

Dia 1

Fundamentos e Ambiente de Trabalho

Vinícius Silva Junqueira

2025-11-02

Sumário

1 Abertura	2
2 Apresentação do Curso (15 min)	2
2.1 Bem-vindos!	2
2.2 Objetivos Gerais do Curso	2
2.3 Metodologia	2
2.4 Estrutura dos 4 Dias	2
2.5 Materiais e Suporte	2
3 Por Que R, GitHub e IA? (15 min)	3
3.1 Por Que R?	3
3.2 Por Que GitHub?	3
3.3 Por Que IA (ChatGPT e Claude)?	3
4 Ambientação e Setup (40 min)	4
4.1 Verificações rápidas	4
4.2 Configurar Git (uma vez só)	4
4.3 Autenticar no GitHub (PAT recomendado)	4
5 Fundamentos de R (50 min)	7
5.1 Objetos básicos e operações	7
5.2 Vetores e indexação	8
5.3 Listas e data.frames	9
5.4 Fatores	10
6 Exploração inicial de dados (40 min)	10
6.1 Comparando str() vs glimpse()	11
7 Exercícios guiados (20 min)	12
7.1 Exercício 1 — Vetores	12
7.2 Exercício 2 — Data frame	12
7.3 Exercício 3 — Exploração palmerpenguins	12
8 Primeiro commit (5 min)	12

9 Checklist de encerramento	13
10 Referências rápidas	13

1 Abertura

Tempo previsto 19h00–22h00 (intervalo 20h30–20h50)

2 Apresentação do Curso (15 min)

2.1 Bem-vindos!

Olá! Seja muito bem-vindo ao **Curso Intensivo de R com GitHub e IA**. Esta jornada de 16 horas foi cuidadosamente estruturada para transformar você de iniciante a alguém capaz de realizar análises de dados completas usando ferramentas modernas e profissionais.

2.2 Objetivos Gerais do Curso

Ao final deste curso, você será capaz de:

- Programar em R com confiança, desde operações básicas até análises complexas
- Manipular e transformar dados usando o ecossistema tidyverse
- Criar visualizações profissionais e informativas com ggplot2
- Versionar seu código com Git e colaborar via GitHub
- Usar inteligência artificial (ChatGPT e Claude) para acelerar seu aprendizado e resolver problemas

2.3 Metodologia

Nossa abordagem é **100% prática e hands-on**:

- **Teoria mínima necessária** seguida de prática imediata
- **Datasets reais** desde o primeiro dia
- **Commits diários** no seu fork do repositório
- **IA como assistente** para explicação, depuração e geração de código
- **Multiplataforma**: todo conteúdo funciona em Windows, macOS e Linux

2.4 Estrutura dos 4 Dias

- **Dia 1 (hoje)**: Fundamentos de R + Ambiente reproduzível (RStudio, Git, GitHub, fork)
- **Dia 2**: Lógica de programação, condicionais, funções e tidyverse básico
- **Dia 3**: Transformações com tidyr/dplyr, leitura/escrita de dados e visualização com ggplot2
- **Dia 4**: Integração prática do ChatGPT e Claude dentro do RStudio

2.5 Materiais e Suporte

- **Repositório GitHub**: <https://github.com/viniciusjunqueira/curso-r-github-ia>
- **Datasets**: incluídos no repositório + pacote palmerpenguins
- **Contato**: junqueiravinicius@hotmail.com

3 Por Que R, GitHub e IA? (15 min)

3.1 Por Que R?

R é uma linguagem poderosa e gratuita, criada especificamente para análise de dados e estatística. Algumas razões para aprender R:

Ecosistema rico - Mais de 20.000 pacotes disponíveis para praticamente qualquer análise - tidyverse: conjunto integrado de ferramentas modernas para ciência de dados - ggplot2: sistema de visualização elegante e profissional

Reprodutibilidade - Tudo que você faz fica documentado em código - Fácil repetir análises com novos dados - R Markdown permite combinar código, resultados e narrativa

Comunidade ativa - Grande comunidade brasileira e internacional - Milhares de tutoriais, cursos e fóruns de ajuda - TidyTuesday: prática semanal com dados reais

Demandा no mercado - Usado em empresas, universidades e governos - Essencial para ciência de dados, bioinformática, economia, ciências sociais - Combina bem com Python em pipelines modernos de dados

3.2 Por Que GitHub?

GitHub não é apenas para programadores! É uma plataforma essencial para:

