Package: pyramidi (via r-universe)

September 1, 2024

Title Generate and Manipulate Midi Data in R Data Frames

Version 0.2.1

Config/testthat/edition 3

Description Import the python libraries miditapyr and mido to read in midi file data in pandas DataFrames. These can then be imported in R via reticulate. The event-based midi data is widened to facilitate the manipulation and plotting of note-based structures as in music21. The data frame format allows for an easy implementation of many music data manipulations.

Config/reticulate list(packages = list(list(package = ``miditapyr", version = ``0.1.1", pip = TRUE)))

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports reticulate, tibble, dplyr, stringr, tidyr, magrittr, R6, fluidsynth, details, zeallot, glue

URL https://github.com/urswilke/pyramidi, https://urswilke.github.io/pyramidi/

Depends R (>= 2.10)

Suggests pichor, purrr, ggplot2, spelling, knitr, rmarkdown, testthat, htmltools, covr, forcats, DiagrammeR, tuneR

Remotes mikldk/pichor

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

RdMacros details

Language en-US

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

RemoteUrl https://github.com/urswilke/pyramidi

RemoteRef HEAD

RemoteSha 4ee6ee9b3296fae1f36c3f1810860406bc58b8f8

Contents

`install_miditapyr` *Install miditapyr python package*

Description

Wrapper around `reticulate::py_install()` to install the miditapyr python package

Usage

```
install_miditapyr(envname = NULL, method = "auto", conda = "auto", pip = TRUE)
```

Arguments

`envname`	The name, or full path, of the environment in which python packages are to be installed. When `NULL` (default), the active environment (`RETICULATE_PYTHON_ENV` variable) will be used; if that is unset, then the `"r-reticulate"` environment will be used.
`method`	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
`conda`	The path to a conda executable. Use "auto" to allow reticulate to automatically find an appropriate conda binary. See Finding Conda for more details.
`pip`	Boolean; use pip for package installation? This is only relevant when Conda environments are used, as otherwise packages will be installed from the Conda repositories.

Details

From `reticulate::py_install()`: On Linux and OS X the "virtualenv" method will be used by default ("conda" will be used if virtualenv isn't available). On Windows, the "conda" method is always used.

merge_midi_frames *Merge dataframes transformed back to long format*

Description

Merge dataframes transformed back to long format, remove added columns and transform to the right chronological order in order to replace the original midi_frame_unnested object.

Usage

```
merge_midi_frames(df_meta, df_notes_long, df_not_notes)
```

Arguments

df_meta, df_notes_long, df_not_notes	Dataframes in the format of split_midi_frame().

Value

Merges the input dataframes, arranges by i_track & (absolute) ticks, and calculates time (in relative ticks since the last event).

See Also

Other Split/merge meta/notes/non-note events: split_midi_frame()

Examples

```
## Not run:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mf <- miditapyr$MidiFrames(midi_file_string)

dfm <- tab_measures(mf$midi_frame_unnested$df, ticks_per_beat = mf$midi_file$ticks_per_beat)

l <- split_midi_frame(dfm)

df_notes_long <- pivot_long_notes(l$df_notes_wide)

merge_midi_frames(l$df_meta, df_notes_long, l$df_not_notes)

## End(Not run)
```

MidiFramer *MidiFramer class*

Description

The class MidiFramer can be used to read midi files to dataframes in order to facilitate to manipulate the data from R. You can also create midi data from R without reading it from a file. The data is transformed to various formats. One of the MidiFramer fields is a MidiFrames object of the python miditapyr package. Its method write_file() can be used to write the data back to a midi file.

Details

- See the vignette("pyramidi") for a brief usage introduction how to manipulate midi data.
- The vignette("compose") shows a more extended example how to generate midi files from scratch.
- vignette("package_workflow") shows in detail the structure of the MidiFramer class.
- vignette("functions_usage") illustrates the low-level functions of the pyramidi package. that MidiFramer objects use under the hood.

