Data Visualization Examples

Leonard Wee

24-1-2022

To begin with, we will again source those useful and re-usable functions which we keep and maintain elsewhere. This keeps our Markdown here focussed, on point and relatively clean. We saw this step in the previous markdown document.

```
## [1] "dplyr"    "magrittr" "httr"     "ggplot2"  "GGally"
```

It is assumed here that the idea behind scatterplots, pie charts, bar charts and such like are already familiar to you. Equivalent code for doing these in R are easily findable in many places, included in the recommended textbooks.

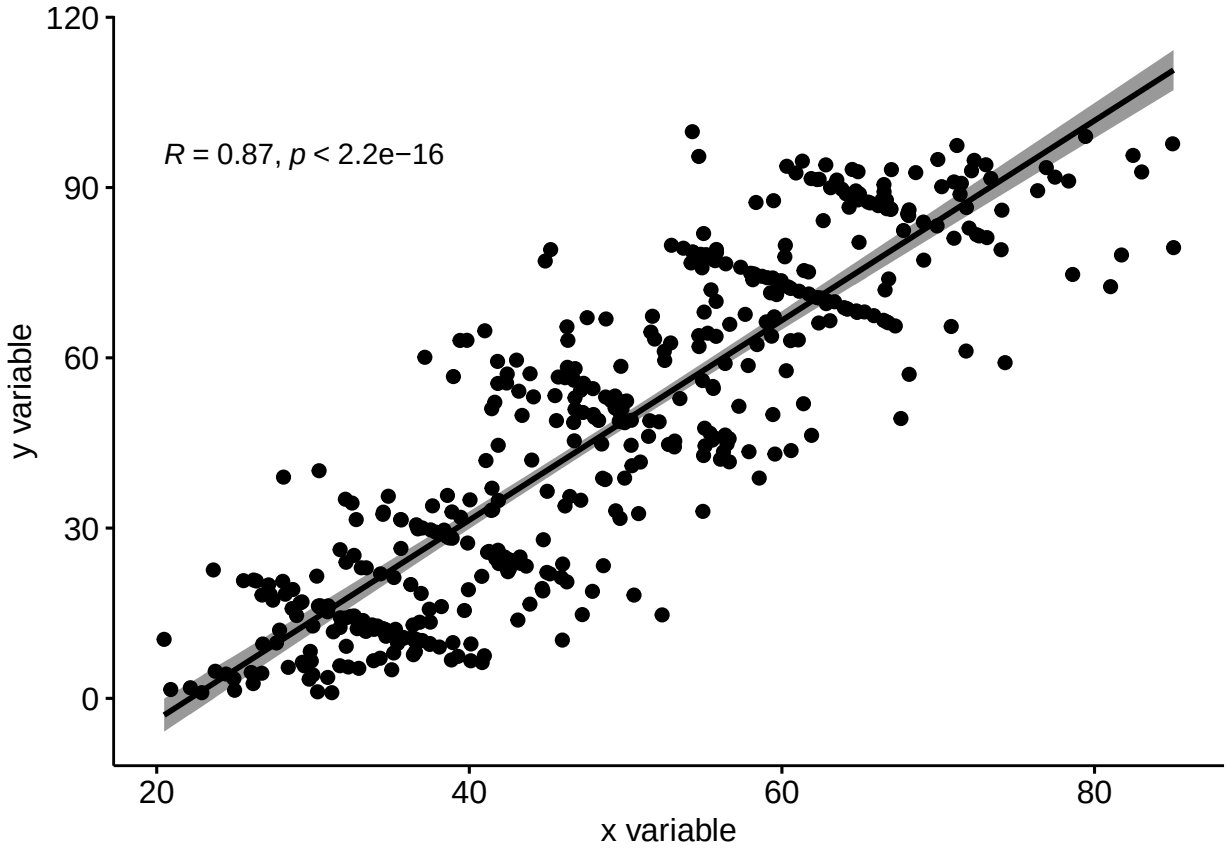
We will use some of these of course, and also show how much more you could do in R and how to make better use of the generalizability and flexibility of R.

Some basic manipulations to summarize data

```
##               Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) -38.975999 2.37104602 -16.43831  9.969771e-48
## x             1.759638 0.04646745  37.86819 7.778496e-141
```

```
##
##  Pearson's product-moment correlation
##
## data:  data_viz_1$x and data_viz_1$y
## t = 37.868, df = 442, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8504273 0.8945716
## sample estimates:
##       cor
## 0.8742953
```

```
## [1] "ggpubr"
```

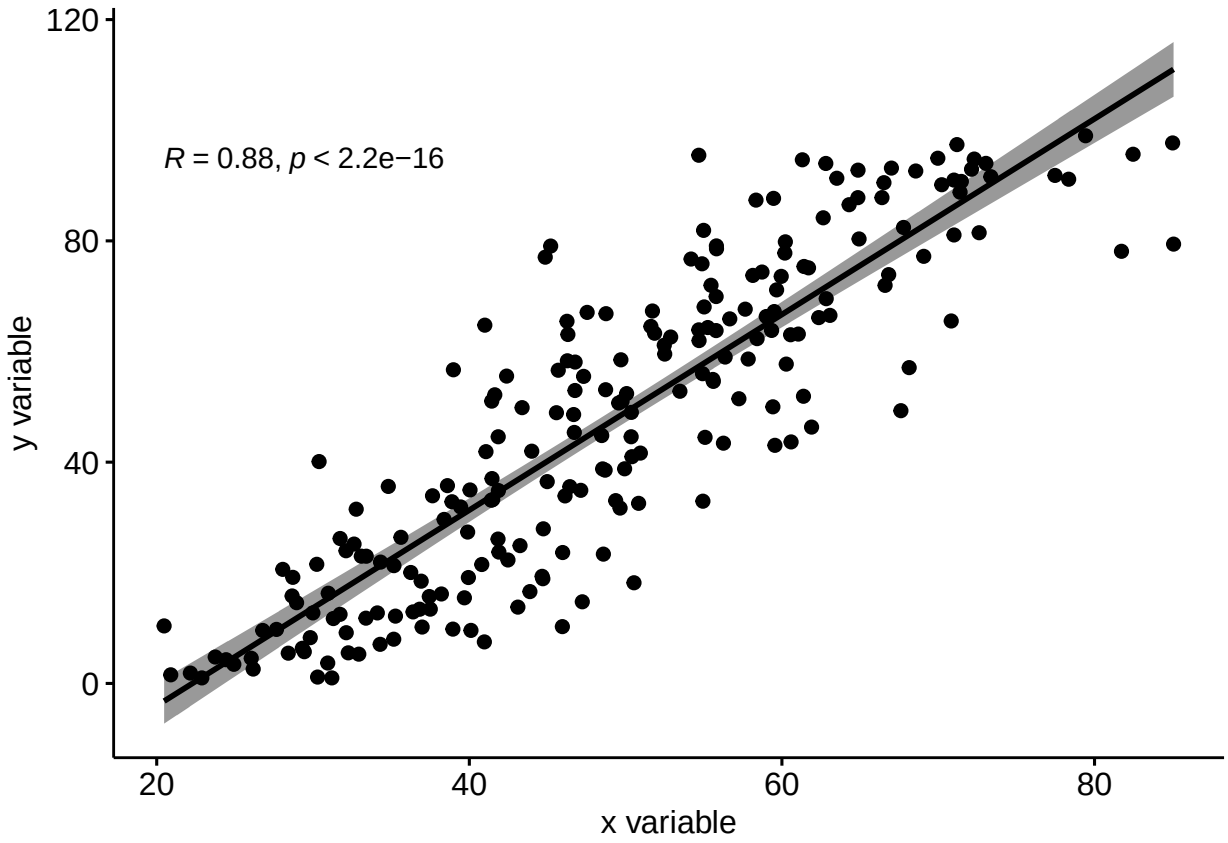


In the above code fragment, it seems like there is "structure" embedded in the data that does not seem to be captured in the model. It is entirely reasonable to feel pleased with ourselves for a highly significant model, but wait - the purpose of DATA EXPLORATION is **not** to crow about how good our model is, at least not yet! The purpose is to look for underlying relationships in the data that maybe the model - a linear regress in this case - might not pick up.

Let's analyse the each subgroup separately, and see where this gets us.

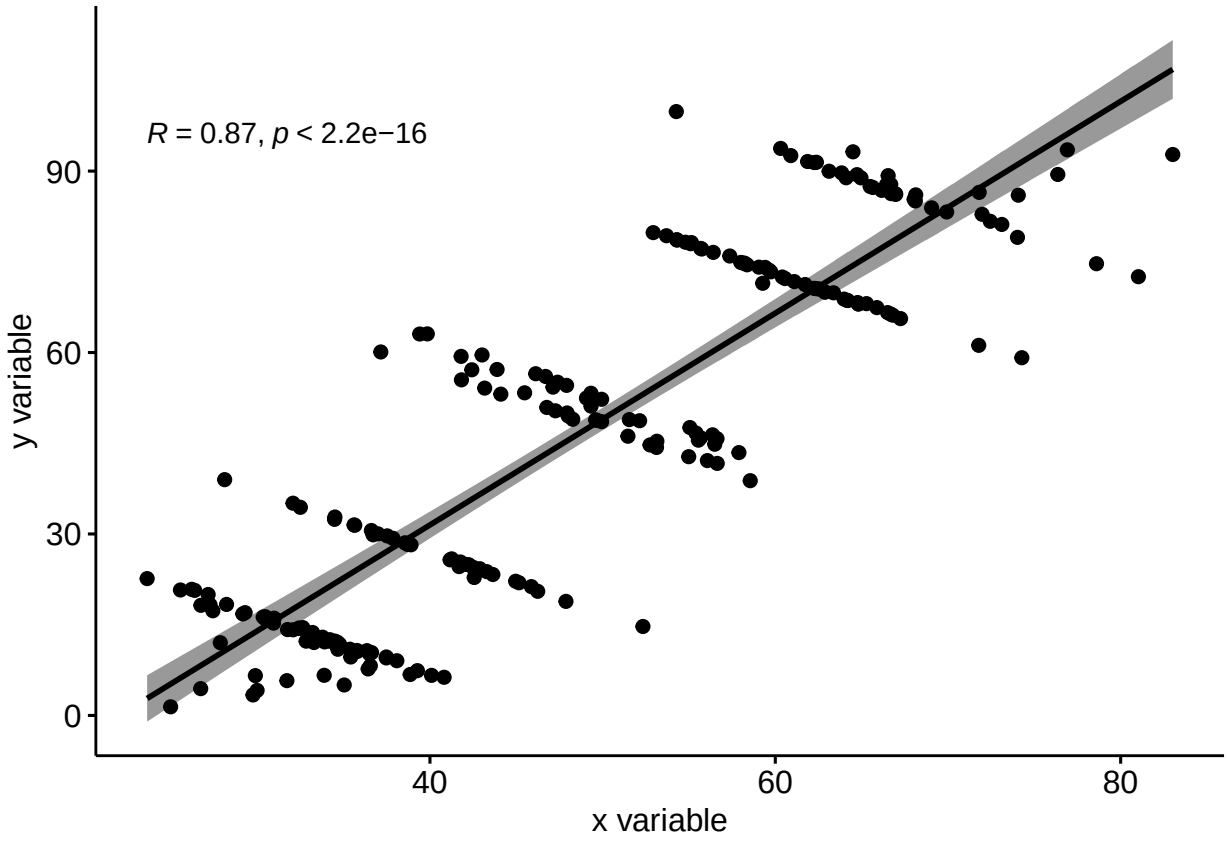
```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -39.397485 3.30652263 -11.91508 1.390323e-25
## x             1.768218 0.06480234  27.28633 1.503025e-72
```

```
##
##  Pearson's product-moment correlation
##
## data:  data_viz_A$x and data_viz_A$y
## t = 27.286, df = 220, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8446238 0.9055045
## sample estimates:
##       cor
## 0.8785857
```




```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -38.554276 3.41394606 -11.29317 1.262304e-23
## x             1.751055 0.06690449  26.17246 1.680236e-69
```

```
##
##  Pearson's product-moment correlation
##
## data:  data_viz_B$x and data_viz_B$y
## t = 26.172, df = 220, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8338529 0.8987222
## sample estimates:
##       cor
## 0.8700031
```



Take a moment here to discuss and describe what you think is going on in this dataset.

In the next code block, I have started it for you by importing a dataset. As an exercise in writing R code, show that each sub-group in the dataset "data-exp-2" has the same mean x, mean y, standard deviation x, standard deviation y and even the same Pearson correlation coefficient as all of the other sub-groups. However, the underlying structure in each the sub-groups are distinct.

