Python in RNotebooks

bash/sed/awk as well

Bob Settlage

2017-11-03

Today's Agenda

- Two objectives today
    - summarize parallel R
    - show how to use Python in Rnotebooks
    - text mining in Python
    - start thinking about final "project"
- Homework 9

Parallel R summary

Sometimes we find that we need our code to run faster and need to do this ourself in R.

```
#dopar to calculate the sum of squares total for y
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)
time_ssc <- system.time({
  ss_dopar <- foreach(i=1:length(y), .combine = "+") %dopar% {
    sum((y[i] - ybar)^2)
  }})

#parSapply to calculate the sum of squares total for y
ss_sap <- function(i){
  return((i - ybar)^2)
}
clusterExport(cl, varlist=c("ss_sap","ybar"))
time_ssd <- system.time({
  ss_parS <- sum(parSapply(cl, y, function(i) ss_sap(i), USE.NAMES = F, si
})
stopCluster(cl)
```

Faster with work

There are other options beyond the scope of this class:

- Rcpp
- GPU computing in R
- Code in Fortran/C and write an interface to this code

Sometimes faster/easier is to use a different language

The real goal is to go from data to results as quickly and reproducibly as possible creating as sound an analysis as the data allows. Factoring in your time to write the code, sometimes this means you work in a different language because the facilities for the perfect analysis are more advanced/easier/faster outside of R.

How do we do this??

One option is to use RNotebooks.

Notebooks

Note that the idea of notebooks is not new and R is a late comer in this arena. The support for multiple languages in any of the Notbeeks is not seamless , BUT, there are advantages.

Today we will introduce Python and some bash in RNotebooks.

Python

http://rmarkdown.rstudio.com/authoring_knitr_engines.html#python

```
{python echo=F, include=T, eval=T}
x = 'hello, python world!'
print(x.split(' '))
```

```
## ['hello,', 'python', 'world!']
```

What just happened??

We told the RNotebook interpreter to use the local Python interpreter to perform a string split on the given sentence based on the space character and then print the result. Cool, not a lot of extra effort here.

Bash

I can do stuff in bash too, even manipulate files …

{bash echo=T, eval=T, include=T}

```
ls -lah | head -n 4
ls -lah > dir_listing.txt
cat dir_listing.txt | head -n 4
```

```
## total 63776
## drwxr-xr-x  10 rsettlag  staff   340B Nov  3 07:35 .
## drwxr-xr-x  34 rsettlag  staff   1.1K Nov  3 06:50 ..
## -rw-r--r--   1 rsettlag  staff   9.7K Nov  3 07:35 Lecture_9_python_Rnd
## total 63768
## drwxr-xr-x  10 rsettlag  staff   340B Nov  3 07:35 .
## drwxr-xr-x  34 rsettlag  staff   1.1K Nov  3 06:50 ..
## -rw-r--r--   1 rsettlag  staff   9.7K Nov  3 07:35 Lecture_9_python_Rnd
```

Functionality

Ok, so this is all very cool, what can't we do (currently):

- directly reference variables defined in one interpreter flavor from another
  Why? The code is being interpreted and called in an external process.
  This is kinda important, so how do we deal with it??
  Temp files. Blah. At least there is a package for it to make it faster…

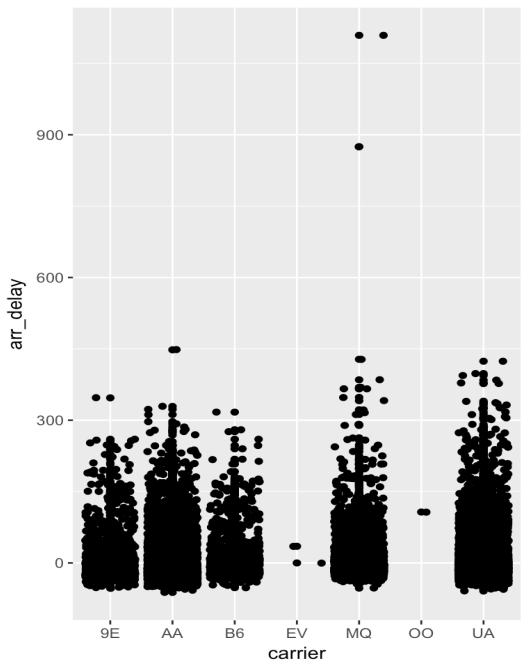
- create visuals using the external process and have them directly incorporate Why? read
  OK, so how do you do it?? Have the external process create a png/jpg/etc. This is how the internals work anyway.

