

PhUSE US Connect 2020
Paper TT14
R & Python for Clinical Workflows

Phil Bowsher, RStudio; Boston, USA

Abstract

The statistical programming space is evolving. Many universities and programs are educating students about Clinical Data Science. Moreover, the field is popular among recent graduates. R, Python, SQL and Javascript are important programming languages within the Clinical Data Science toolbox. For years, people have wondered which language will take the top seat but it is clear now that a new *topic* has taken focus...**Interoperability** and the capability of these different languages to work together harmoniously. If you review any data science job application, you will likely see R and Python listed together among the technologies to know.

The information below will focus on the interoperability of the top 2 programming languages in the data science space, R and Python, and how they can be combined to achieve great results for your clinical workflow.

Please see the PhUSE paper “R for Deep Learning with Tensorflow with Applications in Cancer Immunotherapy” for a deeper dive into how python can be used in the pharmaceutical space.

Introduction

This paper is for clinical statistical programmers that are new to Python and have wondered how it can be combined to enhance their R workflows. For the focus of this paper, information will be provided about the use of R and Python on a server and not on a desktop computer.

Below will discuss the current landscape for combining R and Python in your clinical workflows. Below is not an introduction to programming in R or Python but will provide valuable information for clinical statistical programmers using R and looking to add Python to their knowledge base.

R & Python for Clinical Workflows

First it is important to understand the creation differences among R and Python.

R

R is a programming language purpose built for statistical computing and graphics. Much of the data analysis components of R were built into the language from the beginning (1992) by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. This capability was in R's formation, which started with the S language which initiated in 1976 at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues as an internal statistical analysis environment—originally implemented as Fortran libraries and later into C. From the very beginning, R, like S, was conceived to **provide interfaces and bindings into other languages.**

Base R is the core R programming language and packages added to supplement base R are often referred to as “contributed packages” and are available for download directly from the Comprehensive R Archive Network (CRAN). CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. The source for R includes 3 things: R source code, a set of base packages, and a set of recommended packages. All 3 components follow the release and testing cycle outlined here:

<https://www.r-project.org/doc/R-FDA.pdf>

In addition to distributing R, CRAN is the primary package repository in the R community. A team of individuals approves packages submitted and added to CRAN. The number of R packages published to CRAN has been steadily increasing. In the first 10 years of R, there were roughly 1,000 packages on CRAN. In the second 10 years, there were nearly 13,000.

Python

Python is an interpreted, high-level, general-purpose programming language. Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL). It was conceived as an easier higher-level interpreted language that provides readable code for general purpose programming tasks. Python's standard library is extensive, offering facilities such as built-in modules (written in C) that provide access to system functionality (file I/O) as well as modules written in Python that provide help for work in general programming. Please see: <https://docs.python.org/3/library/>

In addition to the standard library, there are components available from the Python Package Index PyPI. Most Python packages are submitted to PyPI which acts as a central repository. As of the date of this publication, Feb 2020, there are 217,579 total projects.

217,579 projects	1,667,395 releases	2,524,969 files	402,018 users	2/12/20
------------------	--------------------	-----------------	---------------	---------

To add statistical computing to Python, users will often utilize various libraries such as pandas which was created by Wes McKinney and open sourced in 2009 to provide a high performance, flexible tool to perform quantitative analysis on financial data. Below are some of the top Python tools to know:

NumPy - N-dimensional array

SciPy - Scientific & technical computing (linear algebra, numerical integration, optimization, etc.)

Pandas - Data analysis (manipulation & aggregation & provides a data frame structure similar to R)

Statsmodels - Statistical analysis

mlpy - Machine learning

scikit-learn - Machine learning

Theano - Deep learning

Tensorflow - Deep learning

Keras - Deep learning

Sympy - Symbolic mathematics

Nltk - Text processing

gensim - Text processing

networkx - Network analysis & visualization

Bokeh - Visualization

Matplotlib - Visualization & 2D Plotting (similar to Matlab)

Seaborn - Visualization

plotly - Visualization

Beautifulsoup - Web scraping

scrapy - Web scraping

Jupyter Notebook - (the former iPython Notebook) is a web-based interactive data analysis environment

Anaconda - is a free and open-source distribution of the Python that comes with many of the packages needed for scientific computing and data science and helps to simplify package management and deployment. More will be covered on this topic below.

Validation

Validation is an important part of the clinical workflow. There is a lot of work being done in the R ecosystem to make this work more approachable. This is a key focus for the R Validation Hub as well as the R in Pharma conference at Harvard University each year. Please visit these links for more information:

<http://rinpharma.com/>
<https://www.pharmar.org/>
<https://www.pharmar.org/white-paper/>

I suspect more will come soon for Python.

