Aula: Regressão Boladona - O Passo a Passo

Mercel Santos

Introdução

Regressão linear é uma técnica estatística muito utilizada por estabelecer um modelo matemático (equação) usado para determinar (estimar) valores de variáveis ditas como dependentes com base em variáveis independentes.

Nesta aula, mostro o **PASSO a PASSO** necessário para fazer **regressão linear com o R** . Além disso, mostro como escrever uma função que facilita fazer regressão linear com base em dados do tipo **dataframes**.

Preparando os Dados

Para trabalhar com leitura e manipulação de dados vamos ter que usar alguns pacotes. Neste caso, utilizei os seguintes pacotes:

- ggplot2
- ggpubr
- gdata

O **ggplot2** e o **ggpubr** serão usados para plotar os dados. No entanto, nessa aula, **stat_regline_equation()** será a única função utilizada do pacote **ggpubr**, que será usada para inserir a equação de regressão e o valor do R^2 nos gráficos. O pacote **gdata** é destinado para leitura de arquivos do Excel, ou seja, os arquivos com a extensão .xls ou .xlsx.

Além desses pacotes, é claro, você vai precisar de dados, porque a linguagem R foi criada basicamente para análise de dados e que vem sendo muito utilizada para esse propósito.

Bom, uma vez que sabemos quais pacotes utilizaremos e qual a finalidade de cada um deles, o resto agora é programação.

Para começar esse programa vamos garantir que todos os objetos ou variáveis sejam removidos da memória

principal do computador. Então, para remover objetos da memória no R usamos o comando **rm()**. rm é uma abreviação de *remove*, palavra em inglês que significa remover.

Então para entender como funciona o comando **rm()**, vamos declarar as seguintes variáveis:

```
a <- 5
b <- 10
c <- 15
```

Agora sabemos que as variáveis **a,b,c** estão na memória do computador, pois as mesmas foram declaradas. No entanto, podemos ter certeza que essas variáveis estão na memória do computador através do comando **ls()**. Este comando lista os objetos que estão na memória do computador. Veja abaixo:

```
ls()
```

```
## [1] "a"                  "b"                  "c"
```

Podemos remover a variável **a** da seguinte forma:

```
rm(a)
```

Se verificarmos novamente as informações da memória veremos que a variável **a** foi removida, pois não aparece mais na lista.

```
ls()
```

```
## [1] "b"                  "c"
```

Imagine agora que queiramos apagar todos os objetos da memória com apenas uma linha de comando. Isso é possível juntando os comandos **rm()** e **ls()**. No comando **rm()** existe um parâmetro chamado **list** onde é possível informar as variáveis que serão apagadas da memória. Como sabemos, podemos listar essas variáveis através do comando **ls()**. Então juntando esses dois comandos podemos apagar TODAS as variáveis da memória da seguinte forma:

```
rm(list = ls())
```

Estes comandos devem ser escritos na primeira linha de todo programa em R.

Já que sabemos apagar TODAS as variáveis da memória, o próximo passo é ler os dados no formato **xlsx**. Para isso, vamos carregar o pacote **gdata** e usar a função **read.xls()** para ler os dados.

```r
require("gdata") # carregando o pacote gdata

setwd("/home/mercel/aulas/regressao") # definido o diretório de trabalho

dados <- read.xls("dadosPesosAltura.xlsx",sheet = 2) # lendo os dados
```

Observe que foi utilizado o comando **setwd()** para estabelecer o diretório de trabalho.

Durante a leitura dos dados com a função **read.xls()** foi informado o nome do arquivo ("dadosPesosAltura.xlsx") para leitura e o número da planilha (planilha número 2) lida dentro do arquivo **dadosPesosAltura.xlsx**. Como visto no código acima, a especificação da planilha é feita através do parâmetro **sheet**. *Sheet* é uma palavra em inglês que pode ser traduzida como planilha. Os dados contidos na planilha 2 do arquivo **dadosPesosAltura.xlsx** contém informações referentes a peso, altura e idade de pessoas.

Por padrão, a estrutura de dados lidos pela função **read.xls()** é do tipo dataframe. Isso pode ser verificado através do comando **class()**. Veja:

```r
class(dados)
```

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

Podemos fazer uma análise rápida desses dados usando os comandos **head()** e **tail()** (em português, head=cabeça, tail=calda). O comando **head()** quando aplicado a um dataframe mostra para o usuário a parte superior dos dados mais o cabeçalho. Por padrão, são mostradas as 6 primeiras linhas. Veja:

```r
head(dados)
```

```
##   Pessoa PESO ALTURA IDADE  X
## 1      1   86    187    40 NA
## 2      2   77    176    35 NA
## 3      3   84    189    34 NA
## 4      4   79    180    29 NA
## 5      5   68    172    42 NA
## 6      6   81    176    35 NA
```

Caso seja desejado expor um número maior ou menor que 6 linhas, basta especificar. Segue o exemplo.

```r
head(dados,12)
```

```
##    Pessoa PESO ALTURA IDADE  X
## 1       1   86    187    40 NA
## 2       2   77    176    35 NA
## 3       3   84    189    34 NA
## 4       4   79    180    29 NA
```

```
## 5       5   68    172    42 NA
## 6       6   81    176    35 NA
## 7       7   87    181    45 NA
## 8       8   86    183    36 NA
## 9       9   80    184    28 NA
## 10     10   79    184    21 NA
## 11     11   90    182    47 NA
## 12     12   64    171    49 NA
```

O comando **tail()** funciona de forma similar ao **head()**. A diferença é que tail mostra a parte inferior dos dados conforme o seguinte exemplo:

```
tail(dados,8)
```

```
##     Pessoa PESO ALTURA IDADE  X
## 125    125   90    186    38 NA
## 126    126   73    182    25 NA
## 127    127   71    176    20 NA
## 128    128   81    174    45 NA
## 129    129   84    175    46 NA
## 130    130   63    171    23 NA
## 131    131   93    186    22 NA
## 132    132   63    170    33 NA
```

Bom, agora que já sabemos ler os dados e fazer uma rápida análise, podemos então fazer uma regressão linear.

