Package: tidytransit (via r-universe)

October 18, 2024

Type Package

Title Read, Validate, Analyze, and Map GTFS Feeds

Version 1.7.0

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: <https://gtfs.org/>.

License GPL

LazyData TRUE

Depends R (>= 3.6.0)

Imports gtfsio (>= 1.2.0), dplyr (>= 1.1.1), data.table (>= 1.12.8), rlang, sf, jsonlite, hms, digest, geodist

Suggests testthat (>= 3.1.5), knitr, markdown, rmarkdown, ggplot2, scales, lubridate, leaflet

RoxygenNote 7.3.2

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

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

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Encoding UTF-8

Repository https://r-transit.r-universe.dev

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

RemoteRef HEAD

RemoteSha a09a52d4ad3cbf67c3bb1dc74ae00f8ec78e1744

Contents

`as_tidygtfs` *Convert another gtfs like object to a tidygtfs object*

Description

Convert another gtfs like object to a tidygtfs object

Usage

```
as_tidygtfs(x, ...)
```

Arguments

x	gtfs object
...	ignored

Value

a tidygtfs object

`cluster_stops` *Cluster nearby stops within a group*

Description

Finds clusters of stops for each unique value in `group_col` (e.g. stop_name). Can be used to find different groups of stops that share the same name but are located more than `max_dist` apart. `gtfs_stops` is assigned a new column (named `cluster_colname`) which contains the `group_col` value and the cluster number.

Usage

```
cluster_stops(
  gtfs_stops,
  max_dist = 300,
  group_col = "stop_name",
  cluster_colname = "stop_name_cluster"
)
```

Arguments

`gtfs_stops`	Stops table of a gtfs object. It is also possible to pass a tidygtfs object to enable piping.
`max_dist`	Only stop groups that have a maximum distance among them above this threshold (in meters) are clustered.
`group_col`	Clusters for are calculated for each set of stops with the same value in this column (default: stop_name)
`cluster_colname`	Name of the new column name. Can be the same as group_col to overwrite.

Details

`stats::kmeans()` is used for clustering.

Value

Returns a stops table with an added cluster column. If `gtfs_stops` is a tidygtfs object, a modified tidygtfs object is return

Examples

```
library(dplyr)
nyc_path <- system.file("extdata", "nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)
nyc <- cluster_stops(nyc)

# There are 6 stops with the name "86 St" that are far apart
stops_86_St = nyc$stops %>%
  filter(stop_name == "86 St")

table(stops_86_St$stop_name_cluster)

stops_86_St %>% select(stop_id, stop_name, parent_station, stop_name_cluster) %>% head()

library(ggplot2)
ggplot(stops_86_St) +
  geom_point(aes(stop_lon, stop_lat, color = stop_name_cluster))
```

---

empty_strings_to_na *Convert empty strings ("") to NA values in all gtfs tables*

Description

Convert empty strings ("") to NA values in all gtfs tables

Usage

```
empty_strings_to_na(gtfs_obj)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)

Value

a gtfs_obj where all empty strings in tables have been replaced with NA

See Also

na_to_empty_strings()

filter_feed_by_area *Filter a gtfs feed so that it only contains trips that pass a given area*

Description

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

Usage

```
filter_feed_by_area(gtfs_obj, area)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)
area	all trips passing through this area are kept. Either a bounding box (numeric vector with xmin, ymin, xmax, ymax) or a sf object.

Value

tidygtfs object with filtered tables

See Also

filter_feed_by_stops, filter_feed_by_trips, filter_feed_by_date

filter_feed_by_date *Filter a gtfs feed so that it only contains trips running on a given date*

Description

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

Usage

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

Arguments

gtfs_obj	gtfs feed (tidygtfs object)
extract_date	date to extract trips from this day (Date or "YYYY-MM-DD" string)
min_departure_time	(optional) The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.
max_arrival_time	(optional) The latest arrival time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.

