Ebook - Curso Básico de R

Priscilla Normando

2024-07-29

Contents

CURSO BÁSICO DE LINGUAGEM R

Chapter 1

Boas-vindas!

Olá!

Damos as boas-vindas ao curso básico de R do projeto BMClima. Neste espaço você terá acesso a primeira etapa do treinamento para o curso **Análise de Dados de Saúde e Clima: Estatísticas para Políticas Públicas utilizando R** ou para compreender e saber utilizar os scripts, dados e protocolos disponibilizados em nossos repositórios.

Dividimos os trabalhos em três pilares que compreendemos facilitar o trabalho com dados: *organização e versionamento*, *conceitos fundamentais de programação* e *autonomia*.

Organização e versionamento. Começar seu trabalho com dados de forma organizada e mantendo a controle sobre a versão de cada etapa dos processos com os dados faz com que possamos revisar, editar e, caso necessário, recuperar com facilidade o trabalho realizado. Perder arquivos, dados e ideias costuma ser uma fonte de estresse. Mantenha um processo de organização e controle sobre o que está sendo feito e poderá se concentrar naquilo que é importante. Por isso, vamos passar pelas etapas de instalação das ferramentas e configuração de uma espaço de trabalho em que você poderá colaborar com colegas ou apenas manter backups seguro de tudo o que foi feito.

Conceitos fundamentais de programação. Calma! Este não é um curso de programação. Mas saber os conceitos básicos que envolver a construção dos scripts dará a você capacidade de leitura e compreensão do eles fazem e de, por consequência, adaptá-los para suas necessidades. Inclusive, a capacidade de fazer scripts que sejam mais rápido e que exigem computadores mais simples. Além disso, nos cursos e materiais mais avançados você já "falará" a língua dos computadores facilitando seu acesso ao uso de ferramentas e metodologias avançadas na área de dados em saúde.

Autonomia. Ao estudar o que conteúdo deste curso, acreditamos que você será

capaz de encontrar materiais e de contribuir com projetos na área de dados em saúde.

Com isso esperamos dar a você os instrumentos para manipular dados para saúde e políticas públicas utilizando a leitura, escrita, adaptação e aplicação de scripts na linguagem R. Promovendo o uso de dados abertos e dados públicos para pesquisa e tomada de decisão.

1.1 O que você vai aprender

Ao longo do material disponível você terá contato com os fundamentos da programação de computadores aplicada à análise de dados utilizando a linguagem R. Abordaremos temas importantes como a organização de seu trabalho com os códigos e dados, a reutilização de scripts e dados escritos por terceiros e os fundamentos do uso de uma linguagem de programação.

1.1.1 Conceitos fundamentais

- **Introdução ao R**: o que é o R e o que podemos fazer com ele

1.1.2 Seu espaço de trabalho

- **Instalando o R, RStudio, GIT e Colab**: instalação das ferramentas de trabalho e algumas alternativas úteis
- **Boas práticas de organização e versionamento**: configurando as ferramentas de trabalho, diretório do projeto, a ferramenta "projetos" do RStudio e versionadores de código
- **Usando o R**: conheça as interfaces das ferramentas e suas principais funcionalidades
- **Escrevendo e versionando o seu primeiro script**: escreva seu primeiro programa de computador e faça seu primeiro "commit"

1.1.3 Fundamentos

- **Variáveis e constantes**: entenda o que permite uma linguagem de programação manupular dados inseridos (entrada), calcular (processamento) e entregar resultados (saída)
- **Operadores**: use o R como uma calculadora
- **Tipos de dados**: saiba quais tipos de dados podemos usar em R e como tratá-los para suas necessidades
- **Manipulação de strings**: trate caracteres alfanuméricos e encontre insights dentro de textos com o R

- **Estruturas de dados**: como estruturar dados no R
- **Entradas e saídas (input/output)**: as formas de inserir dados no R e encontrar os resultados esperados
- **Dicionários de Dados**: como associar um ou mais tipos de dados no R
- **Estruturas condicionais**: como utilizar o R para realizar tarefas complexas utilizando fluxos de processamento por meio de decisões condicionais
- **Loops**: como utilizar o R para realizar tarefas complexas utilizando fluxos de processamento por meio de repetição de tarefas

1.1.4 O poder do R

- **Pacotes e Scripts de Terceiros**: entenda como reutilizar códigos escritos pela comunidade em seus projetos
- **Manipulação de arquivos**: entenda como ler e escrever arquivos de dados com o R
- **Carregamento de bases de dados**: carregue bases de dados utilizandos boas práticas

1.1.5 Construa seus próprios projetos

- **Inspeção e descritiva de bases de dados**: seu projeto para aplicar tudo o que foi aprendido utilizando estatística descritiva básica

1.2 Como você vai aprender

Para promover sua autonomia de aprendizado e sua capacidade de aprender a partir do material disponibilizado aqui, este curso é baseado em tutoriais e laboratórios.

Os tutoriais possuem explicações seguidas de atividades guiadas de forma tal que você irá aprender praticando o que deve ser feito. **É muito impotante que você siga o passo-a-passo conforme indicado no texto** para que tudo corra bem.

Os laboratórios estarão mais ao final do curso em que você fará a aquisição, leitura e descrição de uma base de dados a sua escolha.

Para entregar seu projeto final você poderá escolher entre duas trilhas: 1. Clonar o repositório do curso no GitHub e seguir o material até a entrega do trabalho. 2. Realizar os laboratórios e enviar os arquivos dos scripts por meio do formulário disponível em https://forms.gle/1WKmHHcJNSx1yrW39

Recomendamos fortemente que você faça o curso utilizando a primeira opção, pois ela fornece uma experiência completa no processo (pipeline) em projetos da área de dados.

1.3 Materiais e referências

Deixamos uma apostila disponível em PDF. Clique aqui para baixá-la!

Ao final dela estão links para materiais de referência como livros, apresentações e vídeos que poderão ajudar na sua jornada com o R.

Você poderá também fazer perguntas por meio do GitHub do curso. Basta abrir uma "issue" clicando aqui.

Figure 1.1: NroH.gif

Vamos aos trabalhos?!

Conceitos básico

Chapter 2

Sobre o R

Chapter 3

Conceitos Fundamentais

Agora que você já sabe o que vamos estudar é hora de conhecer alguns conceitos fundamentais.

3.1 O que é o R?

O R é uma linguagem de programação criada especificamente para cálculos estatísticos e visualização de dados. Foi derivada de outra linguagem vastamente utilizada na estatística, o S. É aberta (open source), o que significa que além de gratuita, possui um ambiente (ou ecossitema) mantido pelo própria comunidade da estatística. O levou a um conjunto de ferramentas poderoso e acessível que torna o trabalho com dados muito mais simples e fácil.

A diferença entre o R e outros ambientes voltados para a área de dados vai depender da ferramenta a ser comparada.

Do ponto de vista das linguagens de programação, há outras que fazem trabalho similar ao R mas não são especializadas em estatística. Esse fator pode levar a problemas como lentidão nos cálculos ou a necessidade de maior quantidade de trabalho para alcançar o mesmo objetivo. No entanto, linguagens como o Python, por exemplo, podem ter mais funções, tais como construção de aplicativos e automatização de processos.

Já quando falamos em ferramentas como SPSS ou Stata, a principal diferença será na capacidade de processamento (veremos que se bem configurado, o R é muito leve e rápido) e a acessibilidade. O ecossistema do R é livre. É possível acessar códigos feitos por outras pessoas, baixar a maior parte das ferramentas e adaptá-las livremente e conseguir ajuda dentro da própria comunidade. Além de um quantidade razoável de livros e tutoriais disponíveis. Ferramentas proprietárias podem possuir capacidade limitada de processamento e necessitarem da compra de licenças.

Outra característica marcante do R é a simplicidade e elegância. Com poucas horas de treinamento (e linhas de código) é possível realizar a carga, limpeza, visualização de bases de dados, além de criar relatórios.

Ok! Mas, exatamente, o que pode ser feito no R?

3.2 O que podemos fazer utilizando o R?

A resposta mais simples é ***cálculos***. De forma mais detalhada podemos afirmar que o R fornece uma grande variedade de técnicas estatísticas disponíveis para uso e que podem ser extendidas de acordo com as necessidades do seu projeto. Por exemplo:

- Modelagem linear e não linear
- Testes estatísticos clássicos
- Classificação
- Grupamento
- Análise de séries temporais

É possível também gerar realtórios, livros técnicos (como este curso), painéis, automatizações e páginas web estáticas.

Há também os pacotes de extensões criadas para tarefas específicas que permitem reutilizar o código de outras pessoas. Essas pessoas trabalham em conjunto para desenvolver e manter novas funcionalidades para o R, são as comunidades. Basicamente, é por meio do uso e contribuição comunitário que o R vem se desenvolvendo ao longo do tempo. Então, uma outra coisa que podemos fazer com o R é contribuir para melhorá-lo.

Um conjunto de ferramentas está disponível para facilitar o nosso trabalho. Destacamos aqui neste curso o RStudio, o Git, o GitHub e o Google Colab.

O ***RStudio*** é considerado o principal software para usar o R. Trata-se de um ambiente integrado de desenvolvimento (IDE). Por meio dele podemos fazer todas as tarefas estatísticas, desde carregar os dados até publicar um documento técnico na internet.

O ***Google Colab*** é uma alternativa em nuvem para o RStudio. Aceita que escrevamos código em R e testemos o que estamos fazendo. Funciona de forma diferente, mas caso você tenha problemas com seu computador ou mesmo em usar o RStudio no computador do trabalho, basta fazer login na ferramenta, realizar pequenas configurações e começar a usar. Não possui o mesmo poder o RStudio, mas permite realizar o trabalho com o R sem a necessidade de instalar softwares adicionais e a flexibidade de podermos utilizá-lo inclusive no celular.

O ***Git*** é um versionador de códigos e textos. Basicamente permite que a gente consiga registrar e rastrear todas as alterações realizadas em um documento

ou conjunto de documentos. Uma outra funcionalidade do Git é permitir que várias pessoas trabalhem no mesmo projeto e que saibamos o que cada uma fez, além de podermos comparar alterações e, caso necessário, recuperar documentos para suas versões anteriores. Sabe quando você estava escrevendo um artigo, trabalho ou TCC, fez anotações ótimas, mas acabou apagando? Com o Git é possível manter a versão final do trabalho e encontrar as anotações que estavam em versões anteriores, mesmo com alterações realizadas. *Uma curiosidade*: é que é essa tecnologia que permite a existência de aplicativos Google Docs e o Office 365.

O ***GitHub*** é um site (plataforma) voltada para o desenvolvimento de código. Permite que busquemos e compartilhemos códigos, relatório, documentos técnicos. Em sua interface é possível automatizar processos, usar edito de códigos com assistente de IA inclusa, fazer backups e colaborar com a comunidade e com sua equipe de trabalho. Funciona baseado em Git, ou seja, é como se fosse o RStudio para o R.

Calma, calma, calma! Se você está com dúvidas sobre essas ferramentas, basta continuar o curso. Veremos cada uma delas com mais detalhes.

3.2.1 *Hora de começar a colocar a mão na massa!*

Figure 3.1: image.png

3.3 Saiba mais!

Uma descrição mais detalhada sobre o ambiente de desenvolvimento com R pode ser encontrada no site oficial do projeto.

3.4 Materiais e referências

Site oficial do R

Chapter 4

Instalação e Configuração

Chapter 5

Seu espaço de trabalho

No capítulo anterior falamos um pouco sobre as ferramentas que utilizaremos. Agora vamos instalar e começar a usar.

Caso tenha pulado alguma coisa, pedimos que volte e leia atentamente.

A primeira coisa que precisamos fazer aqui é combinar algo ***muito importante***: *você precisa seguir o passo-a-passo na ordem indicada*. Caso não faça isso, as coisas provavelmente não vão funcionar. Essa etapa do trabalho pode ser desafiadora, por isso tenha calma e persistência.

Com isso já aprendemos um princípio fundamental da área de TI, "comece organizado, mantenha organizado e termine organizado"! Arquivos, documentos e código bagunçados serão uma dor de cabeça infindável na sua vida. Neste capítulo, você vai aprender a organizar seu trabalho em projetos ou em uma pasta específica com uma hierarquia clara e configurações que melhor atendem a um iniciante.

Seguir os passos na ordem indicada também é importante para que, caso haja algum problema, você possa pedir ajuda com mais facilidade.

Combinado? Então, vamos lá!

Chapter 6

Instalando as ferramentas

6.1 1. Instalando o R

O R é instalado a partir de um conjunto de servidores conhecidos como CRAN. Neles ficam disponíveis os códigos e documentação da linguagem. Recomendamos que você use sempre o servidor geograficamente mais próximo. Por exemplo, se está em São Paulo é melhor baixar os arquivos dos servidores da USP.

> *Os arquivos de instalação estão disponíveis para os sistemas operacionais Windows, Linux e MACOS.* Caso não saiba qual seu sistema confira essa informação antes de baixar os arquivos. Caso preciso de alguma instalação em seu computador do trabalho, converse com o pessoal de suporte de TI da sua instituição.

6.1.1 1.1. Passo-a-passo no Windows

- Entre no seguinte link https://cran.r-project.org/mirrors.html.
- Clique no servidor mais próximo a você.
- Clique em Download R for Windows
- Escolha a opção "base"
- Uma nova página será aberta. Escolha a última versão do R e a pasta de destino do arquivo de instalação.
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela.

