

Título do capítulo	CAPÍTULO 5 MANIPULAÇÃO E VISUALIZAÇÃO DE DADOS GTFS
Autor(es)	Rafael H. M. Pereira Daniel Herszenhut
DOI	DOI: http://dx.doi.org/10.38116/9786556350547cap5

Título do livro	Introdução à Acessibilidade Urbana: um guia prático em R
Organizadores(as)	Rafael H. M. Pereira Daniel Herszenhut
Volume	1
Série	-
Cidade	Rio de Janeiro
Editora	Instituto de Pesquisa Econômica Aplicada (Ipea)
Ano	2023
Edição	1a
ISBN	9786556350547
DOI	DOI: http://dx.doi.org/10.38116/9786556350547

© Instituto de Pesquisa Econômica Aplicada – ipea 2023

As publicações do Ipea estão disponíveis para *download* gratuito nos formatos PDF (todas) e EPUB (livros e periódicos). Acesso: <http://www.ipea.gov.br/portal/publicacoes>

As opiniões emitidas nesta publicação são de exclusiva e inteira responsabilidade dos autores, não exprimindo, necessariamente, o ponto de vista do Instituto de Pesquisa Econômica Aplicada ou do Ministério do Planejamento e Orçamento.

É permitida a reprodução deste texto e dos dados nele contidos, desde que citada a fonte. Reproduções para fins comerciais são proibidas.

5 MANIPULAÇÃO E VISUALIZAÇÃO DE DADOS GTFS

Usualmente, arquivos GTFS provenientes de fontes oficiais são utilizados para desenvolver análises e pesquisas que possuem diversos elementos comuns. Visando facilitar a leitura, o processamento e a análise desses dados, a equipe do projeto AOP vem desenvolvendo o pacote de R `{gtfstools}`, que disponibiliza diversas funções que facilitam a manipulação e a exploração de *feeds*.

Neste capítulo, passaremos por algumas das funcionalidades mais frequentemente utilizadas do pacote. Para isso, vamos utilizar uma amostra do *feed* da SPTrans apresentado no capítulo anterior, disponível dentro do `{gtfstools}`.

5.1 Leitura e manipulação básica de arquivos GTFS

A leitura de arquivos GTFS com o `{gtfstools}` é feita com a função `read_gtfs()`, que recebe uma *string* com o caminho do arquivo. Após sua leitura, o *feed* é representado como uma lista de `data.tables`, uma versão de alta *performance* da classe `data.frame`. Ao longo deste capítulo, vamos nos referir a essa lista de tabelas como um *objeto GTFS*. Por padrão, a função a seguir lê todas as tabelas `.txt` do *feed*:

```
# carrega biblioteca
library(gtfstools)

# aponta para o endereço do arquivo gtfs dentro do {gtfstools}
endereco <- system.file("extdata/spo_gtfs.zip", package =
"gtfstools")

# le o gtfs
gtfs <- read_gtfs(endereco)

# consulta o nome das tabelas dentro da lista
names(gtfs)
```

```
[1] "agency"      "calendar"    "frequencies" "routes"      "shapes"
[6] "stop_times" "stops"       "trips"
```

Como podemos ver, cada `data.table` dentro do objeto GTFS é nomeado de acordo com a tabela que ele representa, sem a extensão `.txt`. Isso nos permite selecionar e manipular cada uma das tabelas separadamente. O código adiante, por exemplo, mostra os seis primeiros registros da tabela `trips`:

```
head(gtfs$strips)
```

	route_id	service_id	trip_id	trip_headsign	direction_id	shape_id
1:	CPTM L07	USD	CPTM L07-0	JUNDIAI	0	17846
2:	CPTM L07	USD	CPTM L07-1	LUZ	1	17847
3:	CPTM L08	USD	CPTM L08-0	AMADOR BUENO	0	17848
4:	CPTM L08	USD	CPTM L08-1	JULIO PRESTES	1	17849
5:	CPTM L09	USD	CPTM L09-0	GRAJAU	0	17850
6:	CPTM L09	USD	CPTM L09-1	OSASCO	1	17851

As tabelas dentro de um objeto GTFS podem ser facilmente manipuladas usando a sintaxe dos pacotes `{dplyr}` ou `{data.table}`. Neste livro, optamos por utilizar a sintaxe do `{data.table}`, pois esse pacote oferece diversas funcionalidades para a manipulação de tabelas com grande quantidade de registros, tal como a edição de colunas por referência, filtros de linhas muito rápidos e agregação de dados eficiente.¹⁵ Para adicionar cem segundos a todos os *headways* listados na tabela `frequencies` e reverter essa mudança em seguida, por exemplo, podemos usar o código a seguir:

```
# salva o headway original
headway_original <- gtfs$frequencies$headway_secs
head(gtfs$frequencies, 3)
```

	trip_id	start_time	end_time	headway_secs
1:	CPTM L07-0	04:00:00	04:59:00	720
2:	CPTM L07-0	05:00:00	05:59:00	360
3:	CPTM L07-0	06:00:00	06:59:00	360

```
# modifica o headway
gtfs$frequencies[, headway_secs := headway_secs + 100]
head(gtfs$frequencies, 3)
```

	trip_id	start_time	end_time	headway_secs
1:	CPTM L07-0	04:00:00	04:59:00	820
2:	CPTM L07-0	05:00:00	05:59:00	460
3:	CPTM L07-0	06:00:00	06:59:00	460

```
# restitui o headway original
gtfs$frequencies[, headway_secs := headway_original]
head(gtfs$frequencies, 3)
```

15. Mais detalhes sobre o uso e a sintaxe do `{data.table}` estão disponíveis em: <https://rdatatable.gitlab.io/data.table/index.html>.

	trip_id	start_time	end_time	headway_secs
1:	CPTM L07-0	04:00:00	04:59:00	720
2:	CPTM L07-0	05:00:00	05:59:00	360
3:	CPTM L07-0	06:00:00	06:59:00	360

