Trabalho de AEDV - R

Zuilho Segundo

Trabalho desenvolvido para
Ciência de Dados e I.A. - 1° Período

Escola de Matemática Aplicada
Fundação Getúlio Vargas
Brasil

Maio 2022

Sumário

1 Funções Base R

A primeira parte do trabalho consistiu na instalação do RStudio e aprender os usos básicos de R. O testes inicial consistiu do clássico de programação, "Hello World!". [1]

```
print("Hello World!")
```

Um próximo passo foi aprender a utilizar o comando source() e o setwd() para executar scripts a partir do terminal.

```
1 setwd("C:/Users/B46838/OneDrive - Fundacao Getulio Vargas -
  ↪ FGV/Área de Trabalho/R Tentativa")
2 source("HelloWorld.R")
```

Em seguida, foi importante aprender sobre as atribuições e criação de dados no R, para que fosse possível manipular os dados que seriam recebidos mais tarde. Nessa parte criamos vetores utiizando diferentes funções e aprendemos a calcular medidas de resumo aprendidas nos momentos inicias da matéria. Com os dados prontos, um exemplo de medidas de resumo podem ser vistas em seguida:

```
media <- mean(v5)
mediana <- median(v5)
desvpad <- sd(v5)
quartis <- summary(v5)
```

Além disso aprendemos a criar gráficos utilizando as funções base do R.

```
boxplot(v5)
hist(v5)
```

Os gráficos gerados são os seguintes:

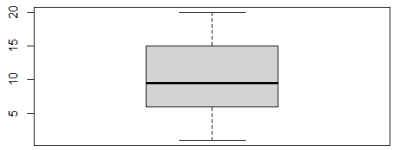

Figura 1: boxplot.png

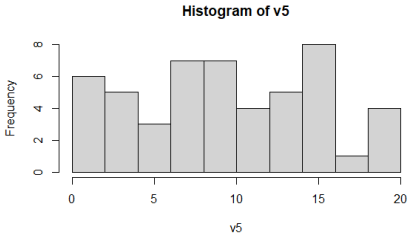

Figura 2: histograma.png

Por fim, nesse primeiro momento, importei o banco de dados mtcars e o .csv pesq_prof, para que pudesse aprender a manipular uma base de dados que já estivesse construída. Trabalhando com a biblioteca fiz medidas de resumo, aprendi a acessar colunas específicas e como plotar os dados necessários em gráficos, além de formas de descobrir os dados existentes no banco de dados.

[1] Os arquivos utilizados na primeira parte são, HelloWorld.R, script.R, funcoes_bases.R, base_dados.R, nessa ordem

```r
#trabalhando com a mtcars
summary(mtcars$disp)
plot(mtcars$wt, mtcars\begin{}{}$disp)
#trabalhando com o .csv
pesq_prof <- read.csv("pesq_prof.csv")
names(pesq_prof)
summary(pesq_prof$Alunos)
```

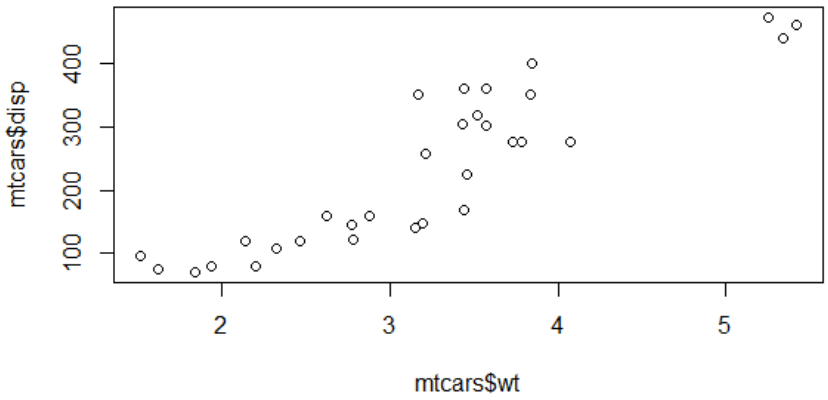

Figura 3: mtcarsplot.png

2 Utilizando ggplot2

A primeira coisa a se fazer nesse segundo momento foi importar a base de dados pesq_prof.csv e a biblioteca do ggplot2. [2]

```
library(ggplot2)
pesq_prof <- read.csv("pesq_prof.csv")
```

Em seguida criamos o primeiro gráfico utilizando geom_histogram para criar um gráfico de barras. Uma coisa que gostei de fazer foi sempre associar o gráfico a uma variável e mandar imprimir depois, fiz isso desde o primero momento. A parte interessante sobre o ggplot é a possibilidade de criar diversas camadas que vão modificando o gráfico desejado.

```
ggplot(pesq_prof, aes(x=Area)) + geom_histogram(stat='count')
graf <- ggplot(pesq_prof, aes(x=?rea)) + geom_histogram(stat='count')
graf2 <- graf + geom_histogram(data=subset(pesq_prof,?rea=="Geometria"),
                               stat="count", fill="darkblue")
print(graf2)
```

