ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DEETC - DEPARTAMENTO DE ENGENHARIA DE

ELETRÓNICA E TELECOMUNICAÇÕES E COMPUTADORES

MEIC

Mestrado em Engenharia Informática e Computadores

Mineração de Dados em Larga Escala

Aula de Laboratório #1

Configuração da linguagem R e do ambiente de trabalho



Nuno Gomes 18364

Ricardo Ramos 46638

Rafael Carvalho 47663

*Fevereiro 2024*

Índice

[2 Introdução 5](#_Toc160291892)

[3 Instruções na linguagem R 6](#_Toc160291893)

[3.1 Escalares e Operadores 6](#_Toc160291894)

[3.2 Manipulação de Vetores 7](#_Toc160291895)

[3.3 Manipulação de matrizes 8](#_Toc160291896)

[*3.4* Manipulação de *data frames* 8](#_Toc160291897)

[3.5 Manipulação de fatores 10](#_Toc160291898)

[3.6 Algumas funções úteis 10](#_Toc160291899)

[3.7 Pacotes 10](#_Toc160291900)

[3.8 Estatísticas descritivas básicas e mais 11](#_Toc160291901)

[4 Introdução ao Spark 13](#_Toc160291902)

[5 Conclusão 14](#_Toc160291903)

[6 Referências Bibliográficas 14](#_Toc160291904)

[6.1 Bibliografia 14](#_Toc160291905)

[6.2 Webgrafia 14](#_Toc160291906)

Índice de Figuras

[Figura 1 - Comparação dos elementos do vetor 'a' com 2 6](#_Toc160291932)

[Figura 2 - Retorno de a[a>2] 6](#_Toc160291933)

[Figura 3 - Demonstração da operação 10 na matriz 'n' 7](#_Toc160291934)

[Figura 4 - Vista de my.data na operação 7 8](#_Toc160291935)

[Figura 5 - Negação da variável "f" 8](#_Toc160291936)

[Figura 6 - Vista de my.data na operação 10 8](#_Toc160291937)

[Figura 7 - Vista de my.data na operação 12 8](#_Toc160291938)

[Figura 8 - EX.1 10](#_Toc160291939)

[Figura 9 - EX.2 11](#_Toc160291940)

[Figura 10 - EX.3 11](#_Toc160291941)

[Figura 11 – Vetor criado pela função range 11](#_Toc160291942)

[Figura 12 - Vetor criado pelo operador ":" 11](#_Toc160291943)

[Figura 13 - Resultado da função describe() 12](#_Toc160291944)

Índice de Tabelas

[Tabela 1 - Escalares e Operadores 5](#_Toc160291945)

# Introdução

A análise de grandes conjuntos de dados para extrair informações valiosas e padrões significativos é o foco da disciplina de Big Data Mining. Com o aumento exponencial do volume de dados na era digital, tornou-se essencial contar com ferramentas e tecnologias especializadas para lidar eficientemente com esses vastos conjuntos de informações.

O Apache Spark destaca-se como uma ferramenta poderosa no contexto do Big Data, proporcionando um ambiente de computação em cluster rápido e versátil para o processamento de dados. A linguagem de programação R é amplamente reconhecida no âmbito da estatística e análise de dados, sendo uma escolha popular devido à sua sintaxe amigável e à extensa coleção de pacotes, especialmente para quem trabalha com dados em grande escala.

Neste trabalho introdutório à unidade curricular, pretende-se testar, ensaiar e familiarizar quer com o ambiente Spark quer com a linguagem de programação R, constituindo estes o alicerce para os demais trabalhos e desenvolvimento das aprendizagens da unidade curricular.

Em síntese, o relatório subdivide-se em dois temas: Instruções na linguagem R, onde é abordado a prática de implementação de instruções R básicas, essenciais, e configuração do Spark, onde, sucintamente são descritos os resultados da aplicação do código.

# Instruções na linguagem R

## Escalares e Operadores

Na linguagem R, um escalar é um valor único, podendo ser numérico ou um caracter.

Os operadores podem ser distinguidos em aritméticos, como é o caso de ‘+’, ‘-‘, ‘\*’, ‘/’ que permitem realizar as operações de soma, subtração, multiplicação, e divisão, respetivamente; de comparação, como é o caso de ‘<’, ‘>’, ‘==’, ‘!=’, para aferir se um valor é maior, menor, igual, ou diferente de outro.