Após editarmos um objeto GTFS no R, frequentemente vamos querer utilizá-lo para fazer análises de diferentes tipos. Para isso, é comum que precisemos do arquivo GTFS em formato .zip novamente, e não como uma lista de tabelas dentro do R. O pacote {gtfstools} disponibiliza a função `write_gtfs()` exatamente com a finalidade de transformar objetos GTFS que existem apenas dentro do R em arquivos GTFS armazenados no seu computador. Para usarmos essa função, precisamos apenas listar o objeto e o endereço no qual o arquivo deve ser salvo:

```
# aponta para o endereço onde arquivo deve ser salvo
endereco_destino <- tempfile("novo_gtfs", fileext = ".zip")

# salva o GTFS no endereço
write_gtfs(gtfs, path = endereco_destino)

# lista arquivos dentro do feed recém-salvo
zip::zip_list(endereco_destino)[, c("filename", "compressed_size", "timestamp")]
```

	filename	compressed_size	timestamp
1	agency.txt	112	2023-06-13 21:17:54
2	calendar.txt	129	2023-06-13 21:17:54
3	frequencies.txt	2381	2023-06-13 21:17:54
4	routes.txt	659	2023-06-13 21:17:54
5	shapes.txt	160470	2023-06-13 21:17:54
6	stop_times.txt	7907	2023-06-13 21:17:54
7	stops.txt	18797	2023-06-13 21:17:54
8	trips.txt	717	2023-06-13 21:17:54

5.2 Cálculo de velocidade das linhas

Arquivos GTFS são frequentemente utilizados em estimativas de roteamento de transporte público e para informar passageiros sobre a tabela de horários das diferentes rotas que operam em uma região. Dessa forma, é extremamente importante que o cronograma das viagens e a velocidade operacional de cada linha estejam adequadamente descritos no *feed*.

O `{gtfstools}` disponibiliza a função `get_trip_speed()` para facilitar o cálculo da velocidade de cada viagem presente no *feed*. Por padrão, a função calcula a velocidade (em km/h) de todas as viagens do objeto GTFS, mas viagens individuais também podem ser especificadas:

```
# calcula a velocidade de todas as viagens
velocidades <- get_trip_speed(gtfs)

head(velocidades)

  trip_id origin_file    speed
1: 2002-10-0    shapes  8.952511
2: 2105-10-0    shapes 10.253365
3: 2105-10-1    shapes  9.795292
4: 2161-10-0    shapes 11.182534
5: 2161-10-1    shapes 11.784458
6: 4491-10-0    shapes 13.203560

nrow(velocidades)

[1] 36

# calcula a velocidade de duas viagens específicas
velocidades <- get_trip_speed(gtfs, trip_id = c("CPTM L07-0",
"2002-10-0"))

velocidades

  trip_id origin_file    speed
1: 2002-10-0    shapes  8.952511
2: CPTM L07-0    shapes 26.787768
```

Calcular a velocidade de uma viagem requer que saibamos o seu comprimento e em quanto tempo ela foi realizada. Para isso, a `get_trip_speed()` utiliza duas outras funções do `{gtfstools}` por trás dos panos: a `get_trip_length()` e a `get_trip_duration()`. O funcionamento das duas é muito parecido com o mostrado anteriormente, calculando o comprimento/duração de todas as viagens por padrão, ou de apenas algumas selecionadas, caso desejado. A seguir, mostramos seus comportamentos padrões.

```
# calcula a distância percorrida de todas viagens
distancias <- get_trip_length(gtfs, file = "shapes")
```

```
head(distancias)
```

```

      trip_id  length origin_file
1: CPTM L07-0 60.71894      shapes
2: CPTM L07-1 60.71894      shapes
3: CPTM L08-0 41.79037      shapes
4: CPTM L08-1 41.79037      shapes
5: CPTM L09-0 31.88906      shapes
6: CPTM L09-1 31.88906      shapes

```

```
# calcula a duração de todas viagens
duracao <- get_trip_duration(gtfs)
```

```
head(duracao)
```

```

      trip_id duration
1: 2002-10-0      48
2: 2105-10-0     108
3: 2105-10-1     111
4: 2161-10-0      94
5: 2161-10-1      93
6: 4491-10-0      69

```

Assim como a `get_trip_speed()` calcula as velocidades em km/h por padrão, a `get_trip_length()` e a `get_trip_duration()` calculam os comprimentos e as durações em quilômetros e em minutos, respectivamente. Essas unidades podem ser ajustadas com o argumento `unit`, presente nas três funções.

5.3 Combinando e filtrando *feeds*

Muitas vezes, o processamento e a edição de arquivos GTFS são realizados, em grande medida, manualmente. Por isso, pequenas inconsistências podem passar batidas pelos responsáveis por esse processamento. Um problema comumente observado em *feeds* é a presença de registros duplicados em uma mesma tabela. O *feed* da SPTrans, por exemplo, possui registros duplicados tanto no `agency.txt` quanto no `calendar.txt`:

```
gtfs$agency
```

```

agency_id  agency_name          agency_url
1:         1      SPTRANS http://www.sptrans.com.br/?versao=011019
2:         1      SPTRANS http://www.sptrans.com.br/?versao=011019

```

```

      agency_timezone agency_lang
1: America/Sao_Paulo          pt
2: America/Sao_Paulo          pt

```

```
gtfs$calendar
```

```

      service_id monday tuesday wednesday thursday friday saturday sunday
1:      USD      1      1      1      1      1      1      1
2:      U__      1      1      1      1      1      0      0
3:      US_      1      1      1      1      1      1      0
4:      _SD      0      0      0      0      0      1      1
5:      __D      0      0      0      0      0      0      1
6:      _S_      0      0      0      0      0      1      0
7:      USD      1      1      1      1      1      1      1
8:      U__      1      1      1      1      1      0      0
9:      US_      1      1      1      1      1      1      0
10:     _SD      0      0      0      0      0      1      1
11:     __D      0      0      0      0      0      0      1
12:     _S_      0      0      0      0      0      1      0

      start_date  end_date
1: 2008-01-01 2020-05-01
2: 2008-01-01 2020-05-01
3: 2008-01-01 2020-05-01
4: 2008-01-01 2020-05-01
5: 2008-01-01 2020-05-01
6: 2008-01-01 2020-05-01
7: 2008-01-01 2020-05-01
8: 2008-01-01 2020-05-01
9: 2008-01-01 2020-05-01
10: 2008-01-01 2020-05-01
11: 2008-01-01 2020-05-01
12: 2008-01-01 2020-05-01

```

O `{gtfstools}` disponibiliza a função `remove_duplicates()` para remover essas duplicatas. Essa função recebe como *input* um objeto GTFS e retorna o mesmo objeto, porém sem registros duplicados:

```

# remove valores duplicados
gtfs_sem_dups <- remove_duplicates(gtfs)

gtfs_sem_dups$agency