Fazendo Regressão Linear com o Comando lm()

Uma regressão linear em R, tanto simples como múltipla, é feita através do comando **lm()**. lm é uma abreviação de *Linear Model*, que significa modelo linear. Esta função requer basicamente dois parâmetros. Uma fórmula que detalha o tipo de regressão e um dado do tipo **dataframe**.

Fórmulas em R são estabelecidas através do operador "~". Veja abaixo como é simples fazer um objeto do tipo fórmula.

```
minha.formula <- Y~X
```

Para demonstrar que a operação **Y~X** resultou em uma fórmula, aplicarei o comando **class()** à variável **minha.formula**.

```
class(minha.formula)
```

```
## [1] "formula"
```

Observe que a função **class()** retornou o seguinte *character* “formula”. Indicando que o objeto ou a variável é do tipo **fórmula**.

Quando tratamos de **Regressão Linear**, sabemos que vamos estabelecer uma equação linear (equação de primeiro grau) onde teremos uma variável dependente (Y) que dependerá de outra variável dita independente (X). Logo, podemos dizer que Y é **função de** X ou que Y **depende de** X. Em fórmulas de modelos estatísticos, "~" é lido como **função de** ou **depende de**. Então quando queremos estabelecer um modelo estatístico e escrevemos **Y~X**, podemos dizer que **Y é função de X** ou **Y depende de X**.

Agora que sabemos criar uma fórmula e interpretar a mesma, vamos fazer uma regressão linear.

O primeiro passo é determinar quem serão as variáveis dependente e independente. Para escolhermos as variáveis vamos rever o cabeçalho dos nossos dados com o comando **head()**.

```
head(dados)
```

```
##   Pessoa PESO ALTURA IDADE  X
## 1      1   86    187    40 NA
## 2      2   77    176    35 NA
## 3      3   84    189    34 NA
## 4      4   79    180    29 NA
## 5      5   68    172    42 NA
## 6      6   81    176    35 NA
```

Como mostrado acima temos cinco colunas com nomes e dados diferentes. Os nomes dessas colunas também podem ser acessados através do comando **names()**, que significa nomes. Veja:

```
names(dados)
```

```
## [1] "Pessoa" "PESO"   "ALTURA" "IDADE"  "X"
```

Bom, neste exemplo quero determinar uma equação linear que calcula o PESO em função da ALTURA. Logo, PESO (Y) será minha variável dependente e ALTURA (X) minha variável independente. Só lembrando que as variáveis **PESO E ALTURA estão no daframe dados**. Então podemos aplicar o comando **lm()** da seguinte forma:

```
lm(PESO~ALTURA,dados)
```

```
##
## Call:
## lm(formula = PESO ~ ALTURA, data = dados)
##
```

```
## Coefficients:
## (Intercept)       ALTURA
##    -179.086        1.437
```

Observe que a função **lm()** retorna os coeficientes de regressão. Com esses coeficientes é possível montar uma equação do primeiro grau simples.

Como o resultado da regressão mostra que o coeficiente da variável ALTURA é igual a 1.437 e o intercepto equivale a -179.0, a equação de regressão fica da seguinte forma, $peso = -179.0 + 1.43 * altura$. Com essa equação é possível calcular o peso das pessoas com base na altura das mesmas. Vamos fazer um exemplo para facilitar o entendimento.

Considerando que o indivíduo possua 186 centímetros (1 metro e 86 centímetros) podemos estimar seu peso da seguinte forma:

```
altura <- 186
peso <- -179.0 + 1.43*altura
print(peso)
```

```
## [1] 86.98
```

O resultado do exemplo acima mostra que para um indivíduo com altura equivalente a 186 centímetros, seu peso estimado corresponde a 86.98 kg. Observe também que escrevi as variáveis propositalmente em minúsculo, isso para mostrar que esse código não possui ligação direta com o **dataframe dados**. O **dataframe dados**, que contém as informações de peso e altura nomeados como PESO E ALTURA respectivamente, foi usado apenas para fazer a regressão linear através da função **lm()**. Uma vez feita a regressão, focamos nos parâmetros estatísticos da mesma.

Falando em parâmetros estatísticos da regressão linear, podemos determinar um resumo estatístico mais detalhado utilizando a função **summary()** — *summary* que significa resumo em português.

Para isso, precisamos atribuir o resultado da regressão a uma variável e aplicar o comando **summary()**. Veja como fica na prática:

```
modelo <- lm(PESO~ALTURA,dados)
summary(modelo)
```

```
##
## Call:
## lm(formula = PESO ~ ALTURA, data = dados)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6096  -5.1607  -0.3621   5.9024  14.5748
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -179.08644   17.13105  -10.45   <2e-16 ***
## ALTURA         1.43688    0.09522   15.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.793 on 130 degrees of freedom
## Multiple R-squared:  0.6366, Adjusted R-squared:  0.6338
## F-statistic: 227.7 on 1 and 130 DF,  p-value: < 2.2e-16
```

Como mostrado acima, a função **summary()** além de resultar os coeficientes de regressão, realiza uma análise estatística do resíduo e mensura a performasse da regressão linear.

