



R vs Python for Data Wrangling and Analysis

Joe Vargovich





Textbook referenced throughout the semester

<https://bookdown.org/dereksonderegger/444/>

Difficulty of Python Solutions

Module 1 - Familiarization

Module 2 - Data Frames

Module 3 - Graphing

Module 4 - Data Wrangling

Module 5 - Statistical Models

Module 6 - Flow Control

Module 7 - Factors

Module 8 - Data Structures

Module 9 - Importing Data

Module 10 - Functions



Easier in Python



Same in Python



More Difficult in Python

Module 11 - String Manipulation

Module 12 - Dates and Times

Module 13 - Data Reshaping

Module 14 - Advanced Graphing

Module 16 - Data Scraping

Module 17 - API Queries

Module 18 - Databases



Github repo for those who are interested.

<https://github.com/DovahCraft/R-and-Python-Data-Wrangling>



Familiarization



- Installed Python Anaconda, which is a version of Python that comes with many useful statistics libraries that I used throughout this course.
- I completed my exercises in a Jupyter notebook
 - Similar to Rmarkdown, was previously referred to as an Ipython notebook
 - Ipython is the backend to Jupyter Notebook, runs a Python kernel behind the notebook file itself.



Jupyter Notebook Structure

Author: Joseph Vargovich

^ [1]
v
+ |

```
▶ Mi  
  
#Import libraries  
import pandas as pd  
import requests  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from numpy import random as rand
```

Exercise 1 - FloridaLakes ordered bar graph

[10]

```
▶ Mi  
  
#Load the data from csv  
FloridaLakes = pd.read_csv("FloridaLakes.csv")  
  
# Sort the lakes by their AvgMercury  
FloridaLakes = FloridaLakes.sort_values(by='AvgMercury', ascending=False)  
FloridaLakes.head()  
  
#Set custom dimensions for the figure  
sns.set(rc={'figure.figsize':(11.7,10.27)})  
#Graph the data  
lakePlot = sns.barplot(x="AvgMercury", y="Lake", data=FloridaLakes, color="b")  
lakePlot.set_title('Average Mercury Content of Lakes in Florida')
```

Data Frames and Data Wrangling





Module 2 - Data Frames

- Data frames are provided by the Pandas library
- Basic reading of csvs and modification of data frames.
- Similar functionality to R data frames and dplyr.
- No pipes in Python!



Piping (R magrittr)

```
```{r}
sampleStr = "Hello world!"

sampleStr = sampleStr %>%
 toupper() %>%
 tolower() %>%
 str_split(" ")
sampleStr
```
```


=

Method Chaining (Python)

```
▶ ▶≡ M4

sampleStr = "Hello world"
sampleStr.lower().upper().split()

['HELLO', 'WORLD']
```



Module 4, 7, 13 - Data Cleaning, Wrangling, and Reshaping

- Joins are implemented in Pandas using the `merge()` method
 - There are a variety of ways to do this operation
 - There is a `concat()` function that mimics `cbind` and an `append()` method that mimics `rbind`.
- Pivots are also implemented through the `pivot()` method
 - There are a variety of ways to do this as well, including different pivot methods to pivot wider or longer like in R. I used `pivot()` to pivot wider for the homework.
- Supports reordering of rows and columns, as well as changing of labels.
- Filtering of rows and selecting of columns are commonplace operations that are supported in Python.

Defining Pandas Dataframes From Scratch Using Python Dictionaries

```
▶ M4

#Dictionaries that define two pandas dataframes and their column->values mapping.
firstDict = {'Name': ["Alice", "Bob", "Charlie"], 'Car': ["Ford F150", "Tesla Model III", "VW Bug"]}
secondDict = {'First.Name': ["Bob", "Charlie", "Alice"], 'Pet': ["Cat", "Dog", "Rabbit"]}

firstDf = pd.DataFrame(data=firstDict)
print("First Dataframe")
print(firstDf)
print("\n\n")

secondDf = pd.DataFrame(data=secondDict)
print("Second Dataframe")
print(secondDf)
print("\n\n")
```

First Dataframe

| | Name | Car |
|---|---------|-----------------|
| 0 | Alice | Ford F150 |
| 1 | Bob | Tesla Model III |
| 2 | Charlie | VW Bug |

Second Dataframe

| | First.Name | Pet |
|---|------------|--------|
| 0 | Bob | Cat |
| 1 | Charlie | Dog |
| 2 | Alice | Rabbit |

Concat (like cbind) vs Joins in Python

▶ M1

```
#a. Join together with pd.concat and join options.  
#Concat (like cbind on axis=1)  
concatResult = pd.concat([firstDf, secondDf], axis=1, sort=False)  
print("\n1. Concatted Tables: \n" )  
print(concatResult)  
  
#Same as above but using a LEFT JOIN operation  
joinResult = firstDf.merge(secondDf, how='left', left_on='Name', right_on='First.Name')  
print("\na. Joined Tables: \n" )  
print(joinResult)
```

