Cursos Integrados em Vigilância em Saúde

Curso

Visualização de dados de interesse para a vigilância em saúde

Instituto Todos pela Saúde

ABRASCO
ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA

UNA-SUS
Universidade Aberta do SUS

FIOCRUZ

UFPEL

UFSC

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Reitor Irineu Manoel de Souza
Vice-Reitora Joana Célia dos Passos
Pró-Reitora de Pós-graduação Werner Kraus
Pró-Reitor de Pesquisa e Inovação Jacques Mick
Pró-Reitor de Extensão Olga Regina Zigelli Garcia

CENTRO DE CIÊNCIAS DA SAÚDE

Diretor Fabrício de Souza Neves
Vice-Diretora Ricardo de Souza Magini

DEPARTAMENTO DE SAÚDE PÚBLICA

Chefe do Departamento Rodrigo Otávio Moretti Pires
Subchefe do Departamento Sheila Rúbia Lindner
Coordenadora do Curso Alexandra Crispim Boing

INSTITUTO TODOS PELA SAÚDE (ITPS)

Diretor presidente Jorge Kalil (Professor titular da Faculdade de Medicina da Universidade de São Paulo; Diretor do Laboratório de Imunologia do Incor)

ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA (ABRASCO)

Presidente Rosana Teresa Onocko Campos

EQUIPE DE PRODUÇÃO

Denis de Oliveira Rodrigues
Kamila de Oliveira Belo
Marcelo Eduardo Borges
Oswaldo Gonçalves Cruz
Alexandra Crispim Boing
Antonio Fernando Boing

Curso

Visualização de dados de interesse para a vigilância em saúde

Dados Internacionais de Catalogação-na-Publicação (CIP)

V834 Visualização de dados de interesse para a vigilância em saúde/ Denis de Oliveira Rodrigues, Oswaldo Gonçalves Cruz, Kamila de Oliveira Belo e Marcelo Eduardo Borges . – Santa Catarina ; São Paulo ; Rio de Janeiro : UFSC ; ITPS ; Abrasco; 2022. 96p. (Cursos Integrados em Vigilância em Saúde).

Publicação Online
10.52582/curso-analise-dados-vigilancia-modulo6

1. Vigilância em saúde 2. Análise de dados I. Título

Sumário

Visualização de dados de interesse para a vigilância em saúde

Fazer uma boa apresentação que valorize sua análise de dados e a torne simples para seu interlocutor pode ser desafiador, não é mesmo?! Na vigilância em saúde comunicar análises é uma etapa essencial do trabalho e para isso podemos utilizar diversas estratégias de **visualização dos dados**.

O profissional de vigilância em saúde produz slides, tabelas, boletins, informes e relatórios diariamente. Estes materiais organizam dados e revelam informações valiosas que apoiam a tomada de decisão de profissionais de saúde e gestores. Neste contexto, a comunicação visual é chave para informar, e os gráficos que compõem a maioria dos relatórios descritivos dão forma às análises da vigilância.

Mas, qual a melhor maneira de se produzir visualização de dados? Qual gráfico utilizar? Existe gráfico específicos para informar um evento ou doença?

Neste curso, você desenvolverá diversas visualizações de dados de forma rápida com poucos comandos. Com o apoio da linguagem de programação R você poderá automatizar a produção gráfica e torná-la mais interessante e atrativa. Você aprenderá a utilizar cores e escolher os gráficos corretos para as suas análises.

Você quer análises ainda mais poderosas e gráficos com excelente qualidade visual? Este curso é para você!

Ao final deste curso, você será capaz de:

1. produzir gráficos de barras e de linhas;

2. criar e interpretar histogramas e *boxplots*;

3. produzir pirâmides etárias;

4. criar gráficos temporais de calor (*calendar heatmaps*);

Atenção

Para seguir com este curso, você deve conhecer as ferramentas básicas para uso da linguagem `R` e do `RStudio`, além de possuir conhecimentos básicos de rotinas de análises e visualização de dados utilizando a linguagem de programação `R`. Lembre-se que você pode acessar a qualquer momento o curso **"Análise de dados para a vigilância em saúde – curso básico"** obtendo os códigos desejados para a confecção de seus gráficos. Caso não tenha feito os cursos, sugerimos fortemente que se inscreva neles. Maiores informações em https://www.abrasco.org.br/site/analise-de-dados-para-a-vigilancia-em-saude/

1. O uso de gráfico na vigilância em saúde

Gráficos são representações de algo que possa ser mensurado, quantificado ou ilustrado de forma mais ou menos lógica. Eles facilitam a análise de dados e quase sempre são primeiramente organizados em tabelas que facilitam sua análise estatística. A partir delas utilizamos algumas ferramentas para interpretação e apresentação de resultados na vigilância em saúde.

Em seu dia a dia provavelmente você já utilizou ferramentas como o `Microsoft Excel` para produção gráfica, ou ainda utilizou outros *softwares* que apoiam a visualização de dados de vigilância como o Epi Info e o Tabwin. Cada vez mais na realidade dos serviços de vigilância em saúde estamos nos deparando com o aumento do volume de dados e da sua complexidade, o que torna essas ferramentas limitadas para a produção de visualização de dados (*DataViz*).

O uso da linguagem de programação `R` para a produção gráfica trará muito mais praticidade, principalmente quando o volume de dados que você utiliza é consideravelmente grande. Além disso, o `R` possibilitará que você possa automatizar a produção gráfica, podendo também optar por uma gama diversificada de gráficos que tornam a interpretação da informação mais precisa e clara.

Neste curso, além de produzir uma grande variedade de tipos de gráficos como os de coluna, em barras, pizza, área, linha, pirâmides etárias e gráficos interativos, você aprenderá a escolher os gráficos mais adequados para a representação correta de sua necessidade.

Para isso, no `R` você encontrará diversos pacotes que te permitirão produzir gráficos arrojados e atraentes, com uma ampla gama de possibilidades. Neste curso, você aprenderá a utilizar dois pacotes do `R`: o módulo *base* (nativo do `R`) e o `ggplot2` (um dos componentes do metapacote `tidyverse`). Vamos lá!

1.1 Preparando os dados

Um dos maiores benefícios da linguagem de programação `R` é a facilidade de estruturação de dados e organização das rotinas para análise de vigilância em saúde. Base de dados como a do Sinan Net é facilmente importada e, além disso, é possível escrever um roteiro (um *script*) para a elaboração de gráfico com excelente qualidade visual podendo ser replicado de forma automática.

Vamos práticar! Para iniciarmos a conversa sobre a capacidade gráfica do `R`, vamos utilizar a base de dados proveniente do inquérito epidemiológico do VIGITEL de 2015. O VIGITEL, sigla para Vigilância de Fatores de Risco e Proteção para Doenças Crônicas por Inquérito Telefônico, compõe o sistema de Vigilância de Fatores de Risco para Doenças Crônicas Não Transmissíveis (DCNT) do Ministério da Saúde.

Para nosso exemplo, selecionaremos apenas algumas das variáveis disponíveis no inquérito para apresentarmos uma introdução dos recursos gráficos. Caso necessite acessar o dicionário de dados completo para conhecer todas as variáveis disponíveis acesse AQUI. Seguindo, acesse o menu lateral "Arquivos" do curso e faça *download* do banco de dados de nome `vigitel_2015.csv`. Lembre-se de manter todas as bases de dados deste curso em uma mesma pasta ou diretório.

Primeiro precisaremos importar a base de dados VIGITEL 2015. Para isso utilizaremos a função `read_csv2()` do pacote `readr`, carregado pelo metapacote `tidyverse`. Acompanhe o código a seguir e replique-o em seu `RStudio`:

```
if(!require(tidyverse)) install.packages("tidyverse");library(tidyverse)
```

```
#> Carregando pacotes exigidos: tidyverse
#> ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
#> ✓ ggplot2 3.3.6     ✓ purrr   0.3.4
#> ✓ tibble  3.1.8     ✓ dplyr   1.0.9
#> ✓ tidyr   1.2.0     ✓ stringr 1.4.0
#> ✓ readr   2.1.2     ✓ forcats 0.5.1
#> ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
#> ✖ dplyr::filter() masks stats::filter()
#> ✖ dplyr::lag()    masks stats::lag()
```

```
# Importando o banco de dados {`vigitel_2015.csv`}
# e armazenando os dados no objeto `vigitel`
vigitel <- read_csv2('Dados/vigitel_2015.csv')
```

Pronto, agora os dados estão armazenados no objeto {`vigitel`} e analisaremos a estrutura do banco de dados utilizando a função `glimpse()` do pacote `dplyr`. Perceba que o objeto {`vigitel`} contém 12 colunas (variáveis) e 54.174 linhas (observações) individuais. Acompanhe o código a seguir e replique-o em seu `RStudio`:

```
# Visualizando a estrutura do objeto {`vigitel`}
glimpse(vigitel)
```

```
#> Rows: 54,174
#> Columns: 12
#> $ cidade      <chr> "aracaju", "aracaju", "aracaju", "aracaju", "aracaju", "vi…
#> $ regiao      <chr> "nordeste", "nordeste", "nordeste", "nordeste", "nordeste"…
#> $ idade       <dbl> 59, 20, 53, 32, 70, 84, 84, 27, 71, 72, 70, 64, 56, 64, 52…
#> $ civil       <chr> "casado legalmente", "solteiro", "casado legalmente", "sol…
#> $ sexo        <chr> "masculino", "feminino", "feminino", "feminino", "feminino…
#> $ racacor     <chr> "branca", "parda", "parda", "preta", "parda", "não sabe", …
#> $ peso        <dbl> 76, 84, 77, 45, 62, NA, 51, 57, 77, 67, 74, 61, 69, 63, 65…
#> $ altura      <dbl> 172, 162, NA, 160, 153, 158, 140, 170, 155, 159, 156, 150,…
#> $ bebe        <chr> "não", "não", "não", "não", "sim", "não", "não", "não", "n…
#> $ fuma        <chr> "não", "não", "sim, diariamente", "não", "não", "não", "nã…
#> $ hipertensao <chr> "não", "sim", "não", "não", "sim", "não", "sim", "não", "s…
#> $ diabetis    <chr> "não", "não", "não", "não", "não", "sim", "sim", "não", "n…
```

Observe que a variável `fuma` contém três categorias de respostas (“não”, “sim, diariamente” e “sim, mas não diariamente”) que, para nossos exemplos, vamos categorizar para apenas duas (“sim” e “não”). Para isso, vamos utilizar a função `if_else()`, aninhada à função `mutate()`, ambas do pacote `dplyr`. Com essa função, vamos definir que se a variável for igual a “não”, substitua para “não” e, caso contrário, substitua para “sim”. É uma operação simples de transformação. Acompanhe o código a seguir e replique-o no *script* em seu computador:

```
vigitel <- vigitel |>

  # Transformando os valores da coluna "fuma" para "sim" e "não" apenas
  mutate(fuma = if_else(fuma == "não", "não", "sim"))
```

Pronto, agora que já organizamos nossos dados podemos seguir para o processo de visualização gráfica.

Atenção

Todos os bancos de dados utilizados para análises neste curso se encontram no menu lateral "Arquivos" do curso. Lembre-se de fazer o *download* do material do curso diretamente do Ambiente Virtual de Aprendizagem do curso e arquivá-los em um diretório que deverá indicar para que o `R` o localize.

1.2 Utilizando a função `plot()`

Destacamos para uso em seu dia a dia o módulo base do `R`. Apesar de apresentar gráficos mais simples, ele é muito bom quando necessitamos obter uma visualização gráfica rápida, com poucos comandos, como por exemplo, traçar gráficos de linhas ou de barras ou analisar a distribuição dos dados que serão analisados utilizando um gráfico do tipo *boxplot*.

Vamos iniciar a produção dos gráficos utilizando então a função `plot()`. Ela é a mais genérica função para visualizar objetos gráficos no `R`. Essa função é nativa da linguagem, não sendo necessário nenhum pacote para acessá-la. É muito utilizada para uma análise rápida, mas possui inúmeros recursos que, quando usados em conjunto, tornam-se poderosos auxílios para visualização de dados.

Os argumentos básicos da função `plot()` são:

- `x`: os dados que serão apresentados no eixo x. Usualmente pode ser utilizado sozinho, a depender do tipo de gráfico;
- `y`: os dados que serão apresentados no eixo y é opcional.

Outros argumentos são complementares e serão utilizados para acessarmos alguns recursos gráficos como:

- `type`: elemento que define qual tipo de gráfico será plotado. Alguns tipos muito úteis são:
  - "p" para gráfico de pontos;
  - "l" para gráfico de linhas;
  - "b" para gráfico de pontos e linhas.
- `pch`: um número inteiro que define a forma geométrica utilizada para gráfico de pontos (quadrado, triângulo);
- `col`: cor dos símbolos, das linhas. Existem muitas formas de especificar as cores. Veremos mais à frente sobre isso;
- `cex`: tamanho de texto e símbolos. O padrão é 1, podendo ser ampliado (por exemplo 1.1, 1.2, 1.3) ou reduzido (por exemplo 0.9, 0.8, 0.5);
- `main`: título do gráfico;
- `xlab`: título do eixo x do gráfico;
- `ylab`: título do eixo y;
- `lty`: um número inteiro que define o tipo de linha dos gráficos. Os mais utilizados serão:
  - 1 para linha sólida,
  - 2 para linha tracejada, e
  - 3 para linha pontilhada.
- `lwd`: espessura da linha.

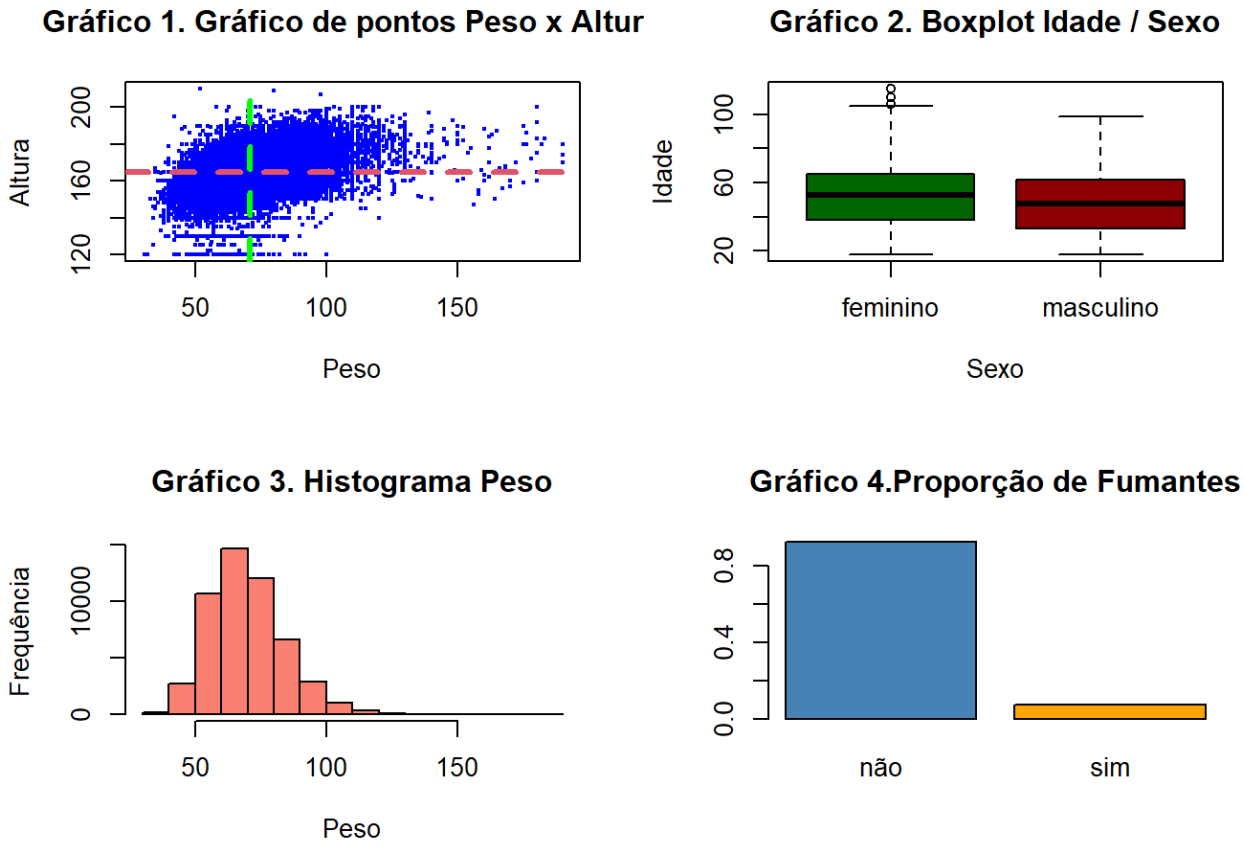
Vamos demonstrar a criação de quatro tipos de gráficos como **exemplo**: gráfico de pontos, *boxplot*, histograma e gráfico de barras. Mais à frente vamos aprofundar nosso conhecimento sobre eles. Neste momento, perceba como podem ser facilmente elaborados com poucas linhas de código. Perceba, também, que a maneira que acessamos as colunas da base é utilizando o cifrão (`$`).

Para visualizar quatro gráficos alinhados e arrumados, utilizaremos a função `par` e o argumento `mfrow`, no qual definimos que queremos dois gráficos por linha divididos em duas colunas (`c(2, 2)`).

Acompanhe o código a seguir e replique-o em seu `RStudio`:

```
par(mfrow = c(2, 2))

plot(
  x = vigitel$peso,
  y = vigitel$altura,
  pch = 19,
  col = "blue",
  cex = 0.3,
  main = "Gráfico 1. Gráfico de pontos Peso x Altura",
  xlab = "Peso",
  ylab = "Altura"
)
abline(v = mean(vigitel$peso, na.rm = T), col = "green", lty = 2, lwd = 3)
abline(h = mean(vigitel$altura, na.rm = T), col = 2, lty = 2, lwd = 3)

# Gráfico 2
boxplot(
  vigitel$idade ~ vigitel$sexo,
  col = c("darkgreen", "darkred"),
  main = "Gráfico 2. Boxplot Idade / Sexo",
  xlab = "Sexo",
  ylab = "Idade"
)

# Gráfico 3
hist(
  x = vigitel$peso,
  col = "salmon",
  main = "Gráfico 3. Histograma Peso",
  xlab = "Peso",
  ylab = "Frequência"
)

# Gráfico 4
barplot(prop.table(table(vigitel$fuma)),
        col = c("steelblue", "orange"),
        main = "Gráfico 4.Proporção de Fumantes")
```

Figura 1: Exemplo de quatro gráficos diferentes gerados em uma mesma imagem.



Conseguiu visualizar? Caso não tenha conseguido revise o código e atente-se para sua escrita seguindo novamente o passo a passo acima. A qualquer momento você poderá copiar e colar o seu erro e pedir ajuda em algum desses locais:

- *Help* ou documentação do R, link de acesso: https://cran.r-project.org/manuals.html.
- *Google*, link de acesso: https://www.google.com.
- Fóruns como o *Stack Overflow*, link de acesso: https://pt.stackoverflow.com.
- Fórum do curso para consultar outros alunos do curso.

Vamos lá! Observe na Figura 1 que o Gráfico 1 apresenta um cruzamento da variável `peso` com a variável `altura` utilizando um **gráfico de pontos**. Ele também plota (imprime) uma linha verde tracejada marcando no *eixo x* o peso médio dos dados utilizados e uma linha vermelha marcando a altura média no* eixo y*. Perceba que aninhamos a função `mean()` dentro da função `abline()`. Esta última é a responsável por criar as linhas. Além disso, para a linha vertical utilizamos o argumento `v` da função, e, para linha horizontal, utilizamos o argumento `h`.

No Gráfico 2 visualizamos um **gráfico do tipo *boxplot*** com o cruzamento da variável categórica `sexo` com a variável numérica `Idade`. Para mostrarmos ao `R` que queríamos este cruzamento utilizamos o operador til (`~`). Observe também que concatenamos as cores num vetor simples, na sequência em que são apresentadas as categorias da variável.

Já no Gráfico 3 conseguimos visualizar **um gráfico do tipo histograma** que apresenta a distribuição da variável `Peso`.

E, finalmente, no Gráfico 4 temos um **gráfico de barras** com a proporção de fumantes, usando o mesmo recurso de cores do Gráfico 2: um vetor com o nome das cores na sequência das categorias apresentadas.

Bem interessante, não é mesmo? Estes são os gráficos mais utilizados no dia a dia de análise de dados. Este exercício inicial, apenas apresenta como é simples e rápido a produção de visualização gráfica com `R`. Agora, precisamos aprofundar os componentes gráficos que nos exigirão um pouco mais de tempo para a personalização. Aqui iniciaremos com um elemento muito importante da visualização de dados: a cor. Siga em frente!

2. Escolhendo as cores

Um importante componente para tornar a visualização gráfica atrativa e intuitiva é a cor. Ela é uma das ferramentas mais poderosas para a comunicação visual e deverá ser escolhida com cuidado e dará a compreensão da identidade visual do seu relatório, tema ou painel de indicadores. Você certamente conhece a marca Ferrari pela cor vermelha de seu carro, por exemplo, ou de empresas como Nu Bank, Mc Donald's, Coca-Cola entre muitas outras que são facilmente reconhecidas por suas cores. As cores despertam emoções e também possuem um papel importante na criação de uma identidade positiva ou negativa para informação que quer comunicar.

Como vimos no exercício praticado na seção anterior, existem muitas maneiras de especificar as cores no `R`. Uma das formas mais usadas é o padrão *RGB* (do inglês: "Red, Green, Blue") que representa a intensidade das cores primárias vermelho, verde e azul. Uma das codificações desse padrão é o formato chamado **hexadecimal**, formado por seis dígitos, dois para cada cor primária. Esses dois dígitos podem ser representados por números que variam de 00 a 99 e pares de letras de AA até FF (por exemplo AB, AC, CE, EF, FF). Assim:

R = 00 a 99 / AA a FF
G = 00 a 99 / AA a FF
B = 00 a 99 / AA a FF

No `R`, quando configuramos cores o símbolo hashtag (`#`) indica um código hexadecimal. Então, um código `#000000` representa a cor preta e `#FFFFFF` representa a cor branca. Combinando as três cores básicas podemos especificar mais de 16 milhões de cores e tons de cinza!

Uma pequena amostra de cores pode ser vista na tabela 1 a seguir, mas é possível visualizar as combinações possíveis clicando em https://www.w3schools.com/colors/colors_hexadecimal.asp. Na tabela 1, mostramos a cor em cada célula e, também, o código hexadecimal.

Tabela 1: Cores com seus códigos hexadecimais.

