

Package ‘parquetize’

May 10, 2023

Type Package

Title Convert Files to Parquet Format

Version 0.5.6.1

Description Collection of functions to get files in parquet format.
Parquet is a columnar storage file format <<https://parquet.apache.org/>>.
The files to convert can be of several formats
(`csv`, `RData`, `rds`, `RSQLite`,
`json`, `ndjson`, `SAS`, `SPSS`...).

License Apache License (>= 2.0)

Encoding UTF-8

Depends R (>= 3.5.0)

URL <https://ddotta.github.io/parquetize/>,
<https://github.com/ddotta/parquetize>

BugReports <https://github.com/ddotta/parquetize/issues>

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports haven, arrow, curl, readr, jsonlite, DBI, RSQLite, cli,
tidyselect, lifecycle, tools, glue, fst, dplyr

VignetteBuilder knitr

NeedsCompilation no

Author Damien Dotta [aut, cre],
Nicolas Chucho [aut]

Maintainer Damien Dotta <damien.dotta@live.fr>

Repository CRAN

Date/Publication 2023-05-10 12:00:02 UTC

R topics documented:

check_parquet	2
csv_to_parquet	3
dbi_to_parquet	5
download_extract	7
fst_to_parquet	8
get_partitions	10
json_to_parquet	11
parquetize_example	12
rbind_parquet	13
rds_to_parquet	14
sqlite_to_parquet	15
table_to_parquet	17
write_parquet_at_once	20
write_parquet_by_chunk	21
Index	24

check_parquet	<i>Check if parquet file or dataset is readable and return basic information</i>
---------------	--

Description

This function checks if a file/dataset is a valid parquet format. It will print the number of lines/columns and return a tibble on columns information.

Usage

```
check_parquet(path)
```

Arguments

path path to the file or dataset

Details

This function will :

* open the parquet dataset/file to check if it's valid * print the number of lines * print the number of columns * return a tibble with 2 columns :

* the column name (string) * the arrow type (string)

You can find a list of arrow type in the documentation [on this page](#).

Value

a tibble with information on parquet dataset/file's columns with three columns : field name, arrow type and nullable

Examples

```
# check a parquet file
check_parquet(parquetize_example("iris.parquet"))

# check a parquet dataset
check_parquet(parquetize_example("iris_dataset"))
```

csv_to_parquet	<i>Convert a csv file to parquet format</i>
----------------	---

Description

This function allows to convert a csv file to parquet format.

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

Usage

```
csv_to_parquet(  
  path_to_file,  
  url_to_csv = lifecycle::deprecated(),  
  csv_as_a_zip = lifecycle::deprecated(),  
  filename_in_zip,  
  path_to_parquet,  
  columns = "all",  
  compression = "snappy",  
  compression_level = NULL,  
  partition = "no",  
  encoding = "UTF-8",  
  ...  
)
```

Arguments

path_to_file	String that indicates the path to the input file (don't forget the extension).
url_to_csv	DEPRECATED use path_to_file instead
csv_as_a_zip	DEPRECATED
filename_in_zip	name of the csv file in the zip. Required if several csv are included in the zip.

path_to_parquet	String that indicates the path to the directory where the parquet files will be stored.
columns	Character vector of columns to select from the input file (by default, all columns are selected).
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", 'partitioning' argument must be filled in. In this case, a folder will be created for each modality of the variable filled in 'partitioning'. Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
encoding	String that indicates the character encoding for the input file.
...	additional format-specific arguments, see <code>arrow::write_parquet()</code> and <code>arrow::write_dataset()</code> for more informations.

Value

A parquet file, invisibly

Note

Be careful, if the zip size exceeds 4 GB, the function may truncate the data (because `unzip()` won't work reliably in this case - see [here](#)). In this case, it's advised to unzip your csv file by hand (for example with [7-Zip](#)) then use the function with the argument 'path_to_file'.

