Package 'rpymat'

August 22, 2023

Type Package

Title Easy to Configure an Isolated 'Python' Environment

Version 0.1.7

Description Aims to create a single isolated 'Miniconda'
and 'Python' environment for reproducible pipeline scripts.
The package provides utilities to run system command within the 'conda'
environment, making it easy to install, launch, manage, and stop
'Jupyter-lab'.

License Apache License (>= 2)

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

URL https://github.com/dipterix/rpymat, http://dipterix.org/rpymat/

BugReports https://github.com/dipterix/rpymat/issues

Imports utils, reticulate (>= 1.21), fastmap (>= 1.1.0), rappdirs (>=
0.3.3), glue (>= 1.4.2), IRkernel (>= 1.3), jsonlite (>=
1.7.3), rstudioapi (>= 0.13)

Suggests spelling

NeedsCompilation no

Author Zhengjia Wang [cph, aut, cre]

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2023-08-22 03:20:02 UTC

R topics documented:

`choose-file` *Choose file or directory to open via 'Python'*

Description

Choose a directory, one or multiple files to open, or choose a file to save.

Usage

```
choose_fileopen(
  initialfile = NULL,
  multiple = FALSE,
  title = ifelse(multiple, "Choose Files", "Choose a File"),
  message = "",
  verbose = FALSE,
  force = FALSE
)

choose_filesave()

choose_directory(
  initialdir = NULL,
  title = "Choose a Directory",
  message = "",
  verbose = FALSE
)
```

Arguments

`initialfile`, `initialdir`	initial selection of file or directory
`multiple`	whether to open multiple files
`title`, `message`	dialogue title and message
`verbose`	whether to verbose debug information
`force`	whether to force using `'Python'` when native R functions are available, default is false

Details

Base-R has `file.choose` function to choose files. However, users cannot select multiple files nor directories. These functions fill the gap by using `'Python' 'tkinter'` package. Please make sure that one-time setup function `configure_conda` has executed before running these functions.

The functions must run as interactive mode. If you run the functions on a server, most likely you will get nothing. The functions themselves do not check if you are running under interactive sessions. You must check by yourself.

Value

User-selected paths. If the users select nothing, then `NULL` will be returned. For multiple file selection, multiple paths will be returned.

Examples

```
if(interactive()) {
  choose_fileopen(multiple = TRUE)
}
```

---

conda-env *'Miniconda' environment*

Description

These functions/variables are used to configure 'Miniconda' environment.

Usage

```
CONDAENV_NAME(env_name)

conda_path()

conda_bin()

env_path()

list_pkgs(...)

configure_matlab(matlab, python_ver = "auto")

configure_conda(
  python_ver = "auto",
  packages = NULL,
```

```
  matlab = NULL,
  update = FALSE,
  force = FALSE,
  standalone = FALSE
)

remove_conda(ask = TRUE)

add_packages(packages = NULL, python_ver = "auto", ...)

ensure_rpymat(verbose = TRUE, cache = TRUE)

matlab_engine()

call_matlab(
  fun,
  ...,
  .options = getOption("rpymat.matlab_opt", "-nodesktop -nojvm"),
  .debug = getOption("rpymat.debug", FALSE)
)
```

Arguments

env_name	alternative environment name to use; default is "rpymat-conda-env"
...	for add_packages, these are additional parameters passing to conda_install; for call_matlab, ... are the parameters passing to fun
matlab	'Matlab' path to add to the configuration path; see 'Details'
python_ver	python version to use; see 'Configuration'
packages	additional python or conda packages to install
update	whether to update conda; default is false
force	whether to force install the 'Miniconda' even a previous version exists; default is false. Setting false=TRUE rarely works. Please see 'Configuration'.
standalone	whether to install conda regardless of existing conda environment
ask	whether to ask for user's agreement to remove the repository. This parameter should be true if your functions depend on remove_conda (see 'CRAN Repository Policy'). This argument might be removed and force to be interactive in the future.
verbose	whether to print messages
cache	whether to use cached configurations; default is true
fun	'Matlab' function name, character (experimental)
.options	'Matlab' compiler options
.debug	whether to enable debug mode

Value

None

Background & Objectives

Package `reticulate` provides sophisticated tool-sets that allow us to call `python` functions within `R`. However, the installation of 'Miniconda' and `python` can be tricky on many platforms, for example, the 'M1' chip, or some other 'ARM' machines. The package `rpymat` provides easier approach to configure on these machines with totally isolated environments. Any modifications to this environment will not affect your other set ups.