Python example code

```
{python echo=T, eval=F, include=T,
engine.path="/Users/rsettlag/miniconda3/bin/python"}
import pandas; import feather; import csv
# Read flights data and select flights to O'Hare
flights = pandas.read_csv("flights.csv")
flights = flights[flights['dest'] == "ORD"]
# Select carrier and delay columns and drop rows
with missing values
flights = flights[['carrier', 'dep_delay',
'arr_delay']]
flights = flights.dropna()
print(flights[:5])
# Write to feather file for reading from R
feather.write_dataframe(flights, "flights.feather")
##<---- I had some trouble with this on the R side
## but in the process of implimenting the csv to get
around it
## discovered the real issue was that I didn't know
where I was in dir str
with open("flights2.csv", "w", newline='') as
csv_file:
writer = csv.writer(csv_file, delimiter=',')
for line in flights:
writer.writerow(line)
```

And back to R

```
library(feather)
library(ggplot2)

# Read from feather and plot
flights <- read_feather(path = "./flights.feather")

# knitr::include_graphics('planes.png')
p <- ggplot(flights, aes(carrier, arr_delay)) + geom_point() +
    geom_jitter()
p
```

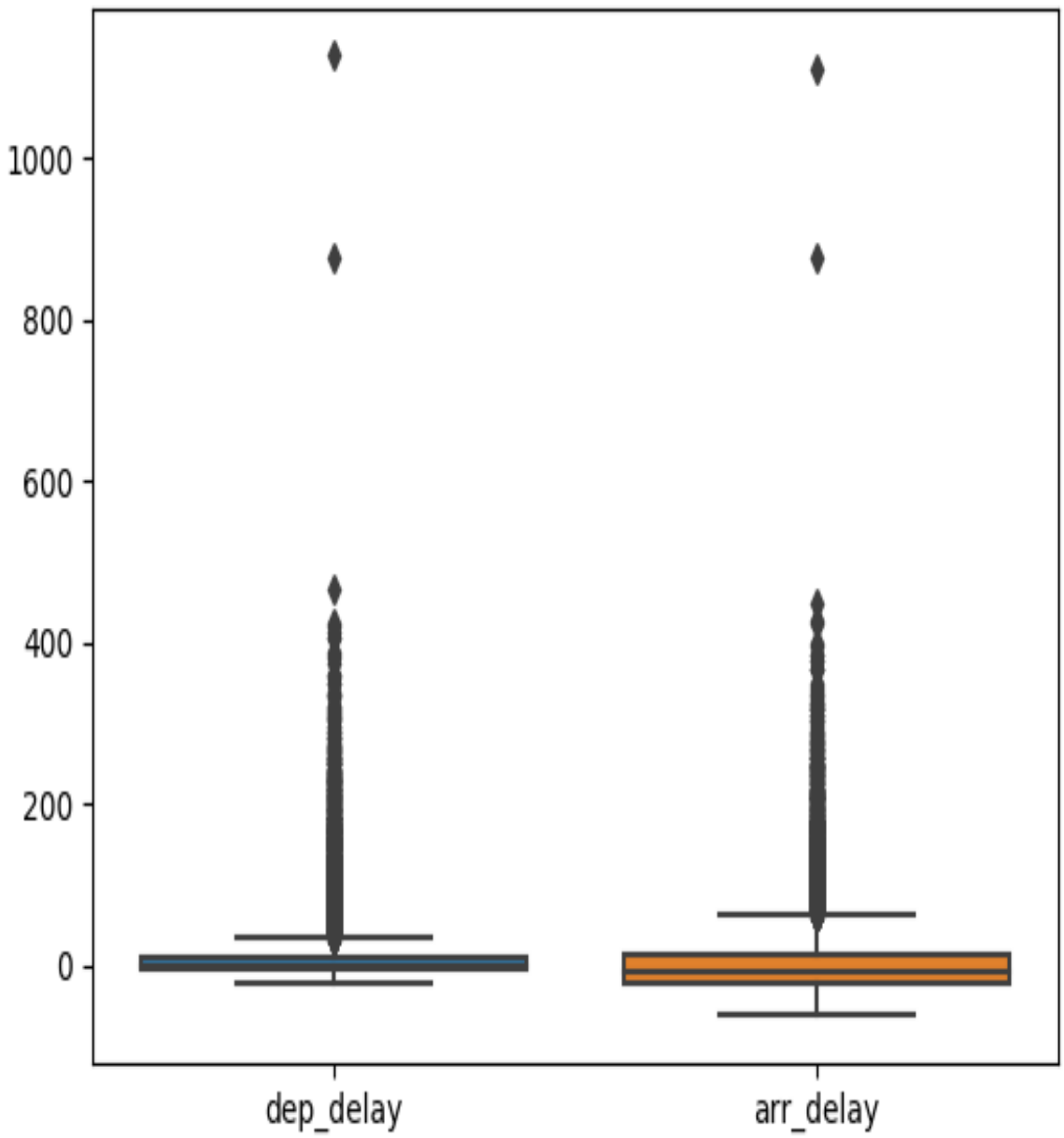


And what if we wanted to use Python to make the plot?

If I were better at python, I could make the plot more similar, but, for now…

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

# Read flights data and select flights to O'Hare
flights = pd.read_csv("flights.csv")
flights = flights[flights['dest'] == "ORD"]

# Select carrier and delay columns and drop rows with missing values
flights = flights[['carrier', 'dep_delay', 'arr_delay']]
flights = flights.dropna()
temp = flights.drop(['carrier'], axis=1)

sns.boxplot(data=temp)
plt.savefig("planes2.png")
```

Sad planes plot



Text mining in R

Fantastic online book from which much of this material originated: http://tidytextmining.com/tidytext.html#contrasting-tidy-text-with-other-data-structures