Examples

```
# Conversion from a local csv file to a single parquet file :

csv_to_parquet(
  path_to_file = parquetize_example("region_2022.csv"),
  path_to_parquet = tempfile(fileext=".parquet")
)

# Conversion from a local csv file to a single parquet file and select only
# few columns :

csv_to_parquet(
  path_to_file = parquetize_example("region_2022.csv"),
  path_to_parquet = tempfile(fileext = ".parquet"),
  columns = c("REG", "LIBELLE")
)

# Conversion from a local csv file to a partitioned parquet file :

csv_to_parquet(
```

```

    path_to_file = parquetize_example("region_2022.csv"),
    path_to_parquet = tempfile(fileext = ".parquet"),
    partition = "yes",
    partitioning = c("REG")
  )

# Conversion from a URL and a zipped file :

csv_to_parquet(
  path_to_file = "https://www.nomisweb.co.uk/output/census/2021/census2021-ts007.zip",
  filename_in_zip = "census2021-ts007-ctry.csv",
  path_to_parquet = tempfile(fileext = ".parquet")
)

```

dbi_to_parquet

Convert a SQL Query on a DBI connection to parquet format

Description

This function allows to convert a SQL query from a DBI to parquet format.

It handles all DBI supported databases.

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

Examples explain how to convert a query to a chunked dataset

Usage

```

dbi_to_parquet(
  conn,
  sql_query,
  path_to_parquet,
  max_memory,
  max_rows,
  chunk_memory_sample_lines = 10000,
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)

```

Arguments

conn	A DBIConnection object, as return by DBI::dbConnect
sql_query	a character string containing an SQL query (this argument is passed to DBI::dbSendQuery)
path_to_parquet	String that indicates the path to the directory where the parquet files will be stored.
max_memory	Memory size (in Mb) in which data of one parquet file should roughly fit.
max_rows	Number of lines that defines the size of the chunk. This argument can not be filled in if max_memory is used.
chunk_memory_sample_lines	Number of lines to read to evaluate max_memory. Default to 10 000.
partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", "partitioning" argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if "max_memory" or "max_rows" argument are not NULL.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	additional format-specific arguments, see arrow::write_parquet() and arrow::write_dataset() for more informations.

Value

A parquet file, invisibly

Examples

```
# Conversion from a sqlite dbi connection to a single parquet file :

dbi_connection <- DBI::dbConnect(RSQLite::SQLite(),
  system.file("extdata","iris.sqlite",package = "parquetize"))

# Reading iris table from local sqlite database
# and conversion to one parquet file :

dbi_to_parquet(
  conn = dbi_connection,
  sql_query = "SELECT * FROM iris",
  path_to_parquet = tempfile(fileext=".parquet"),
)

# Reading iris table from local sqlite database by chunk (using
# `max_memory` argument) and conversion to multiple parquet files

dbi_to_parquet(
  conn = dbi_connection,
```

```

    sql_query = "SELECT * FROM iris",
    path_to_parquet = tempdir(),
    max_memory = 2 / 1024
  )

# Using chunk and partition together is not possible directly but easy to do :
# Reading iris table from local sqlite database by chunk (using
# `max_memory` argument) and conversion to arrow dataset partitioned by
# species

# get unique values of column "iris from table "iris"
partitions <- get_partitions(dbi_connection, table = "iris", column = "Species")

# loop over those values
for (species in partitions) {
  dbi_to_parquet(
    conn = dbi_connection,
    # use glue_sql to create the query filtering the partition
    sql_query = glue::glue_sql("SELECT * FROM iris where Species = {species}",
                              .con = dbi_connection),
    # add the partition name in the output dir to respect parquet partition schema
    path_to_parquet = file.path(tempdir(), "iris", paste0("Species=", species)),
    max_memory = 2 / 1024,
  )
}

# If you need a more complicated query to get your partitions, you can use
# dbGetQuery directly :
col_to_partition <- DBI::dbGetQuery(dbi_connection, "SELECT distinct(`Species`) FROM `iris`")[,1]

```

download_extract	<i>download and uncompress file if needed</i>
------------------	---

Description

This function will download the file if the file is remote and unzip it if it is zipped. It will just return the input path argument if it's neither.

If the zip contains multiple files, you can use 'filename_in_zip' to set the file you want to unzip and use.

You can pipe output on all '*_to_parquet' functions.

Usage

```
download_extract(path, filename_in_zip)
```

Arguments

`path` the input file's path or url.
`filename_in_zip` name of the csv file in the zip. Required if several csv are included in the zip.

Value

the path to the usable (uncompressed) file, invisibly.