```

```

agency_id  agency_name                                agency_url
1:         1      SPTRANS http://www.sptrans.com.br/?versao=011019
          agency_timezone  agency_lang
1: America/Sao_Paulo                pt

```

```
gtfs_sem_dups$calendar
```

```

service_id monday tuesday wednesday thursday friday saturday sunday
1:      USD      1      1          1          1      1          1      1
2:      U__      1      1          1          1      1          0      0
3:      US_      1      1          1          1      1          1      0
4:      _SD      0      0          0          0      0          1      1
5:      __D      0      0          0          0      0          0      1
6:      _S_      0      0          0          0      0          1      0
  start_date  end_date
1: 2008-01-01 2020-05-01
2: 2008-01-01 2020-05-01
3: 2008-01-01 2020-05-01
4: 2008-01-01 2020-05-01
5: 2008-01-01 2020-05-01
6: 2008-01-01 2020-05-01

```

Frequentemente, também, lidamos com múltiplos *feeds* em uma mesma área de estudo. Por exemplo, quando os dados dos sistemas de ônibus e de trens de uma mesma cidade estão salvos em arquivos GTFS separados. Nesse caso, muitas vezes gostaríamos de uni-los em um único arquivo, diminuindo assim o esforço de manipulação e processamento dos dados. Para isso, o {gtfstools} disponibiliza a função `merge_gtfs()`. O exemplo a seguir mostra o resultado da combinação de dois *feeds* distintos, o da SPTrans (sem duplicatas) e o da EPTC, de Porto Alegre:

```

# lê GTFS de Porto Alegre
endereco_poa <- system.file("extdata/poa_gtfs.zip", package =
"gtfstools")
gtfs_poa <- read_gtfs(endereco_poa)

gtfs_poa$agency

agency_id
1:      EPTC

agency_name
1: Empresa Publica de Transportes e Circulação

```



```

          agency_url      agency_timezone
1: http://www.eptc.com.br America/Sao_Paulo
  agency_lang agency_phone
1:          pt          156

          agency_fare_url
1: http://www2.portoalegre.rs.gov.br/eptc/default.php?p_secao=155

```

```
gtfs_sem_dups$agency
```

```

  agency_id agency_name      agency_url
1:         1   SPTRANS http://www.sptrans.com.br/?versao=011019
  agency_timezone agency_lang
1: America/Sao_Paulo      pt

```

```

# combina objetos GTFS de Porto Alegre e São Paulo
gtfs_combinado <- merge_gtfs(gtfs_sem_dups, gtfs_poa)

```

```

# checa resultados
gtfs_combinado$agency

```

```

  agency_id      agency_name
1:         1      SPTRANS
2:      EPTC Empresa Publica de Transportes e Circulação
  agency_url      agency_timezone agency_lang
1: http://www.sptrans.com.br/?versao=011019 America/Sao_Paulo      pt
2:          http://www.eptc.com.br America/Sao_Paulo      pt
  agency_phone      agency_fare_url
1:
2:          156 http://www2.portoalegre.rs.gov.br/eptc/default.php?p_secao=155

```

Como podemos ver, os registros das tabelas de ambos os *feeds* foram combinados em uma única tabela. Esse é o caso quando os dois (ou mais, se desejado) objetos GTFS possuem registros de uma mesma tabela (a *agency*, no exemplo). Caso apenas um dos objetos possua uma das tabelas, a operação copia essa tabela para o resultado final. É o caso, por exemplo, da tabela *frequencies*, que existe no *feed* da SPTrans, mas não no da EPTC:

```
names(gtfs_poa)
```

```

[1] "agency" "calendar" "routes" "shapes" "stop_times"
[6] "stops" "trips"

```

```
names(gtfs_sem_dups)

[1] "agency"      "calendar"    "frequencies" "routes"      "shapes"
[6] "stop_times" "stops"       "trips"

names(gtfs_combinado)

[1] "agency"      "calendar"    "frequencies" "routes"      "shapes"
[6] "stop_times" "stops"       "trips"

identical(gtfs_sem_dups$frequencies, gtfs_combinado$frequencies)

[1] TRUE
```

Um outro tipo de operação muito utilizada no tratamento de dados GTFS é o de filtragem desses arquivos. Frequentemente, *feeds* são usados para descrever redes de transporte público de grande escala, o que pode transformar sua edição, sua análise e seu compartilhamento em operações complexas. Por esse motivo, pesquisadores e planejadores muitas vezes precisam trabalhar com um subconjunto de dados descritos nos *feeds*. Por exemplo, caso desejemos estimar a *performance* da rede de transporte em uma determinada região no horário de pico da manhã, podemos filtrar o nosso arquivo GTFS de modo a manter apenas os registros referentes a viagens que ocorrem nesse intervalo do dia.

O pacote `{gtfstools}` traz diversas funções para facilitar a filtragem de arquivos GTFS. São elas:

- `filter_by_agency_id()`;
- `filter_by_route_id()`;
- `filter_by_service_id()`;
- `filter_by_shape_id()`;
- `filter_by_stop_id()`;
- `filter_by_trip_id()`;
- `filter_by_route_type()`;
- `filter_by_weekday()`;
- `filter_by_time_of_day()`; e
- `filter_by_sf()`.

5.3.1 Filtro por identificadores

As sete primeiras funções mencionadas anteriormente são utilizadas de forma muito similar. Devemos especificar um vetor de identificadores que é usado para manter no objeto GTFS apenas os registros relacionados a esses identificadores. O exemplo a seguir demonstra essa funcionalidade com a `filter_by_trip_id()`:

```
# checa tamanho do feed antes do filtro
utils::object.size(gtfs)
```

864568 bytes

```
head(gtfs$trips[, .(trip_id, trip_headsign, shape_id)])
```

	trip_id	trip_headsign	shape_id
1:	CPTM L07-0	JUNDIAI	17846
2:	CPTM L07-1	LUZ	17847
3:	CPTM L08-0	AMADOR BUENO	17848
4:	CPTM L08-1	JULIO PRESTES	17849
5:	CPTM L09-0	GRAJAU	17850
6:	CPTM L09-1	OSASCO	17851

```
# mantém apenas registros relacionados a duas viagens
gtfs_filtrado <- filter_by_trip_id(
  gtfs,
  trip_id = c("CPTM L07-0", "CPTM L07-1")
)
```

```
# checa tamanho do feed após o filtro
utils::object.size(gtfs_filtrado)
```

71592 bytes

```
head(gtfs_filtrado$trips[, .(trip_id, trip_headsign, shape_id)])
```

	trip_id	trip_headsign	shape_id
1:	CPTM L07-0	JUNDIAI	17846
2:	CPTM L07-1	LUZ	17847

```
unique(gtfs_filtrado$shapes$shape_id)
```

```
[1] "17846" "17847"
```

O código mostra que a função não filtra apenas a tabela `trips`, mas também as outras tabelas que possuem algum tipo de relação com os identificadores especificados. Por exemplo, a trajetória das viagens CPTM L07-0 e CPTM L07-1 é descrita pelos `shape_ids` 17846 e 17847, respectivamente. Esses são, portanto, os únicos identificadores da tabela `shapes` mantidos no objeto GTFS filtrado.

