Week 1

Introduction

Python Data Products

Course 1: Basics

Meet your instructors and about this course

Dr. Julian McAuley

Research

- Assistant professor at UC San Diego since 2014
- Research on Machine Learning, Recommender Systems, and predictive models of human behavior (or so-called "computational social science")
- Lab website: https://cseweb.ucsd.edu/~jmcauley/

Academic Teaching and Workforce Development

- I teach undergraduate and graduate courses on Mining Web Data, and Recommender Systems at UCSD, and on current trends in recommender systems and behavioral modeling

Interdisciplinary Research

- I frequently collaborate with industry, and have conducted research and written papers with *Microsoft, Etsy, Adobe, Pinterest,* and *Flipkart,* among others

Research interests...

I also work on: **Question-Answering Systems**

Q: "I want to use this with my iPad air while taking a jacuzzi bath. Will the volume be loud enough over the bath jets?"

"The sound quality is great, especially for the size, and if you place the speaker on a hard surface it acts as a sound board, and the bass really kicks up." → Yes

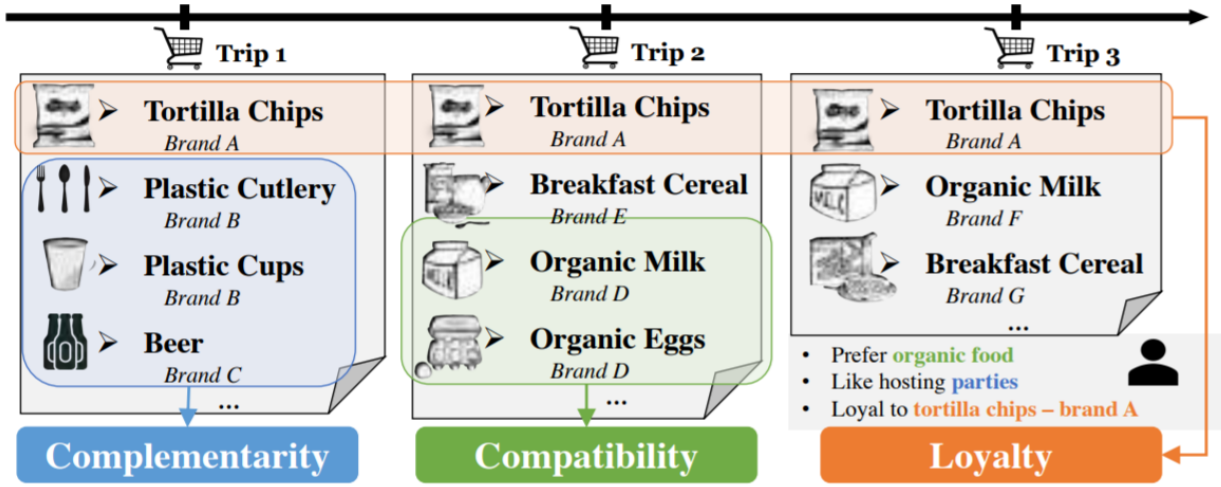
"If you are looking for a water resistant blue tooth speaker you will be very pleased with this product." → Yes

"However if you are looking for something to throw a small party this just doesn't have the sound output." → No

Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems, Mengting Wan, Julian McAuley, **ICDM 2016**
Addressing complex and subjective product-related queries with customer reviews, Julian McAuley, Alex Yang, **WWW 2016**

Research interests...

I also work on: Combining **Economic Models with Recommender Systems**



Modeling consumer preferences and price sensitivities from large-scale grocery shopping transaction logs, Wan, Wang, Goldman, Taddy, Rao, Liu, Lymberopoulos, McAuley, **WWW 2017**

Representing and recommending shopping baskets with complementarity, compatibility, and loyalty, Wan, Wang, Liu, Bennett, McAuley, **CIKM 2018**

Research interests...

I also work on: Systems for **Personalized Fashion Recommendation and Design**

Modeling the visual evolution of fashion trends with one-class collaborative filtering, He, McAuley, **WWW 2016**

Visually-aware fashion recommendation and design with generative image models, Kang, Fang, Wang, McAuley, **ICDM 2017**

Research interests...

I also work on: Systems for **natural language generation**

12 oz. bottle, excited to see a new Victory product around, **A:** Pours a dark brown, much darker than I thought it would be, rich creamy head, with light lace. **S:** Dark cedar/pine nose with some dark bread/pumpernickel. **T:** This ale certainly has a lot of malt, bordering on Barleywine. Molasses, sweet maple with a clear bitter melon/white grapefruit hop flavour. Not a lot of complexity in the hops here for me. Booze is noticable. **M:** Full-bodied, creamy, resinous, nicely done. **D:** A good beer, it isn't exactly what I was expecting. In the end above average, though I found it monotonous at times, hence the 3. A sipper for sure.

Actual review

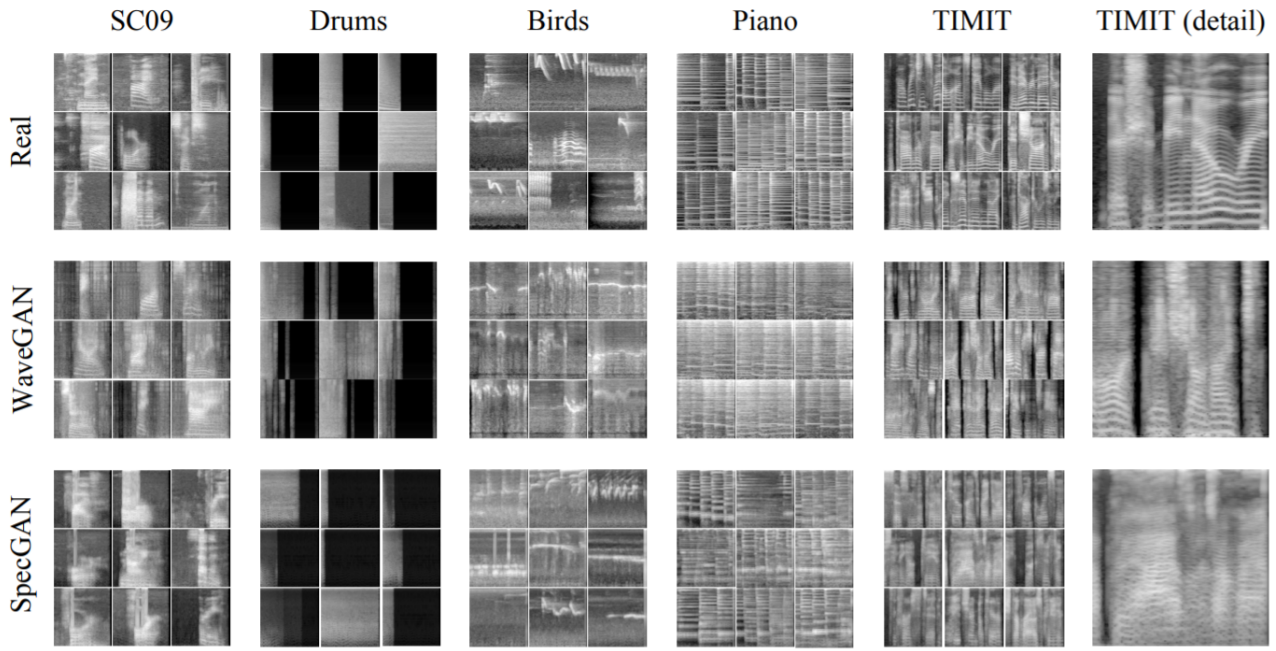
A: Pours a very dark brown with a nice finger of tan head that produces a small bubble and leaves decent lacing on the glass. **S:** Smells like a nut brown ale. It has a slight sweetness and a bit of a woody note and a little cocoa. The nose is rather malty with some chocolate and coffee. The taste is strong but not overwhelmingly sweet. The sweetness is overpowering, but not overwhelming and is a pretty strong bitter finish. **M:** Medium bodied with a slightly thin feel. **D:** A good tasting beer. Not bad.

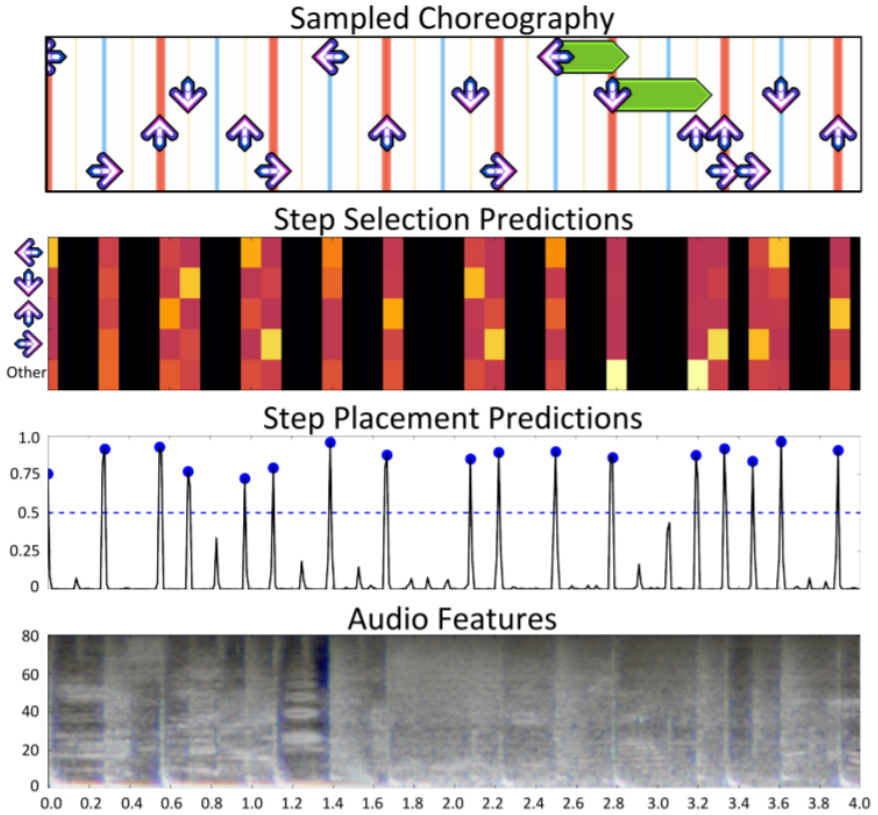
Synthetically generated review

Personalized review generation by expanding phrases and attending on aspect-aware representations, Ni, McAuley, **ACL 2018**
Estimating reactions and recommending products with generative models of reviews, Ni, Lipton, Vikram, McAuley, **IJCNLP 2017**

Research interests...

I also work on: **Music and Audio**





Adversarial audio synthesis, Chris Donahue, Julian McAuley, Miller Puckette, **ICLR 2019**
Dance Dance Convolution, Chris Donahue, Zachary Lipton, Julian McAuley, **ICML 2017**

Dr. Ilkay ALTINTAS

Research, Development and Management

- Chief Data Science Officer, San Diego Supercomputer Center (SDSC) — datascience.sdsc.edu
- Division Director, Cyberinfrastructure Research, Education, and Development @ SDSC — www.sdsc.edu
- Director, Workflows for Data Science (WorDS) Center of Excellence @ SDSC — words.sdsc.edu
- Fellow — Halicioglu Data Science Institute (HDSI) @ UC San Diego — datascience.ucsd.edu

Academic Teaching and Workforce Development

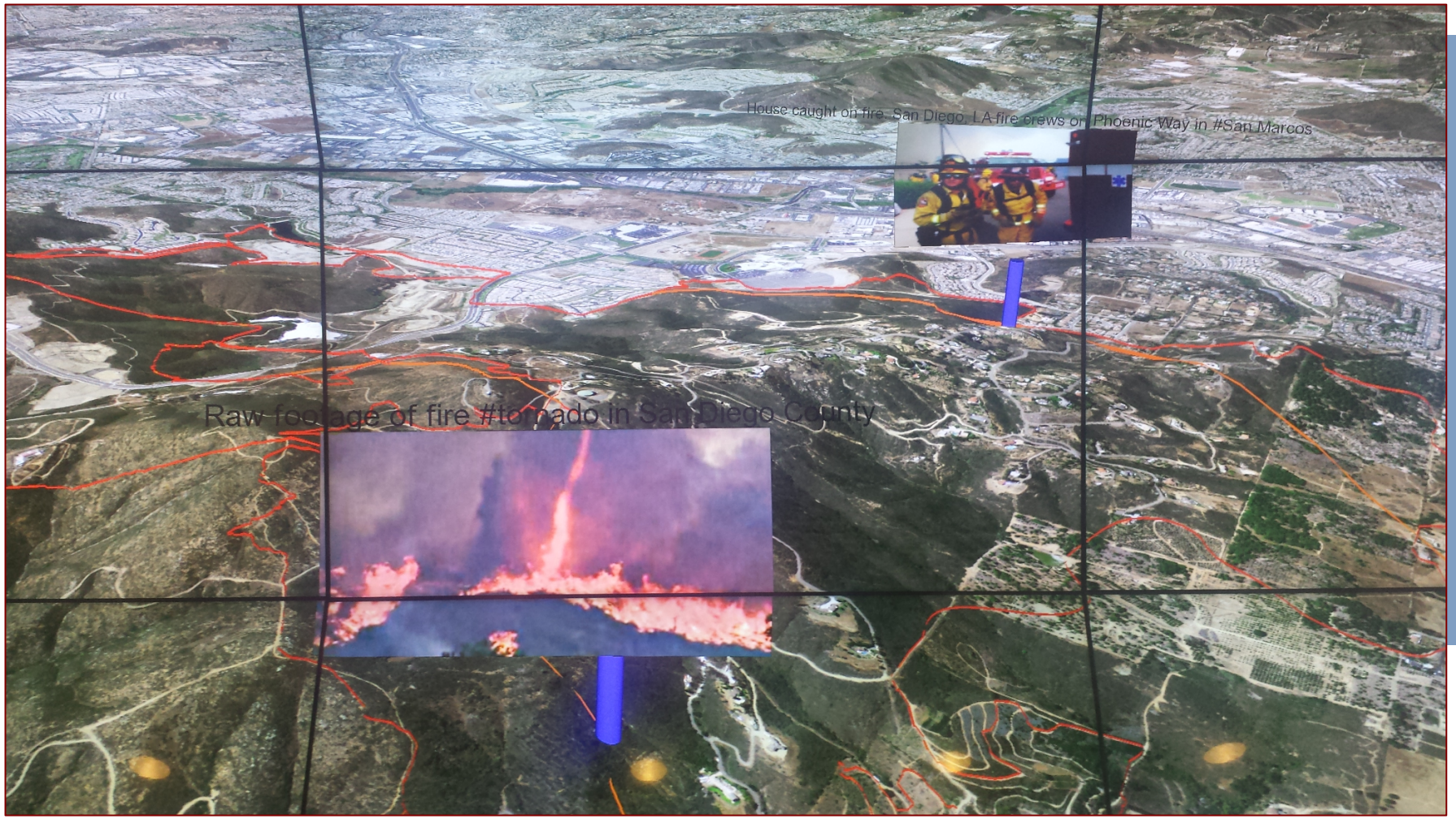
- Faculty Co-Director and Lecturer, UC San Diego Master of Advanced Study Program in Data Science and Engineering
- Lecturer, Computer Science and Engineering at UC San Diego
- Faculty on Coursera UC San Diego Big Data Specialization
- Faculty on edX UC San Diego MicroMasters in Data Science

Interdisciplinary Research

How can I get smart people to collaborate and communicate to analyze data and computing to **generate insight** and **solve a question**?

Building Data Products for Wildfire Prediction and Emergency Response

Wildfire Command Centers of the Future



Welcome!

In this specialization we will...

- Learn how to read and **process** complex, real-world datasets (**Course 1**)
- Learn how to design **predictive pipelines** in order to build predictive models from data (**Course 2**)
- Learn how to **validate and evaluate** our predictions and gain insights from our models (**Course 3**)
- Learn how to **deploy** working systems that implement these models (**Course 4**)

Course overview

Course 1 will cover...

- Background on data products (Week 1)
  - What is the Data-to-Product pipeline?
- Data ingestion and processing (Week 2)
  - Reading data from CSV/TSV and JSON files
- Exploratory data analysis (Week 3)
  - Manipulating semi-structured data such as time and date, and string processing in Python
- Data processing and visualization (Week 4)
  - Data collection & data visualization

Course overview

Course 2 will cover...

- Regression
  - Supervised vs. unsupervised learning, autoregression
- Feature engineering
  - Dealing with temporal and categorical data
- Classification
  - Nearest neighbor and logistic regression
- Libraries and tools for regression and classification

Course overview

Course 3 will cover...

- Regression and classification diagnostics
- Training and validating regressors and classifiers
- Model evaluation

Course overview

Course 4 will cover...

- Recommender Systems
- Our Capstone Project!
- Deployment of data products in practice, and Python libraries for deploying data driven systems

How will you succeed?

- Participating in course activities
  - Lecture videos
  - Discussion prompts
  - Reading materials
- Working through Jupyter Notebooks

We value your feedback!

Python Data Products

Course 1: Basics

Lecture: What is a Data Product?

Learning objectives

In this lecture we will...

- Explain many interpretations of “data product”
- Describe what we focus on as a data product in the Data to Product Specialization
- Illustrate a typical process to go from data to a predictive insight towards an end goal
- Motivate why Python is a good choice for data product development

Data Products

"… a product that facilitates an end goal through the use of data"

-- DJ Patil, Former U.S. Chief Data Scientist

in Data Jujitsu

What are data products?

Data Products are

systems models

that help us to

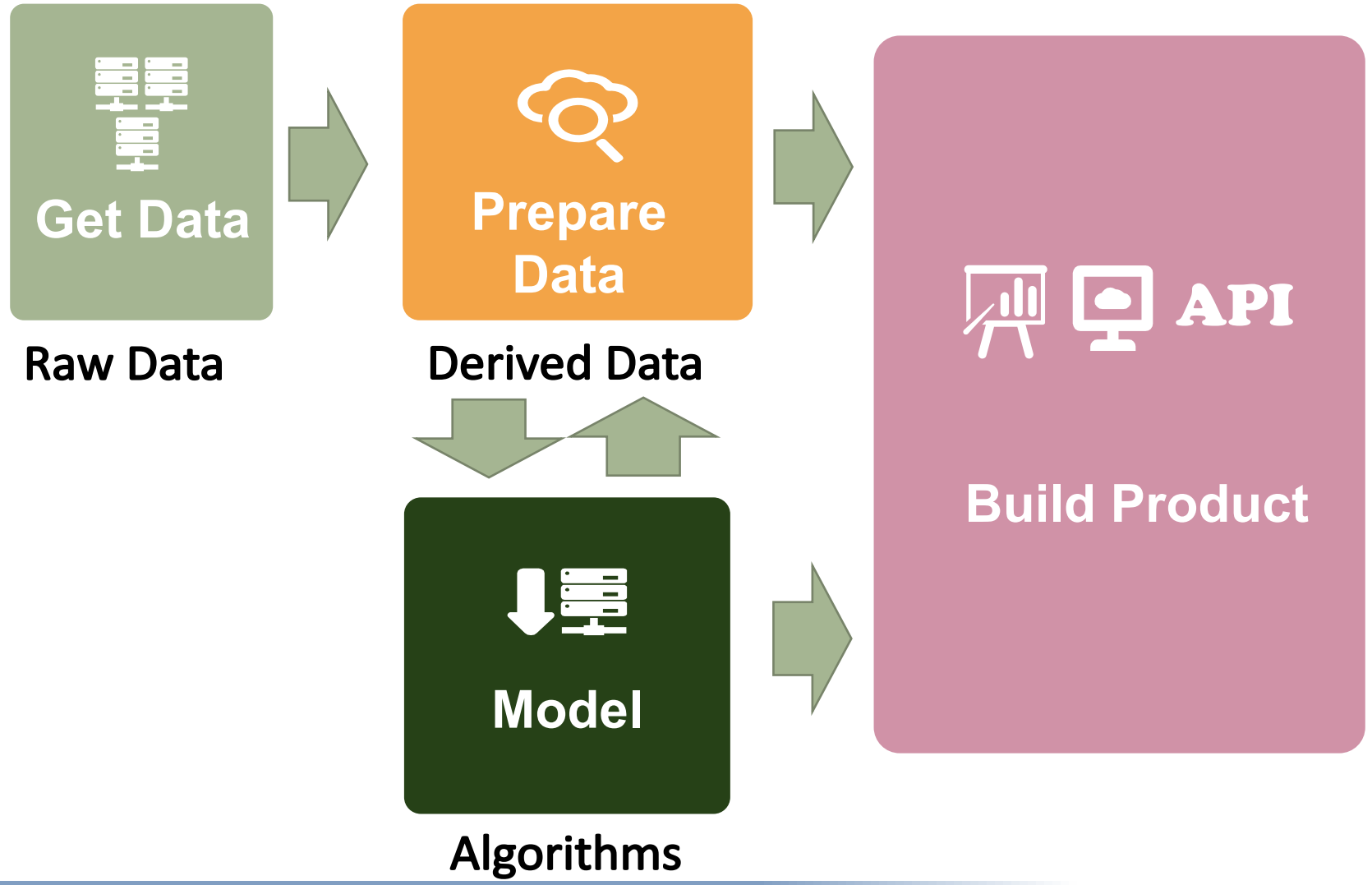
understand data

in order to

gain insights and

make predictions

Going from raw data to a model using data science…



- Visual Dashboards
- Web Interfaces
- Programming Interfaces
- Robotics platforms

Summary of concepts

- Introduced the concept of data product and how data science is a means to generating data products
- Explained how visual dashboard, web interfaces and APIs are used to deploy data products

RECOMMENDED READING:

- **"Data Jujitsu: The Art of Turning Data into Product",** by DJ Patil, 2012.
  http://radar.oreilly.com/2012/07/data-jujitsu.html
- **"What is data science? : The future belongs to the companies and people that turn data into products",** by Mike Loukides, June 2, 2010
  https://www.oreilly.com/ideas/what-is-data-science
- **"The evolution of data products: The data that drives products is shifting from overt to covert",** by Mike Loukides, September 15, 2011
  https://www.oreilly.com/ideas/evolution-of-data-products

Python Data Products

Course 1: Basics

Lecture: Data Product examples in the real world

Learning objectives

In this lecture we will...