Approachability

There are 2 fields that dominate the data science space, statistical computing and computer science/software engineering. It is common for those with a statistical computing background to gravitate towards R. This is especially true by the popularity of R in pharma for statiscal programming. However, this does not mean one with a statiacial background should not utilize some of the awesome capabilities in Python. Below is an overview of how this has evolved.

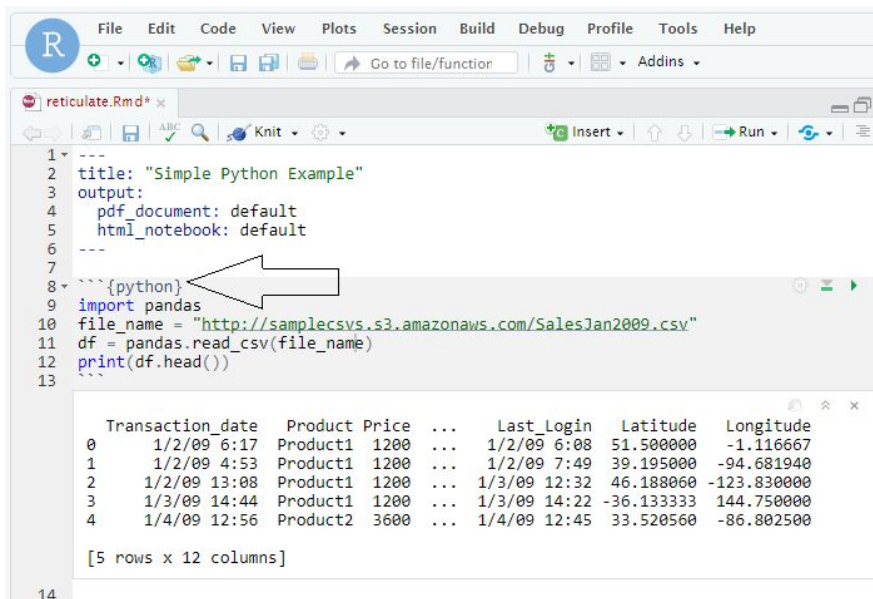
2005-2015

The interoperability between R and Python is a fairly new development. While there were some tools that helped to bridge them, from 2005 to 2015, using both languages together was not an easy undertaking. During this time, much of the R and Python use was disjointed. This is likly one reason that fueled the discussion to choose one language over the other. During this period, researchers, statisticians, and data scientists using both languages would often move from RStudio to the Python shell or IPython or IPython Notebooks (now known as the Jupyter Notebook). In 2009, rpy2 was released as an interface to R running embedded in a Python process. Adoption of this package increased but using Python in R picked up more significantly in 2015...

2015

R Markdown

In 2015, R Markdown had an enhancement that made it possible to specify Python as the language engine as seen here:



```
1 ---
2 title: "Simple Python Example"
3 output:
4   pdf_document: default
5   html_notebook: default
6 ---
7
8 {python}
9 import pandas
10 file_name = "http://samplecsvs.s3.amazonaws.com/SalesJan2009.csv"
11 df = pandas.read_csv(file_name)
12 print(df.head())
13
```

	Transaction_date	Product	Price	...	Last_Login	Latitude	Longitude
0	1/2/09 6:17	Product1	1200	...	1/2/09 6:08	51.500000	-1.116667
1	1/2/09 4:53	Product1	1200	...	1/2/09 7:49	39.195000	-94.681940
2	1/2/09 13:08	Product1	1200	...	1/3/09 12:32	46.188060	-123.830000
3	1/3/09 14:44	Product1	1200	...	1/3/09 14:22	-36.133333	144.750000
4	1/4/09 12:56	Product2	3600	...	1/4/09 12:45	33.520560	-86.802500

