Pré-processamento de dados

Tarefas de pré-processamento;
implementação em R

Preparação dos dados

Para chegar à estrutura desejável dos dados, dada anteriormente, podem ser necessárias diversas etapas a partir dos dados reais, “em bruto”

Primeiro passo deve passar por **selecionar** os dados desejáveis e **carregá-los** para um formato de trabalho, podendo haver necessidade de selecionar dados, juntar dados de várias fontes e formatos, etc.

Esta fase não será coberta neste curso dada a sua heterogeneidade e dependência dos objetivos (cada caso é um caso)

Assumindo que após essa fase, temos os dados num formato matricial (tipo *data frame*), ainda haverão tipicamente várias tarefas a realizar para termos os dados prontos para a análise de dados, que veremos em seguida

Discretização de variáveis numéricas

Uma das operações potencialmente úteis de transformação de variáveis é a **discretização** de variáveis numéricas

Implica dividir o intervalo de valores possíveis em sub-intervalos e considerar cada um deles como uma categoria

Pode permitir usar métodos de análise que não estariam disponíveis; pode simplificar a análise

Função R: **cut** – transforma vetores numéricos em fatores

- Argumentos obrigatórios: *x* - vetor numérico com dados, *breaks* – vetor com pontos de corte ou um nº único indicando nº de intervalos (neste caso assumem-se pontos de corte equidistantes)
- Argumentos opcionais: *labels* – nomes para as categorias criadas

Discretização - exemplo

```
> range(eData$Magnitude)
[1] 1.0 5.4
> md = cut(eData$Magnitude, c(0,1.5,2.5,4,7), labels =
c("VL","L","M","H"))
> levels(md)
[1] "VL" "L"  "M"  "H"
> table(md)
 VL   L   M   H
138 175  43   4
> md[1:10]
 [1] L  VL L  L  VL L  L  VL M  VL
Levels: VL L M H
> eData$MagnitudeFactor = md
> class(eData$MagnitudeFactor)
[1] "factor"
```

Valores omissos

Função ***is.na*** permite verificar os valores em falta (NA) num dado vetor / matriz / campo do data frame (note que comparar um valor com NA, fazendo *val == NA* irá ser sempre falso)

Algumas funções (como a função ***table, mean, ...***) permitem definir argumentos para tratar os NAs

Na leitura de dados é possível indicar uma string que se assume como NA (por omissão "NA")

Existem outros casos de valores erróneos, por exemplo **NaN** (*not a number*) que resulta por ex. de um cálculo indeterminado (e.g. 0/0); estes valores podem ser identificados com função ***is.nan***

Função ***complete.cases*** permite identificar quais as linhas de um data frame têm valores corretos (não NA nem NaN)

Valores omissos- exemplos

```
> cfseal = read.csv("dataset-cfseal.csv")
> names(cfseal)
 [1] "age"        "weight"     "heart"      "lung"       …
> cfseal$lung
 [1]  605.0  436.0  380.0  493.9     NA  550.0  470.0  …
> sum(is.na(cfseal$lung))
[1] 6
> cfseal$lung[cfseal$lung>2000]
[1]   NA   NA   NA   NA 2735   NA 2380   NA
> cfseal$lung[cfseal$lung>2000 & !is.na(cfseal$lung)]
[1] 2735 2380
> mean(cfseal$lung)
[1] NA
> mean(cfseal$lung, na.rm=T)
[1] 1136.846
```

```
> table(c(0,1,2,3,NA,3,3,2,2,3))
0 1 2 3
1 1 3 4
> table(c(0,1,2,3,NA,3,3,2,2,3),
useNA="ifany")
   0    1    2    3 <NA>
   1    1    3    4    1
```

Tratamento de valores omissos

Existem várias formas de tratar valores omissos que podem ser mais ou menos adequadas dependendo da análise de dados a realizar:

- Podem ser mantidos no data frame tendo em atenção a sua presença na aplicação das funções
- Podem ser ignorados, removendo linhas e/ ou colunas com valores omissos (por exemplo com função ***na.exclude***)
- Podem ser substituídos por outros valores de várias formas

  Por um valor constante, escolhido dependendo do domínio

  Por um valor constante por coluna (ou linha)

  Por métodos mais elaborados, usando-se informação de outras linhas (e.g. método dos k vizinhos mais próximos)

Tratamento valores omissos- exemplos

```
> sum(is.na(cfseal$lung))
[1] 6
> mean(cfseal$lung)
[1] NA
> m = mean(cfseal$lung, na.rm=T)
> m
[1] 1136.846
> cfseal$lung[is.na(cfseal$lung)] = m
> sum(is.na(cfseal$lung))
[1] 0
> mean(cfseal$lung)
[1] 1136.846
```

```
> cfn = na.exclude(cfseal)
> sum(is.na(cfn$lung))
[1] 0
```

```
> for (i in 1:ncol(cfseal)) {
  m = mean(cfseal[,i], na.rm = TRUE)
  cfseal[is.na(cfseal[,i]),i] = m
}
> sapply(cfseal,
      function(x) sum(is.na(x)))
```

```
> data(airquality)
> complete.cases(airquality)
  [1]  TRUE  TRUE  TRUE  TRUE …
> dim(airquality)
[1] 153   6
> aqn = airquality[complete.cases(airquality),]
> dim(aqn)
[1] 111   6
> sapply(airquality, function(x) sum(is.na(x)))
  Ozone Solar.R    Wind    Temp   Month     Day
     37       7       0       0       0       0
> sapply(aqn, function(x) sum(is.na(x)))
```

Standardização dos dados

Em muitos casos, para ser possível comparar variáveis é necessário proceder à standardização dos dados para que estes estejam dentro de valores comparáveis

O processo mais comum de normalizar valores passa pela sua subtração pela média e divisão pelo desvio padrão; ao valor obtido é também dado o nome de ***z-score***

Um processo alternativo usa a mediana e o desvio absoluto médio

A função **scale** em R: permite realizar este processo.

Argumentos: dados a normalizar; *center* – indica se se subtrai pela média (por omissão: *T*); *scale* – indica se divide pelo sd (por omissão – *T*)

Standardização: exemplo

```
> mean(iris$Sepal.Length)
[1] 5.843333
> sd(iris$Sepal.Length)
[1] 0.8280661
> iris.st = as.data.frame(scale(iris[,1:4]))
> mean(iris.st$Sepal.Length)
[1] -4.484318e-16
> sd(iris.st$Sepal.Length)
[1] 1
```

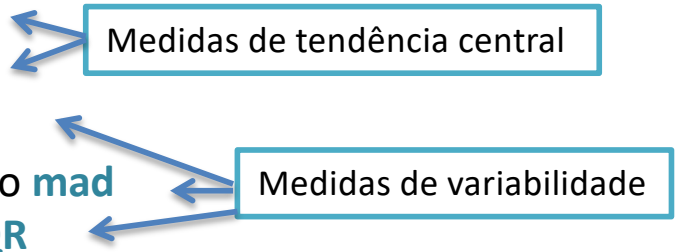
Análise exploratória de dados: sumariação de dados; gráficos

Sumariar os dados

- Dados de grandes dimensões impossíveis de ver em detalhe
- Procuram-se caracterizar os dados através de métricas globais
- Um dos objectivos é procurar problemas nos dados que requeiram ações de pré-processamento
  - Valores atípicos; valores fora de intervalos previstos
  - Problemas com unidades
- Tarefas
  - Verificar **distribuição dos valores** em cada campo: numéricos e discretos (medidas de tendência central, variabilidade, ...)
  - Aplicar **sumários** de dados a linhas e colunas

Sumariar os dados: medidas de estatística descritiva

- **Variáveis numéricas**
  - Maior valor, menor valor e intervalo – funções **min**, **max**, **range**
  - Valor médio – função **mean**
  - Mediana – função **median**
  - Desvio padrão – função **sd**
  - Desvio absoluto médio – função **mad**
  - *Interquartile range* – função **IQR**
  - Funções que resumem várias métricas – **summary**, **quantiles**
  - Sumários de linhas e colunas (apenas para valores numéricos) – **rowSums, rowMeans, colSums, colMeans**



- **Variáveis nominais**
  - Frequências de cada valor – função **table**
  - Valores únicos – função **unique**

Funções de sumarização em R: variáveis numéricas

```
> mean(eData$Magnitude)
[1] 1.860278
> median(eData$Magnitude)
[1] 1.8
> sd(eData$Magnitude)
[1] 0.6674735
> mad(eData$Magnitude)
[1] 0.59304
> quantile(eData$Magnitude)
  0%  25%  50%  75% 100%
 1.0  1.4  1.8  2.2  5.4
> summary(eData$Magnitude)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.40    1.80    1.86    2.20    5.40
> summary(cfseal$lung, na.rm=T)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  380.0   601.9  1055.0  1137.0  1483.0  2735.0       6
```

Funções de sumarização em R: variáveis numéricas

```
> summary(iris)
 …
```

```
> rowMeans(iris[,-5])
  [1] 2.550 2.375 2.350 2.350 …
> colMeans(iris[,-5])
Sepal.Length  Sepal.Width Petal.Length
Petal.Width
5.843333     3.057333     3.758000
1.199333
> colSums(iris[-5])
Sepal.Length  Sepal.Width Petal.Length
Petal.Width
876.5        458.6        563.7       179.9
```

```
> mean(iris$Petal.Length)
[1] 3.758
> sapply(iris[,1:4], IQR)
Sepal.Length  Sepal.Width Petal.Length Petal.Width
1.3          0.5          3.5          1.5
> sapply(iris[,1:4], sd)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
0.8280661    0.4358663    1.7652982    0.7622377
> quantile(iris.st[,1])
 0%  25%  50%  75% 100%
4.3  5.1  5.8  6.4  7.9
```