  Reinicie o Windows e verifique se o R entre os programas instalados.

6.1.2 1.2. Passo-a-passo no Linux

- Entre no seguinte link https://cran.r-project.org/mirrors.html.
- Clique no servidor mais próximo a você.
- Clique em Download R for Linux escolhendo a distribuição equivalente ao seu sistema operacional.
- Siga o processo de instalação equivalente a sua distribuição.

  Os usuários de Linux normalmente são mais experientes ou precisam buscar as formas de instalação dos pacotes em seus sistemas operacionais.

  Teste se a instalação aconteceu verificando a versão do R via terminal.

Abaixo colocamos o exemplo de instalação para Ubuntu > 22.02 (Jelly Fish)

```
# update indices
sudo apt update -qq
# install two helper packages we need
sudo apt install --no-install-recommends software-properties-common dirmngr
# add the signing key (by Michael Rutter) for these repos
# To verify key, run gpg --show-keys /etc/apt/trusted.gpg.d/cran_ubuntu_key.asc
# Fingerprint: E298A3A825C0D65DFD57CBB651716619E084DAB9
wget -qO- https://cloud.r-project.org/bin/linux/ubuntu/marutter_pubkey.asc | sudo tee
# add the R 4.0 repo from CRAN -- adjust 'focal' to 'groovy' or 'bionic' as needed
sudo add-apt-repository "deb https://cloud.r-project.org/bin/linux/ubuntu $(lsb_releas
# Install R base
sudo apt install --no-install-recommends r-base
```

6.1.3 1.3. Passo-a-passo no MacOS

- Entre no seguinte link https://cran.r-project.org/mirrors.html.
- Clique no servidor mais próximo a você.
- Clique em Download R (Mac) OS X
- Escolha a opção correspondente a sua versão do sistema operacional
- Um arquivo .pkg será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela.

  Reinicie o computador e verifique se o R entre os programas instalados.

6.2 2. Instalando o RStudio

O segundo passo é instalar o RStudio.

> Lembre-se de instalar o R antes.

6.2.1 2.1. Passo-a-passo no Windows

- Entre no seguinte link https://posit.co/download/rstudio-desktop/#download.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

> Se você deseja que o aplicativo esteja disponível para todos os usuários do computador, precisará fazer a instalação com uma conta de administrador

> Reinicie o Windows e verifique se o R entre os programas instalados

6.2.2 2.2. Passo-a-passo no Linux

- Entre no seguinte link https://posit.co/download/rstudio-desktop/#download.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

Abaixo colocamos o exemplo de instalação para Ubuntu > 22.02 (Jelly Fish)

```
# update indices
sudo apt update
# install gdebi-core
sudo apt install --no-install-recommends gdebi-core
# download deb pack
wget -qO- https://download1.rstudio.org/electron/jammy/amd64/rstudio-2024.04.2-764-amd64.deb
```

```
# Open terminal on download folder
# Install dependencies
wget http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.1_1.1.0g-2ubuntu4_amd
sudo dpkg -i libssl1.1_1.1.0g-2ubuntu4_amd64.deb
# Install pack from file
sudo gdebi rstudio-2024.04.2-764-amd64.deb
# Open RStudio
rstudio
```

6.2.3 2.3. Passo-a-passo no MacOS

- Entre no seguinte link https://posit.co/download/rstudio-desktop/#download.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

  Reinicie o computador e verifique se o R entre os programas instalados.

6.3 3. Instalando o Git

O terceiro passo é instalar o Git.

6.3.1 3.1. Passo-a-passo no Windows

- Entre no seguinte link https://git-scm.com/download/.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

  Se você deseja que o aplicativo esteja disponível para todos os usuários do computador, precisará fazer a instalação com uma conta de administrador

  Reinicie o Windows e verifique se o R entre os programas instalados

6.3.2 3.2. Passo-a-passo no Linux

- Entre no seguinte link <hhttps://git-scm.com/download/>.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

  Teste se a instalação aconteceu verificando a versão do Git via terminal.

Abaixo colocamos o exemplo de instalação para Ubuntu > 22.02 (Jelly Fish)

```
# update indices
sudo apt update
# install git
sudoapt-get install git
```

6.3.3 3.3. Passo-a-passo no MacOS

- Entre no seguinte link https://git-scm.com/download/.
- Clique no arquivo correspondente ao seu sistema operacional
- O arquivo será baixado
- Vá até a pasta de destino, clique no arquivo e siga as instruções de instação em tela

  Reinicie o computador e verifique se o R entre os programas instalados.

6.4 4. Criando uma conta no GitHub

O quarto passo é criar uma conta no GitHub.

- Entre no seguinte link https://github.com/signup.
- Siga as instruções em tela
- Confirme sua conta

O GitHub possui uma aplicação desktop que funciona apenas no Windows. Se esse for o seu sistema operacional, sugiro que você visite https://desktop.github.com/download/ e experimente a ferramenta.

Caso você utilize Linux o MacOS, veremos mais à frente como utilizar Git e GitHub utilizando RStudio e Google Colab.

6.5 5. Criando uma conta no Google Colab

O quinto passo é criar uma conta no Google Colab.

- Entre no seguinte link https://colab.research.google.com/.
- Clique na opção Signin (botão azul na parte superior da tela à sua direita)
- Siga as instruções em tela
- Confirme sua conta

Por padrão o Google Colab está configurado para ser usado com a linguagem Python, mas nas próximas sessões vamos ver como configurar o ambiente para usar R.

6.6 6. Instalando ferramentas adicionais

No RStudio e Google Colab é possível instalar ferramentas adicionais que podem melhorar a experiência com o R. Dentre elas estão o Latex (formatação de fórmulas matemáticas), o webshot (para geração de páginas web e pdf), o bookdown (para criação de documentos técnicos).

Veremos algumas delas ao longo deste curso, mas se você quiser explorar um pouco do que há disponível para o R, clique aqui.

Chapter 7

Boas práticas de organização e versionamento

Agora que você possui as ferramentas instaladas, vamos fazer as configurações. Ao longo do processo também apresentarei algumas boas práticas que servirão para melhorar a qualidade e o processo de trabalho com o R.

Vamos conhecer a interface do RStudio e usar a ferramenta "projetos" da IDE.

7.1 A interface do RStudio

Vamos conhecer a interface do RStudio.

Abra o programa em seu computador. Assim que ele inicializar,você deverá ver algo como a figura abaixo. No início é normal se sentir um pouco perdide e não compreender completamente todas as opções disponível. Tenha tranquilidade que de pouco em pouco, com o uso e a prática, vamos entendendo as funcionalidades e opções disponíveis.

Figure 7.1: image.png

Observe que há quatro janelas diferentes. Em sentido anti-horário e de cima para baixo, você verá uma janela com o editor de scripts, uma janela onde há o console, uma que mostra o que está "rodando" no ambiente no R e outra mostrando as saídas (arquivos do projeto, visualizações, pacotes instalados, ajuda e permissões).

No editor de scripts escrevermos os códigos (scripts) que serão responsáveis por realizar os cálculos e outras funções que programamos.

No console executamos os códigos e recebemos as saídas.

As outras janelas servem a nos auxiliar. Mostram saídas de gráficos, quais variáveis estão no ambiente, dados carregados, pacotes instalados, etc.

Acima das quatro janelas descritas estão as opções do RStudio.

Como falamos acima, aos poucos vamos entendendo como tudo funciona. Nas próximas seções faremos alguns exercícios que ajudam a explorar e entender melhor as funcionalidades do RStudio.

7.1.1 Algumas configurações importantes

.RData e .Rhistory

O RStudio possui duas funcionalidades que são muito boas para usuários mais experientes, mas que podem atrapalhar o aprendizado do R. Por isso é recomendado que sejam desativadas, por enquanto.

Para isso vá na opção ferramentas (tools) no menu superior e depois em opções gerais (general options). Desmarque a opção *Restore .RData* e *Always save history*.

Suas opções devem ficar como na image abaixo:

Figure 7.2: image.png

Gerenciando projetos

O RStudio possui uma ferramenta chamada "projetos". Ela gera um arquivo chamado .RProject. Nele ficam guardados dados sobre os arquivos que você cria e usa, ou seja, é um arquivo de metadados do projeto em que está trabalhando.

Tá, mas por que isso é importante?

Lembra quando falei de sabermos quando e como as alterações foram realizadas? Ao guardarmos os metadados, podemos manter as versões de nosso trabalho de forma mais segura e garantir que os arquivos criados para uma determinada finalidade estejam sempre na mesma pasta. Nada de perder arquivos depois de horas, porque eles ficaram em alguma pasta temporária do sistema.

Para fazer isso, primeiro vamos alterar as opções no RStudio para que ele sempre guarde os arquivos em uma pasta que sabemos onde está. Para isso siga o seguinte passo-a-passo:

1. Vá em tools (ferramentas) -> general options -> RSessions -> Browser
2. Uma nova janela vai se abrir

3. Clique em New Folder
4. Da mesma forma que você cria novas pastas no sistema operacional, crie uma pasta exclusiva para seus projetos do RStudio. Eu costumo chamar essa pasta de projetosR.
5. Clique em choose (escolher)
6. Cliqeu em applay e depois OK

Agora o RStudio sempre salvará seus arquivos dentro desta pasta.

Figure 7.3: image.png

Uma vez que sua pasta de trabalho padrão está criada. Cada vez que criar um projeto, ele será uma subpasta do diretório principal.

Para criar um projeto vá até o menu superior e clique em *file -> new project*. O programa dará as opções de criar uma nova pasta no computador, de usar um pasta já existente ou de conectar-se a um repositório GIT (incluindo repositórios no GitHub).

Lembra que um dos pilares desse curso é despertar a autonomia no uso do R e suas ferramentas? Agora é sua vez de explorar a interface do RStudio.

Exercício: criar três projetos diferentes, um para cada opção disponível no RStudio.

Figure 7.4: image.png

Agora que nosso ambiente de trabalho está configurado e testado. Vamos começar a usar o R dentro do RStudio.

Chapter 8

Usando o R

Vamos explorar cada uma das janelas? Então faremos um exercício obrigatório. 1. Abra o RStudio e crie um arquivo chamado primeiroScript. 2. Depois salve, usando o comando ctrl+s ou o ícone de disquete na janela de scripts. 3. Agora observe o que mudou na interface da sua IDE. 4. Na janela de scripts copie e cole esse código:

```
x <- 2 + 4
print(x)
```

5. Em seguida, selecione o código e clique em RUN.

Na janela do console (na parte inferior a sua esquerda) deve ter aparecido o número 6, conforme a imagem abaixo:

Figure 8.1: image.png

Chapter 9

Escrevendo o seu primeiro script

Notou que o RStudio mostrou a soma entre 2 e 4? Isso porque o R pode funcionar como uma calculadora.

Exercício: modifique o código de forma que ele imprima a multiplicação entre 2 e 4.

Parabéns! Você acaba de fazer o seu primeiro programa de computador. Um script em R.

No próximo capítulo vamos aprofundar seus conhecimentos na linguagem R. Te aguardamos lá!

Chapter 10

Algumas dicas úteis

10.1 Atalhos do RStudio

Os atalhos são comando que fazemos a partir do teclado e que nos permitem trabalhar de forma mais rápida e assertiva. Abaixo estão os atalhos mais comuns.

- CTRL+ENTER: avalia/roda a linha selecionada no script
- ALT+-: cria no script um sinal de atribuição (<-)
- CTRL+SHIFT+M: (%>%) operador pipe
- CTRL+1: altera cursor para o script
- CTRL+2: altera cursor para o console
- ALT+SHIFT+K: janela com todos os atalhos disponíveis

OBS: *No MacOS, substitua o CTRL por command e o ALT por option.*

Chapter 11

Materiais e referências

- Livro o Zen do R
- Documentação do R

Chapter 12

Fundamentos de programação

Neste capítulo do curso, vamos abordar os fundamentos da programação aplicados ao R.

Programar um computador significa dizer a ele por meio de um conjunto de textos especiais (códigos) o que desejamos que seja feito. Ao escrever um código (script) precisamos ter em mente que o computador só fará aquilo que mandamos. E, como uma máquina, suas capacidades de compreensão são bem limitadas. Por isso, usamos algumas técnicas para que possamos nos fazer "entender" pelo computador.

A melhor parte é que quando aprendemos a "dizer" para o computador o que queremos, somos capazes de automatizar tarefas repetitivas, processar grandes quantidades de dados e realizar cálculos longos.

Mantenha em mente que o que queremos é entrar com os dados, que o computador processe conforme solicitamos e nos devolva a saída de dados ou informações de acordo com o desejado.

ENTRADA -> PROCESSAMENTO -> SAÍDA

No R utilizamos essas mesmas técnicas. Nas próximas sessões daremos as orientações e exemplos aplicados no próprio R para que você possa entender e praticar.

12.1 Variáveis e constantes:

Para que um programa de computador funcione, é precisa entrar com os dados que desejamos. Isso pode ser feito duas maneiras:

1. Inserir manualmente cada campo.
2. Carregar um conjunto de dados.

Um exemplo de inserção manual são as calculadoras, onde colocamos um número, depois inseridos a operação que queremos e depois outro número e assim por diante.