Existem ainda outros operadores, no entanto não são explorados neste primeiro trabalho prático.

De seguida, mostra-se uma tabela com as instruções a realizar, e uma explicação dos resultados obtidos para cada.

Tabela 1 - Escalares e Operadores

|  |  |
| --- | --- |
| 1. var1 <− 3 | Atribui o valor 3 à variável var1 |
| 2. show( var1 ) | Mostra o valor da variável var1 |
| 3. var2 <− var1 ∗ var1 | Atribui à variável var2 o quadrado do valor de var1 |
| 4. var3 <− var1 ∗∗ 2 | Atribui à variável var3 o valor de var1 elevado ao quadrado |
| 5. var4 <− var1 ˆ 2 | (^ operador de potência) Atribui à variável var4 o valor de var1 ao quadrado |
| 6. var1 < var1 | Retorna o resultado booleano da comparação do valor de var1 com o valor de var1 |
| 7. var3 != var4 | Verifica se var3 é diferente de var4 e retorna TRUE ou FALSE |
| 8. var2 == var4 | Verifica se var2 é igual a var4 e retorna TRUE ou FALSE |
| 9. var2 <− var2 − var2 | Subtrai o valor atual de var2 do próprio var2, resultando em 0 e guarda em var2 |
| 10. var5 <− var3 / var2 | Divide var3 pelo valor de var2 e atribui o resultado à variável var5 |
| 11. var5 + 1 | Adiciona 1 ao valor de var5 e retorna o resultado |

## Manipulação de Vetores

O vetor é uma estrutura de dados que pode conter múltiplos escalares do mesmo tipo, e é criado através da função c(). Os vetores podem ter uma dimensão, ou várias, criando matrizes.

|  |  |
| --- | --- |
| 1. a <- c(1,2,5.3,6,-2, 4, 3.14159265359) | A função c() cria um vetor, afeta a variável ‘a’ com o vetor |
| 2. a | Mostra o vetor ‘a’ |
| 3. b <- c("1", "2", "3") | Cria um vetor de caracteres ‘b’ |
| 4. "2" %in% b | Verifica se "2" está presente no vetor ‘b’ |
| 5. "5" %in% b | Verifica se "5" está presente no vetor ‘b’ |
| 6. c <- c(TRUE, TRUE, FALSE, TRUE) | Cria um vetor de booleanos ‘c’ |
| 7. a <- c | Afeta a variável ‘a’ com os valores de ‘c’ |
| 8. a[0] | Acede ao elemento na posição 0 de ‘a’. De notar que a indexação dos elementos começa em 1 |
| 9. a[1] | Acede ao primeiro elemento de ‘a’ |
| 10. a[-1] | Apresenta todos os valores exceto o primeiro elemento de ‘a’ |
| 11. a[8] | Acede ao oitavo elemento de 'a' (se existir) |
| 12. a[c(1, 3)] | Seleciona os elementos nas posições 1 e 3 de ‘a’ |
| 13. a[c(3, 1)] | Seleciona os elementos nas posições 3 e 1 de ‘a’ |
| 14. a[a>2] | Seleciona apenas os elementos maiores que 2 do vetor ‘a’ |

A operação 14 pode ser separada em dois diferentes passos:

1. Compara todos os elementos de ‘a’ com 2:



Figura 1 - Comparação dos elementos do vetor 'a' com 2

1. Indexa o vetor ‘a’ e retorna todos os elementos onde o vetor de a > 2 é TRUE:

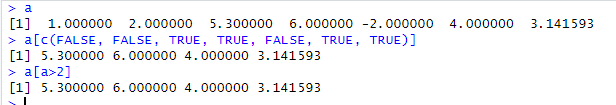


Figura 2 - Retorno de a[a>2]