Public fields

midi_file_string Path to the midi file.

mf miditapyr$MidiFrames object.

dfm result of tab_measures().

df_notes_long Result of pivot_long_notes().

df_meta, df_not_notes, df_notes_wide Results of split_midi_frame().

midi_frame_mod Result of merge_midi_frames().

params Parameters used in internal functions; Named list; params$columns_to_add is passed to tab_measures(columns_to_add).

Active bindings

df_meta, df_not_notes, df_notes_wide Results of split_midi_frame().

ticks_per_beat Set ticks per beat of MidiFrames()mfmidi_file. The value of ticks_per_beat passed should be integer. When a value is passed, the field mf$midi_file$ticks_per_beat is modified.

Methods

Public methods:

- MidiFramer$new()
- MidiFramer$update_notes_wide()
- MidiFramer$populate_r_fields()
- MidiFramer$play()

- MidiFramer$clone()

Method `new()`**:** Initialize a MidiFramer object

Usage:

```
MidiFramer$new(midi_file_string = NULL)
```

Arguments:

`midi_file_string` Path to the midi file; if NULL (the default), an empty `MidiFramer` object is created.

Method `update_notes_wide()`**:** Update a MidiFramer object with modified notes

Usage:

```
MidiFramer$update_notes_wide(mod)
```

Arguments:

`mod` Dataframe or function returning a dataframe of the format of `df_notes_wide`.

Examples:

```
\dontrun{
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
# Function to replace every note with a random midi note between 60 & 71:
mod <- function(dfn) {
  n_notes <- sum(!is.na(dfn$note))
  dfn %>% dplyr::mutate(note = ifelse(
    !is.na(note),
    sample(60:71, n_notes, TRUE),
    note
  ))
}
set.seed(123)
mfr$update_notes_wide(mod)
mfr$play()
# You can pass functions to the $update_notes_wide() method (as above), but
# you can also modify the dataframe directly and pass it. Therefore, the
# following results in the same:
set.seed(123)
df_mod <- mod(mfr$df_notes_wide)
mfr$update_notes_wide(df_mod)
mfr$play()
}
```

Method `populate_r_fields()`**:** Populate the fields of a MidiFramer object

This can also be used to recalculate all the object's attributes, when a value in params is changed (see examples).

Usage:

```
MidiFramer$populate_r_fields()
```

Examples:

```
\dontrun{
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
mfr$params$columns_to_add <- c("m", "b", "t", "time")
mfr$populate_r_fields()
}
```

Method `play()`**:** Play midi from MidiFramer object. Writes a midi file and either playing it in the R console (`live` = `TRUE`), or otherwise (`live` = `FALSE`) writes an audio file and adding an html audio player in an Rmarkdown (/quarto?) document. Calls `player()` helper function.

WARNING: Setting `overwrite` = `TRUE` (the default!!) will DELETE the specified audio files!!! (see more details below)

Usage:

```
MidiFramer$play(
  audiofile = tempfile("mf_out_", fileext = ".mp3"),
  soundfont = fluidsynth::soundfont_path(),
  midifile = gsub("\\....$", ".mid", audiofile),
  live = interactive(),
  verbose = FALSE,
  overwrite = TRUE,
  ...
)
```

Arguments:

`audiofile` Path to the audiofile to be synthesized. If audiofile of type mp3, it will first be synthesized to wav, and then converted to mp3 with ffmpeg; (character string).

`soundfont` path to sf2 sound font (character string); if not specified, the default soundfont of the fluidsynth package (`fluidsynth::soundfont_path()`) will be (downloaded if not present and) used.

`midifile` Path to the midi file to synthesize on; (character string).

`live` logical; if `TRUE` the synthesized midi is directly played in the console. If `FALSE` an audio html tag is written. This will generate a small audio player when knitting an Rmd document (and probably also Quarto qmd files; I didn't check).