O gráfico gerado foi esse:

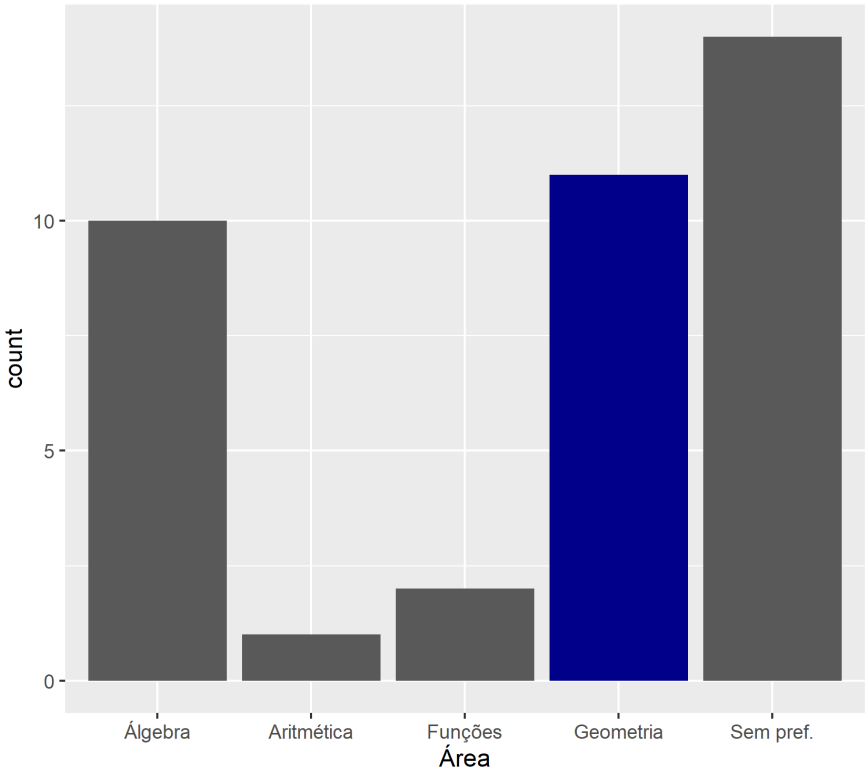

Figura 4: fig3-1.png

[2] O arquivo utilizado aqui foi o ggplot_usos.R

Como eu tinha lido um pouco sobre o R antes, eu utilizei o comando ggsave() para salvar todos os gráficos gerados. Vou colocar aqui apenas uma vez, mas utilizei ele em todo o script.

```
ggsave("fig3-1.png",device="png",dpi = 300)
```

Em seguida me aproveitei da funcionalidade de camadas para gerar um gráfico de altura e peso usando a base de dados pes_prof.csv. Não colocarei todos os graficos gerados, apenas o resultado após a aplicação de todas as camadas.

```
#Organizacaoo por camadas do R
ggplot(pesq_prof, aes(Peso, Altura)) + geom_point()
ggplot(pesq_prof, aes(Peso, Altura, colour=Sexo)) + geom_point()
ggplot(pesq_prof, aes(Peso, Altura, colour=Sexo, size=Peso)) +
↪  geom_point()
graf_disp <- ggplot(pesq_prof, aes(Peso, Altura, colour=Sexo)) +
↪  geom_point() + geom_smooth(method = lm)
print(graf_disp)
graf + xlab("legenda em x - Peso") + ylab("legenda em y -
↪  Altura") + xlim(0,110) + ylim(0, 200)
```

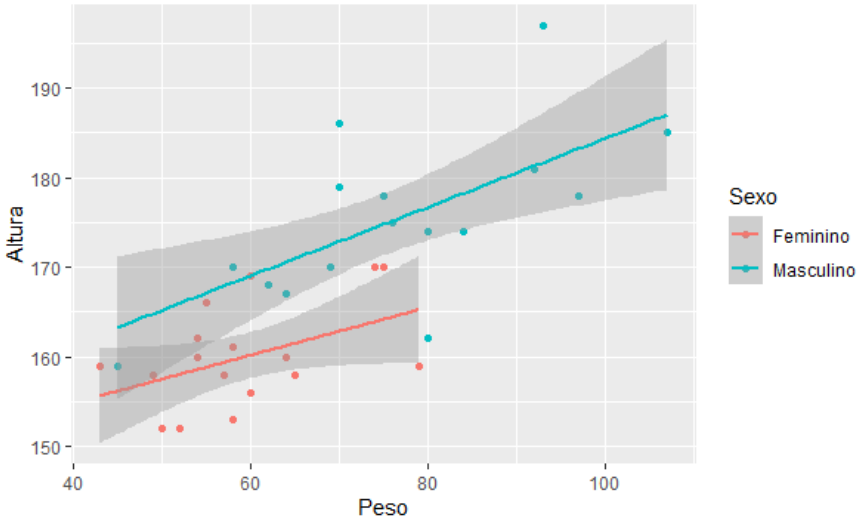

Figura 5: ggplot_plot.png

3 Utilizando o Gapminder

Aqui exploramos as possibilidades biblioteca tidyverse e gapminder para nos aprofundarmos na vizualização. [3]

```
1  library("gapminder")
2  library("ggplot2")
3  #aprendendo com a documentação do gapminder
4  graf <- ggplot(gapminder, aes(x = continent, y = lifeExp)) +
   ↪  geom_boxplot(outlier.colour = "hotpink")+geom_jitter(position
   ↪  = position_jitter(width = 0.1, height = 0), alpha = 1/4)
```

As primeiras linhas de código servem para chamar as bibliotecas e plotar gráficos utilizando os dados do gapminder.

