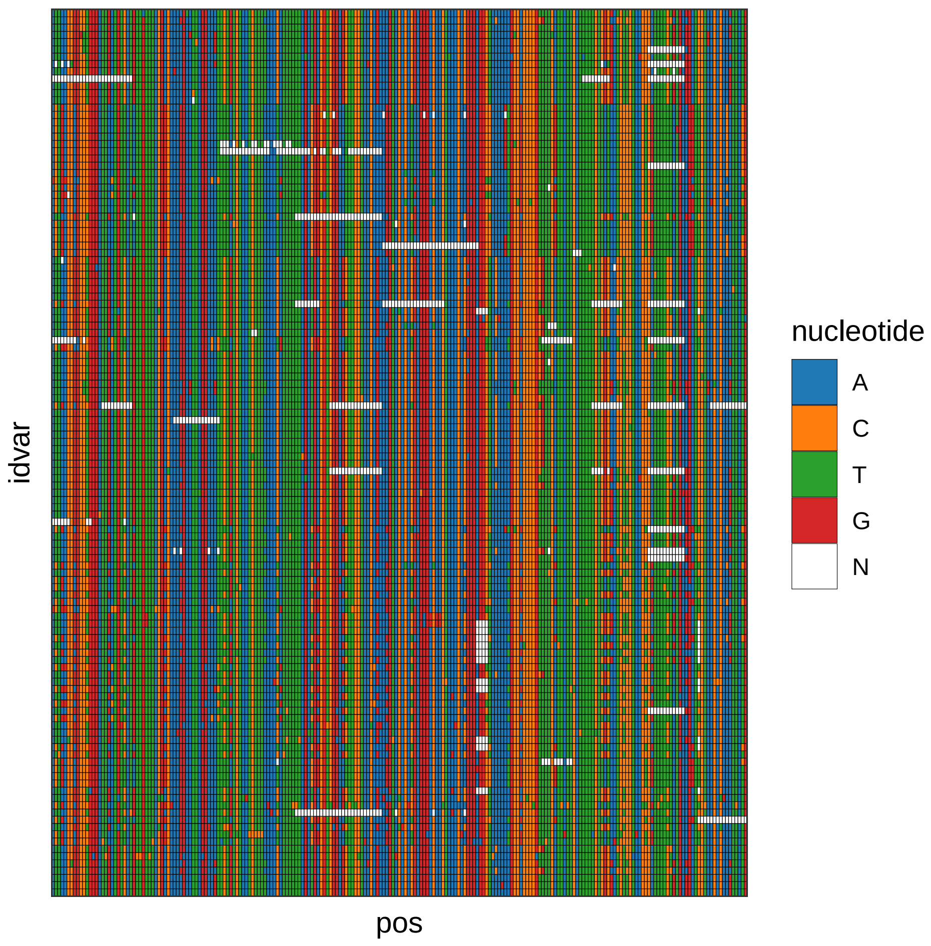
```r
# Phylogenetic Tree
phyloTree_chart <- specify_single(chart_type = "phylogenetic tree", data = "tree_dat")
plot(phyloTree_chart)
```