Funções de sumarização em R: variáveis nominais

O comando table pode ser usado para visualizar tabelas de contingência com mais de 2 variáveis

```
> unique(eData$Src)
[1] nc hv pr uw at
Levels: at hv nc pr uw
> table(eData$Src)
 at  hv  nc  pr  uw
  4  39 227  39  51
```

```
> table(eData$Src,eData$Version)
        0   1   2   3   4   5   6   A
  at    0   4   0   0   0   0   0   0
  hv    0   6  12  11   4   6   0   0
  nc  117  95   8   2   1   2   1   1
  pr   39   0   0   0   0   0   0   0
  uw    0  42   7   2   0   0   0   0
```

Exemplo de distribuição de pares de valores para duas variáveis

```
> table(iris$Species)
setosa versicolor  virginica
50         50         50
> unique(iris$Species)
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
```

Outliers

Outliers são valores que parecem fora dos valores “normais”

Podem significar erros nos dados ou simplesmente valores atípicos mas reais

Quando estes valores são erróneos eles devem ser removidos

Se forem valores reais, ainda assim, podem afectar em demasia algumas medidas de tendência central ou variabilidade (e.g. média e desvio padrão)

Algumas medidas são robustas e este problema, como a média e o desvio absoluto média

Existem também versões “robustas” das funções *mean* e *sd*, usando o parâmetro *trim* que ignora uma proporção pré-definida dos dados

Visualização dos dados: gráficos

Razões para usar gráficos para visualizar dados:

- Explorar os dados
  - Perceber a sua estrutura global
  - Perceber a distribuição dos dados
  - Perceber relações entre variáveis
  - Descobrir padrões
  - Sugerir estratégias de análise
  - Descobrir problemas nos dados e na análise
- Mostrar/ explicar resultados da análise de dados

Nesta fase, concentrar-nos-emos na primeira tarefa (exploração)
A outra irá aparecendo à medida que se estudarem métodos de análise

Packages do R:
- Package standard (incluído distribuição base): ***graphics*** – usado nos exemplos
- Outros packages alternativos: ***lattice, ggplot2***

Boxplots

Servem para visualizar a distribuição de valores de uma variável numérica mostrando algumas medidas de estatística descritiva (média/ mediana, quartis); permitem ver a simetria da distribuição

Podem ajudar a detectar problemas nos dados como outliers

São úteis para comparar distribuições de valores em situações distintas. Podem, por exemplo, ilustrar distribuição de uma variável dependente de um factor, ajudando a perceber a sua relação

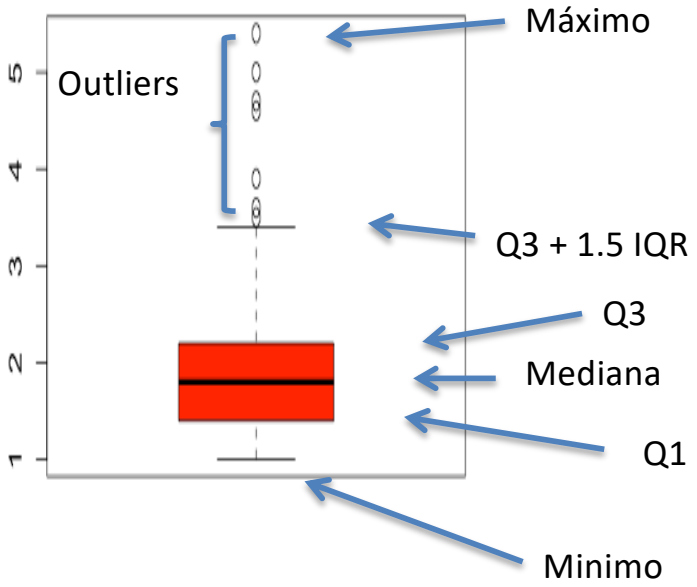
Implementados em R por diversas funções, a mais simples sendo a função ***boxplot***

- Argumentos obrigatórios: variável(is) a representar ou fórmula (pode incluir várias variáveis)
- Opcionais: *horizontal* (lógico, define se boxplot é horizontal; por omissão F; *width* – vetor com larguras das barras, etc.; *range* – valor que multiplicado pelo IQR dá o tamanho dos bigodes (omissão: 1.5), *na.action* (o que fazer com NAs, omissão: ignorar)

Função alternativa: ***bwplot*** (package lattice)

Boxplot & outliers - exemplo

```
> boxplot(eData$Magnitude, col="red")
> range(eData$Magnitude)
[1] 1.0 5.4
> mean(eData$Magnitude)
[1] 1.860278
> mean(eData$Magnitude, trim = 0.1)
[1] 1.785764
> median(eData$Magnitude)
[1] 1.8
> quantile(eData$Magnitude, c(0.25,
0.75))
25% 75%
1.4 2.2
> quantile(eData$Magnitude, 0.75)+
IQR(eData$Magnitude)*1.5
3.4
> quantile(eData$Magnitude, 0.25)
-IQR(eData$Magnitude)*1.5
25%
0.2
```

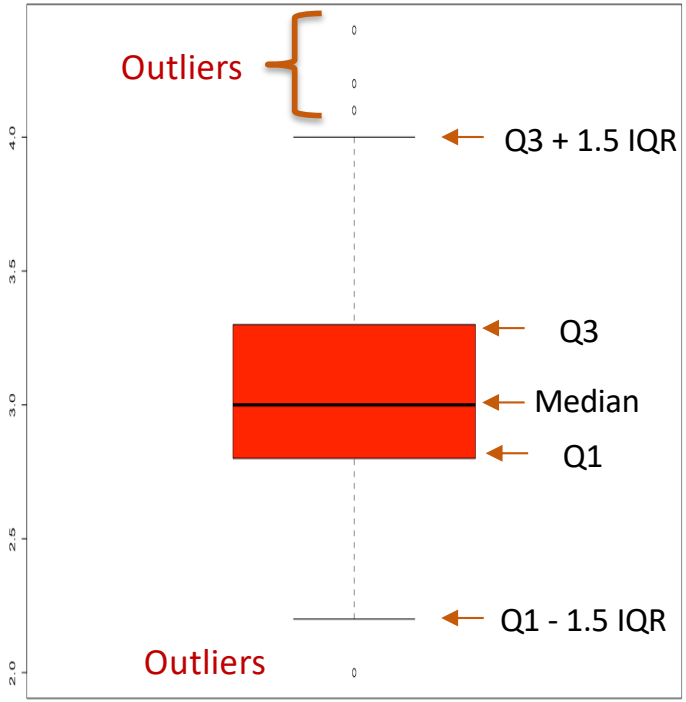


Neste caso, valores atípicos são reais

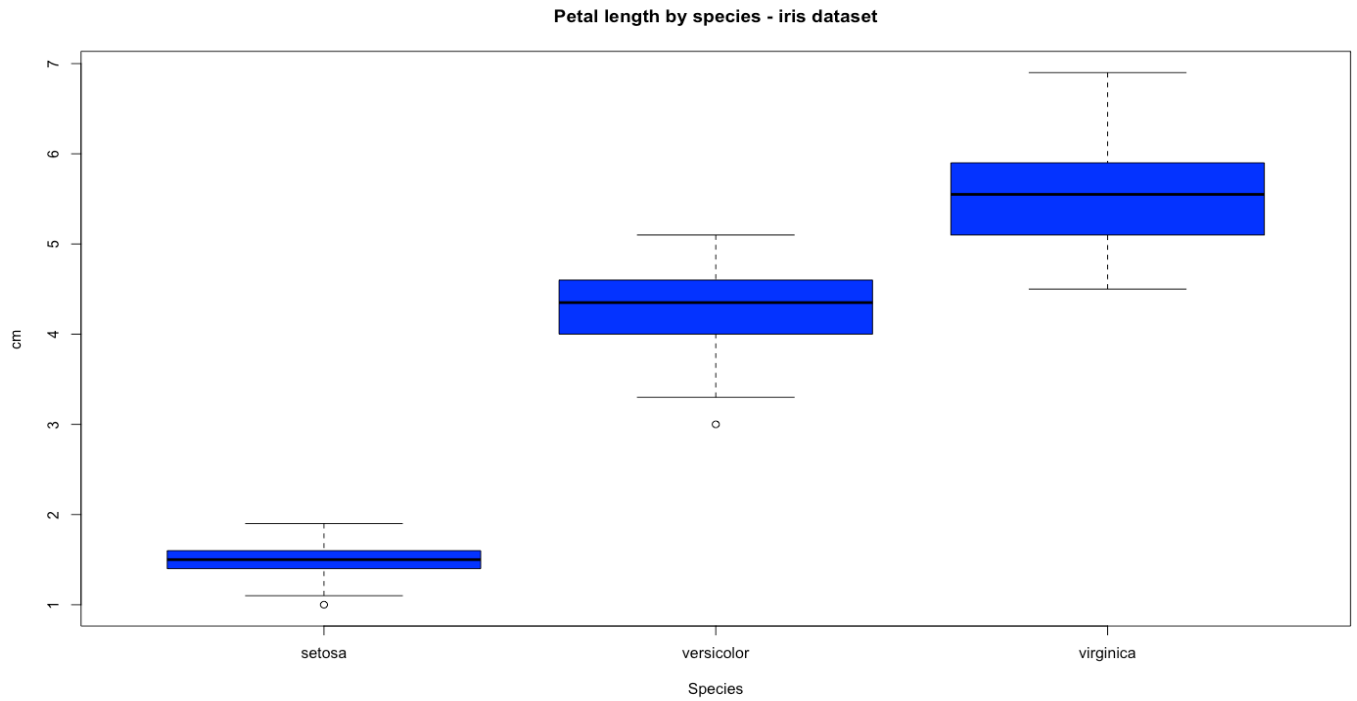
Boxplot & outliers - exemplo

```
> range(iris$Sepal.Width)
[1] 2.0 4.4
> mean(iris$Sepal.Width)
[1] 3.057333
> median(iris$Sepal.Width)
[1] 3
> quantile(iris$Sepal.Width, c(0.25,
0.75))
25% 75%
2.8 3.3
> quantile(iris$Sepal.Width, 0.75) +
IQR(iris$Sepal.Width)*1.5
4.05
> quantile(iris$Sepal.Width, 0.25) -
IQR(iris$Sepal.Width)*1.5
2.05

> boxplot(iris$Sepal.Width, col = "red")
```



