# ISA 414 – Managing Big Data

## Lecture 8 – Data Collection

*Web Scraping (Part II)*

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu

MIAMI UNIVERSITY

# Announcements

- Assignment 2
  - Now available on Canvas
  - **New deadline**: Friday, September 24th, at 11:59 pm
- Homework 5
  - **New deadline**: Monday, September 27th, at 11:59 pm
- Assignment 1
  - I will review the solutions on Tuesday

# Lecture Objectives

- Learn how to scrap data from web pages
  - Using regular expressions

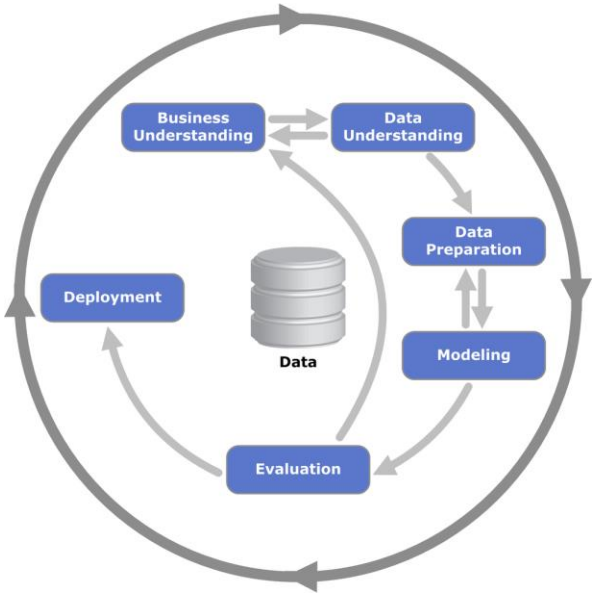
# Lecture Instructions

- Download the script "*Lecture 8.ipynb*" available on Canvas

- Open the file "*Lecture 8.ipynb"* with VS Code

# CRISP-DM

- From previous lecture
  - Big data (data analytics) project management



# CRISP-DM

- We are currently focusing on data understanding and collection
  - To a less degree on data preparation
  - Topic of this and the next three lectures
- One very important source of data is the web
  - Big picture: any file freely available on the web can be downloaded and used in an analytics process
    - One must be very careful with copyright violations
- Eventually, we will move on to data analysis
  - Modeling and evaluation

# Web Scraping

- Let's continue scraping data from the web
  - Recall the background story
    - You work for a sports analytics company, which recently signed a contract with ESPN
      - Project: design a predictive model that predicts the final NFL regular season standings per conference
      - The project manager of the analytics team you belong to follows the CRISP-DM methodology
      - Your group is now in the data understanding phase, brainstorming about the relevant variables/features one should include in the predictive model
      - You suggest the number of NFL titles each team won before might be a relevant feature

# Background Story

- http://www.espn.com/nfl/superbowl/history/winners

| Super Bowl Winners and Results | | | |
|---|---|---|---|
| NO. | DATE | SITE | RESULT |
| I | Jan. 15, 1967 | Los Angeles Memorial Coliseum | Green Bay 35, Kansas City 10 |
| II | Jan. 14, 1968 | Orange Bowl (Miami) | Green Bay 33, Oakland 14 |
| III | Jan. 12, 1969 | Orange Bowl (Miami) | New York Jets 16, Baltimore 7 |
| IV | Jan. 11, 1970 | Tulane Stadium (New Orleans) | Kansas City 23, Minnesota 7 |
| V | Jan. 17, 1971 | Orange Bowl (Miami) | Baltimore 16, Dallas 13 |
| VI | Jan. 16, 1972 | Tulane Stadium (New Orleans) | Dallas 24, Miami 3 |
| VII | Jan. 14, 1973 | Los Angeles Memorial Coliseum | Miami 14, Washington 7 |
| VIII | Jan. 13, 1974 | Rice Stadium (Houston) | Miami 24, Minnesota 7 |
| IX | Jan. 12, 1975 | Tulane Stadium (New Orleans) | Pittsburgh 16, Minnesota 6 |
| X | Jan. 18, 1976 | Orange Bowl (Miami) | Pittsburgh 21, Dallas 17 |
| XI | Jan. 9, 1977 | Rose Bowl (Pasadena, Calif.) | Oakland 32, Minnesota 14 |
| XII | Jan. 15, 1978 | Superdome (New Orleans) | Dallas 27, Denver 10 |
| XIII | Jan. 21, 1979 | Orange Bowl (Miami) | Pittsburgh 35, Dallas 31 |

- How can we collect the above data?

