

aula04  
ME315-2S2025

**brotto**

2025-11-12

## Sumário

§ Natureza . . . . .	2
§ Resumo . . . . .	2
§ Introdução . . . . .	2
§ Conjunto de dados (CD): <code>flights.csv.zip</code> . . . . .	3
<b>Laboratório 2: Processamento de Bases de Dados em Lote</b>	<b>4</b>
§ Instruções . . . . .	4
<b>Desafio 2: Python</b>	<b>10</b>
§ Resultados e Comparações (R-Python) . . . . .	13
<b>Trabalho Final: Julia</b>	<b>14</b>
§ Resultados e Comparações (R-Julia) . . . . .	22

## § Natureza

Este documento apresenta as soluções propostas para as seguintes atividades da disciplina ME315-2S2025:

- **Laboratório nº 2**, apresentado em aula no dia 14 de agosto de 2025;
- **Desafio nº 2**, que consiste na replicação do Laboratório 2 utilizando a linguagem **Python**;
- **Trabalho Final**, proposto em 21 de outubro de 2025, que estende a análise para a linguagem **Julia**.

## § Resumo

São apresentados, a seguir, exemplos de soluções para os problemas mencionados. O **Laboratório 2** consiste em um exercício originalmente elaborado por professores que ministraram a disciplina em semestres anteriores. O **Desafio 2** propõe a reprodução da mesma atividade utilizando a linguagem **Python**, enquanto o **Trabalho Final** expande a abordagem para a linguagem **Julia**.

## § Introdução

A atividade inicial, proposta pelos professores Benilton e Guilherme, tem como objetivo desenvolver habilidades essenciais para estatísticos, especialmente no que se refere à manipulação de conjuntos de dados, incluindo aqueles de grande volume. Com a inclusão das linguagens Python e Julia, o escopo do trabalho ampliou-se, abrangendo agora a comparação entre diferentes ferramentas computacionais.

Em virtude da integração de múltiplas atividades em um único documento, detalhes excessivos sobre cada solução nem sempre são apresentados ao longo do texto. Para uma compreensão mais aprofundada, recomenda-se a consulta ao código-fonte em **RMarkdown** utilizado para gerar este relatório.

Vale ressaltar que a compilação desse código depende da instalação de diversos pacotes e da configuração prévia dos ambientes **R**, **Python** e **Julia**. A instalação das dependências de R é tratada automaticamente pelo documento, porém, os interpretadores de Python e Julia devem estar previamente configurados. Além disso, o arquivo `columns.tex` deve estar presente no mesmo diretório do arquivo Rmd durante a compilação. Interessados em reproduzir ou modificar este relatório devem assegurar-se de atender a todos os requisitos listados.

Tabela 1: Os dois primeiros registros do CD.

YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER
2015	1	1	4	AS	98
2015	1	1	4	AA	2336
TAIL_NUMBER		ORIGIN_AIRPORT		DESTINATION_AIRPORT	
N407AS		ANC		SEA	
N3KUAA		LAX		PBI	
SCHEDULED_DEPARTURE		DEPARTURE_TIME		DEPARTURE_DELAY	TAXI_OUT
0005		2354		-11	21
0010		0002		-8	12
WHEELS_OFF	SCHEDULED_TIME		ELAPSED_TIME	AIR_TIME	DISTANCE
0015	205		194	169	1448
0014	280		279	263	2330
WHEELS_ON	TAXI_IN	SCHEDULED_ARRIVAL		ARRIVAL_TIME	ARRIVAL_DELAY
0404	4	0430		0408	-22
0737	4	0750		0741	-9
DIVERTED	CANCELLED	CANCELLATION_REASON		AIR_SYSTEM_DELAY	
0	0	NA		NA	
0	0	NA		NA	
SECURITY_DELAY		AIRLINE_DELAY		LATE_AIRCRAFT_DELAY	WEATHER_DELAY
NA		NA		NA	NA
NA		NA		NA	NA

## § Conjunto de dados (CD): flights.csv.zip

```
## Nome do arquivo com os dados
file0 = "flights.csv.zip"
```

```
## Amostra do CD com os 2 primeiros registros
a = readr::read_csv(file = file0, n_max = 2)
lista_partes <- split.default(a, cut(1:ncol(a), breaks = seq(0, ncol(a), by = 1)))
```

```
## Trata e imprime Tabela 1
latex = (lista_partes %>% my_kbl())[1]
ss = substring(latex, 14, nchar(latex))
s4 = " Colunas}"; s3 = "\\textit{ "; s2 = "\\caption{Os dois primeiros registros do
↪ CD.}"; s1 = "\\begin{table}"
paste0(s1,s2,ss,s3,file.info(file0)$size * 1e-6," Mb | ",ncol(a),s4) %>% cat()
```

200.91153 Mb / 31 Colunas

## Laboratório 2: Processamento de Bases de Dados em Lote

Benilton Carvalho & Guilherme Ludwig

### § Instruções

1. Quais são as estatísticas suficientes para a determinação do percentual de vôos atrasados na chegada (`ARRIVAL_DELAY` > 10)?

---

O Número Total de Voos no conjunto de dados ( $n$ ) e a Contagem de Voos cujo valor na coluna `ARRIVAL_DELAY` seja maior que 10 ( $x$ ). Com essas duas estatísticas, o percentual  $p$  é calculado por

$$p = (x/n) * 100$$

---

2. Crie uma função chamada `getStats` que, para um conjunto de qualquer tamanho de dados provenientes de `flights.csv.zip`, execute as seguintes tarefas (usando apenas verbos do `dplyr`):
  - a) Filtre o conjunto de dados de forma que contenha apenas observações das seguintes Cias. Aéreas: AA, DL, UA e US;
  - b) Remova observações que tenham valores faltantes em campos de interesse;
  - c) Agrupe o conjunto de dados resultante de acordo com: dia, mês e cia. aérea;
  - d) Para cada grupo em b), determine as estatísticas suficientes apontadas no item 1. e os retorne como um objeto da classe `tibble`;
  - e) A função deve receber apenas dois argumentos:
    - i. `input` : o conjunto de dados (referente ao lote em questão);
    - ii. `pos` : argumento de posicionamento de ponteiro dentro da base de dados. Apesar de existir na função, este argumento não será empregado internamente. É importante observar que, sem a definição deste argumento, será impossível fazer a leitura por partes.

```
getStats = function(input, pos) {  
  
  # Filtra os dados nas cias. de interesse e remove linhas sem valores de atraso  
  input %>%  
    filter(  
      AIRLINE %in% c("AA", "DL", "UA", "US"),  
      !is.na(ARRIVAL_DELAY)  
    ) %>%  
  
  # Agrupa dados por dia, mês e cia.  
  group_by(DAY, MONTH, AIRLINE) %>%  
  
  # Cria coluna de tipo booleano, TRUE para atraso significativo (>10 minutos)  
  mutate(late = (ARRIVAL_DELAY > 10))  
}
```

