Universidade Federal do Ceará

Departamento de Economia Agrícola

Programa de pós-graduação em Economia Rural

Introdução à análise de dados em R

Organizador: Professor Edward Martins Costa

Ministrante: Helson Gomes de Souza

Aula IV: Visualização

As técnicas de importação, limpeza e tratamento dos dados vistas até então são atividades que facilitam a execução de tarefas e tornam a realização de trabalhos mais eficiente. No entanto, a maneira como os dados transmitem as informações pode ser um diferencial interessante. A exposição dos dados pode agregar valor ao trabalho a depender da adequação da exposição ao tipo de informação transmitida, da qualidade da ferramenta de exposição, e da facilidade de compreensão repassada pela exposição dos dados.

Além das ferramentas de manipulação de dados já apresentadas, o R dispõe de um conjunto de bibliotecas que permitem que o usuário escolha dentre um grande conjunto de possibilidades aquela ferramenta que melhor exponha as informações que precisam ser expostas. Aseguir, discutiremos as funcionalidades de algumas delas.

1. plot

Plot é uma função nativa do R que auxilia na criação de gráficos e figuras. A função é composta pelos seguintes parâmetros:

plot(x, y, type, main, sub, xlab, ylab)

Sendo:

x O vetor de valores do eixo x.

y O vetor de valores do eixo x.

main O título do gráfico.

sub O subtítulo do gráfico.

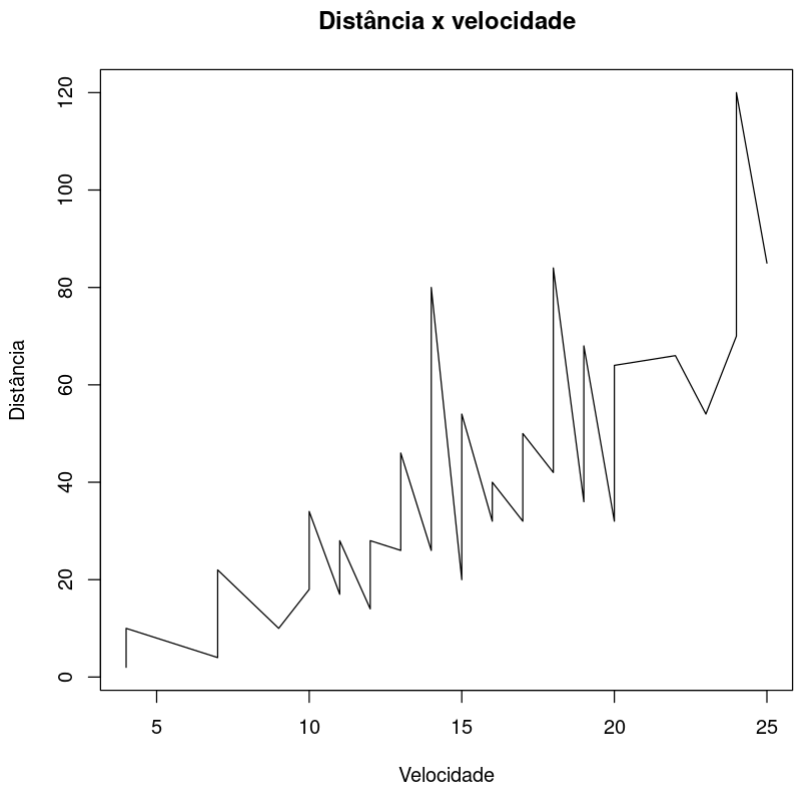
xlab O título do eixo x.

ylab O título do eixo y.

type O tipo do gráfico. Se *type* == *"p"* tem-se um gráfico de pontos; Se *type* == *"l"* tem-se um gráfico de linha; Se *type* == *"b"* tem-se um gráfico de pontos e linha; Se *type* == *"c"* tem-se o mesmo gráfico com *type*== *"b"*, porém, sem os pontos; Se *type* == *"o"* tem-se o mesmo gráfico com *type*== *"b"*, porém, a linha irá sobrepor os pontos; Se *type* == *"h"* tem-se um gráfico de linhas verticais; Se *type* == *"s"* tem-se um gráfico do tipo escada; Se *type* == *"n"* tem-se um gráfico sem feições.

In [23]:

```
plot(x = cars$speed, y = cars$dist, type = "l", main = "Distância x velocidade", xl
     ylab = "Distância")
```

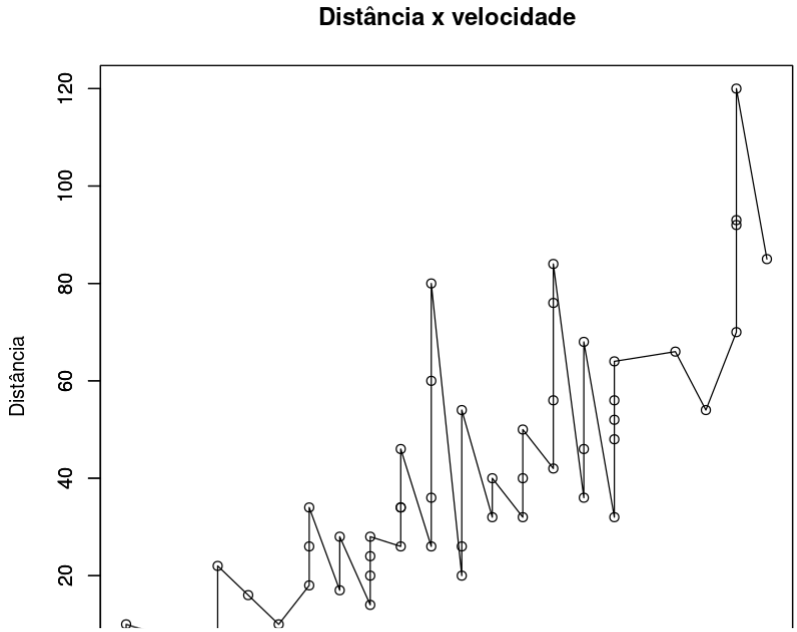


E se o usuário desejar inserir um gráfico adicional na mesma figura?

Nesse caso, deve-se indicar ao R que será icluído um novo gráfico na mesma figura. Podemos fazer isto usando *par(new=TRUE)*.

In [4]:

```
plot(x = cars$speed, y = cars$dist, type = "l", main = "Distância x velocidade", xl
     ylab = "Distância")
par(new=T)
plot(x = cars$speed, y = cars$dist, type = "p", xlab = "", ylab = "")
```



2. Gráfico de barras

Em um gráfico de barras é possível indicar a distribuição das quantidades de uma determinada variável por meio de barras indicativas. No R podemos usar a função nativa *barplot* para executar esta tarefa. Basicamente, os parâmetros da função são:

barplot(height, width, space, names.arg, legend.text, horiz, col, border, main, sub, xlab, ylab, xlim, ylim)

Em que:

height é o vetor de valores que o usuário deseja plotar.

width indica a largura de cada barra.

space indica o tamanho do espaço que irá separar cada barra.

names.arg indica um título opcional para o gráfico que será exposto abaixo do eixo x.

legend.text é o título da legenda caso o usuário queira incluí-la.

horiz quando *horiz = TRUE* as barras são expostas na horizontal.

col indica a cor das barras.

border indica a cor das bordas de cada barra.

main é o título do gráfico.

sub é o subtítulo do gráfico.

xlim são os limites específicos do eixo x que o usuário deseja plotar.

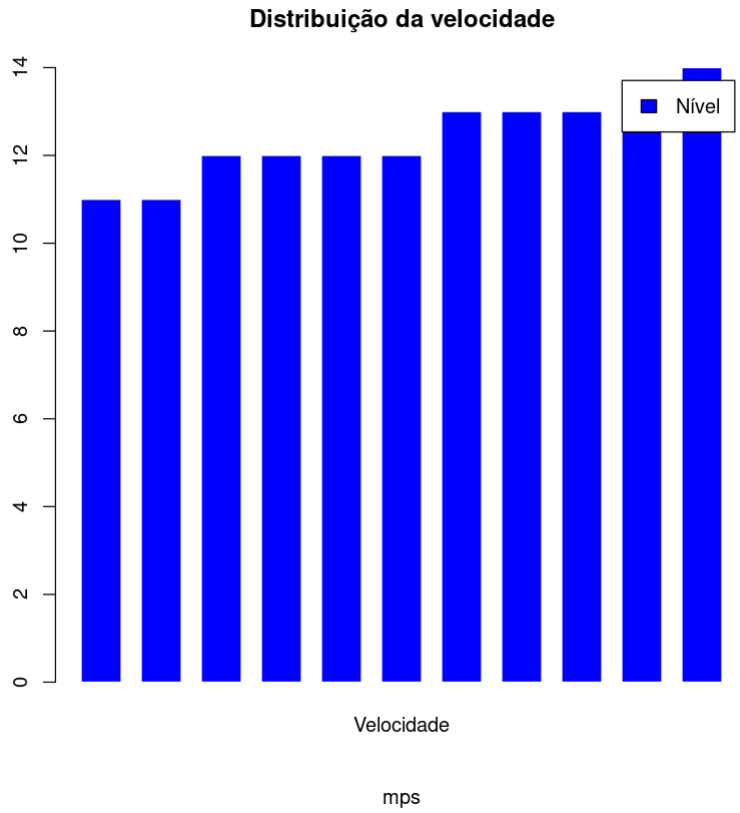
ylim são os limites específicos do eixo y que o usuário deseja plotar.

xlab indica o título do eixo x.

ylab indica o título do eixo y.

In [50]:

```
barplot(height = cars$speed[10:20], width = .2, space = .5, names.arg = "Velocidade
        horiz = F, col = "blue", border = "white", main = "Distribuição da velocidad
```



3. Histograma

Podemos obter um histograma utilizando a função nativa *hist()*. Basicamente, os parâmetros da função são:

hist(x, density, angle, col, border, main, xlim, ylim, xlab, ylab, axes, labels, nclass)

Em que:

x é o vetor de valores que o usuário deseja plotar.

density quando *density* = *TRUE*, um conjunto de linhas é traçado ao longo das barras.

angle é o ângulo de inclinação das linhas traçadas nas barras do histograma quando *density* = *TRUE*.

col indica cor das barras do histograma.

border indica a cor das bordas de cada barra.

main é o título do histograma.

xlim são os limites específicos do eixo x que o usuário deseja plotar.

ylim são os limites específicos do eixo y que o usuário deseja plotar.

xlab indica o título do eixo x.

ylab indica o título do eixo y.

axes quando *axes=FALSE* os eixos x e y não aparecerão.

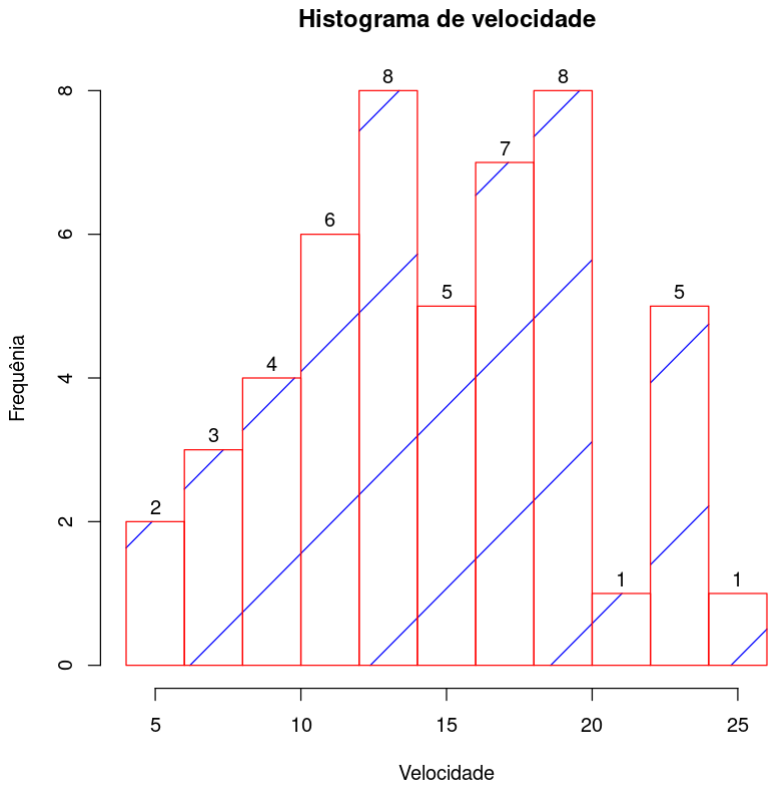
labels se *labels=TRUE* o valor do eixo y será representado no topo de cada barra.

nclass indica o número de barras em que o usuário deseja seguimentar o histograma.

OBSERVAÇÃO Quando *density = TRUE* o parâmetro *col* irá indicar a cor das linhas traçadas sobre as barras ao invés da cor das barras.

In [35]:

```
hist(cars$speed, density = T, angle = 45, border = "red", col = "blue", axes = T, l
    main = "Histograma de velocidade", xlab = "Velocidade", ylab = "Frequênia")
```



4. Gráfico de densidade

Para criarmos um gráfico de densidade, primeiro precisamos criar um objeto contendo as informações da função de densidade da variável. Para tanto, utilizaremos a função density, dada por:

density(x, bw, kernel,...)

Em que:

x é o objeto em que se deseja obter a densidade.

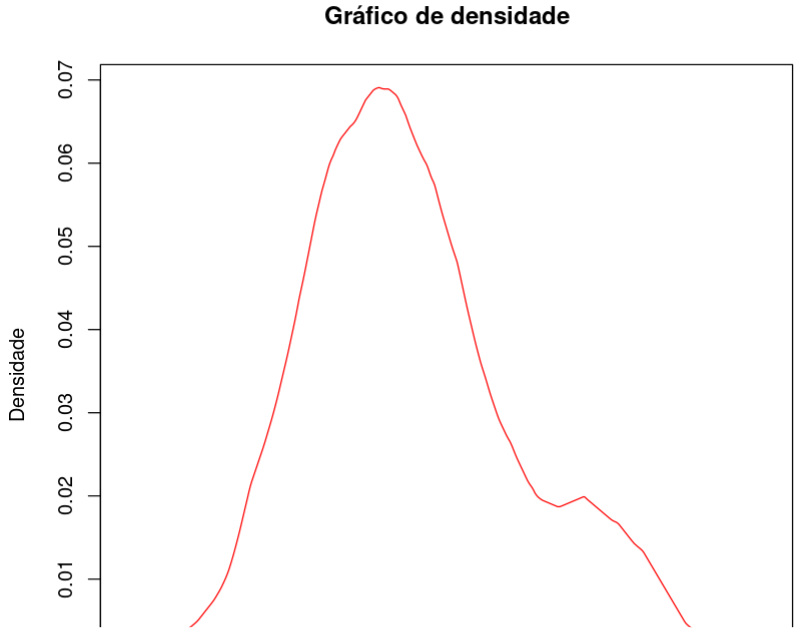
bw é a largura da banda da função de densidade.

kernel serefere a distribuição e pode ser ("gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine")

Em seguida podemos criar o gráfico por meio da função plot.

In [78]:

```
obj <- density(x = mtcars$mpg, kernel = "triangular")
plot(obj, col = "red", main = "Gráfico de densidade", xlab= "Mpg", ylab = "Densidad
```

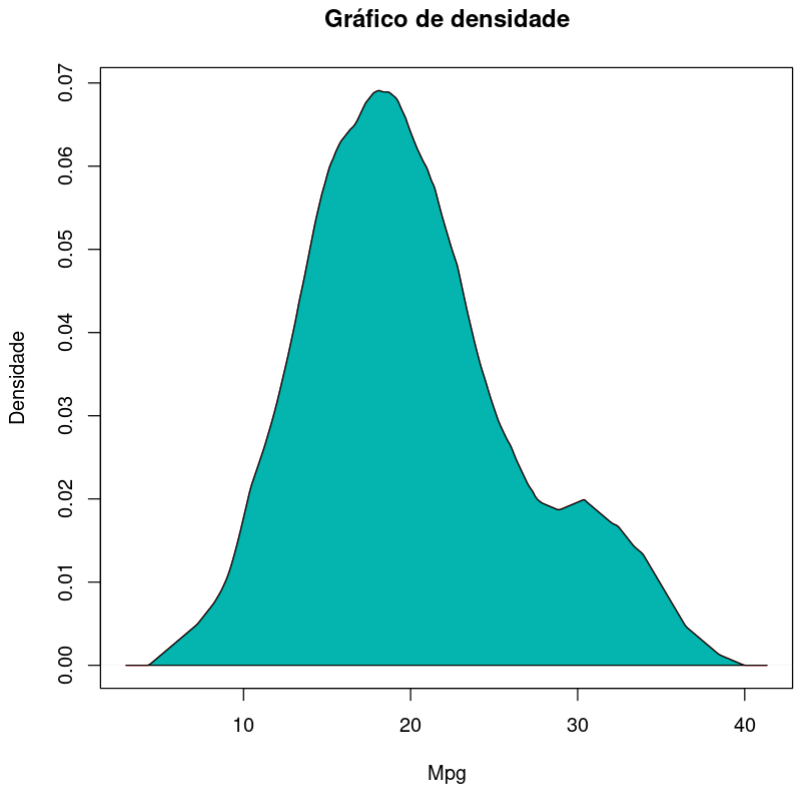


E como faço caso deseje colorir o interior da curva de densidade?

