- Tutorial
- Exercícios
- Apostila

4. Tutoriais de Análise Exploratória de Dados

Antes de iniciar uma análise propriamente dita, precisamos conhecer os dados e avaliar problemas com relação às etapas anteriores da pesquisa (coleta, processamento, estruturação e digitação dos dados brutos). As variáveis coletadas são aleatórias, ou seja, há diferentes fontes de variabilidade associadas a sua representação. Precisamos avaliar essa variabilidade e suas relações com as outras variáveis para poder tomar decisões embasadas sobre as análises que irão testar as hipóteses a respeito dessa variável.

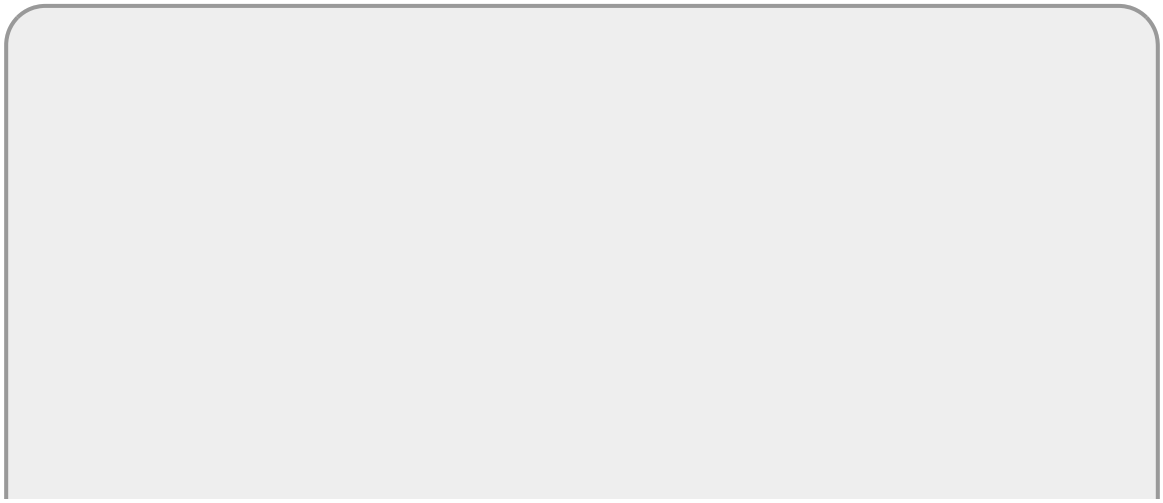
Dentre os principais **objetivos** de uma Análise Exploratória de Dados (AED) podemos listar os seguintes:

- Detectar erros nos dados;
- Detectar valores extremos (*outliers*);
- Compreender a estrutura dos dados coletados;
- Indicar a melhor distribuição para modelar a variabilidade nos dados;
- Avaliar preliminarmente se os dados apresentam dependência/autocorrelação (espacial ou temporal);
- Avaliar se variáveis preditoras apresentam colinearidade;

Não é objetivo desse tutorial passar por todos esses tópicos e sim, apresentar algumas técnicas da linguagem R para auxiliá-lo a iniciar as análises exploratória dos dados. Essas técnicas passam por duas instrumentações básicas: análise numérica com estatísticas descritivas e análises gráficas, explorando a variabilidade das variáveis e sua relação com outras variáveis.

Conferindo Data Frames

Após a leitura dos dados no R é uma boa prática verificá-lo cuidadosamente e investigar as variáveis em busca de erros e incoerências. Nesse tutorial iremos explorar algumas das ferramentas básicas para explorar dados de variáveis.



Nesta sessão apresentamos duas situações bem comuns: variáveis de fatores com códigos errados e valores NA incorretos.

Vamos usar uma planilha com dados fictícios de um estudo de aves no cerrado. Baixe o arquivo do link no seu diretório de trabalho.

