LeaRning Week 5: PCAs and Heatmaps

Nelly Amenyogbe

12/08/2021

```
##
##  -----
## Look mom - my first PCA!
##  ------
##     \          ,'``.._   ,'``.
##      \        :,--._:)\,:,._,.:
##       \       :`--,''   :`...';\
##                `,'       `---'  `.
##                /                 :
##               /                   \
##             ,'                     :\.___,-.
##            `...,---'``````-..._    |:       \
##              (                 )   ;:    )   \  _,-.
##               `.              (   //          `'    \
##                :               `.//  )      )     , ;
##              ,-|`.            _,'/       )    ) ,' ,'
##             (  :`.`-..____..=:.-':     .     _,' ,'
##              `,'\ ``--....-)='    `._,  \  ,') _ '``._
##           _.-/ _ `.       (_)      /     )' ; / \ \`-.'
##          `--(   `-:`.     `' ___..'  _,-'   |/   `.)
##              `-. `.`.``-----``--,  .'
##                |/`.\`'        ,',');  SSt
##                    `         (/  (/
##
```

Let's load the packages that we will need for this week's lesson, then import our data.

```
library(plyr)
library(tidyverse)
library(ggfortify)
```

Today, we will move beyond bar graphs and get into exploratory data analysis - and the most popular ones that you've likely encountered are Principal Component Analysis (PCA) and heatmaps. This is also a good time to get into handling multivariate data - i.e. getting away from ggplot-friendly long data and reviewing some great applications for wide data.

```
 Let's start with PCA.
```

Principal Component Analysis

What is a PCA?

Hopefully, at some point, you will have access to a multi-dimensional data set with the task of finding exciting new patterns… pathway enrichment… new insights… the works! Or 5000 t-tests using the loop function we just learned. Either way, high-dimensional data is very exciting.

What kind of data is this? Really, any kind of data matrix where you have measured multiple rationally-related variables for the same samples. This could be flow cytometry, with many cell types. It could be plasma cytokines measured with a luminex multiplex machine. More typical examples are RNASeq data sets, or any types of OMIC measurements. The key is, same set of samples, and numerical outputs for all your dependent (measured) variables, in some kind of wide format.

Cluster analysis is a very useful way to begin exploring your data set. The reasons being, you are able to assess the behavior of all the data together at once, and see how your samples relate to each other given all the things you measured. There are many ways to perform cluster analysis, but one of the most popular is Principal Components Analysis (PCA).

As always, it is useful to understand the basics of what goes on behind the scenes of any statistical tools we apply. There is a large wealth of resources that do a nice job of introducing the theory behind PCA, and we will not delve deeply into this here, where to focus is on how to appropriately produce one. You are encouraged to read further. A few links below:

Theory of PCA: Coursera: https://www.coursera.org/learn/pca-machine-learning?action=enroll (long and involved)

PCA tools to use in R: PCAtools: https://bioconductor.org/packages/release/bioc/vignettes/PCAtools/inst/doc/PCAtools.html (Great examples with biological data)

Data Camp: https://www.datacamp.com/community/tutorials/pca-analysis-r (quick and dirty, good explanations. Uses ggbiplot, which fails in more current versions of R.)

mixOmics: http://mixomics.org/methods/pca/ (useful, as this suite of tools also transfers to many other types of analysis, including data integration)

In This Lesson

We will build our first PCA using the gapminder.csv data set used for our Week 4 homework. The gapminder data set includes sociodemographic data across 134 countries from 2012, including life expectancy in years, the population, and the GDP per capita. Our mission is to see if these variables can show us any useful trends, by continent, in a multivariate context.

```
library(here)
```

```
# load data
gap <- read.csv(here("LeaRning/gapminder_2012.csv"))
head(gap)
```

```
##       country continent year lifeExp      pop gdpPercap
## 1 Afghanistan      Asia 2012    57.2 30700000      1840
## 2     Albania    Europe 2012    77.0  2920000     10400
## 3     Algeria    Africa 2012    76.8 37600000     13200
## 4      Angola    Africa 2012    61.7 25100000      6000
## 5   Argentina  Americas 2012    76.1 42100000     19200
## 6   Australia   Oceania 2012    82.3 22800000     42600
```

This data set has a very simple structure. We have the country (the independent variable), annotated by the continect and year (our metadata), and our measured variables lifeExp, pop, and gdpPercap. Our first step towards a PCA so to convert this into two data sets: a data matrix that can be used for clustering, and a corresponding metadata file used for plotting aesthetics.

```
# Let's make sure the country variable is unique
length(unique(gap$country)) / length(gap$country)
```

```
## [1] 1
```

```
Great! All unique values. Now we know that we can assign our row names as the country names.  An importa
```

Create a data matrix

Data matrices differ from data frames in a few simple ways: 1. They must be assigned row names useful for your analysis 2. The columns must all be numeric Based on these criteria, we can create a matrix out of our data frame by selecting the columns we want, and putting our row names in place.

```
# Create the data matrix ####

# start by making a new version of our data
mat <- gap

# assign row names
rownames(mat) <- mat$country

colnames(mat)
```

```
## [1] "country"   "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
```

```
# Subset to the columns of interest
mat <- mat[,c("lifeExp", "pop", "gdpPercap")]

head(mat)
```

```
##             lifeExp      pop gdpPercap
## Afghanistan    57.2 30700000      1840
## Albania        77.0  2920000     10400
## Algeria        76.8 37600000     13200
## Angola         61.7 25100000      6000
## Argentina      76.1 42100000     19200
## Australia      82.3 22800000     42600
```

Let's check the class of our new object.

```
class(mat)
```

```
## [1] "data.frame"
```

We see that this is still a data frame. Now, the function we will use to get our PCA (prcomp) will accept an object of the class data frame. However, we can also easily convert this to a data matrix now that it is properly formatted.

```
# Convert to class matrix ####

mat <- as.matrix(mat)
class(mat)
```

```
## [1] "matrix" "array"
```

Create a metadata file

Technically, we can still use our gap data set as the metadata file. However, be warned that the assumption here is that your metadata has observations in the exact same order as your data matrix. Let's add a line of code to ensure this is the case, so you know these steps for your analysis.

```
# Prepare your metadata ####

# subset the gapminder data set to get our metadata
meta <- gap[,c("country", "continent", "year")]
```

```
# re-order according to the row names of our data matrix
meta <- meta[match(rownames(mat), meta$country),]

# check to make sure all is in the same order
length(which(meta$country == rownames(mat))) / nrow(mat)
```

```
## [1] 1
```

Great! Now we make our PCA. Generally, data should be normalized to do this, and most common are the log2 or log10 transformations. For today, let's start with a PCA using our raw data, and compare it to our PCA with log-transformed data.

Create the PCA

The simplest way to generate a PCA is with the prcomp function built into Base R. Let's start with a look at the help.

```
?prcomp
```

```
## starting httpd help server ... done
```

The main variables to look out for are: 1. data: the data set you will use to model (here, mat) 2. na.action: This is important! PCAs (and many other mulivariate tools) do not take missing values. Now, dealing with them is another topic altogether. For now, suffice it to say that the default is for the function to fail if your data matrix contains missing values. If you specify na.action = na.omit, then the rows containing missing values will be removed from the data set prior to any modelling. 3. center: Do you want your PCA to be centred in space? Play around with setting center to TRUE or FALSE to see the difference. 4. scale.: Also very useful. If scale. is set to TRUE, then all your variables will be scaled (think mean of 0, SD of 1). scaling helps to eliminate any skewing of the data by some variables that speak louder than others. It is generally advisable to scale your PCA.

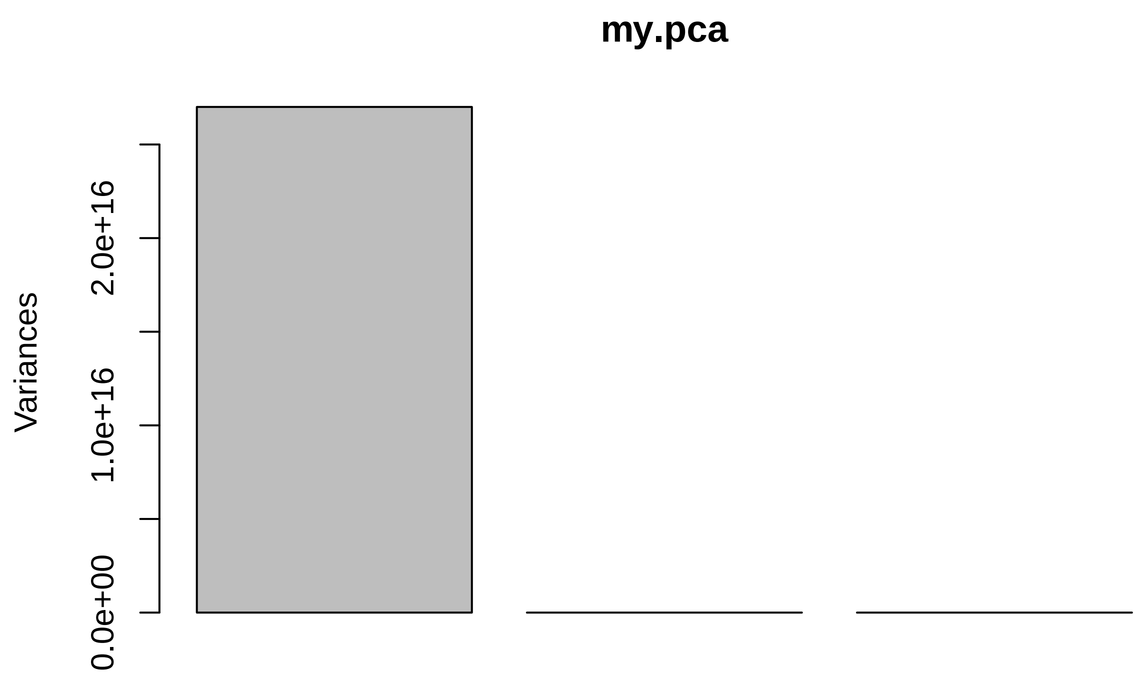
Let's give this a try.

```
# build our first PCA ####

my.pca <- prcomp(x = mat,
                 center = TRUE,
                 scale. = FALSE)

my.pca
```

```
## Standard deviations (1, .., p=3):
## [1] 1.643315e+08 1.671901e+04 6.417889e+00
##
## Rotation (n x k) = (3 x 3):
##                     PC1          PC2           PC3
## lifeExp   -1.608254e-09 3.776992e-04  9.999999e-01
## pop       -1.000000e+00 5.320033e-06 -3.617627e-09
## gdpPercap  5.320034e-06 9.999999e-01 -3.776992e-04
```

Great! something happened. However, it is not the cluster we expected. Notice that our PCA object is a list. We can explore it by first having a feel for how much variance was explained by each component by plotting our PCA.

```
plot(my.pca)
```



This is telling us that most of the variance in our data was captured by the first component. We can also see that only three components were plotted. The default is to set the number of components to the number of variables (columns) in our data set. The PCAs we are accustomed to seeing need some extra work to plot. There are many tools for plotting a PCA. However, we're going to start old-school by extracting the data from our object and building a good 'ol fashion ggplot.

```
# extract your PCA data ####
pca.dat <- as.data.frame(my.pca$x) # here the position of our data along all the principal components.

head(pca.dat)
```

```
##                  PC1        PC2       PC3
## Afghanistan 18914104 -14526.620 -8.067964
## Albania     46694104  -6114.404  8.599427
## Algeria     12014104  -3129.905  7.216410
## Angola      24514104 -10396.411 -5.118934
## Argentina    7514104   2894.034  4.233936
## Australia   26814105  26191.358  1.665595
```