Para isso podemos usar a função *polygon*

In [81]:

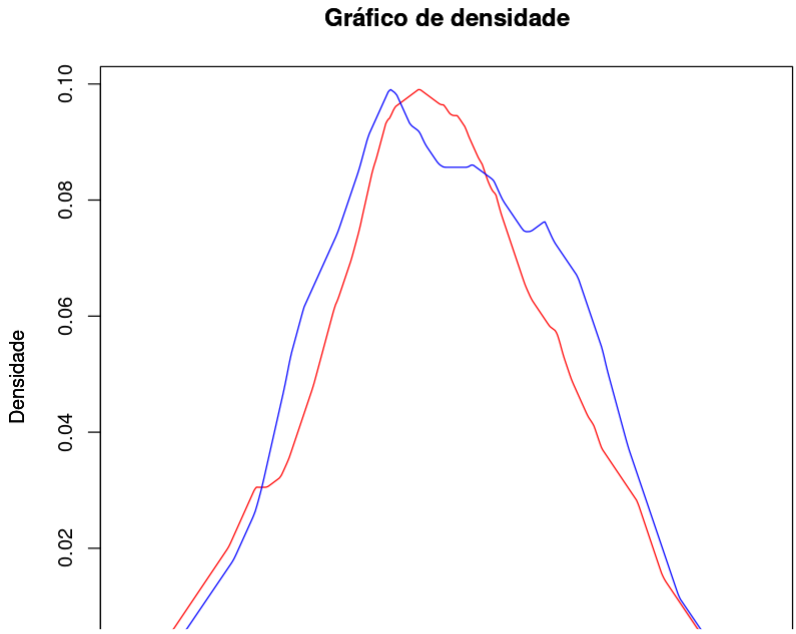
```
plot(obj, col = "red", main = "Gráfico de densidade", xlab= "Mpg", ylab = "Densidad
polygon(obj, col = "#04B4AE")
```



Suponha que tenhamos uma variável categórica no banco de dados e desejamos obter a densidade de outra variável para cada uma das categorias. Para tanto, precisamos criar os objetos de densidade para cada categoria e obter o gráfico com a função *plot*, indicando que iremos adicionar novos gráficos com a função *par*.

In [92]:

```
obj1 <- density(x = mtcars$mpg[mtcars$am == 0], kernel = "triangular")
obj2 <- density(x = mtcars$mpg[mtcars$am == 1], kernel = "triangular")
#
plot(obj1, col = "red", main = "Gráfico de densidade", xlab= "Mpg", ylab = "Densida
par(new = TRUE)
plot(obj2, col = "blue", main = "Gráfico de densidade", xlab= "Mpg", ylab = "Densid
```

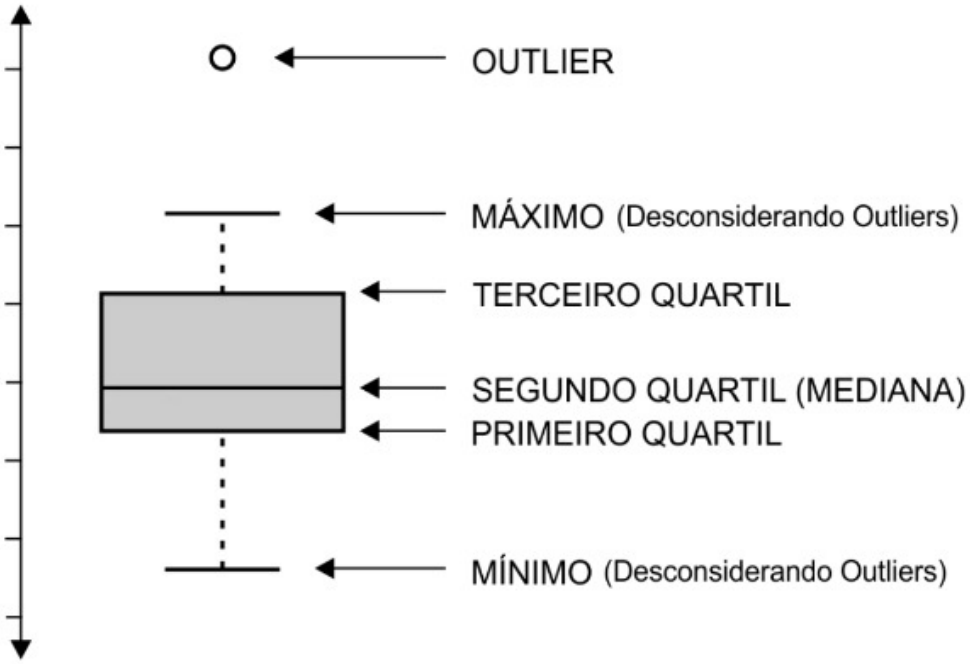


5. Boxplot

O boxplot - ou diagrama de caixa - é uma ferramenta gráfica que permite visualizar a distribuição e valores discrepantes dos dados.

In [96]:

```
library("IRdisplay")
display_png(file="boxplot.png", width = 500)
```



O R disponibiliza uma função nativa na qual podemos obter o diagrama de caixa. Basicamente, os parâmetros da função são:

boxplot(x, range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, outline = TRUE, names, plot = TRUE, border = par("fg"), col = NULL, log = "", pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5), ann = !add, horizontal = FALSE, add = FALSE, at = NULL)

Em que:

x é o vetor de dados que o usuário deseja plotar.

range é o grau de "rigidez" na escolha dos outliers.

width é a largura da caixa.

varwidth se *varwidth= TRUE* as caixas são desenhadas com larguras proporcionais às raízes quadradas do número de observações nos grupos.

outline Se *outline = FALSE* os outliers não são expostos.

horizontal se *horizontal = TRUE* as caixas são apresentadas no sentido horizontal.

col indica a cor da caixa.

border indica a cor das linhas das caixas.

xlim são os limites específicos do eixo x que o usuário deseja plotar.

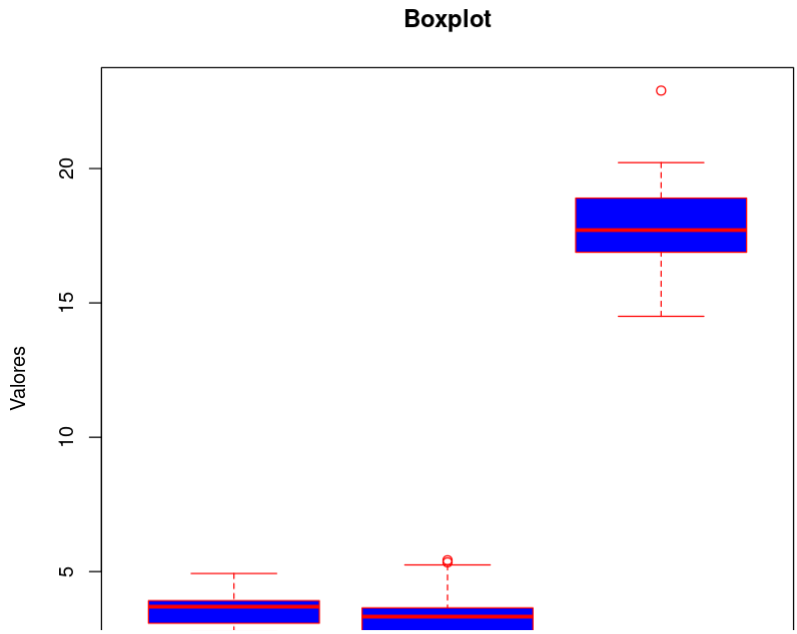
ylim são os limites específicos do eixo y que o usuário deseja plotar.

xlab indica o título do eixo x.

ylab indica o título do eixo y.

In [2]:

```
boxplot(mtcars[,5:7], range= 1.5, width = NULL, varwidth = F, outline= T,
        names = c("Variável 1", "Variável 2", "Variável 3"), col = "blue", border =
       main = "Boxplot", xlab = "Variáveis", ylab = "Valores")
```



6. ggplot2

ggplot2 é uma biblioteca de construção avançada de ferramentas de exposição de dados. Com o ggplot2 o usuário pode desenvolver gráficos e figuras com uma boa qualidade visual e com uma série de opções de edição que não estão disponíveis nas bibliotecas nativas do R.

In [ ]:

```
install.packages("ggplot2")
```

In [1]:

```
library(ggplot2)
```

Existem várias maneiras de elaborar um gráfico usando as ferramentas da biblioteca ggplot2. Aqui, vamos seguir o seguinte padrão.

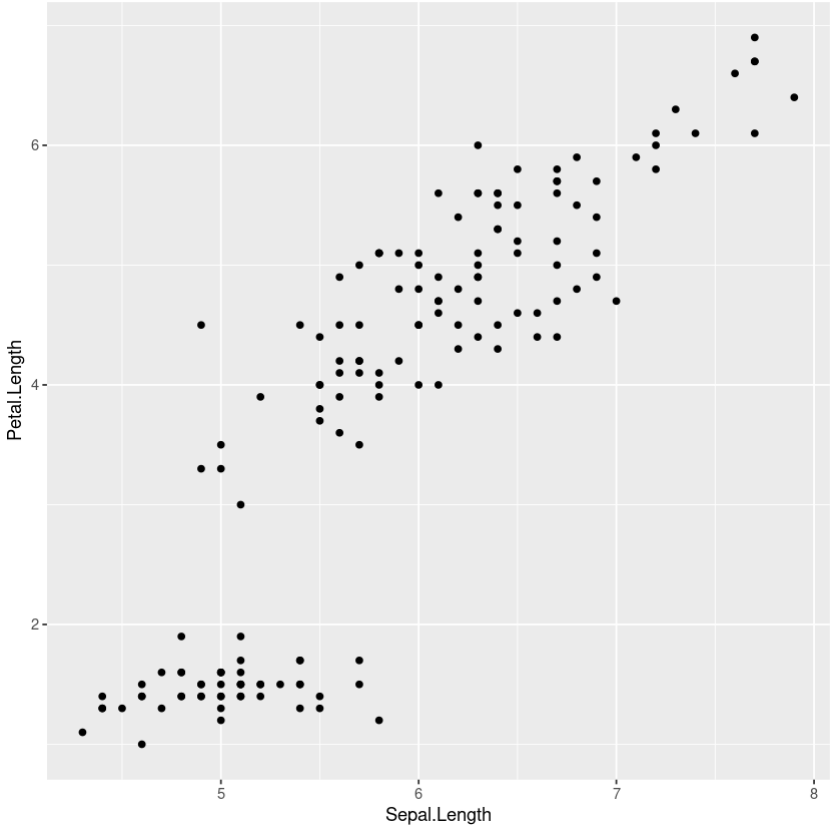
ggplot(data = banco_de_dados) +

geometria(aes(x = eixo_x, y= eixo_y, ...) + ...

6.1 Gráfico de dispersão

In [8]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length))
```



Podemos melhorar a visualização do gráfico usando os seguintes parâmetros:

xlab: Nome do eixo x.

ylab Nome do eixo y.

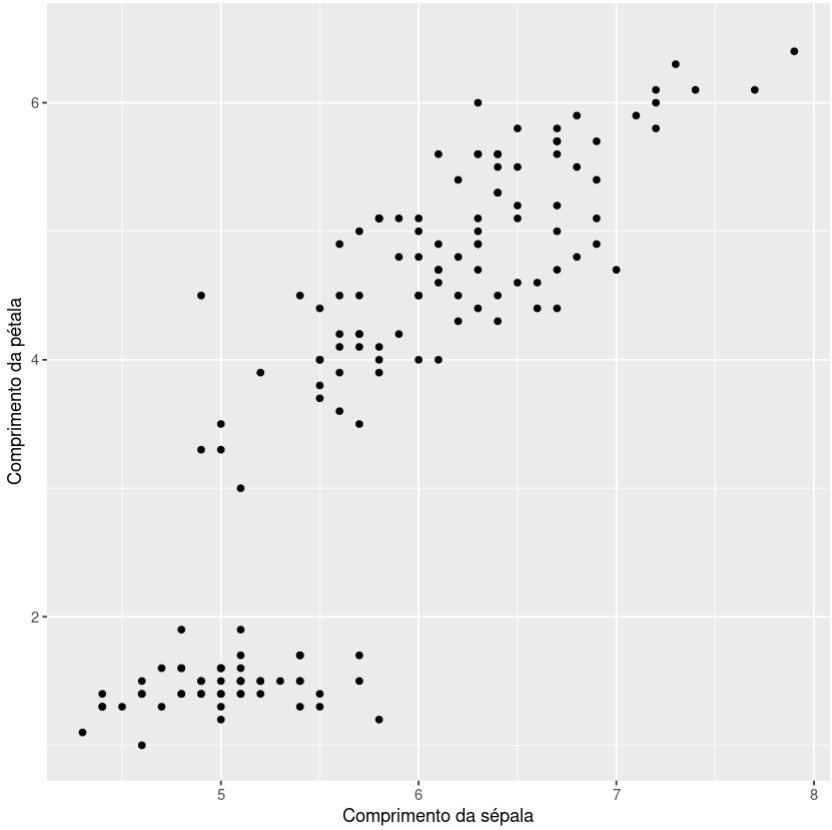
xlim: Limites inferior e superior do eixo x.

ylim Limites inferior e superior do eixo y.

In [9]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
ylim(1, 6.5)
```

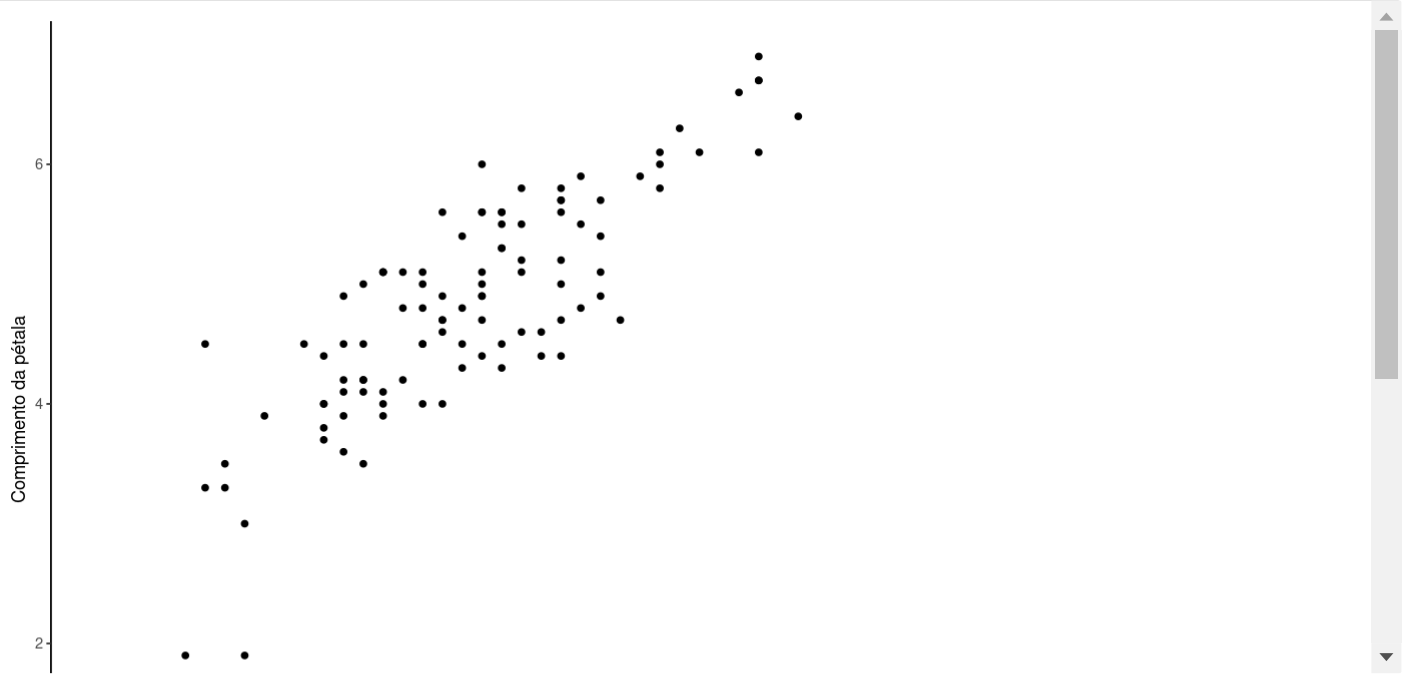
```
Warning message:
“Removed 4 rows containing missing values (geom_point).”
```



Com o parâmetro *theme*, podemos escolher um tema para melhorar a exposição do gráfico.