#FFFFFF	#FFFFEE	#FFFFDD	#FFFFAA	#FFFF50	#FFFF00
#FFDD00	#FFC000	#FFAD48	#FF9900	#D95F0E	#993404
#C7E9C0	#00DD00	#00BB00	#00AA00	#008000	#006000
#00FFFF	#00DDDD	#00AAAA	#008080	#000099	#000060
#00DDFF	#0080FF	#0080D0	#0000FF	#0000DD	#000080
#FEEBE2	#FCC5C0	#FA9FB5	#F768A1	#C51B8A	#7A0177
#FEE5D9	#FCBBA1	#E34A33	#FF0000	#800000	#600000
#FAFAFA	#F2F2F2	#D9D9D9	#BABABA	#404040	#000000

Existem diversas tabelas e ferramentas para ajudar a selecionar as cores como por exemplo w3 color picker

Também estamos disponibilizando um arquivo `tabela_cores.html` contendo todos os nomes das cores que estão nativamente no `R`. Clique no menu lateral "Arquivos", do curso. Lembre-se de fazer o *download* do material do curso diretamente do Ambiente Virtual do curso e arquivá-los no seu computador.

Vamos seguir no passo a passo para que você consiga colocar em prática a escolha das cores no seu dia a dia da vigilância.

3. Criando gráficos

O `ggplot2` é um pacote do `R` criado por Hadley Wickham, que faz parte do universo do *tidyverse*. Esse pacote implementa um sistema de gráficos para visualização de dados, permitindo, inclusive, a criação de gráficos avançados.

O `ggplot2` é diferente dos demais pacotes gráficos existentes no `R`, por simplificar algumas etapas de produção gráfica. Em um primeiro contato você pode encontrar dificuldades para utilizá-lo, mas após se familiarizar com ele é possível obter gráficos completos, com multicamadas e com excelente qualidade visual de forma rápida e efetiva. Acredite, por meio da combinação de seus diversos elementos em poucas linhas de código ele vai revolucionar a sua rotina de trabalho.

Atualmente, podemos encontrar mais de 70 pacotes do `R` que estendem ou implementam funções gráficas da `ggplot2`. Clique aqui e conheça alguns exemplos.

Observe na Tabela 2 a seguir os principais elementos (camadas) que utilizaremos aplicando funções do `ggplot2`.

Tabela 2: Principais camadas, sua descrição e as respectivas funções correspondentes do `ggplot2`.

Elementos	Descrição	Funções
Dado (Data)	o conjunto de dados que será usado	**ggplot()**
Estética (Aesthetics)	variáveis e escalas para mapear o gráfico	**aes()**
Geometria (Geometries)	elementos visuais usados no gráfico	**geom_???()**
Faceta (Facets)	múltiplos gráficos (sub-painéis)	**facet_???()**
Estatística (Statistics)	representação estatística dos dados para melhor entendimento/visualização	**stat_???(),**
Coordenada (Coordinates)	representação espacial dos dados	**coord_???()**
Tema (Themes)	estilos cores aspectos gerias	**theme_???()**

Atenção

Para fazer um gráfico com o `ggplot2` precisamos ao menos das camadas de Dados, Estética e Geometria.

As diversas camadas são aplicadas concomitantemente quando utilizamos o operador **+** (soma). Neste caso, não devemos aplicar o **pipe** (`|>` ou `%>%`) .

Atualmente, diversos pacotes atuam como extensão ao `ggplot2`, adicionando novas funcionalidades. Neste curso usaremos alguns desses pacotes complementares.

3.1 Organizando os dados

O primeiro passo para se fazer um gráfico é conhecer o dado a ser apresentado e adequá-lo à visualização proposta. Cada tipo de gráfico exigirá uma organização específica dos dados para se obter o resultado desejado. Como segundo exercício prático neste curso você construirá gráficos que visualizem os dados **para avaliação das notificações de dengue do Estado de Rosas** (fictício). Para isso você precisará organizar informações que possam apresentar a situação das notificações do estado entre os anos 2007 e 2012 para o secretário de saúde.

Para isso, primeiro precisaremos exportar o nosso banco de dados. Os dados disponibilizados foram exportados do Sinan Net do Estado de Rosas: é o arquivo de nome {`NINDNET.dbf`} que está disponível no menu lateral "Arquivos" do Ambiente Virtual do curso. Clique e faça o *download* para seguirmos com as análises.

Vamos lá?! Carregue e instale os pacotes necessários para produção gráfica. Escreva as linhas de código a seguir no *script* do seu `RStudio`:

```
# Esta primeira linha verifica se o pacote está instalado.
# Caso não esteja, irá prosseguir com a instalação
if(!require(foreign)) install.packages("foreign");library(foreign)
if(!require(knitr)) install.packages("knitr");library(knitr)
if(!require(lubridate)) install.packages("lubridate");library(lubridate)
if(!require(aweek)) install.packages("aweek");library(aweek)
if(!require(xts)) install.packages("xts");library(xts)
if(!require(plotly)) install.packages("plotly");library(plotly)
if(!require(ggTimeSeries)) install.packages("ggTimeSeries");library(ggTimeSeries)
```

Em seguida, importaremos a base de dados {`NINDINET.DBF`}, que se refere às notificações realizadas entre 2007 e 2012 de um estado fictício chamado Rosas. Para isso, vamos utilizar a função `read.dbf()` do pacote `foreign`. Note que o argumento `as.is` marcado como `TRUE` na função transforma as variáveis da base de dados no tipo `character`. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# criando objeto do tipo dataframe (tabela) {`nindi`} com o banco de dados {`NINDINET.dbf`}
nindi <- read.dbf('Dados/NINDINET.dbf', as.is = TRUE)
```

Pronto. Com a base importada, vamos filtrar os registros cujo agravo de notificação foi dengue (campo `ID_AGRAVO` com registro "A90"). Além disso, vamos transformar a variável data dos primeiros sintomas (`DT_SIN_PRI`), armazenada como *character*, para date utilizando a função `ymd` do pacote `lubridate`. Essa função reconhece os separadores típicos de datas e atribui um formato adequado à variável.

A partir da variável `DT_SIN_PRI`, vamos criar três novas variáveis também utilizando funções do pacote `lubridate`. São elas:

- `epiweek()`: retorna a semana epidemiológica a partir de uma variável do tipo data.
- `epiyear()`: retorna o ano conforme o calendário epidemiológico.
- `month()`: retorna o mês referente ao calendário normal.

Também será necessário alterar o tipo da variável `NU_ANO` para o tipo *numeric*. Veja a seguir o *script* completo e replique-o em seu `RStudio`:

```
# Criando a tabela {`dengue`}
dengue <- nindi |>

  # Filtrando os registros de casos de dengue (CID = A90)
  filter(ID_AGRAVO == 'A90') |>

  # Criando novas colunas
  mutate(

    # Transformando a variável `DT_SIN_PRI` para data
    DT_SIN_PRI = ymd(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'sem_epi', referente à semana
    # epidemiológica dos primeiros sintomas
    sem_epi = epiweek(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'ano_epi', referente ao ano epidemiológico
    # dos primeiros sintomas
    ano_epi = epiyear(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'mes', referente ao mês dos primeiros sintomas
    mes = month(DT_SIN_PRI),

    # Transformando a coluna `NU_ANO` no tipo numérico
    NU_ANO = as.numeric(NU_ANO)
  )
```

Pronto já temos os dados necessário para criar as nossas visualizações de dados (*DataViz*).

4. Principais gráficos de interesse da vigilância em saúde

Até aqui você já compreendeu que as representações gráficas devem seguir algumas premissas básicas:

1. Garantia de sua **simplicidade**: o gráfico deve ser simples para que a informação transmitida seja de rápida compreensão.
2. Apresentar **clareza**: o gráfico deve possibilitar a correta interpretação dos valores apresentados. Escolher o melhor tipo de gráfico e suas cores é fundamental para que tenhamos sucesso nesta etapa.
3. Representar informações **verídicas**: ao escolher personalizar um gráfico ou editá-lo você priorizar customizações que garantam a transmissão exata da realidade encontrada nos dados. Ao induzir o leitor ao erro, o gráfico perdeu sua finalidade.

Na epidemiologia, em relação ao seu formato, os gráficos podem ser classificados em três categorias, principais:

- os diagramas: gráficos dispostos em duas dimensões, mais utilizados para representar séries temporais. Os principais são: gráfico em linha ou em curva, gráfico em colunas ou em barras, em barras múltiplas e gráfico em setores (*pizza*);
- os cartogramas: ilustrações que representam posições geográficas ou políticas, os mapas; e
- os estereogramas: gráficos apresentados em três dimensões (3D) e utilizados para indicar volume. É empregado, por exemplo, quando precisamos indicar o número de consultas perdidas em unidade de saúde por dia da semana e sexo dos faltosos.

Nas próximas seções aprofundaremos os principais gráficos utilizados para apresentar dados epidemiológicos, siga em frente. Mas, lembre-se que o objetivo da produção gráfica neste curso é construir uma que tenha o objetivo de **analisar as notificações de dengue do Estado de Rosas** (fictício).

Atenção

A depender do tipo de gráfico escolhido lhe será exigido a reorganização dos dados. Fique atento, pois nas etapas a seguir deste curso você terá que transformar a base de dados todas as vezes que necessitarmos fazer gráficos.

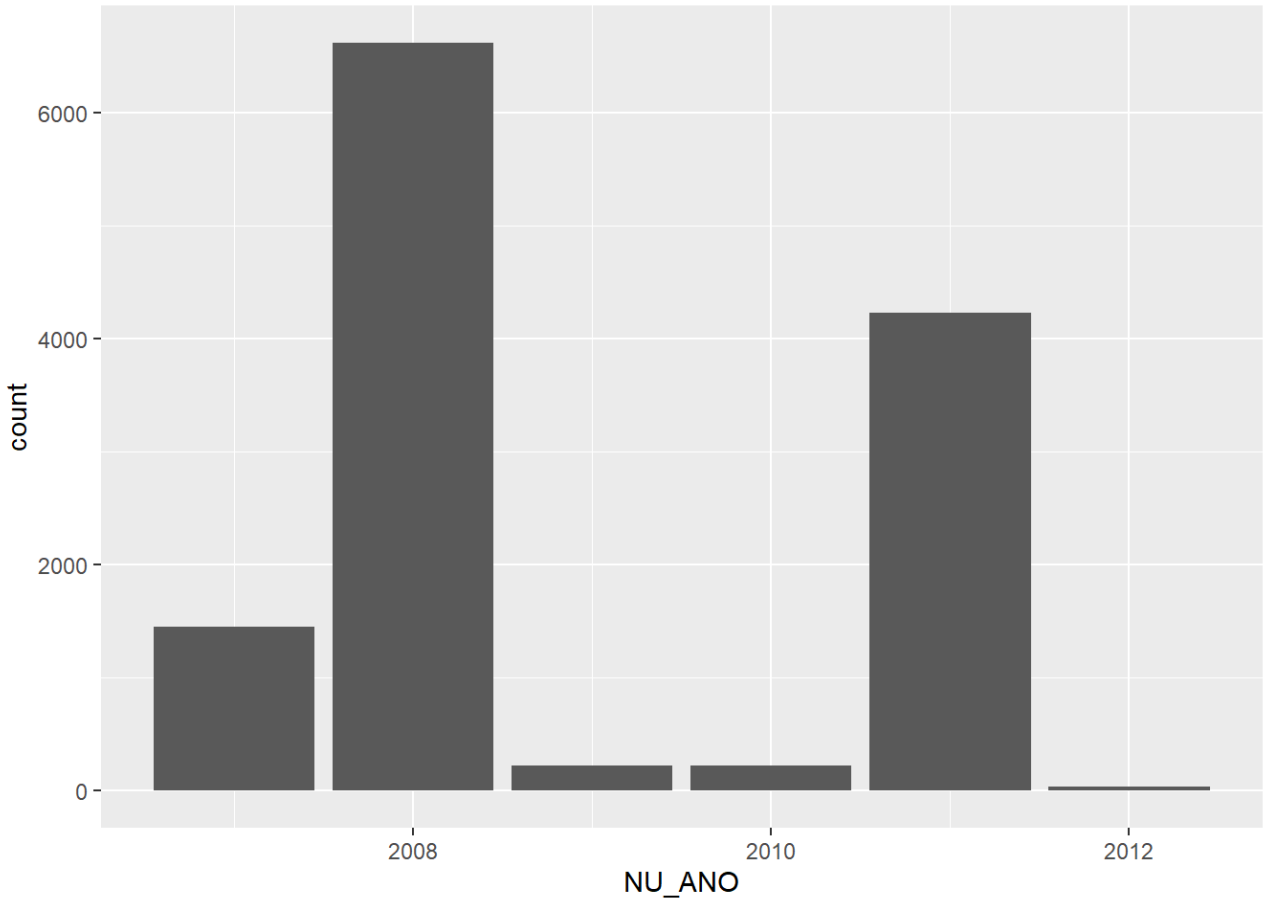
4.1 Gráfico de barras

Vamos iniciar criando um gráfico de barras simples. O gráfico de barras é um dos mais úteis para explorar dados e tem muito destaque na vigilância em saúde. Seja para comparar categorias ou visualizar distribuição de um evento em intervalo de tempo, sempre está presente em análises de vigilância. Basicamente, o tamanho das barras representa os valores a serem representados. É comum usar o gráfico de barras para explorar variáveis categóricas. Em um eixo têm-se as barras representando a variável (e suas categorias ou intervalos) e no outro os valores correspondentes à frequência da variável.

Veja inicialmente o código, replique-o no seu `RStudio` e acompanhe a explicação a seguir.

```
# criando gráfico de barras para avaliação temporal da dengue
ggplot(data = dengue, aes(x = NU_ANO)) + geom_bar()
```

Figura 2: Gráfico de barras com a distribuição dos casos notificados de dengue, por ano.



Percebeu o como foi feito o gráfico. Com o pacote `ggplot2`, informamos o nome do objeto {`dengue`} que contém os dados no argumento data da função `ggplot()`, nesse caso a base {`dengue`}, criada anteriormente. Também definimos a variável que contém a frequência que vamos plotar no argumento `x`, da função `aes()`, no caso a variável `NU_ANO`. Com isso, foi possível fazer um gráfico de barras com a contagem de casos por ano, utilizando o operador `+` e a função de geometria `geom_bar()`.

Bem fácil, não é mesmo? A partir do gráfico acima, você pode notar que 2008 foi um ano com alto número de notificações de dengue no Estado de Rosas, seguido por dois anos de menos casos e certa estabilidade, mas com um novo aumento em 2011.

Observe que a estrutura do código envolveu as funções `ggplot()`, `aes()` e `geom_bar()`. O tema de cores padrão possui fundo cinza e linhas brancas. Mas este tema pode ser alterado especificando a função de um dos temas disponíveis com o pacote. Além disso, também é possível alterar elementos específicos de cada tema, como tamanho da letra do título e eixos. Alguns dos temas disponíveis são:

- `theme_grey()` (ou `theme_gray()`): tema padrão;
- `theme_bw()`: uma variação do padrão que usa fundo branco e linhas cinzas;
- `theme_linedraw()`: um tema com apenas linhas pretas de várias larguras sobre fundo branco;
- `theme_light()`: semelhante ao anterior, mas com linhas e eixos cinza claro , para dirigir mais atenção para os dados;
- `theme_dark()`: "o primo escuro" de `theme_light()`, com linhas de larguras semelhantes, mas com um fundo escuro. Útil para destacar cores de linhas finas;
- `theme_minimal()`: um tema minimalista, sem anotações ou fundo;
- `theme_classic()`: Um tema de aspecto "clássico", com linhas de eixo x e y destacado e nenhuma linha de grade (as linhas internas do gráfico);
- `theme_void()`: um tema completamente vazio.

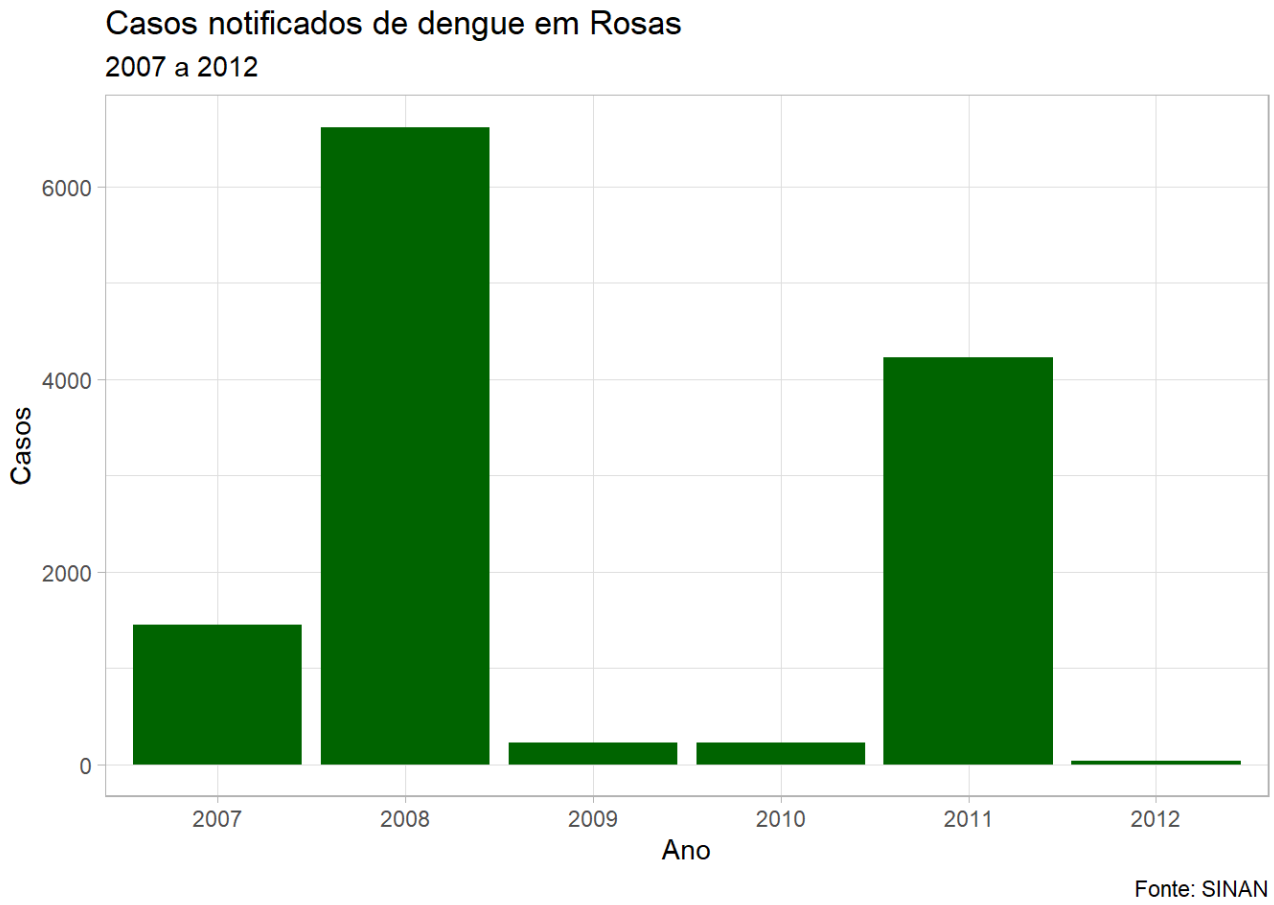
Certo, agora vamos exercitar alterando o mesmo gráfico. Vamos alterar o tema padrão para `theme_light()`. Não é necessário nenhum argumento adicional. Além disso, vamos alterar alguns elementos de comunicação visual como a cor das barras, usando o argumento `fill` na função de geometria, e elementos de comunicação textual como título e subtítulo. Nesta última alteração, vamos usar a função `labs()` e os seguintes argumentos:

• `title`: título do gráfico; • `subtitle`: subtítulo do gráfico; • `caption`: informação de rodapé; •`x`: título do eixo x; • `y`: título do eixo y.

Acompanhe com atenção como fica o *script* a seguir. Nele estamos salvando o gráfico elaborado anteriormente no objeto {`graf_barras`}. Copie e replique os códigos em seu `RStudio`:

```
# Criando o objeto gráfico {`graf_barras`}
graf_barras <- ggplot(dengue, aes(x = factor(NU_ANO))) +

  # Adicionando uma geometria de barras e definindo a cor do preenchimento
  # das barras
  geom_bar(fill = 'darkgreen') +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico
  labs(
    title = 'Casos notificados de dengue em Rosas',
    subtitle =  '2007 a 2012',
    caption = 'Fonte: SINAN',
    x = "Ano",
    y = "Casos"
  ) +

