We're excited to announce that `reticulate` 1.14 is now available on CRAN! You can install it with:

```
install.packages("reticulate")
```

With this release, we are introducing a major new feature: `reticulate` can now automatically configure a Python environment for the user, in coordination with any loaded R packages that depend on `reticulate`. This means that:

- *R package authors* can declare their Python dependency requirements to `reticulate` in a standardized way, and `reticulate` will automatically prepare the Python environment for the user; and
- *R users* can use R packages depending on `reticulate`, without having to worry about managing a Python installation / environment themselves.

Ultimately, the goal is for R packages using `reticulate` to be able to operate just like any other R package, without forcing the R user to grapple with issues around Python environment management.

We'd also like to give a special thanks to Ryan Hafen for his work on the rminiconda package. The work in this release borrows from many of the ideas he put together as part of the `rminiconda` package.

R Packages and Python – The Problem

Currently, *reticulated* R packages typically have to document for users how their Python dependencies should be installed. For example, packages like tensorflow provide helper functions (e.g. `tensorflow::install_tensorflow()`):

```
library(tensorflow)
install_tensorflow()
# use tensorflow
```

This approach requires users to manually download, install, and configure an appropriate version of Python themselves. In addition, if the user has *not* downloaded an appropriate version of Python, then the version discovered on the user's system may not conform with the requirements imposed by the Python TensorFlow package – leading to more trouble.

Fixing this often requires instructing the user to install Python, and then use `reticulate` APIs (e.g. `reticulate::use_python()` and other tools) to find and use that version of Python. This is, understandably, more cognitive overhead than one normally might want to impose on the users of one's package.

R Packages and Python – The Solution

Our goal in this release, then, is to make it possible for `reticulate` to *automatically* prepare a Python environment for the user, without requiring any explicit user intervention. In other words, R packages that wrap Python packages through `reticulate` should *feel* just like any other R package. The R user should only need to write:

```
library(tensorflow)
# use tensorflow
```

and `reticulate` will automatically prepare and install TensorFlow (prompting the user as necessary).

To that end, we've made the following changes. If the user has not explicitly instructed `reticulate` to use a pre-existing Python environment, then:

1. `reticulate` will prompt the user to download and install Miniconda;
2. `reticulate` will prepare a default `r-reticulate` Conda environment, using (currently) Python 3.6

and NumPy;

3. When Python is initialized, `reticulate` will query any loaded R packages for their Python dependencies, and install those dependencies into the aforementioned `r-reticulate` Conda environment.

Ultimately, this leads to an experience where R packages wrapping Python packages can work just like any other R package – the user will normally not need to intervene and manually configure their Python environment.

All that said, all of the pre-existing workflows for configuring Python remain available for users who require them. If you need to manually take control of the Python environment you use in your projects, you can still do so.

Currently, automatic Python environment configuration will only happen when using the aforementioned `reticulate` Miniconda installation. However, you can still call

```
reticulate::configure_environment()
```

to manually install any declared Python dependencies into your active Python environment.

Declaring a Python Dependency

R packages which want to declare a Python package dependency to `reticulate` can do so in their `DESCRIPTION` file. For example, suppose we were building a package `rscipy` which wrapped the Python SciPy package. We could declare the dependency on `scipy` with a field like:

```
Config/reticulate:
  list(
    packages = list(
      list(package = "scipy", pip = TRUE)
    )
  )
```

In particular, this will instruct `reticulate` to install the latest available version of the `scipy` package from PyPI, using `pip`.

`reticulate` will read and parse the `DESCRIPTION` file when Python is initialized, and use that information when configuring the Python environment. See:

```
vignette("python_dependencies")
```

for more information.

Limitations

With automatic configuration, `reticulate` wants to encourage a world wherein different R packages wrapping Python packages can live together in the same Python environment / R session. In essence, we would like to minimize the number of conflicts that could arise through different R packages having incompatible Python dependencies.

Unfortunately, Python projects tend to lean quite heavily upon virtual environments, and so Python packages do sometimes declare fairly narrow version requirements. Ultimately, we are relying on R package authors to work together and avoid declaring similarly narrow or incompatible version requirements. To that end, we ask package authors to please prefer using the latest-available packages on `pip` / the Conda repositories when possible, and to declare version requirements only when necessary.

Pandas Performance

We've also invested some time into improving the performance of conversions between R and Python for Pandas DataFrames – in particular, the conversion performance should be greatly improved for DataFrames

with a large number of columns.

For example, with the following script:

```
library(reticulate)

rdf <- as.data.frame(matrix(0, nrow = 1000, ncol = 10000))
pdf <- r_to_py(rdf)

system.time(r_to_py(rdf))
system.time(py_to_r(pdf))
```

We see the following timings:

```
# reticulate 1.13 ----

> system.time(r_to_py(rdf))
   user  system elapsed
  7.581   0.052   7.640

> system.time(py_to_r(pdf))
   user  system elapsed
 15.363   0.065  15.446

# reticulate 1.14 ----

> system.time(r_to_py(rdf))
   user  system elapsed
  0.303   0.002   0.306

> system.time(py_to_r(pdf))
   user  system elapsed
  1.320   0.025   1.347
```

Over a 10x improvement!

Python 2.7

As you may be aware, Python 2.7 is slowly being phased out in favor of Python 3. On January 1st, 2020, Python 2.7 will officially reach end-of-life. To that end, this will be the last `reticulate` release to officially support Python 2.7 – all future work will focus on supporting Python 3.x. We strongly encourage users of `reticulate` to update to Python 3 if they have not already.