In [11]:

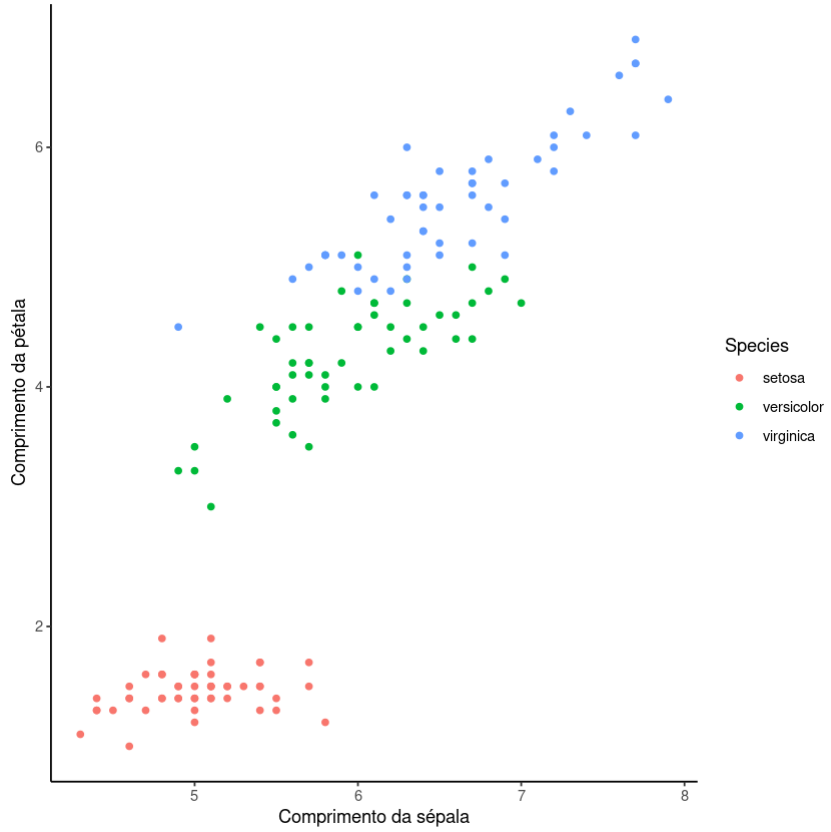
```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
theme_classic()
```



Suponha que desejamos destacar os pontos de acordo com uma variável categórica. Podemos fazer isto indicando esta variável no parâmetro *color*, dentro do bloco *aes*.

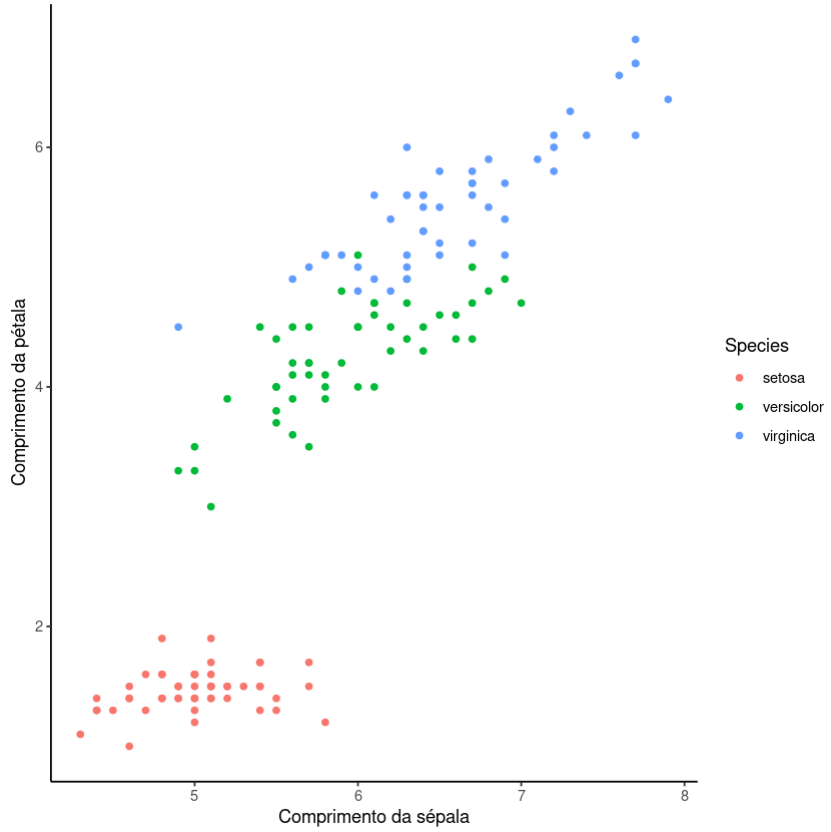
In [14]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
theme_classic()
```



In [17]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length, colour = Species)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
theme_classic()
```

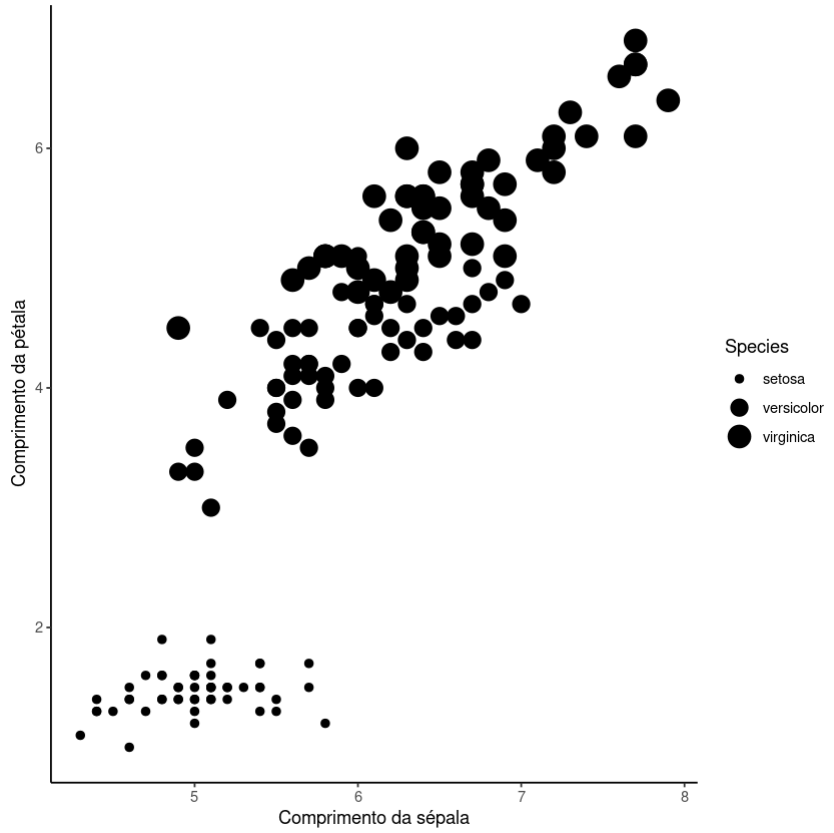


Em vez disso suponha que queiramos diferenciar uma variável contínua de acordo com uma determinada variável categórica por meio de diferentes tamanhos de pontos no gráfico. Podemos fazer isso indicando esta variável categórica no parâmetro *size*, dentro do bloco *aes*.

In [18]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length, size = Species)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
theme_classic()
```

```
Warning message:
“Using size for a discrete variable is not advised.”
```

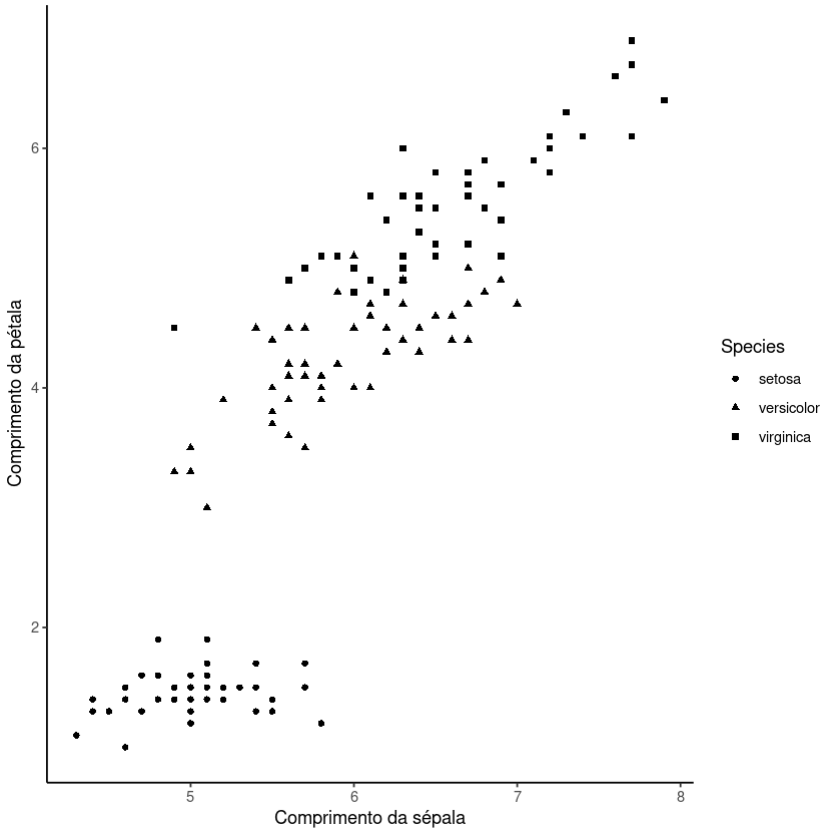


Agora suponha que em vez de diferenciarmos as classes por tamanho dos pontos, queiramos expor um

símbolo para cada categoria no gráfico. Podemos fazer isso indicando as categorias no parâmetro shape, dentro do bloco aes.

In [19]:

```
ggplot(data = iris) +
geom_point(aes(x = Sepal.Length, y = Petal.Length, shape = Species)) +
xlab("Comprimento da sépala")+
ylab("Comprimento da pétala") +
theme_classic()
```



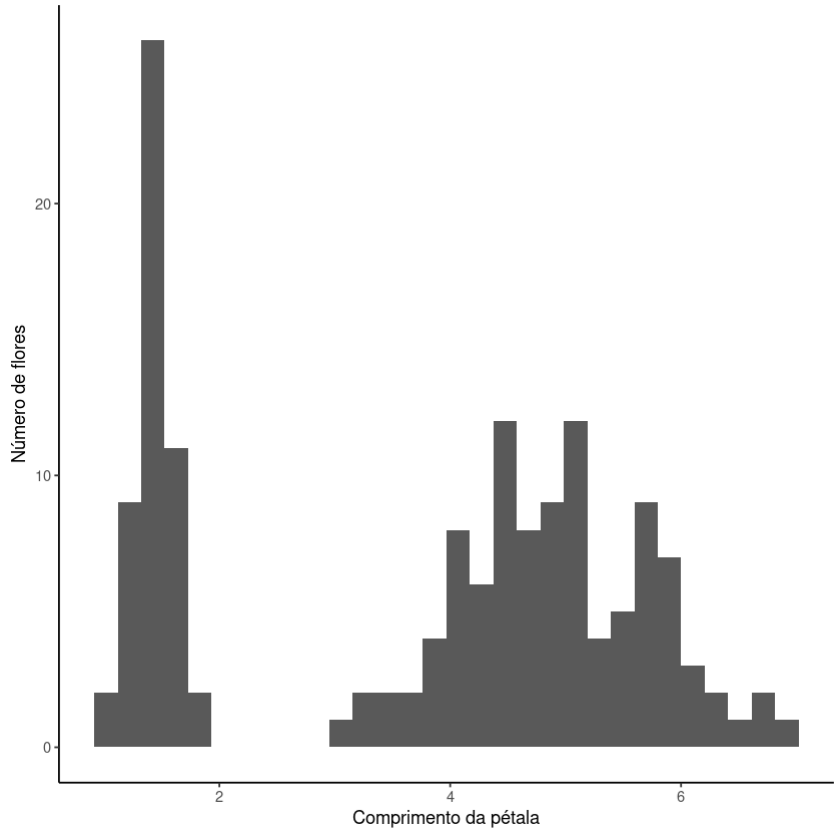
6.2 Histograma

No histograma precisamos indicar apenas a variável que compõe o eixo x e utilizar a geometria do tipo *geom_histogram()*.

In [65]:

```
ggplot(data = iris) +
geom_histogram(aes(x = Petal.Length))+
xlab("Comprimento da pétala")+
ylab("Número de flores") +
theme_classic()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

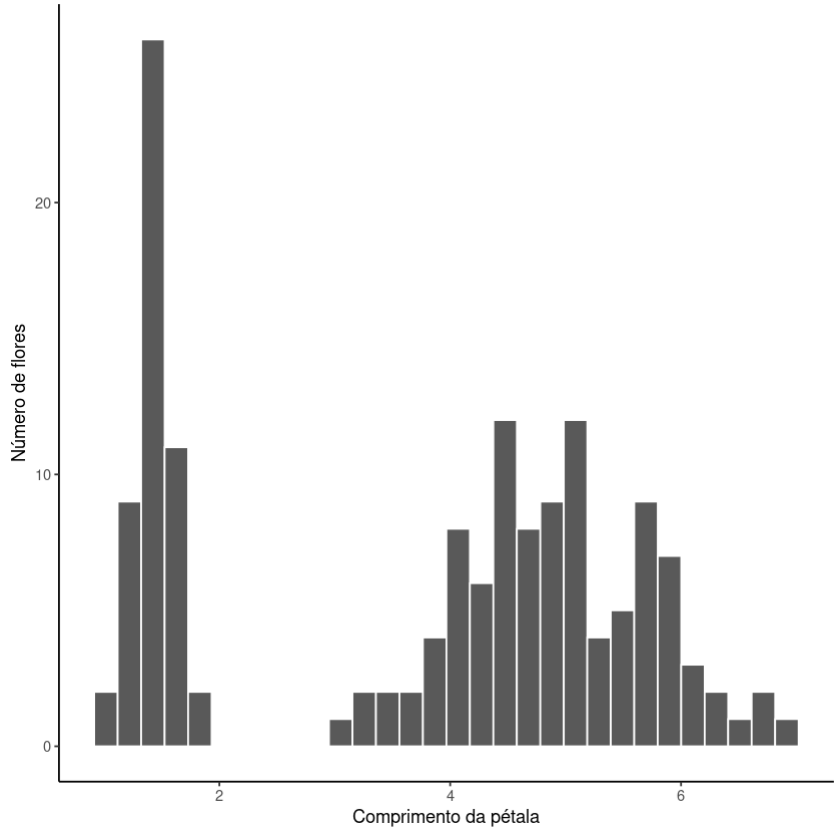


Podemos separar cada barra por uma determinada cor usando o parâmetro *color*.

In [66]:

```
ggplot(data = iris) +
geom_histogram(aes(x = Petal.Length), color = "white")+
xlab("Comprimento da pétala")+
ylab("Número de flores") +
theme_classic()
```

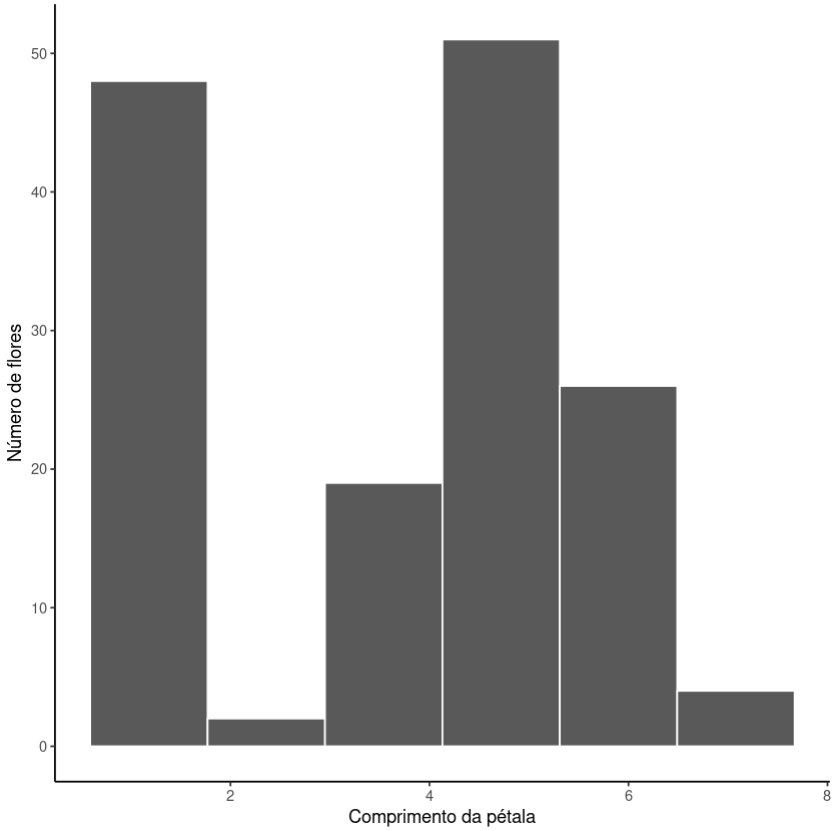
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Podemos delimitar o número de barras do histograma por meio do parâmetro *bins*.

In [67]:

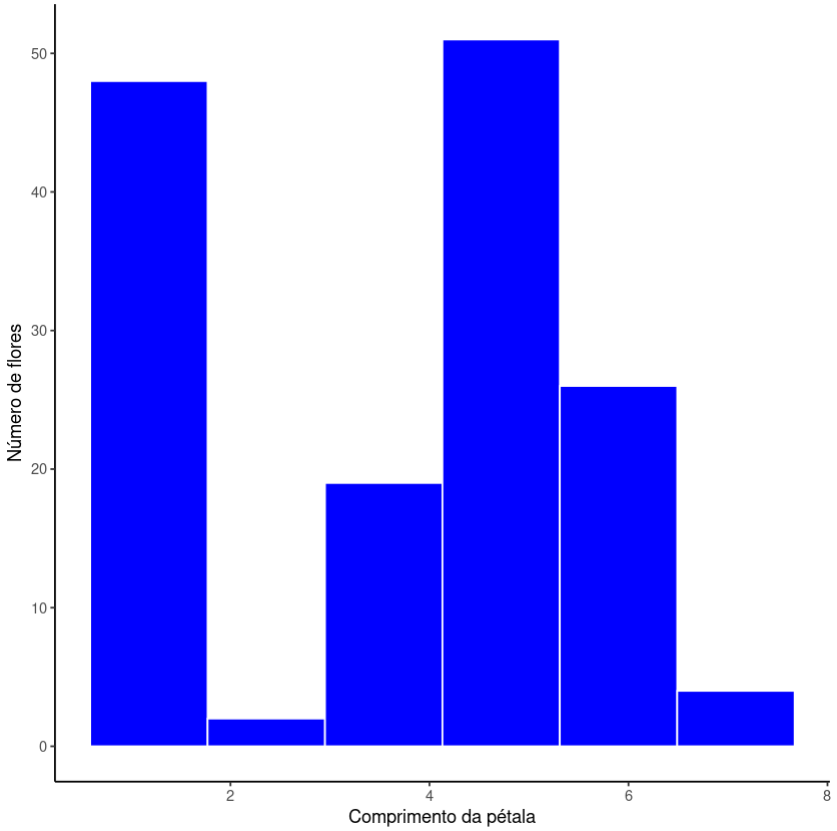
```
ggplot(data = iris) +
geom_histogram(aes(x = Petal.Length), color = "white", bins = 6)+
xlab("Comprimento da pétala")+
ylab("Número de flores") +
theme_classic()
```



Podemos alterar a cor das barras usando o parâmetro *fill*.

