Package 'tidytransit'

May 10, 2021

Type Package

Title Read, Validate, Analyze, and Map Files in the General Transit Feed Specification

Version 1.0.0

Description Read General Transit Feed Specification (GTFS) zipfiles into a list of R dataframes. Perform validation of the data structure against the specification. Analyze the headways and frequencies at routes and stops. Create maps and perform spatial analysis on the routes and stops. Please see the GTFS documentation here for more detail: <http://gtfs.org/>.

License GPL

LazyData TRUE

Depends R (>= 3.6.0)

Imports gtfsio (>= 0.1.0), dplyr, zip (>= 2.0.1), tibble, readr, data.table (>= 1.12.8), httr, assertthat, rlang, sf, lubridate, hms, tidyr, tools, digest, checkmate

Suggests testthat, knitr, markdown, rmarkdown, ggplot2, scales

RoxygenNote 7.1.1

URL https://github.com/r-transit/tidytransit

BugReports https://github.com/r-transit/tidytransit

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author Flavio Poletti [aut],
Daniel Herszenhut [aut] (<https://orcid.org/0000-0001-8066-1105>),
Mark Padgham [aut],
Tom Buckley [aut, cre],
Danton Noriega-Goodwin [aut],
Angela Li [ctb],
Elaine McVey [ctb],
Charles Hans Thompson [ctb],
Michael Sumner [ctb],

Patrick Hausmann [ctb],
Bob Rudis [ctb],
James Lamb [ctb],
Alexandra Kapp [ctb],
Kearey Smith [ctb],
Dave Vautin [ctb],
Kyle Walker [ctb]

Maintainer Tom Buckley <tom@tbuckl.com>

Repository CRAN

Date/Publication 2021-05-10 00:00:02 UTC

R topics documented:

convert_times_to_hms . . . 2
feedlist . . . 3
feed_contains . . . 4
filter_stops . . . 4
filter_stop_times . . . 5
get_feedlist . . . 6
get_route_geometry . . . 6
get_stop_frequency . . . 7
get_trip_geometry . . . 8
gtfs_as_sf . . . 8
gtfs_duke . . . 9
plot.tidygtfs . . . 9
print.tidygtfs . . . 10
raptor . . . 10
read_gtfs . . . 12
route_type_names . . . 13
set_api_key . . . 14
shapes_as_sf . . . 14
stops_as_sf . . . 15
summary.tidygtfs . . . 15
travel_times . . . 16
validate_gtfs . . . 17
write_gtfs . . . 18

Index **20**

`convert_times_to_hms` *Use hms::hms columns in feed*

Description

Overwirtes character columns in stop_times (arrival_time, departure_time) and frequencies (start_time, end_time) with times converted with `hms::hms()`.

Usage

```
convert_times_to_hms(gtfs_obj)
```

Arguments

gtfs_obj	a gtfs object in which hms times should be set, the modified gtfs_obj is returned

Value

gtfs_obj with added hms times columns for stop_times and frequencies

---

feedlist *Dataframe of source GTFS data from Transitfeeds*

Description

A dataset containing a list of URLs for GTFS feeds

Usage

```
feedlist
```

Format

A data frame with 911 rows and 10 variables:

id the id of the feed on transitfeeds.com

t title of the feed

loc_id location id

loc_pid location placeid of the feed on transitfeeds.com

loc_t the title of the location

loc_n the shortname fo the location

loc_lat the location latitude

loc_lng the location longitude

url_d GTFS feed url

url_i the metadata url for the feed

Source

http://www.transitfeeds.com/

`feed_contains` *Returns TRUE if the given gtfs_obj contains the table. Used to check for tidytransit's calculated tables in sublist*

Description

Returns TRUE if the given gtfs_obj contains the table. Used to check for tidytransit's calculated tables in sublist

Usage

```
feed_contains(gtfs_obj, table_name)
```

Arguments

`gtfs_obj`	gtfs object
`table_name`	name as string of the table to look for

`filter_stops` *Get a set of stops for a given set of service ids and route ids*

Description

Get a set of stops for a given set of service ids and route ids

Usage

```
filter_stops(gtfs_obj, service_ids, route_ids)
```

Arguments

`gtfs_obj`	as read by read_gtfs()
`service_ids`	the service for which to get stops
`route_ids`	the route_ids for which to get stops

Value

stops for a given service

Examples

```
library(dplyr)
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(local_gtfs_path)
select_service_id <- filter(nyc$calendar, monday==1) %>% pull(service_id)
select_route_id <- sample_n(nyc$routes, 1) %>% pull(route_id)
filtered_stops_df <- filter_stops(nyc, select_service_id, select_route_id)
```

---

filter_stop_times *Filter a* stop_times *table for a given date and timespan.*

Description

Filter a stop_times table for a given date and timespan.