# Web Scraping

- Our first solution
  - Find the index (position) of all the HTML tags <td> and </td>
  - Extract what is in between the tags

- This solution is independent of the underlying programming language
  - Relies on very basic **regular expressions** to find patterns
    - *E.g.*, “<td>”, “</td>”

# Web Scraping

- Regular expressions
  - Very powerful language with its own syntax and semantics
    - Can be used in conjunction with virtually any programming and querying languages
      - JAVA, C++, SQL, R, …
  - Concise and powerful method for searching for a specific string pattern
    - We will learn how to write more complex regular expressions today
      - Example: <td>*some text*</td>

vague specification

# Web Scraping

- Regular expressions can be used in conjunction with most string functions

- In our classes, we shall focus on the functions search, finditer, and sub
  - search: finds the first occurrence of a pattern
  - finditer:  can be used to find positions and matches of a pattern
  - sub: replace the matches of a pattern with another string

- All the above functions are from the re module
  - Documentation: https://docs.python.org/3/library/re.html

# Regular Expressions

- Up to now, we used regular expressions to search for patterns containing entire strings (no vagueness)
  - *E.g.*, “HTTP” or “<td>”

- But we can be more vague in our expressions
  - This vagueness is often captured by the fundamental building blocks that match a single character
  - *E.g.*, the expression “San Francisco [1-9]” will look for the string “San Francisco ” followed by one character: a digit between 1 and 9
    - **The brackets [ ] in a regular expression indicate that we are looking for one particular character**

# Regular Expressions

- Let’s try the previous expression

```
import requests
import re

r = requests.get("http://www.espn.com/nfl/superbowl/history/winners")
html_code = r.text

hits = re.finditer("San Francisco [0-9]", html_code)
positions = [m.start(0) for m in hits]
positions
```

- Result: positions where the matches start

# Regular Expressions

Syntax:

some of the most important expressions to find individual characters

| Expression | Description |
|---|---|
| . | Any character |
| [0-9] | Any numeric character in the range 0, 1, 2, . . ., 9 |
| [A-Z] | Any uppercase alphabetic character in the range A, B, . . . , Z |
| [a-z] | Any lowercase alphabetic character in the range a, b, . . . , z |
| \s | Space characters: tab, newline, vertical tab, form feed, carriage return, space |

More expressions at: https://docs.python.org/3/library/re.html

# Regular Expressions

- Example #1: has San Francisco ever been to a Super Bowl?
  - Pattern: search for the first occurrence of “San Francisco”
  - Two possible ways:
    1. Search for the whole string “San Francisco”
    2. Search for “San” followed by a space character \s, followed by “Francisco”

```
hits = re.finditer("San Francisco", html_code)
hits = re.finditer("San\sFrancisco", html_code)
```

# Regular Expressions

- Example #2: retrieve the date of each Super Bowl
  - Pattern: [A-Z][a-z][a-z]. [0-9][0-9], [0-9][0-9][0-9][0-9]

    Example: J a n . 1 5 , 1 9 6 7
    - Not a perfect pattern (why?)

```
hits = re.finditer("[A-Z][a-z][a-z]. [0-9][0-9], [0-9][0-9][0-9][0-9]", html_code)
positions = [m.start(0) for m in hits]
```

  - Note that the above code only retrieves the position of the patterns
    - Not the matched values

# Regular Expressions

- Example #2: retrieve the date of each Super Bowl
  - Retrieving matched values
    - Previously, we looked for two separate regular expressions to get data in between the patterns' locations
    - We are now looking for a single regular expression

```
hits = re.finditer("[A-Z][a-z][a-z]. [0-9][0-9], [0-9][0-9][0-9][0-9]", html_code)
matches = [m.group(0) for m in hits]
```

# Regular Expressions

- Difference between start() and group() in finditer
  - start: returns the positions where the hits start
  - group: returns the precise matches
- So, what's with the argument '0' in start() and group()?
  - One can define "groups" in a pattern using parenthesis
    - The value 0 returns the positions/matches for the entire hit
    - The values 1, 2, 3, … return the positions/matches for the first, second, third, … groups
  - Example: creating two groups for the previous pattern

```
hits = re.finditer("([A-Z][a-z][a-z]). ([0-9][0-9]), ([0-9][0-9][0-9][0-9])", html_code)
matches_group_1 = [m.group(1) for m in hits]
```

# Regular Expressions

- Up to now, we coded repetitions of a character by explicitly writing the same pattern multiple times
  - Example: [A-Z][a-z][a-z]. [0-9][0-9], [0-9][0-9][0-9][0-9]
    - *E.g.*, the last [0-9] pattern is written multiple times
  - There are more general ways of defining repetitions:

| Quantifier | Meaning |
|---|---|
| * | The preceding term will be matched zero or more times |
| ? | The preceding term is optional and will be matched at most once |
| + | The preceding term will be matched one or more times |
| {n} | The preceding term is matched exactly n times |
| {n, } | The preceding term is matched n or more times |
| {n, m} | The preceding term is matched at least *n* times, but no more than *m* times |

