Package: neopolars (via r-universe)

September 8, 2024

Title R Bindings for the 'polars' Rust Library

Version 0.0.0.9000

Description Lightning-fast 'DataFrame' library written in 'Rust'. Convert R data to 'Polars' data and vice versa. Perform fast, lazy, larger-than-memory and optimized data queries. 'Polars' is interoperable with the package 'arrow', as both are based on the 'Apache Arrow' Columnar Format.

License MIT + file LICENSE

Depends R (>= 4.2)

Imports rlang (>= 1.1.0)

Suggests bit64, blob, clock, hms, patrick, testthat (>= 3.0.0), tibble, vctrs, withr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

SystemRequirements Cargo (Rust's package manager), rustc

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

RemoteUrl https://github.com/eitsupi/neo-r-polars

RemoteRef HEAD

RemoteSha 2bed891f3ba15a1d20c4a4c478e9e76659885c54

Contents

`as.data.frame.polars_data_frame`

Export the polars object as an R DataFrame

Description

This S3 method is a shortcut of `<DataFrame>$to_struct()$to_r_vector(ensure_vector = FALSE, struct = "dataframe")`.

Usage

```
## S3 method for class 'polars_data_frame'
as.data.frame(
  x,
  ...,
  int64 = "double",
  as_clock_class = FALSE,
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

`x`	A polars object
`...`	Ignored
`int64`	Determine how to convert Polars' Int64, UInt32, or UInt64 type values to R type. One of the followings: • `"double"` (default): Convert to the R's double type. Accuracy may be degraded. • `"character"`: Convert to the R's character type.

- `"integer"`: Convert to the R's integer type. If the value is out of the range of R's integer type, export as NA_integer_.
- `"integer64"`: Convert to the bit64::integer64 class. The bit64 package must be installed. If the value is out of the range of bit64::integer64, export as bit64::NA_integer64_.

`as_clock_class` A logical value indicating whether to export datetimes and duration as the clock package's classes.

- `FALSE` (default): Duration values are exported as difftime and datetime values are exported as POSIXct. Accuracy may be degraded.
- `TRUE`: Duration values are exported as clock_duration, datetime without timezone values are exported as clock_naive_time, and datetime with timezone values are exported as clock_zoned_time. For this case, the clock package must be installed. Accuracy will be maintained.

`ambiguous` Determine how to deal with ambiguous datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. Character vector or Expression containing the followings:

- `"raise"` (default): Throw an error
- `"earliest"`: Use the earliest datetime
- `"latest"`: Use the latest datetime
- `"null"`: Return a `NA` value

`non_existent` Determine how to deal with non-existent datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. One of the followings:

- `"raise"` (default): Throw an error
- `"null"`: Return a `NA` value

Value

An R data frame

Examples

```
df <- as_polars_df(list(a = 1:3, b = 4:6))

as.data.frame(df)
```

`as.list.polars_data_frame`

Export the polars object as an R list

Description

This S3 method calls `<DataFrame>$get_columns()` or `<DataFrame>$to_r_list()` depending on the `as_series` argument.

Usage

```
## S3 method for class 'polars_data_frame'
as.list(
  x,
  ...,
  as_series = FALSE,
  int64 = "double",
  struct = "dataframe",
  as_clock_class = FALSE,
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

x	A polars object
...	Ignored
as_series	Whether to convert each column to an R vector or a Series. If TRUE, return a list of Series, otherwise a list of vectors (default).
int64	Determine how to convert Polars' Int64, UInt32, or UInt64 type values to R type. One of the followings: • "double" (default): Convert to the R's double type. Accuracy may be degraded. • "character": Convert to the R's character type. • "integer": Convert to the R's integer type. If the value is out of the range of R's integer type, export as NA_integer_. • "integer64": Convert to the bit64::integer64 class. The bit64 package must be installed. If the value is out of the range of bit64::integer64, export as bit64::NA_integer64_.
struct	Determine how to convert Polars' Struct type values to R class. One of the followings: • "dataframe" (default): Convert to the R's data.frame class. • "tibble": Convert to the tibble class. If the tibble package is not installed, a warning will be shown.
as_clock_class	A logical value indicating whether to export datetimes and duration as the clock package's classes. • FALSE (default): Duration values are exported as difftime and datetime values are exported as POSIXct. Accuracy may be degraded. • TRUE: Duration values are exported as clock_duration, datetime without timezone values are exported as clock_naive_time, and datetime with timezone values are exported as clock_zoned_time. For this case, the clock package must be installed. Accuracy will be maintained.
ambiguous	Determine how to deal with ambiguous datetimes. Only applicable when as_clock_class is set to FALSE and datetime without timezone values are exported as POSIXct. Character vector or Expression containing the followings:

- "raise" (default): Throw an error
- "earliest": Use the earliest datetime
- "latest": Use the latest datetime
- "null": Return a NA value

non_existent Determine how to deal with non-existent datetimes. Only applicable when as_clock_class is set to FALSE and datetime without timezone values are exported as POSIXct. One of the followings:

- "raise" (default): Throw an error
- "null": Return a NA value

Details

Arguments other than x and as_series are passed to <Series>$to_r_vector(), so they are ignored when as_series=TRUE.