Plotando os Resultados

Uma das formas mais eficientes de analisar dados é através de gráficos. Por isso, vamos plotar aqui, de forma fácil, um gráfico profissional que mostra a distribuição dos dados analisados e a reta de regressão linear que representa a variabilidade dos dados. Poderíamos fazer esse gráfico com os seguintes comandos básicos do R, **plot()** e **abline()**, porém seria mais trabalhoso para torná-lo profissional. Sendo assim, vamos utilizar o comando **ggplot()** do pacote **ggplot2**.

Um plot utilizando o pacote **ggplot2** é composto por três partes: i) *data* (dados), ii) *aesthetic* (estética) e iii) geometry (geometria). Isso significa dizer que para fazer um plot com este pacote temos que especificar esses três componentes. Então, podemos representar um plot com o **ggplot2** da seguinte forma:

$$Plot = dados + estética + geometria$$

- Na parte dados, como o próprio nome sugere, são informados os dados do tipo **dataframe**.
- Na estética são informados as variáveis x e y. Nessa parte podem ser feitas também personalização de cores, formato, tamanho de pontos, espessuras de linha, etc.
- Na geometria é onde são escolhidos os tipos de gráficos (pontos, linhas, barras, etc).

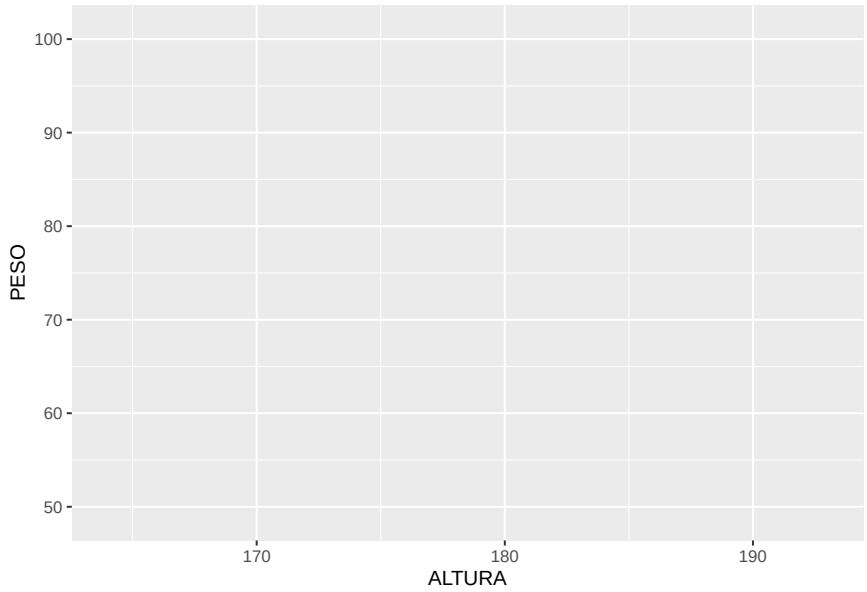
Bom, agora que temos uma boa noção do que precisamos, vamos fazer um gráfico de dispersão usando o comando **ggplot()** para verificarmos visualmente a relação entre as variáveis dependente e independente.

1. Carregando o pacote.

```
require(ggplot2)
```

2. Definindo os dados e as variáveis.

```
ggplot(dados,aes(x=ALTURA,y=PESO))
```



Observe que até o momento foi definido apenas os dados e as variáveis dentro da função **aes()** — aes é uma abreviação de *aesthetic*. É com essa função que podemos especificar a estética do gráfico.

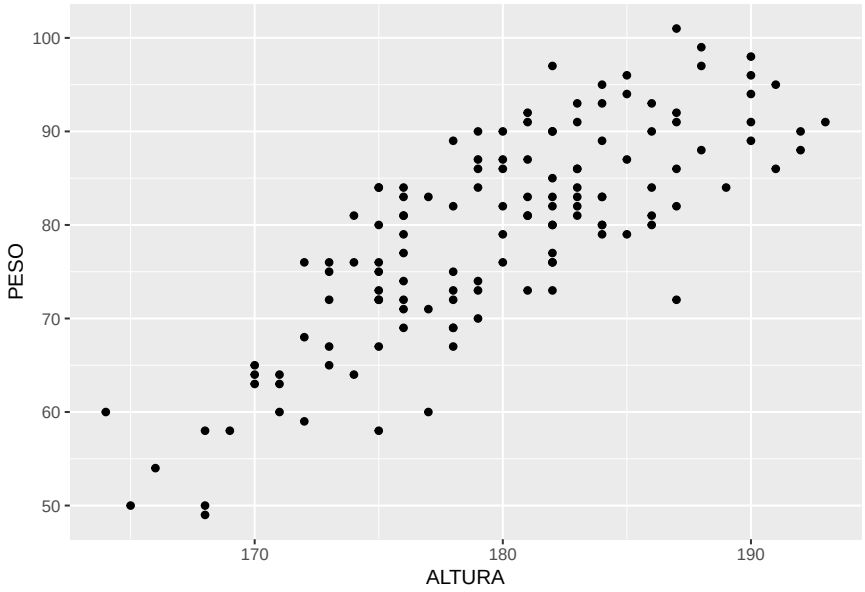
Agora falta estabelecer o tipo de gráfico (geometria) para que finalmente os dados possam aparecer no gráfico.

3. Especificando a geometria.

Como queremos um gráfico de dispersão (scatter plot), devemos escolher uma geometria do tipo pontos. Isto é feito através do comando **geom_point()**.