Vamos ler o arquivo de texto contendo os dados com o `read.table`. Note que o padrão desse arquivo `csv` é o padrão da exportação do excel configurado em português.

```
aves <- read.table("aves_cerrado.csv", row.names = 1, header = TRUE, sep =
";", dec = ",", as.is = TRUE)
```

Logo após a leitura, garanta que os dados foram lidos corretamente, verificando a estrutura e outros atributos:

Verificação inicial do *data frame*

```
str(aves)
head(aves)
tail(aves)
```

Com isso já temos um diagnóstico de um possível problema: os NAs são de fato observações faltantes? Verificando as informações coletadas em campo descobrimos que esses campos são observações em que não houve avistamento da ave. Portanto, o valor correto é zero [1].

Manipulando NA

Vamos indexar para verificar onde estão esses NAs, usando a lógica básica de extração de registros em uma variável, no caso para `urubu`:

```
aves$urubu
aves$urubu == NA
```

O teste lógico para NAs não funciona pois ele é uma palavra reservada e sua operação retorna NA.

Para o teste lógico específico de NA usamos a função `is.na`, da seguinte forma:

```
is.na(aves$urubu)
which(is.na(aves$urubu))
```

O `is.na` retorna um vetor lógico que podemos usar para indexar o data frame ou apenas uma das suas variáveis:

```
aves[is.na(aves$urubu), ]
aves$fisionomia[is.na(aves$urubu)]
```

É possível também operar vários vetores lógicos para retornar os NAs em mais de uma variável:

```
is.na(aves$urubu) | is.na(aves$carcara))
```

Acima o teste lógico com | entre vetores lógicos, opera as posições utilizando a **regra de equivalência**, retornando TRUE se um dos vetores tiver na posição TRUE. O teste retornará FALSE apenas se os vetores tiverem na mesma posição o valor FALSE. Ao final, o vetor resultante terá TRUE quando `urubu` for NA **ou** `carcara` for NA na posição equivalente.

```
aves[is.na(aves$urubu) | is.na(aves$carcara) | is.na(aves$seriema), ]
```

Se indexarmos o objeto `aves` com a operação lógica `is.na` e | para os três vetores, ele retorna as linhas onde o teste é TRUE, ou seja, onde existe algum NA entre as variáveis `urubu`, `caracara` ou `seriema`. Vamos guardar esses registros em um objeto, atribuindo o resultado do código acima a um objeto:

```
avesNAs <- aves[is.na(aves$urubu) | is.na(aves$carcara) |
is.na(aves$seriema), ]
```

Agora vamos sobrescrever os NAs com o valor `0` nas posições associadas aos vetores lógicos produzidos pelo `is.na`. Para garantir que está sobrescrevendo a posição onde está o erro, faça o processo em duas etapas, primeiro verifique o valor daquela posição antes de alterá-lo.

```
aves$urubu[is.na(aves$urubu)]
aves$urubu[is.na(aves$urubu)] <- 0
```

Fizemos para a variável `urubu` e podemos repetir o código para as outras variáveis. Uma maneira mais eficiente é usar o `is.na` em todo o objeto de uma única vez:

```
is.na(aves)
aves[is.na(aves)]
aves[is.na(aves)] <- 0
```

Para garantir que fizemos a substituição corretamente, vamos verificar, comparando as linhas que agora tem zeros com o objeto onde guardamos os registros contendo NAs:

```
aves[aves$urubu == 0 | aves$carcara == 0 | aves$seriema == 0, ]
avesNAs
```

Conferindo Fatores

As variáveis categóricas ou fatores podem ser conferidas com as funções `table` ou `unique` que retornam os níveis únicos, no primeiro caso fazendo a contagem e no segundo apenas os nomes únicos. Vamos aplicá-las para a variável `fisionomia`:

```
unique(aves$fisionomia)
table(aves$fisionomia)
```

O nível ce parece ser um erro de digitação já que todas as fisionomias deveriam ter 20 pontos amostrados. Para corrigir usamos a indexação usual de teste lógico:

```
aves$fisionomia == "ce"
aves[aves$fisionomia == "ce", ]
aves$fisionomia[aves$fisionomia == "ce"]
aves$fisionomia[aves$fisionomia == "ce"] <- "Ce"
table(aves$fisionomia)
```

Vamos agora converter esse vetor de caracteres para fator, ordenando as fisionomias da mais aberta para a mais fechada:

```
aves$fisionomia <- factor(aves$fisionomia, levels = c("CL", "CC", "Ce"))
```

Sempre é bom verificar novamente para ver se a estruturação e os valores do objeto estão corretos:

```
str(aves)
```

Agora parece que tudo está correto!