Examples

```
# 1. unzip a local zip file
# 2. parquetize it

file_path <- download_extract(system.file("extdata", "mtcars.csv.zip", package = "readr"))
csv_to_parquet(
  file_path,
  path_to_parquet = tempfile(fileext = ".parquet")
)

# 1. download a remote file
# 2. extract the file census2021-ts007-ctry.csv
# 3. parquetize it

file_path <- download_extract(
  "https://www.nomisweb.co.uk/output/census/2021/census2021-ts007.zip",
  filename_in_zip = "census2021-ts007-ctry.csv"
)
csv_to_parquet(
  file_path,
  path_to_parquet = tempfile(fileext = ".parquet")
)

# the file is local and not zipped so :
# 1. parquetize it

file_path <- download_extract(parquetize_example("region_2022.csv"))
csv_to_parquet(
  file_path,
  path_to_parquet = tempfile(fileext = ".parquet")
)
```


Description

This function allows to convert a fst file to parquet format.

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

Usage

```
fst_to_parquet(
  path_to_file,
  path_to_parquet,
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)
```

Arguments

path_to_file	String that indicates the path to the input file (don't forget the extension).
path_to_parquet	String that indicates the path to the directory where the parquet files will be stored.
partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", "partitioning" argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	additional format-specific arguments, see arrow::write_parquet() and arrow::write_dataset() for more informations.

Value

A parquet file, invisibly

Examples

```
# Conversion from a local fst file to a single parquet file ::
fst_to_parquet(
```

```

  path_to_file = system.file("extdata","iris.fst",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet")
)

# Conversion from a local fst file to a partitioned parquet file ::

fst_to_parquet(
  path_to_file = system.file("extdata","iris.fst",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet"),
  partition = "yes",
  partitioning = c("Species")
)

```

get_partitions	<i>get unique values from table's column</i>
----------------	--

Description

This function allows you to extract unique values from a table's column to use as partitions.

Internally, this function does "SELECT DISTINCT('mycolumn') FROM 'mytable';"

Usage

```
get_partitions(conn, table, column)
```

Arguments

conn	A 'DBIConnection' object, as return by 'DBI::dbConnect'
table	a DB table name
column	a column name for the table passed in param

Value

a vector with unique values for the column of the table

Examples

```

dbi_connection <- DBI::dbConnect(RSQLite::SQLite(),
  system.file("extdata","iris.sqlite",package = "parquetize"))

get_partitions(dbi_connection, "iris", "Species")

```

json_to_parquet	<i>Convert a json file to parquet format</i>
-----------------	--

Description

This function allows to convert a **json** or **ndjson** file to parquet format.

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

Usage

```
json_to_parquet(
  path_to_file,
  path_to_parquet,
  format = "json",
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)
```

Arguments

path_to_file	String that indicates the path to the input file (don't forget the extension).
path_to_parquet	String that indicates the path to the directory where the parquet files will be stored.
format	string that indicates if the format is "json" (by default) or "ndjson"
partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", 'partitioning' argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	additional format-specific arguments, see arrow::write_parquet() and arrow::write_dataset() for more informations.

Value

A parquet file, invisibly

Examples

```
# Conversion from a local json file to a single parquet file ::

json_to_parquet(
  path_to_file = system.file("extdata","iris.json",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet")
)

# Conversion from a local ndjson file to a partitioned parquet file ::

json_to_parquet(
  path_to_file = system.file("extdata","iris.ndjson",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet"),
  format = "ndjson"
)
```

parquetize_example *Get path to parquetize example*

Description

parquetize comes bundled with a number of sample files in its 'inst/extdata' directory. This function make them easy to access

Usage

```
parquetize_example(file = NULL)
```

Arguments

file Name of file or directory. If 'NULL', the example files will be listed.

Value

A character string

Examples

```
parquetize_example()
parquetize_example("region_2022.csv")
parquetize_example("iris_dataset")
```

rbind_parquet	<i>Function to bind multiple parquet files by row</i>
---------------	---

Description

This function read all parquet files in 'folder' argument that starts with 'output_name', combine them using rbind and write the result to a new parquet file.

It can also delete the initial files if 'delete_initial_files' argument is TRUE.

Be careful, this function will not work if files with different structures are present in the folder given with the argument 'folder'.

Usage

```
rbind_parquet(  
  folder,  
  output_name,  
  delete_initial_files = TRUE,  
  compression = "snappy",  
  compression_level = NULL  
)
```

Arguments

folder	the folder where the initial files are stored
output_name	name of the output parquet file
delete_initial_files	Boolean. Should the function delete the initial files ? By default TRUE.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.