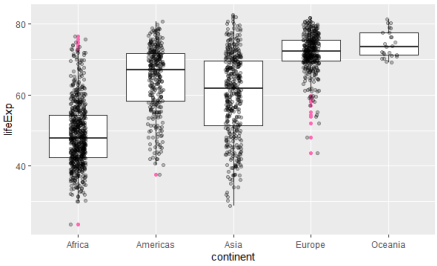

Figura 6: gapminder_teste.png

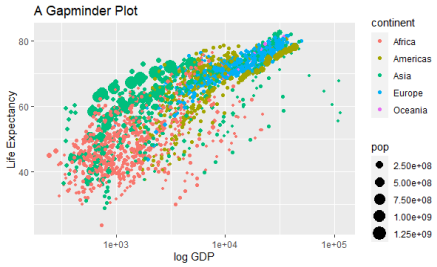

Figura 7: gapminder_teste2.png

Após algumas brincadeira que podem ser vistas no script, aprendi a utilizar as funções geom_point, geom_smooth, e utilizar cores como parâmetros dos gráficos produzidos. A seguir podem ser vistos o código final e as diferentes versões conforme foram se acrescentando camadas.

```
1  h <- ggplot(data = gapminder, mapping = aes(x = gdpPercap, y =
   ↪  lifeExp, color = "purple"))
2  h <- h + geom_point() + geom_smooth(method = "loess") +
   ↪  scale_x_log10()
3  print(h)
```

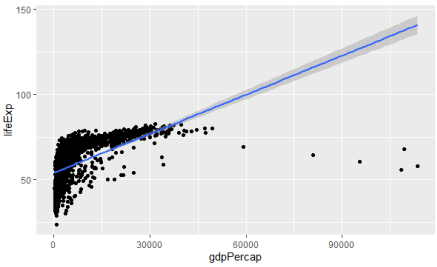

Figura 8: funcoe1.png

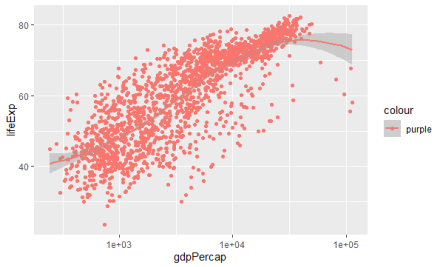

Figura 9: funcoes2.png

[3] O arquivo utilizado nessa seção é o gapminder.R

Após mais algumas modificações, onde modiquei a forma dos marcadores, as legendas, os eixos e as cores utilizadas no gráfico (sempre me aproveitando da função de camadas e salvando os gráficos utilizando variáveis que podiam ser facilmente recuperadas depois), temos uma versão mais apresentável do gráfico.

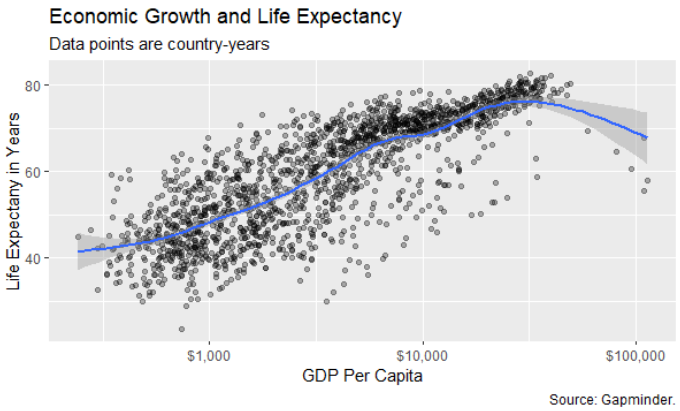

Figura 10: gapminder_1.png

Algumas outras vizualizações e modificações podem ser vistas no script, mas se resumem a mudanças nas cores e formas de expressar marcas gráficas no gráfico.