Um exemplo de carga de dados é seu extrato bancário. Já notou que o extrato é um relatório de todas as transações registradas em certo período de tempo?

Da mesma maneira, os dados coletados durante atendimentos em uma UBS, notificados e depois disponibilizados via DataSUS também são exemplos de registros que podem se tornar relatórios.

É aí que entra a segunda parte, o processamento. Quando você coloca o seu login e senha no aplicativo do banco e solicita o extrato, o computador processa a sua solicitação e imprime os valores na tela ou gera um PDF. Da mesma maneira, com o R podemos construir scripts que nos respondam à perguntas que possuímos sobre os registros disponíveis nas bases de dados de clima e de saúde.

Então, é possível solicitar ao computador que realize as operações necessárias com os dados carregados? Sim. Cada vez que executamos (rodamos) um script, acontece um processamento.

O relatório é a saída do processamento. O seu extrato bancário possui operações consideradas simples, pois trata-se de uma consulta com adição e subtração das operações. Compreender as relações entre a temperatura do ar e morbimortalidade nos exigirá um pouco mais de esforço, mas o fundamento é sempre o mesmo.

Rode (execute) o script abaixo e veja a diferença no valor de x, a cada vez que você atribui um novo valor. (Você pode usar o RStudio ou o Google Colab.)

```
x<-5
print(x)
```

```
[1] 5
```

```
x<-10
print(x)
```

```
[1] 10
```

```
x<-4500
print(x)
```

```
[1] 4500
```

Notou que cada vez que você muda o valor à sua direita, o valor em tela também muda? Por isso chamamos o x, ou qualquer outra letra ou nome à sua esquerda de variável.

Seja qual for a operação que você está fazendo no computador (entrada, processamento ou saída), será necessário "guardar" temporariamente os valores utilizados no cálculo ou consulta. Para isso dizemos ao computador que queremos "declarar uma variável".

Uma variável é uma unidade representada por letra ou números que pode ter seu valor modificado ao longo do processamento. Para isso usamos algo que você certamente conhece, equações.

Vamos a um exemplo:

$x = 3$

Se você ler a expressão acima em voz alta, deverá dizer que *xis é igual a três*. Não é mesmo?

No R, fazemos algo muito parecido, apenas precisamos deixar diferente a igualdade porque na verdade estamos atribuindo um valor para x que poderá ser modificado. Por isso no R utilizamos o símbolo <- para dizer que naquele momento *xis é igual a três*. Segue o exemplo:

x<-3

Pronto, você disse ao R que *a xis está atribuido o número inteiro três*, mas que isso pode mudar. Foi exatamente isso que acabamos de fazer nos exemplos acima. Utilizamos números, mas podemos atribuir qualquer caractere ou conjunto de dados a uma variável. Veremos isso com detalhes mais à frente.

E quando precisamos que um valor atribuído não seja modificado durante o processamento? Utilizamos as constantes. Como o próprio nome diz, ao contrário da variável ela não poderá ter seu valor atualizado.

As constantes são importantes pela clareza, facilidade de manutenção, prevenção de erros, segurança e melhoria no desempenho dos nossos scripts.

No R é comum utilizarmos funções para declarar constantes como o PI e E (constante de euler) ou valores vazios e nulos. No final dessa seção você vai encontrar uma lista com os tipos de contantes mais utilizadas pela linguagem.

Esses valores, por convenção, não serão modificados então os declaramos da seguinte forma:

```
PI <- 3.14159
E <- 2.71828
```

Por boas práticas, as variáveis sempre são declaradas com letras minúsculas e as constantes com letras maiúsculas.

Rode os exemplos abaixo e veja como se comportam as constantes.

```
PI <- 3.14159
E <- 2.71828
print(PI)
print(E)
```

```
[1] 3.14159
[1] 2.71828
```

12.2 Operadores

Agora que já sabemos o que são variávis e constantes, assim como declará-las utilizando o R, vamos ver como fazer operações.

Algumas pessoas costumam dizer que o R pode ser usado como uma calculadora, mas podemos fazer muito mais que isso. O primeiro passo é entender os tipos de operação disponíveis na linguagem.

Basicamente, podemos realizar qualquer operação com dados utilizando o R e armazená-las em uma ou mais variáveis. Podemos utilizar os seguintes tipos de operadores:

Operadores aritméticos:

`Soma (+), subtração (-), multiplicação (*), divisão (/), módulo (%%), exponenciação (^`

Operadores relacionais:

`Igual a (==), diferente de (!=), maior que (>), menor que (<), maior ou igual a (>=), m`

Operadores lógicos:

`E (&&), OU (||), NOT (!).`

12.2.1 Operadores aritméticos

Os operadores aritméticos são:

- Soma (+)

- Subtração (-)
- Multiplicação (*)
- Divisão (/)
- Módulo (%%)
- Exponenciação (^)

Para fazer operações aritméticas, basta utilizar o símbolo correpondente diretamente com os números ou declarar variáveis. Abaixo seguem alguns exemplos.

OBS: Note que para imprimir os resultados precisamos utilizar o 'print()'. Em breve você conhecerá a fundo como esse tipo de comando funciona. Por enquanto, é só clicar em RUN.

```
x <- 50
y <- 17
```

```
print(x + y)
```

```
[1] 67
```

```
print(x - y)
```

```
[1] 33
```

```
print(x * y)
```

```
[1] 850
```

```
print(x / y)
```

```
[1] 2.941176
```

```
print(x %% y)
```

```
[1] 16
```

```
print(x ^ y)
```

```
[1] 7.629395e+28
```

Exercício: Experimente trocar algumas vezes o valor de **x** e de **y** e rodar os comandos novamente.

Aqui precisamos que você note que para trocar o valor da variável você necessita atribuí-la novamente clicando em RUN.

12.2.2 Operadores relacionais:

Os operadores relacionais fazem comparações entre as variáveis ou dados que estão carregados no sistema. São eles:

- Igual a (==)
- Diferente de (!=)
- Maior que (>)
- Menor que (<)
- Maior ou igual a (>=)
- Menor ou igual a (<=).

Para exemplificar vamos utilizar as mesmas variáveis carregadas acima. Note que diferente dos valores impressos nos exemplos anteriores, aqui o R vai apenas dizer se a comparação é verdadeira (TRUE) ou falsa (FALSE). É o que chamamos de operação booleana.

```
print(x == y)
```

```
[1] FALSE
```

```
print(x != y)
```

```
[1] TRUE
```

```
print(x > y)
```

```
[1] TRUE
```

```
print(x < y)
```

```
[1] FALSE
```

```
print(x >= y)
```

```
[1] TRUE
```

```
print(x <= y)
```

```
[1] FALSE
```

Exercício: Experimente atribuir o valor de **x** e de **y** antes das operações relacionais e rodar os comandos novamente.

Aqui precisamos que você note que é possível declarar novamente o valor de uma variável ao longo do mesmo script. Isso é importante porque o R lê as linhas de código na sequência em que estão escritas e isso pode mudar os resultados obtidos em scripts e operações mais complexas.

12.2.3 Operadores lógicos

Assim como os operadores relacionais, os operadores lógicos retornam apenas valores booleanos. São eles:

- E (&)
- OU (|)
- NOT (!)

Se estamos trabalhando com conjuntos, o R também possui operadores específicos para fazermos testes lógicos.

x pertence a y * x %in% y

x ou y são verdadeiros (apenas um deles) * xor(x, y)

```
print(x & y)
```

```
[1] TRUE
```

```
print(x | y)
```

```
[1] TRUE
```

```
print(!y)
```

```
[1] FALSE
```

```
# Atribuindo um conjunto de dado a uma variável chamada 'z'
z <- c(12,3,5,9)
```

```
x %in% z
y %in% z
```

FALSE

FALSE

12.3 Tipos e estruturas de dados

O R possui uma variedade de tipos de dados que podem ser utilizados. Até agora vimos apenas os tipos numéricos, mas basicamente qualquer representação que conseguirmos fazer com um caracter é possível ser tratada e abordada. Abaixo segue uma lista com os principais tipos que utilizamos no dia-a-dia:

1. **Numéricos**:

   - Inteiros (`integer`): Números inteiros, como 1, 2, -3, 0.
   - Reais (`numeric` ou `double`): Números reais, como 3.14, -2.5, 0.0.

   Os números reais devem ser representados utilizando '.' e não ',' para marcar as casas decimais. Por exemplo, x <- 2.5 e não x <- 2,5.

   ```
   x <- 25,98372652 (errado)
   x <- 25.98372652 (correto)
   ```

   Para declarar números inteiros é necessário colocar um L logo depois do número. "' num_parti <- 25 (errado) num_parti <- 25L (correto)
   "'

2. **Caracteres** (`character`):

   - Strings de texto, como "Olá, mundo!", 'R é incrível'.

   Os caracteres sevem para que possamos trabalhar com textos. Eles devem sempre ser colocar entre aspas.

   ```
   x <- Olá, mundo! (errado)
   x <- 'Olá, mundo!' (correto)
   x <- '123' (Aqui o 123 será tratado como texto.)
   ```

3. **Lógicos** (`logical`):

   - Valores booleanos, como `TRUE` e `FALSE`.

   Como vimos na seção anterior, os tipos lógicos servem para fazermos comparações entre valores ou mesmo atribuir esse valor a um campo ou unidade de conjunto.

   ```
   exame_realizado <- TRUE
   exame_realizado <- FALSE
   ```

4. **Fatores** (`factor`):

- Variáveis categóricas, como “masculino”, “feminino”, “alto”, “baixo”.

Utilizamos fatores quando vamos trabalhar categorização de colunas em uma dataframe. Por exemplo, queremos atribuir um nome a determinada escala de valor.

```
genero <- factor(c("Masculino", "Feminino", "Masculino"))
```

5. **Datas e Horas**:

   - `Date`: Datas, como “2023-04-18”.
   - `POSIXct` e `POSIXlt`: Datas e horas, como “2023-04-18 14:30:00”.

   As datas são variáveis importantes em qualquer estudo em que precisamos estabelecer um período de tempo. Por exemplo, quando queremos descobrir qual foi a precipitação ou a temperatura em determinado mês ou ano. Por isso precisam ser cuidadosamente tratadas antes que realizemos qualquer cálculo. O R possui três funções básicas (nativas) para tratamento de datas. `date` nos permite tratar apenas as datas sem hora ou registro de sistema (timestamp). POSIXct e POSIXlt nos permitem utilizar aqueles registros das bases que veem compostos com hora, dia e outras informações de sistema. Muitas vezes precisamos saber quando um determinado evento ocorreu ou foi registrado, em geral, esse registro vem com essas informações suplementares, também conhecidas como `timestamp`. Ao longo do curso você terá contato com vários exemplos que incluem datas e timestamps.

   ```
   data_nascimento <- as.Date("2022-03-15")

   reg_atendimento <- as.POSIXct("2023-10-27 15:30:00")
   ```

   A diferença entre ***POSIXct*** e ***POSIXlt*** está na forma como eles representam datas e horas no R.

   POSIXct: POSIXct é um tipo de dado numérico que armazena datas e horas como o número de segundos desde 1º de janeiro de 1970, 00:00:00 UTC (Tempo Universal Coordenado). Esse formato é amplamente utilizado em sistemas computacionais, pois é fácil de manipular e calcular diferenças entre datas. POSIXct é o formato recomendado para a maioria das aplicações, pois é mais eficiente em termos de memória e processamento.

   POSIXlt: POSIXlt é um tipo de dado de lista que armazena datas e horas de uma forma mais detalhada, com campos separados para ano, mês, dia, hora, minuto, segundo, etc. Esse formato é útil quando você precisa acessar ou manipular componentes individuais da data e hora, como extrair o dia da semana ou o mês. POSIXlt é mais flexível, mas também consome mais memória e é menos eficiente em termos de processamento do que POSIXct.

Veja os exemplos de uso abaixo:

```
# Criando uma data e hora usando POSIXct
data_hora_ct <- as.POSIXct("2023-04-18 14:30:00")
class(data_hora_ct)
# [1] "POSIXct" "POSIXt"
# Criando uma data e hora usando POSIXlt
data_hora_lt <- as.POSIXlt("2023-04-18 14:30:00")
class(data_hora_lt)
# [1] "POSIXlt" "POSIXt"

# Acessando componentes individuais
data_hora_lt$year
# [1] 123
data_hora_lt$mon
# [1] 3
data_hora_lt$mday
# [1] 18
```

Note que aqui você terá possibilidades diferentes com os dados. No POSIXct é possível fazer a localização do dado por meio do registro no sistema, enquanto no POSIXlt você poderá extrair de forma eficiente dia, mês e ano sempre precisar criar colunas extra em seu conjunto de dados.

6. **Vetores** (`vector`):

   - São coleções de valores do mesmo tipo de dado. Utilizamos a seguinte marcação (função) para declará-los.

```
numeros <- c(1, 2, 3, 4, 5)
letras <- c('a', 'b', 'c')
```

ESTRUTURAS DE DADOS

7. **Matrizes** (`matrix`):

   - São estruturas bidimensionais de dados do mesmo tipo. Se pensarmos em outros programas de computador, podemos dizer que são como tabelas, possuem um número específico de linhas e colunas.