- Review three specific examples of recommender data products in the enterprise world

What are data products?

Data Products arise whenever we want to build systems that depend on **predictive models.** For example:

- *Predict* users' future actions based on their past activities
- *Recommend* content to users that they are likely to consume
- *Estimate* demand for a product

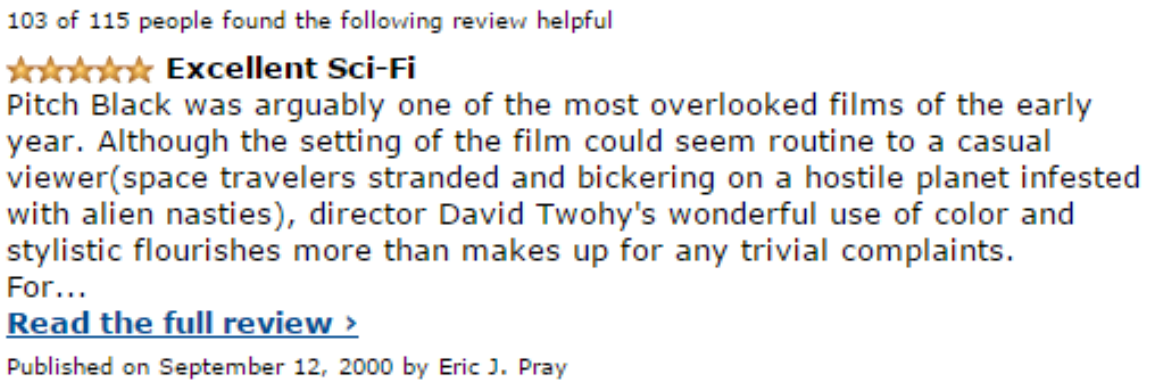
Examples of *Data Products* on the Web

(to name a few)

Examples – Recommender Systems

Modeling task: ***predict*** what rating a person will give to an item
e.g. rating(julian, Pitch Black) = ?

Data Product: build a system to recommend products that people are interested in


103 of 115 people found the following review helpful

Excellent Sci-Fi

Pitch Black was arguably one of the most overlooked films of the early year. Although the setting of the film could seem routine to a casual viewer(space travelers stranded and bickering on a hostile planet infested with alien nasties), director David Twohy's wonderful use of color and stylistic flourishes more than makes up for any trivial complaints. For...

Read the full review ›

Published on September 12, 2000 by Eric J. Pray

Modeling challenges: how are opinions influenced by factors like time, gender, age, and location?

Examples – Social Networks

Modeling task: ***predict*** whether two users of a social network are likely to be friends

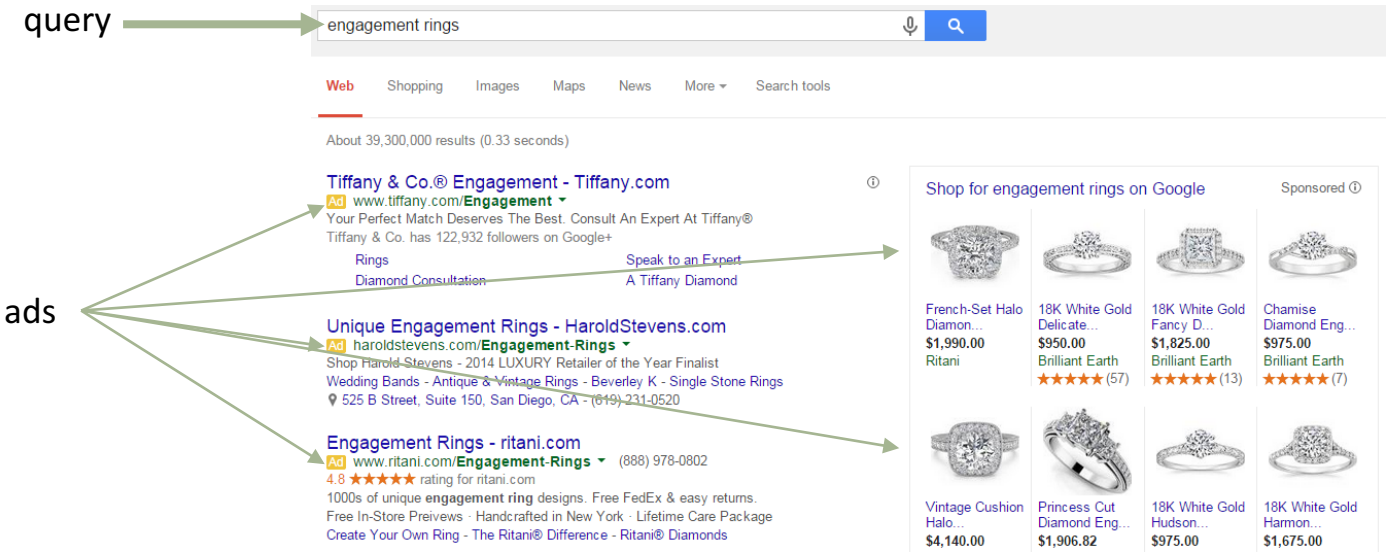
Data Product: "people you may know" and friend recommendation systems

Modeling challenges: what are the features around which friendships form?

Examples – Advertising

Modeling task: ***predict*** whether I will click on an advertisement

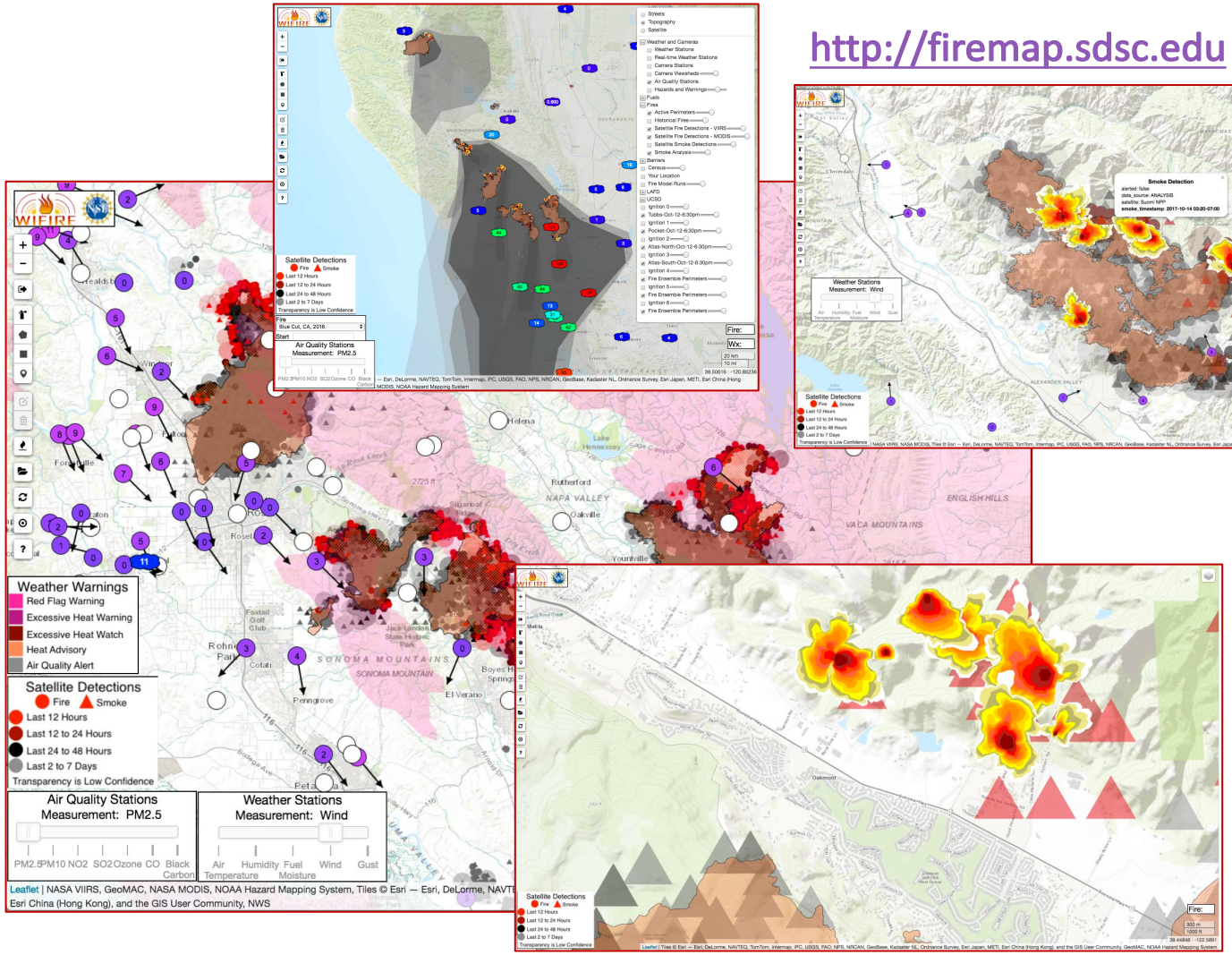
Data product: Ad recommendation systems

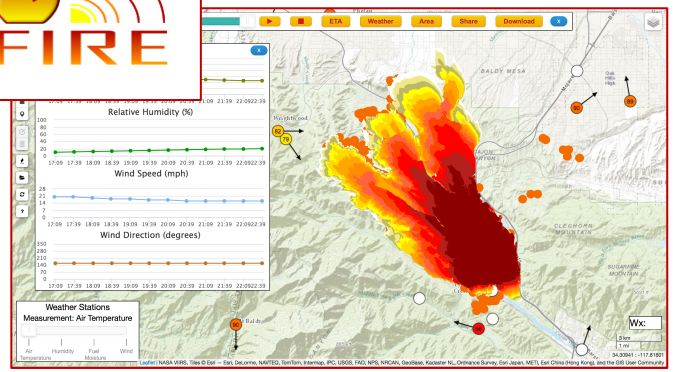


Modeling challenges: what products tend to be purchased together, how do purchases change over time, etc.

A Data Product for predictive wildfire modeling

http://firemap.sdsc.edu





Summary of concepts

- Overviewed recommender systems as a type of data product deployed as a part of most web-based systems
- Summarized a decision-support data product that builds on predictive wildfire behavior modeling

Python Data Products

Course 1: Basics

Lecture: Our Case Study (Recommender Systems)

Learning objectives

In this lecture we will...

- Introduce the concept of Recommender Systems, which we'll use as an ongoing case-study throughout the Specialization
- Describe some of the main datasets we will use to study recommender systems and their main characteristics
- Motivate the use of these datasets for various problems throughout the Specialization

Recommender Systems

As a running example throughout this course, we will build **recommender systems** that model interactions between users and items

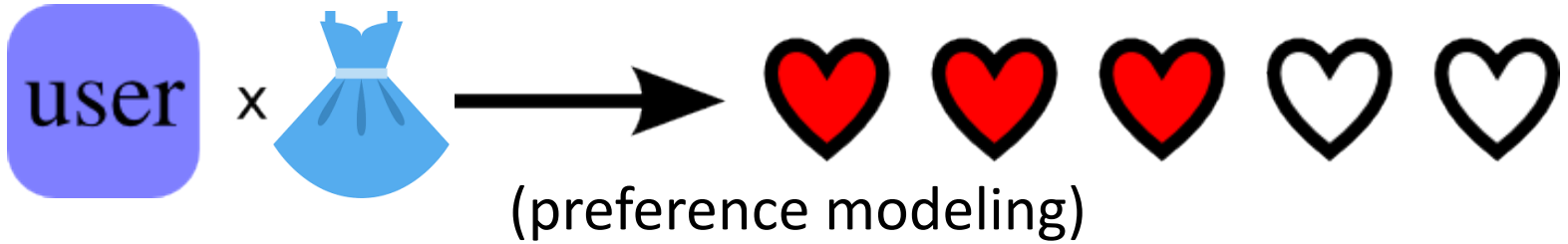
- In particular, we will focus on two publicly-available datasets from Amazon and Yelp:
- https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt
- https://www.yelp.com/dataset/download

Recommender Systems

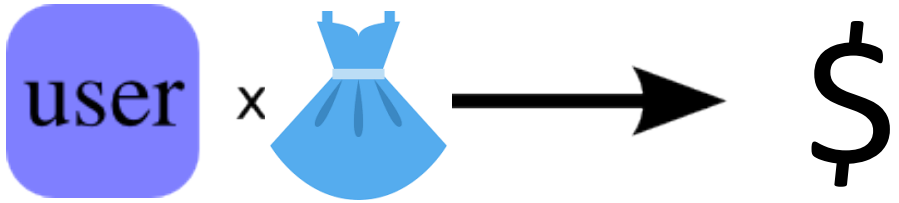
We are particularly interested in tasks including:

- How can we read and process these large datasets, containing complex, structured fields (**Course 1**)?
- How can we make simple predictions from these datasets, such as the sentiment of a review or the category of a business (**Course 2**)?
- How can we validate these predictions, and compare different modeling approaches (**Course 3**)?
- How can we build and deploy a working system using these predictive models (**Course 4**)?

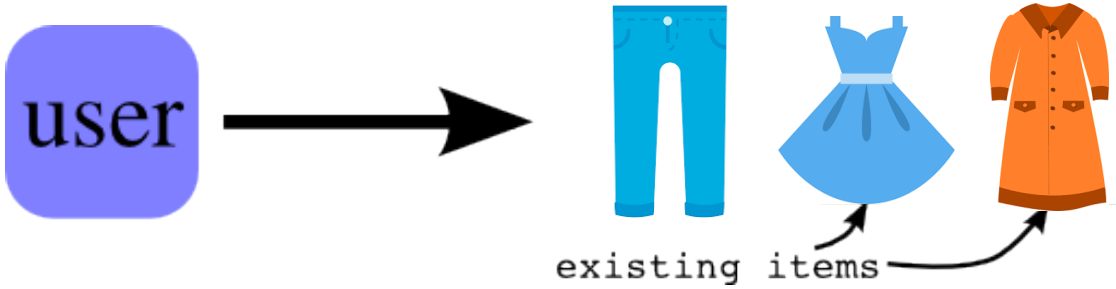
What do recommender systems do?



(preference modeling)



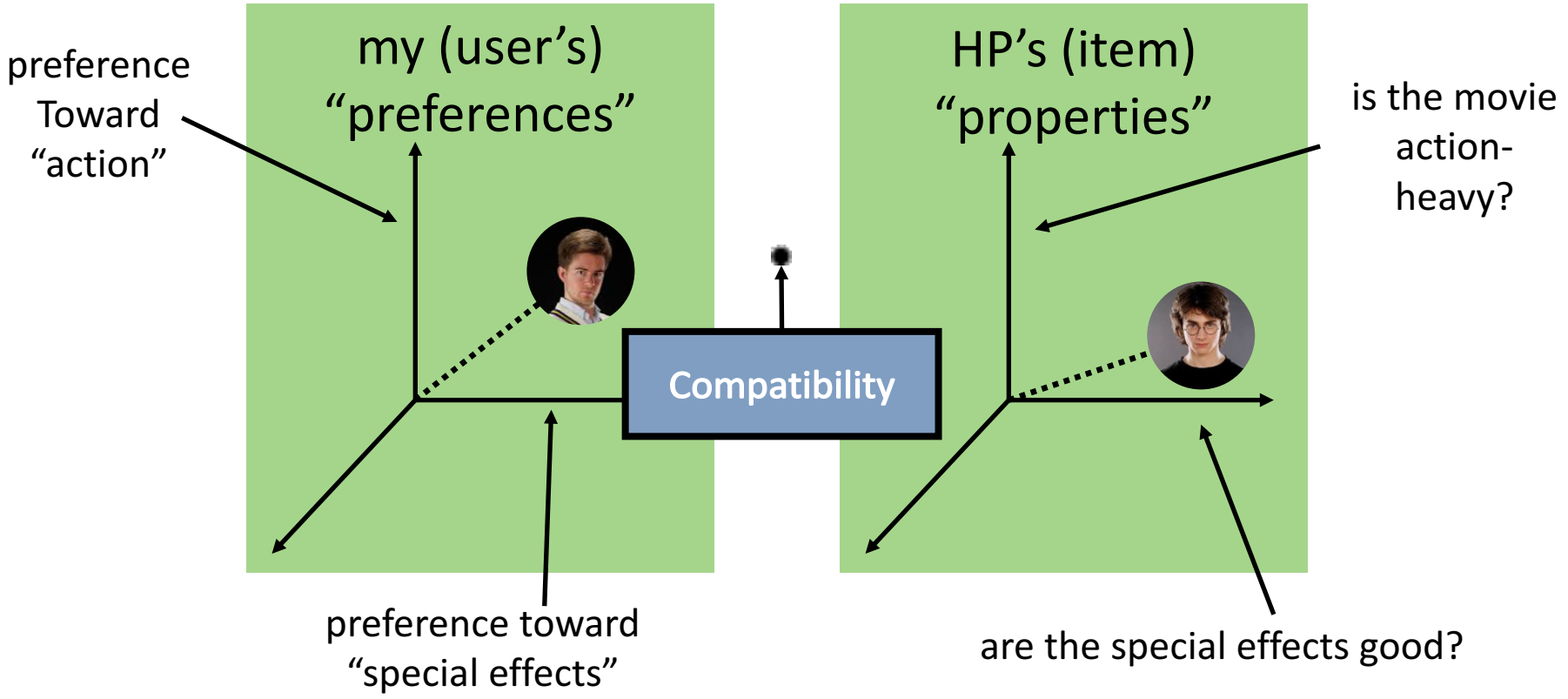
(pricing)



(retrieval)

What do recommender systems do?

In essence, Recommender Systems work by trying to model the **relationships** between people and the items they're evaluating:



Recommender Systems

- In Course 4 we'll look at some of the state-of-the-art (but reasonably common) approaches that are used to implement recommender systems on the web, e.g. rating prediction:

- And "people who bought X also bought Y" etc.:

Recommender Systems

Other than building such systems in Course 4, in the meantime we're also interested in "standard" tasks that can approached using the same type of data, e.g.

- Regression and classification tasks
- Time series modeling
- Text analysis
- Visualization
- (Etc.)