Uma que gostei e acho interessante colocar aqui, é utilizar cores para representar a variação numérica.

```
1    #Mapping Aesthetics in geom
2    h <- ggplot(data = gapminder, mapping = aes(x = gdpPercap, y =
     ↪  lifeExp))
3    h <- h + geom_point(mapping = aes(color = log(pop))) +
4    scale_x_log10()
5    print(h)
```

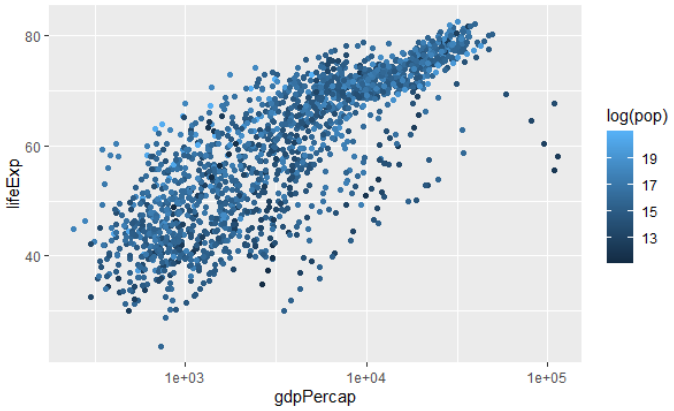

Figura 11: saved.png

4 Minard meets ggplot

4.1 Minard original

Nessa penúltima sessão foi proposto reproduzir os gráficos de Minard utilizando o R e o ggplot. [4]

Após explorar um pouco os dados, foi feito o primeiro gráfico utilizando o geom_path() e como parâmetro da espessura da linha, o número de sobreviventes.

```
1       minard <- ggplot(Minard.troops, aes(long, lat)) +
2       geom_path(aes(size = survivors))
```

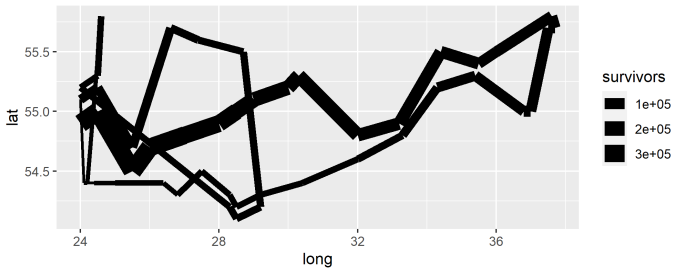

Figura 12: minard_1.png

Melhorando o gráfico para que fique mais similar ao de Minard, passamos a utilizar cores para indicar a direção de movimento das tropas.

```
1       minard <- ggplot(Minard.troops, aes(long, lat)) +
2       geom_path(aes(size = survivors, colour = direction, group =
        ↪  group)) +
3       coord_fixed()
```

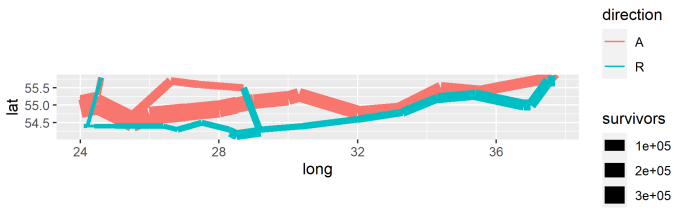

Figura 13: minard_2.png

[4] O script utiizado nessa seção é o minard.R

Nesse momento foram propostos alguns testes para serem realizados com os gráficos, que serão apresentados a seguir.

1 - Não utilizar espeessura no gráfico do path:

```
minard <- ggplot(Minard.troops, aes(long, lat)) +
geom_path() +
coord_fixed()
```

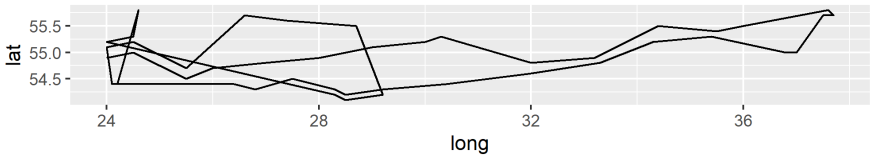

Figura 14: minard_path.png

2 - Utilizar geompoint para mostrar o número de sobreviventes:

```
minard <- ggplot(Minard.troops, aes(long, lat)) +
geom_path() +
coord_fixed() +
geom_point(aes(size=survivors))
```

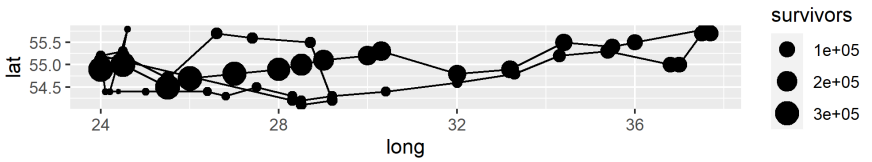

Figura 15: minard_geompoint.png

3 - Utilizar geompoint para mostrar o número de sobreviventes e a cor para o caminho:

```
minard <- ggplot(Minard.troops, aes(long, lat)) +
geom_path() +
coord_fixed() +
geom_point(aes(size=survivors, color=direction))
```