  # Definindo o tema base do gráfico
  theme_light()

# Plotando o objeto `graf_barras`
print(graf_barras)
```

Figura 3: Gráfico de barras, na cor verde, com a distribuição dos casos notificados de dengue, por ano.



Pronto! O objeto {`graf_barras`} criado pode ser visualizado como gráfico sempre que utilizamos a função `print()` utilizando como argumento o nome do objeto ou adicionando camadas utilizando o operador `+`, como veremos a seguir.

Repare no exemplo acima em que a variável `NU_ANO` foi transformada em fator na hora de se fazer o gráfico (`factor(NU_ANO)`). Percebeu o que mudou em relação ao primeiro gráfico?

Além de transformar os dados antes de gerar o gráfico, algumas vezes temos melhores resultados mudando o tipo da variável de numérica para categórica. Neste exemplo, ao transformar a variável `NU_ANO` para categórica, estamos forçando que todos os anos sejam representados no eixo x. Caso a variável se mantivesse como numérica nem todos seriam mostrados, sendo necessário especificar algum argumento adicional mais detalhado para o mesmo procedimento.

É possível também customizar opções para mudar a aparência do gráfico especificando, por exemplo, tipo de fonte, cor, orientação da fonte dos eixos (90º, 45º etc..) e muitas outras opções!

Vamos seguir praticando usando o objeto {`graf_barras`} com os casos de dengue notificados no Estado de Rosas, modificando o tema padrão, alterando o tamanho e a cor (azul-escuro) da fonte dos eixos x e y. No eixo x, vamos girar a fonte em 45 graus.

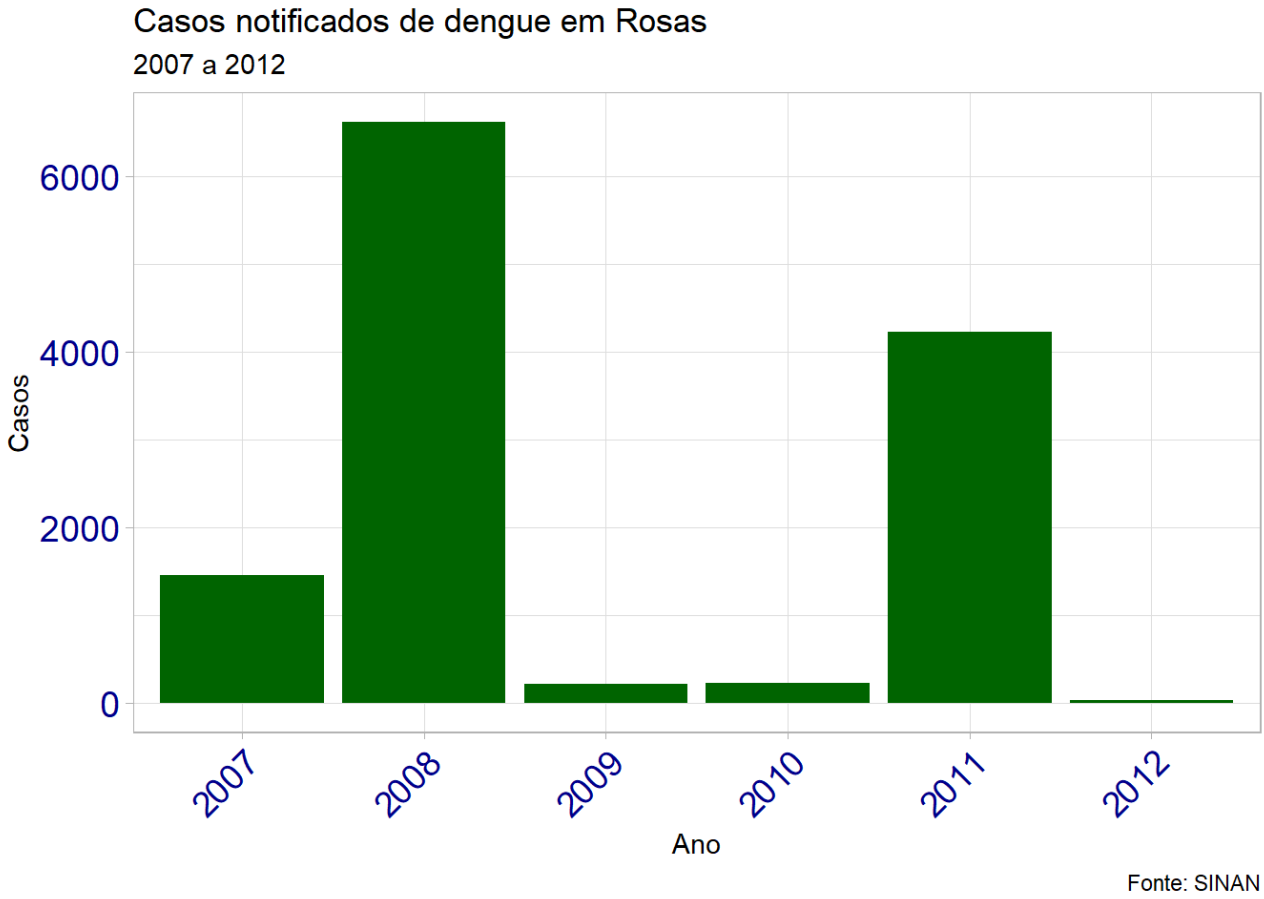
Para as alterações, vamos utilizar a função `theme()`, que possui dezenas de argumentos que se referem a diversas formatações do gráfico. Como vamos fazer alterações no texto dos eixos, vamos precisar dos argumentos `axis.text.x` e `axis.text.y`, utilizando a função `element_text()`. Essa função controla elementos de texto e, aqui, vamos usar:

- `angle`: ângulo da orientação dos títulos dos eixos;
- `hjust`: posição horizontal do texto;
- `size`: tamanho do texto;
- `color`: cor do texto.

Perceba que vamos utilizar o objeto salvo anteriormente {`graf_barras`} e sobrepor novas camadas utilizando o operador (`+`). Essa estratégia é muito útil para evitar repetir o código várias vezes. Acompanhe o *script* a seguir com atenção e replique-o em seu `RStudio`:

```
# Utilizando o objeto gráfico {`graf_barras`} salvo anteriormente
graf_barras +

  # Alterando o tema do gráfico
  theme(

    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 45,         #  ângulo da orientação dos títulos
      hjust = 1,          #  definindo a posição horizontal do texto
      size = 14,          #  tamanho da letra
      color = 'darkblue'  #  definindo a cor da letra
    ),

    # Alterando o texto do eixo y
    axis.text.y = element_text(
      angle = 0,          #  ângulo da orientação dos títulos
      hjust = 1,          #  definindo a posição horizontal do texto
      size = 14,          #  tamanho da letra
      color = 'darkblue'  #  definindo a cor da letra
    )
  )
```

Figura 4: Gráfico de barras, com eixos editados, da distribuição dos casos notificados de dengue, por ano.



Fonte: SINAN

4.2 Gráfico de linhas

Agora, vamos elaborar um gráfico de linhas com a contagem de casos por ano segundo o sexo do paciente notificado com suspeita de dengue, no estado fictício de Rosas, entre os anos de 2007 e 2012.

Para isso, precisaremos preparar uma base de dados que apresente a contagem que precisamos, conforme apresentado na Tabela 3:

1. primeiro, vamos salvar o resultado dos próximos passos ao objeto {`dengue_ano`},
2. em seguida, vamos agrupar os dados por ano e sexo,
3. vamos filtrar os casos cuja categoria da variável sexo (`CS_SEXO`) sejam diferentes de ignorado ('I'),
4. realizaremos a contagem conforme os grupos, e
5. por último, utilizaremos a função `head()`.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando a tabela {`dengue_ano`}
dengue_ano <- dengue |>

  # Filtrando os registros cuja coluna `CS_SEXO` não tenha registros como `I`
  filter(CS_SEXO != 'I') |>

  # Agrupando as notificações pelo ano e sexo
  group_by(NU_ANO, CS_SEXO) |>

  # Contando a frequência de notificações
  count(name = 'n_casos')
```

Agora, vamos visualizar as primeiras linhas com o *script* a seguir. Repita em seu `RStudio`:

```
kable(head(dengue_ano, 10))
```

Tabela 3: Organização dos dados para criação do gráfico de linhas (Figura 5).

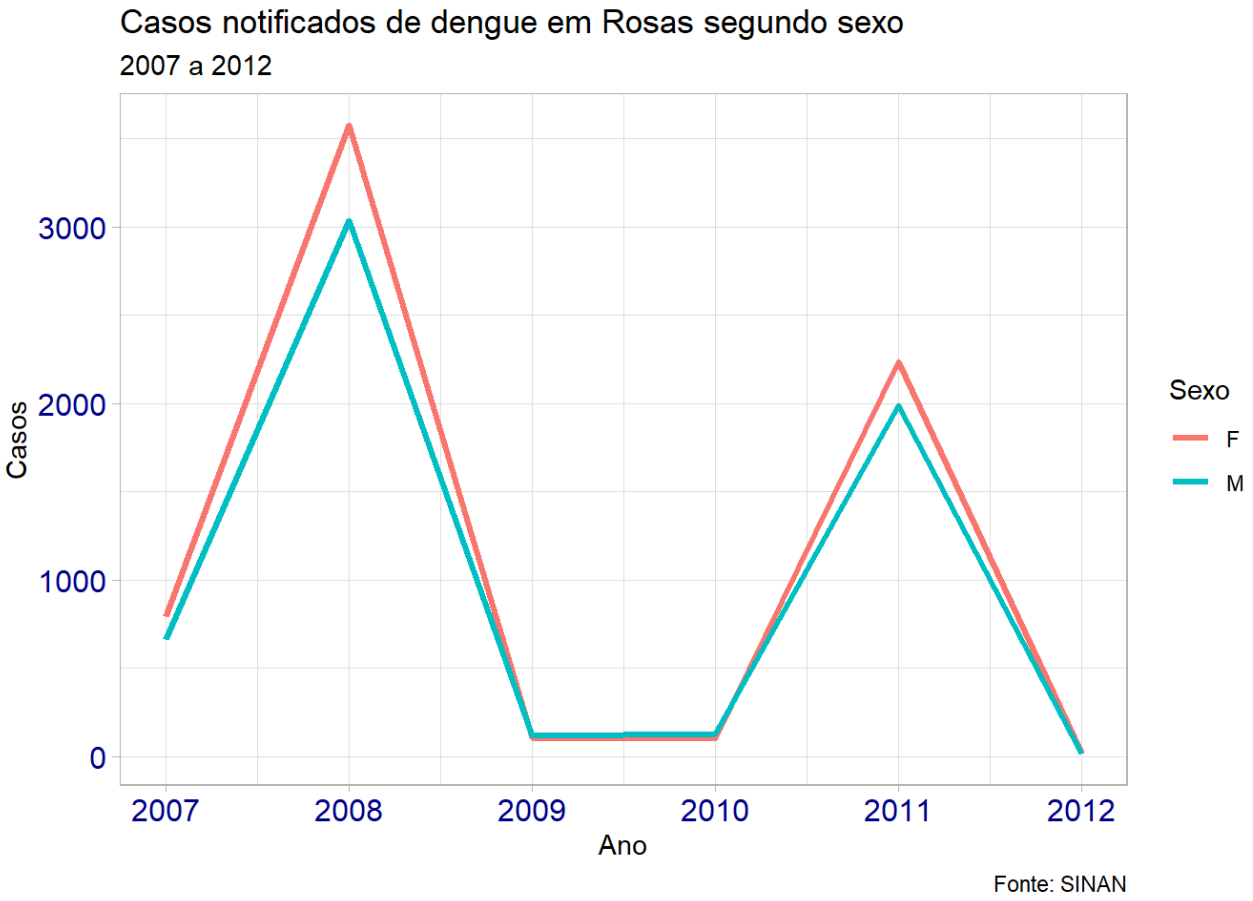
NU_ANO	CS_SEXO	n_casos
2007	F	791
2007	M	662
2008	F	3575
2008	M	3034
2009	F	104
2009	M	118
2010	F	100
2010	M	126
2011	F	2235
2011	M	1988

Com o dado preparado, vamos utilizar na camada de geometria a função `geom_line()`. Como recurso estético, vamos padronizar que cada sexo no gráfico tenha uma cor, inserindo o nome da variável no argumento `color` dentro da função `aes()`. Vamos também aumentar a espessura das linhas usando o argumento `size` na função `geom_line()`. Usaremos as definições de tema apresentadas anteriormente com recursos visuais adicionais ao tema `theme_light()`.

Os resultados desses procedimentos serão salvos em um objeto chamado `graf_linhas`. Acompanhe o código a seguir e copie em seu `RStudio`:

```
# Criando o objeto gráfico {`graf_linhas`}
graf_linhas <- ggplot(data = dengue_ano) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e definindo a variável usada para a cor dos pontos
  aes(x = NU_ANO, y = n_casos, color = CS_SEXO) +

  # Adicionando a geometria de linhas e definindo espessura
  geom_line(size = 1.2) +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico.
  # Para legenda, estamos definindo que o título será "Sexo"
  labs(
    title = 'Casos notificados de dengue em Rosas segundo sexo',
    subtitle = '2007 a 2012',
    caption = 'Fonte: SINAN',
    x = "Ano",
    y = "Casos",
    color = "Sexo"
  ) +

  # Definindo o tema base
  theme_light() +

  # Alterando o tema do gráfico
  theme(

    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 0,
      hjust = 0.5,
      size = 12,
      color = 'darkblue'
    ),

    # Alterando o texto do eixo y
    axis.text.y = element_text(
      angle = 0,
      hjust = 1,
      size = 12,
      color = 'darkblue'
    )
  )

# Plotando o objeto `graf_linhas`
graf_linhas
```

Figura 5: Gráfico de linhas, com legenda à direita, da distribuição dos casos de dengue segundo sexo.

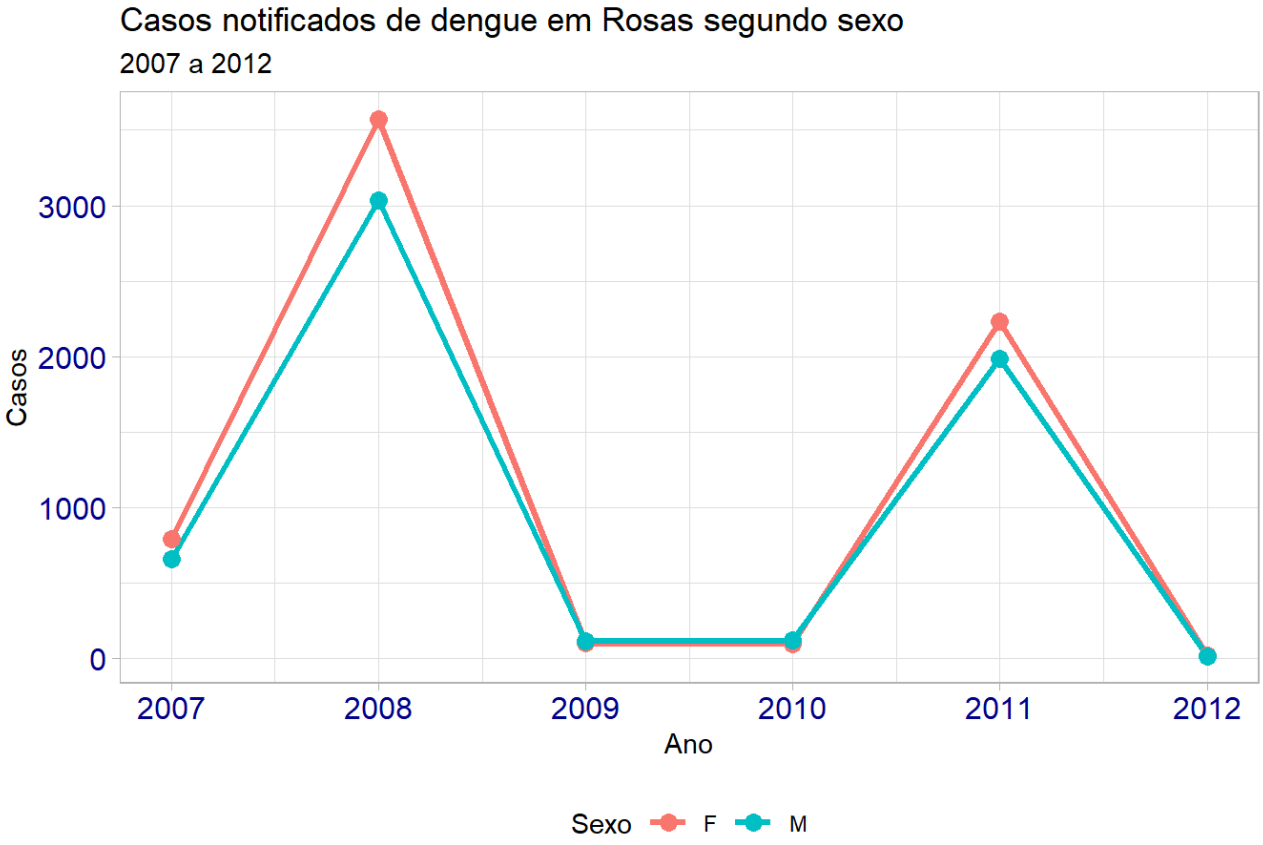


Fonte: SINAN

Você percebeu que estamos, pouco a pouco, aumentando os elementos e complexidade da elaboração de um gráfico? No próximo exemplo vamos adicionar uma camada de geometria de pontos de tamanho maior sobre as linhas (`size = 3`) e alterar a posição da legenda do lado direito para baixo. Como o objeto gráfico foi salvo anteriormente (`graf_linhas`), vamos somente utilizar o operador `+`. Acompanhe o código a seguir e repita o processo no seu `RStudio`:

```
# Utilizando o objeto gráfico {`graf_linhas`} salvo anteriormente
graf_linhas +

  # Adicionando a geometria de pontos e definindo o tamanho
  geom_point(size = 3) +

  # Alterando o tema do gráfico e posicionando a legenda na
  # parte inferior do gráfico
  theme(legend.position = 'bottom')
```

Figura 6: Gráfico de linhas, com legenda inferior central, da distribuição dos casos de dengue segundo sexo.



Fonte: SINAN

No gráfico acima, você pode notar a diferença em números absolutos dos casos notificados de dengue segundo o sexo do paciente, sendo as mulheres mais acometidas.

O gráfico de linhas é muito útil para visualização de séries temporais. Para visualizar um exemplo de um gráfico que aborde esta temática, vamos retornar aos dados da UF de Rosas. Mas, agora, além de criar variáveis para semana e ano referentes ao calendário epidemiológico, vamos determinar a data de início da semana epidemiológica utilizando a função `get_date()` do pacote `aweek`. Essa função tem como argumentos principais a semana (argumento `week`) e o ano de referência (argumento `year`) para retorno da data adequada.

Por fim, vamos contar o número de eventos entre grupos de ano, semana e data inicial da semana e atribuir o resultado a um objeto chamado `dengue_semana`.

Veja a seguir o *script* completo e replique-o em seu computador:

```
# Criando o objeto {`dengue_semana`}
dengue_semana <- dengue |>

  # Filtrando os registros com data de primeiros sintomas maior ou igual
  # a data de primeiro de janeiro de 2007.
  filter(DT_SIN_PRI >= '2007-01-01') |>

  # Utilizando a função `mutate()` para criar novas colunas
  mutate (

    # Criando uma nova coluna chamada 'SEM_EPI` referente à semana
    # epidemiológica dos primeiros sintomas
    SEM_EPI = epiweek(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'ANO_EPI` referente ao ano
    # epidemiológicos dos primeiros sintomas
    ANO_EPI = epiyear(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'DT_INI_SEM` de data de início
    # da semana epidemiológica
    DT_INI_SEM = get_date(week = SEM_EPI, year = ANO_EPI)

  ) |>

  # Agrupando as notificações pelo ano epidemiológico, semana
  # epidemiológica e data de início da semana epidemiológica
  group_by(ANO_EPI, SEM_EPI, DT_INI_SEM) |>

  # Contando a frequência de notificações
  count(name = 'n_casos')
```

O objeto salvo do código acima será utilizado para a criação de um gráfico de linhas com escala de tempo mais detalhada, no caso dia. Usaremos novamente a função **geom_line()** para obter uma linha que representa a evolução do número de casos de dengue no decorrer do tempo.

Atenção

Lembre-se que você deverá sempre verificar se todos os pacotes necessários foram instalados e carregados no `R`, caso encontre algum aviso (warning) ou erro (error) revise seu código cuidadosamente e rode-o novamente.

Como recurso estético, vamos alterar a cor e a espessura da linha e utilizar o tema `theme_classic()`. Outros comandos para alterar o visual do gráfico também estão sendo utilizados como:

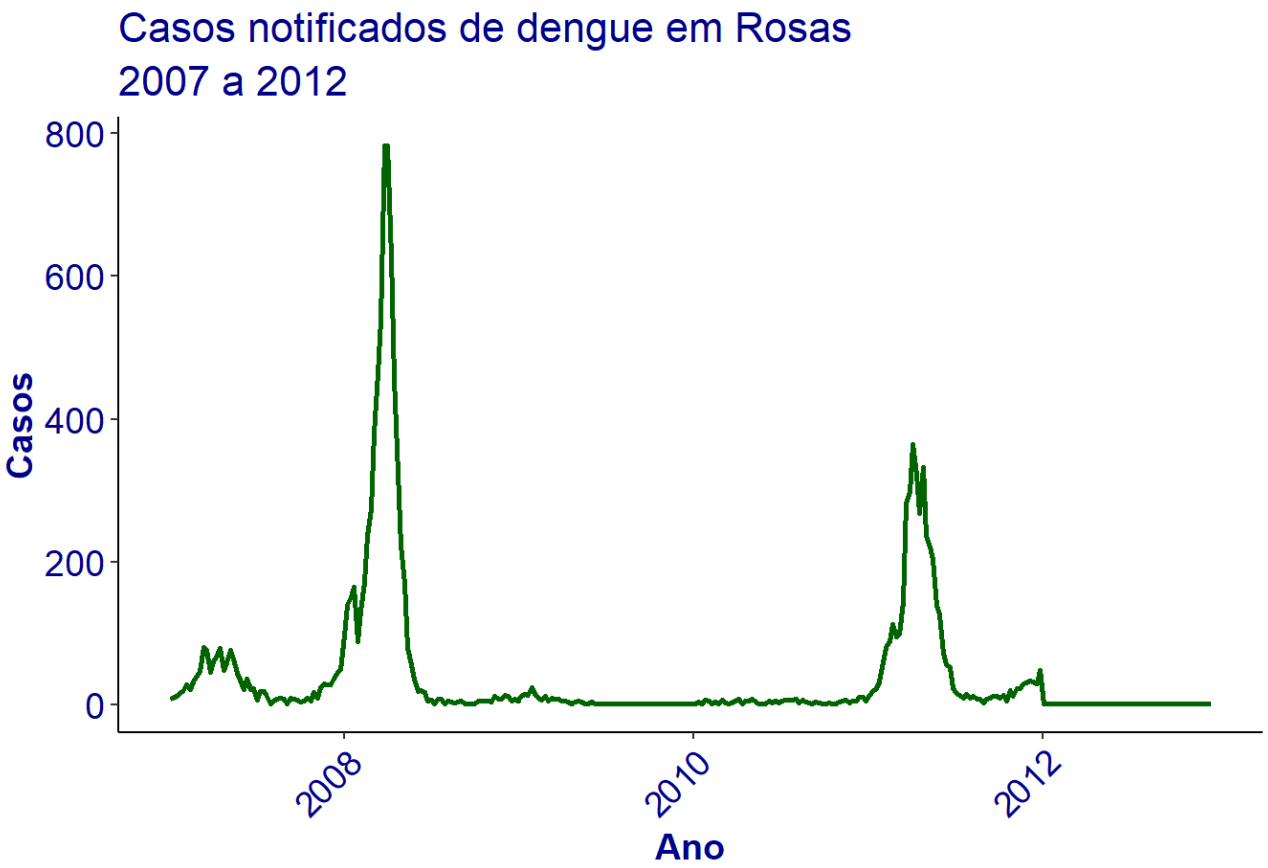
- `plot.title`: altera o visual do título do gráfico;
- `plot.subtitle`: altera o visual do subtítulo do gráfico;
- `axis.title.x`: altera o visual do título do eixo x;
- `axis.title.y`: altera o visual do título do eixo y.

Todas os argumentos que alteram visual de texto devem utilizar a função auxiliar `element_text()`.