3. Utilize alguma função `readr::read_*_*_*_chunked` para importar o arquivo `flights.csv.zip`.

- a) Configure o tamanho do lote (chunk) para 100 mil registros;
- b) Configure a função de `callback` para instanciar DataFrames aplicando a função `getStats` criada acima;
- c) Configure o argumento `col_types` de forma que ele leia, diretamente do arquivo, apenas as colunas de interesse (veja nota de aula para identificar como realizar esta tarefa);

```
## Define colunas específicas para leitura eficiente dos dados CSV
mycols = cols_only(
  DAY = 'i',          # Dia do mês (inteiro)
  MONTH = 'i',        # Mês (inteiro)
  YEAR = 'i',          # Ano (inteiro)
  AIRLINE = 'c',       # Código da Cia. (string)
  ARRIVAL_DELAY = 'i' # Tempo de atraso em minutos (inteiro)
)

## Captura do tempo inicial para monitoramento de performance
t0 = Sys.time()

## Processa os dados em lote
in2 = read_csv_chunked(
  file = file0,          # Nome do arquivo
  callback = DataFrameCallback$new(getStats), # Função a ser usada no processamento de
  ↪ cada lote.
  chunk_size = 1e5,      # Tamanho do lote (100.000 linhas)
  col_types = mycols     # Restrição da leitura às colunas pré
  ↪ especificadas.
)

## Captura do tempo final
t1 <- Sys.time()
```

$$\Delta t = t1 - t0 = 21.6711900234222 \text{ s}$$

4. Crie uma função chamada `computeStats` que:
  - a) Combine as estatísticas suficientes para compor a métrica final de interesse (percentual de atraso por dia/mês/cia aérea);
  - b) Retorne as informações em um `tibble` contendo apenas as seguintes colunas:
    - i. `Cia` : sigla da companhia aérea;
    - ii. `Data` : data, no formato AAAA-MM-DD (dica: utilize o comando `as.Date` );
    - iii. `Perc` : percentual de atraso para aquela cia. aérea e data, apresentado como um número real no intervalo  $[0, 1]$ .

```
computeStats <- function(input) {
  input %>%

  # Cria uma coluna de data a partir das colunas com dados de ano, mês e dia.
  mutate(
    DATE = as.Date(paste(YEAR, MONTH, DAY, sep = "-"))
  ) %>%
```

```

# Agrupa os dados por cia. e data
group_by(AIRLINE, DATE) %>%

# Calcula atraso diário percentual para cada cia.
reframe(
  Perc = sum(1 * (late)) / n() # Converte valores booleanos para inteiros (1 ou
    ↪ 0) e calcula a média da sua soma
)
}

```

5. Produza um mapa de calor em formato de calendário para cada Cia. Aérea.

- a) Instale e carregue os pacotes `ggcal` e `ggplot2`.
- b) Defina uma paleta de cores em modo gradiente. Utilize o comando `scale_fill_gradient`. A cor inicial da paleta deve ser `#4575b4` e a cor final, `#d73027`. A paleta deve ser armazenada no objeto `pal`.
- c) Crie uma função chamada `baseCalendario` que recebe 2 argumentos a seguir: `stats` (`tibble` com resultados calculados na questão 4) e `cia` (sigla da Cia. Aérea de interesse). A função deverá:
  - i. Criar um subconjunto de `stats` de forma a conter informações de atraso e data apenas da Cia. Aérea dada por `cia`.
  - ii. Para o subconjunto acima, montar a base do calendário, utilizando `ggcal(x,y)`. Nesta notação, `x` representa as datas de interesse e `y`, os percentuais de atraso para as datas descritas em `x`.
  - iii. Retornar para o usuário a base do calendário criada acima.
- d) Executar a função `baseCalendario` para cada uma das Cias. Aéreas e armazenar os resultados, respectivamente, nas variáveis: `cAA`, `cDL`, `cUA` e `cUS`.
- e) Para cada uma das Cias. Aéreas, apresente o mapa de calor respectivo utilizando a combinação de camadas do `ggplot2`. Lembre-se de adicionar um título utilizando o comando `ggtitle`. Por exemplo, `verb01`.

```

## Captura do tempo inicial para monitoramento de performance
t0 = Sys.time()

## Aplica funções com computações estatísticas aos dados processados
in4 <- computeStats(in2)

## Define paleta de cores para visualização dos caledários
## Grandiente de azul a vermelho para representação da magnitude dos atrasos
pal <- scale_fill_gradient(low = "#4575b4", # Azul (Pouco)
                           high = "#d73027") # Vermelho (Muito)

## Base do calendário para visualização das estatísticas de atraso
baseCalendario <- function(stats, cia) {

  # Filtra os dados nas cias. de interesse
  input <- stats %>%
    filter(AIRLINE == cia)

  # Cria mapa de calor no calendário usando função do pacote ggcal

```

```

  ggcal(input$DATE, input$Perc)
}

## Gera os calendários para visualização de cada cia.
g1 <- baseCalendario(in4, "AA") +
  pal +
  ggtitle("American Airlines (AA)")

g2 <- baseCalendario(in4, "DL") +
  pal +
  ggtitle("Delta Air Lines (DL)")

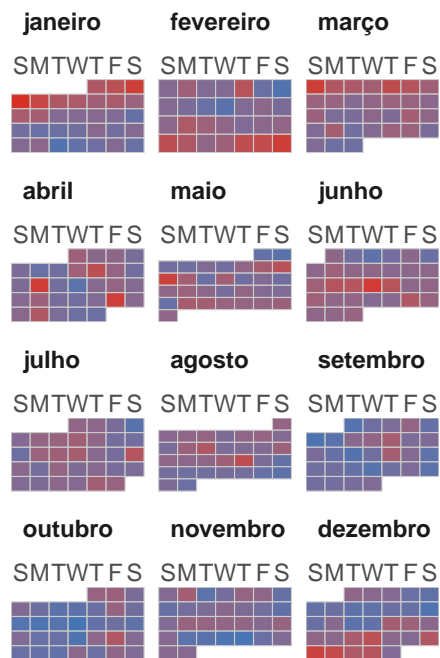
g3 <- baseCalendario(in4, "UA") +
  pal +
  ggtitle("United Airlines (UA)")

g4 <- baseCalendario(in4, "US") +
  pal +
  ggtitle("US Airways (US)")

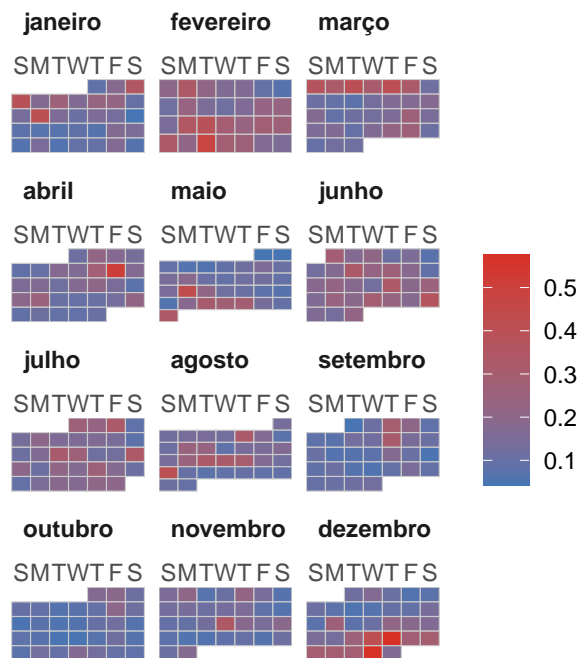
## Imprime calendários
g1 + g2