A função também pode funcionar com o comportamento diametralmente oposto: em vez de definirmos os identificadores cujos registros devem ser *mantidos* no *feed*, podemos especificar os identificadores que devem ser *retirados* dele. Para isso, usamos o argumento `keep` com valor `FALSE`:

```
# remove duas viagens do feed
gtfs_filtrado <- filter_by_trip_id(
  gtfs,
  trip_id = c("CPTM L07-0", "CPTM L07-1"),
  keep = FALSE
)

head(gtfs_filtrado$trips[, .(trip_id, trip_headsign, shape_id)])
```

	trip_id	trip_headsign	shape_id
1:	CPTM L08-0	AMADOR BUENO	17848
2:	CPTM L08-1	JULIO PRESTES	17849
3:	CPTM L09-0	GRAJAU	17850
4:	CPTM L09-1	OSASCO	17851
5:	CPTM L10-0	RIO GRANDE DA SERRA	17852
6:	CPTM L10-1	BRÁS	17853

```
head(unique(gtfs_filtrado$shapes$shape_id))
```

```
[1] "17848" "17849" "17850" "17851" "17852" "17853"
```

Como podemos ver, as viagens especificadas, bem como suas trajetórias, não estão presentes no objeto GTFS filtrado. A mesma lógica aqui demonstrada com a `filter_by_trip_id()` é válida para as funções que filtram objetos GTFS pelos identificadores `agency_id`, `route_id`, `service_id`, `shape_id`, `stop_id` e `route_type`.

5.3.2 Filtro por dia e hora

Outra operação que recorrentemente aparece em análises que envolvem dados GTFS é a de manter serviços que funcionem apenas em determinados horários do dia ou dias da semana. Para isso, o pacote disponibiliza as funções `filter_by_weekday()` e `filter_by_time_of_day()`.

A `filter_by_weekday()` recebe os dias da semana (em inglês) cujos serviços que neles operam devem ser mantidos. Adicionalmente, a função também inclui o argumento `combine`, que define como filtros de dois ou mais dias funcionam. Quando este recebe o valor "and", apenas serviços que operam em todos os dias especificados são mantidos. Quando recebe o valor "or", serviços que operam em pelo menos um dos dias são mantidos:

```
# mantém apenas serviços que operam no sábado E no domingo
gtfs_filtrado <- filter_by_weekday(
  gtfs = gtfs_sem_dups,
  weekday = c("saturday", "sunday"),
  combine = "and"
)

gtfs_filtrado$calendar[, c("service_id", "sunday", "saturday")]
```

```
service_id sunday saturday
1:      USD      1         1
2:      _SD      1         1
```

```
# mantém apenas serviços que operam OU no sábado OU no domingo
gtfs_filtrado <- filter_by_weekday(
  gtfs = gtfs_sem_dups,
  weekday = c("sunday", "saturday"),
  combine = "or"
)

gtfs_filtrado$calendar[, c("service_id", "sunday", "saturday")]
```

```
service_id sunday saturday
1:      USD      1         1
2:      US_      0         1
3:      _SD      1         1
4:      __D      1         0
5:      _S_      0         1
```

A `filter_by_time_of_day()`, por sua vez, recebe o começo e o final de uma janela de tempo e mantém os registros relacionados a viagens que rodam dentro dessa janela. O funcionamento da função depende da presença ou não da tabela `frequencies` no objeto GTFS: o cronograma descrito na `stop_times` das viagens listadas na tabela `frequencies` não deve ser filtrado, pois, como comentado no capítulo anterior, ele serve como um modelo que dita o tempo de viagem entre uma parada e outra. Caso a `frequencies` esteja ausente, no entanto, a `stop_times` é filtrada segundo o intervalo de tempo especificado. Vamos ver como isso funciona com um exemplo:

```
# mantém apenas viagens dentro do período de 5 às 6 da manhã
gtfs_filtrado <- filter_by_time_of_day(gtfs, from = "05:00:00",
to = "06:00:00")
```

```
head(gtfs_filtrado$frequencies)
```

```
      trip_id start_time end_time headway_secs
1: CPTM L07-0 05:00:00 05:59:00          360
2: CPTM L07-1 05:00:00 05:59:00          360
3: CPTM L08-0 05:00:00 05:59:00          480
4: CPTM L08-1 05:00:00 05:59:00          480
5: CPTM L09-0 05:00:00 05:59:00          480
6: CPTM L09-1 05:00:00 05:59:00          480
```

```
head(gtfs_filtrado$stop_times[, c("trip_id", "departure_time",
"arrival_time")])
```

```
      trip_id departure_time arrival_time
1: CPTM L07-0      04:00:00      04:00:00
2: CPTM L07-0      04:08:00      04:08:00
3: CPTM L07-0      04:16:00      04:16:00
4: CPTM L07-0      04:24:00      04:24:00
5: CPTM L07-0      04:32:00      04:32:00
6: CPTM L07-0      04:40:00      04:40:00
```

```
# salva a tabela frequencies e a remove do objeto gtfs
frequencies <- gtfs$frequencies
gtfs$frequencies <- NULL
```

```
gtfs_filtrado <- filter_by_time_of_day(gtfs, from = "05:00:00",
to = "06:00:00")
```

```
head(gtfs_filtrado$stop_times[, c("trip_id", "departure_time",
"arrival_time")])
```

	trip_id	departure_time	arrival_time
1:	CPTM L07-0	05:04:00	05:04:00
2:	CPTM L07-0	05:12:00	05:12:00
3:	CPTM L07-0	05:20:00	05:20:00
4:	CPTM L07-0	05:28:00	05:28:00
5:	CPTM L07-0	05:36:00	05:36:00
6:	CPTM L07-0	05:44:00	05:44:00