Genomic charts

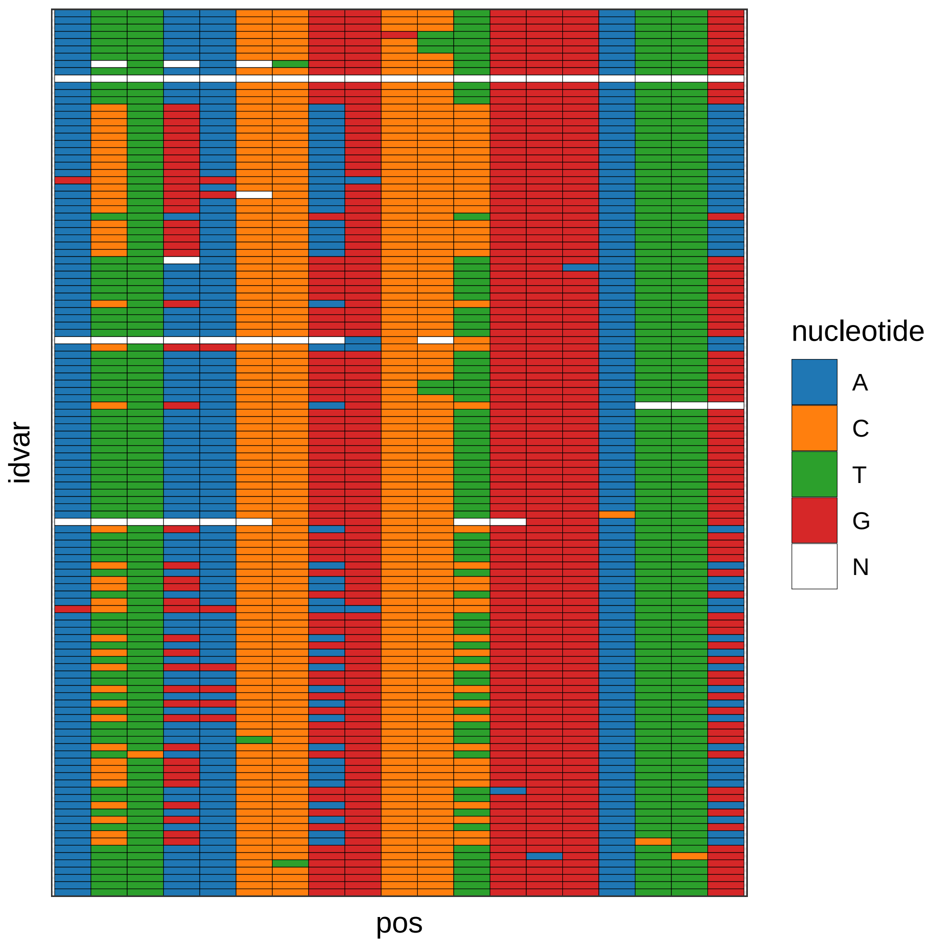
A very standard alignment chart. This tends to still work if you've got very long sequences. There are some options to simply the output too. It's possible to pass a **diff_only** parameter, which will only show variant positions.

```r
# Alignment
genome_chart <- specify_single(data = "genomic_dat", chart_type = "alignment")
plot(genome_chart)
```



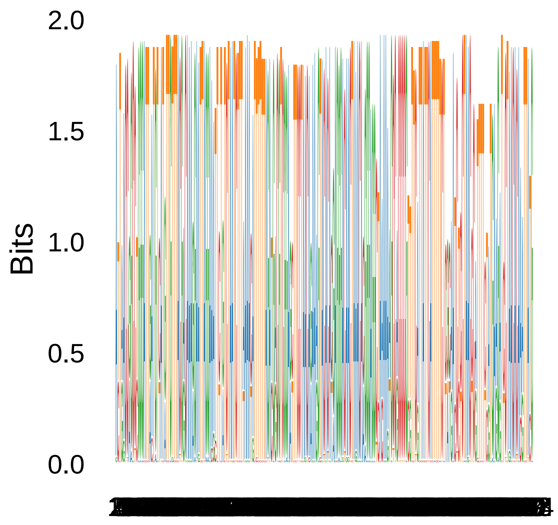
We can also just look at the subset of the data, to make life a little bit easier.

```
# Alignment
diff_seq <- get_diff_pos(genomic_dat)
genome_chart <- specify_single(data = "genomic_dat", chart_type = "alignment",show_pos=diff_seq[1:20])
plot(genome_chart)
```