Value

A list

See Also

- <DataFrame>$get_columns()
- <DataFrame>$to_r_list()

Examples

```
df <- as_polars_df(list(a = 1:3, b = 4:6))

as.list(df, as_series = TRUE)
as.list(df, as_series = FALSE)
```

---

as_polars_df *Create a Polars DataFrame from an R object*

Description

The as_polars_df() function creates a polars DataFrame from various R objects. Polars DataFrame is based on a sequence of Polars Series, so basically, the input object is converted to a list of Polars Series by as_polars_series(), then a Polars DataFrame is created from the list.

Usage

```
as_polars_df(x, ...)

## Default S3 method:
as_polars_df(x, ...)

## S3 method for class 'polars_series'
as_polars_df(x, ..., column_name = NULL, unnest_struct = TRUE)

## S3 method for class 'polars_data_frame'
as_polars_df(x, ...)

## S3 method for class 'polars_group_by'
as_polars_df(x, ...)

## S3 method for class 'polars_lazy_frame'
as_polars_df(
  x,
  ...,
  type_coercion = TRUE,
  predicate_pushdown = TRUE,
  projection_pushdown = TRUE,
  simplify_expression = TRUE,
  slice_pushdown = TRUE,
  comm_subplan_elim = TRUE,
  comm_subexpr_elim = TRUE,
  cluster_with_columns = TRUE,
  no_optimization = FALSE,
  streaming = FALSE
)

## S3 method for class 'list'
as_polars_df(x, ...)

## S3 method for class 'data.frame'
as_polars_df(x, ...)
```

Arguments

x	An R object.
...	Additional arguments passed to the methods.
column_name	A character or NULL. If not NULL, name/rename the Series column in the new DataFrame. If NULL, the column name is taken from the Series name.
unnest_struct	A logical. If TRUE (default) and the Series data type is a struct, the <Series>$struct$unnest() method is used to create a DataFrame from the struct Series. In this case, the column_name argument is ignored.
type_coercion	A logical, indicats type coercion optimization.

`predicate_pushdown`
: A logical, indicats predicate pushdown optimization.

`projection_pushdown`
: A logical, indicats projection pushdown optimization.

`simplify_expression`
: A logical, indicats simplify expression optimization.

`slice_pushdown`
: A logical, indicats slice pushdown optimization.

`comm_subplan_elim`
: A logical, indicats tring to cache branching subplans that occur on self-joins or unions.

`comm_subexpr_elim`
: A logical, indicats tring to cache common subexpressions.

`cluster_with_columns`
: A logical, indicats to combine sequential independent calls to with_columns.

`no_optimization`
: A logical. If `TRUE`, turn off (certain) optimizations.

`streaming`
: A logical. If `TRUE`, process the query in batches to handle larger-than-memory data. If `FALSE` (default), the entire query is processed in a single batch. Note that streaming mode is considered unstable. It may be changed at any point without it being considered a breaking change.

Details

The default method of `as_polars_df()` throws an error, so we need to define methods for the classes we want to support.

S3 method for list:

- The argument `...` (except `name`) is passed to `as_polars_series()` for each element of the list.
- All elements of the list must be converted to the same length of Series by `as_polars_series()`.
- The name of the each element is used as the column name of the DataFrame. For unnamed elements, the column name will be an empty string `""` or if the element is a Series, the column name will be the name of the Series.

S3 method for data.frame:

- The argument `...` (except `name`) is passed to `as_polars_series()` for each column.
- All columns must be converted to the same length of Series by `as_polars_series()`.

S3 method for polars_series:

This is a shortcut for `<Series>$to_frame()` or `<Series>$struct$unnest()`, depending on the `unnest_struct` argument and the Series data type. The `column_name` argument is passed to the `name` argument of the `$to_frame()` method.

S3 method for polars_lazy_frame:

This is a shortcut for `<LazyFrame>$collect()`.

Value

A polars DataFrame

See Also

- `<DataFrame>$to_r_list()`: Export the DataFrame as an R list of R vectors.