O filtro da tabela `stop_times` pode funcionar de duas formas distintas: mantendo intactas todas as *viagens* que *cruzam* a janela de tempo especificada; ou mantendo no cronograma apenas as *paradas* que são visitadas *dentro* da janela (comportamento padrão da função). Esse comportamento é controlado com o parâmetro `full_trips`, como mostrado a seguir (atenção aos horários e aos segmentos presentes em cada exemplo):

```
# mantém apenas viagens inteiramente dentro do período de 5 às 6 da manhã
gtfs_filtrado <- filter_by_time_of_day(
  gtfs,
  from = "05:00:00",
  to = "06:00:00",
  full_trips = TRUE
)

head(
  gtfs_filtrado$stop_times[
    ,
    c("trip_id", "departure_time", "arrival_time", "stop_sequence")
  ]
)
```

	trip_id	departure_time	arrival_time	stop_sequence
1:	CPTM L07-0	04:00:00	04:00:00	1
2:	CPTM L07-0	04:08:00	04:08:00	2
3:	CPTM L07-0	04:16:00	04:16:00	3
4:	CPTM L07-0	04:24:00	04:24:00	4
5:	CPTM L07-0	04:32:00	04:32:00	5
6:	CPTM L07-0	04:40:00	04:40:00	6

```
# mantém apenas paradas que são visitadas entre 5 e 6 da manhã
gtfs_filtrado <- filter_by_time_of_day(
  gtfs,
  from = "05:00:00",
  to = "06:00:00",
  full_trips = FALSE
)

head(
  gtfs_filtrado $stop_times[
    ,
    c("trip_id", "departure_time", "arrival_time", "stop_sequence")
  ]
)
```

	trip_id	departure_time	arrival_time	stop_sequence
1:	CPTM L07-0	05:04:00	05:04:00	9
2:	CPTM L07-0	05:12:00	05:12:00	10
3:	CPTM L07-0	05:20:00	05:20:00	11
4:	CPTM L07-0	05:28:00	05:28:00	12
5:	CPTM L07-0	05:36:00	05:36:00	13
6:	CPTM L07-0	05:44:00	05:44:00	14

5.3.3 Filtro espacial

Por fim, o {gtfstools} também disponibiliza uma função que permite filtrar o objeto GTFS usando um polígono espacial. A `filter_by_sf()` recebe um objeto do tipo `sf/sfc` (representação espacial criada pelo pacote {sf}), ou sua *bounding box*, e mantém os registros cujas viagens são selecionadas por uma operação espacial que também deve ser especificada. Embora aparentemente complicado, esse processo de filtragem é compreendido com facilidade quando apresentado visualmente. Para isso, vamos filtrar a GTFS da SPTrans pela *bounding box* da trajetória 68962. Com o código a seguir, apresentamos a distribuição espacial dos dados não filtrados, com a *bounding box* destacada em vermelho na figura 5.

```
# carrega biblioteca ggplot2 para visualização de dados
library(ggplot2)

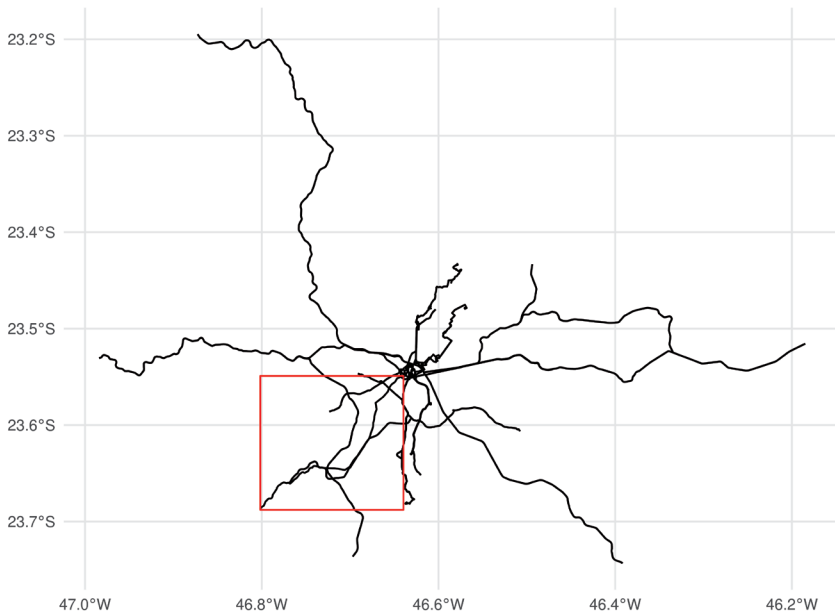
# cria poligono com a bounding box da trajetoria 68962
trajetoria_68962 <- convert_shapes_to_sf(gtfs, shape_id = "68962")
bbox <- sf::st_bbox(trajetoria_68962)
geometria_bbox <- sf::st_as_sfc(bbox)

# gera geometria de todas as trajetorias do gtfs
todas_as_trajetorias <- convert_shapes_to_sf(gtfs)
```



```
ggplot() +
  geom_sf(data = todas_as_trajetorias) +
  geom_sf(data = geometria_bbox, fill = NA, color = "red") +
  theme_minimal()
```

FIGURA 5
Distribuição espacial das trajetórias com a *bounding box* da trajetória 68962 em destaque



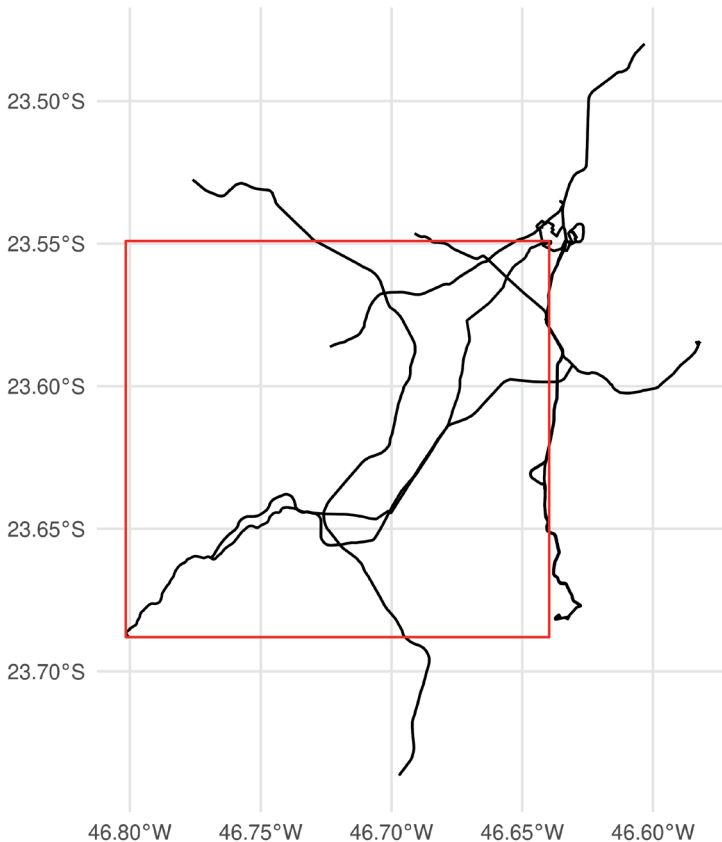
Fonte: Figura gerada pelo código supracitado.

