Data Analysis

Kelly Rivers and Stephanie Rosenthal

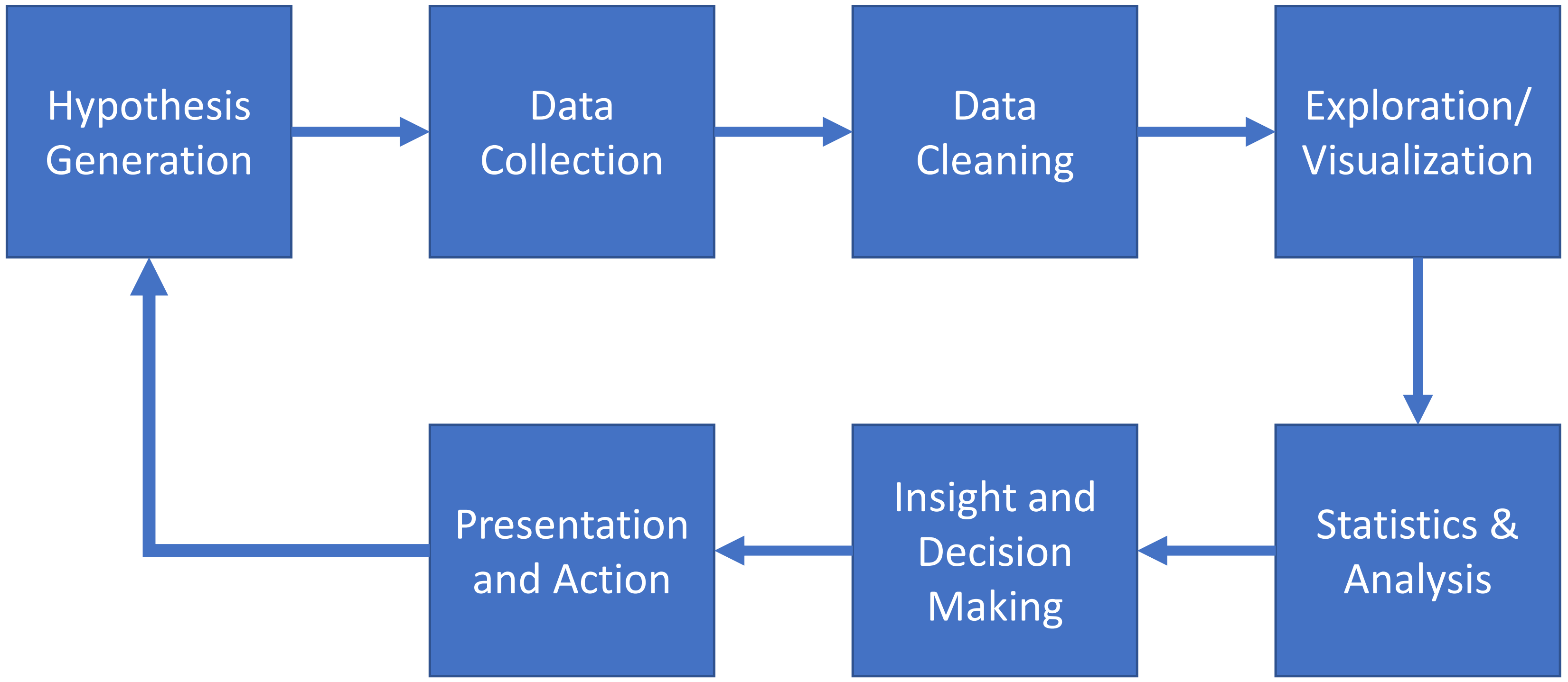
15-110 Fall 2019

Data Science vs Data Analytics

Data science is the application of computational and statistical techniques to address or gain insight into some problem in the real world that can be captured by data

Data analysis is the application of the same techniques to gain insight only into the data collected

