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

Alejandro Leonardo García Navarro

2024-07-10

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:

```r
if (!require("reticulate")) install.packages("reticulate")
```

```
## Le chargement a nécessité le package : reticulate
```

After installation, load the package using:

```r
library(reticulate)
```

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:

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

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

```r
# 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:

```r
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:

```r
# 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:

```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!"
```

```
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:

```
# 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
```

```
# 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:

```
# 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:

```
# 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:

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

# Create a plot using Matplotlib
plt$figure()
```

```
## <Figure size 640x480 with 0 Axes>
```

```
plt$plot(c(1, 2, 3), c(4, 5, 6))
```

```
## [[1]]
## <matplotlib.lines.Line2D object at 0x000001DA42DDFC20>
```

```
plt$show()

# Create a plot using Seaborn
plt$figure()
```

```
## <Figure size 640x480 with 0 Axes>
```

```
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

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

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

```r
# 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: 3058.42081626712"
```

Using Pytorch for Deep Learning

```r
# Import necessary modules
torch <- import("torch")
torchvision <- import("torchvision")
nn <- torch$nn
optim <- torch$optim
transforms <- torchvision$transforms

# Define hyperparameters
batch_size <- as.integer(64)  # Number of samples per batch
learning_rate <- 0.001  # Learning rate for the optimizer
num_epochs <- 5  # Number of times to iterate over the entire training dataset

# Data preparation
# Define a transformation to convert images to tensors and normalize them
transform <- transforms$Compose(list(transforms$ToTensor(), transforms$Normalize(c(0.5), c(0.5))))

# Load the training and test datasets with the defined transformations
train_dataset <- torchvision$datasets$MNIST(root='./data', train=TRUE, transform=transform, download=TRU
train_loader <- torch$utils$data$DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=TRUE)
test_dataset <- torchvision$datasets$MNIST(root='./data', train=FALSE, transform=transform, download=TRU
test_loader <- torch$utils$data$DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=FALSE)

# Define neural network model using Python code
py_run_string("
import torch
import torch.nn as nn
import torch.nn.functional as F

# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        # First fully connected layer (input layer)
        self.fc1 = nn.Linear(28*28, 512)
        # Second fully connected layer (hidden layer)
        self.fc2 = nn.Linear(512, 256)
        # Third fully connected layer (output layer)
        self.fc3 = nn.Linear(256, 10)

    # Define the forward pass
    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten the input image
        x = F.relu(self.fc1(x))  # Apply ReLU activation to the first layer
        x = F.relu(self.fc2(x))  # Apply ReLU activation to the second layer
        x = self.fc3(x)  # Output layer
        return x

model = SimpleNN()
")

# Access the model defined in the Python code
model <- py$model

# Define loss function and optimizer
criterion <- nn$CrossEntropyLoss()
optimizer <- optim$Adam(model$parameters(), lr=learning_rate)

# Training loop
for (epoch in 1:num_epochs) {
  # Iterate over batches of data
  for (batch in reticulate::iterate(train_loader)) {
    images <- batch[[1]]  # Input images
    labels <- batch[[2]]  # True labels

    # Forward pass: compute predicted outputs
    outputs <- model$forward(images)
    # Compute loss
    loss <- criterion(outputs, labels)

    # Backward pass: compute gradients
    optimizer$zero_grad()  # Clear existing gradients
    loss$backward()  # Compute gradients
    optimizer$step()  # Update model parameters

  }
  cat(sprintf('Epoch [%d/%d], Loss: %.4f\n', epoch, num_epochs, loss$item()))
}
```

```
## Epoch [1/5], Loss: 0.0719
## Epoch [2/5], Loss: 0.2977
## Epoch [3/5], Loss: 0.1472
## Epoch [4/5], Loss: 0.0304
## Epoch [5/5], Loss: 0.1768
```

```
# Testing the model
model$eval()
```

```
## SimpleNN(
##   (fc1): Linear(in_features=784, out_features=512, bias=True)
##   (fc2): Linear(in_features=512, out_features=256, bias=True)
##   (fc3): Linear(in_features=256, out_features=10, bias=True)
## )
##  signature: (*args, **kwargs)
```

```r
correct <- 0
total <- 0

# Iterate over batches of test data
for (batch in reticulate::iterate(test_loader)) {
  images <- batch[[1]]
  labels <- batch[[2]]
  outputs <- model$forward(images)
  predicted <- torch$max(outputs$data, dim=1L)$indices
  total <- total + as.integer(labels$size(as.integer(0)))
  correct <- correct + as.integer((predicted == labels)$sum()$item())
}

# Compute accuracy
accuracy <- 100 * correct / total
cat(sprintf('Accuracy of the model on the 10000 test images: %.2f%%\n', accuracy))
```

```
## Accuracy of the model on the 10000 test images: 97.47%
```

Using TensorFlow for Deep Learning

```r
# Import TensorFlow and Keras to create and tranii the neural network
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]]