Value

tidygtfs object with filtered tables

See Also

filter_stop_times, filter_feed_by_trips, filter_feed_by_trips, filter_feed_by_date

---

filter_feed_by_stops *Filter a gtfs feed so that it only contains trips that pass the given stops*

Description

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

Usage

```
filter_feed_by_stops(gtfs_obj, stop_ids = NULL, stop_names = NULL)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)
stop_ids	vector with stop_ids. You can either provide stop_ids or stop_names
stop_names	vector with stop_names (will be converted to stop_ids)

Value

tidygtfs object with filtered tables

Note

The returned gtfs_obj likely contains more than just the stops given (i.e. all stops that belong to a trip passing the initial stop).

See Also

filter_feed_by_trips, filter_feed_by_trips, filter_feed_by_date

filter_feed_by_trips *Filter a gtfs feed so that it only contains a given set of trips*

Description

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

Usage

```
filter_feed_by_trips(gtfs_obj, trip_ids)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)
trip_ids	vector with trip_ids

Value

tidygtfs object with filtered tables

See Also

filter_feed_by_stops, filter_feed_by_area, filter_feed_by_date

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	gtfs feed (tidygtfs object)
service_ids	the service for which to get stops
route_ids	the route_ids for which to get stops

Value

stops table for a given service or route

Examples

```
library(dplyr)
local_gtfs_path <- system.file("extdata", "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`	gtfs feed (tidygtfs object)
`extract_date`	date to extract trips from this day (Date or "YYYY-MM-DD" string)
`min_departure_time`	(optional) The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.
`max_arrival_time`	(optional) The latest arrival time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.

Value

Filtered `stop_times` data.table for `travel_times()` and `raptor()`.

Examples

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

# filter the sample feed
stop_times <- filter_stop_times(g, "2018-10-01", "06:00:00", "08:00:00")
```

`get_route_frequency` *Get Route Frequency*

Description

Calculate the number of departures and mean headways for routes within a given timespan and for given service_ids.

Usage

```
get_route_frequency(
  gtfs_obj,
  start_time = "06:00:00",
  end_time = "22:00:00",
  service_ids = NULL
)
```

Arguments

`gtfs_obj`	gtfs feed (tidygtfs object)
`start_time`	analysis start time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
`end_time`	analysis period end time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
`service_ids`	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.

Value

a dataframe of routes with variables or headway/frequency in seconds for a route within a given time frame

Note

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

Examples

```
data(gtfs_duke)
routes_frequency <- get_route_frequency(gtfs_duke)
x <- order(routes_frequency$median_headways)
head(routes_frequency[x,])
```

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

Calculate the number of departures and mean headways for all stops within a given timespan and for given service_ids.

Usage

```
get_stop_frequency(
  gtfs_obj,
  start_time = "06:00:00",
  end_time = "22:00:00",
  service_ids = NULL,
  by_route = TRUE
)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)
start_time	analysis start time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
end_time	analysis period end time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
service_ids	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) in seconds as columns

Note

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

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,"shape_id"])
```

`gtfs_as_sf` *Convert stops and shapes to Simple Features*

Description

Stops are converted to POINT sf data frames. Shapes are converted to a 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`	gtfs feed (tidygtfs object, created by `read_gtfs()`)
`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

tidygtfs object with stops and shapes as sf dataframes

See Also

`sf_as_tbl`, `stops_as_sf`, `shapes_as_sf`

gtfs_duke *Example GTFS data*

Description

Data obtained from https://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()`

gtfs_transform *Transform coordinates of a gtfs feed*

Description

Transform coordinates of a gtfs feed

Usage

```
gtfs_transform(gtfs_obj, crs)
```

Arguments

`gtfs_obj`	gtfs feed (tidygtfs object)
`crs`	target coordinate reference system, used by sf::st_transform

Value

tidygtfs object with transformed stops and shapes sf dataframes

gtfs object with transformed sf tables

interpolate_stop_times *Interpolate missing stop_times linearly*

Description

Interpolate missing stop_times linearly

Usage

```
interpolate_stop_times(x, use_shape_dist = TRUE)
```

Arguments

x	tidygtfs object or stop_times table
use_shape_dist	If TRUE, use shape_dist_traveled column from the shapes table for time interpolation (if that column is available). If FALSE or shape_dist_traveled is missing, times are interpolated equally between stops.

