A Comprehensive Guide to Combining R and Python with `reticulate`

Alejandro Leonardo García Navarro

2024-06-20

Contents

Introduction

What is `reticulate`?

The `reticulate` package is a tool that allows to combine R and Python. It allows users to call Python from R and R from Python, combining the strengths of both programming languages in a single workflow.

With this library, you can import any Python module and access its functions, classes, and objects from R, enabling a more versatile and flexible approach to data analysis, machine learning, and statistical computing.

Benefits of combining R and Python

Combining R and Python brings together the best of both worlds:

1. Choose the best tool for each task by leveraging R's statistical analysis and Python's programming and machine learning strengths.
2. Access more libraries and packages from both ecosystems.
3. Easy transfer of data between R and Python for flexible data handling in complex analysis pipelines.

Prerequisites and installation

Before using the library, make sure you have the following prerequisites:

1. R Installation: Install R on your system.
2. Python Installation: Install Python on your system.
3. RStudio (Optional but recommended): Using RStudio as your IDE can simplify the process of using `reticulate`. Download RStudio from here.

Once you have completed all the prerequisites, it is time to install the package. Use the following command in your R console:

```
options(repos = "https://cloud.r-project.org")
install.packages("reticulate")
```

```
##
##   Une version binaire est disponible mais la version du source est plus
##   récente:
##            binary source needs_compilation
## reticulate 1.36.1 1.38.0              TRUE
##
##   Binaries will be installed
## le package 'reticulate' a été décompressé et les sommes MD5 ont été vérifiées avec succés
##
## Les packages binaires téléchargés sont dans
##  C:\Users\algar\AppData\Local\Temp\Rtmpyy8HYj\downloaded_packages
```

After installation, load the package using:

```
library(reticulate)
```

```
## Warning: le package 'reticulate' a été compilé avec la version R 4.2.3
```

Basic Usage

Importing Python Modules

To import a Python module in R using the `reticulate` package, you use the `import` function. For example, to import the `numpy` library, you can use:

```
np <- import("numpy")
```

With this, you can use the `np` object to access `numpy` functions and methods just as you would in Python:

```
# Create a numpy array
array <- np$array(c(1, 2, 3, 4, 5))
print(array)
```

```
## [1] 1 2 3 4 5
```

Running Python Code in R

Sometimes, it might be useful to execute Python code directly within an R script, and this can be easily done using the `py_run_string` function. This function allows you to run Python code as a string:

```
py_run_string("print('Hello from Python')")
```

```
## Hello from Python
```

Alternatively, it may be more convenient to directly execute a Python script file. For this, you can use the `py_run_file` function:

```
# py_run_file("path/to/your_script.py")
py_run_file("test.py")
```

```
## The sum of 4 and 6 is 10
```

Accessing Python Objects in R

In the same way, you can access and manipulate Python objects in R. For example, if you create a Python list, you can access it in R:

```
# You can access a Python list
py_run_string("my_list = [1, 2, 3, 4, 5]")
my_list <- py$my_list
print(my_list)
```

```
## [1] 1 2 3 4 5
```

```r
# You can manipulate the list
my_list[1] <- 4
print(my_list)
```

```
## [1] 4 2 3 4 5
```

It is also possible to define functions and call them from R:

```r
py_run_string("
def greet(name):
  return 'Hello, ' + name + '!'
")

greet <- py$greet
print(greet("World"))
```

```
## [1] "Hello, World!"
```

```r
print(greet("James"))
```

```
## [1] "Hello, James!"
```

Data Manipulation

Using Python Libraries Like NumPy and pandas

You can use Python libraries like NumPy and pandas for data manipulation in R:

```r
# Import NumPy and pandas
np <- import("numpy")
pd <- import("pandas")

# Create a numpy array
array <- np$array(c(1, 2, 3, 4, 5))
print(array)
```

```
## [1] 1 2 3 4 5
```

```r
# Create a pandas data frame
py_df <- pd$DataFrame(dict(a=np$array(c(1, 2, 3)), b=np$array(c('x', 'y', 'z'))))
print(py_df)
```

```
##   a b
## 1 1 x
## 2 2 y
## 3 3 z
```

Converting Data Types Between R and Python

It is important to know that the `reticulate` package automatically converts many data types between R and Python. For example, **R vectors become Python lists**, and **R data frames become pandas data frames**.

