MANIPULANDO IMAGENS ATRAVÉS DA REPRESENTAÇÃO MATRICIAL: UMA ABORDAGEM CURIOSA

Lucas Meireles T. de Alvarenga, Luciane Ferreira Alcoforado e Francisco C.S.de Azeredo Pinto

08 de outubro de 2014

Introdução

A proposta desse tutorial é mostrar algumas curiosidades do que o R pode fazer, através da manipulação de imagens que são representadas por uma matriz de cores. Faremos alguns gráficos no R com essas informações. Esse conhecimento de como tratar tais de matrizes podem ser aplitacados também em data frames (banco de dados). Este tutorial foi desenvolvido durante o projeto de monitoria 2014 no departamento de Estatística da UFF. ##Pacotes utilizados

O primeiro pacote que seria utilizado nesse tutorial foi o **biOps**, mas ele acabou por ser descontinuado e posteriormente retiraram-no do CRAN do R. Após esse contratempo, descobrimos novos pacotes:

rimage

```
Também descontinuado
```

EBImage

```
*Para baixar esse pacote temos que usar os seguintes comandos:*
source("http://bioconductor.org/biocLite.R")
biocLite("EBImage")
library("EBImage")
```

ripa *Para baixar esse pacote temos que usar os seguintes comandos:*

```
install.packages("ripa")
```

jpeg *Para baixar esse pacote temos que usar os seguintes comandos:*

```
install.packages("jpeg")
```

Para ler a imagem usamos: readImage(“Nome da imagem com a extensão.jpeg”) Para “plotar” a imagem usamos: display(“Nome do objeto que recebeu a imagem”,method=c(“raster”)) Usamos raster para ele ser plotado direto no R, caso contrário ele é plotado no browser.

```
**ripa** e **jpeg**
```

O jpeg permite ler imagens através do readJPEG(“Nome do arquivo com extensão”), esse pacote também plota a imagem criando um gráfico de dispersão vazio e depois plotando a imagem por cima, da seguitne forma: plot(c(0,10),c(0,10),ylab=“”,xlab=“”,type=“n”,axes=F) rasterImage(“Nome do objeto que recebeu a imagem”,0,0,10,10)

Entretanto, o pacote ripa permite realizar a mesma tarefa de modo mais simples. Para isso usaremos o comando plot(imagematrix(“Nome do objeto que recebeu a imagem”)) do pacote ripa “‘ Agora veremos como esses comando plotam a imagem na prática.

```
#Verifique o diretório onde encontram-se os arquivos de imagem. No nosso caso, a imagem está no arquivo
setwd("C:\\Users\\Luciane\\Documents\\Monitoria 2014\\LucasFinal2014")

library(EBImage)

joker=readImage("joker.jpg")
display(joker,method=c("raster"))
```

```
#Se não tiver trocado o endereço ou reaberto o R, não precisa fazer o passo do setwd() novamente.
setwd("C:\\Users\\Luciane\\Documents\\Monitoria 2014\\LucasFinal2014")
library(jpeg)
joker=readJPEG("joker.jpg")
plot(c(0,10),c(0,10),ylab="",xlab="",type="n",axes=F)
rasterImage(joker,0,0,10,10)
```

```
setwd("C:\\Users\\Luciane\\Documents\\Monitoria 2014\\LucasFinal2014")

library(jpeg)
require(ripa)
```

```
## Loading required package: ripa
## Loading required package: tcltk
## Loading required package: parallel
##
## Attaching package: 'ripa'
##
## The following object is masked from 'package:EBImage':
##
##     normalize
```

```
joker=readJPEG("joker.jpg")
plot(imagematrix(joker))
```

Matrizes

Rode o seguinte comando e observe a informação sobre a dimensão da imagem:

```
imagematrix(joker)
```

```
## size:  540 x 960
## type:  rgb
```

Podemos ver que essas são as dimensões de nosssa imagem e o sistema de coloração usada para determinar as cores. A pergunta agora é, por que vamos trabalhar com matriz se temos uma imagem? A imagem é salva em formato de uma matriz tridimensional, ou seja, cada uma das células da matriz vai conter três valores em vez de um só valor, como é o normal. Em uma matriz normal só temos as linhas e colunas para nos preocuparmos, logo qualquer alteração que quisermos fazer em relação a linhas e colunas é so modificar os valores referentes as linhas e os valores referentes a coluna, como podemos ver no exemplo:

```
#Criando a matriz
matriz=matrix(1:6,nrow=2,ncol=3,byrow=T)

#Rodando a matriz nova
matriz
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```r
#Se eu fixar uma só linha
matriz[1,]
```

```
## [1] 1 2 3
```

```r
#Fixando duas colunas
matriz[,c(1,3)]
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    6
```

```r
#E caso queira alterar um valor específico é só seguir essa ideia
matriz[2,2]=20
matriz
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4   20    6
```

Mas a matriz tridimensional não só tem linha e coluna, ela tem uma terceira opção como podemos ver no exemplo:

```r
matriz3d=array(1:12,dim=c(2,2,3))
matriz3d
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
##
## , , 3
##
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
```

```r
#Fixando a linha e a coluna
matriz3d[1,1,]
```

```
## [1] 1 5 9
```

```
#Essa nova parte varia só entre um, dois ou três, não importando o tamanho das linhas e das colunas
#Então vou fixar cada uma dessas opções, sem fixar a linha ou a coluna
#Terceiro valor como 1
matriz3d[,,1]
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
#Terceiro valor como 2
matriz3d[,,2]
```

```
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
#Terceiro valor como 3
matriz3d[,,3]
```

```
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
```

```
#Toda mudança que podemos fazer na matriz bidimensional podemos fazer também na tridimensional
```

Mas por hora esqueçamos essa nova parte da matriz tridimensional e trabalharemos sem mexer nela, mais para frente eu irei explicar o que são esses valores exatamente. Por enquanto aprenderemos como modificar e mexer nas linhas e colunas da matriz usando essa imagem.