## Manipulação de matrizes

|  |  |
| --- | --- |
| 1. m <- matrix(1:6, nrow=3, ncol=2) | Cria uma matriz 3x2 com valores de 1 a 6 |
| 2. show(m) | Mostra a matriz ‘m’ |
| 3. n <- matrix(2:7, nrow=2, ncol=3) | Cria uma matriz 2x3 com valores de 2 a 7 |
| 4. n | Mostra a matriz ‘n’ |
| 5. m[, 2] | Seleciona a segunda coluna da matriz ‘m’ |
| 6. n[1, ] | Seleciona a primeira linha da matriz ‘n’ |
| 7. m[2:3, 1:2] | Seleciona as linhas 2 e 3 e as colunas 1 e 2 da matriz ‘m’ |
| 8. n %\*% m | Multiplica as matrizes ‘n’ e ‘m’ |
| 9. m %\*% n | Multiplica as matrizes ‘m’ e ‘n’ |
| 10. n %\*% n | Multiplica a matriz ‘n’ por ela mesma |
| 11. nˆ2 | Eleva cada elemento da matriz ‘n’ ao quadrado |
| 12. sqrt(n) | Calcula a raiz quadrada de cada elemento da matriz ‘n’ |

Para a matriz construída ‘n’, de 2 linhas e 3 colunas, a operação 10 não é matematicamente possível. Este tipo de operações apenas é possível em situações onde a matriz é quadrada.

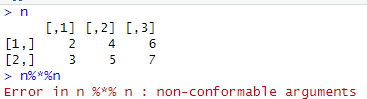


Figura 3 - Demonstração da operação 10 na matriz 'n'

## Manipulação de *data frames*

Um *data frame* é uma estrutura de dados tabular semelhante a uma tabela, onde cada coluna pode conter valores escalares de diferentes tipos de dados.

No exemplo que se segue constrói-se um *data frame* inicialmente com três colunas, com um identificador numérico, uma string, e um valor booleano. Nomeiam-se as colunas para “ID”, “Name” e “Passed” com a função names(). Adiciona-se então uma nova coluna de *features* de nome “Failed” com a função cbind() (*Column Bind*), e por fim uma nova entrada (linha) com a função rbind() (*Row Bind*).

|  |  |
| --- | --- |
| 1. d <- c(1, 2, 3, 4) | Cria um vetor ‘d’ com valores de 1 a 4 |
| 2. e <- c("Bob", "Alice", NA, "Joe") | Cria um vetor ‘e’ de strings |
| 3. f <- c(TRUE, TRUE, FALSE, TRUE) | Cria um vetor lógico ‘f’ |
| 4. my.data <- data.frame(d, e, f, stringsAsFactors=FALSE) | Cria um data frame chamado 'my.data' |
| 5. show(my.data) | Mostra o conteúdo do data frame 'my.data' |
| 6. names(my.data) <- c("ID", "Name", "Passed") | Renomeia as colunas do data frame 'my.data' |
| 7. View(my.data) | Visualização da matriz do data frame 'my.data'. A visualização é apresentada na Figura 4. |
| 8. my.data$Name | Seleciona a coluna 'Name' do data frame 'my.data' |
| 9. my.data <- cbind(my.data, Failed=!f) | Adiciona uma nova coluna 'Failed' ao data frame 'my.data' |
| 10. View(my.data) | Mostra o data frame 'my.data' após a alteração. O data frame é mostrado na Figura 6. |
| 11. my.data <- rbind(my.data, c(5, "Carol", FALSE, TRUE)) | Adiciona uma nova linha ao data frame 'my.data' |
| 12. View(my.data) | Mostra o data frame 'my.data' após a alteração. O data frame é mostrada na Figura 7. |
| 13. ?data.frame | Abre a documentação para a função data.frame |

Na operação 7 é mostrada a vista de my.data. essa vista é apresentada na seguinte figura:

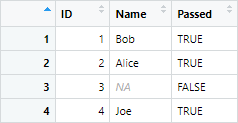


Figura 4 - Vista de my.data na operação 7

Na operação 9 é adicionada a coluna Failed ao *data frame* my.data. Esta nova coluna contém os valores resultantes da negação da variável “f”.



Figura 5 - Negação da variável "f"

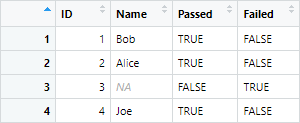


Figura 6 - Vista de my.data na operação 10

Na operação 11 é adicionada uma linha ao *data frame* my.data. Esta nova linha contém os valores 5, “Carol”, FALSE, TRUE.