O resultado desses procedimentos está sendo salvo no objeto chamado {`graf_linhas2`}. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando o objeto gráfico {`graf_linhas2`}
graf_linhas2 <- ggplot(data = dengue_semana) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  aes(x = DT_INI_SEM, y = n_casos) +

  # Adicionando a geometria de linhas e definindo cor e espessura
  geom_line(color = 'darkgreen', size = 1) +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico
  labs(
    title = 'Casos notificados de dengue em Rosas',
    subtitle =  '2007 a 2012',
    caption = 'Fonte: SINAN',
    x = "Ano",
    y = "Casos"
  ) +

  # Definindo o tema base
  theme_classic() +

  # Alterando o tema do gráfico
  theme(
    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 45,
      hjust = 1,
      size = 14,
      color = 'darkblue'
    ),

    # Alterando o texto do eixo y
    axis.text.y = element_text(
      angle = 0,
      hjust = 1,
      size = 14,
      color = 'darkblue'
    ),

    # Alterando o tamanho e cor da letra do título do gráfico
    plot.title = element_text(size = 16, color = 'darkblue'),

    # Alterando o tamanho e cor da letra do subtítulo do gráfico
    plot.subtitle = element_text(size = 16, color = 'darkblue'),

    # Alterando o tamanho, cor e estilo da letra do título do eixo x
    axis.title.x = element_text(size = 14, color = 'darkblue', face = "bold"),

    # Alterando o tamanho, cor e estilo da letra do título do eixo y
    axis.title.y = element_text(size = 14, color = 'darkblue', face = "bold")
  )

# Plotando o objeto `graf_linhas2`
graf_linhas2
```

Figura 7: Gráfico de linhas da distribuição dos casos de dengue por dia e ano.



Fonte: SINAN

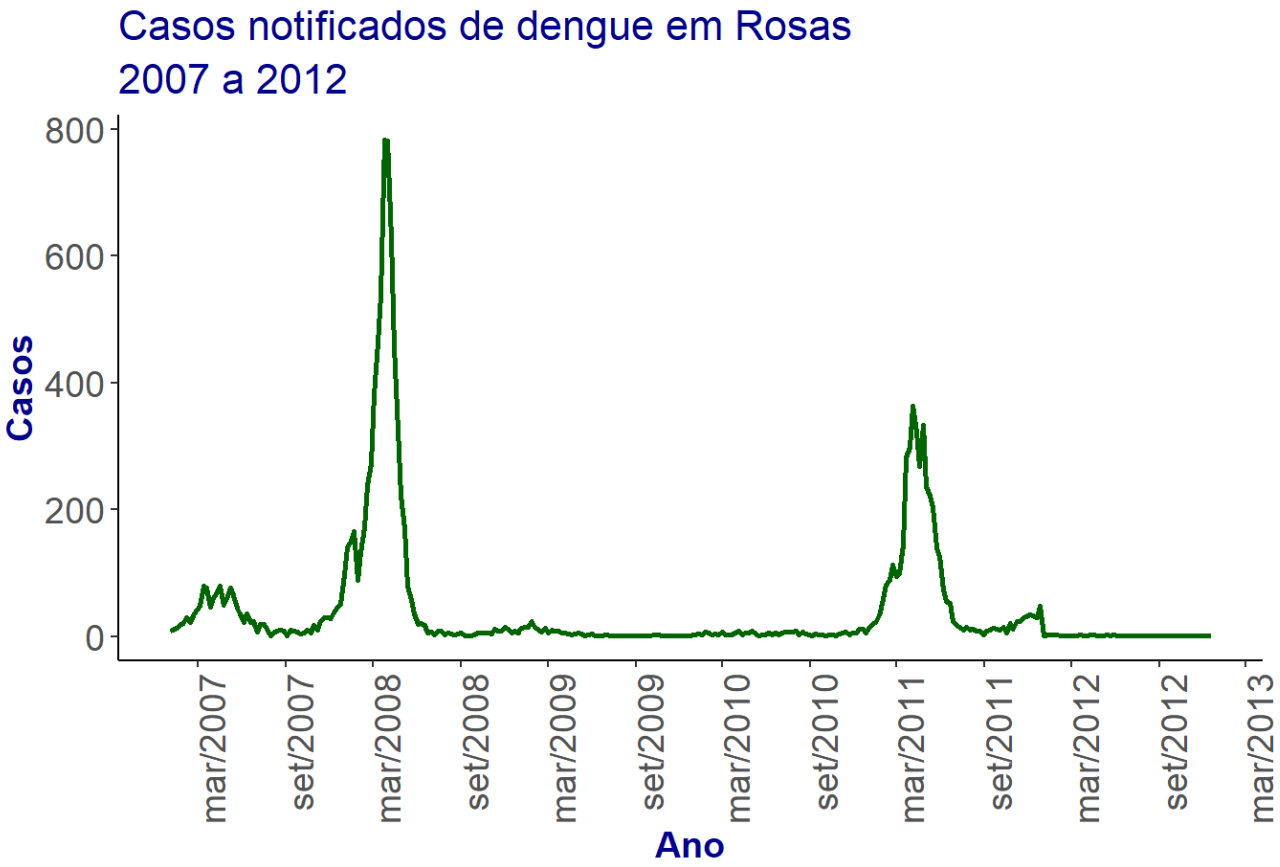
O gráfico {`graf_linhas2`} (Figura 7) possui o eixo x apresentando a escala de tempo em anos. Mas o objeto que utilizamos para a criação do gráfico está na escala de dias. Caso mudemos o eixo para dias, ficará difícil de diferenciar, pois são muitos dias e o eixo ficará poluído com muito texto. Vamos formatar a escala para apresentar as datas a cada 6 meses. Vamos utilizar uma função de escala de data chamada `scale_x_date()` para isso.

Com a função `scale_x_date()` vamos definir dois argumentos: um para a quebra das datas chamado `date_breaks` e outro para o rótulo das datas que aparecerá no eixo x, chamado `date_labels`. Note que no argumento `date_labels` estamos usando símbolos para repassar ao `R` o formato de datas que queremos. Na expressão `%b/%Y`, o `%b` significa mês abreviado (jan, fev, mar) e o `%Y` significa que o ano será no formato completo, com quatro dígitos. Para separar mês e ano estamos usando uma barra `/`.

Replique o código a seguir no *script* que você está construindo em seu `RStudio`:

```
# Utilizando o objeto gráfico {`graf_linhas2`} salvo anteriormente
graf_linhas2 +

  # Arrumando o eixo x, definindo os intervalos dos marcadores de datas
  # (`date_breaks`) serão mostrados que, no caso, será uma sequência de
  # 6 em 6 meses
  scale_x_date(date_breaks = '6 months', date_labels = '%b/%Y') +

  # Alterando o tema do gráfico
  theme(
    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 90,
      hjust = 1,
      size = 14,
      color = 'grey32'
    ),

    # Alterando o texto do eixo y
    axis.text.y = element_text(
      hjust = 1,
      size = 14,
      color = 'grey32'
    )
  )
```

Figura 8: Gráfico de linhas da distribuição dos casos de dengue por mês e ano



Fonte: SINAN

Na Figura 8 você pode notar a distribuição dos casos notificados de dengue de 2007 a 2012. Como a escala dos dados e do eixo foram alterados, temos mais detalhes da distribuição. Em 2008 o pico de notificação foi entre março e abril, meses com pico aproximado de 800 casos notificados. Em 2011, o cenário foi similar, reproduzindo o mesmo padrão, mas com aproximados 400 casos no mês de pico.

Séries Temporais

Existem dois tipos básicos de séries temporais: as séries regulares e os irregulares, também conhecidas como "séries calendário".

- Nas séries regulares existe a necessidade de se definir um período regular de tempo, por exemplo, todos os anos têm 12 meses (ou seja, uma frequência = 12) e isso não muda, é regular. Assim é possível analisarmos elementos que compõem uma série temporal como tendência, sazonalidade e ciclicidade, mesmo que nem todos os meses tenham o mesmo número de dias.

- Já as séries irregulares não possuem uma estrutura regular. Por exemplo, as semanas epidemiológicas que, em geral, possuem 52 semanas, mas, às vezes, o ano possui 53 semanas. Essa irregularidade impossibilita tratá-la com as mesmas ferramentas das séries regulares. Existem estratégias que permitem transformar uma série irregular, normalizando, no exemplo das semanas, o ano em 52 semanas.

O `R` possui muitos pacotes para construir, aprimorar e melhorar as séries temporais.

4.3 Gráfico de barras sobreposto

O gráfico de barras sobreposto ou empilhado é uma variante do gráfico de barras. Neste último temos, basicamente, duas variáveis sendo visualizadas e, no gráfico de barras sobrepostas, temos que cada barra é subdividida em outras barras que ficam empilhadas. Essas "sub-barras" representam uma espécie de cruzamento de uma variável com outra e que, juntas, definem o tamanho das barras. Vejamos na prática.

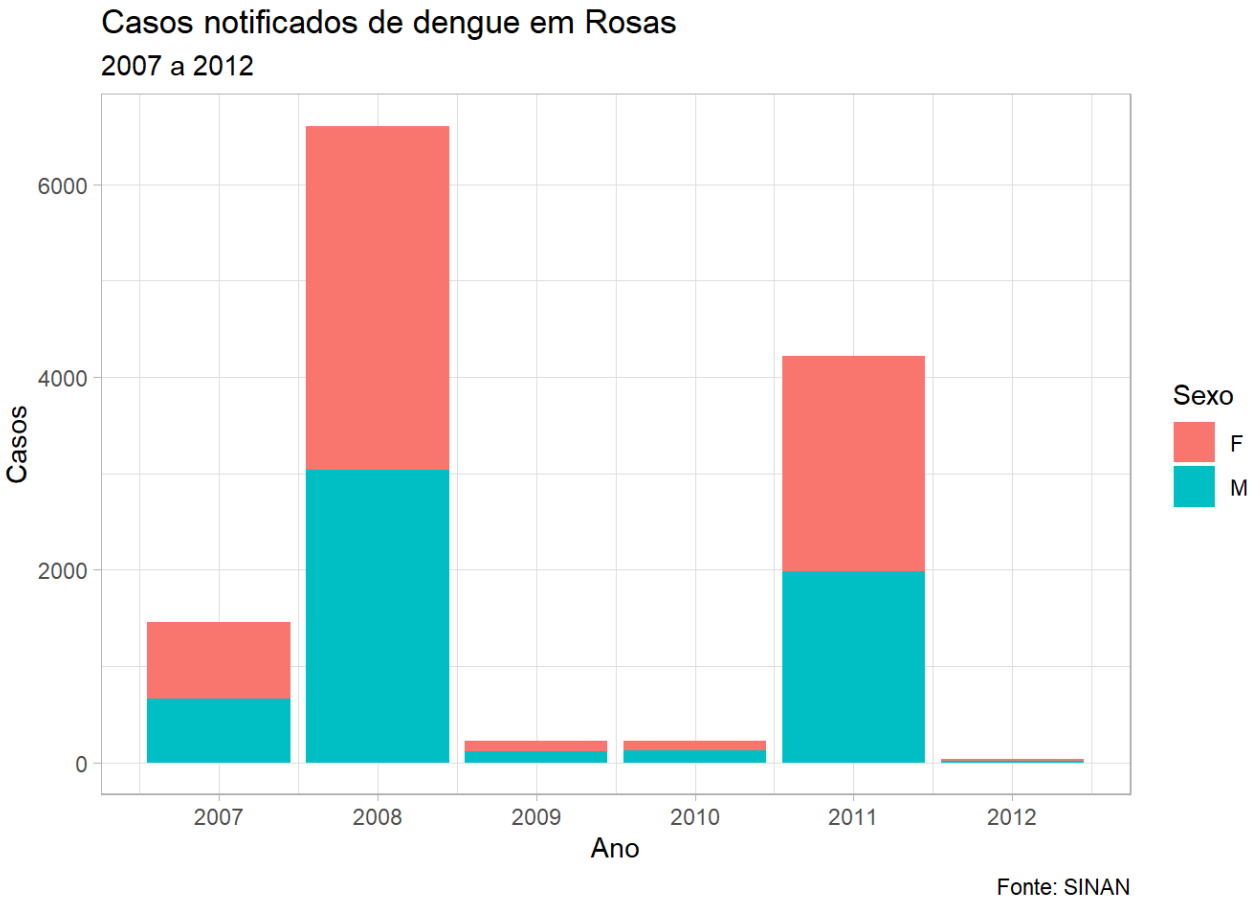
Vamos elaborar um gráfico de barras sobrepostas usando as variáveis sexo (`CS_SEXO`) e ano (`NU_ANO`), bem como a contagem de casos notificados de dengue por ano. A base de dados será a que preparamos no exemplo anterior. Aqui, vamos comparar o número de casos de dengue por ano e analisar a "contribuição" de cada categoria da variável sexo no total de casos em cada ano.

No `ggplot2`, esse gráfico utiliza a geometria `geom_col()`. Temos que a estrutura básica é a base de dados `dengue_ano`, a variável do eixo x será o ano (`NU_ANO`) e no eixo y o número de casos (`n_casos`). Agora, para que o `R` entenda que temos uma terceira variável, vamos preencher as barras com as categorias da variável sexo (`CS_SEXO`) usando o argumento `fill`. Mas cuidado! Esse argumento está presente apenas nas geometrias que possuem uma área a ser preenchida.

Para que as barras estejam sobrepostas, vamos utilizar o argumento `position` dentro da função `geom_col()`. Esse argumento recebe a definição de `stack`, que remete, em português, a algo empilhado. Veja o *script* a seguir e replique no seu computador:

```
# Criando o objeto gráfico {`graf_barras_sobrepostas`}
graf_barras_sobrepostas <- ggplot(data = dengue_ano) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento das colunas
  aes(x = NU_ANO, y = n_casos, fill = CS_SEXO) +

  # Adicionando a geometria de colunas e definindo o tipo empilhado
  geom_col(position = 'stack') +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico.
  # Para a legenda, estamos definindo o título como "Sexo"
  labs(
    title = 'Casos notificados de dengue em Rosas',
    subtitle =  '2007 a 2012',
    caption = 'Fonte: SINAN',
    x = "Ano",
    y = "Casos",
    fill = 'Sexo'
    ) +

  # Arrumando o eixo x, definindo quais os marcadores (`breaks`)
  # serão mostrados que, no caso, será uma sequência de 2007 a 2012,
  # referente aos anos.
  scale_x_continuous(breaks = 2007:2012) +

  # Definindo o tema base
  theme_light()

# Plotando o objeto `graf_barras_agrupadas`
graf_barras_sobrepostas
```

Figura 9: Gráfico de barras empilhadas da distribuição de casos de dengue por sexo.

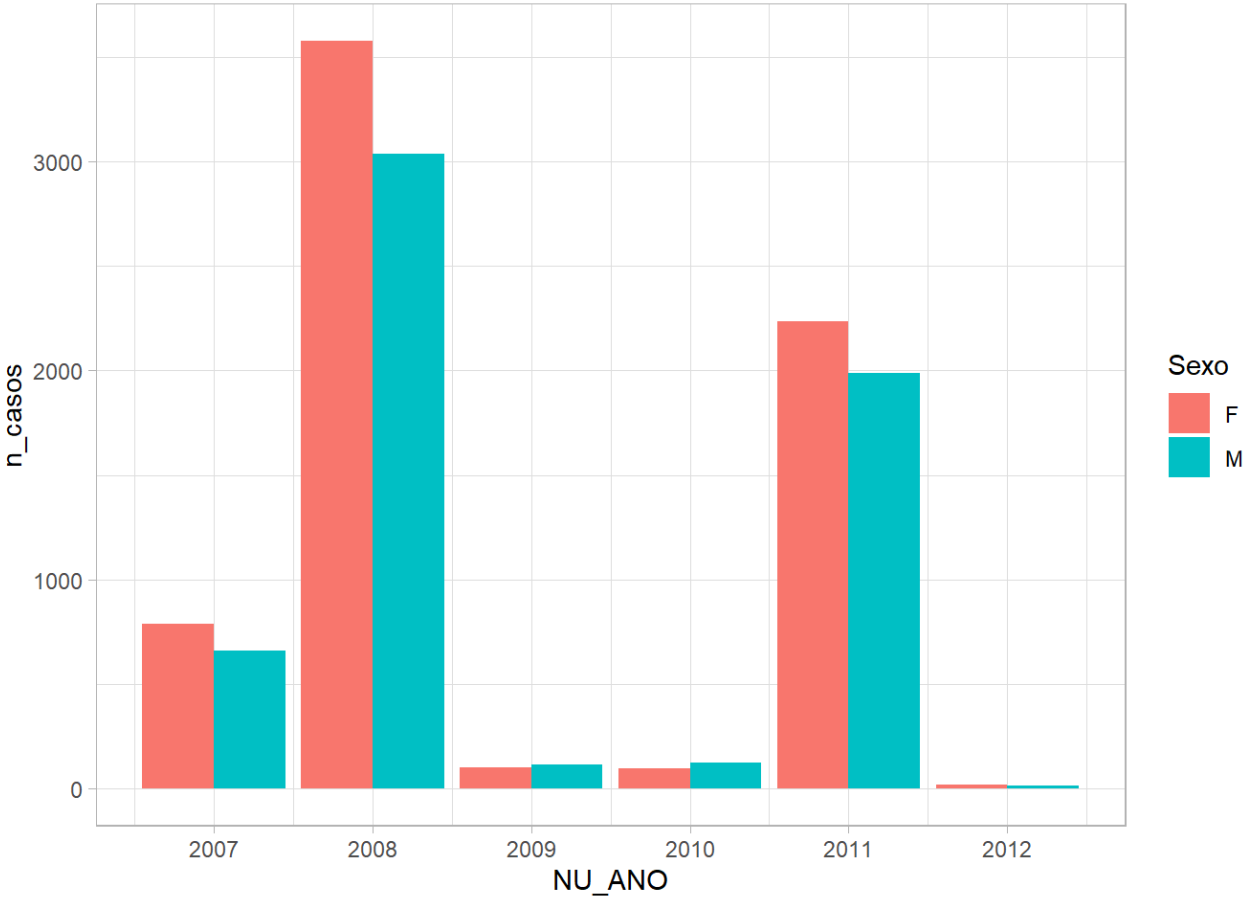


Note que para a plotar o gráfico de barras empilhado (Figura 9) usamos um novo argumento dentro da função `labs()`, chamado `fill`. Isso foi necessário para que o `ggplot2` coloque um nome na legenda. Note também que usamos uma escala para o eixo x (`scale_x_continuous`). Com essa escala podemos definir quebras (os `breaks`) para que o eixo x tenha os rótulos definidos.

Para alterar a posição das barras de empilhadas para agrupadas, basta especificar o argumento `position` para `dodge` dentro da função `geom_col`. Veja o *script* a seguir e replique no seu computador:

```
# Criando o objeto gráfico {`graf_barras_agrupadas`}
graf_barras_agrupadas <- ggplot(data = dengue_ano) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento das colunas
  aes(x = NU_ANO, y = n_casos, fill = CS_SEXO) +

  # Adicionando a geometria de colunas e definindo o tipo agrupado
  geom_col(position = 'dodge') +

  # Definindo o título da legenda das cores usadas para preenchimento
  # das colunas
  labs(fill = 'Sexo') +

  # Arrumando o eixo x, definindo quais os marcadores (`breaks`)
  # serão mostrados que, no caso, será uma sequência de 2007 a 2012,
  # referente aos anos.
  scale_x_continuous(breaks = 2007:2012) +

  # Definindo o tema base
  theme_light()

# Plotando o objeto `graf_barras_agrupadas`
graf_barras_agrupadas
```

Figura 10: Gráfico de barras agrupadas da distribuição de casos de dengue por sexo.



4.4 Gráfico de pontos

O gráfico de pontos é também conhecido como diagrama de dispersão, gráfico de dispersão ou *scatterplot*. Esse gráfico representa um conjunto de observações devidamente posicionadas nos eixos x e y a partir de duas **variáveis numéricas**. São úteis para visualizar correlações entre as variáveis. No `ggplot2`, esse gráfico utiliza a geometria `geom_point()`.

Usaremos a base de dados do VIGITEL 2015, que apresentamos no início. Para visualizar a diferença das observações segundo o sexo, vamos definir o argumento `colour` com esta variável.

Veja a seguir o *script* e replique-o em seu computador:

```
ggplot(data = vigitel) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para as cores dos pontos
  aes(x = peso, y = altura, colour = sexo) +

  # Adicionando a geometria de pontos
  geom_point()
```

```
#> Warning: Removed 5557 rows containing missing values (geom_point).
```

Figura 11: Gráfico de pontos da distribuição da variável peso com a variável altura, segundo sexo, dos entrevistados no VIGITEL 2015.

Ao gerar este gráfico, um aviso (*warning*) aparece, informando que 5.557 linhas foram removidas do gráfico por conter valores faltantes (`NA`). **Isso é devido a não ser possível posicionar o ponto sem que tenha algum valor registrado na variável** e no banco do VIGITEL 2015 algumas pessoas não responderam a algumas perguntas, sendo os dados classificados como faltantes (ou *missings*).

Atenção

Algumas vezes encontramos bancos de dados com valores "9", "99", ou "999" que representam valores ignorados. Esses valores devem ser excluídos das análises e representações gráficas para não gerar análises equivocadas. Desta forma é muito importante que você conheça os vieses e limitações da base de dados que está analisando, e tenha sempre seu dicionário de variáveis em mãos.

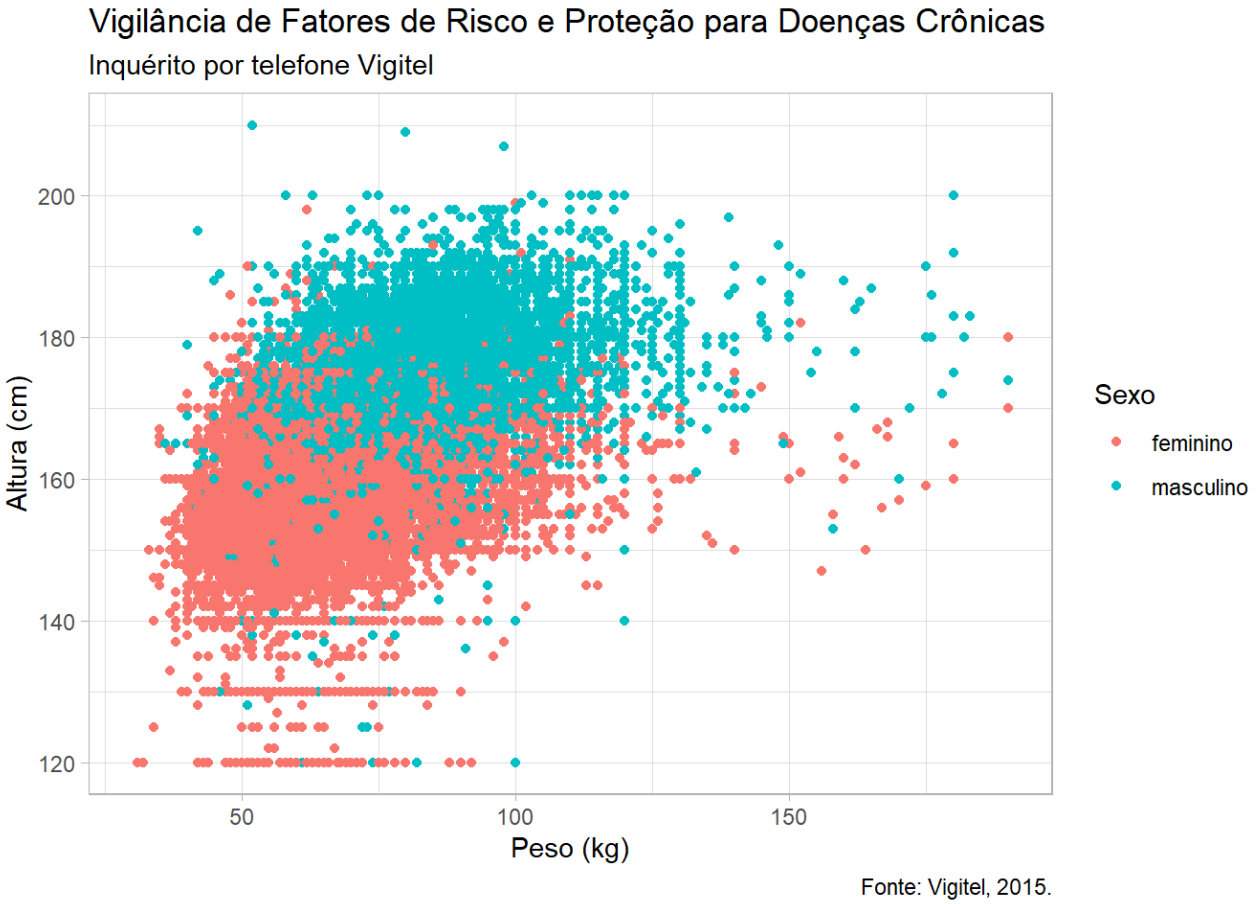
É recomendável que se faça a avaliação de estatísticas descritivas básicas antes de qualquer análise e construção gráfica. Para mais informações acesse o curso **"Análise de dados para a vigilância em saúde – curso básico"** disponível em https://www.abrasco.org.br/site/analise-de-dados-para-a-vigilancia-em-saude/

Dando continuidade, vamos inserir alguns elementos de comunicação e outros visuais que tornam o gráfico mais agradável. Acompanhe o código e replique em seu `RStudio`:

```
ggplot(data = vigitel) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento das barras do histograma
  aes(x = peso, y = altura, colour = sexo) +

  # Adicionando a geometria de pontos e definindo o tamanho do ponto
  geom_point(size = 1.5) +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico.
  # Para legenda, estamos definindo o título para "Sexo".
  labs(
    title = 'Vigilância de Fatores de Risco e Proteção para Doenças Crônicas',
    subtitle =  'Inquérito por telefone Vigitel',
    caption = 'Fonte: Vigitel, 2015.',
    x = "Peso (kg)",
    y = "Altura (cm)",
    colour = "Sexo"
  ) +

  # Definindo o tema base
  theme_light()
```

```
#> Warning: Removed 5557 rows containing missing values (geom_point).
```

Figura 12: Gráfico de pontos, com título e subtítulo, da distribuição da variável peso com a variável altura, segundo sexo, dos entrevistados no VIGITEL 2015.



Fonte: Vigitel, 2015.

No gráfico acima, cada ponto representa uma pessoa entrevistada na pesquisa do VIGITEL. A posição em x representa o peso em quilos da pessoa e a posição em y representa a altura em centímetros. A partir do gráfico percebe-se que, quando o peso aumenta, a altura também aumenta e homens têm altura e peso maior.

4.5 Histogramas

O histograma possui um visual muito parecido com o gráfico de barras, mas possui, de certa forma, uma interpretação diferente. Cada barra representa um intervalo de valores da variável analisada, geralmente no eixo x. No eixo y, é mostrada a frequência dos intervalos mostrados no eixo x e, dessa forma, a altura da barra significa sua maior ou menor frequência.

Os histogramas são muito úteis para visualizar a distribuição de variáveis numéricas. Entretanto, na área da vigilância em saúde, é também útil para visualizar a curva epidêmica de uma doença. Para exemplificar seu uso, vamos utilizar a base de casos notificados de dengue utilizados em exemplos anteriores e oriundos da base do estado fictício de Rosas.

Vamos filtrar os dados para o ano epidemiológico de 2008 usando a função `filter()` do pacote `dplyr` e salvar em um objeto chamado {`dengue_2008`} e depois utilizá-lo no nosso gráfico. Observe o código a seguir e replique-o em seu `RStudio`:

```
# Criando o objeto dataframe {`dengue_2008`}
dengue_2008 <- dengue |>

  # Filtrando os registros com ano epidemiológico igual a 2008
  filter(ano_epi == 2008)
```

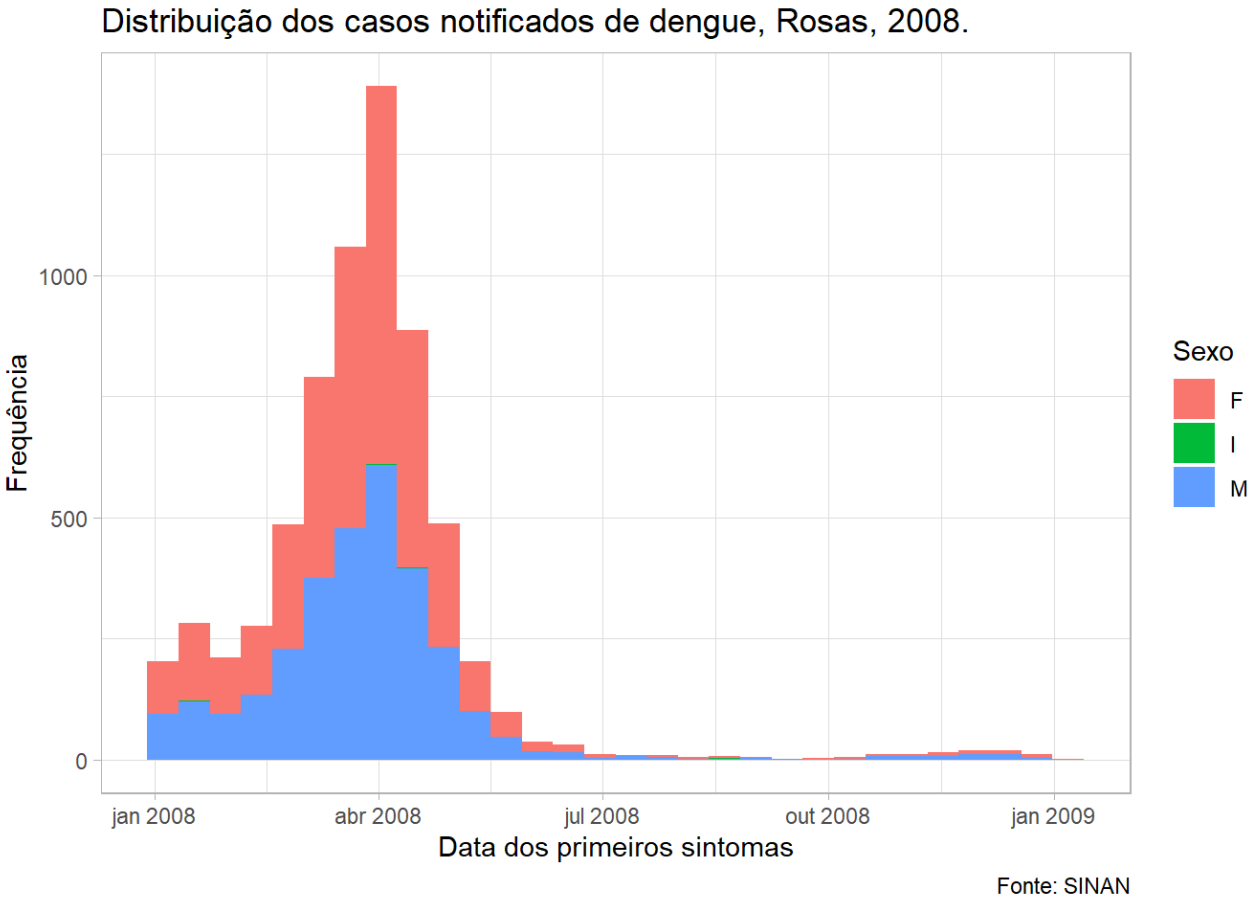
Agora observe no código a seguir que no eixo x definiremos a variável de data dos primeiros sintomas (`DT_SIN_PRI`) e "preencheremos" o gráfico com as categorias da variável sexo (`CS_SEXO`), utilizando o argumento `fill`. A geometria para histograma é representada pela função `geom_histogram()`.

Acompanhe o *script* a seguir e replique-o em seu `RStudio` para produzir o gráfico apresentado na Figura 13.

```
# Utilizando a tabela {`dengue_2008`} criada anteriormente
ggplot(data = dengue_2008) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento das barras do histograma
  aes(x = DT_SIN_PRI, fill = CS_SEXO) +

  # Adicionado geometria de colunas de histograma
  geom_histogram() +

  # Definindo os títulos dos eixos x e y, rodapé, subtítulo e título do gráfico.
  # Para legenda, estamos definindo o título para "Sexo".
  labs(
    title = "Distribuição dos casos notificados de dengue, Rosas, 2008.",
    caption = 'Fonte: SINAN',
    x = "Data dos primeiros sintomas",
    y = "Frequência",
    fill = 'Sexo'
  ) +

  # Definindo o tema base
  theme_light()
```

```
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Figura 13: Histograma da distribuição dos casos de dengue em Rosas segundo sexo em 2018.



Fonte: SINAN

O histograma acima, Figura 13, apresenta a distribuição da frequência dos casos de dengue no decorrer de 2008. Percebemos que a concentração dos casos notificados foi no primeiro semestre, com pico em abril de 2008. Note que a Figura 13 difere do gráfico de linhas apresentado na Figura 7. No histograma as barras representam a frequência de casos em intervalos de classes (chamados pelo pacote `ggplot2` de *bins*).

5. Outros gráficos de interesse da vigilância em saúde

5.1 Gráficos temporais para séries irregulares

As séries temporais irregulares costumam apresentar frequência na unidade de tempo diferentes durante o período avaliado, ou seja, é irregular conforme o tempo é modificado. Por exemplo, alguns anos têm 52 semanas e outros 53. Isso é uma irregularidade. Para trabalhar com séries com esta característica, vamos usar o pacote `xts`.

Com este pacote é possível organizar uma série calendário (irregular) baseada na data dos primeiros sintomas, por exemplo, e visualizá-la em um gráfico. Para isso, o processo é similar ao que estamos utilizando: preparamos os dados, atribuímos a um objeto e plotamos em um gráfico.

Então vamos lá. Continuaremos utilizando a base de dados de dengue do estado fictício de Rosas. Acompanhe os passos para preparar os dados:

Primeiro, vamos definir um período utilizando a função `filter()`.

Em seguida, vamos fazer uma contagem de casos notificados de dengue por ano e semana epidemiológica utilizando a função `count()`. Note que as variáveis `ano_epi` e `sem_epi` foram criadas logo após a importação do {`NINDINET.dbf`} quando filtramos os casos de dengue.

Agora, vamos extrair o dia de início das semanas utilizando a função `get_date()` do pacote `aweek`. Vamos definir o argumento `start` como `7`, que significa que a semana inicia no domingo, como padrão das semanas epidemiológicas.

Vamos organizar a base considerando a ordem do menor ano e menor semana para maior ano e maior semana.Vamos atribuir o resultado desse procedimento ao objeto `dengue_semana`.

Veja o *script* completo a seguir e replique no seu computador:

```
# Criando o objeto {`dengue_semana`}
dengue_semana <- dengue |>

  # Filtrando os registros dentre intervalo de datas
  filter(DT_SIN_PRI >= '2007-01-01' & DT_SIN_PRI <= '2012-12-31') |>

  # Contando a frequência de notificações por ano epidemiológico esemana epidemiológica
  count(ano_epi, sem_epi, name = 'n_casos') |>

  # Utilizando a função `mutate()` para criar a coluna de data de início
  # da semana epidemiológica
  mutate(data = get_date(sem_epi, ano_epi, start = 7)) |>

  # Ordenando pelo ano e semana epidemiológica
  arrange(ano_epi, sem_epi)
```

Atenção

Lembre-se que você deverá sempre verificar se todos os pacotes necessários foram instalados e carregados no `R`, caso encontre algum aviso (warning) ou erro (error) revise seu código cuidadosamente e rode-o novamente.

Pronto. Agora, vamos visualizar as primeiras linhas da Tabela 4, armazenada no objeto {`dengue_semana`}, que acabamos de criar utilizando a função `head()` alinhada à função `kable()` do pacote `knitr`. Essa função cria uma tabela para ser visualizada em arquivos `HTML`. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
kable(head(dengue_semana, 10))
```

Tabela 4: Organização dos dados para criação de um gráfico de linhas (Figura 14).

ano_epi	sem_epi	n_casos	data
2007	1	9	2006-12-31
2007	2	10	2007-01-07
2007	3	13	2007-01-14
2007	4	17	2007-01-21
2007	5	21	2007-01-28
2007	6	29	2007-02-04
2007	7	22	2007-02-11
2007	8	32	2007-02-18
2007	9	41	2007-02-25
2007	10	47	2007-03-04

Pacotes úteis para lidar com datas

Já vimos que o pacote `lubridate` tem funções bem interessantes para a manipulação de datas, entre elas `epiweek()` e `epiyear()`, que retornam a semana e o ano epidemiológico na definição do *Centers for Disease Control (CDC)*, ou seja, a mesma usada no Brasil.

O pacote *aweek* também tem funções para lidar com dados por semana. No exemplo a seguir, estamos calculando a data de início da semana epidemiológica usando a função `aweek()`. No entanto, você tem de estar atento ao valor padrão dessa função, que não é semana epidemiológica. Por isso, é necessário definir o argumento `start` como 7.

```
# exemplo de como extrair a data de início da semana 1, no ano 2020
get_date(1, 2020, start = 7)
```

```
#> [1] "2019-12-29"
```

O resultado deve ser "2019-12-29" que indica que a primeira semana de 2020 se iniciou em 29/12/2020.

Com os dados prontos, iremos agora utilizar a função `xts` do pacote `xts`. Para isso, será necessário definir dois argumentos:

- `x`: argumento que deverá indicar os dados a serem plotados. Ou seja, colocaremos no `x` a coluna da base de dados {`dengue_foz`} com os valores da incidência de casos (`n_casos`).
- `order.by`: argumento que deverá indicar a data correspondente aos dados indicados no argumento `x`. Aqui, utilizaremos a coluna com os valores da data de início das semanas epidemiológicas (`data`).

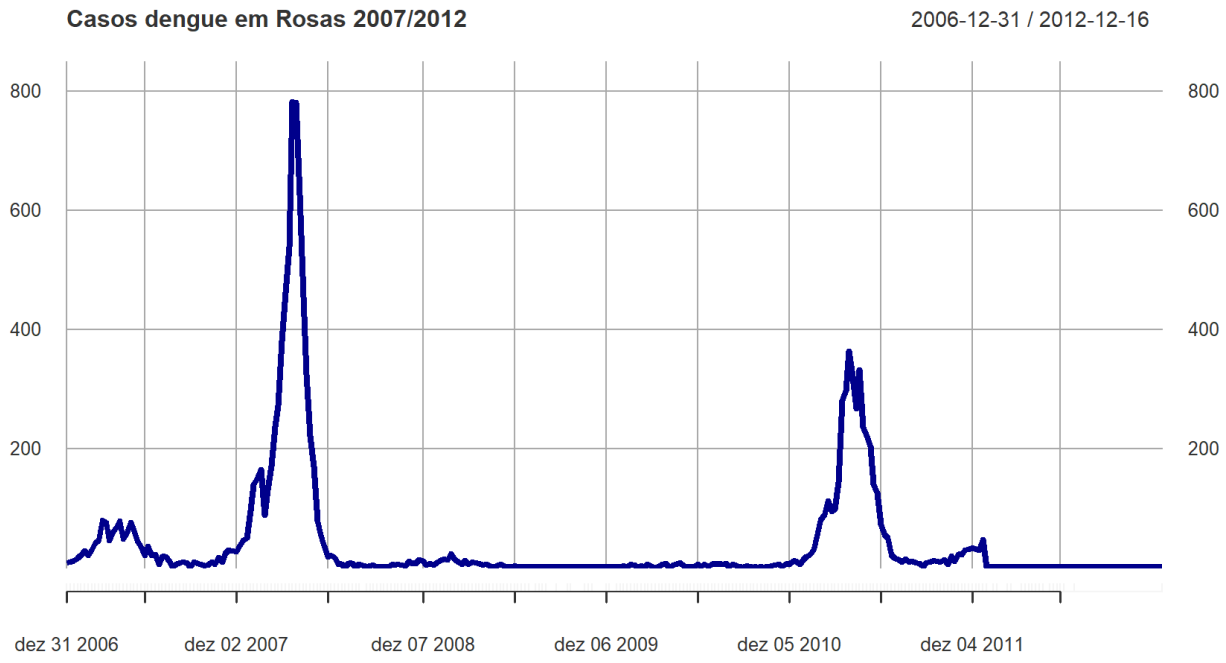
Perceba que utilizaremos o operador `$` (cifrão) para indicar a seleção das variáveis dentro da base de dados. Observe o *script* a seguir e replique-o em seu `RStudio`:

```
# criando o no objeto {`dengue_irregular`}
dengue_irregular <- xts(

  # selecionando a variável com os dados dos casos
  x = dengue_semana$n_casos,

  # selecionando a variável que contém as datas correspondentes
  order.by = dengue_semana$data)
```

Também poderemos visualizar a série como um gráfico de linhas na função `plot()`, apresentado na Figura 14:

```
par(mfrow=c(1,1))
plot(

  # indicando a série temporal
  dengue_irregular,

  # Definindo o título
  main = 'Casos dengue em Rosas 2007/2012',

  # Definindo a cor das linhas
  col = 'darkblue',

  # Definindo a espessura das linhas
  lwd = 4,

  # Definindo o intervalo do eixo y
  ylim = c(0, 850)
)
```

Figura 14: Gráfico de linhas da distribuição dos casos de dengue do Estado de Rosa, entre 2007 e 2012.



5.2 Gráficos temporais de calor (*heatmaps*)

O gráfico de calor, também conhecido pelo seu nome em inglês *heatmap* descreve valores para uma variável dividido entre duas outras variáveis em uma grade de quadrados que, conforme o valor da variável principal, a cor fica mais intensa.

Mais uma vez vamos usar os casos de dengue como exemplo:

1. em primeiro lugar vamos filtrar os casos de 2007 a 2011;
2. em seguida vamos contar o número de casos por dia de primeiros sintomas e atribuir a um objeto *dengue_heat*.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando o objeto {`dengue_heat`}
dengue_heat <- dengue |>

  # Filtrando os registros dentre intervalos de datas
  filter(DT_SIN_PRI >= "2007-01-01" & DT_SIN_PRI <= "2011-12-31") |>

  # Contando a frequência de notificações por data dos primeiros sintomas
  count(DT_SIN_PRI, name = 'casos') |>

  # Utilizando a função `mutate()` para criar a coluna de ano de início
  # dos primeiros sintomas
  mutate(Ano = year(DT_SIN_PRI))
```

Agora, vamos visualizar as primeiras linhas, conforme Tabela 5, com o *script* a seguir. Repita em seu `RStudio`:

```
kable(head(dengue_heat, 10))
```

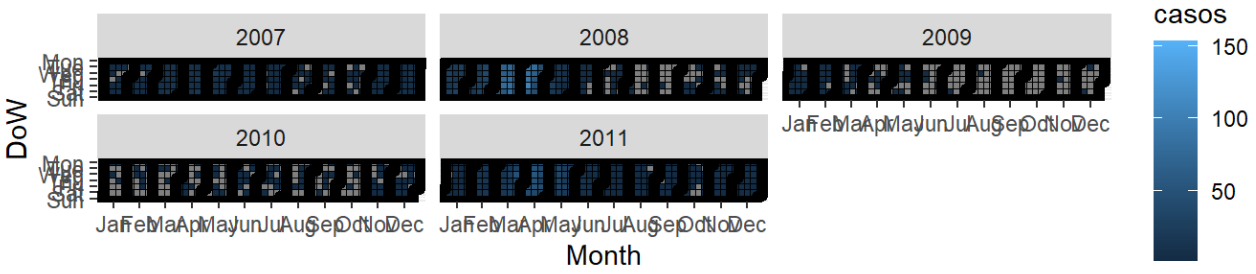
Tabela 5: Tabela {`dengue_semana`} criada para armazenar os dados que serão utilizados no gráfico temporal (Figura 15).

DT_SIN_PRI	casos	Ano
2007-01-01	2	2007
2007-01-02	1	2007
2007-01-03	1	2007
2007-01-04	3	2007
2007-01-06	2	2007
2007-01-07	1	2007
2007-01-08	1	2007
2007-01-09	2	2007
2007-01-10	2	2007
2007-01-12	3	2007

O pacote **ggTimeSeries** adiciona uma série de funções, entre elas a que vamos utilizar para fazer mapas de calor tipo calendário. Entretanto, as funções básicas do pacote não oferecem um visual que seja fácil de interpretar. Veja a Figura 15 a seguir, observe o padrão no código e replique-o no seu computador:

```
ggplot_calendar_heatmap(dengue_heat, 'DT_SIN_PRI','casos')
```

Figura 15: Gráfico temporal de calor da distribuição dos casos de dengue do Estado de Rosa, entre 2007 e 2011.