# Regular Expressions

- Example #2: retrieve the date of each Super Bowl
  - Previous expression: [A-Z][a-z][a-z]. [0-9][0-9], [0-9][0-9][0-9][0-9]
  - Alternative: [A-Z][a-z]{2}. [0-9]{2}, [0-9]{4}
  - Note that there are dates that are missing in the above result (wrong pattern)
    - *E.g.*, Feb. 1, 2004

# Regular Expressions

- Example #2: retrieve the date of each Super Bowl
  - Fixing the previous expression
    - Expression: [A-Z][a-z][a-z]. [0-9]{1,2}, [0-9]{4}
      - Explanation: the day of the game can have at least 1 and at most 2 digits

# Regular Expressions

- Example #3:
  - Obtaining all the relevant data from http://www.espn.com/nfl/superbowl/history/winners
  - Cell tags
    - <td> and </td>

| NO. | DATE | SITE | RESULT |
|---|---|---|---|
| I | Jan. 15, 1967 | Los Angeles Memorial Coliseum | Green Bay 35, Kansas City 10 |
| II | Jan. 14, 1968 | Orange Bowl (Miami) | Green Bay 33, Oakland 14 |
| III | Jan. 12, 1969 | Orange Bowl (Miami) | New York Jets 16, Baltimore 7 |
| IV | Jan. 11, 1970 | Tulane Stadium (New Orleans) | Kansas City 23, Minnesota 7 |

```
<td colspan="4">Super Bowl Winners and Results</td>
</tr>
<tr class="colhead">
<td>NO.</td>
<td>DATE</td>
<td>SITE</td>
<td>RESULT</td>
</tr>
<tr class="oddrow">
<td>I</td>
<td>Jan. 15, 1967</td>
<td>Los Angeles Memorial Coliseum</td>
<td>Green Bay 35, Kansas City 10</td>
</tr>
<tr class="evenrow">
<td>II</td>
<td>Jan. 14, 1968</td>
<td>Orange Bowl (Miami)</td>
<td>Green Bay 33, Oakland 14</td>
</tr>
<tr class="oddrow">
<td>III</td>
<td>Jan. 12, 1969</td>
<td>Orange Bowl (Miami)</td>
<td>New York Jets 16, Baltimore 7</td>
</tr>
<tr class="evenrow">
<td>IV</td>
<td>Jan. 11, 1970</td>
<td>Tulane Stadium (New Orleans)</td>
<td>Kansas City 23, Minnesota 7</td>
</tr>
```

# Regular Expressions

- Example #3: (last class)
  - Obtaining all the relevant data from http://www.espn.com/nfl/superbowl/history/winners
    - Pattern: <td>.+?</td>
      - Rationale: Find a pattern that begins with <td>, ends with </td>, and has one or more characters in between
        - . = any character
        - + = the preceding item will be matched one or more times
        - ? = the preceding item will be matched at most once
          - This is to avoid “greediness,” i.e., the pattern stops as soon as it finds </td>

# Regular Expressions

- Understanding the role of '?'
  - The * and + qualifiers are all greedy
    - They match as much text as possible
    - Adding ? after the qualifier makes it perform the match in non-greedy or minimal fashion
  - Example
    - Consider the string <a> b c>
      - The pattern <.+> returns the whole string <a> b c>
      - The pattern <.+?> returns the string <a>

# Regular Expressions

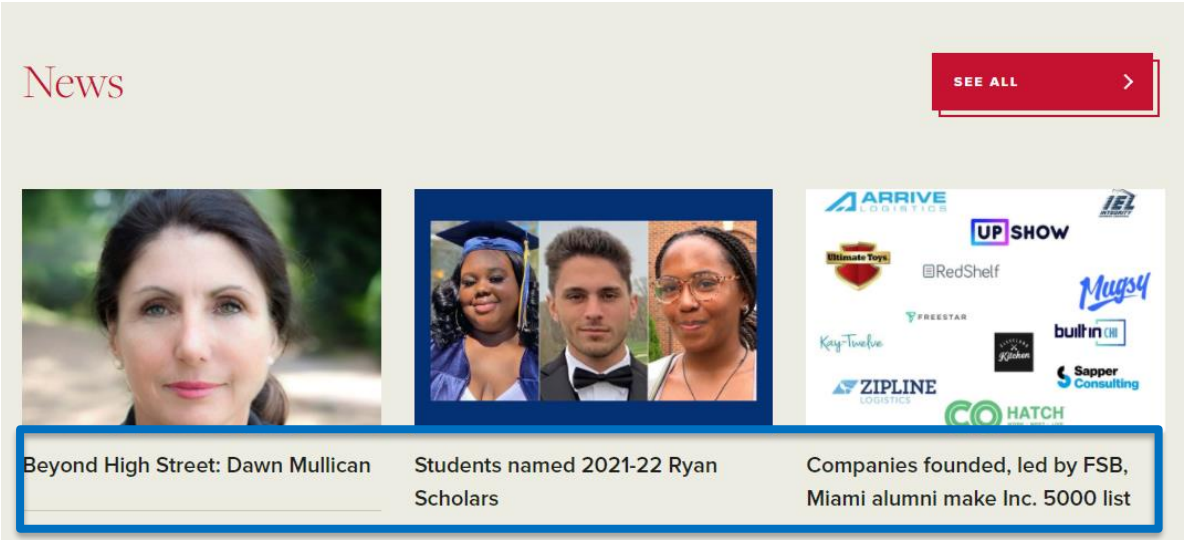
- Escape characters
  - How does one search for a pattern containing a dot, a parenthesis, a bracket, *etc*.?
    - For example, the character . cannot be used because it means “all characters”
    - We have to use “escaping”
      - Escaping is done with a single backslash \ in regular expressions in Python
  - Example: look at the difference between the codes below

```
hits = re.finditer(".", html_code)
hits = re.finditer("\.", html_code)
```

