## Capítulo 2 - Gráficos com plot() 01

## a - *plot()* ando gráficos

A criaçao de gráficos é uma arte e um processo totalmente iterativo enquanto encontramos, modificando diversos argumentos, a aparência final do nosso gráfico.

Ainda que apenas para conhecer ou explorar os dados, a interação com o gráfico nos permite ter diferentes visões dos dados e começar a decidir sobre o tipo e apariencia do(s) gráfico(s) final(is).

Nos exercícios a seguir, toda instrução que tenha a

#### ***seguinte formatação***

é para sua execução no fragmento de código R que segue ou na consola do RStudio (janela 3).

Para ver algumas das possibilidades da função **plot()** execute na consola o seguinte comando:

**demo("graphics")**

#### Criação rápida de gráficos

A função *plot()* do R base, permite elaborar em forma rápida e simples, diversos gráficos que facilitam a análise exploratória de dados. Esses gráficos não são complexos ou aprimorados como para apresentações ou publicações, mas são extremamente úteis para definir análises iniciais e ainda definir o tipo de gráfico mais adequado para a elaboração de resultados finais.

A função é genérica e, acorde os dados de entrada, podemos ter diferentes tipos de gráficos.

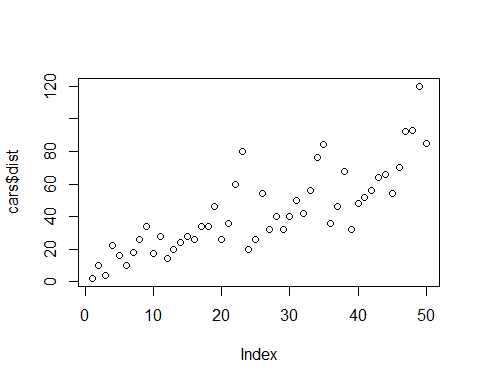
A forma mais básica é *plot(x, y, ...)*. Sendo *"..."* argumentos adicionais que podem ser passados.

O argumento *y* também é opcional e dependo do tipo dos seus dados ele será obrigatório. Por exemplo quando desejamos fazer um diagrama de dispersão de uma variável, ele não é necessário. Vejamos o seguinte exemplo, usando o conjunto de dados "cars":

head(cars)

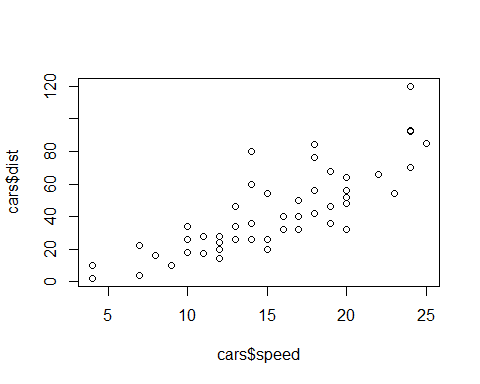
## speed dist  
## 1 4 2  
## 2 4 10  
## 3 7 4  
## 4 7 22  
## 5 8 16  
## 6 9 10

plot(cars$dist)



Quando desejamos um gráfico de duas variáveis (uma explicatória ou independente no eixo x e uma explicada ou resposta no eixo y), basta fazer:

plot(cars$speed, cars$dist)



Note em ambos casos que *plot()* escolheu automaticamente diversos atributos do gráfico para a plotagem.

Temos duas maneiras de especificar nossos dados ou variáveis: através da forma cartesiana, **x, y**, como nos exemplos anteriores ou na forma de fórmula **y ~ x**.

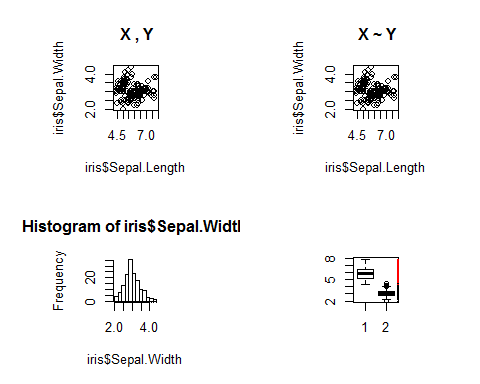
Note a ordem das variáveis em cada uma e o uso do **~**.

Veja o seguinte exemplo com os dados do conjunto de dados **iris**, o qual veremos a sua estrutura primeiro:

str(iris)

## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

par(mfrow = c(2,2), pty = "s")  
plot(iris$Sepal.Length, iris$Sepal.Width, main = " X , Y")  
plot(iris$Sepal.Width ~ iris$Sepal.Length, main = " X ~ Y")  
hist(iris$Sepal.Width)  
rug(iris$Sepal.Width)  
boxplot(iris$Sepal.Length, iris$Sepal.Width)  
rug(iris$Sepal.Length, side = 4, col = "red") # função que plota as observações no eixo (padrão=1)  
rug(iris$Sepal.Width, side = 4)



No painel anterior o parâmetro **par** determina como vamos configurar o espaço gráfico. O argumento *mfrow = c(2,2)*, indica que vamos ter 4 gráficos, organizados numa matriz de 2 linhas por duas colunas. O preenchimento dos espaços pelos gráficos, por padrão, é por linha. A inclusão do argumento *pty = "s"*, faz que os gráficos mantenham um aspecto similar para ambos eixos (gráfico quadrado).

Também observa-se que a própria função *plot()* escolhe em forma automática os atributos de exibição dos gráficos, como limites dos eixos x e y, nomes das variáveis, forma, tamanho e cores dos pontos, nos casos de cima, cores divisão de intervalos no caso do histograma e posição e cores no caso dos boxplot.

Os títulos nos dois primeiros gráficos foram definidos através do argumento *main*.

Os argumentos adicionais que podem ser passados na função *plot()* são:

1- *type*, determina o tipo de gráfico que queremos, por padrão será usado "p" de pontos, mas temos:

* + "l" para linhas;
  + "b" para pontos e linhas;
  + "c" para as linhas apenas da opção "b";
  + "o" para sobreposição de ambos;
  + "h" para gráfico tipo ‘histograma’(linhasverticais);
  + "s" degraus,
  + "S" outros tipos de degraus
  + "n" não plotar

2- *main*, título principal;

3- *sub*, subtítulo, debaixo do principal. Se aconselha para colocar a fonte dos nossos dados;

4- *xlab*, título do eixo x;

5- *ylab*, título do eixo y;

6- *xlim*, vetor que indica os limites mínimo e máximo do eixo x;

7- *ylim*, vetor que indica os limites mínimo e máximo do eixo y;

8- *pch*, determina o tipo de símbolo a ser usado, veja ?pch

9, *col*, especifica qual cor a ser usada na simbologia (em inglês), exemplo *col = "blue"*

**DICA**

Para a realização dos exercícios, vai encontrar no seu diretório de trabalho um script R, "graficos01\_exerc.R", copie os exercícios a realizar e sua própria solução. No futuro podem ser uma referência, junto com o material da oficina para resolver em forma rápida problemas de visualização de dados.

#### ***Exercício 01***

Usando o comando já mostrado de:

*plot(cars$speed, cars$dist)*

1- adicionar um título principal, um subtítulo (sugestão: "Fonte: R datasets::cars"), legendas em ambos eixos e redefina os limites de cada um.

2- mude o tipo de símbolo para triângulo cheio e de cor vermelha

#### ***Exercício 02a***

Considerando o comando *hist* no segundo exemplo apresentado, fazer o histograma da variável *Sepal.Lenght* dos dados *iris*.

Redefina os limites dos eixos, coloque título principal, subtítulo e nos nomes nos eixos e defina uma cor verde para o histograma.

A função *hist* escolhe automáticamente o número de intervalos ótimo, em função do tamanho da amostra e a dispersão dos dados, porém nem sempre é a melhor escolha visual. Através do argumento *breaks* e o número de intervalos desejado podemos mudar nosso histograma.

#### ***Exercício 02b***

Com o comando do exercício anterior, experimente usar o argumento *breaks* com 5,10,20 e 50 intervalos. Que acontece com o eixo de Frequências? e com 50 intervalos, o que acontece com o histograma?

**Gráficos de probabilidade amostral e histogramas**

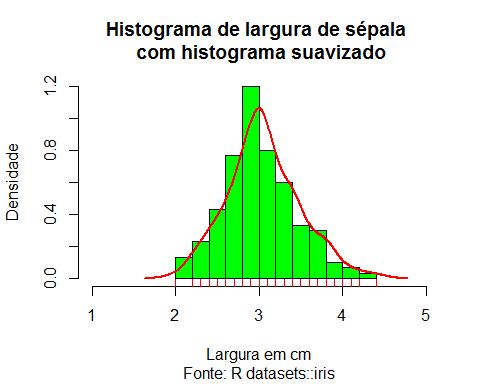
Ainda que não tão conhecidos como os histogramas, as probabilidades calculadas podem ser graficadas como um histograma "suavizado", o que fornece uma melhor estimativa da função de probabilidade para uma variável aleatória.

A função *hist* com o argumento *"probability =T"*, calcula a função de probabilidade amostral e fornece os valores de densidade no eixo "y". Adicionando uma linha de densidade sobre o histograma, a linha pode ser vista com um histograma "suavizado".

Para isto, após o gráfico do histograma, adicionar uma linha com a função lines e a densidade calculada. No seguinte exemplo adicionamos uma linha em vermelho de largura = 2 e retiramos os limites do eixo "y". A função *lines()* é uma função adicional que se coloca fora da função *plot()*.

Note o uso de “” no título principal para quebrar a legenda em dois parâgrafos.

hist(iris$Sepal.Width,  
 probability = T,  
 breaks = 10,  
 xlim = c(1,5),  
 main = "Histograma de largura de sépala \n com histograma suavizado",  
 sub = "Fonte: R datasets::iris",  
 xlab = "Largura em cm",  
 ylab = "Densidade",  
 col = "green")  
lines(density(iris$Sepal.Width), lwd=2, col= "red")  
  
rug(iris$Sepal.Width, lwd = 1, col ="red")



Um outro tipo de gráfico muito usado é o de **barra**. Ele serve para mostrar as quantidades de observações contidas nos diferentes níveis de uma variável categórica.

Por exemplo, nos dados *iris* a função *table(iris$Species)* (execute na consola de comandos), nos informa que as três espécies da planta tem 50 observações cada. função *plot()* pode fazer um gráfico de barras, mas visualmente não é atrativo.

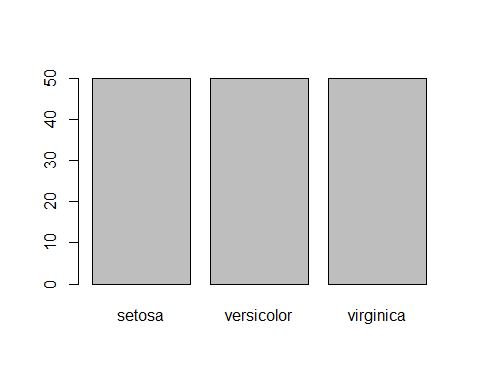
#### ***Exercício 03***

Na linha de comando experimente:

plot(table(iris$Species))

A função específica *barplot()* elabora a partir de uma tabela prévia, o gráfico de barras correspondente. Vejamos agora a mesma tabela com a função *barplot*:

barplot(table(iris$Species))



#### ***Exercício 04***

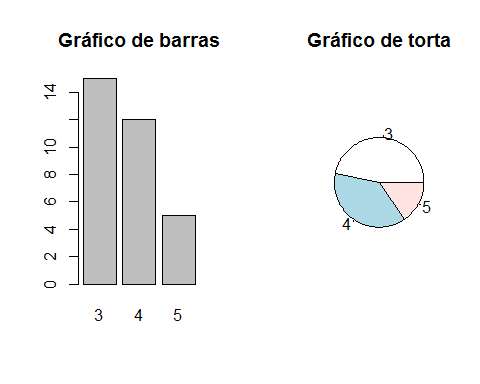
Com os conhecimentos de argumentos já conhecidos: coloque título principal, subtítulo, nome no eixo "y" e dê uma cor diferente para cada barra (dica: o argumento *col* aceita um vetor com os nomes das cores (inglês), tipo *c("cor1", "cor2",...)*).

Finalmente algumas considerações acerca do tipo de gráfico de torta ou *pie chart*. Ele pode ser usado quando as categorias a serem mostradas são em número pequeno e não muito diferentes nas suas quantidades ou valores. Para ter categorias em número diferentes, vamos usar a base de dados *mtcars* e fazer uma tabela de carros por tipo de quantidade de marchas (3, 4 ou 5). Logo após vamos apresentar a tabela e o gráfico com *barpot()* e com *pie* lado a lado *(mfrow)*.

tab <- table (mtcars$gear)  
tab

##   
## 3 4 5   
## 15 12 5

# Configurando o espaço gráfico para dois gráficos na mesma linha  
par(mfrow = c(1,2))  
  
barplot(tab, main="Gráfico de barras")  
pie(tab, main="Gráfico de torta")



No exemplo apresentado, a informação tem diferentes características.

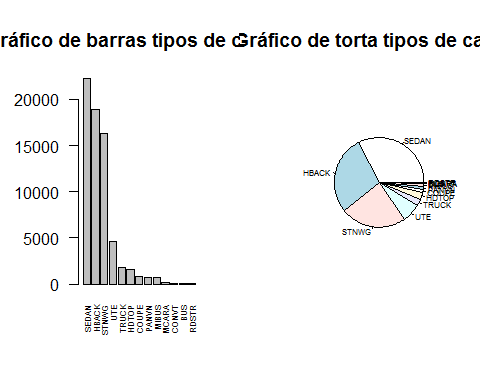
No primeiro gráfico, pode-se estimar pelo eixo *y* a quantidade de dados em cada categoria, assim como temos a impressão do tamanho relativo de cada uma.

No segundo gráfico, temos uma visão mais clara do tamanho relativo, mas não temos as quantidades expressas.

Ambas dificuldades podem ser contornadas com a inclusão de argumentos adicionais em cada função. Porém o gráfico de torta fica comprometido no caso de maior número de variáveis e grande diferença de elementos entre as categorias.

No exemplo seguinte, vamos usar uma base de dados de seguros de carros e criar uma tabela por tipo de carro. Na sequência vamos plotar dois gráficos lado a lado com o gráfico de barras com *barplot()* e com *pie*, lado a lado.

# Carrega o pacote insuranceData  
library(insuranceData)  
  
# Uso a função "data" para carregar a base "dataCar" do pacote "insuranceData"  
data("dataCar")  
  
# Configurando o espaço gráfico para dois gráficos na mesma linha  
par(mfrow = c(1,2))  
  
# Criar a tabela de tipo de carro e ordenar em ordem decrescente  
tab <- sort(table(dataCar$veh\_body),  
 decreasing = T)  
# Gráfico de barras com nomes reduzidos pela metade cex.names = 0.5  
# o argumento las = 2, coloca os nomes na vertical para melhorar o espaço  
barplot(tab, las = 2, cex.names = 0.5)  
  
# Adiciono um título, usando a função "title"  
title("Gráfico de barras tipos de carros")  
  
# criar o gráfico de torta  
pie(tab, cex = 0.5)  
title("Gráfico de torta tipos de carros")



A pesar da diferença grande de dados em cada variável, vemos que no caso do gráfico de barras, ainda temos uma visão clara do tamanho relativo, assim como das quantidades envolvidas. No caso do gráfico de torta, a leitura torna-se impossível para as menores quantidades, superpondo as últimas.

A escolha da melhor apresentação é uma tarefa subjetiva, porém em toda a literatura de visualização de dados, não se recomenda usar o gráfico de torta e se ainda assim é usado, **nunca colocar ele em forma 3D!**

FIM DA PRIMEIRA PARTE

## Capítulo 2 - Gráficos com plot() 02

## b - **Eixos controlados!!**

**DICA**

As variáveis de ambiente que controlam o espaço gráfico são determinadas pela função *par()* (global).

Quando trabalhamos com diversos gráficos, com frequência modificamos o ambiente, por exemplo para plotar diversos gráficos numa só janela, ou modificar as fontes do texto.

Por isso é de boa prática guardar as variáveis com seus valores padrão antes de qualquer modificação, para poder retornar (ou limpar) ao ambiente original. Uma das formas pode ser colocar os valores padrão numa variável que pode ser chamada antes de um novo gráfico ser construído, por exemplo:

pro<- par(no.readonly=TRUE)

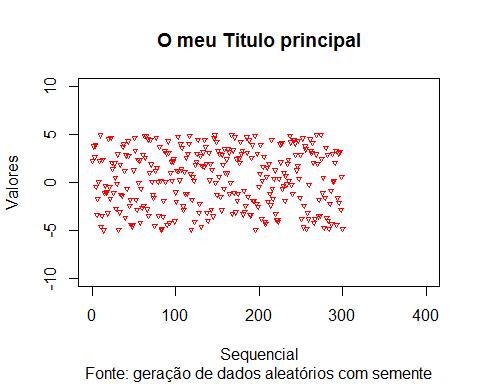
Por outro lado as variáveis modificadas diretamente nas funções gráficas são apenas válidas para o gráfico em que são usadas (local)

Vamos ver como personalizar os eixos em alguns casos práticos.

Primeiro uma plotagem simples com todos os elementos vistos até aqui, intencionalmente fixamos o limites dos eixos com valores maiores dos necessários.

Simulamos dados, 300 valores consecutivos e aleatorios entre -5 e +5...veja *?runif*

set.seed(12345) # semente serve para reprodução de valores iguais em dados aleatórios (ou seudo...)  
x<-1:300  
y<-runif(x,-5,5)  
  
# gráfico ...Dica, ordene sempre os argumentos de forma semelhante para encontrar de forma rápida o que quer modificar. Separe cada elemento em linhas separadas...  
# Nesse exemplo: primeira linha tem os dados a serem graficados  
# Segunda e terceira, os limites dos eixos  
# Quarta até sexta, aparência dos pontos, cor, tipo e tamanho  
# Sétima até o final, títulos principais, sub e títulos dos eixos  
plot(x, y,   
 ylim = c(-10,10),   
 xlim = c(0,400),  
 col = "red",   
 pch = 25,   
 cex = 0.5,  
 main = "O meu Titulo principal",  
 sub = "Fonte: geração de dados aleatórios com semente",  
 xlab = "Sequencial",   
 ylab = "Valores")



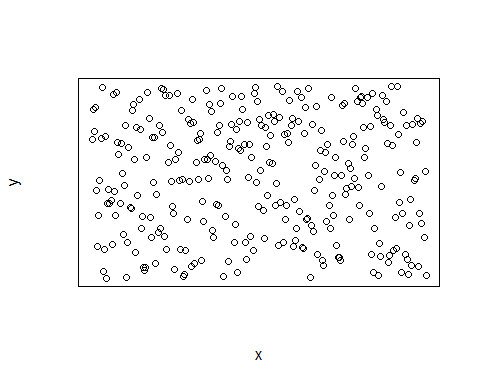
Uma função muito útil para ter mais controle dos eixos é *axis*. Para poder fazer uso da função, primeiro tem de desabilitar o eixo desejado na função *plot()*. Isso é feito incluíndo o parâmetro **xaxt="n"** e ou **yaxt="n"**.