Na prática fica da seguinte forma:

```
ggplot(dados,aes(x=ALTURA,y=PESO)) + geom_point()
```

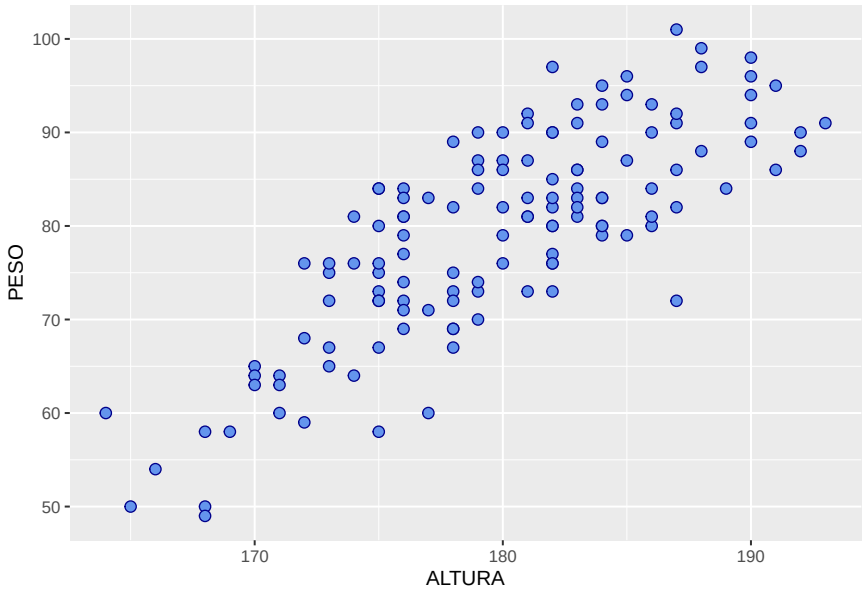


Observe que o comando **geom_point()** foi adicionado como uma camada no **ggplot()**.

Podemos melhorar a aparência desses pontos adicionando os seguintes parâmetros com os respectivos valores na função **geom_point()**.

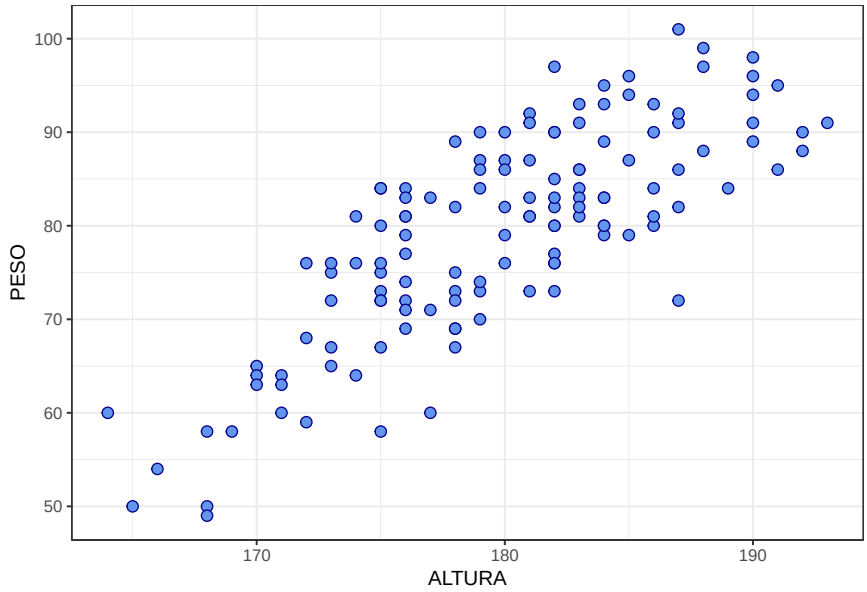
- pch = 21 (pontos do tipo circunferência com preenchimento)
- col = “blue4” (cor da borda da circunferência)
- bg = “cornflowerblue” (cor do interior da circunferência)
- size= 2.5 (tamanho do ponto)

```
ggplot(dados,aes(x=ALTURA,y=PESO)) + geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5)
```



Veja que essas mudanças simples já melhorou a aparência do gráfico, tornando-o mais profissional. Também podemos definir um tema mais atrativo usando o comando **theme_set()**.

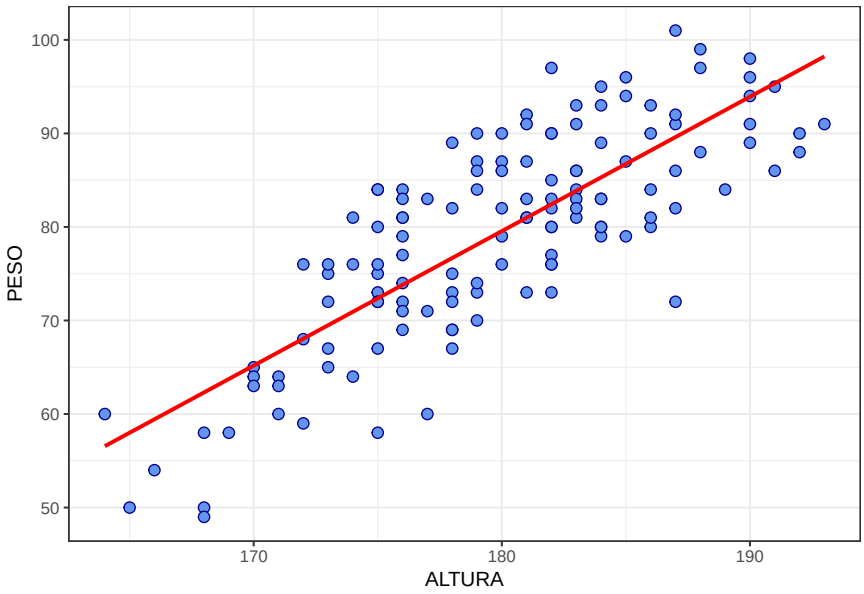
```
theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5)
```



Veja que no comando **theme_set()** foi estabelecido o tema **bw** (abreviação de *white background* — fundo branco) usando a função **theme_bw()**.