Scatter and grouped scatter

Simple checks for outliers and unusual responses

Scatterplots are one of the most basic plots you could make, and for continuous variables can be quite telling. In the next example, I am re-using the **recodedHeartFailure** dataset from before.

The plot library **ggplot2** allows for many different kinds of high quality plots, and you may need to take it slow building your plots as a beginner.

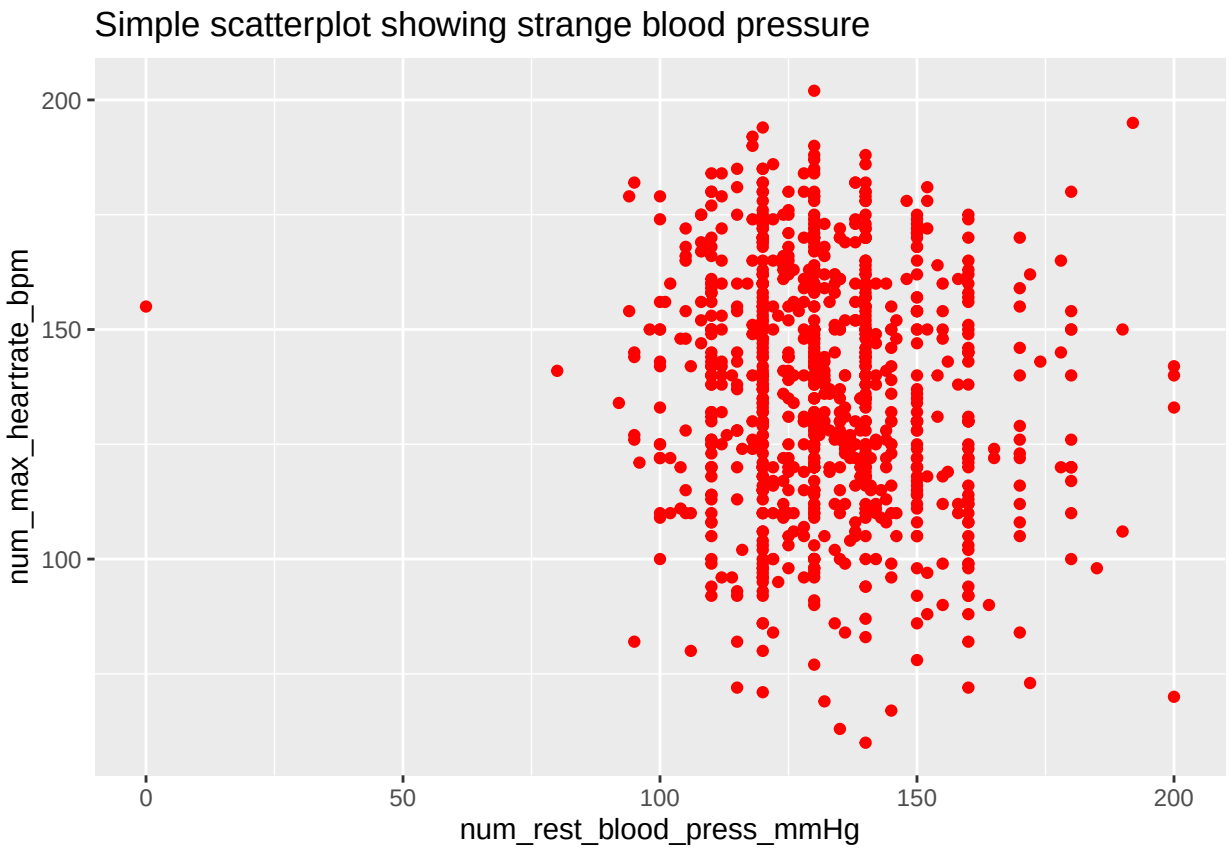
We generally begin by defining the data to plot, add "aesthetics" in terms of x and y variables, and then define a scatter plot by plotting points (in the example below in red colour).

Later, we can then add titles, axes labels etc but it is always a good idea to just get the key elements down first and then add improvements little bits at a time.

In the example scatter plot below, I might already suspect that (a) there are two distinct types of blood pressure measuring devices used in this study, and that (b) zero values might actually be intended to convey missing values ie they are not really numerically zero.

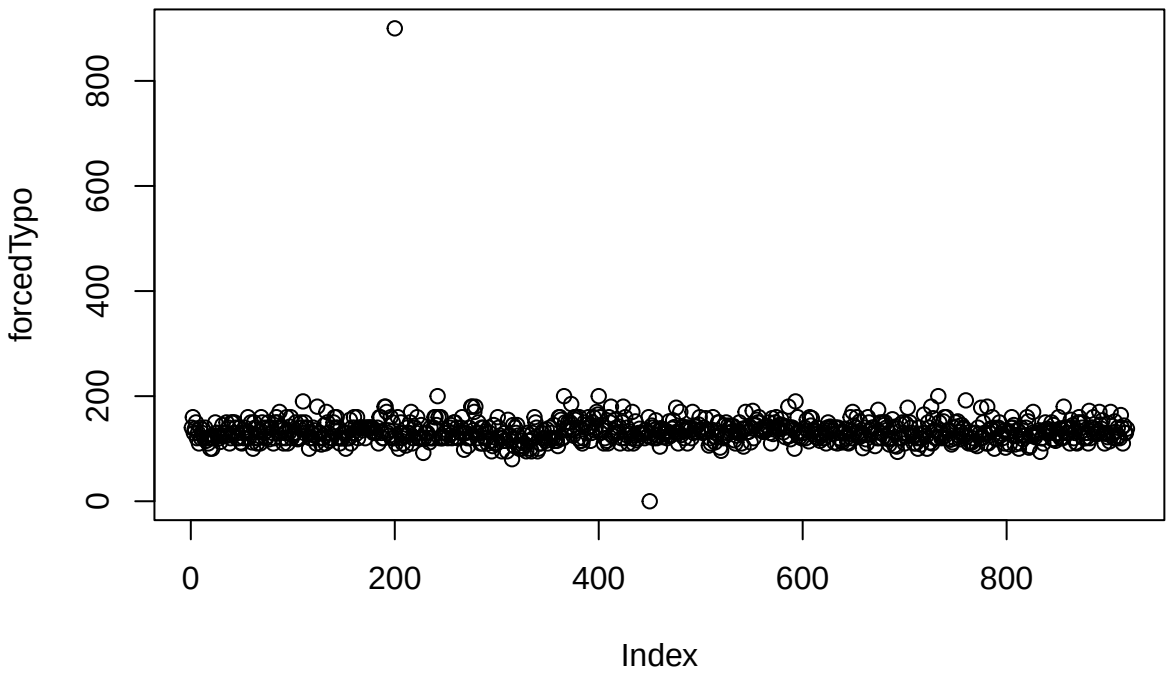
Can you see from the figure below why I might wonder about this?

```
ggplot(data = recodedHeartFailure,
  aes(x=num_rest_blood_press_mmHg, y=num_max_heartrate_bpm)) +
  geom_point(color="red") +
  ggtitle("Simple scatterplot showing strange blood pressure")
```



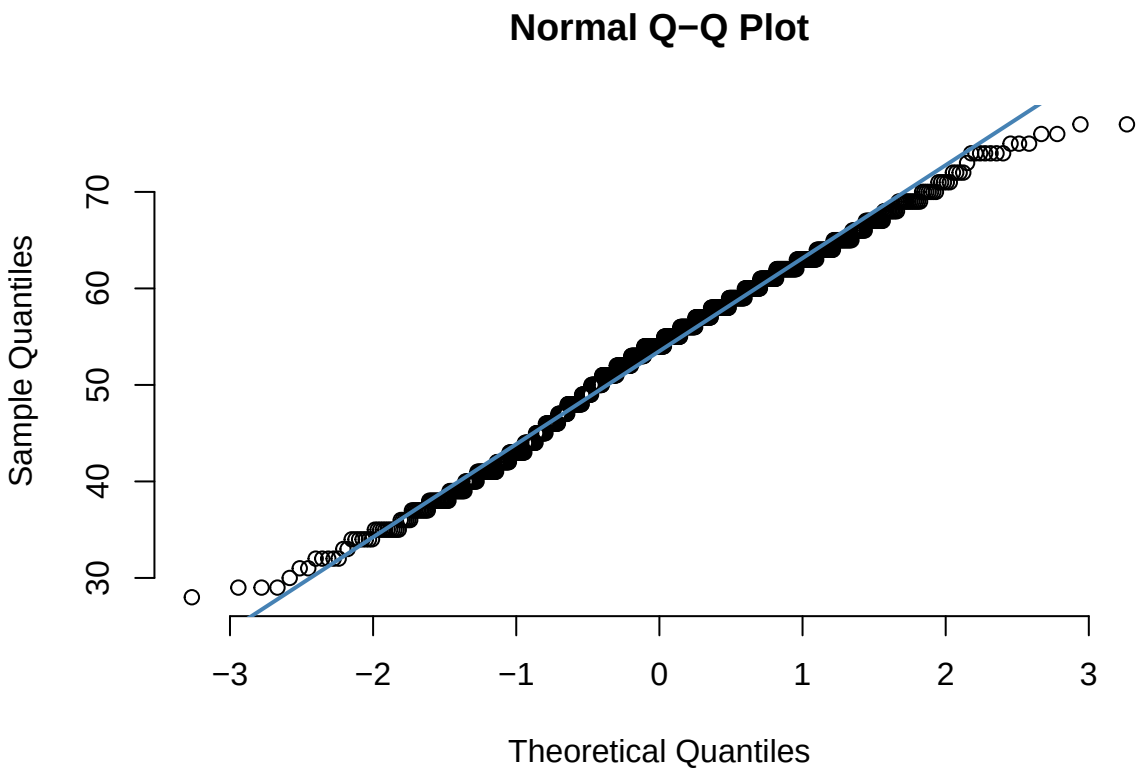
In R, you can also simply plot just one variable at a time, using the built-in basic **plot** function, to look for strange values and outliers. I am going to intentionally introduce a very obvious typo to one of the values as follows :

```
forcedTypo <- recodedHeartFailure$num_rest_blood_press_mmHg
forcedTypo[200] <- 900 #intentionally made a typo, perhaps it was meant to be 90, most likely not 900
plot(forcedTypo)
```



Another question that is sometimes asked is whether or not a variable conforms to some kind of well known statistical distribution such as a normal (Gaussian) distribution. For that, R also has a straightforward built-in function to perform a Q-Q plot as follows, but you can look up tests for other distributions. Note that you can always include some kind of statistical test (e.g. a Shapiro-Wilks test of normality) but don't underestimate the power of just looking very quickly at the distribution - this is a data visualization module after all!

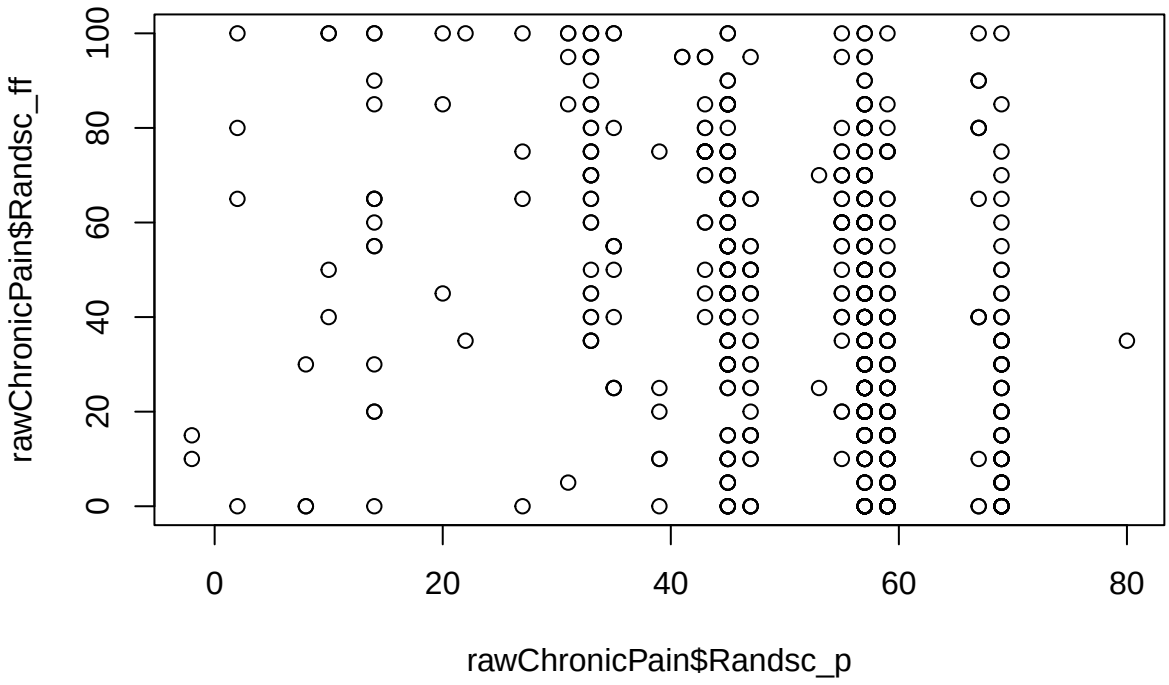
```
my_data <- recodedHeartFailure$num_age_in_years
qqnorm(my_data, pch = 1, frame = FALSE)
qqline(my_data, col = "steelblue", lwd = 2)
```



Continuous variables that may not be actually continuous?