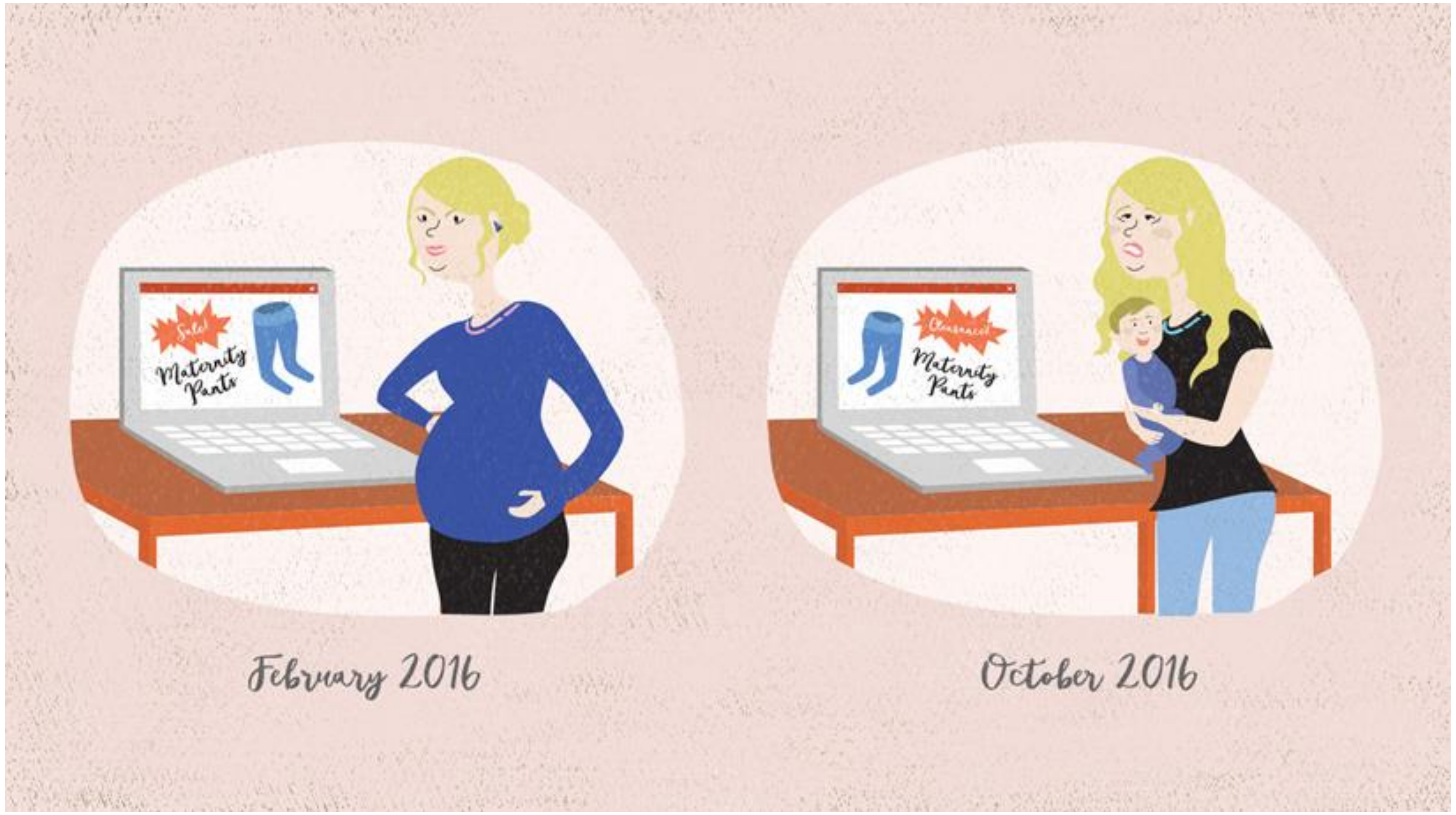
Data Science/Analysis Process



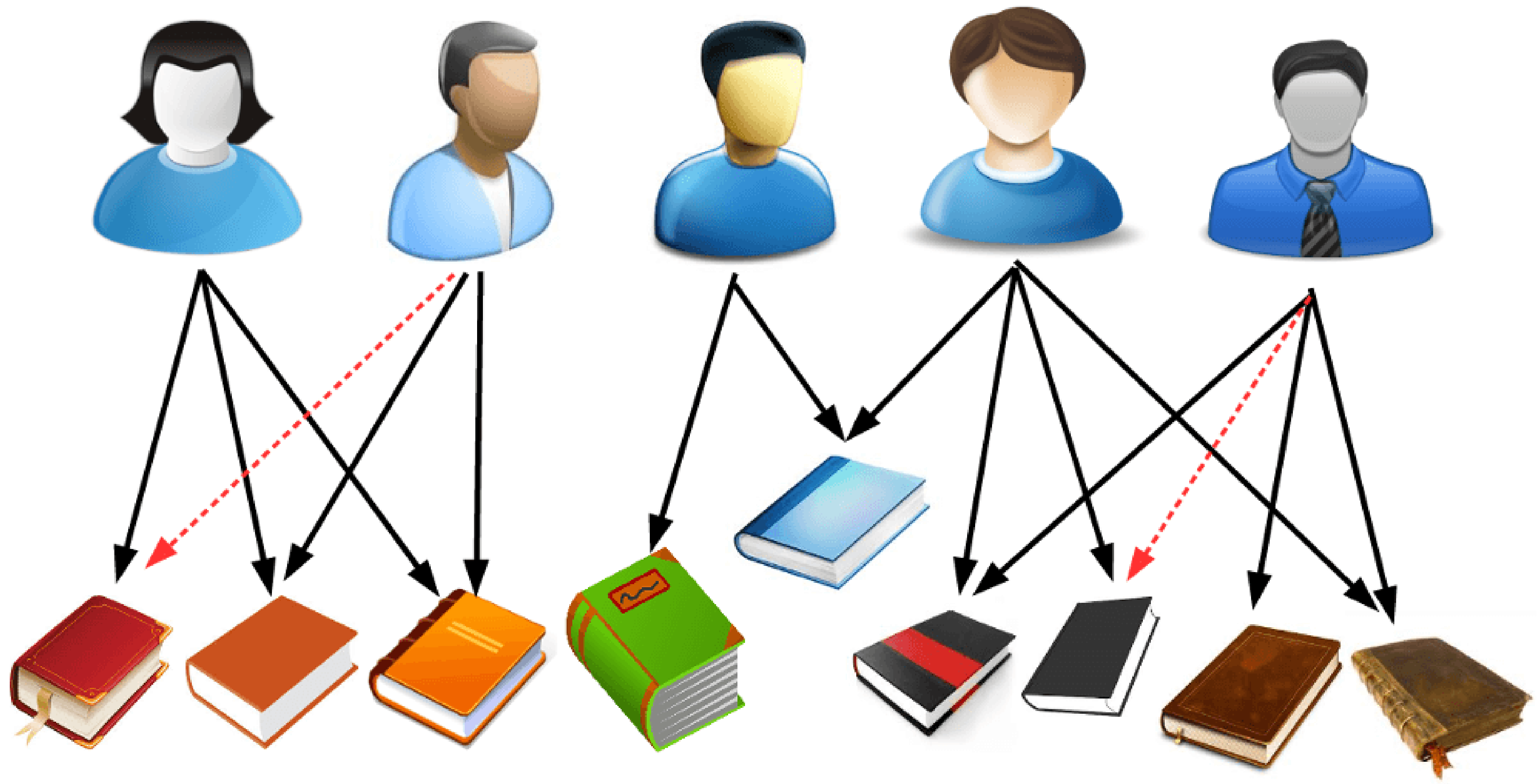
Applications

Web Search

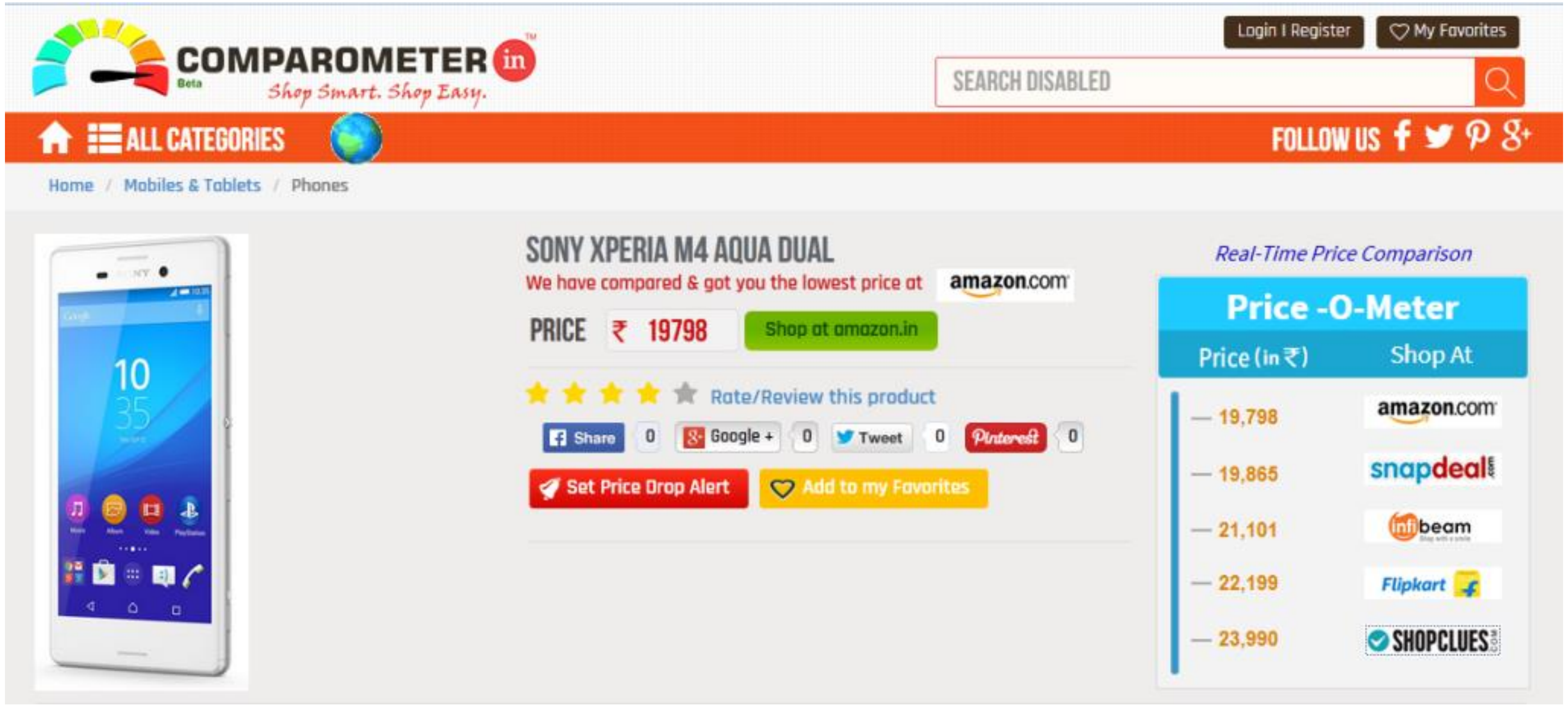
Targeted Advertising



Recommender Systems



Price Comparison Websites



SONY XPERIA M4 AQUA DUAL