```

### American Airlines (AA)

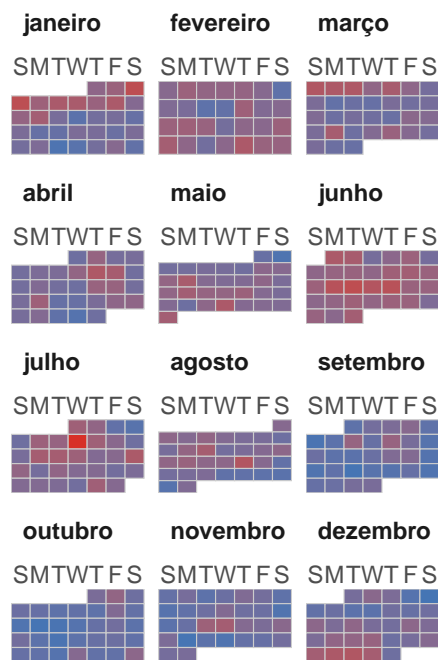


### Delta Air Lines (DL)



g3 + g4

## United Airlines (UA)



## US Airways (US)

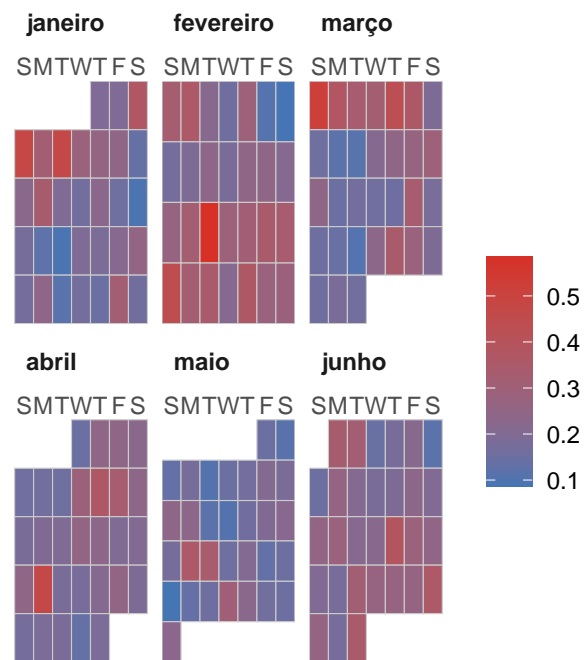


Figura 1: Mapa de calor dos vôos atrasados das Cias. aéreas AA, DL, UA e US, respectivamente

```
computeMoreStats = function(stats) {
  stats %>%

  # Seleciona
  filter(AIRLINE %in% c("AA", "DL", "UA", "US")) %>%

  # Agrupa os dados por cia.
  group_by(AIRLINE) %>%

  # Calcula o atraso percentual de cada cia.
  reframe(
    Média = round(mean(Perc) * 100, 2),
    Máx = round(max(Perc) * 100, 2)
  ) %>%
  arrange(AIRLINE)
}
```

Tabela 2: Média de vôos atrasados por Cia estudada.

```
## Tabela kable()
computeMoreStats(in4) %>% my_kbl()
```

AIRLINE	Média	Máx
AA	22.37	60.21
DL	16.26	57.51
UA	23.64	79.13
US	22.85	58.49

Tabela 3: Média de vôos atrasados por Cia estudada e por dia.

```
## Tabela kable()
in4 %>% my_kbl_foot(header = 10)
```

AIRLINE	DATE	Perc
AA	2015-01-01	0.3727339
AA	2015-01-02	0.4303714
AA	2015-01-03	0.5382456
AA	2015-01-04	0.6021433
AA	2015-01-05	0.4912869
AA	2015-01-06	0.3656934
AA	2015-01-07	0.3549075
AA	2015-01-08	0.3074753
AA	2015-01-09	0.3422724
AA	2015-01-10	0.2797147

*1-10 de 1276 linhas / 3 colunas*

```
## Captura do tempo final
t1 = Sys.time()
```