Since 2014, 'Matlab' has introduced its official compiler for `python`. The package `rpymat` provides a simple approach to link the compiler, provided that you have proper versions of 'Matlab' installed. Here is a list of 'Matlab' versions with official compilers and their corresponding `python` versions.

Configuration

If 'Matlab' compiler is not to be installed, In most of the cases, function `configure_conda` with default arguments automatically downloads the latest 'Miniconda' and configures the latest `python`. If any other versions of 'Miniconda' is ought to be installed, please set options `"reticulate.miniconda.url"` to change the source location.

If 'Matlab' is to be installed, please specify the 'Matlab' path when running `configure_conda`. If the environment has been setup, `configure_matlab` can link the 'Matlab' compilers without removing the existing environment. For 'ARM' users, unfortunately, there will be no 'Matlab' support as the compilers are written for the 'Intel' chips.

Initialization

Once `conda` and `python` environment has been installed, make sure you run `ensure_rpymat()` before running any `python` code. This function will make sure correct compiler is linked to your current `R` session.

Examples

```
# The script will interactively install \code{conda} to `R_user_dir`
## Not run:

# Install conda and python 3.9

configure_conda(python_ver = '3.9')


# Add packages h5py, pandas, jupyter

add_packages(c('h5py', 'pandas', 'jupyter'))

# Add pip packages

add_packages("itk", pip = TRUE)

# Initialize the isolated environment

ensure_rpymat()
```

```
# Remove the environment

remove_conda()


## End(Not run)
```

---

jupyter *Install, register, launch 'Jupyter' notebook to the virtual environment*

Description

Install, register, launch 'Jupyter' notebook to the virtual environment

Usage

```
add_jupyter(..., register_R = TRUE)

jupyter_bin()

jupyter_register_R(
  user = NULL,
  name = "ir",
  displayname = "R",
  rprofile = NULL,
  prefix = NULL,
  sys_prefix = NULL,
  verbose = getOption("verbose")
)

jupyter_options(
  root_dir,
  host = "127.0.0.1",
  port = 8888,
  open_browser = FALSE,
  token = rand_string(),
  base_url = "/jupyter/"
)

jupyter_launch(
  host = "127.0.0.1",
  port = 8888,
  open_browser = TRUE,
  workdir = getwd(),
  async = FALSE,
  ...,
```

```
  dry_run = FALSE
)

jupyter_check_launch(
  port = 8888,
  host = "127.0.0.1",
  open_browser = TRUE,
  workdir = getwd(),
  async = "auto",
  ...
)

jupyter_server_list()

jupyter_server_stop(port, ...)

jupyter_server_stop_all(...)
```

Arguments

`...`	for `add_jupyter`, these are additional parameters passed to `jupyter_register_R`; for `jupyter_launch`, these are additional parameters passed to `jupyter_options`
`register_R`	whether to register `IRkernel` to the notebook
`user, name, displayname, rprofile, prefix, sys_prefix, verbose`	see `installspec`
`root_dir, workdir`	default root directory of the notebook
`host, port`	'IP' and port of the hosting 'URL'
`open_browser`	whether to open the browser once launched
`token`	access token of the notebook
`base_url`	base address, default is `'/jupyter/'`
`async`	whether to open the notebook in the background
`dry_run`	whether to display the command instead of executing them; used to debug the code

Value

`jupyter_bin` returns the 'Jupyter' notebook binary path; `jupyter_options` returns the 'Jupyter' configuration in strings; `jupyter_server_list` returns a table of existing local 'Jupyter' server hosts, ports, and tokens; `jupyter_check_launch` returns true if a new server has been created, or false if there has been an existing server at the port; other functions return nothing.