Controle de versão - Histórico completo de todas as mudanças no seu código - Possibilidade de voltar a versões anteriores - Nunca mais perder trabalho por acidente

Colaboração - Trabalhe em equipe sem conflitos - Contribua para projetos open-source - Receba feedback e sugestões

Portfólio profissional - Mostre seus projetos para empregadores - Demonstre evolução e consistência - Compartilhe conhecimento com a comunidade

Integração moderna - Funciona perfeitamente com RStudio - Base para deployment de aplicações - Padrão da indústria para ciência de dados

3.3 Por Que IA (ChatGPT e Claude)?

A inteligência artificial revolucionou o aprendizado de programação. **Não é trapaça, é trabalhar de forma inteligente!**

Acelera o aprendizado - Explicações personalizadas para seu nível - Respostas imediatas para dúvidas específicas - Exemplos sob medida para seu contexto

Assistência na depuração - Interpretação de mensagens de erro - Sugestões de correção - Identificação de problemas de lógica

Aumenta produtividade - Geração de código boilerplate - Refatoração e otimização - Criação de documentação

Ferramentas do curso - ChatGPT (OpenAI): excelente para explicações didáticas e geração rápida de código - **Claude (Anthropic)**: ótimo para análises mais profundas e revisão de código complexo

Importante: IA é uma ferramenta, não uma substituição do aprendizado. Use-a para entender conceitos, não apenas copiar código!

4 Ambiente e Setup (40 min)

Objetivos desta seção

- Verificar instalações (R, RStudio, Git)
- Configurar Git e autenticar no GitHub
- Entender e aplicar o **workflow com fork**
- Preparar ambiente reproduzível com projetos .Rproj e here()

4.1 Verificações rápidas

```
R.version.string          # Versão do R
RStudio.Version()$version # Versão do RStudio
system("git --version")   # Confirma Git disponível
```

4.2 Configurar Git (uma vez só)

No **Terminal do RStudio** (funciona em Windows/macOS/Linux):

```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
# Verificar
git config --global --list
```

4.3 Autenticar no GitHub (PAT recomendado)

O que é um PAT?

Um Personal Access Token (PAT) é como uma “senha especial” que permite ao RStudio se comunicar com o GitHub de forma segura. O GitHub não aceita mais senhas normais para operações via linha de comando, então o PAT é obrigatório.

Passo a passo para criar e configurar o PAT:

4.3.1 Instalar pacotes necessários

```
install.packages("usethis")
install.packages("gitcreds")
```

4.3.2 Criar o token no GitHub

```
usethis::create_github_token()
```

Este comando abrirá seu navegador automaticamente na página de criação de tokens do GitHub. Você verá uma página pré-configurada com as permissões necessárias.

No navegador:

1. **Faça login no GitHub** (se ainda não estiver logado)
2. **Note (New personal access token - classic):**
 - O campo “Note” já virá preenchido com algo como “DESCRIBE THE TOKEN’S USE CASE”
 - Renomeie para algo descriptivo como: RStudio-Curso-R-2024
3. **Expiration:** escolha a duração do token
 - Para o curso: 90 days é suficiente
 - Para uso contínuo: No expiration (menos seguro, mas mais prático)
4. **Permissões (Scopes):** o usethis já marca as principais
 - repo (controle total de repositórios privados)
 - workflow (atualizar workflows do GitHub Actions)
 - gist (criar gists)
 - user (atualizar dados do usuário)
 - **Não altere nada**, as permissões pré-selecionadas são ideais
5. **Clique em “Generate token”** no final da página
6. **ATENÇÃO:** copie o token que aparece (começa com ghp_...)
 - **VOCÊ SÓ VERÁ ESTE TOKEN UMA VEZ!**
 - Cole em um lugar seguro temporariamente (bloco de notas)

4.3.3 Salvar o token no RStudio

```
gitcreds::gitcreds_set()
```

Quando executar este comando, você verá algo assim no Console:

? Enter password or token:

Cole o token que você copiou do GitHub e pressione **Enter**.

Você verá uma mensagem de confirmação:

```
-> Adding new credentials...
-> Removing credentials from cache...
-> Done.
```

```
usethis::git_sitrep()
```

4.3.3.1 Verificar se funcionou Este comando mostra o status da sua configuração Git/GitHub. Procure por:

```
GitHub user: 'seu-usuário'
Token: '<discovered>'
```

Se você ver isso, está tudo configurado!