We have compared & got you the lowest price at amazon.com

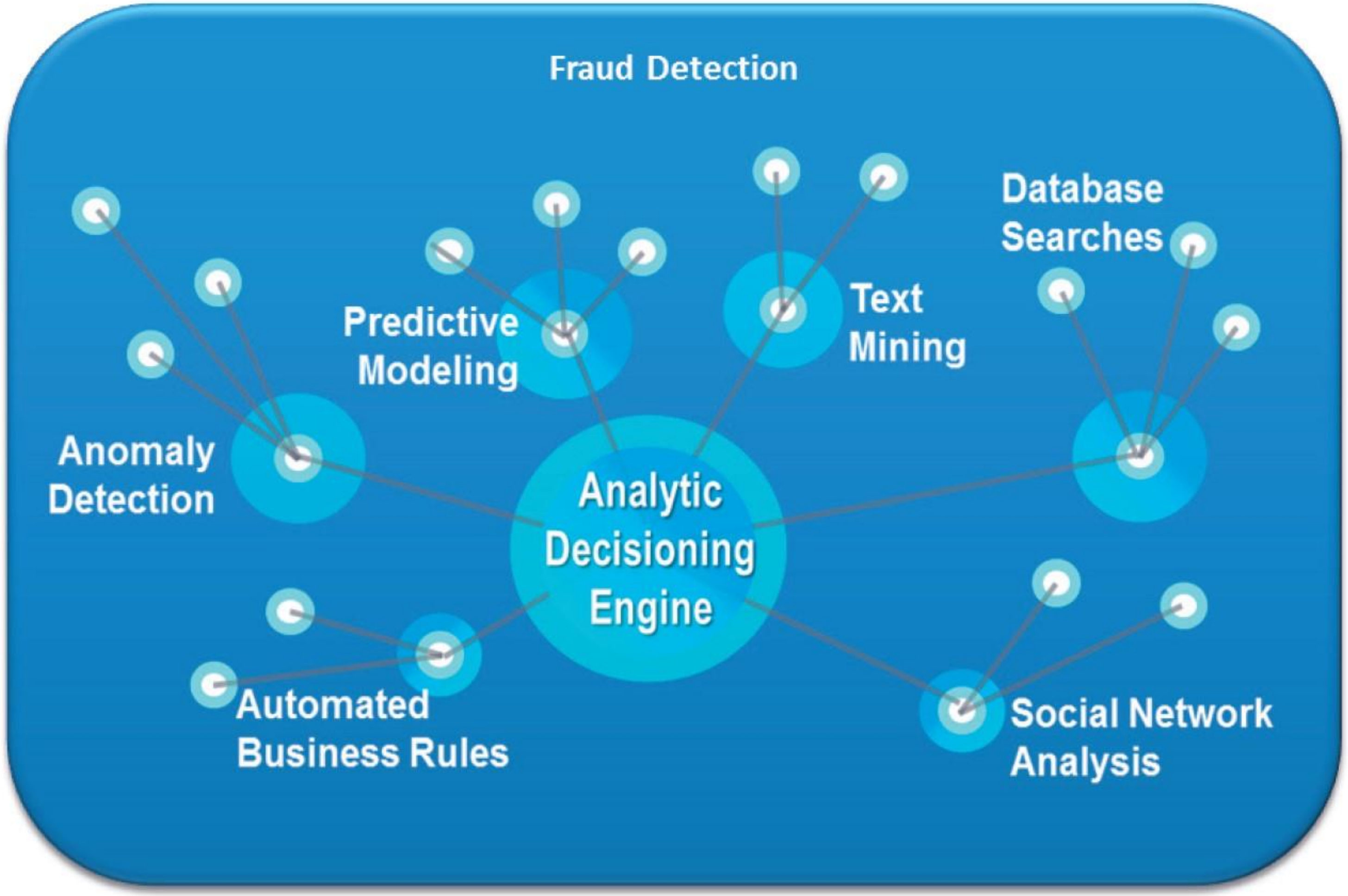
PRICE ₹ 19798 Shop at amazon.in

Real-Time Price Comparison

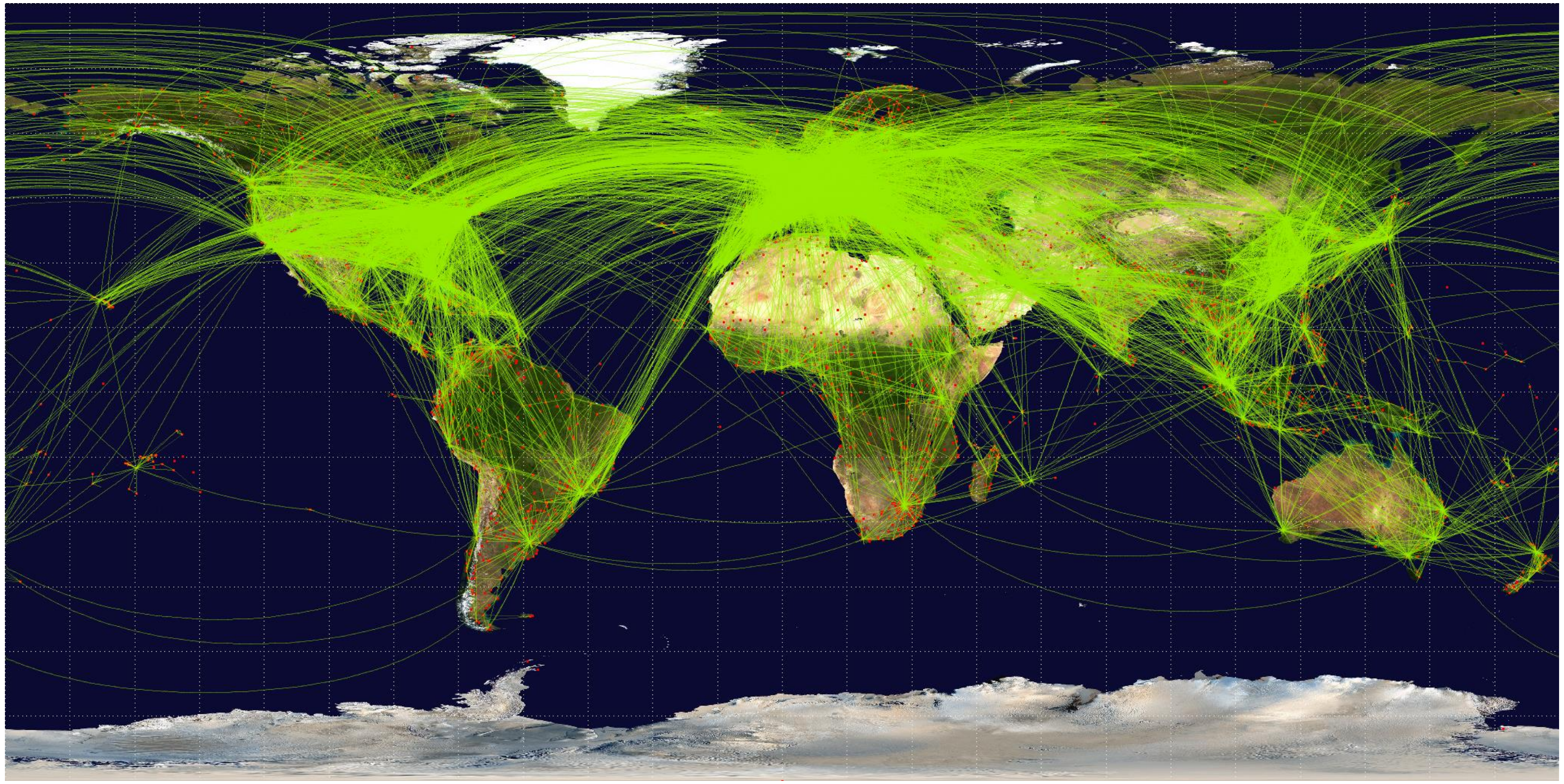
Price -O-Meter

Price (in ₹)	Shop At
19,798	amazon.com
19,865	snapdeal.com
21,101	infibeam