You can manually convert data types using specific functions if needed:

1. To convert an R data frame to a pandas data frame:

```r
# Define data frame
df <- data.frame(a = 1:3, b = c('x', 'y', 'z'))

# Convert R data frame to pandas data frame
py_df <- r_to_py(df)
print(py_df)
```

```
##    a  b
## 0  1  x
## 1  2  y
## 2  3  z
```

2. To convert a pandas data frame back to an R data frame:

```r
# Convert pandas data frame to R data frame
r_df <- py_to_r(py_df)
print(r_df)
```

```
##   a b
## 1 1 x
## 2 2 y
## 3 3 z
```

Visualization

Using Python Visualization Libraries

Sometimes, you might have more experience plotting in Python than in R. Thanks to this package, Python libraries like Matplotlib and Seaborn can be used:

```r
# Import libraries
plt <- import("matplotlib.pyplot")
sns <- import("seaborn")

# Create a plot using Matplotlib
plt$plot(c(1, 2, 3), c(4, 5, 6))
```

```
## [[1]]
## <matplotlib.lines.Line2D object at 0x00000204BC0C22D0>
```

```r
plt$show()

# Create a plot using Seaborn
sns$set_theme()
df <- sns$load_dataset("iris")
sns$scatterplot(data=df, x="sepal_length", y="sepal_width", hue="species")
```

```
## <Axes: xlabel='sepal_length', ylabel='sepal_width'>
```

```r
plt$show()
```

Machine Learning

Using Scikit-Learn for Classification

```r
# Import scikit-learn
sklearn <- import("sklearn")
datasets <- sklearn$datasets
svm <- sklearn$svm
metrics <- sklearn$metrics

# Load dataset and train a model
iris <- datasets$load_iris()
X <- iris$data
y <- iris$target
model <- svm$SVC()
model$fit(X, y)
```

```
## SVC()
```

```r
# Make predictions
predictions <- model$predict(X)

# Evaluate the model
accuracy <- metrics$accuracy_score(y, predictions)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.973333333333333"
```

Building and Evaluating a Regression Model

```r
# Import necessary libraries
sklearn <- import("sklearn")
datasets <- sklearn$datasets
linear_model <- sklearn$linear_model
metrics <- sklearn$metrics
```

```r
# Load the diabetes dataset
diabetes <- datasets$load_diabetes()
X <- diabetes$data
y <- diabetes$target

# Split the data into training and testing sections
library(zeallot)
```

```
## Warning: le package 'zeallot' a été compilé avec la version R 4.2.3
```

```r
train_test_split <- sklearn$model_selection$train_test_split
c(X_train, X_test, y_train, y_test) %<-% train_test_split(X, y, test_size = 0.2)

# Train a linear regression model
model <- linear_model$LinearRegression()
model$fit(X_train, y_train)
```

```
## LinearRegression()
```

```r
# Make predictions
predictions <- model$predict(X_test)

# Evaluate the model
mse <- metrics$mean_squared_error(y_test, predictions)
print(paste("Mean Squared Error:", mse))
```

```
## [1] "Mean Squared Error: 3034.34598842021"
```

Using TensorFlow for Deep Learning

```r
# Import TensorFlow
tf <- import("tensorflow")
keras <- import("keras")

# Load and preprocess data
mnist_data <- keras$datasets$mnist$load_data()
train_images <- mnist_data[[1]][[1]]
train_labels <- mnist_data[[1]][[2]]
test_images <- mnist_data[[2]][[1]]
test_labels <- mnist_data[[2]][[2]]
train_images <- train_images/255
test_images <- test_images/255

# Ensure the input shape is specified correctly as an integer tuple
input_shape <- as.integer(c(28, 28))

# Build the model
model <- keras$Sequential()
model$add(keras$layers$Flatten(input_shape = input_shape))
```

```r
model$add(keras$layers$Dense(128, activation = "relu"))
model$add(keras$layers$Dense(10, activation = "softmax"))

# Compile the model
model$compile(optimizer = "adam", loss = "sparse_categorical_crossentropy", metrics = "accuracy")

# Train the model
model$fit(train_images, train_labels, epochs = as.integer(5), verbose = 0)
```

```
## <keras.src.callbacks.History object at 0x00000204D414ABD0>
```

```r
# Evaluate the model
metrics <- model$evaluate(test_images, test_labels, verbose = 0)
test_loss <- metrics[[1]]
test_acc <- metrics[[2]]
print(paste("Test accuracy:", test_acc))
```

```
## [1] "Test accuracy: 0.975300014019012"
```

