RStudio Reticulates: R Chunks to Python & Back Again

By Brett Simms on Fri Jul 06 13:28:23 2018

Background

For data analysts, notebooks are a powerful way to document code, display output and describe stories in the data. R and Python are two of the most popular data science languages each with its own merits and weaknesses. As an analyst who uses both languages there are times when you wish to cherry pick functions or data structures from one of the languages, while working in a notebook which principally supports the other.

RStudio as the name suggests principally supports R, while Jupyter Notebook (usually associated with an Anaconda installation) focuses on Python. Jupyter Notebook is a great product complete with "magics" for running R code inline, but I will focus on RStudio here as its new preview release has introduced an exciting feature. **RStudio notebooks can now run Python chunks and provide Python objects back to the user to process in R!**

How It Works

First, the user must insert a R code chunk to declare 'reticulate' as the virtual Python environrment. Second, create a test variable in R to pass to Python.

```
#Must first declare reticulate as virtual env
library(reticulate)
use_virtualenv("r-reticulate")

#Then write some R code
test <- as.character("This variable started out in R")
```

Next insert a Python code chunk and pull across the R variable using .r syntax.

```
#Pull across R variable
test_py = r.test + ", was pushed to Python"
```

Finally, push the Python variable back to R using py$ snytax and print.

```
#Back in R!
test_py_r <- paste0(py$test_py, " and is back in R again!")

print(test_py_r)
```

```
## [1] "This variable started out in R, was pushed to Python and is back in R again!"
```

The code in the notebook below more elaborately tests this functionality by alternating between R and Python chunks. Both Python and R are used to read-in csv data, process it to a common data set (i.e. Unemployment Rates for Alberta) and then plot using either ggplot2 or plotly.

Please note to run this script you will need to install the newest version of the reticulate package, have a KNITR version > 1.8 and of course RStudio 1.2 or greater. The script was created using R 3.51 (64 bit) and Python 3.6 (64 bit) on Windows 7.

Testing :-)

```
#Declare working directory to KNITR for this script
knitr::opts_knit$set(root.dir = "P:/Brett_Personal/R Work")

#KNITR setup
knitr::opts_chunk$set(echo=TRUE, tidy=TRUE, fig.keep = "high", fig.show = "hold", fig.align = "center",
```

```
#This chunk downloads, unzips the raw data from Stats Canada
library(downloader)
library(data.table)

#Create directory for Stats Can Unemployment data
if (!file.exists("Canada_Unemployment")) {
dir.create("Canada_Unemployment")
}

#Download data and unzip if necessary
if (!file.exists("Canada_Unemployment/14100287.csv")) {

  url <- "https://www150.statcan.gc.ca/n1/tbl/csv/14100287-eng.zip"
  download(url, dest="Canada_Unemployment.zip", mode="wb")
  unzip (zipfile = "Canada_Unemployment.zip", exdir = "Canada_Unemployment")
  dateDownloaded <- date()

}
```

```
#Begin timing R read-in of data
r_start_time <-proc.time()

#Read-in Unemployment data from file using R's data.table package
if (!exists("r_data")) {

  r_data<-fread("file://Canada_Unemployment/14100287.csv", colClasses = "character",
                header = TRUE, data.table = FALSE)

}

#Calculate time taken for R to read-in data using fread()
r_total_read_time <- as.numeric(round((proc.time() - r_start_time)[3]))

remove(r_start_time)
print(r_total_read_time)
```

```
## [1] 174
```

```
import pandas as pd
import time
#Begin timing python read-in
python_start_time = time.time()
#Test for presence of pydata object
try:
  pydata

except NameError:
  #Read in fresh copy of py_data using pandas.read_csv()
```

```
  py_data = pd.read_csv("Canada_Unemployment/14100287.csv", dtype = "str")
  print("py_data has just been created")

#Determine total time taken for Python read-in
```

```
## py_data has just been created
```

```
python_total_read_time = round(time.time() - python_start_time)
print(python_total_read_time)
```

```
## 154
```

```
#Bring python_total_read_time over to RStudio to view (note the special py$ syntax)
python_total_read_time <-as.numeric(py$python_total_read_time)
print(python_total_read_time)
```

```
## [1] 154
```

```
#Perform various munging tasks to shape R data as desired
library(dplyr)
library(lubridate)

#Put variable names into proper format (i.e. underscores instead of spaces between words)
names(r_data) <- make.names(names(r_data), unique = TRUE)
names(r_data) <-gsub(x = names(r_data), pattern = "\\.", replacement = "_")

#Select variables of interest, change formats as necessary and filter data. Filtered data will include
r_data_2 <- rename_all(r_data, toupper) %>%
                select(-c(DGUID, VECTOR, COORDINATE, STATUS, SYMBOL, TERMINATED)) %>%
                    mutate(DATE = paste0(REF_DATE, "-01")) %>% mutate_at(vars(DATE), ymd) %>%
                    mutate_at(vars(c(GEO, SEX, LABOUR_FORCE_CHARACTERISTICS,
                                     AGE_GROUP, STATISTICS, DATA_TYPE, UOM, SCALAR_FACTOR)), as.factor
                    mutate_at(vars(VALUE), as.numeric) %>%
                filter(GEO == "Alberta" & LABOUR_FORCE_CHARACTERISTICS == "Unemployment rate" &
                         (SEX == "Females" | SEX == "Males") &
                         (AGE_GROUP == "15 to 24 years" |
                            AGE_GROUP =="25 to 54 years" |
                            AGE_GROUP == "55 years and over") &
                         DATA_TYPE == "Seasonally adjusted" & STATISTICS == "Estimate")

#Drop defunct factor levels
r_data_2 <-droplevels.data.frame(r_data_2)

#Remove an VALUE elements which are missing following steps above
r_data_2 <-filter(r_data_2, !is.na(VALUE))
```

```
#Test for presence of py_data_2 which is the Python replicate of r_data_2. Or in other words, captures
#Make variable names uppercase (note this must occur before rename as upper() removes underscores)
py_data.columns = [x.upper() for x in py_data.columns]
#Rename and drop fields as necessary, also filter to equate to r_data_2
py_data_2 = py_data.rename(index=str, columns={"REF_DATE":"DATE", "LABOUR FORCE CHARACTERISTICS":"LABOU
#Add first of the month to every row in ['Date'] and convert to datetime format
py_data_2['DATE'] = pd.to_datetime(py_data_2['DATE'] + '-01', format = '%Y-%m-%d')
#Convert VALUE to numeric
py_data_2['VALUE'] = pd.to_numeric(py_data_2['VALUE'])
print("py_data_2 has just been created")
```