22,199	Flipkart
23,990	SHOPCLUES.com

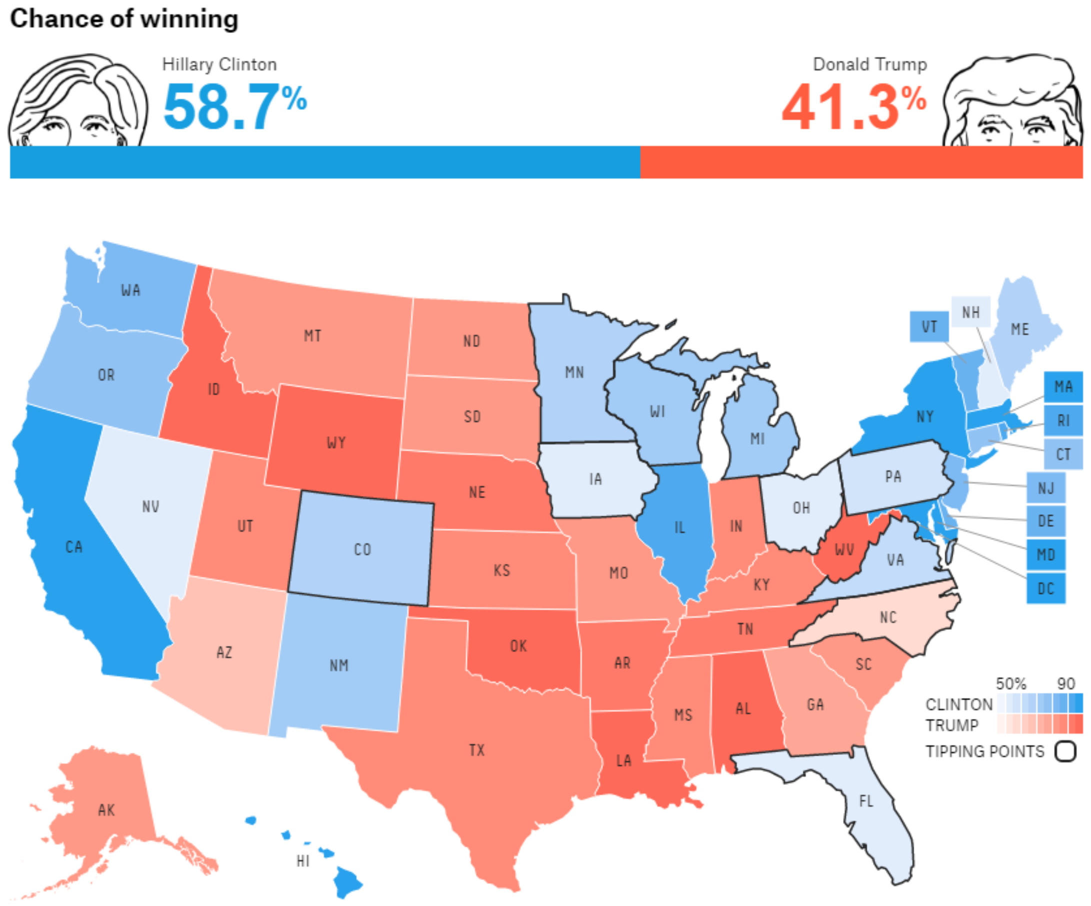
Fraud Detection



Airplane Routing



Politics


Chance of winning
Hillary Clinton
58.7%
Donald Trump
41.3%
WA
OR
CA
NV
ID
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
FL
SC
NC
VA
WV
PA
NY
VT
NH
ME
MA
RI
CT
NJ
DE
MD
DC
AK
HI
50%
90
CLINTON
TRUMP
TIPPING POINTS

Logistics

- What to sell
- When to sell it
- How to move it around

- Walmart
- UPS
- IKEA

Data

Data Formats

Data comes in many different formats

- HTML
- XML
- JSON
- CSV
- Raw text
- …

The main types of data you'll encounter if you do a lot of analysis

- CSV are comma separated files
- JSON are javascript object notation files
- HTML/XML are markup language

Open to Read

```
openfile = open("path/to/file.txt","r")
```

- Where is the file located?
- What am I doing with the file?
- What format is it in?