# Case Study: FSB Website

# Regular Expressions

- Obtaining Farmer School news
  - Headlines only




- Open the source code of http://miamioh.edu/fsb/

# Regular Expressions

- Step #1: find an appropriate regular expression
  - How would you retrieve the headline "*Beyond High Street: Dawn Mullican*"

```
                                </div>
        <article class="cards-news__card">
                                                                    <a class="cards-news__card-link" href="https://miamioh.edu/fsb/news/2021/09/Beyond-High-Street-Dawn-Mullican.html" target="_parent">
                        <div class="cards-news__image-container">
    <img class="cards-news__image" alt="Dawn Mullican head shot" src="https://miamioh.edu/fsb/_files/images/news/2021/09/Dawn-Mullican-800.jpg

</div>
<div class="cards-news__card-inner">
                                <p class="cards-news__headline">Beyond High Street: Dawn Mullican</p>
</div>
                        </a>
                                    </article>
        <article class="cards-news__card">
                                                                    <a class="cards-news__card-link" href="https://miamioh.edu/fsb/news/2021/09/Students-named-2021-22-Ryan-Scholars.html" target="_parent">
                        <div class="cards-news__image-container">
    <img class="cards-news__image" alt="Riley Acton head shot" src="https://miamioh.edu/fsb/_files/images/news/2021/09/Ryan-Scholars-2021-22-800.jpg

</div>
```

# Regular Expressions

- Obtaining headlines from FSB website
  - Let's use the pattern "**cards-news__headline.+</p>"**, but first let's remove unnecessary characters from the HTML page

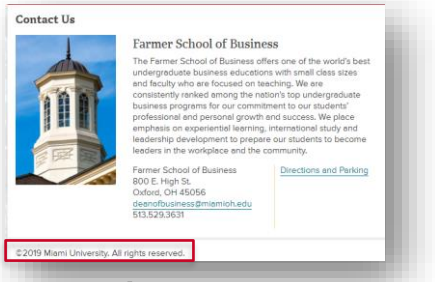
```
r = requests.get("https://miamioh.edu/fsb/")
html_code = r.text
hits = re.finditer("cards-news__headline.+?</p>", html_code)
headlines = [m.group(0) for m in hits]
```

# Regular Expressions

- Obtaining headlines from FSB  website
  - Let's look inside the headlines variable
    - The data of interest is there alongside lots of garbage (undesirable HTML tags)
    - Nongraded homework: clean our data with regular expressions

# Web Scraping

- Can one scrap all the data on the web? No!!!
  - Legal issues
    - Copyright
  - Website owners can:
    - Block an IP address who is requesting too many webpages in a short period of time
    - Use captchas
    - Make small/random changes to HTML code
  - Controlled access to data is often done via API
    - Next lecture



# Nongraded Homework

- Let's clean our headlines using regular expressions
  - Task #1: remove the start of the pattern immediately before the headline

    For example, after this step, the entry: "*cards-news__headline">Beyond High Street: Dawn Mullican</p>*"

    becomes: "*Beyond High Street: Dawn Mullican</p>*"

  - Task #2: remove the end of the pattern

    For example, after this step, the entry: "*Beyond High Street: Dawn Mullican</p>*"

    becomes: "*Beyond High Street: Dawn Mullican*"

  - Hint: Use the function re.sub to replace a pattern with "" (*i.e.*, nothing)
    - Syntax: re.sub(*pattern*, *replacement*, *text_data*)

# Summary

- We learned how to scrap data from the web
  - Using more complex regular expressions
- Further readings
  - https://en.wikipedia.org/wiki/Web_scraping
  - https://en.wikipedia.org/wiki/Regular_expression
  - https://docs.python.org/3/library/re.html
- Next Lecture
  - Collecting data via APIs