Examples

```
# list
as_polars_df(list(a = 1:2, b = c("foo", "bar")))

# data.frame
as_polars_df(data.frame(a = 1:2, b = c("foo", "bar")))

# polars_series
s_int <- as_polars_series(1:2, "a")
s_struct <- as_polars_series(
  data.frame(a = 1:2, b = c("foo", "bar")),
  "struct"
)

## Use the Series as a column
as_polars_df(s_int)
as_polars_df(s_struct, column_name = "values", unnest_struct = FALSE)

## Unnest the struct data
as_polars_df(s_struct)
```

---

as_polars_series *Create a Polars Series from an R object*

Description

The `as_polars_series()` function creates a polars Series from various R objects. The Data Type of the Series is determined by the class of the input object.

Usage

```
as_polars_series(x, name = NULL, ...)

## Default S3 method:
as_polars_series(x, name = NULL, ...)

## S3 method for class 'polars_series'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'polars_data_frame'
```

```
as_polars_series(x, name = NULL, ...)

## S3 method for class 'double'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'integer'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'character'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'logical'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'raw'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'factor'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'Date'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'POSIXct'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'difftime'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'hms'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'blob'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'array'
as_polars_series(x, name = NULL, ...)

## S3 method for class '`NULL`'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'list'
as_polars_series(x, name = NULL, ..., strict = FALSE)

## S3 method for class 'AsIs'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'data.frame'
```

```
as_polars_series(x, name = NULL, ...)

## S3 method for class 'integer64'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'vctrs_unspecified'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'vctrs_rcrd'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'clock_time_point'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'clock_sys_time'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'clock_zoned_time'
as_polars_series(x, name = NULL, ...)

## S3 method for class 'clock_duration'
as_polars_series(x, name = NULL, ...)
```

Arguments

x	An R object.
name	A single string or NULL. Name of the Series. Will be used as a column name when used in a polars DataFrame. When not specified, name is set to an empty string.
...	Additional arguments passed to the methods.
strict	A logical value to indicate whether throwing an error when the input list's elements have different data types. If FALSE (default), all elements are automatically cast to the super type, or, casting to the super type is failed, the value will be null. If TRUE, the first non-NULL element's data type is used as the data type of the inner Series.

Details

The default method of `as_polars_series()` throws an error, so we need to define S3 methods for the classes we want to support.

S3 method for list and list based classes:

In R, a list can contain elements of different types, but in Polars (Apache Arrow), all elements must have the same type. So the `as_polars_series()` function automatically casts all elements to the same type or throws an error, depending on the `strict` argument. If you want to create a list with all elements of the same type in R, consider using the `vctrs::list_of()` function.

Since a list can contain another list, the `strict` argument is also used when creating Series from the inner list in the case of classes constructed on top of a list, such as data.frame or vctrs_rcrd.

S3 method for polars_data_frame:
This method is a shortcut for `<DataFrame>$to_struct()`.

Value

A polars Series

See Also

- `<Series>$to_r_vector()`: Export the Series as an R vector.
- `as_polars_df()`: Create a Polars DataFrame from an R object.

Examples

```
# double
as_polars_series(c(NA, 1, 2))

# integer
as_polars_series(c(NA, 1:2))

# character
as_polars_series(c(NA, "foo", "bar"))

# logical
as_polars_series(c(NA, TRUE, FALSE))

# raw
as_polars_series(charToRaw("foo"))

# factor
as_polars_series(factor(c(NA, "a", "b")))

# Date
as_polars_series(as.Date(c(NA, "2021-01-01")))

# POSIXct with timezone
as_polars_series(as.POSIXct(c(NA, "2021-01-01 00:00:00"), "UTC"))

# POSIXct without timezone
as_polars_series(as.POSIXct(c(NA, "2021-01-01 00:00:00")))

# difftime
as_polars_series(as.difftime(c(NA, 1), units = "days"))

# NULL
as_polars_series(NULL)

# list
as_polars_series(list(NA, NULL, list(), 1, "foo", TRUE))

## 1st element will be `null` due to the casting failure
as_polars_series(list(list("bar"), "foo"))
```

```
# data.frame
as_polars_series(
  data.frame(x = 1:2, y = c("foo", "bar"), z = I(list(1, 2)))
)

# vctrs_unspecified
if (requireNamespace("vctrs", quietly = TRUE)) {
  as_polars_series(vctrs::unspecified(3L))
}

# hms
if (requireNamespace("hms", quietly = TRUE)) {
  as_polars_series(hms::as_hms(c(NA, "01:00:00")))
}

# blob
if (requireNamespace("blob", quietly = TRUE)) {
  as_polars_series(blob::as_blob(c(NA, "foo", "bar")))
}

# integer64
if (requireNamespace("bit64", quietly = TRUE)) {
  as_polars_series(bit64::as.integer64(c(NA, "9223372036854775807")))
}

# clock_naive_time
if (requireNamespace("clock", quietly = TRUE)) {
  as_polars_series(clock::naive_time_parse(c(
    NA,
    "1900-01-01T12:34:56.123456789",
    "2020-01-01T12:34:56.123456789"
  ), precision = "nanosecond"))
}

# clock_duration
if (requireNamespace("clock", quietly = TRUE)) {
  as_polars_series(clock::duration_nanoseconds(c(NA, 1)))
}
```