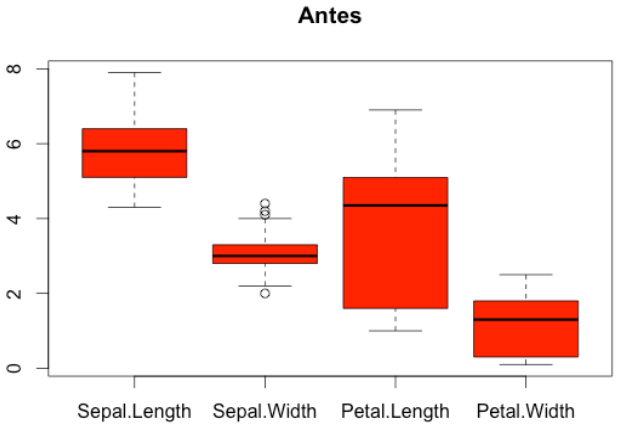
Exemplos boxplots

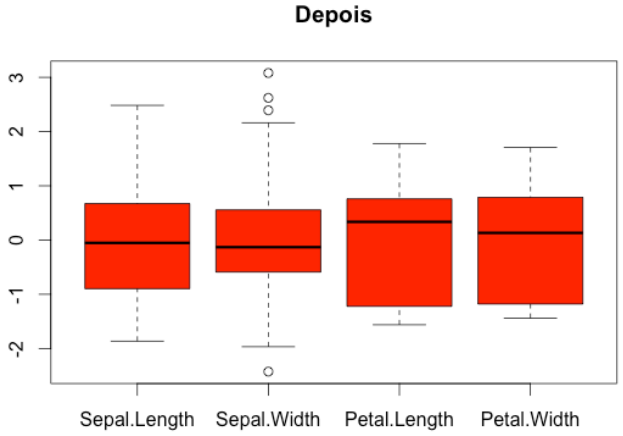



```
boxplot(iris$Petal.Length ~iris$Species, col = "blue", xlab = "Species",
ylab = "cm", main = "Petal length by species - iris dataset")
```

Exemplos boxplots: standardização

```
> iris.st = scale(iris[,1:4])
> boxplot(iris[,1:4], col = "red", main = "Antes")
> boxplot(iris.st[,1:4], col = "red", main =
"Depois")
```





Gráficos de barras

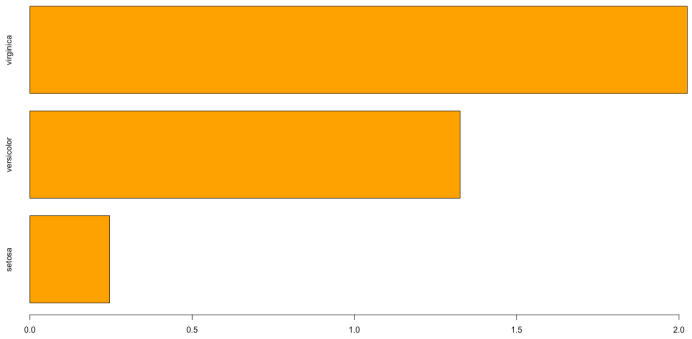
- São uma forma de representar valores numéricos associados a categorias

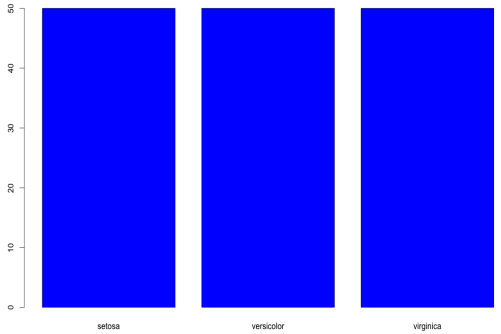
- Podem ser usados para representar frequências de fatores ou para sumariar métricas numéricas em função de categorias

- Implementado em R pela função **barplot**
  - Argumento obrigatório: dados a representar
  - Argumentos opcionais: *horiz*: define se são barras horizontais ou verticais

Exemplos gráficos de barras

```
> barplot(table(iris$Species), col = "blue")
```





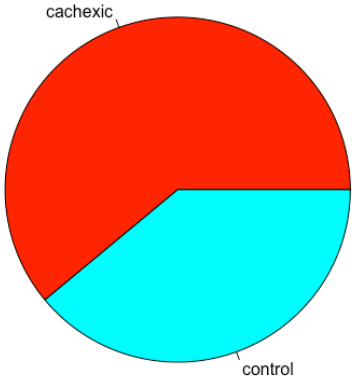

```
> barplot(tapply(iris$Petal.Width, iris$Species, mean), col =
"orange", horiz = T)
```

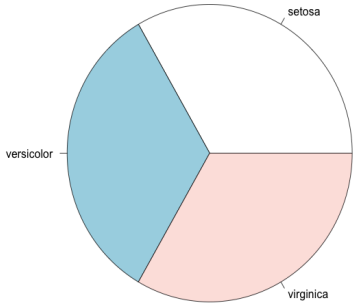
Gráficos tipo pie

Permitem representar frequências de ocorrência de valores distintos de uma variável nominal

Não são muito populares para os estatísticos dada a pouca percepção das relações entre os valores

```
> pie(table(iris$Species))
```






```
pie(table(cachexia$Muscle.loss), col = rainbow(2))
```

Histogramas

Gráfico simples que permite caracterizar a **distribuição de frequências** de valores de um conjunto de dados

Área de cada rectângulo é proporcional ao número de observações do intervalo de valores na base

Permite ter uma primeira ideia da distribuição de dados: por exemplo verificar se os dados seguem uma curva “em forma de sino” que indicie uma distribuição normal; se há enviesamentos, ou mesmo se há problemas como outliers.

Visualização depende bastante do nº de pontos de quebra

Pode ser interessante visualizar também um **gráfico de densidade** (gráfico contínuo e normalizado para a probabilidade de cada valor)

Histogramas

Implementação dos histogramas em R: função ***hist***

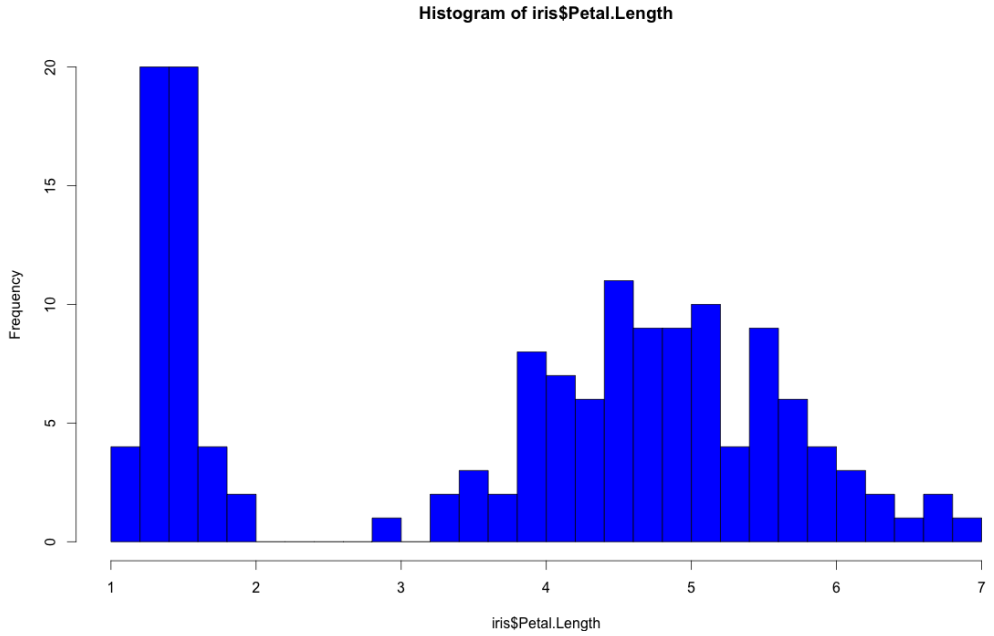
- Argumento obrigatório: variável de dados a representar
- Argumentos opcionais: *breaks* – nº de pontos ou a sua definição num vetor, *col* – cor das barras, *xlim* – vetor com intervalo no eixo dos xx, *ylim* - vetor com intervalo no eixo dos yy, *probability* – variável lógica indicando se valores mostrados são probabilidades (neste caso a área das barras do histograma somam 1), ...
- Função alternativa: ***histogram*** (package lattice)

Implementação da densidade: função **density**

- Gera um conjunto de pontos que pode ser representado independentemente ou sobreposto ao histograma