Estatística Descritiva

Vamos usar o mesmo arquivo da sessão anterior para explorar as estatísticas descritivas básicas, começando pela média e pela mediana. Nós já usamos o `apply` para aplicar uma função a alguma dimensão de um objeto:

```
apply(X = aves[ ,2:4], MARGIN = 2, FUN = mean)
apply(aves[ ,c("urubu", "carcara", "seriema")], 2, median)
```

O que significaria a média ser muito diferente da mediana? A mediana é pouco sensível aos valores extremos e mais sensível à assimetria da distribuição do que a média.

Podemos também fazer uma média truncada, usando o argumento `trim` da função `mean`. Note que usamos uma outra indexação, com o mesmo resultado das anteriores, e que podemos incluir no `apply` os argumentos da função que iremos aplicar:

```
apply(aves[ , -1], 2, mean, trim = 0.1)
```

O `trim` retira do cálculo da média os valores extremos com o corte definido pelo fator estipulado a partir das observações extremas. Nesse caso, retiramos 10% dos maiores valores e 10% dos menores. Como a média é muito sensível a valores extremos, se houver algum valor muito grande ou pequeno em relação ao resto, a média truncada seria bem diferente da média com todos os dados.

Os quantis são também é uma forma de verificar se a distribuição dos valores é simétrica. O padrão da função `quantile` é retornar os quartis, que é a divisão dos dados em seus quartos depois de ordená-los: mínimo, 1/4, 1/2, 3/4 e máximo, sendo que a mediana é o segundo quartil onde os dados são divididos pela metade (menores e maiores). Quando usamos a função `summary` em um vetor de dados numéricos, esses valores também são apresentados:

```
quantile(aves$urubu)
summary(aves$urubu)
```

O argumento `probs` da função `quantile` permite retornar o valor de qualquer quantil, vamos separar os dados em 10 quantis, com 10% dos dados cada um:

```
quantile(aves$urubu, probs = seq(from = 0, to = 1, by = 0.1))
```

A função `summary` ajuda a resumir essas estatísticas básicas para todas as variáveis numéricas ao mesmo tempo:

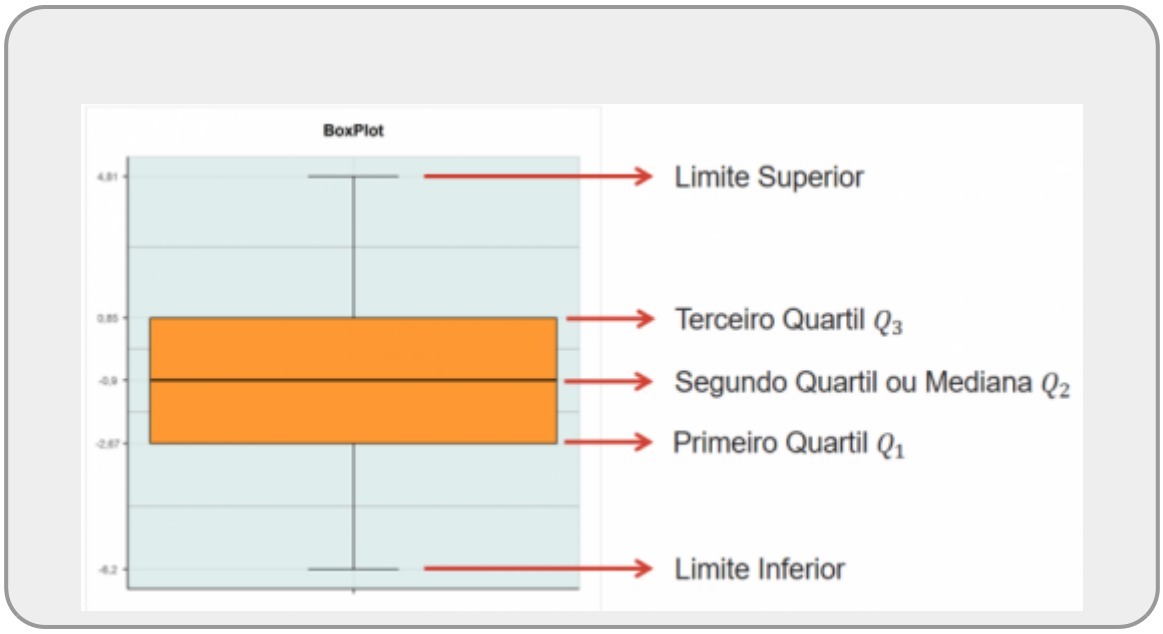
```
summary(aves[ ,-1])
```

Gráficos Univariados

Os gráficos são ferramentas importantes para avaliarmos as variáveis dos nossos dados. No tópico de gráficos vamos estudar mais fundo as funções associadas a elaboração de gráficos no R, por enquanto vamos apenas aplicar algumas funções básicas sem nos preocuparmos muito com o acabamento dos gráficos. Normalmente a análise exploratória de dados é introspectiva, buscando entender as variáveis e não apresentá-las para alguma audiência. Vejamos alguns gráficos básicos de diagnóstico de uma variável numérica, usando como exemplo o avistamento de urubus no Cerrado:

Diagrama de Caixas

O diagrama de caixas (boxplot) é a representação gráfica dos quartis que a função `quantile` retorna.




```
boxplot(aves$urubu)
```

Histograma e Gráfico de Barras

Os histogramas são usados para representar a frequência absoluta (contagem) ou densidade de variáveis contínuas e os gráficos de barras para variáveis categóricas ou discretas. Abaixo construímos as duas formas, mas antes repartimos nossa janela gráfica em quatro porções utilizando a função `par` que modifica os parâmetros gráficos e mudamos o parâmetro `mfrow`.

```
par(mfrow = c(2, 2))
barplot(table(aves$urubu))
hist(aves$urubu)
```

O formato dos gráficos é muito diferente. Isso se deve ao fato do histograma agrupar as observações em intervalos, geralmente regulares. O número de intervalos vai influenciar muito o formato do gráfico, o R usa o algoritmo padrão da função `nclass`. Vejam o que acontece se utilizarmos o argumento `breaks` do `hist`:

```
hist(aves$urub, breaks = 25)
```

Vamos usar a função `stripchart` para fazer mais um gráfico dessa variável e retornar o parâmetro `mfrow` para o seu padrão:

```
stripchart(aves$urubu, method="stack")
par(mfrow = c(1, 1))
```

No código abaixo incluímos os principais gráficos diagnósticos em uma única janela:

```
par(mfrow = c(2, 2))
boxplot(aves$urubu)
```

```
hist(aves$urubu)
barplot(table(aves$urubu))
stripchart(aves$urubu, method = "stack")
par(mfrow = c(1, 1))
```

Comparações com distribuições teóricas

Vamos agora retornar aos dados reais de caixetais para explorar a variável numérica contínua cap das árvores:

```
caixeta <- read.table(file = "caixeta.csv", sep = ",", header = TRUE)
#certifique-se que o arquivo esta em seu diretório de trabalho
summary(caixeta$cap)
par(mfrow = c(1, 2))
hist(caixeta$cap)
boxplot(caixeta$cap)
par(mfrow = c(1, 1))
```

Essa distribuições de valores não parece nada com uma distribuição normal. Parece muito assimétrica com os dados concentrados nos valores menores. Além disso, tem uma longa cauda para os valores maiores. Várias características desses dados levam a uma distribuição como essa, as principais são:

- o cap mínimo de inclusão foi 20 mm, ou seja não há como ter valores menores. Mesmo que pudesse, os valores estariam restritos a valores positivos e seria truncado no zero;
- a estrutura de tamanho de populações estáveis de plantas tem a tendência a ter uma distribuição como essa de J invertido, devido às variações nas taxas vitais (crescimento, sobrevivência e reprodução) ao longo da sua ontogênese.