   Observe o exemplo abaixo:

```
matriz <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
```

Ao executar esse script, o R criará uma variável chamada matriz que contém os números de 1 a 6 armazenados em uma tabela com 2 linhas (nrow) e três colunas (ncol). Ficando da seguinte forma:

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

OBS: Até aqui, você só precisa compreender que essa é função disponível no R, ao longo do curso e com o uso da ferramenta vamos aprofundando nos usos das matrizes e outras estruturas de dados.

8. **Listas** (`list`):

   - São coleções de objetos de diferentes tipos de dados. Enquanto as matrizes só atribuição dos mesmos tipos de dados, as listas nos permitem realizar a mesma tarefa "mesclando" tipos diferentes de dados. Observe o exemplo abaixo:

   ```
   programas_saude <- list("Programa de Vacinação" = list(objetivo = "Aumentar a cobertura vaci
   ```

   Note que agora é possível colocar diferentes categorias atribuídas a uma mesma variável, incluindo qualquer tipo de dado. Em outras linguagens de programação, como o Python, isso também é chamado de dicionário de dados. Com eles podemos interagir com diversas categorias, nomear elementos dentro de um dataset e outras funcionalidade muito úteis para preparação e manipulação de dados.

   Se você imprimir a variável `programas_saude` imprimir no console utilizando print() verá a seguinte saída.

   ```
   $`Programa de Vacinação`
   $`Programa de Vacinação`$objetivo
   [1] "Aumentar a cobertura vacinal"

   $`Programa de Vacinação`$publico_alvo
   [1] "Crianças e idosos"

   $`Programa de Prevenção ao Tabagismo`
   $`Programa de Prevenção ao Tabagismo`$objetivo
   [1] "Reduzir o número de fumantes"

   $`Programa de Prevenção ao Tabagismo`$publico_alvo
   [1] "Adolescentes e adultos"

   $`Programa de Saúde Mental`
   $`Programa de Saúde Mental`$objetivo
   [1] "Melhorar o acesso a serviços de saúde mental"

   $`Programa de Saúde Mental`$publico_alvo
   [1] "População em geral"
   ```

9. **Dataframes** (`data.frame`):

    - Enquanto as listas e matrizes se assemelham às tabelas, os dataframes são semelhantes às planilhas. A diferença entre uma lista e um dataframe é de que cada categoria vai corresponder a uma coluna e não a uma linha de dados. Dessa maneira, é possível armazenar uma quantidade maior de dados, homegeneizá-los e transformá-los. Observe o exemplo abaixo:

```
pacientes <- data.frame(
  nome = c("João Silva", "Maria Oliveira", "Pedro Souza", "Ana Pereira"),
  idade = c(45, 32, 58, 27),
  diagnostico = c("Diabetes", "Hipertensão", "Asma", "Gripe"),
  tratamento = c("Medicação", "Medicação", "Medicação", "Repouso"),
  data_admissao = as.Date(c("2023-04-01", "2023-03-15", "2023-02-20", "2023-04-05"
```

Note que agora todos os nomes estão agrupados na categoria nome, assim como as idades, diagnósticos e assim por diante. Quando declaramos uma lista fazermos o caminho contrário, todas as características do objeto que estamos descrevendo ficam agrupadas junto a ele.

Se você imprimir a variável `pacientes` imprimir no console utilizando print() verá a seguinte saída.

```
            nome idade diagnostico tratamento data_admissao
1     João Silva    45    Diabetes  Medicação    2023-04-01
2 Maria Oliveira    32 Hipertensão  Medicação    2023-03-15
3    Pedro Souza    58        Asma  Medicação    2023-02-20
4    Ana Pereira    27       Gripe    Repouso    2023-04-05
```

Agora é possível ver os dados tabulados e organizados em formato de linhas e colunas.

EXPRESSÕES E FUNÇÕES

10. **Expressões** (`expression`):

    - O R nos permite escrever a representação de expressões matemáticas de forma nativa. Basta usar a função expression.

11. **Fórmulas** (`formula`):

    - Representação de modelos estatísticos também é realizada com facilidade, por meio da expressão formula.

12. **Funções** (`function`):

- Enquanto linguagem de programação é possível criar conjuntos de scripts que podem ser chamados em conjunto apenas com um comando. Sabe o print(), o c(), o list(), o vector() e o data.frame() são todos funções que já vem instaladas com o R. A parte mais interessante é que você pode criar as suas próprias funções, de acordo com as suas necessidades. Mais adiante vamos nos aprofundar nisso.

Exercício:

1. Imagine que você está realizando a gestão de três programas de saúde em uma secretaria estadual. Utilizando o RStudio, escreva um script que realize as seguintes tarefas:

   Dados de entrada (variáveis):

   - Nome dos programas
   - Objetivos dos programas
   - Público-alvo dos programas
   - Orçamento anual de cada programa
   - Datas de início e fim de cada programa

   Tarefas:

   - Criar uma lista com as informações de cada programa
   - Calcular o orçamento total destinado a todos os programas
   - Identificar o programa com o maior orçamento
   - Listar os programas que estão em andamento atualmente

Desafio: - Exibir o programa com a data de início mais recente

Orientações:

```
- O primeiro objetivo desta atividade é treinar a atribuição de tipos de dados às variáveis e cor
- O segundo objetivo é exercitar o uso de operadores. Então, busque testar os operadores que melh
- Procure fazer os exercícios antes de procurar por soluções prontas. Lembre-se que não estamos r
```

Chapter 13

Algumas dicas úteis

13.1 Lista de Constantes Nativas do R

Como vimos acima as contantes são valores que não se modificam ao longo do processamento. Facilitam o processamento e a escrita dos códigos. Apenas esse tema já daria por si só um capítulo ou mesmo um curso.

Abaixo deixamos uma lista de contantes que já vem junto com o R que podem facilitar muito o nosso trabalho. Aos poucos, vá pesquisando como utilizá-las e experimentando compor os seus scripts com elas. Dois exemplos úteis, são as contantes de valor ausente (NA) e valos não numérico (NaN), assim como as contantes que fazem a leitura de caracteres latinos (encoding).

1. **Constantes matemáticas**:
   - `pi`: Valor de pi (aproximadamente 3.14159)
   - `e`: Valor da constante de Euler (aproximadamente 2.71828)
2. **Constantes lógicas**:
   - `TRUE`: Valor lógico verdadeiro
   - `FALSE`: Valor lógico falso
   - `NA`: Valor ausente (Not Available)
   - `NaN`: Valor não numérico (Not a Number)
3. **Constantes de precisão**:
   - `.Machine$double.eps`: Menor número positivo `x` tal que `1 + x != 1`
   - `.Machine$double.xmin`: Menor número positivo normalizado
   - `.Machine$double.xmax`: Maior número finito positivo
4. **Constantes de sistema**:

- `.Platform$OS.type`: Tipo de sistema operacional (Windows, Unix, etc.)
- `.Platform$GUI`: Interface gráfica do usuário (X11, Aqua, RStudio, etc.)
- `.Platform$endian`: Ordem de bytes do sistema (little-endian ou big-endian)

5. **Constantes de datas e horas**:

   - `Sys.Date()`: Data atual do sistema
   - `Sys.time()`: Hora atual do sistema
   - `as.Date("1970-01-01")`: Data de referência (1 de janeiro de 1970)

6. **Constantes de codificação de caracteres**:

   - `Encoding("UTF-8")`: Codificação de caracteres UTF-8
   - `Encoding("Latin-1")`: Codificação de caracteres Latin-1 (ISO-8859-1)

OBS: Esses são apenas alguns exemplos das contantes mais utilizadas.

Chapter 14

Materiais e referências

- Livro MQ
- Documentação do R

Chapter 15

Fundamentos de programação

Neste capítulo do curso, vamos continuar a abordar os fundamentos da programação aplicados ao R.

Estudaremos mais a fundo o uso dos objetos, das funções e dos pacotes, nossas três principais ferramentas dentro do R.

Chapter 16

Funções

Na seção anterior falamos que *print()* é um função, mas não explicamos a fundo o que isso significa.

De maneira geral, as funções no R são blocos de código reutilizáveis que realizam uma tarefa específica. Elas são fundamentais para a programação em R, pois permitem organizar e modularizar o código, tornando-o mais curto, legível, eficiente e fácil de manter.

Pense o seguinte, no exercício anterior atribuímos algumas variáveis valores com tipos de dados específicos. Eram três programas de saúde, o que tornava razoavelmente tranquilo repetir os cálculos para cada um deles. Agora imagine que você trabalha com dados de 200 milhoẽs de pessoas. Repetir a mesma operação, pelo menos, 200 milhões de vezes é inviável. As funções são uma das opções para evitarmos repetir as mesmas tarefas. Há outras como as classes, mas não abordaremos neste curso.

16.0.1 Entendendo melhor os objetos no R

Antes, porém, precisamos entender o que é um objeto. Um objeto é um nome que guarda um valor. Ah, então variáveis são objetos? Podemos dizer que sim. Mas há objetos que não são variáveis, como as contantes.

Existem algumas regras para nomear objetos. São elas:

1. O nome do objeto deve sempre começar com uma letra.
2. O nome pode conter números, mas não pode ser iniciado com um número.
3. O nome de constantes deve ser colocado em letras maiúsculas.
4. Pode-se usar pontos . e underlines _ para separar palavras, mas não podemos iniciar o nome de um objeto com esses caracteres.
5. Não se pode usar hífen (-) para iniciar ou separar o nome de um objeto.

6. Em um mesmo conjunto de scripts use-se sempre a mesma forma para nomear os objetos. Isso quer dizer que se utilizamos . para separar as palavras que nomeiam variáveis, não utilizaremos _.
7. O R diferencia letras maiúsculas de minúsculas, ou seja, 'a' é difente de 'A' em qualquer contexto. A essa característica chamamos de *camel sensitive*.

Veja alguns exemplos:

```
Nomes de objeto bem formados

  objeto <- valor do objeto
  x <- 23
  primeiro_objeto <- 34
  segundo.objeto <- 'Como vai?'
  PI <- 3.14

Nomes de objeto mal formados

  1x -> 23
  2x > 34
  PrimeiroObjeto <- 56
  _objeto <- "Olá, mundo!"
  segundo-objeto <- print("Como vai?")
  pi <- 3.14
```

O operador de atribuição <- é sempre utilizado para valorar os objetos que não são funções.

16.0.2 Funções nativas ou embutidas

No R existem funções que são nativas ou embutidas (built-in), ou seja, já vem programadas para a gente utilizar. É o caso do print() ou c() que você já utilizou bastante neste curso. Agora vamos conhecer outras.

As funções embutidas do R, pois são muitas e variam entre manipulação de dados, operações matemáticas, estatística, gráficos, caracteres e texto, controle de fluxo, data e hora, entrada e saída de dados, funções de sistema.

Em seções anteriores vimos as funções de impressão (print()), de criação de vetores (c()), de data e hora (as.Date()). Caso não se recorde sugerimos que revise o conteúdo visto até aqui.

Segue uma lista com as principais funções embutidas do R:

1. Manipulação de Dados:

- c(…): Cria vetores combinando elementos. A função pode ser chamada dentro de estruturas de dados como listas e dataframes.

```
numeros <- c(1,2,3,4,5,6)
letras <- c('a', 'b', 'c', 'd')
```

- length(x): Retorna o número de elementos de um vetor ou lista.

```
length(numeros)  # Resultado: 6
length(letras)  # Resultado: 4
```

- seq(from, to, by): Cria sequências de números, onde 'from' é o primeiro número a ser gerado, 'to' é o número limite, 'by' são os passos da sequência.

```
seq(1, 10, by = 2)  # Resultado: 1 3 5 7 9
seq(1, 10, by = 3)  # Resultado: 1 4 7 10
seq(1, 10, by = 4)  # Resultado: 1 5 9
```

- rep(x, times): Repete um elemento ou vetor.

```
rep(2, 5)   # Resultado: 2 2 2 2 2
```

- sort(x): Ordena um vetor.

```
sort(numeros)  # Resultado: 1 2 5 8
```

- rev(x): Inverte a ordem dos elementos de um vetor.

```
rev(numeros)  # Resultado: 8 2 5 1
```

- toupper() e tolower(): Convertem uma string para maiúsculas ou minúsculas.

```
toupper("hello") # Resultado: "HELLO"
tolower("HELLO") # Resultado: "hello"
```

2. Operações Matemáticas:

- sum(x) / prod(x): Calcula a soma / produto dos elementos de um vetor.

```
sum(numeros)  # Resultado: 10

prod(numeros)  # Resultado: 24
```

- mean(x) / median(x): Calcula a média / mediana de um vetor.

```
mean(numeros)  # Resultado: 2.5

median(numeros)  # Resultado: 2.5
```

- sd(x) / var(x): Calcula o desvio padrão / variância de um vetor.

```
sd(numeros)  # Resultado: 1.290994

var(numeros)  # Resultado: 1.666667
```

- min(x) / max(x): Encontra o valor mínimo / máximo de um vetor.

```
min(numeros)  # Resultado: 1
max(numeros)  # Resultado: 4
```

- abs(x): Retorna o valor absoluto de um número.

```
abs(-5)  # Resultado: 5
```

- round(x, digits): Arredonda um número para um número especificado de dígitos.

```
round(3.14159, digits = 2)  # Resultado: 3.14
```

3. Estatística:

- summary(x): Fornece estatísticas descritivas de um vetor ou data frame.

```
summary(numeros)

# Resultado: Mínimo, 1º Quartil, Mediana, Média, 3º Quartil, Máximo
```

- table(x): Cria uma tabela de frequências de um vetor.

```
table(c("A", "B", "A", "C"))  # Resultado: A B C
```

- cor(x, y): Calcula a correlação entre dois vetores.

```
cor(c(1, 2, 3), c(4, 5, 6))  # Resultado: 1 (correlação perfeita)
```

- lm(formula, data): Ajusta um modelo de regressão linear.

```
# Exemplo com uma regressão linear simples
data <- data.frame(x = c(1, 2, 3, 4, 5), y = c(2, 4, 6, 8, 10))
model <- lm(y ~ x, data = data)
summary(model)
```

- t.test(x, y): Realiza um teste t de Student.

```
# Exemplo de teste t de Student
set.seed(123)
group1 <- rnorm(20, mean = 10, sd = 2)
group2 <- rnorm(20, mean = 11, sd = 3)
t.test(group1, group2, var.equal = TRUE)
```

4. Entrada e Saída:

- print(x): Imprime um objeto no console.

```
print("Olá, mundo!")
```

- cat(..., sep = " "): Concatena e imprime strings.

```
cat("Olá", "mundo!", sep = "\n")

# Resultado: Olá
# mundo!
```

- read.csv(file): Lê um arquivo CSV.

```
dados <- read.csv("dados.csv")
```

- write.csv(x, file): Escreve um data frame em um arquivo CSV.

```
write.csv(dados, "novos_dados.csv")
```

AS FUNÇÕES DISPONÍVEIS NOS ITENS 5 E 6 SERÃO ABORDADAS SEPARADAMENTE

5. Controle de Fluxo:

- if (condição) { ... } else { ... }: Executa código condicionalmente.
- for (i in vetor) { ... }: Itera sobre os elementos de um vetor.
- while (condição) { ... }: Executa código repetidamente enquanto uma condição for verdadeira.