Value

Parquet files, invisibly

Examples

```
## Not run:  
library(arrow)  
if (file.exists('output')==FALSE) {  
  dir.create("output")  
}  
  
file.create(fileext = "output/test_data1-4.parquet")
```

```

write_parquet(data.frame(
  x = c("a", "b", "c"),
  y = c(1L, 2L, 3L)
),
"output/test_data1-4.parquet")

file.create(fileext = "output/test_data4-6.parquet")
write_parquet(data.frame(
  x = c("d", "e", "f"),
  y = c(4L, 5L, 6L)
), "output/test_data4-6.parquet")

test_data <- rbind_parquet(folder = "output",
                           output_name = "test_data",
                           delete_initial_files = FALSE)

## End(Not run)

```

rds_to_parquet

Convert a rds file to parquet format

Description

This function allows to convert a rds file to parquet format.

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument ‘path_to_parquet’ must then be used;
- Convert to a partitioned parquet file. Additional arguments ‘partition’ and ‘partitioning’ must then be used;

Usage

```

rds_to_parquet(
  path_to_file,
  path_to_parquet,
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)

```

Arguments

`path_to_file` String that indicates the path to the input file (don’t forget the extension).

`path_to_parquet` String that indicates the path to the directory where the parquet files will be stored.

partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", "partitioning" argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	additional format-specific arguments, see <code>arrow::write_parquet()</code> and <code>arrow::write_dataset()</code> for more informations.

Value

A parquet file, invisibly

Examples

```
# Conversion from a local rds file to a single parquet file ::

rds_to_parquet(
  path_to_file = system.file("extdata","iris.rds",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet")
)

# Conversion from a local rds file to a partitioned parquet file ::

rds_to_parquet(
  path_to_file = system.file("extdata","iris.rds",package = "parquetize"),
  path_to_parquet = tempfile(fileext = ".parquet"),
  partition = "yes",
  partitioning = c("Species")
)
```

sqlite_to_parquet *Convert a sqlite file to parquet format*

Description

This function allows to convert a table from a sqlite file to parquet format.

The following extensions are supported : "db","sdb","sqlite","db3","s3db","sqlite3","s13","db2","s2db","sqlite2","s12".

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

Usage

```
sqlite_to_parquet(
  path_to_file,
  table_in_sqlite,
  path_to_parquet,
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)
```

Arguments

<code>path_to_file</code>	String that indicates the path to the input file (don't forget the extension).
<code>table_in_sqlite</code>	string that indicates the name of the table to convert in the sqlite file
<code>path_to_parquet</code>	String that indicates the path to the directory where the parquet files will be stored.
<code>partition</code>	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", 'partitioning' argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
<code>compression</code>	compression algorithm. Default "snappy".
<code>compression_level</code>	compression level. Meaning depends on compression algorithm.
<code>...</code>	additional format-specific arguments, see arrow::write_parquet() and arrow::write_dataset() for more informations.

Value

A parquet file, invisibly

Examples

```
# Conversion from a local sqlite file to a single parquet file :

sqlite_to_parquet(
  path_to_file = system.file("extdata", "iris.sqlite", package = "parquetize"),
  table_in_sqlite = "iris",
  path_to_parquet = tempfile(fileext = ".parquet")
)

# Conversion from a local sqlite file to a partitioned parquet file :
```

```
sqlite_to_parquet(
```



```

path_to_file = system.file("extdata","iris.sqlite",package = "parquetize"),
table_in_sqlite = "iris",
path_to_parquet = tempfile(),
partition = "yes",
partitioning = c("Species")
)

```

table_to_parquet	<i>Convert an input file to parquet format</i>
------------------	--

Description

This function allows to convert an input file to parquet format.

It handles SAS, SPSS and Stata files in a same function. There is only one function to use for these 3 cases. For these 3 cases, the function guesses the data format using the extension of the input file (in the 'path_to_file' argument).

Two conversions possibilities are offered :

- Convert to a single parquet file. Argument 'path_to_parquet' must then be used;
- Convert to a partitioned parquet file. Additional arguments 'partition' and 'partitioning' must then be used;

To avoid overcharging R's RAM, the conversion can be done by chunk. One of arguments 'max_memory' or 'max_rows' must then be used. This is very useful for huge tables and for computers with little RAM because the conversion is then done with less memory consumption. For more information, see [here](<https://ddotta.github.io/parquetize/articles/aa-conversions.html>).