*axis()* permite controlar os valores das marcas, tipo de linha, tamanho, espaçamento das linhas, cor, etc.

Vejamos:

Uso da função axis: vamos suprimir os eixos "x" e "y"

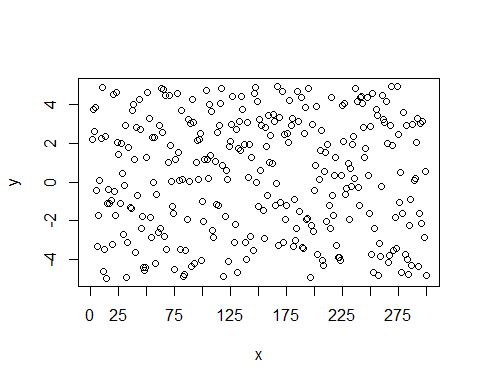
par(pro) # voltamos aos valores padrão..  
plot(x,y,  
 xaxt = "n", # o valor xaxt="n", impede de plotar o eixo x  
 yaxt = "n") # o valor yaxt="n", impede de plotar o eixo y



Vamos adicionar o eixo x, para isso usamos a função *axis* (veja todas as opções com **?axis**), especificando o lado em que vamos adicionar (1, 2, 3 ou 4, contados em sentido horário a partir da borda inferior).

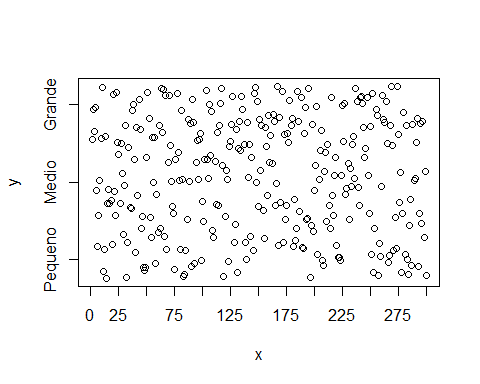
No eixo adicionado, vamos colocar uma sequência de valores de 0 a 300, com marcas cada 25. Para isso usamos a função *seq* como argumento dentro de *axis*.

plot(x,y,  
 xaxt="n") # o valor xaxt="n", impede de plotar o eixo x  
  
axis(side= 1,  
 at = c(seq(0, 300, 25))) # valores no eixo x de x=0 a 300, cada 25



Podemos colocar etiquetas texto em determinados valores. No seguinte exemplo, colocaremos três etiquetas nos valores -4, 0 e 4 do eixo "y" (número 2).

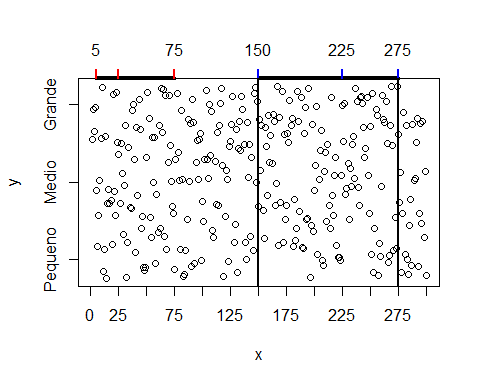
plot(x,y,  
 xaxt="n",  
 yaxt="n")  
  
axis(side= 1, # no eixo x inferior  
 at = c(seq(0, 300, 25))) # plotar os valores no eixo x de x=0 a 300, cada 25  
  
axis(side = 2, # no eixo y, lateral esquerdo  
 at = c(-4,0,4), # nas posições de vetor indicado  
 labels = c("Pequeno","Medio","Grande")) # plotar os textos indicados



Um argumento que ajuda a controlar a posição relativa dos textos e etiquetas é *las =*. Quando *las = 1*, o texto será colocado paralelo ao eixo, quando *las = 2*, o texto fica perpendicular. O valor 1 é o padrão, note que para o eixo "x" o texto fica na horizontal do gráfico, enquanto no eixo "y", fica perpendicular.

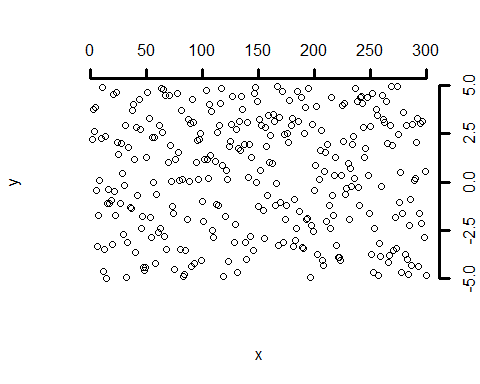
A função *axis()*, também controla o tipo de linha e espessura das marcas. Ao gráfico anterior vamos adicionar uma marca preta entre os valores (5,25 e 75) e outra nos valores (150, 225 e 275), finalmente serão colocadas duas linhas verticais para demarcar o espaço entre os valores (150 e 275).

plot(x,y,  
 xaxt="n",  
 yaxt="n")  
  
axis(side = 1, # no eixo x inferior  
 at = c(seq(0, 300, 25))) # plotar os valores no eixo x de x=0 a 300, cada 25  
  
axis(side = 2, # no eixo y, lateral esquerdo  
 at = c(-4,0,4), # nas posições de vetor indicado  
 labels = c("Pequeno","Medio","Grande")) # plotar os textos indicados  
  
axis(side = 3, # adicionamos no eixo x superior  
 at = c(5,25,75), # nos valores...  
 lwd = 4, # espessura de 4   
 lwd.ticks = 2, # espessura das marcas  
 col.ticks = "red") # cor das marcas  
  
axis(side =3, # adicionamos no eixo x superior  
 at = c(150,225,275), # nos valores...  
 lwd = 4, # espessura de 4  
 lwd.ticks = 2, # espessura das marcas   
 col.ticks = "blue") # cor das marcas  
  
abline(v=150, lwd =2) # linha vertical no valor de 150, espessura 2  
abline(v=275, lwd =2) # linha vertical no valor de 150, espessura 2



Ou se desejar pode tirar toda a borda da plotagem e mostrar apenas os eixos.

plot(x,y,  
 bty ="n", # suprimir a caixa do gráfico (borda exterior)  
 xaxt="n", # suprimir eixo x  
 yaxt="n") # suprimir eixo y  
  
axis(side = 3,  
 at = seq(0,300,50),  
 lwd = 3)  
axis(side = 4,  
 at = seq(-5,5,2.5),  
 lwd = 3)



Marcas nos eixos, ou etiquetas

Se tem um controle mais fino através de *par()* ou *axis()*:

o argumento *tcl* controla como as marcas serão posicionadas:

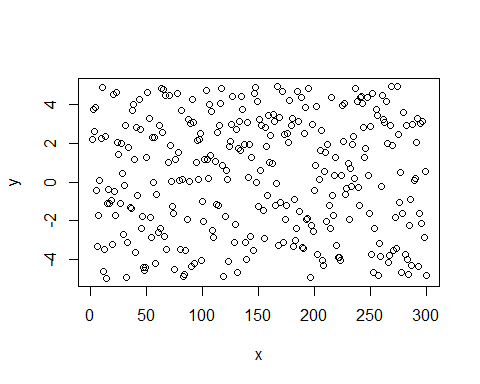
- valores positivos fazem as marcas estar dentro do gráfico  
  
- valores negativos fazem as marcas estar fora da área do grafico

o argumento *mgp* dentro de *par()* toma três valores:

- o primeiro controla o afastamento entre o gráfico e o eixo do título  
  
 - o segundo entre o gráfico e o eixo de etiquetas e   
 - o terceiro entre o gráfico e a linha do eixo,  
 - o padrão é mgp=c(3,1,0)

Vejamos no seguinte exemplo usando *mgp* e *tcl* com os valores padrão para o desenho.

par(tcl = -0.5, # valor padrão, controla o tamanho das marcas e a direção, valor negativo = marca fora  
 mgp = c(3,1,0)) #padrão  
  
plot(x,y)



#### ***Exercício 05***

Usando o comando anterior, modique o argumento *tcl* para 0.5 e experimente modificar apenas o primeiro valor de *mgp* para 1. Que acontece com o nome dos eixos? Experimente outros valores.

#### ***Exercício 06***

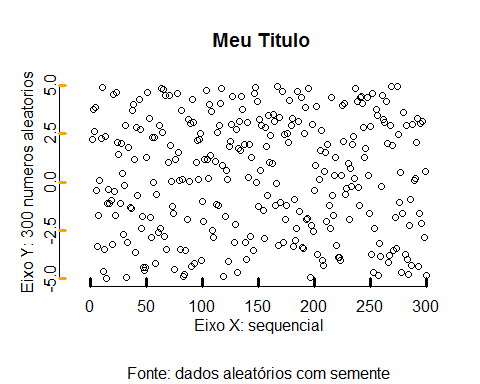
Usando o comando anterior, modique o argumento *mgp*, deixe o primeiro valor em 3 e modifique os dois seguintes para 2 e 1. Que muda nas etiquetas? Experimente outros valores.

Um outro exemplo usando *axis()* e *mtext()*. A última é uma função útil para personalizar os eixos quando eles são suprimidos inicialmente. No exemplo seguinte mostram-se duas formas diferentes de personalizar eixos. Também foi suprimida a caixa do gráfico

par(pro)

plot(x,y,  
 bty ="n", # suprimir a caixa do gráfico (borda exterior)  
 xaxt = "n",  
 yaxt = "n",  
 xlab = "",  
 ylab = "",   
 main = " Meu Titulo",   
 sub = "Fonte: dados aleatórios com semente")  
  
axis(side = 1,  
 at = seq(0,300,50),  
 tcl = 0.4,  
 lwd.ticks= 3,  
 mgp = c(0,0.5,0))

#usamos mtext porque o eixo foi desabilitado em plot..  
mtext(side = 1,   
 text = "Eixo X: sequencial",  
 line = 1.5)  
  
axis(side = 2,  
 at = seq(-5,5,2.5),  
 tcl = 0.3,  
 lwd.ticks= 3 ,  
 col.ticks= "orange",  
 mgp = c(0,1,1))  
  
mtext(side=2,  
 text="Eixo Y: 300 numeros aleatorios",  
 line=2.2)



Vejamos o caso quando temos uma área de plotagem com vários gráficos e eixos semelhantes. No uso do *mtext*, notar o argumento **outer** que determina que o texto será colocado na área exterior ao gráfico

par(pro)

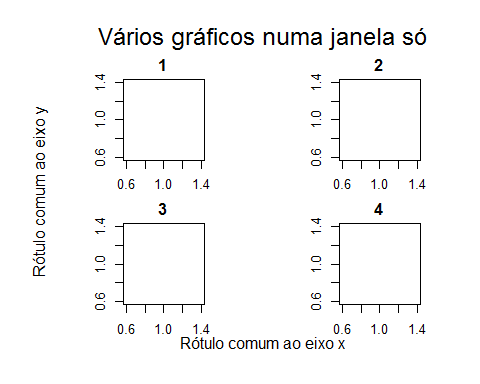
par(oma = c(3,3,3,0), # parâmetro que estabelece a margem exterior da janela comum

mar = c(2,2,2,1), # parâmetro que estabelece a margem exterior de cada gráfico(em linhas)

mfrow= c(2,2), # parâmetro que determina a quantidade e posição dos gráficos

pty = "s") # parâmetro que ajusta os eixos para deixar eles de igual tamanho  
  
plot(1,1, ylab="", xlab="", type="n", main = "1")  
plot(1,1, ylab="", xlab="", type="n", main = "2")  
plot(1,1, ylab="", xlab="", type="n", main = "3")  
plot(1,1, ylab="", xlab="", type="n", main = "4")  
mtext(text = "Rótulo comum ao eixo x",  
 side = 1,  
 line = 0,  
 outer= TRUE)  
  
mtext(text ="Rótulo comum ao eixo y",  
 side = 2,  
 line = 0,  
 outer= TRUE)

mtext(text ="Vários gráficos numa janela só",  
 side = 3,  
 line = 0,  
 outer= TRUE,  
 cex = 1.5)



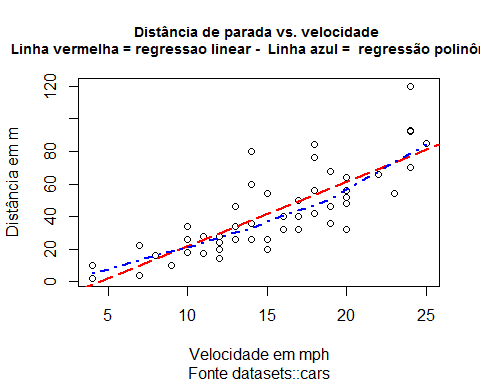
Plotar e ou sobrepor linhas e pontos

Com a função *lines()* podemos adicionar uma linha ao gráfico. A linha é produzida a partir da união de um conjunto de pontos em forma sequêncial.

A função *abline()* é um caso especial de *lines()* quando queremos graficar uma regressão linear simples e suprimos os parâmetros "a" e "b", por exemplo ao plotar um modelo produzido pela função *lm()*.

No seguinte exemplo vamos usar os dados da base datasets::**cars** para produzir um gráfico de dispersão das variáveis "speed" e "dist" e adicionar posteriormente duas linhas, uma de regressão simples,através de *abline()* e uma de regressão polinômica suavizada usando a função *lowess()* do pacote *stats*, dentro da função *lines()*.

cars <- datasets::cars # salvamos os dados  
linha.reg <- lm(cars$dist ~ cars$speed)# calculamos uma regressão linear simples  
linha.pol <- stats::lowess(cars)  
  
plot(cars,   
 cex.main = 0.9,  
 main = "Distância de parada vs. velocidade \n Linha vermelha = regressao linear - Linha azul = regressão polinômica",  
 sub = "Fonte datasets::cars",  
 xlab = "Velocidade em mph",  
 ylab = "Distância em m") # diagrama de dispersão  
abline(linha.reg,   
 lwd = 2,  
 lty = 5,  
 col ="red")  
lines(linha.pol, # plotar a lina  
 lwd = 2, # largura da linha  
 lty = 4, # tipo de linhas veja o argumento em graphics::par   
 col = "blue")



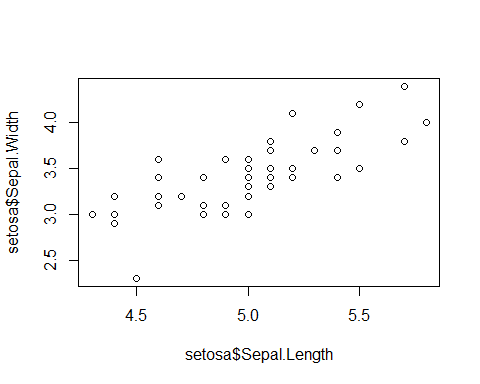
Adicionando uma camada de pontos

A função *points* realiza uma função parecida com as anteriores mostradas. Uma vez realizado um gráfico podemos adicionar camadas personalizadas de pontos.

No exemplo seguinte vamos criar um gráfico da variável "Sepal.Length" da espécie "setosa" dos dados "iris". Sabemos que os dados tem 50 observações de cada espécie.

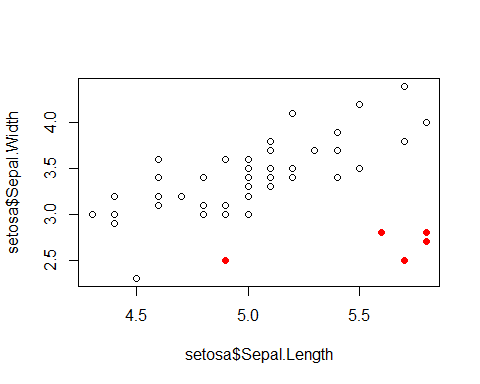
setosa <-iris[which(iris$Species == "setosa"),]

plot(setosa$Sepal.Length, setosa$Sepal.Width)



Agora vamos adicionar outra camada de pontos, desta vez da especie "virginica", em color vermelho. Primeiro plotamos o mesmo gráfico anterior da especie setosa e depois adicionamos a camada.

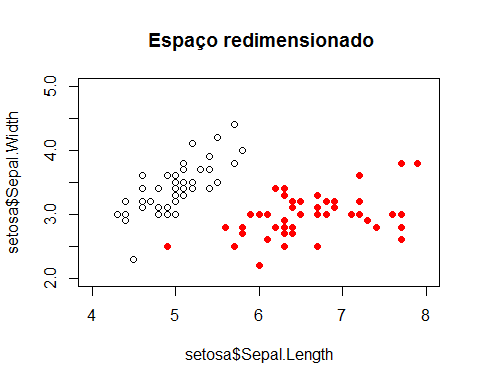
# gráfico anterior  
plot(setosa$Sepal.Length, setosa$Sepal.Width)  
# separamos os dados da espécie "virgínica"  
virginica <-iris[which(iris$Species == "virginica"),]  
# adicionamos os pontos com cor vermelha e ponto cheio  
points(virginica $Sepal.Length, virginica $Sepal.Width, col = "red", pch = 19)



O que aconteceu com as 50 observações da espécie adicionada??

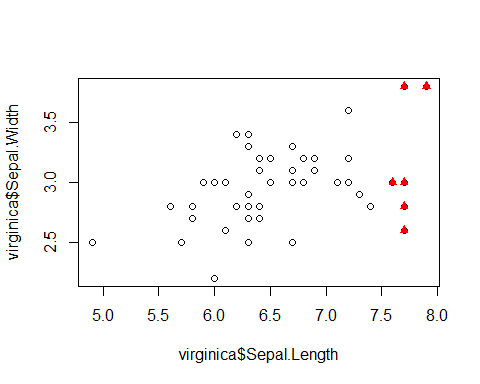
A função *plot()* tem uma limitação na sua execução e é que uma vez aberto o espaço gráfico, ele não é redimensionado pela chegada de novos dados superpostos. O correto é analisar os novos dados e ver de dimensionar adequadamente os eixos. No caso anterior seria...