Uma forma de avaliar o acoplamento de um variável a uma distribuição normal é através dos gráficos `qqnorm` e da relação `qqline` que compara a distribuição dos valores dos quantis (pontos) da variável com os quantis de uma distribuição normal teórica (linha):

```
qqnorm(caixeta$cap)
qqline(caixeta$cap)
```

Uma outra forma de fazer essa comparação é com o histograma de densidade, com o argumento `freq = FALSE` e utilizar a função `curve` que produz a curva da distribuição teórica:

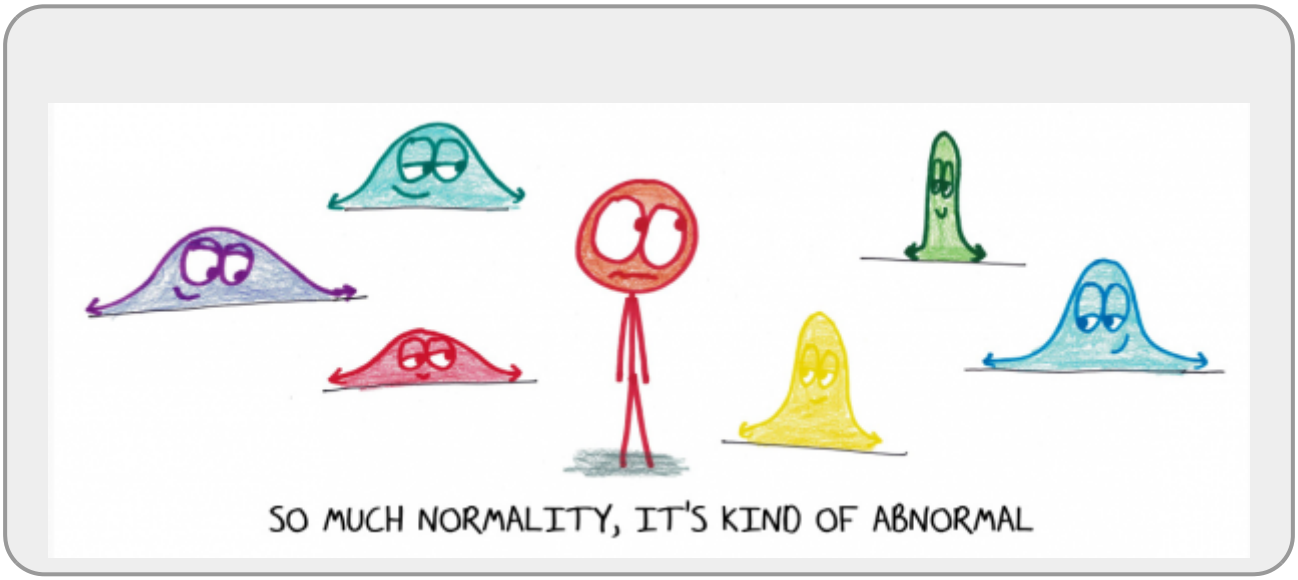
```
hist(caixeta$cap, freq = FALSE)
curve(dnorm(x, mean = mean(caixeta$cap), sd = sd(caixeta$cap)), add = TRUE)
```

Para resolver o problema de distribuição de valores assimétricos e truncados no zero é muito comum utilizar a transformação `log`, vamos explorar essa transformação:

```
par(mfrow = c(1, 2))
hist(caixeta$cap, freq = FALSE)
curve(dnorm(x, mean = mean(caixeta$cap), sd = sd(caixeta$cap)), add = TRUE)
hist(log(caixeta$cap), freq = FALSE)
curve(dnorm(x, mean = mean(log(caixeta$cap)), sd = sd(log(caixeta$cap))),
```

```
add = TRUE)
par(mfrow = c(1, 1))
```

Parece que a transformação logarítmica ajudou no acoplamento da variável à distribuição normal.



Análise exploratória Bivariada

A relação entre duas ou mais variáveis categóricas pode ser explorada com tabelas cruzadas, por exemplo:

```
table(caixeta$especie,caixeta$local)
```

Quando temos uma variável categórica (fator) e uma numérica, as funções `aggregate` e `tapply` são muito úteis.

