ISA 414 – Managing Big Data

Lecture 4 – Preliminaries (Part III)

Advanced Variable Types

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu

MIAMI UNIVERSITY

Lecture Objectives

- Review Homework #2
- Learn about multivalued variable types
  - List
  - Data Frames
- “Case study”
  - Learn about web logs
  - Learn about the CSV file format
    - Learn how to load/save CSV files to/from Python

Lecture Instructions

1. Download the notebook "*Lecture 4.ipynb*" available on Canvas
2. Download the data file "*web_log.csv*" available on Canvas
   - **Make sure both files are inside the same folder**
3. Open the notebook "*Lecture 4.ipynb*" with VS Code

Introduction to Python

- In the previous lecture, we learned how to define single-valued variables
  - Types: int, float, bool, str
- More complex variable types can be defined by allowing multivalued variables
  - List
  - Data Frame

Text Type:	**str**
Numeric Types:	**int, float, complex**
Sequence Types:	**list, tuple, range**
Mapping Type:	**dict**
Set Types:	**set, frozenset**
Boolean Type:	**bool**
Binary Types:	**bytes, bytearray, memoryview**

Not covered in this course

List

- Used to store multiple values in a single variable
- One of 4 built-in data types in Python used to store collections of data
  - The other three are *Tuple*, *Set*, and *Dictionary*
- Important points to remember
  - List items are ordered, changeable, and allow for duplicate values
  - List items are indexed
    - **The first item has index [0]**

List

- Lists are created using square brackets
  - Example: my_first_list = ["Paul", "John", "Ringo", "George"]
- A list can contain different data types
  - Example: my_second_list = ["Paul",1,"John", True,"Ringo", 2.0, "George"]
- One can obtain the length (*i.e.*, number of elements) of a list by using the len() function
  - Example: len(my_second_list)

List

- Accessing elements
  - List elements are indexed
    - One can access an element by referring to its index number
    - **The first item has index 0**
    - Example: my_second_list[2]
  - Negative indexing
    - Negative indexing means start from the end
    - Example: accessing the last element: my_second_list[-1]

List

- Accessing elements
  - One can create a range of elements to be retrieved
    - Syntax: start:end:step
      - Important: the end-value is not included, and step is optional
      - Example: access all the elements from position 0 to 6 by 2

```
my_second_list[0:6:2]
```

    - By leaving out the start (end) value, the range will start at the first (end at the last) item
      - Example: my_second_list[0:]

```
my_second_list[:5]
```

List

- Adding elements
  - The append() function adds an element to the end of the list
    - Example: my_second_list.append("ISA 414")
  - The insert() function adds an element to a specific locations of the list
    - Example: my_second_list.insert(0, "first element")
  - One can concatenate two lists by summing them
    - Example: my_third_list = my_first_list + my_second_list

List

- Removing elements
  - The remove() function removes elements based on values
    - Example: my_third_list.remove("ISA 414")
  - The pop() function removes elements based on indexes
    - Example: my_third_list.pop(0)

List

- Lists and loops
  - We will often loop through a list of items

index	values
0	Paul
1	John
2	Ringo
3	George
4	first el…
5	Paul
6	1
7	John
8	true
9	Ringo
10	2
11	George
12	ISA 414

- Looping through item values:

```
for x in my_third_list:
    print(x)
```

- Looping through index numbers:

```
for x in range(len(my_third_list)):
    print(my_third_list[x])
```

- Note that len() gives the number of elements
  - Then, range(len( )) creates a range from position 0 to the total number of elements in a list minus one

List

- Lists and loops
  - We will often create loops to add elements to a list
    - Example:

```
my_fourth_list = []
for i in my_third_list:
    if type(i) == str:
        my_fourth_list.append(i)
```

  - Overall, loops are very slow in interpreted languages, such as Python and R
    - They work on each element at a time, instead of a whole list