# gráfico anterior  
plot(setosa$Sepal.Length, setosa$Sepal.Width,  
 xlim = c(4,8),  
 ylim = c(2,5),  
 main = " Espaço redimensionado")  
  
# separamos os dados da espécie "virgínica"  
virginica <-iris[which(iris$Species == "virginica"),]  
# adicionamos os pontos com cor vermelha e ponto cheio  
points(virginica $Sepal.Length, virginica $Sepal.Width, col = "red", pch = 19)



Um outro uso da função *points()* que pode ser de utilidade, é quando precisamos destacar ou mostrar valores determinados ou discrepantes. Vamos supor que nesses dados graficados, todas as medidas sepal.lenght superiores a 7.5 são valores a destacar.

plot(virginica$Sepal.Length, virginica$Sepal.Width)  
  
destacados <- virginica[which(virginica$Sepal.Length > 7.5),]  
  
points(destacados$Sepal.Length,destacados$Sepal.Width, pch =17, col = "red")



A função **plot()** vai sempre tratar de determinar qual o melhor gráfico para seus dados, veja o seguinte exemplo usando o dataframe *iris*, do qual vamos imprimir as primeiras 6 linhas para relembrar a estrutura.

iris <- datasets::iris  
  
head(iris)

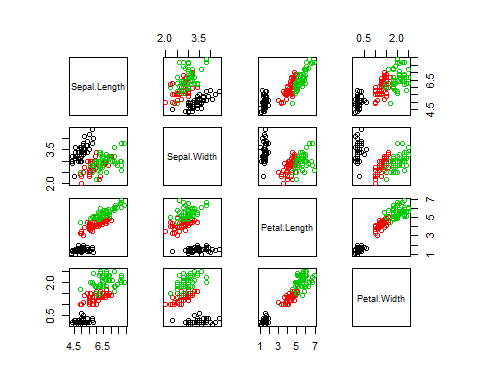
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

plot(iris)



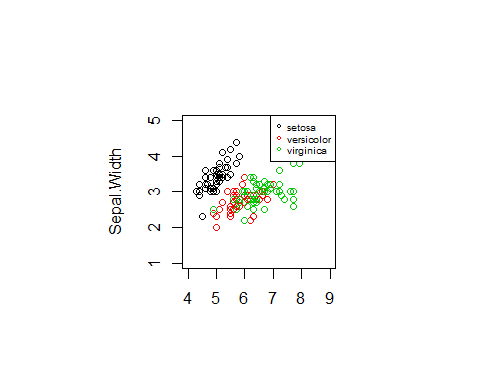
É claro que o gráfico com as espécies como mais uma variável não é informativa. Vamos agora a graficar apenas as primeiras 4 variáveis de interesse e plotar os pontos com diferentes cores, acorde a variável "Species".

iris <- (datasets::iris)  
attach(iris)  
  
par(pro)  
par(oma = c(5,5,5,3),   
 mar = c(1,1,1,1),  
 pty = "s")  
plot(iris[1:4],   
 col= iris$Species)



plot(Sepal.Length,   
 xlim = c(4,9),  
 ylim = c(1,5),  
 Sepal.Width, col=Species)

legend("topright", legend = levels(Species), col = 1:3, cex = 0.6, pch = 1)



FIM DA SEGUNDA PARTE

## Capítulo 3 - Visualização de dados com *ggplot2()*

## a - Conceitos, elementos e primeiro gráfico

#### Filosofia da "Gramática dos Gráficos" - Grammar of Graphics, Leland Wilkinson, 1999

##### princípios básicos:

* + o gráfico está composto de diferentes camadas de elementos
  + a mensagem nos gráficos se obtém adicionando camadas de elementos estéticos

O gráfico é um mapeamento de dados, a partir de atributos estéticos de objetos geométricos

Baseado nesses conceitos, Hadley Wickman em *Layered Grammar of Graphics*, estabelece que um gráfico pode ser construído a partir de camadas que são adicionadas sucessivamente.

Assim surgiu o pacote **ggplot2** (Create Elegant Data Visualisations Using the Grammar of Graphics) de Hadley Wickman e Winston Chang (ver na ajuda *?ggplot2* ).

##### **Elementos básicos no gráfico**

|  |  |
| --- | --- |
| Elemento | Descrição |
| Dados (data) | dados a serem apresentados |
| Estetica (aesthetics) | atributos estéticos como posição, forma, cor, ... |
| Geometrias (Geometries) | Os elementos visuais com os quais representamos os dados (pontos, linhas, barras,...) |

##### **Elementos complementares**

|  |  |
| --- | --- |
| Elemento | Descrição |
| Fatores (Facets) | Gráficos múltiplos por categoria |
| Estatísticas (Statistics) | Representação dos dados para ajudar na compreensão |
| Coordenadas (coordinates) | O espaço onde os dados serão representados |
| Temas (themes) | Aparência ou aspecto |

#### primeiro gráfico

**Template para gráficos com ggplot**

**ggplot (data = <DADOS>, aes(<MAPPINGS>)) +  
   
 <GEOM\_FUNCTION>**

#### Ordenamentos dos dados

dados + aesthetics +   
 geometrias +   
 fatores +   
 estatísticas +   
 coordenadas +  
 temas

Nos próximos exemplos começaremos com um gráfico simples até apresentar um gráfico elaborado e, como diz Hadley, "elegante", perfeitamente adequado para a publicação, inclusão num relatório ou trabalho acadêmico.

#### 1- gráfico simples

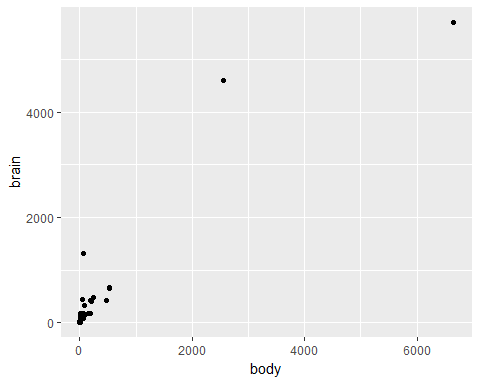
Vamos fazer um gráfico de um conjunto de dados da biblioteca **MASS**, que contém a média de peso corporal e do cérebro de 62 animais mamíferos.

Para o gráfico vamos indicar ao **ggplot** que vamos trabalhar com o dataframe **mammals** e no eixo "x" vamos colocar o peso do corpo e no eixo "y" o do cérebro.

library(MASS)

# estes são os dados e o mapeamento  
ggplot(mammals, aes(x = body, y = brain)) **+**

# camada de geometria adicionada, no caso, pontos  
 geom\_point()



No gráfico acima temos apenas duas camadas: os dados na primeira e o tipo de representação na segunda, apenas pontos. O **ggplot** vai escolher por nós o resto dos elementos como cor dos pontos, limites dos eixos, legendas e aspecto do gráfico.

Todos esses elementos e muitos outros podem ser personalizados como veremos mais adiante.

***Exercício, experimente na consola executar apenas a primeira linha do mapeamento dos dados. O que acontece e porquê? Coloque a segunda camada e veja o resultado,***

#### 2- gráfico com linha de regressão

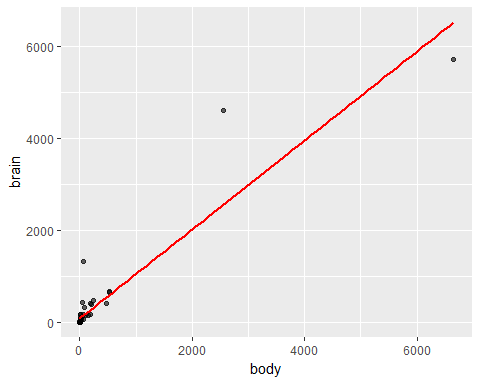
Adicionamos às camadas anteriores uma camada complementar de estatística, o gráfico da linha simples de regressão linear.

Notem que não é necessário fazer nenhum tipo de cálculo prévio, apenas incluir na camada que a metodologia é **"lm"** e indicar uma cor vermelha para o traçado e que não vamos precisar do intervalo de confiança da regressão, indicado por **"se = F"**.

A adição de \*\*\_smooth\*\* na chamada da estatística, introduz um fator de suavizado na linha graficada (ver a ajuda).

Além disso será adicionada uma transparência aos pontos graficados para melhor visualização de sobreposição.

ggplot(mammals, aes(x = body, y = brain)) +  
 geom\_point(alpha = 0.6) +   
 stat\_smooth(method = "lm", col = "red", se = F)



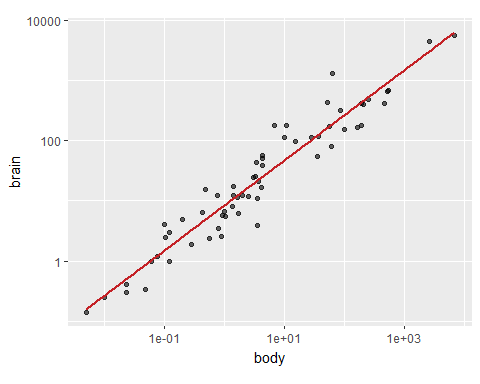
Analisado o gráfico pareceria que os dados necessitam de uma transformação nas unidades, já que existe disparidade de valores e concentração dos valores menores.

Uma transformação log pode resolver o problema. Para manter a relação de unidades entre os eixos, adicionamos uma camada **coord\_fixed()**, que por padrão vai manter a relação y/x = 1.

Na sequência transforma-se os eixos com log10 e finalmente adiciona-se a linha de regressão, indicando a cor em sistema RGB de representação e o tamanho como 1.

#### 3- gráfico com transformação da escala para melhor representação

ggplot(mammals, aes(x = body, y = brain)) +  
 geom\_point(alpha = 0.6) +  
 coord\_fixed() +  
 scale\_x\_log10() +  
 scale\_y\_log10() +  
 stat\_smooth(method = "lm",   
 col = "#C42126",   
 se = F,  
 size = 1)



Nota-se que a transformação melhorou o aspecto do gráfico e o análise dos dados, já que a relação entre as variáveis era logarítmica.

#### Para publicação, com padronização dos intervalos de escala em ambos eixos

Para deixar finalmente nosso gráfico num formato adequado para publicação e apresentação, será necessário padronizar os intervalos em ambos eixos, adicionar um título e subtítulo, indicando a fonte de donde os dados foram obtidos e finalmente deixar o gráfico apenas em preto e branco, já que muitas editoras não permitem outras cores, sem fundo colorido nem grade.

library(scales)

# dados  
ggplot(mammals, aes(x = body, y = brain)) +

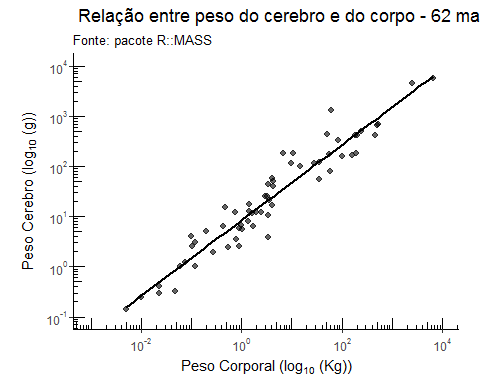
# tipo de geometria   
 geom\_point(size = 1.8, alpha = 0.6) +

# escala em ambos eixos   
 coord\_fixed(xlim = c(10^-3, 10^4), ylim = c(10^-1, 10^4)) +   
 annotation\_logticks() + # adiciona espaços cada vez menores, faz sentido apenas para eixos log

# escala logaritmica "x", titulo  
 scale\_x\_log10(expression ("Peso Corporal (log" ["10"]\*" (Kg))"), # intervalos  
 breaks = trans\_breaks("log10", function(x) 10^x),   
# etiquetas

labels = trans\_format("log10", math\_format(10^.x))) +   
# escala logaritmica "y", titulo

scale\_y\_log10(expression ("Peso Cerebro (log" ["10"]\*" (g))"),   
 breaks = trans\_breaks("log10", function(x) 10^x), # intervalos  
 labels = trans\_format("log10", math\_format(10^.x))) + # etiquetas  
 stat\_smooth(method = "lm", # linha de regressão  
 col = "black", # cor  
 se = F, # sem intervalo de confiança  
 size = 1) + # tamanh0  
 labs(title = " Relação entre peso do cerebro e do corpo - 62 mamiferos", # título e sub  
 subtitle = "Fonte: pacote R::MASS ") +  
 theme\_classic() # tema de fundo



### Adicionando todas as camadas num gráfico.

Já vimos que um gráfico na metodologia **ggplot** se compõe de diferentes camadas. Por questões de simplicidade clareza, a ordem de adição das camadas é:

#### Ordenamentos dos dados

**dados + aesthetics + geometrias + fatores + estatísticas + coordenadas + temas**

#### Um exemplo completo:

Os dados a serem usados são de **datasets::iris** e contém medições de sépala e pétalas de 50 amostras de três especies da flor **iris** (ver artigo na wikipedia <https://en.wikipedia.org/wiki/Iris_flower_data_set>).

##### 1- conhecendo os dados: iris

head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 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

str(iris)

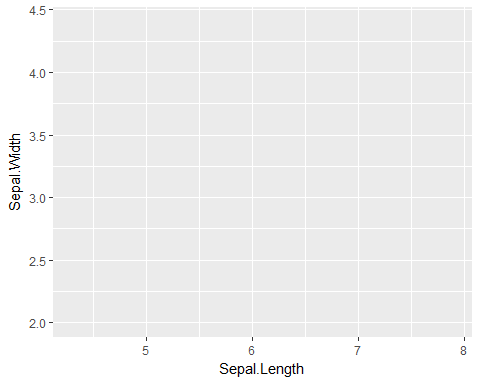
## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

Os dados a serem graficados serão os que pertencem às primeiras e segunda coluna (variáveis) e correspondem à largura e comprimento da sépala das três espécies. Assim sendo, o argumento estético do mapeamento dos valores vai ser

**aes(x = Sepal.Length, y = Sepal.Width)**

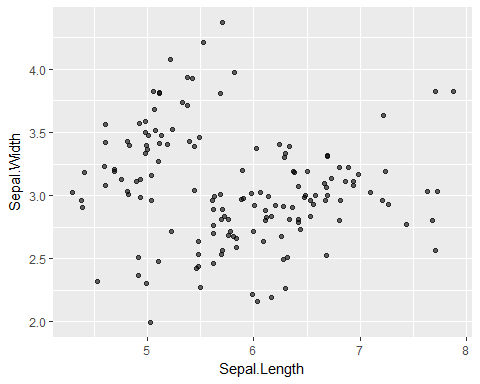
A *primeira camada* contendo os dados e mapeamento dos valores (aesthetics) é:

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))



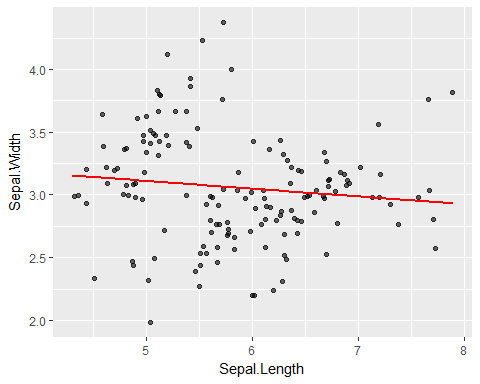
A *segunda camada* corresponde a geometria que representará os dados, neste caso, pontos. Para melhorar a visualização vamos permitir um pequeno deslocamento dos pontos que se sobrepõem. Isso é feito através de **geom\_jitter**, que é uma variação de **geom\_point**. Adicionamos uma transparência 0.6, que permite distinguir os pontos sobrepostos, isso é feito na própria camada de geometria.

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
 geom\_jitter(alpha = 0.6)



A *terceira camada* incorpora uma linha de regressão, na cor vermelha e sem intervalo de confiança (se = F).

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
 geom\_jitter(alpha = 0.6) +  
 stat\_smooth(method = "lm", se = F, col = "red")



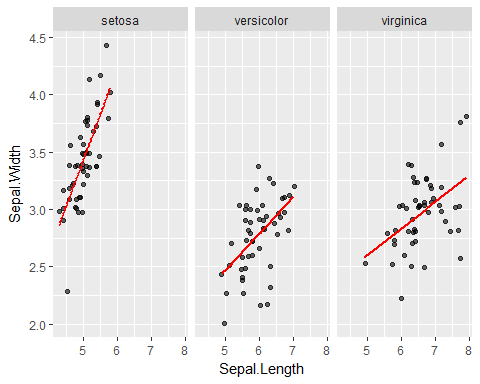
Como temos três espécies diferentes no gráfico (setosa, versicolor e virginica), a linha de regressão não é representativa dos diferentes grupos, precisamos de uma regressão por espécie.

A *quarta camada* vai permitir dividir os dados por espécies (fatores) e apresentar três gráficos separados, um para cada espécie.

Isto permite comparar as distribuições e valores para cada espécie, além de apresentar as linhas de regressão por separado.

A fatorização dos dados e a divisão em três gráfico é feita com a camada **facet\_grid**, indicando no primeiro argumento com o ponto "**.**", os dados a serem fatorizados (que vem da primeira camada) e sendo o segundo argumento o critério ou coluna usada como fator.

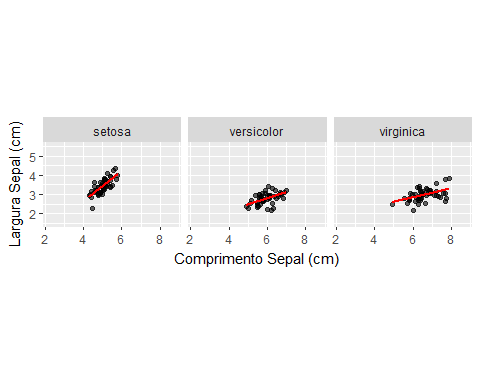
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
 geom\_jitter(alpha = 0.6) +  
 stat\_smooth(method = "lm", se = F, col = "red") +  
 facet\_grid(. ~ Species)



Na *quinta camada* (coordenadas) vamos incluir escalas de coordenadas personalizadas. Ver ajuda para **scale\_x\_continuos, scale\_y\_continuos e coord\_equal**

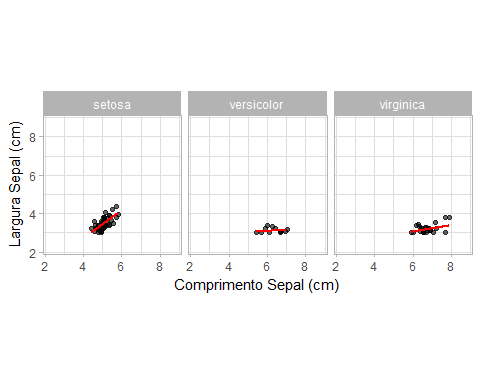
Com isto colocamos todos os gráficos na mesma escala numérica e de aparência. Note que nas camadas **scale**, colocamos a legenda de cada eixo, os limites e a distância do eixo que serão colocados (veja ajuda para mais detalhes).