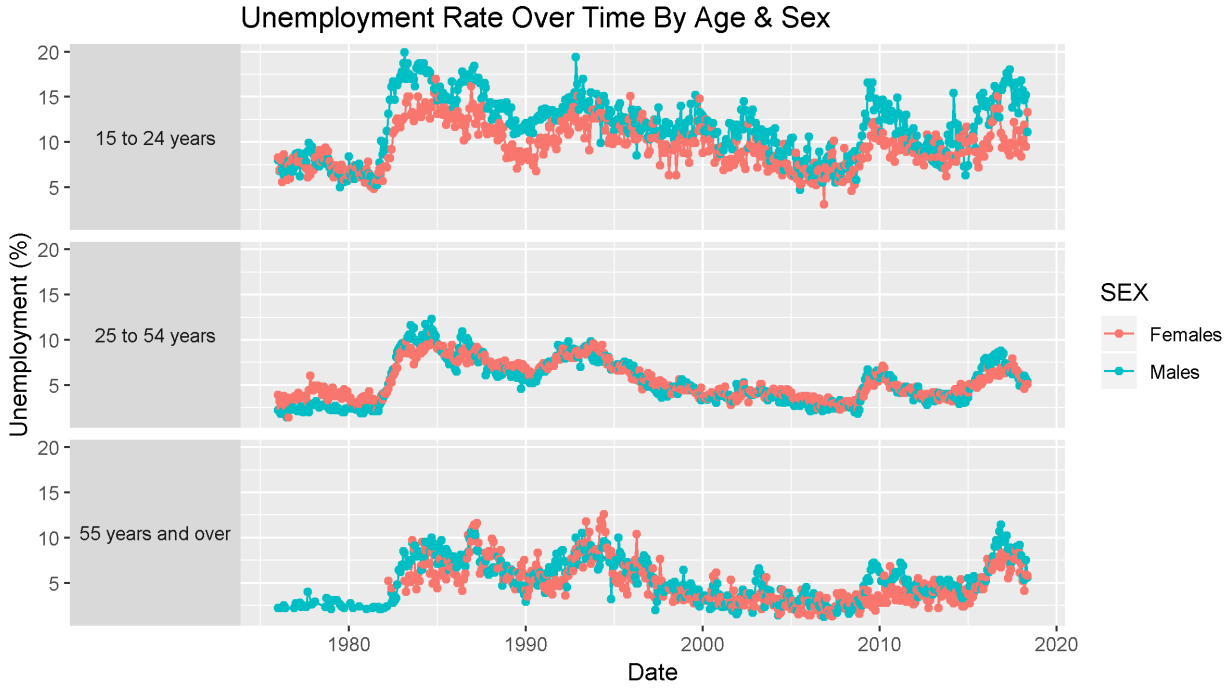
```
## py_data_2 has just been created
```

```
#Plot the R data as a punctuated line graph
library(ggplot2)

#Plot unemployment rate over time
p <- ggplot(data = r_data_2, aes(x=DATE, y=VALUE, colour = SEX)) +
      geom_line() +
      geom_point() +
      labs(y = "Unemployment (%)", x = "Date", title = "Unemployment Rate Over Time By Age & Sex")


#Facet by AGE_GROUP & SEX
p <- p + facet_grid(AGE_GROUP ~., switch = "y") +
      theme(strip.text.y = element_text(angle = 180))

print(p)
```




```
#Test whether Python processed data can produce R graph (note py$ syntax)
py_to_r_data = py$py_data_2

#Plot unemployment rate over time
p <- ggplot(data = py_to_r_data, aes(x=DATE, y=VALUE, colour = SEX)) +
      geom_line() +
      geom_point() +
      labs(y = "Unemployment (%)", x = "Date", title = "Unemployment Rate Over Time By Age & Sex")


#Facet by AGE_GROUP & SEX
p <- p + facet_grid(AGE_GROUP ~., switch = "y") +
      theme(strip.text.y = element_text(angle = 180))
```

```
print(p)
```


```
#Facetting with seaborn fails
# import seaborn as sns
# import matplotlib.pyplot as plt
#pp = sns.FacetGrid(py_data_2, row = "AGE_GROUP", hue = "SEX")
#pp = (pp.map(plt.scatter, "DATE", "VALUE").add_legend()) ---- this line causes crash
#Use plotly in lieu of matplotlib/seaborn
import plotly as py
import plotly.figure_factory as ff
#Create facet grid
pp = ff.create_facet_grid(
        py_data_2, x = 'DATE', y = 'VALUE',
        facet_row = 'AGE_GROUP', color_name = 'SEX',
        show_boxes = False, marker = {'size':10, 'opacity': 0.8},
        colormap={'Males': 'rgb(165, 242, 242)', 'Females': 'rgb(253, 174, 216)'}
        )
#Augment layout to look like R output as much as possible
pp['layout'].update(
        xaxis1 = dict(title = "", tickvals = [1980, 1990, 2000, 2010, 2020]),
        yaxis1 = dict(title = "", range =[0,20], autotick = False, tick0=0, dtick=5),
        xaxis2 = dict(title = ""), yaxis2 = dict(title = "", range =[0,20], tick0=0, dtick=5),
        xaxis3 = dict(title = ""), yaxis3 = dict(title = "", range =[0,20], tick0=0, dtick=5),
        title = "Unemployment Rate Over Time By Age & Sex"
        )
#Plot Ploty graph
py.offline.plot(pp)
```

```
#Test whether Python generated data can produce R graph (notice r. syntax)
r_to_py_data = r.r_data_2
#Create facet grid
```

```
pp = ff.create_facet_grid(
          r_to_py_data, x = 'DATE', y = 'VALUE', facet_row = 'AGE_GROUP',
          color_name = 'SEX', show_boxes = False, marker = {'size':10,
          'opacity': 0.8}, colormap={'Males': 'rgb(165, 242, 242)',
          'Females': 'rgb(253, 174, 216)'}
          )
#Augment layout to look like R output
pp['layout'].update(
          xaxis1 = dict(title = "", tickvals = [1980, 1990, 2000, 2010, 2020]),
          yaxis1 = dict(title = "", range =[0,20], autotick = False, tick0=0, dtick=5),
          xaxis2 = dict(title = ""), yaxis2 = dict(title = "", range =[0,20], tick0=0, dtick=5),
          xaxis3 = dict(title = ""), yaxis3 = dict(title = "", range =[0,20], tick0=0, dtick=5)
          , title = "Unemployment Rate Over Time By Age & Sex"
          )
#Plot Ploty graph
py.offline.plot(pp)
```