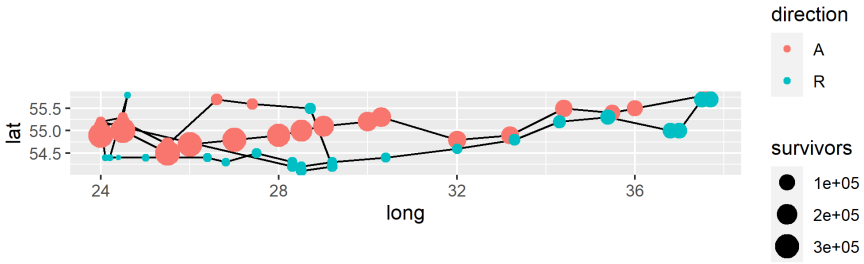

Figura 16: minard_geompoint_color.png

Depois dessa brincadeira com as formas de se utilizar o ggplot, continuamos desenvolvendo o gráfico para que ficasse mais parecido do o de Minard.

```
breaks <- c(1, 2, 3) * 10^5
minard <- ggplot(Minard.troops, aes(long, lat)) +
geom_path(aes(size = survivors, colour = direction, group =
↪  group),
        lineend="round") +
scale_size("Sobreviventes", range = c(1,10), #c(0.5, 15),
         breaks=breaks, labels=scales::comma(breaks)) +
scale_color_manual("Direção",
                   values = c("#e6cbad", "#1f1a1b"),
                   labels = c("Avanço", "Retirada"))
```

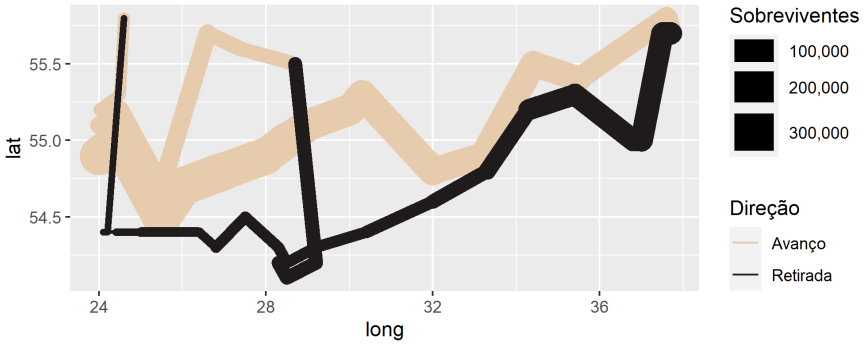

Figura 17: minard_3.png

Mais algumas mudanças foram feitas para modificar a aparência, como remover as legendas, modificar as escalas e o tema de fundo do gráfico. Essas mudanças podem ser vistas a seguir.

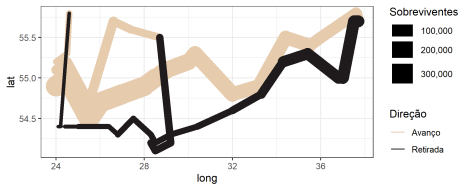

Figura 18: minard_4.png

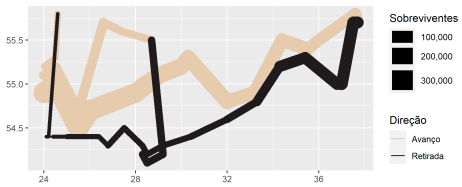

Figura 19: minard_5.png

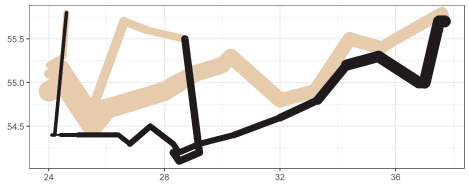

Figura 20: minard_6.png

Agora continuando as mudanças, vamos inserir os nomes das cidades no gráfico, conforme as tropas avançam.

```
bgraf <- plot_troops +
geom_point(data = Minard.cities) +
geom_text_repel(data = Minard.cities, aes(label = city))
print(graf)
```

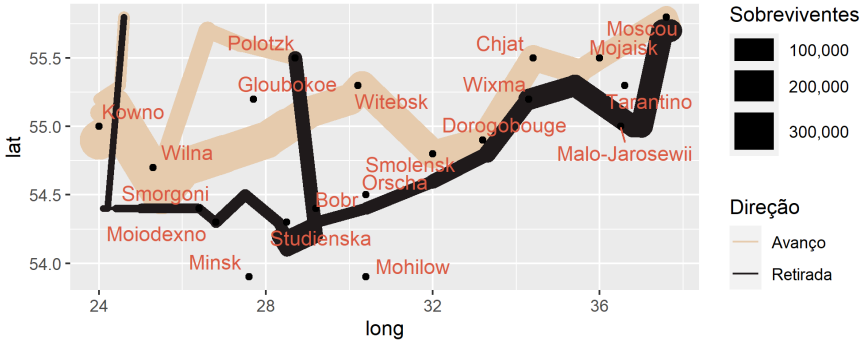

Figura 21: minard_names_adjusted.png

Um outro gráfico de Minard que trabalhamos foi a temperatura encontrada pelos soldados a cada avanço. Nesse momento, o site utilizou o comando pipe $> \% >$ para criar as labels para serem apresentadas no gráfico, o que me fez pesquisar um pouco para entender melhor o que ele fazia. O código do gráfico e o gráfico ficaram assim:

```
Minard.temp <- Minard.temp %>%
mutate(label = paste0(temp, "° ", date))
head(Minard.temp$label)
temp <- ggplot(Minard.temp, aes(long, temp)) +
geom_path(color="grey", size=1.5) +
geom_point(size=1) +
geom_text(aes(label=label), size=2, vjust=-1)
print(temp)
```