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
 geom\_jitter(alpha = 0.6) +  
 facet\_grid(. ~ Species) +  
 stat\_smooth(method = "lm", se = F, col = "red") +  
 scale\_x\_continuous("Comprimento Sepal (cm)",  
 limits = c(3,8),  
 expand = c(0.2,0.1)) +  
 scale\_y\_continuous("Largura Sepal (cm)",  
 limits = c(2,5),  
 expand = c(0.2,0.1)) +  
 coord\_equal()



Finalmente na *sexta camada* adicionamos um tema ou aparença personalizada, no caso adicionamos um tema já preexistente no pacote o, **theme\_light**.

ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
 geom\_jitter(alpha = 0.6) +  
 facet\_grid(. ~ Species) +  
 stat\_smooth(method = "lm", se = F, col = "red") +  
 scale\_x\_continuous("Comprimento Sepal (cm)",  
 limits = c(3,8),  
 expand = c(0.2,0.1)) +  
 scale\_y\_continuous("Largura Sepal (cm)",  
 limits = c(3,8),  
 expand = c(0.2,0.1)) +  
 coord\_equal() +  
 theme\_light()



FIM DA PRIMEIRA PARTE

## Capítulo 3 - Visualização de dados com *ggplot2()*

## Reforço de Exercícios

Vamos a praticar os conhecimentos vistos e completar o primeiro ggplot2.

Usaremos a base de dados ***datasets::mtcars***.

#### ***Procure na consola ?mtcars e conheça os dados.***

Como primeiro passo:

#### ***carregue a biblioteca ggplot2, use head() e str() para conhecer os dados, se deseja ver o intervalo de valores de cada variável, use summary(), e finalmente salve os dados como "mtc".***

library(ggplot2)  
head(mtcars)

## mpg cyl disp hp drat wt qsec vs am gear carb  
## Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4  
## Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
## Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1  
## Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
## Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
## Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

str(mtcars)

## 'data.frame': 32 obs. of 11 variables:  
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...  
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...  
## $ disp: num 160 160 108 258 360 ...  
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...  
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...  
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...  
## $ qsec: num 16.5 17 18.6 19.4 17 ...  
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...  
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...  
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...  
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...

summary(mtcars)

## mpg cyl disp hp   
## Min. :10.40 Min. :4.000 Min. : 71.1 Min. : 52.0   
## 1st Qu.:15.43 1st Qu.:4.000 1st Qu.:120.8 1st Qu.: 96.5   
## Median :19.20 Median :6.000 Median :196.3 Median :123.0   
## Mean :20.09 Mean :6.188 Mean :230.7 Mean :146.7   
## 3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0 3rd Qu.:180.0   
## Max. :33.90 Max. :8.000 Max. :472.0 Max. :335.0   
## drat wt qsec vs   
## Min. :2.760 Min. :1.513 Min. :14.50 Min. :0.0000   
## 1st Qu.:3.080 1st Qu.:2.581 1st Qu.:16.89 1st Qu.:0.0000   
## Median :3.695 Median :3.325 Median :17.71 Median :0.0000   
## Mean :3.597 Mean :3.217 Mean :17.85 Mean :0.4375   
## 3rd Qu.:3.920 3rd Qu.:3.610 3rd Qu.:18.90 3rd Qu.:1.0000   
## Max. :4.930 Max. :5.424 Max. :22.90 Max. :1.0000   
## am gear carb   
## Min. :0.0000 Min. :3.000 Min. :1.000   
## 1st Qu.:0.0000 1st Qu.:3.000 1st Qu.:2.000   
## Median :0.0000 Median :4.000 Median :2.000   
## Mean :0.4062 Mean :3.688 Mean :2.812   
## 3rd Qu.:1.0000 3rd Qu.:4.000 3rd Qu.:4.000   
## Max. :1.0000 Max. :5.000 Max. :8.000

mtc <- mtcars

#### ***Responda às seguintes preguntas:***

1- qual as dimensões do dataframe?

2- quantas variáveis tem e de que classe?

3- que variáveis ao seu critério poderiam ser fatores para um posterior análise?

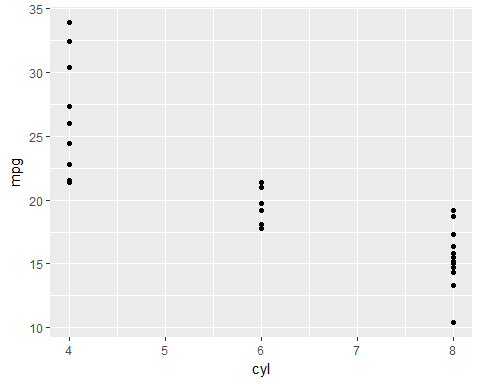
#### ***Segundo passo: fazer o primeiro gráfico ggplot usando o dataframe criado, "mtc".***

O gráfico a ser feito, deve mostrar como pontos os valores que terão:

* eixo "x" a variável "cyl" (número de cilindros) e
* eixo "y" a variável "mpg" (milhas por galão).

**Solução:**

ggplot(mtc, aes(x = cyl, y = mpg)) +  
 geom\_point()



Notamos duas coisas no gráfico:

1- o consumo tende a ser maior quanto maior o número de cilindros

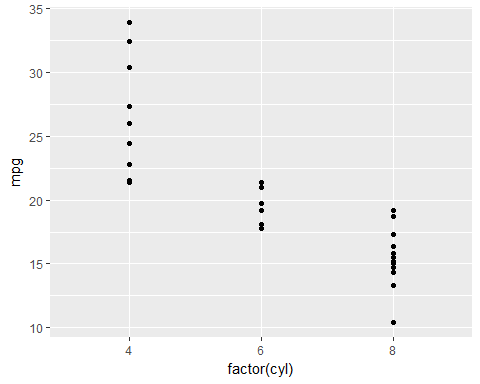
2- como a variável "cyl" foi tratada como contínua, temos valores de quantidade de cilindros que não existem nos dados, por exemplo: 3, 5 ou 7.

Para solucionar esse problema, deve ser explicitamente colocado que a variável "cyl" não é contÍnua, mas categórica. Isso pode ser feito através do uso de **fator(cyl)** na estética dos dados.

Terceiro passo:

#### ***usando o fragmento de código anterior, refaza o gráfico com "cyl" como fator.***

ggplot(mtc, aes(x = factor (cyl), y = mpg)) +  
 geom\_point()



Veja que agora o gráfico contém apenas a quantidade certa de cilindros.

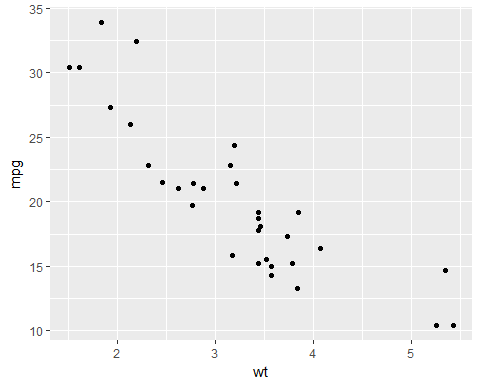
#### ***Repita o exercício escolhendo outras variáveis, use factor() se necessário.***

Vamos explorar um pouco mais as camadas complementares que já vimos

#### ***A seguir faça um gráfico do peso do carro, "wt" versus a quantidade de milhas por galão, "mpg", use a geometria de pontos sem nenhuma especificidade.***

Espera-se ver que o rendimento diminua enquanto o peso aumenta.

ggplot(mtc, aes(x = wt, y = mpg)) +  
 geom\_point()



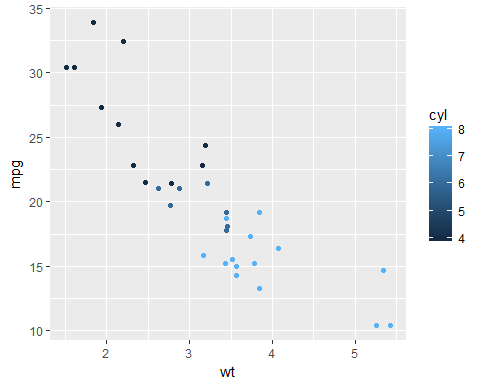
No seguinte passo, vamos adicionar ao código anterior, uma cor acorde o número de cilindros.

Para definir o color dos nossos pontos, usamos **col = ..."** dentro do **aes** onde tanto podemos indicar tanto uma cor só como uma escala de cores.

Como a intenção é colorir os pontos acorde ao número de cilindros podemos usar a variável "cyl" (col = cyl).

**Solução:**

ggplot(mtc, aes(x = wt, y = mpg, col = cyl)) +  
 geom\_point()



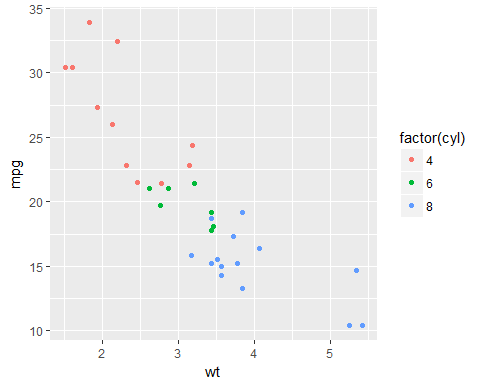
Veja que a legenda foi automáticamente gerada e colocada à direita.

Note também o mesmo problema anterior da variável "cyl" sendo tratada como contínua, nos introduze no gráfico cores e números de cilindros indesejados.

O problema se resolve, da mesma forma, colocando a variável como **factor(...)**, ou se desejar uma solução mais radical, transforme a variável em fator no dataframe.

#### ***Refaça o gráfico com as cores como variável categórica.***

ggplot(mtcars, aes(x = wt, y = mpg, col =factor(cyl))) +  
 geom\_point()



No gráfico, conseguimos individualizar o rendimento olhando para três variáveis ao mesmo tempo, peso, milhas por galão e número de cilindros, esse último representado pela cor.

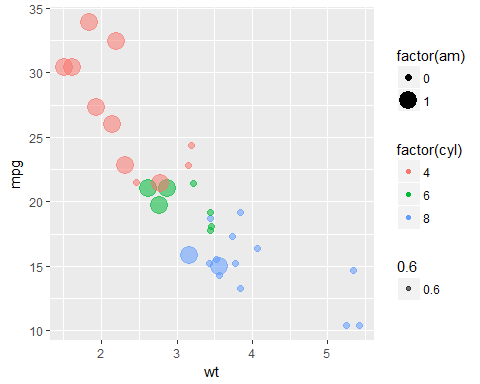
E se fosse possível adicionar uma outra variável, como por exemplo, tipo de transmissão?

A variável "am" nos indica se o carro é automático (0) o não é (1).

Podemos colocar o tamanho do ponto (outra estética ou aparência), como mais um elemento gráfico. Isso é feito com o argumento **size\* =** e colocando a variável "am " com fator novamente.

Finalmente é adequado adicionar uma transparência como já feito anteriormente. Essas novas opções são adicionadas no **aes()** da primeira camada.

ggplot(mtcars, aes(x = wt, y = mpg, col =factor(cyl), size = factor(am), alpha = 0.6)) +  
 geom\_point()



Mais sobre estética...

Nos exercícios anteriores mapeamos as cores tanto numa escala contínua como categórica.

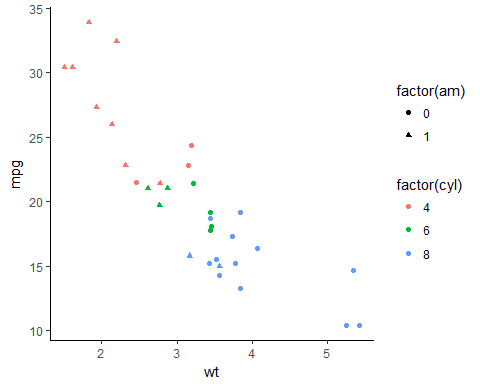
Um outro elemento estético é a forma dos pontos, dada por **shape**. Porém existem uma quantidade finita de formas a serem atribuídas a pontos, por isso se tentamos mapear a forma com uma variável contínua, teremos uma mensagem de erro.

***A forma como elemento estético diferenciador tem sentido apenas para variáveis categóricas!!!***

#### ***Mude o fragmento anterior de código para mapear em vez do tamanho, a forma (shape).***

#### ***Retire o argumento alpha, adicione o tema clássico e veja o resultado.***

ggplot(mtcars, aes(x = wt, y=mpg, col =factor(cyl), shape = factor(am)))+  
 geom\_point() +  
 theme\_classic()



#### Um cuidado muito especial deve-se ter em ter sempre o gráfico o menos poluído possível. Se desejamos comparar ou mostrar várias variáveis, analise se não é melhor preparar mais de um gráfico.

No gráfico acima estamos mostrando quatro variáveis, "wt" e "mpg" nos eixos X e Y e duas com argumentos de estética, "cyl" e "am".

FIM DE EXERCÍCIOS 01

### Resumo de elementos estéticos

| Elemento estético (aes) | Descrição |

|---------------------------- ----|----------------------------------------------|

| x | variável a ser mapeada no eixo x |

| y | variável a ser mapeada no eixo y |

|colour ou col |cor dos pontos ou formas geométrica|

| fill | cor de preenchimento |

| size | tamanho dos pontos ou da geometria|

| alpha | transparência em percentagem |

| linetype | tipo de linha a ser desenhada |

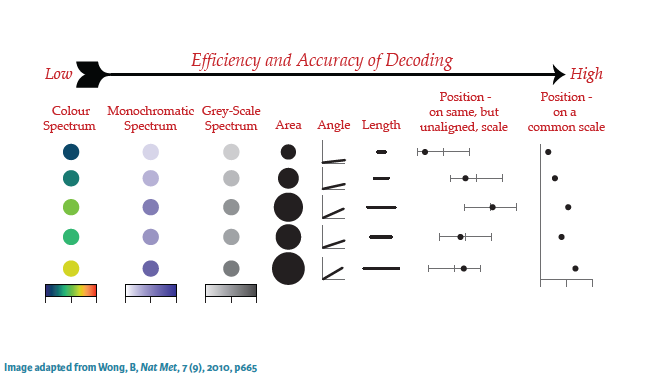
| labels | texto a colocar do gráfico ou eixos |

| shape | forma |

Recomendações para uma eficiente decodificação das variáveis num gráfico

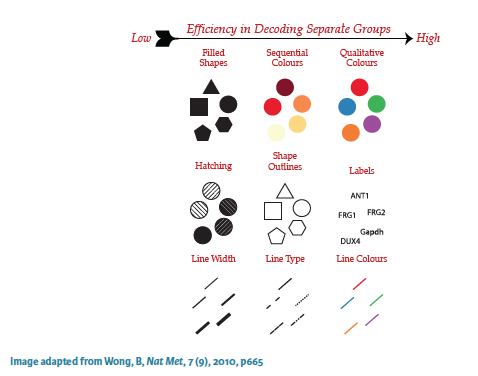
**Variáveis contínuas (**do menos eficiente ao mais eficiente):

* + - * espectro ou paleta de cores
      * espectro monocromático
      * espectro na escala de cinzas
      * área (elementos diferenciados pelo tamanho)
      * ângulo e largura de linhas
      * posição de diversos elementos no mesmo gráfico
      * posição de diversos elementos numa escala comun



**variáveis categóricas**

* formas de tipo cheio ou preenchidas
  + diferentes formas numa cor só
  + uma forma em cores sequenciais
  + uma forma seguindo teoría das cores
* formas 2
  + uma forma hachurada com diferentes grossores
  + formas diferentes e vazias
  + etiquetas com texto
* linhas
  + diferente largura
  + diferentes tipos
  + diferentes cores



FIM DO EXERCÍCIO 01

## Capítulo 3 - Visualização de dados com *ggplot2()*

## c - Elementos gráficos

Na continuação dos nossos exercícios com ggplot, vamos trabalhar com a base de dados **ggplot2::diamond**.

Como elemento informativo carat(quilate) da Wikipédia:

"No que se refere a pedras preciosas, como o diamante, **um quilate representa uma massa igual a duzentos miligramas**.

A unidade de massa foi adotada em 1907 na Quarta Conferência Geral de Pesos e Medidas. O quilate pode ser subdividido ainda em 100 pontos de 2 mg cada.""

O objetivo desta parte é reforçar o aprendido e colocar alguns elementos novos em **ggplot**.

Como a base de dados é grande, vamos selecionar um conjunto de 2000 registros em forma aleatória. Com fins de compatibilidade, colocaremos uma semente, faremos a seleção usando a função **dplyr::sample()** e chamaremos os novos dados de "diamantes".

library(dplyr)

set.seed(20170620)  
diamantes <- sample\_n(diamonds, 2000)

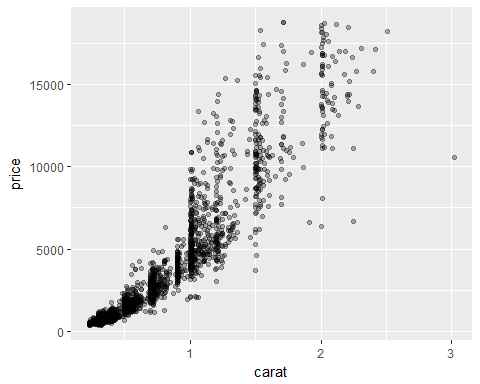
#### ***Use str() para conhecer os dados, veja o tamanho do arquivo e as variáveis.***

Os dados contém informação referente a diversas características de diamantes, como preço, corte, color, dimensões de cada um, etc.

Nos exercícios anteriores, vimos como mapear os pontos através de **aes()** e graficar com **geom\_point()** e **geom\_jitter**.

No primeiro gráfico vamos fazer um gráfico de pontos de "carat" e "price", colocando uma transparencia de 0.3 na estética da camada de pontos.

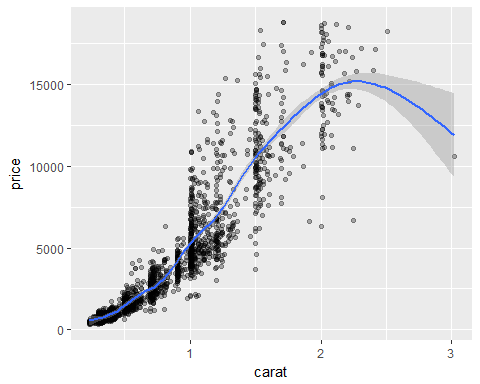
ggplot(diamantes, aes(x = carat, y = price)) +  
 geom\_point(alpha = 0.3)



Na sequência vamos adicionar ao gráfico uma camada complementária de estatística. Para isso usamos sempre o "+" e vamos colocar apenas uma camada **geom\_smooth()**.