Examples

```
## Not run:

# Requires installation of conda
```

```
library(rpymat)

# Install conda, if you have done so, skip
configure_conda()


# Install Jupyter notebook
add_jupyter(register_R = TRUE)


# Utility functions
jupyter_bin()

# Please install `dipsaus` package to enable `async=TRUE` with
# better experience
jupyter_launch(async = FALSE, open_browser = TRUE)



## End(Not run)
```

---

py_builtin *Get 'Python' built-in object*

Description

Get 'Python' built-in object

Usage

```
py_builtin(name, convert = FALSE)
```

Arguments

name	object name
convert	see import_builtins

Value

A python built-in object specified by name

Examples

```
if(interactive() && dir.exists(env_path())) {


# ------ Basic case: use python `int` as an R function ---------
```

```
py_int <- py_builtin("int", convert = TRUE)

# a is an R object now
a <- py_int(9)
print(a)
class(a)

# ------ Use python `int` as a Python function -----------------
py_int2 <- py_builtin("int", convert = FALSE)

# b in a python object
b <- py_int2(9)

# There is no '[1] ' when printing
print(b)
class(b)

# convert to R object
py_to_r(b)



}
```

`py_list` *List in 'Python'*

Description

List in 'Python'

Usage

```
py_list(..., convert = FALSE)
```

Arguments

`...`	passing to `list` ('Python')
`convert`	whether to convert the results back into R; default is no

Value

List instance, or an R vector if converted

Examples

```
if(interactive() && dir.exists(env_path())) {

  py_list(list(1,2,3))
  py_list(c(1,2,3))

  py_list(array(1:9, c(3,3)))
  py_list(list(list(1:3), letters[1:3]))

}
```

`py_slice` *Slice index in 'Python' arrays*

Description

Slice index in 'Python' arrays

Usage

```
py_slice(...)
```

Arguments

`...`	passing to `slice` ('Python')

Value

Index slice instance

Examples

```
if(interactive() && dir.exists(env_path())) {

  x <- np_array(array(seq(20), c(4, 5)))

  # equivalent to x[::2]
  x[py_slice(NULL, NULL, 2L)]

}
```

read_xlsx *Read data frame from a 'xlsx' file*

Description

Tries to use 'readxl' package or 'pandas' to read data frame.

Usage

```
read_xlsx(
  path,
  sheet = NULL,
  method = c("auto", "pandas", "readxl"),
  n_max = Inf,
  ...
)
```

Arguments

path	'xlsx' file path
sheet	either a character or an integer of which spread-sheet to read; the number starts from 1
method	which method to use for reading the 'xlsx' file; choices are 'auto' (automatically find proper method), 'pandas' (use pandas.read_xlsx), or 'readxl' (use the corresponding R package)
n_max	maximum number of rows (excluding headers) to read
...	passed to 'Python' function pandas.read_xlsx or readxl::read_excel, depending on method

Value

A data.frame table

Examples

```
## Not run:

rpymat::read_xlsx("Book1.xlsx", sheet = 1)

rpymat::read_xlsx("Book1.xlsx", sheet = "sheet1")


## End(Not run)
```

repl_python *Enable interactive 'python' from R*

Description

Allows users to type 'python' command from R console for quick code evaluation or debugging.

Usage

```
repl_python(...)
```

Arguments

...	passed to repl_python in 'reticulate' package

Value

See repl_python

reticulate-reexports *Wrappers around 'reticulate' package*

Description

Almost the same with 'reticulate' functions, with rpymat enabled by default and some minor changes (see parameter convert and local)

Usage

```
import_main(convert = FALSE)

tuple(..., convert = FALSE)

py_tuple(..., convert = FALSE)

py_help(object)

np_array(data, ...)

import(module, as = NULL, convert = FALSE, delay_load = FALSE)

r_to_py(x, convert = FALSE)

py_to_r(x)

py_to_r_wrapper(x)
```

```
py_str(object, ...)

py_run_string(code, local = TRUE, convert = FALSE)

py_bool(x)

py_dict(keys, values, convert = FALSE)

py_call(x, ...)

py_del_attr(x, name)

py_del_item(x, name)

py_eval(code, convert = FALSE)

py_get_attr(x, name, silent = FALSE)

py_set_attr(x, name, value)

py_get_item(x, key, silent = FALSE)

py_set_item(x, name, value)

py_len(x, default = NULL)

py_none()
```

Arguments

`convert`	whether to convert `'Python'` objects to R; default is `FALSE`. This is different to `'reticulate'`, but less error prone: users must explicitly convert `'Python'` objects to R.
`object, data, x, code, keys, values, ...`	passed to corresponding `'reticulate'` functions as data inputs
`module, as, delay_load`	import `'Python'` module as alias
`local`	whether to execute code locally so the memory sets free when the function ends; default is true
`name, silent, key, value, default`	other parameters passing to the `'reticulate'` functions