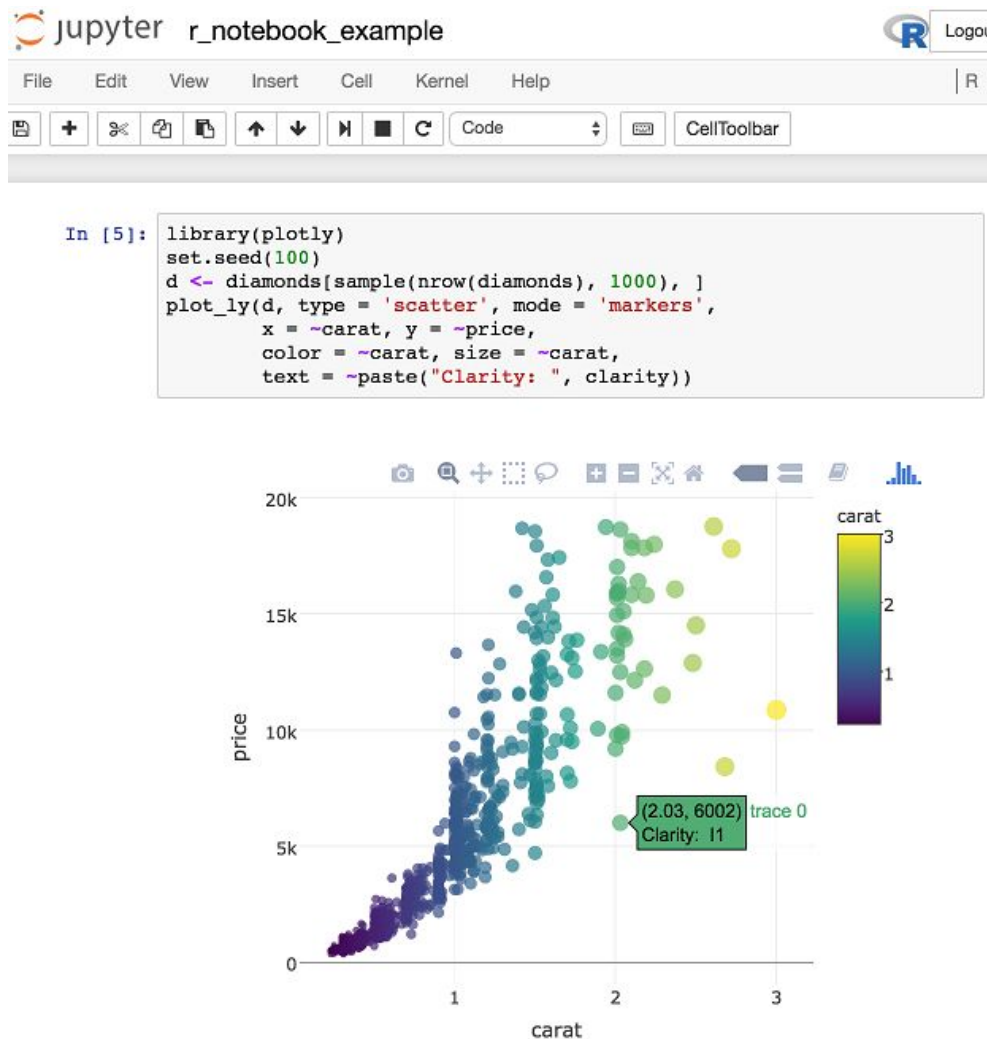
[5 rows x 12 columns]

Please note that a less well-known fact about R Markdown is that many other languages are also supported besides Python, including SAS, Julia, C++, and SQL etc. A full list is here:

<https://bookdown.org/yihui/rmarkdown/language-engines.html>

Jupyter

In 2015, support for R (as well as other languages other than Python) was also added to Jupyter with the IRkernel. IRkernel, an R kernel for Jupyter, allows you to write and execute R code in a Jupyter notebook which allows users to run kernels with R and Python. Below is an example:



<https://irkernel.github.io/installation/>

TensorFlow & Keras

TensorFlow was developed by the Google Brain team for internal Google use. It was publicly released under the Apache License 2.0 on November 9, 2015 and written in Python, C++ and CUDA. Working with TensorFlow directly was challenging and to interface with TensorFlow, most people use the user-friendly Keras package, developed by François Chollet (a Google engineer) and written in Python. Keras can run on top of many deep learning deep learning computing environments like TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML.

The challenge for the R community, was that TensorFlow and Keras were originally released for Python programmers and the R user had limited ways to work with these tools.