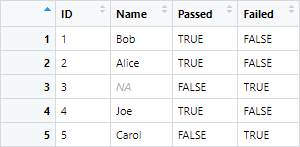


Figura 7 - Vista de my.data na operação 12

## Manipulação de fatores

Os fatores são um tipo de dados utilizados para classificar categoricamente diferentes variáveis com um conjunto fixo, e bem determinado, de valores. Os valores destas variáveis podem ser textuais, no entanto são representados de forma numérica nos fatores. Podem ser usados para distinguir, por exemplo, duas cores, como demonstrado no exemplo abaixo, onde é definido um vetor com a função rep(), que replica um determinado valor passado como argumento, neste caso “red” 20 vezes, e “blue” 30 vezes.

Ao aplicar a função factor() sobre este vetor, passamos a ter 2 níveis: “Red” e “Blue”, que depois permite inferir resumos estatísticos deste vetor com a função summary().

|  |  |
| --- | --- |
| 1. colour <- c(rep("red", 20), rep("blue", 30)) | Cria um vetor chamado 'colour' com repetição de "red" 20 vezes e "blue" 30 vezes |
| 2. colour <- factor(colour) | Converte 'colour' num fator apresentado todos os valores diferentes presentes |
| 3. summary(colour) | Apresenta um resumo estatístico do fator 'colour' |
| 4. dimensions <- c("large", "medium", "small") | Cria um vetor de strings chamado 'dimensions' |
| 5. show(dimensions) | Mostra o vetor 'dimensions' |
| 6. dimensions <- ordered(dimensions) | Converte 'dimensions' num fator ordenado, a ordenação é feita alfabeticamente |
| 7. show(dimensions) | Mostra o vetor 'dimensions' após a alteração |

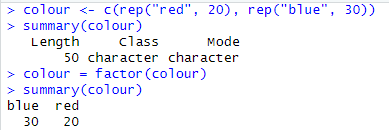


Figura 8 - Transformação do vetor 'colour' em fatores, e sumário descritivo

## Algumas funções úteis

O R disponibiliza também outras funções que podem por vezes ser úteis, como por exemplo a função length(), que retorna a dimensão de um determinado vetor, class(), que retorna a classe do objeto passado como argumento, nrow() e ncol() para obter a informação de linhas e colunas de um *data frame*.

Para obter informação acerca da sessão atual, usa-se a função sessionInfo(); a listagem dos objetos criados é feita com ls() e podem ser removidos com rm(), passando como argumento o objeto a apagar.

Para situações onde a *data frame* é muito extensa, a função glimpse() é útil uma vez que apresenta os dados de todas as colunas na vertical, e os dados de cada, na horizontal, como demonstrado abaixo:

|  |  |
| --- | --- |
| 1. length(colour) | Retorna o comprimento (número de elementos) do vetor 'colour' |
| 2. length(a) | Retorna o comprimento do vetor 'a' |
| 3. class(colour) | Retorna a classe do objeto 'colour' |
| 4. class(dimensions) | Retorna a classe do objeto 'dimensions' |
| 5. class(a) | Retorna a classe do objeto 'a' |
| 6. class(my.data) | Retorna a classe do objeto 'my.data' |
| 7. nrow(my.data) | Retorna o número de linhas do data frame 'my.data' |
| 8. ncol(my.data) | Retorna o número de colunas do data frame 'my.data' |
| 9. str(my.data) | Apresenta a estrutura do data frame 'my.data' |
| 10. sessionInfo() | Mostra informações sobre a sessão do R |
| 11. ls() | Lista os objetos no ambiente de trabalho |
| 12. rm(a) | Remove o objeto 'a' do ambiente de trabalho |
| 13. glimpse(my.data) | Apresenta uma visão resumida do data frame 'my.data' |

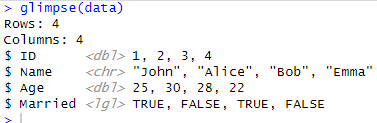


Figura 9 - Exemplo de utilização da função glimpse()

## Pacotes

O R permite adicionar novos pacotes com mais funcionalidades. Estes pacotes podem ser listados com a função library(), e quando passado um pacote como argumento, por exemplo “MASS”, carrega o mesmo. Para confirmar que foi carregado, utiliza-se a função search().

A instalação de novos pacotes é feito com a função install.packages().