Remember the tidy concepts?

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

Let's use those concepts in a text mining framework.

Tidy text

Make a table with one token per row.

Token: meaninful unit of text
- word
- sentence
- email

Tokenization: process of splitting text into tokens

How do we get from raw text to tokens in tidy format??

dplyr, tidyr

How do we look at the data?

ggplot2, broom

Others.

tm, quanteda, Nulty

Tidy vs other text mining

- String: simple character vectors
- Corpus: raw strings plus metadata
- Document-term matrix: sparse matrix of documents vs terms

First text

```
text <- c("Because I could not stop for Death -", "He kindly stopped for m
    "The Carriage held but just Ourselves -", "and Immortality")

text
```

```
## [1] "Because I could not stop for Death -"
## [2] "He kindly stopped for me -"
## [3] "The Carriage held but just Ourselves -"
## [4] "and Immortality"
```

Tidy'ing, tokenizing,

```
original_books <- austen_books() %>% group_by(book) %>%
    mutate(linenumber = row_number(), chapter = cumsum(str_detect(text,
        regex("^chapter [\\divxlc]", ignore_case = TRUE)))) %>%
    ungroup()

# original_books

tidy_books <- original_books %>% unnest_tokens(word, text)

tidy_books
```

```
## # A tibble: 725,055 x 4
##                   book linenumber chapter        word
##                 <fctr>      <int>   <int>       <chr>
##  1 Sense & Sensibility          1       0       sense
##  2 Sense & Sensibility          1       0         and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0          by
##  5 Sense & Sensibility          3       0        jane
##  6 Sense & Sensibility          3       0      austen
##  7 Sense & Sensibility          5       0        1811
##  8 Sense & Sensibility         10       1     chapter
##  9 Sense & Sensibility         10       1           1
## 10 Sense & Sensibility         13       1         the
## # ... with 725,045 more rows
```