Summary of concepts

- Introduced the concept of Recommender Systems, which we'll use as an ongoing case-study throughout the Specialization

Python Data Products

Course 1: Basics

Lecture: Python and Jupyter basics

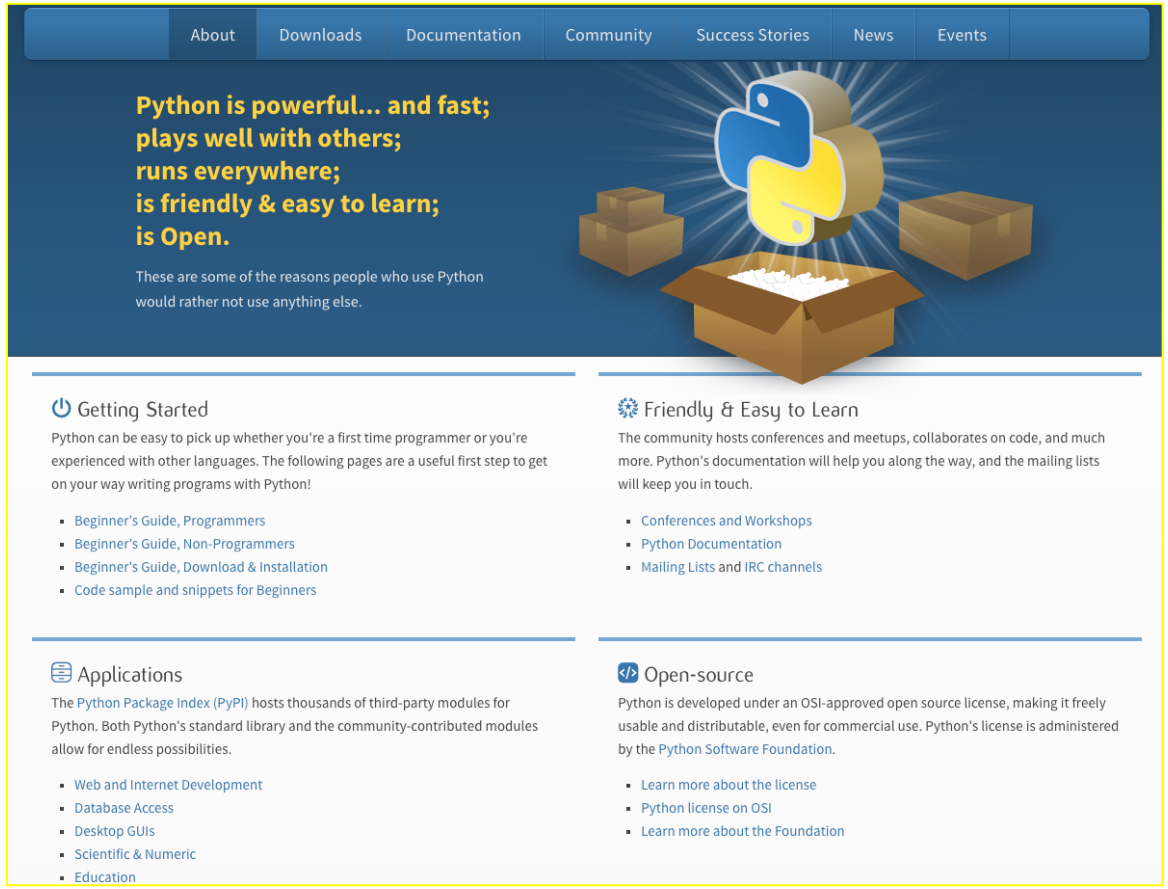
Learning objectives

In this lecture we will...

- Explain why we chose Python as the programming environment for this course
- Talk about some useful Python modules you will be seeing throughout this specialization
- Describe the features of Python-based Jupyter notebooks

What is Python?

https://www.python.org/about/



Why Python for Data Science?

- Easy-to-read and learn
- Vibrant community
- Growing and evolving set of libraries
  - Data management
  - Analytical processing
  - Visaualization
- Applicable to each step in the data science process
- Notebooks

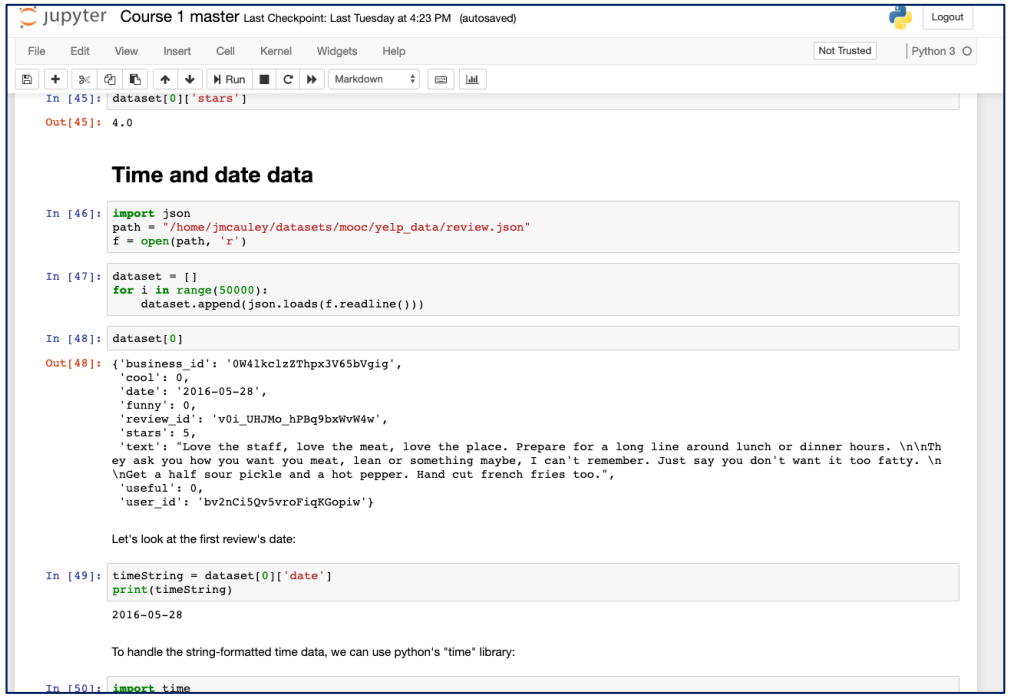
Python libraries we will be learning…

- Jupyter notebooks
- scipy
- numpy
- pandas
- matplotlib
- sklearn
- nltk

Why Jupyter notebooks?

Key features

- Documented Data Science
- Reproducible Science
- Presentation of Results
- Support for Julia, Python, and R



Summary of concepts

- Introduced Python and the Jupyter notebook environment

RECOMMENDED READING:

- **"Advantages and Disadvantages of Python Programming Language"**, by Mindfire Solutions, Apr 23, 2017.
  https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121

- **"Why I'm Learning Python in 2018",** by Alexus Strong, January 12, 2018.
  https://news.codecademy.com/why-learn-python/

Week 2

Data Ingestion

Python Data Products

Course 1: Basics

Lecture: Python Recap

Scripting Languages

Python scripting is similar to a number of other languages

- Interpreted, not compiled
- Prompt/Shell, Scripts and Notebooks
- Easier to learn and code in

https://www.python.org/about/

Hello World!

C

```
#include "stdio.h"
int main() {
  printf("Hello\n");
}
```

Python

```
print("hello")
```

Notice: no ;

Java

```
public class Hi {
  public static void main (String [] args) {
    System.out.println("Hello");
  }
}
```

Python uses Indentations

Python

```
for i in range(0,10):
    print(i)
```

Python uses indentation rather than brackets.

Dynamic Typing

C

```
#include "stdio.h"

int main() {
  int x = 3;
  int y = 4;
  printf("%s"\n,x+y);
}
```

Python

```
x = 3
y = 4
print(x+y)
```

Notice: no types

Dynamic Typing

C

```c
#include "stdio.h"


int main() {
  int x = 3;
  x = 4.5;
}
```

Python

```
x = 3
x = 4.5
```

Notice: no types

Python is Object-Oriented

Python

```
>>> x = "Hello"
>>> x.lower()
'hello'
```

<var_name>.<method_name>(params)

Lists, Dictionaries, and more…

```
>>> list = [11,22,33]
>>> list
```

```
[11, 22, 33]
```

More info:
https://www.python.org/about/gettingstarted/

```
>>> dict = {('Ghostbusters', 2016): 5.4,
('Ghostbusters', 1984): 7.8, ('Cars', 2006):7.1}
>>> x = dict[('Cars',2006)]
>>> x
```

```
7.1
```

Summary of concepts

- Reminders on the basic Python features
- Pointers to beginners resources for Python

RECOMMENDED STARTING POINTS FOR BEGINNERS:

- **"Python Beginners Guide",**
  https://www.python.org/about/gettingstarted/
- **"Python Overview",**
  https://wiki.python.org/moin/BeginnersGuide/Overview
- **"Simple Programs",**
  https://wiki.python.org/moin/SimplePrograms

Python Data Products

Course 1: Basics

Lecture: CSV and JSON files

Learning objectives

In this lecture we will...

- Demonstrate the CSV/TSV and JSON formats
- Compare the main advantages and disadvantages of both formats

CSV and JSON

CSV and JSON are two formats that are easy to read and manipulate in Python

- We'll work through two examples for much of this course:
  - https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt (TSV)
- https://www.yelp.com/dataset/download (JSON)

CSV and JSON

First let's look at the Amazon dataset:

- https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt
- We'll look at data from the "Gift Card" category

Concept: CSV

- CSV is a simple format that allows us to store **tabular** data
- It is a human-readable format, meaning it can easily be read or written via a text-editor or spreadsheet application

E.g. CSV (or rather TSV) from Amazon

```
marketplace    customer_id    review_id    product_id    product_parent    product_title    product_category    star_rating    help
US    24371595    R27ZP1F1CD0C3Y    B004LLIL5A    346014806    Amazon eGift Card - Celebrate    Gift Card    5    0    0    N    Y
US    42489718    RJ7RSBCHUDNNE    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    5    0    0    N    Y    Gift
US    861463    R1HVYBSKLQJI5S    B00IX1I3G6    926539283    Amazon.com Gift Card Balance Reload    Gift Card    5    0    0    N
US    25283295    R2HAXF0IIYQBIR    B00IX1I3G6    926539283    Amazon.com Gift Card Balance Reload    Gift Card    1    0    0    N
US    397970    RNYLPX611NB7Q    B005ESMGV4    379368939    Amazon.com Gift Cards, Pack of 3 (Various Designs)    Gift Card    5
US    18513645    R3ALA9XXMBEDZR    B004KNWWU4    326384774    Amazon Gift Card - Print - Happy Birthday (Birds)    Gift Card    5
US    22484620    R3R8PHAVJFTPDF    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    5    0    0    N    Y    Five
US    14765851    R18WWEK8OIXE30    BT00CTP2EE    775486538    Amazon.com Gift Card in a Greeting Card (Various Designs)    Gift Ca
US    18751931    R1EGUNQON2J277    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    1    0    0    N    Y    One
US    15100528    R21Z4M4L98CPU2    B004W8D102    595099956    Amazon Gift Card - Print - Amazon Boxes    Gift Card    5    0    0
US    3559726    R6JH7A117FHFA    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    5    0    0    N    Y    Five S
US    23413911    R1XZHS8M1GCGI7    B004KNWWU4    326384774    Amazon Gift Card - Print - Happy Birthday (Birds)    Gift Card    5
US    2026222    R1DAI0N03SKRJN    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    5    1    1    N    Y    Five
US    32956435    R2F6SKZOEYQRU3    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    5    0    0    N    N    Five
US    20241560    RIBOP6OEAZA47    B00H5BNLUS    637715957    Amazon eGift Card - Hoops and Yoyo Thank You Very Much (Animated) [Ha
US    10670435    R15H8E7WD6XD29    B004KNWX6C    763371347    Amazon Gift Card - Print - Celebrate    Gift Card    5    0    0
US    48872127    RVN4P3RU4F8IE    BT00CTOYC0    506740729    Amazon.com $15 Gift Card in a Greeting Card (Amazon Surprise Box Desi
US    460630    RCS8F9JCAAXC7    B004LLIKVU    473048287    Amazon.com eGift Cards    Gift Card    4    0    0    N    Y    Four St
US    41238378    R6811C4E7UYL2    B00H5BMH44    81025991    Amazon eGift Card - Hoops and Yoyo Cake Face (Animated) [Hallmark]
```

TSV example

From Amazon's public dataset ("Gift Card" category – amazon_reviews_us_Gift_Card_v1_00.tsv.gz):
https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	
US	24371595	R27ZP1F1CD0C3Y	B004LLIL5A	346014806	Amazon eGift Card - Celebrate	Gift Card	5	0	0	N
US	42489718	RJ7RSBCHUDUNE	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y	G
US	861463	R1HVYBSKLQJI5S	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	5	0	0
US	25283295	R2HAXF0IIYQBIR	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	1	0	0
US	397970	RNYLPX611NB7Q	B005ESMGV4	379368939	Amazon.com Gift Cards, Pack of 3 (Various Designs)	Gift Card
US	18513645	R3ALA9XXMBEDZR	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)	Gift Card
US	22484620	R3R8PHAVJFTPDF	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y
US	14765851	R18WWEK8OIXE30	BT00CTP2EE	775486538	Amazon.com Gift Card in a Greeting Card (Various Designs)	Gif
US	18751931	R1EGUNQON2J277	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	1	0	0	N	Y
US	15100528	R21Z4M4L98CPU2	B004W8D102	595099956	Amazon Gift Card - Print - Amazon Boxes	Gift Card	5	0
US	3559726	R6JH7A117FHFA	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y	Fi
US	23413911	R1XZHS8M1GCGI7	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)	Gift Card
US	2026222	R1DAI0N03SKRJN	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	1	1	N	Y	F
US	32956435	R2F6SK[illegible]	[illegible]KVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	N
US	20241560	RIBOP[illegible]	[illegible]US	637715957	Amazon eGift Card - Hoops and Yoyo Thank You Very Much (Animated)
US	10670435	R15H8[illegible]	[illegible]X6C	763371347	Amazon Gift Card - Print - Celebrate	Gift Card	5	0	0
US	48872127	RVN4P[illegible]	[illegible]C0	506740729	Amazon.com $15 Gift Card in a Greeting Card (Amazon Surprise Box
US	460630	RCS8F9JC[illegible]	[illegible]	473048287	Amazon.com eGift Cards	Gift Card	4	0	0	N	Y	Fou
US	41238378	R6811C4E7UYL2	B00H5BMH44	81025991	Amazon eGift Card - Hoops and Yoyo Cake Face (Animated) [Hallmark]

Data are separated by *tabs* (tsv) or *commas* (csv)

TSV example

From Amazon's public dataset ("Gift Card" category – amazon_reviews_us_Gift_Card_v1_00.tsv.gz):
https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	
US	24371595	R27ZP1F1CD0C3Y	B004LLIL5A	346014806	Amazon eGift Card - Celebrate	Gift Card	5	0	0	N
US	42489718	RJ7RSBCHUDNNE	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y	G
US	861463	R1HVYBSKLQJI5S	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	5	0	0			
US	25283295	R2HAXF0IIYQBIR	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	1	0	0			
US	397970	RNYLPX611NB7Q	B005ESMGV4	379368939	Amazon.com Gift Cards, Pack of 3 (Various Designs)	Gift Card						
US	18513645	R3ALA9XXMBEDZR	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)	Gift Card						
US	22484620	R3R8PHAVJFTPDF			mazon.com eGift Cards	Gift Card	5	0	0	N	Y	
US	14765851	R18WWEK8OIXE30			mazon.com Gift Card in a Greeting Card (Various Designs)	Gif						
US	18751931	R1EGUNQON2J277			mazon.com eGift Cards	Gift Card	1	0	0	N	Y	
US	15100528	R21Z4M4L98CPU2			mazon Gift Card - Print - Amazon Boxes	Gift Card	5	0				
US	3559726	R6JH7A117FHFA			zon.com eGift Cards	Gift Card	5	0	0	N	Y	Fi
US	23413911	R1XZHS8M1GCGI7	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)	Gift Card						
US	2026222	R1DAI0N03SKRJN	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	1	1	N	Y	F
US	32956435	R2F6SKZOEYQRU3	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	N	
US	20241560	RIBOP6OEAZA47	B00H5BNLUS	637715957	Amazon eGift Card - Hoops and Yoyo Thank You Very Much (Animated)							
US	10670435	R15H8E7WD6XD29	B004KNWX6C	763371347	Amazon Gift Card - Print - Celebrate	Gift Card	5	0	0			
US	48872127	RVN4P3RU4F8IE	BT00CTOYC0	506740729	Amazon.com $15 Gift Card in a Greeting Card (Amazon Surprise Box							
US	460630	RCS8F9JCAAXC7	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	4	0	0	N	Y	Fou
US	41238378	R6811C4E7UYL2	B00H5BMH44	81025991	Amazon eGift Card - Hoops and Yoyo Cake Face (Animated) [Hallmark]							

The first row is the *header*, which indicates what value each field represents

TSV example

From Amazon's public dataset ("Gift Card" category – amazon_reviews_us_Gift_Card_v1_00.tsv.gz):
https://s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	
US	24371595	R27ZP1F1CD0C3Y	B004LLIL5A	346014806	Amazon eGift Card - Celebrate	Gift Card	5	0	0	N
US	42489718	RJ7RSBCHUDNNE	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y	G
US	861463	R1HVYBSKLQJI5S	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	5	0	0			
US	25283295	R2HAXF0IIYQBIR	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload	Gift Card	1	0	0			
US	397970	RNYLPX611NB7Q	B005ESMGV4	379368939	Amazon.com Gift Cards, Pack of 3 (Various Designs)	Gift Card						
US	18513645	R3ALA9XXMBEDZR	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)	Gift Card						
US	22484620	R3R8PHAVJFTPDF	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	Y	
US	14765851	R18WWEK8OIXE30	BT00CTP2EE	775486538	Amazon.com Gift Card in a Greeting Card (Various Designs)	Gif						
US	18751931	R1EGUNQON2J277	B004LLIKV	m eGift Cards	Gift Card	1	0	0	N	Y		
US	15100528	R21Z4M4L98CPU2	B004W8D1C	ft Card - Print - Amazon Boxes	Gift Card	5	0					
US	3559726	R6JH7A117FHFA	B004LLIKVU	eGift Cards	Gift Card	5	0	0	N	Y	Fi	
US	23413911	R1XZHS8M1GCGI7	B004KNWWU	ft Card - Print - Happy Birthday (Birds)	Gift Card							
US	2026222	R1DAI0N03SKRJN	B004LLIKVU	eGift Cards	Gift Card	5	1	1	N	Y	F	
US	32956435	R2F6SKZOEYQRU3	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	5	0	0	N	N	
US	20241560	RIBOP6OEAZA47	B00H5BNLUS	637715957	Amazon eGift Card - Hoops and Yoyo Thank You Very Much (Animated)							
US	10670435	R15H8E7WD6XD29	B004KNWX6C	763371347	Amazon Gift Card - Print - Celebrate	Gift Card	5	0	0			
US	48872127	RVN4P3RU4F8IE	BT00CTOYC0	506740729	Amazon.com $15 Gift Card in a Greeting Card (Amazon Surprise Box							
US	460630	RCS8F9JCAAXC7	B004LLIKVU	473048287	Amazon.com eGift Cards	Gift Card	4	0	0	N	Y	Fou
US	41238378	R6811C4E7UYL2	B00H5BMH44	81025991	Amazon eGift Card - Hoops and Yoyo Cake Face (Animated) [Hallmark]							

Each other row corresponds to a single product review from *Amazon*

TSV example

Can be a little easier to visualize if we align the columns vertically:

Note that the data is essentially ***tabular*** and is much like an *Excel* (or similar) spreadsheet

marketplace	customer_id	review_id	product_id	product_parent	product_title
US	24371595	R27ZP1F1CD0C3Y	B004LLIL5A	346014806	Amazon eGift Card - Celebrate
US	42489718	RJ7RSBCHUDNNE	B004LLIKVU	473048287	Amazon.com eGift Cards
US	861463	R1HVYBSKLQJI5S	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload
US	25283295	R2HAXF0IIYQBIR	B00IX1I3G6	926539283	Amazon.com Gift Card Balance Reload
US	397970	RNYLPX611NB7Q	B005ESMGV4	379368939	Amazon.com Gift Cards, Pack of 3 (Various Designs)
US	18513645	R3ALA9XXMBEDZR	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)
US	22484620	R3R8PHAVJFTPDF	B004LLIKVU	473048287	Amazon.com eGift Cards
US	14765851	R18WWEK8OIXE30	BT00CTP2EE	775486538	Amazon.com Gift Card in a Greeting Card (Various D
US	18751931	R1EGUNQON2J277	B004LLIKVU	473048287	Amazon.com eGift Cards
US	15100528	R21Z4M4L98CPU2	B004W8D102	595099956	Amazon Gift Card - Print - Amazon Boxes
US	3559726	R6JH7A117FHFA	B004LLIKVU	473048287	Amazon.com eGift Cards
US	23413911	R1XZHS8M1GCGI7	B004KNWWU4	326384774	Amazon Gift Card - Print - Happy Birthday (Birds)
US	2026222	R1DAI0N03SKRJN	B004LLIKVU	473048287	Amazon.com eGift Cards
US	32956435	R2F6SKZOEYQRU3	B004LLIKVU	473048287	Amazon.com eGift Cards
US	20241560	RIBOP6OEAZA47	B00H5BNLUS	637715957	Amazon eGift Card - Hoops and Yoyo Thank You Very
US	10670435	R15H8E7WD6XD29	B004KNWX6C	763371347	Amazon Gift Card - Print - Celebrate
US	48872127	RVN4P3RU4F8IE	BT00CTOYC0	506740729	Amazon.com $15 Gift Card in a Greeting Card (Amazc
US	460630	RCS8F9JCAAXC7	B004LLIKVU	473048287	Amazon.com eGift Cards
US	41238378	R6811C4E7UYL2	B00H5BMH44	81025991	Amazon eGift Card - Hoops and Yoyo Cake Face (Anim

TSV example

Let's look at some more columns of the data:

product_category	star_rating	helpful_votes	total_votes	vine	verified_purchase	review_headline	review_body	rev
Gift Card	5	0	0	N	Y	Five Stars	Great birthday gi	
Gift Card	5	0	0	N	Y	Gift card for the greatest	selection of iter	
Gift Card	5	0	0	N	Y	Five Stars	Good 2015-08-31	
Gift Card	1	0	0	N	Y	One Star	Fair 2015-08-31	
Gift Card							I can't believe ho	
Gift Card							casion! Perfect	
Gift Card	5	0	0	N	Y	Five Stars	excelent 2015-(	
Gift Card	5	0	0	N	Y	Five Stars	Great and Safe Gi	
Gift Card	1	0	0	N	Y	One Star	What????????? 2	
Gift Card	5	0	0	N	Y	Five Stars	This was just too	
Gift Card	5	0	0	N	Y	Five Stars	Bien 2015-08-31	
Gift Card	5	1	1	N	Y	Always good	Easy to print fron	
Gift Card	5	1	1	N	Y	Five Stars	Amazing with 10 do	
Gift Card	5	0	0	N	N	Five Stars	Remember Matthew T	
Gift Card	5	1	1	N	Y	Five Stars	good 2015-08-31	
Gift Card	5	0	0	N	Y	Five Stars	Awesome way to ser	
Gift Card	5	0	0	N	Y	Quick Solution for Forgotten Occasion	I l	
Gift Card	4	0	0	N	Y	Four Stars	Good gift. Easy to	
Gift Card	5	0	0	N	Y	Satisfied customer	Satisfied as usual	

What was the rating, how many helpful votes were received (out of how many), was the purchase verified, etc.