Agora que temos o gráfico de dispersão, podemos inserir a reta de regressão usando o comando **geom_smooth()**. Esse comando será mais uma camada adicionada ao comando **ggplot()**.

```
theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)
```



Observe que na função **geom_smooth()** foram definidos os seguintes parâmetros com os respectivos valores.

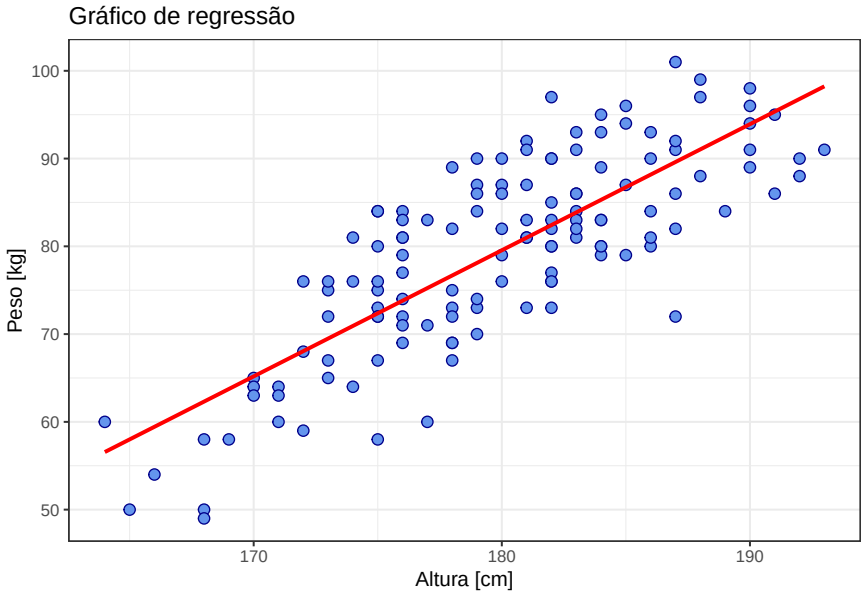
- method = lm
- col = “red”
- se = FALSE

No parâmetro **method** foi definido o método de ajuste dos dados. Nesse caso, foi escolhido método **lm** que é uma abreviação de *linear model* (modelo linear). A cor da reta foi definida como vermelho (*red*) no parâmetro **col** . No parâmetro **se**, foi definido que NÃO será mostrado o intervalo de confiança da reta de regressão, uma vez que **se = FALSE**.

Agora para deixarmos o nosso gráfico mais informativo, podemos inserir título, legendas, reta de regressão e o R^2.

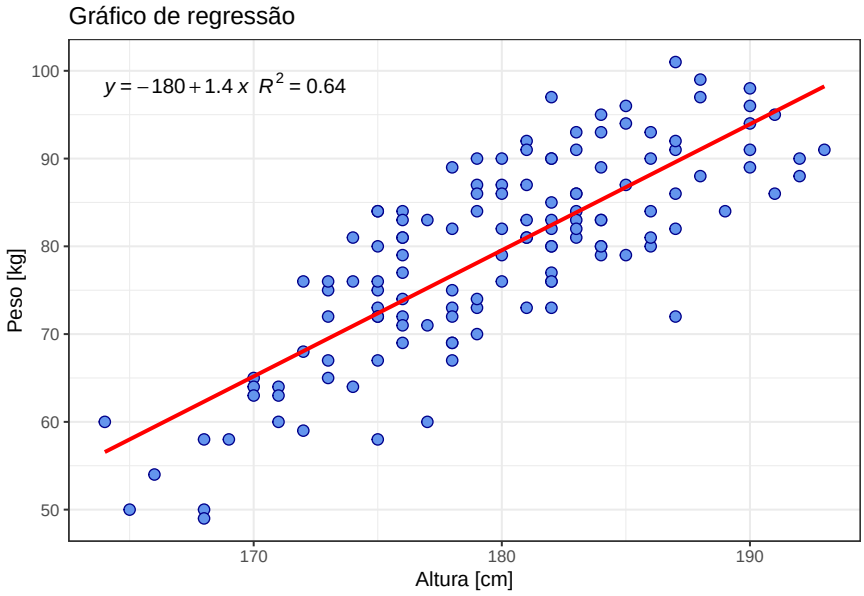
O título e as legendas são inseridas no gráfico através do comando **labs()**, que é mais uma camada adicionada ao comando **ggplot()**. No comando **labs()**, o título é adicionado através do parâmetro **title** e as legendas dos eixos x e y através dos parâmetros **x** e **y** respectivamente. Observe na prática como é simples,

```
theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
  labs(title = "Gráfico de regressão",
       x = "Altura [cm]",
       y = "Peso [kg]"
       )
```



De forma similar, podemos inserir a equação da reta de regressão e o R^2 ao gráfico. Para isso, vamos usar o comando **stat_regline_equation()** juntamente com a função **aes()**. Na função **aes()** inserimos a equação e o R^2 no parâmetro **label**. Veja como fica:

```
require(ggpubr)

theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
  labs(title = "Gráfico de regressão",
       x = "Altura [cm]",
       y = "Peso [kg]"
       ) +
  stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
```



Observe que no parâmetro **label** da função **aes()** foram inseridos a equação e o R^2, representados por **..eq.label..** e **..rr.label..** nesta ordem. Os códigos **..eq.label..** e **..rr.label..** são convertidos em objetos do tipo “expression”, que é o formato de expressão matemática usado pelo R. Para juntar esses objetos foi usado o comando **paste()**, onde foi especificado que os objetos foram separados por dois espaços ("~~"). Espaços em expressão matemática é representado com "~".