Tratamento das imagens usando matriz

Para começarmos a ter essa noção vamos começar com exercícios simples, o primeiro deles será plotar somente metade da foto, da esquerda para a direita.

```
#Para fazer isso só precisamos saber o tamanho da coluna e pedir para ele desenhar a imagem só até meta
imagematrix(joker)
```

```
## size:  540 x 960
## type:  rgb
```

```
#Podemos ver que de largura a imagem tem 960 pixels
#ou seja queremos que ele plot da coluna 1 até a 480 e podemos fazer isso usando o comando 1:n
#Que o R gera um vetor de valores inteiros do um até o valor do n
plot(imagematrix(joker[,1:(960/2),]))

#Também podemos ter feito de outra forma, para não nos preocuparmos de olhar o tamanho
#Podemos usar o comando ncol(nome da matrix), esse comando já retorna o tamanho das colunas
plot(imagematrix(joker[,1:(ncol(joker)/2),]))
```

Fazer a mesma coisa, mas plotando só a imagem de cima até a metade

```
#Podemos olhar a dimensão de novo
imagematrix(joker)
```

```
## size:  540 x 960
## type:  rgb
```

```
plot(imagematrix(joker[1:540/2,,]))

#Ou usar o comando nrow
plot(imagematrix(joker[1:nrow(joker)/2,,]))
```

Pegando agora somente um quarto da foto, ou seja, metade de cima para baixo e metade da esquerda para a direita

```
plot(imagematrix(joker[1:540/2,1:960/2,]))

plot(imagematrix(joker[1:nrow(joker)/2,1:ncol(joker)/2,]))
```

```
#Ou o último um quarto da imagem, seguindo a mesma idéia, só que pegando da metade até o final
plot(imagematrix(joker[(nrow(joker)/2):nrow(joker),(ncol(joker)/2):ncol(joker),]))
```

Podemos pegar áreas aleatórias da imagem

```
#Close no sorriso

plot(imagematrix(joker[275:450,300:650,]))
```

Podemos alterar a ordem também que a matriz é desenhanda, por exemplo vamos alterar o primeiro exemplo, fazer ele ser plotado da metade até o ínicio

```
#A idéia continua a mesma só que dessa vez não começaremos a criar os números inteiro com o um
#E sim com a coluna que representa metade da foto
plot(imagematrix(joker[,(ncol(joker)/2):1,]))
```

Pegando essa idéia, podemos modificar essa imagem e por exemplo criar uma imagem totalmente simétrica só mudando as colunas ou linhas

```
#O comando c() serve para juntar valores em um mesmo vetor
plot(imagematrix(joker[,c(1:(ncol(joker)/2),(ncol(joker)/2):1),]))
```

```
#Outra forma de construir essa imagem simétrica
plot(imagematrix(joker[,c((ncol(joker)/2):ncol(joker),ncol(joker):(ncol(joker)/2)),]))
```

```
#Ou podemos seguir a idéia anterior mas criar um rosto em vez de deixa as duas metades para fora da ima
plot(imagematrix(joker[,c(ncol(joker):(ncol(joker)/2),(ncol(joker)/2):ncol(joker)),]))
```

Como já aprendemos a mexer com as linhas e colunas, podemos pegar qualquer parte da imagem, no nosso caso, ou poderíamos tratar informações de uma matriz ou de um banco de dados só fazendo esse tipo de mudança.

Retirada de informações de uma matriz

Além de mover informações, também podemos retirar informações dos bancos de dados, ou tirar informações da nossa imagem, que nesse caso seria a mesma coisa que reduzir a qualidade da imagem. Como podemos ver abaixo:

```
#Nesse exemplo eu decidi retirar todas as linhas e colunas pares.
#Para isso vou criar um vetor de todos números pares até o tamanho total da linha e outro vetor seguind
L=seq(2,nrow(joker),by=2)
C=seq(2,ncol(joker),by=2)

#Agora eu simplesmente escrevo o L na parte das linhas e o C na parte das colunas, com um menos na fren
plot(imagematrix(joker[-L,-C,]))
```

```
#Podemos rodar o imagedata para ver qual a nova resolução
imagematrix(joker[-L,-C,])
```

```
## size:  270 x 480
## type:  rgb
```

Inserindo informação em uma matriz

Agora para falarmos dessa parte, primeiro precisamos preparar uma nova imagem e seria aconselhável fazer um novo objeto para ser um backup rápido da primeira imagem.

```
#Lendo a nova imagem
batman=readJPEG('batman.jpg')

#Criando um novo objeto com a imagem antiga
joker1=joker
plot(imagematrix(batman))
```

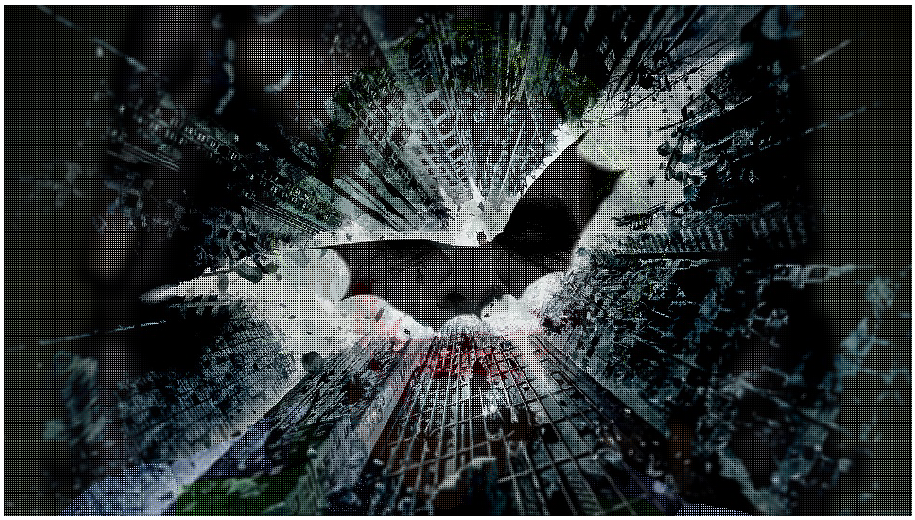
Vamos começar substituindo metade dos valores da primeira imagem com a imagem nova.

```
#Seguimos a mesma ideia de quando retiramos metade da imagem, mas em vez de só plotarmos a imagem vamos
joker1[,(ncol(joker1)/2):ncol(joker1),]=batman[,(ncol(batman)/2):ncol(batman),]
plot(imagematrix(joker1))
```