Value

tidygtfs or stop_times with interpolated arrival and departure times

Examples

```
## Not run:
data(gtfs_duke)
print(gtfs_duke$stop_times[1:5, 1:5])

gtfs_duke_2 = interpolate_stop_times(gtfs_duke)
print(gtfs_duke_2$stop_times[1:5, 1:5])

gtfs_duke_3 = interpolate_stop_times(gtfs_duke, FALSE)
print(gtfs_duke_3$stop_times[1:5, 1:5])

## End(Not run)
```

na_to_empty_strings *Convert NA values to empty strings ("")*

Description

Convert NA values to empty strings ("")

Usage

```
na_to_empty_strings(gtfs_obj)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)

Value

a gtfs_obj where all NA strings in tables have been replaced with ""

See Also

empty_strings_to_na()

---

plot.tidygtfs *Plot GTFS stops and trips*

Description

Plot GTFS stops and trips

Usage

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

Arguments

x	a tidygtfs object as read by read_gtfs()
...	ignored for tidygtfs

Value

plot

Examples

```
local_gtfs_path <- system.file("extdata",
                               "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 and hiding the validation_result attribute, created with `validate_gtfs()`.

Usage

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

Arguments

`x`	a tidygtfs object as read by `read_gtfs()`
`...`	Optional arguments ultimately passed to `format`.

Value

The GTFS object that was printed, invisibly

Examples

```
 ## Not run:
path = system.file("extdata",
           "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. Can be omitted if stop_times has been prepared with `filter_stop_times()`.
stop_ids	Character vector with stop_ids from where journeys should start (or end). It is recommended to only use stop_ids that are related to each other, like different platforms in a train station or bus stops that are reasonably close to each other.
arrival	If FALSE (default), all journeys *start* from `stop_ids`. If TRUE, all journeys *end* at `stop_ids`.
time_range	Either a range in seconds or a vector containing the minimal and maximal departure time (i.e. earliest and latest possible journey departure time) as seconds or "HH:MM:SS" character. If `arrival` is TRUE, `time_range` describes the time window when journeys should end at `stop_ids`.
max_transfers	Maximum number of transfers allowed, no limit (NULL) as default.
keep	One of c("all", "shortest", "earliest", "latest"). By default, `all` journeys between stop_ids are returned. With `shortest` only the journey with the 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 exact journeys (with each intermediate stop and route ids for example) are *not* returned.

For most cases, `stop_times` needs to be filtered, as it should only contain trips happening on a single day, 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", "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 = seq(0,2*60)*60)
```

---

read_gtfs *Read and validate GTFS files*

Description

Reads a GTFS feed 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 validated against the GTFS specification without the file extension (txt or geojson). If NULL (the default), all existing files are read.
quiet	Whether to hide log messages and progress bars (defaults to TRUE).
...	Can be used to pass on arguments to gtfsio::import_gtfs(). The parameters files and quiet are passed on by default.

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(), write_gtfs()

Examples

```
## Not run:
local_gtfs_path <- system.file("extdata", "nyc_subway.zip", package = "tidytransit")
gtfs <- read_gtfs(local_gtfs_path)
summary(gtfs)

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

## End(Not run)
```

---

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

Description

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

Usage

```
route_type_names
```

Format

A data frame with 136 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_servicepattern *Calculate service pattern ids for a GTFS feed*

Description

Each trip has a defined number of dates it runs on. This set of dates is called a service pattern in tidytransit. Trips with the same `servicepattern` id run on the same dates. In general, `service_id` can work this way but it is not enforced by the GTFS standard.

Usage

```
set_servicepattern(
  gtfs_obj,
  id_prefix = "s_",
  hash_algo = "md5",
  hash_length = 7
)
```

Arguments

`gtfs_obj`	gtfs feed (tidygtfs object)
`id_prefix`	all servicepattern ids will start with this string
`hash_algo`	hashing algorithm used by digest
`hash_length`	length the hash should be cut to with `substr()`. Use `-1` if the full hash should be used

Value

modified gtfs_obj with added servicepattern list and a table linking trips and pattern (trip_servicepatterns), added to `gtfs_obj$.` sublist.