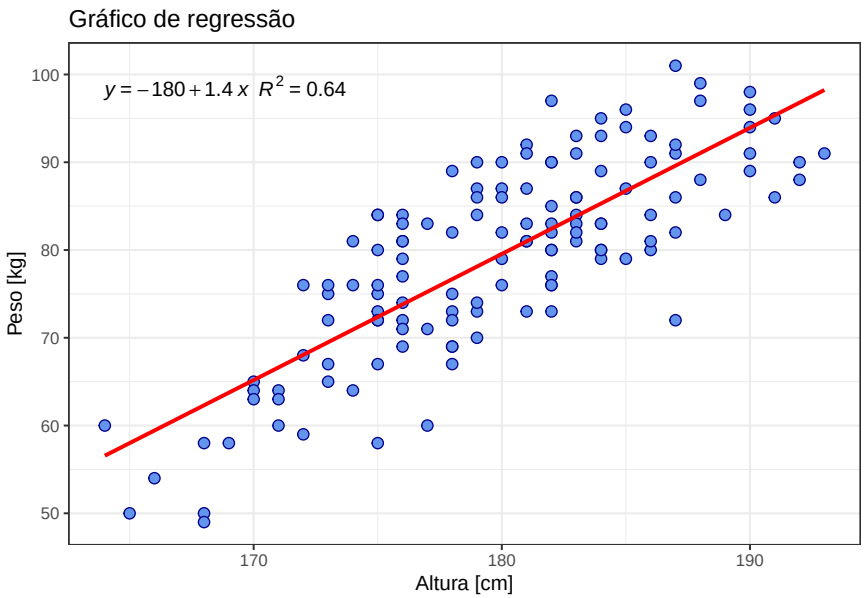
Criando a função regressao.boladona()

Agora que foi mostrado o PASSO a PASSO de como se faz uma regressão linear no R, podemos fazer uma função para incorporar todos esses passos em um único comando.

O primeiro passo para fazer a função é juntar todos os comandos e verificar se está tudo funcionando como planejado. Juntando todos os comandos necessários temos:

```
rm(list = ls())

require("gdata") # carregando o pacote gdata
require(ggpubr)
require(ggplot2)

setwd("/home/mercel/aulas/regressao") # definido o diretório de trabalho
dados <- read.xls("dadosPesosAltura.xlsx",sheet = 2) # lendo os dados

modelo <- lm(PESO~ALTURA,dados)

theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
```

```
geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
geom_smooth(method = lm,col="red",se=FALSE)+
labs(title = "Gráfico de regressão",
     x = "Altura [cm]",
     y = "Peso [kg]"
     ) +
stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
```



Observe que os comandos acima reproduz o gráfico de regressão da forma desejada.

Feito esse primeiro passo, podemos fazer uma função para "empacotar" esses comandos e evitar repetição de código quando uma próxima regressão linear for realizada.

Para fazer uma função em R usamos o comando **function()**. O esquema abaixo mostra a estrutura do comando **function()** ao passo que exemplifica como nomear uma função.

```
minha.funcao <- function(argumentos){

    Bloco de comandos da função

}
```

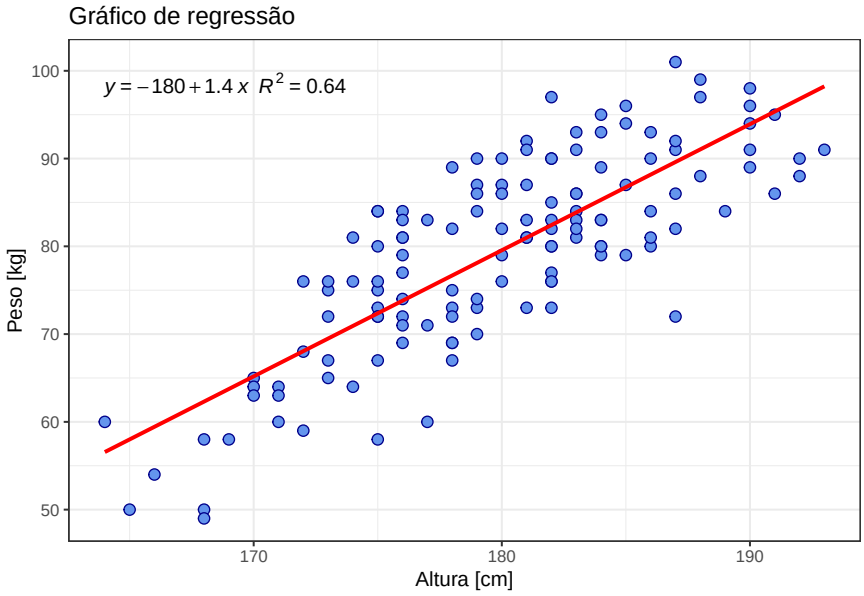
Observe acima que uma função é composta por argumentos e o bloco de comandos. Os argumentos ou parâmetros são os dados de entrada usados pelo bloco de comandos da função. No bloco de comandos ficam os comandos necessários para execução de tarefas, neste caso, as tarefas são o cálculo da regressão e a plotagem do gráfico. Verifique também que a função do esquema acima está sendo atribuída a um nome (**minha.funcao**) através do comando de atribuição (**<-**)