In [68]:

```
ggplot(data = iris) +
geom_histogram(aes(x = Petal.Length), color = "white", bins = 6, fill = "blue")+
xlab("Comprimento da pétala")+
ylab("Número de flores") +
theme_classic()
```

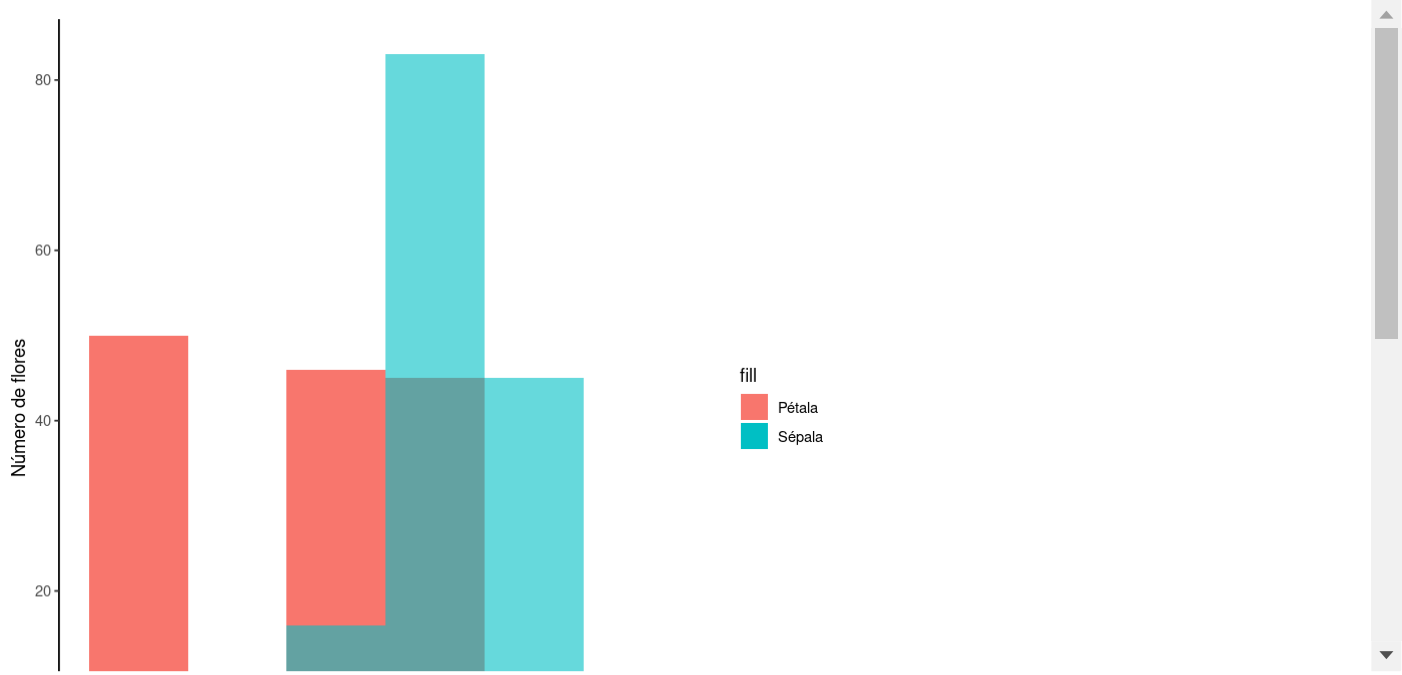


E se quisermos inserir um outro histograma no mesmo gráfico?

Primeiro precisamos usar cores diferentes. Depois podemos usar o parâmetro *alpha* para inserir um determinado grau de transparência em um dos gráficos.

In [22]:

```
ggplot(data = iris) +
  geom_histogram(aes(x = Petal.Length, fill= "Pétala"), bins = 6)+
  geom_histogram(aes(x = Sepal.Length, fill = "Sépala"), bins = 6, alpha = .6)+
  xlab("Comprimento")+
  ylab("Número de flores") +
  theme_classic()
```



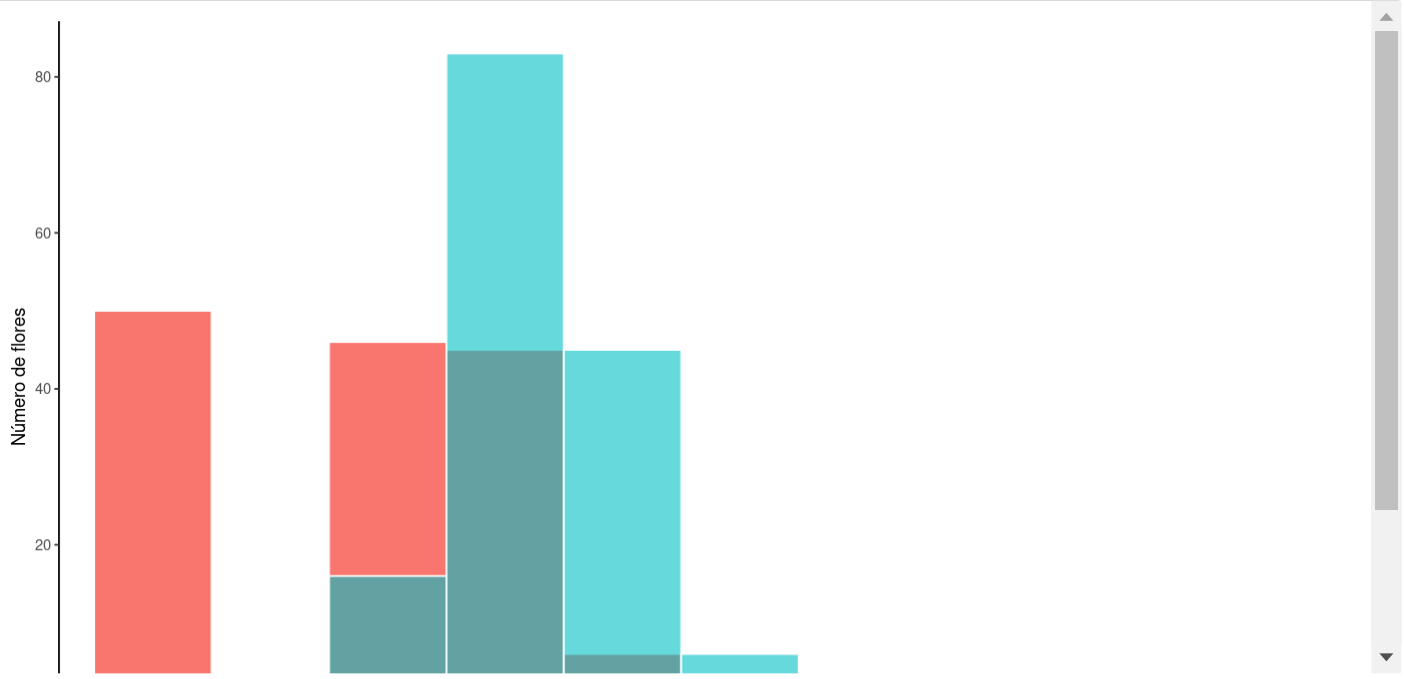
6.2.1 Legendas

E se quisermos alterar os elementos da legenda?

Para isso, precisamos configurar as opções do tema utilizado por meio da função *theme*

In [26]:

```
ggplot(data = iris) +
  geom_histogram(aes(x = Petal.Length, fill= "Pétala"), bins = 6, color ="white")+
  geom_histogram(aes(x = Sepal.Length, fill = "Sépala"), bins = 6, color ="white",
  xlab("Comprimento")+
  ylab("Número de flores") +
  theme_classic()+
  # Alterando a fonte, cor e tamanho da legenda
  theme(legend.title = element_text(family = "Times", colour = "blue", size = 12))+
  # Alterando a posição da legenda
  theme(legend.position = "bottom")+
  # Alterando o título da legenda
  guides(fill=guide_legend(title= "Parte da flor:"))
```

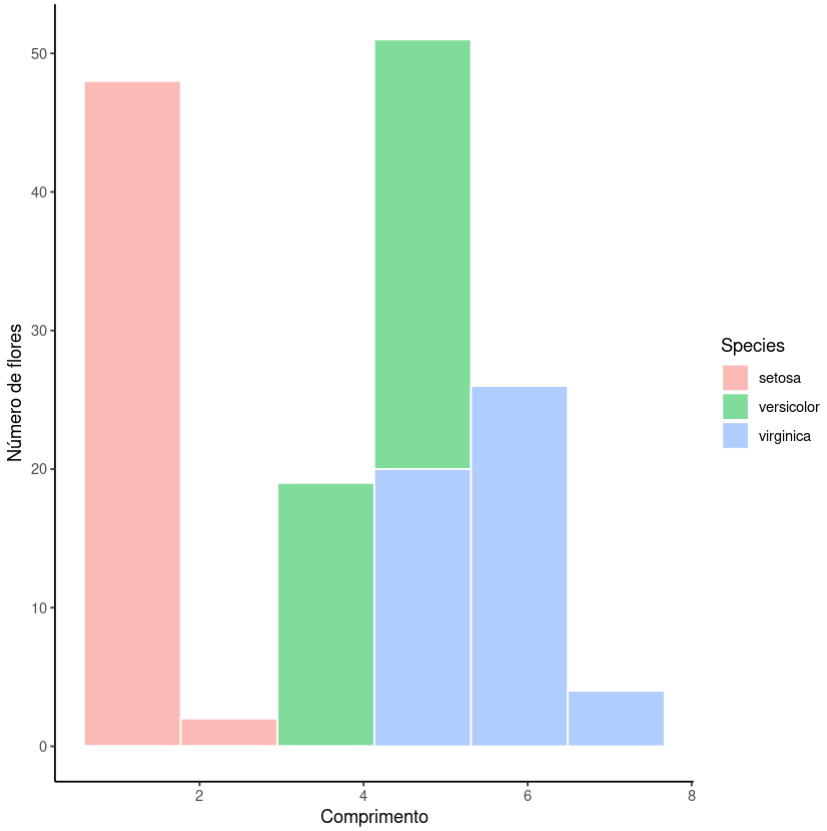


6.2.2 Condicionando o histograma a uma variável categórica

Podemos criar um histograma de uma determinada coluna de acordo com as categorias de uma variável categórica. Para isso, podemos indicar esta variável no parâmetro *fill*.

In [31]:

```
ggplot(data = iris) +
  geom_histogram(aes(x = Petal.Length, fill= Species), bins = 6, color ="white", al
  xlab("Comprimento")+
  ylab("Número de flores") +
  theme_classic()
```

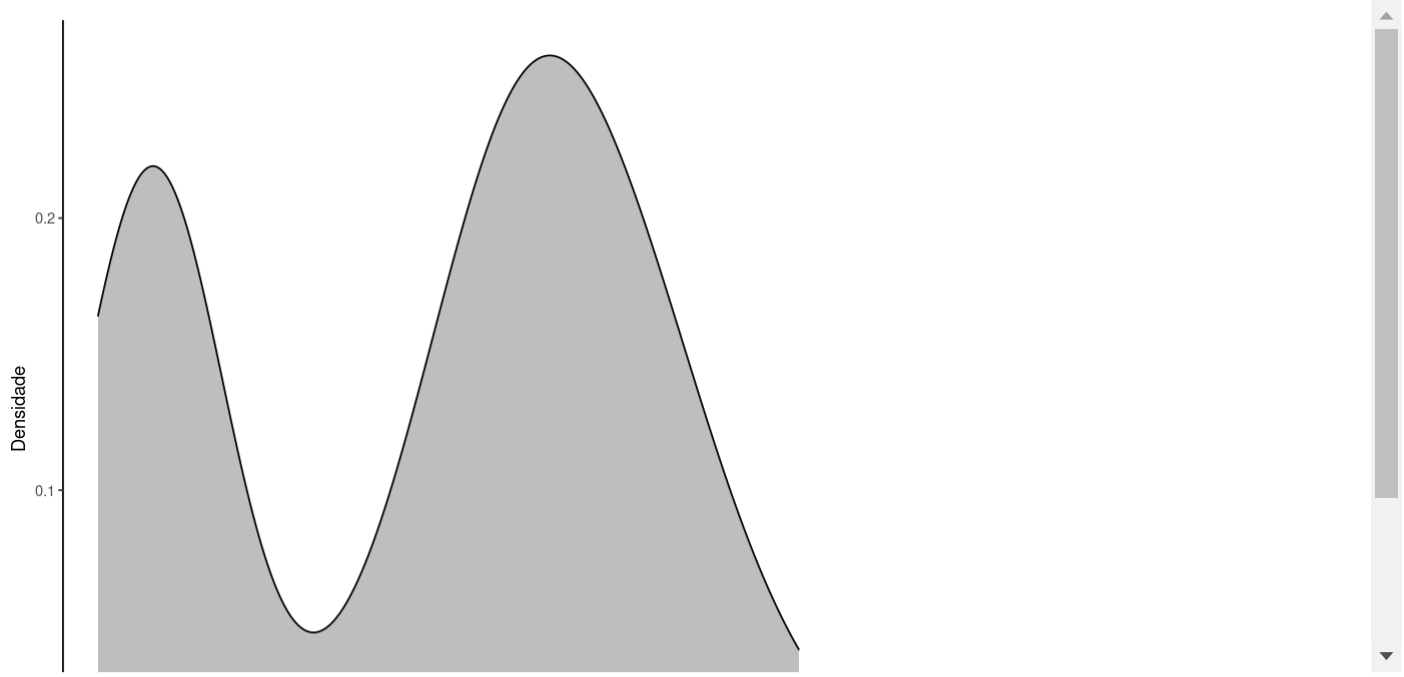


6.3 Gráfico de densidade

Assim como no histograma, no gráfico de densidade precisamos indicar apenas a variável que compõe o eixo x. No entanto, nesse tipo de gráfico iremos utilizar a geometria do tipo geom_density().

In [50]:

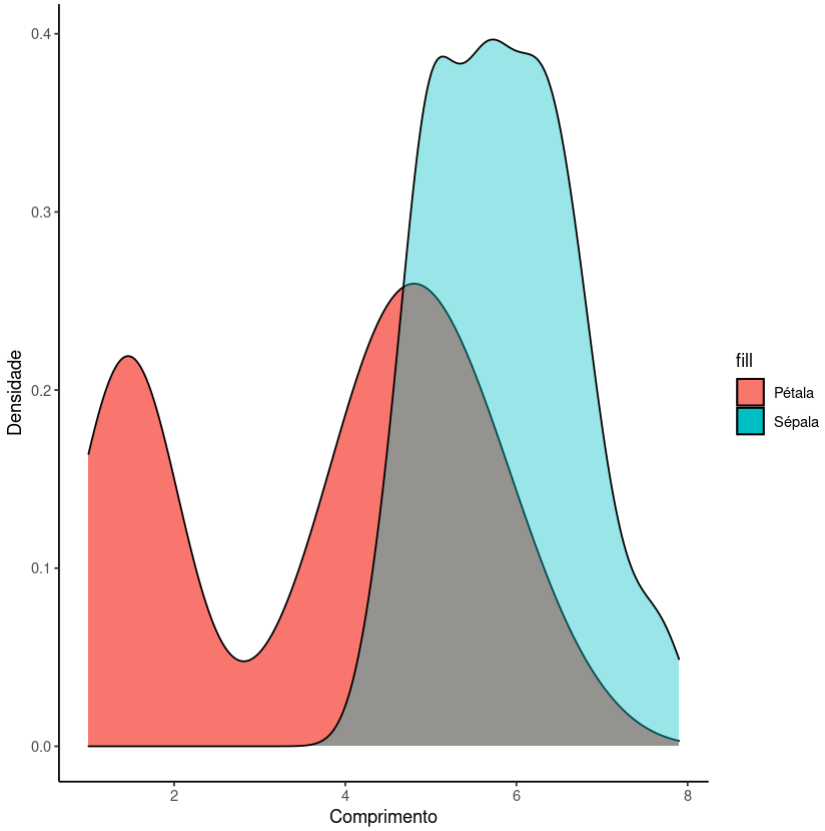
```
ggplot(data = iris) +
geom_density(aes(x = Petal.Length), fill = "gray") +
xlab("Comprimento")+
ylab("Densidade") +
theme_classic()
```



Para duas densidades em um único gráfico, o procedimento é semelhante ao que foi feito para os histogramas.