In the next examples, I am going to use the chronic pain dataset provided in this course. In the documentation for this dataset you will be told that the variables are all continuous.

Here is a simple plot of the the Physical Functioning Domain Score against the Pain Domain Score. Do you think the variables are continuous? What do you suggest might the better way to handle such data? Below the plot is a suggested method to **cut** the data into discrete groups.




```
cutpoints <- c(-Inf, 10, 20, 30, 40, 50, 60, 70, 80, 90, Inf)
Randsc_p_cut <- cut(rawChronicPain$Randsc_p, cutpoints, right=FALSE)
summary(Randsc_p_cut)
```

```
## [-Inf,10)   [10,20)   [20,30)   [30,40)   [40,50)   [50,60)   [60,70)   [70,80)
##        11        18        10        56       158       397        83         0
##   [80,90) [90, Inf)      NA's
##         1         0         8
```

This function **cut** to force groups onto an otherwise continuous (or sort-of continuous) variable will turn out to be very useful later on, so just bear in mind that it exists to break up a contiguous range into arbitrary subgroups.

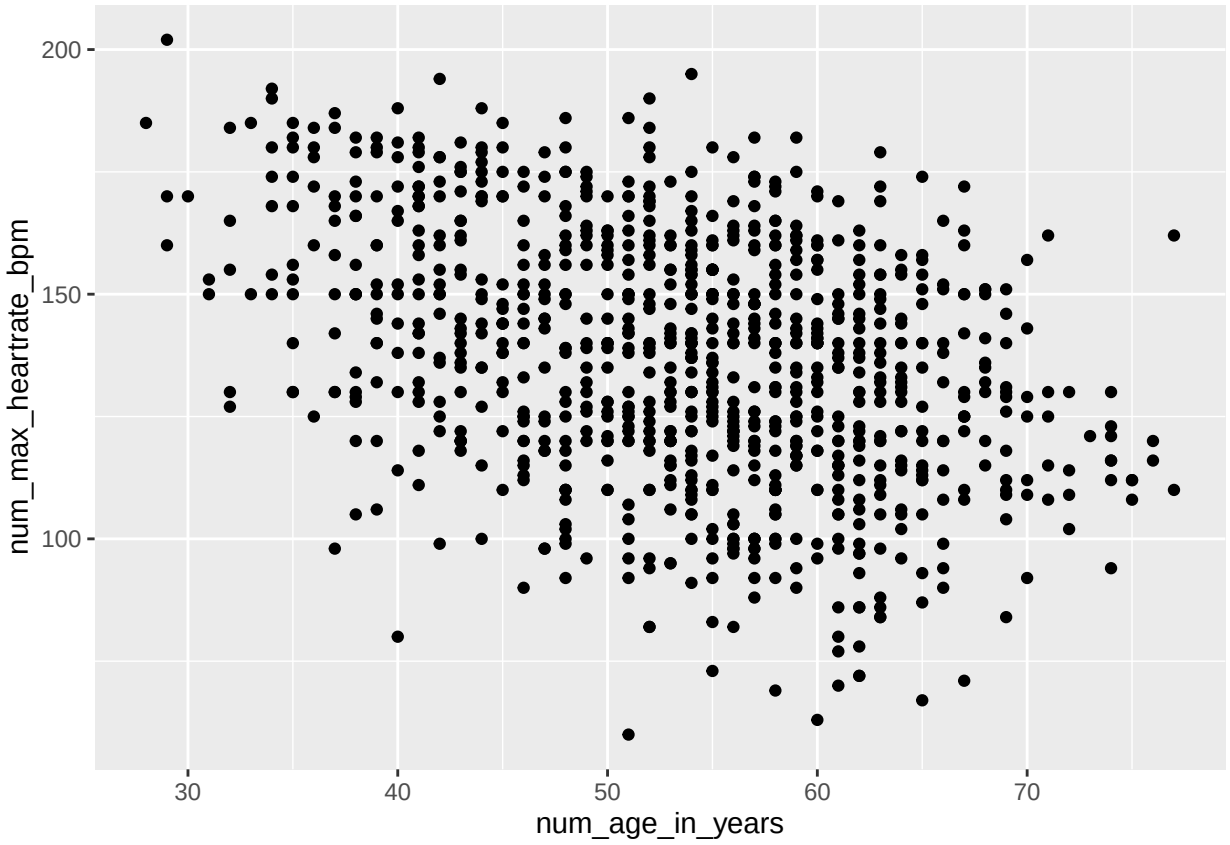
Folding multiple dimensions into a 2D scatterplot

By far the most common visualizations are all in 2D, i.e. they go on a page or a screen, etc. At this stage, we will not depend on animations and such fancy things to portray additional dimensions - usually this degree of complexity is not necessary. For now, we will use colour coding or symbol shape coding, and other simpler techniques to visualize extra dimensions down in a static 2D figure.

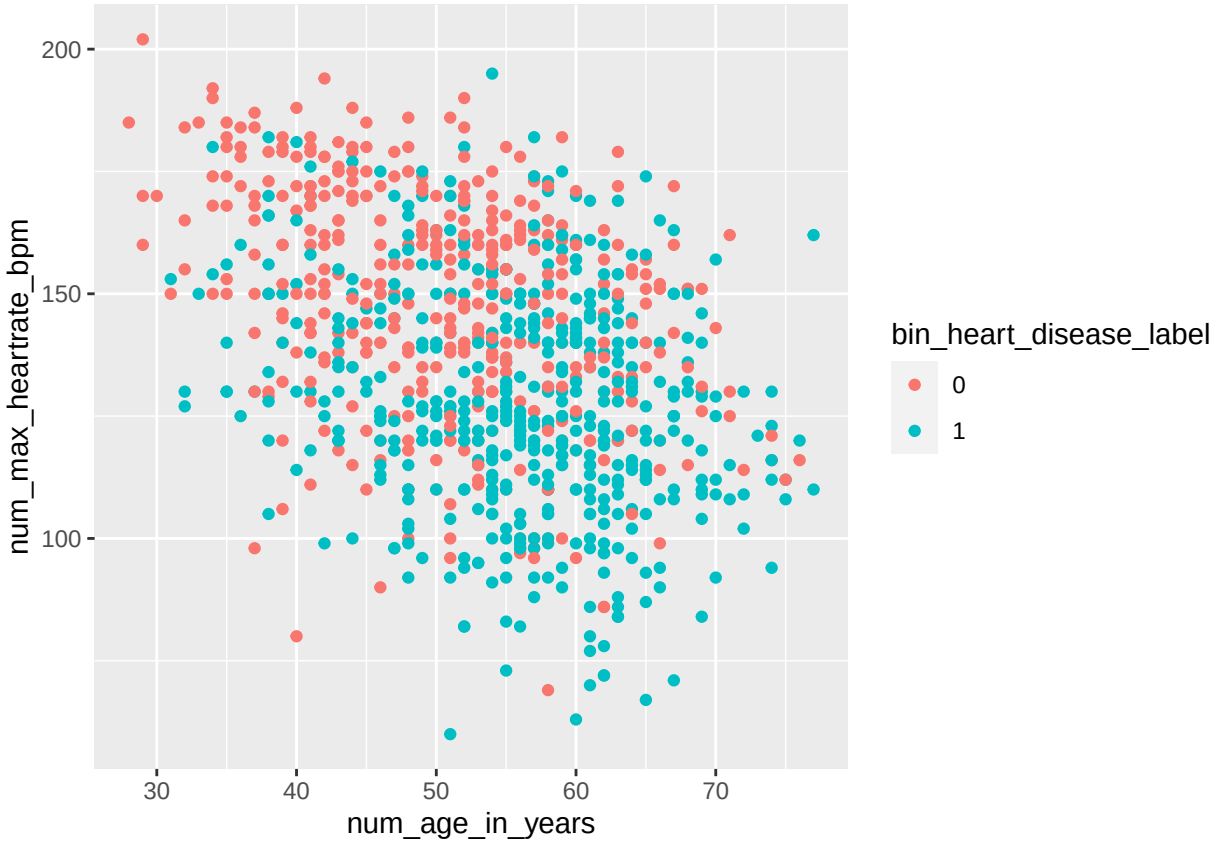
The key steps are as follows, first plot your continuous variables as a plain 2D scatter plot. Later on, add groups and subgroups as colour coding or symbol coding. The advice is to get your principal plot down first, and then add things to it one at a time.

The following have been taken from the R Graphics Cookbook, and then modified to suit. The dataset is the recoded heart failure dataset.

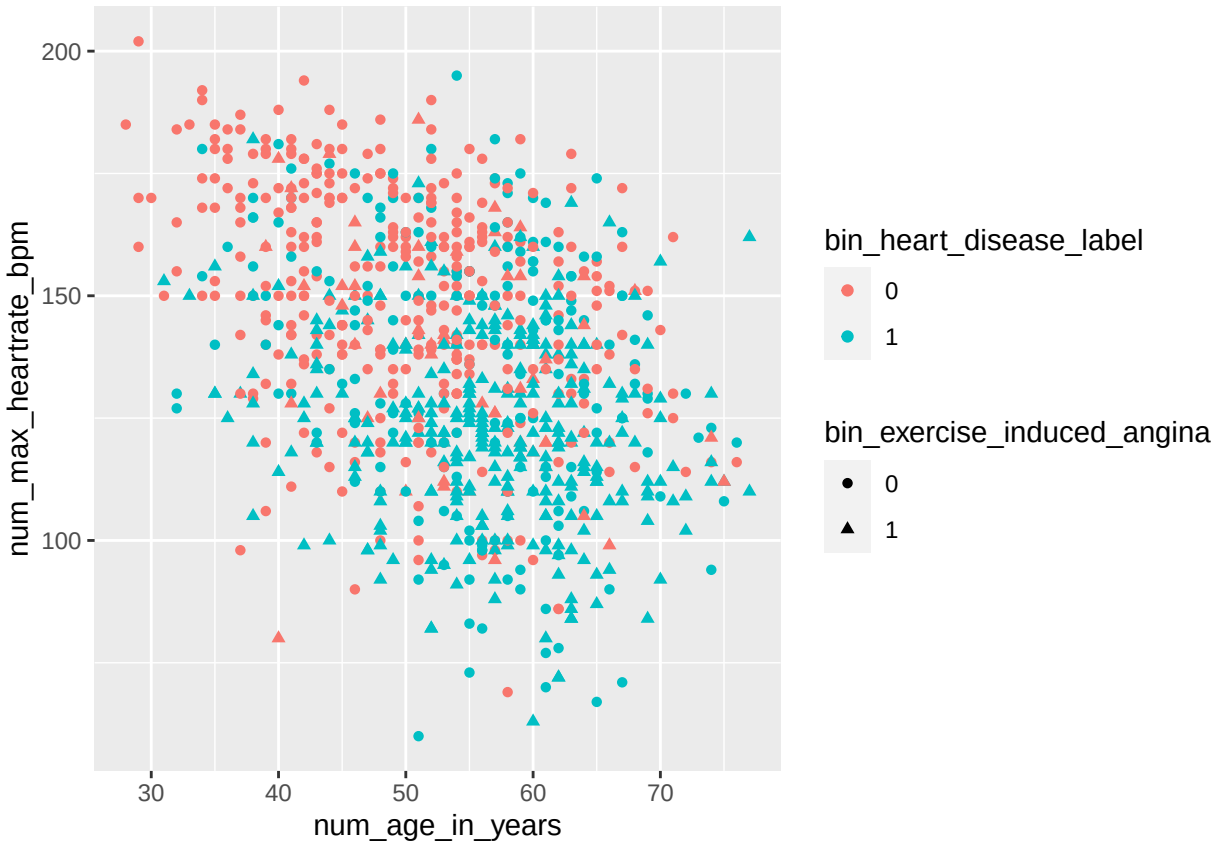
```
ggplot(recodedHeartFailure,
       aes(x=num_age_in_years,
           y=num_max_heartrate_bpm)) +
  geom_point()
```




```
ggplot(recodedHeartFailure,
       aes(x=num_age_in_years,
           y=num_max_heartrate_bpm,
           colour = bin_heart_disease_label)) +
  geom_point()
```