6. Outras Funções Úteis:

- str(x): Mostra a estrutura de um objeto.
- getwd(): Mostra o diretório de trabalho atual no R
- setwd(): Configura o diretório de trabalho nos qual desejamos trabalhar
- class(x): Retorna a classe de um objeto.
- help(função): Abre a documentação de uma função.
- install.packages("nome_do_pacote"): Instala um pacote.
- library(nome_do_pacote): Carrega um pacote.

Note que para cada função há algumas características ou regras que ela deve seguir para realizar a tarefa. Por exemplo, em *write.csv(x, file)*, x diz ao R qual objeto deve ser usado para escrever o arquivo, enquanto file será o nome do arquivo salvo em nossa pasta de trabalho. Essas regras são conhecidas para parâmetros. Por isso, sempre que for utilizar uma função leia a documentação para saber quais parâmetros serão necessário para seu correto funcionamento. No R temos uma função específica para isso, a *help()*.

Pedindo ajuda no R

Sempre que precisar saber o que uma função faz, basta usar a função help que ela retornará a documentação no console ou no ambiente gráfico do RStudio. Abaixo segue um exemplo:

```
help(print)
```

Ao rodar este comando no RStudio você verá na tela o objetivo da função, quais parâmetros recebe e exemplos de uso. Sempre que precisar de ajuda utilize a help().

Obs: A melhor forma de compreender as funções embutidas no R é treinando em cenários específicos. Ao final dessa sessão você encontrará alguns exercícios. Leia o problema proposto e procure responder antes de buscar os scripts de gabarito disponíveis no repositório do curso.

16.0.3 Funções construídas ou próprias

Muitas vezes precisamos realizar tarefas específicas, as quais não estão disponíveis de forma nativa. O R nos permite construir nossas próprias funções. Para isso precisamos conhecer a estrutura de função e como acioná-la quando necessário em nossos scripts.

Apesar de não estarem no escopo do nosso curso, pois são um tópico mais avançado no R, colocamos aqui uma breve explicação sobre a estrutura das funções. Caso não tenha interesse, você poderá pular este tópico.

Algumas características importantes das funções no R:

1. **Definição**: As funções são definidas usando a palavra-chave `function`. A sintaxe básica é:

```r
nome_da_funcao <- function(argumentos) {
  # Corpo da função
  return(resultado)
}
```

2. **Argumentos**: As funções podem receber zero ou mais argumentos, que são valores passados para a função quando ela é chamada. Esses argumentos são usados dentro do corpo da função.

3. **Corpo**: O corpo da função contém o código que será executado quando a função for chamada. Esse código pode incluir instruções, cálculos, chamadas a outras funções, etc.

4. **Retorno**: A função pode retornar um ou mais valores usando a palavra-chave `return()`. Se nenhum valor for especificado, a função retornará `NULL`.

5. **Reutilização**: Após definir uma função, você pode chamá-la quantas vezes quiser em seu código, passando os argumentos necessários. Isso evita a repetição de código e torna o programa mais modular e fácil de manter.

6. **Escopo**: As funções têm seu próprio escopo, o que significa que as variáveis definidas dentro da função são acessíveis apenas dentro dela. Isso ajuda a evitar conflitos de nomes de variáveis.

Aqui está um exemplo simples de uma função em R:

```r
# Definição da função
calcular_area_retangulo <- function(largura, altura) {
  area <- largura * altura
  return(area)
}

# Chamada da função
resultado <- calcular_area_retangulo(5, 10)
print(resultado)  # Saída: 50
```

Nesse exemplo, a função `calcular_area_retangulo` recebe dois argumentos (largura e altura) e retorna a área do retângulo. Ao chamar a função, passamos os valores 5 e 10 como argumentos, e o resultado (50) é armazenado na variável `resultado`.

As funções são uma parte essencial da programação em R, pois permitem organizar e modularizar o código, tornando-o mais legível, reutilizável e fácil de manter.

Chapter 17

Pacotes

Um pacote é um conjunto de funções que podem ser instaladas no R com o objetivo de estender suas capacidades.

Lembra que falamos que toda a base do desenvolvimento da linguagem R está na comunidade. Se as funções são uma mão-na-roda, facilitando nosso trabalho e permitindo que consigamos descrever uma base de dados inteira com 5 ou 6 linhas de código, os pacotes existem para acrescentar funções que o R não traz embutidas.

Exitem pacotes para resolver praticamente todo tipo de problema estatístico. É difícil listar pacotes úteis sem saber exatamente para qual área estamos trabalhando. Porém, há pacotes que são abrangentes, como o conjunto de pacotes conhecido como Tidyverse. Com ele já é possível fazer toda a pipeline de ciência de dados sem necessitarmos de ferramentas adicionais. Mas se você precisa usar dados do Datasus poderá usar o pacote *microdatasus* e realizar as tarefas de extração (aquisição de dados) por meio dele.

Para instalar um pacote basta utilizar os seguintes comandos:

```
install.package("[nome_do_pacote]")
```

O R se encarregará de fazer a instalação para você. Depois de instalado é só incluir no seu script a seguinte função:

```
library("[nome_do_pacote]")
```

Sempre que precisar de um pacote específico para realizar seu trabalho, lembre-se de verificar se ele está instalado antes de tentar fazer uma nova instalação ou tentar carregar suas funções. Para isso uso o seguinte comando:

```
installed.packages()
```

Esse comando retornará a lista de pacotes instalados no seu sistema.

Caso você necessite atualizar os pacotes já instalados, basta utilizar o comando update.packages(checkBuilt=TRUE, ask=FALSE).

OBS: Se você estiver usando o RStudio poderá utilizar o gerenciador disponível no canto inferior direito da tela. Nele é possível instalar e carregar pacotes utilizando uma interface gráfica.

17.0.1 EXERCÍCIOS

1.

Objetivo:

Este exercício visa familiarizar os alunos com as funções embutidas do R para manipulação e análise de dados numéricos, além de introduzir conceitos básicos de estatística descritiva. Ao praticar com dados reais, os alunos poderão aplicar seus conhecimentos de forma prática e relevante.

Cenário:

Você é um professor que precisa analisar as notas de seus alunos em uma prova. As notas estão armazenadas em um vetor chamado notas. Utilize as funções embutidas do R para realizar as seguintes tarefas:

```
Organização:
    Ordene as notas em ordem crescente.
    Inverta a ordem das notas, mostrando-as da maior para a menor.

Estatísticas Descritivas:
    Calcule a média das notas.
    Encontre a mediana das notas.
    Determine a nota mais alta e a nota mais baixa.
    Calcule o desvio padrão das notas.

Frequência:
    Crie uma tabela de frequências para mostrar quantas vezes cada nota aparece.
```

Dados: Fragmento do código

notas <- c(8.5, 7.0, 9.2, 6.8, 8.0, 7.5, 9.0, 5.5, 8.8, 7.2)

Use o código com cuidado.

Dicas:

```
Use as funções sort(), rev(), mean(), median(), min(), max(), sd() e table().
Explore a função summary() para obter um resumo estatístico completo das notas.
```

Desafio Extra:

```
Crie um novo vetor chamado notas_arredondadas que contenha as notas arredondadas para uma casa de
Calcule a média das notas arredondadas e compare com a média das notas originais.
```

2.

Neste exercício, você irá utilizar algumas das funções embutidas do R para realizar uma análise básica de um conjunto de dados. Dados

Vamos utilizar o conjunto de dados mtcars, que está disponível no R por padrão. Esse conjunto de dados contém informações sobre 32 modelos de carros, incluindo características como consumo de combustível, número de cilindros, potência do motor, entre outras. Tarefas

```
Carregar o conjunto de dados:
    Utilize a função data() para carregar o conjunto de dados mtcars em seu ambiente de trabalho.

Explorar a estrutura dos dados:
    Utilize a função str() para exibir a estrutura do data frame mtcars.
    Observe quantas linhas (observações) e colunas (variáveis) o data frame possui.

Obter informações básicas sobre as variáveis:
    Utilize a função summary() para obter um resumo estatístico das variáveis no data frame.
    Identifique quais são as variáveis numéricas e quais são as variáveis categóricas.

Calcular estatísticas descritivas:
    Selecione uma variável numérica, por exemplo, mpg (milhas por galão).
    Utilize as funções mean(), median(), sd(), min() e max() para calcular a média, mediana, desv

Ordenar os dados:
    Selecione uma variável numérica, por exemplo, hp (potência do motor).
    Utilize a função sort() para ordenar os valores dessa variável em ordem crescente.
    Utilize a função rev() para inverter a ordem dos valores.

Criar uma sequência de números:
    Utilize a função seq() para criar uma sequência de números de 1 a 10, com incrementos de 2.
    Crie outra sequência de números de 5 a 15, com incrementos de 3.

Repetir elementos:
    Utilize a função rep() para criar um vetor com o número 7 repetido 4 vezes.
    Crie outro vetor com a letra 'a' repetida 10 vezes.
```

```
Converter entre maiúsculas e minúsculas:
    Selecione uma variável de texto, por exemplo, cyl (número de cilindros).
    Utilize as funções toupper() e tolower() para converter os valores dessa variável p
```

Após concluir todas as tarefas, você terá praticado o uso de diversas funções embutidas do R, como data(), str(), summary(), mean(), median(), sd(), min(), max(), sort(), rev(), seq(), rep(), toupper() e tolower(). Essas funções são fundamentais para a manipulação e análise básica de dados em R.

###EXEMPLO DE SCRIPT PARA O PROJETO FINAL DO CURSO

Agora que você já exercitou o uso de funções e pacotes no R por meio de exemplos genéricos vamos abordar um pouco o uso dessas mesmas ferramentas para os dados de clima e saúde no Brasil. Trata-se de um script inicial que será aprimorado e desenvolvido ao longo do curso. Por isso, apenas copie e cole o código em seu RStudio ou Google Colab e ir acompanhando as explicações abaixo conforme roda o sript.

Todo início de script R, costuma seguir a sequência de instalação e chamada da função dos pacotes que serão utilizados.

Programar fica divertido quando a gente experimenta. Fique à vontade para copiar e fazer as alterações e testes que desejar. Procure por soluções similares nos repositórios, sites e livros indicados nas referências do curso.

Lembre-se, nas primeiras vezes que rodamos um script precisamos fazer a tarefa linha a linha. Especialmente, as instalações e "chamadas" dos pacotes com os quais vamos trabalhar.

Os dados estão disponíveis neste link -> https://github.com/bmclima/cursoRBasico/blob/main/docs/dados/

```r
#Instala e configura o remotes
install.packages('remotes')
library(remotes)

#Instala os pacotes de leitura de arquivo e ciência de dados
remotes::install_github("rfsaldanha/microdatasus")
install.packages('tidyverse')
install.packages('readr')

#Chama as bibliotecas
library(microdatasus)
library(tidyverse)
library(readr)

#Configura as pastas de trabalho
```

```r
getwd()
setwd('[caminho da pasta de trabalho onde desejamos trabalhar]')

#Baixa os arquivos com os dados
#dados das estações
url <- "https://github.com/bmclima/cursoRBasico/blob/main/docs/dados/stations.csv"
estacoes <- read_csv(url)
#dados do SIM
sim <- fetch_datasus(year_start = 2010,
year_end = 2010,
uf = "DF",
information_system = "SIM-DO")

#Função do pacote microdatasus para pré-processar os dados coletados
sim_df <- process_sim(sim)

#Lê o cabeçalho dos bancos de dados
head(estacoes)
head(sim)

#Sumariza o conjunto de dados
summary(estacoes)
summary(sim_df)

#Verifica a estrutura dos dados
str(estacoes)
str(sim_df)
```

17.0.1.1 Explicando o código

Comentários

Comentar o código que estamos escrevendo é uma boa prática, pois permite que lembremos o que estávamos querendo fazer. Acredite daqui a um mês, provavelmente você não vai lembrar o que estava pensando na hora que escreveu seus scripts. Por isso, comente. Seja breve e tente deixar o mais claro possível o que aquele trecho específico faz.

