
ISA 414 – Managing Big Data

Lecture 7 – Data Collection

Web Scraping (Part I)

Dr. Arthur Carvalho

arthur.carvalho@miamioh.edu



MIAMI UNIVERSITY

Copyright © 2021 Arthur Carvalho

Announcements

- ISA Recruiting Event
 - September 17
 - 1 – 3 pm
 - FSB West Commons

NETWORK WITH COMPANIES ON THE
ISA ADVISORY BOARD BEFORE
CAREER FAIR!

ISA RECRUITING EVENT

September 17, 1-3 pm
FSB West Commons

Deloitte
EY
FIS/Worldpay
GE Aviation
KPMG
Kroger

Nationwide
PwC
TransImpact
VNDLY
West Monroe



SCAN THE QR CODE TO LEARN
MORE ABOUT THE COMPANIES
AND READ JOB DESCRIPTIONS!

OR VISIT:
[HTTP://WWW.FSB.MIAMIOH.EDU/
ISAEVENTS](http://www.fsb.miamioh.edu/isaevents)

Reminders

- We have in-person office hours and classes
 - The virtual option is only in case you have a serious reason
 - *E.g., health, religious, serious accidents, etc.*
 - You should let me know of that at least 24h before class
 - Be prepared to provide documentation
- Please, avoid “Hail Marys”
 - Last minute requests for help, extending deadlines, *etc.*
 - It is very unlikely I will reply to those emails

Lecture Objectives

- Review of Homework #4
- Learn about the CRISP-DM methodology
- Learn how to scrap data from web pages

Lecture Instructions

1. Download the script “*Lecture 7.ipynb*” available on Canvas
2. Open the file “*Lecture 7.ipynb*” with VS Code
3. [Optional] Download the file *webpage.html* available on Canvas

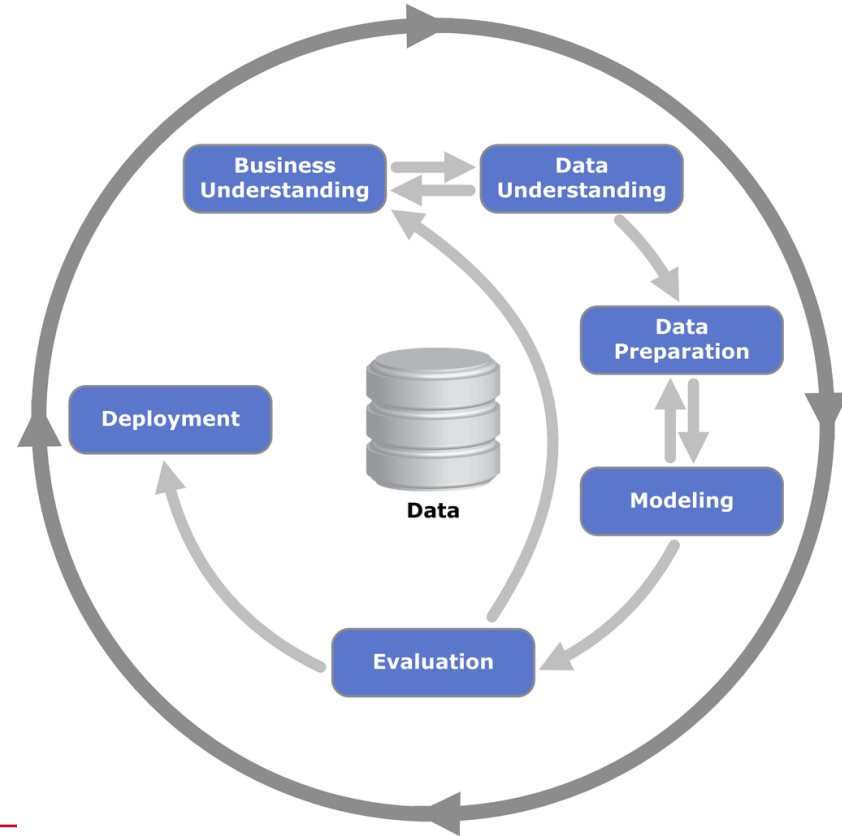
CRISP-DM

- We are reasonably proficient in Python now
 - It is time to start working on more complex data analytics tasks
- Starting and successfully completing an analytics project is no easy endeavor
 - Requires serious project management
 - Standard process model: CRISP-DM methodology
 - Cross Industry Standard Process for Data Mining
 - Series of steps to follow in order to solve an analytics-related business problem