Fazendo outro exemplo, vamos substituir só as linhas pares de um pelas linhas pares do outro

```
#Restaurando a imagem joker1, para o seu formato original
joker1=joker
#Usando aquele mesmo L e C que já foi feito.
joker1[L,C,]=batman[L,C,]
plot(imagematrix(joker1))
```

```
#Podemos fazer o contrário também
batman1=batman
batman1[L,C,]=joker[L,C,]
plot(imagematrix(batman1))
```



No caso de imagens, não tem nenhuma maneira fácil de adicionar mais dimensões, por exemplo plotar uma imagem lado a lado sem alterar o tamanho das duas imagens antes de juntar. Se fosse um banco de dados ou uma matriz normal, seria só ver o último valor da linha ou da coluna e adicionar o valores.

```
plot(c(0,10),c(0,10),ylab="",xlab="",type="n",axes=F)
rasterImage((joker),0,0,5,10)
rasterImage((batman),5,0,10,10)
```

Cores e a matriz tridimensional

Primeiro vamos falar de cores, no R existe o comando rgb(), que é o esquema de cores usados por muitos programas. mais abaixo podemos ver um detalhamento da função.

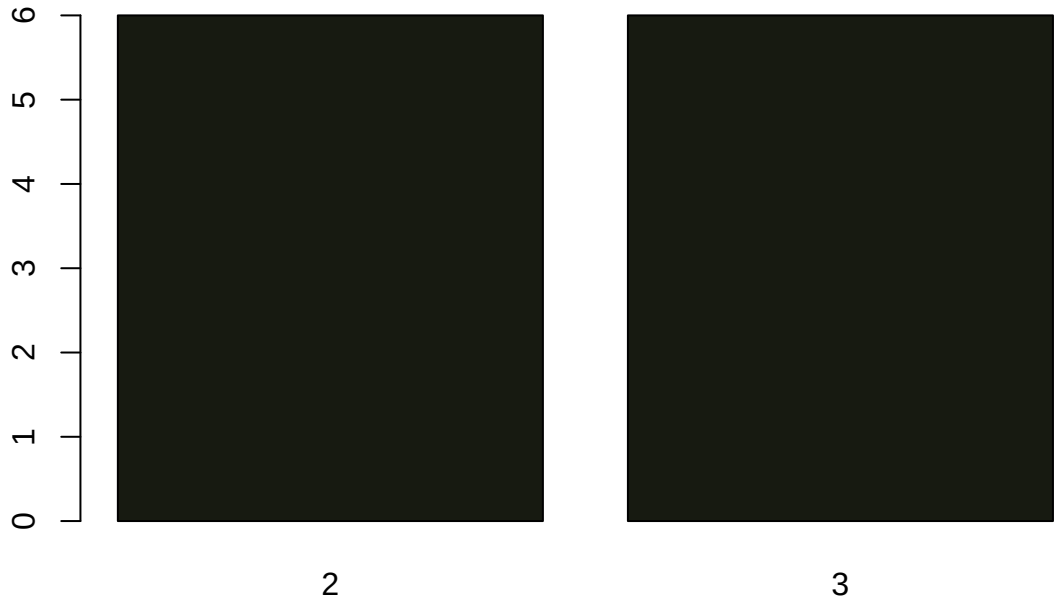
```
rgb(r=,g=,b=,max=,alpha=)

r é o valor referente a quantidade de vermelho que você quer dessa cor (rED), é também um valor que vari

g e b seguem a mesma ideia do r só que g é referente a verde (gREEN) e b é referente ao azul (bLUE)

max é um valor que podemos alterar o quanto os valores do r, g e b variam proporcionalmente ao 0 e o 1.

alpha recebe a transparência da imagem, sendo 1 sem transparência e 0 totalmente transparente. Ele recel
```

Podemos ver abaixo um exemplo de como o rgb pode ser utilziado

```
rgb(23,26,17,max=255)
```

```
## [1] "#171A11"
```

```r
#"#171A11" é o valor que o rgb retorna.
barplot(table(matrix(c(2,2,2,2,2,2,3,3,3,3,3,3),ncol=2)),col="#171A11")
```



Agora falando da nossa matriz tridimensional, temos a parte que é referente a linha, a coluna e uma terceira parte que não usamos até agora que é referente a parte tridimensional, mas por que esse nome? É porque cada linha e cada coluna recebem ao mesmo tempo três informações, como podemos ver abaixo:

```r
joker[1,1,]
```

```
## [1] 0.09019608 0.10196078 0.06666667
```

A primeira informação é a quantidade de vermelho que tem naquela linha e naquela coluna, ou seja, nosso r do rgb; A segunda informação é a quantidade de verde que tem naquela linha e naquela coluna, ou seja, nosso g do rgb; A terceira informação é a quantidade de azul que tem naquela linha e naquela coluna, ou seja, nosso b do rgb.