Usage

```
filter_stop_times(gtfs_obj, extract_date, min_departure_time, max_arrival_time)
```

Arguments

gtfs_obj	a gtfs feed
extract_date	date to extract trips from in YYYY-MM-DD format
min_departure_time	The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.
max_arrival_time	The latest arrival time. Can be given as "HH:MM:SS", hms object or numeric value in seconds This function creates filtered stop_times for travel_times() and raptor().

Examples

```
feed_path <- system.file("extdata", "sample-feed-fixed.zip", package = "tidytransit")
g <- read_gtfs(feed_path)

# filter the sample feed
stop_times <- filter_stop_times(g, "2007-01-06", "06:00:00", "08:00:00")
```

get_feedlist *Get list of all available feeds from transitfeeds API*

Description

Get list of all available feeds from transitfeeds API

Usage

```
get_feedlist()
```

Value

a data frame with the gtfs feeds on transitfeeds.

See Also

feedlist_df

Examples

```
## Not run:
feedlist_df <- get_feedlist()

## End(Not run)
```

get_route_geometry *Get all trip shapes for a given route and service*

Description

Get all trip shapes for a given route and service

Usage

```
get_route_geometry(gtfs_sf_obj, route_ids = NULL, service_ids = NULL)
```

Arguments

gtfs_sf_obj	tidytransit gtfs object with sf data frames
route_ids	routes to extract
service_ids	service_ids to extract

Value

an sf dataframe for gtfs routes with a row/linestring for each trip

Examples

```
data(gtfs_duke)
gtfs_duke_sf <- gtfs_as_sf(gtfs_duke)
routes_sf <- get_route_geometry(gtfs_duke_sf)
plot(routes_sf[c(1,1350),])
```

`get_stop_frequency` *Get Stop Frequency*

Description

Note that some GTFS feeds contain a frequency data frame already. Consider using this instead, as it will be more accurate than what tidytransit calculates.

Usage

```
get_stop_frequency(
  gtfs_obj,
  start_hour = 6,
  end_hour = 22,
  service_ids = NULL,
  by_route = FALSE
)
```

Arguments

`gtfs_obj`	a list of gtfs dataframes as read by `read_gtfs()`.
`start_hour`	(optional) an integer indicating the start hour (default 6)
`end_hour`	(optional) an integer indicating the end hour (default 22)
`service_ids`	(optional) a set of service_ids from the calendar dataframe identifying a particular service id. If not provided the service_id with the most departures is used
`by_route`	default TRUE, if FALSE then calculate headway for any line coming through the stop in the same direction on the same schedule.

Value

dataframe of stops with the number of departures and the headway (departures divided by timespan) as columns.

Examples

```
data(gtfs_duke)
stop_frequency <- get_stop_frequency(gtfs_duke)
x <- order(stop_frequency$mean_headway)
head(stop_frequency[x,])
```

get_trip_geometry *Get all trip shapes for given trip ids*

Description

Get all trip shapes for given trip ids

Usage

```
get_trip_geometry(gtfs_sf_obj, trip_ids)
```

Arguments

gtfs_sf_obj	tidytransit gtfs object with sf data frames
trip_ids	trip_ids to extract shapes

Value

an sf dataframe for gtfs routes with a row/linestring for each trip

Examples

```
data(gtfs_duke)
gtfs_duke <- gtfs_as_sf(gtfs_duke)
trips_sf <- get_trip_geometry(gtfs_duke, c("t_726295_b_19493_tn_41", "t_726295_b_19493_tn_40"))
plot(trips_sf[1,])
```

gtfs_as_sf *Convert stops and shapes to Simple Features*

Description

Stops are converted to POINT sf data frames. Shapes are created as LINESTRING data frame. Note that this function replaces stops and shapes tables in gtfs_obj.

Usage

```
gtfs_as_sf(gtfs_obj, skip_shapes = FALSE, crs = NULL, quiet = TRUE)
```

Arguments

gtfs_obj	a standard tidytransit gtfs object
skip_shapes	if TRUE, shapes are not converted. Default FALSE.
crs	optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates of stops and shapes
quiet	boolean whether to print status messages

Value

gtfs_obj a tidytransit gtfs object with stops and shapes as sf data frames

---

gtfs_duke *Example GTFS data*

Description

Data obtained from http://data.trilliumtransit.com/gtfs/duke-nc-us/duke-nc-us.zip.