Concept: JSON

- CSV/TSV format is **simple and effective**, but **limited** in the types of data that can be stored
- I.e., it is limited to **tabular** data
- For example, how would you store
  - A business's opening hours (e.g. in *Yelp's* data)?
  - A set of categories for a product?
  - Playlist entries in the "million playlist dataset"? (https://labs.spotify.com/2018/05/30/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/)
- Fields best represented as **lists** or **sets** are inconvenient to store as CSV/TSV entries
- **JSON** attempts to address this by allowing for more general structured data to be represented

Concept: JSON

- See e.g. yelp's dataset: https://www.yelp.com/dataset/download
- Let's look at the first line of the "business.json" file:


```
{'business_id': 'FYWN1wneV18bWNgQjJ2GNg', 'attributes':
{'BusinessAcceptsCreditCards': True, 'AcceptsInsurance': True,
'ByAppointmentOnly': True}, 'longitude': -111.9785992,
'state': 'AZ', 'address': '4855 E Warner Rd, Ste B9',
'neighborhood': '', 'city': 'Ahwatukee', 'hours': {'Tuesday':
'7:30-17:00', 'Wednesday': '7:30-17:00', 'Thursday': '7:30-
17:00', 'Friday': '7:30-17:00', 'Monday': '7:30-17:00'},
'postal_code': '85044', 'review_count': 22, 'stars': 4.0,
'categories': ['Dentists', 'General Dentistry', 'Health &
Medical', 'Oral Surgeons', 'Cosmetic Dentists',
'Orthodontists'], 'is_open': 1, 'name': 'Dental by Design',
'latitude': 33.3306902}
```

JSON

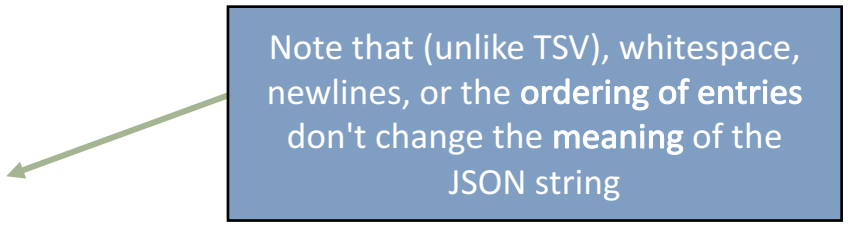
Might look a little cleaner if we format it more carefully:


```
{
  'business_id': 'FYWN1wneV18bWNgQjJ2GNg',
  'attributes':
  {
    'BusinessAcceptsCreditCards': True,
    'AcceptsInsurance': True,
    'ByAppointmentOnly': True
  },
  'longitude': -111.9785992,
  'latitude': 33.3306902,
  'state': 'AZ',
  'address': '4855 E Warner Rd, Ste B9',
  'neighborhood': '',
  'city': 'Ahwatukee',
  'hours':
  {
    'Tuesday': '7:30-17:00',
    'Wednesday': '7:30-17:00',
    'Thursday': '7:30-17:00',
    'Friday': '7:30-17:00',
    'Monday': '7:30-17:00'
  },
  'postal_code': '85044',
  'review_count': 22,
  'stars': 4.0,
  'categories':
  ['Dentists', 'General Dentistry', 'Health & Medical', 'Oral Surgeons', 'Cosmetic Dentists', 'Orthodontists'],
  'is_open': 1,
  'name': 'Dental by Design'
}
```

JSON

Might look a little cleaner if we format it more carefully:

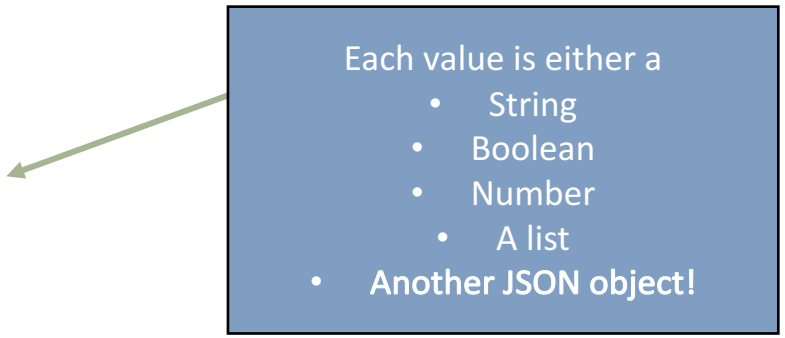
```
{
  'business_id': 'FYWN1wneV18bWNgQjJ2GNg',
  'attributes':
  {
    'BusinessAcceptsCreditCards': True,
    'AcceptsInsurance': True,
    'ByAppointmentOnly': True
  },
  'longitude': -111.9785992,
  'latitude': 33.3306902,
  'state': 'AZ',
  'address': '4855 E Warner Rd, Ste B9',
  'neighborhood': '',
  'city': 'Ahwatukee',
  'hours':
  {
    'Tuesday': '7:30-17:00',
    'Wednesday': '7:30-17:00',
    'Thursday': '7:30-17:00',
    'Friday': '7:30-17:00',
    'Monday': '7:30-17:00'
  },
  'postal_code': '85044',
  'review_count': 22,
  'stars': 4.0,
  'categories':
  ['Dentists', 'General Dentistry', 'Health & Medical', 'Oral Surgeons', 'Cosmetic Dentists', 'Orthodontists'],
  'is_open': 1,
  'name': 'Dental by Design'
}
```



JSON

Might look a little cleaner if we format it more carefully:

```
{
  'business_id': 'FYWN1wneV18bWNgQjJ2GNg',
  'attributes':
  {
    'BusinessAcceptsCreditCards': True,
    'AcceptsInsurance': True,
    'ByAppointmentOnly': True
  },
  'longitude': -111.9785992,
  'latitude': 33.3306902,
  'state': 'AZ',
  'address': '4855 E Warner Rd, Ste B9',
  'neighborhood': '',
  'city': 'Ahwatukee',
  'hours':
  {
    'Tuesday': '7:30-17:00',
    'Wednesday': '7:30-17:00',
    'Thursday': '7:30-17:00',
    'Friday': '7:30-17:00',
    'Monday': '7:30-17:00'
  },
  'postal_code': '85044',
  'review_count': 22,
  'stars': 4.0,
  'categories':
  ['Dentists', 'General Dentistry', 'Health & Medical', 'Oral Surgeons', 'Cosmetic Dentists', 'Orthodontists'],
  'is_open': 1,
  'name': 'Dental by Design'
}
```



CSV/TSV vs. JSON

CSV/TSV:

Advantages:

+ Simple, human-readable format
+ Can be easily manipulated in "tabular" form
– e.g can be read & modified using *Excel* or similar tools

Disadvantages:

- Cannot represent complex, flexible (I.e., non-tabular) data

JSON:

Advantages:

+ Allows for manipulation of complex, semi-structured data

Disadvantages:

- More difficult to explore and manipulate using "GUI" tools

Summary of concepts

- You should understand the **format** of json and csv files
- Understand the relative **merits** of both formats

On your own...

- Download the Amazon and Yelp datasets
- Try opening the (smaller) files in a text editor

Python Data Products

Course 1: Basics

Lecture: Reading CSV and JSON into Python

Learning objectives

In this lecture we will...

- Demonstrate the main **methods** to read CSV/TSV and JSON files in Python
- **Understand** some of the edge cases that make reading these formats difficult

CSV/TSV in Python

In this lecture we'll look through a few functions to read CSV/TSV and JSON data in Python:

- string.split()
- csv.reader (library)
- eval() and ast.eval()
- json.loads (library)

Code: String.split()

```
In [1]: x = "marketplace customer_id review_id product_id product_parent"

In [2]: x.split()
Out[2]: ['marketplace', 'customer_id', 'review_id', 'product_id', 'product_parent']

In [3]: x = "marketplace; customer_id; review_id; product_id; product_parent"

In [4]: x.split(';')
Out[4]: ['marketplace', ' customer_id', ' review_id', ' product_id', ' product_parent']
```

Note: preserves whitespace!

- Converts a **string** to a **list**, given a **separator**
- By default, any whitespace separator is used (tab, space, newline)
- But different separators can be provided via an optional argument

Code: String.split()

What happens when the delimiter appears in the column?

```
In [1]: x = '4.0, "good product, would buy again"'

In [2]: x.split(',')
Out[2]: ['4.0', ' "good product', ' would buy again"']
```

Note: splits into three columns rather than two!

- This could be addressed by using a different delimiter (e.g. ';'), though this doesn't generalize for fields containing arbitrary text
- Normally, the field will be escaped by quotes

Code: CSV.reader

```
In [1]: import csv

In [2]: path = "datasets/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv"

In [3]: f = open(path)

In [4]: reader = csv.reader(f, delimiter = '\t')

In [5]: next(reader)

Out[5]: ['marketplace',
         'customer_id',
         'review_id',
         'product_id',
         'product_parent',
         'product_title',
         'product_category',
         'star_rating',
         'helpful_votes',
         'total_votes',
         'vine',
         'verified_purchase',
         'review_headline',
         'review_body',
         'review_date']
```

Note: specify what delimiter to use (tab)

first line is the **header**

Code: CSV.reader

```
In [6]: next(reader)
```

```
Out[6]: ['US',
         '24371595',
         'R27ZP1F1CD0C3Y',
         'B004LLIL5A',
         '346014806',
         'Amazon eGift Card - Celebrate',
         'Gift Card',
         '5',
         '0',
         '0',
         'N',
         'Y',
         'Five Stars',
         'Great birthday gift for a young adult.',
         '2015-08-31']
```

next line is the first review in the dataset

Code: eval()

Reading json files is even easier as they're very similar to Python's built-in dictionaries:

```
In [1]: path = "datasets/yelp_data/review.json"

In [2]: f = open(path)

In [3]: line = f.readline()

In [4]: line

Out[4]: '{"review_id":"v0i_UHJMo_hPBq9bxWvW4w","user_id":"bv2nCi5Qv5vroFiqKGopiw","business_id":"0W4lkclzZThpx3V65bVgig","st
ars":5,"date":"2016-05-28","text":"Love the staff, love the meat, love the place. Prepare for a long line around lun
ch or dinner hours. \\n\\nThey ask you how you want you meat, lean or something maybe, I can\'t remember. Just say y
ou don\'t want it too fatty. \\n\\nGet a half sour pickle and a hot pepper. Hand cut french fries too.","useful":
0,"funny":0,"cool":0}\n'
```

Note: first line of Yelp's review data

Code: eval()

Reading json files is even easier as they're very similar to Python's built-in dictionaries:

```
In [5]: d = eval(line)

In [6]: d

Out[6]: {'business_id': '0W4lkclzZThpx3V65bVgig',
         'cool': 0,
         'date': '2016-05-28',
         'funny': 0,
         'review_id': 'v0i_UHJMo_hPBq9bxWvW4w',
         'stars': 5,
         'text': "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nT
        hey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty.
        \n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
         'useful': 0,
         'user_id': 'bv2nCi5Qv5vroFiqKGopiw'}

In [7]: d['user_id']

Out[7]: 'bv2nCi5Qv5vroFiqKGopiw'
```

Code: eval()

Note that the "eval" function just treats an arbitrary string as if it were python code:

```
In [1]: eval("4 + 2")
Out[1]: 6
```

- While convenient, this could be **dangerous** to run on untrusted datasets since it could execute arbitrary code
- We can use some library functions to make sure that only valid json data gets executed
- We'll look at the **ast** (abstract syntax tree) and **json** libraries

Code: ast and json libraries

```
In [5]: ast.literal_eval(line)

Out[5]: {'business_id': '0W4lkclzZThpx3V65bVgig',
         'cool': 0,
         'date': '2016-05-28',
         'funny': 0,
         'review_id': 'v0i_UHJMo_hPBq9bxWvW4w',
         'stars': 5,
         'text': "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nT
        hey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty.
        \n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
         'useful': 0,
         'user_id': 'bv2nCi5Qv5vroFiqKGopiw'}
```

- Note that the outputs are identical, the code is merely "safer" to execute

Code: ast and json libraries

```
In [6]: import json

In [7]: json.loads(line)

Out[7]: {'business_id': '0W4lkclzZThpx3V65bVgig',
 'cool': 0,
 'date': '2016-05-28',
 'funny': 0,
 'review_id': 'v0i_UHJMo_hPBq9bxWvW4w',
 'stars': 5,
 'text': "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nT
hey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty.
\n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
 'useful': 0,
 'user_id': 'bv2nCi5Qv5vroFiqKGopiw'}
```

- Note that the outputs are identical, the code is merely "safer" to execute

Summary of concepts

- Understand the **methods** .split() and eval()
- Understand the **libraries** ast and json
- Be able to read JSON and CSV data in Python

On your own...

- Try reading the Amazon dataset (or the first few lines) using csv.reader
- Try reading the Yelp dataset using json.loads()

Week 3

Exploratory Data Analysis

Python Data Products

Course 1: Basics

Lecture: Processing Structured Data in Python

Learning objectives

In this lecture we will...

- Demonstrate how to read JSON/CSV files into python objects
- Introduce the "gzip" library

Reading data into data structures

- In a previous lecture we saw the basics of how to use the CSV/JSON libraries to read structured data
- What comes next? I.e., how to we read the data into appropriate data structures?

```
In [1]: import csv

In [2]: path = "datasets/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv"

In [3]: f = open(path)

In [4]: reader = csv.reader(f, delimiter = '\t')

In [5]: next(reader)
Out[5]: ['marketplace',
         'customer_id',
         'review_id',
         'product_id',
         'product_parent',
```

Reading data into data structures

- In a previous lecture we saw the basics of how to use the CSV/JSON libraries to read structured data
- What comes next? I.e., how to we read the data into appropriate data structures?

1. How do we read larger csv/json files without having to unzip them?
2. How do we extract relevant parts of the data for performing analysis?
3. What structures make access to the data more convenient?

Code: The gzip library

```
In [1]: import gzip
        path = "datasets/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
        f = gzip.open(path, 'rt')

In [2]: import csv
        reader = csv.reader(f, delimiter = '\t')

In [3]: header = next(reader)

In [4]: header

Out[4]: ['marketplace',
         'customer_id',
         'review_id',
         'product_id',
         'product_parent',
         'product_title',
         'product_category',
         'star_rating',
         'helpful_votes',
         'total_votes',
         'vine',
         'verified_purchase',
         'review_headline',
```

"rt" indicates that the file is a text file (default is to read as bytes)

Even this small file is 12mb zipped and 39mb unzipped

Otherwise, the file can be treated like a regular file

- Often we'll want to manipulate files that are cumbersome to fit on disk if we extract them
- The gzip library allows us to read zipped files (.gz) without unzipping them

Code: Reading and filtering files line by line

```
In [5]: dataset = []

In [6]: for line in reader:
            line = line[:-3]
            if line[-1] == 'Y':
                dataset.append(line)

In [7]: dataset[0]

Out[7]: ['US',
         '24371595',
         'R27ZP1F1CD0C3Y',
         'B004LLIL5A',
         '346014806',
         'Amazon eGift Card - Celebrate',
         'Gift Card',
         '5',
         '0',
         '0',
         'N',
         'Y']
```

File is read one line at a time

Drop the text fields

Discard unverified reviews

Two ideas:

1. Read the file one line at a time (rather than reading the whole thing and *then* processing it)
2. Perform filtering as we read the data, so that it is never stored in memory

Code: Reading CSV files into key-value pairs

```
In [5]: dataset = []
```

```
In [6]: for line in reader:
            d = dict(zip(header, line))
            for field in ['helpful_votes', 'star_rating', 'total_votes']:
                d[field] = int(d[field])
            for field in ['verified_purchase', 'vine']:
                if d[field] == 'Y':
                    d[field] = True
                else:
                    d[field] = False
            dataset.append(d)
```

dict(zip(header,line)) makes the line into a **dictionary**

Convert numeric and boolean fields to Python types

```
In [7]: dataset[0]
```

```
Out[7]: {'customer_id': '24371595',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False}
```

Two ideas:

1. The "dict" operator makes the line into a dictionary, allowing us to index fields by keys (rather than numbers)
2. Convert strings to numbers/booleans where possible

Summary of concepts

- Introduced the gzip library
- Saw some techniques for preprocessing datasets as we read them

On your own...

- Try reading some of the larger Amazon datasets (or the Yelp review data) and compiling statistics from them
- Experiment with the dict() and zip() operators

Python Data Products

Course 1: Basics

Lecture: Extracting simple statistics from datasets

Learning objectives

In this lecture we will...

- Introduce data structures that help us to compile statistics (like "defaultdict")
- Compute simple statistics like counts, sums, and averages from data

Simple statistics from data

Let's try to compute the following from the Amazon data:

- What is the average star rating?
- What is the distribution of star ratings?
- What fraction of purchases are verified?
- Which products are the most popular (purchases)?
- Which products have the highest average ratings?

Reading the data

First let's read the Amazon data into a list, exactly as we did in the previous lecture:

In [1]:
```
import gzip
path = "datasets/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
f = gzip.open(path, 'rt')
```

In [2]:
```
import csv
reader = csv.reader(f, delimiter = '\t')
```

In [3]:
```
header = next(reader)
```

In [4]:
```
dataset = []
for line in reader:
    d = dict(zip(header, line))
    for field in ['helpful_votes', 'star_rating', 'total_votes']:
        d[field] = int(d[field])
    for field in ['verified_purchase', 'vine']:
        if d[field] == 'Y':
            d[field] = True
        else:
            d[field] = False
    dataset.append(d)
```

Code: Average rating and rating distribution

- Average rating can be computed straightforwardly with a list comprehension:

```
In [5]: ratings = [d['star_rating'] for d in dataset]

In [6]: sum(ratings) / len(ratings)
Out[6]: 4.731333018677096
```

- Rating distribution can be computed by using a dictionary to store counts:

```
In [7]: ratingCounts = {1: 0, 2: 0, 3: 0, 4: 0, 5:0}

In [8]: for d in dataset:
            ratingCounts[d['star_rating']] += 1

In [9]: ratingCounts
Out[9]: {1: 4766, 2: 1560, 3: 3147, 4: 9808, 5: 129029}
```

Code: defaultdict

- Note that we counted ratings by initializing a dictionary with all zero counts:

```
In [7]: ratingCounts = {1: 0, 2: 0, 3: 0, 4: 0, 5:0}
```

- The "defaultdict" structure from the "collections" library allows us to automate this functionality, which is useful for counting different types of object
  - Let's compute the rating distribution using defaultdict:

```
In [10]: from collections import defaultdict

In [11]: ratingCounts = defaultdict(int)

In [12]: for d in dataset:
             ratingCounts[d['star_rating']] += 1

In [13]: ratingCounts
Out[13]: defaultdict(int, {1: 4766, 2: 1560, 3: 3147, 4: 9808, 5: 129029})
```

Code: verified purchases

- Similarly we can use the defaultdict function to count verified vs. non-verified purchases

```
In [14]: verifiedCounts = defaultdict(int)

In [15]: for d in dataset:
             verifiedCounts[d['verified_purchase']] += 1

In [16]: verifiedCounts
Out[16]: defaultdict(int, {False: 13021, True: 135289})
```

Code: most popular products

- Again we can use defaultdict to determine product popularity (here we just want to count which products appear most in the dataset)

```
In [17]: productCounts = defaultdict(int)

In [18]: for d in dataset:
             productCounts[d['product_id']] += 1

In [19]: counts = [(productCounts[p], p) for p in productCounts]

In [20]: counts.sort()

In [21]: counts[-10:]

Out[21]: [(2038, 'B004KNWWO0'),
          (2173, 'B0066AZGD4'),
          (2630, 'BT00DDC7CE'),
          (2643, 'B004LLIKY2'),
          (3407, 'BT00DDC7BK'),
          (3440, 'BT00CTOUNS'),
          (4283, 'B00IX1I3G6'),
          (5034, 'BT00DDVMVQ'),
          (6037, 'B00A48G0D4'),
          (28705, 'B004LLIKVU')]
```

- Following this, we build a list of counts followed by product IDs, which we can sort to get the most popular

Code: top rated products

- Here we need to compute the average rating for each product, which requires that we first construct the **list** of ratings for each product
  - This can also be done using defaultdict, with the "list" subclass:

```
In [22]: ratingsPerProduct = defaultdict(list)

In [23]: for d in dataset:
             ratingsPerProduct[d['product_id']].append(d['star_rating'])

In [24]: averageRatingPerProduct = {}
         for p in ratingsPerProduct:
             averageRatingPerProduct[p] = sum(ratingsPerProduct[p]) / len(ratingsPerProduct[p])
```

- We now have two data structures: one which stores the **list** of ratings for each product, and one which stores the **average** rating for each product

Code: top rated products

- Now we can sort by ratings, and also filter to only include reasonably popular products:

```
In [25]: topRated = [(averageRatingPerProduct[p], p) for p in averageRatingPerProduct if len(ratingsPerProduct[p]) > 50]

In [26]: topRated.sort()

In [27]: topRated[-10:]

Out[27]: [(4.918918918918919, 'B004KNWX94'),
          (4.919354838709677, 'B00CRQ496G'),
          (4.923076923076923, 'B00PMLDNBA'),
          (4.931034482758621, 'B00CT77E60'),
          (4.936842105263158, 'B004KNWX76'),
          (4.9423076923076925, 'B00SNMPQYC'),
          (4.944444444444445, 'B007V6EWKK'),
          (4.947368421052632, 'B004LLIL5K'),
          (4.955882352941177, 'B00H5BNKYA'),
          (4.966101694915254, 'B00P8N49M4')]
```

Only products with more than 50 reviews

Summary of concepts

- Saw how to compute simple statistics from datasets
- Introduced the "defaultdict" structure

On your own...