`as_tibble.polars_data_frame` *Export the polars object as a tibble data frame*

Description

This S3 method is basically a shortcut of `<DataFrame>$to_struct()$to_r_vector(ensure_vector = FALSE, struct = "tibble")`. Additionally, you can check or repair the column names by specifying the `.name_repair` argument. Because polars DataFrame allows empty column name, which is not generally valid column name in R data frame.

Usage

```
## S3 method for class 'polars_data_frame'
as_tibble(
  x,
  ...,
  .name_repair = "check_unique",
  int64 = "double",
  as_clock_class = FALSE,
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

x	A polars object
...	Ignored
.name_repair	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
int64	Determine how to convert Polars' Int64, UInt32, or UInt64 type values to R type. One of the followings: • "double" (default): Convert to the R's double type. Accuracy may be degraded. • "character": Convert to the R's character type. • "integer": Convert to the R's integer type. If the value is out of the range of R's integer type, export as NA_integer_. • "integer64": Convert to the bit64::integer64 class. The bit64 package must be installed. If the value is out of the range of bit64::integer64, export as bit64::NA_integer64_.
as_clock_class	A logical value indicating whether to export datetimes and duration as the clock package's classes. • FALSE (default): Duration values are exported as difftime and datetime values are exported as POSIXct. Accuracy may be degraded. • TRUE: Duration values are exported as clock_duration, datetime without timezone values are exported as clock_naive_time, and datetime with timezone values are exported as clock_zoned_time. For this case, the clock package must be installed. Accuracy will be maintained.

`ambiguous` Determine how to deal with ambiguous datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. Character vector or Expression containing the followings:

- `"raise"` (default): Throw an error
- `"earliest"`: Use the earliest datetime
- `"latest"`: Use the latest datetime
- `"null"`: Return a `NA` value

`non_existent` Determine how to deal with non-existent datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. One of the followings:

- `"raise"` (default): Throw an error
- `"null"`: Return a `NA` value

Value

A tibble

See Also

- `as.data.frame(<polars_data_frame>)`: Export the polars object as a basic data frame.

Examples

```
# Polars DataFrame may have empty column name
df <- pl$DataFrame(x = 1:2, c("a", "b"))
df

# Without checking or repairing the column names
tibble::as_tibble(df, .name_repair = "minimal")

# You can make that unique
tibble::as_tibble(df, .name_repair = "unique")
```

---

cs *Polars column selector function namespace*

Description

`cs` is an environment class object that stores all selector functions of the R Polars API which mimics the Python Polars API. It is intended to work the same way in Python as if you had imported Python Polars Selectors with `import polars.selectors as cs`.

Usage

```
cs
```

Format

An object of class `polars_object` of length 2.

Examples

```
cs

# How many members are in the `cs` environment?
length(cs)
```

`dataframe__get_columns`

Get the DataFrame as a List of Series

Description

Get the DataFrame as a List of Series

Usage

```
dataframe__get_columns()
```

Value

A list of Series

See Also

- `<DataFrame>$to_r_list()`: Similar to this method but returns a list of vectors instead of Series.

Examples

```
df <- pl$DataFrame(foo = c(1, 2, 3), bar = c(4, 5, 6))
df$get_columns()

df <- pl$DataFrame(
  a = 1:4,
  b = c(0.5, 4, 10, 13),
  c = c(TRUE, TRUE, FALSE, TRUE)
)
df$get_columns()
```

`dataframe__to_r_list` *Export the polars DataFrame as an R list of R vectors*

Description

This method is a shortcut of `<DataFrame>$to_struct()$to_r_vector(ensure_vector = TRUE)`. Different from `<DataFrame>$get_columns()`, all columns are exported as R objects.

Usage

```
dataframe__to_r_list(
  ...,
  int64 = "double",
  as_clock_class = FALSE,
  struct = "dataframe",
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