Dito isso, podemos agora fazer uma função da maneira mais simples (porém não versátil) sem a especificação de argumentos. No exemplo abaixo, mostro como fica a função **regressao.boladona()** sem o uso de argumentos.

```
rm(list = ls())

require("gdata") # carregando o pacote gdata
require(ggpubr)
require(ggplot2)

setwd("/home/mercel/aulas/regressao") # defindo o diretório de trabalho
dados <- read.xls("dadosPesosAltura.xlsx",sheet = 2) # lendo os dados


regressao.boladona <- function(){

modelo <- lm(PESO~ALTURA,dados)

theme_set(theme_bw())
ggplot(dados,aes(x=ALTURA,y=PESO)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
  labs(title = "Gráfico de regressão",
       x = "Altura [cm]",
       y = "Peso [kg]"
       ) +
  stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
}
```

Para usar a função, basta digitar o nome da função seguido de abre e fecha parênteses. Veja:

```
regressao.boladona()
```



Que MARAVILHA MEUS AMIGOS, isso chama-se o poder das funções! Agora sempre que quisermos determinar a regressão linear, com os dados acima, basta digitar **regressao.boladona()**.

O grande porém é que essa função só funcionará corretamente com os dados acima, ou seja, a mesma não possui versatilidade para funcionar com qualquer dado do tipo **dataframe**. Para solucionar essa questão, vamos colocar argumentos ou variáveis de entrada na função.

Sendo assim, primeiramente, deve-se escolher os argumentos. Neste caso, os argumentos serão:

- entraDados (receberá o nome do dado de entrada do tipo dataframe)
- dep.var (receberá o nome da variável dependente)
- indep.var (receberá o nome da variável independente)
- leg.x (receberá a legenda do eixo x)
- leg.y. (receberá a legenda do eixo y)

Vale lembrar que o nome desses argumentos foi escolhido por mim, isso quer dizer que você pode nomear da sua maneira também.

Agora podemos inserir esses argumentos na função **regressao.boladona()** para torná-la versátil.

```
regressao.boladona <- function(entraDados,dep.var,indep.var,
                               leg.x="Legenda X",leg.y="Legenda Y"){

modelo <- lm(as.formula(paste(dep.var,indep.var,sep="~")),entraDados)
summary(modelo)

theme_set(theme_bw())
ggplot(entraDados,aes_string(x=indep.var,y=dep.var)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
```

```
  labs(title = "Gráfico de regressão",
       x = leg.x,
       y = leg.y
       ) +
  stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
}
```

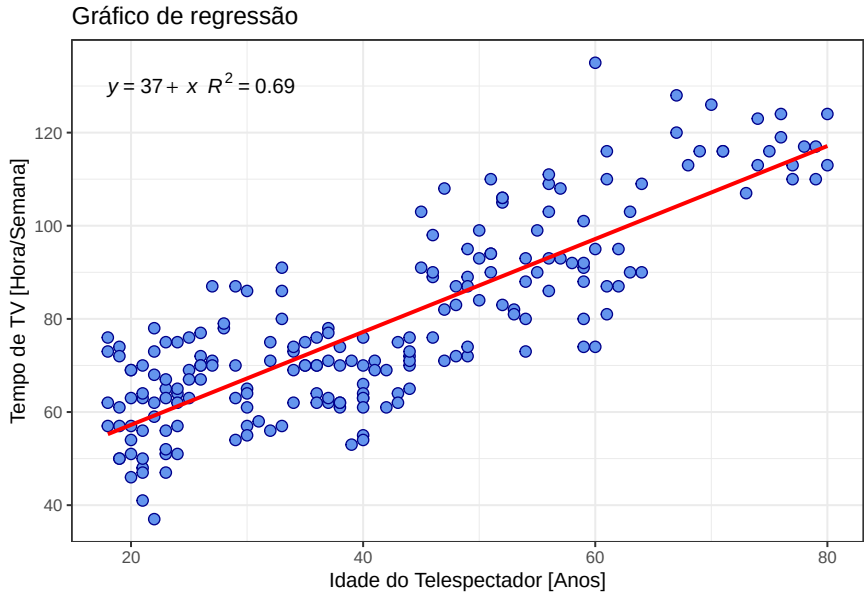
Veja que a "GRANDE MÁGICA" foi inserir os argumentos (dados de entrada) dentro do bloco de comandos da função. Sendo assim, toda vez que for alterado algum dado de entrada da função **regressao.boladona()**, a mesma produzirá um resultado diferente. Observe também que adicionei a linha de comando **print(summary(modelo))** para que apareça no terminal um resumo estatístico da regressão linear toda vez que a função **regressao.boladona()** for executada.

Agora que a função **regressao.boladona()** está versátil, com ela é possível fazer uma regressão linear para quaisquer variáveis de um determinado dado de entrada do tipo **dataframe**. Para exemplificar, vamos fazer uma regressão linear entre o número de horas de TV assistidos por semana e a idade dos telespectadores. Para fazermos isso, como já sabemos, precisamos carregar os pacotes, ler os dados, selecionar as variáveis (dependente e independente) e por fim executar a função **regressao.boladona()**. Veja como fica:

```
rm(list = ls())

require("gdata") # carregando o pacote gdata
require(ggpubr)
require(ggplot2)

setwd("/home/mercel/aulas/regressao") # defindo o diretório de trabalho
dados <- read.xls("idadeTV.xls",sheet = 1) # lendo os dados


regressao.boladona <- function(entraDados,dep.var,indep.var,
                               leg.x="Legenda X",leg.y="Legenda Y"){

modelo <- lm(as.formula(paste(dep.var,indep.var,sep="~")),entraDados)

print(summary(modelo))

theme_set(theme_bw())
ggplot(entraDados,aes_string(x=indep.var,y=dep.var)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
  labs(title = "Gráfico de regressão",
       x = leg.x,
       y = leg.y
       ) +
  stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
}
```

```
regressao.boladona(dados, "Horas","Idade","Idade do Telespectador [Anos]",
                   "Tempo de TV [Hora/Semana]")
```

```
##
## Call:
## lm(formula = as.formula(paste(dep.var, indep.var, sep = "~")),
##     data = entraDados)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.204  -9.199  -0.686   6.823  37.837
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.27958    2.10369   17.72   <2e-16 ***
## Idade        0.99806    0.04695   21.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.33 on 204 degrees of freedom
## Multiple R-squared:  0.689,  Adjusted R-squared:  0.6874
## F-statistic: 451.9 on 1 and 204 DF,  p-value: < 2.2e-16
```



