SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Data handling

PROF. D. HERREMANS

50.038 Computational data science

Hacking skills required

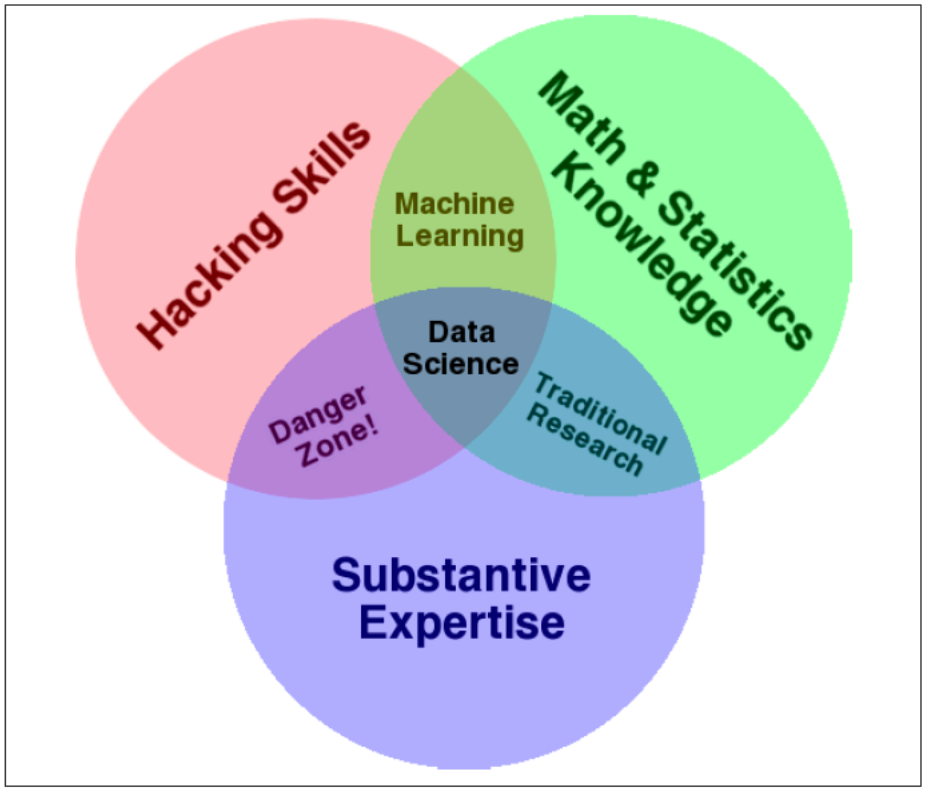

Figure 1-1. Drew Conway's Venn diagram of data science

Data handling - overview

Have you ever tried to open a large file with Excel?

→Need for tools

- Unix command line tools
- Web parsing in Python:
  - Writing to csv
  - Storing in SQL database

Unix text handling

Basic commands

- Download our data:

```
wget http://dorienherremans.com/sites/default/files/parsing/data.csv
```

- Reading text:
  - `vi / cat`
    - `vi manual: https://www.ccsf.edu/Pub/Fac/vi.html`
    - `cat filename`
- View large text and pipe to only see last 5:
  - `cat filename | tail -5`
  - `tail -n 5 data.csv`
- View top 5 lines of large file:
  - `head -n 5 data.csv`

Counting words

- Count words, lines and characters in a file
  - `wc data.csv`
- Count number of lines in a file
  - `wc -l data.csv`

Grep

- **Grep** allows you to search through plain text files (even using regular expressions). Useful when searching through log files for a particular event.
  - parameters: -i (ignore case), -r (recursively search directories), -B N (N lines before), -A N (N lines after).

```
grep –i –B 1 –A 1 'tangible assets’ data.csv
```

  - Sample output:

```
# 17:25||2-4|Darius Theus Turnover.
# 17:25|Terrell Vinson tangible assets.|2-4|
# 17:18|Chaz Williams made Layup.  Assisted by X
```

Sed

- Sed is similar to grep and awk in many ways, however more used for find and replace magic on a very large file.

```
sed -e 's/Block/Rejection/g' data.csv > newdata.csv
```

Replaces all instances of the word 'Block' in data.csv with 'Rejection’ and sends the results to a file called newdata.csv.

Sorting

- head -n 5 data.csv | sort

  Sorts the text file on the first character of each line

- head -n 5 data.csv | sort -f -t',' -**k2**

  Sorts on the second columns (-k2), using ',' as a column separator, case insensitive (-f)

Sorting and removing duplicate rows

- `uniq` command options:
  - -c : shows a count (occurrence) before each line
  - -d : shows only duplicate lines
  - -u : shows only unique lines

- sort data.csv | **uniq -c** | **sort -nr**
  => shows sorted list of all lines with their count
- sort data.csv | uniq –d
  => shows sorted list of all duplicate lines

Find and replace

Imagine you have a 4.4GB csv file. It has over 14 million records and 60 columns.