$\Delta t = 31.8071849346161 \text{ s}$

## Desafio 2: Python

```
## Inicia sessão python
venv_python = "temp.venv"
if(!virtualenv_exists(venv_python)){ virtualenv_create(venv_python, packages =
  ↪ c("pip","setuptools","wheel")) }

py_install("pandas", envname = venv_python)

## Using virtual environment "temp.venv" ...

py_install("calplot", envname = venv_python)

## Using virtual environment "temp.venv" ...

py_install("pyarrow", envname = venv_python)

## Using virtual environment "temp.venv" ...

use_virtualenv(venv_python)

def getStats(flight_chunk, chunk_list):

  # Seleção das cias.
  major_airlines = ['AA', 'DL', 'UA', 'US']

  # Seleciona colunas relevantes no processamento
  filtered_chunk = flight_chunk[['DAY', 'MONTH', 'YEAR', 'AIRLINE', 'ARRIVAL_DELAY']]

  # Filtra os dados nas cias.
  filtered_chunk = filtered_chunk[filtered_chunk['AIRLINE'].isin(major_airlines)]

  # Remove linhas sem valores de atraso nos vôos
  filtered_chunk = filtered_chunk.dropna(subset=['ARRIVAL_DELAY'])

  # Identifica vôos com atraso significativos (> 10 minutos)
  filtered_chunk['late'] = filtered_chunk['ARRIVAL_DELAY'] > 10

  # Anexa lote processado a lista de resultados
  chunk_list.append(filtered_chunk)

def computeStats(flight_data):

  # Cria coluna com valores de data a partir de colunas individuais com informações de
  ↪ tempo
  flight_data['date'] = pd.to_datetime(
    flight_data[['YEAR', 'MONTH', 'DAY']],
    format="%Y-%m-%d"
  )

  # Seleciona apenas as colunas relevantes para agregação
  processed_data = flight_data[['date', 'AIRLINE', 'ARRIVAL_DELAY', 'late']]

  # Agrupa por cia. e data para calcular estatísticas de atraso
  daily_stats = processed_data.groupby(['AIRLINE', 'date']).agg(
```

```

        delayed_count=('late', 'sum'),      # Count of delayed flights
        total_flights=('AIRLINE', 'count') # Total flights in group
    ).reset_index()

    # Cálculo do atraso percentual
    daily_stats['Perc'] = daily_stats['delayed_count'] / daily_stats['total_flights']

    # Retorna apenas colunas relevantes
    return daily_stats[['AIRLINE', 'date', 'Perc']]

```

```

def computeMoreStats(stats):
    import pandas as pd

    airlines = ['AA', 'DL', 'UA', 'US']
    resultados = []

    for airline in airlines:
        airline_data = stats[stats['AIRLINE'] == airline]

        atraso_medio = round(airline_data['Perc'].mean() * 100, 2)
        atraso_maximo = round(airline_data['Perc'].max() * 100, 2)

        resultados.append({
            'AIRLINE': airline,
            'atraso_medio': atraso_medio,
            'atraso_maximo': atraso_maximo,
        })

    return pd.DataFrame(resultados)

```

```

# Bibliotecas
import matplotlib.pyplot as plt
import calplot
import pandas as pd

## Nome do arquivo com os dados
arquivo_csv = 'flights.csv.zip'

# Define o tamanho de cada lote a ser lido
tamanho_chunck = 1e5

# Lista para armazenar os subconjuntos processados
subconjuntos_processados = []

# Cria um objeto leitor que vai iterar sobre o CD
leitor_csv = pd.read_csv(arquivo_csv, chunksize=tamanho_chunck)

# Itera sobre cada chunk do CD
for chunk in leitor_csv:
    getStats(chunk, subconjuntos_processados)

# Concatena todos os subconjuntos em um único DataFrame
df_final = pd.concat(subconjuntos_processados, ignore_index=True)

```

```
# Computa estatísticas
stats = computeStats(df_final)
```

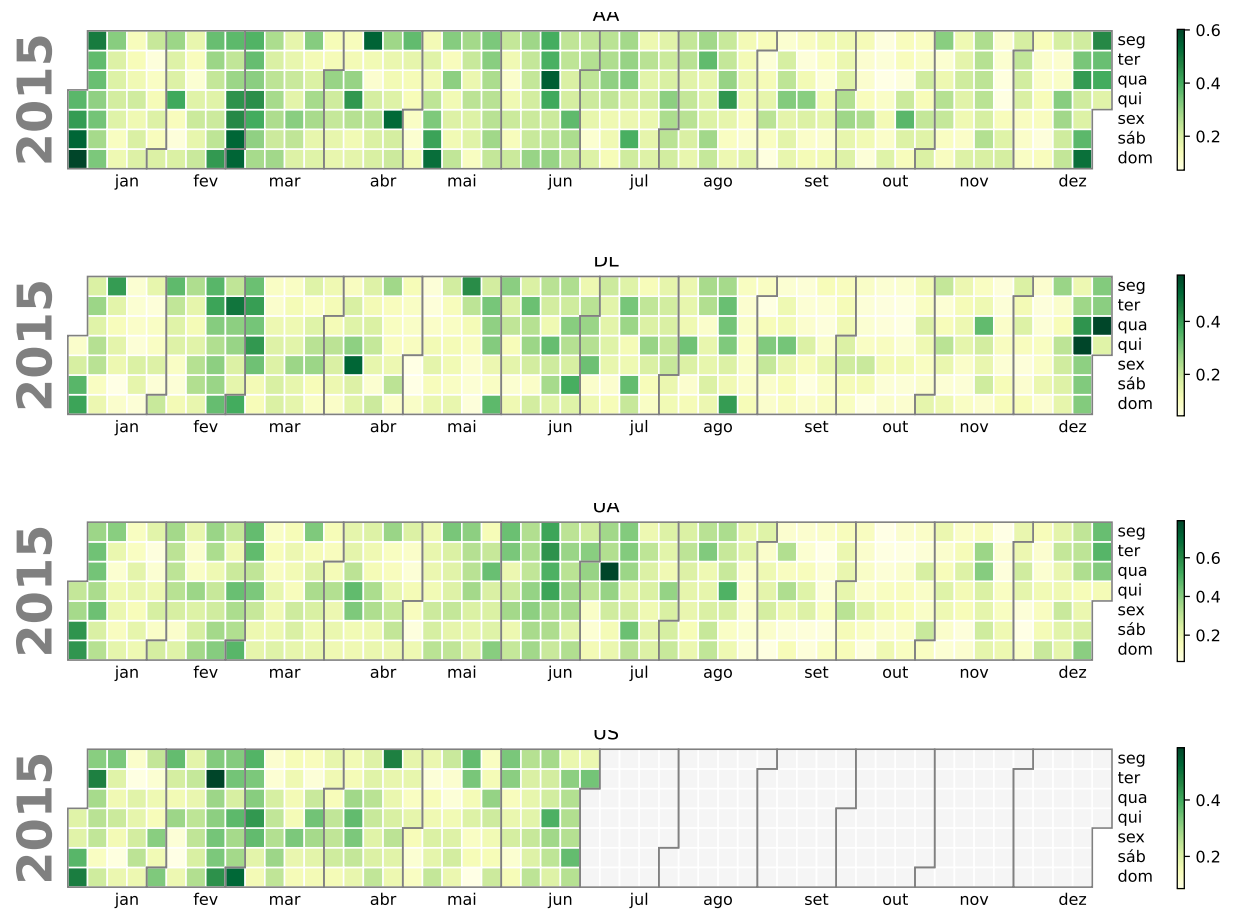


Figura 2: Mapa de calor dos vôos atrasados das Cias. aéreas AA, DL, UA e US, respectivamente. (Obtida através do Python)