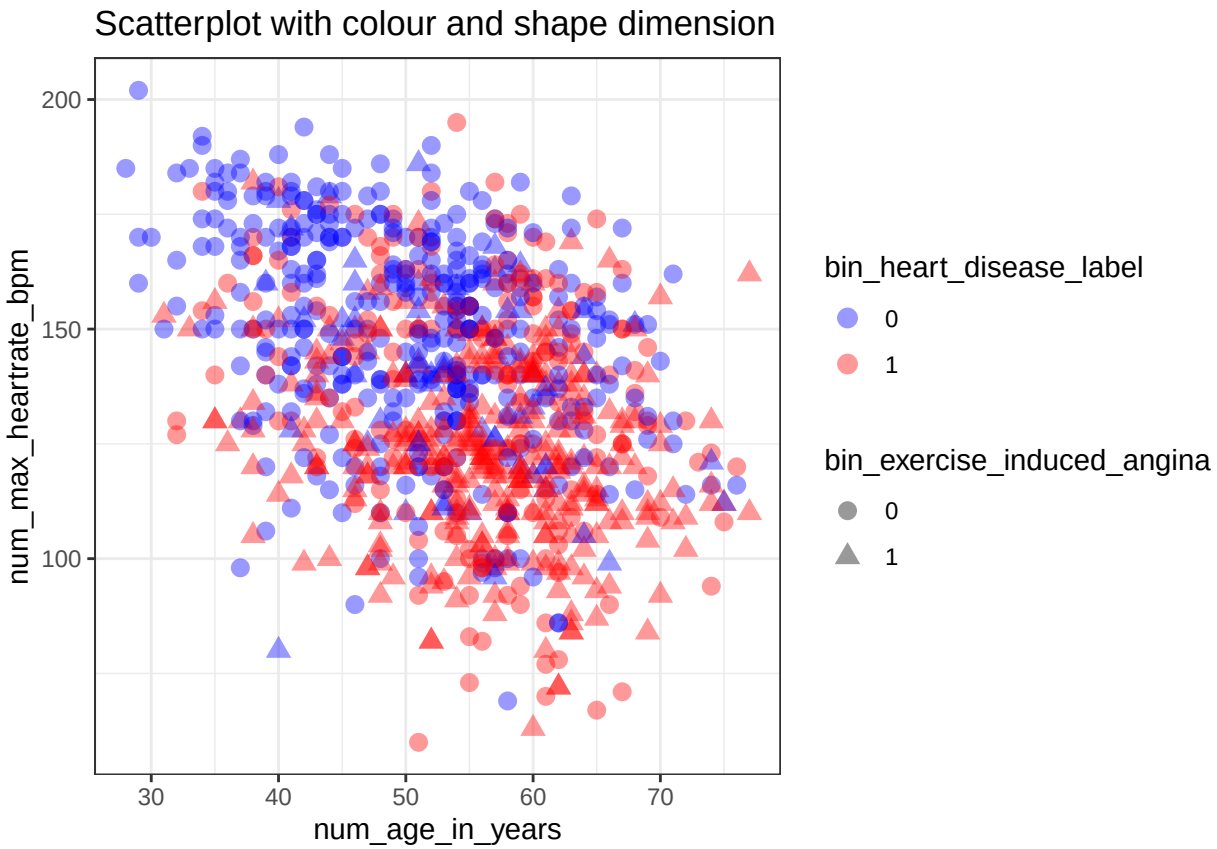



```
ggplot(recodedHeartFailure,
       aes(x=num_age_in_years,
           y=num_max_heartrate_bpm,
           colour = bin_heart_disease_label,
           shape = bin_exercise_induced_angina)) +
  geom_point()
```



Now I think I see a pattern by my own eyes, but I am not sure. I am going to see if I can highlight it more clearly by the following - I want to set my own colours, then I want to control the symbols, then I want to change the schema to be a bit more publication friendly and also add a title. Lastly, I try to deal a typical visualization issue called "overplottting" where data points fall on top of each other. Let's see how we can build this up bit by bit.

```
ggplot(recodedHeartFailure,
       aes(x=num_age_in_years,
           y=num_max_heartrate_bpm,
           colour = bin_heart_disease_label,
           shape = bin_exercise_induced_angina)) +
  geom_point(size = 3, alpha = 0.4) +
  scale_shape_manual(values=c(16, 17)) +
  scale_colour_manual(values=c("blue","red")) +
  ggtitle("Scatterplot with colour and shape dimension") +
  theme_bw()
```



Special case of a survival model with risk groups

We will only touch on this very briefly here, because you would be familiar with time-to-event types of curves from elsewhere. The purpose here is not to show how to compute risk groups etc but simply how to visualize them when needed. This is particularly geared towards those working in clinical research since you will encounter time to event data quite often.

```
## Loading required package: survival
```

```
## Loading required package: survivalAnalysis
```

```
## Loading required package: survminer
```

```
##
## Attaching package: 'survminer'
```

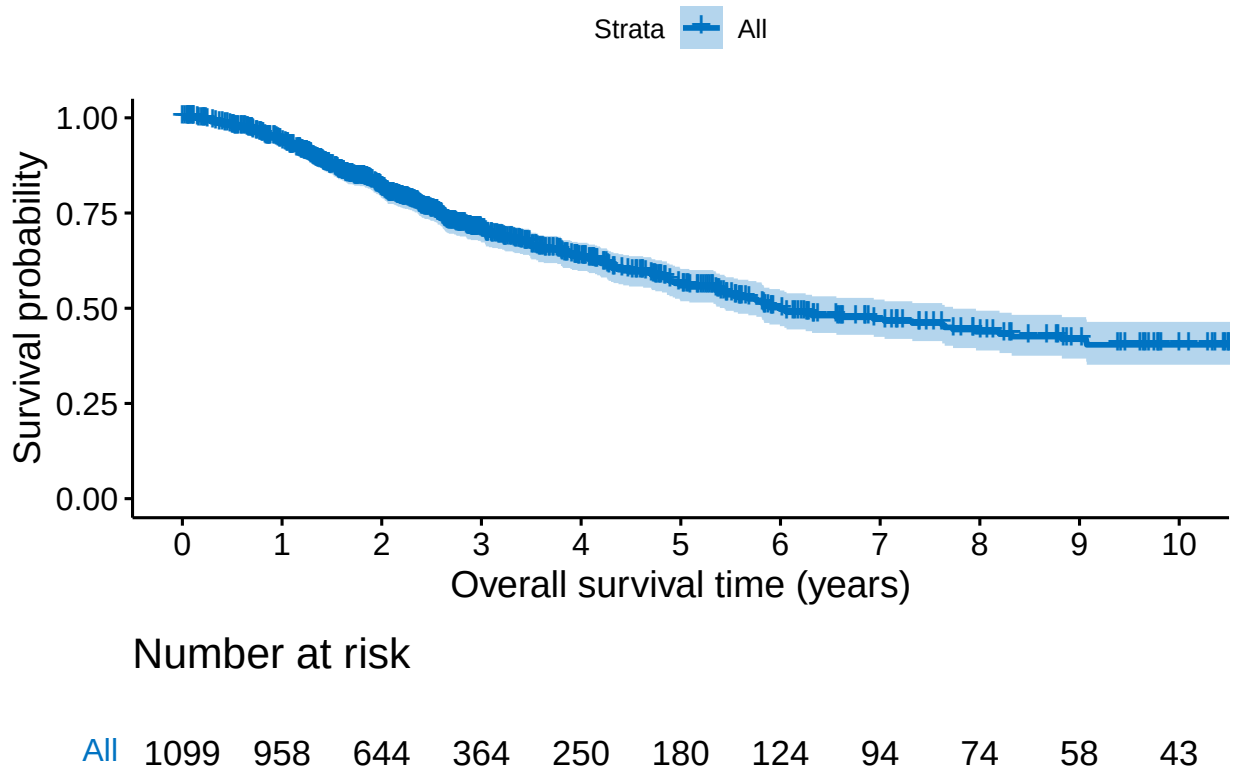
```
## The following object is masked from 'package:survival':
##
##     myeloma
```

```
## [1] "survival"         "survivalAnalysis" "survminer"
```

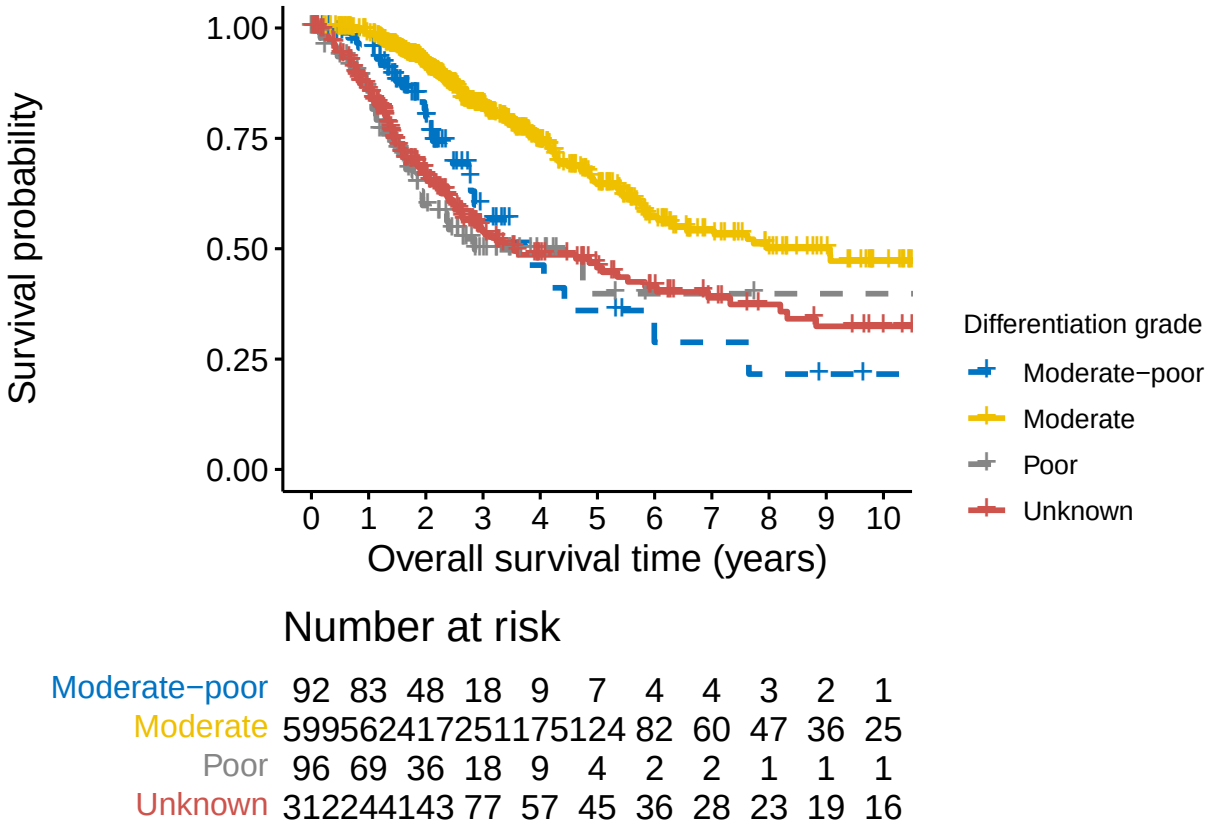
```r
head(recoded_crc_clinical)
```

```
##   id_patient num_age_at_diagnosis bin_sex_is_male bin_vital_status
## 1  P-0000119                  67               0                1
## 2  P-0000241                  49               0                1
## 3  P-0000493                  24               1                1
## 4  P-0000511                  39               0                1
## 5  P-0000520                  64               1                0
## 6  P-0000526                  24               0                0
##   num_survival_months fac_differentiation_grade num_survival_years
## 1               27.37                         M           2.280833
## 2               24.00                        MP           2.000000
## 3               23.87                         U           1.989167
## 4               23.27                         P           1.939167
## 5               41.50                        MP           3.458333
## 6               85.97                         M           7.164167
```

```r
ggsurvplot(
  survfit(Surv(num_survival_years, bin_vital_status) ~ 1, data = recoded_crc_clinical),
  data = recoded_crc_clinical,
  censor = T, #show right-censored events as symbol
  tables.theme = theme_cleantable(),
  risk.table = TRUE,
  palette = "jco",
  xlab = "Overall survival time (years)",
  xlim = c(0,10),
  break.time.by = 1,
  pval = F
)
```




```
ggsurvplot(
  survfit(Surv(num_survival_years, bin_vital_status) ~ fac_differentiation_grade, data = recoded_crc_cli
  data = recoded_crc_clinical,
  censor = T, #show right-censored events as symbol
  linetype = c("dashed","solid","dashed","solid"),
  tables.theme = theme_cleantable(),
  risk.table = TRUE,
  risk.table.height = 0.3,
  palette = "jco",
  xlab = "Overall survival time (years)",
  xlim = c(0,10),
  break.time.by = 1,
  pval = F,
  legend = "right",
  legend.title = "Differentiation grade",
  legend.labs = c("Moderate-poor","Moderate","Poor","Unknown")
)
```