---

sf_as_tbl *Convert stops and shapes from sf objects to tibbles*

Description

Coordinates are transformed to lon/lat columns (`stop_lon`/`stop_lat` or `shape_pt_lon`/`shape_pt_lat`)

Usage

```
sf_as_tbl(gtfs_obj)
```

Arguments

gtfs_obj	gtfs feed (tidygtfs object)

Value

tidygtfs object with stops and shapes converted to tibbles

See Also

gtfs_as_sf

---

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

See Also

gtfs_as_sf

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

See Also

gtfs_as_sf

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[,"stop_name"])
```

stop_distances *Calculate distances between a given set of stops*

Description

Calculate distances between a given set of stops

Usage

```
stop_distances(gtfs_stops)
```

Arguments

gtfs_stops	gtfs stops table either as data frame (with at least stop_id, stop_lon and stop_lat columns) or as sf object.

Value

Returns a data.frame with each row containing a pair of stop_ids (columns `from_stop_id` and `to_stop_id`) and the `distance` between them (in meters)

Note

The resulting data.frame has `nrow(gtfs_stops)^2` rows, distances calculations among all stops for large feeds should be avoided.

Examples

```
## Not run:
library(dplyr)

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

nyc$stops %>%
  filter(stop_name == "Borough Hall") %>%
  stop_distances() %>%
  arrange(desc(distance))

#> # A tibble: 36 × 3
#>    from_stop_id to_stop_id  distance
#>    <chr>        <chr>          <dbl>
#>  1 423          232             91.5
#>  2 423N         232             91.5
#>  3 423S         232             91.5
#>  4 423          232N            91.5
#>  5 423N         232N            91.5
#>  6 423S         232N            91.5
#>  7 423          232S            91.5
#>  8 423N         232S            91.5
#>  9 423S         232S            91.5
#> 10 232          423             91.5
#> # . . .  with 26 more rows

## End(Not run)
```

---

stop_group_distances *Calculates distances among stop within the same group column*

Description

By default calculates distances among stop_ids with the same stop_name.

Usage

```
stop_group_distances(gtfs_stops, by = "stop_name")
```

Arguments

gtfs_stops	gtfs stops table either as data frame (with at least stop_id, stop_lon and stop_lat columns) or as sf object.
by	group column, default: "stop_name"

Value

data.frame with one row per group containing a distance matrix (distances), number of stop ids within that group (n_stop_ids) and distance summary values (dist_mean, dist_median and dist_max).

Examples

```
## Not run:
library(dplyr)

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

stop_group_distances(nyc$stops)
#> # A tibble: 380 × 6
#>    stop_name   distances       n_stop_ids dist_mean dist_median dist_max
#>    <chr>       <list>              <dbl>     <dbl>       <dbl>    <dbl>
#>  1 86 St       <dbl [18 × 18]>        18     5395.       5395.   21811.
#>  2 79 St       <dbl [6 × 6]>           6    19053.      19053.   19053.
#>  3 Prospect Av <dbl [6 × 6]>           6    18804.      18804.   18804.
#>  4 77 St       <dbl [6 × 6]>           6    16947.      16947.   16947.
#>  5 59 St       <dbl [6 × 6]>           6    14130.      14130.   14130.
#>  6 50 St       <dbl [9 × 9]>           9     7097.       7097.   14068.
#>  7 36 St       <dbl [6 × 6]>           6    12496.      12496.   12496.
#>  8 8 Av        <dbl [6 × 6]>           6    11682.      11682.   11682.
#>  9 7 Av        <dbl [9 × 9]>           9     5479.       5479.   10753.
#> 10 111 St      <dbl [9 × 9]>           9     3877.       3877.    7753.
#> # . . .  with 370 more rows

## End(Not run)
```

---

summary.tidygtfs *GTFS feed summary*

Description

GTFS feed summary

Usage

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

Arguments

object	a tidygtfs object as read by read_gtfs()
...	ignored for tidygtfs

Value

the tidygtfs object, invisibly

---

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 stop_names of a feed. filtered_stop_times needs to be created before with filter_stop_times() or filter_feed_by_date().

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,
  stop_dist_check = 300
)
```