## § Resultados e Comparações (R–Python)

Tabela 4: Média de vôos atrasados por Cia estudada e dia (Obtida através do Python)

```
## Atribui o conteúdo da variável stats  
↪ no ambiente python para a variável  
↪ in5 no R  
## Tabela kable()  
in5 = py$stats  
in5 %>% my_kbl_foot(header = 10)
```

AIRLINE	date	Perc
AA	2014-12-31 22:00:00	0.3727339
AA	2015-01-01 22:00:00	0.4303714
AA	2015-01-02 22:00:00	0.5382456
AA	2015-01-03 22:00:00	0.6021433
AA	2015-01-04 22:00:00	0.4912869
AA	2015-01-05 22:00:00	0.3656934
AA	2015-01-06 22:00:00	0.3549075
AA	2015-01-07 22:00:00	0.3074753
AA	2015-01-08 22:00:00	0.3422724
AA	2015-01-09 22:00:00	0.2797147

1-10 de 1276 linhas / 3 colunas

Tabela 5: Média de vôos atrasados por Cia estudada (Obtida através do Python)

```
## Estatísticas extras  
MoreStats = computeMoreStats(stats)
```

```
## Tabela kable()  
in6 = py$MoreStats  
in6 %>% my_kbl()
```

AIRLINE	atraso_medio	atraso_maximo
AA	22.37	60.21
DL	16.26	57.51
UA	23.64	79.13
US	22.85	58.49

```
## Encerra sessão Python  
virtualenv_remove(venv_python)
```

Tabela 3: Média de vôos atrasados por Cia estudada e dia.

```
## Tabela kable()  
in4 %>% my_kbl_foot(header = 10)
```

AIRLINE	DATE	Perc
AA	2015-01-01	0.3727339
AA	2015-01-02	0.4303714
AA	2015-01-03	0.5382456
AA	2015-01-04	0.6021433
AA	2015-01-05	0.4912869
AA	2015-01-06	0.3656934
AA	2015-01-07	0.3549075
AA	2015-01-08	0.3074753
AA	2015-01-09	0.3422724
AA	2015-01-10	0.2797147

1-10 de 1276 linhas / 3 colunas

$$\Delta t = 33.1782040596008 \text{ s}$$

## Trabalho Final: Julia

```
julia_setup(JULIA_HOME = NULL)
julia_assign("file0",file0)

# Instalação de pacotes
import Pkg

# Verifica se o ambiente já existe e está atualizado
function setup_environment()
    env_path = "temp.venv"

    if isdir(env_path)
        println("Ambiente Julia já existe. Verificando pacotes...")
        Pkg.activate(env_path)

        # Lista de pacotes necessários
        required_pkgs = ["CSV", "DataFrames", "Dates", "Statistics", "ZipFile", "Plots"]
        installed_pkgs = keys(Pkg.project().dependencies)

        # Encontra pacotes faltantes
        missing_pkgs = setdiff(required_pkgs, installed_pkgs)

        if !isempty(missing_pkgs)
            println("Instalando pacotes faltantes: ", missing_pkgs)
            Pkg.add(collect(missing_pkgs))
        else
            println("Todos os pacotes já estão instalados.")
        end
    else
        println("Criando novo ambiente Julia...")
        Pkg.activate(env_path, shared = true)
        Pkg.add(["CSV", "DataFrames", "Dates", "Statistics", "ZipFile", "Plots"])
    end
end

## setup_environment (generic function with 1 method)

# Executa setup
setup_environment()

## Criando novo ambiente Julia...

# Carrega pacotes
using CSV, DataFrames, Dates, ZipFile, Statistics, Plots

function getStats(chunk::DataFrame)::DataFrame

    # Filtra companhias aéreas de interesse
    major_airlines = ("AA", "DL", "UA", "US")
    filtered = filter(row -> row.AIRLINE in major_airlines, chunk)

    # Remove valores missing no atraso de chegada
```

```

    filtered = filter(row -> !ismissing(row.ARRIVAL_DELAY), filtered)

    # Calcula se o voo está atrasado (>10 minutos)
    filtered[!, :late] = filtered.ARRIVAL_DELAY .> 10

    return filtered
end

## getStats (generic function with 1 method)
function computeStats(processed_data)

    # Cria coluna de data
    processed_data[!, :Date] = Date.(
        string.(processed_data.YEAR, "-",
            lpad.(string.(processed_data.MONTH), 2, "0"), "-",
            lpad.(string.(processed_data.DAY), 2, "0"))
    )
    # Agrupa por companhia e data para calcular estatísticas
    gdf = groupby(processed_data, [:AIRLINE, :Date])
    result = combine(gdf) do group
        total_flights = nrow(group)
        delayed_flights = sum(group.late)
        perc_delayed = delayed_flights / total_flights
        return (Perc = perc_delayed,)
    end

    return result
end

## computeStats (generic function with 1 method)
function plot_calendar_heatmap(stats, airline_code)
    airline_data = filter(row -> row.AIRLINE == airline_code, stats)

    # Prepara dados para heatmap mensal
    months = 1:12
    p = plot(layout = (4, 3), size = (800, 600))

    for (i, month) in enumerate(months)
        month_data = filter(row -> month == Dates.month(row.Date), airline_data)

        if nrow(month_data) > 0
            heatmap_data = zeros(7, 5) # 7 dias da semana, 5 semanas máximas

            for row in eachrow(month_data)
                day_of_week = Dates.dayofweek(row.Date)
                week_of_month = ceil(Int, Dates.day(row.Date) / 7)
                heatmap_data[day_of_week, week_of_month] = row.Perc
            end

            # Paleta customizada - branco para 0, gradiente para valores > 0
            custom_palette = cgrad([
                RGB(1.0, 1.0, 1.0), # Branco puro para 0
                RGB(0.69, 0.85, 0.90), # Azul muito claro
            ])
        end
    end
end

```

```

        RGB(0.49, 0.75, 0.93), # Azul claro
        RGB(0.27, 0.46, 0.71), # Azul médio (#4575b4)
        RGB(0.18, 0.20, 0.28)  # (#473027)
    ])

    color_limits = (0.0, maximum(heatmap_data) > 0 ? maximum(heatmap_data) : 0.1)

    heatmap!(subplot = i, heatmap_data,
              title = Dates.monthname(month),
              color = custom_palette,
              clim = color_limits,
              xticks = false,          # Remove ticks do eixo X
              yticks = (1:7, ["Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sáb"]))

    end
end
return p
end

