Exercício 3: Boas Práticas de Programação em R

1. **Organização do Código e Registros de Operações:** Crie um script em R que organize o código em seções bem-definidas usando comentários para cada bloco. Implemente um sistema de log em um arquivo externo para registrar operações críticas, como carregamento de pacotes e salvamento de resultados. Use o comando `write()` para salvar mensagens com timestamps no log.

```
# Organizando o código com seções e criando um arquivo de log
setwd("caminho/para/diretorio")  # Define o diretório de trabalho
log_file <- "registro_log.txt"
write("Iniciando script: " %>% Sys.time(), file = log_file)

# Log de carregamento de pacotes
if(!require(dplyr)) install.packages("dplyr")
library(dplyr)
write("Pacote dplyr carregado às " %>% Sys.time(), file =
log_file, append = TRUE)
```

2. **Funções Bem Documentadas:** Escreva uma função que, dado um vetor de números, retorna a média e o desvio padrão, e registra essa operação no arquivo de log. Documente cada etapa da função com comentários e inclua uma verificação de erro caso o vetor esteja vazio.

```
# Função para calcular a média e desvio padrão com registro no
log
calcula_media_sd <- function(numeros) {
  if (length(numeros) == 0) {
    stop("Erro: Vetor vazio")
  }
  media <- mean(numeros)
  sd_valor <- sd(numeros)
  write(paste("Cálculo de média e desvio padrão realizado: ",
media, sd_valor, Sys.time()), file = log_file, append = TRUE)
  return(list(media = media, desvio_padrao = sd_valor))
}
```

3. **Divisão de Código em Módulos e Uso de `source():** Separe o código em múltiplos arquivos, por exemplo, um arquivo para funções e outro para execução principal. Use `source()` para importar o arquivo de funções e organize o script principal com seções comentadas.

```
# Suponha que salvamos o cálculo acima em um arquivo chamado
"funcoes.R"
```

```
# No script principal, usamos source() para importar e utilizar
as funções.

source("funcoes.R")
numeros_exemplo <- c(10, 20, 30)
resultado <- calcula_media_sd(numeros_exemplo)
print(resultado)
```

4. **Organização e Limpeza de Dados com Comentários:** Crie um dataframe com dados fictícios que incluam valores NA. Desenvolva um script para limpar o dataframe, removendo ou substituindo valores NA, e registre a operação em um log. Comente cada linha para explicar o que o código está realizando.

```
# Criando um dataframe fictício com valores NA e limpando-o
df_exemplo <- data.frame(
  nome = c("Ana", "Beto", "Carlos", NA, "Eduardo"),
  idade = c(25, NA, 30, NA, 45),
  salario = c(5000, 7000, NA, 6000, 8000)
)

# Removendo linhas com valores NA e registrando
df_limpo <- na.omit(df_exemplo)
write("Dados limpos às " %>% Sys.time(), file = log_file, append
= TRUE)
```

5. **Tratamento de Erros e Condicionais:** Crie uma função que recebe um dataframe e verifica se contém colunas com mais de 50% de valores NA. Caso encontre essas colunas, registre a operação no log e remova-as do dataframe, retornando o resultado. Explique cada etapa com comentários detalhados.

```
# Função para verificar e remover colunas com muitos valores NA
remove_colunas_na <- function(dados) {
  colunas_na <- sapply(dados, function(col) mean(is.na(col)) >
0.5)
  dados <- dados[, !colunas_na]
  write("Colunas com NA removidas às " %>% Sys.time(), file =
log_file, append = TRUE)
  return(dados)
}

df_exemplo_limpo <- remove_colunas_na(df_exemplo)
```

6. **Exportação de Resultados e Registros de Comando:** Salve o dataframe processado em um arquivo CSV, registrando a operação de exportação no log. Inclua mensagens de conclusão no log usando `write()` e adicione comentários explicativos em cada passo.

```
# Salvando dataframe limpo em um arquivo CSV e registrando
operação
write.csv(df_exemplo_limpo, "dados_limpos.csv", row.names =
FALSE)
write("Dados exportados para CSV às " %>% Sys.time(), file =
log_file, append = TRUE)
```