Usage

```
gtfs_duke
```

Format

An object of class `tidygtfs` (inherits from `gtfs`) of length 25.

See Also

read_gtfs

---

plot.tidygtfs *Plot GTFS stops and trips*

Description

Plot GTFS stops and trips

Usage

```
## S3 method for class 'tidygtfs'
plot(x, ...)
```

Arguments

x	a gtfs_obj as read by read_gtfs()
...	further specifications

Examples

```
local_gtfs_path <- system.file("extdata",
                               "google_transit_nyc_subway.zip",
                               package = "tidytransit")
nyc <- read_gtfs(local_gtfs_path)
plot(nyc)
```

print.tidygtfs *Print a GTFS object*

Description

Prints a GTFS object suppressing the `class` attribute.

Usage

```
## S3 method for class 'tidygtfs'
print(x, ...)
```

Arguments

`x`	A GTFS object.
`...`	Optional arguments ultimately passed to `format`.

Value

The GTFS object that was printed, invisibly.

Examples

```
 ## Not run:
path = system.file("extdata",
           "google_transit_nyc_subway.zip",
           package = "tidytransit")

g = read_gtfs(path)
print(g)

## End(Not run)
```

raptor *Calculate travel times from one stop to all reachable stops*

Description

`raptor` finds the minimal travel time, earliest or latest arrival time for all stops in `stop_times` with journeys departing from `stop_ids` within `time_range`.

Usage

```
raptor(
  stop_times,
  transfers,
  stop_ids,
  arrival = FALSE,
  time_range = 3600,
  max_transfers = NULL,
  keep = "all"
)
```

Arguments

stop_times	A (prepared) stop_times table from a gtfs feed. Prepared means that all stop time rows before the desired journey departure time should be removed. The table should also only include departures happening on one day. Use `filter_stop_times()` for easier preparation.
transfers	Transfers table from a gtfs feed. In general no preparation is needed.
stop_ids	Character vector with stop_ids from where journeys should start (or end)
arrival	If FALSE (default), all journeys *start* from `stop_ids`. If TRUE, all journeys *end* at `stop_ids`.
time_range	Departure or arrival time range in seconds. All departures from the first departure of stop_times (not necessarily from stop_id in `stop_ids`) within `time_range` are considered. If `arrival` is TRUE, all arrivals within `time_range` before the latest arrival time of stop_times are considered.
max_transfers	Maximum number of transfers allowed, no limit (NULL) as default.
keep	One of c("all", "shortest", "earliest", "latest"). By default, `all` journeys arriving at a stop are returned. With `shortest` the journey with shortest travel time is returned. With `earliest` the journey arriving at a stop the earliest is returned, `latest` works accordingly.

Details

With a modified Round-Based Public Transit Routing Algorithm (RAPTOR) using data.table, earliest arrival times for all stops are calculated. If two journeys arrive at the same time, the one with the later departure time and thus shorter travel time is kept. By default, all journeys departing within `time_range` that arrive at a stop are returned in a table. If you want all journeys *arriving* at stop_ids within the specified time range, set `arrival` to TRUE.

Journeys are defined by a "from" and "to" stop_id, a departure, arrival and travel time. Note that the exact journeys (with each intermediate stop and route ids for example) is *not* returned.

For most cases, `stop_times` needs to be filtered, as it should only contain trips happening on a single day and departures later than a given journey start time, see `filter_stop_times()`. The algorithm scans all trips until it exceeds `max_transfers` or all trips in `stop_times` have been visited.

Value

A data.table with journeys (departure, arrival and travel time) to/from all stop_ids reachable by `stop_ids`.

See Also

`travel_times()` for an easier access to travel time calculations via stop_names.

Examples

```
nyc_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)

# you can use initial walk times to different stops in walking distance (arbitrary example values)
stop_ids_harlem_st <- c("301", "301N", "301S")
stop_ids_155_st <- c("A11", "A11N", "A11S", "D12", "D12N", "D12S")
walk_times <- data.frame(stop_id = c(stop_ids_harlem_st, stop_ids_155_st),
                         walk_time = c(rep(600, 3), rep(410, 6)), stringsAsFactors = FALSE)

# Use journeys departing after 7 AM with arrival time before 11 AM on 26th of June
stop_times <- filter_stop_times(nyc, "2018-06-26", 7*3600, 9*3600)

# calculate all journeys departing from Harlem St or 155 St between 7:00 and 7:30
rptr <- raptor(stop_times, nyc$transfers, walk_times$stop_id, time_range = 1800,
               keep = "all")

# add walk times to travel times
rptr <- merge(rptr, walk_times, by.x = "from_stop_id", by.y = "stop_id")
rptr$travel_time_incl_walk <- rptr$travel_time + rptr$walk_time

# get minimal travel times (with walk times) for all stop_ids
library(data.table)
shortest_travel_times <- setDT(rptr)[order(travel_time_incl_walk)][, .SD[1], by = "to_stop_id"]
hist(shortest_travel_times$travel_time, breaks = 360)
```