Alternativas ao PAT: - **GitHub Desktop** (aplicativo com interface gráfica - mais simples para iniciantes) - **SSH** (método avançado, requer configuração de chaves públicas/privadas)

4.3.4 Workflow com Fork (obrigatório para a turma)

Original (instrutor) → FORK (sua conta) → CLONE (seu PC) → PUSH (para seu fork)

1. Abra: <https://github.com/viniciusjunqueira/curso-r-github-ia>
2. Clique **Fork** → escolha **sua conta** → *Create fork*.

3. Clone **SEU fork**:

```
git clone https://github.com/SEU-USUARIO/curso-r-github-ia.git
cd curso-r-github-ia
```

4. Abra o projeto .Rproj no RStudio.

5. Cheque o remote:

```
git remote -v
# Deve mostrar seu usuário em origin
```

Por que fork? Você controla seu repositório, faz commits/push à vontade e não altera o repo do instrutor.

4.3.5 Estrutura de projeto e portabilidade

```
curso-r-github-ia/
  curso-r-github-ia.Rproj
  data/
    raw/
    processed/
  scripts/
  output/
    figures/
    tables/
  docs/
# Caminhos: sempre prefira here::here()
install.packages("here")
library(here)
caminho <- here("data", "raw", "dados.csv")
caminho
```

UTF-8: salve arquivos com File → Save with Encoding → UTF-8 (evita problemas de acentuação em todos os SOs).

5 Fundamentos de R (50 min)

5.1 Objetos básicos e operações

O que são objetos em R?

Em R, tudo é um objeto! Quando você cria uma variável, você está criando um objeto que armazena informação na memória do computador. Os tipos básicos mais importantes são:

- **Numérico (numeric)**: números decimais como 3.14, 10.5, -2.7
- **Inteiro (integer)**: números inteiros como 1L, 100L (o L indica inteiro)
- **Lógico (logical)**: valores verdadeiro/falso - TRUE ou FALSE
- **Caractere (character)**: texto entre aspas como "Olá", "R", "2024"

Você cria objetos usando o operador de atribuição <- (preferido) ou =.

```
# Números, lógicos, strings
x_num <- 3.14
x_log <- TRUE
x_chr <- "Olá, R!"
class(x_num); typeof(x_num)
class(x_log); typeof(x_log)
class(x_chr); typeof(x_chr)

# Aritmética
10 + 2 # somatório
10 - 2 # subtração
10 * 2 # multiplicação
10 / 3 # divisão
2 ^ 3 # exponecial

# Infinito positivo. Significa que o valor tende ao infinito positivo.
a <- 1 / 0
a
# Resultado: Inf

# Infinito negativo. Significa que o valor tende ao infinito negativo.
b <- -1 / 0
b
# Resultado: -Inf

# Not a Number (NaN). Indica uma operação indefinida matematicamente.
c <- 0 / 0
c
# Resultado: NaN

# Valor ausente (NA). Representa um valor desconhecido ou faltante.
d <- c(2, 4, NA, 8)
d
# Resultado: 2 4 NA 8
```

```

# -----
# Testando os tipos de valores
# -----

is.infinite(a)      # TRUE
is.nan(c)           # TRUE
is.na(d)            # TRUE para o elemento ausente
is.nan(NaN)         # TRUE - NaN é considerado um tipo especial de NA
is.finite(a)         # FALSE - porque Inf não é finito
is.finite(10)        # TRUE - número normal é finito

# -----
# Operações que produzem resultados especiais
# -----

Inf + (-Inf)       # NaN (infinito positivo menos infinito negativo é indefinido)
Inf / Inf           # NaN
0 * Inf             # NaN
NA + 1              # NA

```

5.2 Vetores e indexação

O que são vetores?

Vetores são a estrutura de dados mais fundamental do R. Um vetor é uma **coleção de elementos do mesmo tipo** (todos números, ou todos textos, ou todos lógicos). Você pode pensar em um vetor como uma linha de dados em uma planilha.

Características importantes: - Criados com a função `c()` (de “combine” ou “concatenar”) - Todos os elementos devem ser do mesmo tipo - R é **1-indexed** (o primeiro elemento está na posição 1, não 0) - Operações são **vetorizadas** (aplicadas a todos elementos automaticamente)

Indexação é o processo de acessar elementos específicos de um vetor usando colchetes `[]`.

```

v <- c(10, 20, 30, 40, 50)
length(v); mean(v); sum(v)

v[1]; v[2:4]; v[-1]
sel <- v > 25; sel; v[sel]