In [35]:

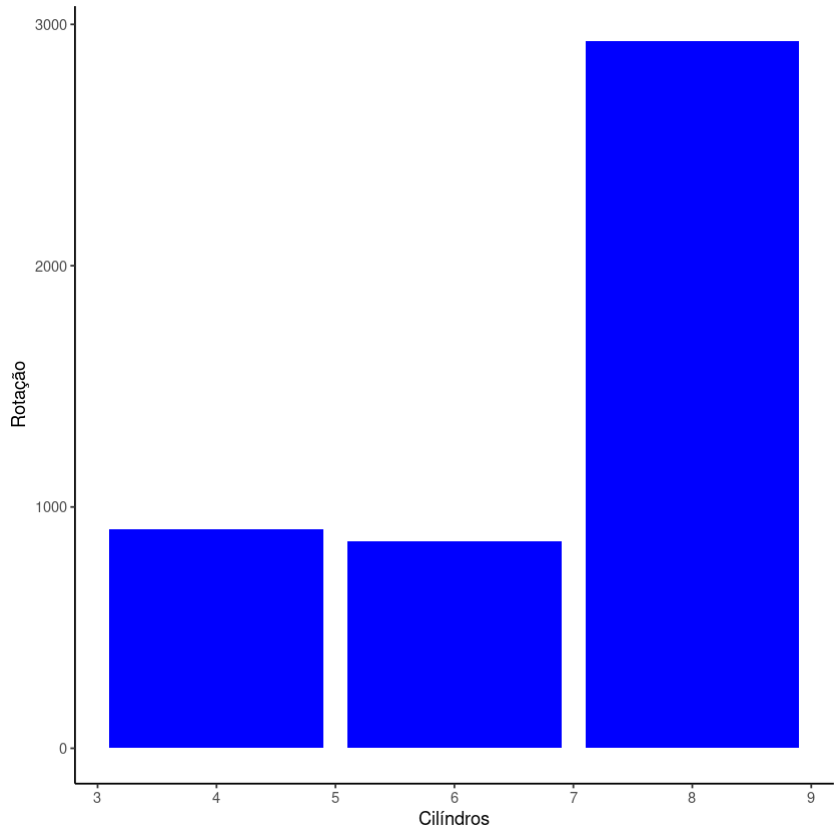
```
ggplot(data = iris) +
geom_density(aes(x = Petal.Length, fill = "Pétala")) +
geom_density(aes(x = Sepal.Length, fill = "Sépala"), alpha = .4) +
xlab("Comprimento")+
ylab("Densidade") +
theme_classic()
```



6.4 Gráfico de barras

In [149]:

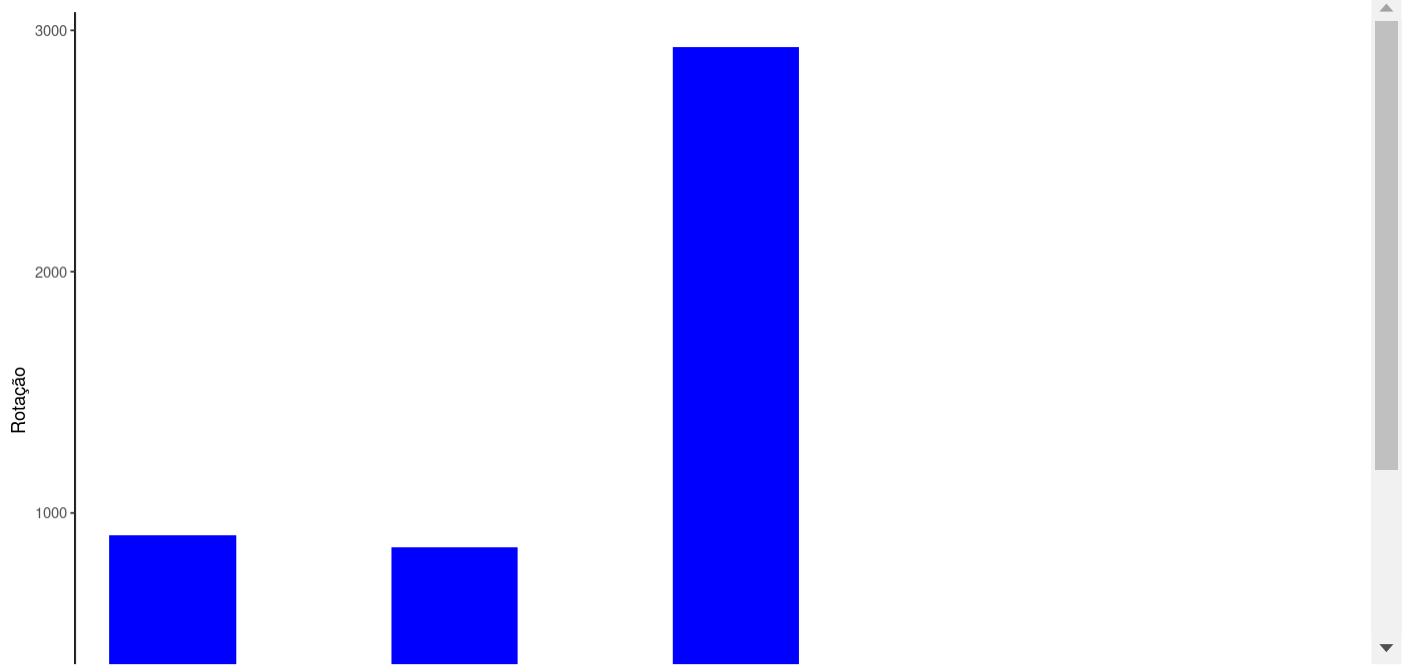
```
ggplot(data = mtcars) +
geom_bar(aes(y = hp, x = cyl), fill = "blue", stat="identity") +
xlab("Cilíndros")+
ylab("Rotação") +
theme_classic()
```



Podemos alterar a largura das barras por meio do parâmetro *width*

In [151]:

```
ggplot(data = mtcars) +
geom_bar(aes(y = hp, x = cyl), fill = "blue", stat="identity", width = .9) +
xlab("Cilíndros")+
ylab("Rotação") +
theme_classic()
```



Podemos adicionar o parâmetro *coord_flip()* para indicar que as barras devem ser expostas na horizontal.

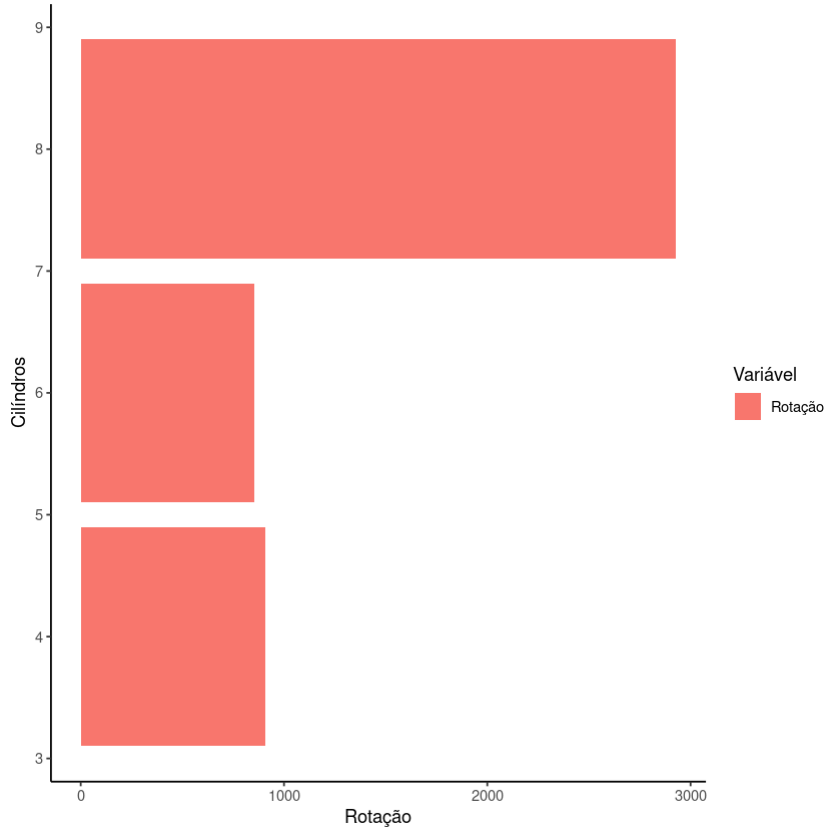
In [48]:

```
ggplot(data = mtcars) +
geom_bar(aes(y = hp, x = cyl), stat="identity", fill = "blue") +
xlab("Cilíndros")+
ylab("Rotação") +
theme_classic() +
coord_flip()
```

Para adicionar uma legenda às barras basta fazer a indicação no parâmetro *fill* dentro do bloco *aes*.

In [54]:

```
ggplot(data = mtcars) +
geom_bar(aes(y = hp, x = cyl, fill = "Rotação"), stat="identity") +
xlab("Cilíndros")+
ylab("Rotação") +
theme_classic() +
coord_flip() +
guides(fill=guide_legend(title= "Variável"))
```



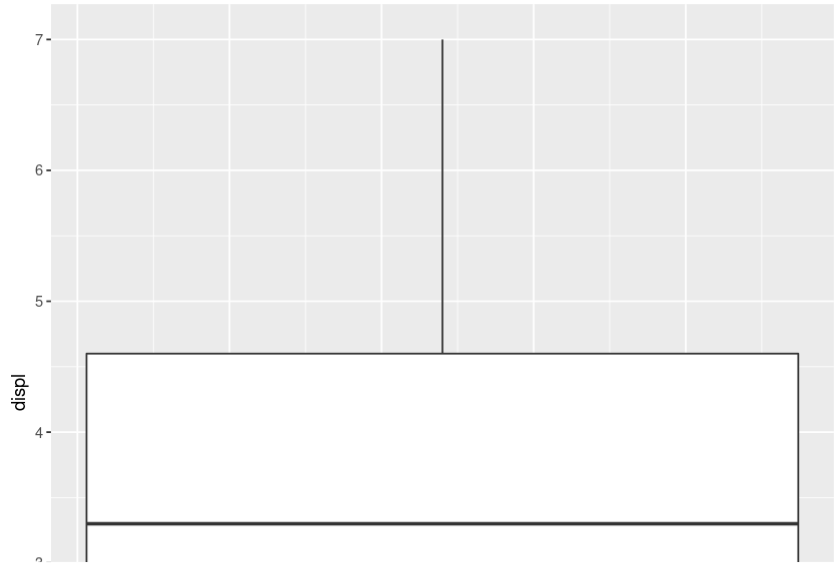
6.5 Boxplot

Para construir um gráfico do tipo boxplot com o ggplot2 devemos usar a geometria *geom_boxplot*. Dentro do bloco *aes* devem ser especificadas as variáveis que irão aparecer de acordo com o eixo x e a variável do eixo y.

In [131]:

```
ggplot(data = mpg, aes(x = cty, y = displ)) + geom_boxplot()
```

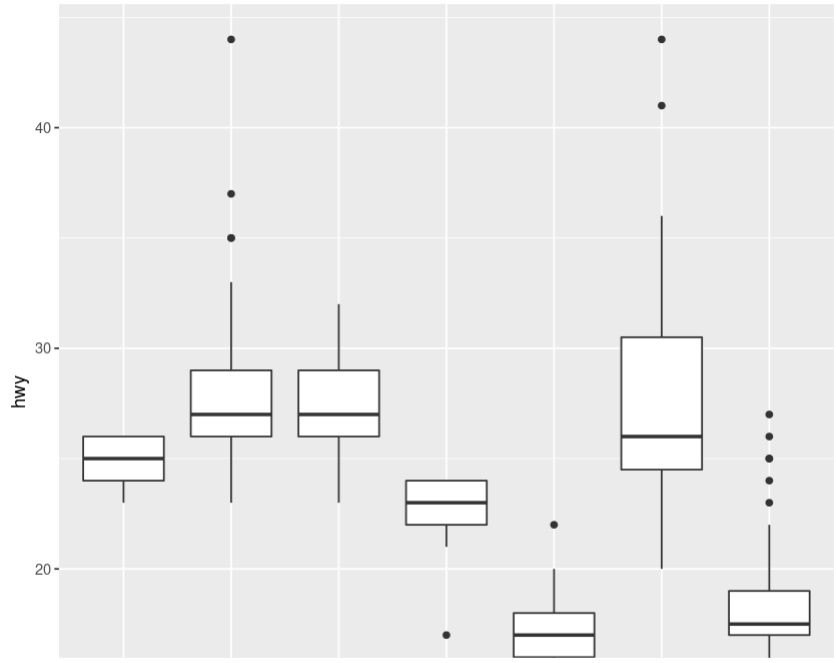
```
Warning message:
“Continuous x aesthetic -- did you forget aes(group=...)?”
```



Caso a variável x seja categórica, o output gerado será semelhante ao gráfico abaixo:

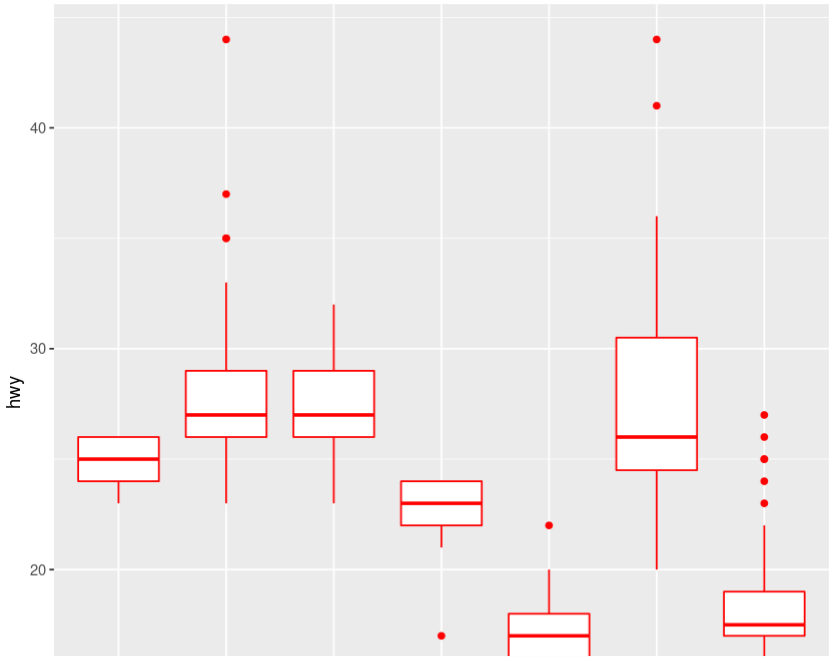
In [133]:

```
ggplot(data = mpg, aes(x = class, y = hwy)) + geom_boxplot()
```



In [136]:

```
ggplot(data = mpg, aes(x = class, y = hwy)) + geom_boxplot(fill = "white", colour =
```

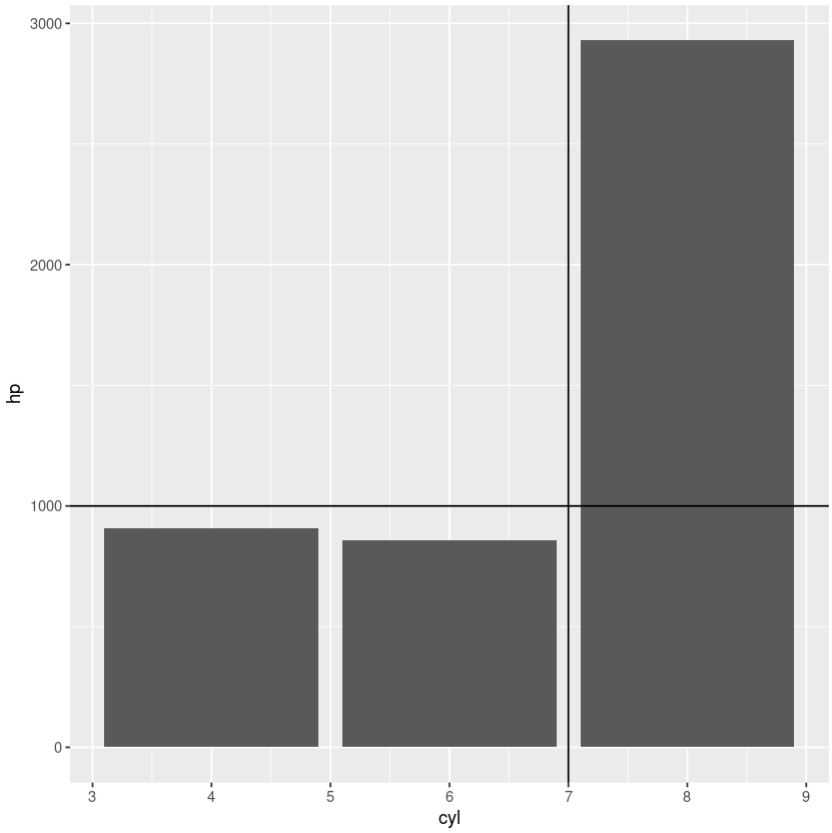


6.6 Adicionando linhas verticais e horizontais

Para adicionar uma linha vertical, use a geometria *geom_vline*. Para adicionar uma linha horizontal, use a geometria *geom_hline*.