1. Concatted Tables:

| | Name | Car | First.Name | Pet |
|---|---------|-----------------|------------|--------|
| 0 | Alice | Ford F150 | Bob | Cat |
| 1 | Bob | Tesla Model III | Charlie | Dog |
| 2 | Charlie | VW Bug | Alice | Rabbit |

a. Joined Tables:

| | Name | Car | First.Name | Pet |
|---|---------|-----------------|------------|--------|
| 0 | Alice | Ford F150 | Alice | Rabbit |
| 1 | Bob | Tesla Model III | Bob | Cat |
| 2 | Charlie | VW Bug | Charlie | Dog |

Exercise 1 of Chapter 13 showing some data wrangling and pivoting (wider)

```
▶ ML

#a. Load the data, na_values in the list will be removed. '' and ' ' are na values now.
Survey = pd.read_csv("http://www.lock5stat.com/datasets/StudentSurvey.csv", na_values=['', ' '])
Survey.head()

#b. Cleanup and select what we need.
Survey = Survey[['Year', 'Gender']]

#Grab the amount of Female and Male Students in each Year.
# Groupby will convert our dataframe into a Series object, ugh!
Survey = Survey.groupby(['Gender', 'Year']).size()

#Convert our Series back to a dataframe.
Survey = Survey.to_frame(name = 'n').reset_index()
print(Survey)

#Create a pivot table
pivotTable = Survey.pivot(index='Gender', columns='Year', values='n')
#Set the column order
pivotTable = pivotTable[['FirstYear', 'Sophomore', 'Junior', 'Senior']]
#Set the row order
pivotTable.reindex(["M", "F"]).head()
```

| | Gender | Year | n |
|---|--------|-----------|----|
| 0 | F | FirstYear | 43 |
| 1 | F | Junior | 18 |
| 2 | F | Senior | 10 |
| 3 | F | Sophomore | 96 |
| 4 | M | FirstYear | 51 |
| 5 | M | Junior | 17 |
| 6 | M | Senior | 26 |
| 7 | M | Sophomore | 99 |

| | Year | FirstYear | Sophomore | Junior | Senior |
|--------|------|-----------|-----------|--------|--------|
| Gender | | | | | |
| M | | 51 | 99 | 17 | 26 |
| F | | 43 | 96 | 18 | 10 |

Graphing and Statistical Modeling





Module 3 - Graphing

- Graphing with two main libraries: Matplotlib and Seaborn
- Seaborn is the easier of the two, and I used it the most.
 - <https://seaborn.pydata.org/examples/index.html> Example gallery for seaborn
- Matplotlib is more detailed and necessary for complex graphs, but it is less readable and harder to use than ggplot2.
- Ggplot2 is as, if not more usable than seaborn with better flexibility and power than Matplotlib. Certainly a win for R.



Module 5 - Statistical Modeling

- Very similar to the information/functions available in R, but R is easier to work with and more suited to in-depth statistical analysis.
 - Summary statistics, fitting models to data, etc.
 - Used libraries: Statsmodels, numpy, seaborn
- In the statistical models homework, it was hard and less straightforward to display regression lines on my graphs
- Graphing was a challenge overall when combined with statistical modeling, but I could access the information I needed.

Functions, Control Flow, Loops, and Basic Data Structures





Modules 6 and 10 - Functions, Conditionals, and Loops

- Function scope is created via indentation, not curly braces!
- This is pretty much a Python only thing, but it makes code look a lot cleaner and forces a readable style of indentation and formatting.
- Follows PEP8 standard for Python.

R Fizzbuzz

```
# Exercise 4 - FizzBuzz function in R
```{r}
fizzbuzz = function(n){
 output = NULL
 for(i in 1:n){
 if((i %% 3 == 0) && (i %% 5 == 0)){
 output[i] = "FizzBuzz"
 }
 else if(i %% 3 == 0){
 output[i] = "Fizz"
 }
 else if(i %% 5 == 0){
 output[i] = "Buzz"
 }
 else{
 output[i] = i
 }
 }
 return(output)
}
fizzbuzz(20)
```
```

Python Fizzbuzz

Exercise 4 - FizzBuzz function

```
[14] ▶ ⌵ M4
def fizzbuzz(n):
    for i in range(1,n):
        if((i%3 == 0) and (i%5==0)):
            print('FizzBuzz')
        elif(i % 3 == 0):
            print('Fizz')
        elif(i % 5 == 0):
            print('Buzz')
        else:
            print(i)

    fizzbuzz(21)
```



Module 8 - Data Structures

- Pretty much the same data structures in both languages.
- Dataframes are formed differently
 - I would first create a Python dictionary that maps column names to values
 - Then I would
- No recycling rule in Python! Cannot add arrays (vectors) of different sizes!