Usage

```

table_to_parquet(
  path_to_file,
  path_to_parquet,
  max_memory = NULL,
  max_rows = NULL,
  chunk_size = lifecycle::deprecated(),
  chunk_memory_size = lifecycle::deprecated(),
  columns = "all",
  by_chunk = lifecycle::deprecated(),
  skip = 0,
  partition = "no",
  encoding = NULL,
  chunk_memory_sample_lines = 10000,
  compression = "snappy",
  compression_level = NULL,
  ...
)

```

Arguments

path_to_file	String that indicates the path to the input file (don't forget the extension).
path_to_parquet	String that indicates the path to the directory where the parquet files will be stored.
max_memory	Memory size (in Mb) in which data of one parquet file should roughly fit.
max_rows	Number of lines that defines the size of the chunk. This argument can not be filled in if max_memory is used.
chunk_size	DEPRECATED use max_rows
chunk_memory_size	DEPRECATED use max_memory
columns	Character vector of columns to select from the input file (by default, all columns are selected).
by_chunk	DEPRECATED use max_memory or max_rows instead
skip	By default 0. This argument must be filled in if 'by_chunk' is TRUE. Number of lines to ignore when converting.
partition	String ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", "partitioning" argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning". Be careful, this argument can not be "yes" if 'max_memory' or 'max_rows' argument are not NULL.
encoding	String that indicates the character encoding for the input file.
chunk_memory_sample_lines	Number of lines to read to evaluate max_memory. Default to 10 000.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	Additional format-specific arguments, see arrow::write_parquet() and arrow::write_dataset() for more informations.

Value

Parquet files, invisibly

Examples

```
# Conversion from a SAS file to a single parquet file :

table_to_parquet(
  path_to_file = system.file("examples", "iris.sas7bdat", package = "haven"),
  path_to_parquet = tempfile(fileext = ".parquet")
)

# Conversion from a SPSS file to a single parquet file :
```

```
table_to_parquet(  
  path_to_file = system.file("examples", "iris.sav", package = "haven"),  
  path_to_parquet = tempfile(fileext = ".parquet"),  
)  
# Conversion from a Stata file to a single parquet file without progress bar :  
  
table_to_parquet(  
  path_to_file = system.file("examples", "iris.dta", package = "haven"),  
  path_to_parquet = tempfile(fileext = ".parquet")  
)  
  
# Reading SPSS file by chunk (using `max_rows` argument)  
# and conversion to multiple parquet files :  
  
table_to_parquet(  
  path_to_file = system.file("examples", "iris.sav", package = "haven"),  
  path_to_parquet = tempfile(),  
  max_rows = 50,  
)  
  
# Reading SPSS file by chunk (using `max_memory` argument)  
# and conversion to multiple parquet files of 5 Kb when loaded (5 Mb / 1024)  
# (in real files, you should use bigger value that fit in memory like 3000  
# or 4000) :  
  
table_to_parquet(  
  path_to_file = system.file("examples", "iris.sav", package = "haven"),  
  path_to_parquet = tempfile(),  
  max_memory = 5 / 1024,  
)  
  
# Reading SAS file by chunk of 50 lines with encoding  
# and conversion to multiple files :  
  
table_to_parquet(  
  path_to_file = system.file("examples", "iris.sas7bdat", package = "haven"),  
  path_to_parquet = tempfile(),  
  max_rows = 50,  
  encoding = "utf-8"  
)  
  
# Conversion from a SAS file to a single parquet file and select only  
# few columns :  
  
table_to_parquet(  
  path_to_file = system.file("examples", "iris.sas7bdat", package = "haven"),  
  path_to_parquet = tempfile(fileext = ".parquet"),  
  columns = c("Species", "Petal_Length")  
)  
  
# Conversion from a SAS file to a partitioned parquet file :  
  
table_to_parquet(  

```

```

path_to_file = system.file("examples", "iris.sas7bdat", package = "haven"),
path_to_parquet = tempfile(),
partition = "yes",
partitioning = c("Species") # vector use as partition key
)

# Reading SAS file by chunk of 50 lines
# and conversion to multiple files with zstd, compression level 10

if (isTRUE(arrow::arrow_info()$capabilities[['zstd']])) {
  table_to_parquet(
    path_to_file = system.file("examples", "iris.sas7bdat", package = "haven"),
    path_to_parquet = tempfile(),
    max_rows = 50,
    compression = "zstd",
    compression_level = 10
  )
}

```

write_parquet_at_once *write parquet file or dataset based on partition argument*

Description

Low level function that implements the logic to write a parquet file or a dataset from data

