

Using python for postgresql uploads

James P. Gilbert

2025-09-24

Contents

Introduction	1
Installing psycpg2	1
Using a virtualenv	1
Using conda or system python installs	1
Usage within functions	2

Introduction

Note, this feature should be considered experimental until further notice.

The use of DatabaseConnector with postgresql without bulk uploads functionality will often be slow and require the installation and configuration of postgresql binaries on the system. This may be challenging or restricted in many environments. Similarly, this method requires writing a data.frame to disk which will be prohibitively slow if data is already in a csv format.

As a consequence, this package supports bulk uploading through python with a small amount of configuration. This uses no r-memory; csv files are transferred directly through python and will be considerably faster

This process uses the **psycpg2** python library, which can be installed via compilation or in binary form. This process demonstrates usage with the **psycpg2-binary** package.

Installing psycpg2

Using a virtualenv

Result model manager provides an interactive function for enabling the python library. If psycpg2 this function will do nothing. However, if there is no available binary (and the **reticulate** package is not installed) you will be asked to install these packages. Do do this run the following:

```
ResultModelManager::enablePythonUploads()
```

Alternatively you can specify this manually

```
ResultModelManager::install_psycpg2()
```

Using conda or system python installs

Please consult the reticulate documentation on how to install the **psycpg2-binary** package.

Usage within functions

By default, this functionality will not be enabled when uploading tables and the function `pyUploadCsv` will fail. To enable, and directly upload a csv, try the following example code.

```
ResultModelManager::enablePythonUploads()
connectionDetails <- DatabaseConnector::createConnectionDetails(
  dbms = "postgresql",
  server = "myserver.com",
  port = 5432,
  password = "s",
  user = "me",
  database = "some_db"
)
connection <- DatabaseConnector::connect(connectionDetails)
readr::write_csv(
  data.frame(
    id = 1:1e6,
    paste(1:1e6, "bottle(s) on the wall")
  ),
  "my_massive_csv.csv"
)

ResultModelManager::pyUploadCsv(connection,
  table = "my_table",
  filepath = "my_massive_csv.csv",
  schema = "my_schema"
)
```

Note that you are not required to call `ResultModelManager::enablePythonUploads()` every time. As an alternative, add the following line to your `.Renv` file (note that this will automatically assume that setup of python libraries has been completed)

```
RMM_USE_PYTHON_UPLOADS=TRUE
```

The astute reader will realize that this approach requires an IO call, writing the CSV to disk. In many situations this will be a major bottleneck. A much more sensible approach is to use a string buffer. Thankfully, the author of this package has provided such an interface!

```
ResultModelManager::pyUploadDataframe(connection,
  table = "my_table",
  filepath = "my_massive_csv.csv",
  schema = "my_schema"
)
```

Note - that this approach is actually already implemented for you when you use `uploadResults` functionality. That's right - if you call `ResultModelManager::enablePythonUploads()` (and you are using postgres) you will be able to upload your large R `data.frames` to postgres!

```
ResultModelManager::enablePythonUploads()

ResultModelManager::uploadResults(
  connectionDetails,
  schema = "my_schema",
  resultsFolder = "my_results_folder",
  tablePrefix = "cm_",
  purgeSiteDataBeforeUploading = FALSE,
```

```
specifications = getResultsDataModelSpec()  
)
```

Better yet, calling `ResultModelManager::enablePythonUploads()` before uploading results from any OHDSI package will automatically give you this fast(er) upload functionality.