Sendo assim, faremos alguns exemplos para entender melhor essa relação.

```r
#Trabalhando com nosso backup
joker1=joker

#Trocando toda quantidade de vermelho da imagem por um valor fixo.
joker1[,,1]=0.405
plot(imagematrix(joker1))
```

```
#Trocando toda quantidade de verde da imagem por um valor fixo.
joker2=joker
joker2[,,2]=0.405
plot(imagematrix(joker2))
```

```
#Trocando toda quantidade de azul da imagem por um valor fixo.
joker3=joker
joker3[,,3]=0.405
plot(imagematrix(joker3))
```

Lembrando que o 0 é o preto e o branco é o 1, como a foto é mais escura adicionando um valor fico no vermelho, podemos ver que toda a imagem ficou mais avermelhada e nas outras demonstrações não ficou tão clara a mudança que ocorreu mas podemos ver que toda a imagem reage de uma forma parecida, principalmente porque estamos puxando o que era 1 para baixo e o que era 0 para cima, quase para seu ponto médio.

O que aconteceria se fizermos as mudanças das cores para cores fixas em uma mesma imagem?

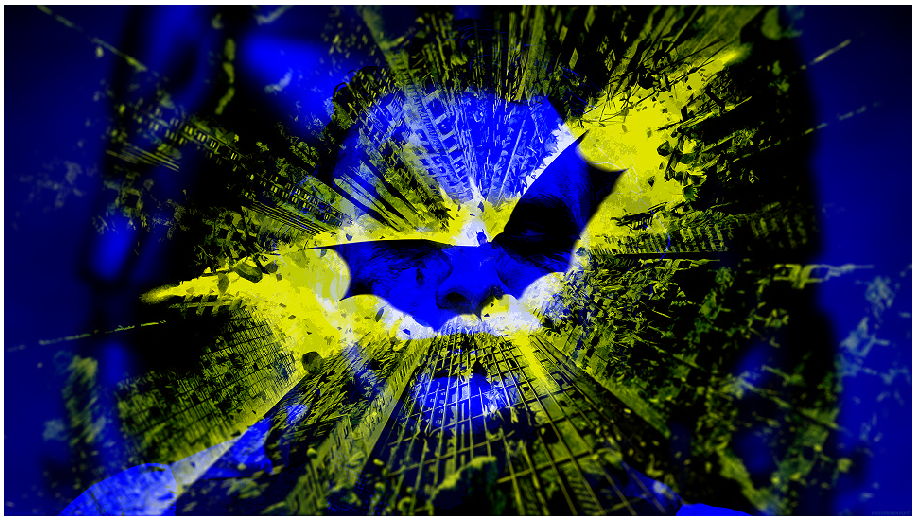
```
joker4=joker
joker4[,,1]=0.125
joker4[,,2]=0.24
joker4[,,3]=0.75
plot(imagematrix(joker4))
```

Como podemos ver ela ficou toda em uma mesma cor, porque todas as suas linhas e colunas estão com o mesmo valor.

Mas em vez de substituir com uma cor fixa, podemos também substituir com as cores de uma outra imagem, no próximo exemplo fazemos isso, primeiro trocando o vermelho, depois o verde e depois o azul na mesma imagem.

```
#Backup
joker5=joker

#Primeira modificação
joker5[,,1]=batman[,,1]
plot(imagematrix(joker5))
```

```
#Segunda modificação
joker5[,,2]=batman[,,2]
plot(imagematrix(joker5))
```



```
#Terceira modificação
joker5[,,3]=batman[,,3]
plot(imagematrix(joker5))
```

Podemos ver como a primeira imagem vai assumindo as cores da segunda, poderiamos também trocar as ordem, ou simplesmente só fazer uma modificação.

Como podemos reconhecer as cores e se o dejeso fosse trocar alguma cor específica? Nesse caso terá que usar um pouco de programação, para que o programa veja qual é a cor daquela posição e se for a cor que você quer substituir para faze essa mudança, no exemplo a ideia será mudar a cor preta pela cor vermelha.

```
#A cor que eu vou querer modificar, será o preto.
#Nosso Backup
batman1=batman

#Essa linha é para que ele passe por todas as linhas da imagem
for(i in 1:nrow(batman1)){
  #Essa linha é para que ele passe por todas colunas da nossa imagem, a cada linha que ele passar.
  for(j in 1:ncol(batman1)){
    #Essa é a nossa pergunta, se a cor naquele ponto é preta ou não
    if(batman1[i,j,1]==0&batman1[i,j,2]==0&batman1[i,j,3]==0){
      #Caso seja acontece essa modificação para o vermelho
      batman1[i,j,1]=1
      batman1[i,j,2]=0
      batman1[i,j,3]=0
    }
  }
}
plot(imagematrix(batman1))
```

Não ficou uma coisa bonita, mas é válida para qualquer cor ou até mesmo range de cores, por exemplo pegar todas cores com valores menores do que 0.1, só seria preciso trocar o ==0 por <=0.1.

Tabela de frequência e gráficos

Primeiro vamos pegar cada nome das cores de cada uma das posições.

```
#Uma matriz vazia com o mesmo tamanho de linhas e de colunas que a nossa imagem
#OBS: não precisa ser tridimensional, pois só irá receber o nome da cor por cada linha e coluna, ou sej
mt=matrix(0,ncol=length(joker[1,,1]),nrow=length(joker[,1,1]))

#Essa linha é para que ele passe por todas as linhas da imagem
for(i in 1:length(joker[,1,1])){
  #Essa linha é para que ele passe por todas colunas da nossa imagem, a cada linha que ele passar.
  for(j in 1:length(joker[1,,1])){
    #O rgb está recebendo exatamente a cor respectiva ao seu r, g e b
    mt[i,j]=rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])
}}
#Peguei só uma amostra de 20 linhas e 20 colunas, porque seria absurda a quantidade de
table(mt[1:20,1:20])
```

```
##
## #171A11 #181B12 #181D16 #191C13 #191E17 #1A1D14 #1A1F18 #1B1E15 #1B2019
##       8      15       5      34      14      30      33      33      22
```

```
## #1C1F16 #1C211A #1C2218 #1D2017 #1D221B #1D2319 #1E2118 #1E231C #1E241A
##      23      30       4      10      25       9       7      34       8
## #1F241D #1F251B #20251E #21261F #222720
##      29       2      13       8       4
```