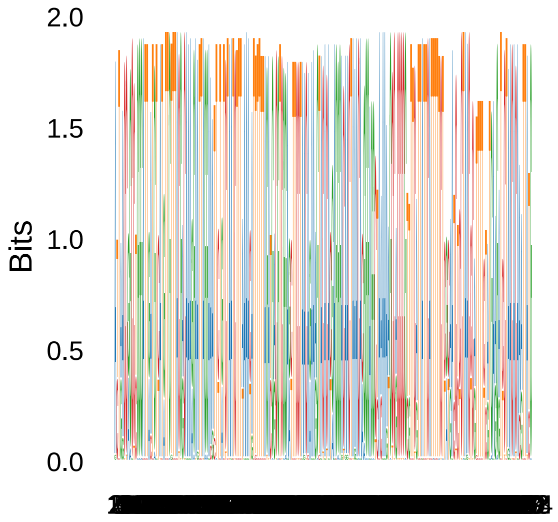
Sequence logo plots are also supported. These work well when there are a small number of variant positions, but when there are many, it's actually not a very nice visual. The user will recieve a prompt in those circumstances.

```r
seqlogo_chart <- specify_single(data = "genomic_dat", chart_type = "sequence logo")
plot(seqlogo_chart)
```



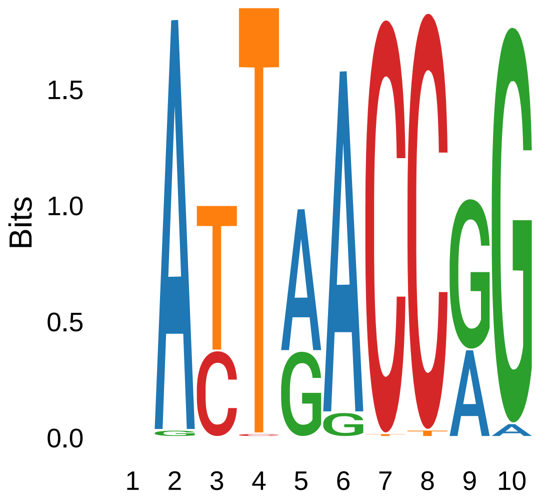
See? Too many. Instead, it might be better to select a few specific regions to show. In the first case, there will be a warning that nothing is very different and the plot won't show:

```r
seqlogo_chart <- specify_single(data = "genomic_dat", chart_type = "sequence logo", show_pos = 1:20)
plot(seqlogo_chart)
```



But there is a slightly more clever way to go about this too:

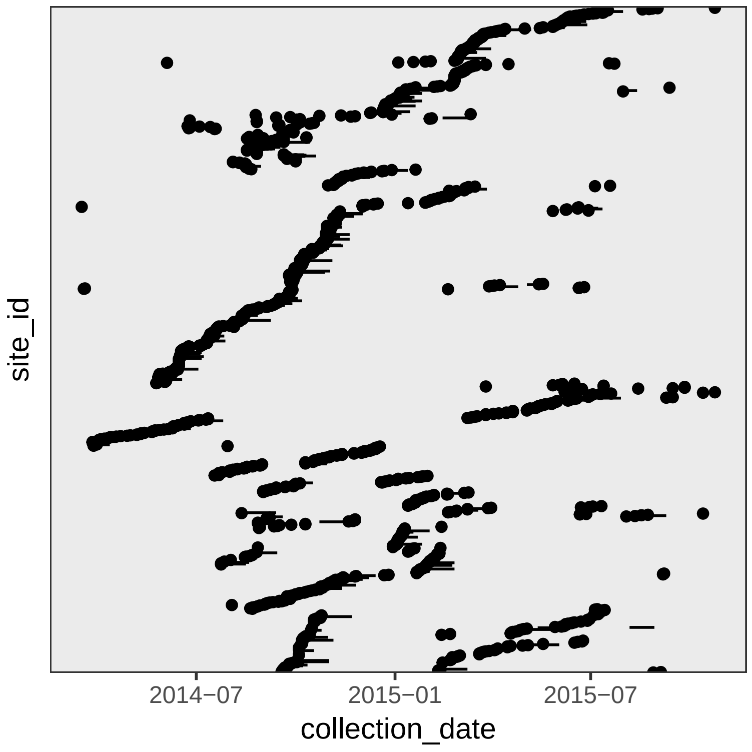
```r
# Here is a little helper function that extracts the similar sequences
diff_seq <- get_diff_pos(genomic_dat)

# Sequence Logo
seqlogo_chart <- specify_single(data = "genomic_dat", chart_type = "sequence logo", show_pos = diff_seq
plot(seqlogo_chart)
```