`...`	These dots are for future extensions and must be empty.
`int64`	Determine how to convert Polars' Int64, UInt32, or UInt64 type values to R type. One of the followings: • `"double"` (default): Convert to the R's double type. Accuracy may be degraded. • `"character"`: Convert to the R's character type. • `"integer"`: Convert to the R's integer type. If the value is out of the range of R's integer type, export as NA_integer_. • `"integer64"`: Convert to the bit64::integer64 class. The bit64 package must be installed. If the value is out of the range of bit64::integer64, export as bit64::NA_integer64_.
`as_clock_class`	A logical value indicating whether to export datetimes and duration as the clock package's classes. • `FALSE` (default): Duration values are exported as difftime and datetime values are exported as POSIXct. Accuracy may be degraded. • `TRUE`: Duration values are exported as clock_duration, datetime without timezone values are exported as clock_naive_time, and datetime with timezone values are exported as clock_zoned_time. For this case, the clock package must be installed. Accuracy will be maintained.
`struct`	Determine how to convert Polars' Struct type values to R class. One of the followings: • `"dataframe"` (default): Convert to the R's data.frame class. • `"tibble"`: Convert to the tibble class. If the tibble package is not installed, a warning will be shown.

`ambiguous` Determine how to deal with ambiguous datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. Character vector or Expression containing the followings:

- `"raise"` (default): Throw an error
- `"earliest"`: Use the earliest datetime
- `"latest"`: Use the latest datetime
- `"null"`: Return a `NA` value

`non_existent` Determine how to deal with non-existent datetimes. Only applicable when `as_clock_class` is set to `FALSE` and datetime without timezone values are exported as POSIXct. One of the followings:

- `"raise"` (default): Throw an error
- `"null"`: Return a `NA` value

Value

A list containing vectors and R dataframes

See Also

- `<DataFrame>$get_columns()`: Export the polars DataFrame as a list of Series.
- `<Series>$to_r_vector()`: Export the polars Series as an R vector.

Examples

```
df <- pl$DataFrame(foo = c(1, 2, 3), bar = c(4, 5, 6))
df$to_r_list()

df <- pl$DataFrame(
  a = 1:4,
  b = c(0.5, 4, 10, 13),
  c = c(TRUE, TRUE, FALSE, TRUE)
)
df$to_r_list()
```

---

dataframe__to_struct *Convert a DataFrame to a Series of type Struct*

Description

Convert a DataFrame to a Series of type Struct

Usage

```
dataframe__to_struct(name = "")
```

Arguments

name	A character. Name for the struct Series.

Value

A Series of the struct type

See Also

- as_polars_series()

Examples

```
df <- pl$DataFrame(
  a = 1:5,
  b = c("one", "two", "three", "four", "five"),
)
df$to_struct("nums")
```

---

expr_dt_replace_time_zone

Replace time zone for an expression of type Datetime

Description

Different from convert_time_zone, this will also modify the underlying timestamp and will ignore the original time zone.

Usage

```
expr_dt_replace_time_zone(
  time_zone,
  ...,
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

time_zone	NULL or a character time zone from base::OlsonNames(). Pass NULL to unset time zone.
...	These dots are for future extensions and must be empty.
ambiguous	Determine how to deal with ambiguous datetimes. Character vector or Expression containing the followings: • "raise" (default): Throw an error • "earliest": Use the earliest datetime

- "latest": Use the latest datetime
- "null": Return a null value

non_existent Determine how to deal with non-existent datetimes. One of the followings:

- "raise" (default): Throw an error
- "null": Return a null value

---

lazyframe__collect *Materialize this LazyFrame into a DataFrame*

Description

By default, all query optimizations are enabled. Individual optimizations may be disabled by setting the corresponding parameter to FALSE.

Usage

```
lazyframe__collect(
  ...,
  type_coercion = TRUE,
  predicate_pushdown = TRUE,
  projection_pushdown = TRUE,
  simplify_expression = TRUE,
  slice_pushdown = TRUE,
  comm_subplan_elim = TRUE,
  comm_subexpr_elim = TRUE,
  cluster_with_columns = TRUE,
  no_optimization = FALSE,
  streaming = FALSE,
  `_eager` = FALSE
)
```

Arguments

...	These dots are for future extensions and must be empty.
type_coercion	A logical, indicats type coercion optimization.
predicate_pushdown	A logical, indicats predicate pushdown optimization.
projection_pushdown	A logical, indicats projection pushdown optimization.
simplify_expression	A logical, indicats simplify expression optimization.
slice_pushdown	A logical, indicats slice pushdown optimization.
comm_subplan_elim	A logical, indicats tring to cache branching subplans that occur on self-joins or unions.

comm_subexpr_elim
: A logical, indicats tring to cache common subexpressions.

cluster_with_columns
: A logical, indicats to combine sequential independent calls to with_columns.

no_optimization
: A logical. If TRUE, turn off (certain) optimizations.

streaming
: A logical. If TRUE, process the query in batches to handle larger-than-memory data. If FALSE (default), the entire query is processed in a single batch. Note that streaming mode is considered unstable. It may be changed at any point without it being considered a breaking change.

_eager
: A logical, indicates to turn off multi-node optimizations and the other optimizations. This option is intended for internal use only.

Value

A polars DataFrame

Examples