Comma-separated (CSV) files

Not always comma separated, but have to be separated by something (tab, etc) – these are called delimiters

```
data.csv
Semester,Course,Lecture,LastName,FirstName,Email
F19,15110,01,Rivers,Kelly,krivers
F19,15110,02,Rosenthal,Stephanie,srosenth
```

```
import csv
csvfile = open("data.csv", "r")
data = csv.reader(csvfile, delimiter=',', quotechar='"')
my2dList = []
for row in data: #each row in data is a list of the delimited values
    my2dlist.append(row)
```

JSON files

- Can represent the following data types:
  - Numbers (floating point)
  - Booleans
  - Strings
  - Lists
  - Dictionaries
  - Combinations of those

```
[
        {
        "key1": [value1, [value2, value3]]
        "key2": value4
        }
        {
        "key1": [value5,value6,value7]
        "key2": {"k":1, "m":2, "n":3}
        }
]
```

JSON files

- To read a json file properly, use the json module

```
import json

#INPUT
f = open("data.json","r")
content = f.read()
data = json.loads(content)
#OR data = json.load(f) # load json from file

#OUTPUT
w = open("output.json","w")
s = json.dumps(obj) # return json string, can write it to w
#OR json.dump(obj, w) # write json to file
```

Markup Language

- The main format for the web
- XML seems to be losing popularity to JSON for use in APIs / file formats
- XML files contain hiearchical content delineated by tags

```
<tag attribute="value">
     <subtag> Some content for the subtag
     </subtag>
     <openclosetag attribute="value2"/>
</tag>
```

- HTML is syntactically like XML but horrible (e.g., open tags are not always closed)

Markup Language

Use BeautifulSoup as the package to parse HTML/XML in this course

You'll need to install it to use it- we'll go over that later in the unit

```
#get all the links in a website
from bs4 import BeautifulSoup

f = open("data.html","r")
content = f.read()
root = BeautifulSoup(content)
root.find("section",id="schedule").find("table").find("tbody").findAll("a")
```

Raw Text

- What are you looking for?
- Can you find it easily?

read() vs readline()

- The variable returned by open() is a reference/pointer to the data in the file
- Once you have that pointer, you can read
  - all = openfile.read(): the string all contains the entire contents of the file
  - some = openfile.read(5): the string some contains the next 5 characters of the file
  - line = openfile.readline(): the string line contains characters up to and including the next "\n" character

Printing the File – good for debugging

```
openfile = open("path/to/file.txt","r")
all = openfile.read()
print(all)

#OR

openfile = open("path/to/file.txt","r")
for line in openfile:
     <do something to each line>
#can do all.split("\n") and make a for loop to produce the
#same results with .read()
```

Raw Text: What is the pattern in your file?

Parsing Strings

Where is the pattern in the file? Do you know the part of the file you are looking for?

- Do you know where it is in the line?
- Do you know what you are looking for?
- Do you know what comes before or after it?
- Is there a character you know will be there?

This works for any string, even if it is in a file that isn’t raw text

Parsing Strings

Do you know where it is in the line and how long it is?

substring = string[start:stop:step]

```
end = line[5:]
begin = line[:5]
middle = line[5:10]
secondtolast = line[:-1]
backwards = line[::-1]
```

Parsing Strings

Do you know what you are looking for?

index = line.find("cat")

Returns character index if cat exists or -1

```
index = line.find("cat")
if index > -1:
        print(line[index:index+3])
        print(line[index:])
```

Parsing Strings

Do you know what comes before or after it?

```
line = "abcdefghijklmnopqrstuvwxyz"
idx1 = line.find("d")
idx2 = line.find("n")
substring = line[idx1:idx2]
```

Parsing Strings

Is there a character you know will be on either side of the content in question?

parts = line.split("\t")

```
line = "5 \t cat \t s abcd s def s ghi \t dog \t 4 \n"
parts = line.split("\t")
print(parts[2])
subparts = parts[2].split("s")
print(subparts[3])
```

Iterating Through Many Files

You have lots of files, how can you iterate through many files?