```

```
## plot_calendar_heatmap (generic function with 1 method)
```

```

function computeMoreStats(stats)
    # Primeiro, filtra apenas as companhias de interesse
    stats_filtrado = filter(row -> row.AIRLINE in ["AA", "DL", "UA", "US"], stats)

    # Agrupa e calcula estatísticas
    return combine(
        groupby(stats_filtrado, :AIRLINE),
        :Perc => (x -> round(mean(x) * 100, digits=2)) => :Atraso_Medio,
        :Perc => (x -> round(maximum(x) * 100, digits=2)) => :Atraso_Maximo
    )
end

```

```
## computeMoreStats (generic function with 1 method)
```

```

# Especificação de tipos para leitura mais rápida
types_dict = Dict{
    :DAY => Int32,
    :MONTH => Int32,
    :YEAR => Int32,
    :AIRLINE => String,
    :ARRIVAL_DELAY => Union{Int32, Missing}
};

# Leitura apenas das colunas necessárias
selected_cols = ["DAY", "MONTH", "YEAR", "AIRLINE", "ARRIVAL_DELAY"];

# z
vetor_dataframes = [];
zip_archive = ZipFile.Reader(file0);
try
    csv_file = zip_archive.files[1]
    chunks = CSV.Chunks(csv_file, select = selected_cols, types = types_dict, ntasks = 5)
    for (i, chunk) in enumerate(chunks)
        df_chunk = DataFrame(chunk)
        println("Processando chunk #\$i (#\$nrow(df_chunk)) linhas")
    end
catch
    println("Erro ao ler o arquivo zip")
end

```

```

        push!(vetor_dataframes, getStats(df_chunk))
    end
finally
    close(zip_archive)
end

```

```

## Processando chunk #1 (#1175972 linhas)
## Processando chunk #2 (#1169190 linhas)
## Processando chunk #3 (#1168127 linhas)
## Processando chunk #4 (#1155813 linhas)
## Processando chunk #5 (#1149977 linhas)

```

```

# Agregando resultados
df_final = reduce(vcat, vetor_dataframes);

# Computando estatísticas
final_stats = computeStats(df_final);

# Ordenação final
sort!(final_stats, [:AIRLINE, :Date]);

# Gera gráficos para cada companhia
g1_julia = plot_calendar_heatmap(final_stats, "AA");
g2_julia = plot_calendar_heatmap(final_stats, "DL" );
g3_julia = plot_calendar_heatmap(final_stats, "UA");
g4_julia = plot_calendar_heatmap(final_stats, "US");