7. **Exemplo Completo de Documentação e Boas Práticas:** Crie um script que simule uma pequena análise de dados (ex., cálculo de média, filtros com `dplyr`, etc.), seguindo todas as boas práticas de documentação, organização em seções, e uso de logs para registrar etapas críticas do processo.

```
# Criando um dataframe e calculando a média, com documentação e
logs
setwd("caminho/para/diretorio")
log_file <- "registro_analise.txt"

# Carregando pacotes
if(!require(dplyr)) install.packages("dplyr")
library(dplyr)
write("Pacote dplyr carregado às " %>% Sys.time(), file =
log_file)

# Função para análise de idade média
calcula_idade_media <- function(dados) {
  if ("idade" %in% names(dados)) {
    media <- mean(dados$idade, na.rm = TRUE)
    write(paste("Média de idade calculada:", media, Sys.time()),
file = log_file, append = TRUE)
    return(media)
  } else {
    warning("Coluna 'idade' não encontrada")
    return(NA)
  }
}

df_idades <- data.frame(idade = c(25, NA, 30, 40, 35))
idade_media <- calcula_idade_media(df_idades)
```

8. **Uso de `message()` e `warning()` para Feedback ao Usuário:** Implemente um feedback ao usuário usando `message()` e `warning()` dentro das funções. Por exemplo, avise se o dataframe contém valores NA ou se o cálculo de média resultou em um valor inesperado. Comente as mensagens de feedback explicando seu propósito.

```
# Criando um dataframe e calculando a média, com documentação e
logs
setwd("caminho/para/diretorio")
```

```
log_file <- "registro_analise.txt"

# Carregando pacotes
if(!require(dplyr)) install.packages("dplyr")
library(dplyr)
write("Pacote dplyr carregado às " %>% Sys.time(), file =
log_file)

# Função para análise de idade média
calcula_idade_media <- function(dados) {
  if ("idade" %in% names(dados)) {
    media <- mean(dados$idade, na.rm = TRUE)
    write(paste("Média de idade calculada:", media, Sys.time()),
file = log_file, append = TRUE)
    return(media)
  } else {
    warning("Coluna 'idade' não encontrada")
    return(NA)
  }
}

df_idades <- data.frame(idade = c(25, NA, 30, 40, 35))
idade_media <- calcula_idade_media(df_idades)
```

9. **Testes de Unidade e Validação:** Escreva um teste simples com `stopifnot()` para verificar se a média de um vetor é calculada corretamente em uma função, documentando o objetivo de cada teste. Adicione verificações adicionais, como testar se o vetor não é vazio.

```
# Teste de unidade com stopifnot()
teste_calcula_media <- function() {
  valores <- c(10, 20, 30)
  resultado <- calcula_media_sd(valores)$media
  stopifnot(resultado == mean(valores))  # Teste: a média
calculada deve ser correta
  message("Teste passou.")
}

teste_calcula_media()
```

10. **Prática de Boas Práticas e Revisão de Código:** Revise um código R fornecido e identifique áreas de melhoria. Reescreva o código implementando as boas práticas aprendidas e comente cada modificação. Em seguida, compare a performance do código antes e depois das alterações.

```
# Revisão de um código com correções implementadas
# Original:
# numeros = c(10,20,30); media_num = mean(numeros); sd(numeros)
```

```
# Revisado:
numeros <- c(10, 20, 30)  # Uso de <- para atribuição
media_num <- mean(numeros)  # Calculo da média
desvio_padrao <- sd(numeros)  # Calculo do desvio padrão
print(paste("Média:", media_num, "Desvio padrão:",
desvio_padrao))

# Comparar desempenho com uso de microbenchmark
if(!require(microbenchmark)) install.packages("microbenchmark")
library(microbenchmark)
microbenchmark(
  original = { media_num = mean(numeros); sd(numeros) },
  revisado = { media_num <- mean(numeros); desvio_padrao <-
sd(numeros) }
)
```