# Normalize pixel values
train_images <- train_images/255
test_images <- test_images/255

# Build the model
  # Allows stacking layers linearly
model <- keras$Sequential()
  # Flatten layer that reshapes each 28x28 image into a 1D of 784 elements (28x28)
model$add(keras$layers$Flatten(input_shape = as.integer(c(28, 28))))
  # Dense layer with 128 units and ReLU activation function
model$add(keras$layers$Dense(units = as.integer(128), activation = "relu"))
  # Dense layer with 10 units and softmax activation function, used for multi-class classifcation
```

```r
model$add(keras$layers$Dense(units = as.integer(10), activation = "softmax"))

# Compile the model (prepare it for training)
model$compile(optimizer = "adam", loss = "sparse_categorical_crossentropy", metrics = list("accuracy"))

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

```
## <keras.src.callbacks.history.History object at 0x000001DA9FAF0290>
```

```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: 0.97790002822876"
```

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]]
    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: 24"
## [1] "Episode: 2 Total Reward: 20"
## [1] "Episode: 3 Total Reward: 14"
## [1] "Episode: 4 Total Reward: 17"
## [1] "Episode: 5 Total Reward: 30"
## [1] "Episode: 6 Total Reward: 18"
## [1] "Episode: 7 Total Reward: 22"
## [1] "Episode: 8 Total Reward: 17"
## [1] "Episode: 9 Total Reward: 76"
## [1] "Episode: 10 Total Reward: 20"
## [1] "Episode: 11 Total Reward: 29"
## [1] "Episode: 12 Total Reward: 16"
## [1] "Episode: 13 Total Reward: 21"
## [1] "Episode: 14 Total Reward: 11"
## [1] "Episode: 15 Total Reward: 10"
## [1] "Episode: 16 Total Reward: 20"
## [1] "Episode: 17 Total Reward: 14"
## [1] "Episode: 18 Total Reward: 21"
## [1] "Episode: 19 Total Reward: 22"
## [1] "Episode: 20 Total Reward: 9"
## [1] "Episode: 21 Total Reward: 11"
## [1] "Episode: 22 Total Reward: 11"
## [1] "Episode: 23 Total Reward: 37"
## [1] "Episode: 24 Total Reward: 19"
## [1] "Episode: 25 Total Reward: 12"
## [1] "Episode: 26 Total Reward: 17"
## [1] "Episode: 27 Total Reward: 11"
## [1] "Episode: 28 Total Reward: 11"
## [1] "Episode: 29 Total Reward: 13"
## [1] "Episode: 30 Total Reward: 29"
## [1] "Episode: 31 Total Reward: 11"
## [1] "Episode: 32 Total Reward: 21"
## [1] "Episode: 33 Total Reward: 8"
## [1] "Episode: 34 Total Reward: 12"
## [1] "Episode: 35 Total Reward: 17"
## [1] "Episode: 36 Total Reward: 16"
## [1] "Episode: 37 Total Reward: 14"
## [1] "Episode: 38 Total Reward: 19"
## [1] "Episode: 39 Total Reward: 13"
## [1] "Episode: 40 Total Reward: 18"
## [1] "Episode: 41 Total Reward: 12"
## [1] "Episode: 42 Total Reward: 16"
## [1] "Episode: 43 Total Reward: 42"
## [1] "Episode: 44 Total Reward: 36"
## [1] "Episode: 45 Total Reward: 16"
## [1] "Episode: 46 Total Reward: 14"
## [1] "Episode: 47 Total Reward: 33"
## [1] "Episode: 48 Total Reward: 17"
## [1] "Episode: 49 Total Reward: 13"
## [1] "Episode: 50 Total Reward: 23"
```

```
env$close()
```

Web Scraping

Using BeautifulSoup for Web Scraping

```
# Import BeautifulSoup and requests
bs4 <- import("bs4")
requests <- import("requests")

# Fetch webpage content
url <- "https://www.uc3m.es/Home"
response <- requests$get(url)

# Parse HTML content
soup <- bs4$BeautifulSoup(response$content, "html.parser")

# Extract all div elements
divs <- soup$find_all("div")

# Print the content of the first 10 divs
for (i in 1:10) {
  div <- divs[[i]]
  print(div$get_text())
}
```

```
## [1] "\n\n\n\n\n\n\n\n\n\n\n\nES\n\nE_administration\n\nFaculty/Staff Intranet\n\nAula Global\n\n\n\nSea
## [1] "\n\n\n\n\n\n\n\n\n\n\nES\n\nE_administration\n\nFaculty/Staff Intranet\n\nAula Global\n\n\n\nSearc
## [1] "\n\n\n\n\n\n\n"
## [1] ""
## [1] ""
## [1] ""
## [1] "\n\nES\n\nE_administration\n\nFaculty/Staff Intranet\n\nAula Global\n\n\n\nSearch:\n\n\n\nBuscar
## [1] "\nES\n\nE_administration\n\nFaculty/Staff Intranet\n\nAula Global\n"
## [1] "\n\nSearch:\n\n\n\nBuscar en:\n\nuc3m.es\n\nPeople\n\n\n\n\n\n"
## [1] "\nBuscar en:\n\nuc3m.es\n\nPeople\n"
```