```

```
display(g1_julia)
```

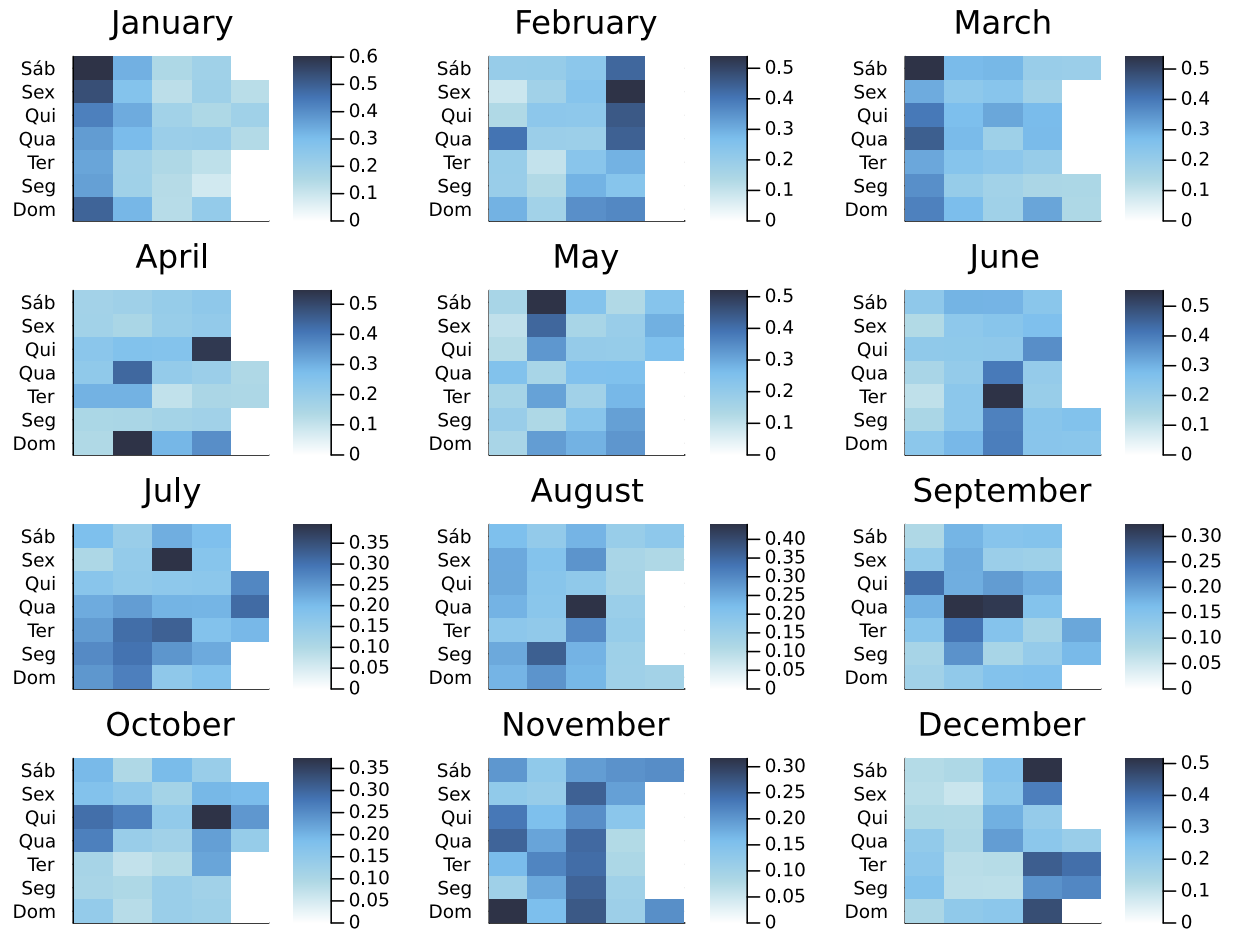


Figura: American Airlines (AA) - Julia

```
display(g2_julia)
```

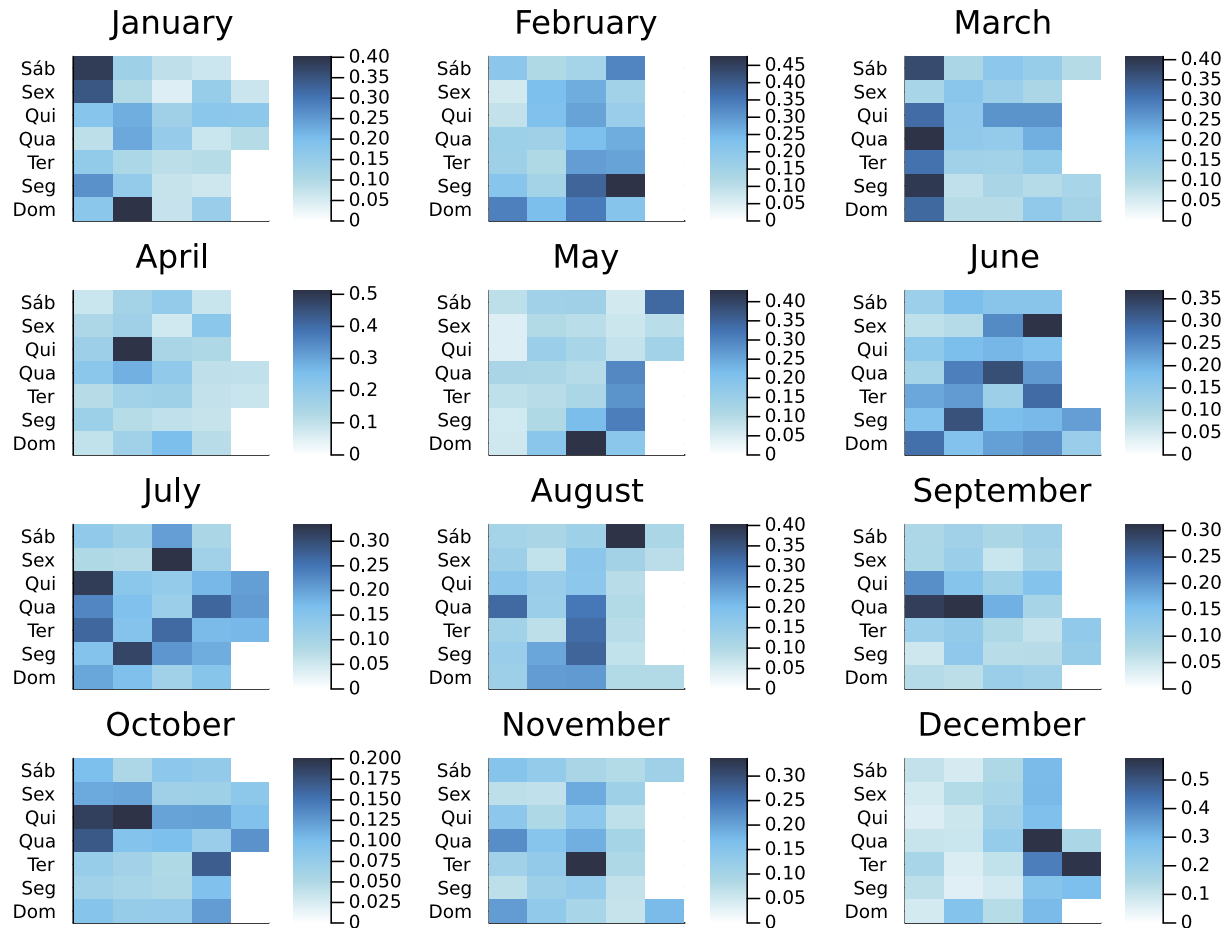


Figura: Delta Air Lines (DL) - Julia

```
display(g3_julia)
```

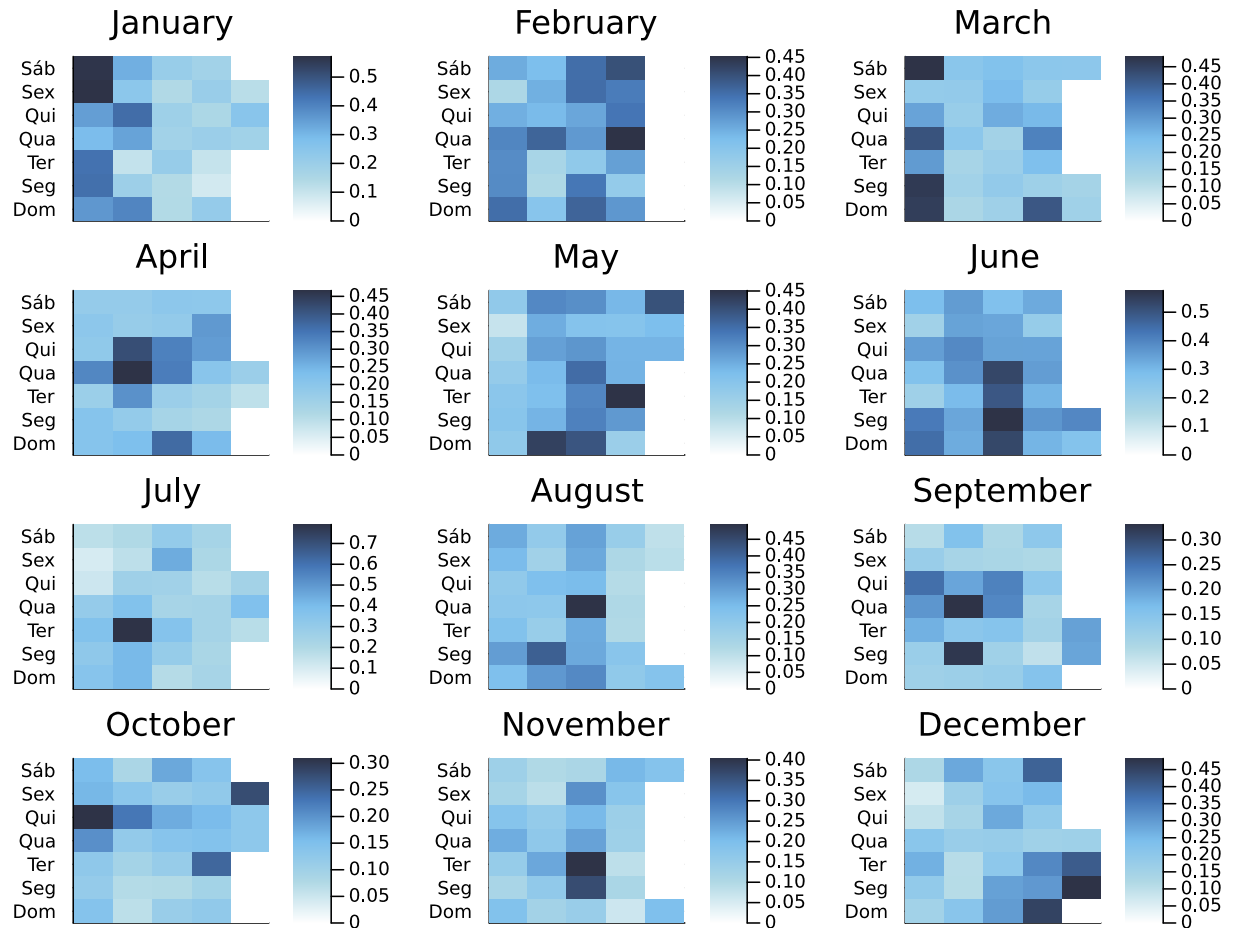


Figura: United Airlines (UA) - Julia

```
display(g4_julia)
```

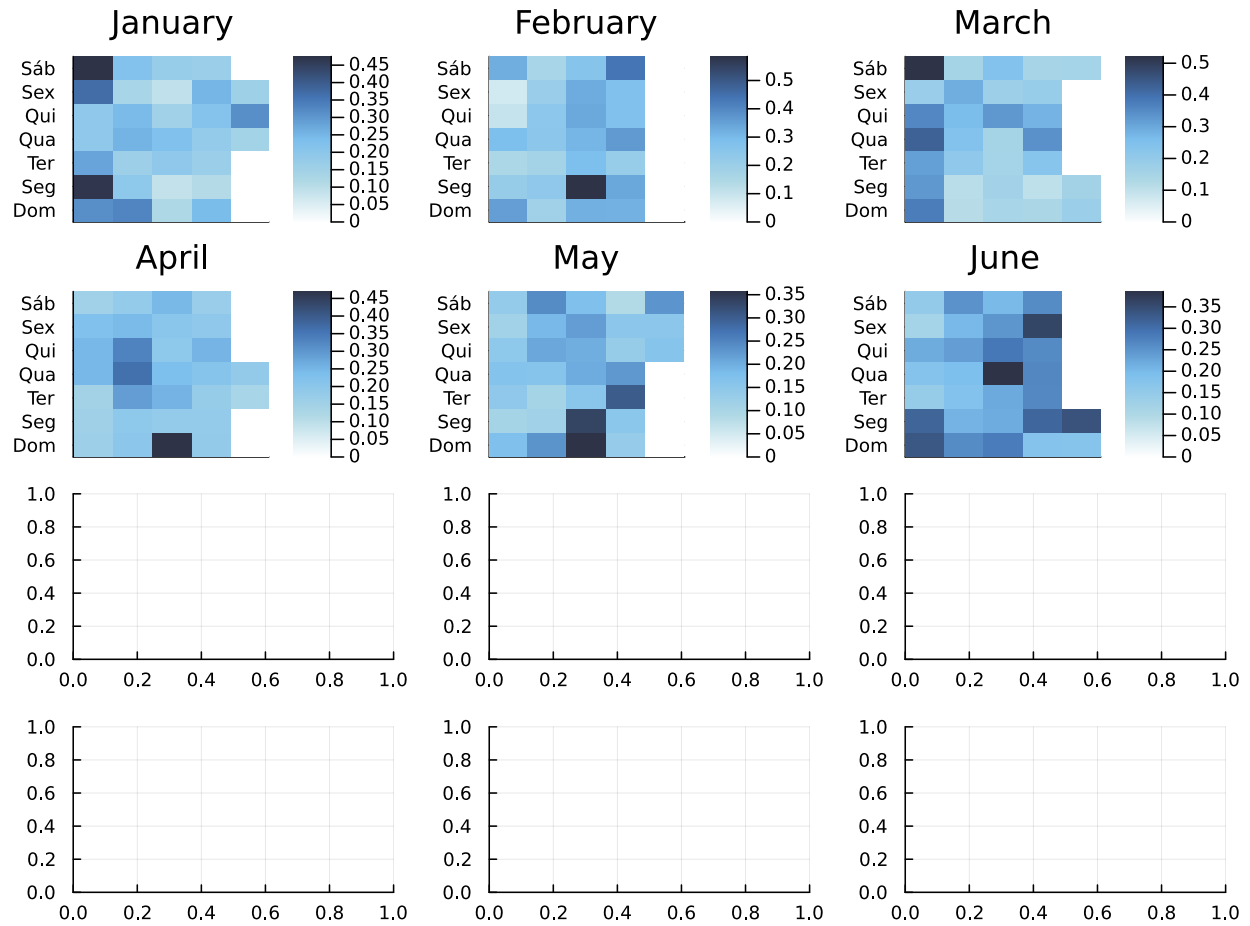


Figura: US Airways (US) - Julia

## § Resultados e Comparações (R–Julia)

Tabela 6: Média de vôos atrasados por Cia estudada e dia (Obtida através do Julia)

```
## Tabela kable()
in7 = julia_eval("final_stats")
in7 %>% my_kbl_foot(header = 10)
```

AIRLINE	Date	Perc
AA	2015-01-01	0.3727339
AA	2015-01-02	0.4303714
AA	2015-01-03	0.5382456
AA	2015-01-04	0.6021433
AA	2015-01-05	0.4912869
AA	2015-01-06	0.3656934
AA	2015-01-07	0.3549075
AA	2015-01-08	0.3074753