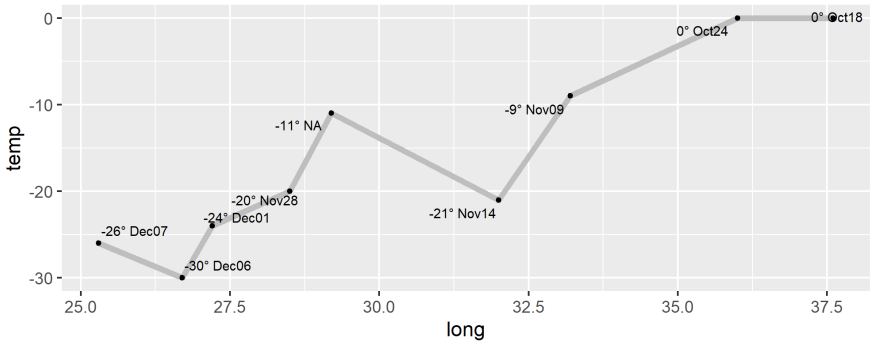

Figura 22: minard_temp2_adjusted.png

Por fim, finalizei juntando os dois gráficos e exportando. Precisei procurar um comando novo para exportar os gráficos juntos, o arrangeGrob(), que retorna a junção de dois gráficos.

```
plot_troops_cities +
coord_cartesian(xlim = c(24, 38)) +
labs(x = NULL, y = NULL) +
guides(color = FALSE, size = FALSE) +
theme_void()

plot_troops_cities_fixed <- last_plot()

plot_temp +
coord_cartesian(xlim = c(24, 38)) +
labs(x = NULL, y="Temperature") +
theme_bw() +
theme(panel.grid.major.x = element_blank(),
    panel.grid.minor.x = element_blank(),
    panel.grid.minor.y = element_blank(),
    axis.text.x = element_blank(), axis.ticks = element_blank(),
    panel.border = element_blank())

plot_temp_fixed <- last_plot()

grid.arrange(plot_troops_cities_fixed, plot_temp_fixed, nrow=2,
↪  heights=c(3.5, 1.2))
```

```
    grid.rect(width = .99, height = .99, gp = gpar(lwd = 2, col =
    ↪  "gray", fill = NA))

    junto <- arrangeGrob(plot_troops_cities_fixed, plot_temp_fixed)
    print(junto)
```

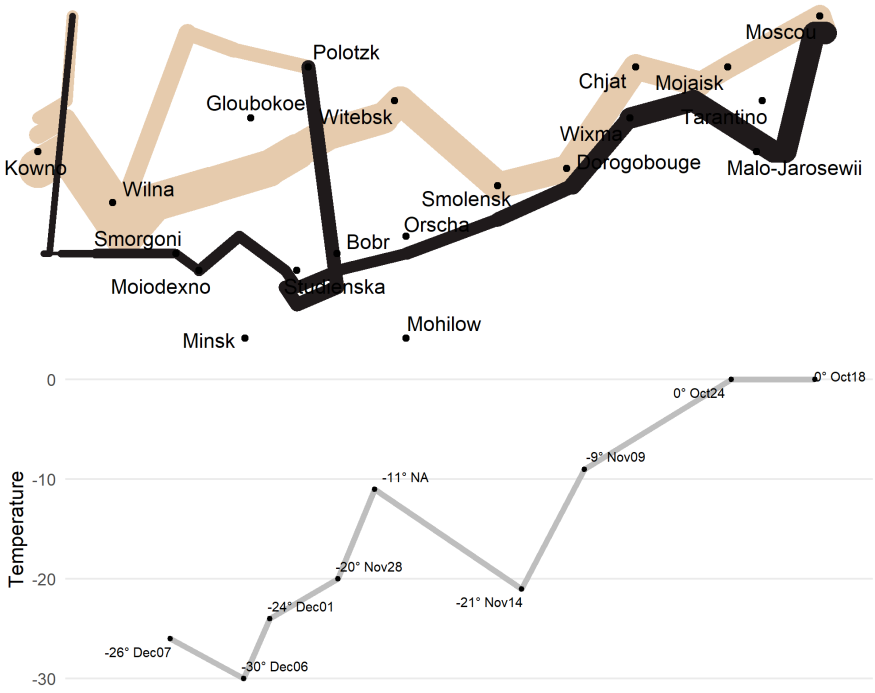

Figura 23: minard_temp_troops_fixed.png

4.2 Modificações

Aqui fiz uma modificação no gráfico utilizando o ggmap. Para fazer isso foi necessário acessar os dados atráves de outra fonte, um arquivo .txt.

```
    troops <- read.table("minard/troops.txt",
                         header = TRUE, stringsAsFactors = FALSE)
    cities <- read.table("minard/cities.txt",
                         header = TRUE, stringsAsFactors = FALSE)
    troops %>% head() %>% pandoc.table()
```