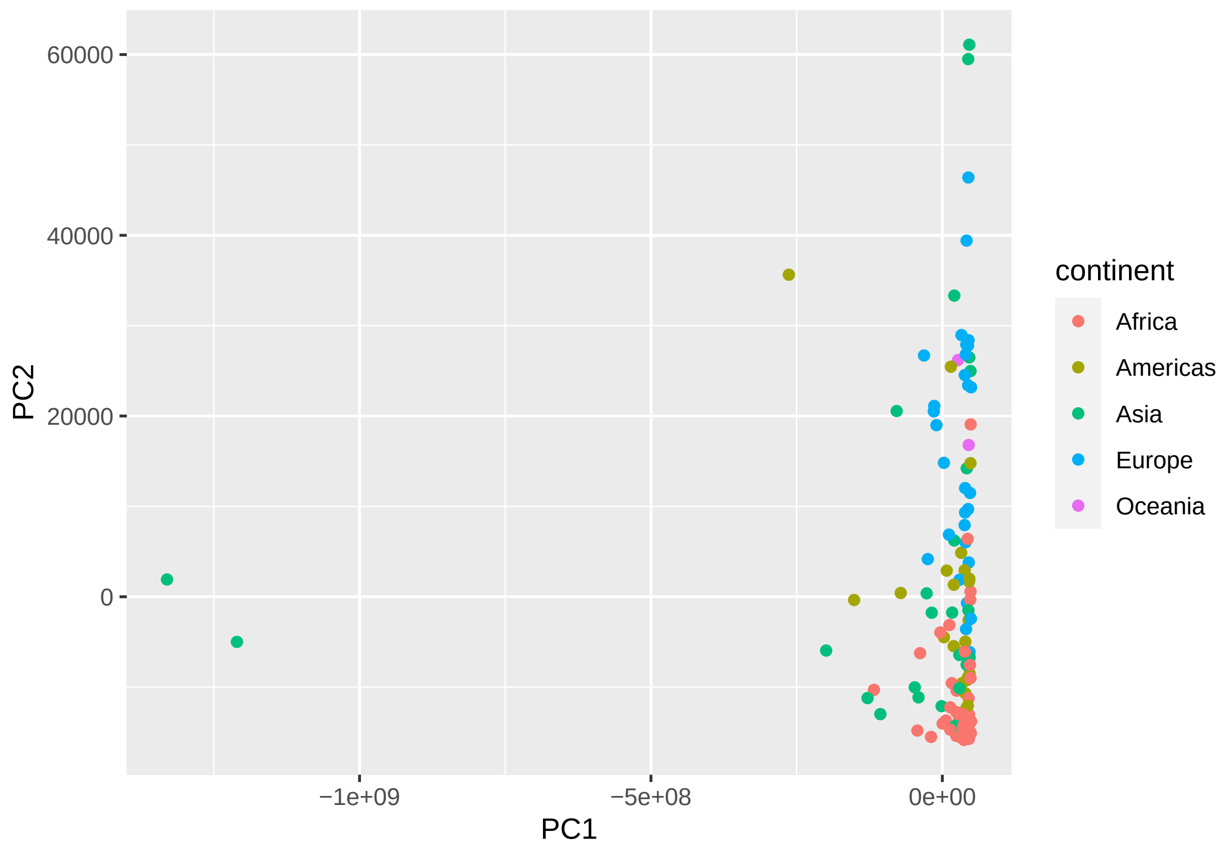
to use our new data set with ggplot, we will need to add back our metadata.

```
# add your metadata ####
pca.dat$country <- rownames(pca.dat)

pca.dat <- join(pca.dat, meta, by = "country")

head(pca.dat)
```

```
##        PC1        PC2       PC3     country continent year
## 1 18914104 -14526.620 -8.067964 Afghanistan      Asia 2012
## 2 46694104  -6114.404  8.599427     Albania    Europe 2012
## 3 12014104  -3129.905  7.216410     Algeria    Africa 2012
## 4 24514104 -10396.411 -5.118934      Angola    Africa 2012
## 5  7514104   2894.034  4.233936   Argentina  Americas 2012
## 6 26814105  26191.358  1.665595   Australia   Oceania 2012
```

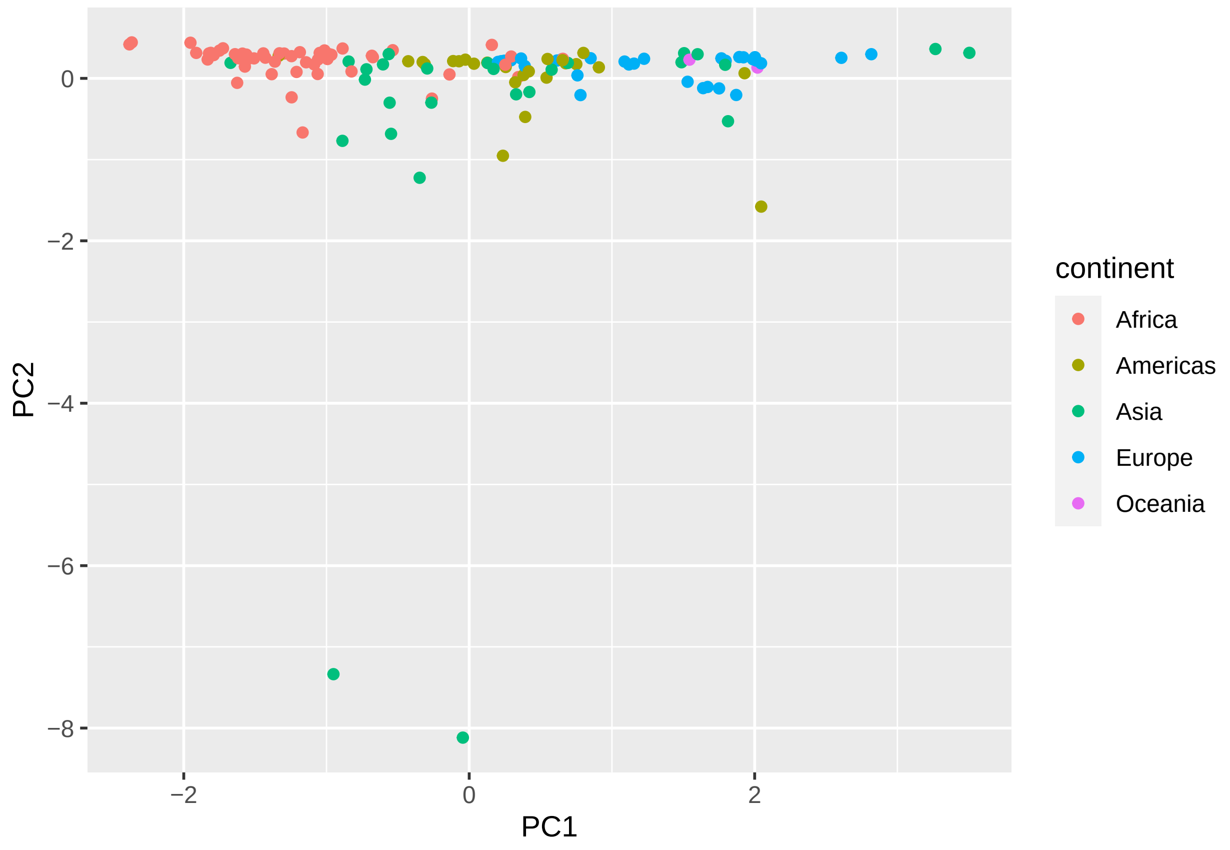
We can now plot our data with ggplot.

```
# visualize our PCA ####
ggplot(pca.dat, aes(x = PC1, y = PC2, color = continent)) +
  geom_point()
```



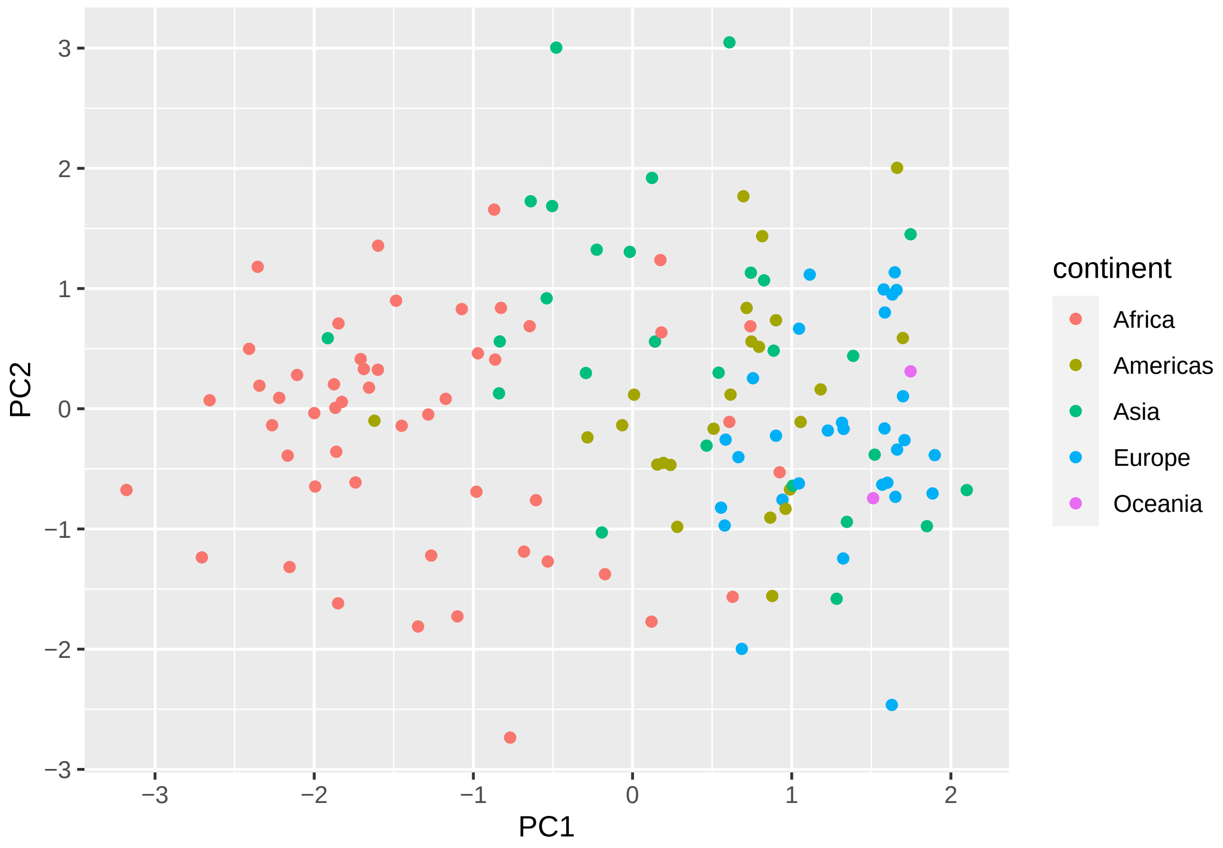
Wow! nice. but not quite what we expected. Let's see if scaling our PCA will make a difference.

```
# scale your PCA ####
my.pca <- prcomp(x = mat,
                 center = TRUE,
                 scale. = TRUE)

# extract the data
pca.dat <- as.data.frame(my.pca$x)

# add the metadata
pca.dat$country <- rownames(pca.dat)
pca.dat <- join(pca.dat, meta, by = "country")

# plot again
ggplot(pca.dat, aes(x = PC1, y = PC2, color = continent)) +
  geom_point()
```



This didn't help too much. However, recall that our variables had very different scales. Some were log-distrubuted, others were not. If you build a PCA and it looks like this, it is likely that you need to perform some normalization steps prior to plotting your data. Let's try one last time, but this time apply a log2 transformation to our data matrix as well.

```
# log-transformed PCA ####
my.pca <- prcomp(x = log2(mat),
                 center = TRUE,
                 scale. = TRUE)

# extract the data
pca.dat <- as.data.frame(my.pca$x)

# add the metadata
pca.dat$country <- rownames(pca.dat)
pca.dat <- join(pca.dat, meta, by = "country")

# plot again
ggplot(pca.dat, aes(x = PC1, y = PC2, color = continent)) +
  geom_point()
```



That's more like it!