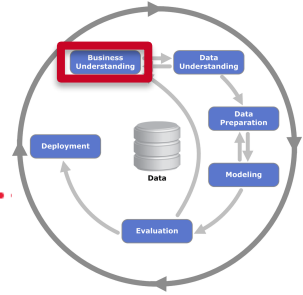
CRISP-DM

➤ Process model: 6 phases

- Goal: extract information from data
- Your final course project will follow the first five phases of this model



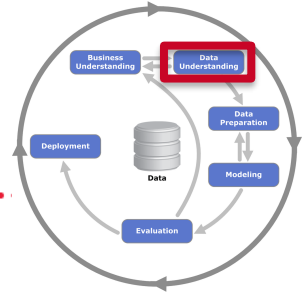
CRISP-DM



➤ Business understanding

- What is exactly the business problem to be solved?
 - Not at all trivial
 - Requires a number of meetings with key stakeholders/sponsors
- Can we formulate a data science solution to solve the business problem?
 - If we can, then is the problem a supervised or unsupervised problem?
 - Is there a clear target variable? (more on this in future lectures)
- What decisions should the solution support?
- Who will sponsor the project?

CRISP-DM



➤ Data understanding (and collection)

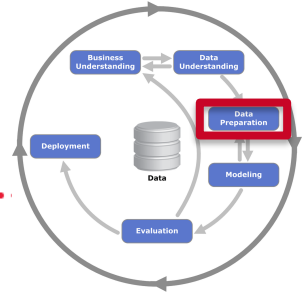
- How does the data generating process look like?
 - Surveys, questionnaires, GPS, social media, experiments,...
- Data come at a price (but it is an asset)
 - Can previously collected data sets or data stored in internal databases (relational, Hadoop, etc.) be used?
 - Otherwise, how will the required data be collected?
 - How costly will that be? It might require complex IT operations

▪ Potential issues

- E.g., privacy (see the ING issue

<https://www.bloomberg.com/news/articles/2014-03-10/ing-plan-to-share-customer-payment-data-spurs-privacy-concerns>)

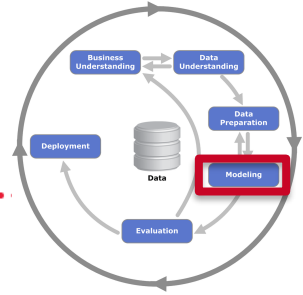
CRISP-DM



➤ Data preparation

- Data sets are rarely in the required form for analysis
 - *E.g.*, unstructured data such as tweets, Facebook posts
- Missing values, wrong values, mixed data types, transformations
- Merging data from different sources

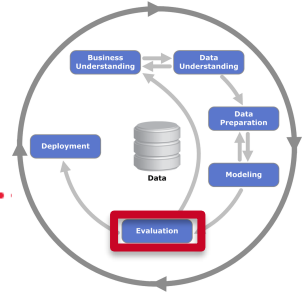
CRISP-DM



Modeling

- Extract information using various techniques
 - *E.g.*, face recognition, voice analytics, sentiment analysis, ...
- Often, the most exciting and easiest part of the data-mining process
- The most appropriate technique to be used depends on a series of factors
 - Interpretability, predictive power, speed of learning, speed of application, ...
 - *E.g.*, Netflix paid one million dollars for a model that is not being used (<http://www.forbes.com/sites/ryanholiday/2012/04/16/what-the-failed-1m-netflix-prize-tells-us-about-business-advice/#617cb97757c7>)

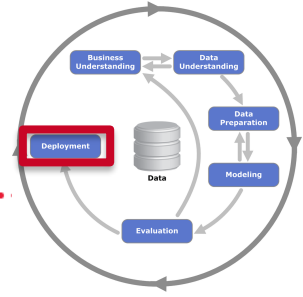
CRISP-DM



➤ Evaluation

- Internal metrics, domain experts, comparison against baselines
- Acceptance of the solution may be subject to socio-cultural factors
- Feedback to business problem understanding