A função `aggregate` é o equivalente das tabelas dinâmicas das planilhas eletrônicas: ela aplica uma função a partes dos dados separadamente. Por exemplo, para obter um *data frame* com a altura média dos fustes de cada espécie de árvore por local você executa o comando:

```
caixetaH <- aggregate(caixeta$h, by = list(local = caixeta$local, especie =
caixeta$especie), FUN=mean)
```

A função `aggregate` recebe uma ou mais variáveis contínuas e aplica a ela(s) a função que é recebida no argumento `FUN` agrupadas pelas variáveis que foram colocadas no argumento `by` organizadas em uma lista. Veja a estrutura do objeto resultante:

```
str(caixetaH)
```

A função `tapply` faz a mesma operação, mas organiza a objeto resultante em uma `matrix`, veja como ficam organizadas as médias de altura nesse caso:

```
caixAlt <- tapply(caixeta$h, INDEX = list(local = caixeta$local, especie =
caixeta$especie), FUN = mean)
str(caixAlt)
class(caixAlt)
```

```
dim(caixetaH)
dim(caixAlt)
```

Os dados originais continha 1027 observações, agora o `caixetaH` tem 48, pois compilou os dados de altura em sua média para cada espécie em cada uma das localidades. O objeto `caixAlt` por sua vez tem 43 linhas e 3 colunas que contém as mesmas médias. Como é possível?
No primeiro caso nos temos 3 variáveis: `local`, `especie` e x (média de altura). No `caixAlt` temos as linhas representando as `espécies`, as colunas representando as `localidades` e os valores no interior da `matrix` como as médias das alturas.

Consulte a ajuda da função `aggregate` e experimente outras combinações de fatores e funções, com este conjunto de dados.

Gráficos Bivariados

O gráfico de dispersão nos permite avaliar a relação entre duas variáveis numéricas:

```
plot(x = caixeta$cap, caixeta$h)
```

A função `pairs` auxilia na avaliação da relação entre múltiplos pares de variáveis, em um painel:

```
str(aves)
pairs(aves[ , c("urubu", "carcara", "seriema")])
```

A notação de fórmula estatística do R, usa o símbolo ~ para indicar a relação entre duas variáveis.

```
fun(y ~ x, data = dados)
```

- `fun` = função aplicada no objeto `dados`
- y = variável resposta
- x = variável preditora
- `dados` = um data frame contendo os vetores x e y

Esta notação foi criada para os modelos estatísticos, como a regressão linear, mas foi estendida para várias funções gráficas no R onde as relações entre as variáveis pode ser representada. Essa notação facilita muito a produção de gráficos exploratórios da relação entre variáveis. Os comandos abaixo utilizam esta notação:

```
plot(h ~ cap, data = caixeta)
plot(h ~ cap, data = caixeta, subset = local == "jureia")
```

A notação em fórmula nos gráficos tornam o código mais sintético e permitem algumas formulações interessantes, por exemplo entre variáveis contínuas e categóricas:

```
boxplot(cap ~ local, data = caixeta)
```

Esta fórmula pode ainda incluir um fator condicionante, que aplica a relação proposta dentro de cada nível dos condicionais:

```
fun(y ~ x | z, data = dados)
```

- z = variável condicionantes

O pacote `lattice` implementa esta ideia para gráficos:

```
library(lattice)
xyplot(h ~ cap | local, data = caixeta)
```

O quarteto de Anscombe

O pacote `datasets` tem vários conjuntos de dados que se tornaram clássicos da estatística. Vamos analisar um dos mais usados para treino de análises exploratórias. Como o R já carrega o `datasets` por padrão, podemos utilizar os dados diretamente [2)]:

```
str(anscombe)
```

Este objeto é composto de 4 pares de variáveis, nomeadas x1 a x4 (variáveis independentes ou preditoras) e y1 a y4 (variáveis dependentes ou resposta):

```
names(anscombe)
```

Compare as médias de cada uma das variáveis. Para isto, use a função `apply` para aplicar a função `mean` a cada coluna[3)] deste data frame:

```
apply(anscombe[1:4], MARGIN = 2, FUN = mean)
apply(anscombe[5:8], 2, mean)
```