Essa tabela de frequência não é muito útil, pois traz muita (MUITA!) informação com poucas repetições de valores, porque as combinações de cores são quase infinitas. Veremos agora como agrupar essas cores em grupos para podermos contruir uma tabela mais fácil de ser observada, usando um pouco mais de programação.

```
#Variável auxiliar que vai ser utilziada como contador
cont=1

#Um vetor para receber o nome das cores, de tamanho 960*540, ou seja, todas as cores possiveis.
cores=c(rep(0,960*540))

for(i in 1:length(joker[,1,1])){
  for(j in 1:length(joker[1,,1])){
    #Verificando se a cor é puramente vermelha
    if(rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])==rgb(1,0,0,)){
      #Caso seja vermelho o vetor cores vai receber a palavra vermelho naquela posição do contador
      cores[cont]="Vermelho"
    }
    else{
      #Verificando se a cor é puramente verde, caso ela não tenha sido vermelha
    if(rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])==rgb(0,1,0)){
      #Caso seja verde o vetor cores vai receber a palavra Verde naquela posição do contador
      cores[cont]="Verde"
    }
    else{
      #Verificando se a cor é puramente azul, caso ela não tenha sido vermelha ou verde
    if(rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])==rgb(0,0,1)){
      #Caso seja vermelho o vetor cores vai receber a palavra Azul naquela posição do contador
      cores[cont]="Azul"
    }
    else{
      #Verificando se a cor é puramente branca, caso ela não tenha sido vermelha ou verde ou azul
    if(rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])==rgb(1,1,1)){
      #Caso seja branca o vetor cores vai receber a palavra Branca naquela posição do contador
      cores[cont]="Branca"
    }
    else{
      #Verificando se a cor é puramente preta, caso ela não tenha sido vermelha ou verde ou azul ou bra
    if(rgb(joker[i,j,1],joker[i,j,2],joker[i,j,3])==rgb(0,0,0)){
      #Caso seja preto o vetor cores vai receber a palavra Preto naquela posição do contador
      cores[cont]="Preto"
    }
    else{
      #Caso ela não tinha sido nenhuma cor, ela vai receber Outras.
      cores[cont]="Outras"
    }}}}}
    cont=cont+1

    #Parar a porgramação, caso passe o número máximo correto de possíveis cores
```

```
    if(cont==960*540+1){
      break
    }
  }
}

(table(cores))
```

```
## cores
## Branca Outras  Preto
##   8966 380005 129429
```

Quando olhamos a tabela, vemos que só tem três cores, preto, branco e outras, isso se dá porque pegando valores exatatos, ou seja, embora tenha vermelho na imagem não necessariamente é o vermelho rgb(1,0,0), logo precisamos aumentar nossa tolerância quanto ao grupo das cores.

```
#Boa parte dessa programação segue a ideia do exemplo anterior, só mudando que uxamos ">=" ou "<=" em v
#Os valores que representam os grupos das cores, foram escolhidos por critério pessoal.

cont2=1
cores2=c(rep(0,960*540))


for(i in 1:length(joker[,1,1])){
  for(j in 1:length(joker[1,,1])){
    if(joker[i,j,1]>=0.8519608 &joker[i,j,2]<=0.098431375 &joker[i,j,3]<=0.09843137){
      cores2[cont2]="Vermelho"
    }
    else{
    if(joker[i,j,1]<=0.12843137 &joker[i,j,2]>=0.8519608 &joker[i,j,3]<=0.12843137){
      cores2[cont2]="Verde"
    }
    else{
    if(joker[i,j,1]<=0.09843137 &joker[i,j,2]<=0.098431375 &joker[i,j,3]>=0.8519608){
      cores2[cont2]="Azul"
    }
    else{
    if(joker[i,j,1]>=0.9019608&joker[i,j,2]>=0.9019608&joker[i,j,3]>=0.9019608){
      cores2[cont2]="Branco"
    }
    else{
    if(joker[i,j,1]<=0.07843137&joker[i,j,2]<=0.07843137&joker[i,j,3]<=0.07843137){
      cores2[cont2]="Preto"
    }
    else{
      cores2[cont2]="Outras"
    }}}}}
    cont2=cont2+1
    if(cont2==960*540+1){
      break
    }
  }
}
```

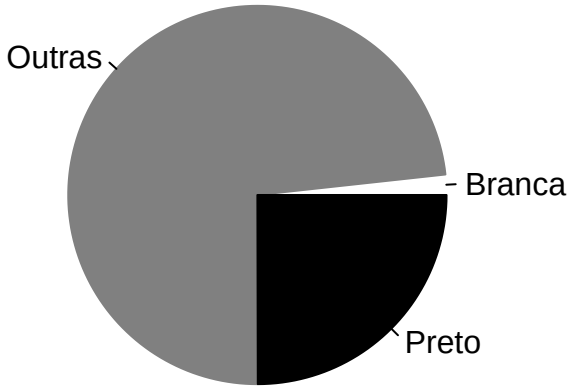
```r
table(cores2)
```

```
## cores2
##   Branco   Outras    Preto Vermelho
##    26010   283089   209265       36
```

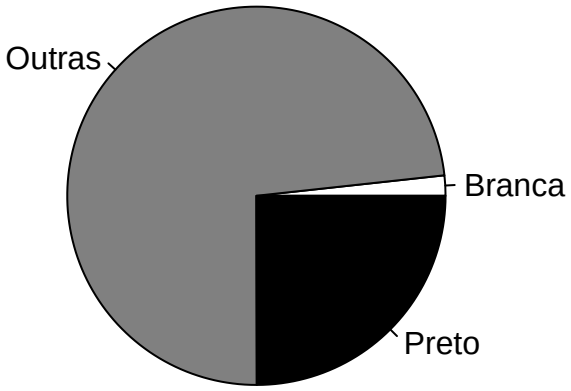
Para deixar essa tabela mais fácil de ser representada em uma apresentação, podemos fazer gráficos e os dois gráficos que desenham essa tabela corretamente é o gráfico de pizza e o gráfico de barras. Faremos exemplos com as nossas duas tabelas.