**Solução**

ggplot(diamantes, aes(x = carat, y = price)) +  
 geom\_point(alpha = 0.3) +  
 geom\_smooth() ## `geom\_smooth()` using method = 'gam'

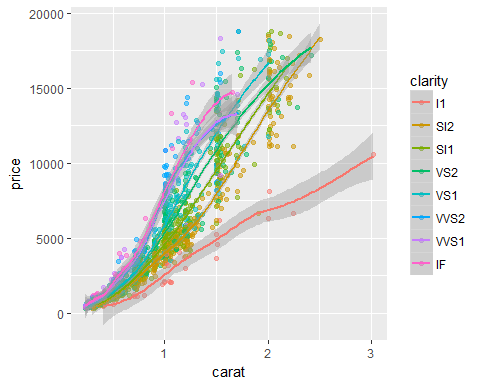


Devido ao tamanho do arquivo, **geom\_smooth()** avisa que para suavizar a curva usou o método ***gam***. Veja na ajuda referente a metodologia e parâmetros do método.

Vamos continuar combinando geometrias.

#### ***Com o fragmento de código anterior, adicione uma cor usando a variável "clarity".***

ggplot(diamantes, aes(x = carat, y = price, col = clarity)) +  
 geom\_point( alpha = 0.5) +  
 geom\_smooth() ## `geom\_smooth()` using method = 'loess'



Note que como a cor foi colocada na camada primeira, ela é considerada um argumento global para todo o gráfico e além de determinarcores diferentes aos pontos, para cada grupo de "clarity", também calculou e colocou uma curva para cada grupo de "clarity", adicionando a legenda do lado direito.

O gráfico acima ficou muito confuso. Temos várias possibilidades de melhorá-lo:

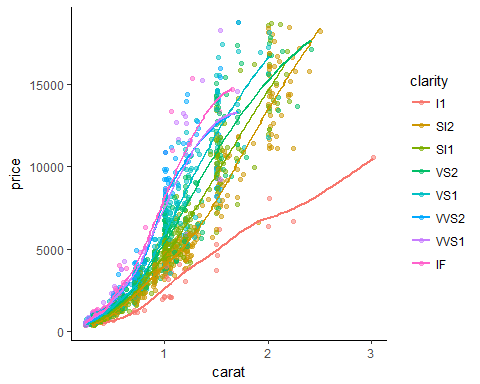
#### ***1- Adicione o argumento "se = F"" para tirar o intervalo de confiança das linhas, ou***

#### ***2- Deixe apenas as linhas e retire a camada de pontos,***

#### ***3- Adicione um tema menos poluído, como "theme\_classic()".***

#### ***Experimente !!***

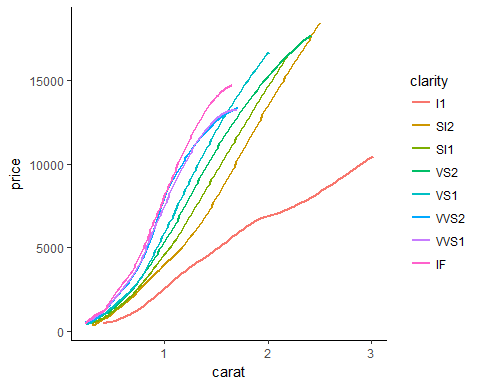
ggplot(diamantes, aes(x = carat, y = price, col = clarity)) +  
 geom\_point( alpha = 0.5) +  
 geom\_smooth(se = F) +  
 theme\_classic() ## `geom\_smooth()` using method = 'loess'



Podemos ainda limpar mais o gráfico e mostrar apenas as linhas para cada grupo.

#### ***Usando o código anterior, elimine a camada de pontos e veja o resultado.***

ggplot(diamantes, aes(x = carat, y = price, col = clarity)) +  
 geom\_smooth(se = F) +  
 theme\_classic()



### Mais sobre a **gramática dos gráficos**

A possibilidade de combinar diversos elementos gráficos em forma clara e inteligente é uma das melhores caraterísticas do **ggplot**.

A primeira camada, ou as que desejar, podem ser salvas como um objeto **ggplot** e adicionar as restantes a esse objeto.

Por exemplo,

graf1 = ggplot(dados, aes(x = "X", y = "Y"))  
  
graf2 = graf1 +  
 geom\_points()

Na sequência iniciaremos um fragmento de código a partir de **diamantes** e voces irão adicionando elementos gráficos em camadas para finalizar gráficos elegantes e informativos.

#### ***Faça um gráfico, mapeando "carat" no eixo x e "price" no eixo y, salve como "graf.diam".***

graf.diam <- diam <- ggplot(diamantes, aes(x = carat, y = price))

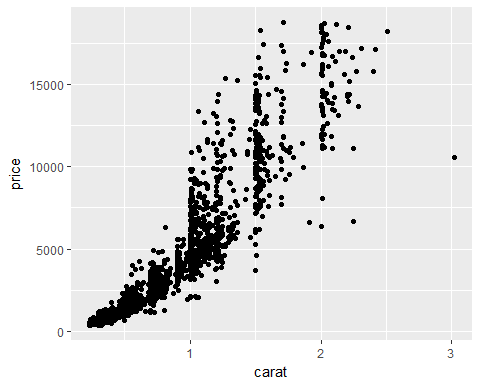
O objeto ggplot "graf.diam", deve aparecer na sua janela de ambiente (janela 2),

#### ***tome um tempo para analisar o objeto, clicando na seta do lado do nome.***

#### ***adicione ao objeto criado uma camada de pontos, geom\_point(), salve como graf.diam2.***

#### ***Para visualizar deve chamar o objeto.***

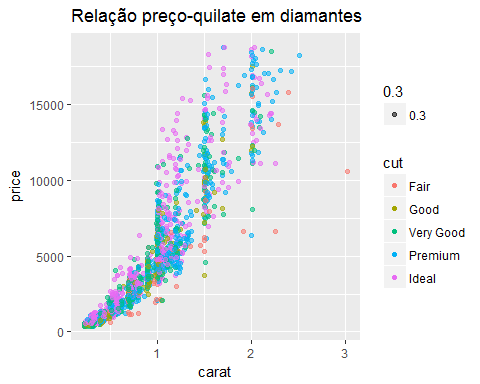
graf.diam2 <- graf.diam +  
 geom\_point()  
graf.diam2



O argumento da estética **aes()** pode ser colocado dentro da geometria para personalizar a camada.

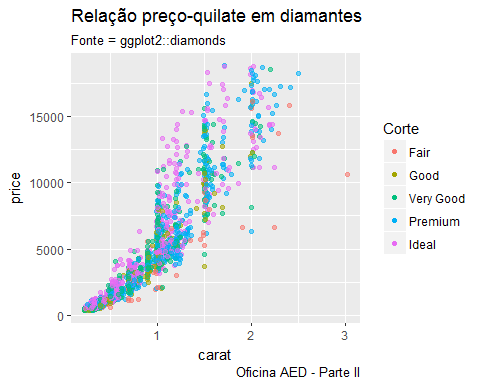
***coloque aes() dentro da camada de pontos, especificando a cor acorde a variável "cut", uma transparência de 0.3 e mude a camada para geom\_jitter, adicione um titulo com "labs(title = 'Seu titulo')". Salve como graf.diam2.***

graf.diam2 <- graf.diam +  
 geom\_jitter(aes(col = cut, alpha = 0.3)) +  
 labs(title = 'Relação preço-quilate em diamantes')  
graf.diam2



Se desejamos remover a legenda da transparência que não contribui no gráfico, apenas adicione a linha **guides(alpha=FALSE)**, mas isso deve ser feito no passo anterior já que se usamos o objeto salvo graf.diam2, ele já traz a legenda embutida. Aproveitamos e mudamos o título da legenda para "Quilates" e adicionamos um subtitulo e um pê de página. Veja no exemplo:

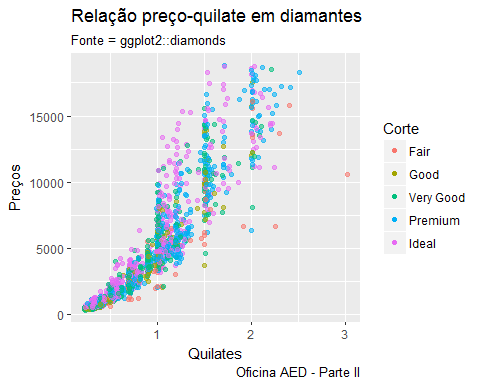
graf.diam2 <- graf.diam +  
 geom\_jitter(aes(col = cut, alpha = 0.3)) +  
 labs(title = 'Relação preço-quilate em diamantes', subtitle = "Fonte = ggplot2::diamonds") +  
 guides(alpha=FALSE) +  
 labs(col = "Corte")+  
 labs(caption = "Oficina AED - Parte II")  
graf.diam2



As legendas nos eixos podem ser trocadas e personalizadas com o argumento "labs( x = "novo nome")"

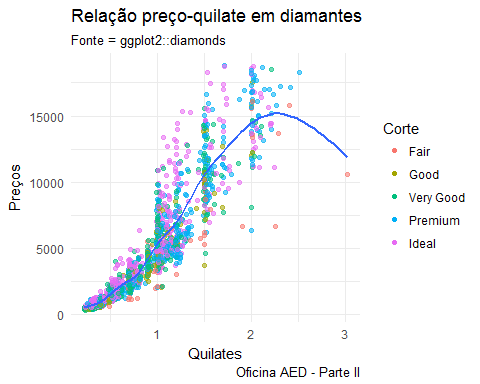
#### ***Coloque em graf.diam2 o título "Quilates" no eixo x e "Preço" no eixo y. Salve como graf.diam3.***

graf.diam3 <- graf.diam2 +  
 labs(x = "Quilates") +  
 labs(y = "Preços")  
graf.diam3



#### ***Finalmente, adicione uma camada de estatística (geom\_smooth, sem intervalo de confiança) e escolha um tema do seu gosto para finalizar o exercício. Salve como graf.diam4 e visualize.***

graf.diam4 <- graf.diam3 +  
 geom\_smooth(se = FALSE) +  
 theme\_minimal()  
graf.diam4



FIM DE EXERCÍCIOS 02

## Capítulo 3 - Visualização de dados com *ggplot2()*

## d- modificação de parâmetros estéticos

#### Exercícios 03

Até agora vimos apenas o elemento estético geométrico de ponto e linha suavizada. Temos diversos outros elementos geométricos que podemos usar na camada de geometria, os mais usados:

* + geom\_bar
  + geom\_boxplot
  + geom\_histogram
  + geom\_curve
  + geom\_abline
  + geom\_quantile
  + …

#### Gráfico de barras

O gráfico de barras é um dos mais usados para visualizar dados categóricos.

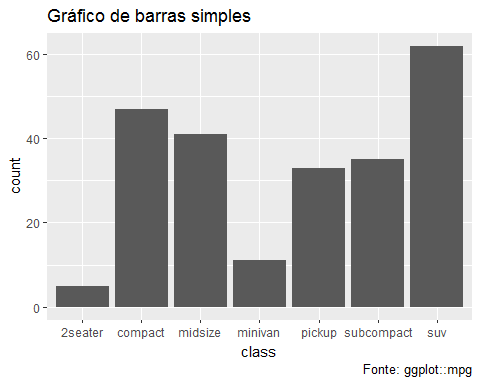
As barras representam o número ou proporção de elementos em cada categoria, fator ou nivel.

Primeiro um gráfico simples de barra a partir dos dados em **ggplot::mpg**, em que usaremos a variável "class".

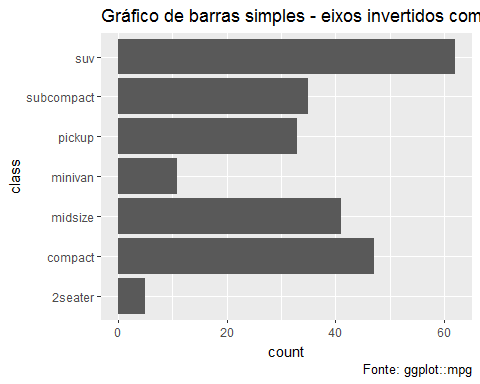
#### ***conheça seus dados!***

--> ***head, str, summary !!***

# primeira camada com os dados e o mapeamento das variáveis  
graf <- ggplot(mpg, aes(class))  
# Gráfico de barras com número de tipos de carros  
graf +   
 geom\_bar() +  
 labs(title ="Gráfico de barras simples") +  
 labs(caption ="Fonte: ggplot::mpg")



graf +   
 geom\_bar() +   
 coord\_flip() +  
 labs(title ="Gráfico de barras simples - eixos invertidos com coord\_flip()", caption ="Fonte: ggplot::mpg")



A contagem por categoria pode ser dividida em diversas classes acorde uma outra variável. Vamos usar a variável **drv**, que indica o tipo de transmissão usado pelo carro, **f** = transmissão dianteira, **r** = traseira e **4wd** = nas quatro rodas.

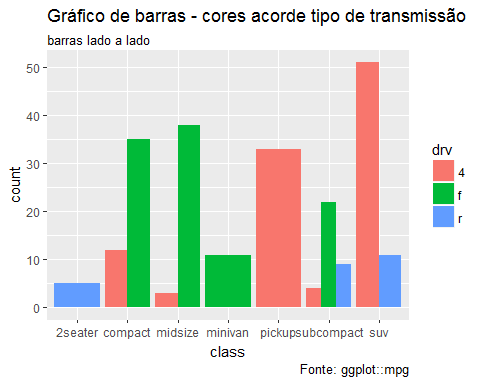
Para cada classe de carro vamos então a ter até três barras diferentes, dependendo do tipo de transmissão usado.

As barras para cada categoria podem ser colocadas *lado a lado* ou *empilhadas* na categoria para completar o 100% (1.0 na escala) de observações em cada categoria.

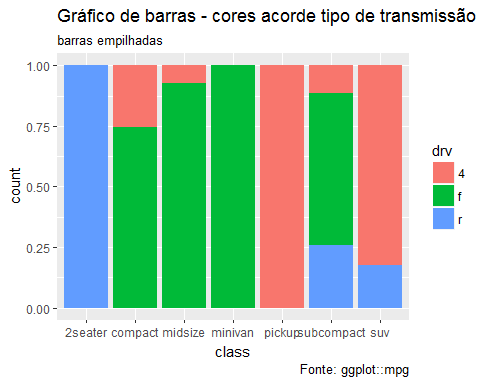
O gráfico será colorido, através do argumento **"fill= ..."**, e as barras colocadas lado a lado, **position = "dodge"** ou empilhadas **position = "fill"** como no segundo gráfico.

Note o uso da camada **labs** para colocar título, subtítulo e nota de pê de página. A camada pode ser única ou dividida por argumentos para maior clareza ou conveniência.

# Gráfico com barras lado a lado  
graf +   
 geom\_bar(aes(fill = drv), position = "dodge") +  
 labs(title = "Gráfico de barras - cores acorde tipo de transmissão",   
 subtitle = "barras lado a lado",  
 caption = "Fonte: ggplot::mpg")



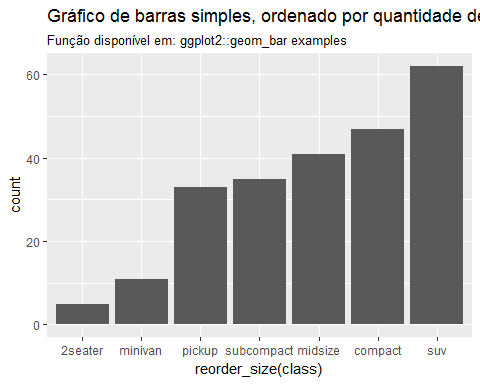
# Gráfico com barras empilhadas  
graf +   
 geom\_bar(aes(fill = drv), position = "fill") +  
 labs(title = "Gráfico de barras - cores acorde tipo de transmissão",   
 subtitle = "barras empilhadas",  
 caption = "Fonte: ggplot::mpg")



Um problema recorrente quando queremos fazer um gráfico por categorias é a ordem em que aparecem, que obedece apenas a uma classificação alfabética ou as vezes à ordem em que se apresentam as variáveis.

Para trocar o ordenamento, podemos fazer uma pequena função que calcula uma tabela de frequências dos dados e logo após ordena em forma crescente por número de elementos em cada categoria.

# função que ordena as categorias por ordem crescente  
reorder\_size <- function(x) {  
 factor(x, levels = names(sort(table(x))))  
}  
  
# gráfico de barras com as categorias ordenadas por número de elementos  
ggplot(mpg, aes(reorder\_size(class))) +   
 geom\_bar() +  
 labs(title ="Gráfico de barras simples, ordenado por quantidade de elementos") +  
 labs(subtitle = "Função disponível em: ggplot2::geom\_bar examples")

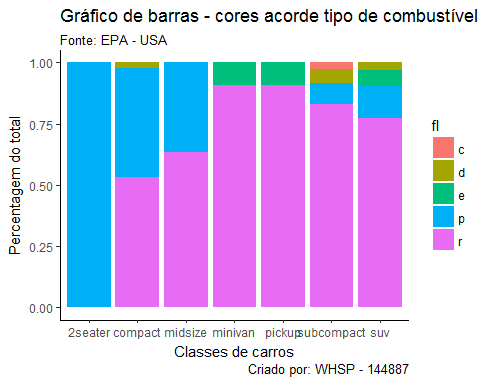


#### ***EXERCICIO***

# Exercício  
# faça um gráfico de barras empilhado do tipo de carro, separado por tipo de combustivel  
# adicione um título, subtítulo e pê de página (seu nome e SIAPE) e tema a sua escolha  
# coloque nomes nos eixos

# Solução:  
# Gráfico com barras empilhadas

ggplot(mpg, aes(class)) +   
 geom\_bar(aes(fill = fl), position = "fill") +  
 labs(title = "Gráfico de barras - cores acorde tipo de combustível",   
 subtitle = "Fonte: EPA - USA",  
 caption = "Criado por: WHSP - 144887") +  
 xlab ("Classes de carros") +  
 ylab ("Percentagem do total") +  
 theme\_classic()

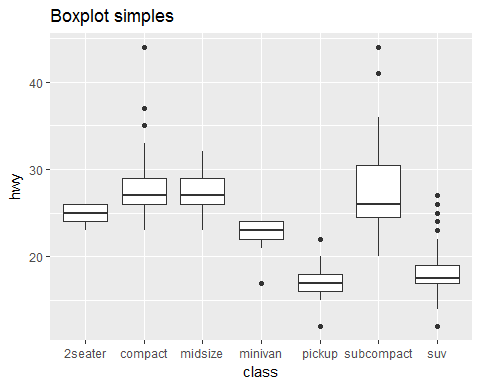


#### Gráfico boxplot

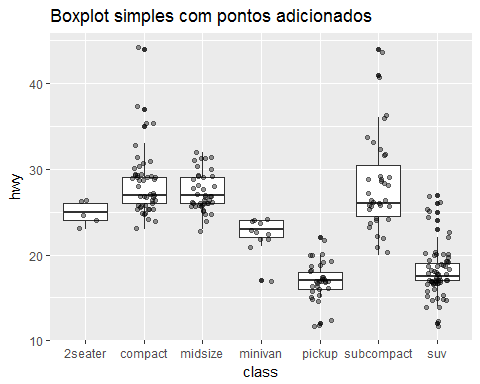
Outro gráfico muito usado é o **boxplot**. Ele nos ajuda a identificar valores discrepantes ou **outliers**. Como sempre começamos com uma camada de dados e o seu mapeamento (aes) para logo após adicionar as camadas complementares.