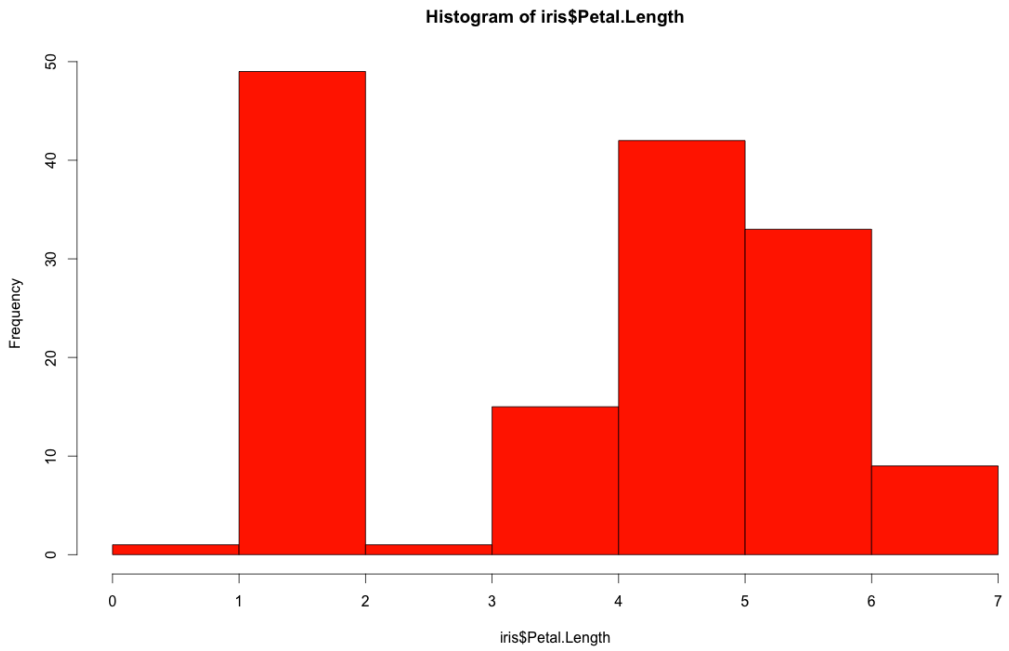
Exemplos histogramas

```
> hist(iris$Petal.Length, breaks= 30, col="blue")
```



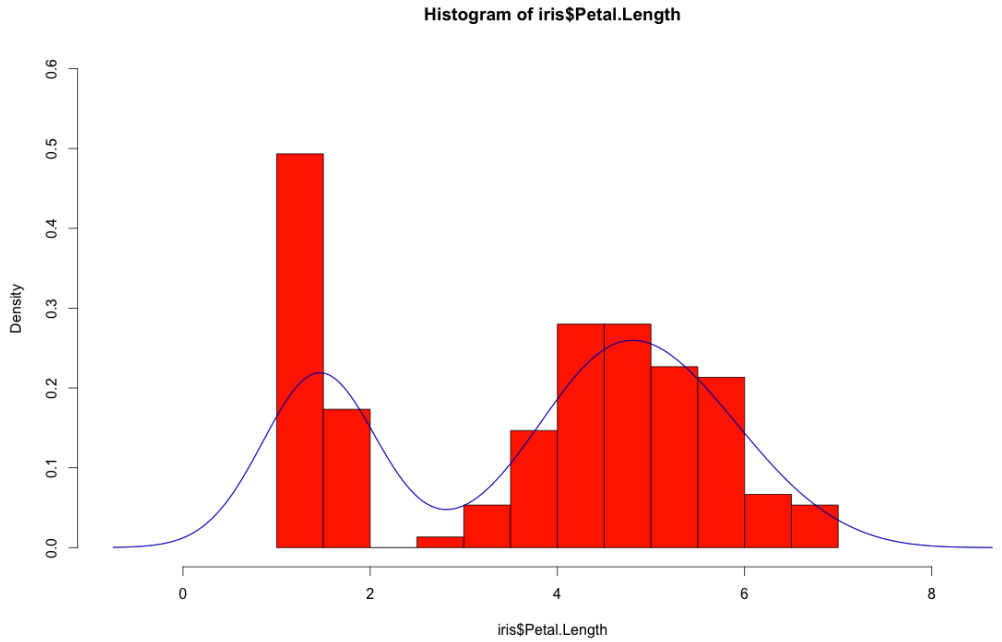
Exemplos histogramas

```
> hist(iris$Petal.Length, breaks= 0:7, col = "red")
```



Exemplos histogramas

```
> dens = density(iris$Petal.Length)
> hist(iris$Petal.Length, breaks= 10, xlim=range(dens$x),
   ylim =c(0,0.6), probability = T, col = “red”)
> lines(dens, col = “blue”)
```



Gráficos de dispersão

Os gráficos de dispersão (ou *scatter plots*) permitem representar uma (ou várias) variáveis numéricas (ditas dependentes) no eixo dos yy em função de outra no eixo dos xx (dita independente)

Implementado pela função ***plot***

Principais argumentos:

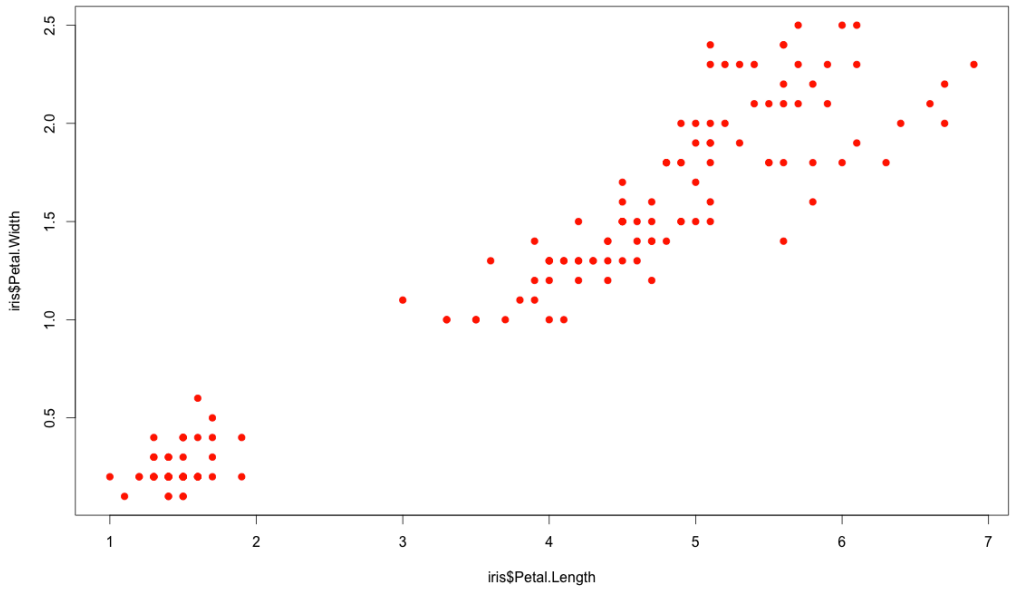
- *x*, *y* – objetos a representar no gráfico em cada eixo (y é obrigatório)
- *type* – tipo do gráfico (linhas – "l", pontos – "p", ...)
- *main* – título do gráfico
- *xlab*, *ylab* – labels dos eixos x, y
- *col* – cor(es)
- *cex* – tamanho dos pontos
- pch – caracter do ponto
- ...

Função ***curve*** serve para representar funções de uma forma mais simples

Função alternativa: ***xyplot*** (package lattice)

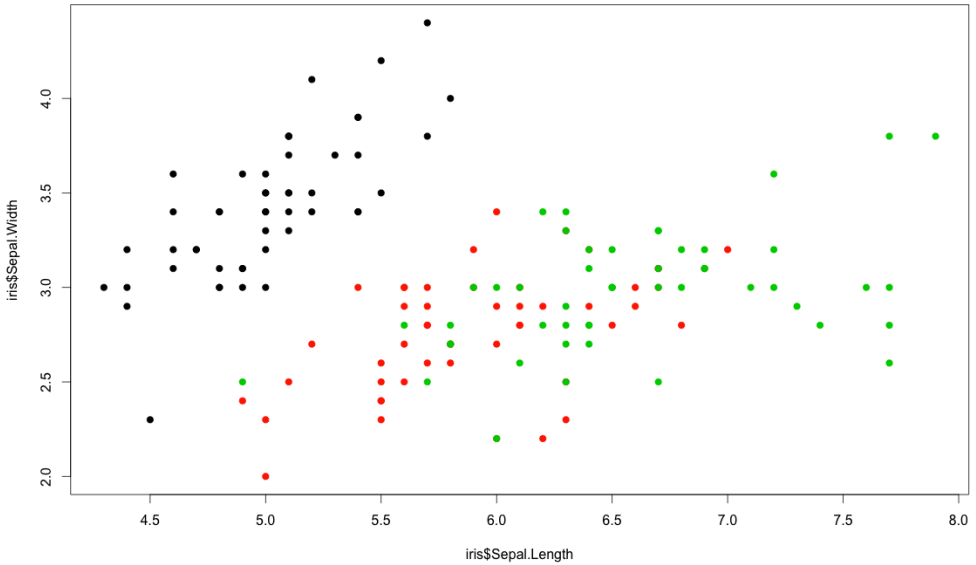
Gráficos de dispersão: exemplos

```
> plot(iris$Petal.Length, iris$Petal.Width, col="red", pch = 19)
```



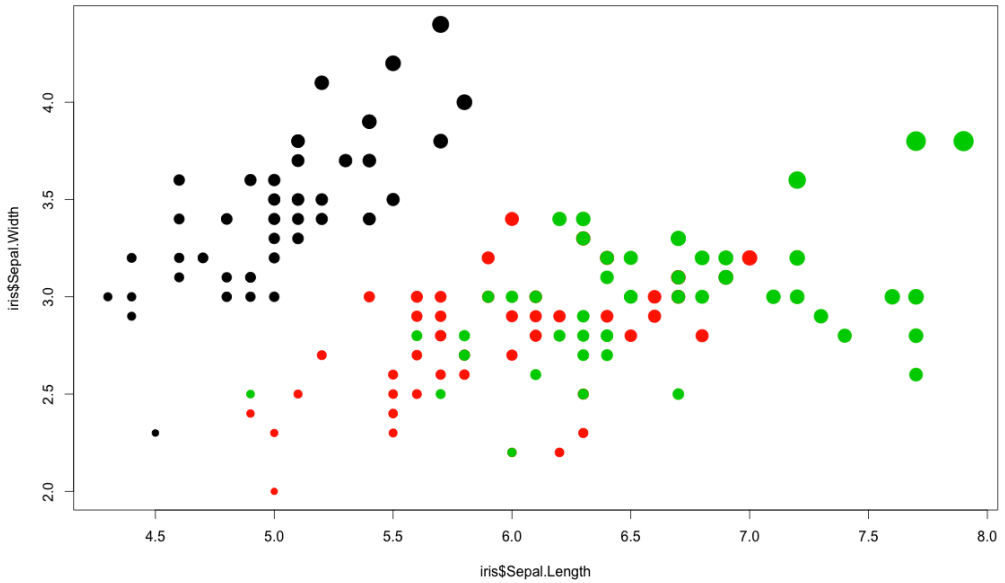
Gráficos de dispersão: exemplos

```
> plot(iris$Petal.Length, iris$Petal.Width, col=iris$Species, pch = 19)
```