All you need from this file, is the sum of all values in one particular column.

How do you do that?

Find and replace

Imagine you have a 4.4GB csv file. It has over 14 million records and 60 columns. All you need from this file is the sum of all values in one particular column.

```
cat data.csv | awk -F "," '{ sum += $4 } END { printf "%.2f\n", sum }'
```

=> Cat sends the text to awk using the pipe symbol (|). Awk then splits columns based on ‘,’, then the 4th column is summarised and printed as a float (.2f)

Exercise

1. Download the data.csv file
2. How many words are there in the file?
3. Find the record of funds that cost exactly ‘8898’ and print out two lines below as well.
4. Replace all instances of , by ;
5. Show the first 7 lines of the file
6. Count the number of unique lines in the file

Scraping

IN PYTHON

Scraping with Python

Objective: Scraping a website or data files, parsing them and and save in a csv file (or database).

- Scraping: automatically extracting data.

- Parsing: processing the extracted data in a format you can easily make sense of.

- Libraries: Beautiful Soup, lxml, jSoup (java),…

Best practices

- Check a website's **Terms and Conditions** before scraping. Usually, the data you scrape cannot be used for commercial purposes.
- Don’t put too much stress on the website. (i.e. no 10,000 requests a minute), this may break the website. Make it **behave human-like**, e.g. one request for one webpage per second is good practice.
- **Revisit the site at times** to check if the layout has changed → adapt code.
- NOTE: often parsing is not necessary as companies are offering APIs to extract data:
  - https://iextrading.com/developer/. -> stock market data API
  - https://developer.yahoo.com/boss/search/boss_api_guide/ -> Yahoo news API

Html

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h1> First Scraping </h1>
        <p> Hello World </p>
    <body>
</html>
```

Other useful tags include <a> for hyperlinks, <table> for tables, <tr> for table rows, and <td> for table columns.

Tags often have id and class as attributes. (id is unique, and class can occur multiple times as it specifies the style)

Id & class

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h1> First Scraping </h1>
        <p id='first_paragraph' class='plain_text'> Hello
          World </p>
    <body>
</html>
```

Each of the tags can have id and class labels. These are typically used to format the webpage (layout). We can use them to pinpoint the text we want to parse!

Exercise

- Switch to iPython Notebook (using Python3):
  - Execute the code using iPython Notebook
  - or an IDE such as PyCharm…

Beautiful Soup

- Let's parse the site pythonprogramming.net/parsememcparseface/. The code below will extract the website's source code and store it in the soup variable, a BeautifulSoup object. You can print this out to make sure that you were successful.

Loading the soup object

```
1  from bs4 import BeautifulSoup
2  import urllib.request
3  from requests import get
4
5  url = 'https://pythonprogramming.net/parsememcparseface/'
6  response = get(url)
7  soup = BeautifulSoup(response.text, 'html.parser')
8
9  print(soup)
10
```

salmon: #FF476F

Extracting basic tags

We can now extract all sorts of html tags from our soup variable. For instance:

```
# title of the page (between <title> tags, this prints the full
# tag, including the tags itself)
print(soup.title)

# get attributes:  (prints the name of the tag)
print(soup.title.name)

# get values:  (only prints the value between the <title> tags)
print(soup.title.string)

# beginning navigation:  (prints the parent tag in
# which title is contained (here it is in the header <head>))
print(soup.title.parent.name)

# getting specific values: (prints the first paragraph <p>)
print(soup.p)
```

Extracting basic tags

The last comment soup.p, gives us the first paragraph, there are actually more then one paragraphs! To find them all:

```
1 print(soup.find_all('p'))
```

Alternatively, we can iterate through all paragraphs:

```
1 for paragraph in soup.find_all('p'):
2     print(str(paragraph.text))
3
```

Finding something specific

What other html tags are there? Link! Let's get all links...'

```
1 for url in soup.find_all('a'):
2     print(url.get('href'))
```

,

Let's print the paragraph that contains the introduction (class = introduction)

```
1 name_box = soup.find('p', attrs={'class': 'introduction'})
2 print(name_box.text)
```

Find the tags that contain the text yesnojs:

```
1 thistext = soup.find_all(id="yesnojs")
2 print(thistext[0].text)
```

Find Today's internet points of Python in the table and append to a csv file

```
table = soup.find('table')
table_rows = table.find_all('tr')  #note: <tr> tags are
# used for table rows, <td> for table columns within each row

pythonrow = table_rows[1] #in the html code we can see
# there is an empty row above the python row
cells_in_pythonrow = pythonrow.find_all('td') #finds
# all cells in the current row

ip_python = cells_in_pythonrow[1] #second column of
# the python row contains our value

print(ip_python.text)
```

write to csv:

```
import csv
from datetime import datetime

# open a csv file with append (a instead of w), so old data will not be erased
with open('index.csv', 'a') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow([datetime.now(), 'Python', ip_python.text])

```

Check our work:

```
with open('index.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for row in csv_reader:
        print(row)

```