Variance Explained

The one thing that is missing here are the labels indicating the variance explained by each component. For many PCA-tools and packages, this information is grabbed out of the PCA object for you. However, it's nice to know where it comes from. So, our last step will be to get this from our object and incorporate it into our plot.

```
# get the PCA summary ####
sum <- summary(my.pca)
class(sum)
```

```
## [1] "summary.prcomp"
```

```
We can see that the proportion of variabnce explaned is included in the second row of the summary.  How
```

```
# get PCA importance ####
imp <- sum$importance
imp
```

```
##                              PC1      PC2       PC3
## Standard deviation     1.346723 1.002052 0.4268817
## Proportion of Variance 0.604550 0.334700 0.0607400
## Cumulative Proportion  0.604550 0.939260 1.0000000
```

```
class(imp)
```

```
## [1] "matrix" "array"
```

Great! Using the data frame navigation skills we have learned, we know that we can get the variance expl

```
# get variance explained ####

pc1.var <- imp[2,1]*100
pc2.var <- imp[2,2]*100

# create axis labels
pc1.lab <- paste("PC1:", pc1.var, "% expl. var")
pc2.lab <- paste("PC2:", pc2.var, "% expl. var")

pc1.lab
```

```
## [1] "PC1: 60.455 % expl. var"
```

```
pc2.lab
```

```
## [1] "PC2: 33.47 % expl. var"
```

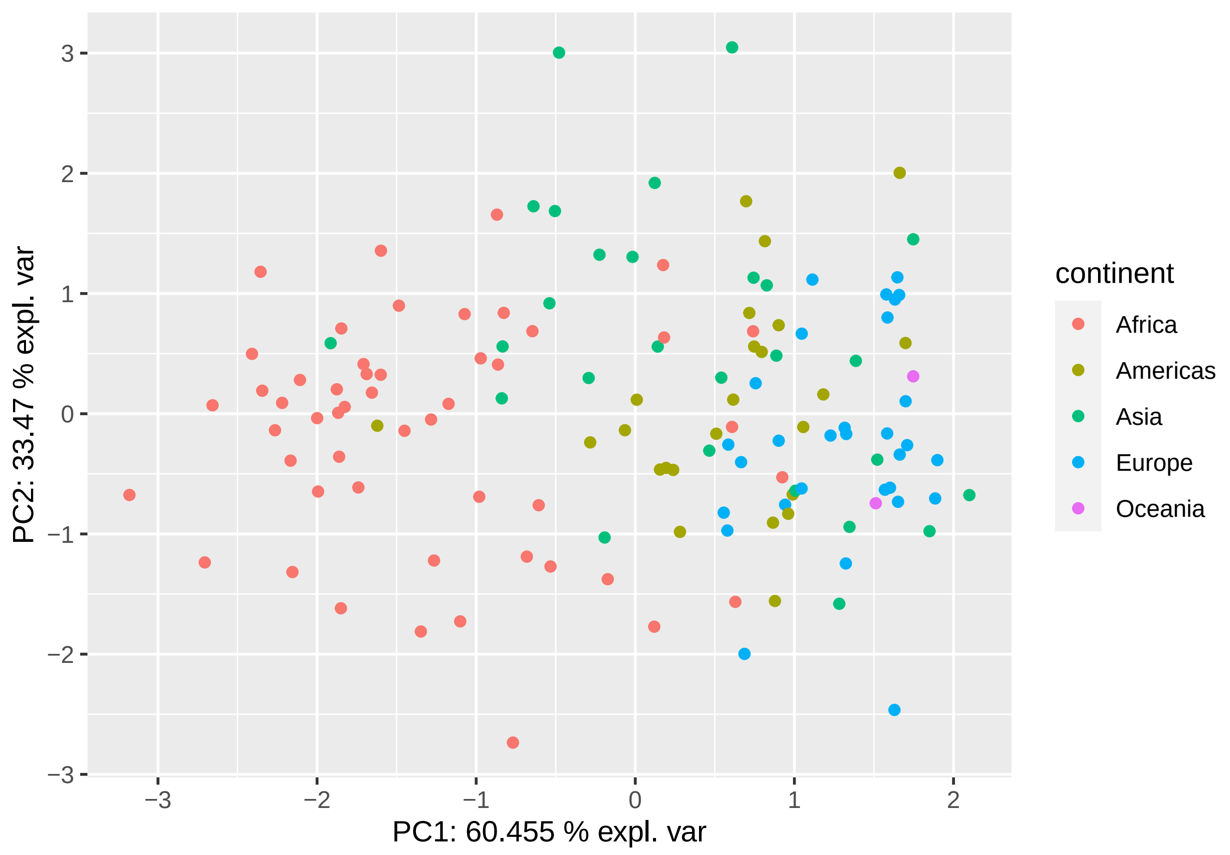
BONUS: For you keeners out there, you can create a function in a separate script and save it for later u

```
# make a function for PCA labels ####
get.pca.dat <- function(pca.object){


  pc.dat <- summary(pca.object)
  pc.dat <- pc.dat$importance


  pc1.var <- paste("PC1:", 100*pc.dat[2,1], "% expl. var")
  pc2.var <- paste("PC2:", 100*pc.dat[2,2], "% expl. var")
  pc3.var <- paste("PC3:", 100*pc.dat[2,3], "% expl. var")

  to.return <- list(pc1.var, pc2.var, pc3.var)
  names(to.return) <- c("pc1", "pc2", "pc3")

  return(to.return)

}

# try it out
my.labs <- get.pca.dat(my.pca)
my.labs
```

```
## $pc1
## [1] "PC1: 60.455 % expl. var"
##
## $pc2
## [1] "PC2: 33.47 % expl. var"
##
## $pc3
## [1] "PC3: 6.074 % expl. var"
```

Now we can complete our PCA by adding the appropriate axis labels.

```
ggplot(pca.dat, aes(x = PC1, y = PC2, color = continent)) +
  geom_point() +
  labs(x = pc1.lab, y = pc2.lab)
```



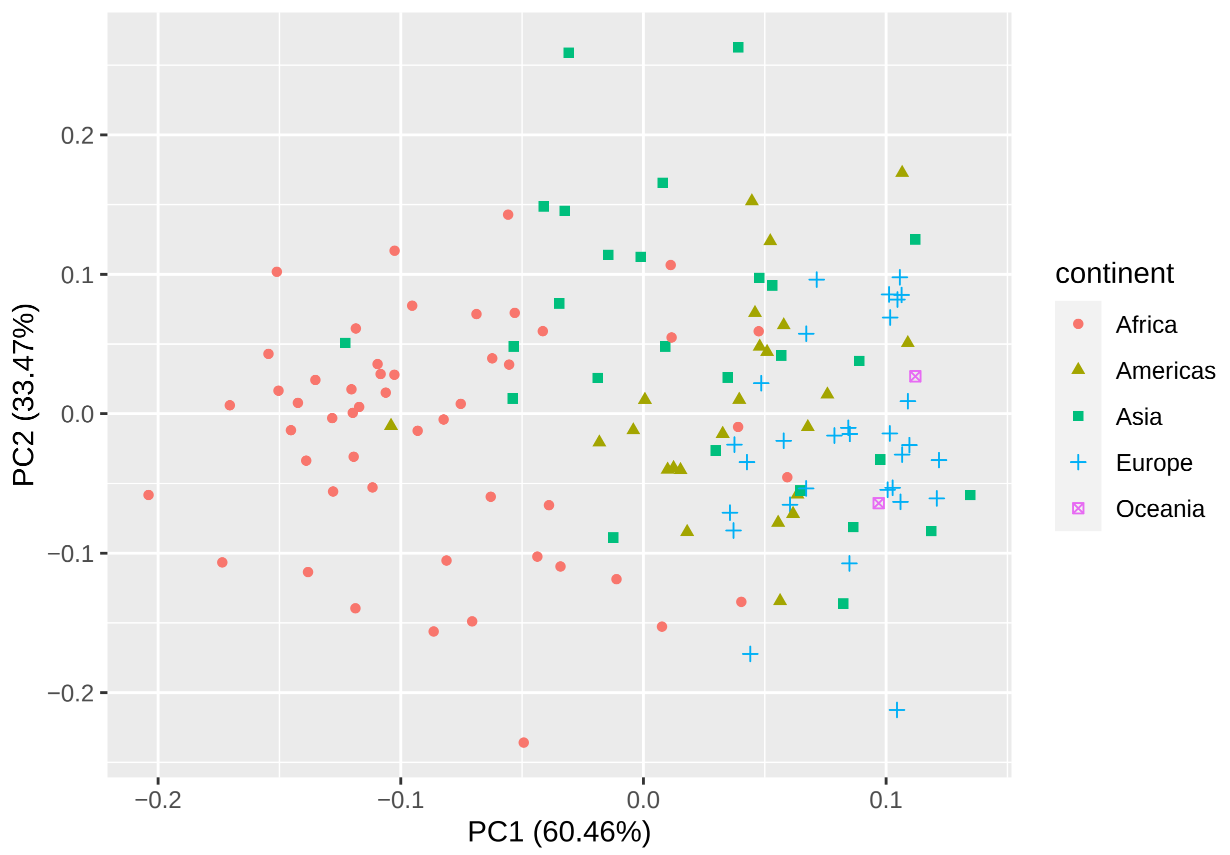
Congrats! you've made your first PCA.

PCA the ggfortify way

While this strategy has now given you a great idea of how PCAs are generated in R and how to access the important information, there are several R packages that utilize the ggplot2 aesthetics but take out a lot of the leg work for you to go from your prcomp object to your ggplot. We will give you one quick example below, and provide references to other well-annotated tools that you can explore on your own.

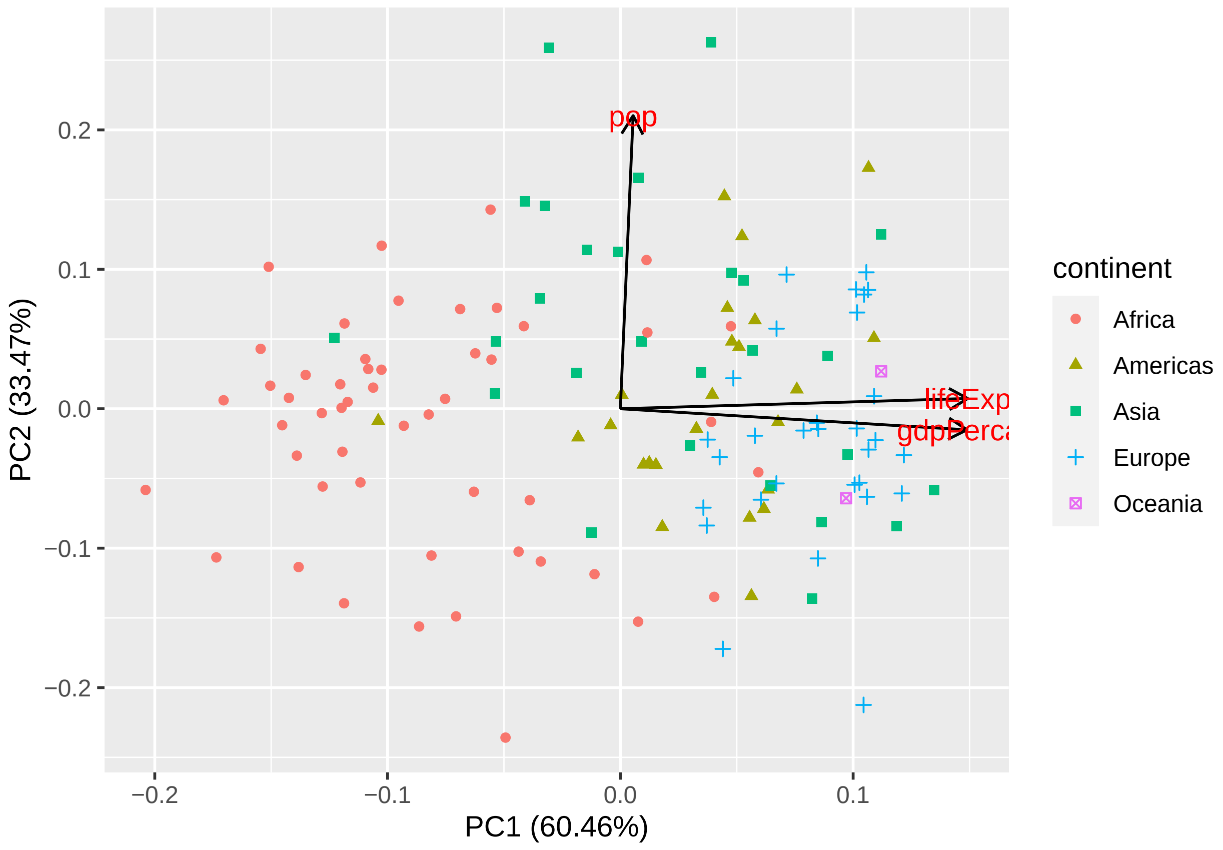
ggfortify is a simple strategy that will take your prcomp object, your metadata file, and build your ggplot using the autoplot function.

```
# PCA the autoplot way ####
autoplot(my.pca,
         data = meta,
         colour = "continent",
         shape = "continent")
```