Primeiro o gráfico de Pizza:

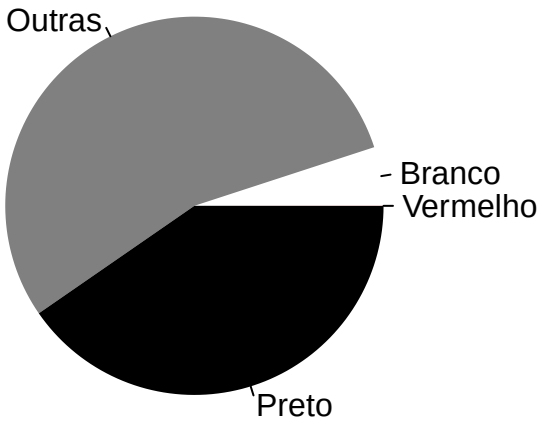
```r
#pie é o comando utilziado para construir gráfico de pizza
pie(table(cores),labels=names(table(cores)),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0)),
    border=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0)))
```




```r
#prop.table está sendo utilizado para fazer um gráfico usando frequência relativa
pie(prop.table(table(cores)),labels=names(prop.table(table(cores))),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rg
```




```
#Cores 2
pie(table(cores2),labels=names(table(cores2)),
    col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0),rgb(1,0,0)),
    border=NA)
```

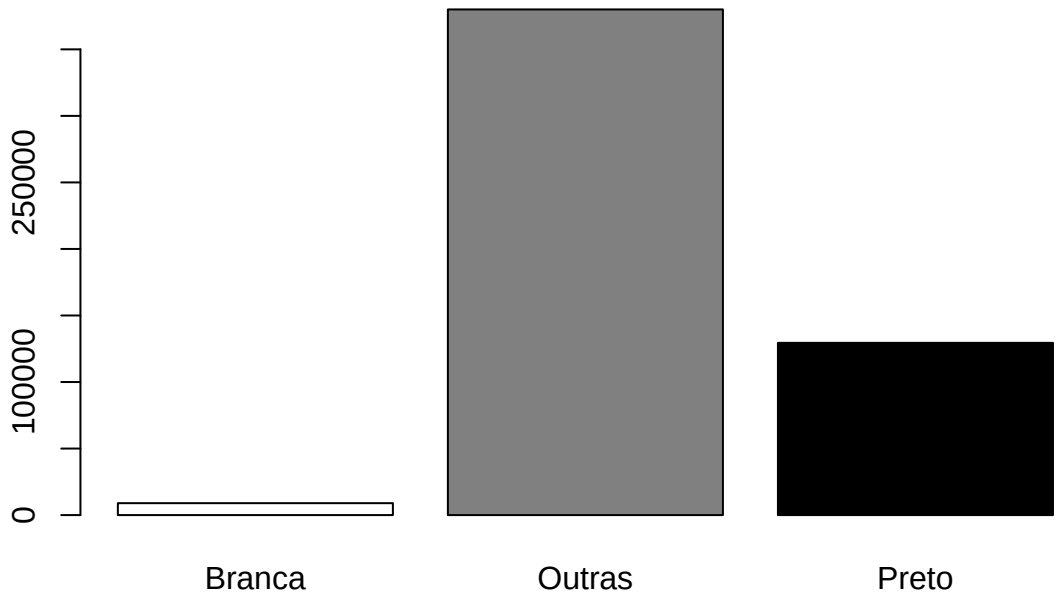


O que significa cada comando utilizado para fazer o gráfico de pizza?

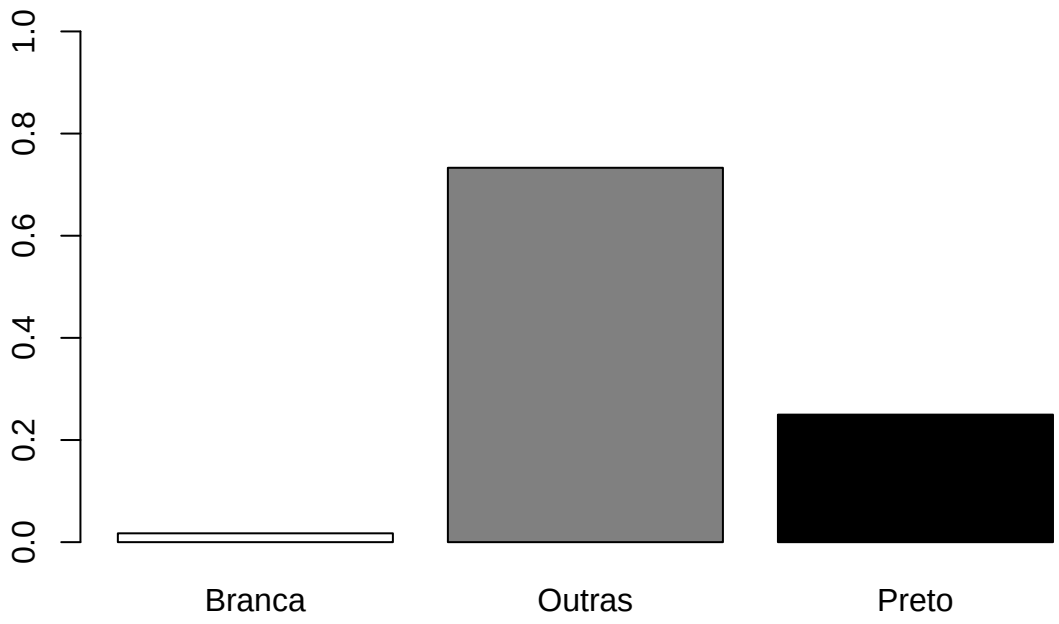
```
Pie é o comando para fazer o gráfico de pizza;
Na primeira parte nos inserimos nossa tabela na função;
Labels é a função que recebe o nome que queremos para nossas variáveis;
names(table(cores)) estou usando para pegar o nome da tabela na ordem certa;
col recebe a cor de cada pedaço da pizza;
border recebe a cor de cada borda de cada pedaço da pizza.
```