`verbose` logical whether to print command line output; defaults to FALSE

`overwrite` logical; defaults to TRUE; if file exists and overwrite = FALSE, the existing files will not be overwritten and the function errors out.

`...` Arguments passed to the fluidsynth functions (`fluidsynth::midi_play` or `fluidsynth::midi_convert` depending on the value of `live`).

`@seealso` player

Examples:

```
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
mfr$play()
# The play method does basically this:
\dontrun{
midi_out <- "my_output.mid"
mp3file <- "test.mp3"
```

```
mfr$mf$write_file(midi_out)
fluidsynth::midi_convert(midi_out, output = mp3file)
# `overwrite` = TRUE overwrites midi_out & mp3file
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MidiFramer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
## Create a MidiFramer object from a midi file:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
MidiFramer$new(midi_file_string)

## ------------------------------------------------
## Create empty MidiFramer object to illustrate
## the use of the `ticks_per_beat` active binding:
## ------------------------------------------------

mfr <- MidiFramer$new()
# Print default value of empty MidiFile object:
mfr$mf$midi_file$ticks_per_beat
# Modify it with the active binding ticks_per_beat:
mfr$ticks_per_beat <- 960L
# Print it again:
mfr$mf$midi_file$ticks_per_beat

## End(Not run)

## ------------------------------------------------
## Method `MidiFramer$update_notes_wide`
## ------------------------------------------------

## Not run:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
# Function to replace every note with a random midi note between 60 & 71:
mod <- function(dfn) {
  n_notes <- sum(!is.na(dfn$note))
  dfn %>% dplyr::mutate(note = ifelse(
    !is.na(note),
    sample(60:71, n_notes, TRUE),
    note
  ))
}
set.seed(123)
mfr$update_notes_wide(mod)
```

```
mfr$play()
# You can pass functions to the $update_notes_wide() method (as above), but
# you can also modify the dataframe directly and pass it. Therefore, the
# following results in the same:
set.seed(123)
df_mod <- mod(mfr$df_notes_wide)
mfr$update_notes_wide(df_mod)
mfr$play()

## End(Not run)

## ------------------------------------------------
## Method `MidiFramer$populate_r_fields`
## ------------------------------------------------

## Not run:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
mfr$params$columns_to_add <- c("m", "b", "t", "time")
mfr$populate_r_fields()

## End(Not run)

## ------------------------------------------------
## Method `MidiFramer$play`
## ------------------------------------------------

midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mfr <- MidiFramer$new(midi_file_string)
mfr$play()
# The play method does basically this:
## Not run:
midi_out <- "my_output.mid"
mp3file <- "test.mp3"
mfr$mf$write_file(midi_out)
fluidsynth::midi_convert(midi_out, output = mp3file)
# `overwrite` = TRUE overwrites midi_out & mp3file

## End(Not run)
```

`miditapyr` *Miditapyr python module*

Description

Miditapyr python module

Usage

```
miditapyr
```

Format

An object of class `python.builtin.module` (inherits from `python.builtin.object`) of length 1.

Value

reticulate python module

Examples

```
## Not run:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
miditapyr$MidiFile(midi_file_string)

## End(Not run)
```

---

midi_defs *Table of midi notes*

Description

A table containing the midi pitches and their note names

Usage

```
midi_defs
```

Format

A data frame with 128 rows and 2 variables:

note integer from 0 - 127 containing the midi pitches

note_name name of the note and the octave

Examples

```
midi_defs
```

mido *Mido python module*

Description

Mido python module

Usage

```
mido
```

Format

An object of class `python.builtin.module` (inherits from `python.builtin.object`) of length 1.

Value

reticulate python module

Examples

```
## Not run:
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mido$MidiFile(midi_file_string)

## End(Not run)
```

piano_keys_coordinates *Table of piano keys coordinates*

Description

A table containing the piano keys coordinates. Each piano key is centered at the midi note value it represents.