---

read_gtfs *Read and validate GTFS files*

Description

Reads GTFS text files from either a local `.zip` file or an URL and validates them against GTFS specifications.

Usage

```
read_gtfs(path, files = NULL, quiet = TRUE)
```

Arguments

path	The path to a GTFS .zip file.
files	A character vector containing the text files to be read from the GTFS (without the .txt extension). If NULL (the default) all existing files are read.
quiet	Whether to hide log messages and progress bars (defaults to TRUE).

Value

A tidygtfs object: a list of tibbles in which each entry represents a GTFS text file. Additional tables are stored in the . sublist.

See Also

validate_gtfs

Examples

```
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
gtfs <- read_gtfs(local_gtfs_path)
names(gtfs)

gtfs <- read_gtfs(local_gtfs_path, files = c("trips", "stop_times"))
names(gtfs)
```

---

route_type_names *Dataframe of route type id's and the names of the types (e.g. "Cable Car")*

Description

Extended GTFS Route Types: https://developers.google.com/transit/gtfs/reference/extended-route-types

Usage

```
route_type_names
```

Format

A data frame with 132 rows and 2 variables:

route_type the id of route type

route_type_name name of the gtfs route type

Source

https://gist.github.com/derhuerst/b0243339e22c310bee2386388151e11e

set_api_key *Set TransitFeeds API key for recall*

Description

Set TransitFeeds API key for recall

Usage

```
set_api_key()
```

shapes_as_sf *Convert shapes into Simple Features Linestrings*

Description

Convert shapes into Simple Features Linestrings

Usage

```
shapes_as_sf(gtfs_shapes, crs = NULL)
```

Arguments

gtfs_shapes	a gtfs$shapes dataframe
crs	optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates

Value

an sf dataframe for gtfs shapes

stops_as_sf *Convert stops into Simple Features Points*

Description

Convert stops into Simple Features Points

Usage

```
stops_as_sf(stops, crs = NULL)
```

Arguments

stops	a gtfs$stops dataframe
crs	optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates

Value

an sf dataframe for gtfs routes with a point column

Examples

```
data(gtfs_duke)
some_stops <- gtfs_duke$stops[sample(nrow(gtfs_duke$stops), 40),]
some_stops_sf <- stops_as_sf(some_stops)
plot(some_stops_sf)
```

summary.tidygtfs *GTFS feed summary*

Description

GTFS feed summary

Usage

```
## S3 method for class 'tidygtfs'
summary(object, ...)
```

Arguments

object	a gtfs_obj as read by read_gtfs()
...	further specifications

`travel_times` *Calculate shortest travel times from a stop to all reachable stops*

Description

Function to calculate the shortest travel times from a stop (given by `stop_name`) to all other stops of a feed. `filtered_stop_times` needs to be created before with `filter_stop_times()`.

Usage

```
travel_times(
  filtered_stop_times,
  stop_name,
  time_range = 3600,
  arrival = FALSE,
  max_transfers = NULL,
  max_departure_time = NULL,
  return_coords = FALSE,
  return_DT = FALSE
)
```

Arguments

`filtered_stop_times`	stop_times data.table (with transfers and stops tables as attributes) created with `filter_stop_times()` where the departure or arrival time has been set.
`stop_name`	Stop name for which travel times should be calculated. A vector with multiple names is accepted.
`time_range`	All departures within this range in seconds after the first departure of `filtered_stop_times` are considered for journeys. If arrival is TRUE, all journeys arriving within time range before the latest arrival of `filtered_stop_times` are considered.
`arrival`	If FALSE (default), all journeys *start* from `stop_name`. If TRUE, all journeys *end* at `stop_name`.
`max_transfers`	The maximimum number of transfers
`max_departure_time`	Either set this parameter or `time_range`. Only departures before `max_departure_time` are used. Accepts "HH:MM:SS" or seconds as a numerical value. Unused if `arrival` is TRUE.
`return_coords`	Returns stop coordinates as columms. Default is FALSE.
`return_DT`	travel_times() returns a data.table if TRUE. Default is FALSE which returns a tibble/tbl_df.