List

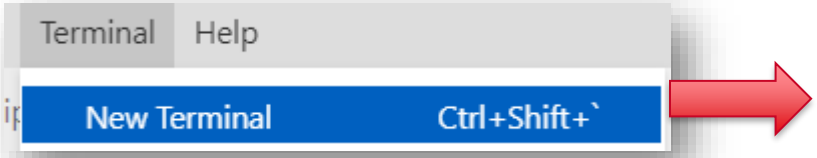
- Lists and loops
  - **List comprehension** offers a shorter and more efficient way of creating a new list
    - Syntax: newlist = [*value* for *item* in *list* if <condition>]
      - The if-statement is optional; it filters values from *list*
      - Example: my_fourth_list = [i for i in my_third_list if type(i) == str]
    - Note that the syntax changes if there is an else statement
      - Syntax: newlist = [*value_1* if <condition> else *value_2* for *item* in *list*]
      - Example: another_list = [i if type(i) == str else None for i in my_third_list]
    - List comprehension will be incredibly useful when manipulating textual data

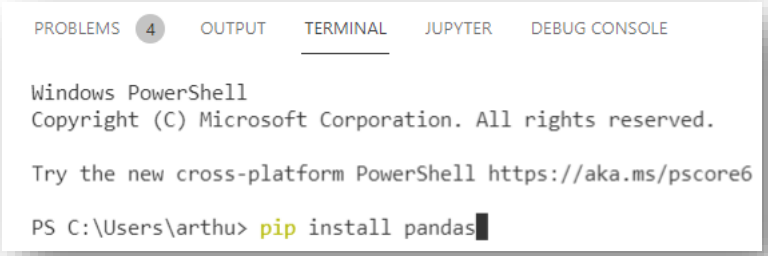
Data Frames

- Data frames are two dimensional structures that follow a tabular format
  - There is no native data frame structure in Python
- We need to install the **Pandas** “package” that offers such a functionality
  - Packages are called **modules** or **libraries** in Python
  - More about modules in our next class
- Note that columns in a Pandas data frame are called **Series**
  - Special type of lists

Data Frames

- Go to the terminal type: pip install pandas
  - Menu item *Terminal* -> "*New Terminal*"





- Let's import Pandas and create a data frame with two columns

```
import pandas
df = pandas.DataFrame({"Col_1":another_list, "Col_2":my_third_list})
```

Data Frames

- Note how each row has a “name” (index)
  - Unique identifier of the row

	Col_1	Col_2
0	Paul	Paul
1	John	John
2	Ringo	Ringo
3	George	George

  - Accessing columns by name
    - Example: df["Col_1"]
      df[["Col_1", "Col_2"]]

Data Frames

- Accessing rows and/or columns by location
  - Example: selecting all rows (:) for the second column

```
df.iloc[:,1]
```

  - Example: selecting all the columns (:) for the first and third rows

```
df.iloc[ [0,2], :]
```

Data Frames

- Adding rows to a data frame
  - The append () function adds rows similar to how we created data frames
    - Example: df = df.append({"Col_1":1, "Col_2":2}, ignore_index=True)

- Removing rows from a data frame
  - The drop() function removes elements based on indexes
    - Example: df = df.drop(10)

Case Study: Analyzing Web Logs

Case Study: Analyzing Web Logs

- We are now in a position to start talking about and practicing the management of big data
- Keep in mind that:
  - Big data management ≠ data storage
    - Narrow perspective
  - Big data management is about how to:
    - Effectively collect, process, store, and analyze potentially large and unstructured data sets

Case Study: Analyzing Web Logs

- Throughout this course, we will learn about different data types and formats
  - Today we learn about (web) logs and the CSV format