Usage

```
piano_keys_coordinates
```

Format

A data frame with 127 rows and 6 variables:

midi integer from 0 - 127 containing the midi pitches

layer integer 1 or 2 depending on whether key is white (1) or black (2)

xmin xmin coordinate of the key

ymin ymin coordinate of the key

xmax xmax coordinate of the key

ymax ymax coordinate of the key

Examples

```
library(ggplot2)
# Print dataframe:
piano_keys_coordinates
# Plot the keyboard:
piano_keys_coordinates %>%
  # plot white keys first that they don't cover half of the black keys:
  dplyr::arrange(layer) %>%
  ggplot(aes(ymin = ymin, ymax = ymax, xmin = xmin, xmax = xmax, fill = factor(layer))) +
  geom_rect(color = "black", show.legend = FALSE) +
  scale_fill_manual(values = c("#ffffdd", "#113300")) +
  coord_fixed(ratio = 10)
```

---

pivot_long_notes *Write* note_on *and* note_off *events in two lines (wide to long)*

Description

Write note_on and note_off events in two lines (wide to long)

Usage

```
pivot_long_notes(df_notes_wide)
```

Arguments

df_notes_wide	notes dataframe in wide format

Value

Transforms notes in wide dataframe format to long format.

See Also

Other Pivot midi frame functions: pivot_wide_notes()

Examples

```
## Not run:
mid_file_str <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mido_mid_file <- mido$MidiFile(mid_file_str)
dfc <- miditapyr$frame_midi(mido_mid_file)
ticks_per_beat = mido_mid_file$ticks_per_beat
df <- dfc %>%
  miditapyr$unnest_midi()

## End(Not run)
## Not run:
dfm <- tab_measures(df, ticks_per_beat)
library(zeallot)
c(df_meta, df_notes) %<-% miditapyr$split_df(dfm)
dfw <- df_notes %>% pivot_wide_notes()
dfw %>% pivot_long_notes()

## End(Not run)
```

---

pivot_wide_notes *Write* note_on *and* note_off *events in the same line (long to wide)*

Description

Write note_on and note_off events in the same line (long to wide)

Usage

```
pivot_wide_notes(df_measures)
```

Arguments

df_measures	data.frame resulting of miditapyr$mido_midi_df() and then running tab_measures() (see example)

Value

A data.frame with the following columns pivoted to wide: c("m", "b", "t", "ticks", "time", "velocity"). Every column is pivoted to wide with the suffix "_note_on" & "_note_off" showing the values of the original column as a prefix. See ?tab_measures for an explanation of the meaning of these columns.

See Also

Other Pivot midi frame functions: pivot_long_notes()

Examples

```
## Not run:
mid_file_str <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mido_mid_file <- mido$MidiFile(mid_file_str)
dfc <- miditapyr$frame_midi(mido_mid_file)
ticks_per_beat = mido_mid_file$ticks_per_beat
df <- dfc %>%
  miditapyr$unnest_midi()

## End(Not run)
## Not run:
dfm <- tab_measures(df, ticks_per_beat)
library(zeallot)
c(df_meta, df_notes) %<-% miditapyr$split_df(dfm)
df_notes %>% pivot_wide_notes()

## End(Not run)
```

`player` *Play midi file*

Description

Play midi from MidiFramer object. Helper function to synthesize midi, either:

- directly in the console,
- or generate audio files and include a small player in html documents.

This helper function is also called in the `MidiFramer$play()` method.