A camada para boxplot é **geom\_boxplot**. Faremos no primeiro gráfico um boxplot simples, por categoria de carro e no segundo gráfico vamos adicionar uma terceira camada com os pontos, veja as opções de estética dessa camada. ***comente***.

# primeira camada com os dados e o mapeamento das variáveis  
graf2 <- ggplot(mpg, aes(class, hwy))  
  
# segunda camada com o boxplot + terceira com título  
graf2 +   
 geom\_boxplot() +  
 labs(title ="Boxplot simples")



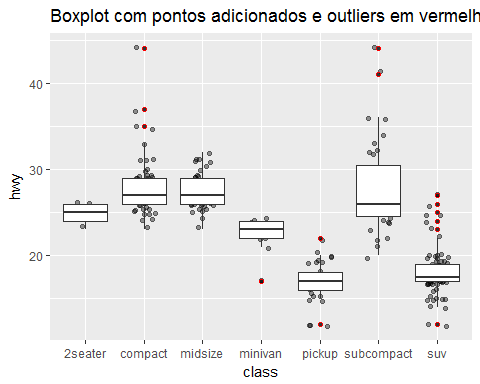
# segunda camada com o boxplot + terceira com pontos + quarta com título  
graf2 +   
 geom\_boxplot() +   
 geom\_jitter(width = 0.2, alpha=0.4) +  
 labs(title ="Boxplot simples com pontos adicionados")



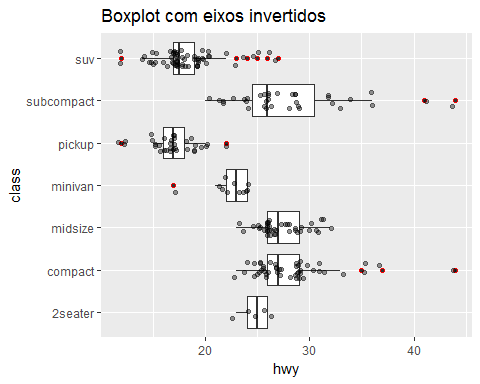
Note que no segundo gráfico, e devido ao tipo de geometria adicionado (geom\_jitter) foi possível discriminar os diversos pontos que componem os outliers visualizados como únicos no primeiro gráfico.

Esses outliers podem ser destacados da seguinte forma (ver a ordem das camadas e o gráfico resultado):

# segunda camada com o boxplot   
# + terceira com pontos  
# + quarta boxplot colorindo de vermelho os outliers   
# + quinta camada com título  
graf2 +   
 geom\_boxplot() +   
 geom\_jitter(width = 0.2, alpha=0.4) +  
 geom\_boxplot(outlier.colour = "red", outlier.shape = 1.8) +  
 labs(title ="Boxplot com pontos adicionados e outliers em vermelho")



# o boxplot também pode ter os eixos invertidos  
graf2 + geom\_boxplot() +  
 geom\_boxplot(outlier.colour = "red", outlier.shape = 1.3) +  
 geom\_jitter(width = 0.2, alpha=0.4) +   
 coord\_flip() +  
 labs(title ="Boxplot com eixos invertidos")



#### ***EXERCICIO***

# Crie um gráfico de boxplot das variáveis "drv" e "cty"  
# grafique os pontos com separação (jitter)  
# coloque títulos nos eixos e principal  
# deixe os boxplot na horizontal  
# adicione um tema a sua escolha

ggplot(mpg, aes(drv, cty)) +

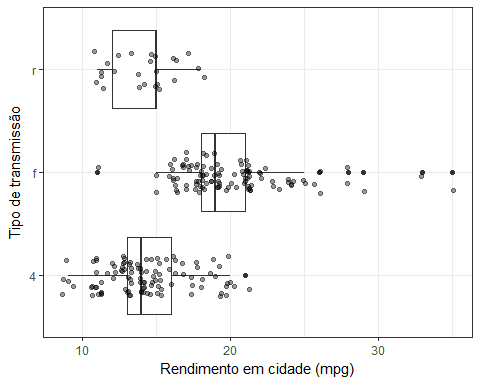
geom\_boxplot() +  
 geom\_jitter(width = 0.2, alpha=0.4) +

ylab ("Rendimento em cidade (mpg)") +

xlab ("Tipo de transmissão") +

coord\_flip() +

theme\_bw()



#### Gráfico de histogramas

O histograma calcula-se em forma semelhante ao gráfico de barras mas para variáveis quantitativas. O procedimento consiste em dividir os elementos da variável em grupos ou "bins" acorde seu valor e contar a quantidade de elementos em cada um eles.

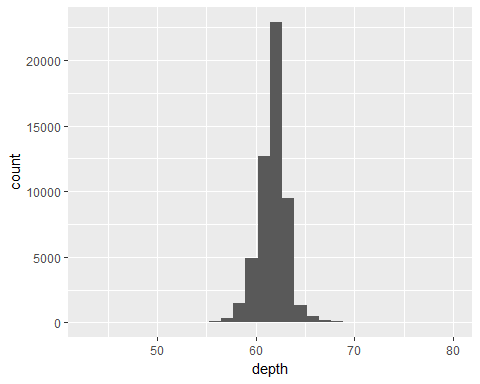
Feito isso o gráfico é construído com os valores da variável de cada "bin" no eixo "x" e a quantidade de elementos no eixo "y". Diferente do gráfico de barras, os grupos são contínuos, como a variável, e graficados sem espaço entre eles.

A **amplitude** dos grupos é calculada automáticamente pela argumento "binwidth", ou pode ser modificada manualmente. O histograma também pode ser construído através da definição manual da **quantidade de grupos** em vez da **amplitude**.

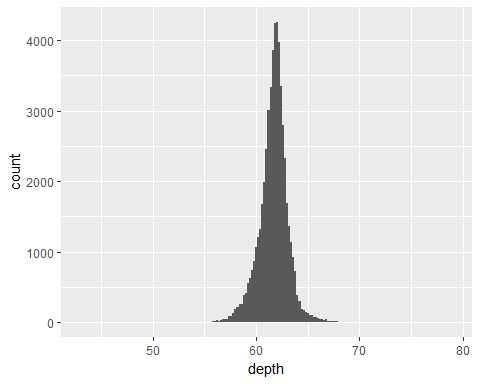
EXEMPLOS

#histograma simples  
ggplot(diamonds, aes(depth)) +  
 geom\_histogram()

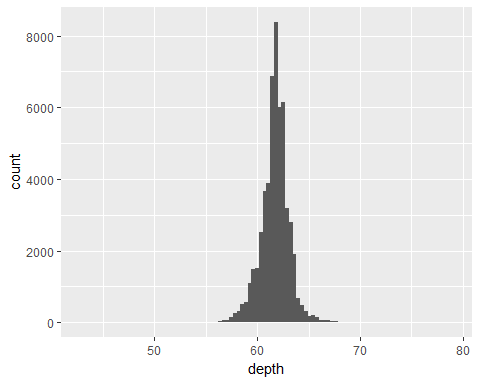
## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



#histograma determinando um largo de banda de 0.2  
ggplot(diamonds, aes(depth)) +  
 geom\_histogram(binwidth = 0.2)



#histograma determinando uma quantidade de 100 grupos  
ggplot(diamonds, aes(depth)) +  
 geom\_histogram(bins = 100)



Observe que dependendo da escolha do tamanho do grupo ou sua quantidade, o histograma muda de apariência, a escolha final é sua!

Os histogramas também podem ser calculados para diversos fatores ou níveis de uma categoria.

No próximo gráfico vamos calcular histogramas para a variável "price", usando os níveis da variável "cut". O procedimento seria: primeiro separar a variável por níveis, depois fazer o histograma para cada nível e finalmente empilhar os histogramas.

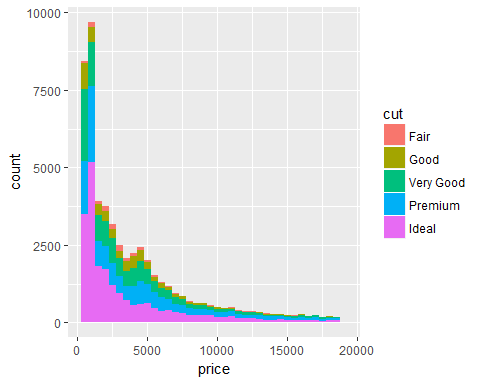
Como essa visualização pode ficar muito poluída, as vezes é melhor fazer a comparação por frequências.

Nos próximos dois gráficos apresentaremos as duas opções.

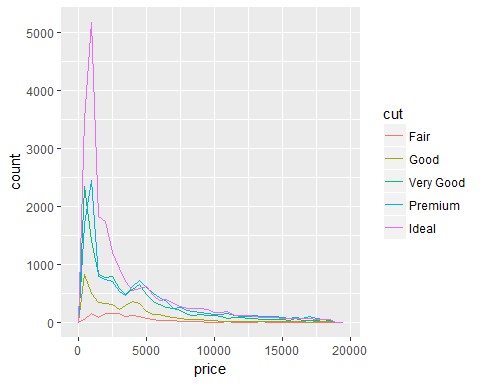
# histograma por factor, empilhados

ggplot(diamonds, aes(price, fill = cut)) +

geom\_histogram(binwidth = 500)



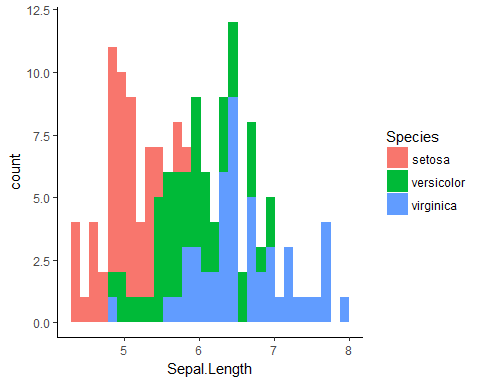
# histograma por frequências  
ggplot(diamonds, aes(price, colour = cut)) +  
 geom\_freqpoly(binwidth = 500)



#### ***EXERCICIO***

# Usando a base de dados "iris", faça o histograma da variável Sepal.Length e dando cor separado por espécies e um tema de fundo tipo classico  
  
ggplot(iris, aes(Sepal.Length, fill = Species)) +  
 geom\_histogram() +  
 theme\_classic()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



#### Desenho de linhas

Para desenhar linhas em **ggplot** usamos **geom\_line()**, que é parte de uma geometria maior que permite conectar observações. Veja **?geom\_path** para maiores detalhes.

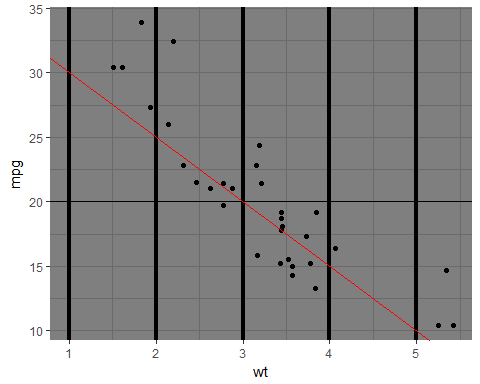
**geom\_line()** conecta observações na ordem em que aparecem no eixo "x".

Para desenhar linhas de referência podemos usar:

1. geom\_hline, para linhas horizontais, definindo o argumento *yintercept = ...*
2. geo\_vline, para linhas verticais, definindo o argumento *xintercept = ...*
3. geom\_abline, para linhas retas em qualquer direção, definindo os argumentos, *slope = ...* e *intercept = ...* para determinar sua posição.

Como exemplo vamos fazer um gráfico com os três tipos de linhas.

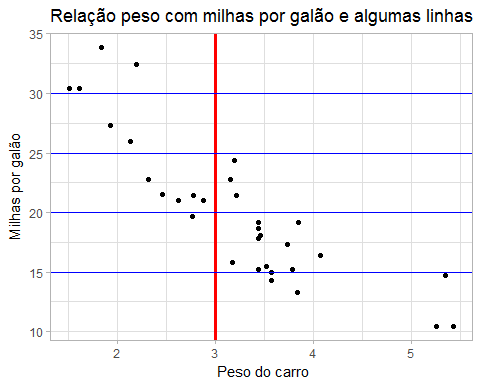
# primeira camada, dados e mapeamento das variáveis  
graf4 <- ggplot(mtcars, aes(wt, mpg)) + geom\_point()  
  
graf4 +   
 geom\_vline(xintercept = 1:5, size = 1.5) + # + camada com várias linhas verticais   
 geom\_hline(yintercept = 20) + # + camada com linha horizontal  
 geom\_abline(slope = -5, intercept = 35, colour = "red") + # + camada com linha diagonal  
 theme\_dark() # + mas tema escuro de fundo



#### ***EXERCICIO***

# Usando "graf4",   
# desenhar linhas horizontais em 30, 25, 20 e 15, de cor azul  
# desenhar uma linha vertical de valor 3, na cor vermelha e largura 1.2  
# adicione legendas nos eixos e titulo principal a sua escolha  
# finalmente um tema light...

graf4 +   
 geom\_vline(xintercept = 3, size = 1.2, col = "red") + # + camada com várias linhas verticais   
 geom\_hline(yintercept = c(15,20,25,30), col = "blue") + # + camada com linha horizontal  
 xlab(" Peso do carro") + # + legenda eixo x  
 ylab("Milhas por galão") + # + legenda eixo y  
 labs(title = "Relação peso com milhas por galão e algumas linhas") + # titulo principal  
 theme\_light() # tema light



#### Gráfico de linhas 2

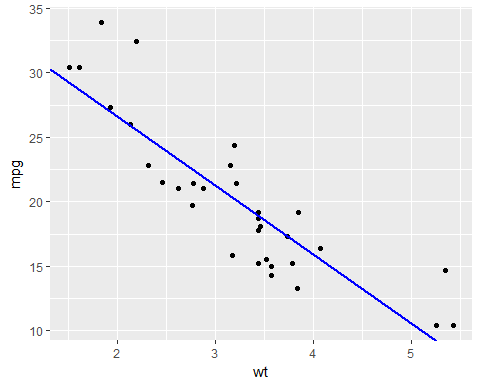
Vamos mostrar como plotar uma linha de regressão (existem diversas formas no R, como sempre!).

A regressão vai ser feita com as variáveis "wt" e "mpg", já conhecidas, reaproveitando o cálculo de graf4, que tem os dados e os pontos num objeto gráfico só.

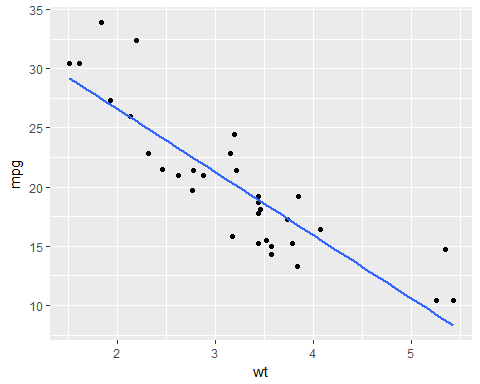
Para isso a calculamos a regressão com a função **lm()** e guardamos os coeficientes, que serão chamados posteriormente dentro da camada **geom\_abline**.

Na sequência fazemos o mesmo procedimento duma forma mais direta com **geom\_smooth()**

# Calcular a regressão e guardar os coeficientes em "coeficientes" (dataframe)  
coeficientes <- coef(lm(mpg ~ wt, data = mtcars))  
  
# adicionar a graf4 uma camada de linha diagonal, usando os argumentos da regressão calculada  
graf4 + geom\_abline(intercept = coeficientes[1], slope = coeficientes[2], col = "blue", size = 0.8)



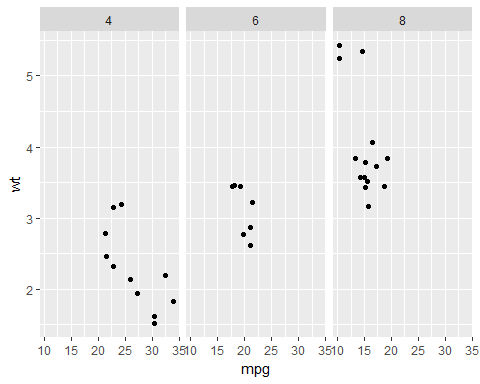
# mais fácil ainda com geom\_smooth e determinando o método lm:  
graf4 + geom\_smooth(method = "lm", se = FALSE)



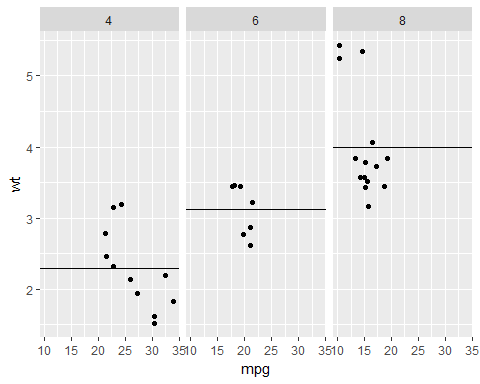
#### ***compare os dois gráficos, porquê são levemente diferentes?***.