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 can be used.
time_range	Either a range in seconds or a vector containing the minimal and maximal departure time (i.e. earliest and latest possible journey departure time) as seconds or "HH:MM:SS" character. If arrival is TRUE, time_range describes the time window when journeys should end at stop_name.
arrival	If FALSE (default), all journeys *start* from stop_name. If TRUE, all journeys *end* at stop_name.
max_transfers	The maximum number of transfers. No limit if NULL

max_departure_time
: Deprecated. Use time_range to set the latest possible departure time.

return_coords
: Returns stop coordinates (lon/lat) as columns. Default is FALSE.

return_DT
: travel_times() returns a data.table if TRUE. Default is FALSE which returns a tibble/tbl_df.

stop_dist_check
: stop_names are not structured identifiers like stop_ids or parent_stations, so it's possible that stops with the same name are far apart. travel_times() errors if the distance among stop_ids with the same name is above this threshold (in meters). Use FALSE to turn check off. However, it is recommended to either use raptor() or fix the feed (see cluster_stops()) in case of warnings.

Details

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

Note however that stop_name might not be a suitable identifier for a feed. It is possible that multiple stops have the same name while not being related or geographically close to each other. stop_group_distances() and cluster_stops() can help identify and fix issues with stop_names.

Value

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

Examples

```
library(dplyr)

# 1) Calculate travel times from two closely related stops
# The example dataset gtfs_duke has missing times (allowed in gtfs) which is
# why we run interpolate_stop_times beforehand
gtfs = interpolate_stop_times(gtfs_duke)

tts1 = gtfs %>%
  filter_feed_by_date("2019-08-26") %>%
  travel_times(c("Campus Dr at Arts Annex (WB)", "Campus Dr at Arts Annex (EB)"),
               time_range = c("14:00:00", "15:30:00"))

# you can use either filter_feed_by_date or filter_stop_times to prepare the feed
# the result is the same
tts2 = gtfs %>%
 filter_stop_times("2019-08-26", "14:00:00") %>%
 travel_times(c("Campus Dr at Arts Annex (WB)", "Campus Dr at Arts Annex (EB)"),
              time_range = 1.5*3600) # 1.5h after 14:00

all(tts1 == tts2)
# It's recommended to store the filtered feed, since it can be time consuming to
# run it for every travel time calculation, see the next example steps
```

```
# 2) separate filtering and travel time calculation for a more granular analysis
# stop_names in this feed are not restricted to an area, create clusters of stops to fix
nyc_path <- system.file("extdata", "nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)
nyc <- cluster_stops(nyc, group_col = "stop_name", cluster_colname = "stop_name")

# 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)

# Calculate travel times from "34 St - Herald Sq"
tts <- travel_times(stop_times, "34 St - Herald Sq", return_coords = TRUE)

# only keep journeys under one hour for plotting
tts <- tts %>% filter(travel_time <= 3600)

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

# 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 feed*

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, warnings = TRUE)
```

Arguments

`gtfs_obj`	gtfs object (i.e. a list of tables, not necessary a tidygtfs object)
`files`	A character vector containing the text files to be validated against the GTFS specification without the file extension (`txt` or `geojson`). If `NULL` (the default), the provided GTFS feed is validated against all possible GTFS text files.
`warnings`	Whether to display warning messages (defaults to `TRUE`).

Details

Note that this function just checks if required files or fields are missing. There's no validation for internal consistency (e.g. no departure times before arrival times or calendar covering a reasonable period).

Value

A `validation_result` 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 GTFS Schedule 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

```
validate_gtfs(gtfs_duke)

## Not run:
local_gtfs_path <- system.file("extdata", "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)

## End(Not run)
```

`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`	gtfs feed (tidygtfs object)
`zipfile`	path to the zip file the feed should be written to. The file is overwritten if it already exists.
`compression_level`	a number between 1 and 9, defaults to 9 (best compression).
`as_dir`	if `TRUE`, the feed is not zipped and zipfile is used as a directory path. The directory will be overwritten if it already exists.

Value

Invisibly returns `gtfs_obj`

Note

Auxiliary tidytransit tables (e.g. `dates_services`) are not exported. Calls `gtfsio::export_gtfs()` after preparing the data.

See Also

`read_gtfs()`

Index