Note que usamos a função `convert_shapes_to_sf()`, também disponibilizada pelo `{gtfstools}`, que converte uma determinada trajetória descrita no arquivo GTFS em um objeto espacial do tipo `sf`. Por padrão, a `filter_by_sf()` mantém os dados relacionados aos registros de viagens cujas trajetórias possuem alguma interseção com o polígono espacial selecionado:

```
gtfs_filtrado <- filter_by_sf(gtfs, bbox)
trajetorias_filtradas <- convert_shapes_to_sf(gtfs_filtrado)

ggplot() +
  geom_sf(data = trajetorias_filtradas) +
  geom_sf(data = geometria_bbox, fill = NA, color = "red") +
  theme_minimal()
```

FIGURA 6
Distribuição espacial das trajetórias com interseções com a *bounding box* da trajetória 68962

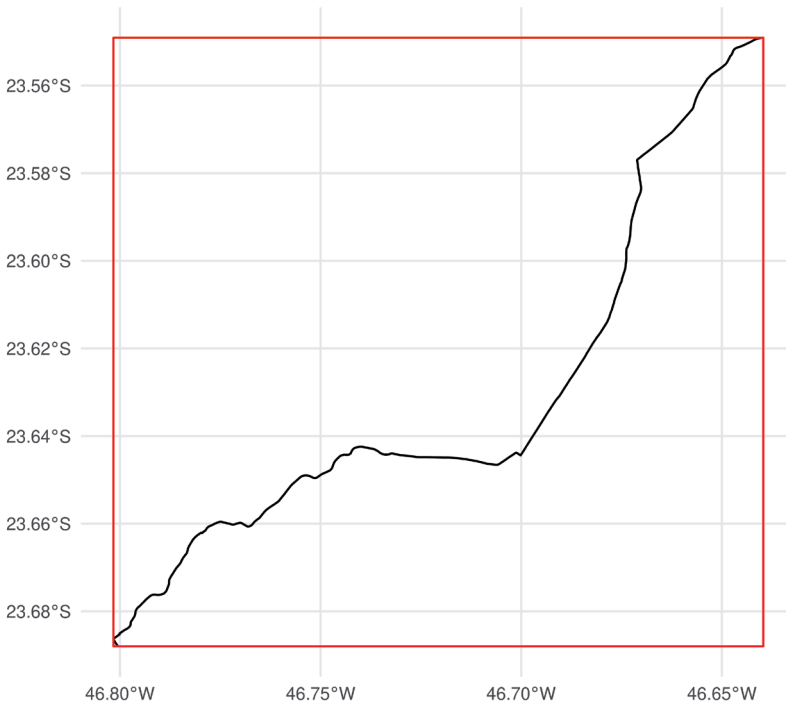


Fonte: Figura gerada pelo código supracitado.

Podemos, no entanto, controlar a operação espacial usada no processo de filtragem. Por exemplo, o código adiante mostra como podemos manter os dados relacionados a viagens que estão *contidas* dentro do polígono espacial:

```
gtfs_filtrado <- filter_by_sf(gtfs, bbox, spatial_operation =  
sf::st_contains)  
trajetorias_filtradas <- convert_shapes_to_sf(gtfs_filtrado)  
  
ggplot() +  
  geom_sf(data = trajetorias_filtradas) +  
  geom_sf(data = geometria_bbox, fill = NA, color = "red") +  
  theme_minimal()
```

FIGURA 7
Distribuição espacial das trajetórias contidas na *bounding box* da trajetória 68962



Fonte: Figura gerada pelo código supracitado.

5.4 Validação de arquivos GTFS

Planejadores e pesquisadores frequentemente querem avaliar a qualidade dos arquivos GTFS que estão utilizando em suas análises ou que estão produzindo. Os *feeds* estão estruturados conforme boas práticas adotadas pela comunidade que usa a especificação? As tabelas e colunas estão formatadas corretamente? A informação descrita pelo *feed* parece verossímil (velocidade das viagens, localização das paradas etc.)? Essas são algumas das perguntas que podem surgir ao lidar com dados no formato GTFS.

Para responder a essas e outras perguntas, o `{gtfstools}` inclui a função `validate_gtfs()`, que serve como interface entre o R e o Canonical GTFS Validator, *software* desenvolvido pela MobilityData. O uso do validador requer que o Java versão 11 ou superior esteja instalado.¹⁶

16. Para informações sobre como checar a versão do Java instalado em seu computador e como instalar a versão correta, caso necessário, conferir o capítulo 3.