Try computing other statistics, e.g.

- Who are the most active users?
- What are the most commonly used words?
- What is the different in average rating between verified versus non-verified purchases?

Python Data Products

Course 1: Basics

Lecture: Data filtering and cleaning

Learning objectives

In this lecture we will...

- Introduce basic protocols for filtering datasets
- Provide Python code to extract subsets of our data

Why preprocessing?

Although the datasets we're working with have already been "cleaned" to some extent, in general there may be many reasons we'd want to further clean or pre-process datasets, e.g.:

- Certain fields may be missing from some entries
- Certain entries may be garbled / poorly formatted
- Parts of a dataset may be "stale" or otherwise unusable
- Parts of the dataset may contain statistical outliers (which we may or may not want to keep)
- We may want to remove data pertaining to rare or inactive users
- May want to restrict our dataset to a certain demographic or region
- Etc.

Data preprocessing in Python

In this lecture we'll look at a few such examples in Python

In [1]:
```
import gzip
path = "/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
f = gzip.open(path, 'rt')
```

In [2]:
```
import csv
reader = csv.reader(f, delimiter = '\t')
```

In [3]:
```
header = next(reader)
```

In [4]:
```
dataset = []
for line in reader:
    d = dict(zip(header, line))
    for field in ['helpful_votes', 'star_rating', 'total_votes']:
        d[field] = int(d[field])
    for field in ['verified_purchase', 'vine']:
        if d[field] == 'Y':
            d[field] = True
        else:
            d[field] = False
    dataset.append(d)
```

Data preprocessing in Python

Recall that our data looks something like this:

```
In [5]: len(dataset)

Out[5]: 148310

In [6]: dataset[0]

Out[6]: {'customer_id': '24371595',
         'helpful_votes': 0,
         'marketplace': 'US',
         'product_category': 'Gift Card',
         'product_id': 'B004LLIL5A',
         'product_parent': '346014806',
         'product_title': 'Amazon eGift Card - Celebrate',
         'review_body': 'Great birthday gift for a young adult.',
         'review_date': '2015-08-31',
         'review_headline': 'Five Stars',
         'review_id': 'R27ZP1F1CD0C3Y',
         'star_rating': 5,
         'total_votes': 0,
         'verified_purchase': True,
         'vine': False}
```

Code: Filtering by date

First let's try to filter reviews by date. We'll see more on processing time/date data later, but for the moment we'll just filter based on the review's year. We first need to convert this to an integer:

Year portion of date

```
In [7]: for d in dataset:
            d['yearInt'] = int(d['review_date'][:4])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-0343dcce14e0> in <module>()
      1 for d in dataset:
----> 2     d['yearInt'] = int(d['review_date'][:4])

KeyError: 'review_date'
```

That threw an error! Why?

- The 'review_date' field must be missing from some reviews

Code: Filtering by date

So, we'll first have to preprocess our dataset to exatract only those entries containing a 'review_date' field:

```
In [8]: dataset = [d for d in dataset if 'review_date' in d]

In [9]: len(dataset)
Out[9]: 148309
```

- Looks like it was just one reivew!
- Now we can try again:

```
In [10]: for d in dataset:
             d['yearInt'] = int(d['review_date'][:4])
```

Code: Filtering by date

Finally let's filter out old reviews, e.g. those written before 2010:

```
In [11]: dataset = [d for d in dataset if d['yearInt'] > 2009]

In [12]: len(dataset)
Out[12]: 148095
```

- Note that we did this preprocessing (and will do most preprocessing) using a fairly simple list comprehension

Code: Filtering by review quality

Similarly, we might filter reviews based on their "helpfulness". Let's write another list comprehension to exclude reviews with low helpful rates:

```
In [13]: dataset = [d for d in dataset if d['total_votes'] < 3 or d['helpful_votes']/d['total_votes'] >= 0.5]

In [14]: len(dataset)
Out[14]: 147801
```

Keep reviews that haven't received many votes yet

Otherwise, delete any with less than 50% helpfulness

Code: Filtering by user activity

Next, let's filter our dataset to discard inactive users (in this case, users who have written only a single review in this category)

First we'll use the "defaultdict" class, as we did in the "simple statistics" lecture:

```
In [15]: from collections import defaultdict

In [16]: nReviewsPerUser = defaultdict(int)

In [17]: for d in dataset:
             nReviewsPerUser[d['customer_id']] += 1
```

Then we can filter to keep only users with 2 or more reviews:

```
In [18]: dataset = [d for d in dataset if nReviewsPerUser[d['customer_id']] >= 2]

In [19]: len(dataset)
Out[19]: 11172
```

Note: this cuts the dataset down **a lot**

Code: Filtering by review length

Finally, let's filter very short reviews, which may be uninformative

```
In [20]: dataset = [d for d in dataset if len(d['review_body'].split()) >= 10]

In [21]: len(dataset)
Out[21]: 7033
```

Data preprocessing

These are just a few of the possible ways that we could filter our dataset, for instance we could extend this by:

- Filtering products that have few reviews
- Filtering users who have only given '5-star' ratings
- Filtering reviews that aren't part of the "vine" program
- Filter users who haven't written a review for more than a year
- Etc.

But note that this type of filtering **quickly and drastically** decreases the amount of data we have to work with!

Summary of concepts

- Gave examples of protocols we may use to filter data
- Developed Python code to apply simple preprocessing schemes
- Saw the effect of preprocessing on real data

On your own...

- Try applying the same preprocessing protocols to the Yelp data (or a small subset of it) and see what effect it has on the amount of workable data

Python Data Products

Course 1: Basics

Lecture: text and string processing in Python

Learning objectives

In this lecture we will...

- Perform simple manipulations of string data in Python
- Discover a few useful library functions for string processing

Strings in Python

In this lecture me'll look through a few functions to manipulate string data in Python:

- string.split() and string.join()
- List operations on strings
- index() and find()
- The "string" library

Strings in Python

First let's read in a review from the Yelp dataset:

In [1]:
```
import json
import string
```

In [2]:
```
path = "/home/jmcauley/datasets/mooc/yelp_data/review.json"
```

In [3]:
```
f = open(path)
```

In [4]:
```
d = json.loads(f.readline())
```

In [5]:
```
d
```

Out[5]:
```
{'business_id': '0W4lkclzZThpx3V65bVgig',
 'cool': 0,
 'date': '2016-05-28',
 'funny': 0,
 'review_id': 'v0i_UHJMo_hPBq9bxWvW4w',
 'stars': 5,
 'text': "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nT
hey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty.
\n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
 'useful': 0,
 'user_id': 'bv2nCi5Qv5vroFiqKGopiw'}
```

In [6]:
```
review = d['text']
```

Code: String.split()

```
In [7]: reviewWords = review.split()

In [8]: reviewWords

Out[8]: ['Love',
         'the',
         'staff,',
         'love',
         'the',
         'meat,',
         'love',
         'the',
         'place.',
         'Prepare',
         'for',
         'a',
         'long',
         'line',
         'around',
         'lunch',
         'or',
         'dinner',
         'hours.',
         'They',
         'ask',
         'you',
```

- We saw string.split() previously when reading CSV/TSV files
- Here, .split() can be used to convert a string to a list of words (or we could split it based on another character)
- This process is known as **tokenization**

Code: String.join()

```
In [9]: ' '.join(reviewWords)

Out[9]: "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. They ask you h
         ow you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty. Get a half so
         ur pickle and a hot pepper. Hand cut french fries too."
```

- String.join() is like .split() in reverse: it takes a list (here the **list of words** in the review), and converts them to a string, by placing the same token (here a **space character**) in between each one

Code: String.lower()

```
In [10]: review.lower()

Out[10]: "love the staff, love the meat, love the place. prepare for a long line around lunch or dinner hours. \n\nthey ask y
         ou how you want you meat, lean or something maybe, i can't remember. just say you don't want it too fatty. \n\nget a
         half sour pickle and a hot pepper. hand cut french fries too."
```

- String.lower() converts a string to lower case
- This operation can be useful before we compute statistics on strings – it allows for easier comparison between different variants of the same word
- Similarly **string.upper()** converts a string to upper case

Code: List operations on strings

- Regular python list operations will work on strings

```
In [11]: len(review)
Out[11]: 289
```

Note: # characters

```
In [12]: len(reviewWords)
Out[12]: 56
```

Note: # words

```
In [13]: review[:10]
Out[13]: 'Love the s'
```

```
In [14]: reviewWords[:10]
Out[14]: ['Love',
          'the',
          'staff,',
          'love',
          'the',
          'meat,',
          'love',
          'the',
          'place.',
          'Prepare']
```

```
In [15]: reviewWords.index("pickle")
Out[15]: 46
```

Note: word position

```
In [16]: review.find("pickle")
Out[16]: 238
```

Note: # characters into review that the word appears

```
In [17]: review.find("cucumber")
Out[17]: -1
```

```
In [18]: review.count("love")
Out[18]: 2
```

```
In [19]: review.lower().count("love")
Out[19]: 3
```

Code: removing punctuation characters

```
In [20]: string.punctuation

Out[20]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

In [21]: [x for x in review if not x in string.punctuation]

Out[21]: ['L',
          'o',
          'v',
          'e',
          ' ',
          't',
          'h',
          'e',
          ' ',
          's',
          't',
          'a',
          'f',
          'f',
          ' ',
          'l',
          'o',
          'v',
          'e',
          ' '

In [22]: ''.join([x for x in review if not x in string.punctuation])

Out[22]: 'Love the staff love the meat love the place Prepare for a long line around lunch or dinner hours \n\nThey ask you h
         ow you want you meat lean or something maybe I cant remember Just say you dont want it too fatty \n\nGet a half sour
         pickle and a hot pepper Hand cut french fries too'
```

- We can remove punctuation by performing a list comprehension on the string
- The "string" library contains utility functions that we can use (for e.g.) to get the list of punctuation tokens
- Finally, we have to use join to convert the output back to a string

Strings in Python

These are just a few of the most basic operations, see also:

- string.startswith() (etc.)
- string.isalpha() (etc.)
- string.strip()
- Other operations in the string library
- (later) the NLTK library

Summary of concepts

- Understand a few of the basic Python string operations
- Apply list operations to strings
- "Tokenize" strings into lists and vice versa

On your own...

- Try computing simple statistics from string data, e.g. how often does a particular word appear among Yelp reviews, and which words are the most common?

Python Data Products

Course 1: Basics

Lecture: Time and date data

Learning objectives

In this lecture we will...

- Consider various **formats and structures** to represent time and date data
- Demonstrate the main **methods** to manipulate time data in python
- **Convert** time and date data between various formats

Time and date data

Dealing with time and date data can be difficult as string-formatted data doesn't admit easy comparison or feature representation:

- Which date occurs first, 4/7/2003 or 3/8/2003?
- How many days between 4/5/2003 - 7/15/2018?
- e.g. how many hours between 2/6/2013 23:02:38 - 2/7/2013 08:32:35?

Time and date data

Most of the data we've seen so far include plain-text time data, that we need to carefully manipulate:


```
{'business_id': 'FYWN1wneV18bWNgQjJ2GNg', 'attributes':
{'BusinessAcceptsCreditCards': True, 'AcceptsInsurance': True,
'ByAppointmentOnly': True}, 'longitude': -111.9785992,
'state': 'AZ', 'address': '4855 E Warner Rd, Ste B9',
'neighborhood': '', 'city': 'Ahwatukee', 'hours': {'Tuesday':
'7:30-17:00', 'Wednesday': '7:30-17:00', 'Thursday': '7:30-
17:00', 'Friday': '7:30-17:00', 'Monday': '7:30-17:00'},
'postal_code': '85044', 'review_count': 22, 'stars': 4.0,
'categories': ['Dentists', 'General Dentistry', 'Health &
Medical', 'Oral Surgeons', 'Cosmetic Dentists',
'Orthodontists'], 'is_open': 1, 'name': 'Dental by Design',
'latitude': 33.3306902}
```

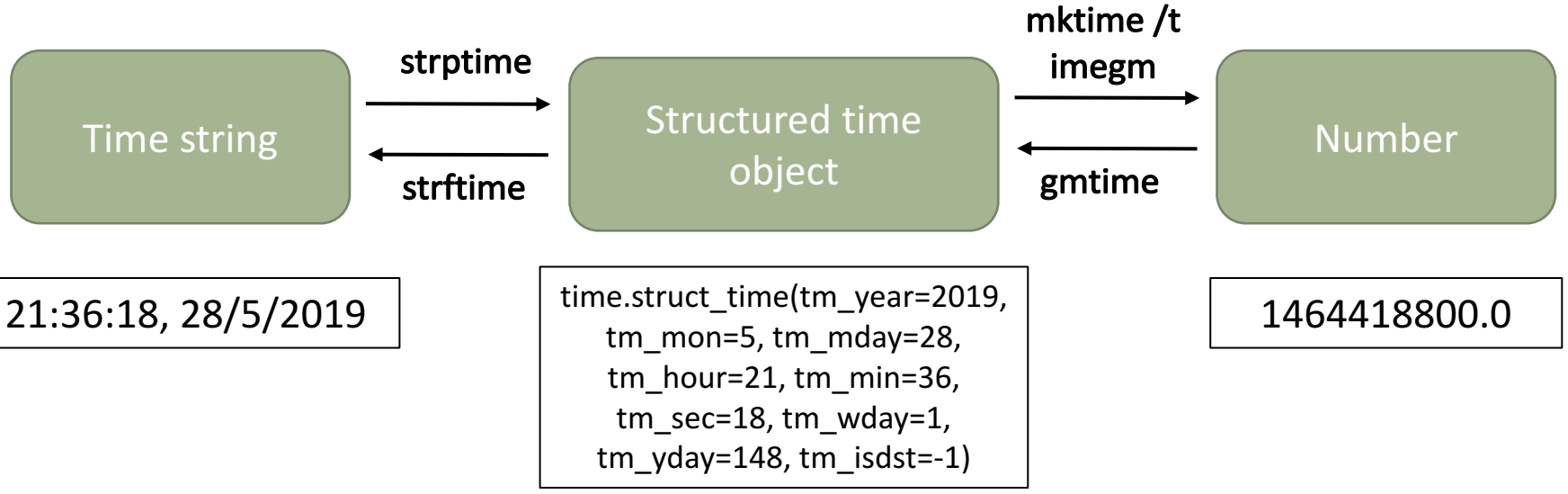
Time and date data

In this lecture we'll cover a few functions:

- Time.strptime: convert a time **string** to a structured time **object**
- Time.strftime: convert a time **object** to a **string**
- Time.mktime / calendar.timegm: convert a time **object** to a **number**
- Time.gmtime: convert a **number** to a time **object**

Time and date data

In this lecture we'll cover a few functions:



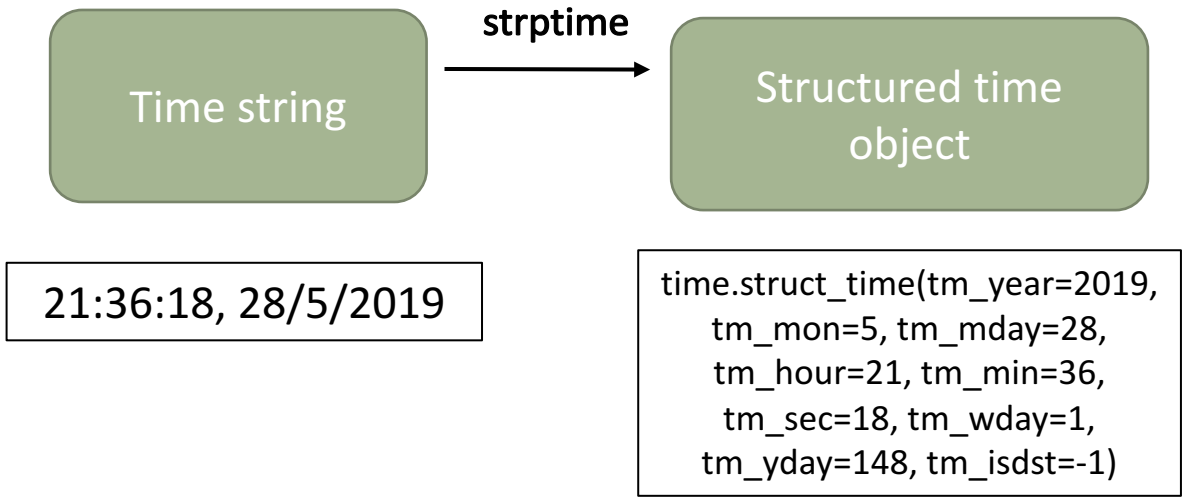
Concept: Unix time

Internally, time is often represented as a number, which allows for easy manipulation and arithmetic

- The value (Unix time) is the **number of seconds since Jan 1, 1970 in the UTC timezone**
- so I made this slide at 1532568962 = 2018-07-26 01:36:02 UTC (or 18:36:02 in my timezone)
- But real datasets generally have time as a "human readable" string
- Our goal here is to convert between these two formats

strptime

First, let's look at converting a string to a structured object (strptime)



Code: time.strptime()

```
In [1]: import time
        import calendar
```

String-formatted time data

```
In [2]: timeString = "2018-07-26 01:36:02"

In [3]: timeStruct = time.strptime(timeString, "%Y-%m-%d %H:%M:%S")

In [4]: timeStruct

Out[4]: time.struct_time(tm_year=2018, tm_mon=7, tm_mday=26, tm_hour=1, tm_min=36, tm_sec=2, tm_wday=3, tm_yday=207, tm_isds
        t=-1)

In [5]: timeStruct.tm_wday

Out[5]: 3
```

Note: this day is a Wednesday!

```
In [6]: help(time.strptime)

    Help on built-in function strptime in module time:

    strptime(...)
        strptime(string, format) -> struct_time

        Parse a string to a time tuple according to a format specification.
```

Note: different time formatting options in the help page

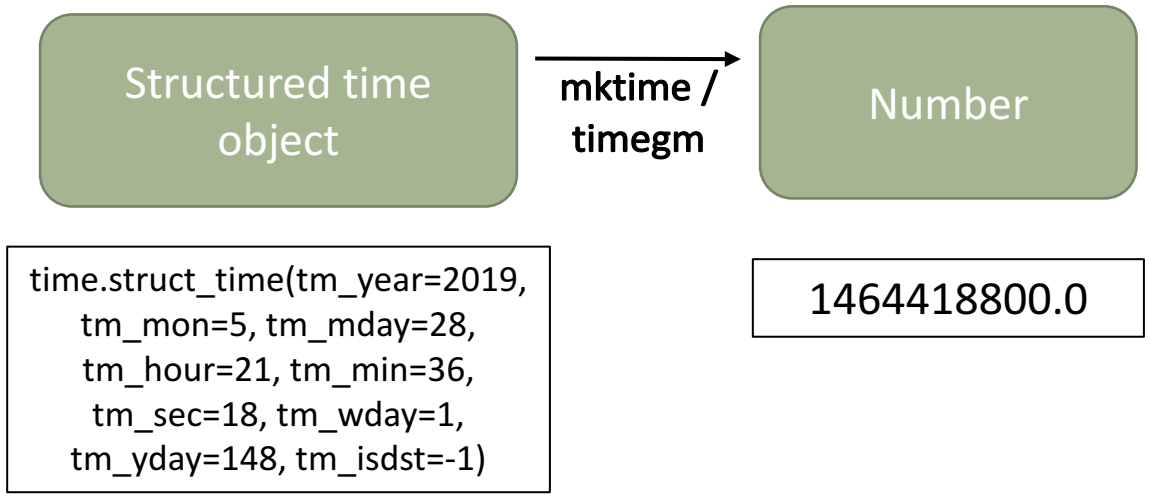
strptime

Strptime is convenient when we want to extract **features** from data

- E.g. does a date correspond to a weekday or a weekend?
- Converting month names or abbreviations (e.g. "Jan") to month numbers
- Dealing with mixed-format data by converting it to a common format
- But if we want to perform arithmetic on timestamps, converting to a number may be easier

time.mktime and calendar.timegm

For this we'll use mktime to convert our structured time object to a number:



Code: time.mktime() and calendar.timegm()

Structured time data from previous slide

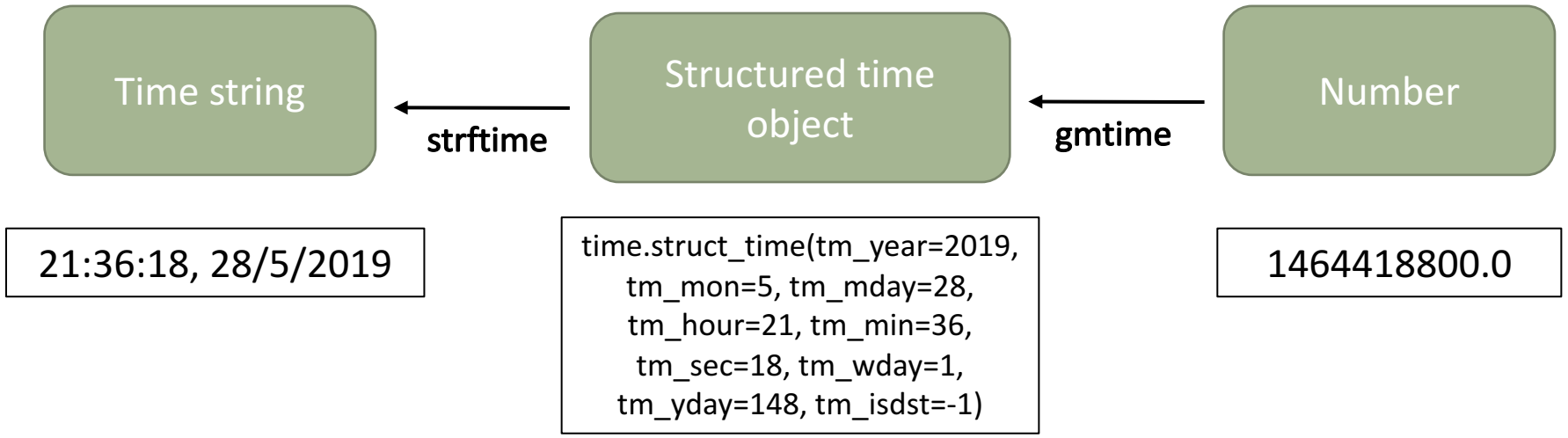
```
In [7]: t1 = calendar.timegm(timeStruct)

In [8]: t2 = time.mktime(timeStruct)

In [9]: t1, t2
Out[9]: (1532568962, 1532594162.0)

In [10]: t1 + 60*60*24*5
Out[10]: 1533000962
```

Five days later

- time.mktime() allows us to convert our structured time object to a number
- **NOTE:** mktime assumes the structure is a *local* time whereas timegm assumes the structure is a *UTC* time
- This allows for easy manipulation, arithmetic, and comparison (e.g. sorting) of time data

time.strftime and time.gmtime

Finally, both of these operations can be *reversed*, should we wish to format time data as a string or structure



Code: time.strftime() and time.gmtime()

```
In [11]: time.gmtime(t1 + 60*60*24*5)
```

Five days later than the previous time

```
Out[11]: time.struct_time(tm_year=2018, tm_mon=7, tm_mday=31, tm_hour=1, tm_min=36, tm_sec=2, tm_wday=1, tm_yday=212, tm_isdst=0)
```

```
In [12]: time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime(t1 + 60*60*24*5))
```

```
Out[12]: '2018-07-31 01:36:02'
```

- These methods can be used to put adjusted times back into string format

Summary of concepts

- Understand the idea and motivation behind **unix time**
- Understand the methods strptime, strftime, mktime, and gmtime
- Be able to convert between various time formats
- Be able to read and manipulate string-formatted time data from real datasets

On your own...

- Try converting the dates in Yelp or Amazon reviews to unix time, and sorting the reviews by their date

Week 4

Tools for data processing and visualization

Python Data Products

Course 1: Basics

Lecture: Data Frames and Pandas

Learning objectives

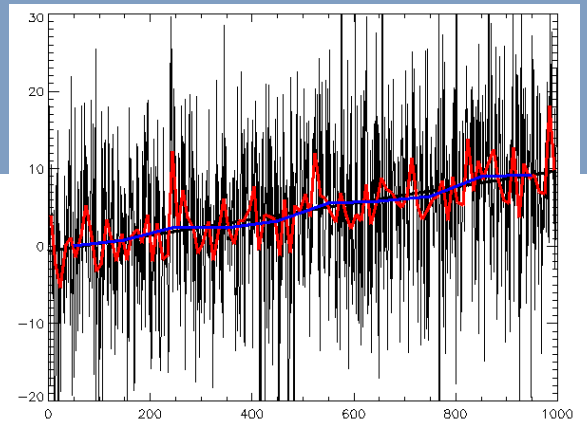
In this lecture we will...

- Describe the efficient and easy to use methods that pandas provides for importing data into memory
- Identify functions such as ‘read_csv’ for reading a CSV file into a DataFrame
- Describe the capabilities of Pandas for performing statistical analysis on data
- Leverage frequently used functions such as describe()
- Identify key plotting functions of Pandas

pandas Benefits

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Support for time-series data

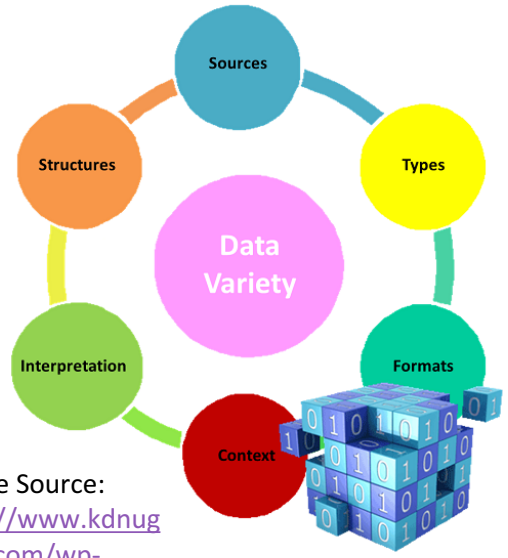
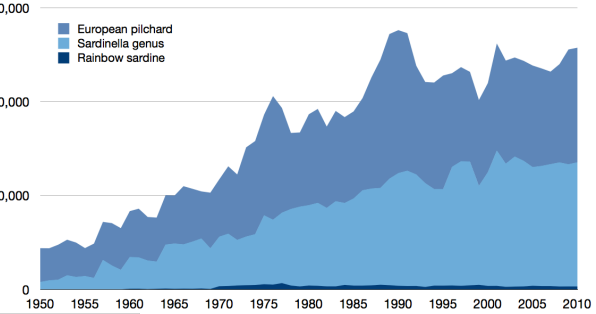
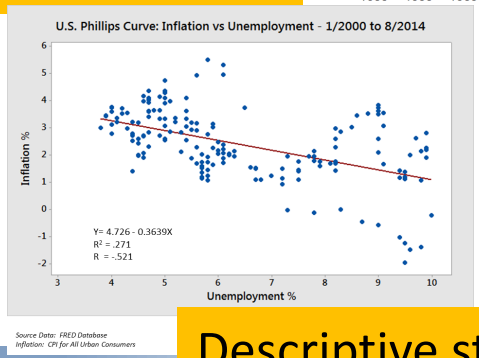


Image Source: http://www.kdnuggets.com/wp-content/uploads/data-variety.png



Visualizations



Descriptive statistics

- Data variety support
- Data integration
- Data transformation

pandas Data Structures

```
In [3]: ser = pd.Series(data = [100, 200, 300, 400, 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])

In [6]: ser
Out[6]: tom      100
        bob      200
        nancy    300
        dan      400
        eric     500
        dtype: int64

In [7]: ser.index
Out[7]: Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')

In [9]: ser[[4, 3, 1]]
Out[9]: eric    500
        dan     400
        bob     200
        dtype: int64

In [10]: ser['nancy']
Out[10]: 300

In [11]: 'bob' in ser
Out[11]: True

In [16]: ser * 2
Out[16]: tom       200
         bob       400
         nancy     600
         dan       800
         eric     1000
         dtype: int64

In [17]: ser ** 2
Out[17]: tom       10000
         bob       40000
         nancy     90000
         dan      160000
         eric     250000
         dtype: int64
```

pandas Series

```
In [46]: d = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
              'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cerill', 'dancy'])}

In [47]: df = pd.DataFrame(d)
         df
```

Out[47]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

```
In [48]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'])
```

Out[48]:

	one	two
dancy	NaN	4444.0
ball	200.0	222.0
apple	100.0	111.0

```
In [49]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

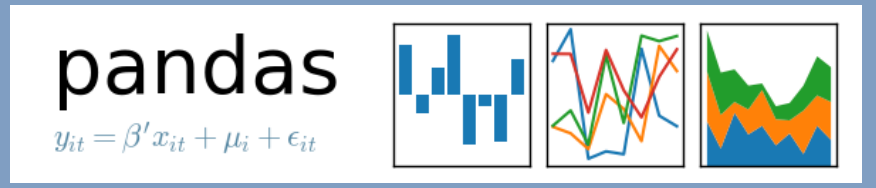
Out[49]:

	two	five
dancy	4444.0	NaN
ball	222.0	NaN
apple	111.0	NaN

```
In [50]: df.index
Out[50]: Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')

In [51]: df.columns
Out[51]: Index(['one', 'two'], dtype='object')
```

pandas DataFrame

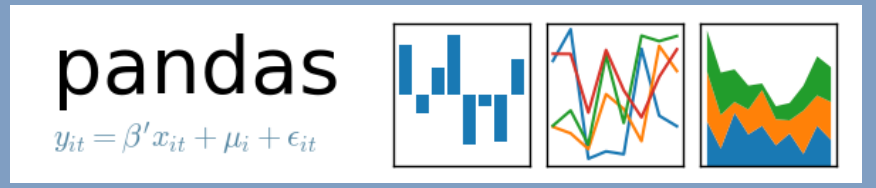
pandas Series



- A 1-dimensional labeled array
- Supports many data types
- Axis labels → index
  - get and set values by index label
- Valid argument to most NumPy methods

```
In [3]: ser = pd.Series(data = [100, 200, 300, 400, 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])

In [6]: ser

Out[6]: tom      100
        bob      200
        nancy    300
        dan      400
        eric     500
        dtype: int64

In [7]: ser.index

Out[7]: Index(['tom', 'bob', 'nancy', 'dan', 'eric'], dtype='object')

In [9]: ser[[4, 3, 1]]

Out[9]: eric    500
        dan     400
        bob     200
        dtype: int64

In [10]: ser['nancy']

Out[10]: 300

In [11]: 'bob' in ser

Out[11]: True

In [16]: ser * 2

Out[16]: tom      200
         bob      400
         nancy    600
         dan      800
         eric     1000
         dtype: int64

In [17]: ser ** 2

Out[17]: tom      10000
         bob      40000
         nancy    90000
         dan      160000
         eric     250000
         dtype: int64
```

pandas DataFrame



- A 2-dimensional labeled data structure
- A dictionary of Series objects
  - Columns can be of potentially different types
  - Optionally parameters for fine-tuning:
    - index (row labels)
    - columns (column labels)

```
In [46]: d = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
              'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cerill', 'dancy'])}

In [47]: df = pd.DataFrame(d)
         df
```

Out[47]:

	one	two
apple	100.0	111.0
ball	200.0	222.0
cerill	NaN	333.0
clock	300.0	NaN
dancy	NaN	4444.0

```
In [48]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'])
```

Out[48]:

	one	two
dancy	NaN	4444.0
ball	200.0	222.0
apple	100.0	111.0

```
In [49]: pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

Out[49]:

	two	five
dancy	4444.0	NaN
ball	222.0	NaN
apple	111.0	NaN

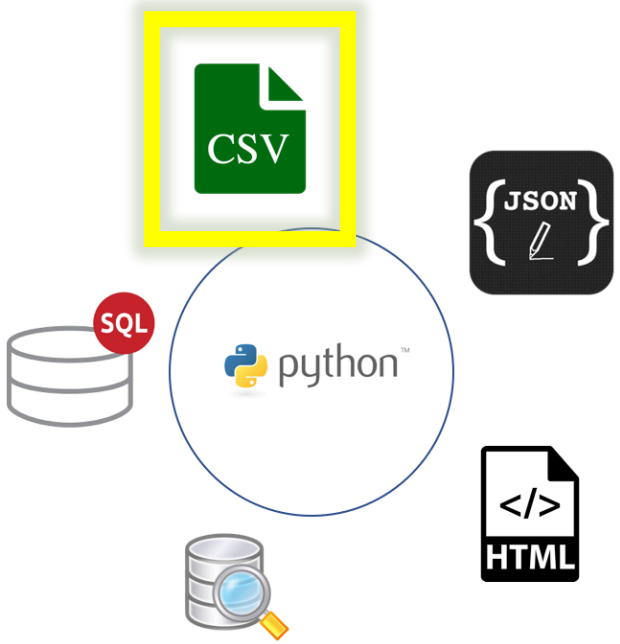
```
In [50]: df.index
Out[50]: Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')

In [51]: df.columns
Out[51]: Index(['one', 'two'], dtype='object')
```

Pandas provides many constructors to create DataFrames!

Data Ingestion

read_csv

- Input : Path to a Comma Separated File
- Output: Pandas DataFrame object containing contents of the file

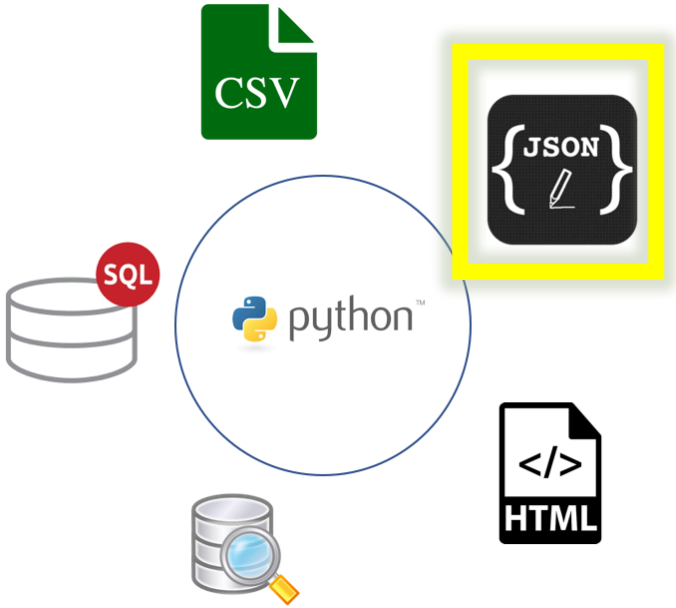


movies.csv

```
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
6,Heat (1995),Action|Crime|Thriller
7,Sabrina (1995),Comedy|Romance
8,Tom and Huck (1995),Adventure|Children
9,Sudden Death (1995),Action
10,GoldenEye (1995),Action|Adventure|Thriller
11,"American President, The (1995)",Comedy|Drama|Romance
12,Dracula: Dead and Loving It (1995),Comedy|Horror
13,Balto (1995),Adventure|Animation|Children
14,Nixon (1995),Drama
15,Cutthroat Island (1995),Action|Adventure|Romance
16,Casino (1995),Crime|Drama
17,Sense and Sensibility (1995),Drama|Romance
18,Four Rooms (1995),Comedy
19,Ace Ventura: When Nature Calls (1995),Comedy
20,Money Train (1995),Action|Comedy|Crime|Drama|Thriller
21,Get Shorty (1995),Comedy|Crime|Thriller
22,Copycat (1995),Crime|Drama|Horror|Mystery|Thriller
23,Assassins (1995),Action|Crime|Thriller
24,Powder (1995),Drama|Sci-Fi
25,Leaving Las Vegas (1995),Drama|Romance
26,Othello (1995),Drama
27,Now and Then (1995),Children|Drama
28,Persuasion (1995),Drama|Romance
29,"City of Lost Children, The (Cité des enfants perdus, La) (1995)",Adventure|Drama|
Fantasy|Mystery|Sci-Fi
```

read_json

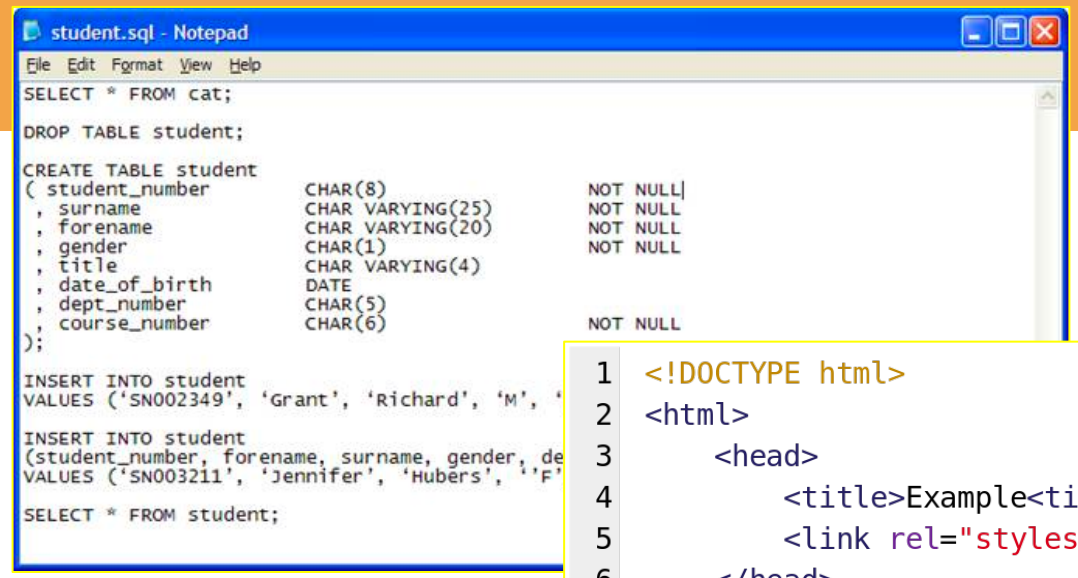
- **Input** : Path to a JSON file or a valid JSON String
- **Output:** Pandas DataFrame or a Series object containing the contents