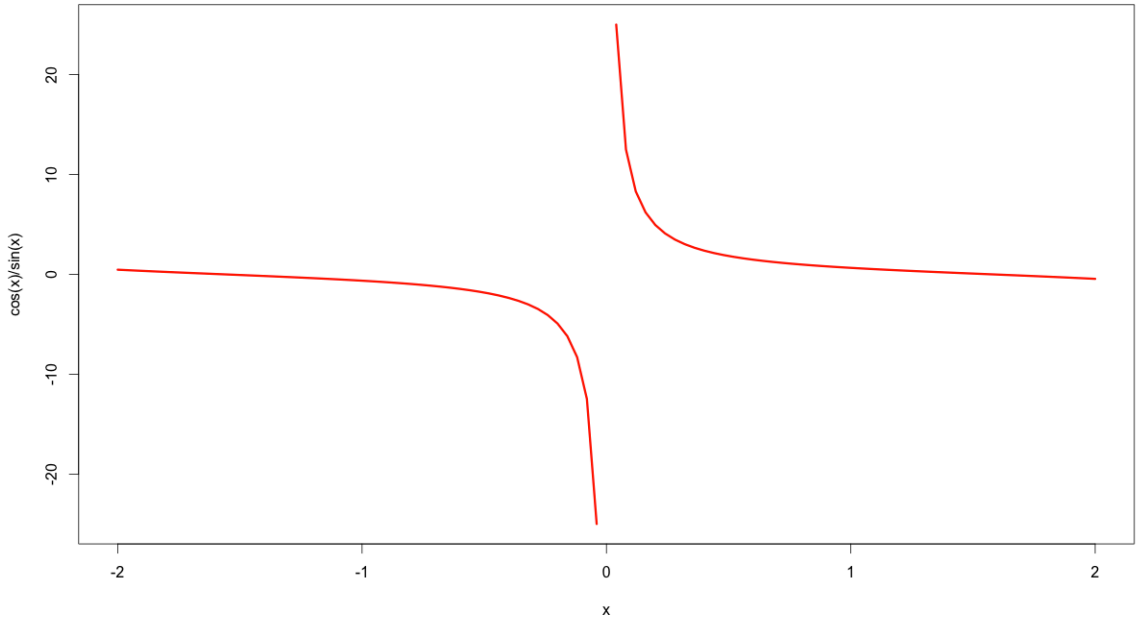
Gráficos de dispersão: exemplos

```
> plot(iris$Sepal.Length, iris$Sepal.Width,
col=iris$Species, pch = 19, cex =
iris$Sepal.Length*iris$Sepal.Width*0.1)
```



Exemplo: função curve

```
> curve(cos(x)/sin(x), -2, 2, col="red")
```



Comparação de todas as variáveis

A função **pairs** permite a comparação num único gráfico de todos os pares de variáveis de um data frame

```
> pairs(iris)
```

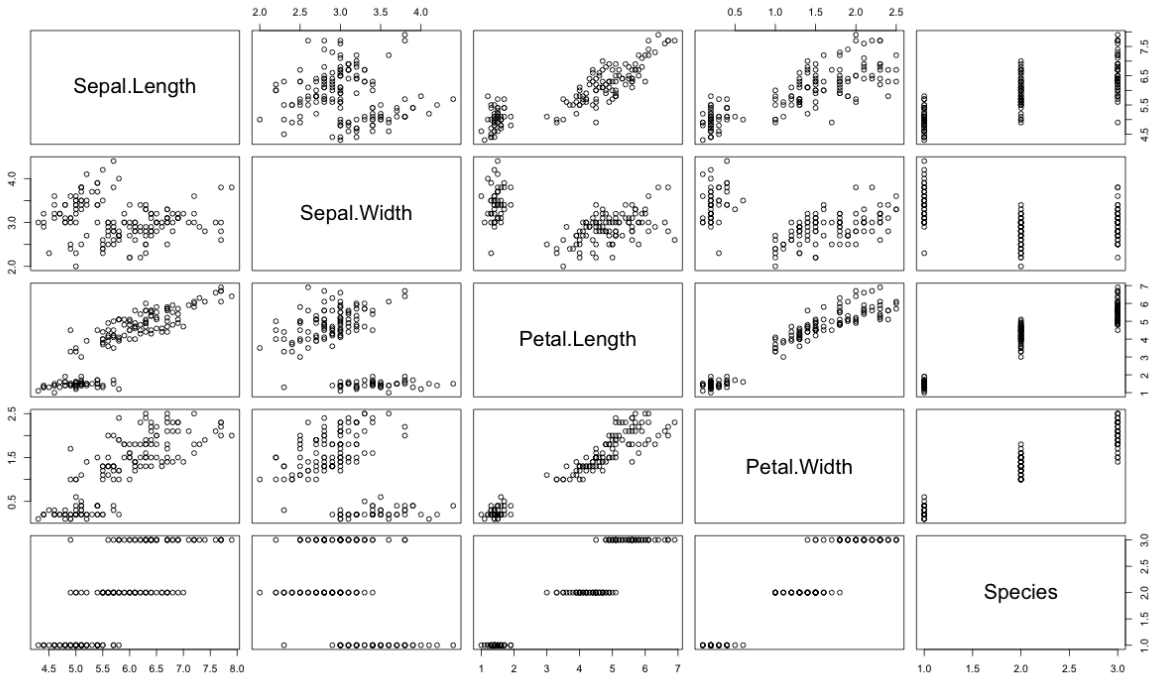
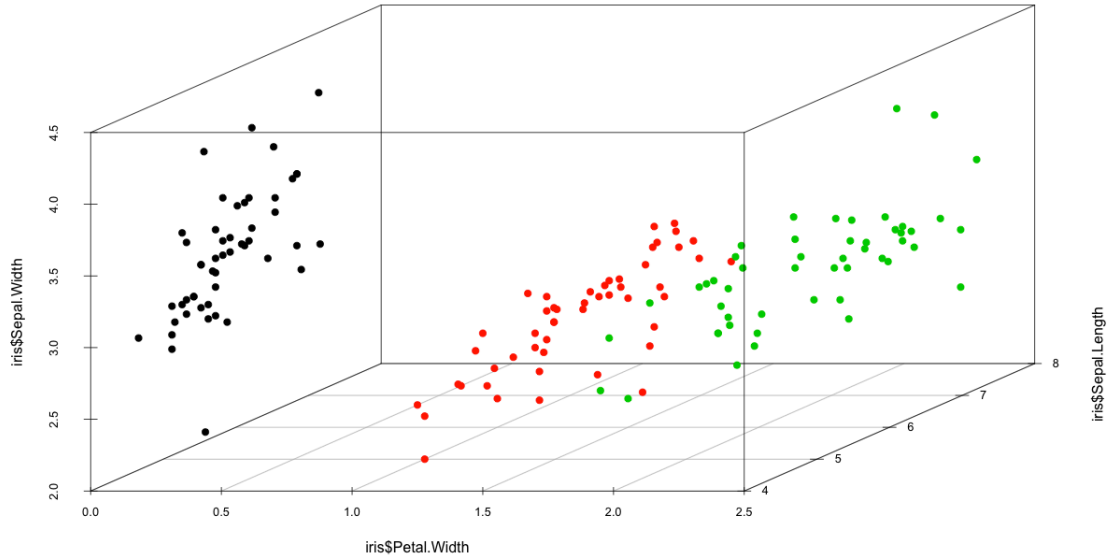


Gráfico 3D

```
> library(scatterplot3d)
> scatterplot3d(iris$Petal.Width,iris$Sepal.Length,iris$Sepal.Width,
pch=19, color=as.integer(iris$Species))
```

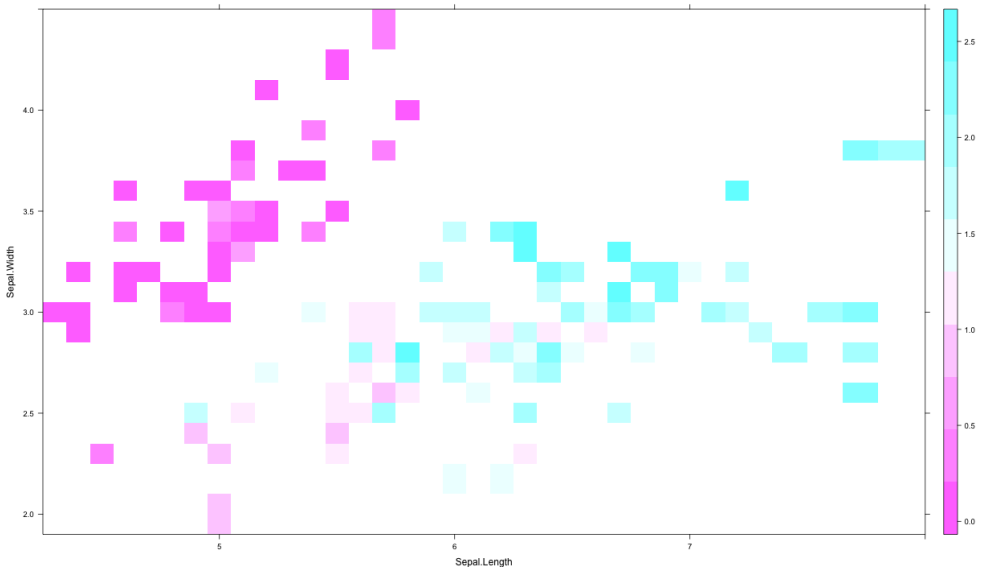


Levelplot

O package "*lattice*" tem um conjunto de funções avançadas para representação gráfica