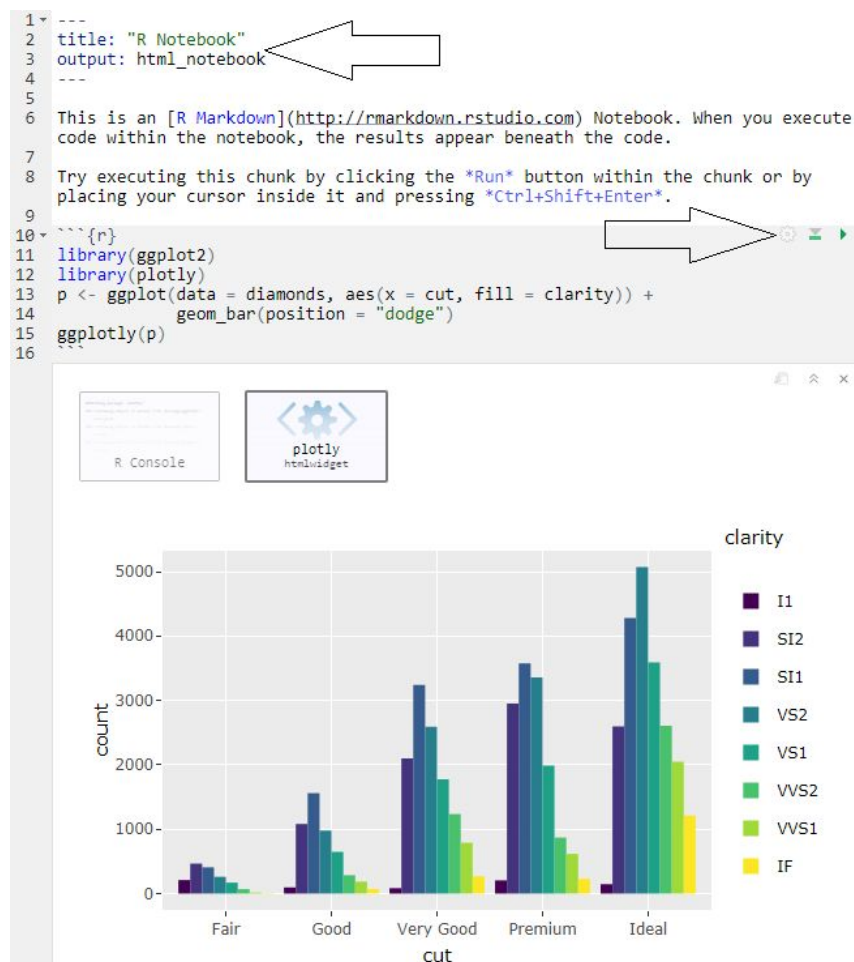
2016

Feather

Even with the advancements above in 2015, it was challenging to get the most out of R and Python working together. One of the key challenges was interchanging data sets among workflows and reading and writing to CSV was cumbersome. In 2015, R's data frames and Python's pandas data frames utilize very different internal memory representations. In early 2016, Wes McKinney (author of pandas) and Hadley Wickham (author of ggplot2) collaborated on Feather, a fast on-disk format for data frames. Feather made it possible for the first time to easily exchange data frames within code chunks of R Markdown and or Jupyter.

R Notebooks

On 2016-10-05, R Notebooks were officially released, which added a powerful notebook authoring engine to R Markdown. This gave R users for the first time, a notebook similar to Jupyter. Prior to this release, code in R Markdown documents were typically executed in batch but now users code executed interactively, one cell at a time for example, with a play button. A final report could still be generated from the notebook. As mentioned previously with R Markdown, R notebook code chunks could execute Python code.



2017

Revisiting TensorFlow & Keras

In 2017, TensorFlow™ 1.0 for R was released on Jan 9th, 2017 and Keras for R on Sep 3, 2017. These important packages made it possible for the first time for R users to use packages that were primarily written in Python and allowed R users to utilize state of the art tools in deep learning. This was possible as the authors of the packages wrote the bindings to Python from R.

These packages laid the foundation for an important package to help further the Interoperability between R and Python... **reticulate**.

After the heavy lifting had been completed in 2017 to write the bindings to Python for TensorFlow and Keras, it was time to further support working with Python from R for other use cases besides deep learning workflows.

reticulate

In March of 2017, the reticulate 0.7 package was officially released. It provided a comprehensive set of tools for interoperability between Python and R:

- Calling Python from R in a variety of ways
- Translation between R and Python objects

reticulate helps to streamline a **bilingual** workflow by allowing a user to utilize Python directly in R.

Calling Python

As before, R users can use Python directly in R Markdown:

```
13
14 ▾ ```{python}
15 import pandas
16 flights = pandas.read_csv("flights.csv")
17 flights = flights[flights['dest'] == "ORD"]
18 flights = flights[['carrier', 'dep_delay', 'arr_delay']]
19 flights = flights.dropna()
20 ```
21
22 ▾ ```{r, fig.width=7, fig.height=3}
23 library(ggplot2)
24 ggplot(py$flights, aes(carrier, arr_delay)) + geom_point() + geom_jitter()
25 ```
26
```