```
lf <- pl$LazyFrame(
  a = c("a", "b", "a", "b", "b", "c"),
  b = 1:6,
  c = 6:1,
)
lf$group_by("a")$agg(pl$all()$sum())$collect()

# Collect in streaming mode
lf$group_by("a")$agg(pl$all()$sum())$collect(
  streaming = TRUE
)
```

---

pl *Polars top-level function namespace*

Description

pl is an environment class object that stores all the top-level functions of the R Polars API which mimics the Python Polars API. It is intended to work the same way in Python as if you had imported Python Polars with `import polars as pl`.

Usage

```
pl
```

Format

An object of class polars_object of length 37.

Examples

```
pl

# How many members are in the `pl` environment?
length(pl)

# Create a polars DataFrame
# In Python:
# ```python
# >>> import polars as pl
# >>> df = pl.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6]})
# ```
# In R:
df <- pl$DataFrame(a = c(1, 2, 3), b = c(4, 5, 6))
df
```

---

pl_api_register_series_namespace

Registering custom functionality with a polars Series

Description

Registering custom functionality with a polars Series

Usage

```
pl_api_register_series_namespace(name, ns_fn)
```

Arguments

name	Name under which the functionality will be accessed.
ns_fn	A function returns a new environment with the custom functionality. See examples for details.

Examples

```
# s: polars series
math_shortcuts <- function(s) {
  # Create a new environment to store the methods
  self <- new.env(parent = emptyenv())

  # Store the series
  self$`_s` <- s

  # Add methods
  self$square <- function() self$`_s` * self$`_s`
  self$cube <- function() self$`_s` * self$`_s` * self$`_s`

  # Set the class
```

```
  class(self) <- c("polars_namespace_series", "polars_object")

  # Return the environment
  self
}

pl$api$register_series_namespace("math", math_shortcuts)

s <- as_polars_series(c(1.5, 31, 42, 64.5))
s$math$square()$rename("s^2")

s <- as_polars_series(1:5)
s$math$cube()$rename("s^3")
```

`pl__DataFrame` *Polars DataFrame class (*`polars_data_frame`*)*

Description

DataFrames are two-dimensional data structure representing data as a table with rows and columns. Polars DataFrames are similar to R Data Frames. R Data Frame's columns are R vectors, while Polars DataFrame's columns are Polars Series.

Usage

```
pl__DataFrame(...)
```

Arguments

`...`	<dynamic-dots> Name-value pairs of objects to be converted to polars Series by the `as_polars_series()` function. Each Series will be used as a column of the DataFrame. All values must be the same length. Each name will be used as the column name. If the name is empty, the original name of the Series will be used.

Details

The `pl$DataFrame()` function mimics the constructor of the DataFrame class of Python Polars. This function is basically a shortcut for `list(...) |> as_polars_df()`, so each argument in `...` is converted to a Polars Series by `as_polars_series()` and then passed to `as_polars_df()`.

Value

A polars DataFrame

Active bindings

- `columns`: `$columns` returns a character vector with the names of the columns.
- `dtypes`: `$dtypes` returns a nameless list of the data type of each column.
- `schema`: `$schema` returns a named list with the column names as names and the data types as values.
- `shape`: `$shape` returns a integer vector of length two with the number of rows and columns of the DataFrame.
- `height`: `$height` returns a integer with the number of rows of the DataFrame.
- `width`: `$width` returns a integer with the number of columns of the DataFrame.

Examples

```
# Constructing a DataFrame from vectors:
pl$DataFrame(a = 1:2, b = 3:4)