Temporal charts

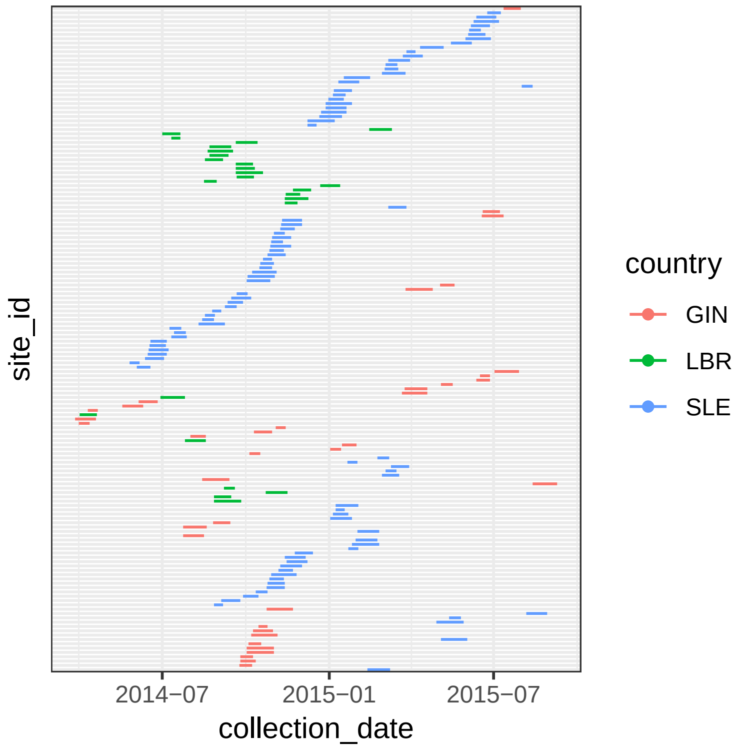
Sometimes data contains start and end dates and its desirable to show these as different types of temporal charts.
There are two times of timelines that can be created:
1. A "gantt chart" type timeline that will show both ranges and single events.
2. A standard epidemic curve, which is essentially a very special case of a histogram or bar chart, depending upon the user.

```
# Using the existing tabular data:
tmp <- tab_dat@data[[1]]
tmp$collection_date <- as.Date(tmp$collection_date)

# Let's add some end dates to keep it interesting:
tmp$collection_date_end <- tmp$collection_date + sample(10:30,nrow(tmp), replace = TRUE)

tmp$collection_date_end <- sapply(as.character(tmp$collection_date_end),
                                  function(x){
                                    if(runif(1)>0.9)
                                      return(x)
                                    return(NA)
                                    })

timeline_chart <- specify_single(chart_type = "timeline",
                                data = "tmp",
                                start = "collection_date",
                                end ="collection_date_end",
                                y = "site_id")
plot(timeline_chart)
```



Since there's a lot going on, let's subset this to get a better view.
So here, only plotting those items that are ranges.

```
tmp_sub <- dplyr::filter(tmp,!is.na(collection_date_end))

timeline_chart <- specify_single(chart_type = "timeline",
                                data = "tmp_sub",
                                start = "collection_date",
                                end = "collection_date_end",
                                y = "site_id",
                                color = "country")
plot(timeline_chart)
```