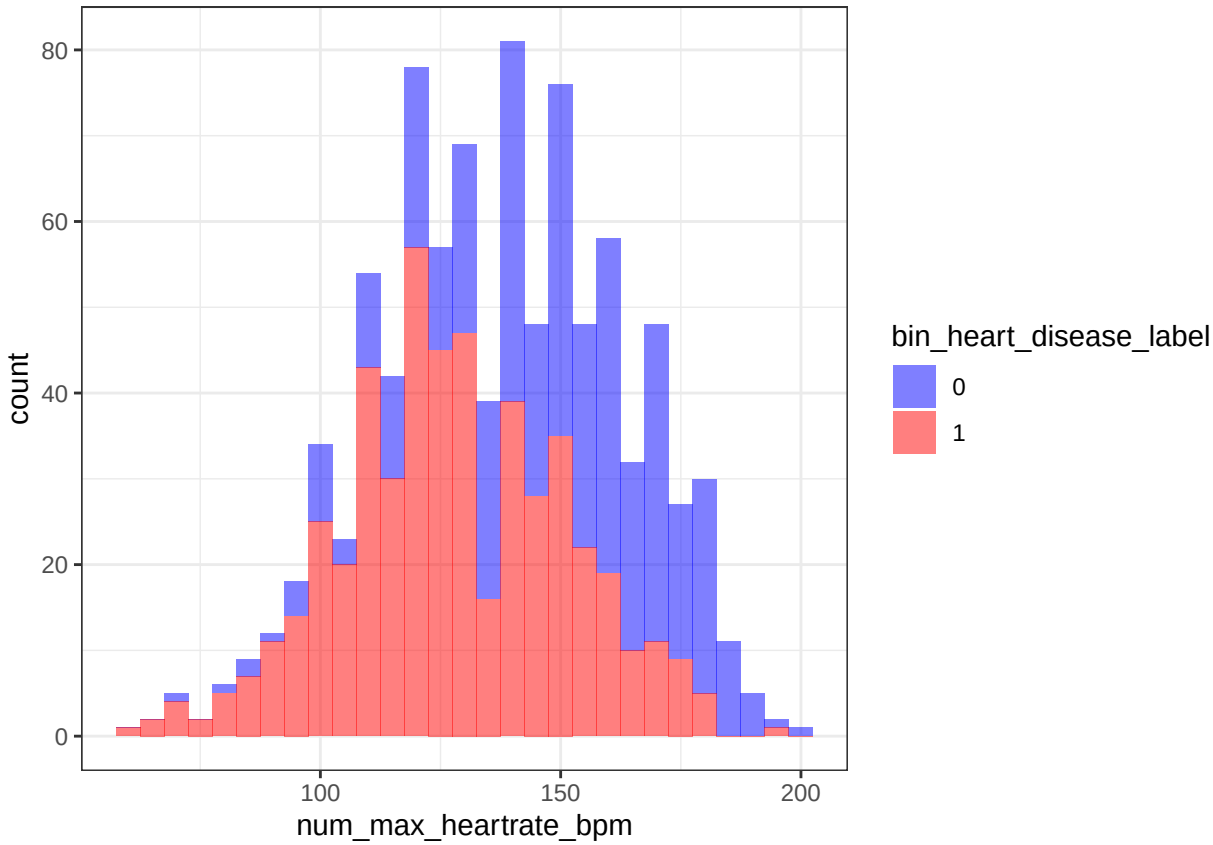
Distributions of continuous variables where the grouping is principal

In the previous section, we were working from the point of view that the relationship between numerical variables was of principal interest, and the groupings might help to explain the observed patterns. Now we flip the picture a little, and consider that the categorical factors are principal, and contiguous distributions within groups might help explain what is going on.

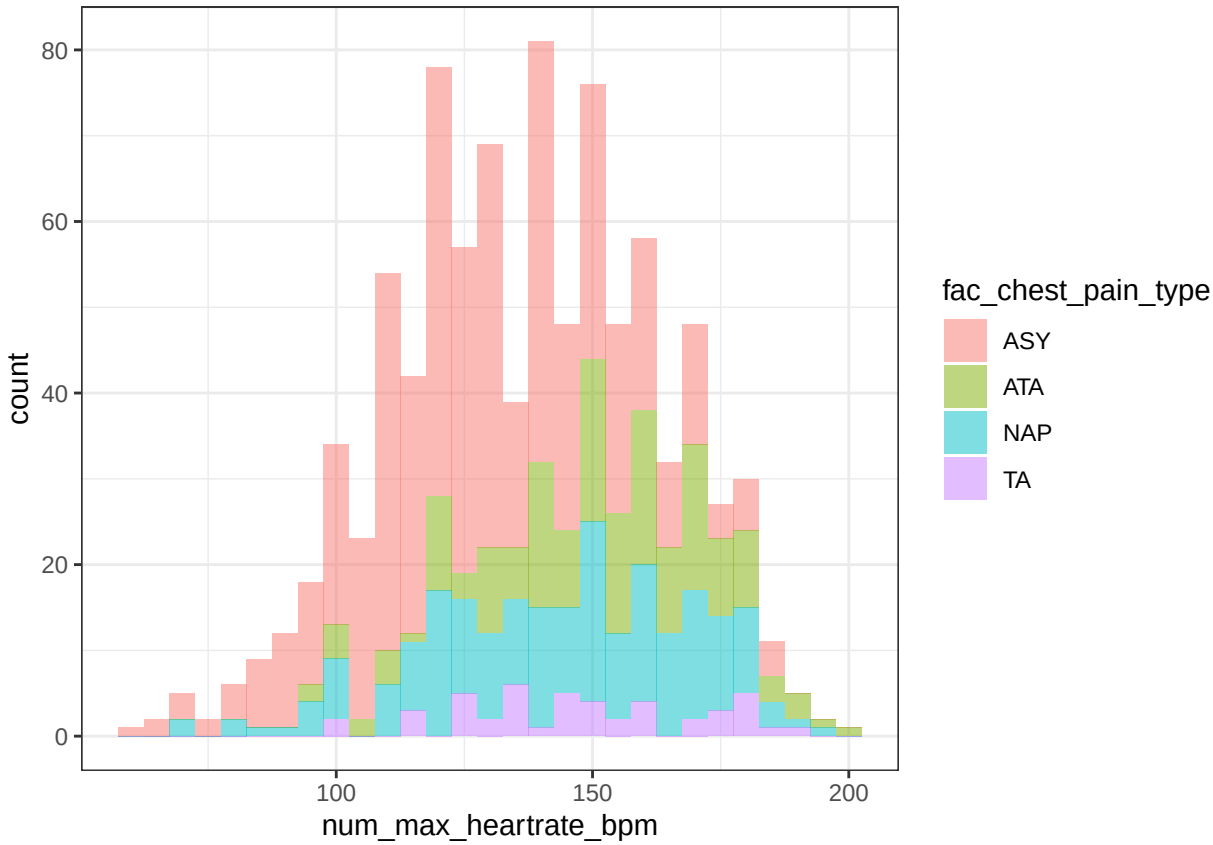
Histogram sets

Let's look at the heartrate distribution among those who do and do not have the heart disease diagnosis, using the **recodedHeartFailure** dataset. However be wary of plotting too many categories on the same axes, and there is also potential for confusion if you try to fit too many variables in.

```
ggplot(recodedHeartFailure,
       aes(x=num_max_heartrate_bpm,
           fill = bin_heart_disease_label)) +
  geom_histogram(binwidth = 5, alpha = 0.5) +
  scale_fill_manual(values=c("blue","red")) +
  theme_bw()
```



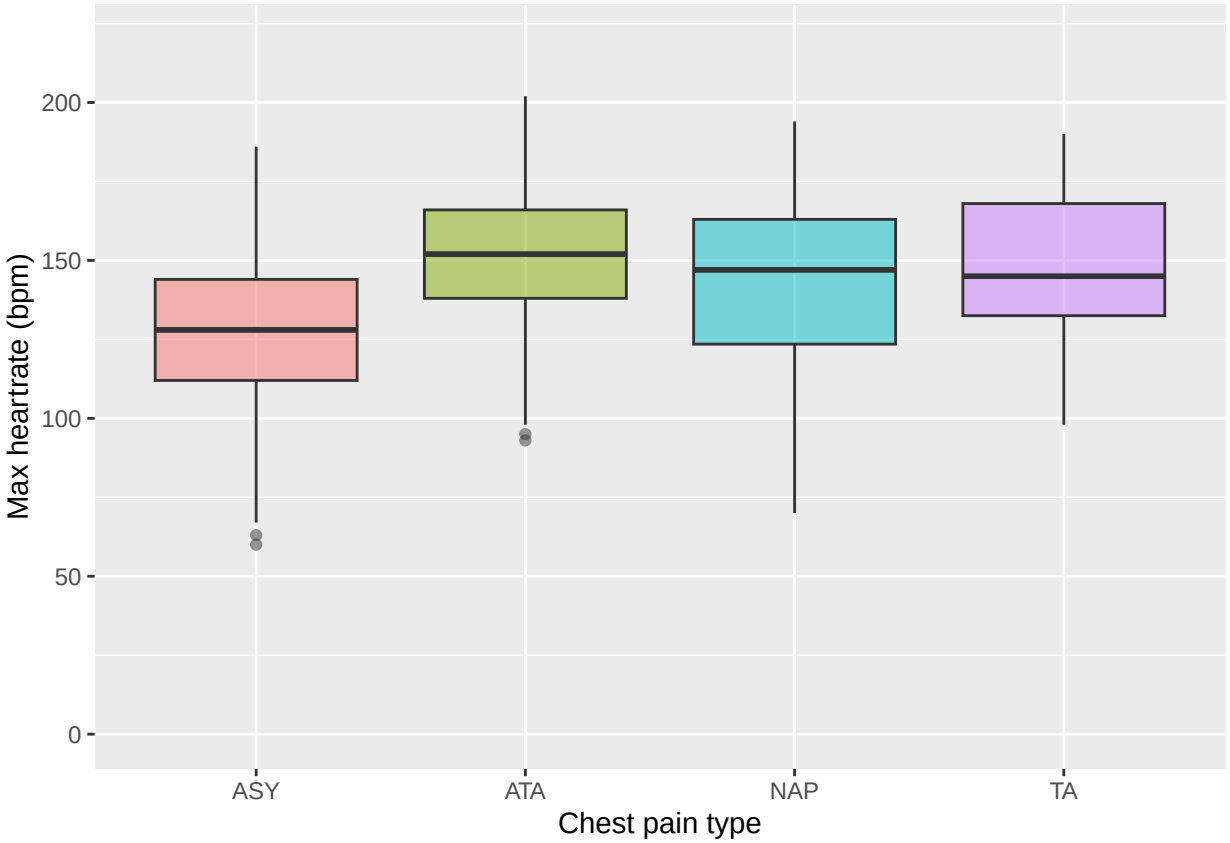

```
ggplot(recodedHeartFailure,
       aes(x=num_max_heartrate_bpm,
           fill = fac_chest_pain_type)) +
  geom_histogram(binwidth = 5, alpha = 0.5) +
  theme_bw()
```



Box-whisker sets

A box-and-whisker plot is useful if you simply want to see the broad descriptive statistics e.g. median, interquartile range, etc. without having too much distracting detail of the histogram itself. A box-whisker is potentially more useful for multi-category variables (whereas you may find the overlapped histograms more easy to digest for binary labels). Here is the same information as the previous, but presented as a box-whisker graphic.