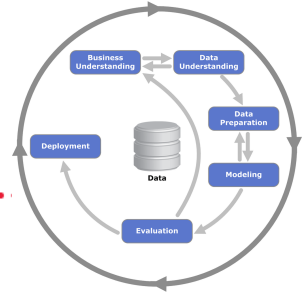
CRISP-DM



➤ Deployment

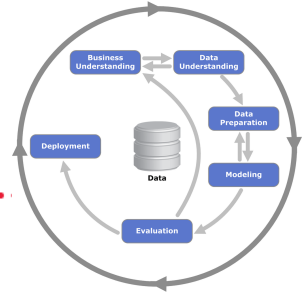
- Put things to use
 - Model will be part of something bigger, *e.g.*, a web page, software, decision support systems
- Evaluation of the acceptance and usage
- Feedback to subsequent business problems
- Reports and presentations

CRISP-DM



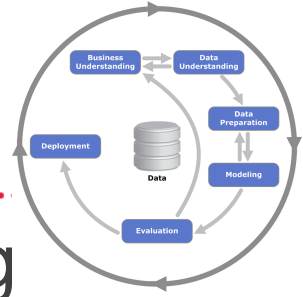
- Example: Butler company (Lectures 3 – 5)
 - Business problem: estimate service demand
 - Data understanding: logs from in-house web server
 - Data preprocessing
 - Extract IP addresses from logs
 - Find latitude/longitude from IP addresses
 - Modeling: simple descriptive analysis (frequencies)
 - Evaluation: solution is not satisfactory
 - Strong assumption (service demand = web access)

CRISP-DM



- CRISP-DM can be, and often is, used in conjunction with other project management frameworks (e.g., agile methodologies)
 - SCRUM
 - Kanban
 - Extreme programming
- Again, you will experience CRISP-DM during the course project

CRISP-DM



- In the next five lectures, we will be focusing on the data understanding (collection) step
- This is followed by five lectures on data modeling/evaluation (supervised learning) and data preparation (text mining)
- Focus on unstructured data

Data Collection

➤ Where do data come from?

- Internal data

- Relational and NoSQL databases
- Spreadsheets/pivot tables
- Files such as pdfs, slides, doc files
- Logs, e.g., web logs, transaction logs

- External data

- Data freely available on the web
- Private data accessed through APIs, often for a price
- Questionnaires, opinion polls, surveys

Data Collection

- This first lecture is about collecting data “freely” available on the web
 - Textual data embedded in HTML code
 - Other data types/formats (e.g., pdf files)
- Before going any further, let’s review important definitions related to HTML
 - HTML (HyperText Markup Language) is a markup language for formatting the content of web pages
 - Main characteristic: links texts via hyperlinks, which creates a *web*
 - Often used in conjunction with other markup and programming languages such as CSS, JavaScript, and PHP

HTML

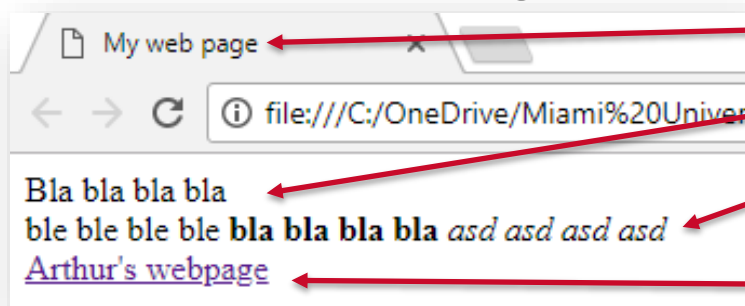
- HTML uses "markups" (tags) to annotate text, images, and other content for display in a Web browser
 - *E.g.*, <head>, <title>, <body>, <header>, <section>, <p>, <div>, , , and many others
- Let's play with HTML
 - Open the file *webpage.html*

Web Scraping

- Let's see the source code of *webpage.html*
 - For Chrome browsers, right click on the page and select “View page source” (or press CTRL + U)
 - For Internet Explorer/MS Edge, press CTRL + U
 - For other browsers see <http://www.computerhope.com/issues/ch000746.htm>

HTML

➤ Understanding the code



<title> My web page </title>

Bla bla bla bla

ble ble ble ble bla bla bla bla <i> asd asd asd
asd</i>

Arthur's webpage

➤ So, why do we care about HTML?