In [137]:

```
ggplot(data = mtcars) +
geom_bar(aes(y = hp, x = cyl), stat="identity") +
geom_vline(aes(xintercept = 7)) +
geom_hline(aes(yintercept = 1000))
```

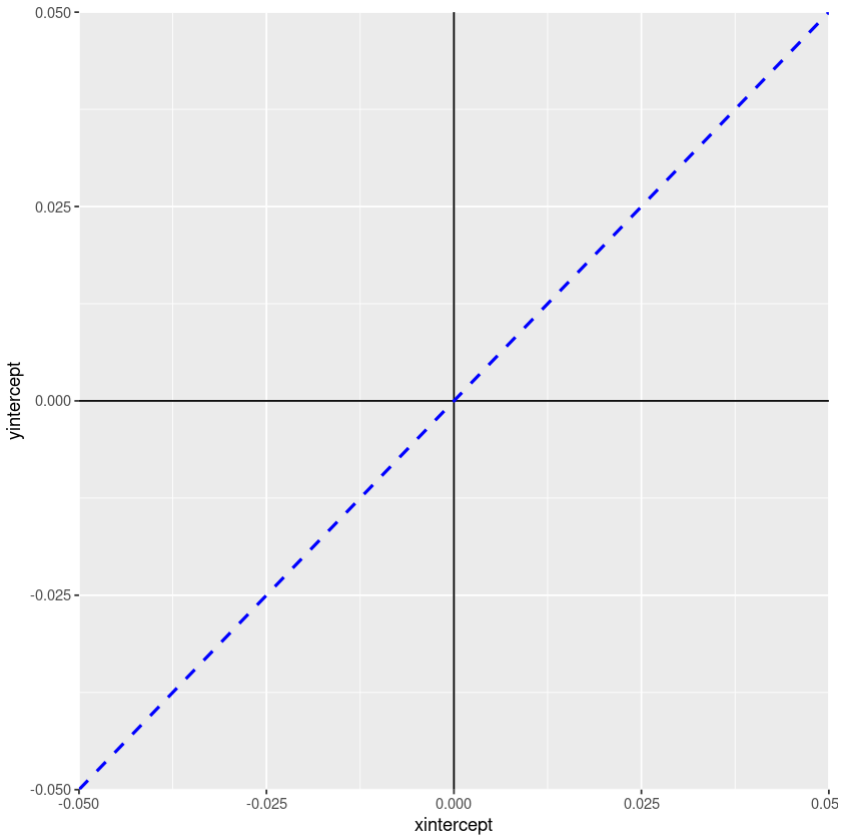


6.7 Adicionando linhas longitudinais

Para adicionar uma linha longitudinal, use a geometria *geom_abline*.

In [91]:

```
ggplot()+
geom_vline(xintercept = 0)+
geom_hline(yintercept = 0) +
geom_abline(intercept = 0, slope = 1, linetype = "dashed", size = .9, color = "blue
```

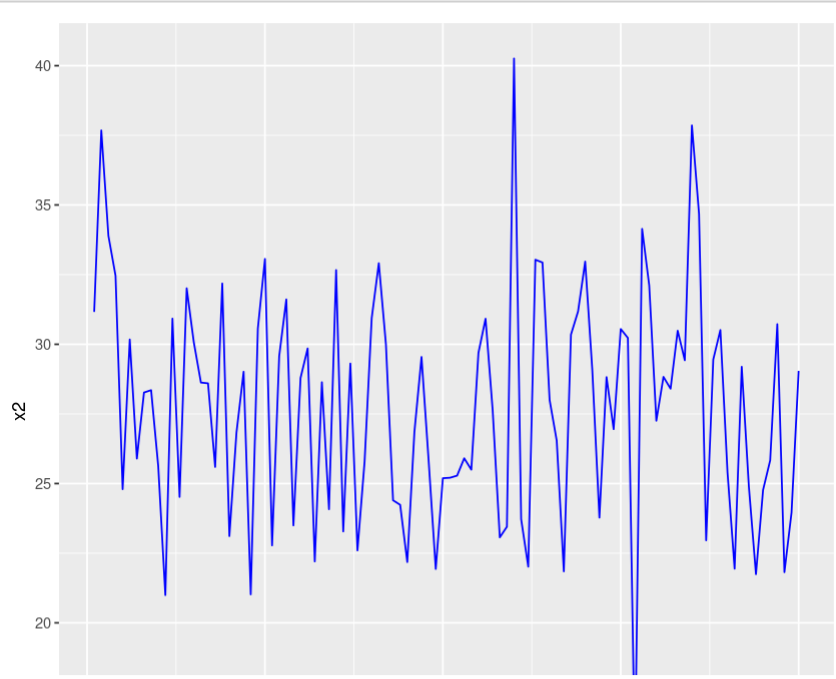


6.8 Gráfico de linhas

Para construir um gráfico de linhas, devemos usar a geometria *geom_line*.

In [16]:

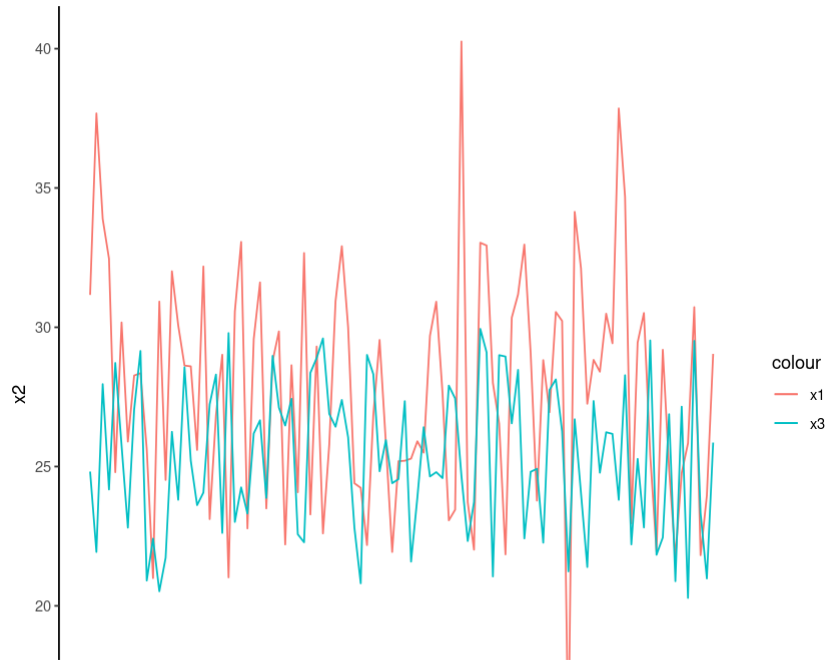
```
df  <- data.frame(x1 = 1:100, x2 = rnorm(100, 28, 4), x3 = runif(100, 20, 30))
ggplot(data = df, aes(x = x1, y = x2)) + geom_line(color = "blue")
```



6.8.1 Múltiplas linhas em un único gráfico

In [35]:

```
ggplot(data = df, aes(x = x1))+
geom_line(aes(y = x2, color = "x1"))+
geom_line(aes(y = x3, color = "x3")) +
theme_classic()
```

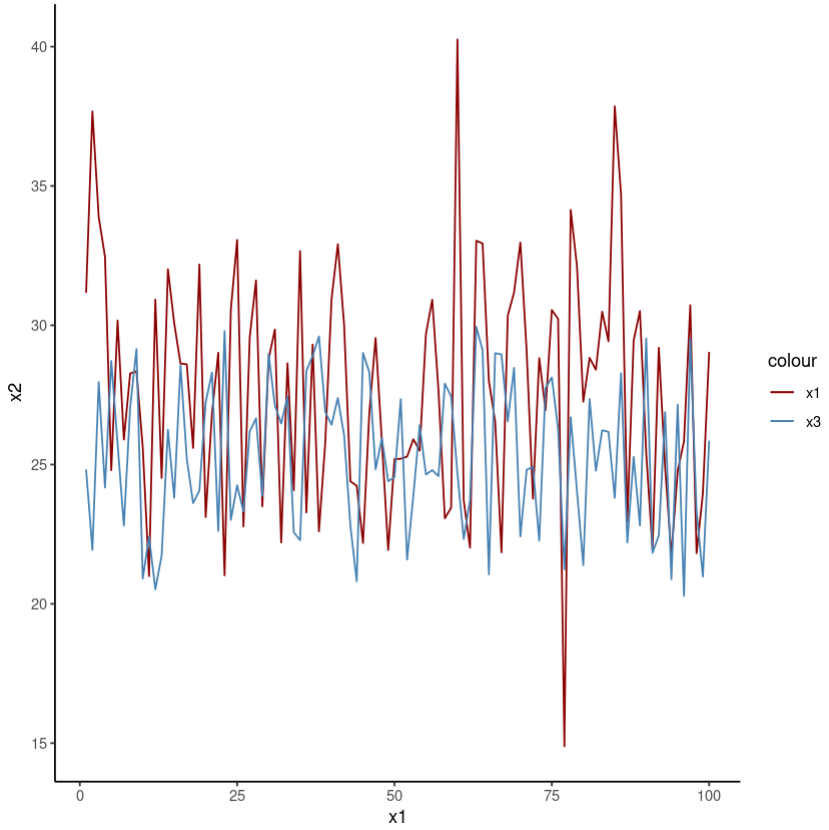


E caso o usuário queira alterar as cores de cada linha?

Nesse caso, estas alterações devem ser indicadas no bloco *scale_color_manual*.

In [29]:

```
ggplot(data = df, aes(x = x1))+
geom_line(aes(y = x2, color = "x1"))+
geom_line(aes(y = x3, color = "x3")) +
scale_color_manual(values = c("darkred", "steelblue"))+
theme_classic()
```

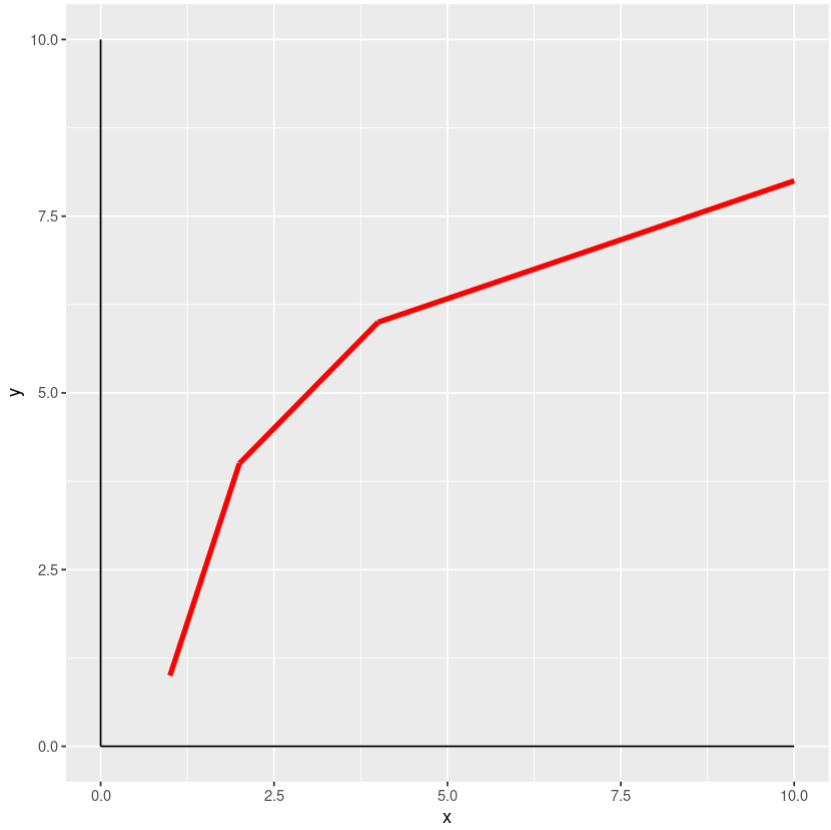


6.9 Inserindo coordenadas manualmente

Caso o usuário deseje construir um gráfico de linhas inserindo cada informação de maneira manual, entãoele deve usar a geometria *geom_segment*.

In [59]:

```
ggplot()+
geom_segment(aes(x = 0, y = 0, xend = 10, yend = 0))+
geom_segment(aes(x = 0, y = 0, xend = 0, yend = 10))+
geom_segment(aes(x = 1, y = 1, xend = 2, yend = 4), color = "red", size = 1.5)+
geom_segment(aes(x = 2, y = 4, xend = 4, yend = 6), color = "red", size = 1.5) +
geom_segment(aes(x = 4, y = 6, xend = 10, yend = 8), color = "red", size = 1.5)
```

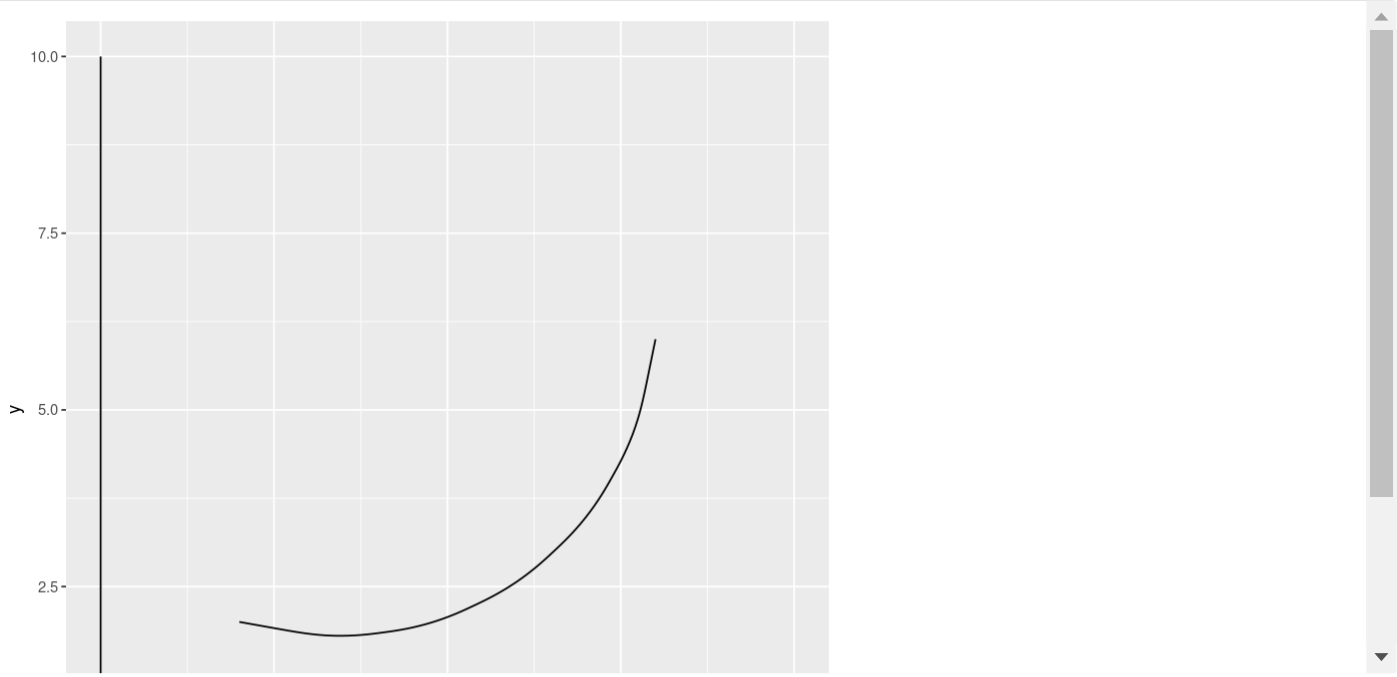


6.10 Criando uma curva

Para criar uma curva de acordo com as propriedades desejadas, o usuário pode usar a geometria *geom_curve*.

In [76]:

```
ggplot()+
geom_segment(aes(x = 0, y = 0, xend = 10, yend = 0))+
geom_segment(aes(x = 0, y = 0, xend = 0, yend = 10))+
geom_curve(aes(x = 2, y = 2, xend = 8, yend = 6), curvature = 0.5)
```



6.11 Mapas

Para plotar variáveis distribuídas no espaço em um mapa de dispersão, precisamos usar um shapefile da área territorial de estudo. Também precisamos usar o ggplot2 em congunto com o pacote *sf*. Para isso, o usuário deverá instalar e importar a biblioteca *sf*.

In [77]:

```
install.packages(sf)
library(sf)
```

```
Linking to GEOS 3.8.0, GDAL 3.0.4, PROJ 6.3.1
```

Para exemplificar, vamos usar um shapefile do estado do Ceará disponibilizado pelo Instituto de Pesquisa Econômica Aplicada (IPEA) no endereço https://www.ipea.gov.br/ipeageo/malhas (https://www.ipea.gov.br/ipeageo/malhas). Inicialmente, vamos fazer o download do arquivo em formato .zip

In [79]:

```
temp <- tempfile()
download.file("https://www.ipea.gov.br/ipeageo/arquivos/malhas/CE_Mun97_region.zip"
# A seguir, iremos descompactar a pasta
unzip("CE_Mun97_region.zip")
unlink(temp)
```

Tendo baixado e descompactado os arquivos do shapefile, o próximo passo é fazer a leitura do arquivo no formato *shp*. Para tanto, usaremos a função *st_read* do pacote *sf*.