Value

`'Python'` built-in objects

Examples

```
library(rpymat)
if(interactive() && dir.exists(env_path())) {

  # tuple
  x <- tuple(1, 2, "a")
  print(x)

  # convert to R object
  py_to_r(x)

  # convert R object to python
  y <- r_to_py(list(a = 1, b = "s"))

  # get element
  py_get_item(y, "a")

  # get missing element
  py_get_item(y, "c", silent = TRUE)

}
```

`rpymat-python-main` *Get 'Python' main process environment*

Description

`py` automatically converts 'Python' objects to R objects. `import_main` does not convert by default; see 'Examples' for details.

Usage

```
py
```

Format

An object of class `NULL` of length 0.

Value

The 'Python' main process as a module

Examples

```
if(interactive() && dir.exists(env_path())) {

py_no_convert <- rpymat::import_main(convert = FALSE)

py$a <- matrix(seq_len(16), 4)

py_no_convert$a

py$a

}
```

---

run_command *Execute command with additional environments*

Description

Enables 'conda' environment

Usage

```
cmd_create(command, shell, use_glue = TRUE)

cmd_set_env(command, key, value, quote = TRUE, quote_type = "cmd")

cmd_set_workdir(command, workdir)

cmd_set_conda(command, conda_path, env_path)

cmd_build(command, .env = parent.frame(), ...)

detect_shell(suggest = NULL)

run_command(
  command,
  shell = detect_shell(),
  use_glue = FALSE,
  enable_conda = TRUE,
  stdout = "",
  stderr = "",
  stdin = "",
  input = NULL,
  env_list = list(),
  wait = TRUE,
  timeout = 0,
```

```
  ...,
  workdir = getwd(),
  dry_run = FALSE,
  print_cmd = dry_run,
  glue_env = parent.frame()
)
```

Arguments

command	system command
shell	shell type
use_glue	whether to glue the command; default is false
key, value	environment variable key and value
quote, quote_type	whether to quote the environment variables and what quote type should use; see shQuote
workdir	the working directory
conda_path	'conda' path; default is conda_path
env_path	'conda' environment path; default is env_path
suggest	suggested shell type; default is 'cmd' on windows, or 'bash' on others
enable_conda	whether to activate 'conda'
stdout, stderr, stdin, input, wait, timeout, ...	passed to system2
env_list	a key-value pairs of environment variables
dry_run	whether to dry-run the command (do not execute, simply returns the command), useful to debug
print_cmd	whether to print the command out
glue_env, .env	the environment to evaluate variables when use_glue is true

Value

All the functions return a list with class rpymat_system_command except for run_command, which returns the exit code by system2.

Examples

```
run_command("conda install -y numpy", dry_run = TRUE)


a <- "This is a message"
run_command('echo "{a}"', dry_run = TRUE,
            enable_conda = FALSE, use_glue = TRUE)


## Not run:
```

```
# Use `jupyter_launch()` instead. This is just a demonstration
run_command('"{jupyter_bin()}" server list', use_glue = TRUE)


## End(Not run)
```

`run_pyscript` *Run 'Python' script*

Description

A wrapper of `py_run_file`, but with `rpymat` enabled

Usage

```
run_script(
  x,
  work_dir = NULL,
  local = FALSE,
  convert = FALSE,
  globals = list()
)

run_pyscript(
  x,
  work_dir = NULL,
  local = FALSE,
  convert = FALSE,
  globals = list()
)

run_pystring(
  code,
  work_dir = NULL,
  local = FALSE,
  convert = FALSE,
  globals = list()
)
```

Arguments

`x`	'Python' script path
`work_dir`	working directory of the script
`local, convert`	passed to `py_run_file`
`globals`	named list of global R variables used by 'Python' script
`code`	'Python' code

Value

The values returned by py_run_file

Examples

```
## Not run:

# Please configure conda environment first

x <- tempfile()
writeLines(c(
  "import re",
  "zipcode = re.findall(r'[0-9]{5,6}', r.address)"
), con = x)

address <- '2341 Main St., 72381'
rpymat::run_script(x)

py$zipcode


## End(Not run)
```

Index