# Constructing a DataFrame from Series:
pl$DataFrame(pl$Series("a", 1:2), pl$Series("b", 3:4))

# Constructing a DataFrame from a list:
data <- list(a = 1:2, b = 3:4)

## Using the as_polars_df function (recommended)
as_polars_df(data)

## Using dynamic dots feature
pl$DataFrame(!!!data)

# Active bindings:
df <- pl$DataFrame(a = 1:3, b = c("foo", "bar", "baz"))

df$columns
df$dtypes
df$schema
df$shape
df$height
df$width
```

`pl__LazyFrame` *Polars LazyFrame class (*`polars_lazy_frame`*)*

Description

Representation of a Lazy computation graph/query against a DataFrame. This allows for whole-query optimisation in addition to parallelism, and is the preferred (and highest-performance) mode of operation for polars.

Usage

```
pl__LazyFrame(...)
```

Arguments

...	<dynamic-dots> Name-value pairs of objects to be converted to polars Series by the as_polars_series() function. Each Series will be used as a column of the DataFrame. All values must be the same length. Each name will be used as the column name. If the name is empty, the original name of the Series will be used.

Details

The pl$LazyFrame(...) function is a shortcut for pl$DataFrame(...)$lazy().

Value

A polars LazyFrame

Examples

```
# Constructing a LazyFrame from vectors:
pl$LazyFrame(a = 1:2, b = 3:4)

# Constructing a LazyFrame from Series:
pl$LazyFrame(pl$Series("a", 1:2), pl$Series("b", 3:4))

# Constructing a LazyFrame from a list:
data <- list(a = 1:2, b = 3:4)

## Using dynamic dots feature
pl$LazyFrame(!!!data)
```

---

pl__Series *Polars Series class (*polars_series*)*

Description

Series are a 1-dimensional data structure, which are similar to R vectors. Within a series all elements have the same Data Type.

Usage

```
pl__Series(name = NULL, values = NULL)
```

Arguments

name	A single string or NULL. Name of the Series. Will be used as a column name when used in a polars DataFrame. When not specified, name is set to an empty string.
values	An R object. Passed as the x param of as_polars_series().

Details

The pl$Series() function mimics the constructor of the Series class of Python Polars. This function calls as_polars_series() internally to convert the input object to a Polars Series.

Active bindings

- dtype: $dtype returns the data type of the Series.
- name: $name returns the name of the Series.
- shape: $shape returns a integer vector of length two with the number of length of the Series and width of the Series (always 1).

See Also

- as_polars_series()

Examples

```
# Constructing a Series by specifying name and values positionally:
s <- pl$Series("a", 1:3)
s

# Active bindings:
s$dtype
s$name
s$shape
```

---

series_struct_unnest *Convert this struct Series to a DataFrame with a separate column for each field*

Description

Convert this struct Series to a DataFrame with a separate column for each field

Usage

```
series_struct_unnest()
```

Value

A polars DataFrame

See Also

- as_polars_df()

Examples

```
s <- as_polars_series(data.frame(a = c(1, 3), b = c(2, 4)))
s$struct$unnest()
```

---

series__to_frame *Cast this Series to a DataFrame*

Description

Cast this Series to a DataFrame

Usage

```
series__to_frame(name = NULL)
```

Arguments

name	A character or NULL. If not NULL, name/rename the Series column in the new DataFrame. If NULL, the column name is taken from the Series name.

Value

A polars DataFrame

See Also

- as_polars_df()

Examples

```
s <- pl$Series("a", c(123, 456))
df <- s$to_frame()
df

df <- s$to_frame("xyz")
df
```

`series__to_r_vector` *Export the Series as an R vector*

Description

Export the Series as an R vector. But note that the Struct data type is exported as a data.frame by default for consistency, and a data.frame is not a vector. If you want to ensure the return value is a vector, please set `ensure_vector = TRUE`, or use the `as.vector()` function instead.

Usage