Images

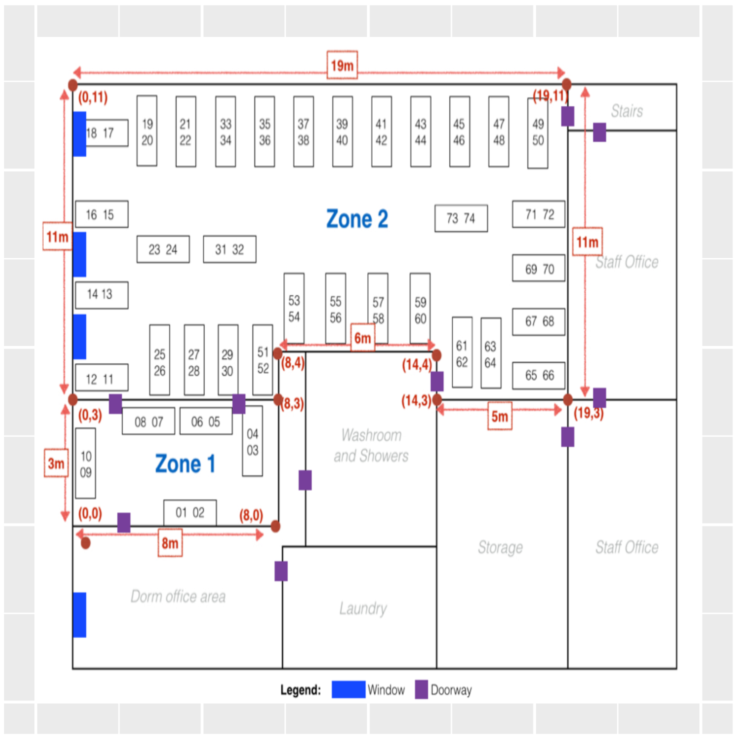
Sometimes it's desirable to use image data, these can be from interior maps or from gel images. Getting image data to work is a little bit more involved because it requires some annotation data that allows you to link pixel space to something useful. There's a small application embedded within minCombinR that lets you do that.

Here's a workflow with a few different types of images:

```
#Load in interior map image data into minCombinR's input_data function
interior_img <- input_data(file = system.file("extdata", "random_interior_map.tiff", package = "mincombi
                                  dataType = "image")
```

It's still possible to simply draw a plot that has no metadata

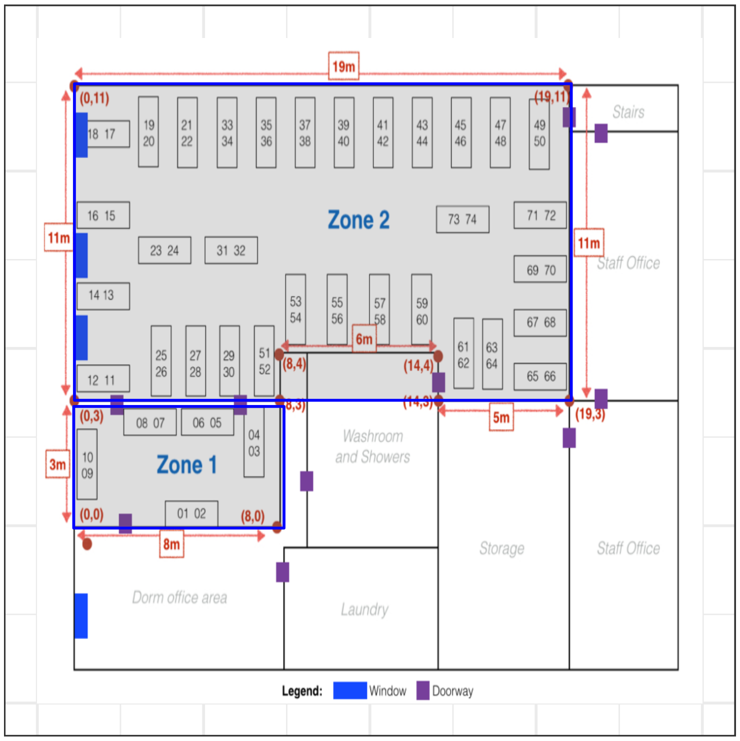
```
img_chart <- specify_single(chart_type = "image", data = "interior_img")
plot(img_chart)
```



But it’s nice to do something more useful with a picture.
The error you saw when loading the data is to remind the user that there needs to be some image file loaded.
To add metadata, we’ll run the special app.
Note that in building this markdown file, this block of code is not run, however, **should** be done by the user if they would like to add some metadata.