Como exemplo, vejamos como fazer um "levelplot" que permite cruzar o valor de 3 variáveis, usando os eixos x e y e cores para a terceira variável

```
> library("lattice")
> levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris)
```



Visualizar muitos pontos

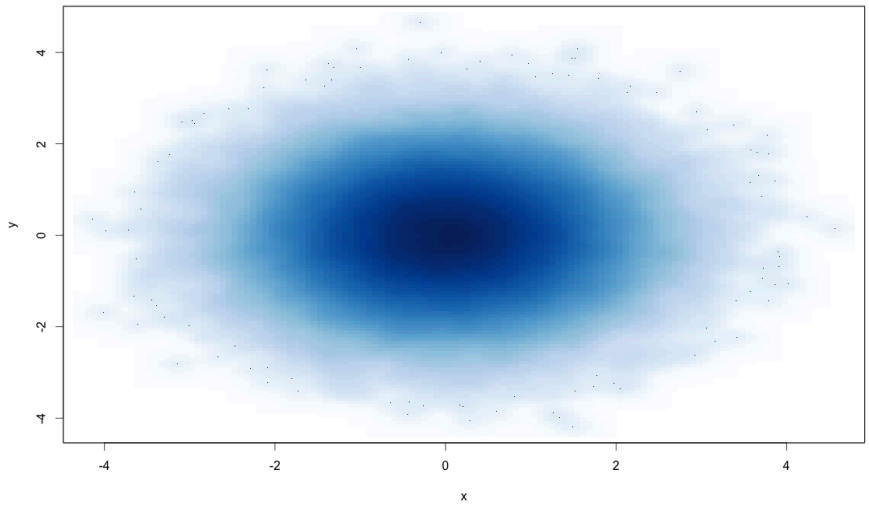
Em alguns casos, a quantidade de pontos a visualizar é demasiado grande para que um gráfico de dispersão traga informação útil (o gráfico fica um borrão gigante)

Pode-se abordar este problema através de métodos de amostragem, i.e. considerando apenas uma parte dos pontos (por exemplo usando a função ***sample*** em R)

Outra abordagem é o uso de outros tipos de gráficos que representam conjuntos de pontos com densidade de cores – e.g. função ***smoothScatter*** do R

Visualizar muitos pontos

```
> X = rnorm(100000)
> y = rnorm(100000)
> smoothScatter(x,y)
```



Sobreposição de objetos gráficos

Em muitos casos, é muito útil a capacidade de sobrepor diversos tipos de objetos sobre gráficos

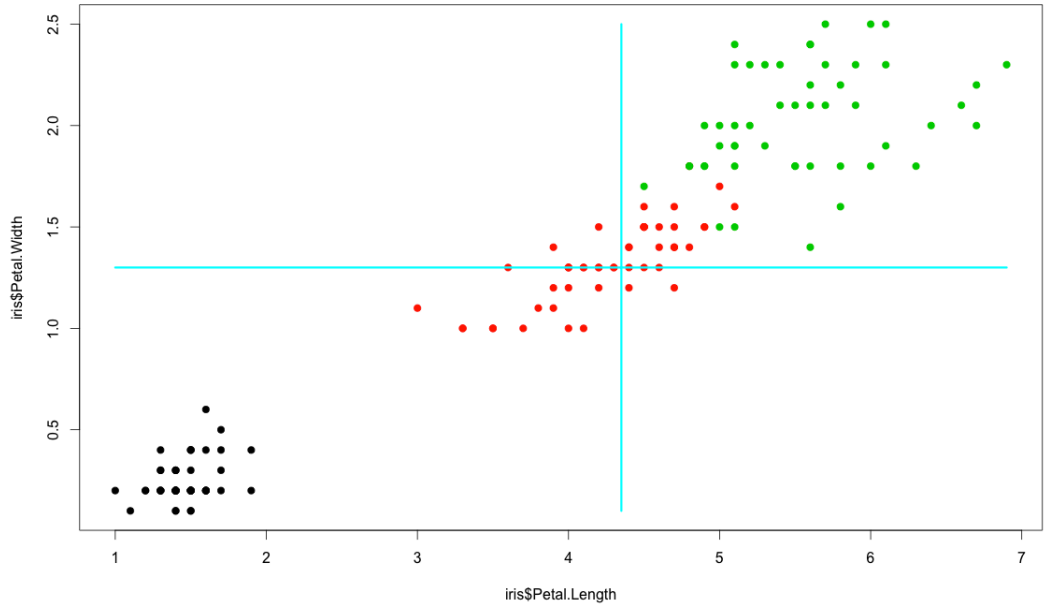
Em R, essa operação pode ser realizada com várias funções:

- *lines* – sobrepõe linhas conectando conjunto de pontos dado por segmentos de reta
- *abline* – linha definida pelo declive e ordenada na origem
- *points* – sobrepõe um conjunto de pontos
- *legend* – permite colocar legendas no gráfico
- *text* – texto
- *symbols* – símbolos matemáticos
- *axis* – textos nos eixos
- *polygon* – áreas “pintadas”

Algumas funções (e.g. *curve, boxplot, barplot)* podem também ser usadas se usarmos o argumento *add = T*

Sobreposição: exemplos

```
> plot(iris$Petal.Length, iris$Petal.Width, col=iris$Species,
pch = 19)
> abline(v = median(iris$Petal.Length), col = "cyan", lwd = 3)
> abline(h = median(iris$Petal.Width), col = "cyan", lwd = 3)
```

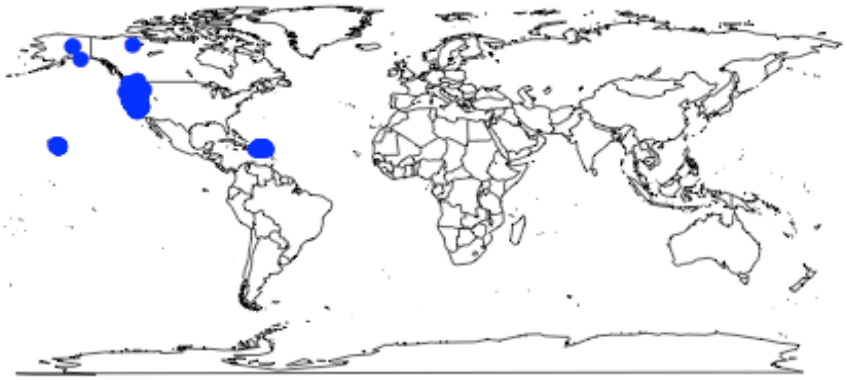


Sobreposição: exemplos

```
> plot(iris$Petal.Length, iris$Petal.Width, col=iris$Species, pch = 19, main = "Relacao
entre largura e comprimento das petalas", ylab="Largura", xlab="Comprimento")
> legend(1.5, 2.2, legend=levels(iris$Species), col=c("black","red","green"),
pch=c(19,19,19))
```

Mapas: exemplo

```
> library("maps”)
> map("world")
> points(eData$Lon, eData$Lat, pch=19, col="blue")
```



Formatando os gráficos

- O R permite que se construam vários gráficos em painéis distintos juntos numa única figura (usando funções ***par*** e ***mfrow***)

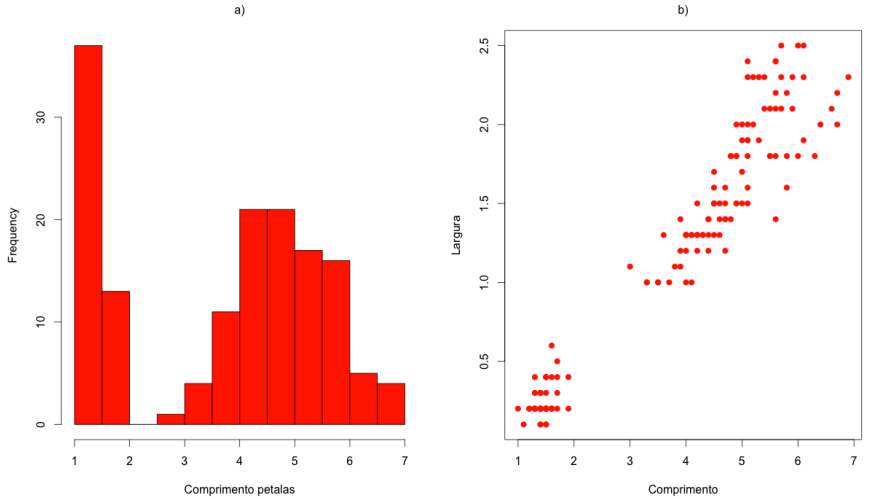
- Usando a função ***mtext*** pode adicionar-se texto a estes painéis

- É ainda possível gravar os gráficos em diversos formatos usando o sistema de devices do R. Por exemplo, a função ***pdf*** permite abrir um device para gravar o gráfico em PDF, definindo o ficheiro. Todos os gráficos são gravados terminando o processo com ***dev.off()***. O mesmo pode ser feito para outros formatos como PNG (função ***png*** no início)

Exemplo de formatação de gráficos

```
> pdf(file = “grafico.pdf”, height=4, width=8)
> par(mfrow=c(1,2))
> hist(iris$Petal.Length, breaks= 10, col = "red", xlab = "Comprimento petalas",
main="")
> mtext(text="a)", side=3, line=1)
> plot(iris$Petal.Length, iris$Petal.Width, col="red", pch = 19, xlab="Comprimento",
ylab="Largura", main="")
> mtext(text="b)", side=3, line=1)
> dev.off()
```