Para aprender a usar, fiz alguns testes, como por exemplo, mostrar o mapa da Europa utilizando o ggmap.

```
    #Mapa da Europa
    march.1812.europe <- c(left = -13.10, bottom = 35.75, right =
    ↪  41.04, top = 61.86)
```

```
march.1812.europe.map <- get_stamenmap(bbox = march.1812.europe, zoom = 5,
                                       maptype = "terrain", where = "cache")
ggmap(march.1812.europe.map)

#Mapa mais cartonesco
march.1812.europe.map.wc <- get_stamenmap(bbox = march.1812.europe, zoom = 5,
                                          maptype = "watercolor", where = "cache")
ggmap(march.1812.europe.map.wc)
```

Por fim, utilizei o ggmap para superpor o gráfico de Minard no mapa correspondente. O código e o gráfico ficaram assim:

```
#Sobreposição do mapa com o gráfico de minard
march.1812.ne.europe <- c(left = 23.5, bottom = 53.4, right = 38.1, top = 56.3)
march.1812.ne.europe.map <- get_stamenmap(bbox = march.1812.ne.europe, zoom = 8,
                                          maptype = "terrain-background", where = "cache")
march.1812.plot <- ggmap(march.1812.ne.europe.map) +
geom_path(data = troops, aes(x = long, y = lat, group = group,
                             color = direction, size = survivors),
          lineend = "round") +
geom_point(data = cities, aes(x = long, y = lat),
           color = "#DC5B44") +
geom_text_repel(data = cities, aes(x = long, y = lat, label = city),
                color = "#DC5B44", family = "Open Sans Condensed Bold") +
scale_size(range = c(0.5, 10)) +
scale_colour_manual(values = c("#DFC17E", "#252523")) +
guides(color = FALSE, size = FALSE) +
theme_nothing()
march.1812.plot
```

Figura 24: minard_modificado.png

5 Gráficos de Pizza

O objetivo do quinto exercício é ver como utilizar coordenadas polares no ggplot. A principal aplicação é para criar gráficos de pizza. Em suma, a coordenada polar é aplicada num gráfico de barras empilhadas como se fosse uma camada. Aqui temos um exemplo simples de gráfico de barras. [5]

```
1    pie <- ggplot(mtcars, aes(x = factor(1), fill = factor(cyl))) +
2    geom_bar(width = 1) + coord_polar(theta = "y")
```

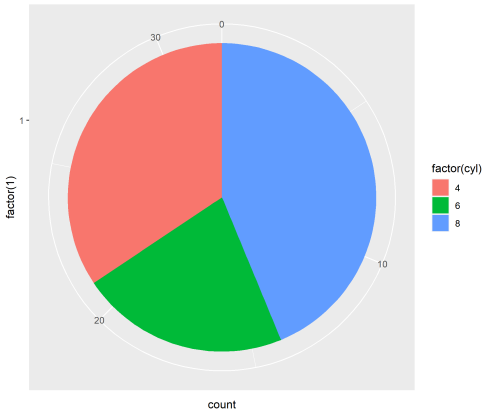

Figura 25: pizza1.png

[5] Aqui o arquivo de script utilizado é o pizza.R

A cada modificação feita no gráfico de barras, temos um correspondente no gráfico com coordenadas polares.

```
cxc <- ggplot(mtcars, aes(x = factor(cyl))) +
geom_bar(width = 1, colour = "black") + coord_polar()
```

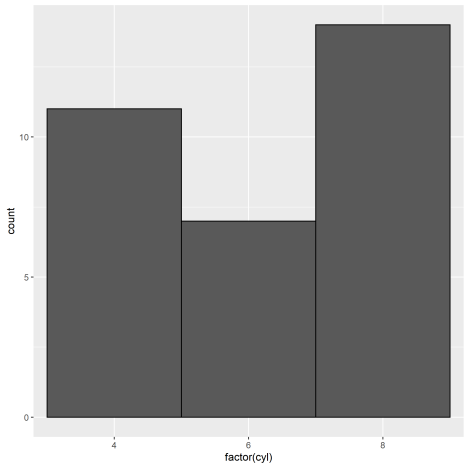

Figura 26: barras2.png

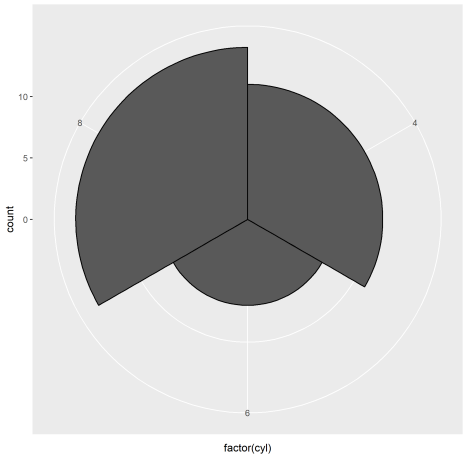

Figura 27: pizza2.png

Em seguida tentamos mudar um pouco o gráfico colocando parâmetros dentro da função de coordenadas polares.

```
cxc2 <- cxc + coord_polar(theta = "y")
```