One valuable peice of information that we can add to our PCA is the loading vectors. In other words, how do the variables in our data set drive the separation of our data? Autoplot includes an argument to include this.

```
autoplot(my.pca,
         data = meta,
         colour = "continent",
         shape = "continent",
         loadings  = TRUE,
         loadings.colour = "black",
         loadings.label = TRUE,
         loadings.label.size = 4)
```



There you have it. PCA two-ways. However, there are a wealth of strategies that you can use to generate your PCA. We encourage you to read further.

PCA resources

mixOmics

mixOmics is a fantastic suite of tools for multivariate analysis: supervised, unsupervised, multi or single omics. Thus, getting used to the syntax has it's advantages, as they may transfer to other tools in this package. This framework also includes the sparse PCA. If you have many variables, but want a convenient way to eliminate some of them that do not contribute to variance in your experiment, then sparse PCA will be useful to you.

For more information: http://mixomics.org/methods/pca/

PCA tools

```
PCAtools is a well-annotated package that, similarly to ggbiplot, gives you quick ways to build and plo
```

For the vignette, see: https://bioconductor.org/packages/release/bioc/vignettes/PCAtools/inst/doc/PCAtools.html#a-bi-plot

Heatmaps three ways

Another very common exploratory data analysis tool is the heat map: a graphical representation of data where values are depicted by colour. Heatmaps are extremely versatile and are an efficient way to visualise trends. In addition to colour coding values, heatmaps often make use of unsupervised hierarchical clustering to group similar observations. Here, we will look at three ways of creating heatmaps in R: 1) using ggplot, 2) using the base R function `heatmap`, and 3) using the `pheatmap` package.

Prepare the data

First, let's prepare the gapminder data in a way that makes sense for visualisation with a heatmap.

```
library(gapminder)
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

If we want to compare values across countries and over time, would it make sense to include lifeExp, pop and gdPercap all together in a heatmap given their different scales? Perhaps a better approach would be to create separate heatmaps for these variables. Here, let's focus on the "lifeExp" column, and remove the other unwanted columns. Let's also focus our analysis on Africa, to keep the plots manageable. We'll use some tidyverse syntax to do this:

```
gap2 <- gapminder %>%
  filter(continent %in% c("Africa")) %>%
  dplyr::select(country, continent, year, lifeExp)
head(gap2)
```

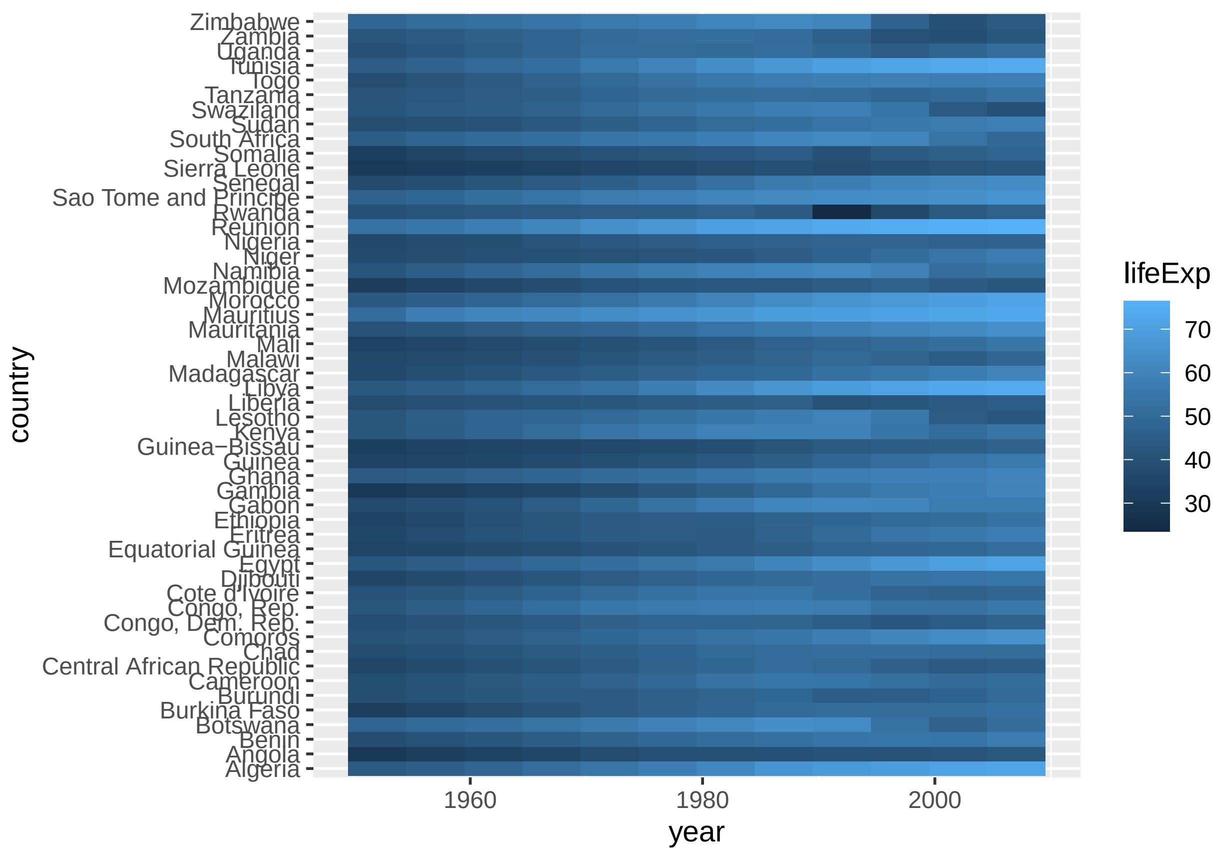
```
## # A tibble: 6 x 4
##   country continent  year lifeExp
##   <fct>   <fct>     <int>   <dbl>
## 1 Algeria Africa     1952    43.1
## 2 Algeria Africa     1957    45.7
## 3 Algeria Africa     1962    48.3
## 4 Algeria Africa     1967    51.4
## 5 Algeria Africa     1972    54.5
## 6 Algeria Africa     1977    58.0
```

Great! Now we're reading to plot a heatmap using ggplot!

1) ggplot

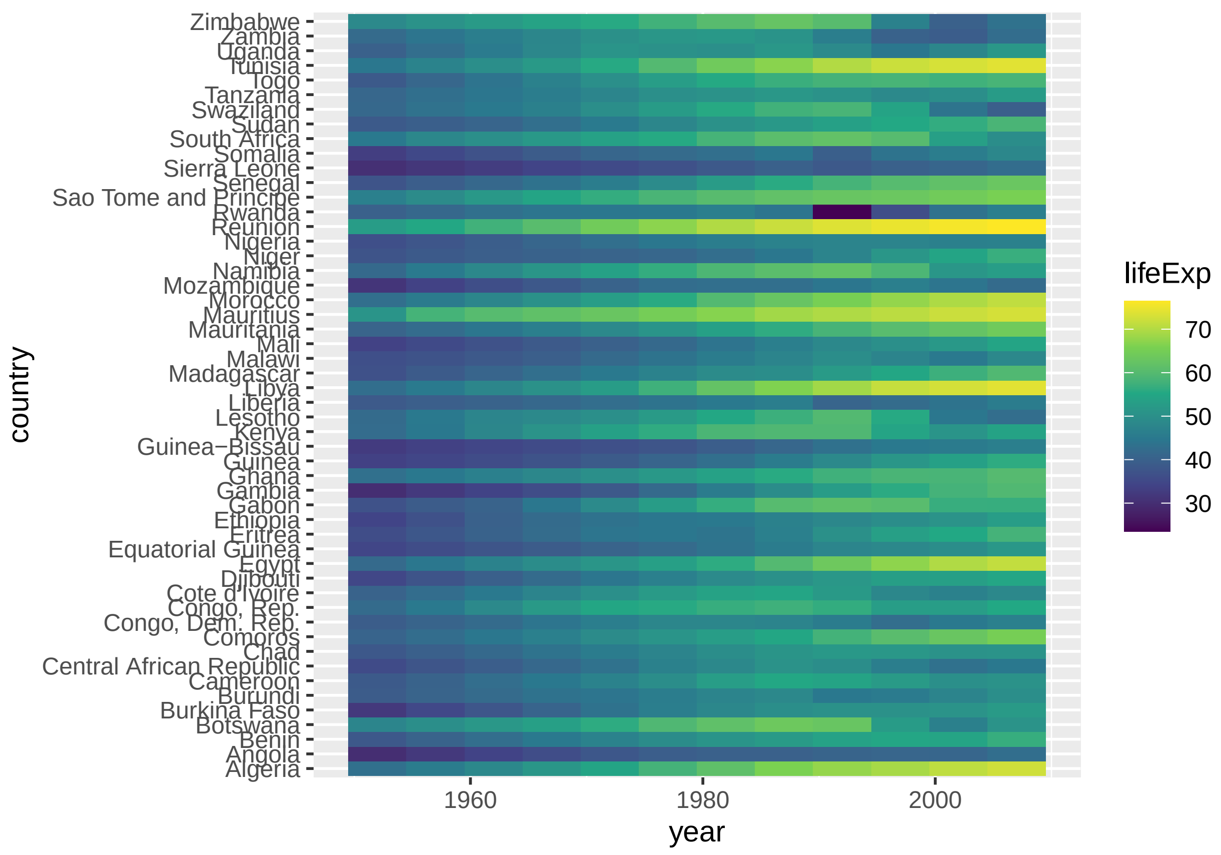
Previously we have used the `reshape` package to melt data before plotting with ggplot, however, the data we have is already in long-format so we can proceed straight to plotting using the geom_tile layer:

```
ggplot(gap2, aes(year, country)) +
  geom_tile(aes(fill = lifeExp))
```



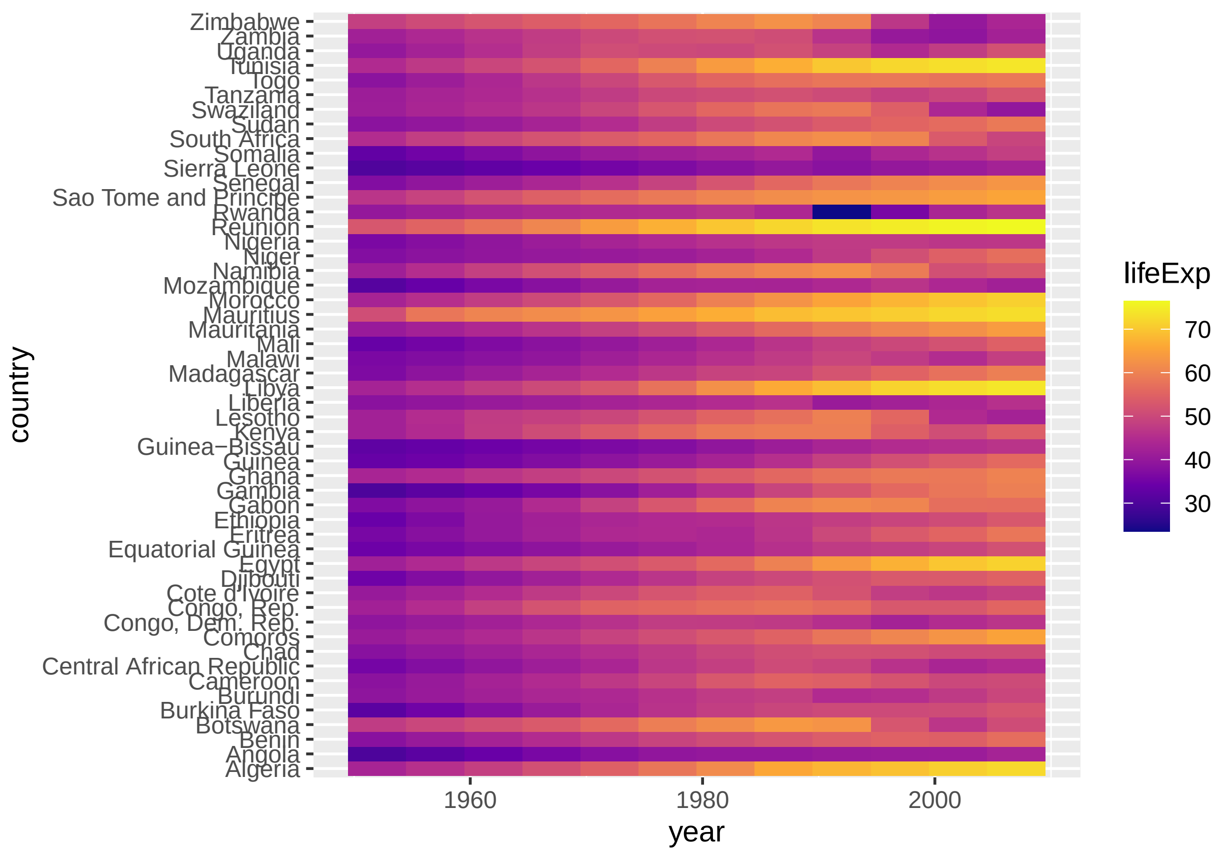
By default, ggplot using a continuous scale of blues to fill the heatmap. Let's change up the colours to better see the trends over time! We can do this using the layer `scale_fill_viridis_c()` - the "c" at the end stands for continuous. The viridis colour scales have been designed especially to be perceived by viewers with common forms of colour blindness.

```
ggplot(gap2, aes(year, country)) +
  geom_tile(aes(fill = lifeExp)) +
  scale_fill_viridis_c()
```



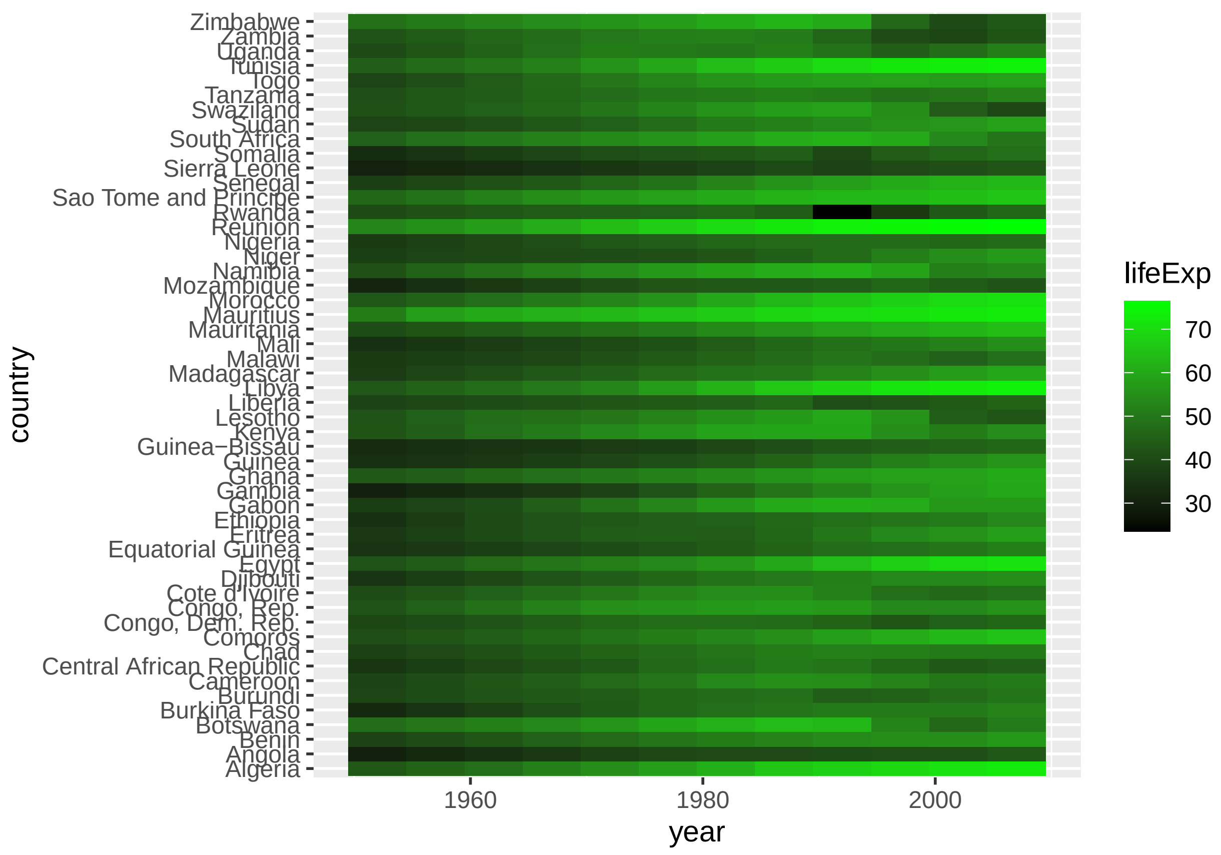
Let’s try one of the other colour options called “plasma”:

```
ggplot(gap2, aes(year, country)) +
  geom_tile(aes(fill = lifeExp)) +
  scale_fill_viridis_c(option = "plasma")
```



If you want to specify your own colours, you can do so using the `scale_fill_gradient()` layer:

```
ggplot(gap2, aes(year, country)) +
  geom_tile(aes(fill = lifeExp)) +
  scale_fill_gradient(low = "black", high = "green")
```



2) Base R

OK now let's go back to basics, and see how we can plot a heatmap without using that somewhat confusing ggplot code! First, we need our data in wide-format:

```
gap3 <- gap2 %>%
  pivot_wider(
    names_from = year,
    values_from = c(lifeExp)
  )

head(gap3)
```

```
## # A tibble: 6 x 14
##   country      continent `1952` `1957` `1962` `1967` `1972` `1977` `1982` `1987`
##   <fct>        <fct>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Algeria      Africa      43.1   45.7   48.3   51.4   54.5   58.0   61.4   65.8
## 2 Angola       Africa      30.0   32.0   34     36.0   37.9   39.5   39.9   39.9
## 3 Benin        Africa      38.2   40.4   42.6   44.9   47.0   49.2   50.9   52.3
## 4 Botswana     Africa      47.6   49.6   51.5   53.3   56.0   59.3   61.5   63.6
## 5 Burkina Faso Africa      32.0   34.9   37.8   40.7   43.6   46.1   48.1   49.6
## 6 Burundi      Africa      39.0   40.5   42.0   43.5   44.1   45.9   47.5   48.2
## # ... with 4 more variables: 1992 <dbl>, 1997 <dbl>, 2002 <dbl>, 2007 <dbl>
```

Next, we need the data to be in the form of a numeric matrix - see the help page for the base R `heatmap` function:

```
?heatmap
```

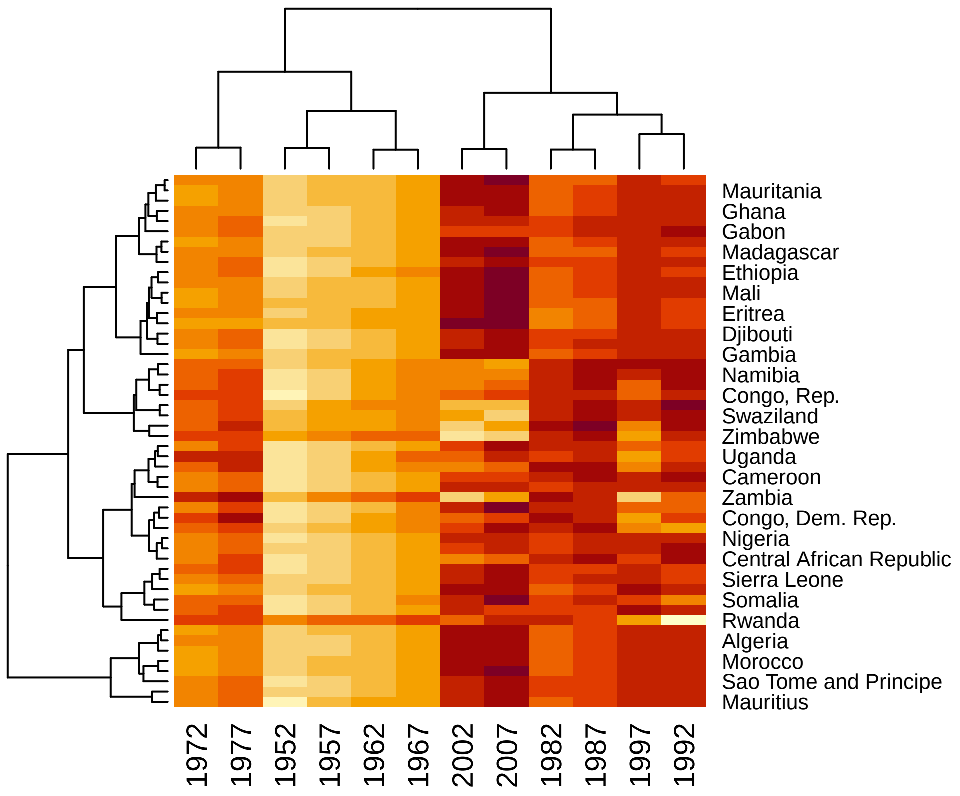
Let's convert our data into the right format by converting it to a numeric matrix (we need to exclude the "country" data here):

```
gap.mat <- as.matrix(gap3[,3:14])