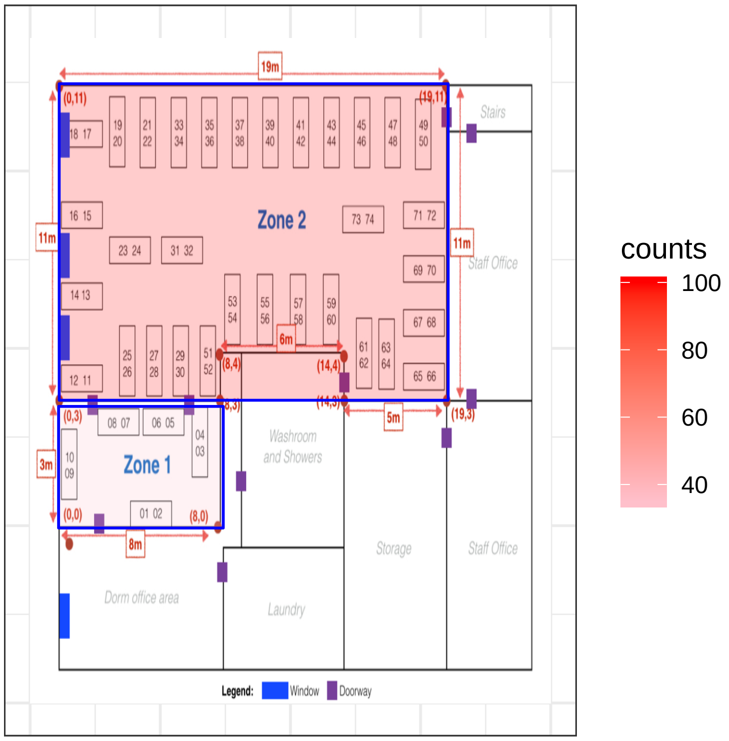
```
interior_img <- annotate_image(interior_img)

# The annotations are automatically there now:
metadata <- interior_img@data$metadata

save(file="../inst/extdata/img_meta.rds",metadata)
```

```
# We'll load an already annotated image to keep things going
# Normally, the user doesn't do this, but we have to do so because
# the previous line of code was not run in the markdown notebook
load(file = system.file("extdata", "img_meta.rds", package = "mincombinr"))
interior_img@data$metadata<-metadata

# Now specify and plot the interior map
img_chart <- specify_single(chart_type = "image", data = "interior_img")
plot(img_chart)
```



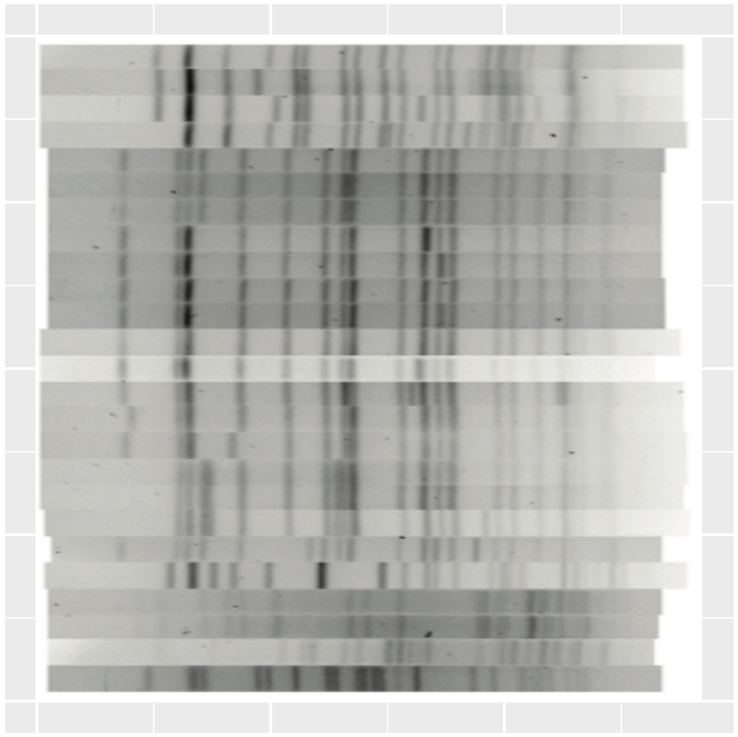
Let’s do something more interesting and color these regions:

```r
# Let's add some arbitrary case counts to the room
interior_img@data$metadata$counts <- c(100,35)

# Now specify and plot the interior map
img_chart <- specify_single(chart_type = "image", data = "interior_img", color = "counts")
plot(img_chart)
```