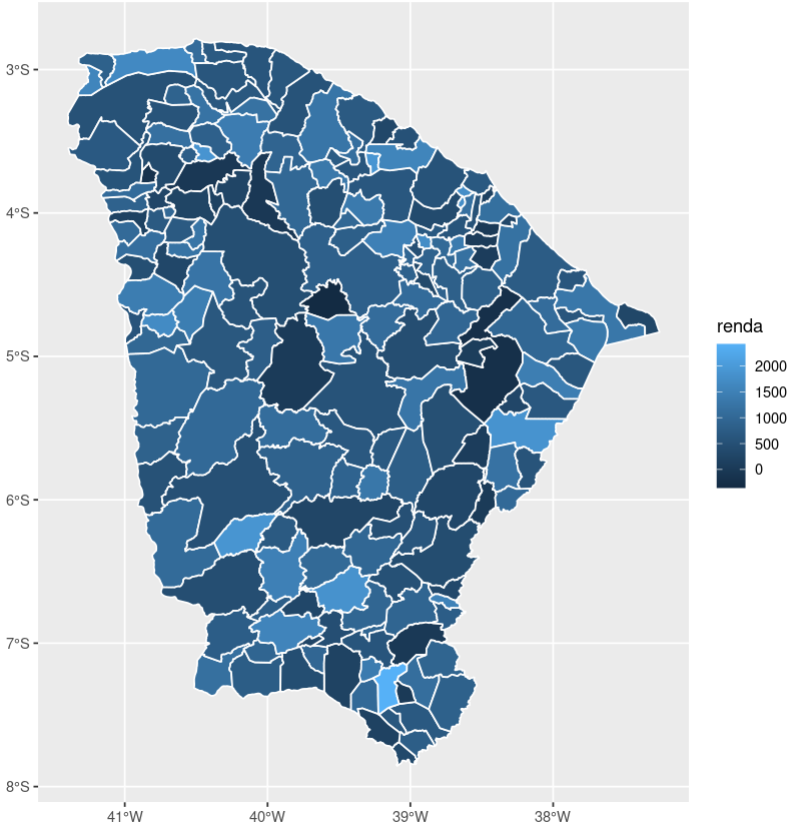
In [82]:

```
shape <- st_read("CE_Mun97_region.shp")
```

```
Reading layer `CE_Mun97_region' from data source `/home/helson/trabalh
os/CURSO_R_PPGER/Aula4/CE_Mun97_region.shp' using driver `ESRI Shapefi
le'
Simple feature collection with 184 features and 19 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -41.3999 ymin: -7.855916 xmax: -37.25418 ymax: -
2.783276
geographic CRS: WGS84 Lat/Long's, Degrees, -180 ==> +180
```

In [89]:

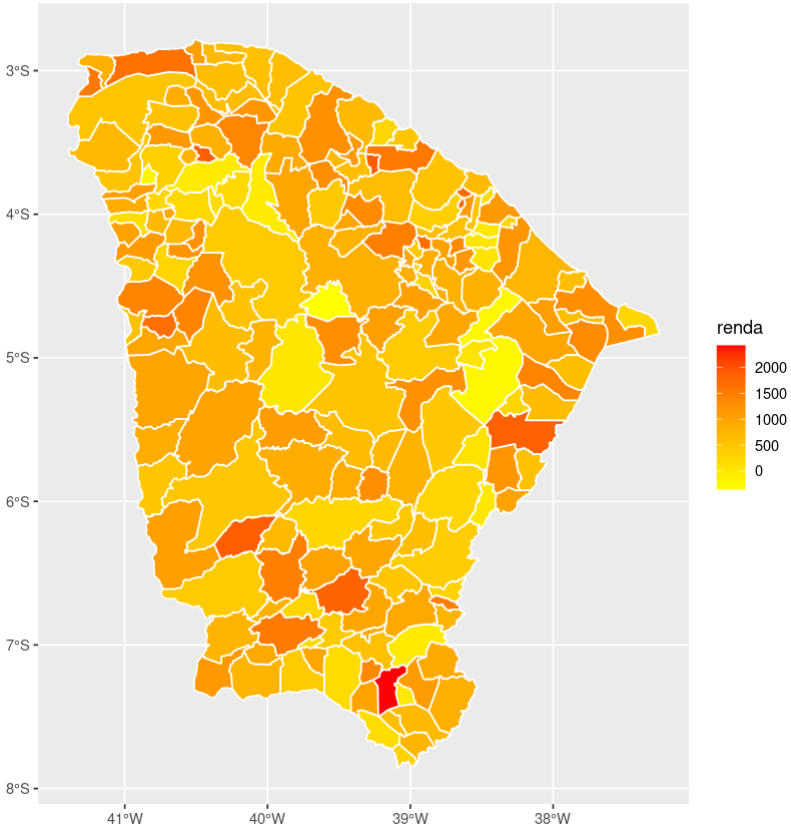
```
# Suponha que a seguinte variável represente a rendamédia dos municipios do estado
set.seed(123)
shape$renda <- rnorm(184, 800, 500)
# Caso queiramos plotar a variável renda em um mapa, podemos fazer isso usando a ge
ggplot(data= shape) +
geom_sf(aes(fill = renda), color = "white")
```



Agora imagine que eu queira definir as cores do mapa manualmente. Imagine que desejo que os menores valores da renda média municipal devam ser plotados na cor amarela e que conforme a renda se eleve a cor dos municípios ganhe uma tonalidade avermelhada. Podemos indicar este propósito por meio do parâmetro *colors* no bloco *scale_fill_gradientn*.

In [102]:

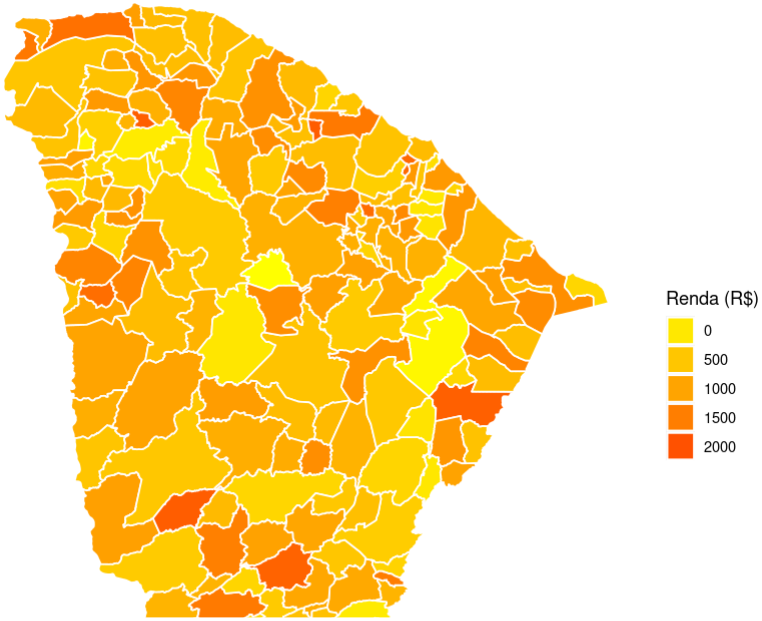
```
ggplot(data= shape) +
geom_sf(aes(fill = renda), color = "white")+
scale_fill_gradientn(colors = c(low = "yellow", high = "red"))
```



Com alguns detalhes adicionais o usuário pode deixar o mapa com uma exposição bem mais agradável.

In [113]:

```
# Possibilidades de edição do mapa
ggplot(data= shape) +
geom_sf(aes(fill = renda), color = "white")+
scale_fill_gradientn(colors = c(low = "yellow", high = "red"))+
theme(axis.line=element_blank(), # retira as linhas dos eixos
axis.text.x=element_blank(), # Retira o texto de indicação do eixo x
axis.text.y=element_blank(), # Retira o texto de indicação do eixo y
axis.title.x=element_blank(), # retira o título do eixo x
axis.title.y=element_blank(), # retira o título do eixo y
panel.border=element_blank(), # Retira as bordas do gráfico
axis.ticks=element_blank(), # retira os traços ao longo dos eixos
panel.background=element_blank())+ # Retira a cor de fundo do gráfico
guides(fill=guide_legend(title= "Renda (R$)")) # Altera o título da legenda
```



7. Qplot

qplot é uma ferramenta gráfica que auxilia na criação de figuras junto ao pacote *ggplot2*. O uso das ferramentas do *qplot* simplifica a linguagem usada no *ggplot2*, deixando a leitura dos códigos mais simples e a execução mais rápida e eficiente. De forma simplificada, um comando *qplot* possui a seguinte estrutura:

qplot(x, y, data, geom="auto", xlim = c(NA, NA), ylim =c(NA, NA))

Em que:

x é o vetor de informaçõesque compõe o eixo x.

y é o vetor de informações que compõe o eixo y.

data é o banco de dados onde as informações dos eixos x e y estão armazenadas.

geom é a geometria que irá definir o tipo de gráfico.

xlim são os limites inferior e superior do eixo x.

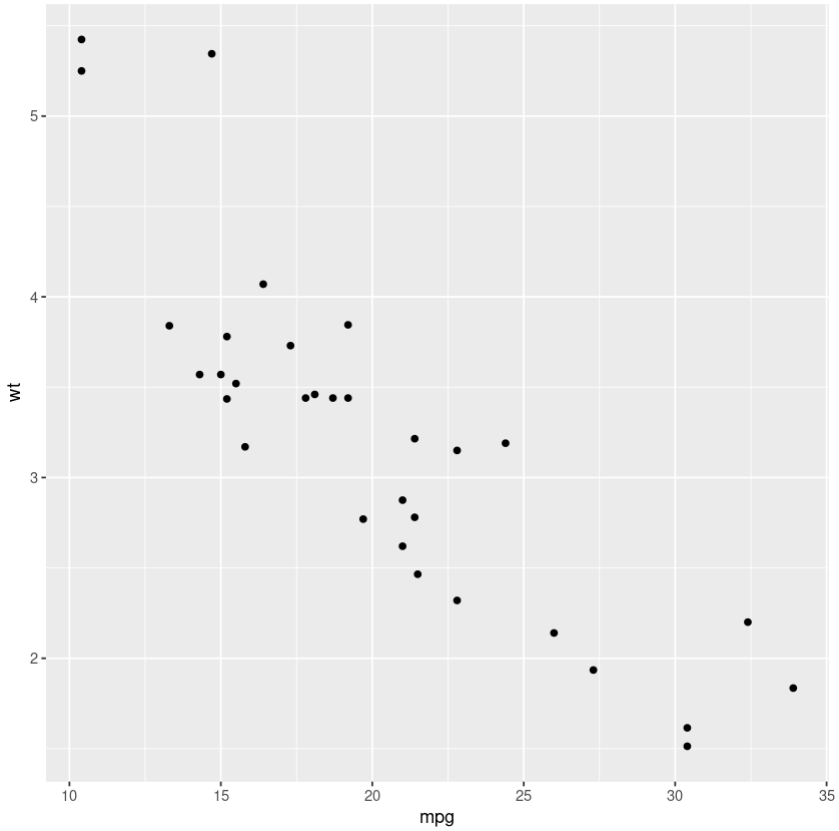
ylim são os limites inferior e superior do eixo y.

Caso o usuário não indique nenhuma geometria, será retornado um gráfico de pontos.

7.1 Gráfico de dispersão

In [2]:

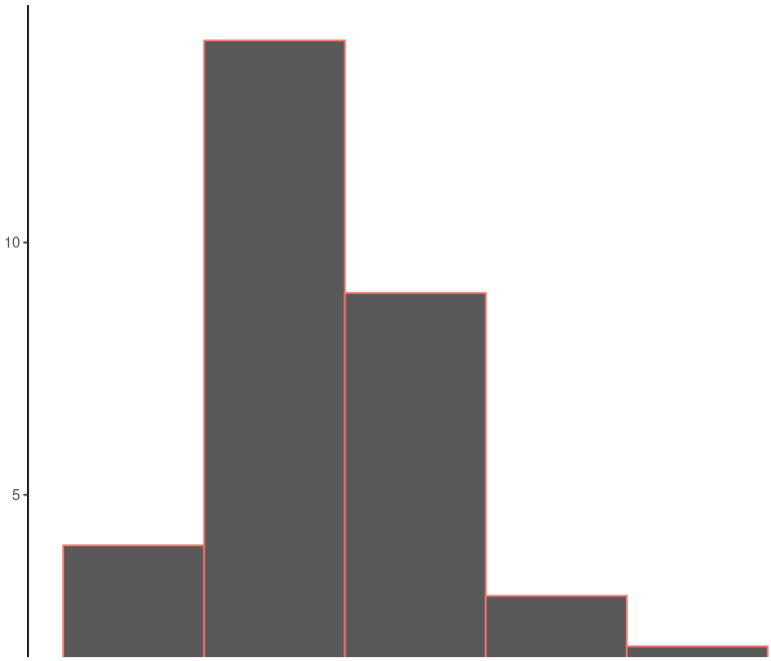
```
qplot(x = mpg, y = wt, data = mtcars)
```



7.2 Histograma

In [23]:

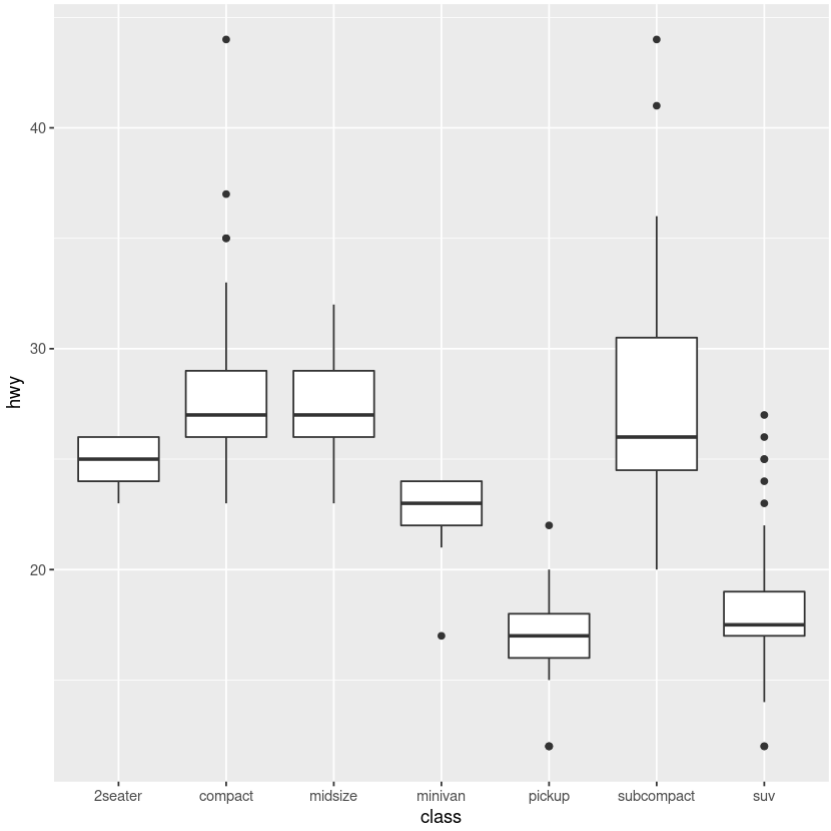
```
qplot(x = mpg, data = mtcars, geom = "histogram", bins = 5, col = "white")+
theme_classic()+
theme(legend.position = "none")
```



7.3 Boxplot

In [24]:

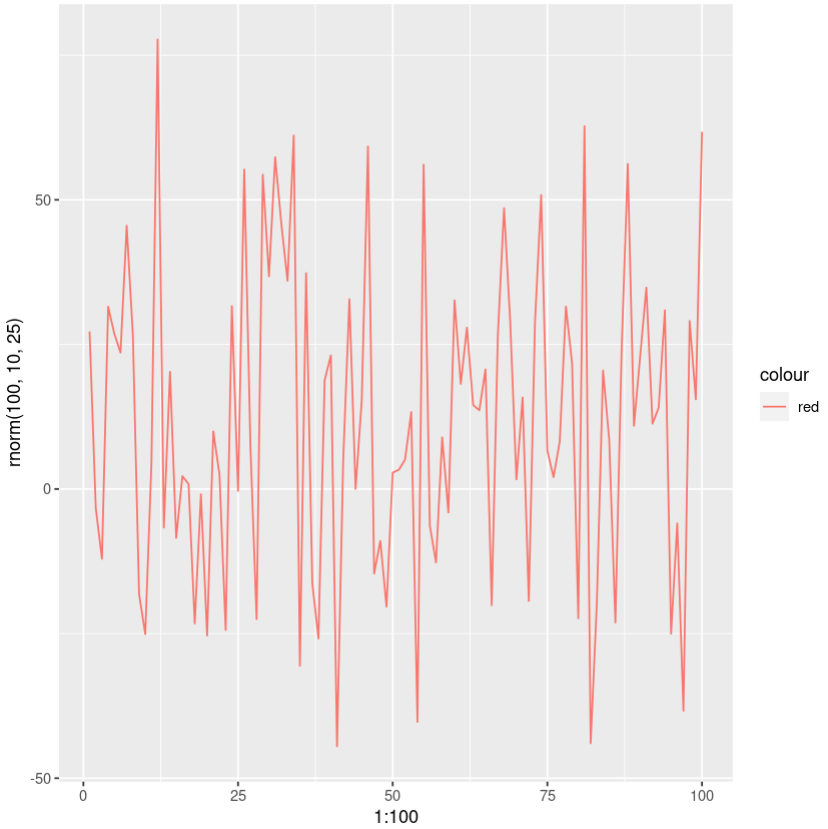
```
qplot(data = mpg, x = class, y = hwy, geom = "boxplot")
```



7.4 Gráfico de linhas

In [12]:

```
qplot(x = 1:100, y = rnorm(100, 10, 25), geom = "line", colour = "red")
```



8. plotly

Plotly é uma ferramenta gratuita de desenvolvimento avançado de gráficos em R. Esta ferramenta possibilita a exposição de dados em maneiras que as funções nativas do R não conseguem executar.

In [1]:

```
install.packages('plotly')
library(plotly)
```

```
Loading required package: ggplot2


Attaching package: ‘plotly’


The following object is masked from ‘package:ggplot2’:

    last_plot


The following object is masked from ‘package:stats’:

    filter


The following object is masked from ‘package:graphics’:

    layout
```

Para indicar como se realiza a construção dos gráficos usando o ploty, vamos usar uma base de dados bastante conhecida sobre as características de um conjunto de tipos de flores.