Usar a `validate_gtfs()` é muito simples. Primeiro, precisamos baixar o *software* de validação. Para isso, podemos usar a função `download_validator()`, também disponível no pacote, que recebe o endereço de uma pasta na qual o validador deve ser salvo e a versão do validador desejada (por padrão, a mais recente). Como resultado, a função retorna o endereço do arquivo baixado:

```
pasta_temporaria <- tempdir()
endereco_validador <- download_validator(pasta_temporaria)
endereco_validador

[1] "/tmp/RtmpYawkxY/gtfs-validator-v4.0.0.jar"
```

A segunda (e última) etapa consiste em de fato rodar a função `validate_gtfs()`. Essa função aceita que os dados GTFS a serem validados sejam passados de diferentes formas: i) como um objeto GTFS existente apenas no R; ii) como o endereço de um arquivo GTFS salvo localmente em formato `.zip`; iii) como a URL para um *feed*; ou iv) como uma pasta que contém os dados GTFS não compactados. A função também recebe o endereço para uma pasta onde o resultado da validação deve ser salvo e o endereço para o validador que deve ser usado. Nesse exemplo, vamos fazer a validação a partir do endereço do arquivo GTFS da SPTrans, lido anteriormente:

```
pasta_resultado <- tempfile("validacao_com_endereco")
validate_gtfs(
  endereco,
  output_path = pasta_resultado,
  validator_path = endereco_validador
)
list.files(pasta_resultado)

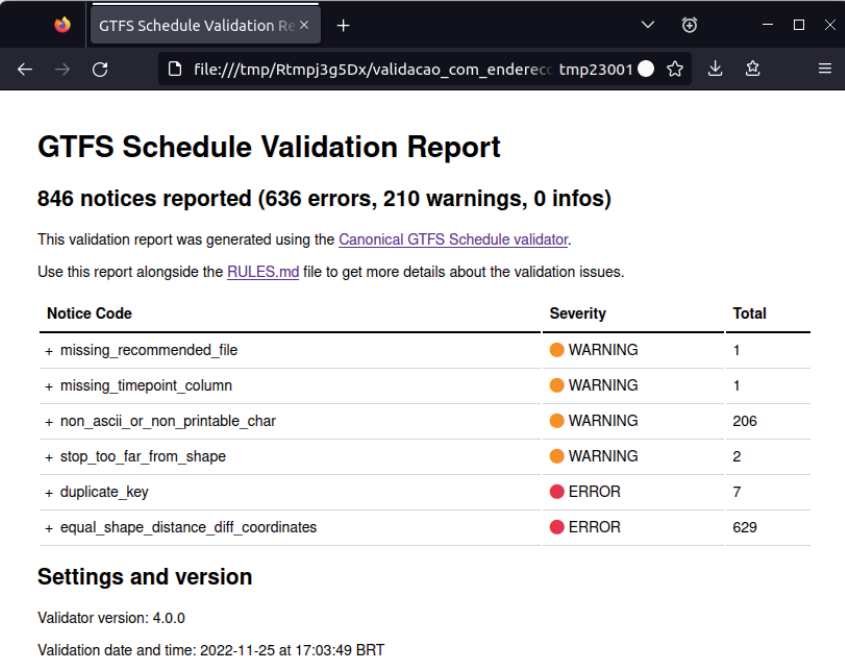
[1] "report.html"          "report.json"         "system_errors.json"
[4] "validation_stderr.txt"
```

Como podemos ver, a validação gera alguns arquivos como resultado:

- `report.html`, mostrado na figura 8, que resume os resultados da validação em uma página HTML (apenas disponível quando utilizado o validador v3.1.0 ou superior);
- `report.json`, que resume a mesma informação apresentada na página HTML, porém em formato JSON, que pode ser usado para processar e interpretar os resultados de forma programática;

- `system_errors.json`, que apresenta eventuais erros de sistema que tenham ocorrido durante a validação e que podem comprometer os resultados; e
- `validation_stderr.txt`, que lista mensagens informativas enviadas pelo validador, incluindo uma lista dos testes realizados, eventuais mensagens de erro etc.¹⁷

FIGURA 8
Exemplo de relatório gerado durante a validação



GTFS Schedule Validation Report

846 notices reported (636 errors, 210 warnings, 0 infos)

This validation report was generated using the [Canonical GTFS Schedule validator](#).

Use this report alongside the [RULES.md](#) file to get more details about the validation issues.

Notice Code	Severity	Total
+ missing_recommended_file	WARNING	1
+ missing_timepoint_column	WARNING	1
+ non_ascii_or_non_printable_char	WARNING	206
+ stop_too_far_from_shape	WARNING	2
+ duplicate_key	ERROR	7
+ equal_shape_distance_diff_coordinates	ERROR	629

Settings and version

Validator version: 4.0.0

Validation date and time: 2022-11-25 at 17:03:49 BRT

Elaboração dos autores.

Obs.: Figura cujos leiaute e textos não puderam ser padronizados e revisados em virtude das condições técnicas dos originais (nota do Editorial).

5.5 Fluxo de trabalho com o `{gtfstools}`: mapeando o *headway* das linhas

Como mostrado nas seções anteriores, o `{gtfstools}` disponibiliza uma grande caixa de ferramentas que podem ser usadas no processamento e na análise de arquivos GTFS. O pacote, no entanto, oferece diversas outras funções que não puderam ser apresentadas neste livro, por limitação de espaço.¹⁸

17. Mensagens informativas podem também ser listadas no arquivo `validation_stdout.txt`. As mensagens listadas no `validation_stderr.txt` e no `validation_stdout.txt` dependem da versão do validador utilizada.

18. A lista completa de funções está disponível em: <https://ipeagit.github.io/gtfstools/reference/index.html>.

A apresentação das funções feita até aqui tem um importante caráter demonstrativo, porém não mostra como elas podem ser usadas de forma conjunta na análise de um arquivo GTFS. Esta seção preenche essa lacuna, mostrando como o pacote pode ser usado, por exemplo, para responder à seguinte pergunta: como se distribuem espacialmente os tempos entre veículos de uma mesma linha (os *headways*) no arquivo GTFS da SPTrans?

A primeira etapa é definir o escopo da nossa análise. Para exemplificar, vamos considerar o *headway* no pico da manhã, entre 7h e 9h, em uma típica terça-feira de operação. Para isso, precisamos filtrar o nosso *feed*:

```
# lê o arquivo GTFS
gtfs <- read_gtfs(endereco)

# filtra o GTFS
gtfs_filtrado <- gtfs |>
  remove_duplicates() |>
  filter_by_weekday("tuesday") |>
  filter_by_time_of_day(from = "07:00:00", to = "09:00:00")

# checa resultado do filtro
gtfs_filtrado$frequencies[trip_id == "2105-10-0"]
```

```
  trip_id start_time end_time headway_secs
1: 2105-10-0 07:00:00 07:59:00          900
2: 2105-10-0 08:00:00 08:59:00         1200
```

```
gtfs_filtrado$calendar
```

```
  service_id monday tuesday wednesday thursday friday saturday sunday
1:      USD      1         1           1         1         1         1
2:      U__      1         1           1         1         1         0
  start_date  end_date
1: 2008-01-01 2020-05-01
2: 2008-01-01 2020-05-01
```

Em seguida, precisamos calcular o *headway* dentro do período estabelecido. Essa informação pode ser encontrada na tabela *frequencies*, mas há um elemento complicador: cada viagem está associada a mais de um *headway*, como vimos anteriormente (um registro para o período entre 7h e 7h59 e outro para o período entre 8h e 8h59). Para resolver essa questão, portanto, vamos calcular o *headway* médio no intervalo entre 7h e 9h.