```
series__to_r_vector(
  ...,
  ensure_vector = FALSE,
  int64 = "double",
  struct = "dataframe",
  as_clock_class = FALSE,
  ambiguous = "raise",
  non_existent = "raise"
)
```

Arguments

`...`	These dots are for future extensions and must be empty.
`ensure_vector`	A logical value indicating whether to ensure the return value is a vector. When the Series has the Struct data type and this argument is `FALSE` (default), the return value is a data.frame, not a vector (`is.vector(<data.frame>)` is `FALSE`). If `TRUE`, return a named list instead of a data.frame.
`int64`	Determine how to convert Polars' Int64, UInt32, or UInt64 type values to R type. One of the followings: • `"double"` (default): Convert to the R's double type. Accuracy may be degraded. • `"character"`: Convert to the R's character type. • `"integer"`: Convert to the R's integer type. If the value is out of the range of R's integer type, export as NA_integer_. • `"integer64"`: Convert to the bit64::integer64 class. The bit64 package must be installed. If the value is out of the range of bit64::integer64, export as bit64::NA_integer64_.
`struct`	Determine how to convert Polars' Struct type values to R class. One of the followings: • `"dataframe"` (default): Convert to the R's data.frame class. • `"tibble"`: Convert to the tibble class. If the tibble package is not installed, a warning will be shown.
`as_clock_class`	A logical value indicating whether to export datetimes and duration as the clock package's classes.

- FALSE (default): Duration values are exported as difftime and datetime values are exported as POSIXct. Accuracy may be degraded.
- TRUE: Duration values are exported as clock_duration, datetime without timezone values are exported as clock_naive_time, and datetime with timezone values are exported as clock_zoned_time. For this case, the clock package must be installed. Accuracy will be maintained.

ambiguous — Determine how to deal with ambiguous datetimes. Only applicable when as_clock_class is set to FALSE and datetime without timezone values are exported as POSIXct. Character vector or Expression containing the followings:

- "raise" (default): Throw an error
- "earliest": Use the earliest datetime
- "latest": Use the latest datetime
- "null": Return a NA value

non_existent — Determine how to deal with non-existent datetimes. Only applicable when as_clock_class is set to FALSE and datetime without timezone values are exported as POSIXct. One of the followings:

- "raise" (default): Throw an error
- "null": Return a NA value

Details

The class/type of the exported object depends on the data type of the Series as follows:

- Boolean: logical.
- UInt8, UInt16, Int8, Int16, Int32: integer.
- Int64, UInt32, UInt64: double, character, integer, or bit64::integer64, depending on the int64 argument.
- Float32, Float64: double.
- Decimal: double.
- String: character.
- Categorical: factor.
- Date: Date.
- Time: hms::hms.
- Datetime (without timezone): POSIXct or clock_naive_time, depending on the as_clock_class argument.
- Datetime (with timezone): POSIXct or clock_zoned_time, depending on the as_clock_class argument.
- Duration: difftime or clock_duration, depending on the as_clock_class argument.
- Binary: blob::blob.
- Null: vctrs::unspecified.
- List, Array: vctrs::list_of.
- Struct: data.frame or tibble, depending on the struct argument. If ensure_vector = TRUE, the top-level Struct is exported as a named list for to ensure the return value is a vector.

Value

A vector

Examples

```
# Struct values handling
series_struct <- as_polars_series(
  data.frame(
    a = 1:2,
    b = I(list(data.frame(c = "foo"), data.frame(c = "bar")))
  )
)
series_struct

## Export Struct as data.frame
series_struct$to_r_vector()

## Export Struct as data.frame,
## but the top-level Struct is exported as a named list
series_struct$to_r_vector(ensure_vector = TRUE)

## Export Struct as tibble
series_struct$to_r_vector(struct = "tibble")

## Export Struct as tibble,
## but the top-level Struct is exported as a named list
series_struct$to_r_vector(struct = "tibble", ensure_vector = TRUE)

# Integer values handling
series_uint64 <- as_polars_series(
  c(NA, "0", "4294967295", "18446744073709551615")
)$cast(pl$UInt64)
series_uint64

## Export UInt64 as double
series_uint64$to_r_vector(int64 = "double")

## Export UInt64 as character
series_uint64$to_r_vector(int64 = "character")

## Export UInt64 as integer (overflow occurs)
series_uint64$to_r_vector(int64 = "integer")

## Export UInt64 as bit64::integer64 (overflow occurs)
if (requireNamespace("bit64", quietly = TRUE)) {
  series_uint64$to_r_vector(int64 = "integer64")
}

# Duration values handling
series_duration <- as_polars_series(
  c(NA, -1000000000, -10, -1, 1000000000)
)$cast(pl$Duration("ns"))
```

```
series_duration

## Export Duration as difftime
series_duration$to_r_vector(as_clock_class = FALSE)

## Export Duration as clock_duration
if (requireNamespace("clock", quietly = TRUE)) {
  series_duration$to_r_vector(as_clock_class = TRUE)
}

# Datetime values handling
series_datetime <- as_polars_series(
  as.POSIXct(
    c(NA, "1920-01-01 00:00:00", "1970-01-01 00:00:00", "2020-01-01 00:00:00"),
    tz = "UTC"
  )
)$cast(pl$Datetime("ns", "UTC"))
series_datetime

## Export zoned datetime as POSIXct
series_datetime$to_r_vector(as_clock_class = FALSE)

## Export zoned datetime as clock_zoned_time
if (requireNamespace("clock", quietly = TRUE)) {
  series_datetime$to_r_vector(as_clock_class = TRUE)
}
```

Index