Usage

```

write_parquet_at_once(
  data,
  path_to_parquet,
  partition = "no",
  compression = "snappy",
  compression_level = NULL,
  ...
)

```

Arguments

data	the data.frame/tibble to write
path_to_parquet	String that indicates the path to the directory where the output parquet file or dataset will be stored.
partition	string ("yes" or "no" - by default) that indicates whether you want to create a partitioned parquet file. If "yes", "partitioning" argument must be filled in. In this case, a folder will be created for each modality of the variable filled in "partitioning".

compression compression algorithm. Default "snappy".
 compression_level compression level. Meaning depends on compression algorithm.
 ... Additional format-specific arguments, see `arrow::write_parquet()`

Value

a dataset as return by `arrow::open_dataset`

Examples

```
write_parquet_at_once(iris, tempfile())  

write_parquet_at_once(iris, tempfile(), partition = "yes", partitioning = c("Species"))
```

```
write_parquet_by_chunk
```

read input by chunk on function and create dataset

Description

Low level function that implements the logic to to read input file by chunk and write a dataset.

It will:

- calculate the number of row by chunk if needed;
- loop over the input file by chunk;
- write each output files.

Usage

```
write_parquet_by_chunk(  

  read_method,  

  input,  

  path_to_parquet,  

  max_rows = NULL,  

  max_memory = NULL,  

  chunk_memory_sample_lines = 10000,  

  compression = "snappy",  

  compression_level = NULL,  

  ...  

)
```

Arguments

read_method	a method to read input files. This method take only three arguments ‘input’: some kind of data. Can be a ‘skip’: the number of row to skip ‘n_max’ : the number of row to return This method will be called until it returns a dataframe/tibble with zero row. If you need to pass more argument, you can use a [closure](http://adv-r.had.co.nz/Functional-programming.html#closures). See the last example.
input	that indicates the path to the input. It can be anything you want but more often a file’s path or a data.frame.
path_to_parquet	String that indicates the path to the directory where the output parquet file or dataset will be stored.
max_rows	Number of lines that defines the size of the chunk. This argument can not be filled in if max_memory is used.
max_memory	Memory size (in Mb) in which data of one parquet file should roughly fit.
chunk_memory_sample_lines	Number of lines to read to evaluate max_memory. Default to 10 000.
compression	compression algorithm. Default "snappy".
compression_level	compression level. Meaning depends on compression algorithm.
...	Additional format-specific arguments, see <code>arrow::write_parquet()</code>

Value

a dataset as return by `arrow::open_dataset`

Examples

```
# example with a dataframe

# we create the function to loop over the data.frame

read_method <- function(input, skip = 0L, n_max = Inf) {
  # if we are after the end of the input we return an empty data.frame
  if (skip+1 > nrow(input)) { return(data.frame()) }

  # return the n_max row from skip + 1
  input[(skip+1):(min(skip+n_max, nrow(input))),]
}

# we use it

write_parquet_by_chunk(
  read_method = read_method,
  input = mtcars,
  path_to_parquet = tempfile(),
```

```
    max_rows = 10,
  )

#
# Example with haven::read_sas
#

# we need to pass two argument beside the 3 input, skip and n_max.
# We will use a closure :

my_read_closure <- function(encoding, columns) {
  function(input, skip = 0L, n_max = Inf) {
    haven::read_sas(data_file = input,
                    n_max = n_max,
                    skip = skip,
                    encoding = encoding,
                    col_select = all_of(columns))
  }
}

# we initialize the closure

read_method <- my_read_closure(encoding = "WINDOWS-1252", columns = c("Species", "Petal_Width"))

# we use it
write_parquet_by_chunk(
  read_method = read_method,
  input = system.file("examples", "iris.sas7bdat", package = "haven"),
  path_to_parquet = tempfile(),
  max_rows = 75,
)
```

Index

[check_parquet](#), [2](#)
[csv_to_parquet](#), [3](#)

[dbi_to_parquet](#), [5](#)
[download_extract](#), [7](#)

[fst_to_parquet](#), [8](#)

[get_partitions](#), [10](#)

[json_to_parquet](#), [11](#)

[parquetize_example](#), [12](#)

[rbind_parquet](#), [13](#)
[rds_to_parquet](#), [14](#)

[sqlite_to_parquet](#), [15](#)

[table_to_parquet](#), [17](#)

[write_parquet_at_once](#), [20](#)
[write_parquet_by_chunk](#), [21](#)