Podemos representar a mesma tabela usando o barplot:

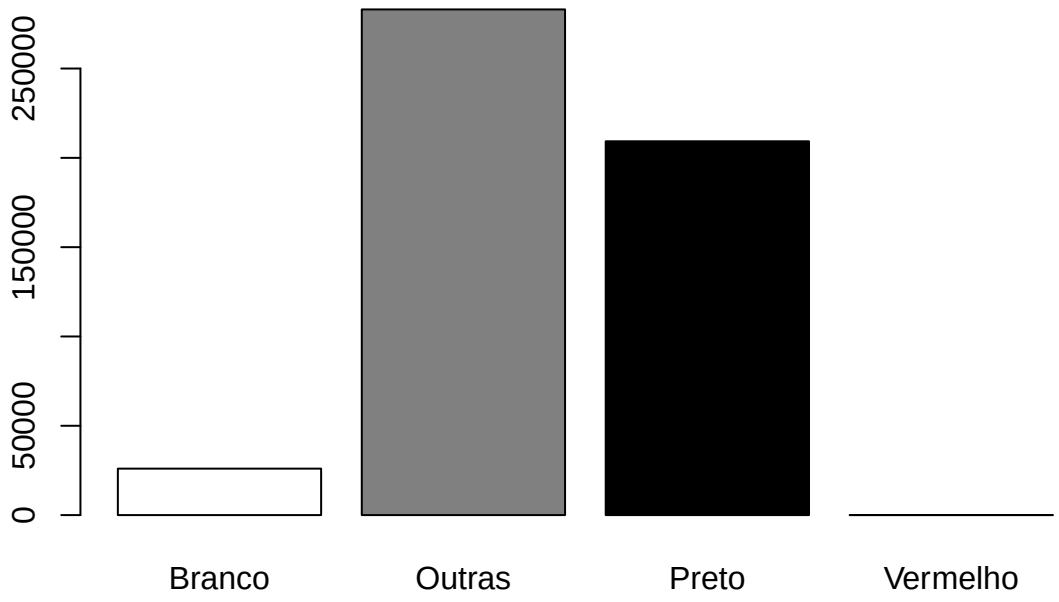
```r
#barplot é o comando utilizado para fazer gráficos de barras
barplot(table(cores),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0)))
```



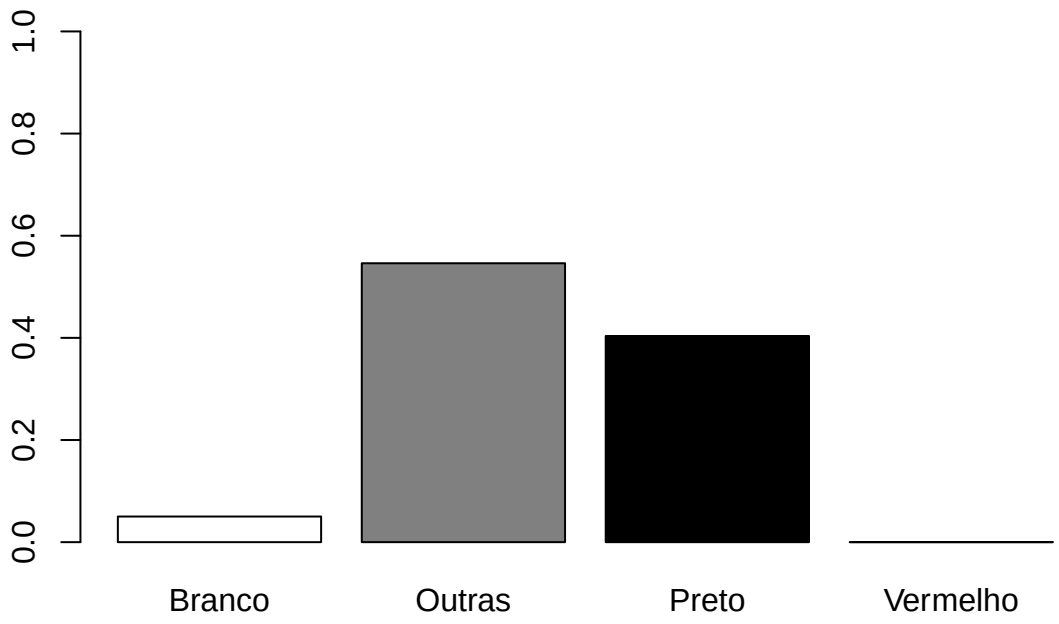

```
barplot(prop.table(table(cores)),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0)),ylim=c(0,1))
```




```
barplot(table(cores2),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0),rgb(1,0,0)))
```