AA	2015-01-09	0.3422724
AA	2015-01-10	0.2797147

1-10 de 1276 linhas / 3 colunas

Tabela 3: Média de vôos atrasados por Cia estudada e dia.

```
## Tabela kable()
in4 %>% my_kbl_foot(header = 10)
```

AIRLINE	DATE	Perc
AA	2015-01-01	0.3727339
AA	2015-01-02	0.4303714
AA	2015-01-03	0.5382456
AA	2015-01-04	0.6021433
AA	2015-01-05	0.4912869
AA	2015-01-06	0.3656934
AA	2015-01-07	0.3549075
AA	2015-01-08	0.3074753
AA	2015-01-09	0.3422724
AA	2015-01-10	0.2797147

1-10 de 1276 linhas / 3 colunas

Tabela 7: Média de vôos atrasados por Cia estudada (Obtida através do Julia)

```
## Estatísticas extras
MoreStats =
  ↪ computeMoreStats(final_stats);
```

```
## Tabela kable()
in8 = julia_eval("MoreStats")
in8 %>% my_kbl()
```

AIRLINE	Atraso_Medio	Atraso_Maximo
AA	22.37	60.21
DL	16.26	57.51
UA	23.64	79.13
US	22.85	58.49

$$\Delta t = 132.365192174911 \text{ s}$$