In [12]:

```
head(iris)
```

A data.frame: 6 × 5

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

8.1 Gráfico de distribuição

Para criar um gráfico de distribuição, apenas precisamos indicar o banco de dados, e as variáveis dos eixos x e y

In [2]:

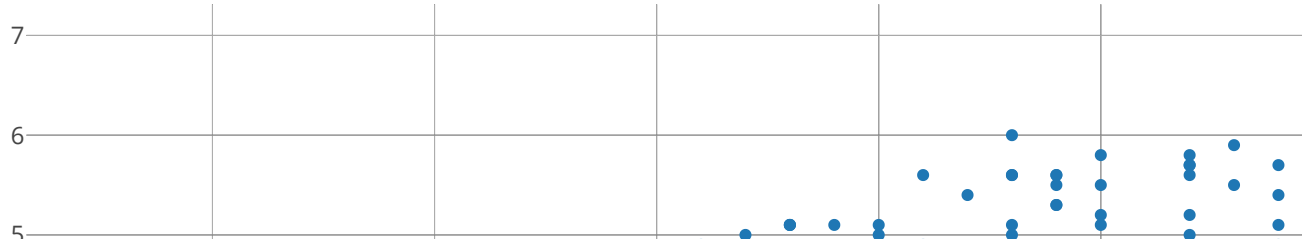
```
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length)
```

```
No trace type specified:
  Based on info supplied, a 'scatter' trace seems appropriate.
  Read more about this trace type -> https://plot.ly/r/reference/#scat
ter (https://plot.ly/r/reference/#scatter)

No scatter mode specifed:
  Setting the mode to markers
  Read more about this attribute -> https://plot.ly/r/reference/#scatt
er-mode (https://plot.ly/r/reference/#scatter-mode)

Warning message:
“`arrange_()` is deprecated as of dplyr 0.7.0.
Please use `arrange()` instead.
See vignette('programming') for more help
This warning is displayed once every 8 hours.
Call `lifecycle::last_warnings()` to see where this warning was genera
ted.”
No trace type specified:
  Based on info supplied, a 'scatter' trace seems appropriate.
  Read more about this trace type -> https://plot.ly/r/reference/#scat
ter (https://plot.ly/r/reference/#scatter)

No scatter mode specifed:
  Setting the mode to markers
  Read more about this attribute -> https://plot.ly/r/reference/#scatt
er-mode (https://plot.ly/r/reference/#scatter-mode)
```



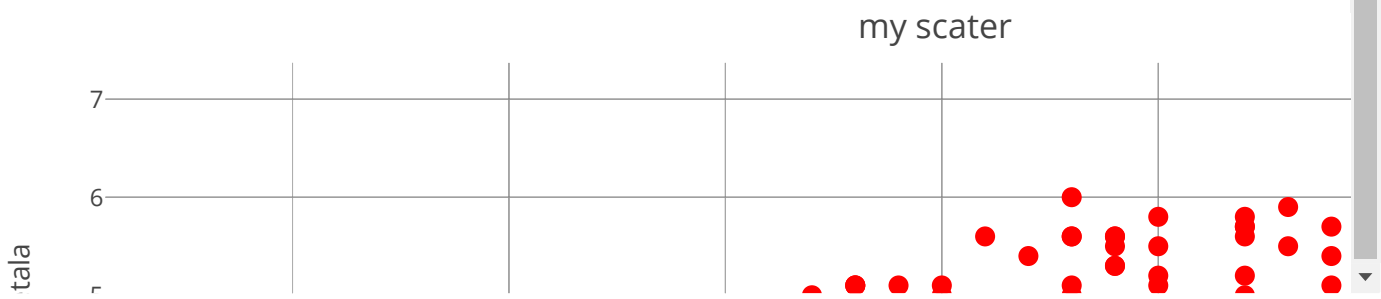
Para personalisar o gráfico, precisamos adicionar um *layout* e um *marker*.

In [3]:

```
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length,
        marker =  list(size = 10, color = "red")) %>%
  layout(title = "my scater", yaxis=list(title = "Comprimento da pétala"),
         xaxis=list(title = "Comprimento da sépala"))
```

```
No trace type specified:
  Based on info supplied, a 'scatter' trace seems appropriate.
  Read more about this trace type -> https://plot.ly/r/reference/#sc
atter (https://plot.ly/r/reference/#scatter)

No scatter mode specifed:
  Setting the mode to markers
  Read more about this attribute -> https://plot.ly/r/reference/#sca
tter-mode (https://plot.ly/r/reference/#scatter-mode)
```

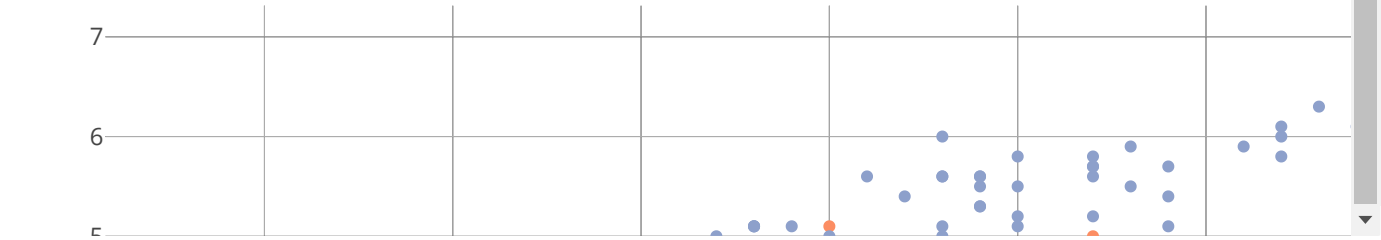


Agora suponha que queiramos deixar as cores do gráfico condicionadas a uma variável categórica. Basta indicar a variável no parâmetro *color*

In [4]:

```
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length, color = ~Species)
```

```
  Setting the mode to markers
  Read more about this attribute -> https://plot.ly/r/reference/#sca
tter-mode (https://plot.ly/r/reference/#scatter-mode)
```



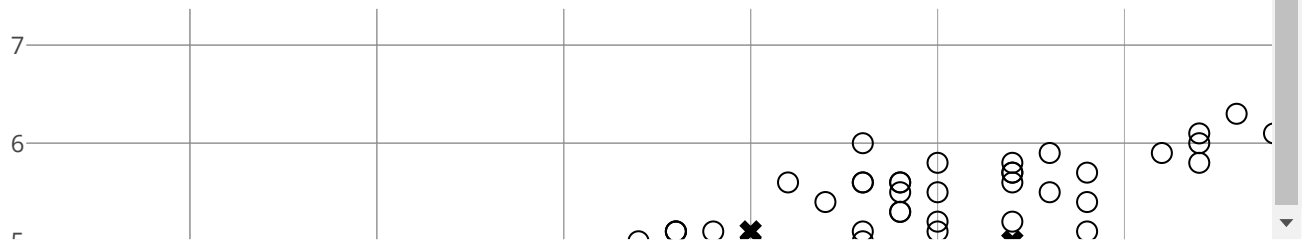
Agora suponha que em vez de condicionar as cores a uma variável categórica, queiramos diferenciar os símbolos do gráfico de acordo com essa variável. Para isso, asta indicar esta variável no parâmetro *symbol*.

In [5]:

```
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length, symbol = ~Species,
        symbols = c('circle','x','o'),
  color = I('black'), marker = list(size = 10))
```

```
No trace type specified:
  Based on info supplied, a 'scatter' trace seems appropriate.
  Read more about this trace type -> https://plot.ly/r/reference/#sc
atter (https://plot.ly/r/reference/#scatter)

No scatter mode specifed:
  Setting the mode to markers
  Read more about this attribute -> https://plot.ly/r/reference/#sca
tter-mode (https://plot.ly/r/reference/#scatter-mode)
```



Podemos alterar o tamanho dos pontos a partir do parâmetro *size*.

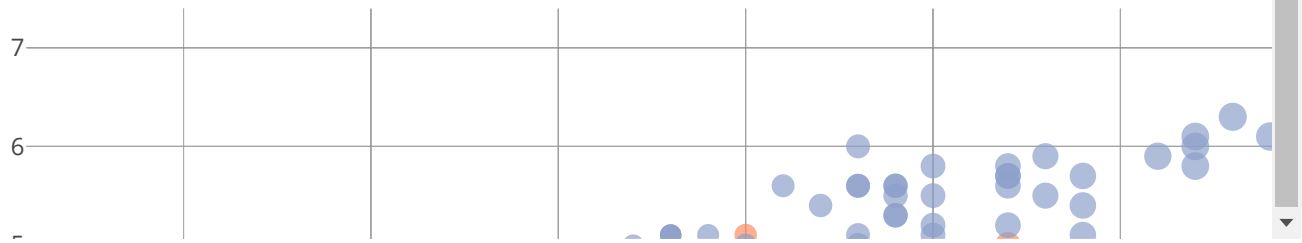
In [6]:

```
plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length, color = ~Species, size =
```

```
  Read more about this attribute -> https://plot.ly/r/reference/#sca
tter-mode (https://plot.ly/r/reference/#scatter-mode)

Warning message:

“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
```

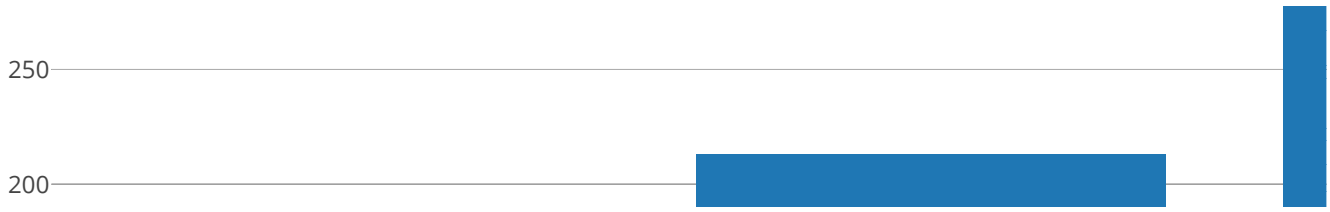


8.2 Gráfico de barras

Para criar um gráfico de barras devemos usar *type* = *bar*.

In [7]:

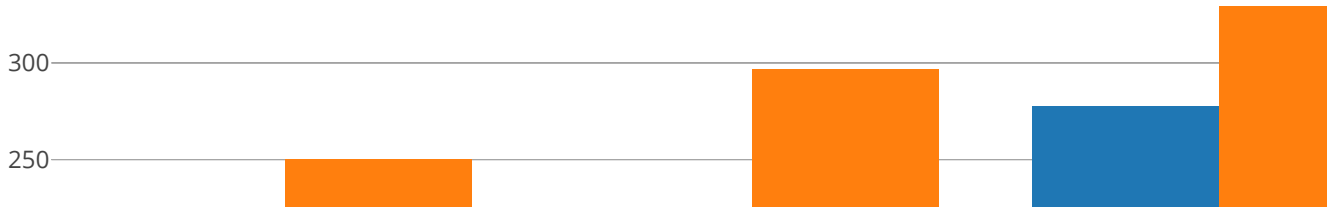
```
plot_ly(data = iris,
  x = ~Species,
  y = ~Petal.Length,
  type = "bar")
```



Para adicionar outra variável ao gráfico de barras, o usuário precisa adicionar um novo traçõ nas feições do gráfico. Esse procedimento é feito pelo bloco *add_trace*, como mostrado abaixo.

In [8]:

```
plot_ly(data = iris,
  x = ~Species,
  y = ~Petal.Length,
  type = "bar", name = "Comprimento da pétala")  %>%
add_trace(y = ~ Sepal.Length, name = "Comprimento da sépala")
```

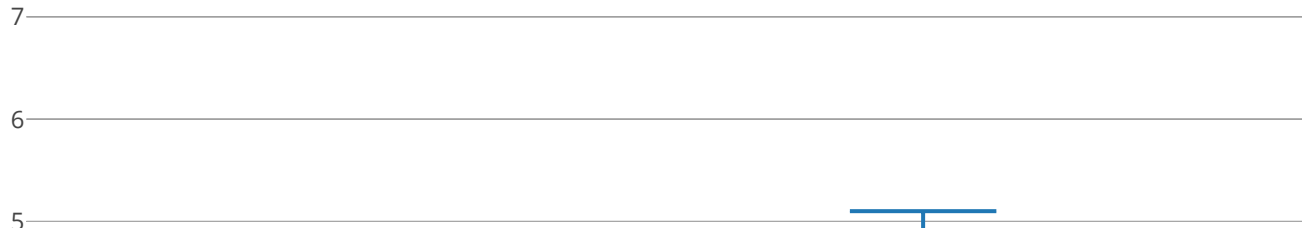


8.3 Boxplot

Para criar um gráfico do tipo boxplot, o usuário deverar indicar a opção "box" no parâmetro type. Por exemplo:

In [9]:

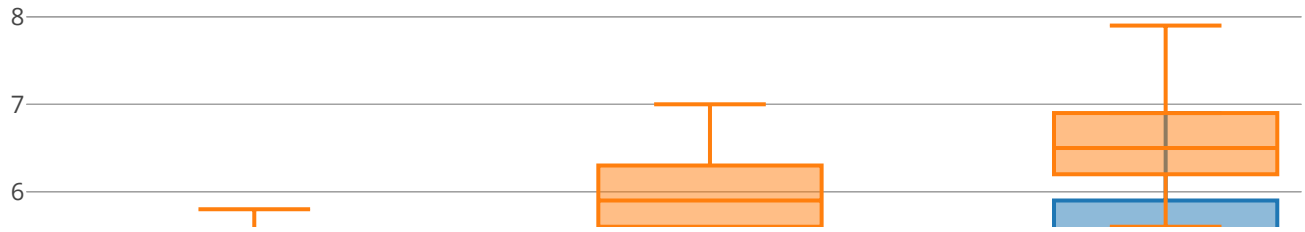
```
plot_ly(data = iris, x = ~Species, y = ~Petal.Length, type = "box")
```



Caso o usuário desege incluir mais uma variável no gráfico, ele devereá fazer isto adicionando um traço por meio da inclusão do bloco *add_trace*.

In [10]:

```
plot_ly(data = iris, x = ~Species, y = ~Petal.Length, type = "box", name = "Comprim
add_trace(y = ~Sepal.Length, name = "Comprimento da sépala")
```



8.4 Histograma

Para construir um histograma, o usuário deverá indicar *type* = *"histogram*, conforme demonstrado abaixo.

In [11]:

```
plot_ly(data = iris, x = ~Petal.Length, type = "histogram") %>%
layout(bargap=0.1)
```

8.5 Gráfico de linhas

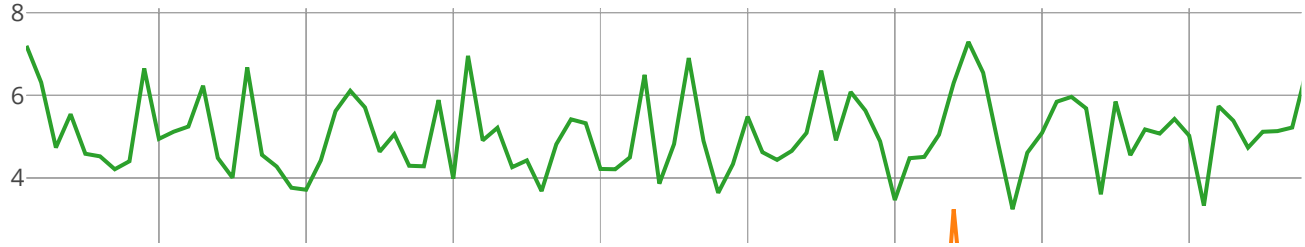
Para indicar que se trata de um gráfico de linhas, o usuário deverá indicar *mode = lines"*. A partir da primeira linha, cada linha adicional deverá ser inserida por meio da inclusão do bloco *add_trace*.

In [12]:

```
set.seed(123)
df <- data.frame(x1 = 1:100, x2 = rnorm(100, mean = -5), x3 = rnorm(100, mean = 0),
```

In [13]:

```
plot_ly(data = df, x = ~x1, y = ~x2, mode = "lines", type = "scatter", name = "Disp
add_trace(y = ~x3,mode = "lines", type = "scatter", name = "Dispersão de x3")%>%
add_trace(y = ~x4,mode = "lines", type = "scatter", name = "Dispersão de x4")
```



8.6 Gráficos dinâmicos

Uma das vantagens do uso da biblioteca plotly é que podemos adicionar um caráter dinâmico aos gráficos. Por exemplo, imagine que queiramos plotar a relação entre crescimento econômico e expectativa de vida entre os países no decorrer dos anos. Em uma situação convencional, seria necessário criar um gráfico de dispersão

para cada ano. No entanto, podemos fazer isso em um único gráfico indicando qual a variável responsável pela dinâmica da análise por meio do parâmetro *frame*. Por exemplo:

In [14]:

```
library(gapminder)

df <- gapminder

df %>%
  plot_ly(
    x = ~log(gdpPercap),
    y = ~log(lifeExp),
    size = ~pop,
    color = ~continent,
    frame = ~year,    # Frame = year indica que os pontos irão se alterar conforme
    text = ~country,
    hoverinfo = "text",
    type = 'scatter',
    mode = 'markers')
```

```
`line.width` does not currently support multiple values.
Warning message:
“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
Warning message:
“`line.width` does not currently support multiple values.”
```