www.chicagohealthatlas.org/places.json


```
{"community_areas":[{"centroid":"[-87.721558,41.968059]","created_at":"2013-05-28T11:43:08-05:00","geo_type":"Community
Area","geometry":"{\"coordinates\":[[[[-87.704038,41.973552],[-87.703532,41.971863],[-87.702648,41.97034],
[-87.701083,41.968521],[-87.696051,41.963682],[-87.695477,41.962931],[-87.694746,41.961273],[-87.729693,41.960927],
[-87.737851,41.96654],[-87.737902,41.968134],[-87.747722,41.968029],[-87.747502,41.968604],[-87.747851,41.969724],
[-87.747993,41.975341],[-87.733171,41.97548],[-87.733117,41.973651],[-87.731899,41.973627],[-87.731831,41.971845],
[-87.728132,41.971884],[-87.728233,41.975532],[-87.727308,41.975542],[-87.726657,41.975976],[-87.725966,41.976116],
[-87.724442,41.975809],[-87.722663,41.974502],[-87.719723,41.973819],[-87.715666,41.971921],[-87.714026,41.971969],
[-87.712316,41.972443],[-87.711014,41.97396],[-87.708737,41.974675],[-87.707791,41.974668],[-87.706681,41.974426],
[-87.70478,41.973955],[-87.704038,41.973552]]]],\"type\":\"MultiPolygon\"}","id":14,"name":"Albany
Park","slug":"albany_park","updated_at":"2013-05-28T11:43:08-05:00"},{"centroid":"[-87.726322,41.810887]","created_at":"2013-
05-28T11:43:10-05:00","geo_type":"Community Area","geometry":"{\"coordinates\":[[[[-87.714369,41.826041],
[-87.713789,41.806023],[-87.714028,41.805155],[-87.714369,41.804585],[-87.715228,41.803755],[-87.722967,41.800553],
[-87.723373,41.800546],[-87.723365,41.800223],[-87.728208,41.79831],[-87.728168,41.796831],[-87.735512,41.796755],
[-87.735412,41.795464],[-87.737905,41.79529],[-87.738557,41.818706],[-87.714374,41.826321],
[-87.714369,41.826041]]]],\"type\":\"MultiPolygon\"}","id":57,"name":"Archer
Heights","slug":"archer_heights","updated_at":"2013-05-28T11:43:10-05:00"},{"centroid":"
[-87.633975,41.842087]","created_at":"2013-05-28T11:43:10-05:00","geo_type":"Community Area","geometry":"{\"coordinates\":
[[[[-87.629168,41.845557],[-87.629965,41.845545],[-87.629561,41.830971],[-87.628996,41.830975],[-87.628923,41.827679],
[-87.629092,41.82733],[-87.628968,41.823681],[-87.633964,41.823613],[-87.636064,41.82521],[-87.636203,41.827233],
[-87.636404,41.827231],[-87.636576,41.834147],[-87.636532,41.834526],[-87.63589,41.834535],[-87.635932,41.83636],
[-87.636431,41.836354],[-87.636512,41.843643],[-87.638291,41.843617],[-87.638358,41.846183],[-87.639525,41.846374],
[-87.6416,41.846301],[-87.64185,41.847111],[-87.641727,41.847283],[-87.642426,41.84811],[-87.641624,41.848193],
[-87.642248,41.848926],[-87.643233,41.848522],[-87.643999,41.849505],[-87.64264,41.850024],[-87.641148,41.851793],
[-87.639394,41.854285],[-87.637479,41.85533],[-87.636161,41.856292],[-87.635494,41.856938],[-87.635158,41.857722],
[-87.630226,41.857779],[-87.630061,41.852877],[-87.62935,41.852882],
[-87.629168,41.845557]]]],\"type\":\"MultiPolygon\"}","id":34,"name":"Armour
Square","slug":"armour_square","updated_at":"2013-05-28T11:43:10-05:00"},{"centroid":"
[-87.708347,41.74574]","created_at":"2013-05-28T11:43:11-05:00","geo_type":"Community Area","geometry":"{\"coordinates\":
[[[[-87.712548,41.757337],[-87.711005,41.757168],[-87.707432,41.757259],[-87.707447,41.757134],[-87.678588,41.757653],
[-87.678388,41.754791],[-87.678242,41.747447],[-87.678118,41.747273],[-87.678114,41.744296],[-87.67789,41.742543],
[-87.676889,41.740111],[-87.675299,41.738675],[-87.673015,41.735944],[-87.672791,41.735898],[-87.672785,41.735654],
[-87.741067,41.734524],[-87.740933,41.736865],[-87.741496,41.753292],[-87.739119,41.753118],[-87.736582,41.753164],
[-87.736582,41.753041],[-87.732329,41.753321],[-87.73213,41.753165],[-87.722036,41.753374],[-87.722287,41.76147],
[-87.718345,41.759625],[-87.716226,41.758776],[-87.716222,41.758886],[-87.713632,41.757645],
[-87.712548,41.757337]]]],\"type\":\"MultiPolygon\"}","id":70,"name":"Ashburn","slug":"ashburn","updated_at":"2013-05-
28T11:43:11-05:00"},{"centroid":"[-87.656307,41.744196]","created_at":"2013-05-28T11:43:12-05:00","geo_type":"Community
Area","geometry":"{\"coordinates\":[[[[-87.6399,41.756146],[-87.639759,41.750705],[-87.639369,41.750711],
[-87.639485,41.748858],[-87.639097,41.748871],[-87.63919,41.743434],[-87.636477,41.743518],[-87.635946,41.744162],
[-87.634108,41.743498],[-87.633963,41.739474],[-87.634081,41.739145],[-87.633883,41.738207],[-87.63381,41.735948],
[-87.634128,41.735942],[-87.633734,41.728853],[-87.644615,41.728701],[-87.644681,41.729835],[-87.645183,41.729835],
[-87.646433,41.729506],[-87.647492,41.728635],[-87.647951,41.7287],[-87.646513,41.732527],[-87.665387,41.732086],
[-87.665331,41.730237],[-87.668117,41.730201],[-87.673083,41.735657],[-87.672785,41.735654],[-87.672791,41.735898],
[-87.673015,41.735944],[-87.675299,41.738675],[-87.676889,41.740111],[-87.677571,41.741531],[-87.678094,41.743934],
[-87.678588,41.757653],[-87.669692,41.757784],[-87.669697,41.757631],[-87.665475,41.757674],[-87.644235,41.758144],
[-87.644184,41.756088],[-87.6399,41.756146]]]],\"type\":\"MultiPolygon\"}","id":71,"name":"Auburn
Gresham","slug":"auburn_gresham","updated_at":"2013-05-28T11:43:12-05:00"},{"centroid":"
[-87.763116,41.894102]","created_at":"2013-05-28T11:43:12-05:00","geo_type":"Community Area","geometry":"{\"coordinates\":
[[[[-87.789415,41.91751],[-87.787228,41.916722],[-87.785271,41.916452],[-87.766396,41.916721],[-87.763814,41.916626],
```

Image Source: http://www.sqa.org.uk/e-learning/SQLIntro01CD/images/pic024.jpg



read_sql_query

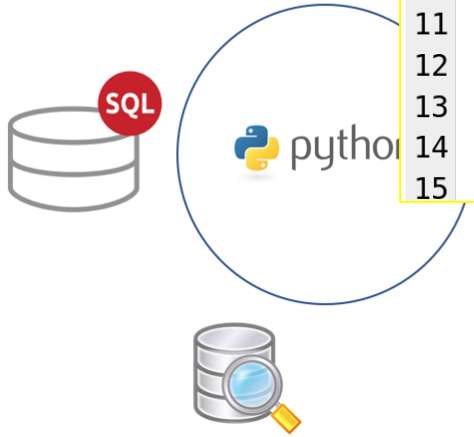
```
<!DOCTYPE html>
<html>
    <head>
        <title>Example<title>
        <link rel="stylesheet" href="sty
    </head>
    <body>
        <h1>
            <a href="/">Header</a>
        </h1>
        <nav>
            <a href="one/">One</a>
            <a href="two/">Two</a>
            <a href="three/">Three</a>
        </nav>
```

read_sql_table

select * from bookstore

ISBN_NO	SHORT_DESC	AUTHOR	PUBLISHER	PRICE
0201703092	The Practical SQL, Fourth Edition	Judith S. Bowman	Addison Wesley	39
0471777781	Professional Ajax	Jeremy McPeak, Joe Fawcett	Wrox	32
0672325764	Sams Teach Yourself XML in 21 Days, Third Edition	Steven Holzner	Sams Publishing	49
0764557599	Professional C#	Simon Robinson and Jay Glynn	Wrox	42
0764579088	Professional JavaScript for Web Developers	Nicholas C. Zakas	Wrox	35
1861002025	Professional Visual Basic 6 Databases	Charles Williams	Wrox	38
1861006314	GDI+ Programming: Creating Custom Controls Using C#	Eric White	Wrox	29

Image Source: http://www.w3processing.com/SQL/images/SQL002.png



read_html

describe()

- Syntax : data_frame.describe()
- Output: Shows summary statistics of the dataframe

```
In [103]: giftcard.describe()
```

Out[103]:

	customer_id	product_parent	star_rating	helpful_votes	total_votes
count	1.483100e+05	1.483100e+05	148310.000000	148310.000000	148310.000000
mean	2.628931e+07	5.406163e+08	4.731333	0.397424	0.490493
std	1.587236e+07	2.661563e+08	0.829255	20.701385	22.823494
min	1.063700e+04	1.100879e+06	1.000000	0.000000	0.000000
25%	1.289732e+07	3.612555e+08	5.000000	0.000000	0.000000
50%	2.499530e+07	4.730483e+08	5.000000	0.000000	0.000000
75%	4.139731e+07	7.754865e+08	5.000000	0.000000	0.000000
max	5.309648e+07	9.992742e+08	5.000000	5987.000000	6323.000000

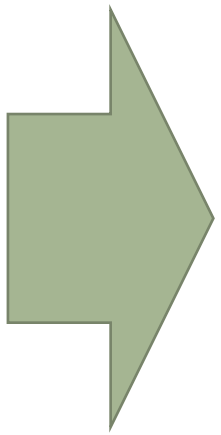
func = min(), max(), mode(), median(), mean(), std()

- The general syntax for calling these functions is
  - **data_frame.func()**
  - Frequently used optional parameter:
    - axis = 0 (rows) or 1 (columns)

```
In [106]: giftcard['star_rating'].max()
Out[106]: 5

In [107]: giftcard['star_rating'].min()
Out[107]: 1

In [108]: giftcard['star_rating'].mean()
Out[108]: 4.731333018677096
```

Real-world data is messy!

- Missing values
- Outliers in the data
- Invalid data (e.g. negative values for age)
- NaN value (np.nan)
- None value



- Replace the value
- Fill gaps forward / backward
- Drop fields
- Interpolation

any() and dropna()

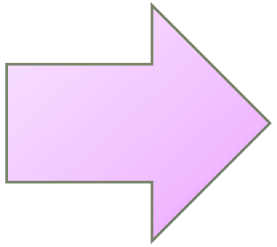
```
In [110]: giftcard.isnull().any()

Out[110]: marketplace          False
          customer_id          False
          review_id            False
          product_id           False
          product_parent       False
          product_title        False
          product_category     False
          star_rating          False
          helpful_votes        False
          total_votes          False
          vine                 False
          verified_purchase    False
          review_headline       True
          review_body           True
          review_date           True
          dtype: bool
```

```
In [112]: giftcard.shape

Out[112]: (148310, 15)

In [113]: giftcard = giftcard.dropna()

In [114]: giftcard.shape

Out[114]: (148304, 15)
```

df.replace()

	0	1
0	-0.349596	-2.017159
1	9999.000000	9999.000000
2	9999.000000	9999.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	9999.000000	9999.000000
6	1.233087	0.720138
7	9999.000000	9999.000000
8	9999.000000	9999.000000
9	9999.000000	9999.000000

9999.0000



```
df=df.replace(9999.0, 0)
df
```

	0	1
0	-0.349596	-2.017159
1	0.000000	0.000000
2	0.000000	0.000000
3	0.113889	0.616122
4	0.014707	-1.731660
5	0.000000	0.000000
6	1.233087	0.720138
7	0.000000	0.000000
8	0.000000	0.000000
9	0.000000	0.000000

0.0000

Fill missing data gaps forward and backward

http://pandas.pydata.org/pandas-docs/stable/missing_data.html

```
df
```

	0	1
0	0.061038	1.339673
1	NaN	NaN
2	1.578293	0.637435
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	-1.145787	0.052887

```
df.fillna(method='ffill')
```

	0	1
0	0.061038	1.339673
1	0.061038	1.339673
2	1.578293	0.637435
3	1.578293	0.637435
4	1.578293	0.637435
5	1.578293	0.637435
6	1.578293	0.637435
7	1.578293	0.637435
8	1.578293	0.637435
9	-1.145787	0.052887

```
df.fillna(method='backfill')
```

	0	1
0	0.061038	1.339673
1	1.578293	0.637435
2	1.578293	0.637435
3	-1.145787	0.052887
4	-1.145787	0.052887
5	-1.145787	0.052887
6	-1.145787	0.052887
7	-1.145787	0.052887
8	-1.145787	0.052887
9	-1.145787	0.052887

df.plot.bar() AND df.plot.hist() AND df.plot()

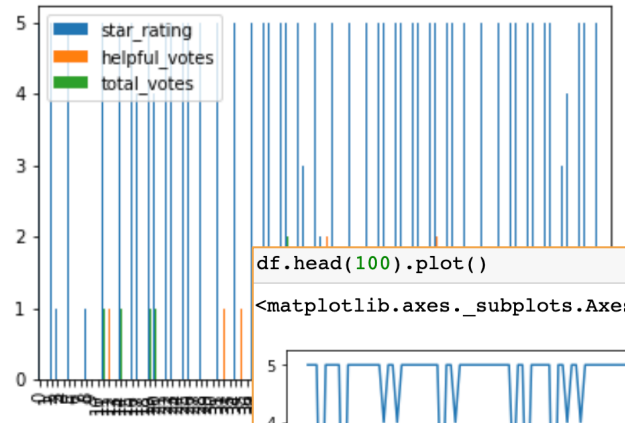
```
In [11]: df = giftcard[['star_rating','helpful_votes','total_votes']]
         df
```

Out[11]:

	star_rating	helpful_votes	total_votes
0	5	0	0
1	5	0	0
2	5	0	0
3	1	0	0
4	5	0	0
5	5	0	0
6	5	0	0
7	5	0	0
8	1	0	0
9	5	0	0
10	5	0	0
11	5	1	1
12	5	1	1
13	5	0	0
14	5	1	1
15	5	0	0
16	5	0	0

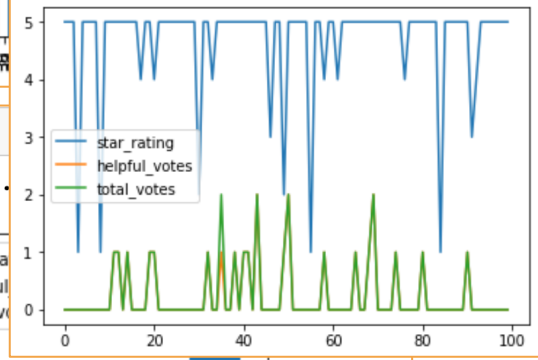
```
In [18]: df.head(100).plot.bar()
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x10d8036d8>
```



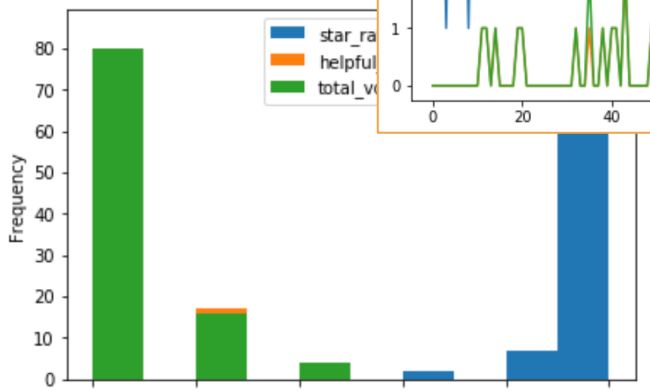

```
df.head(100).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10eb86320>
```




```
df.head(100).plot.hist()
```

```
<matplotlib.axes._subplots.
```



Slice Out Columns

```
In [11]: df = giftcard[['star_rating','helpful_votes','total_votes']]
         df
```

Out[11]:

	star_rating	helpful_votes	total_votes
0	5	0	0
1	5	0	0
2	5	0	0
3	1	0	0
4	5	0	0
5	5	0	0
6	5	0	0
7	5	0	0
8	1	0	0
9	5	0	0
10	5	0	0
11	5	1	1
12	5	1	1
13	5	0	0
14	5	1	1
15	5	0	0
16	5	0	0

Filter Out Rows

```
df_hepful_votes = df[df['helpful_votes'] > 0]
print(df.shape)
print(df_hepful_votes.shape)
```

```
(148304, 3)
(5548, 3)
```

Group By and Aggregate

```
df_helpful_votes
df_helpful_votes.groupby('total_votes').mean
```

total_votes	star_rating	helpful_votes
1	4.066760	1.000000
2	3.463059	1.565036
3	3.106494	2.174026
4	2.843750	2.897321
5	2.602339	3.666667
6	2.184783	4.521739
7	2.250000	5.062500
8	1.760563	5.408451

Delete a Column

```
df
```

	sensor1	sensor2	sensor3	sensor4
0	-0.260156	-1.666998	-0.492616	0.242671
1	-0.762055	-1.114774	0.396817	0.157464
2	-1.263953	-0.562550	-1.670459	2.790434
3	-0.765183	-0.619429	0.316981	0.100477
4	-0.165719	-0.678431	0.485722	0.235925
5	-1.243191	0.494006	0.145171	0.021075
6	0.373786	-0.769120	-0.929956	0.864819
7	-0.081455	0.229613	-2.251816	5.070676
8	-0.536697	1.228345	-1.040728	1.083115
9	0.254220	-0.021794	1.268333	1.608669

```
del df['sensor1']
```

```
df
```

	sensor2	sensor3	sensor4
0	-1.666998	-0.492616	0.242671
1	-1.114774	0.396817	0.157464
2	-0.562550	-1.670459	2.790434
3	-0.619429	0.316981	0.100477
4	-0.678431	0.485722	0.235925
5	0.494006	0.145171	0.021075
6	-0.769120	-0.929956	0.864819
7	0.229613	-2.251816	5.070676
8	1.228345	-1.040728	1.083115
9	-0.021794	1.268333	1.608669

Summary of concepts

- Understand the pandas library and the basic data structures in pandas
- List a few common data ingestion methods in pandas
- Describe a pandas DataFrame
- Clean and explore a DataFrame

On your own...

- Use Yelp review to try and ingest JSON files
- There are many other methods available in Pandas to ingest data:
  http://pandas.pydata.org/pandas-docs/stable/api.html#input-output
- Some other functions that are worth exploring: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats
- There are many other ways to transform missing data:

  http://pandas.pydata.org/pandas-docs/version/0.15.2/missing_data.html#numeric-replacement
- Explore graphs here: http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-plotting
- Explore transformations here : http://pandas.pydata.org/pandas-docs/stable/api.html

Python Data Products

Course 1: Basics

Lecture: Matrix processing and numpy

Learning objectives

In this lecture we will...

- Briefly introduce the numpy library
- Perform simple matrix operations on datasets

Loading data

In [1]:
```
import numpy
import json
```

In [2]:
```
path = "datasets/yelp_data/review.json"
f = open(path)
```

In [3]:
```
dataset = []
```

In [4]:
```
while len(dataset) < 50000:
    dataset.append(json.loads(f.readline()))
```

In [5]:
```
dataset[0]
```

Out[5]:
```
{'business_id': '0W4lkclzZThpx3V65bVgig',
 'cool': 0,
 'date': '2016-05-28',
 'funny': 0,
 'review_id': 'v0i_UHJMo_hPBq9bxWvW4w',
 'stars': 5,
 'text': "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nT
hey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty.
\n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
 'useful': 0,
 'user_id': 'bv2nCi5Qv5vroFiqKGopiw'}
```

Code: Extracting basic statistics from the data

- First let's extract a few simple numerical features from the dataset:

```
In [6]: ratings = [d['stars'] for d in dataset]
```

```
In [7]: cool = [d['cool'] for d in dataset]
```

```
In [8]: funny = [d['funny'] for d in dataset]
```

```
In [9]: useful = [d['useful'] for d in dataset]
```

- These are just lists, but we can convert them to numpy arrays:

```
In [10]: ratings = numpy.array(ratings)
         cool = numpy.array(cool)
         funny = numpy.array(funny)
         useful = numpy.array(useful)
```

```
In [11]: ratings
```

```
Out[11]: array([5, 5, 5, ..., 5, 5, 5])
```

Code: Statistical operations

- For the most part, numpy arrays can be treated much like regular python arrays, though they support a variety of additional operations, such as statistical operations:

```
In [12]: numpy.mean(ratings)
Out[12]: 3.73154

In [13]: numpy.var(ratings)
Out[13]: 1.9213092284000002
```

Code: array type

- We can also compose vectors to build ND-arrays, e.g.:

```
In [14]: numpy.stack([cool, funny, useful])
Out[14]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]])
```

- Once we have an array, we can perform other matrix operations like computing the transpose, e.g. to get a feature matrix (*X*) we might do the following:

```
In [15]: features = numpy.stack([cool, funny, useful]).T
```

Code: matrix type

- Note that with an array, most operations (in particulary multiplication) are overloaded to "elementwise" operations. For many linear algebra routines, it is more convenient to use the "matrix" type instead:

```
In [16]: features = numpy.matrix(features)
```

- This supports operations like standard matrix multiplication:

```
In [17]: features.T * features
Out[17]: matrix([[328903, 219632, 381580],
                 [219632, 213077, 288115],
                 [381580, 288115, 734845]])
```

- Or matrix inverse, etc.

```
In [18]: numpy.linalg.inv(features.T * features)
Out[18]: matrix([[ 1.22273023e-05, -8.55225321e-06, -2.99608975e-06],
                 [-8.55225321e-06,  1.59703874e-05, -1.82070967e-06],
                 [-2.99608975e-06, -1.82070967e-06,  3.63045498e-06]])
```

Code: matrix type

- Finally, numpy overloads primitive operations on matrices, allowing matrices to be used within complex mathematical expressions, in order to perform transformations of our data:

```
In [19]: 2*numpy.sin(features) + 3
Out[19]: matrix([[3., 3., 3.],
                 [3., 3., 3.],
                 [3., 3., 3.],
                 ...,
                 [3., 3., 3.],
                 [3., 3., 3.],
                 [3., 3., 3.]])

In [20]: 2*numpy.sin(features) + 3 > 4
Out[20]: matrix([[False, False, False],
                 [False, False, False],
                 [False, False, False],
                 ...,
                 [False, False, False],
                 [False, False, False],
                 [False, False, False]])
```

Other features...