Incluindo linhas a castanho o gráfico é gravado no ficheiro “*grafico.pdf*” da sua pasta de trabalho. Se as excluir o gráfico é mostrado como normalmente



Package ggplot2

O R tem vários sistemas gráficos distintos que têm filosofias distintas de representação dos objetos

Até ao momento, vimos essencialmente funções do sistema base do R, bem como algumas funções do sistema lattice

Um terceiro sistema bastante poderoso para elaborar gráficos é o **ggplot**, atualmente na versão 2 (package ***ggplot2***)

É uma implementação das ideias de Leland Wilkinson, *Grammar of graphics*

Web site: http://ggplot2.org

Package ggplot2

A função básica do ggplot2 é a função **qplot**, que funciona de forma semelhante à função plot do sistema base

Os dados a representar num gráfico ggplot são guardados tipicamente num data.frame

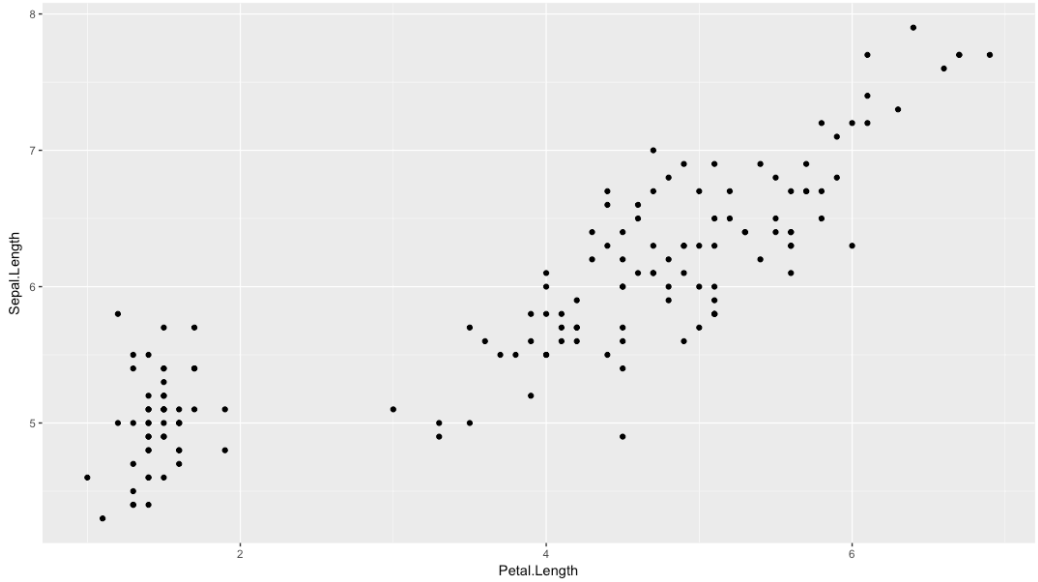
Os plots são definidos por: *aesthetics* (tamanhos, formas, cores) e *geoms* (pontos, linhas)

Fatores no data.frame importantes para definirem sub-conjuntos de dados

Gráfico pode ser programado por layers adicionadas usando função **ggplot** (de mais baixo nível)

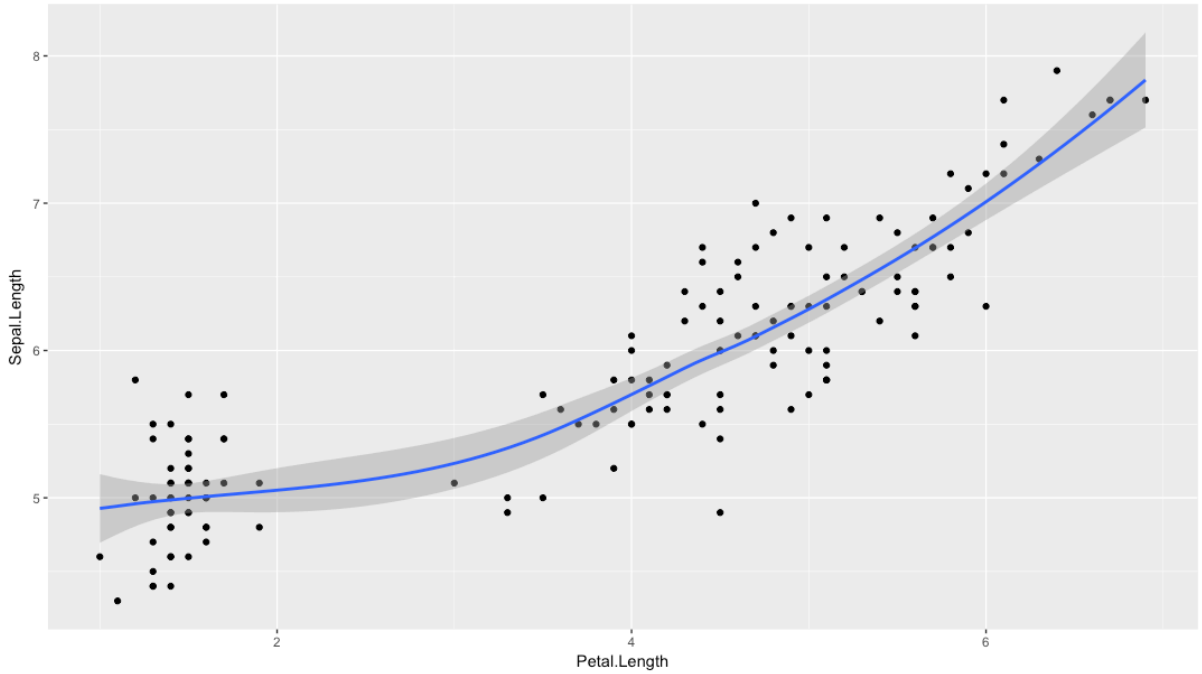
Package *ggplot2*: exemplos

```
> library(ggplot2)
> qplot(Petal.Length, Sepal.Length, data = iris)
```



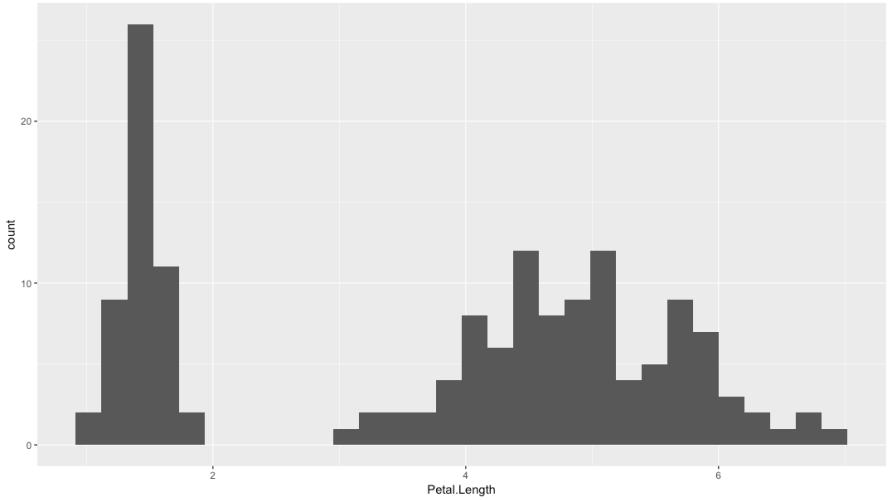
Package *ggplot2*: exemplos

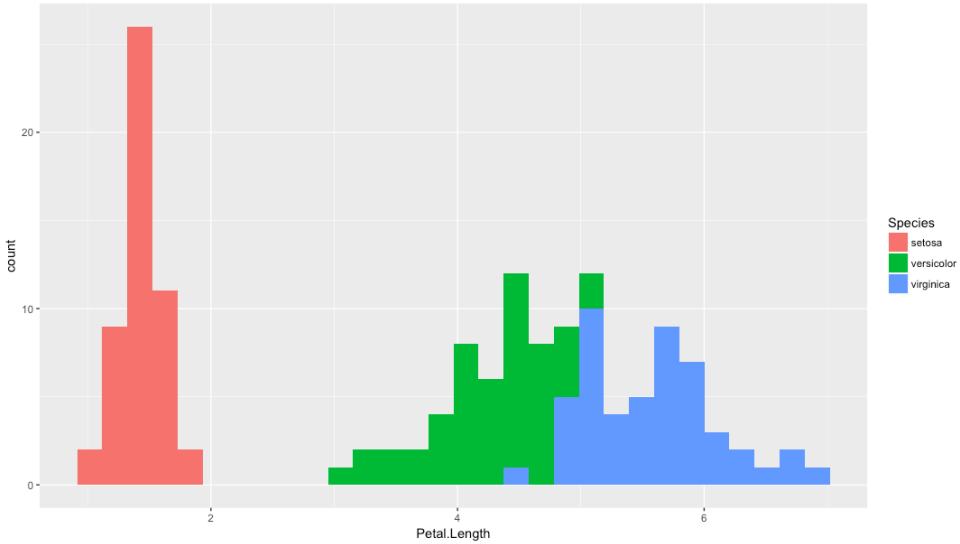
```
> qplot(Petal.Length, Sepal.Length, geom = c("point", "smooth"), data
= iris)
```