Faça o mesmo para obter as variâncias:

```
apply(anscombe[1:4], 2, var)
apply(anscombe[5:8], 2, var)
```

A pergunta principal para este conjunto de dados é se há relação entre cada variável x e y. Isso pode ser avaliado com o coeficiente de correlação de Pearson, que vai de zero (nenhuma correlação) a um (positivo ou negativo, correlação perfeita).

```
cor(anscombe$x1, anscombe$y1)
```

O código acima pode ser aplicado através da função `with` que simplifica a digitação das variáveis assim como o argumento `data` na função `plot`:

```
with(anscombe, cor(x1, y1))
```

```
with(anscombe, cor(x2, y2))
with(anscombe, cor(x3, y3))
with(anscombe, cor(x4, y4))
```

Uma outra forma de avaliar a relação é através dos coeficientes da reta de um modelo linear entre as duas variáveis. Essa relação é obtida pela função `coef` aplicada ao um objeto de modelo linear [4)]. Vamos avaliar esse coeficientes para os quatro conjuntos de variáveis:

```
(coef1 <- coef(lm(y1 ~ x1, data = anscombe)))
(coef2 <- coef(lm(y2 ~ x2, data = anscombe)))
(coef3 <- coef(lm(y3 ~ x3, data = anscombe)))
(coef4 <- coef(lm(y4 ~ x4, data = anscombe)))
```

Todos os coeficientes estão muito próximos a `3.00` e `0.50`, indicando que os dados são muito similares e tem as mesmas características: médias, desvios, correlações e a relação linear. Vamos fazer os gráficos de dispersão entre as variáveis `x` e `y` para os quatro casos e vamos incluir a relação linear para cada par com a função `abline` que grafa essa relação. Nossa única inclusão no `plot` foram os argumentos `xlim` e `ylim` para que todos os gráficos tenham as mesmas escalas nos eixos [5)]:

```
par(mfrow = c(2, 2)) # 4 graficos em uma janela
plot(y1 ~ x1, data = anscombe,
     xlim = range(anscombe[,1:4]), ylim = range(anscombe[,5:8]))
abline(coef1)
plot(y2 ~ x2, data = anscombe,
     xlim = range(anscombe[,1:4]), ylim = range(anscombe[,5:8]))
abline(coef2)
plot(y3 ~ x3, data = anscombe,
     xlim = range(anscombe[,1:4]), ylim = range(anscombe[,5:8]))
abline(coef3)
plot(y4 ~ x4, data = anscombe,
     xlim = range(anscombe[,1:4]), ylim = range(anscombe[,5:8]))
abline(coef4)
par(mfrow=c(1,1))
```

Este conjunto de dados foi criado pelo estatístico Frank Anscombe para demonstrar a importância da análise visual de dados, veja aqui. Note como são diferentes, o primeiro é uma relação que parece estar adequada às premissas dos modelos lineares, o segundo mostra uma clara relação não linear entre as variáveis, já o terceiro tem um dado influente que promove uma inclinação que não acompanha o conjunto dos dados e promove além da mudança da relação uma não homogeneidade da variância, por fim, o quarto mostra também um ponto muito influente e com alta alavancagem que define a relação entre as variáveis e, caso seja retirado da amostra, as variáveis y4 em função de x4 não apresenta nenhuma relação!

Leia mais sobre análise exploratória de dados

Artigo com um protocolo de análise exploratória de dados:

- http://onlinelibrary.wiley.com/doi/10.1111/j.2041-210X.2009.00001.x/pdf

Capítulo do livro “Analysing Ecological Data” sobre análise exploratória de dados:

- zuur_cap_4_exploration_statistics.pdf

1)

o problema inverso é comum e mais difícil de diagnosticar: colocar zero onde a observação não estava disponível!

2)

a função `data` carrega os dados do pacote no nosso `workspace`, mas isto não é necessário já que ele está disponível na sessão a partir do `environment` `datasets`

3)

argumento `MARGIN=2`

4)

isso será alvo das aulas de modelo linear

5)

veja a documentação do `plot`