Você pode fazer comentários no R, usando o caracter cerquilha (#). Basta colocar uma # antes de cada linha que será usada como comentário. Há outras formas de fazer comentários longos que não serão abordadas aqui, mas que estão disponíveis nos materiais do curso no GitHub.

Então cada linha iniciada com uma # é apenas para indicar o que aquele trecho de código faz.

Instalação do pacote 'remotes'

```
install.packages('remotes')
library(remotes)
```

Esse pacote permite a instalação de pacotes que não estão incluídos no repositório oficial do R. O pacote microdatasus é mantido via GitHub, então para instalá-lo precisamos do remotes. Chamamos ele antes de instalar os outros pacotes porque caso façamos após a tentativa de instalação do microdatasus o script não funcionaria.

Instalação dos outros pacotes

```
remotes::install_github("rfsaldanha/microdatasus")
install.packages('tidyverse')
install.packages('readr')
```

Note qua a primeira linha de código chama o remotes e a função install_github para realizar a intalação do microdatasus. Os outros dois pacotes estão no repositório oficial do R então apenas utilizamos o install.packages().

Chamada dos pacotes para usar no script

```
library(microdatasus)
library(tidyverse)
library(readr)
```

Usando a função library() indicamos ao R quais ferramentas não embutidas vammos utilizar em nosso script. Neste caso indicamos o microdatasus, o tidyverse e o readr. Os dois primeiros já explicamos, o terceiro realiza a leitura de datasets em diversos formatos, tais como csv, xlsx, txt.

Configuração da pasta de trabalho

```
getwd()
setwd('[caminho da pasta de trabalho onde desejamos trabalhar]')
```

Quando falamos sobre instalação e configuração do ambiente de trabalho, ressaltamos o quanto organizar nossos arquivos em pastas de projeto é importante. Nessa parte do script verificamos em que pasta estamos trabalhando e apontados (setamos) o nosso script para onde estarão nossos arquivos.

Baixar os dados

```
url <- "https://github.com/bmclima/cursoRBasico/blob/main/docs/dados/stations.csv"
estacoes <- read_csv(url)
#dados do SIM
```

```
sim <- fetch_datasus(year_start = 2010,
year_end = 2010,
uf = "DF",
information_system = "SIM-DO")
```

Aqui informamos um link onde o R pode baixar os dados que vamos usar diretamente e depois usamos usamos a variável 'estacoes' para armazenar o arquivo. Uma outra opção é ir diretamente até o link, baixar o arquivo manualmente e importar o dataset como dataframe no RStudio.

Em seguida baixamos os dados do SIM utilizando a função 'fetch_datasus' do pacote 'microdatasus'. Ela possui quatro parâmetros básicos, ano de início dos dados, ano final, uf e o sistema de informação que queremos acessar. Neste caso indicamos que queremos baixar os dados do SIM relativos ao ano de 2010 para o DF. Para maiores detalhes sobre o função você pode consultar a documentação do pacote no seguinte link https://github.com/rfsaldanha/microdatasus.

Pré-processando os dados do SIM

```
sim_df <- process_sim(sim)
```

Uma vez que baixamos os dados do SIM e guardamos na variável 'sim', precisamos realizar o pré-processamento para que possamos trabalhar com eles. O pacote microdatasus possui uma outra função chamada process_sim() que já entrega os dados tabulados e limpos para serem usados.

Note que atribuímos o processo de baixar os dados a uma variável e depois atribuímos o pré-processamento a outra. Isso é uma boa prática, pois evita que utilizemos um objeto para mais de uma tarefa dentro do script. Essa boa prática é chamada de especialização. Cada objeto serve para um único propósito dentro da nossa área de trabalho.

Verificando se as bases estão no formato e com as variáveis que queremos

```
head(estacoes)
head(sim_df)
```

Utilizamos a função head() para verificar o cabeçalho dos datasets. Por padrão os cinco primeiros registros dos dataframes carregados no ambiente serão mostrados. É momento de você fazer uma verificação e saber se tudo está ok. Se os dados realmente estão lá, se o nome das colunas corresponde com o dicionário de dados, etc. Chamamos esse trabalho de inspeção visual.

Verificando as dimensões das bases de dados

```
summary(estacoes)
summary(sim_df)
```

Após realizar a inspeção visual, vamos resumir os dados do dataframe para saber as dimensões dele. A função summary faz uma descrição estatística básica do seu dataframe.

Verificando a estrutura dos dados

```
str(estacoes)
str(sim_df)
```

Com a função str() verificamos quais os tipos dos dados presentes no dataframe. Se são caracteres, numéricos, datas, etc. Isso é importante para sabermos se precisaremos realizar algum ajuste nos dados antes de rodarmos nossos modelos.

No caso de nossas dados, veremos que os dados presentes no dataframe 'estacoes' estão todos normalizados, mas que no 'sim_df' todos os dados foram armazenados como caracteres. Logo, precisaremos realizar ajustes nos dados presentes em 'sim_df'.

Chapter 18

Materiais e referências

- Repositório oficial de pacotes do R
- Documentação do R
- Site oficial do Tidyverse
- Link para os exercícios e scripts de exemplo do curso
- Artigo sobre o pacote microdatasus

Chapter 19

Fundamentos de programação

UFA! Se você chegou até aqui já consegue realizar as atividades de uso básico do R. Agora vamos conhecer algumas ferramentas mais avançadas e conhecer pipeline de análise de dados.

Neste capítulo do curso, vamos continuar a abordar os fundamentos da programação aplicados ao R. Primeiro falaremos do operador pipe (%>%). Estudaremos mais a fundo o uso das funções de entrada e saída de dados e da manipulação de strings.

Em seguida falaremos como programar o script para tomar decisões condicionais (if..else) e repetir (loops) até tarefas até alcançar um determinado objetivo.

Chapter 20

Funções e operadores no R - parte 2

20.1 O operador pipe

Em alguns momentos de nosso curso usamos o termo "pipeline de dados" para nos referir ao conjunto de processos que nos permitem tratar, analisar e visualizar um conjunto de dados.

O operador pipe nos permite colocar nossas funções em linha para serem aplicadas em um mesmo objeto. É representado pelo símbolo %>%. Com ele evitamos escrever uma linha de código separada para cada tarefa que desejamos que o script execute.

Observe o código abaixo:

```
dado_original %>%
  funcao1() %>%
  funcao2() %>%
  funcao3()
```

Note que primeiro declaramos o objeto, depois aplicamos as funções uma seguida da outra. Você deve lembrar que o R aplicará cada função na ordem em que está escrita. Portanto, se você precisa baixar e processar um conjunto de dados, deve primeiro colocar a função de baixar e depois a de processar.

Lembra do nosso script para tratar dados do Datasus. Pode reescrevê=lo da seguinte forma:

```
sim_df <- fetch_datasus(year_start = 2010,
                        year_end = 2010,
                        uf = "DF",
                        information_system = "SIM-DO") %>%
          process_sim()
```

Aqui ao invés declararmos dois objetos, declarmos apenas um e realizamos todas as tarefas por meio do pipe. As vantagens são muitas, mas as principais são o tempo de processamento, quantidade de linhas código, clareza e a diminuição do uso de memória durante o trabalho com os dados.

OBS:

- Em alguns sistemas pode ser que o operador pipe não esteja instalado ou funcionando corretamente. Caso isso ocorra com você use os seguintes comandos:

```
install.packages("magrittr")
library(magrittr)
```

- Há outros operadores com o mesmo objetivo do pipe, permitir que escrevamos códigos mais suscintos. Não vamos abordá-los aqui, mas se você quiser estudá-los, recomendamos utilizar o livro do Grupo Adar referênciado ao final desta seção.

##Entradas e saídas (input/output)

A entrada e saída de dados é uma parte fundamental da programação em R, por isso vamos reforçar as funções disponíveis para realizar essas tarefas. Isso é importante pois dependendo dos objetos que estamos querendo alcançar, da fonte dos dados ou do ambiente de análise precisaremos de funções suplementares para baixar, ler, manipular e escrever arquivos. O R possui um conjunto de funções embutidas que nos permitem realizar essas tarefas e mais um conjunto extenso de pacotes auxiliares.

Abaixo separamos essas funções em quatro categorias:

1. **Entrada de dados**:
    - **Teclado**: Usar a função `readline()` para solicitar a entrada de dados do usuário.
        - Exemplo: `nome <- readline(prompt = "Digite seu nome: ")`
    - **Arquivos**: Usar a função `read.table()` ou `read.csv()` para ler dados de arquivos.

  - Exemplo: `dados <- read.csv("dados.csv")`

- **Banco de dados**: Usar pacotes como `DBI` e `RMySQL` para se conectar e ler dados de bancos de dados.
  - Exemplo: `library(DBI) con <- dbConnect(MySQL(), user="usuario", password="senha", dbname="banco_de_dados") dados <- dbGetQuery(con, "SELECT * FROM tabela")`

Destacamos as funções readline() e os pacotes para leitura de dados diretamente de um banco de dados.

- Em alguns projetos vamos fazer interfaces que coletam dados diretamente dos usuários de nossos sistemas e análise. A readline() é a funções que nos permite fazer perguntas aos usuários. Por exemplo, você pode precisar que seu usuário escolha um determinado período de tempo ou UF para gerar uma visualização de dados.
- Grande parte dos dados disponíveis para uso está armazenado em bancos de dados com pacotes como DBI ou RMySQL é possível baixar os dados diretamente dessas fontes sem precisar que eles sejam exportados para formatos como csv ou txt. Neste curso não abordaremos esse tipo de operação em detalhes mas vale a pena estudar esse tópico por conta própria.
- A função read.csv() pode ser substituída pelas funções do pacote {readr}. As vantagens é que a sintaxe é exatamente a mesma, mas é possível passar parâmetros mais detalhados na hora de importar os dados, evitando realizar processamentos desnecessários. Os parâmetros disponíveis para o read.csv() são:
  - col_names=: indica se a primeira linha da base contém ou não o nome das colunas. Também pode ser utilizado para (re)nomear colunas.
  - col_types=: caso alguma coluna seja importada com a classe errada (uma coluna de números foi importada como texto, por exemplo), você pode usar esse argumento para especificar a classe das colunas.
  - locale=: útil para tratar problema de encoding.
  - skip=: pula linhas no começo do arquivo antes de iniciar a importação. Útil para quando o arquivo a ser importado vem com metadados ou qualquer tipo de texto nas primeiras linhas, antes da base.
  - na=: indica quais strings deverão ser considaras NA na hora da importação.
- O pacote {readr} também fornece outras funções de leitura de arquivo muito úteis a qualquer analista de dados. Recomendamos utilizar a função help do R e explorar a documentação da ferramenta.

2. **Saída de dados**:

- **Console**: Usar a função `print()` ou `cat()` para exibir dados no console.
  - Exemplo: `print(nome)` ou `cat("Olá,", nome, "!")`
- **Arquivos**: Usar a função `write.table()` ou `write.csv()` para salvar dados em arquivos.
  - Exemplo: `write.csv(dados, "dados.csv", row.names = FALSE)`
- **Gráficos**: Usar funções de plotagem, como `plot()` e `ggplot()`, para gerar gráficos.
  - Exemplo: `plot(x, y)`

Neste tópico não há destaques específicos. Os três tipos de saída apresentados são fundamentais.

- O primeiro nos permite ser visualizar o resultado dos nossos scripts enquanto estamos trabalhando e fazer impressões simples em tela.
- As funções write.table() e write.csv() permitem que salvemos os dados transformados no HD do computador ou em nuvem. Lembre-se sempre que cada objeto carregado está apenas na memória RAM do computador. Quando a sessão finalizar seu trabalho poderá ser perdido se não for salvo em um arquivo permanente. Isso também vale para seus dados.
- As visualizações gráficas são de longe a parte mais esperada ao longo da pipeline. As funções plot() e ggplot() nos permitem plotar gráficos dos mais simples aos mais complexos.

3. **Manipulação de diretórios**:

- **Configuração**: `setwd()` para definir o diretório de trabalho.
- Exemplo: `setwd('C:\dir_projeto')`. O R entenderá que naquela sessão o diretório dir_projeto é o diretório de trabalho.
- **Listagem de arquivos**: `list.files()` para listar os arquivos em um diretório.
  - Exemplo: `list.files('C:\dir_projeto')`. Todos os arquivos disponíveis em dir_projeto serão listados em tela.
- **Manipulação de arquivos:** `file.create()`, `file.remove()` e `file.rename()` para criar, remover e renomear arquivos.
  - Exemplo: `file.create('meus_scripts.R')`. Criará um arquivo vazio no diretório de trabalho atual.

Já vimos o setwd() na seção anterior, inclusive sugerimos que o script final do curso conte com ele como função. Como ele podemos dizer ao R onde queremos que nosso trabalho seja executado e onde, por padrão, os arquivos e dados serão salvos. Porém, é possível usar o R para realizar qualquer tarefa relacionada à listagem, criação, renomeação e remoção de arquivos do HD.