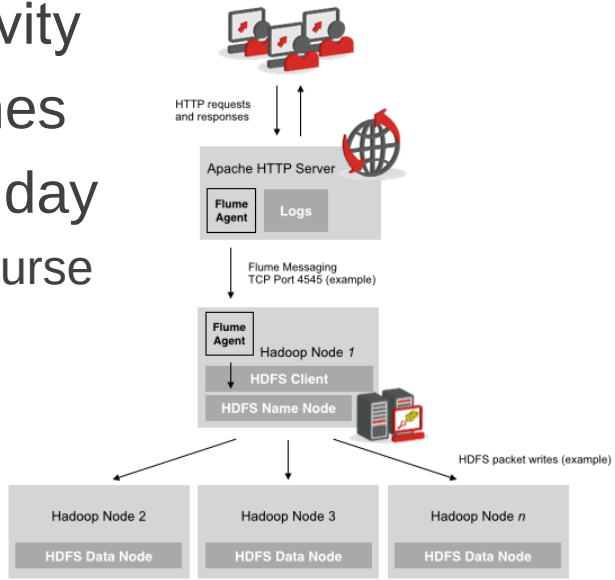
- Background story
  - You work for a company that offers very expensive butler services
    - Target market: European billionaires
  - Due to the recent economic recession, your company is considering to expand to emerging markets

Case Study: Analyzing Web Logs

- Background story
  - Question: which countries are expected to have high demand for the offered butler services?
    - The answer will guide marketing efforts
  - Don Draper, the Chief Marketing Officer (CMO), asks you to find an answer

Case Study: Analyzing Web Logs

- Talking to the IT staff, you learn the following about the company's data infrastructure
  - Whenever a user accesses the company's website, an Apache web server logs the user's activity
  - These logs are eventually sent in batches to a Hadoop cluster at the end of each day
    - We will learn more about that later in this course



Case Study: Analyzing Web Logs

- Web logs
  - Whenever a user access a company's web page (*e.g.*, www.butlerforyou.com), that is how Apache web server stores the user's activity

64.246.220.203 - - [14/Jun/2014:10:30:20 -0400] "GET /home HTTP/1.1" 200 760 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"

64.246.220.203 - - [14/Jun/2014:10:30:30 -0400] "GET /about/butlers HTTP/1.1" 200 1671 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153"

Case Study: Analyzing Web Logs

- Web logs
  - (Web) logs are often (semi-) structured data
    - Not ready for statistical analysis (must be pre-processed)
    - There are patterns (structures) in the data

```
1 79.133.215.123 - - [14/Jun/2014:10:30:13 -0400] "GET /home HTTP/1.1" 200 1671 "-"
  "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/35.0.1916.153 Safari/537.36"
```

IP_Address	Access_Date	HTML_Method	Requested_Page	HTTP_Version	Code_1	Code_2	User_Agent
79.133.215.123	14/Jun/2014:10:30:13	GET	/home	1.1	200	1671	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36

Case Study: Analyzing Web Logs

- Knowing that, you ask the IT staff to do the following for you
  1. Extract the IP addresses from the web logs
  2. From each IP address, obtain the user's latitude and longitude
  3. From the user's latitude and longitude, obtain the user's city and country
  - Your ingenious idea is to count the number of accesses per country, and use such a number as a **proxy** for the butler service demand

- The IT staff complies with your request and gives you a <u>CSV file</u> called "*web_logs.csv*" (available on Canvas)
  - You will learn how to do all of the above steps in the upcoming lectures

Case Study: Analyzing Web Logs

- Comma-Separated Values (CSV) files
  - A CSV-file is a straightforward text file
    - Each line is an observation
    - The values or columns are separated by a semicolon (;) or a comma (,)
  - Standard format used in data analysis
  - Usually, column names are in the first line of the file
  - We will analyze the data in the CSV file "*web_log.csv*", which is available on Canvas

Case Study: Analyzing Web Logs

- Comma-Separated Values (CSV) files
  - If you open the file "*web_log.csv*" with VS Code