Atenção

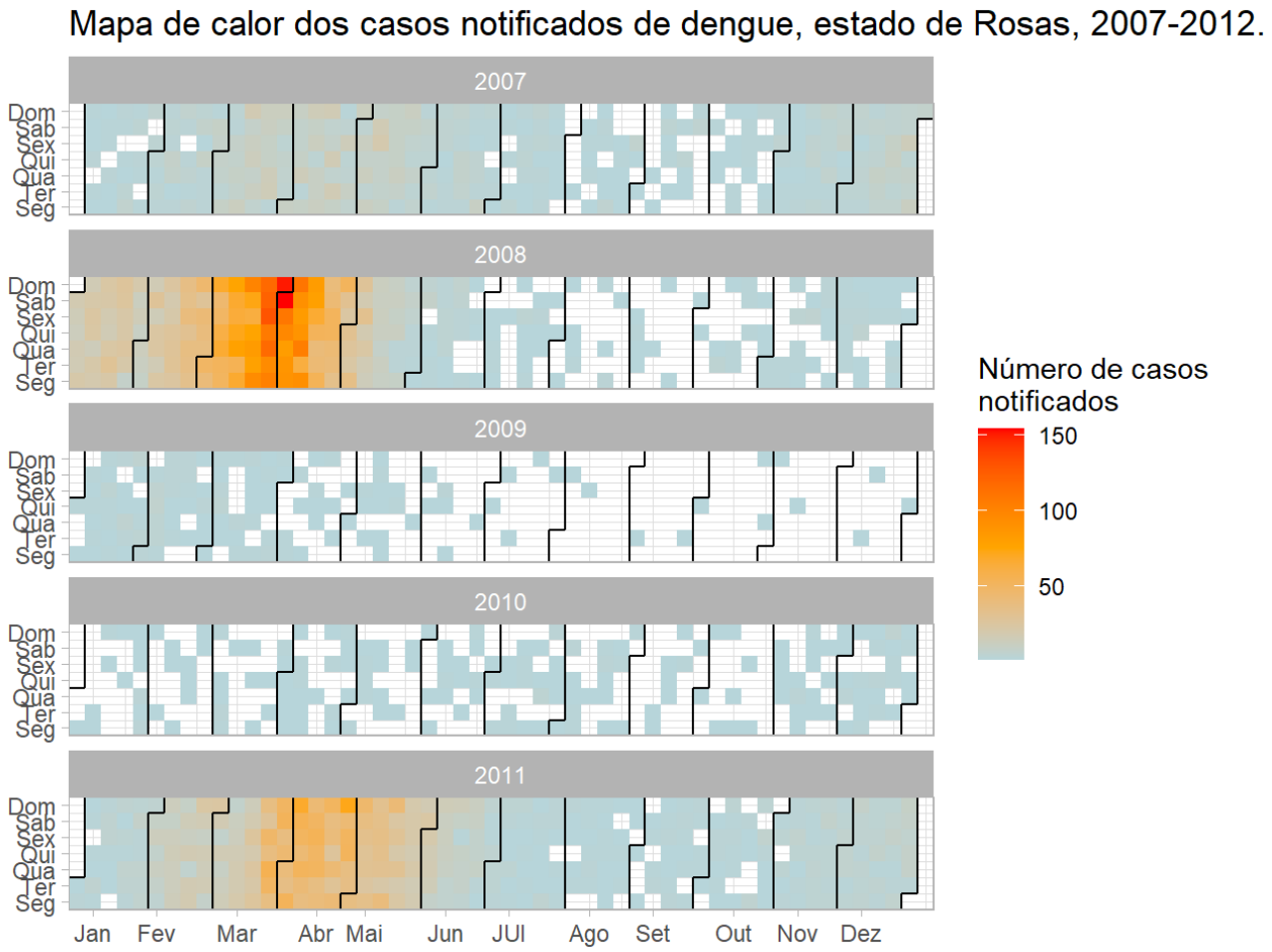
Lembre-se que você deverá sempre verificar se todos os pacotes necessários foram instalados e carregados no `R`, caso encontre algum aviso (warning) ou erro (error) revise seu código cuidadosamente e rode-o novamente.

Fica bem difícil de visualizar os dias, não é mesmo? Vários comandos adicionais podem ser usados para estabelecer os rótulos em meses e dias da semana e também para atribuir uma escala diferente de cores.

Vamos, então, montar um gráfico (Figura 16) com funções do pacote `ggplot2` para deixar mais profissional. Observe o *script* a seguir e replique-o em seu `RStudio`:

```
ggplot(data = dengue_heat) +

    # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento da série
  aes(date = DT_SIN_PRI, fill = casos) +

  # Adicionando geometria de `heatmap`
  stat_calendar_heatmap() +

  # Definindo a rampa de cores de preenchimento
  scale_fill_gradient2(
    low = 'lightblue',
    high = 'red',
    mid = 'orange',
    midpoint = 75
  ) +

  # Arrumando o eixo y, definindo quais os marcadores (`breaks`)
  # serão mostrados que, no caso, será uma sequência de 1 a 7, referente aos
  # dias da semana. O argumento `expand` ajuda nesse processo.
  scale_y_continuous(
    breaks = c(1:7),
    labels = c("Seg", "Ter", "Qua", "Qui", "Sex", "Sab", "Dom"),
    expand = c(0, 0)
  ) +

  # Arrumando o eixo x, definindo quais os marcadores (`breaks`) e
  # os rótulos do gráfico `labels`.
  scale_x_continuous(
    breaks = c(2, 6, 11, 16, 19, 24, 28, 33, 37, 42, 46, 50),
    labels = c(
      'Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun',
      'JUl', 'Ago', 'Set', 'Out', 'Nov', 'Dez'
    ),
    expand = c(0, 0)
  ) +

  # Definindo a estratificação por ano e o visual com uma coluna
  facet_wrap( ~ Ano, ncol = 1) +

  # Definindo o tema base
  theme_light() +

  # Definindo título do gráfico
  labs(
    title = "Mapa de calor dos casos notificados de dengue, estado de Rosas, 2007-2012.",
    fill = "Número de casos \nnotificados"
  )
```

Figura 16: Gráfico temporal de calor, utilizando `ggplot2`, da distribuição dos casos de dengue do Estado de Rosa, entre 2007 e 2011.



Uau! Perceba no gráfico acima que o conjunto extra de comandos torna o gráfico em algo muito diferente e bem mais fácil de interpretar e visualizar.

5.3 Gráficos temporais com média móvel

Na covid-19 muitos profissionais de vigilância utilizaram gráficos temporais com a inclusão da média móvel de casos e óbitos de covid-19. A média móvel é sempre utilizada quando precisamos incluir no gráfico o cálculo da média aritmética das observações mais recentes, ou seja, quando precisamos saber a tendência média ao longo do tempo. Vamos ver como isso se dá na prática?

Primeiro, organizaremos os dados para o gráfico de linhas padrão, demonstrando o comportamento da doença que estamos analisando. Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
dengue_semana <- dengue |>

  # Filtrando os registros com data de primeiros sintomas maior ou igual
  # a data de primeiro de janeiro de 2007.
  filter(DT_SIN_PRI >= '2007-01-01') |>

  # Utilizando a função `mutate()`pra criar novas colunas
  mutate (

    # Criando uma nova coluna chamada 'SEM_EPI` referente à semana
    # epidemiológica dos primeiros sintomas
    SEM_EPI = epiweek(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'ANO_EPI` referente ao ano
    # epidemiológicos dos primeiros sintomas
    ANO_EPI = epiyear(DT_SIN_PRI),

    # Criando uma nova coluna chamada 'DT_INI_SEM` referente à
    # data de início da semana epidemiológica
    DT_INI_SEM = get_date(SEM_EPI, ANO_EPI)

  ) |>

  # Agrupando as notificações pelo ano epidemiológico, semana
  # epidemiológica e data de início da semana epidemiológica
  group_by(ANO_EPI, SEM_EPI, DT_INI_SEM) |>

  # Contando a frequência de notificações
  count(name = 'casos')
```

Agora vamos preparar os dados para a média móvel. Observe o *script* a seguir e replique-o em seu `RStudio`:

```
dengue_sem2 <- dengue_semana |>

  # Filtrando os registros com data  maior ou igual
  # a data de primeiro de janeiro de 2007.
  filter(DT_INI_SEM >= '2010-01-01') |>

  # Desagrupando os dados
  ungroup() |>

  # Substituindo os valores faltantes por zero
  replace_na(list(casos = 0)) |>

  # Utilizando a função `mutate()` para criar colunas
  mutate(

    # Criando uma nova coluna chamada 'mm14` referente à média
    # móvel dos casos notificados de dengue
    mm14 = zoo::rollmean(casos, 14, fill = TRUE, align = 'center'),

    # Criando uma nova coluna referente ao rótulo que será
    # usado no gráfico
    rotulo = sprintf('%02d-%s', SEM_EPI, str_sub(ANO_EPI, 3, 4))


  )
```

Agora, vamos visualizar as primeiras linhas, conforme Tabela 6, com o *script* a seguir. Repita em seu `RStudio`:

```
kable(head(dengue_sem2, 10))
```

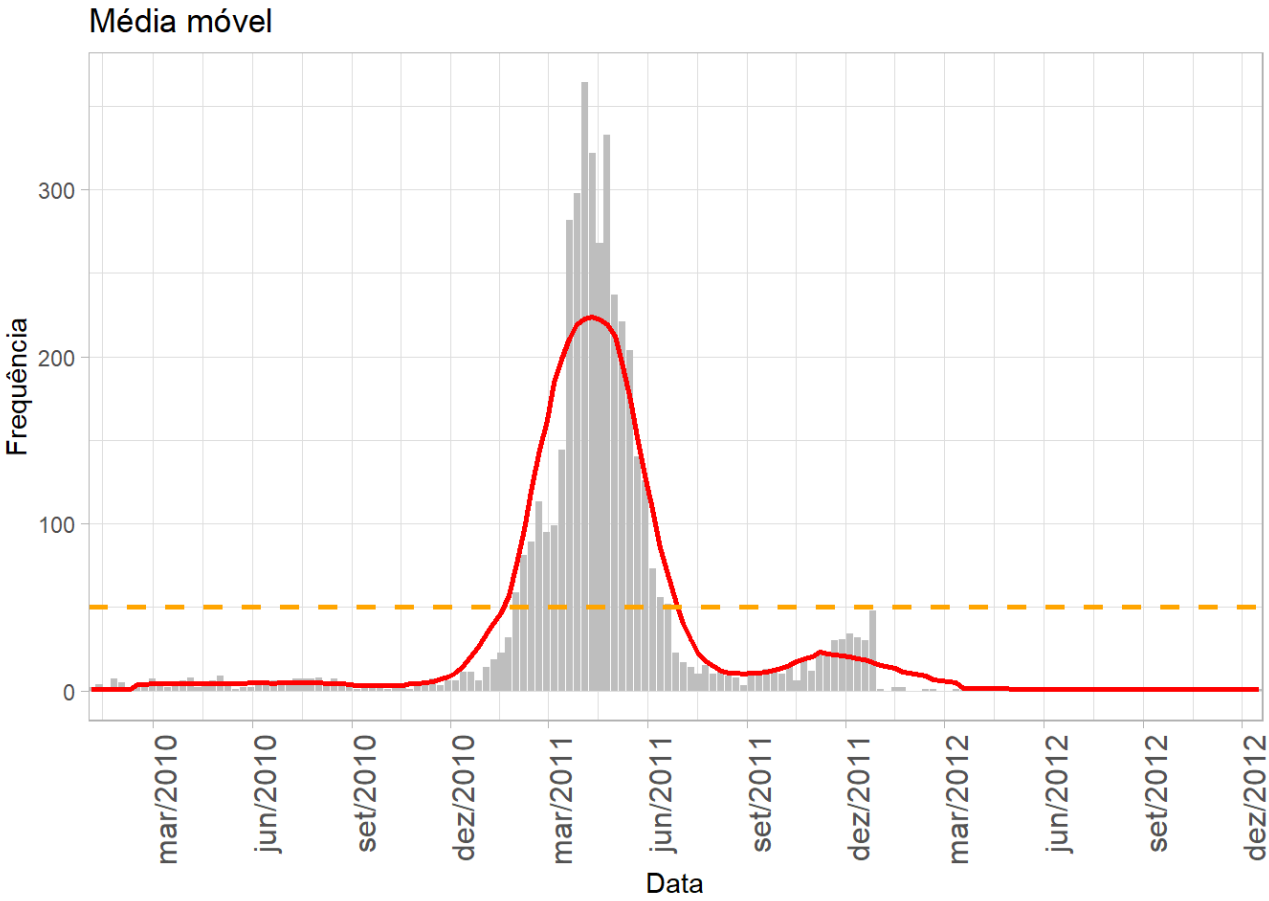
Tabela 6: Tabela {`dengue_sem2`} com dados organizados para gerar um gráfico de média móvel (Figura 17).

ANO_EPI	SEM_EPI	DT_INI_SEM	casos	mm14	rotulo
2010	1	2010-01-04	2	1.000000	01-10
2010	2	2010-01-11	4	1.000000	02-10
2010	3	2010-01-18	2	1.000000	03-10
2010	4	2010-01-25	7	1.000000	04-10
2010	5	2010-02-01	5	1.000000	05-10
2010	6	2010-02-08	2	1.000000	06-10
2010	7	2010-02-15	4	4.071429	07-10
2010	8	2010-02-22	2	4.071429	08-10
2010	9	2010-03-01	7	4.142857	09-10
2010	10	2010-03-08	3	4.428571	10-10

Pronto. Agora, faremos um gráfico de barras em cinza (Figura 17), com gráfico de linha com a média móvel em vermelho e, por fim, uma linha pontilhada laranja mostrando 50 casos. Observe o *script* a seguir e replique-o em seu `RStudio`:

```
# Criando o objeto gráfico {`gsem1`}
gsem1 <- ggplot(data = dengue_sem2) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  aes(x = DT_INI_SEM, y = casos) +

  # Adicionado a geometria de barras e
  # definindo a variável usada para o preenchimento
  geom_col(fill = "gray") +

  # Adicionando a geometria de linhas e definindo a cor e
  # a espessura
  geom_line(aes(y = mm14), color = 'red', size = 1) +

  # Adicionado uma geometria de linha cruzando com o
  # eixo y no valor 50 e definindo a linha como
  # tracejada (`dashed`), cor laranja e espessura
  geom_hline(
    yintercept = 50,
    linetype = 'dashed',
    color = 'orange',
    size = 1
  ) +

  # Arrumando a data do eixo x, definindo os intervalos
  # (de 3 em 3 meses) e o rótulo
  scale_x_date(date_breaks = '3 months',
               date_labels = '%b/%Y',
               expand = c(0, 0))  +

  # Definindo o tema base
  theme_light() +

  # Alterando o tema do gráfico
  theme(

    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 90,
      hjust = 1,
      size = 12,
      color = 'grey32'
    )) +

  # Definindo o título d gráfico e títulos dos eixos x e y
  labs(
    title = "Média móvel",
    x = "Data",
    y = "Frequência"
  )

gsem1
```

Figura 17: Gráfico da média móvel e marcação dos 50 casos de dengue em Rosas, entre 2010 e 2012.



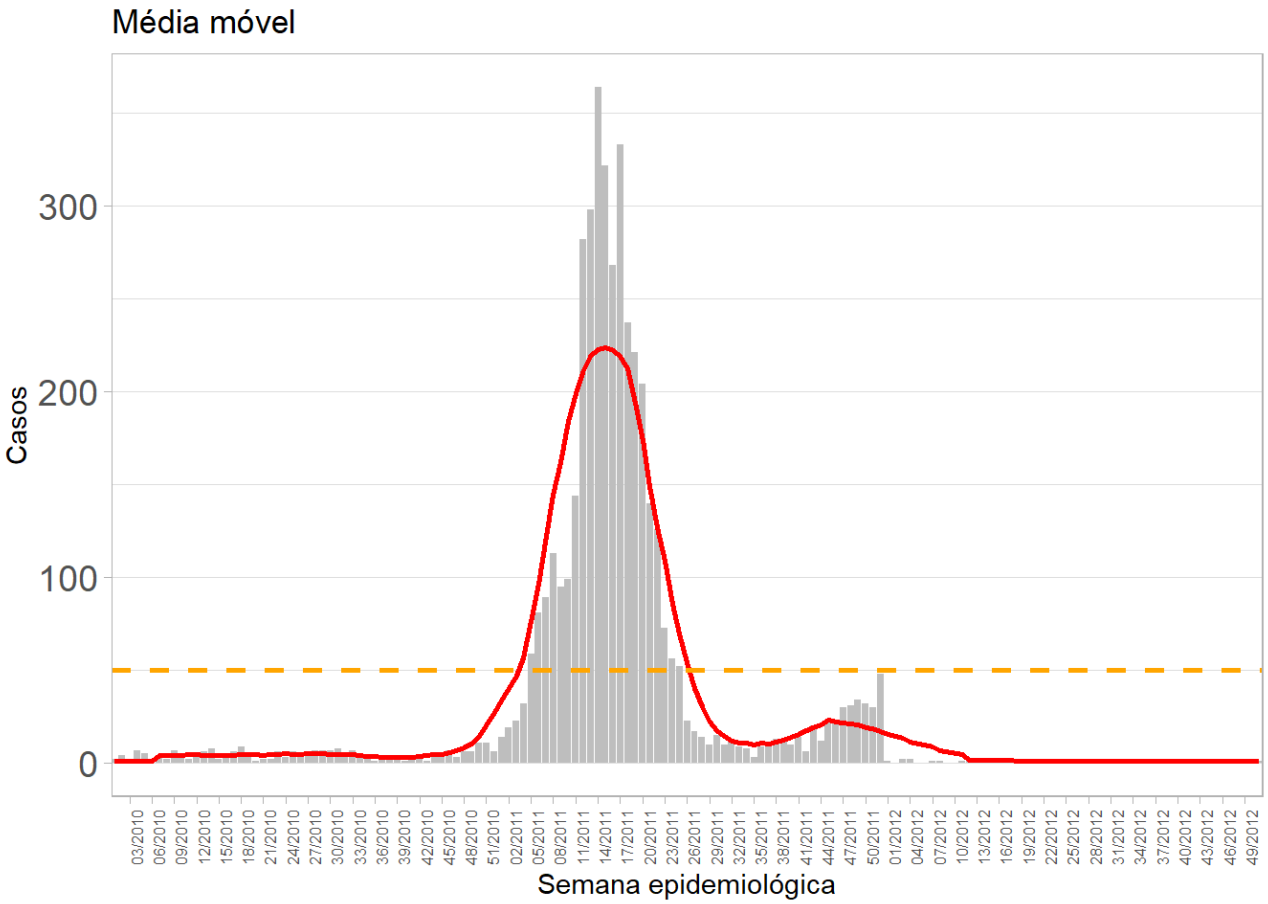
Com o objeto `gsem1` será possível modificar alguns aspectos como a escala do eixo X para representar as datas por semanas epidemiológicas e alterar o tamanho das letras dos respectivos eixos. Observe essas modificações plotadas na Figura 18 a seguir.

Acompanhe o *script* e replique-o em seu computador:

```
gsem1 +

  # Definindo os títulos dos eixos x e y
  labs(x = "Semana epidemiológica", y = "Casos") +

  # Arrumando o eixo x, definindo os marcadores (`breaks`)
  # e os rótulos
    scale_x_date(date_breaks = '3 weeks',
                 date_labels = '%W/%Y',
                 expand = c(0, 0))  +
  # Alterando o tema do gráfico
  theme(

    # Alterando o texto do eixo x
    axis.text.x = element_text(
      angle = 90,
      hjust = 1,
      size = 6,
      color = 'grey32'
    ),

    # Alterando o texto do eixo y
    axis.text.y = element_text(
      hjust = 1,
      size = 14,
      color = 'grey32'
    ),

    # Definindo que as grades do gráfico
    # nulas
    panel.grid.major.x =  element_blank() ,
    panel.grid.minor.x =  element_blank()
  )
```

```
#> Scale for 'x' is already present. Adding another scale for 'x', which will
#> replace the existing scale.
```

Figura 18: Gráfico da média móvel e marcação dos 50 casos de dengue em Rosas, entre 2010 e 2012, por semana epidemiológica.



Ficou bacana, não é mesmo? Experimente aplicar o cálculo da média móvel para outras doenças ou agravos analisados em seu dia a dia.

5.4 Pirâmides etárias

Pirâmides etárias são amplamente utilizadas na vigilância em saúde. E com o apoio da linguagem `R` será possível produzi-las sempre que desejar. Aqui iremos construir pirâmide utilizando sexo e idade das notificações de dengue do Estado de Rosas. O primeiro passo será remover os poucos casos em que a variável `sexo` é igual a "I" (ignorados) e depois criar as faixas etárias com que vamos trabalhar na pirâmide para isso usaremos a função `cut()`, que permite categorizar uma variável contínua (`NU_IDE_N`, no presente caso), em categorias. Para isso informamos os pontos de corte (breaks) e, de forma opcional, os *labels*.

Em seguida vamos contar os casos por sexo e faixa etária criando uma variável chamada **n** e, por fim, vamos multiplicar **n** por -1 quando sexo for igual a "F". Essa configuração é necessária para espelhar o eixo de frequência do lado oposto ao padrão e, assim, criarmos as barras para o sexo masculino e feminino (um à esquerda e outro à direita, como uma pirâmide etária). Acompanhe o código a seguir e replique-o no seu `RStudio`:

```
# Criando o objeto {`piramide`}
piramide <- dengue |>

  # Filtrando os registros com sexo diferente de "I" (ignorado)
  filter(CS_SEXO != 'I') |>

  # Utilizando a função `mutate()` para criar colunas
  mutate(
    # Criando uma coluna de idade conforme a codificação da variável NU_IDADE_N
   idade_anos = if_else(str_sub(NU_IDADE_N, 1, 1) == "4", as.numeric(str_sub(NU_IDADE_N, 2, 4)), 0),

    # Criando uma coluna de faixa etária a partir da variável idade dos casos notificados
    # utilizando a função `cut()`
    fx_etaria = cut(

      # Definindo qual variável será classificada em faixas
      x = idade_anos,

      # Definindo os pontos de corte das classes
      breaks = c(-Inf, 10, 20, 30, 40, 50, 60, Inf),

      # Definindo o tipo do ponto de corte
      right = FALSE,
      include.lowest = FALSE,

      # Definindo os rótulos das classes
      labels = c("0-9", "10-19", "20-29", "30-39",  "40-49",  "50-59", "60 anos e+")
    )
  ) |>

  # Contando a frequência da faixa etária e sexo
  count(fx_etaria, CS_SEXO) |>


  # Utilizando a função `mutate()` para criar a coluna que
  # vai receber as configurações da pirâmide etária
  mutate(n = ifelse (CS_SEXO == 'F', n * -1, n))
```

Agora, vamos visualizar as primeiras linhas, conforme Tabela 7, com o *script* a seguir. Repita em seu `RStudio`:

```
kable(head(piramide, 10))
```

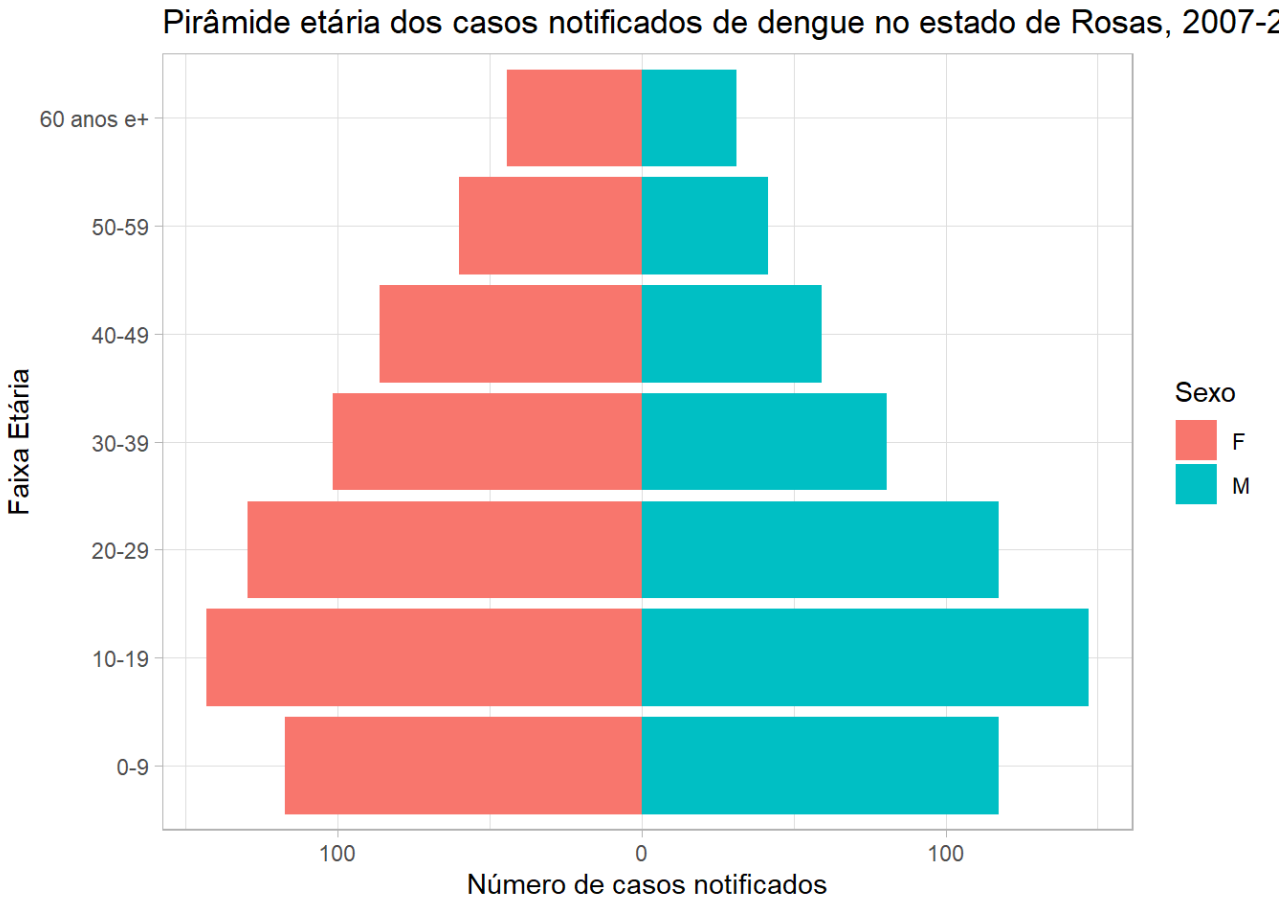
Tabela 7: Tabela {`piramide`} com dados organizados para gerar um gráfico no formato pirâmide (Figura 19).

fx_etaria	CS_SEXO	n
0-9	F	-1175
0-9	M	1173
10-19	F	-1433
10-19	M	1469
20-29	F	-1296
20-29	M	1175
30-39	F	-1015
30-39	M	807
40-49	F	-863
40-49	M	593

Agora, perceba que vamos configurar o gráfico, Figura 19, para a visualização da pirâmide etária de casos de dengue em Rosas. Observe o *script* a seguir e replique-o em seu `RStudio`:

```
ggplot(data = piramide) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para o preenchimento das barras da pirâmide
  aes(x = n,
      y = factor(fx_etaria),
      fill = fct_drop(CS_SEXO)) +

  # Adicionando a geometria de colunas, definindo
  # a largura e o tipo de colunas
  geom_col(width = 0.9, position = "identity") +

  # Arrumando o eixo x, definindo os rótulos
  # a serem mostrados
  scale_x_continuous(labels = abs(c(-200, -100, 0, 100, 200))) +

  # Definindo os títulos dos eixos x, y e título da legenda do gráfico
  labs(
    title = 'Pirâmide etária dos casos notificados de dengue no estado de Rosas, 2007-2012',
    x = 'Número de casos notificados',
    y = 'Faixa Etária',
    fill = 'Sexo'
    ) +

  # Definindo o tema base
  theme_light()
```

Figura 19: Gráfico no formato pirâmide com a distribuição dos casos de dengue em Rosas por sexo.



5.5 Gráficos interativos

Com o `R` você também poderá criar gráficos interativos que aceitam zoom, seleção ou alguma função com clique do mouse e, assim, apresentá-los de forma mais dinâmica os gráficos produzidos na vigilância em saúde. Para isso o pacote `plotly` é uma ótima ferramenta. Esse pacote transforma os gráficos estáticos criados pelo pacote `ggplot2` em gráficos dinâmicos. Essa é uma funcionalidade muito atrativa para páginas online, painéis de dados e arquivos offline disponibilizados via e-mail, por exemplo.

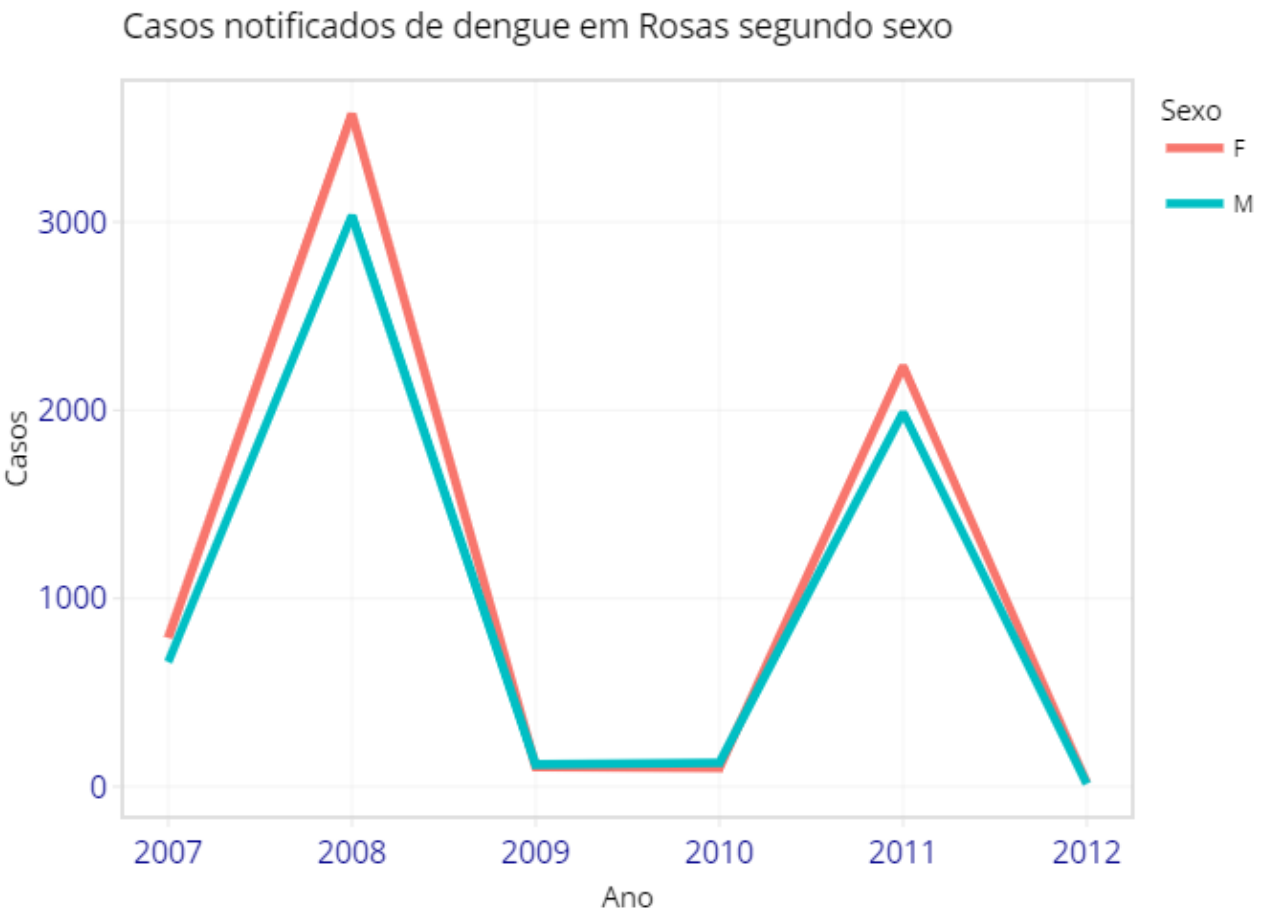
Para transformar os gráficos do `ggplot2` em gráficos interativos, basta utilizarmos a função `ggploty()` do pacote `plotly`, utilizando apenas um argumento essencial que é o objeto gráfico. Vamos adicionar interatividade ao gráfico de linhas criado na seção anterior.

Acompanhe a linha de código a seguir e repita no seu computador:

```
ggplotly(graf_linhas)
```

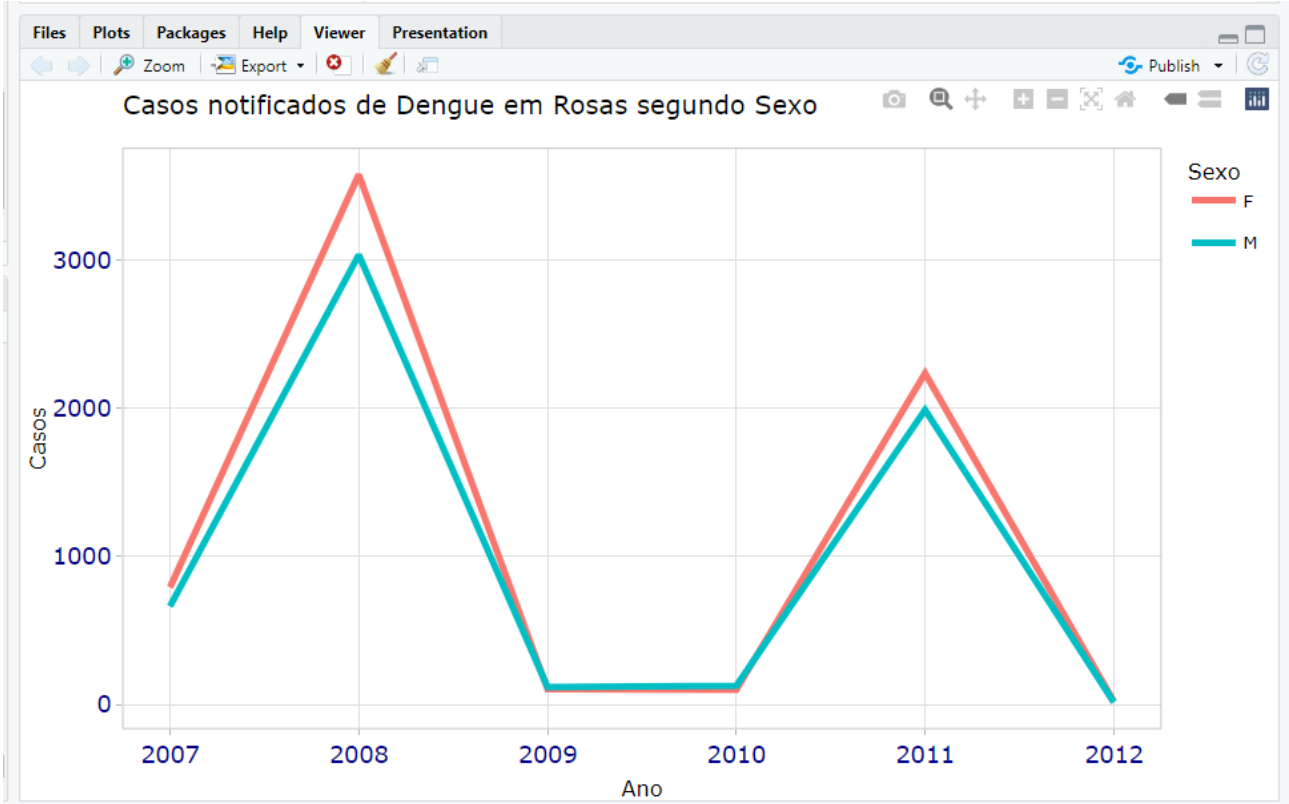
Agora observe o gráfico apresentado na Figura 20.

Figura 20: Gráfico interativo.



Perceba que o gráfico gerado na Figura 20 será visualizado no painel Viewer do `RStudio`. Ao mover o *mouse* pelo gráfico, você verá etiquetas que mostram dados da tabela {`graf_linhas`} utilizada para montá-lo. No canto direito deste espaço, você perceberá que há vários comandos de zoom, de mover e salvar o gráfico como imagem. Explore as possibilidades de interação, veja como fica na Figura 21 a seguir.

Figura 21: Visualização do gráfico interativo no Painel Viewer.



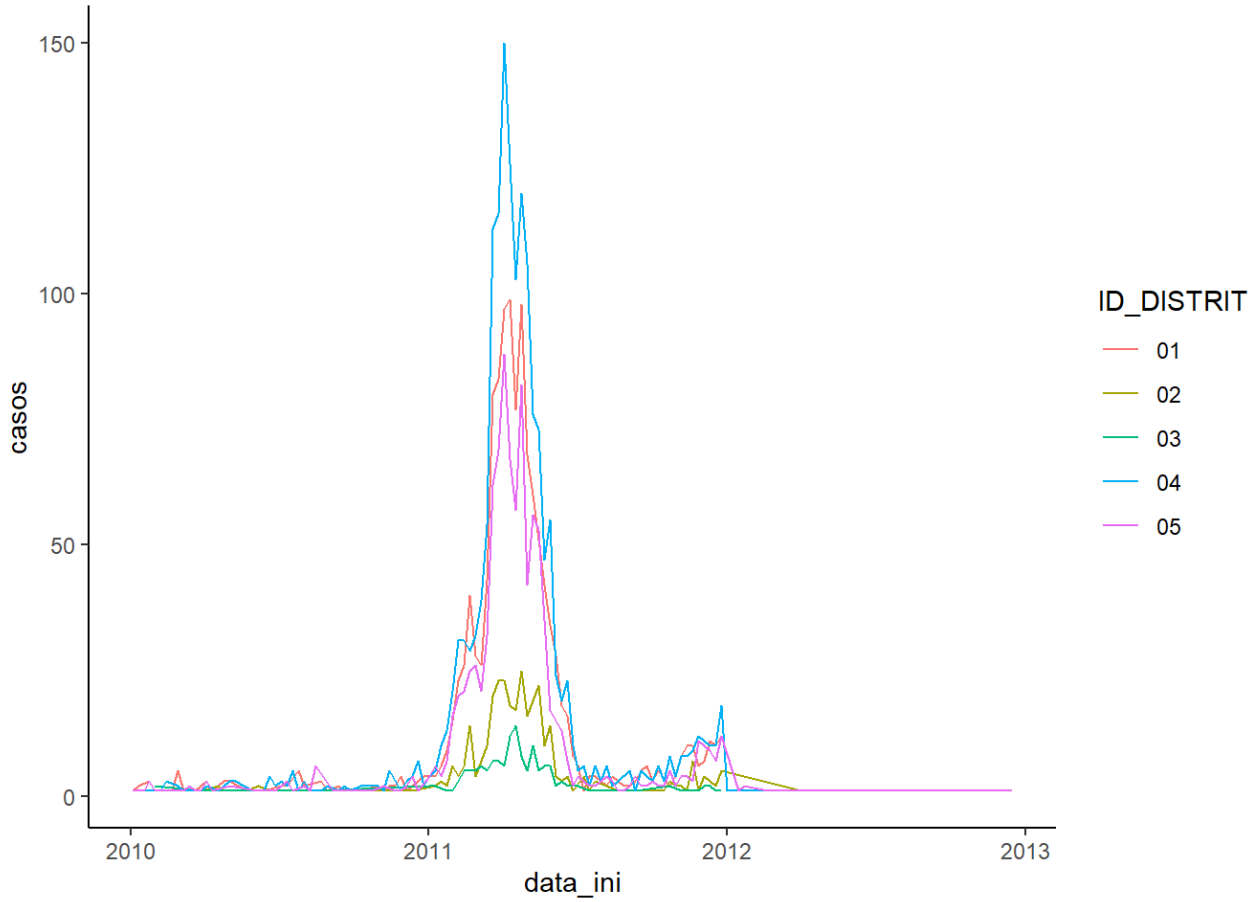
5.6 Estratificação por variáveis de interesse

Uma opção importante com o pacote **ggplot2** é a possibilidade de criar painéis que estratificam um gráfico em sub-gráficos de uma maneira simples podendo controlar vários aspectos desses como arranjo, escalas, cores e outros.

No exemplo a seguir, você pode ver o gráfico de linhas para os 5 distritos administrativos do estado de Rosas. Como sempre, vamos preparar o dado contando a ocorrência de casos suspeitos de dengue por semana epidemiológica em cada um dos distritos e, em seguida, criar um gráfico de linhas onde cada distrito recebe uma cor. Acompanhe os scripts a *seguir* e replique-os no seu `RStudio`:

```
# Criando o objeto {`dengue_distritos`}
dengue_distritos <- dengue |>

  # Filtrando os registros com data de início dos
  # sintomas maior ou igual a 1 de janeiro de 2010
  filter(DT_SIN_PRI >= '2010-01-01') |>

  # Agrupando as notificações pelo ano epidemiológico, semana
  # epidemiológica e distritos administrativos
  group_by(ano_epi, sem_epi, ID_DISTRIT) |>

  # Contando a frequência de notificações
  count(name = "casos") |>

    # Utilizando a função `mutate()` para criar a coluna
    # de data de início da semana epidemiológica
  mutate(data_ini = get_date(sem_epi, ano_epi, start = 7))

# Criando o gráfico
ggplot(data = dengue_distritos) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para a cor
  aes(x = data_ini, y = casos, color = ID_DISTRIT) +

  # Adicionando a linha referente aos casos
  geom_line() +

  # Definindo o tema base
  theme_classic()
```

Figura 22: Gráfico de linhas da distribuição de casos de dengue por distrito de Rosas, entre 2010 e 2013.



O *script* acima criou uma tabela do tipo *dataframe* com dados de dengue notificados a partir de janeiro de 2010, agrupados e sumarizados por ano e semana epidemiológica em cada distrito administrativo do Estado de Rosas, no objeto {`dengue_distritos`}. Em seguida, conforme Figura 22, plotou um gráfico de linhas representa cada distrito apresentado.

Mas perceba que na Figura 22 as linhas se sobrepõem e fica um pouco difícil visualizar o comportamento da doença em cada distrito. Ainda mais porque alguns tiveram muitos casos e outros bem menos. Para melhorar esta visualização, conforme Figura 23, vamos criar sub-gráficos para cada distrito.

No `ggplot2` há duas funções para criar sub-gráficos estratificados por alguma variável:

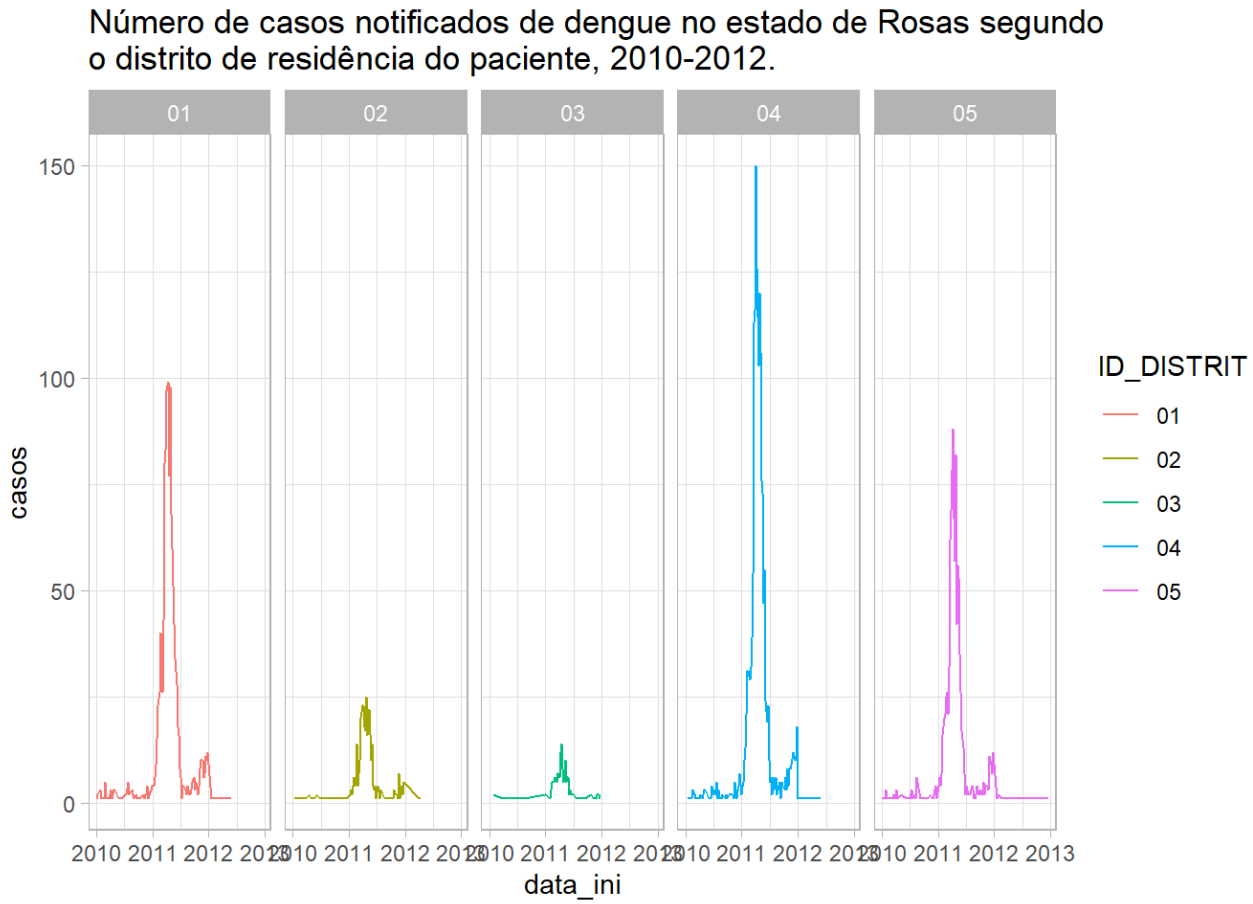
- `facet_grid()`,
- `facet_wrap()`.

A primeira, `facet_grid()`, utiliza como argumento principal o operador til (`~`) seguido da variável que será estratificada em sub-gráficos. Isso significa que a função irá espalhar as categorias da variável por uma linha, um ao lado do outro.

Acompanhe o *script* a seguir e replique-os no seu `RStudio`:

```
ggplot(data = dengue_distritos) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para a cor
  aes(x = data_ini, y = casos, color = ID_DISTRIT) +