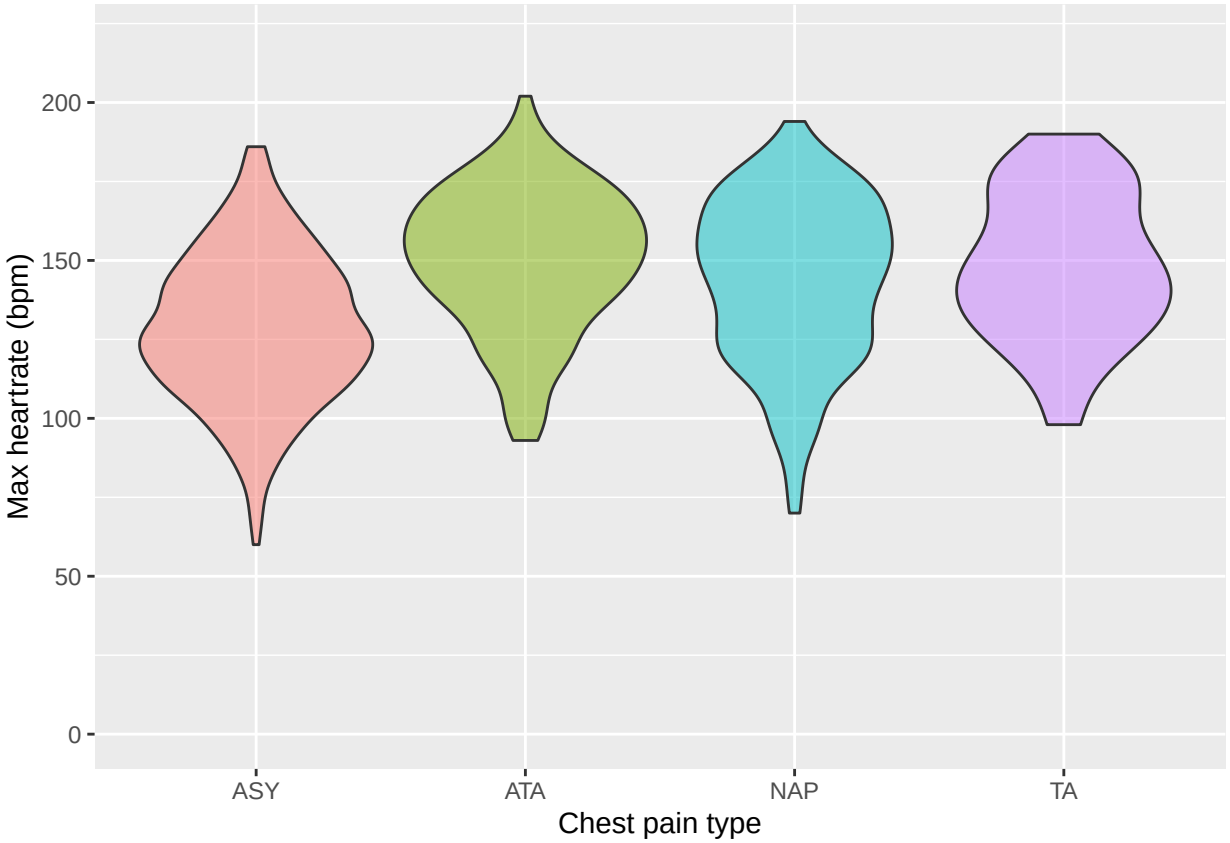
```
ggplot(recodedHeartFailure,
       aes(x = fac_chest_pain_type,
           y = num_max_heartrate_bpm,
           fill = fac_chest_pain_type)) +
  geom_boxplot(alpha = 0.5) +
  xlab("Chest pain type") +
  ylab("Max heartrate (bpm)") +
  ylim(0,220) +
  theme(legend.position = "none")
```



Violin plots

There does happen to be a way if you want to see both the distribution within each groups AND the groups themselves for two or more categorical levels. A violin plot (so-called because of the shapes it tends to generate) might be a way to do this. Note that a violin tries to "smooth" out all the bumps and noise that us present in your histrogram, so beware that you do not get fooled into thinking data exists when it actually is just visually smooth.

```
ggplot(recodedHeartFailure,
       aes(x = fac_chest_pain_type,
           y = num_max_heartrate_bpm,
           fill = fac_chest_pain_type)) +
  geom_violin(alpha = 0.5) +
  xlab("Chest pain type") +
  ylab("Max heartrate (bpm)") +
  ylim(0,220) +
  theme(legend.position = "none")
```



Ridgelines

When you have quite a large sample size and multiple subgroups to plot, the width of your box-whisker or violin plots might become unmanageable. Typically four to maximum six categories is about as much as you can visually fit into the box-whisker or violin. When there are more categories, you may wish to try ridgelines. To get a high sample size here, I am going to use the **colorectal cancer data**.

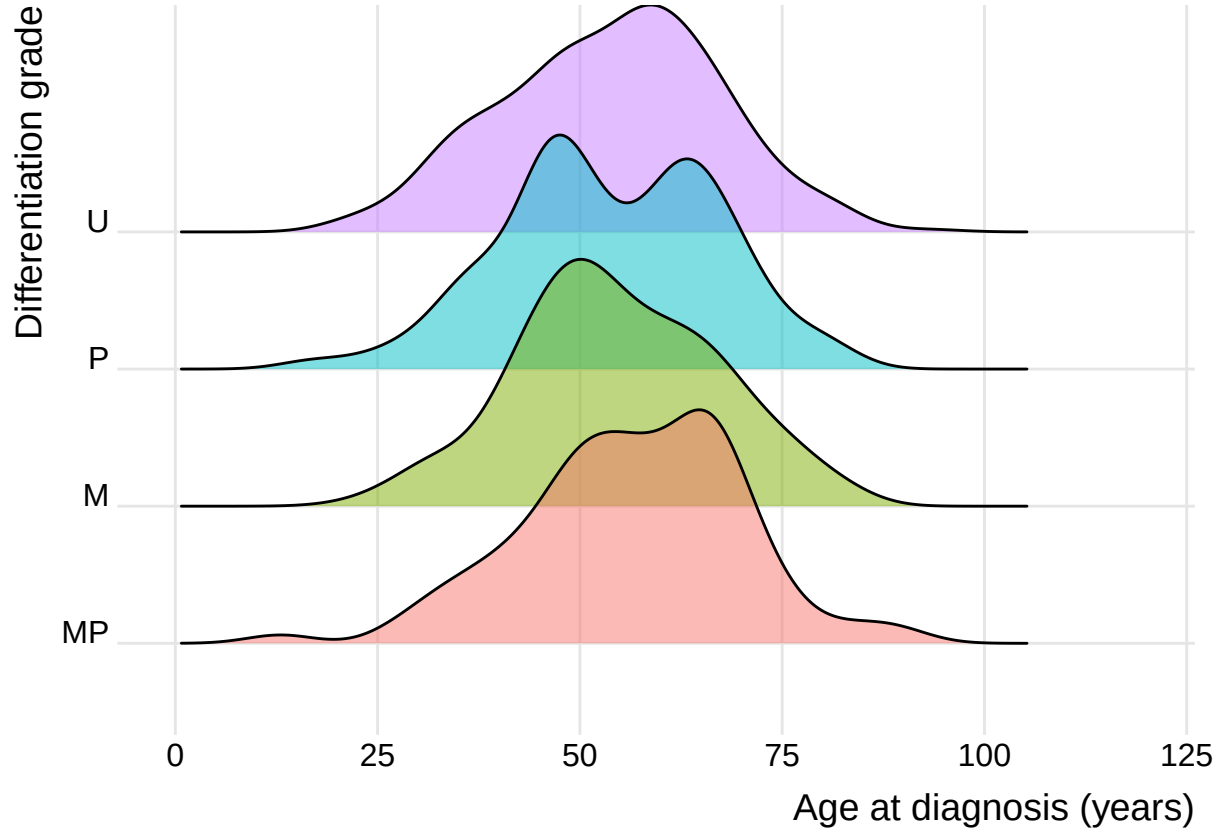
```
installRequiredPackages(c("ggridges"))
```

```
## Loading required package: ggridges
```

```
## [1] "ggridges"
```

```
ggplot(recoded_crc_clinical,
       aes(x = num_age_at_diagnosis,
           y = fac_differentiation_grade,
           fill = fac_differentiation_grade)) +
  geom_density_ridges(alpha = 0.5) +
  theme_ridges() +
  xlab("Age at diagnosis (years)") +
  ylab("Differentiation grade") +
  xlim(0,120) +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 4.09
```



Time-correlated and spatially (geographically) correlated data

Information direct from world wide web

With this kind of data, we typically pick up the numerical / epidemiological data separately from the geographic data i.e. the map boundaries of countries, regions, etc. The latter is easily found in many places on the web, you need some R libraries specifically for getting these geographic outlines. We have seen a previous example of grabbing data directly from webservers previously.

```
urlData <- 'https://api.coronavirus.data.gov.uk/v2/data?areaType=utla&metric=newDeaths28DaysByPublishDa

#load the data from the web source
coronavirusData = read.csv(file=urlData)

#define geographic data endpoint
ons_api <- "https://opendata.arcgis.com/datasets/687f346f5023410ba86615655ff33ca9_4.geojson"
```

Here is what a tiny snippet of the coronavirus data looks like.

```
head(coronavirusData)
```

```
##    areaCode                areaName areaType       date newCasesByPublishDate
```

```
## 1 E06000003      Redcar and Cleveland     utla 2023-02-02                  98
## 2 E06000014                      York     utla 2023-02-02                  70
## 3 E06000050 Cheshire West and Chester     utla 2023-02-02                 108
## 4 E08000001                    Bolton     utla 2023-02-02                  90
## 5 E08000016                  Barnsley     utla 2023-02-02                 125
## 6 E08000031             Wolverhampton     utla 2023-02-02                  66
##   newDeaths28DaysByPublishDate
## 1                            4
## 2                            4
## 3                            3
## 4                            4
## 5                            3
## 6                            3
```

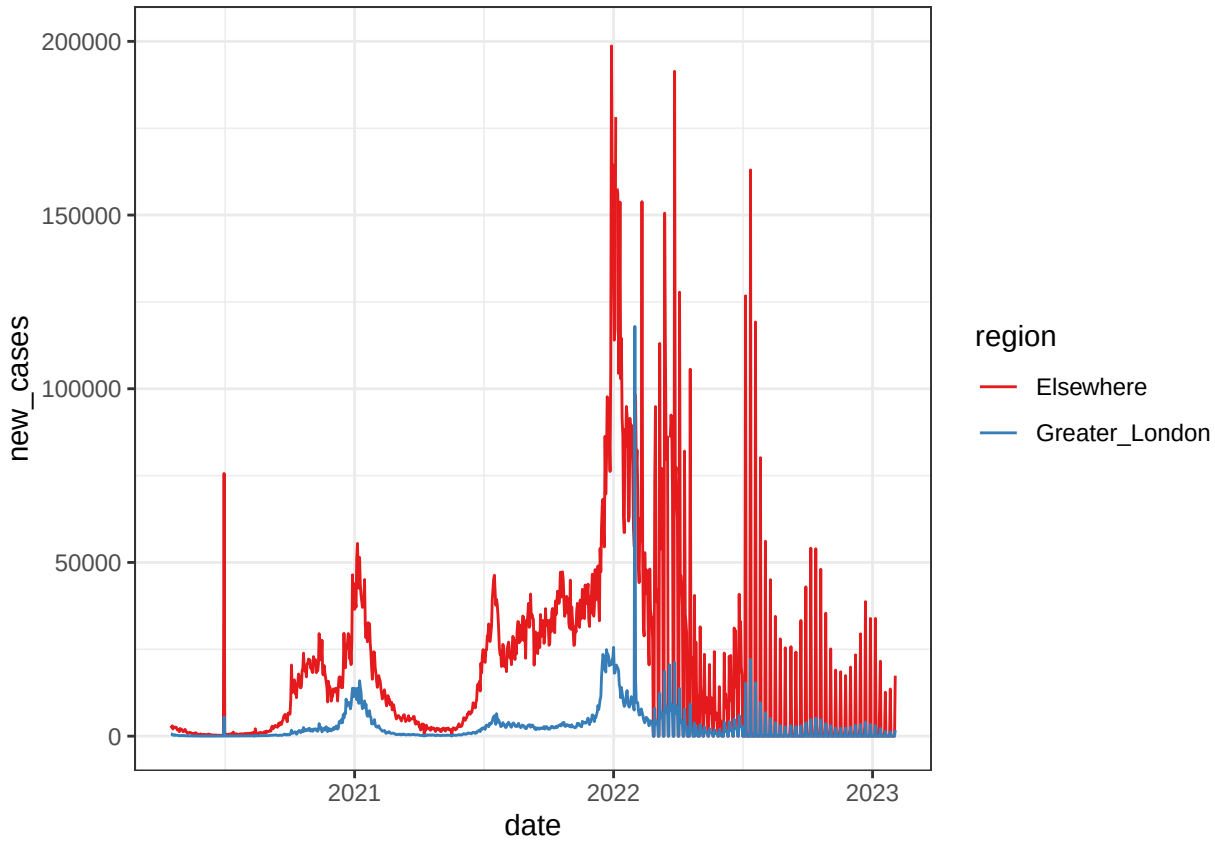
Time series

Note that complex transformations are not taught in this course, but you will find that your data exploratory and visualization power amplifies tremendously if you simply learn a couple of very basic ***data transformation*** tricks - such as aggregation and merging. These are detailed in the recommended textbooks. It is not mandatory to learn, but you will find your abilities grow tremendously if you can learn how to do simple merging and simple aggregations.