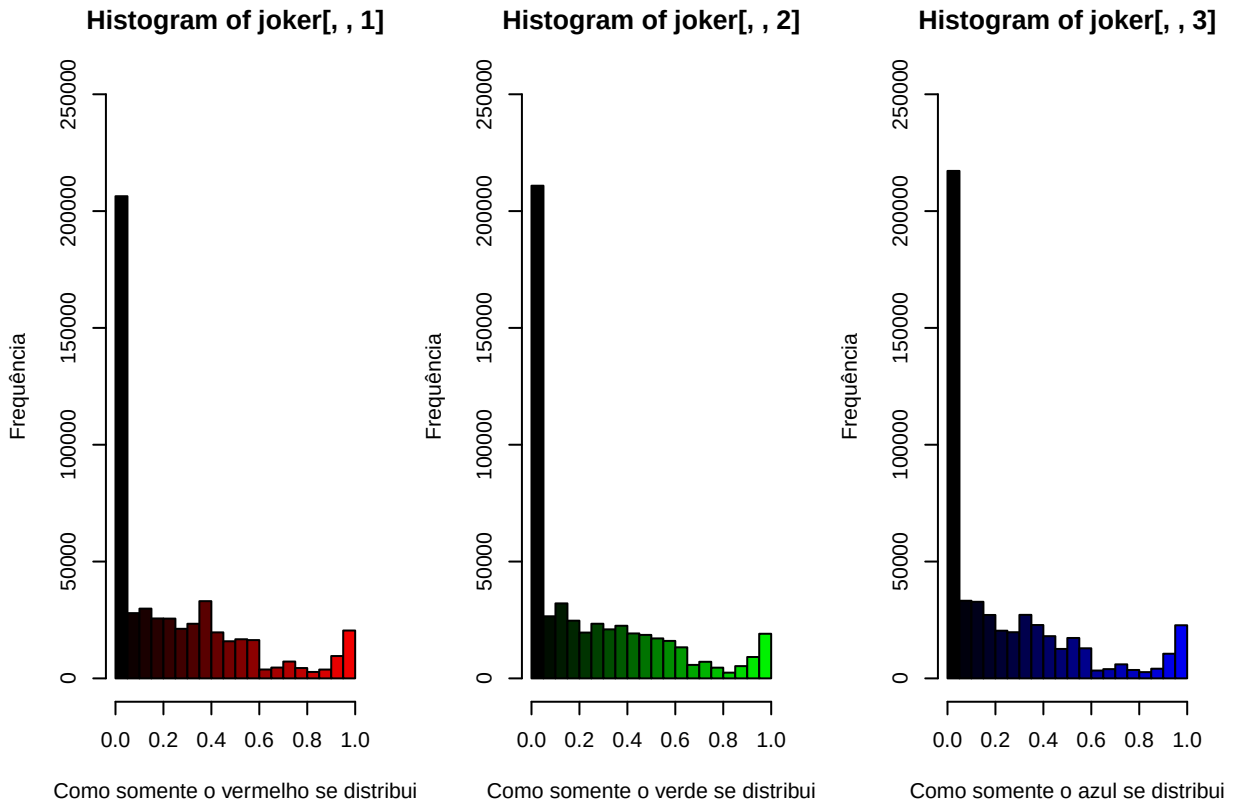

```
barplot(prop.table(table(cores2)),col=c(rgb(1,1,1),rgb(0.5,0.5,0.5),rgb(0,0,0),rgb(1,0,0)),ylim=c(0,1))
```



Os comandos são parecidos com os do gráfico de pizza, só muda o ylim que é usado para definir o limite do eixo y que você quer que apareça.

Esses dois gráficos trabalham com variáveis qualitativas e se quisessemos trabalhar com o gráfico quantativamente? Teríamos que esquecer essa divisão de cores que usamos para poder trabalhar com o gráfico quantativamente e usariamos outro tipo de gráfico, como o histograma. Então vamos usar esse gráfico para ver como o r, g e o b estão distribuídos nessa imagem. Lembrado que esse gráfico não vai levar em conta somente cada cor separada, por isso vai parecer diferente da tabela que fizemos anteriormente, pois ela trata todas cores como um conjunto do r, g e b.

```
par(mfrow=c(1,3))

#hist serve para fazer o histograma
hist(joker[,,1],col=rgb(seq(0,1,0.05),0,0),xlab="Como somente o vermelho se distribui",ylab="Frequência
hist(joker[,,2],col=rgb(0,seq(0,1,0.05),0),xlab="Como somente o verde se distribui",ylab="Frequência",yl
hist(joker[,,3],col=rgb(0,0,seq(0,1,0.05)),xlab="Como somente o azul se distribui",ylab="Frequência",yli
```



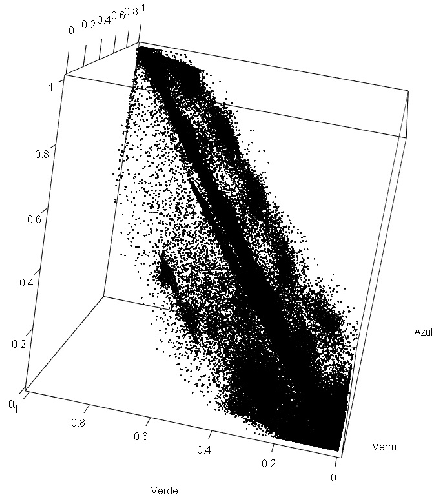

```
par(mfrow=c(1,1))
```

Podemos observar que a maior concentração em todos os gráficos está perto de 0, sendo assim, vemos que a imagem deve ser mais escuro, o que realmente é verdade. Agora vamos explicar os comandos utlizados nessa função.

```
par(mfrow=()) é para desenhar mais de uma imagem no mesmo plot;
xlab é o rótulo do eixo X
ylab é o rótulo do eixo y
```

Essa última linha de comando, não sai o desenho no formato do tutorial porque é um caso particular do R, ele cria uma janela a parte toda vez que é rodado esse comando, logo ele não é plotado dentro do tutorial, mas eu tirei uma printScreen dele e li com os readJPEG, esse gráfico não passa de um gráfico de dispersão mas feito para trÊs valores, em vez de dois que é o normal.

```
library(rgl)
plot3d(joker[,,1],joker[,,2],joker[,,3],xlab="Verm",ylab="Verde",zlab="Azul")
```



Fim do primeiro tutorial!