Package *ggplot2*: exemplos

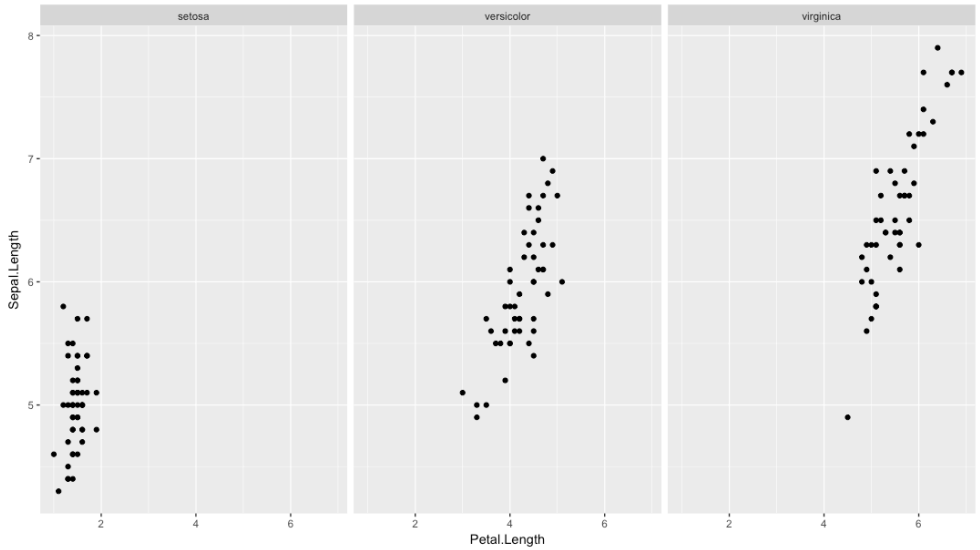
```
> qplot(Petal.Length, data = iris)
> qplot(Petal.Length, fill = Species, data = iris)
```

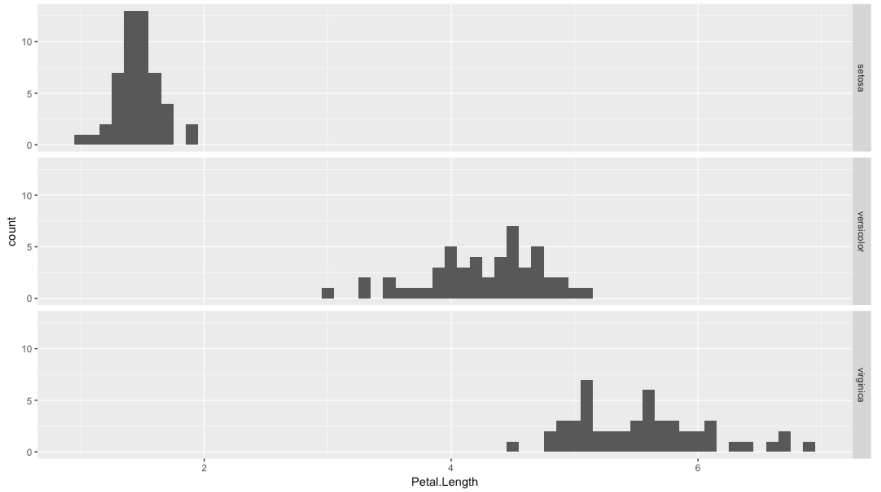




Package *ggplot2*: exemplos

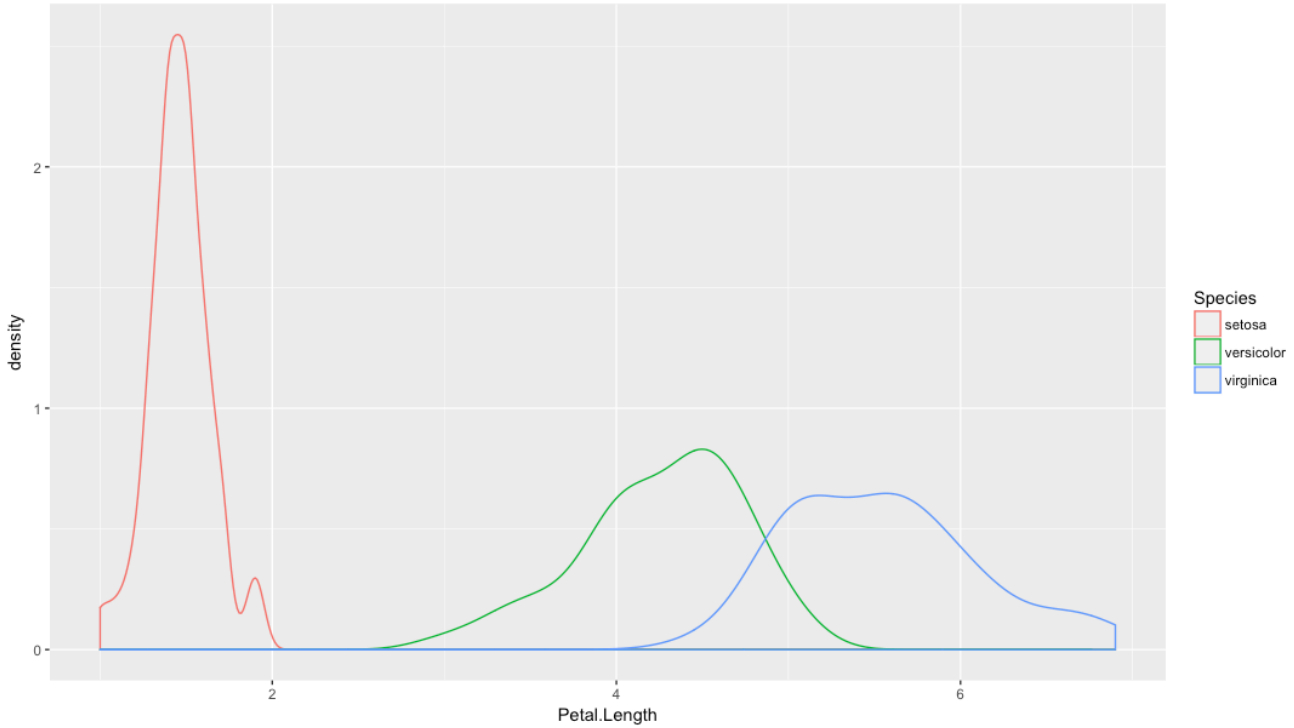
```
> qplot(Petal.Length, Sepal.Length, data = iris, facets = . ~ Species)
> qplot(Petal.Length, data = iris, facets = Species ~., binwidth = 0.1)
```





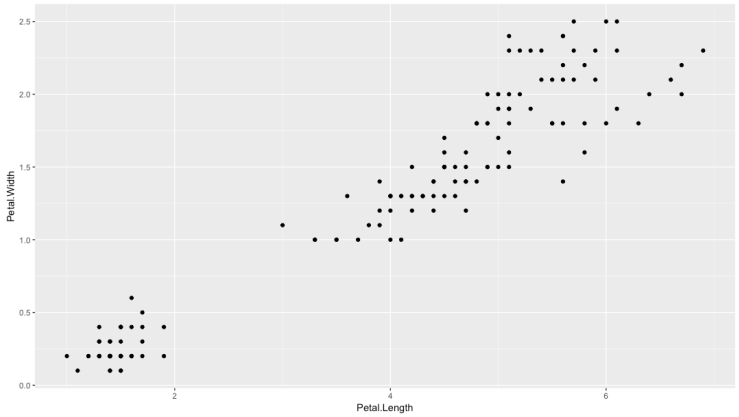
Package *ggplot2*: exemplos

```
> qplot(Petal.Length, data = iris, geom = "density", color = Species)
```



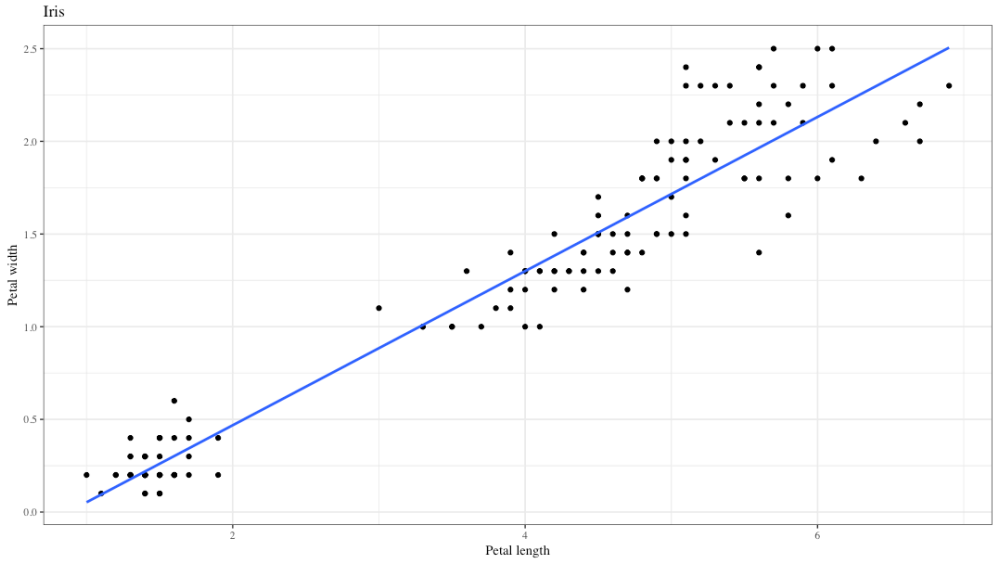
Package *ggplot2*: exemplo com ggplot

```
> g = ggplot(data=iris)
> g = g + aes(Petal.Length, Petal.Width)
> g = g + geom_point()
> print(g)
```



Package *ggplot2*: exemplo com ggplot

```
> g = g + geom_smooth(method = "lm", se = F)
> g = g + theme_bw(base_family = "Times")
> g = g + labs(x = "Petal length") + labs(y = "Petal width")
+ labs(title = "Iris")
> print(g)
```



Um exemplo mais elaborado

A representação gráfica de dados tem como objetivo, em muitos casos, apresentar resultados e como tal, em diversas tarefas variantes destes gráficos e outros tipos de gráficos serão usados com intuitos diversos

O R permite criar gráficos muito elaborados … damos em seguida um exemplo simples destas capacidades

Um exemplo paradigmático pode ser visto em:

http://www.facebook.com/notes/facebook-engineering/visualizing-friendships/469716398919