4. **Pacotes adicionais**:

    - Existem diversos pacotes no R que fornecem funcionalidades adicionais para entrada e saída de dados, como `readxl` (leitura de arquivos Excel), `jsonlite` (leitura e escrita de arquivos JSON) e `httr` (interação com APIs).

    Algumas fontes de dados estão em formatos especiais que são específicos para que outros sistemas e computadores interajam com elas. Quando precisamos baixar e pré-processar esses dados necessitamos instalar pacotes que façam a leitura do formato desejado. Os dois exemplos mais comuns são o formato JSON e a interação com APIs. Outro uso desses formatos são as famosas raspagens de dados em sites e painéis disponíveis na web. Como se trata de uma tópico avançado, não abordaremos neste curso.

Abaixo segue um exemplo de script que utiliza algumas das funções de entrada e saída de dados.

```r
# Entrada de dados
# Solicitar o nome do usuário
nome <- readline(prompt = "Digite seu nome: ")
cat("Olá,", nome, "!\n")

# Ler dados de um arquivo CSV
dados <- read.csv("temperatura_df.csv")

# Manipulação de diretórios
# Definir o diretório de trabalho
setwd("C:/dir_projeto")

# Listar os arquivos no diretório
arquivos <- list.files()
print(arquivos)

# Criar um novo arquivo
file.create("meus_scripts.R")

# Saída de dados
# Exibir os primeiros 6 registros dos dados
print(head(dados))

# Salvar os dados em um novo arquivo CSV
write.csv(dados, "temperatura_df_processado.csv", row.names = FALSE)

# Gerar um gráfico simples
plot(dados$temperatura, main = "Temperatura no DF")
```

20.2 Manipulação de strings

A manipulação de strings (caracteres) é uma tarefa comum em análise de dados, saber realizá é fundamental a construção de modelos e painéis. Abaixo seguem as técnicas de manipulação mais comuns em R.

1. **Concatenação de strings**:

   - Usar o operador `+` ou a função `paste()` para concatenar strings.
   - Exemplo: `nome <- "João" + " " + "Silva"`

   O operador + em R pode ser usado para concatenar strings. Neste exemplo, "João" + " " + "Silva" irá concatenar as três strings, resultando em "João Silva". É importante notar que o operador + só pode ser usado para concatenar strings se todas as partes envolvidas forem strings. Caso contrário, o R tentará realizar uma operação matemática.

2. **Extração de substrings**:

   - Usar a função `substr()` para extrair uma substring de uma string.
   - Exemplo: `sobrenome <- substr(nome, 6, 11)`

   Essa função é útil quando você precisa extrair uma parte específica de uma string, como o sobrenome de uma pessoa a partir do nome completo. No exemplo acima supondo que a variável nome contenha a string "João Silva". A função substr() é aplicada a essa string, extraindo os caracteres da posição 6 até a posição 11. O resultado dessa extração é armazenado na variável sobrenome. Portanto, o valor de sobrenome será "Silva".

3. **Substituição de substrings**:

   - Usar a função `gsub()` para substituir padrões em uma string.
   - Exemplo: `nome_corrigido <- gsub("João", "João", nome)`

   A função gsub() é muito poderosa e permite o uso de expressões regulares para definir padrões mais complexos de substituição. As expressões regulares são padrões de conjunto de caracteres que indicamos ao computador de forma que o script possa buscar e encontrar esse padrão ao longo de um objeto. Essa função nos permite realizar essa tarefa sem a necessidade de programar as expressões regulares manualmente. No exemplo acima se nome tiver a atribuição de "João da Silva", A função gsub() é aplicada a essa string, substituindo todas as ocorrências do padrão "João" pela string "João". O resultado dessa substituição é armazenado na variável nome_corrigido. Portanto, o valor de nome_corrigido será "João Silva".

4. **Conversão de maiúsculas e minúsculas**:

- Usar as funções `toupper()` e `tolower()` para converter strings em maiúsculas ou minúsculas.
- Exemplo: `nome_maiusculo <- toupper(nome)`

Anteriormente abordamos o uso dessas funções, caso tenha dúvidas, pedimos que retorne para o capítulo 2 do curso e revise o que foi estudado.

5. **Remoção de espaços em branco**:

   - Usar a função `trim()` do pacote `stringr` para remover espaços em branco no início e no final da string.
   - Exemplo: `nome_sem_espacos <- stringr::trim(nome)`

   A remoção de espaços em branco é uma das tarefas mais comuns no préprocessamento de uma base de dados. A função trim() remove esses espaços a partir de um objeto atribuído. No exemplo qualquer espaço antes ou depois dos caracteres seria retirado e o novo valor atribuído à variável nome_sem_espacos. Abaixo colocamos um exemplo mais completo:

```
nome <- "   João Silva   "
nome_sem_espacos <- stringr::trim(nome)
print(nome_sem_espacos)  # Saída: "João Silva"
```

6. **Divisão de strings**:

   - Usar a função `strsplit()` para dividir uma string em um vetor de substrings.
   - Exemplo: `partes_nome <- strsplit(nome, " ")`

   Essa função é útil quando você precisa separar uma string em suas partes constituintes, como palavras em uma frase ou campos em uma string delimitada. Costuma ser muito útil para realizar a extração, a limpeza e a manutenção de bases de dados. Além de ser vastamente utilizada na contrução de datasets para treinamento e validação de modelos de aprendizagem de máquina.

7. **Verificação de padrões**:

   - Usar a função `grepl()` para verificar se uma string contém um determinado padrão.
   - Exemplo: `tem_silva <- grepl("Silva", nome)`

   Assim como na divisão de strings, utilizamos a verificação de padrões para inspecionar bases de dados, processar bases e validar modelos. Essa função aceita dois parâmetros, primeiro colocarmos o padrão que queremos verificar e depois o objeto a que o conjunto de dados está atribuído. A função passará pelos dados e retornará quantas vezes e onde o padrão está presente.

8. **Formatação de strings**:

    - Usar a função `sprintf()` para formatar strings com valores numéricos.
    - Exemplo: `mensagem <- sprintf("O paciente %s tem %d anos.", nome, idade)`

    A formatação de strings é muito útil para criar saídas para relatórios utilizando os registros presentes no dataframe como preenchimento de campos dinâmicos. No exemplo acima %s e %d são preenchidos com os registros de nome e idade dos pacientes registrados no dataset.

A biblioteca `stringr` também fornece uma interface mais intuitiva e consistente para muitas das tarefas listadas acima. Encorajamos o estudo desta biblioteca como tópico de aprofundamento no uso do R.

Chapter 21

Estruturas de Decisão

Quando estudamos o operador pipe, apresentamos uma característica dos programas de computador, eles sempre seguem uma sequência para realizar as atividades. As estruturas de decisão nos permitem realizar tarefas de acordo com um fluxo. No R temos cinco estruturas de decisão mais utilizadas:

{Se... então..} 1. **if-else**: - Sintaxe: `if (condição) { código } else { código }` - Permite executar um bloco de código se a condição for verdadeira e outro bloco se a condição for falsa.

2. **if-else if-else**:
    - Sintaxe: `if (condição1) { código } else if (condição2) { código } else { código }`
    - Permite testar múltiplas condições e executar o bloco de código correspondente.

{Se isso... então escolha esse..}

3. **switch**:
    - Sintaxe: `switch(expressão, caso1 = código, caso2 = código, ..., caso_else = código)`
    - Permite selecionar um bloco de código com base em uma expressão.
4. **ifelse**:
    - Sintaxe: `ifelse(condição, valor_se_verdadeiro, valor_se_falso)`
    - Permite criar um vetor com base em uma condição, atribuindo um valor se a condição for verdadeira e outro valor se a condição for falsa.

{Quando a condição for essa então escolha esse valor...} 5. **case_when**: - Sintaxe: `case_when(condição1 ~ valor1, condição2 ~ valor2, ..., TRUE ~ valor_else)` - Permite criar um vetor com base em múltiplas condições, similar ao `if-else if-else`.

Aqui estão alguns exemplos de uso dessas estruturas de decisão:

```r
# if-else
x <- 10
if (x > 0) {
  print("x é positivo")
} else {
  print("x é negativo")
}

# if-else if-else
score <- 85
if (score >= 90) {
  print("Excelente")
} else if (score >= 80) {
  print("Bom")
} else {
  print("Precisa melhorar")
}

# switch
day <- 3
day_name <- switch(day,
                   "Segunda-feira",
                   "Terça-feira",
                   "Quarta-feira",
                   "Quinta-feira",
                   "Sexta-feira",
                   "Sábado",
                   "Domingo")
print(day_name)

# ifelse
temperatures <- c(20, 25, 18, 30)
weather_status <- ifelse(temperatures > 25, "Quente", "Frio")
print(weather_status)

# case_when
grades <- c(85, 92, 75, 88)
letter_grades <- case_when(
  grades >= 90 ~ "A",
```

```
  grades >= 80 ~ "B",
  grades >= 70 ~ "C",
  TRUE ~ "D"
)
print(letter_grades)
```

Chapter 22

Estruturas de Controle (Loop)

Loops em R são estruturas de controle que permitem a execução repetida de um bloco de código. Existem diferentes tipos de loops disponíveis em R, cada um com suas próprias características e usos específicos. Vamos explorar as principais opções:

1. **for loop**:
   - Sintaxe: `for (variável in vetor) { código }`
   - Executa o bloco de código uma vez para cada elemento do vetor atribuído à variável.
   - Útil quando você sabe antecipadamente o número de iterações necessárias.
   - Exemplo:

```r
for (i in 1:5) {
  print(i)
}
```

2. **while loop**:
   - Sintaxe: `while (condição) { código }`
   - Executa o bloco de código repetidamente enquanto a condição for verdadeira.
   - Útil quando você não sabe antecipadamente o número de iterações necessárias.
   - Exemplo:

```
i <- 1
while (i <= 5) {
  print(i)
  i <- i + 1
}
```

3. **repeat loop**:
   - Sintaxe: `repeat { código }`
   - Executa o bloco de código repetidamente até que uma instrução `break` seja encontrada.
   - Útil quando você não sabe antecipadamente o número de iterações necessárias, mas tem uma condição de parada clara.
   - Exemplo:

```
i <- 1
repeat {
  print(i)
  i <- i + 1
  if (i > 5) {
    break
  }
}
```

4. **lapply() e sapply()**:
   - Funções de ordem superior que aplicam uma função a cada elemento de um vetor ou lista.
   - `lapply()` retorna uma lista, enquanto `sapply()` tenta retornar um vetor.
   - Úteis para aplicar a mesma operação a múltiplos elementos de forma concisa.
   - Exemplo:

```
numbers <- c(1, 2, 3, 4, 5)
squared <- sapply(numbers, function(x) x^2)
```

5. **apply(), tapply() e mapply()**:
   - Funções de ordem superior que aplicam uma função a arrays, matrizes ou listas multidimensionais.
   - `apply()` aplica uma função a linhas ou colunas de uma matriz.
   - `tapply()` aplica uma função a subconjuntos de um vetor.
   - `mapply()` aplica uma função a elementos correspondentes de múltiplos vetores.

- Úteis para operações em estruturas de dados multidimensionais.
- Exemplo:

```
matrix <- matrix(1:6, nrow = 2)
row_sums <- apply(matrix, 1, sum)
```

Abaixo colocamos um script que ilustra o uso das estruturas de controle para baixar e préprocessar dados do SUS.

```
# Criar um dataframe para armazenar as taxas de mortalidade
taxas_mortalidade <- data.frame(
  faixa_etaria = c("0-19 anos", "20-39 anos", "40-59 anos", "60 anos ou mais"),
  obitos = c(0, 0, 0, 0),
  populacao = c(1000000, 1000000, 1000000, 1000000),
  taxa_mortalidade = c(0, 0, 0, 0)
)

# Calcular a taxa de mortalidade por faixa etária
for (i in 1:nrow(df_sim)) {
  idade <- df_sim$IDADE[i]

  if (idade >= 0 && idade <= 19) {
    taxas_mortalidade$obitos[1] <- taxas_mortalidade$obitos[1] + 1
  } else if (idade >= 20 && idade <= 39) {
    taxas_mortalidade$obitos[2] <- taxas_mortalidade$obitos[2] + 1
  } else if (idade >= 40 && idade <= 59) {
    taxas_mortalidade$obitos[3] <- taxas_mortalidade$obitos[3] + 1
  } else {
    taxas_mortalidade$obitos[4] <- taxas_mortalidade$obitos[4] + 1
  }
}

# Calcular a taxa de mortalidade por 100.000 habitantes
taxas_mortalidade$taxa_mortalidade <- (taxas_mortalidade$obitos / taxas_mortalidade$populacao) *

# Exibir os resultados
print(taxas_mortalidade)
```

- Criamos um dataframe taxas_mortalidade para armazenar as informações sobre a taxa de mortalidade por faixa etária.
- Utilizamos um loop for para percorrer cada linha do dataframe df_sim e classificar os óbitos em diferentes faixas etárias (0-19 anos, 20-39 anos, 40-59 anos, 60 anos ou mais) usando uma estrutura de decisão if-else.
- Após contabilizar os óbitos por faixa etária, calculamos a taxa de mortalidade por 100.000 habitantes, assumindo uma população de 1 milhão de pessoas em cada faixa.