MAGNÍFICO, agora podemos aplicar a função **regressao.boladona()** aos dataframes e especificar variáveis e legendas.

OBS. para que facilite o entendimento do script acima, aplicarei o comando **head()** aos dados lidos.

```
head(dados)
```

```
##   Horas Idade
## 1    48    21
## 2    47    21
## 3    73    18
## 4    65    23
## 5    74    19
## 6    50    19
```

Colocando a função em um arquivo separado (Bônus)

Agora que a função **regressao.boladona()** está finalizada, podemos colocá-la em um script separado e carregá-la no script principal. Sendo assim, os scripts ficarão mais organizados.

Os dois arquivos serão nomeados da seguinte forma:

- fazer_regressao.R (script principal)
- regressao.R (script com a função **regressao.boladona()**)

O script **regressao.R** terá o seguinte conteúdo:

```
require(ggpubr)
require(ggplot2)

regressao.boladona <- function(entraDados,dep.var,indep.var,
                               leg.x="Legenda X",leg.y="Legenda Y"){

modelo <- lm(as.formula(paste(dep.var,indep.var,sep="~")),entraDados)
print(summary(modelo))

theme_set(theme_bw())
ggplot(entraDados,aes_string(x=indep.var,y=dep.var)) +
  geom_point(pch=21,col="blue4",bg="cornflowerblue",size=2.5) +
  geom_smooth(method = lm,col="red",se=FALSE)+
  labs(title = "Gráfico de regressão",
       x = leg.x,
       y = leg.y
       ) +
  stat_regline_equation(aes(label=paste(..eq.label..,..rr.label..,sep="~~")))
}
```

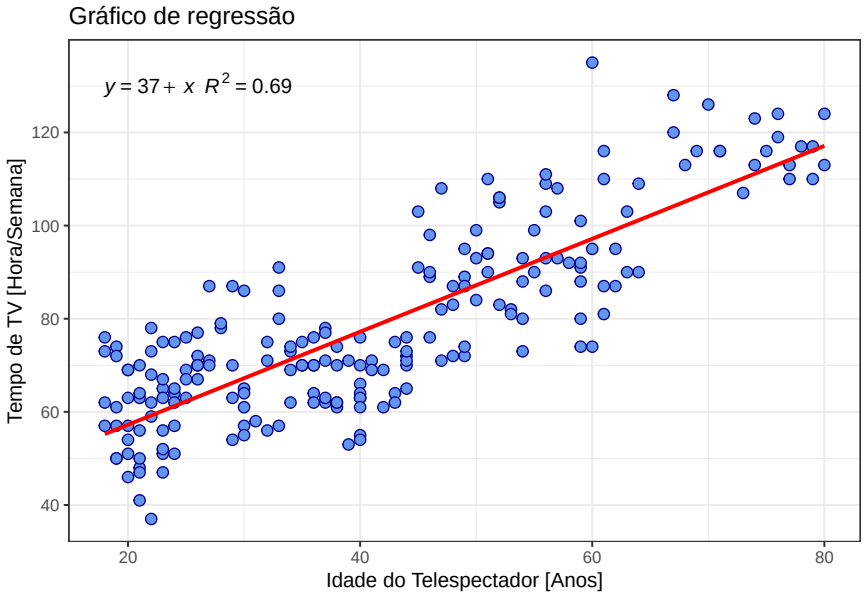
Observe que os pacotes **ggpubr** e **ggplot2** serão carregados dentro do **regressao.R**, uma vez que as funções desses pacotes são usadas nesse script. Isso não é regra, mas deixa o programa mais organizado.

O script **fazer_regressao.R** terá o seguinte conteúdo:

```
rm(list = ls())

require("gdata") # carregando o pacote gdata
```

```r
setwd("/home/mercel/aulas/regressao") # definido o diretório de trabalho
source("regressao.R")

dados <- read.xls("idadeTV.xls",sheet = 1) # lendo os dados

regressao.boladona(dados, "Horas","Idade","Idade do Telespectador [Anos]",
                   "Tempo de TV [Hora/Semana]")
```

```
##
## Call:
## lm(formula = as.formula(paste(dep.var, indep.var, sep = "~")),
##     data = entraDados)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.204  -9.199  -0.686   6.823  37.837
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.27958    2.10369   17.72   <2e-16 ***
## Idade        0.99806    0.04695   21.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.33 on 204 degrees of freedom
## Multiple R-squared:  0.689,  Adjusted R-squared:  0.6874
## F-statistic: 451.9 on 1 and 204 DF,  p-value: < 2.2e-16
```



Observe que, dessa forma, foi possível produzir o mesmo resultado de forma mais organizada. Verifique também, que a função **regressao.boladona()** foi incorporada ao programa principal quando o

script **regressao.R** foi carregado através do comando **source()**. O script **regressao.R** foi salvo no diretório de trabalho definido no script principal (**"/home/mercel/aulas/regressao"**).