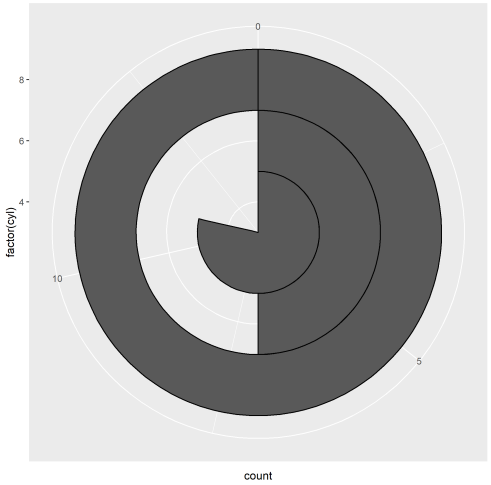

Figura 28: pizza3.png

Uma outra forma é, em cima de uma gráfico de barras empilhadas colocar uma camada de coordenadas polares. Fica parecendo um alvo de dardos.

```
eye <- pie + coord_polar()
```

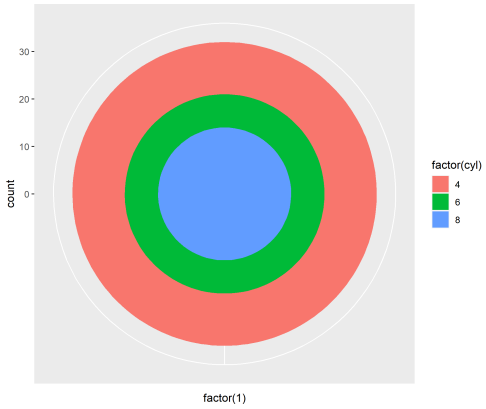

Figura 29: pizza4.png

Dá pra fazer o gráfico pacman, quando comparamos apenas duas variáveis num gráfico de pizza.

```
df <- data.frame(
variable = c("does not resemble", "resembles"),
value = c(20, 80))
pacman <- ggplot(df, aes(x = "", y = value, fill = variable)) +
geom_col(width = 1) +
scale_fill_manual(values = c("red", "yellow")) +
coord_polar("y", start = pi / 3) +
labs(title = "Pac man")
```

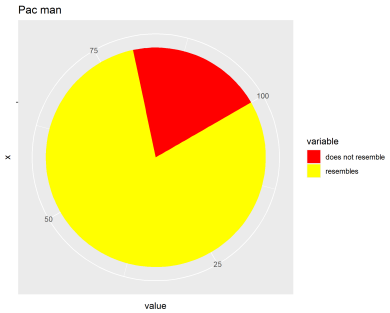

Figura 30: pizza5.png

E por fim um gráfico mais elaborado, chamado de gráfico Windrose.

```
movies$budgetq <- cut_number(movies$budget, 4)
movies$rrating <- cut_interval(movies$rating, length = 1)
doh <- ggplot(movies, aes(x = rrating, fill = budgetq))
doh1 <- doh + geom_bar(width = 1) + coord_polar()
doh2 <- doh + geom_bar(width = 0.9, position = "fill") +
↪  coord_polar(theta = "y")
```

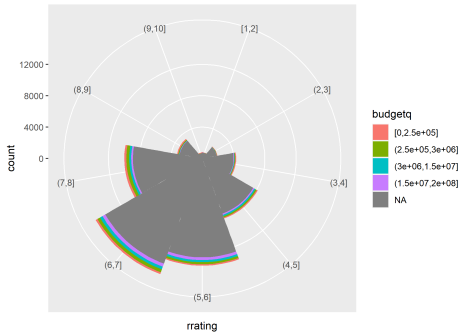

Figura 31: pizza6.png

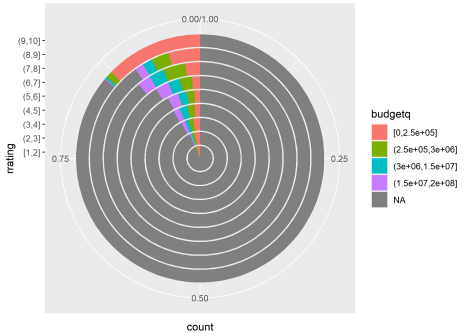

Figura 32: pizza7.png

6 Código

Os scripts e todas as imagens produzidas para esse trabalho podem ser encontradas nesse link: https://github.com/ZuilhoSe/R-Estudos.