Strengths

1) Passing objects between R and Python works very well – this cannot be emphasized enough
2) RStudio/reticulate offers auto-complete while scripting Python
3) Nearly all Python packages loaded into RStudio functioned as expected

Misses

1) Cannot yet view Python objects in either the global or reticulate environments which also means:
    a) Clearing the global environment does not clear Python objects
    b) Two variables can be named identically (one in each language), so careful what you reference in which code chunk!
    c) Python functions that "update" as opposed to create/delete may be performed in perpetuity
2) Python code formatting is not enforced in RStudio
3) Using Python's plotly in "offline mode" produced graphics which did not embed in the notebook

Interesting Tidbits

1) Python's seaborn package (used to facet matplotlib graphics) caused RStudio to crash
2) The matplotlib alternative (plotly) had several formatting nuisances I was unable to fix when combining "offline mode" with faceting
3) Despite parallel processing and the fastest read-in package for .csv files available in R (data.table), Python's pandas.read_csv() loaded the 457k records faster than R, usually by a margin of 20-50% in the notebook. Interestingly this difference disappeared when the notebook was knitted to either HTML or PDF.

Conclusion

This new functionality in RStudio is very exciting and gives Jupyter a run for its money in the chunk-by-chunk battle to produce the ultimate, agnostic notebook!