Em situações onde são necessários carregar múltiplos pacotes de uma só vez, pode ser útil definir um vetor com o nome de cada um, como mostrado no exemplo 9 deste subcapítulo, e utilizar a função lapply() passando o vetor e a função a ser aplicada ao mesmo, neste caso, aplica-se a função require() ao vetor ‘x’, como demonstrado em 10.

|  |  |
| --- | --- |
| 1. library() | Mostra os pacotes atualmente carregados |
| 2. search() | Mostra a lista de pacotes disponíveis |
| 3. library("MASS") | Carrega o pacote "MASS" |
| 4. search() | Mostra a lista de pacotes após a carga do "MASS" |
| 5. .libPaths() | Mostra os caminhos do diretório onde os pacotes são instalados |
| 6. install.packages("e1071") | Instala o pacote "e1071" |
| 7. install.packages("funModeling") | Instala o pacote "funModeling" |
| 9. x <- c("MASS", "dplyr", "e1071") | Cria um vetor 'x' com os nomes dos pacotes |
| 10. lapply(x, require, character.only=TRUE) | Carrega os pacotes usando *lapply* |
| 12. require(funModeling) | Carrega o pacote "funModeling" |

## Estatísticas descritivas básicas e mais

(descrever algo aqui, especialmente a função fivenum e status)

|  |  |
| --- | --- |
| 1. iris | Retorna o conjunto de dados 'iris' |
| 2. summary(iris) | Apresenta um resumo estatístico do conjunto de dados 'iris' |
| 3. fivenum(iris$Sepal.Length) | Calcula os cinco números resumidos para a variável Sepal.Length em 'iris' |
| 5. status(iris) | Chama a função 'status' com o conjunto de dados 'iris' |

(c) A explicação que se segue é feita com base nos exercícios propostos para o ficheiro anexo ao trabalho, “ControlFlow.R”.

**ii. Explique o propósito dos três exemplos chamados EX.1, EX.2 e EX.3.**

O primeiro exemplo apresenta um bloco ‘*for*’, com uma variável ‘i’ iniciada a 0, e terminada em 10, com iterações unitárias, definido pela função seq().

Por cada iteração, a variável ‘x’, posteriormente iniciada com o valor 0, é atualizada com a soma do seu valor, e do valor de ‘i’. Desta forma, pode-se concluir que está a ser feito o somatório de 0 a 10, e que ‘x’ termina com o valor 55.

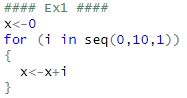


Figura 10 - EX.1

No segundo exercício, novamente com um bloco ‘*for*’, a variável de incremento ‘i’ toma o valor 0 durante 10 iterações, definida pela função rep(), em que o primeiro argumento é o valor a replicar, e o segundo o número de vezes a ser replicado. Desta forma, em cada iteração, é mostrado na consola (print()), o valor de ‘i’, que é sempre 0.

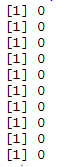


Figura 11 - Output do exercício 2

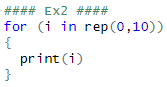


Figura 12 - EX.2

No terceiro exercício, inicia-se um vetor ‘v’, e, num bloco ‘*while*’, verifica-se o vetor tem dimensão positiva. Por cada iteração, o vetor ‘v’ é atualizado, removendo o primeiro elemento. De seguida, é mostrado na consola o novo vetor. O bloco ‘*if*’ verifica se o valor 6 ainda está presente em ‘v’, e, caso não se encontre, termina o ciclo. Desta forma, o exercício remove todos os elementos de um vetor ‘v’, até que o número 6 não esteja presente.

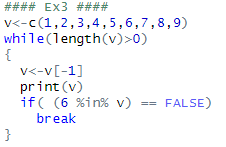


Figura 13 - EX.3

**(d) Implemente e mostre um código que obtém as duas primeiras colunas de my.data, usando a indexação do operador de intervalo.**

**(Falta confirmar)**

O operador de intervalo no R é a função range(). Desta forma, uma vez que pretendemos obter todas as linhas das duas primeiras colunas, podemos utilizar o comando abaixo:

Como não foi introduzido nenhum valor para a indexação das linhas são mostradas todas as linhas disponíveis no *data frame*. Para a indexação das colunas é utilizada a função range(1,2). Esta função cria um vetor que contém os valores mínimo (1) e máximo (2), por omissão o valor de incremento é 1 e por isso obtém-se o seguinte vetor.