The code for creating the time series data is in the markdown file not but in the knitted output, you can refer to that to learn by example.

```
## [1] "lubridate"
```

```
ggplot(timeData, aes(x=date, y=new_cases, colour=region)) +
  geom_line() +
  scale_colour_brewer(palette="Set1") +
  theme_bw() +
  ylim(0,200000)
```

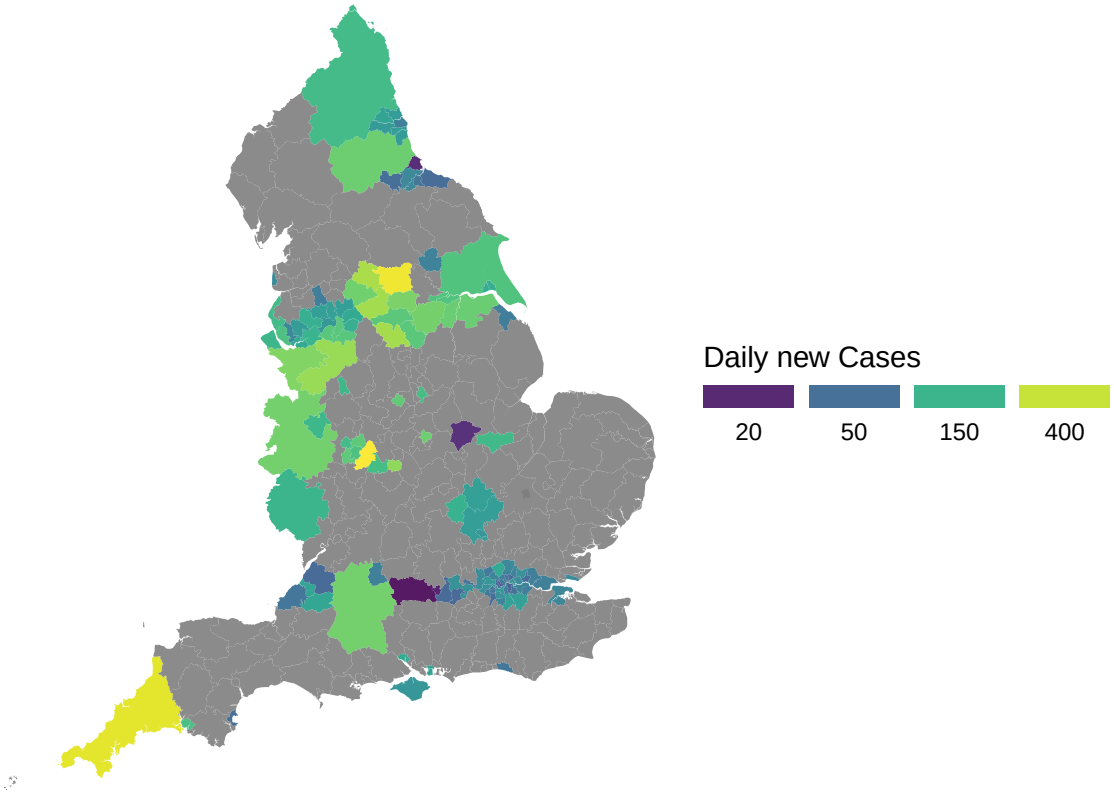


Choropleths

A choropleth is where we **merge** geographic data to epidemiological data, but the most essential ingredient is there has to be a common key between the two. In this example I use the unique Area Code identifier to map the coronovirus data of each local authority (somewhat like our own concept of commune/gemeente) to its respective geographical boundaries.

```
## [1] "sf"        "geojsonio" "mapproj"   "broom"     "viridis"
```

```
#now use this function to call up the choro at the time point that I see corresponds to one of the peak
singleFrameChoro("2021-10-01")
```

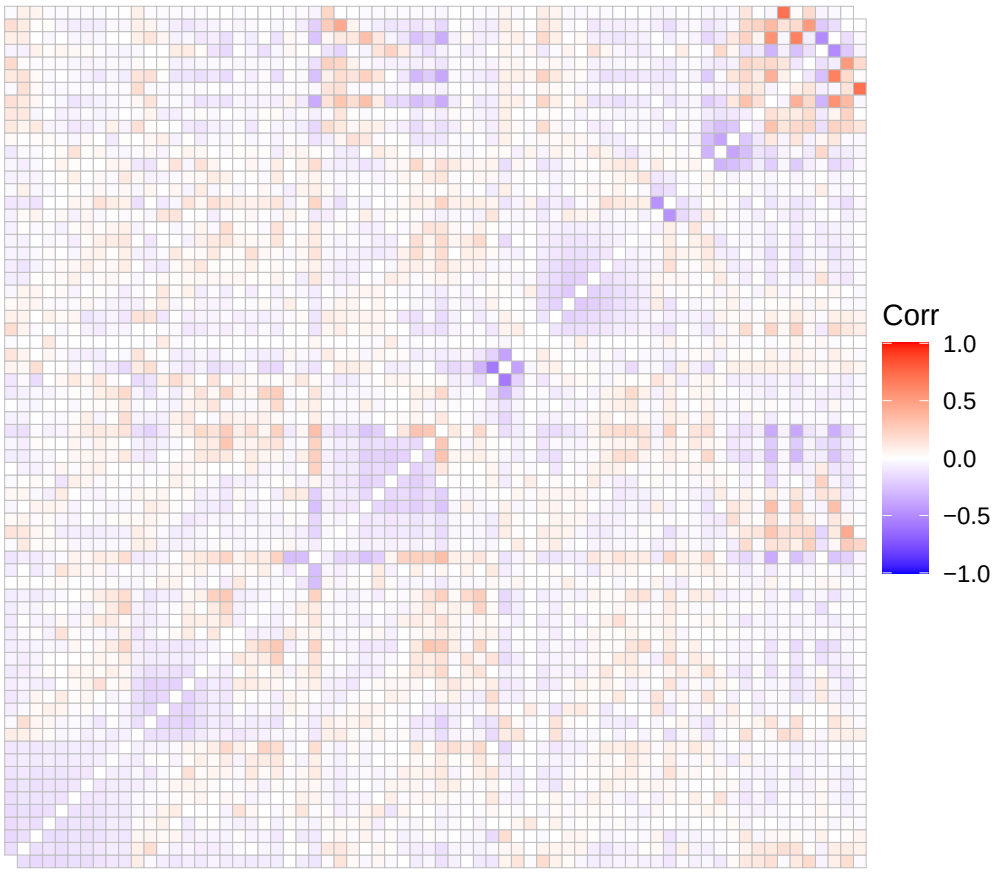


Relationships between categorical variables

```
installRequiredPackages(c("ggcorrplot"))
```

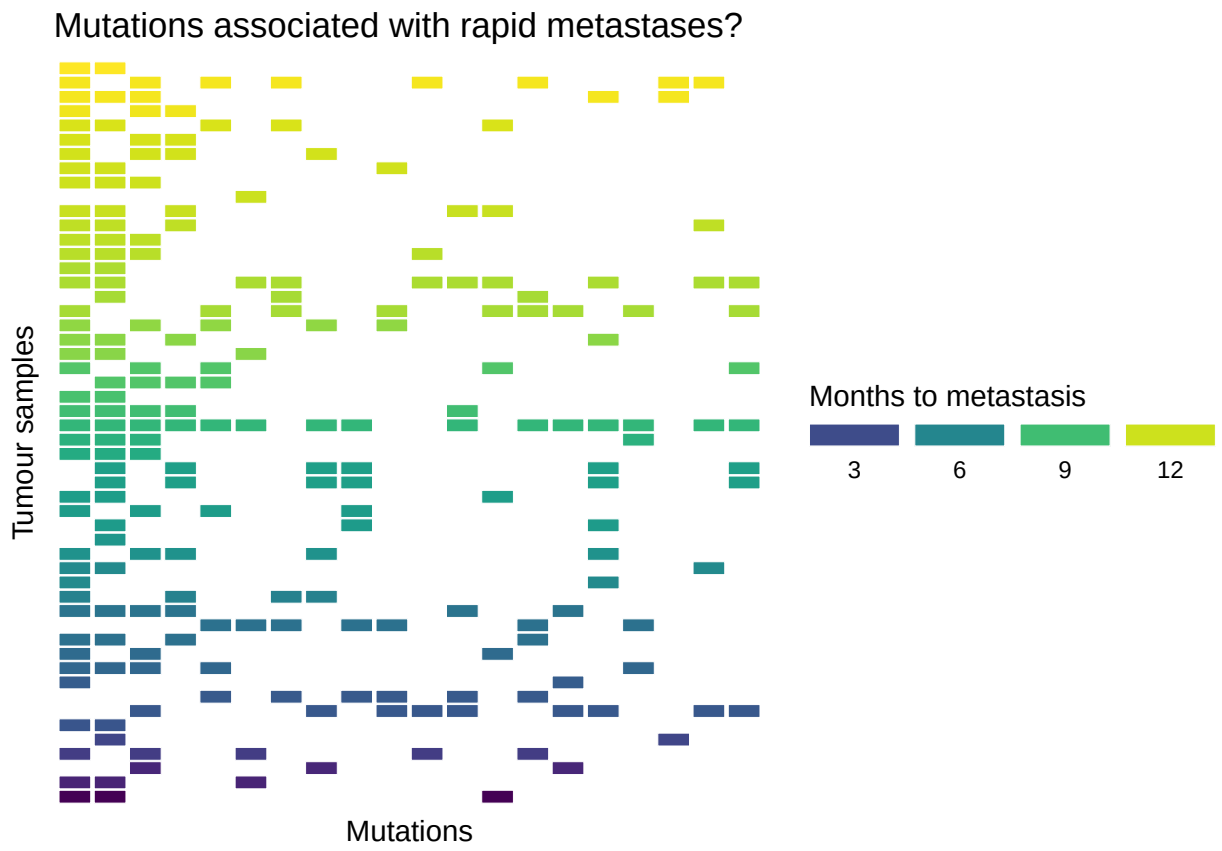
```
## [1] "ggcorrplot"
```

```
model.matrix(~0+., data=scoresOnlyChronic) %>%
  cor(use="pairwise.complete.obs") %>%
  ggcorrplot(show.diag = F, method="square") +
  theme_void()
```



Heatmaps

```
ggplot(mutations,
       aes(x = Hugo_Symbol, y = Tumor_Sample_Barcode, fill = TIME_TO_METASTASIS_MONTHS)) +
  geom_tile(colour="white", width=.9, height=.9) +
  scale_x_discrete(limits=top20Mutations) +
  scale_y_discrete(limits=sampleOrder) +
  theme(panel.background = element_blank()) +
  xlab("Mutations") +
  ylab("Tumour samples") +
  ggtitle("Mutations associated with rapid metastases?") +
  scale_fill_viridis(breaks = c(3,6,9,12),name="Months to metastasis",
                     guide = guide_legend(
                       keyheight = unit(3, units = "mm"),
                       keywidth=unit(12, units = "mm"),
                       label.position = "bottom",
                       title.position = 'top', nrow=1) ) +
  theme(axis.text.x=element_blank(), #remove x axis labels
        axis.ticks.x=element_blank(), #remove x axis ticks
        axis.text.y=element_blank(),  #remove y axis labels
        axis.ticks.y=element_blank()  #remove y axis ticks
        )
```