Column (variable) names

Data (values)

web_log.csv

```
City,Client_IP,Country_Name,Latitude,Longitude
Singapore,128.199.234.236,Singapore,1.293099999,103.8557968
Singapore,128.199.234.236,Singapore,1.293099999,103.8557968
Singapore,128.199.234.236,Singapore,1.293099999,103.8557968
Singapore,128.199.234.236,Singapore,1.293099999,103.8557968
Singapore,128.199.234.236,Singapore,1.293099999,103.8557968
Mountain View,66.249.76.236,United States,37.38600159,-122.0838013
Guiyang,222.85.131.87,China,26.58329964,106.7166977
Guiyang,222.85.131.87,China,26.58329964,106.7166977
Shanghai,101.226.168.225,China,31.04560089,121.3996964
Mountain View,66.249.76.225,United States,37.38600159,-122.0838013
Shanghai,101.226.166.230,China,31.04560089,121.3996964
Shanghai,101.226.166.231,China,31.04560089,121.3996964
Zhengzhou,182.118.25.227,China,34.68360138,113.5325012
Zhengzhou,182.118.25.227,China,34.68360138,113.5325012
Zhengzhou,182.118.25.228,China,34.68360138,113.5325012
Zhengzhou,182.118.25.230,China,34.68360138,113.5325012
Zhengzhou,182.118.25.229,China,34.68360138,113.5325012
Hyderabad,122.175.18.128,India,17.37529945,78.47440338
Shanghai,101.226.166.230,China,31.04560089,121.3996964
Shanghai,101.226.166.234,China,31.04560089,121.3996964
Shanghai,101.226.166.233,China,31.04560089,121.3996964
Shanghai,101.226.166.231,China,31.04560089,121.3996964
```

Case Study: Analyzing Web Logs

- Comma-Separated Values (CSV) files
  - If you open the file "*web_log.csv*" with spreadsheet software, like Excel, that is how it will look like

1	City	Client_IP	Country_Name	Latitude	Longitude
2	Singapore	128.199.234.236	Singapore	1.293099999	103.8557968
3	Singapore	128.199.234.236	Singapore	1.293099999	103.8557968
4	Singapore	128.199.234.236	Singapore	1.293099999	103.8557968
5	Singapore	128.199.234.236	Singapore	1.293099999	103.8557968
6	Singapore	128.199.234.236	Singapore	1.293099999	103.8557968
7	Mountain View	66.249.76.236	United States	37.38600159	-122.0838013
8	Guiyang	222.85.131.87	China	26.58329964	106.7166977
9	Guiyang	222.85.131.87	China	26.58329964	106.7166977
10	Shanghai	101.226.168.225	China	31.04560089	121.3996964

Case Study: Analyzing Web Logs

- Let's analyze the data
  - Importing CSV files into Python
    - In order to perform our analysis, we need to import the data from the CSV file into Python
    - We can do so by using the function read_csv in the Pandas module

```
web_data = pandas.read_csv("web_log.csv")
```

    - The first argument of the above function is the location of the CSV file
      - In the above example, the file "*web_log.csv*" must be located in same the same directory as the notebook

Case Study: Analyzing Web Logs

- We can now count the number of web pages requests per country
  - In statistical terms, we want to calculate the frequency table for the variable "*Country_Name*"
  - To do so, we will use the function *value_counts*
    - This function can be applied to data frames or series

```
pandas.value_counts(web_data["Country_Name"])
```

```
India            357
Thailand         232
United States    190
```

Case Study: Analyzing Web Logs

- One can also save a data frame or a series in Python to a CSV file using the function to_csv
  - Example: suppose one wants to save the frequency table to a csv file

```
results = pandas.value_counts(web_data["Country_Name"])
results.to_csv("ftable.csv")
```

Homework #3

Suppose your company now wants to determine the demand for butlers at the city level, as opposed to country level

- Write a script that solves the following tasks and report you answers on Canvas
- Task #1: save the frequency table per city to a variable called city_freq
- Task #2: visualize the created series
- Task #3: which city should your company focus its marketing efforts on?

Summary

- We have learned about more advanced data types/structures in Python
  - List and Data Frames
- We learned about web logs and the CSV format
- Next lecture
  - Introduction to Python: user-defined functions and modules