WARNING: Setting `overwrite = TRUE` (the default!!) will DELETE the specified audio files!!! (see more details below)

Usage

```
player(
  midifile,
  soundfont = fluidsynth::soundfont_path(),
  output = gsub(".mid$", ".mp3", midifile),
  live = interactive(),
  verbose = interactive(),
  overwrite = TRUE,
  ...
)
```

Arguments

midifile	Path to the midi file to synthesize on; character string.
soundfont	path to sf2 sound font (character string); if not specified, the default soundfont of the fluidsynth package (fluidsynth::soundfont_path()) will be (downloaded if not present and) used.
output	Path to the audiofile to be synthesized. (character string).
live	logical; if TRUE the synthesized midi is directly played in the console. If FALSE an audio html tag is written. This will generate a small audio player when knitting an Rmd document (and probably also Quarto qmd files; I didn't check).
verbose	logical whether to print command line output; defaults to FALSE
overwrite	logical; defaults to TRUE; if file exists and overwrite = FALSE, the existing files will not be overwritten and the function errors out.
...	Arguments passed to the fluidsynth functions (fluidsynth::midi_play or fluidsynth::midi_convert depending on the value of live).

Value

If live = TRUE, nothing is returned. If live = FALSE, a html audio tag is returned that will render as a small audio player when knitting an Rmd document. The audio player can then play the generated output audio file.

Examples

```
midi_file_string <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
midi_file_string |> player()

# The player is a small helper function to do basically this:
## Not run:
midifile <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mp3file <- "test.mp3"
fluidsynth::midi_convert(midifile, output = mp3file)
# `overwrite` = TRUE overwrites the mp3 file if previously existing.

## End(Not run)
```

---

split_midi_frame *Split unnested midi dataframe into parts*

Description

Unnested midi frame with time data from tab_measures() is split into 3 parts:

- df_meta: Consisting of all the meta events in the midi data.
- df_not_notes: Containing all midi events that are not meta and not note_on or note_off.
- df_notes_wide: All note_on or note_off events and furthermore transformed to wide format by pivot_note_wide.

Usage

```
split_midi_frame(dfm)
```

Arguments

dfm	result of tab_measures()

Value

df_meta, df_not_notes & df_notes_wide

See Also

Other Split/merge meta/notes/non-note events: merge_midi_frames()

Examples

```
## Not run:
mid_file_str <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mido_mid_file <- mido$MidiFile(mid_file_str)
dfc <- miditapyr$frame_midi(mido_mid_file)
ticks_per_beat = mido_mid_file$ticks_per_beat
df <- dfc %>%
  miditapyr$unnest_midi()

## End(Not run)
## Not run:
dfm <- tab_measures(df, ticks_per_beat)
split_midi_frame(dfm)

## End(Not run)
```

---

tab_measures *Tabulate measure related data in the midi event data*

Description

This function mainly adds ticks which is the absolute value of mido's time (measured in ticks) for every track. Furthermore, this absolute time value is translated to the further columns of beats (quarter notes; "b"), measures (bars; "m") and the time in seconds ("t"). The function outputs these additional columns only if they are specified in the argument columns_to_add (see details).

Usage

```
tab_measures(df, ticks_per_beat, columns_to_add = "b")
```

Arguments

df	Note event data.frame resulting of miditapyr$unnest_midi()
ticks_per_beat	integer resulting of miditapyr$mido_midi_df()
columns_to_add	Character vector of the columns to be added to the result (see section "Details"). Subset of c("m", "b", "t", "time").

Details

The function transforms the unnested midi frame by first transforming the midi time (measured in ticks passed relative to the previous event) to absolute time for every track. Also the auxiliary variable i_note is added (indexing the occurrence of each note for every track). This is needed to allow to pivot the notes to wide format and back to long with pivot_wide_notes() and pivot_long_notes().

- m: measure (bar)
- b: beat (quarter note)
- t: time in seconds

Value

A data.frame with the following columns added: ticks, i_note and the columns specified in columns_to_add.

Examples

```
## Not run:
mid_file_str <- system.file("extdata", "test_midi_file.mid", package = "pyramidi")
mido_mid_file <- mido$MidiFile(mid_file_str)
dfc <- miditapyr$frame_midi(mido_mid_file)
ticks_per_beat = mido_mid_file$ticks_per_beat
df <- dfc %>%
  miditapyr$unnest_midi()

## End(Not run)
## Not run:
tab_measures(df, ticks_per_beat)

## End(Not run)
```

Index