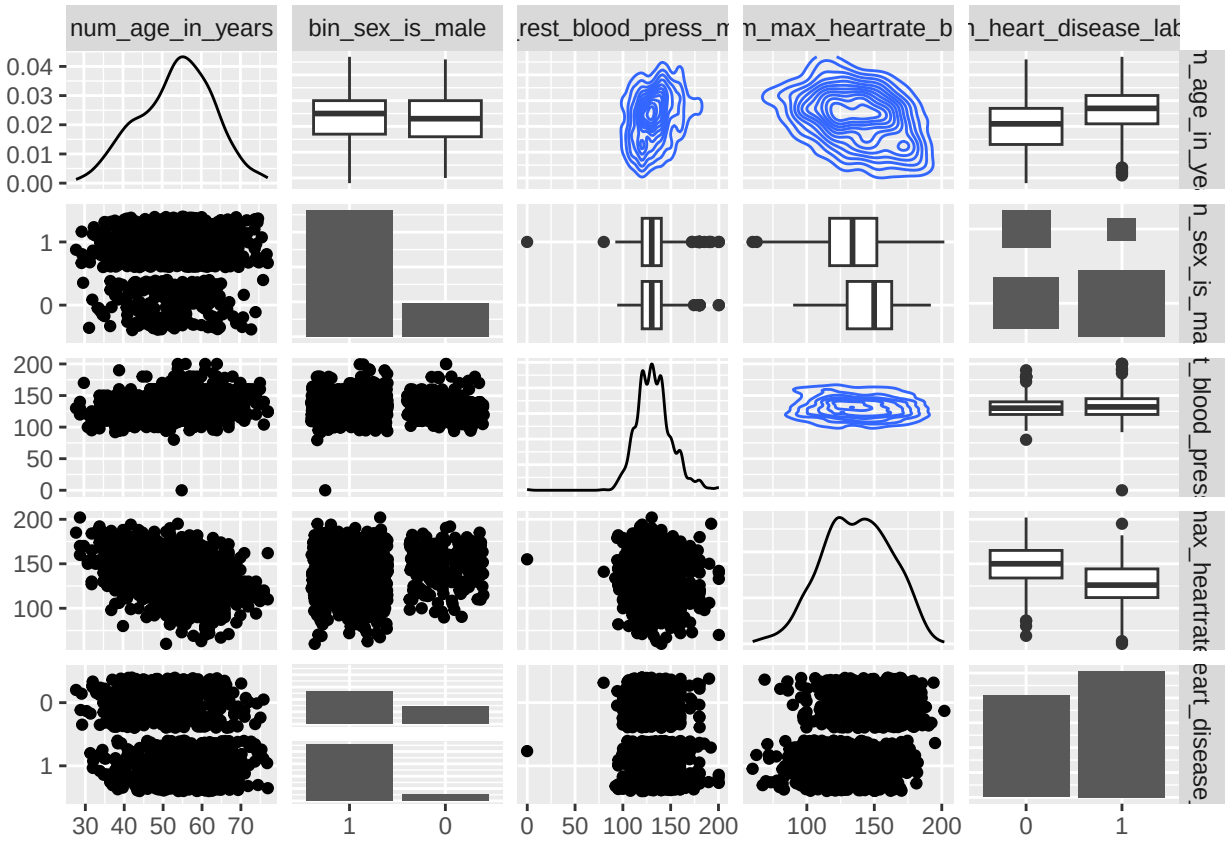
Correlation matrices

From a previous example, we have already seen how to make a fast overview of co-variations between variables using the **ggpairs** package.

```
installRequiredPackages(
  c("ggplot2", "GGally")
) #I realize I need to ensure a couple of packages are available so I re-use a previous function
```

```
## [1] "ggplot2" "GGally"
```

```
subsetHeartFailure <- dplyr::select(
  recodedHeartFailure,
  num_age_in_years,
  bin_sex_is_male,
  num_rest_blood_press_mmHg,
  num_max_heartrate_bpm,
  bin_heart_disease_label
)

ggpairs(
  subsetHeartFailure,
  upper = list(continuous = "density", combo = "box_no_facet"),
  lower = list(continuous = "points", combo = "dot_no_facet")
)
```

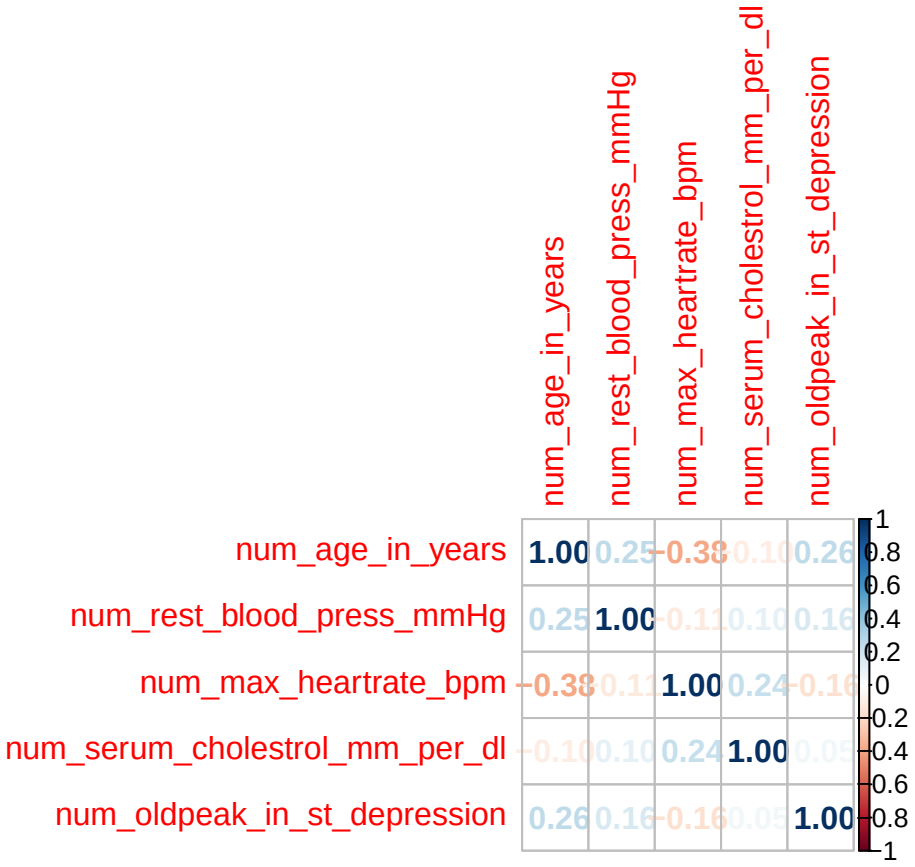


Related to this are various types of output available from the (possibly more simplified) **corr** package. This is mainly geared toward calculating and displaying either linear or rank correlations between variables, therefore there are going to be problems with categorical variables. Here are some examples of how it may be done using a subset of the same recoded heart failure data.

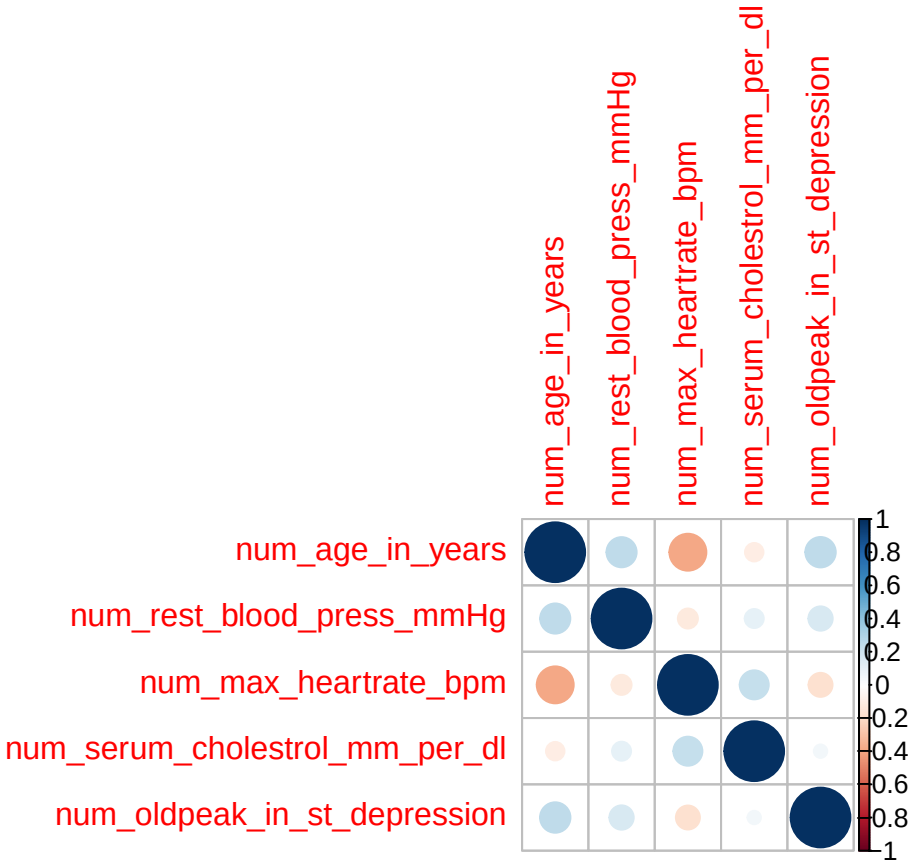
```
installRequiredPackages(c("corrplot"))
```

```
## [1] "corrplot"
```

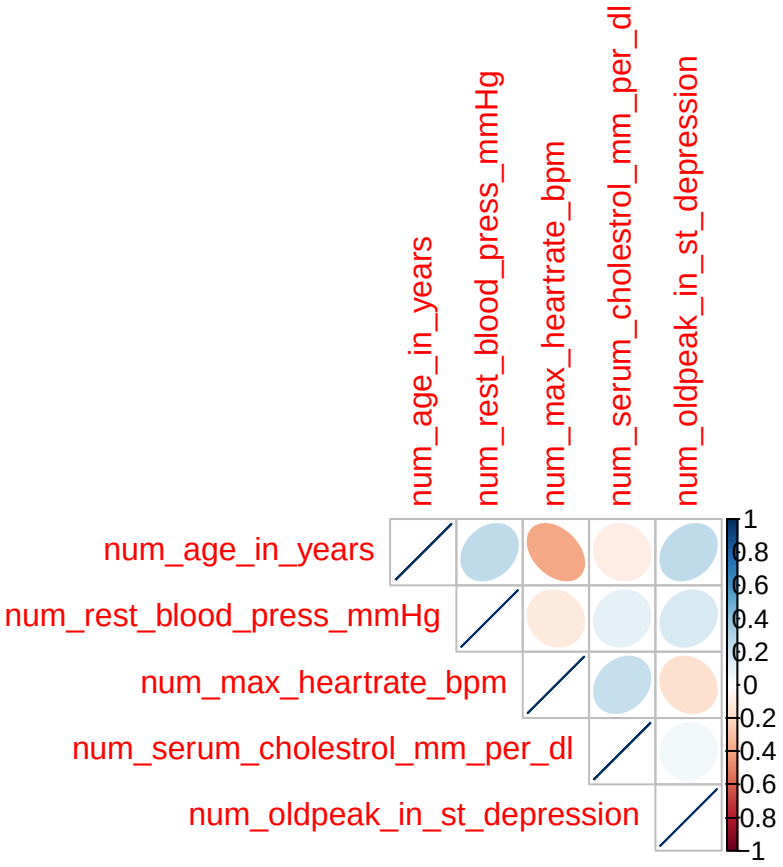
```
subsetHeartFailure <- dplyr::select(recodedHeartFailure,
                                    num_age_in_years, num_rest_blood_press_mmHg, num_max_heartrate_bpm,
                                    num_serum_cholestrol_mm_per_dl,num_oldpeak_in_st_depression)

corr_matrix <- cor(subsetHeartFailure)

corrplot(corr_matrix, method = "number")
```




```
corrplot(corr_matrix, method = "circle")
```




```
corrplot(corr_matrix, method = "ellipse", type="upper")
```


num_age_in_years
num_rest_blood_press_mmHg
num_max_heartrate_bpm
num_serum_cholestrol_mm_per_dl
num_oldpeak_in_st_depression
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1