# label the rows by country
rownames(gap.mat) <- gap3$country

head(gap.mat)
```

```
##                1952   1957   1962   1967   1972   1977   1982   1987   1992
## Algeria      43.077 45.685 48.303 51.407 54.518 58.014 61.368 65.799 67.744
## Angola       30.015 31.999 34.000 35.985 37.928 39.483 39.942 39.906 40.647
## Benin        38.223 40.358 42.618 44.885 47.014 49.190 50.904 52.337 53.919
## Botswana     47.622 49.618 51.520 53.298 56.024 59.319 61.484 63.622 62.745
## Burkina Faso 31.975 34.906 37.814 40.697 43.591 46.137 48.122 49.557 50.260
## Burundi      39.031 40.533 42.045 43.548 44.057 45.910 47.471 48.211 44.736
##                1997   2002   2007
## Algeria      69.152 70.994 72.301
## Angola       40.963 41.003 42.731
## Benin        54.777 54.406 56.728
## Botswana     52.556 46.634 50.728
## Burkina Faso 50.324 50.650 52.295
## Burundi      45.326 47.360 49.580
```

Now, let's plot the heatmap, technically there is only one argument we need to specify, and that is the data to be plotted:

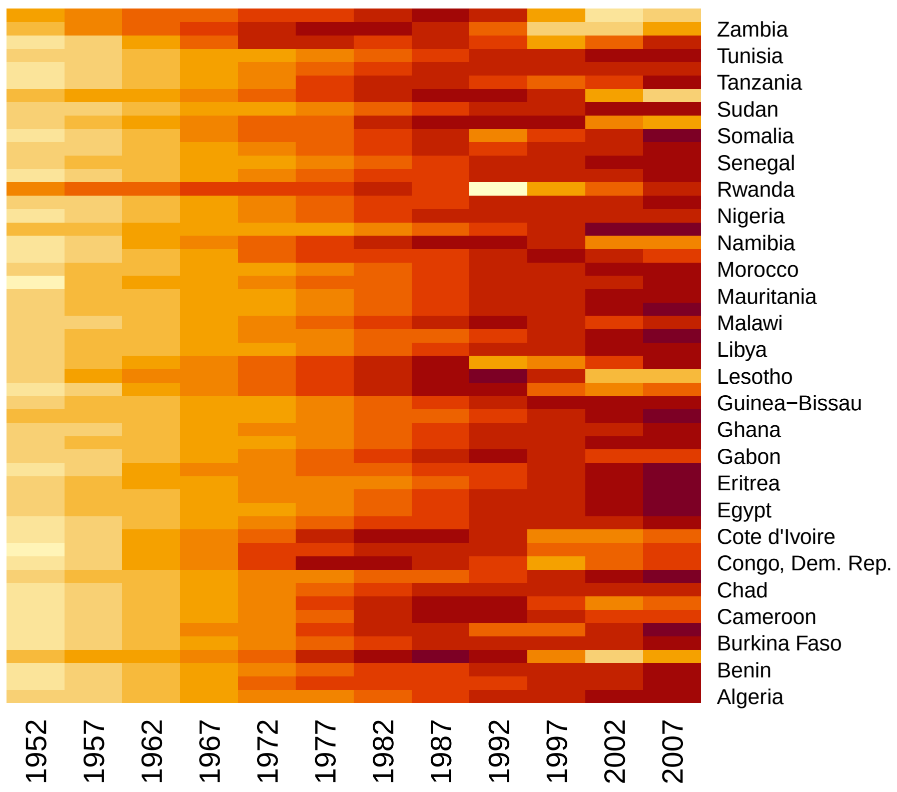
```
heatmap(gap.mat)
```



You can see that this heatmap looks a lot different to the ggplot heatmap. What are those lines on the sides and on the top? In contrast to the heatmaps we created earlier with ggplot, the base R `heatmap` function performs some clustering of the rows and columns, and provides a dendrogram to visualise how dissimilar the observations are - this can be very useful for spotting trends!

If you don't want those to be included, you can remove these by setting the arguments Rowv and Colv to "NA":

```
heatmap(gap.mat, Rowv = NA, Colv = NA)
```

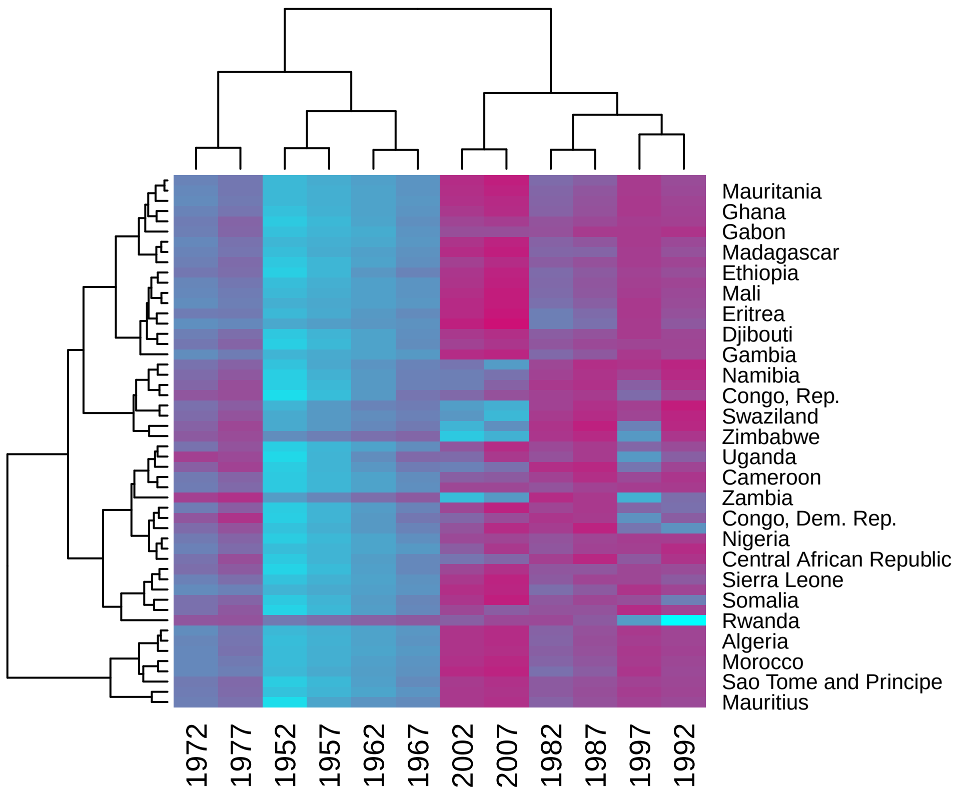


How about changing up the colour? Easily done in two steps - step 1 is to create your own colour palette using the `colorRampPalette` function:

```
my_colours <- colorRampPalette(c("cyan","deeppink3"))
```

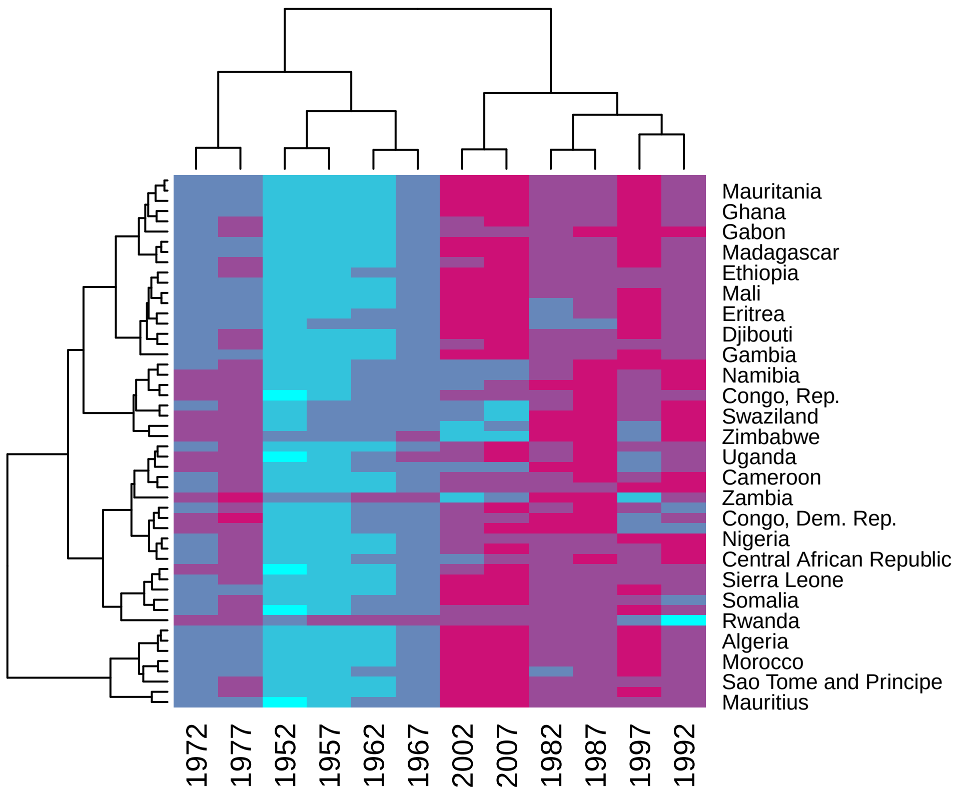
Step 2 - specify the colours using the `col` argument within the heatmap function:

```
heatmap(gap.mat, col = my_colours(100))
```



The number inside the parentheses specifies how granular or fine the colours will appear - the higher the number, the finer the scale. See how it looks with a low number in comparison:

```
heatmap(gap.mat, col = my_colours(5))
```



3) Pheatmap

Pheatmap offers a nice middle ground between the beautiful and easily customizable heatmaps create by ggplot, and the simple code of base R heatmaps. Similar to the base R heatmap function, pheatmap takes the data in wide-format. Load (and install if needed) the package `pheatmap`:

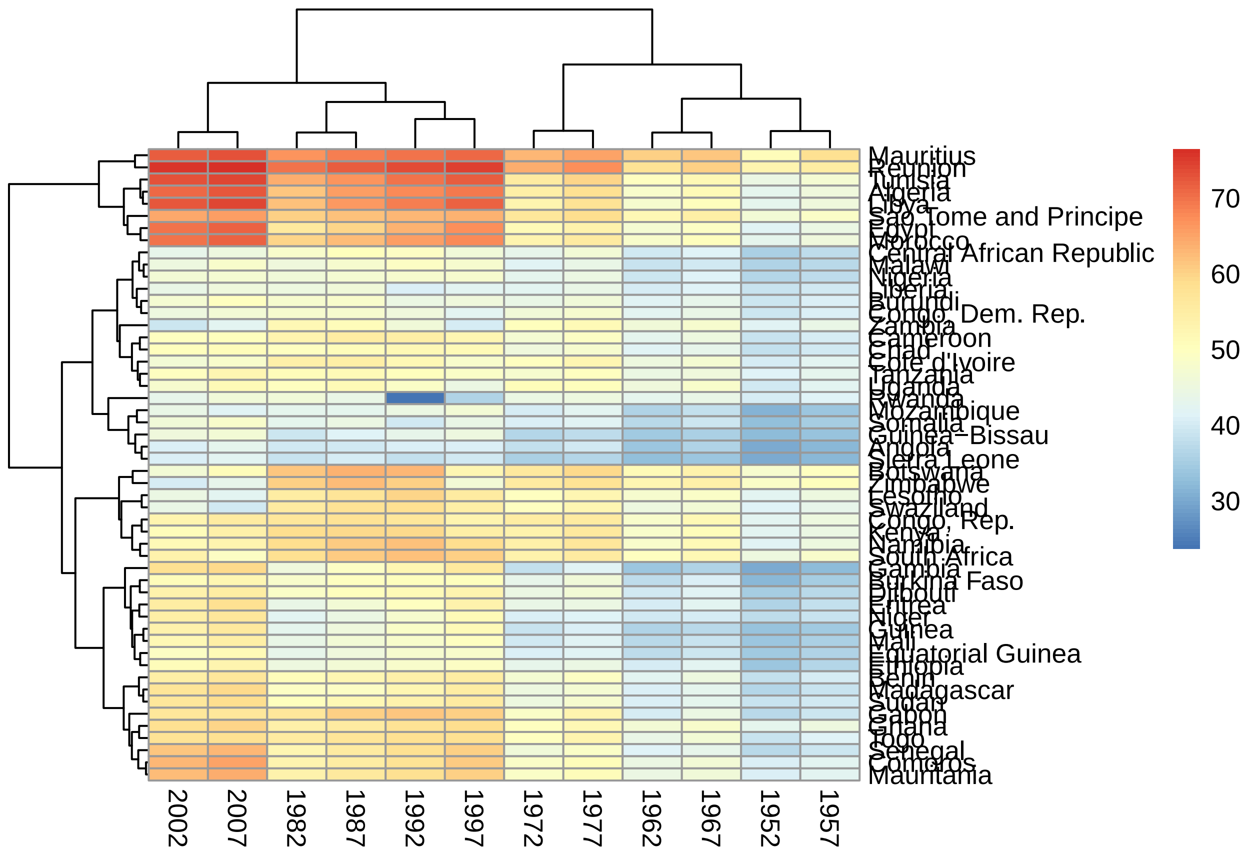
```
library(pheatmap)
```

Check out the help documentation, there is a lot you can customize!