No caso de ter que desenhar linhas em diferentes grupos de variáveis, podemos usar **facets**

# vamos salvar um gráfico com três diferentes grupos, separados por número de cilindros  
graf5 <- ggplot(mtcars, aes(mpg, wt)) +  
 geom\_point() +  
 facet\_wrap(~ cyl)  
# usando o conhecimento das ferramentas tidyverse, calculamos a media de peso do carro por número de cilindros  
mean\_wt <- mtcars %>% group\_by(cyl) %>% summarise(wt\_mean = mean(wt))   
  
# finalmente vamos adicionar uma linha de valor médio calculado em cada grupo  
graf6 <- ggplot(mtcars, aes(mpg, wt)) +  
 geom\_point() +  
 facet\_wrap(~ cyl) +   
 geom\_hline(aes(yintercept = mean\_wt[,2]), mean\_wt[,1])  
graf5



graf6

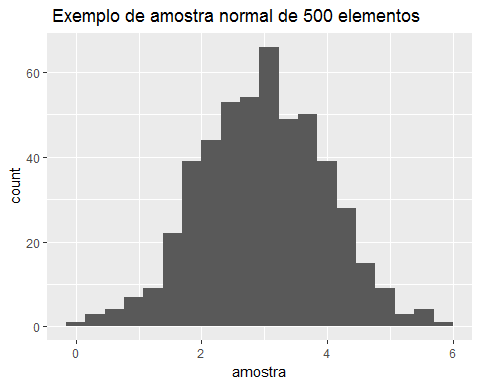


#### Gráfico Quantile-quantile

Um gráfico que pode ser interessante quando analisada a "normalidade" de uma amostra, é o chamado **quantile-quantile** o **QQ**. Ele apresenta os valores dos quantís teóricos e amostrais num gráfico, permitindo visualizar se a amostra apresenta normalidade. Uma amostra normal, apresentaria os pontos numa linha diagonal ascendente, vejamos o seguinte exemplo.

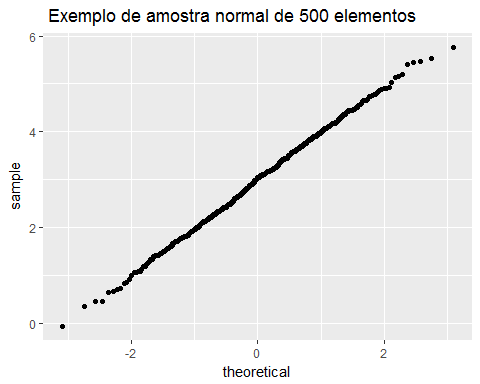
Primeiro criamos a amostra normal e graficamos para conferir.

# Criando uma amostra normal de 500 valores  
amostra <- as.data.frame(rnorm(500, 3,1))  
names(amostra) <- c("Valores")  
  
# graficando para ver a normalidade da amostra  
ggplot(data = amostra, aes(amostra)) +  
 geom\_histogram(bins = 20) +  
 labs(title = " Exemplo de amostra normal de 500 elementos")

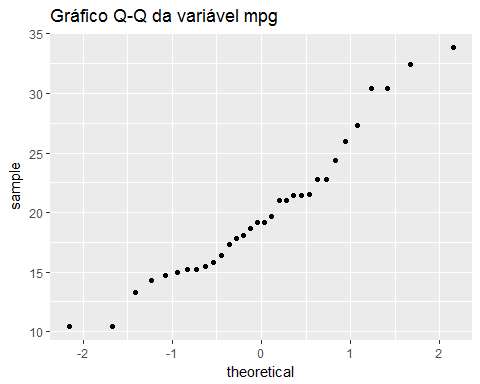


O gráfico QQ da amostra é:

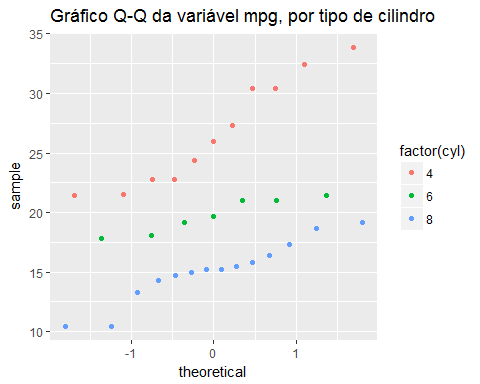
# graficando Q-Q  
ggplot(amostra) +  
 stat\_qq(aes(sample = amostra)) +  
 labs(title = " Exemplo de amostra normal de 500 elementos")



# usando o gráfico para analisar uma variável...  
ggplot(mtcars) +  
 stat\_qq(aes(sample = mpg)) +  
 labs(title = "Gráfico Q-Q da variável mpg")

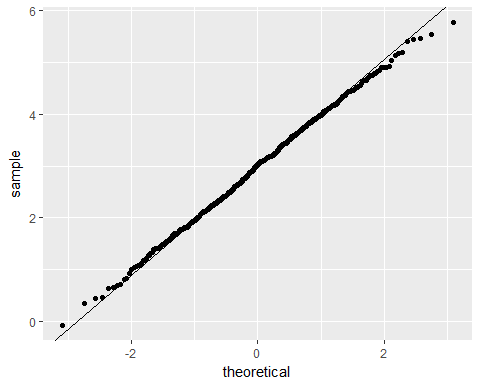


# usando o gráfico para analisar uma variável por fator  
ggplot(mtcars) +  
 stat\_qq(aes(sample = mpg, colour = factor(cyl))) +  
 labs(title = "Gráfico Q-Q da variável mpg, por tipo de cilindro")

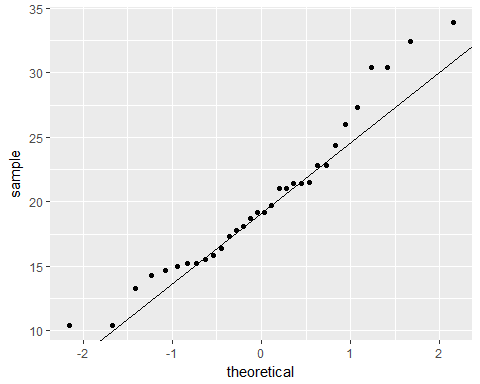


A seguinte função, extraída de <https://stackoverflow.com/questions/4357031/qqnorm-and-qqline-in-ggplot2/>, permite calcular e plotar a linha de regressão da amostra, permitindo assim uma visualização mais detalhada da sua normalidade.

# function extracted from https://stackoverflow.com/questions/4357031/qqnorm-and-qqline-in-ggplot2/  
qqplot.data <- function (vec){ # argumento: vetor de números  
 # as seguintes 4 linhas são da função qqline() do base R  
 y <- quantile(vec[!is.na(vec)], c(0.25, 0.75))  
 x <- qnorm(c(0.25, 0.75))  
 slope <- diff(y)/diff(x)  
 int <- y[1L] - slope \* x[1L]  
 # criação do dataframe  
 d <- data.frame(resids = vec)  
  
 ggplot(d, aes(sample = resids)) +   
 stat\_qq() +   
 geom\_abline(slope = slope, intercept = int)  
}  
  
# gráfico dos valores da amostra normal de 500 elementos   
qqplot.data(amostra$Valores)



# gráfico da variável mpg  
qqplot.data(mtcars$mpg)



Finalmente podemos aplicar esta metodologia para vários fatores de uma variável.

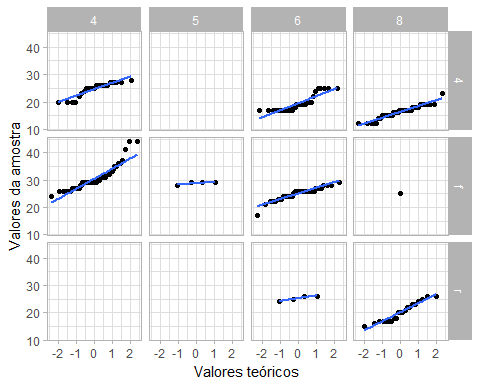
O seguinte exemplo, está baseado numa função disponível na mesma página de **stackoverflow** mencionada acima.

Cria-se uma função para calcular os valores teóricos dos quantís a partir de uma amostra determinada, a diferença é será aplicada sobre uma variável separada em categorias ou fatores.

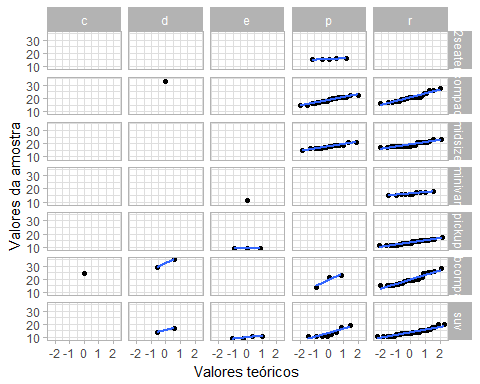
Os dados a serem usados são os da base "ggplot2:mpg", a variável a ser analisada é "hwy" e as variáveis de classificação serão "cyl" e "drv".

Assim sendo, vamos analisar o rendimento dos carros em milhas por galão em estradas, acorde ao número de cilindros e tipo de transmissão de cada carro. Verificaremos se os valores tem uma distribuição normal com gráficos individuais QQ.

# criar a função  
qqnorm\_data <- function(x){  
 Q <- as.data.frame(qqnorm(x, plot = FALSE))  
 names(Q) <- c("xq", substitute(x))  
 Q  
}  
  
# agrupa os dados por categorias e aplica a função  
df5 <- mpg %>%  
 group\_by(drv, cyl) %>%  
 do(with(., qqnorm\_data(hwy)))  
  
# realiza o gráfico QQ  
ggplot(data = df5, aes(x = xq, y = hwy)) +  
 geom\_point() +  
 geom\_smooth(method = "lm", se = FALSE) +  
 xlab("Valores teóricos") +  
 ylab("Valores da amostra") +  
 facet\_grid(drv ~ cyl) +  
 theme\_light()



# Exercício  
# Analisar o rendimento em cidade dos carros, baseado no tipo de carro e combustível usado  
# Solução:  
  
# agrupa os dados por categorias e aplica a função  
df5 <- mpg %>%  
 group\_by(class, fl) %>%  
 do(with(., qqnorm\_data(cty)))  
  
# realiza o gráfico QQ  
ggplot(data = df5, aes(x = xq, y = cty)) +  
 geom\_point() +  
 geom\_smooth(method = "lm", se = FALSE) +  
 xlab("Valores teóricos") +  
 ylab("Valores da amostra") +  
 facet\_grid(class ~ fl) +  
 theme\_light()



FIM DE EXERCÍCIOS 03

## Capítulo 3 - Visualização de dados com *ggplot2()*

## d- modificação de parâmetros estéticos (cont.)

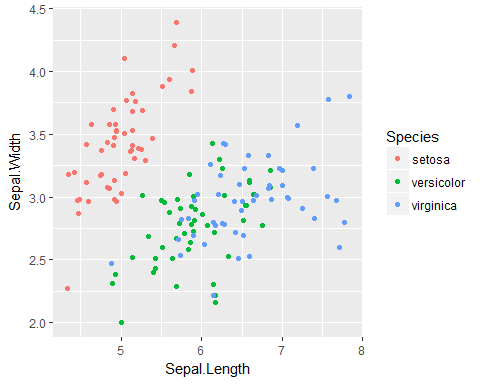
Alguns parâmetros estéticos permitem ser modificados para uma melhor representação.

Referente à posição dos elementos gráficos, o padrão de posicionamento é "identity", ou seja os elementos serão colocados na sua correta posição, mapeada pelas variáveis respectivas.

Porém, já vimos o efeito "jitter" na posição, quando queremos visualizar elementos sobrepostos parcial ou totalmente.

A separação no caso dos pontos pode ser especificamente determinada através da função **position\_jitter**, veja o seguinte exemplo:

posn.j <- position\_jitter(width = 0.2)  
  
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
 geom\_point(position = posn.j)



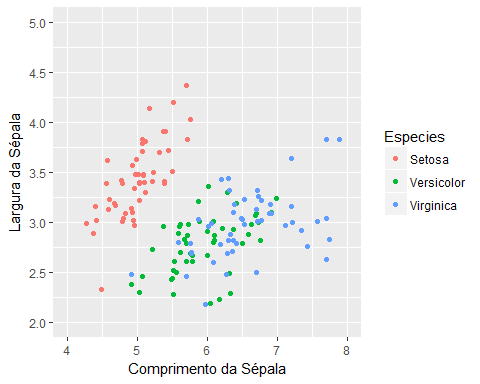
Em relação à posição, com gráficos de barras já vimos que podemos colocarlas lado a lado ou empilhadas, ou seja "dodge" ou "stack", ou ainda mudar a posição dos eixos com "coord\_flip()".

#### Escalas

Existem diversas funções de escala, vamos mostrar duas interessantes e muito usadas, elas são a que determina a apariência das escalas em ambos eixos, "x" e "y" e a que controla a escala de cores, colocando nomes nas cores escolhidas de forma de refletir os fatores usados.

Veja o exemplo:

# as primeiras duas camadas são as básicas de plotagem, com dados e geometria  
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
 geom\_point(position = "jitter") +  
# esta camada controla a apariência do eixo "x", se estabelece o nome do eixo, os limites, assim como o intervalo em que serão colocadas as etiquetas (breaks)  
 scale\_x\_continuous("Comprimento da Sépala",   
 limits = c(4, 8),  
 breaks = seq(4, 8, 1)) +  
# esta camada controla o eixo "y", nome, limites e intervalos   
 scale\_y\_continuous("Largura da Sépala",   
 limits = c(2, 5),  
 breaks = seq(2, 5, 0.5)) +  
# esta camada controla a apariência da escala de cores, de tipo discreta. Coloca nome na legenda e nas categorias   
 scale\_color\_discrete("Especies",labels = c("Setosa", "Versicolor", "Virginica"))



As escalas usadas foram para escala de variáveis contínuas nos eixos e discreta na cor, consulte a ajuda para escalas de variáveis discretas e cor contínua.

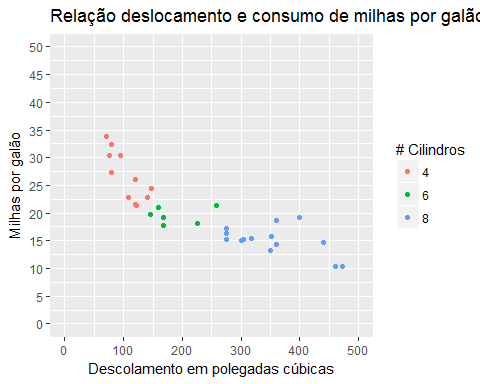
Nos exercícios a seguir, toda instrução que tenha a

#### ***seguinte formatação***

é para sua execução no fragmento de código R que segue ou na consola do RStudio (janela 3).

#### ***Com a base de dados mtcars, faça um gráfico das variáveis "disp" e "mpg", use a variável "cyl" para separar por fatores e dê cor em cada um. Adicione título e legendas nos eixos. As escalas dos eixos devem ser: 0-500 e 0-40 respectivamente, escolha o intervalo que crea adequado,***

# Solução:  
# primeira camada, dados e mapeamento de variáveis, cor pelo cilindro  
ggplot(mtcars, aes(x=disp, y = mpg, col = factor(cyl))) +  
  
# segunda camada, pontos  
 geom\_point() +  
  
# terceira camada, título  
 labs(title = "Relação deslocamento e consumo de milhas por galão acorde # de cilindros") +  
  
# quarta camada, escala no eixo "x", contínua, define os limites e os intervalos  
 scale\_x\_continuous("Descolamento em polegadas cúbicas",   
 limits = c(0,500),   
 breaks = seq(0,500,100)) +  
  
# quinta camada, escala no eixo "y", contínua, define os limites e os intervalos  
 scale\_y\_continuous("Milhas por galão",   
 limits = c(0,50),   
 breaks = seq(0,50,5)) +  
# sexta camada, escala de cor discreta com nome  
 scale\_color\_discrete("# Cilindros")



Os parâmetros estéticos podem ser colocados de duas forma:

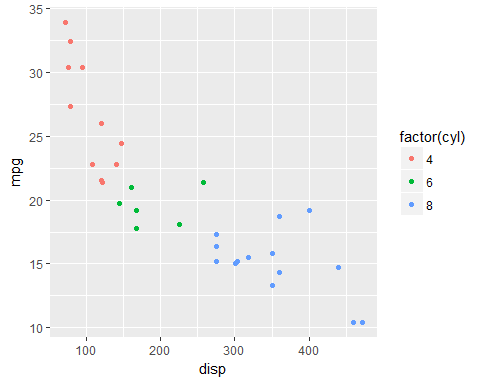
* inicialmente na primeira camada, junto à definição dos dados e o mapeamento de variáveis, ou
* em cada camada individualmente.

No primeiro caso, os parâmetros são definidos como globais e vão determinar o aspecto de toda camada que não tenha definido esses parâmetros na sua definição, ou seja **elas vão herdar os atributos globais**.

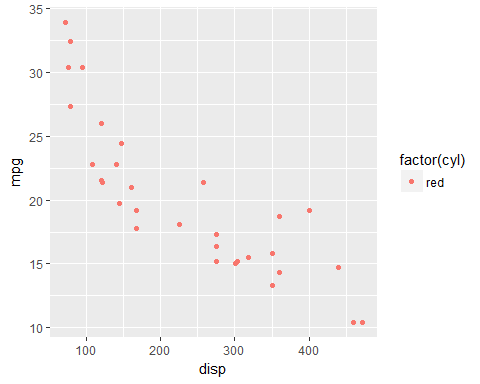
No segundo caso, **cada parâmetro definido na camada vai ter prioridade** sobre os valores que estejam definidos como globais.

Exemplo:

# primeiro gráfico, a cor está definida globalmente acorde factor(cyl)  
ggplot(mtcars, aes(x=disp, y = mpg, col = factor(cyl))) +  
 geom\_point()



# segundo gráfico, a cor está definida na camada da geometria e tem prioridade...  
ggplot(mtcars, aes(x=disp, y = mpg, col = factor(cyl))) +  
 geom\_point(aes(col = "red"))



Cada geometria pode ter seus parâmetros estéticos próprios, independiente das definições globais, e cada geometria tem seus parâmetros específicos.

No caso da geometria de pontos, o esencial é "x" e "y", que pode estar definido globalmente, mas tem as opções de: **alpha, colour, fill, shape e size**.

**Assim sendo temos a possibilidade controlar o aspecto de cada camada independentemente das outras.**

Vejamos o seguinte exemplo. Nele vamos graficar os pontos e as médias das três especies da flor "iris".

Para isso vamos calcular primeiro a média por espécie e guardar numa variável **"iris.resumo"**. Para isso veja a função ?aggregate, que divide um dataframe em subconjuntos e para cada um deles, aplica uma função.