- Finalmente, exibimos os resultados armazenados no dataframe taxas_mortalidade.

OBS:

Não incluímos no exemplo as atividades de baixar e préprocessar os dados, pois isso já foi abordado durante o curso.

22.0.1 EXERCÍCIOS

1.

Objetivo:

Este exercício visa lhe familiarizar com o uso do operador pipe.

Cenário:

Suponha que você tenha um conjunto de dados que contém informações sobre a incidência de doenças respiratórias em uma determinada região, bem como dados meteorológicos como temperatura, umidade e precipitação. Você deve usar o operador pipe (%>%) na construção do seu script.

Exercício:

```
Carregue os pacotes tidyverse e microdatasus.
Importe o conjunto de dados que contém as informações de nascimento.
Utilize o operador pipe para:
  a. Filtrar os dados do SINASC para uma determinada UF de um ano a sua escolha.
  b. Calcular a média mensal de nascimentos nesta UF.
```

Regras:

```
- Você deve declarar uma única variável para realizar o exercício.
```

2.

Neste exercício, você irá utilizar algumas das funções de manipulação de strings e arquivos do R.

Cenário:

Você trabalha em uma prefeitura e recebeu um arquivo CSV de uma UBS (pacientes.csv) contendo os nomes completos dos pacientes atendidos em um determinado dia. Sua tarefa é extrair os primeiros e últimos nomes de cada paciente e salvá-los em um novo arquivo CSV.

Dados:

O arquivo pacientes.csv tem a seguinte estrutura:

nome_completo João da Silva Maria Souza Santos Pedro Henrique Alves Ana Clara Oliveira

data_do_atendimento 2024-07-12 2024-07-12 2024-07-12 2024-07-11

Tarefa:

````
Crie um arquivo csv com a estrutura de dados indicada. Seu arquivo deve ser separado por vírgulas
    Use a função file.create(pacientes.csv) para criar o arquivo de dados vazio no diretório de t
    Crie duas variáveis e atribua os valores indicados acima utilizando uma das funções de estrut
    Inclua os dados no arquivo criado.

    Use o exemplo de scritp abaixo como inspiração:

    ```r
    # Definir o diretório de trabalho
    setwd("C:/dir_projeto")

    # Criar o arquivo CSV vazio
    file.create("dados.csv")

    # Criar as variáveis
    nome <- "João"
    idade <- 35

    # Criar o dataframe com os dados
    dados <- data.frame(nome, idade)

    # Escrever os dados no arquivo CSV
    write.csv(dados, "dados.csv", row.names = FALSE)
    ```


Leitura dos Dados:
    Use a função read.csv() para carregar os dados do arquivo pacientes.csv em um dataframe.

Manipulação de Strings:
    Crie duas novas colunas no dataframe: primeiro_nome e ultimo_nome.
    Utilize a função strsplit() para separar os nomes completos em vetores de palavras.
    Extraia a primeira palavra de cada vetor e armazene na coluna primeiro_nome.
    Extraia a última palavra de cada vetor e armazene na coluna ultimo_nome.

Escrita dos Dados:
    Use a função write.csv() para salvar o dataframe modificado (contendo as colunas primeiro_nom
````

22.0.1.1 3. SCRIPT FINAL DO CURSO

As estruturas condicionais nos permitem interar com os registros das bases de dados, realizar cálculo ou fazer transformações. Abaixo colocamos uma proposta de uso dessas estruturas que poderiam ser utilizadas no script final deste curso. Fique à vontade para copiar e fazer as alterações e testes que desejar.

Os dados estão disponíveis neste link -> https://github.com/bmclima/cursoRBasico/blob/main/docs/dados/stations.csv

```
# Criar um dataframe para armazenar as taxas de mortalidade
taxas_mortalidade <- data.frame(
  faixa_etaria = c("0-19 anos", "20-39 anos", "40-59 anos", "60 anos ou mais"),
  obitos = c(0, 0, 0, 0),
  populacao = c(1000000, 1000000, 1000000, 1000000),
  taxa_mortalidade = c(0, 0, 0, 0)
)

# Calcular a taxa de mortalidade por faixa etária
for (i in 1:nrow(df_sim)) {
  idade <- df_sim$IDADE[i]

  if (idade >= 0 && idade <= 19) {
    taxas_mortalidade$obitos[1] <- taxas_mortalidade$obitos[1] + 1
  } else if (idade >= 20 && idade <= 39) {
    taxas_mortalidade$obitos[2] <- taxas_mortalidade$obitos[2] + 1
  } else if (idade >= 40 && idade <= 59) {
    taxas_mortalidade$obitos[3] <- taxas_mortalidade$obitos[3] + 1
  } else {
    taxas_mortalidade$obitos[4] <- taxas_mortalidade$obitos[4] + 1
  }
}

# Calcular a taxa de mortalidade por 100.000 habitantes
taxas_mortalidade$taxa_mortalidade <- (taxas_mortalidade$obitos / taxas_mortalidade$po

# Exibir os resultados
print(taxas_mortalidade)
```

Chapter 23

Materiais e referências

- Repositório oficial de pacotes do R
- Documentação do R
- Site oficial do Tidyverse
- Link para os exercícios e scripts de exemplo do curso

Chapter 24

Pipeline de Dados e Projeto Final

Essa é a última parte do curso. Chegou o momento de aplicar o que foi aprendido em um projeto curto.

Chapter 25

Pipeline de Dados

Uma pipeline de dados é uma sequência de etapas interconectadas que permitem a coleta, processamento, transformação e armazenamento de dados de forma automatizada e eficiente. É como uma linha de produção, onde os dados brutos entram em uma ponta e, após passarem por diversas etapas de refinamento, saem na outra ponta como registros úteis e prontos para análise.

Os componentes de uma Pipeline de Dados são:

- **Coleta**: Extração de dados de diversas fontes, como bancos de dados, APIs, arquivos CSV, etc.
- **Limpeza**: Remoção de dados duplicados, inconsistentes ou incorretos, além de padronização de formatos.
- **Transformação**: Conversão dos dados em um formato adequado para análise, como agregação, filtragem, cálculo de novas métricas, etc.
- **Armazenamento**: Carregamento dos dados processados em um local de destino, como um banco de dados, data warehouse ou data lake.
- **Análise**: Utilização de ferramentas de análise de dados para extrair insights e gerar relatórios.
- **Visualização**: Criação de gráficos e dashboards para apresentar os resultados de forma clara e intuitiva.

Em nosso projeto final vamos simular cada uma dessas etapas.

Chapter 26

Script comentado

Antes de darmos os detalhes do que deverá ser feito no projeto final, vamos apresentar um script que realizar as tarefas relacionadas a uma pipeline de dados. É um código resumido e simplificado, apenas para ilustrar o que foi estudado e inspirar você no seu projeto final.

26.1 1. Coleta de Dados

```
# Instala e carrega os pacotes necessários
install.packages('remotes')
library(remotes)
remotes::install_github("rfsaldanha/microdatasus")
install.packages(c('tidyverse', 'readr'))
library(microdatasus)
library(tidyverse)
library(readr)

# Define o diretório de trabalho (substitua pelo caminho correto)
setwd('[caminho da pasta de trabalho]')

# Baixa os dados do SIM (substitua 'DF' pelo estado desejado)
sim <- fetch_datasus(year_start = 2010, year_end = 2010, uf = "DF", information_system = "SIM-DO"

# Lê os dados das estações (substitua pelo caminho correto)
estacoes <- read_csv("/cloud/project/projetosR/stations.csv")
```

26.2 2. Limpeza e Transformação dos Dados

```
# Pré-processa os dados do SIM
sim_df <- process_sim(sim)

# Cria um dataframe para armazenar as taxas de mortalidade (opcional)
taxas_mortalidade <- data.frame(
  faixa_etaria = c("0-19 anos", "20-39 anos", "40-59 anos", "60 anos ou mais"),
  obitos = c(0, 0, 0, 0),
  populacao = c(1000000, 1000000, 1000000, 1000000),  # Substitua pelos valores reais
  taxa_mortalidade = c(0, 0, 0, 0)
)
```

26.3 3. Análise dos Dados

```
# Calcula a taxa de mortalidade por faixa etária (opcional)
for (i in 1:nrow(sim_df)) {
  idade <- sim_df$IDADE[i]

  if (idade >= 0 && idade <= 19) {
    taxas_mortalidade$obitos[1] <- taxas_mortalidade$obitos[1] + 1
  } else if (idade >= 20 && idade <= 39) {
    taxas_mortalidade$obitos[2] <- taxas_mortalidade$obitos[2] + 1
  } else if (idade >= 40 && idade <= 59) {
    taxas_mortalidade$obitos[3] <- taxas_mortalidade$obitos[3] + 1
  } else {
    taxas_mortalidade$obitos[4] <- taxas_mortalidade$obitos[4] + 1
  }
}

# Calcula a taxa de mortalidade por 100.000 habitantes (opcional)
taxas_mortalidade$taxa_mortalidade <- (taxas_mortalidade$obitos / taxas_mortalidade$po
```

26.4 4. Visualização dos Dados

```
# Exibe um resumo dos dados (opcional)
summary(estacoes)
summary(sim_df)
```

```r
# Exibe a estrutura dos dados (opcional)
str(estacoes)
str(sim_df)

# Exibe os resultados das taxas de mortalidade (opcional)
print(taxas_mortalidade)
```

26.5 5. Armazenamento dos Dados (Opcional)

```r
# Salva o dataframe processado em um arquivo CSV
write.csv(sim_df, "sim_processado.csv", row.names = FALSE)

# Salva o dataframe de taxas de mortalidade em um arquivo CSV (opcional)
write.csv(taxas_mortalidade, "taxas_mortalidade.csv", row.names = FALSE)
```

26.6 Script completo

```r
# Instala e carrega os pacotes necessários
install.packages('remotes')
library(remotes)
remotes::install_github("rfsaldanha/microdatasus")
install.packages(c('tidyverse', 'readr'))
library(microdatasus)
library(tidyverse)
library(readr)

# Define o diretório de trabalho (substitua pelo caminho correto)
setwd('[caminho da pasta de trabalho]')

# Baixa os dados do SIM (substitua 'DF' pelo estado desejado)
sim <- fetch_datasus(year_start = 2010, year_end = 2010, uf = "DF", information_system = "SIM-DO

# Pré-processa os dados do SIM
sim_df <- process_sim(sim)

# Cria um dataframe para armazenar as taxas de mortalidade (opcional)
taxas_mortalidade <- data.frame(
  faixa_etaria = c("0-19 anos", "20-39 anos", "40-59 anos", "60 anos ou mais"),
  obitos = c(0, 0, 0, 0),
```

```r
  populacao = c(1000000, 1000000, 1000000, 1000000),  # Substitua pelos valores reais
  taxa_mortalidade = c(0, 0, 0, 0)
)

# Calcula a taxa de mortalidade por faixa etária (opcional)
for (i in 1:nrow(sim_df)) {
  idade <- sim_df$IDADE[i]

  if (idade >= 0 && idade <= 19) {
    taxas_mortalidade$obitos[1] <- taxas_mortalidade$obitos[1] + 1
  } else if (idade >= 20 && idade <= 39) {
    taxas_mortalidade$obitos[2] <- taxas_mortalidade$obitos[2] + 1
  } else if (idade >= 40 && idade <= 59) {
    taxas_mortalidade$obitos[3] <- taxas_mortalidade$obitos[3] + 1
  } else {
    taxas_mortalidade$obitos[4] <- taxas_mortalidade$obitos[4] + 1
  }
}

# Calcula a taxa de mortalidade por 100.000 habitantes (opcional)
taxas_mortalidade$taxa_mortalidade <- (taxas_mortalidade$obitos / taxas_mortalidade$po

# Exibe um resumo dos dados (opcional)
summary(sim_df)

# Exibe a estrutura dos dados (opcional)
str(sim_df)

# Exibe os resultados das taxas de mortalidade (opcional)
print(taxas_mortalidade)

# Salva o dataframe processado em um arquivo CSV
write.csv(sim_df, "sim_processado.csv", row.names = FALSE)

# Salva o dataframe de taxas de mortalidade em um arquivo CSV (opcional)
write.csv(taxas_mortalidade, "taxas_mortalidade.csv", row.names = FALSE)
```

Chapter 27

Projeto Final

1.

Objetivo:

Este projeto visa revisar os conceitos e técnicas aprendidos no curso.

Cenário:

Suponha que você é um gestor de saúde do estado de TO e precisa cálcular a taxa de nascidos vivos para o ano de 2019.

Exercício:

```
Construa um script em R que solucione o cenário proposto e inclua as cinco etapas de uma pipeline
```

Regras:

```
- Você deve utilizar o pacote microdatasus para coletar os dados e preprocessar os dados.
- Use o script inserido acima como inspiração, mas use a criatividade. Explore o material do curs
- Seu script deve ser salvo em um arquivo chamado projeto_final_[seunomecompleto].R
- Você deve enviar o arquivo do script para o formulário disponível em <https://docs.google.com/f
- Deixamos alguns scripts e referências no repositório do curso. Fique à vontade para usá-los na
```

Chapter 28

Materiais e referências

- Repositório oficial de pacotes do R
- Documentação do R
- Site oficial do Tidyverse
- Link para os exercícios e scripts de exemplo do curso