  # Adicionando a linha referente aos casos
  geom_line() +

  # Estratificando pelo distrito
  facet_grid(~ID_DISTRIT) +

  # Definindo o tema base
  theme_light() +

  # Definindo o título do gráfico.
  labs(title = "Número de casos notificados de dengue no estado de Rosas segundo
o distrito de residência do paciente, 2010-2012.")
```

Figura 23: Gráfico de linhas da distribuição de casos de dengue por distrito de Rosas, entre 2010 e 2013, segundo distrito de residência.



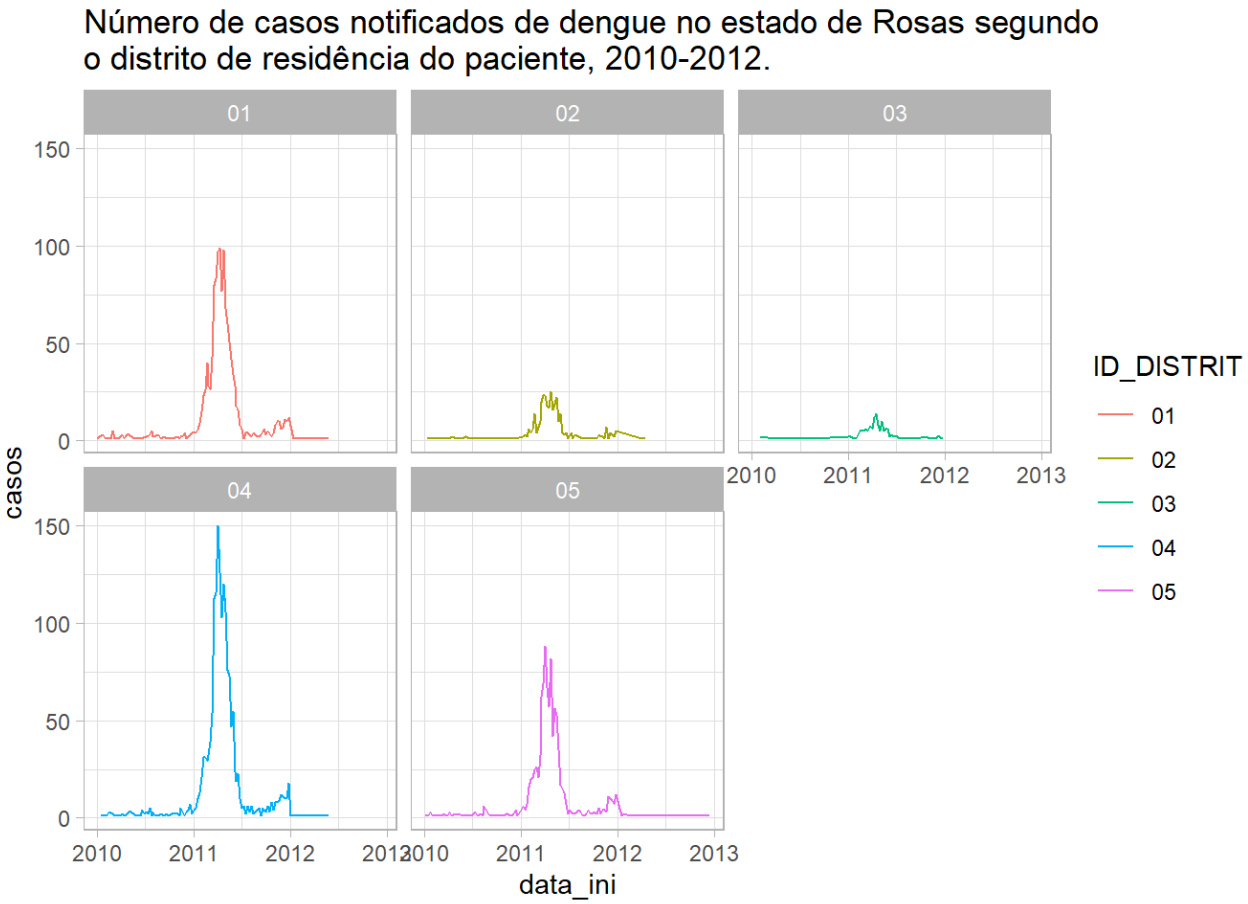
Percebeu o quanto a visualização melhorou? O título dos sub-gráficos são as categorias da variável distrito, o eixo x repete para cada sub-gráfico, mas o eixo y é o mesmo para todos.

A função `facet_wrap()` funciona da mesma forma e tem o mesmo tipo de argumento. Uma das diferenças entre as duas é o output da função. Utilizando a mesma tabela criada anteriormente, {`dengue_distritos`}, vamos criar o mesmo gráfico anterior, só mudando para `facet_wrap()`.

Acompanhe o *script* a seguir e replique-os no seu `RStudio`:

```
ggplot(data =dengue_distritos) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para a cor
  aes(x = data_ini, y = casos, color = ID_DISTRIT) +

  # Adicionando a linha referente aos casos
  geom_line() +

  # Estratificando pelo distrito
  facet_wrap( ~ ID_DISTRIT) +

  # Definindo o tema base
  theme_light() +
  # Definindo o título do gráfico.
  labs(
    title = "Número de casos notificados de dengue no estado de Rosas segundo
o distrito de residência do paciente, 2010-2012."
  )
```

Figura 24: Gráfico de linhas, com a função `facet_wrap()`, da distribuição de casos de dengue por distrito de Rosas, entre 2010 e 2013.



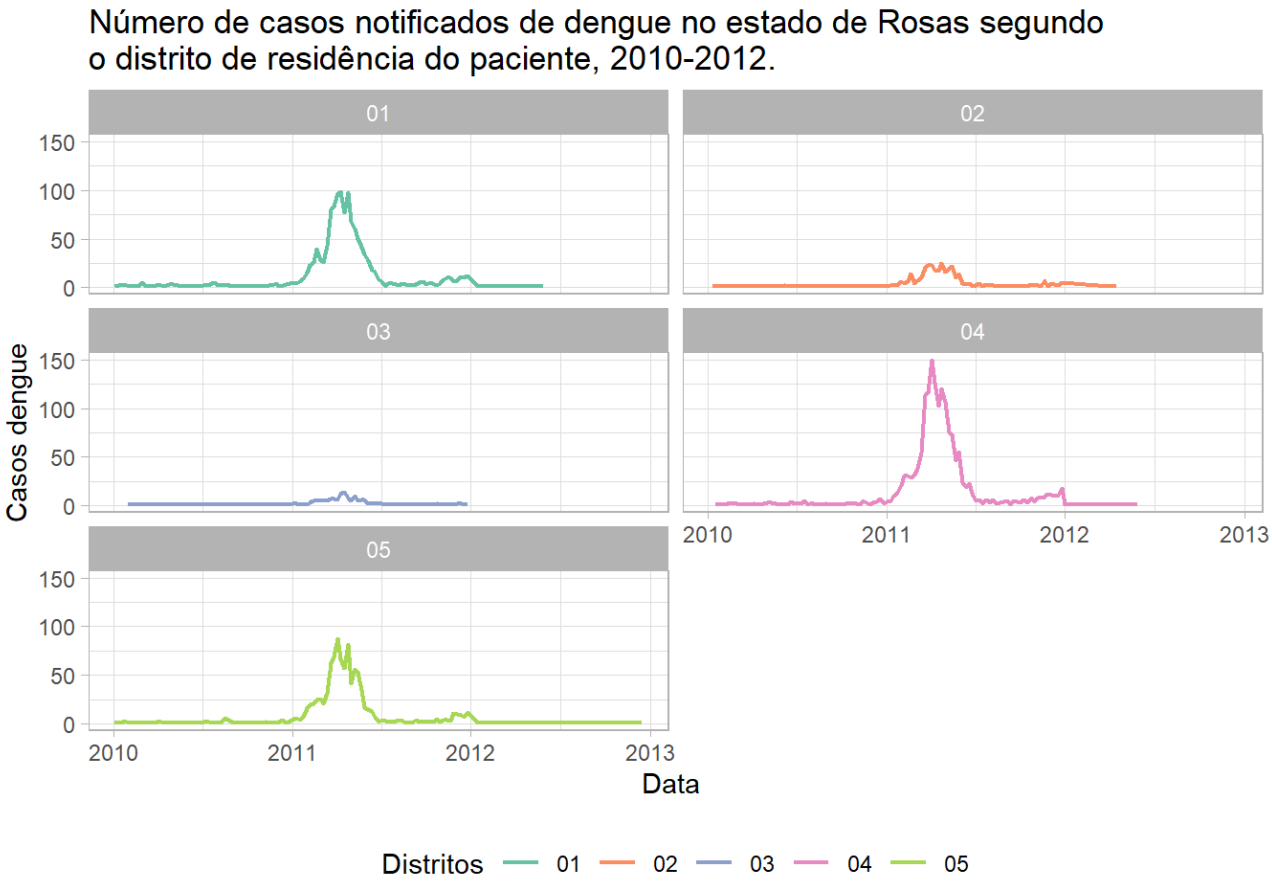
Percebeu a diferença? Os sub-gráficos foram posicionados em três colunas, sendo a linha "quebrada" para outra, na qual foram posicionados mais dois. Outra diferença é que agora o eixo x é compartilhado com alguns sub-gráficos e o eixo y se repete quando troca de linha de sub-gráficos.

A quantidade de colunas e linhas pode ser controlada. Perceba o código a seguir, no qual definimos que a disposição dos sub-gráficos (Figura 25) será em duas colunas, deixando livre o número de linhas. Além disso, vamos posicionar a legenda a seguir de todos os sub-gráficos e definir uma escala de cores diferentes do padrão dado pelo pacote, ou seja, vamos criar uma rampa de cores utilizando códigos hexadecimais.

Acompanhe o código a seguir e repita no seu computador:

```
# Criando uma rampa de cores utilizando códigos hexadecimais
cores <- c("#66C2A5" , "#FC8D62" , "#8DA0CB" , "#E78AC3", "#A6D854")


ggplot(data =dengue_distritos) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para a cor
  aes(x = data_ini, y = casos, color = ID_DISTRIT) +

  # Adicionando geometria de linha e definindo a espessura
  geom_line(size = 0.8) +

  # Estratificando pelo distrito e definindo 2 colunas de disposição dos
  # sub-gráficos
  facet_wrap(~ ID_DISTRIT, ncol = 2) +

  # Definindo os títulos dos eixos x e y
  #   # Definindo o título do gráfico
  labs(
    title = "Número de casos notificados de dengue no estado de Rosas segundo
o distrito de residência do paciente, 2010-2012.",
    x = 'Data ',
    y = 'Casos dengue')+

  # Arrumando a rampa de cores conforme o vetor criado (`cores`)
  scale_color_discrete(type = cores, name = 'Distritos') +

  # Definindo o tema base
  theme_light() +

  # Alterando o tema do gráfico, posicionando a legenda para baixo
  theme(legend.position = "bottom")
```

Figura 25: Gráfico de linhas, com duas colunas e legenda inferior central, da distribuição de casos de dengue por distrito de Rosas, entre 2010 e 2013.

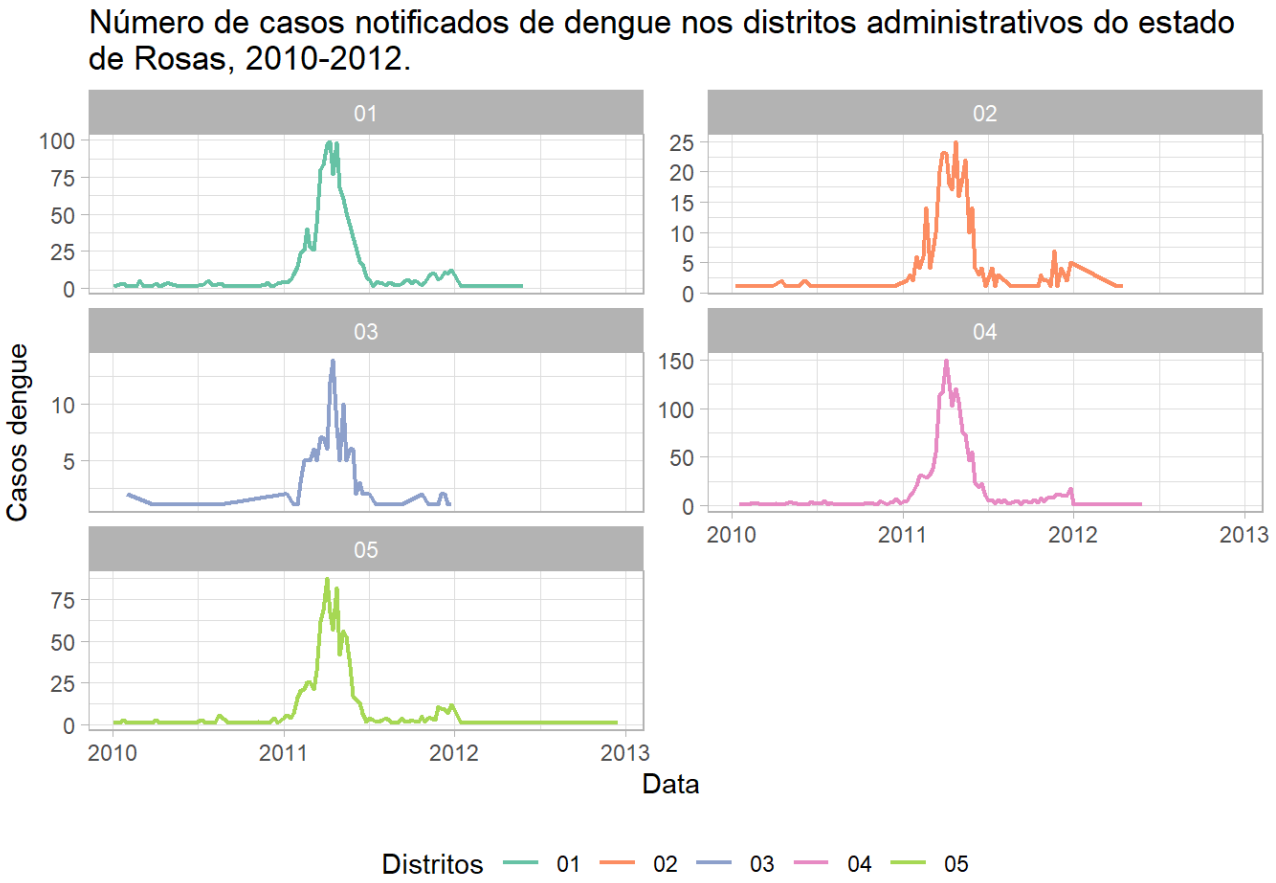


A melhor disposição dos sub-gráficos depende da forma que você deseja representar o evento. Por exemplo, para comparação entre distritos, é de grande importância ter a mesma escala no eixo x. Isso permite que a magnitude do acometimento da doença em cada região seja representada. Mas, considere que o eixo x dos sub-gráficos não esteja representando os distritos administrativos que possuem poucos casos notificados e você decida alterar a escala do eixo x (Figura 26).

Uma das formas de adequar o gráfico é definir o argumento `scales` como `free_y` dentro da função `facet_wrap()`. Acompanhe o *script* a seguir e repita no seu `RStudio`, fazendo a comparação entre os sub-gráficos:

```
# Criando uma rampa de cores utilizando códigos hexadecimais
cores <- c("#66C2A5" , "#FC8D62" , "#8DA0CB" , "#E78AC3", "#A6D854")

# Criando o objeto gráfico {`painel`}
painel <- ggplot(data =dengue_distritos) +

  # Definindo argumentos estéticos com as variáveis usadas em x e em y
  # e a variável usada para a cor
  aes(x = data_ini, y = casos, color = ID_DISTRIT) +

  # Adicionando a geometria de linha e definindo
  # a espessusra
  geom_line(size = 0.8) +

  # Estratificando pelo distrito e definindo
  # mudança no eixo y conforme cada sub-gráfico
  facet_wrap(~ ID_DISTRIT, ncol = 2, scales = "free_y") +

  # Definindo os títulos dos eixos x, y e título do gráfico
  labs(
    title = "Número de casos notificados de dengue nos distritos administrativos do estado
de Rosas, 2010-2012.",
     x = 'Data ',
     y = 'Casos dengue') +

  # Arrumando a rampa de cores
  scale_color_discrete(type = cores, name = 'Distritos') +

  # Definindo o tema base
  theme_light() +

  # Alterando o tema do gráfico
  theme(legend.position = "bottom")

# Plotando o objeto `painel`
painel
```

Figura 26: Gráfico de linhas, alterando a escala do eixo x, da distribuição de casos de dengue por distrito de Rosas, entre 2010 e 2013.



Observe que agora cada sub-gráfico possui um eixo y diferente. Dessa forma, é possível visualizar o comportamento da doença em cada distrito.

Interessante, não é mesmo? Na próxima seção, você aprenderá como salvar estes gráficos no seu computador.

6. Exportar os gráficos como imagem

Muitas vezes na rotina da vigilância precisamos construir nossas apresentações em slides, relatórios ou até mesmo enviar elementos gráficos por e-mail de forma rápida. Quando os gráficos são criados utilizando o pacote `ggplot2` isso é possível.

Para isso, poderemos exportá-los para dois tipos de arquivos:

1. Arquivo **raster**: são imagens digitais convertidas em matrizes com número fixo de linhas e colunas, onde cada célula corresponde a um pixel. Quanto mais linhas e colunas, maior é a resolução da imagem. Devido à ampla compatibilidade com vários tipos de equipamentos são muito populares para visualização de gráficos, embora sofra com distorções caso tenham seu tamanho alterado.
2. Arquivo **vetorial**: são arquivos que armazenam geometrias como pontos, linhas e polígonos, como no caso dos mapas por exemplo. Não sofrem distorção quando têm seu tamanho alterado e são recomendados para gráficos de alta qualidade, embora possam ocupar mais espaço no computador devido a seu tamanho em *bytes*. Para saber mais sobre os tipos de arquivos vetoriais mais usados atualmente, clique em https://www.adobe.com/br/creativecloud/file-types/image/vector.html

O tamanho do arquivo de imagem corresponde ao tamanho do arquivo digital, medido em *bytes*, *megabytes* e *gigabytes*. Quanto mais *pixels* a imagem possui, maior é o tamanho digital dela.

Os principais formatos que são aceitos para exportação no dia a dia são:

- **"jpeg"**: o arquivo `JPEG` (sigla para *Joint Photographic Experts Group*) é um formato rasterizado que possui um protocolo de compactação de dados que diminui a qualidade para reduzir o tamanho em bytes, embora seja muito utilizado.
- **"png"**: o arquivo `PNG` (sigla para *Portable Network Graphic*) é um formato rasterizado mais utilizado para ambiente web.
- **"tiff"**: o arquivo `TIFF` (sigla para *Tag Image File Format*) é um formato rasterizado que possui um protocolo de compactação com menos perda da qualidade, mas o tamanho em bytes é maior que as demais.
- **"svg"**: o arquivo `SVG` (sigla para *Scalable Vector Graphics*) é um formato vetorial mais utilizado para exibição de imagens no ambiente web. Por ser vetorial não perde qualidade quando redimensionado e possuem tamanho (em bytes) menor que as imagens rasterizadas.
- **"pdf"**: o arquivo `PDF` (sigla para *Portable Document Format*) é conhecido para armazenamento de textos, mas é também útil para gráficos.
- **"eps"** e **"ps"**: o arquivo `EPS` e `PS` (siglas em inglês para *Encapsulated PostScript* e *Post Script*, respectivamente) é um formato vetorial usado para impressão de imagens profissionais e de alta qualidade,

Agora vamos praticar! No pacote `ggplot2` usando a função `ggsave()` é possível exportar os gráficos produzidos aqui neste curso de maneira muito simples. Para isso, basta inserir o nome do objeto gráfico a ser exportado e o caminho com a extensão desejada.

Acompanhe o *script* a seguir e replique-o em seu `RStudio`:

```
# Sintaxe para exportação de imagens do `ggplot2`
ggsave(
  plot = graf_barras,
  filename = "grafico_barras_dengue_2007_2012.png"
)
```

Pronto! Caso deseje exportar a imagem com uma definição de tamanho em largura e altura especificas é só utilizar os argumentos `width` e `height`. Eles devem ser definidos juntamente com a unidade (centímetro, milímetro, etc). Observe o *script* a seguir e replique-o em seu `RStudio`:

```
# Sintaxe para exportação de imagens do `ggplot2`
ggsave(
  plot = graf_barras,
  filename = "grafico_barras_dengue_2007_2012.png",
  width = unit(15, "cm"),
  height = unit(10, "cm")
)
```

Você também poderá personalizar a qualidade da imagem que está gerando. Para isso você poderá incluir o uso do argumento `dpi`, que definirá a resolução em *pixels*. Quanto maior o *dpi* melhor a resolução da imagem salva. Por padrão, muitas impressoras utilizam o valor de 300 *dpi*.

Acompanhe o *script* a seguir com atenção e teste-o no seu `RStudio`:

```
# Sintaxe para exportação de imagens do `ggplot2`
ggsave(
  plot = graf_barras,
  filename = "grafico_barras_dengue_2007_2012.png",
  dpi = 300
)
```

7. Exportar os gráficos como arquivo PDF

Mas e se você desejar exportar no formato PDF? Para exportar com uma extensão diferente, basta trocar no nome do arquivo que será salvo. Perceba a seguir que estamos exportando o mesmo gráfico de antes, mudando apenas a extensão para `PDF`. Copie e cole o código a seguir e replique-o no seu `RStudio`:

```
# Sintaxe para exportação de imagens do `ggplot2`
ggsave(
  plot = graf_barras,
  filename = "grafico_barras_dengue_2007_2012.pdf",
)
```

Pronto, agora você já consegue gerar um documento com as análises gráficas da situação epidemiológica da dengue no Estado de Rosas e enviá-lo por e-mail ao secretário de saúde, por exemplo. Muito fácil, não é mesmo?

Nossos cursos

Pronto, chegamos ao final deste curso! Agora você já conhece as principais ações para construir *gráficos* com o apoio da linguagem de programação `R`. Quer seguir a diante no aprendizado? Você encontrará outras etapas para aprofundamento das análises de dados em vigilância em saúde nos outros cursos. Aproveite e já faça sua inscrição nos cursos a seguir clicando nos *links*:

- Análises de dados para vigilância em saúde - curso básico.
- Construção de diagramas de controle na vigilância em saúde.
- Produção automatizada de relatórios na vigilância em saúde.
- Linkage de bases de dados de saúde.
- Análise espacial de dados para a vigilância em saúde.
- Construção de painéis (dashboards) para monitoramento de indicadores de saúde.

Aproveite!

Instituto Todos pela Saúde

ABRASCO
ASSOCIAÇÃO BRASILEIRA DE SAÚDE COLETIVA

UNA-SUS
Universidade Aberta do SUS

FIOCRUZ

UFPEL

UFSC