DATA STRUCTURES





```
[3] ▶ MI

#Create a numpy vector
vec_a = np.array([2,4,6])
vec_b = np.array([8,10])

vec_c = vec_a + vec_b

vec_c

-----
ValueError                                Traceback (most recent call last)
<ipython-input-3-02a24d3397ae> in <module>
      4
      5
----> 6 vec_c = vec_a + vec_b
      7
      8 vec_c

ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

Python No Recycling Rule
ERROR

```
##{r}
vec_a = c(2,4,6)
vec_b = c(8,10)

vec_c = vec_a + vec_b

vec_c
```

longer object length is not a multiple of shorter object length[1] 10 14 14

R Using Recycling Rule
WARNING

Other Key Functions





Module 9 - Reading Excel Files



- Pandas had a full equivalent of the readxl R package used in the R solution.
- I was able to read in and clean the data successfully without much issue in both R and Python.



Module 11 - String Manipulation

- I used the re Python library to create and match regular expressions.
- Very similar in difficulty to the R solution. Regular expressions are a widely used tool for parsing and input matching, no surprise there.

Module 16 - Web Scraping

BeautifulSoup

- I used BeautifulSoup + Pandas to do web scraping
 - Pandas did a LOT of the data cleaning we needed to do ourselves in R for me!
- It was very convenient to process and use the data once I figured out how to access what I wanted with BeautifulSoup
- Scrapy is another library for Python web scraping but I did not use it.



Scrapy

Python
(No numeric cleanup
needed!)

```
#Pandas has a built in method to convert web tables to a dataframe
url = "https://www.iihs.org/topics/fatality-statistics/detail/state-by-state"

#Use this special request call to mask our User-Agent and avoid a 403 forbidden HTTP error.
req = Request(url, headers={'User-Agent': 'Mozilla/5.0'})
page = urlopen(req)
#Parse our page and create a soup object.
html_bytes = page.read()
html = html_bytes.decode("utf-8")
soup = BeautifulSoup(html, "html.parser")
#Filter out the U.S. total row from the data
tables = soup.find_all('table')
#This autoformatted the numeric values into usable values.
df = pd.read_html(str(tables), header=1)[0]
df = df[['State', 'Deaths per 100,000 population']]
df = df[df.State != "U.S. total"]
print(df.head())

#Plot State vs Deaths in a bar chart.
sns.barplot(x="Deaths per 100,000 population", y="State", data=df, color="b").set(title="Crash Deaths per state per 100,000 people.")
```

R

```
```{r, fig.height=10}
url = "https://www.iihs.org/topics/fatality-statistics/detail/state-by-state"
page = read_html(url)

CrashData = page %>%
 html_nodes('table') %>%
 .[[1]] %>%
 html_table(header=FALSE, fill=TRUE) %>%
 select(X1,X2, X6) %>%
 slice(-1:-2) %>%
 magrittr::set_colnames(c('State','Population', 'Deaths')) %>%
 mutate_at(vars(matches('Deaths')), str_remove_all, ',') %>% # remove all commas
 mutate_at(vars(matches('Deaths')), str_remove, '\\[[0-9]+\\]') %>% # remove [7] stuff
 mutate_at(vars(matches('Deaths')), as.numeric) %>%
 mutate_at(vars(matches('Population')), str_remove_all, ',') %>% # remove all commas
 mutate_at(vars(matches('Population')), str_remove, '\\[[0-9]+\\]') %>% # remove [7] stuff
 mutate_at(vars(matches('Population')), as.numeric) %>%
 filter(State != "U.S. total")
head(CrashData)

#Plot our data
plotOfDeaths = ggplot(CrashData, aes(x=State, y=Deaths)) +
 geom_col() +
 coord_flip()
plotOfDeaths
```
```

Module 18 - Databases



- Both R and Python have Sqlite3 support.
- Very easy to use and SQL code can be executed dynamically using function calls to the Sqlite3 API.

Summary





Tidyverse vs Pandas



Tidyverse

Excellent function names that are descriptive (such as select, apply, filter)

Includes ggplot2! The best data visualization package around!

Pipes, if you're into that from magrittr.

Code is written in R, which yields slower performance

Pandas

Not very descriptive function names, at least not as readable as Tidyverse

Matplotlib + Pandas is more difficult to use than ggplot2's syntax

Method chaining accomplished the same thing as magrittr pipes.

Written in C, which is lightning fast. Must use optimal methods though!



Overall, who wins?

- Overall I like R better as it allows you to create and customize a more readable workflow through better function names and pipes!
- Python Pandas is a very good library, it does a lot of work for you that you would have to manually figure out in R. (Web scraping is the best example of this).
- It's really up to personal preference. Learning R will put you in a **very** good position to learn Python and its syntax.
- R looks more like a C-like programming language than Python. Users of C, Java, C++, etc will transition into learning and using R nearly seamlessly.