```

Utilizando a função `names()`: Essa função serve para dar nomes (ou cabeçalhos) aos elementos de um vetor, lista ou outro objeto em R. Esses nomes tornam os dados mais organizados e permitem acessar valores pelo nome, em vez de usar apenas posições numéricas. É útil quando você quer que cada elemento tenha um rótulo identificador, como nomes de amostras, variáveis, tratamentos ou categorias — o que deixa o código mais legível e fácil de interpretar.

```

names(v) <- letters[1:5]

# Mostrando o vetor com nomes
v
#   a   b   c   d   e

```

```
# 10 20 30 40 50

# Acessando um elemento pelo nome
v["c"]
# 30
```

Type coercion. Em R, um vetor é uma estrutura homogênea, ou seja, todos os seus elementos precisam ser do mesmo tipo. Quando você cria um vetor com elementos de tipos diferentes, o R automaticamente converte (ou “coage”) todos os valores para um tipo comum que consiga representar todos eles. Esse processo é chamado de coerção de tipos (type coercion).

Regras básicas de coerção: O R segue uma hierarquia de tipos, do mais simples para o mais geral: logical → integer → double → character

Isso significa: Se você misturar lógicos (TRUE, FALSE) com números, eles viram números (TRUE = 1, FALSE = 0).

Se misturar números com texto, tudo vira texto.

O tipo character tem sempre prioridade, porque é o único que pode representar qualquer coisa como texto.

```
x <- c(1, "a")
# "1" "a"

class(x)
# "character"
```

5.3 Listas e data.frames

Listas: estruturas flexíveis

Uma **lista** é uma estrutura que pode conter elementos de **diferentes tipos** - ao contrário dos vetores. Listas são extremamente versáteis e podem armazenar números, textos, vetores, outras listas e até data.frames!

Uso típico de listas: - Armazenar resultados complexos de análises - Combinar diferentes tipos de informação - Retornar múltiplos valores de uma função

Data.frames: a estrutura tabular

Um **data.frame** é a estrutura mais importante para análise de dados em R. É similar a uma planilha do Excel ou uma tabela de banco de dados: tem linhas (observações) e colunas (variáveis).

Características do data.frame: - Cada coluna pode ser de um tipo diferente (uma coluna numérica, outra texto) - Cada coluna é um vetor e deve ter o mesmo comprimento - É como uma lista especial onde todos os elementos têm o mesmo tamanho - Ideal para dados tabulares (como datasets de pesquisa)

```
# Lista: tipos mistos
lst <- list(id = 1, nome = "Ana", aprovado = TRUE)
lst$nome
```

```
# Data frame
alunos <- data.frame(
  id = 1:4,
  nome = c("Ana", "Bruno", "Caio", "Dani"),
  nota = c(8.5, 7.2, 9.1, 6.8),
  ativo = c(TRUE, TRUE, FALSE, TRUE),
  stringsAsFactors = FALSE
)
str(alunos); nrow(alunos); ncol(alunos); names(alunos)
head(alunos, 2); tail(alunos, 2)

# Acesso e novas colunas
alunos$nome
alunos$aprov <- ifelse(alunos$nota >= 7, "Aprovado", "Recuperação")
```

5.4 Fatores

O que são fatores?

Fatores são a forma do R representar **variáveis categóricas** (também chamadas de qualitativas). São usados para dados que podem assumir um número limitado de valores distintos, chamados de “níveis” (levels).

Quando usar fatores: - Variáveis categóricas: sexo (M/F), região (Norte/Sul/Leste/Oeste), tratamento (Controle/Teste) - Variáveis ordinais: nível de escolaridade, grau de satisfação (Baixo/Médio/Alto) - Respostas de questionários com opções fixas

Vantagens dos fatores: - Economizam memória (armazenam códigos internos, não strings repetidas) - Permitem ordenação lógica (ex: Baixo < Médio < Alto) - Facilitam análises estatísticas e gráficos - Controlam quais valores são válidos

Tipos de fatores: - **Nominais** (sem ordem): cores, categorias - **Ordinais** (com ordem): níveis de satisfação, graus acadêmicos

```
sexo <- factor(c("F", "M", "M", "F"), levels = c("F", "M"))
levels(sexo)

conceito <- factor(c("B", "A", "C", "A"), levels = c("C", "B", "A"), ordered = TRUE)
summary(conceito)
```