The other most common type of image is a gel image - this too can now be part of the analysis fun.

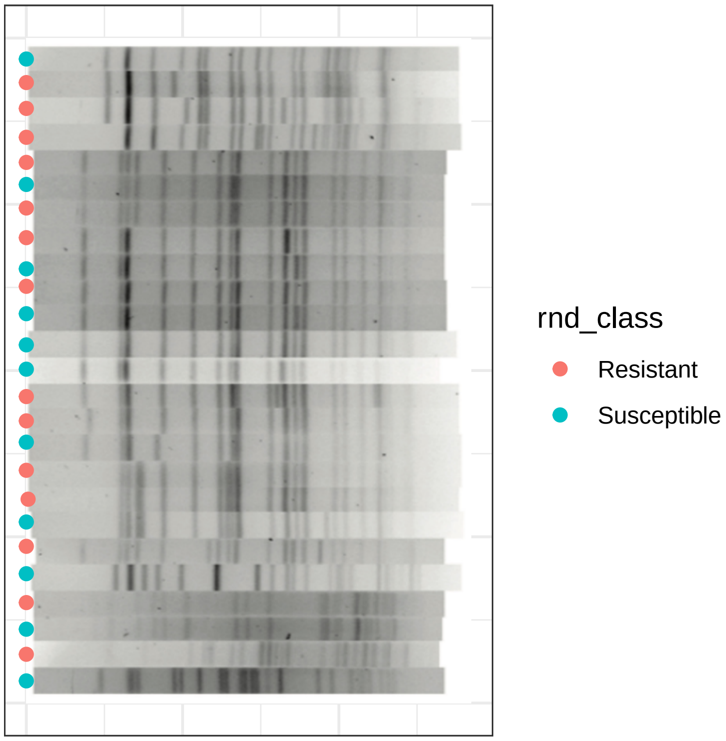
```r
gel_img <- input_data(file = system.file("extdata", "gel_image.tiff", package = "mincombinr"),
                      dataType = "image")

gel_img_chart <- specify_single(chart_type = "image", data = "gel_img")
plot(gel_img_chart)
```



Just like the interior map, this block of code is not run by the notebook but should be used to annotate an image as a user.

```
gel_img <- annotate_image(gel_img)
metadata <- gel_img@data$metadata
metadata$element_name <- paste("item", 1:nrow(metadata), sep = "_")

save(metadata,file="../inst/extdata/gel_img_meta.rds")
```

```
# Just like the interior map, we will load in an annotated image for the sake of demonstration:
load(file = system.file("extdata", "gel_img_meta_shipped.rds", package = "mincombinr"))
gel_img@data$metadata <- metadata

gel_img@data$metadata$rnd_class <- sample(c("Resistant","Susceptible"), replace = TRUE, size = nrow(meta

#Now we specify and plot the gel image
gel_chart <- specify_single(data = "gel_img", chart_type = "image", color = "rnd_class")
plot(gel_chart)
```


rnd_class
● Resistant
● Susceptible