Os primeiros registros da tabela frequências do arquivo GTFS da SPTrans parecem sugerir que os períodos do dia estão listados sempre de uma em uma hora, porém essa não é uma regra estabelecida na especificação nem é a prática adotada em outros *feeds*. Por isso, vamos calcular a *média ponderada* do *headway* no período especificado. Para isso, precisamos multiplicar cada *headway* pelo intervalo de tempo em que ele é válido e dividir o total dessa soma pelo intervalo de tempo total (duas horas). Para calcular o intervalo de tempo em que cada *headway* é válido, calculamos o começo e o fim do intervalo com a função `convert_time_to_seconds()` e subtraímos o valor do fim pelo do começo, como mostrado a seguir:

```
gtfs_filtrado <- convert_time_to_seconds(gtfs_filtrado)

gtfs_filtrado$frequencias[trip_id == "2105-10-0"]

  trip_id start_time end_time headway_secs start_time_secs end_time_secs
1: 2105-10-0 07:00:00 07:59:00         900         25200         28740
2: 2105-10-0 08:00:00 08:59:00        1200         28800         32340
```

```
gtfs_filtrado$frequencias[, time_interval := end_time_secs -
  start_time_secs]
```

Em seguida, calculamos o *headway* médio:

```
headway_medio <- gtfs_filtrado$frequencias[,
  .(headway_medio = weighted.mean(x = headway_secs,
  w = time_interval)),
  by = trip_id
]

headway_medio[trip_id == "2105-10-0"]
```

```
  trip_id headway_medio
1: 2105-10-0          1050
```

```
head(headway_medio)
```

```
  trip_id headway_medio
1: CPTM L07-0          360
2: CPTM L07-1          360
3: CPTM L08-0          300
4: CPTM L08-1          300
5: CPTM L09-0          240
6: CPTM L09-1          240
```

Precisamos agora gerar a trajetória espacial de cada viagem e juntar essa informação à do *headway* médio. Para isso, vamos utilizar a função `get_trip_geometry()`, que, dado um objeto GTFS, gera a trajetória espacial de suas viagens. Essa função nos permite especificar as viagens cujas trajetórias queremos gerar, logo vamos calcular apenas as trajetórias daquelas que estão presentes na tabela de *headways* médios:

```
viagens_selecionadas <- headway_medio$trip_id

trajetorias <- get_trip_geometry(
  gtfs = gtfs_filtrado,
  trip_id = viagens_selecionadas,
  file = "shapes"
)

head(trajetorias)
```

Simple feature collection with 6 features and 2 fields

Geometry type: LINESTRING

Dimension: XY

Bounding box: xmin: -46.98404 ymin: -23.73644 xmax: -46.63535
ymax: -23.19474

Geodetic CRS: WGS 84

	trip_id	origin_file	geometry
1	CPTM L07-0	shapes	LINESTRING (-46.63535 -23.5...
2	CPTM L07-1	shapes	LINESTRING (-46.87255 -23.1...
3	CPTM L08-0	shapes	LINESTRING (-46.64073 -23.5...
4	CPTM L08-1	shapes	LINESTRING (-46.98404 -23.5...
5	CPTM L09-0	shapes	LINESTRING (-46.77604 -23.5...
6	CPTM L09-1	shapes	LINESTRING (-46.69711 -23.7...

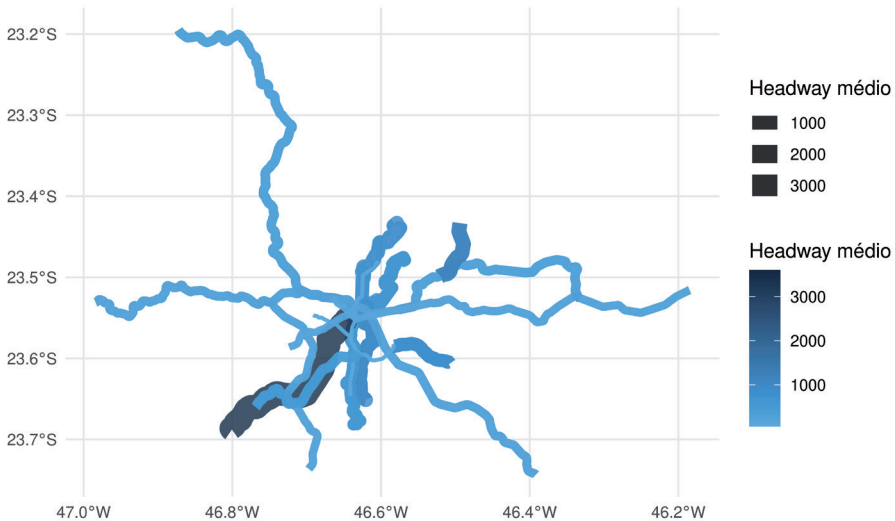
Geradas as trajetórias espaciais de cada viagem, precisamos juntá-las à informação de *headway* médio e, em seguida, configurar o nosso mapa como desejado. No exemplo a seguir, usamos cores e espessuras de linhas que variam de acordo com o *headway* de cada viagem:

```
traj_com_headways <- merge(
  trajetorias,
  headway_medio,
  by = "trip_id"
)
```



```
# configura figura
ggplot(traj_com_headways) +
  geom_sf(aes(color = headway_medio, size = headway_medio),
    alpha = 0.8) +
  scale_color_gradient(high = "#132B43", low = "#56B1F7") +
  labs(color = "Headway médio", size = "Headway médio") +
  theme_minimal()
```

FIGURA 9
Distribuição espacial dos *headways* no arquivo GTFS da SPTrans



Fonte: Figura gerada pelo código supracitado.

Como podemos ver, o pacote `{gtfstools}` torna o desenvolvimento de análises de *feeds* de transporte público algo fácil e que requer apenas o conhecimento básico de pacotes de manipulação de tabelas (como o `{data.table}` e o `{dplyr}`). O exemplo apresentado nesta seção mostra como muitas de suas funções podem ser usadas conjuntamente para revelar aspectos importantes de sistemas de transporte público descritos no formato GTFS.