- Web pages sometimes contain valuable textual data (e.g., tweets)
- Web scraping
 - We will import HTML files into Python and preprocess the HTML code via string manipulations to obtain the desired data

Case Study

- 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 a predictive model
 - You suggest the number of NFL titles each team won before might be a relevant feature
 - Good news: this info is freely available online

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 Scrapping

➤ First approach: manual data collection

- Just copy the data from the website and paste it on a spreadsheet
- Works well for our problem
- Can be tedious and very time-consuming for massive data sets
 - You will see this in Assignment 2

Web Scraping

➤ Second approach: automated data collection

- We will write scripts that automatically collect data for us
- Works well when the layout of the web pages do not change often
- Common approach:
 1. Look for relevant HTML tags/markups
 2. Obtain the data between the open markups (< >) and the close markups (</ >)

Web Scrapping

- Let's begin
 - Install required package
 - Go to the Terminal and run `pip install requests`

Web Scrapping

- Let's load the web page into Python
 - Function from the `requests` module: `get`
 - More on this function during the API classes

```
import requests
```

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

```
html_code = r.text
```

- The whole HTML code is now inside Python
 - One string variable `r.text`

Web Scrapping

- Let's look at the HTML code of the ESPN webpage
- Table tags
 - Row tags `<tr class="...">` and `</tr>`
 - 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>
```

Web Scrapping

➤ Our approach

1. Find the positions of the pattern `<td>` (open tag)
2. Find the positions of the pattern `</td>` (close tag)
3. Obtain the data between the first and second positions
 - String functions
4. Build a matrix (optional)

Web Scraping

- Finding the positions of the pattern `<td>`
 - Recall the function `finditer`
 - Obtains **all** the positions where a given pattern occurs

```
import re
```

```
matches = re.finditer("<td>", r.text)
```

```
open_tags = [m.start(0) for m in matches]
```

Web Scrapping


- Finding the positions of the pattern `</td>`

```
matches = re.finditer("</td>", r.text)
```

```
close_tags = [m.start(0) for m in matches]
```

- There is one more element in `close_tags` than in `open_tags`
 - The first element in `close_tags` appears first than the first element in `open_tags`
 - Let's remove that element from `close_tags`
 - Pay attention to the index 0

```
close_tags.pop(0)
```



```
83 <td colspan="4">Super Bowl Winners and Results</td>
84 </tr>
85 <tr class="colhead">
86 <td>NO.</td>
87 <td>DATE</td>
88 <td>SITE</td>
89 <td>RESULT</td>
90 </tr>
```

Web Scrapping

➤ Obtaining the data between the first and second positions

- The code below is not the most efficient one (remember: loops are bad in Python), but it is relatively easy to understand
- Try to rewrite the same code using list comprehension

```
web_data = []  
for i in range(len(open_tags)):  
    web_data.append(html_code[open_tags[i] + 4:close_tags[i] ])
```

web_data

- Why `close_tags[i]` instead of `close_tags[i] - 1`?
- Why `open_tags[i] + 4` instead of `open_tags[i]`?

Web Scraping

➤ Building a data frame

```
import pandas as pd
```

```
rows = []
```

```
#creating a list of lists, each one having 4 elements
```

```
for i in range(4, len(web_data), 4):
```

```
    rows.append(web_data[i:i+4])
```

```
#creating a data frame
```

```
df = pd.DataFrame(rows, columns = web_data[0:4])
```

```
df
```

Web Scraping

- There are easier ways of scraping data from the web with language-dependent solutions
 - E.g., using Python modules such as [beautiful soap](#)
 - Not always 100% accurate
- We can write more effective code by using more complex regular expressions
 - Example: find all patterns `<td>something</td>`
 - We will learn how to do so in the next lecture

Web Scrapping

- Web scrapping is not only about HTML
 - One can scrape data from any files on the web
 - PDF, Word/text, Excel/spreadsheet, ...

Nongraded Homework

- Can you spot anything wrong with the resulting `web_data` matrix?
 - Look at the last 13 rows
- Can you further preprocess the data to fix that problem?
 - Hint: for the problematic rows only, you might consider finding what is in between `>` and ``

Summary

- We learned how to scrap data from the web
 - Finding appropriate patterns (e.g., HTML tags)
 - Retrieving content based on patterns (e.g., inside open and close tags)
- Next lecture: web scraping part II (tough lecture)
 - Scraping web data via complex regular expressions

Copyright 2021 Arthur Carvalho. All rights reserved. This material may not be published, broadcast, rewritten, or redistributed without explicit written consent.