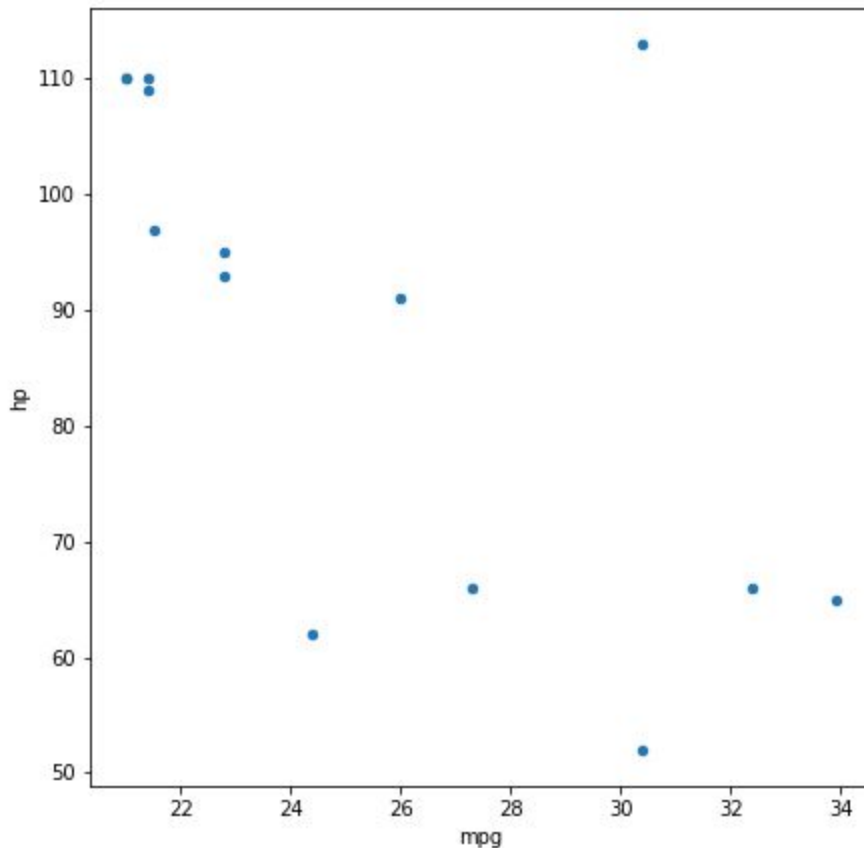
Users can also source any Python script just as you would source an R script using the `source_python()` function.

```
source_python("flights.py")
flights <- read_flights("flights.csv")

library(ggplot2)
ggplot(flights, aes(carrier, arr_delay)) + geom_point() + geom_jitter()
```

Moreover, one can display matplotlib plots within both R notebook and R console execution modes.

```
```{python matplotlib}
import matplotlib.pyplot as plt
my_python_object.plot.scatter(x='mpg', y = 'hp')
plt.show()
```
```



Object Translation

When calling work into Python, R data types are automatically converted to their equivalent Python types and viceversa when returned. You can access previously ran python functions/objects in your r chunks using the prefix `py` combined with the R's dollar sign syntax `$`. Likewise, you can access R objects/functions in your python chunks using the prefix `r` combined with a punctuation mark. `R_to_py` can be used to convert the R object into a python object while still running R with `r_to_py`.

Reticulate provides an important interface into Python for R users, which is especially important when working with data engineers who use primarily Python. Please note that for data science projects that are Python-only, it is still recommended to use IDEs (integrated development environment) optimized for that, such as JupyterLab, PyCharm,

Visual Studio Code, Rodeo, and Spyder. However, if you are using reticulated Python within an R project then the RStudio IDE provides a set of tools to support your workflow.

2018

Apache Arrow & Ursa Labs

While this section is not directly related to R and Python, it does build on the work of Feather discussed previously...

This paper is about the interoperability between R and Python. What about R to Julia, or Julia to Python, or Python to Apache Spark? To solve this problem, there has to be a way to share data between multiple languages without having to write a translation layer (like reticulate) between every pair of languages. The Apache Arrow project has set out to define a standardized, language independent, columnar *memory format for analytics and data science*. Modern hardware platforms provide huge opportunities for optimization (cache pipelining, CPU parallelism, GPUs, etc.), which should allow scientists to use a laptop to interactively analyze 100GB datasets. Apache Arrow sets out as a modern data science runtime environment that will take advantage of computational advances and can be used from many languages.

Ursa Labs, an independent open-source development lab that will serve as the focal point for the development of this cross-language data science runtime powered by Apache Arrow. RStudio serves as the host organization for Ursa Labs, providing operational support and infrastructure. Hadley Wickham is a key technical advisor to Ursa, and collaborates with Ursa on the design and implementation of the runtime. The tidyverse team is planning to build a dplyr back end, as well as other tidy interfaces to arrow. Please see below for more information:

<https://cran.r-project.org/package=arrow>

<https://ursalabs.org/about/>

<https://arrow.apache.org/>

2019

reticulate 1.14

The initial release of reticulate sought to help with the interoperability between Python and R. Configuring Python though to work within R was still a challenge for many users.