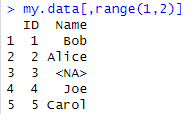


Figura 14 - Indexação com operador de intervalo range()



Figura 15 - Vetor criado pela função range()

**(e) Implemente e mostre um código que obtém as duas primeiras colunas de my.data, usando a indexação vetorial.**

Utilizando a indexação vetorial é possível obter as duas primeiras colunas através da linha seguinte:

O operador “:” é muito semelhante à função range() mas apenas funciona com incrementos ou decrementos de 1 em 1.

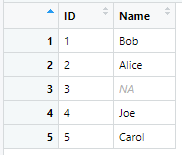


Figura 18 - Representação(View()) das primeiras duas colunas de my.data



Figura 16 - Vetor criado pelo operador ":"

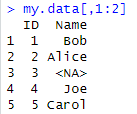


Figura 17 - Indexação vetorial

**(f) Instale o pacote Hmisc e use a função describe na variável my.data.**

(**Falta comentar**)

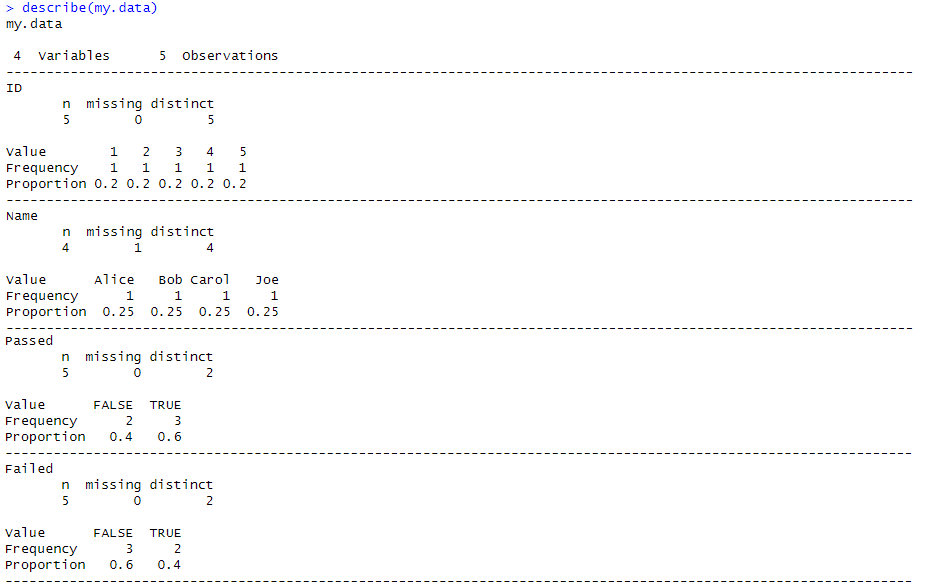


Figura 19 - Resultado da função describe()

# Introdução ao Spark

Descreva os comandos usados (se não listados), os problemas (se houver) e os resultados dos seguintes pontos, linha

por linha:

1. Verifique programaticamente se o pacote sparklyr está carregado. Em seguida, conecte-se ao Spark usando

(Falta meter aqui o código do chatGPT e descrever os comandos)

ss <- spark\_connect('local', version = '3.4.2', hadoop\_version = '3', config = list()).

(b) Veja o conteúdo da variável ss.

(Figura)

(c) Use a função copy do pacote dplyr para copiar o conjunto de dados iris para o Spark.

1. library(dplyr)

2. df <− copy\_to(ss, iris)

3. show(df)

(d) Mostre alguns dados amostrais do conjunto de dados carregado.

(Meter prints)

1. head(select(df, Petal.Width, Species))

2. head(filter(df, Petal.Width > 0,3)

3. df %>% head

(Meter prints)

(e) Utilizando SQL

1. library (DBI)

2. df\_sql <− dbGetQuery ( ss , ”SELECT \* FROM iris WHERE Petal\_Width > 0 . 3 LIMIT 5” )

3. show ( df\_sql )

(Meter prints)

# Conclusão

A simplicidade da linguagem R complementada pela capacidade de processamento distribuído do Spark, proporciona uma poderosa solução para explorar padrões complexos e extrair insights valiosos em ambientes de Big Data, contribuindo para uma análise robusta e informada.

# Referências Bibliográficas

## Bibliografia

## Webgrafia

Anexos