Details

This function allows easier access to `raptor()` by using stop names instead of ids and returning shortest travel times by default.

Value

A table with travel times to/from all stops reachable by `stop_name` and their corresponding journey departure and arrival times.

Examples

```
nyc_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)

# Use journeys departing after 7 AM with arrival time before 9 AM on 26th June
stop_times <- filter_stop_times(nyc, "2018-06-26", 7*3600, 9*3600)

tts <- travel_times(stop_times, "34 St - Herald Sq", return_coords = TRUE)
library(dplyr)
tts <- tts %>% filter(travel_time <= 3600)

# travel time to Queensboro Plaza is 810 seconds, 13:30 minutes
tts %>% filter(to_stop_name == "Queensboro Plaza") %>% pull(travel_time) %>% hms::hms()

# plot a simple map showing travel times to all reachable stops
# this can be expanded to isochron maps
library(ggplot2)
ggplot(tts) + geom_point(aes(x=to_stop_lon, y=to_stop_lat, color = travel_time))
```

`validate_gtfs` *Validate GTFS file*

Description

Validates the GTFS object against GTFS specifications and raises warnings if required files/fields are not found. This function is called in `read_gtfs`.

Usage

```
validate_gtfs(gtfs_obj, files = NULL, quiet = TRUE, warnings = TRUE)
```

Arguments

`gtfs_obj`	A GTFS object
`files`	A character vector containing the text files to be validated against the GTFS specification (without the `.txt` extension). If `NULL` (the default) the provided GTFS is validated against all possible GTFS text files.
`quiet`	Whether to hide log messages (defaults to TRUE).
`warnings`	Whether to display warning messages (defaults to TRUE).

Value

A GTFS object with a `validation_result` attribute. This attribute is a `tibble` containing the validation summary of all possible fields from the specified files.

Details

GTFS object's files and fields are validated against the GTFS specifications as documented in Google's Static GTFS Reference:

- GTFS feeds are considered valid if they include all required files and fields. If a required file/field is missing the function (optionally) raises a warning.
- Optional files/fields are listed in the reference above but are not required, thus no warning is raised if they are missing.
- Extra files/fields are those who are not listed in the reference above (either because they refer to a specific GTFS extension or due to any other reason).

Note that some files (`calendar.txt`, `calendar_dates.txt` and `feed_info.txt`) are conditionally required. This means that:

- `calendar.txt` is initially set as a required file. If it's not present, however, it becomes optional and `calendar_dates.txt` (originally set as optional) becomes required.
- `feed_info.txt` is initially set as an optional file. If `translations.txt` is present, however, it becomes required.

Examples

```
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
gtfs <- read_gtfs(local_gtfs_path)
attr(gtfs, "validation_result")

gtfs$shapes <- NULL
validation_result <- validate_gtfs(gtfs)

# should raise a warning
gtfs$stop_times <- NULL
validation_result <- validate_gtfs(gtfs)
```

---

write_gtfs *Write a tidygtfs object to a zip file*

Description

Write a tidygtfs object to a zip file

Usage

```
write_gtfs(gtfs_obj, zipfile, compression_level = 9, as_dir = FALSE)
```

Arguments

gtfs_obj	a gtfs feed object
zipfile	path to the zip file the feed should be written to
compression_level	a number between 1 and 9.9, passed to zip::zip
as_dir	if TRUE, the feed is not zipped and zipfile is used as a directory path. Files within the directory will be overwritten.

Note

Auxilliary tidytransit tables (e.g. dates_services) are not exported.

Index

∗ **datasets**
feedlist, 3
gtfs_duke, 9
route_type_names, 13

convert_times_to_hms, 2

feed_contains, 4
feedlist, 3
filter_stop_times, 5
filter_stop_times(), *11*, *16*
filter_stops, 4

get_feedlist, 6
get_route_geometry, 6
get_stop_frequency, 7
get_trip_geometry, 8
gtfs_as_sf, 8
gtfs_duke, 9

hms::hms(), *2*

plot.tidygtfs, 9
print.tidygtfs, 10

raptor, 10
raptor(), *5*, *16*
read_gtfs, 12, *17*
read_gtfs(), *7*, *15*
route_type_names, 13

set_api_key, 14
shapes_as_sf, 14
stops_as_sf, 15
summary.tidygtfs, 15

travel_times, 16
travel_times(), *5*, *12*

validate_gtfs, *13*, 17

write_gtfs, 18