The 1.14 sought to help with this key part of using Python in R with *automatic configuration*. The objective being to make it possible for reticulate to automatically prepare a Python environment for the R user, without requiring any explicit user intervention. In this way, R users can use R packages depending on reticulate, without managing a Python installation / environment themselves which is often a key blocker from using Python by statistical programmers. The goal is for R packages using reticulate to be treated just like other R packages, and not requiring R users wanting to use Python to have to engage in Python environment management.

reticulate wants to help where different R packages wrapping Python packages can live together in the same Python environment / R session by minimizing the number of conflicts through different R packages having incompatible Python dependencies. The solution was to help R package authors declare their Python dependency requirements to reticulate in a standardized way, and reticulate will automatically prepare the Python environment for the user. Furthermore, this release will:

1. prompt the user to download and install **Miniconda**
2. prepare a default r-reticulate Conda environment, using (currently) Python 3.6 and NumPy
3. Lastly, 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

The following resource has step by step instructions for adding Python to an R environment:

<https://support.rstudio.com/hc/en-us/articles/360023654474-Installing-and-Configuring-Python-with-RStudio>

Please note that on January 1st, 2020, Python 2.7 will officially reach end-of-life. reticulate 1.14 will be the last reticulate release to officially support Python 2.7 – all future work will focus on supporting Python 3.x. It is encouraged users of reticulate to update to Python 3.

R/Python Environment:

For information on installing Python, especially on Linux servers, take advantage of these resources:

<https://environments.rstudio.com/python.html>

<https://docs.rstudio.com/resources/install-python/>

<https://docs.rstudio.com/rsp/integration/jupyter-multiple-python-versions/>

<https://rstudio.github.io/reticulate/articles/versions.html>

<https://rstudio.github.io/renv/articles/python.html>

For current information on managing mixed R and Python projects, see the `reticulate::use_*` functions and the `renv::use_python` options.

Python in clinical

R and SAS are both well established in the clinical statistical programming space. Python is growing in popularity in many areas of drug development including:

- Various areas of bio-sciences such as genetics. It is popular for automating tasks.
- Python is often used in creating folder structures.
- Python excels at mining and handling text data.
- As a powerful text processing language, Python can be used to automatically create batch files.
- Researchers in genomics and image analytics
- Complex pharma pipelines
- Molecular dynamics workflows and conformational sampling of cyclic peptides
- Deep Learning and machine learning
- Genomic Data Analysis with Pandas

You may be wondering where Python fits in to a *clinical workflow* and why one would need to combine R and Python. Many of the public Python clinical examples focus heavily on machine learning and deep learning. But what about a classic clinical workflow - where does Python fit in? A review of the top repositories on github or recent conference talks highlight that Python is of interest primarily for machine learning and deep learning. It is also worth pointing out that a review of the top topics at the *2019 R in Pharma conference* shows increased interest in more advanced R topics, which highlights a level of adoption evolving as usage is adding data science tools and software development methodologies.

One trend seen with pharmaceutical companies is the expectation of statistical programmers to be aware of data science tooling and languages. The value clinical statistical programmers bring to the drug development process is

invaluable. As the clinical workflow evolves and adds data science processes (machine learning and deep learning etc.) a critical challenge will be **communication**. Below is a post that highlights this need:



Leon Eyrych Jessen @jessenleon · Feb 19

Domain specific knowledge is crucial in #dataScience, this is my new favourite picture to illustrate why! You may be an absolute #rstats or #python ninja, but be sure to team up with a domain specialist if you find yourself in unknown territory 🖥️📊📈📉👍



10

430

1.2K



The challenge often heard from pharmaceutical companies is that data science professionals do not know the drug development processes and statistical programmers may not know some of the data sciences topics. One could see a future where statistical programmers know both their domain and a working knowledge of data science topics. For example, pharmaceutical companies want support from people that have knowledge in the discovery of promising biomarker results in a traditional clinical trial environment. The data can be smaller clinical datasets or working with omics data. It is often heard that machine learning and deep learning approaches can be challenging to work with in precision medicine as the inner workings of these models are harder to understand. Organizations are looking to transform machine learning models to outcomes that fit in a clinical workflow which often includes stratifying patients to the optimum treatment and monitoring the patient's response to treatment.

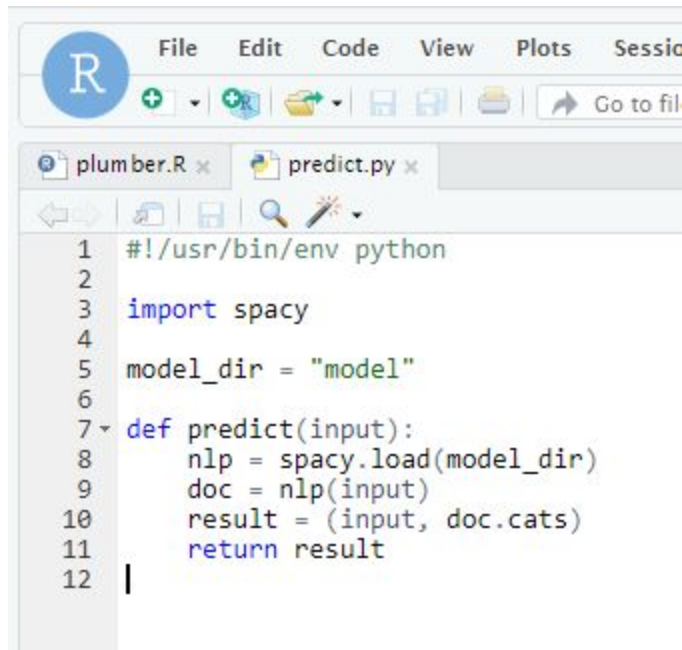
The issue that can arise is the data for these processes can be big (HDFS) and/or unstructured. Both of these challenges are common tasks in data science work, using spark for example or working with images for deep learning. These challenges are also further exacerbated by the infrastructure or engineering required for data science work. Docker, cloud, CI/CD are common tools in a data science workflow but often not in a productionalized clinical development environment with regulatory requirements, though this could change into the future.