No nosso caso, vamos usar o dataframe "iris", columnas 1 a 4, agrupadas pela coluna 5, "Species" e para cada grupo calculamos a média:

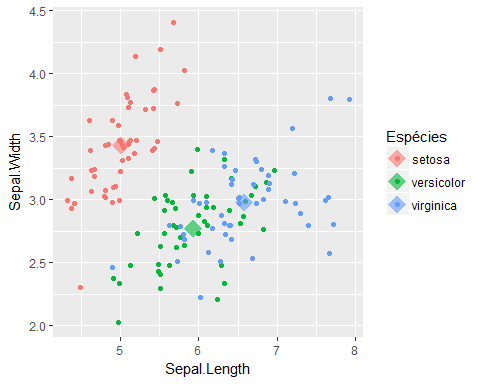
iris.resumo <- aggregate(iris[1:4], by = list(iris$Species), mean)  
  
# para evitar conflitos de nomes na determinação das cores, colocamos o nome  
names(iris.resumo)[1] <- "Species"  
  
iris.resumo

## Species Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 setosa 5.006 3.428 1.462 0.246  
## 2 versicolor 5.936 2.770 4.260 1.326  
## 3 virginica 6.588 2.974 5.552 2.026

Na sequência vamos graficar os pontos, evitando a sobreposição e colorindo por espécie e adicionar depois uma terceira camada com as médias de cada espécie, com simbolo e tamanho diferente, mas cor idêntica e transparência de 0.6.

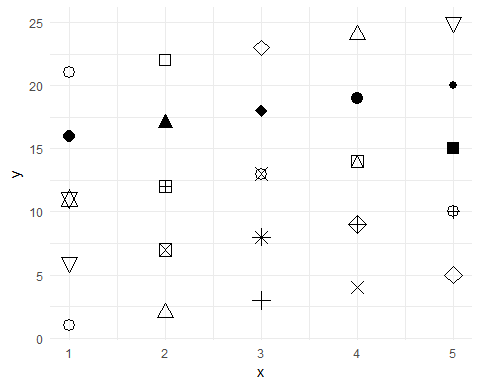
A quarta camada vai dar o nome na legenda, como já feito em exemplos anteriores.

# primeira camada com a definição dos dados e cor  
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +  
  
#segunda camada, apenas pontos, estética determinada globalmente  
 geom\_jitter() +  
  
# terceira camada, definimos os dados, tamanho e etiqueta  
 geom\_point(data = iris.resumo, shape = 18, size = 6, alpha = 0.6)+  
  
# quarta camada com o nome da legenda  
 scale\_color\_discrete("Espécies")

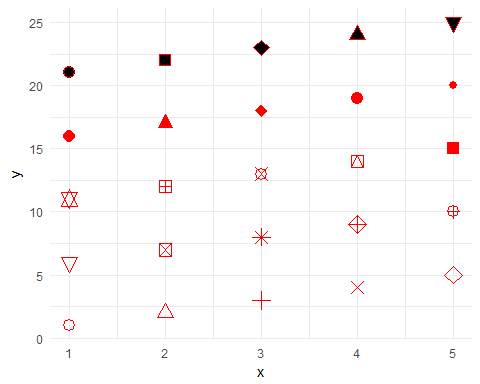


Na escolha das formas, temos 25 diferentes, sendo as últimas 5 com preenchimento de cor e borda diferenciado, veja ?aes\_linetype\_size\_shape.

# gráfico das 25 formas diferentes  
df2 <- data.frame(x = 1:5 , y = 1:25, z = 1:25)  
s <- ggplot(df2, aes(x, y))  
  
# primeiro grafico todos com cor preto e tamanho 4  
s + geom\_point(aes(shape = z), size = 4) +  
 scale\_shape\_identity() +  
 theme\_minimal()



# simbolos 20-25 tem também borda e preenchimento de cor  
s + geom\_point(aes(shape = z), size = 4, colour = "Red", fill = "Black") +  
 scale\_shape\_identity() +  
 theme\_minimal()



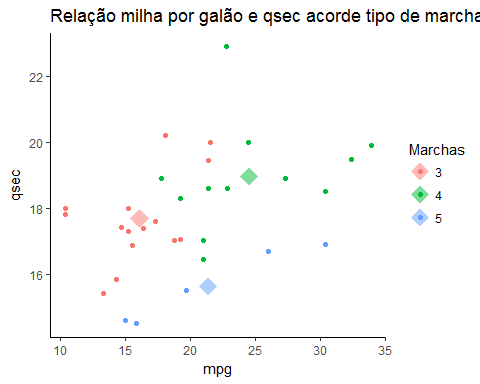
#### ***Com a base de dados "mtcars"", calcule primeiro as médias dos valores de "mpg" e "qsec", depois faça um gráfico das variáveis "mpg" e "qsec", usando a variável "gear" e dê cor em cada um.***

#### ***Adicione uma camada com as médias calculadas, com forma 24, de borda vermelha e preenchimento preto, use alpha =0.5. Adicione título e tema a sua escolha.***

# Solução:  
# cálculo das médias  
mtcars.resumo <- aggregate(mtcars[,c(1,7)], by = list(mtcars$gear), mean)  
  
# para evitar conflitos de nomes na determinação das cores, colocamos o nome "gear"  
names(mtcars.resumo)[1] <- "gear"  
# olhe o resumo  
mtcars.resumo

## gear mpg qsec  
## 1 3 16.10667 17.692  
## 2 4 24.53333 18.965  
## 3 5 21.38000 15.640

# gráfico  
# primeira camada com a definição dos dados e cor  
ggplot(mtcars, aes(x = mpg, y = qsec, col = factor(gear)) ) +  
  
#segunda camada, apenas pontos com separação jitter, estética determinada globalmente  
 geom\_jitter() +  
  
# terceira camada, definimos os dados, tamanho e etiqueta  
 geom\_point(data = mtcars.resumo,   
 shape = 18,   
 size = 6,   
 alpha = 0.5) +  
  
# quarta camada com o nome da legenda  
 scale\_color\_discrete("Marchas") +  
   
# titulos  
 labs(title = "Relação milha por galão e qsec acorde tipo de marchas")+  
 theme\_classic()



### Gráficos de séries temporais

Existem pacotes específicos para tratar e visualizar séries temporais. O próprio lines() do R base pode ser usado, assim como o pacote **stats** permite transformar dados em série temporal (**ts**) e graficar em forma fácil (**ts.plt**). O uso de **plot()** + **lines()** também é possível, com o cuidado de ver a variação de escala quando adicionamos mais de uma linha.

Vamos mostrar um exemplo de gráfico temporal. Criamos o dataframe ou importamos os dados e logo fazemos um gráfico temporal. Neste caso específico a variável que representa os dados não está definida como tipo "data", mas como fator, isso porque os dados contém apenas valores de médias sem especificar outro tipo de data.

Atentar como se cria uma determinada ordem para graficar depois os dados, usando "levels = …"

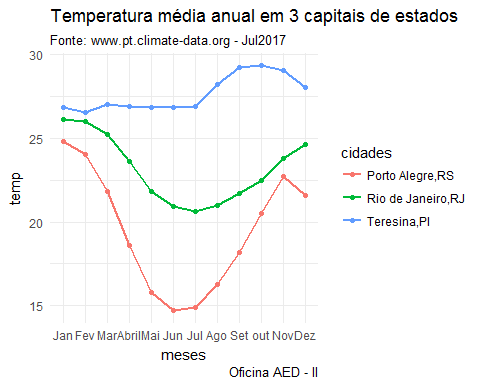
# crio um vetor com o nome dos meses  
nome.meses <- c("Jan", "Fev", "Mar", "Abril","Mai", "Jun", "Jul", "Ago", "Set", "out", "Nov", "Dez")  
  
# crio o dataframe, com os meses, as cidades e as temperaturas  
media.temp <- data.frame(meses = factor((nome.meses), levels = nome.meses),  
 cidades = rep(c("Porto Alegre,RS", "Rio de Janeiro,RJ", "Teresina,PI"), each = 12),  
 temp = c(24.8, 24.0, 21.8, 18.6, 15.8, 14.7, 14.9, 16.3, 18.2, 20.5, 22.7, 21.6,  
 26.1, 26.0, 25.2, 23.6, 21.8, 20.9, 20.6, 21.0, 21.7, 22.5, 23.8, 24.6,  
 26.8, 26.5, 27.0, 26.9, 26.8, 26.8, 26.9, 28.2, 29.2, 29.3, 29.0, 28.0))

head(media.temp)

## meses cidades temp  
## 1 Jan Porto Alegre,RS 24.8  
## 2 Fev Porto Alegre,RS 24.0  
## 3 Mar Porto Alegre,RS 21.8  
## 4 Abril Porto Alegre,RS 18.6  
## 5 Mai Porto Alegre,RS 15.8  
## 6 Jun Porto Alegre,RS 14.7

# gráfico do dataframe, primeiro os dados e mapeamento de variáveis  
ggplot(media.temp, aes(x = meses, y = temp,

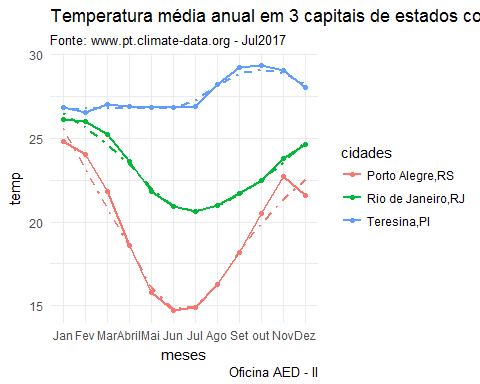
group = cidades, color = cidades)) +  
# adiciona pontos  
 geom\_point(size = 1.3) +   
# adiciona linha entre pontos  
 geom\_line(size = 1) +  
# adiciona titulos e tema  
 labs(title = "Temperatura média anual em 3 capitais de estados",  
 subtitle = "Fonte: www.pt.climate-data.org - Jul2017",  
 caption = "Oficina AED - II") +  
 theme\_minimal()



#### ***Adicione no fragmento de código anterior uma linha suavizada sem intervalo de confiança. Especifique o tipo da linha diferente da linha anterior (linetype). Experimente colocar o intervalo de confiança e diferentes temas. Mude o título para refletir a mudança do gráfico.***

# Solução:  
# gráfico do dataframe, primeiro os dados e mapeamento de variáveis  
ggplot(media.temp, aes(x = meses, y = temp, group = cidades, color = cidades)) +  
  
# adiciona pontos  
 geom\_point(size = 1.3) +   
  
# adiciona linha entre pontos, tamanha 1 e tipo 4  
 geom\_line(size = 1) +  
  
# adiciona linha suave sem intervalo de confiança  
 geom\_smooth(se = F, linetype = 4) +  
  
# adiciona titulos e tema  
 labs(title = "Temperatura média anual em 3 capitais de estados com linha de regressão",  
 subtitle = "Fonte: www.pt.climate-data.org - Jul2017",  
 caption = "Oficina AED - II") +  
 theme\_minimal()

## `geom\_smooth()` using method = 'loess'



As séries temporais podem vir em dois tipos diferentes, como dado tabular, geralmente importado como dataframe, ou como objeto R **"Time-Series"**.

Como dataframe, uma variável deve indicar o tempo de cada registro, seu formato pode ser de apenas ano, ano-mes-dia ou combinações incluindo horas, mas o importante e que primeiro devemos definir a variável que representa as datas como de tipo "Date". Isso pode ser feito com diversos pacotes, incluindo "as.Date" ou "lubridate::as\_Date" (veja a documentação da AED – I). Feito ou verificado isso, **ggplot2** reconhece a variável como data e trata como tal.

# grafico de série temporal  
head(economics)

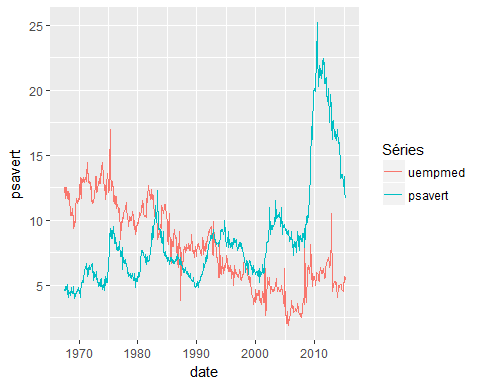
## # A tibble: 6 x 6  
## date pce pop psavert uempmed unemploy  
## <date> <dbl> <int> <dbl> <dbl> <int>  
## 1 1967-07-01 507.4 198712 12.5 4.5 2944  
## 2 1967-08-01 510.5 198911 12.5 4.7 2945  
## 3 1967-09-01 516.3 199113 11.7 4.6 2958  
## 4 1967-10-01 512.9 199311 12.5 4.9 3143  
## 5 1967-11-01 518.1 199498 12.5 4.7 3066  
## 6 1967-12-01 525.8 199657 12.1 4.8 3018

# uma e duas variáveis   
ggplot(economics, aes(date, psavert)) +

geom\_line(aes(col = "blue")) +

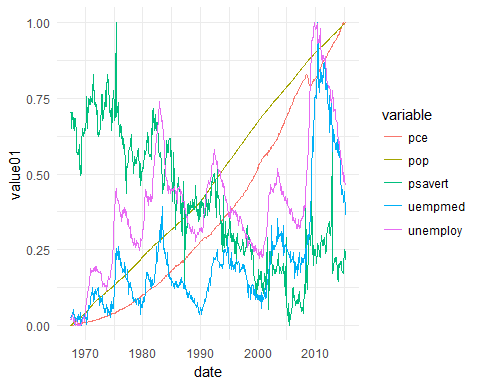
geom\_line(aes(date, uempmed, col = "red")) +

# quarta camada com o nome da legenda  
 scale\_color\_discrete("Séries",   
 labels = c("uempmed", "psavert"))



# diversas variáveis, note a compatibilização das variáveis para ter o eixo comumy entre 0 e 1

ggplot(economics\_long, aes(date, value01, colour = variable)) +  
 geom\_line() +  
 theme\_minimal()



No caso de objetos R do tipo **"Time-Series"**, para ser visualizada corretamente com ggplot2, ela deve ser convertida em dataframe. Temos diversas maneiras de fazer a operação:

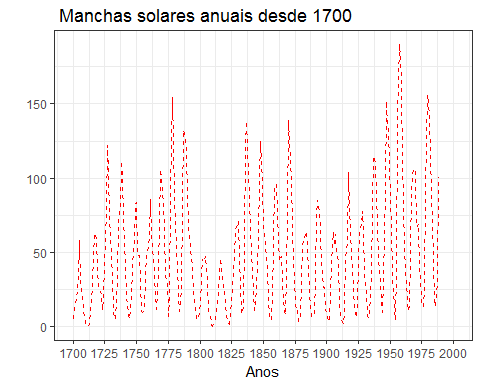
1- forçando a variável com **"as.data.frame.ts()"**;

2- usando o pacote **ggfortity**; ou

3- usando o novo pacote **broom::ts**.

Feita a transformação, a função **autoplot** do própio ggplot faz a visualização

# usando ggfortify  
library(ggfortify)  
autoplot(sunspot.year, ts.colour = 'red', ts.linetype = 'dashed') +  
 labs(title = " Manchas solares anuais desde 1700") +  
 scale\_x\_continuous("Anos",   
 limits = c(1700, 2000),  
 breaks = seq(1700, 2000, 25)) +  
 theme\_bw()



#### ***No seguinte fragmento do código, personaliza os eixos "x" e "y", dando nome, limites e intervalos a sua escolha***

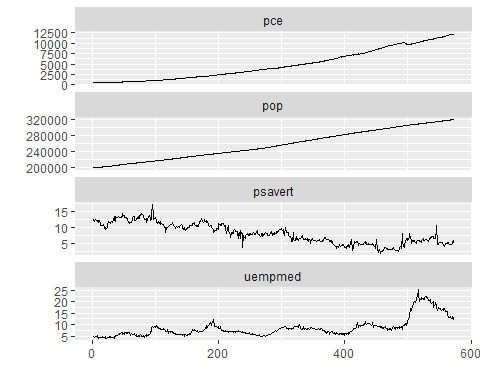
# usando broom  
library(broom)  
autoplot(sunspot.year, ts.colour = 'red', ts.linetype = 'dashed') +  
 labs(title = " Manchas solares anuais desde 1700") +  
  
# esta camada controla o eixo "y", nome, limites e intervalos  
 scale\_x\_continuous("Anos",   
 limits = c(1700, 2000),  
 breaks = seq(1700, 2000, 25)) +  
  
# esta camada controla o eixo "y", nome, limites e intervalos   
 scale\_y\_continuous("Número de manchas",   
 limits = c(0, 200),  
 breaks = seq(0, 200, 25)) +  
  
 theme\_bw()

## Scale for 'y' is already present. Adding another scale for 'y', which  
## will replace the existing scale.

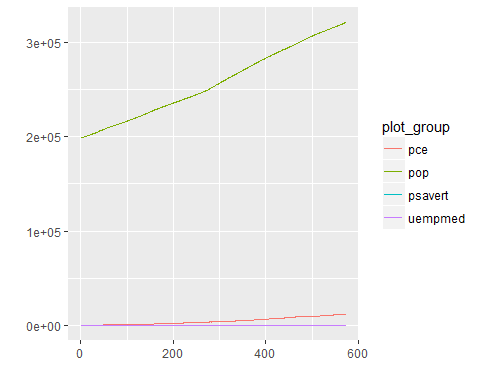
## 

Autoplot sirve também para graficar diversas séries temporais de uma mesma época, através da função **facets = …**

library(ggfortify)  
# a série que tem formato de dataframe é transformada em objeto R "Time-Series"  
# gráfico separado por variável  
eco <- ts(economics)  
# facets  
autoplot(eco[,2:5], facets = T)



# eixo unificado  
autoplot(eco[,2:5], facets = F)



#### DICA

No gráfico da série temporal para diversas variáveis, a amplitude apresenta uma grande variação, fazendo que um gráfico com elas num eixo único, fique confuso, como ocorre nesse último gráfico com facets.

Para colocar todas as variáveis desejadas num intervalo pre-determinado, usamos uma fórmula aplicada em forma individual para cada uma delas.

Fórmula de re-escala em intervalos definidos: , a fórmula seria:

O procedimento seria criar uma função baseada na fórmula apresentada e aplicar a mesma na(s) coluna(s) desejadas, através da função **apply** ou alguma das sus variações.

No caso específico de *normalizar* uma variável, ou seja, deixar sua média com valor 0 e seu desvio padrão em 1, dispomos da função *scale*, que realize o cálculo diretamente.

FIM DE EXERCICÍOS 4 E CURSO DE AED – PARTE II