```
?pheatmap
```

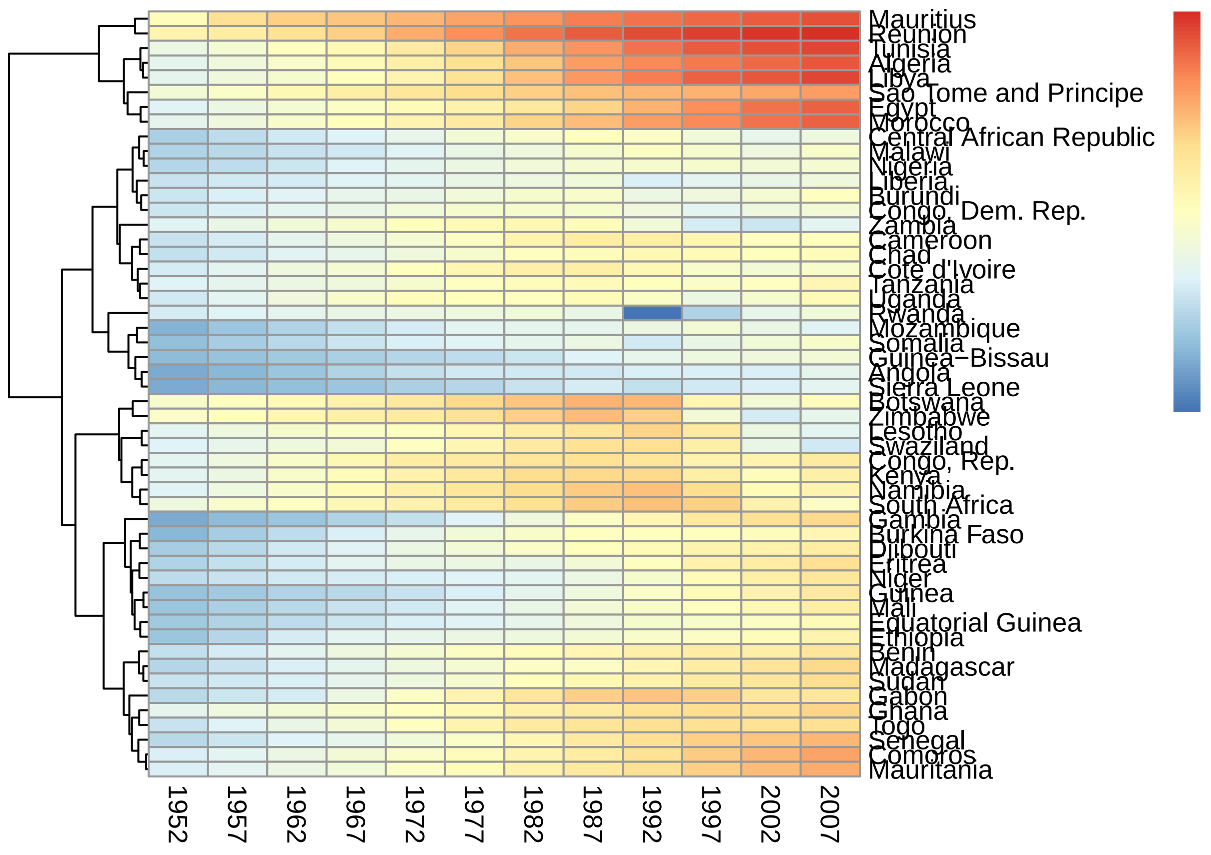
Similar to the base R heatmap function, pheatmap requires a numeric matrix. Let's plot a heatmap and see what the default settings create for us:

```
pheatmap(gap.mat)
```



If we want to turn off the column clustering so that the years appear in order, we can set the `cluster_cols` argument to FALSE:

```
pheatmap(gap.mat,
         cluster_cols = FALSE)
```

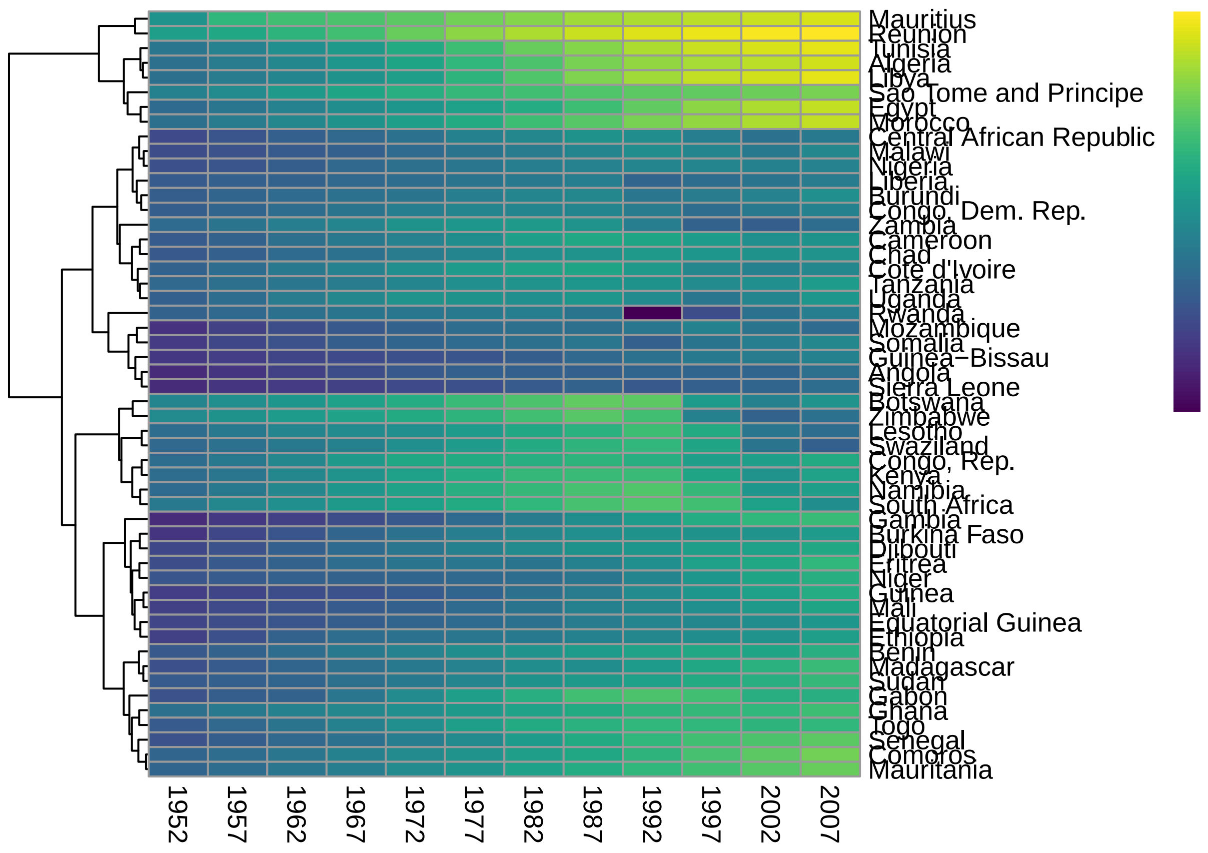


That looks a bit better! How about changing the colour scheme? We can use the viridis colour palette by installing and loading (and installing if needed) the `virids` package:

```
library(viridis)
```

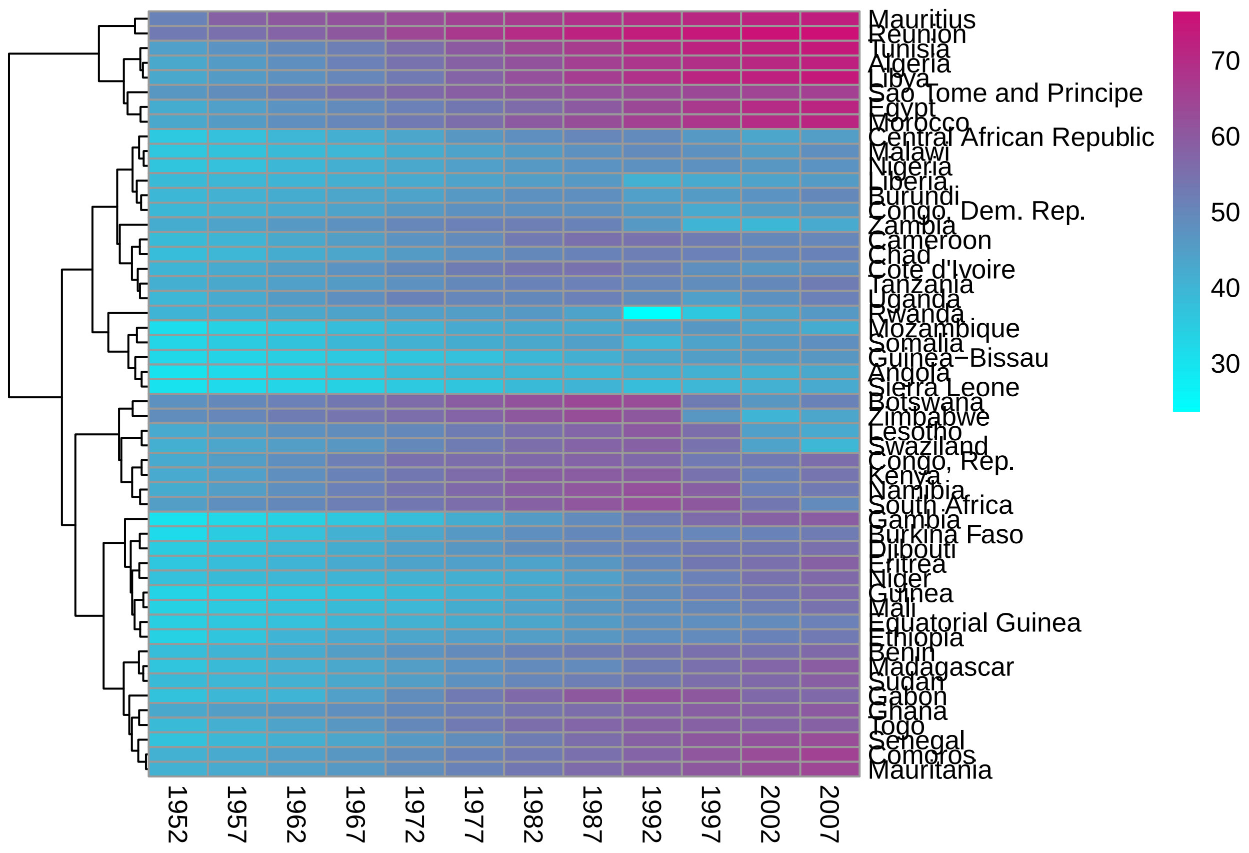
```
## Loading required package: viridisLite
```