6 Exploração inicial de dados (40 min)

Vamos usar um dataset real (`palmerpenguins`) para praticar **inspeção e resumo** com diferentes funções, incluindo `dplyr::glimpse()`.

```
install.packages("palmerpenguins")
library(palmerpenguins)
library(dplyr)
```

```

# Visão geral do dataset
str(penguins)          # estrutura detalhada
dplyr::glimpse(penguins) # visão compacta e moderna (tidyverse style)

# Informações básicas
names(penguins)         # nomes das colunas
nrow(penguins)          # número de linhas
ncol(penguins)          # número de colunas
dim(penguins)           # dimensões (linhas x colunas)

# Primeiras e últimas observações
head(penguins)          # 6 primeiras linhas
tail(penguins, 3)        # 3 últimas linhas

# Resumo estatístico
summary(penguins)        # resumo de cada coluna
colSums(is.na(penguins)) # contagem de NAs por coluna

# Selecionar colunas principais (R base)
peng_min <- penguins[, c("species", "bill_length_mm", "bill_depth_mm",
                         "flipper_length_mm", "body_mass_g")]
head(peng_min)

# Criar nova variável: razão do bico
penguins$raz_bico <- with(penguins, bill_length_mm / bill_depth_mm)
head(penguins$raz_bico)

# Estatísticas descritivas
mean(penguins$flipper_length_mm, na.rm = TRUE)
sd(penguins$body_mass_g, na.rm = TRUE)
range(penguins$bill_length_mm, na.rm = TRUE)

# Estatísticas por grupo
tapply(penguins$flipper_length_mm, penguins$species, mean, na.rm = TRUE)
tapply(penguins$body_mass_g, penguins$species, median, na.rm = TRUE)

# Tabelas de frequência
table(penguins$species)
# table(penguins$species, penguins$island)

```

6.1 Comparando str() vs glimpse()

Ambas mostram a estrutura dos dados, mas com estilos diferentes:

```
# str(): estilo tradicional do R, mais verboso
str(penguins)
```

```
# glimpse(): estilo tidyverse, mais compacto e legível
dplyr::glimpse(penguins)
```

Vantagens do `glimpse()`: - Mostra tipo de cada coluna de forma clara - Apresenta primeiros valores de forma compacta - Melhor para datasets com muitas colunas - Estilo moderno e consistente com tidyverse

Dica: quando houver NAs, sempre use `na.rm = TRUE` nas funções de resumo estatístico.

7 Exercícios guiados (20 min)

7.1 Exercício 1 — Vetores

1. Crie um vetor numérico com 8 valores quaisquer.
2. Calcule média, mediana e desvio-padrão.
3. Filtre apenas os valores **acima da média**.

```
# Seu código aqui
```

7.2 Exercício 2 — Data frame

1. Crie um `data.frame` com colunas: `id`, `nome`, `nota`, `ativo`.
2. Crie uma nova coluna `situacao` usando `ifelse(nota >= 7, "Aprovado", "Recuperação")`.
3. Mostre apenas as colunas `nome` e `situacao` das 2 primeiras linhas.

```
# Seu código aqui
```

7.3 Exercício 3 — Exploração palmerpenguins

1. Use `glimpse()` para ter uma visão geral dos dados.
2. Conte quantos NAs existem em cada coluna.
3. Crie uma nova coluna `massa_kg` convertendo `body_mass_g` para quilogramas.
4. Calcule a **média de flipper_length_mm por espécie** usando `tapply()`.

```
# Seu código aqui
```

8 Primeiro commit (5 min)

No Terminal do RStudio:

```
git add scripts/01_fundamentos.R  
git commit -m "Dia 1: fundamentos de R e setup"  
git push origin main
```

Confirme no seu repositório **forkado** no GitHub se o commit apareceu.

9 Checklist de encerramento

- R, RStudio e Git instalados e funcionando.
 - Git configurado com `user.name` e `user.email`.
 - Fork criado** no GitHub e **clone realizado do SEU fork**.
 - Projeto `.Rproj` aberto; função `here()` testada.
 - Entendeu a diferença entre `str()` e `glimpse()`.
 - Script `01_fundamentos.R` criado e salvo em UTF-8.
 - Commit e push realizados com sucesso para **SEU fork**.
-

10 Referências rápidas

- **R for Data Science (2e)**: <https://r4ds.hadley.nz/>
 - **Happy Git with R**: <https://happygitwithr.com/>
 - **Cheatsheets Posit**: <https://posit.co/resources/cheatsheets/>
 - **palmerpenguins**: <https://allisonhorst.github.io/palmerpenguins/>
 - **dplyr documentation**: <https://dplyr.tidyverse.org/>
-

Nos vemos no Dia 2 para explorarmos lógica de programação e tidyverse!