```
# remove stop words

data(stop_words)

tidy_books %>% count(word, sort = TRUE)
```

```
## # A tibble: 14,520 x 2
##     word     n
```

Word clouds

```
tidy_books %>% anti_join(stop_words) %>% count(word) %>%
    with(wordcloud(word, n, max.words = 100))
```

Python text mining

In my opinion, text mining is currently more natural and easier to accomplish in python. The reason:

1. packages
   beautifulsoup, and a bazillion others
2. lists, dicts, and tuples

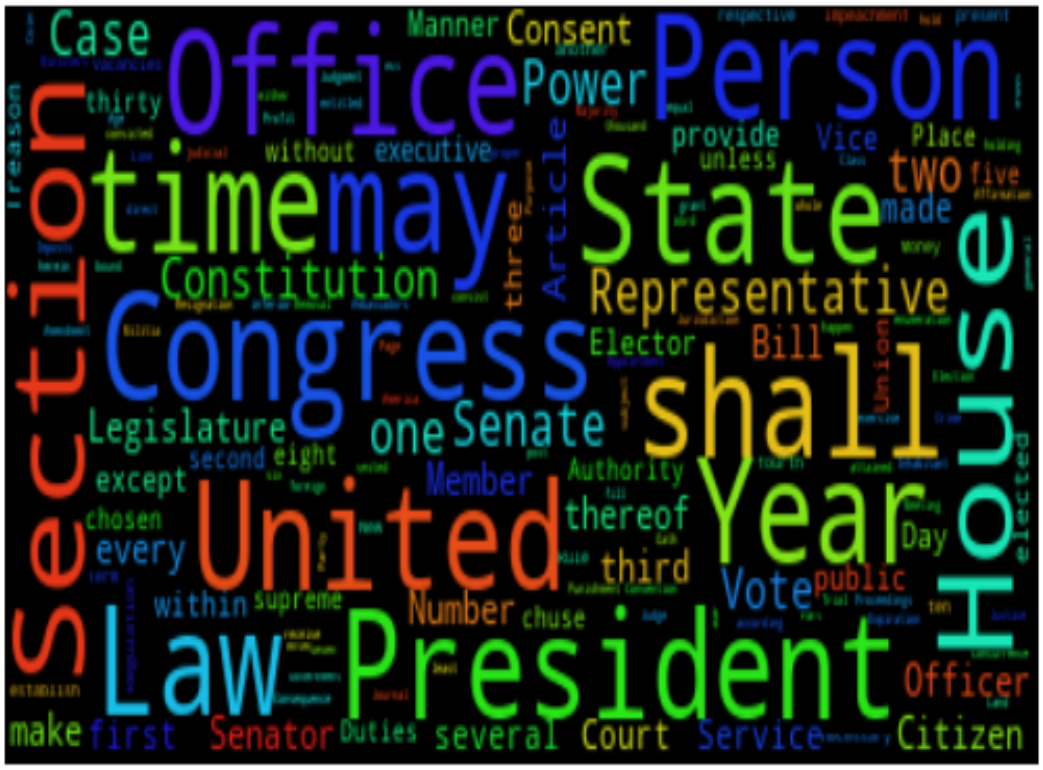
List – we know what that is from R
Dict – dictionary, essentially a lookup table
Tuples – essentially lists, but faster, immutable and can be used as keys on dictionaries

Quick text mine example

Ripped directly from the simple.py here:
https://github.com/amueller/word_cloud/blob/master/examples/simple

```
from os import path
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Read the whole text.

text = open('constitution.txt').read()

# Generate a word cloud image
wordcloud = WordCloud().generate(text)

# Display the generated image:
# the matplotlib way:

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")

# lower max_font_size
wordcloud = WordCloud(max_font_size=40).generate(text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")

plt.savefig("constitution.png")
```

Constitution as a word cloud



Homework 9