Glob searches your files for a particular pattern and returns a list of strings of file locations matching the pattern.

* matches anything

*.txt matches all .txt files

abc matches any file with abc in the middle

```
import glob
files = glob.glob("path/to/files/*.txt")
for file in files:
        openfile = open(file,"r")
        #do something here like "for line in openfile"
```

Saving Data from Python

You know how to write files

```
file = open("path/to/filename","w")
file.write(response.text)
file.close()
```

It's a good idea to save files by their date so it is easy to sort or find when you made it, especially as you are debugging

```
import datetime
file = open("path/to/file"+str(datetime.datetime.now(),"w")
#use glob to match any file that starts with the path part
```

Pickle Files

Python also has its own data format from before JSON existed

You can save your data as pickle (.pyc) files, and when you load them back into Python, they will already be in your data structures (including dictionaries and lists)

```
import pickle
favorite_color = { "lion": "yellow", "kitty": "red" }
pickle.dump( favorite_color, open("save.p", "w") )
reload_color = pickle.load( open("save.p","r") )
```

Data Organization

Your goal is to create a table of consistent information like an excel spreadsheet. If you can get your data into a table format, then you can do things like loop through a column of data, or a row at a time.

Python Pandas

Python Pandas is a framework to help you organize your data and edit it in a table/matrix/2D list format.

http://pandas.pydata.org/pandas-docs/stable/10min.html

To start, you must install pandas

Then, you can import pandas in your python file

The `df` variable is now a dataframe, which is a table.

```
import pandas as pd
df = pd.read_csv("data.csv",delimiter=",", quotechar='"')
```

Pandas without a File

```
import pandas as pd

# dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}

# creating a dataframe from a dictionary
df = pd.DataFrame(dict)

#makes a table with one column name, one column degree, and then score
```

Iterating through rows of a dataframe

```
# iterating over rows using iterrows() function
df = pd.DataFrame(dict)
#index is row index, row is the row itself
for index, row in df.iterrows():
    print(index, row)
    #index into row using column names
```

Data Conversion

- What are some ways that data might be insufficient for your task?
  - Dates hard to parse
  - Upper and lower case characters
  - Integers and floating points, and other conversions
  - New columns with new mathematical computations
  - Change synonyms to match each other
  - Converting words to numbers

Data Conversion

Data Types

- Pandas assumes everything is a float, to make it think a column is an integer

```
df['id'].astype(int)
```

- There are other common types to convert to

```
astype('category')
astype('int32')
astype('int64')
to_datetime(…)
to_timedelta(…)
to_numeric(…)
```

- How would you do this with the original 2D list?

Data Conversion

- Upper and lower case
  - Pandas can do some string handling, same as regular strings in Python

```
df['columnname'].str.lower()
```

- Dates using datetime objects
  - There are many ways that the date could be represented in your data
    - year, month, day
    - date/time
    - unix epoch time (number of seconds since January 1, 1970)
  - Representing it as a datetime data structure allows you to add/subtract dates, convert timezone, etc

```
to_datetime(…)
```

Mathematical Functions

- You can perform mathematical functions on a dataframe

```
import numpy #need to download it first
#can use other math functions by importing math instead
df.apply(numpy.sqrt) # returns new DataFrame
df.apply(numpy.sum, axis=0) #columns
df.apply(numpy.sum, axis=1) #rows
```

Replacing or Splitting Strings

See more https://pandas.pydata.org/pandas-docs/stable/text.html

```
#for each column, it's a string, strip whitespace, make it lowercase,
#replace spaces with underscores
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
```

```
# importing pandas module
import pandas as pd
# reading csv file from url
data = pd.read_csv("data.csv")
# new data frame column Col with split value columns
#split on \t, do it only n=1 time even if more \t in string, return a dataframe
data["Col"]= data["Col"].str.split("\t", n=1, expand = True)
# df display
print(data)
```

You may want to find/replace synonyms

- If you know the synonyms or the replacements, you can use the string replacements
  - Examples:
    - "one"-->1
    - "freshman"-->"first year"
- Some aren’t so easy, so use NLTK (Natural Language Tool Kit)
  - Stemming: want, wants, wanted --> want
  - Synonyms: "dog", "canine", …

Natural Language Tool Kit (NLTK)

- You can download and import nltk to help with text tasks
  - Use with caution!
- Some things NLTK is great for
  - Datasets
  - Natural Language extraction
  - Common text analytics like TF-IDF
- Synonyms and other language nuances are more challenging to code and especially to automate
  - Same with sentiment analysis (deciding if a sentence is happy or sad)