In the short term however, **communication between these fields will be crucial**. R and Python interoperability is one small step in helping to unite these areas - clinical workflows and data science. It is well understood that Python plays an important part of the growth of data science and **reticulate** allows statistical programmers (that work primarily with R) to work with data scientist/data engineers who use Python. For example, the data science team may have a

Python model that would be valuable to the clinical workflow. R statistical programmers could easily access this with a reticulated plumber application program interface (API). Below is an example:

Imagine that the data science group has written a model in Python that captures the sentiment of text as positive or negative:

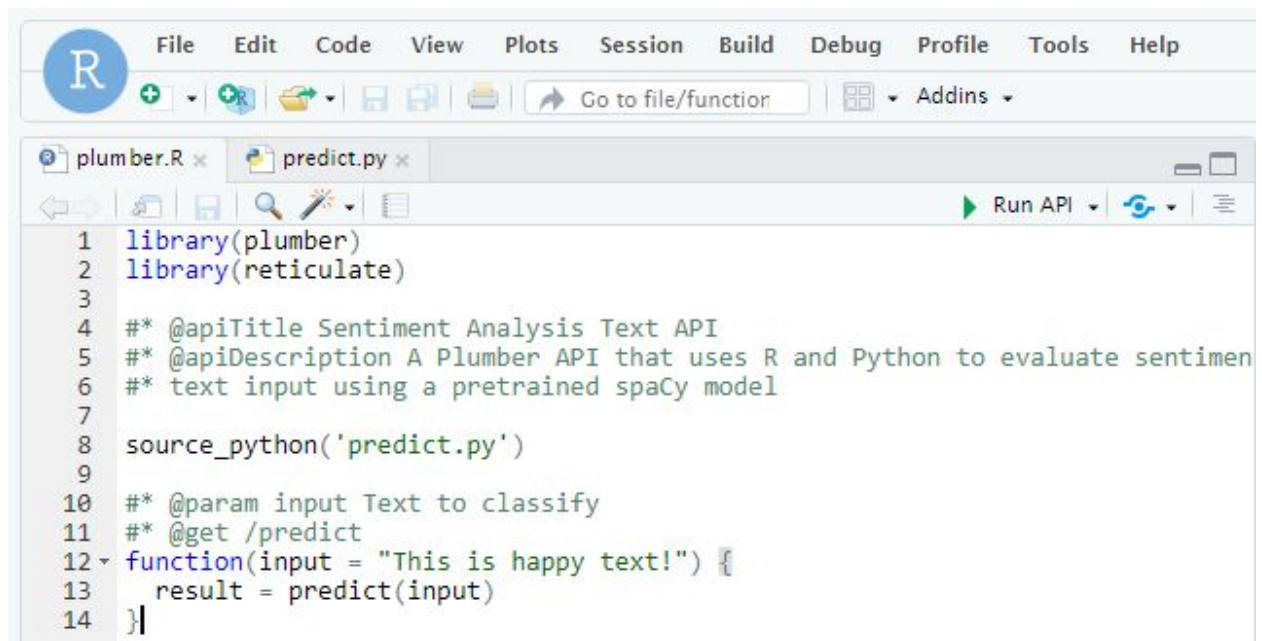
<https://github.com/sol-eng/python-examples/blob/master/sentiment-analysis/predict.py>



The screenshot shows the RStudio interface with the 'predict.py' file open. The code is as follows:

```
1 #!/usr/bin/env python
2
3 import spacy
4
5 model_dir = "model"
6
7 def predict(input):
8     nlp = spacy.load(model_dir)
9     doc = nlp(input)
10    result = (input, doc.cats)
11    return result
12
```

The clinical team would like to use this Python model in the R workflow. The R programmer could utilize this python code and turn it into a R plumber API like this:

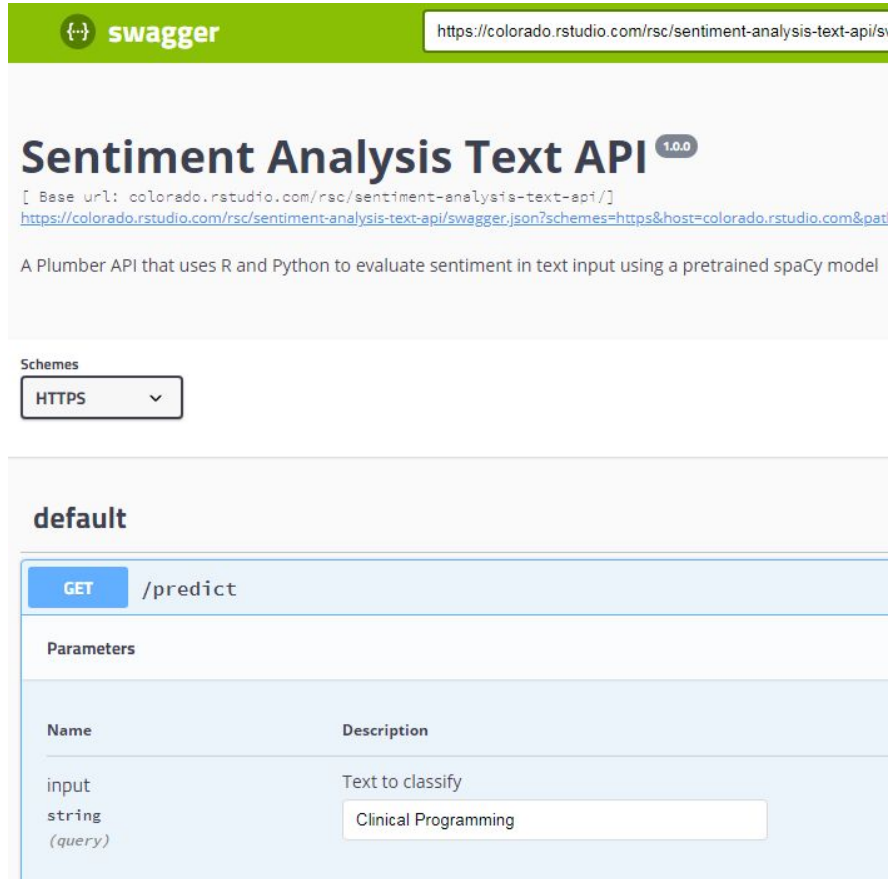


The screenshot shows the RStudio interface with the 'plumber.R' file open. The code is as follows:

```
1 library(plumber)
2 library(reticulate)
3
4 #* @apiTitle Sentiment Analysis Text API
5 #* @apiDescription A Plumber API that uses R and Python to evaluate sentiment
6 #* text input using a pretrained spaCy model
7
8 source_python('predict.py')
9
10 #* @param input Text to classify
11 #* @get /predict
12 function(input = "This is happy text!") {
13   result = predict(input)
14 }
```

This plumber API will then be deployed to a production API environment and kept up to be consumed. After it is live, it can be tested by a human via swagger. A swagger UI is an open source project to visually render documentation for an API defined with the OpenAPI (Swagger) Specification:

https://colorado.rstudio.com/rsc/sentiment-analysis-text-api/__swagger__/



The image shows the Swagger UI for the "Sentiment Analysis Text API" version 1.0.0. The header is green with the Swagger logo and the API URL. Below the header, the API title "Sentiment Analysis Text API" is displayed with a version badge "1.0.0". A description states: "A Plumber API that uses R and Python to evaluate sentiment in text input using a pretrained spaCy model". A "Schemes" dropdown menu is set to "HTTPS". Under the "default" tab, the "/predict" endpoint is shown with a "GET" method. The "Parameters" section lists a query parameter "input" of type "string". A text input field contains the example value "Clinical Programming".

The term "Clinical Programming" returns a positive sentiment:



The image shows the "Details" tab for the "/predict" endpoint in the Swagger UI. It displays a "200" status code with the label "Undocumented". The "Response body" is shown in a dark-themed code editor, containing the following JSON structure:

```
[
  [
    "clinical"
  ],
  {
    "POSITIVE": [
      1
    ]
  }
]
```

Now that the Python model that computes the sentiment of text is surfaced as a RESTful API, the team could now hand off the model predictions to another system. The plumber package can be used to turn any R function into a REST API, and now with reticulate, that means Python too. These APIs can be used by external tools like production ETL processes, Java apps, web apps, or even Excel!

In my paper “R for Deep Learning with Tensorflow with Applications in Cancer Immunotherapy” I will cover an real example where the R and Python interoperability can come together.

Conclusion

As interest in Python increases by clinical statistical programers, the interoperability between R and Python is a crucial bridge between clinical work and data science. The reticulate R package provides R users a way to use R and Python together. The information outlined in this paper highlights the history of R and Python interoperability and advice when incorporating Python into a clinical space with R.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Phil Bowsher

phil@rstudio.com