Reinforcement Learning

Using OpenAI Gym for Reinforcement Learning

```r
# Import necessary libraries
gym <- import("gym")
np <- import("numpy")

# Create the environment
env <- gym$make("CartPole-v1")

# Define the number of episodes and the maximum number of steps per episode
num_episodes <- 50
max_steps <- 100

# Initialize a list to store total rewards per episode
total_rewards <- numeric(num_episodes)

# Run episodes
for (episode in 1:num_episodes) {
  state <- env$reset()
  total_reward <- 0

  for (step in 1:max_steps) {
    # Take a random action
    action <- env$action_space$sample()

    # Perform the action in the environment
    result <- env$step(action)
    new_state <- result[[1]]
```

```r
    reward <- result[[2]]
    done <- result[[3]]

    # Accumulate the reward
    total_reward <- total_reward + reward

    # Update the state
    state <- new_state

    # Break the loop if the episode is finished
    if (done) {
      break
    }
  }
  total_rewards[episode] <- total_reward
  print(paste("Episode:", episode, "Total Reward:", total_reward))
}
```

```
## [1] "Episode: 1 Total Reward: 27"
## [1] "Episode: 2 Total Reward: 11"
## [1] "Episode: 3 Total Reward: 20"
## [1] "Episode: 4 Total Reward: 14"
## [1] "Episode: 5 Total Reward: 15"
## [1] "Episode: 6 Total Reward: 59"
## [1] "Episode: 7 Total Reward: 29"
## [1] "Episode: 8 Total Reward: 21"
## [1] "Episode: 9 Total Reward: 18"
## [1] "Episode: 10 Total Reward: 21"
## [1] "Episode: 11 Total Reward: 10"
## [1] "Episode: 12 Total Reward: 21"
## [1] "Episode: 13 Total Reward: 16"
## [1] "Episode: 14 Total Reward: 14"
## [1] "Episode: 15 Total Reward: 16"
## [1] "Episode: 16 Total Reward: 13"
## [1] "Episode: 17 Total Reward: 12"
## [1] "Episode: 18 Total Reward: 14"
## [1] "Episode: 19 Total Reward: 21"
## [1] "Episode: 20 Total Reward: 17"
## [1] "Episode: 21 Total Reward: 36"
## [1] "Episode: 22 Total Reward: 24"
## [1] "Episode: 23 Total Reward: 13"
## [1] "Episode: 24 Total Reward: 15"
## [1] "Episode: 25 Total Reward: 21"
## [1] "Episode: 26 Total Reward: 30"
## [1] "Episode: 27 Total Reward: 16"
## [1] "Episode: 28 Total Reward: 23"
## [1] "Episode: 29 Total Reward: 32"
## [1] "Episode: 30 Total Reward: 18"
## [1] "Episode: 31 Total Reward: 95"
## [1] "Episode: 32 Total Reward: 10"
## [1] "Episode: 33 Total Reward: 16"
## [1] "Episode: 34 Total Reward: 12"
## [1] "Episode: 35 Total Reward: 30"
```