```
pheatmap(gap.mat,
         cluster_cols = FALSE,
         color = viridis(100))
```



Or again, we can plug in our own colour palette:

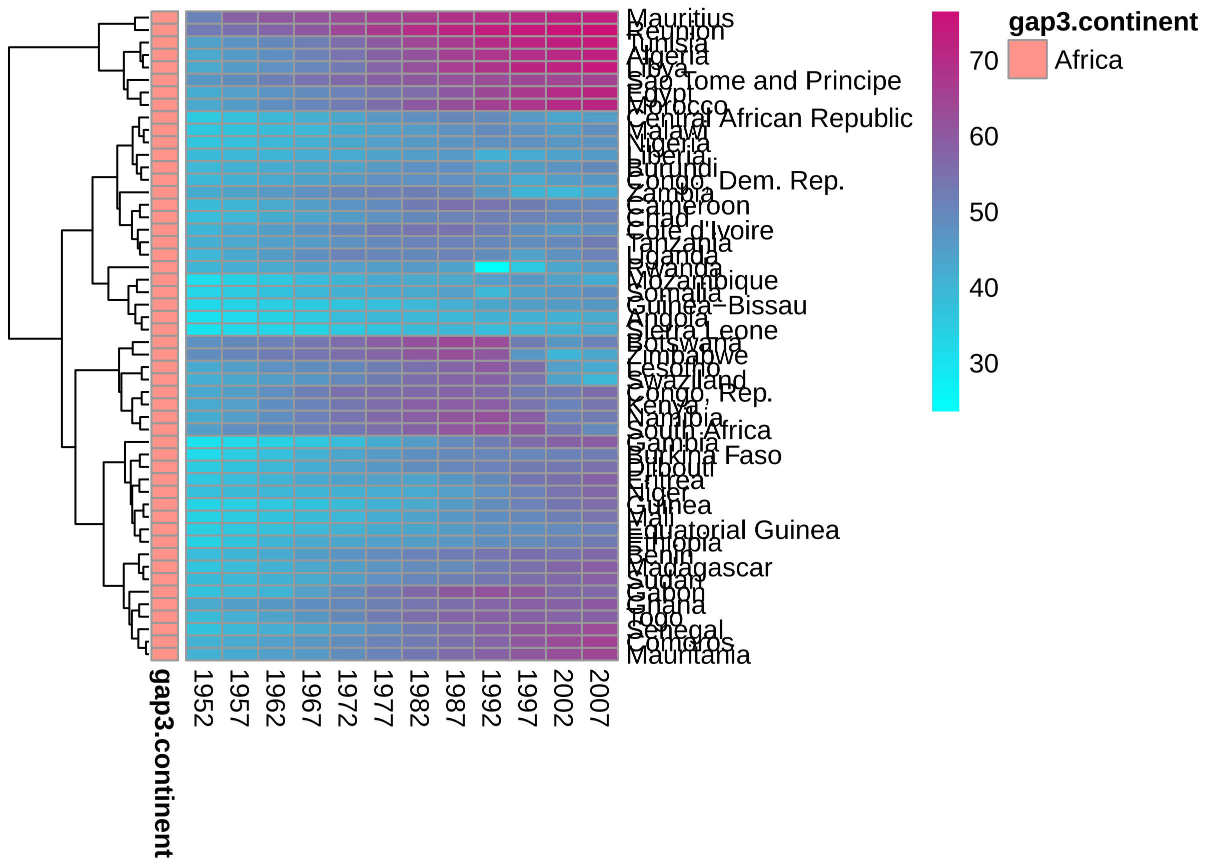
```
pheatmap(gap.mat,
         cluster_cols = FALSE,
         color = my_colours(100))
```



One of the most powerful features of pheatmap is including additional annotation data to the plot. These can be other descriptors of the observations. We can include “continent” as a feature by creating a small data frame containing this information for each country - this won’t be informative for us, since we restricted our analysis to the continent “Africa”, but hopefully it will be a helpful example for you to adapt to your own heatmaps in the future.

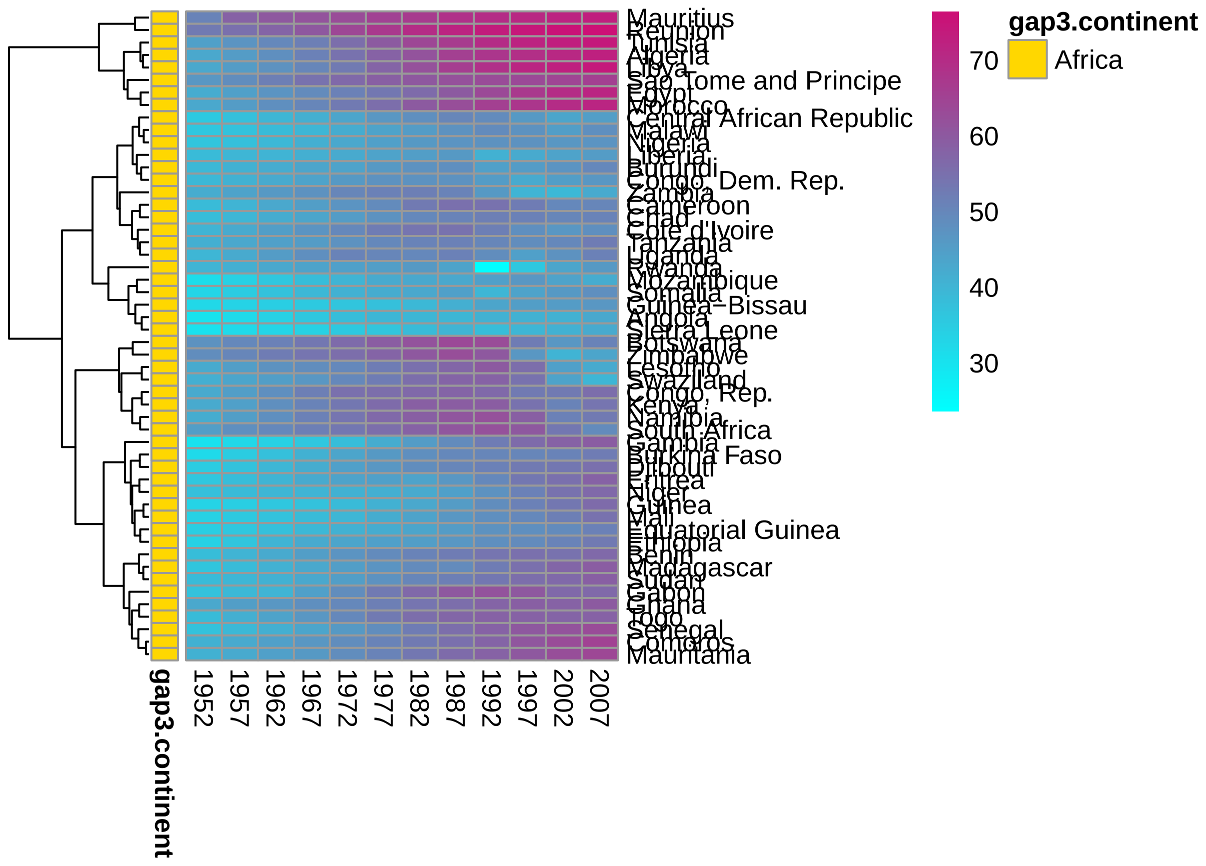
```r
annotation_df <- data.frame(gap3$continent)
rownames(annotation_df) <- gap3$country
```

```r
pheatmap(gap.mat,
         cluster_cols = FALSE,
         color = my_colours(100),
         annotation_row = annotation_df)
```



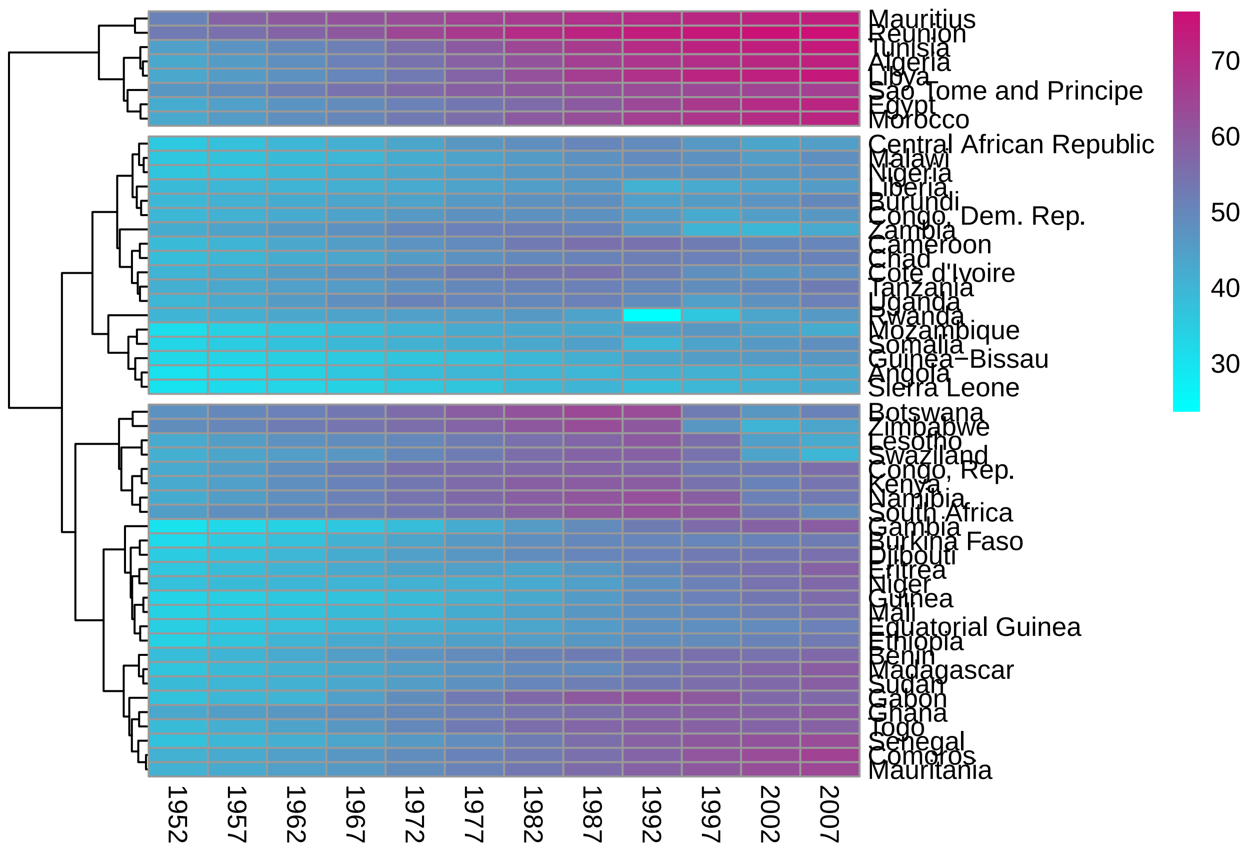
If we don’t like the default colour for this annotation, we can specify our own:

```
annotation_colours <- list(
  gap3.continent = c(Africa = "gold"))

pheatmap(gap.mat,
         cluster_cols = FALSE,
         color = my_colours(100),
         annotation_row = annotation_df,
         annotation_colors = annotation_colours)
```



Another powerful feature of pheatmap is the ability to cut the heatmap into chunks, which can sometimes give clearer visualisation. We can do this either by rows or columns, using the `cutree_rows` or `cutree_cols` arguments:

```
pheatmap(gap.mat,
         cluster_cols = FALSE,
         color = my_colours(100),
         cutree_rows = 3) # the number 3 specifies that we want 3 chunks
```



There you have it, heatmaps three ways! Each with their own pros and cons, and all are appropriate depending on what is important to you at the time - do you just need a quick and simple heatmap? Or are you creating a figure for a publication? Whatever is is, happy heatmap-ing!