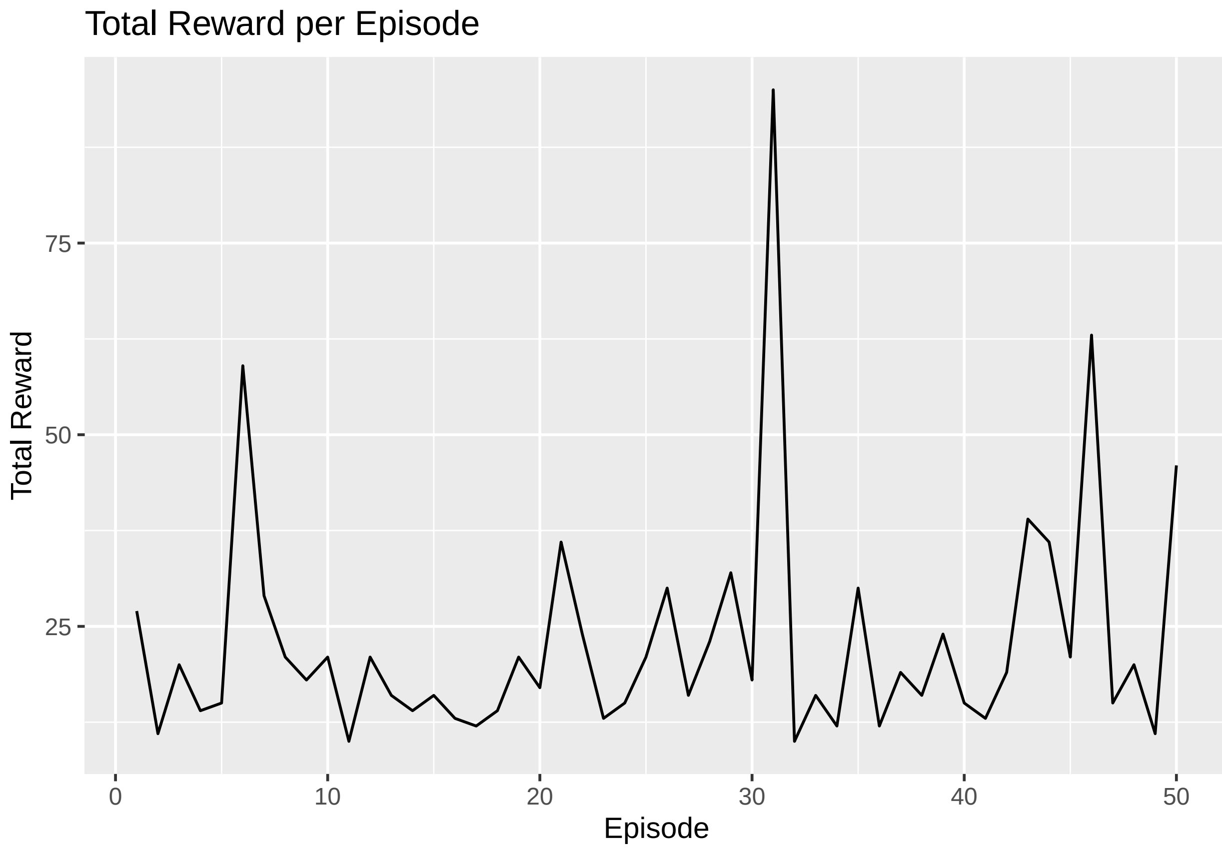
```
## [1] "Episode: 36 Total Reward: 12"
## [1] "Episode: 37 Total Reward: 19"
## [1] "Episode: 38 Total Reward: 16"
## [1] "Episode: 39 Total Reward: 24"
## [1] "Episode: 40 Total Reward: 15"
## [1] "Episode: 41 Total Reward: 13"
## [1] "Episode: 42 Total Reward: 19"
## [1] "Episode: 43 Total Reward: 39"
## [1] "Episode: 44 Total Reward: 36"
## [1] "Episode: 45 Total Reward: 21"
## [1] "Episode: 46 Total Reward: 63"
## [1] "Episode: 47 Total Reward: 15"
## [1] "Episode: 48 Total Reward: 20"
## [1] "Episode: 49 Total Reward: 11"
## [1] "Episode: 50 Total Reward: 46"
```

```r
env$close()

# Plot the total rewards per episode
library(ggplot2)
```

```
## Warning: le package 'ggplot2' a été compilé avec la version R 4.2.3
```

```r
df <- data.frame(episode = 1:num_episodes, total_reward = total_rewards)
ggplot(df, aes(x = episode, y = total_reward)) +
  geom_line() +
  labs(title = "Total Reward per Episode",
       x = "Episode",
       y = "Total Reward")
```



Total Reward per Episode
Total Reward
75
50
25
0
10
20
30
40
50
Episode