- **ndarray.shape:** Get the shape of an array
- **reshape:** change the dimensions of an array/matrix
- **arange:** Create an array containing a range of numbers
- **numpy.random:** generate (arrays of) random numbers
- **sum, min, max, etc.:** reduction operations on matrices
- **eye:** identity matrix
- **trace, eig, etc.:** linear algebra operations
- See https://docs.scipy.org/doc/numpy/user/quickstart.html for more

Summary of concepts

- Introduced the numpy library
- Demonstrated basic operations for data manipulation in numpy

On your own...

- Try reading the numerical features from the Amazon data into a numpy array, and compiling basic statistics about them (max, min, avg values, etc.)

Python Data Products

Course 1: Basics

Lecture: Data Visualization

Learning objectives

In this lecture we will...

- Motivate the need for exploratory data analysis
- Explain the common terms that refer to data
- Discuss how plots can be useful in exploring data
- Describe common plotting styles

Exploratory Data Analysis

Goal: To understand your data

Two categories:

- Data visualization
- Summary statistics

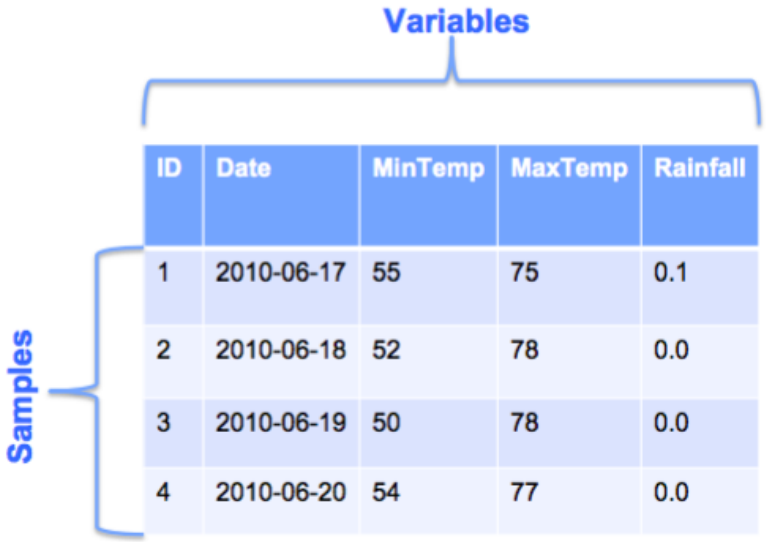
How to refer to data?

Sample

- Instance
- Record
- Row
- Observation
- …

Variable

- Feature
- Field
- Column
- …



Categorical

Qualitative
Nominal

Numeric

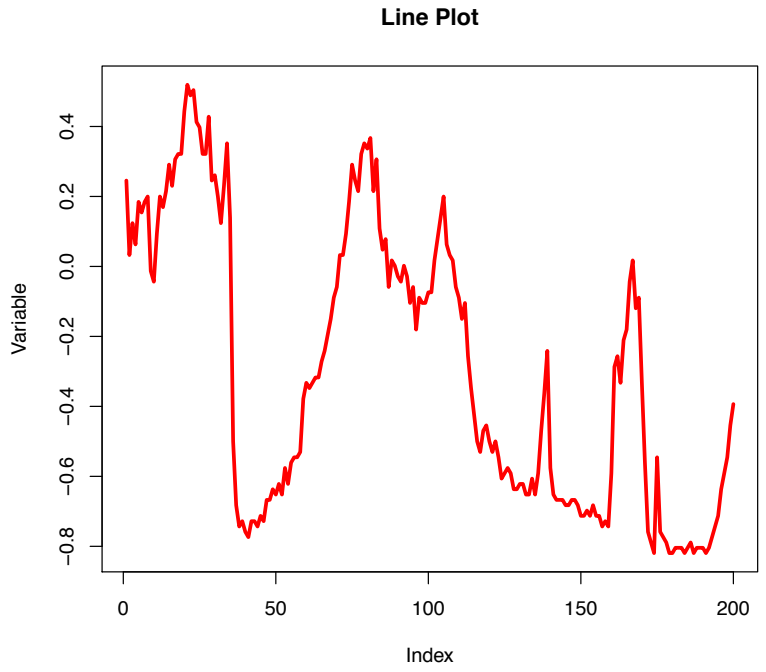
Quantitative

Graphing data

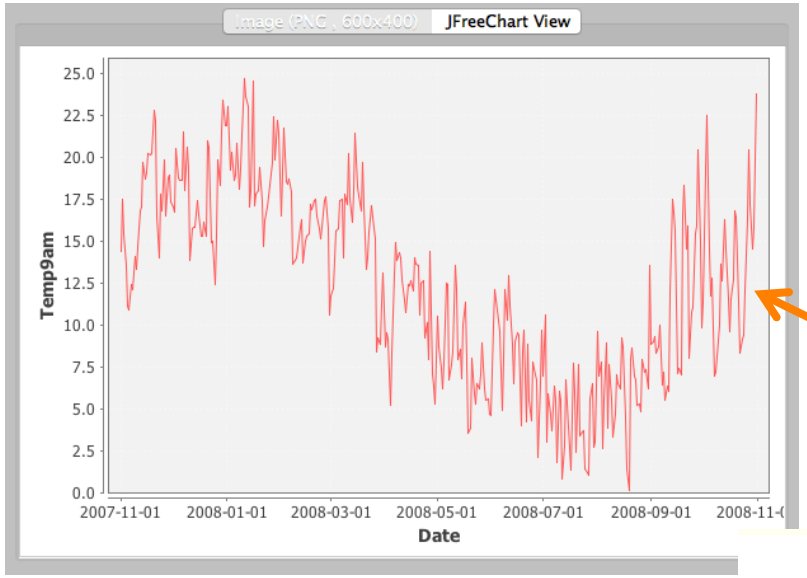
- Line plot
- Histogram
- Scatter plot
- Bar plot
- Box plot

Line Plot

Shows change in data over time



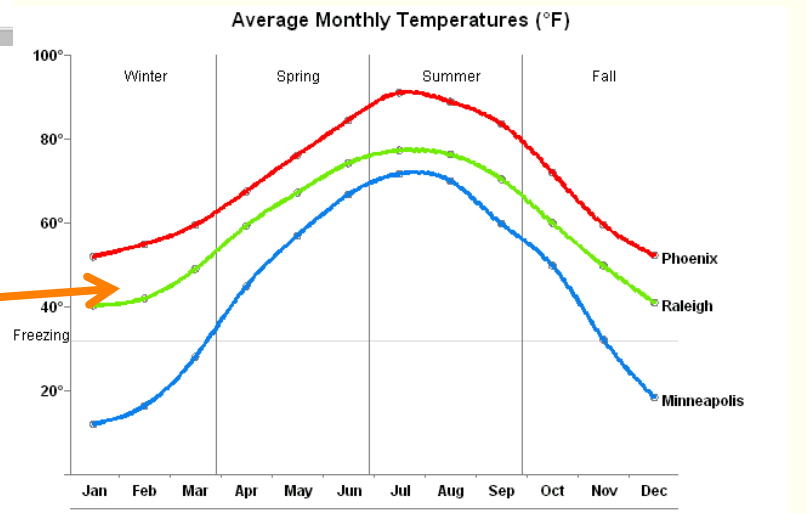
What a Line Plot Shows



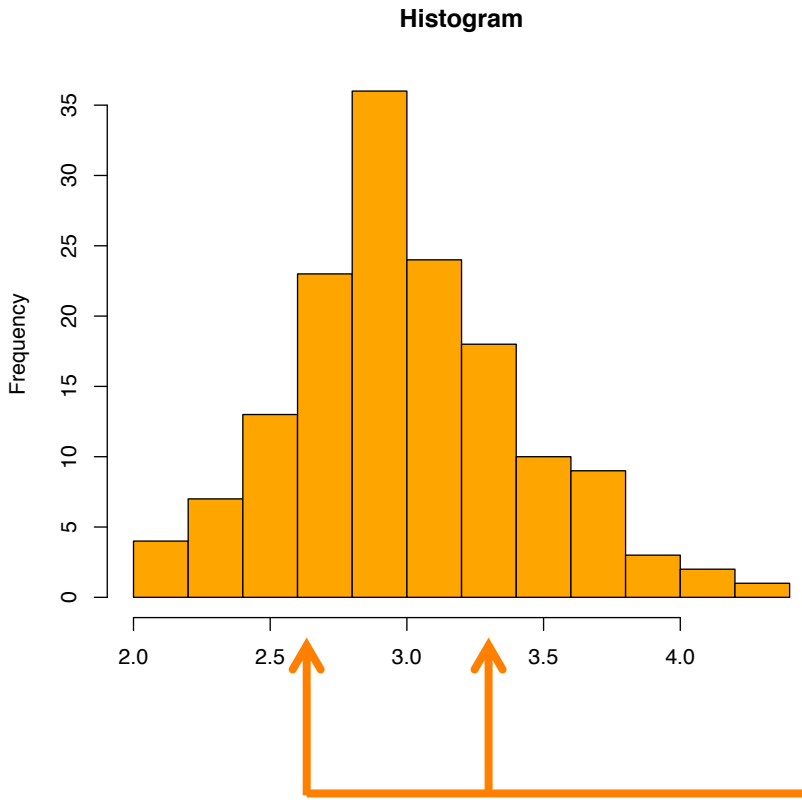
Trend

Cyclical pattern

Compare variables



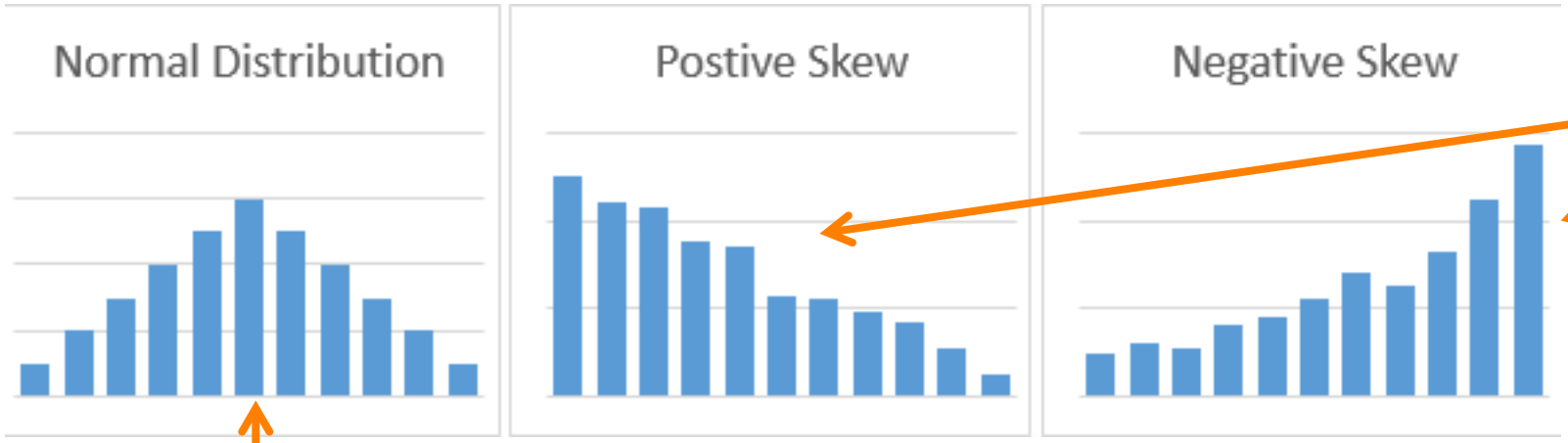
Histogram



Shows distribution of numeric variable

Bins

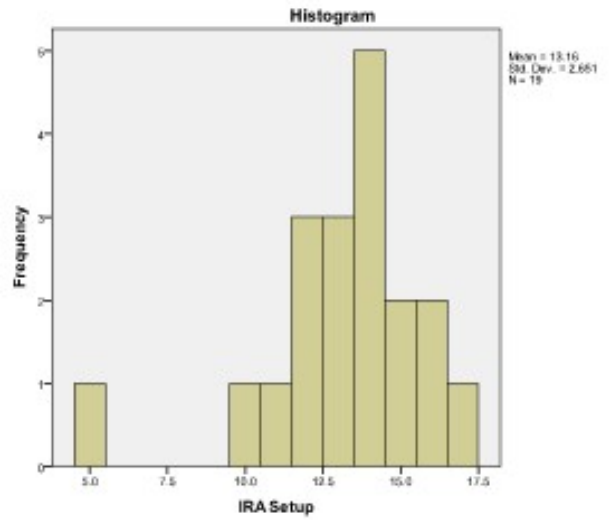
What a Histogram Shows



Skewness

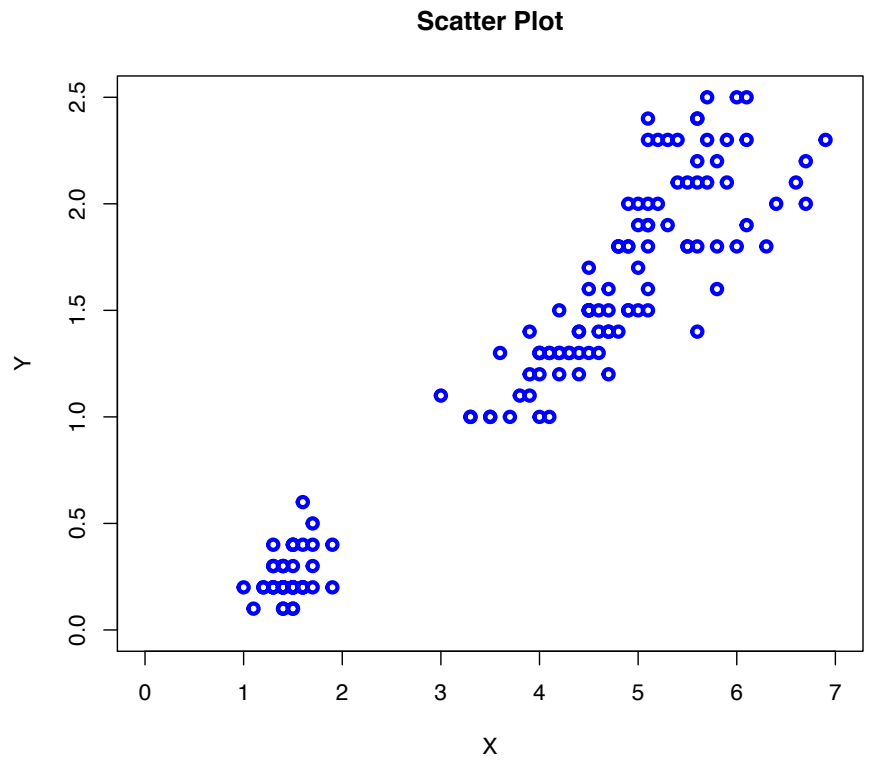
Central Tendency

Outlier



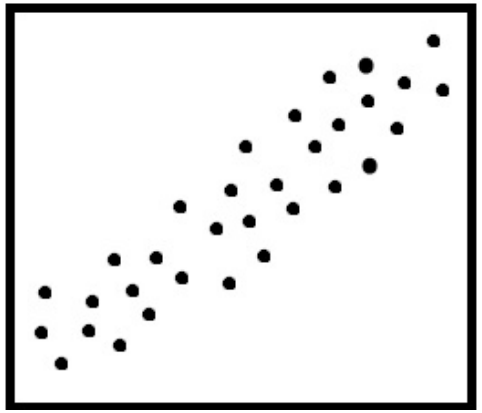
Scatter Plot

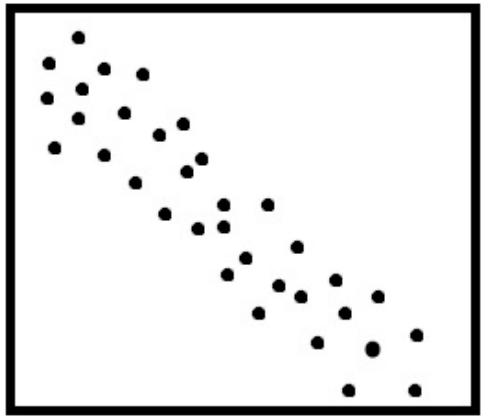
Shows relationship between two variables



What a Scatter Plot Shows

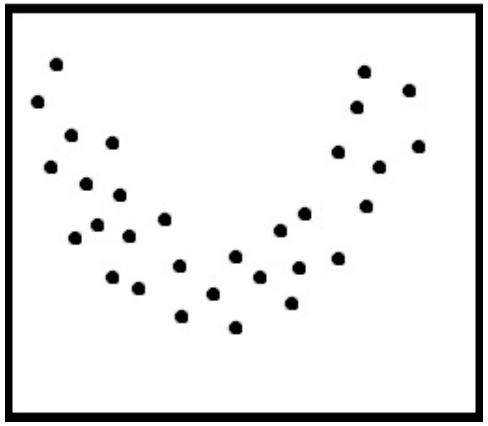
Positive Correlation

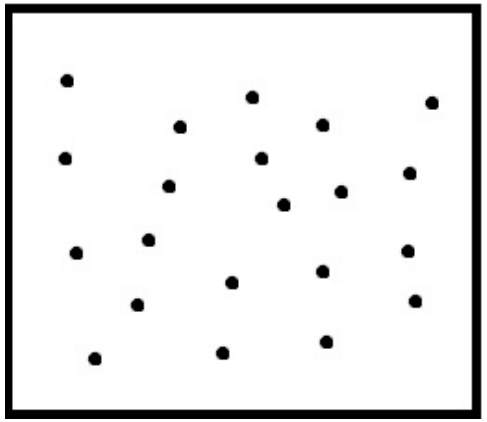




Negative Correlation

Non-Linear Correlation

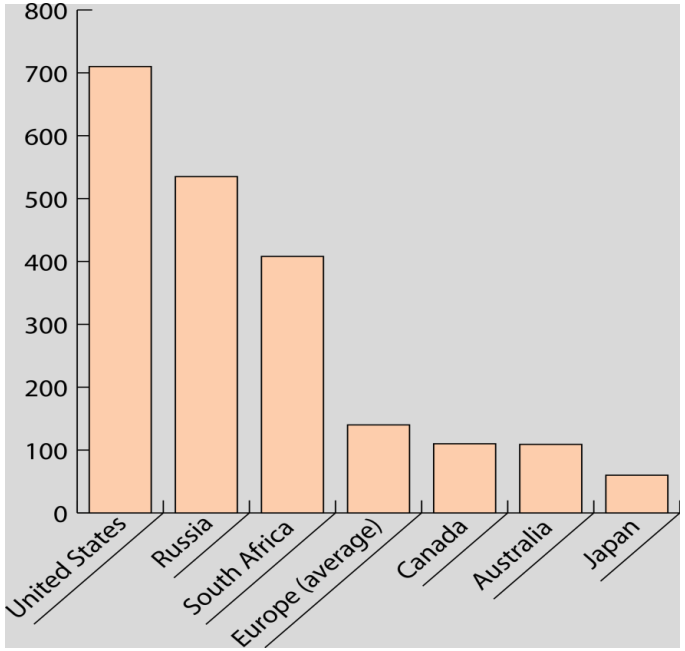




No Correlation

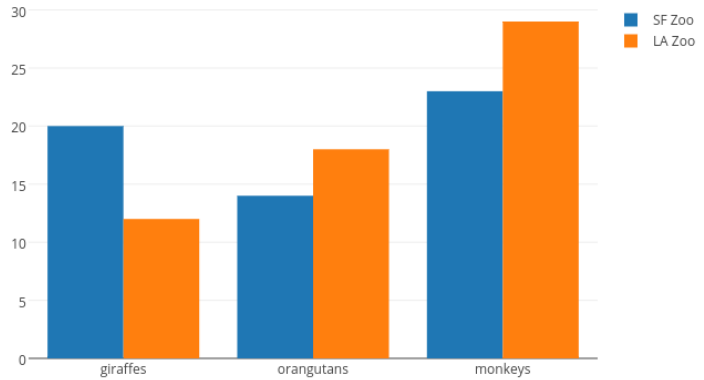
Bar Plot

Shows distribution of categorical variable

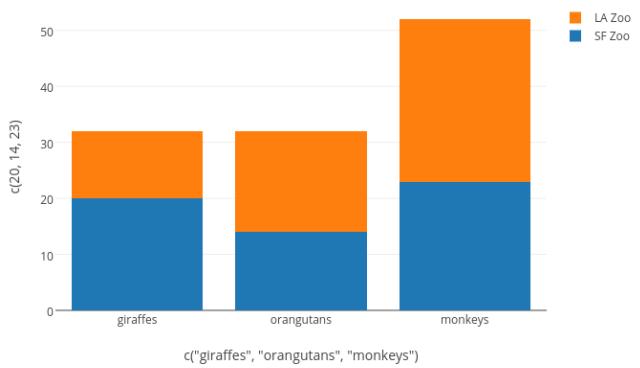


What a Bar Plot Shows

Grouped Bar Chart

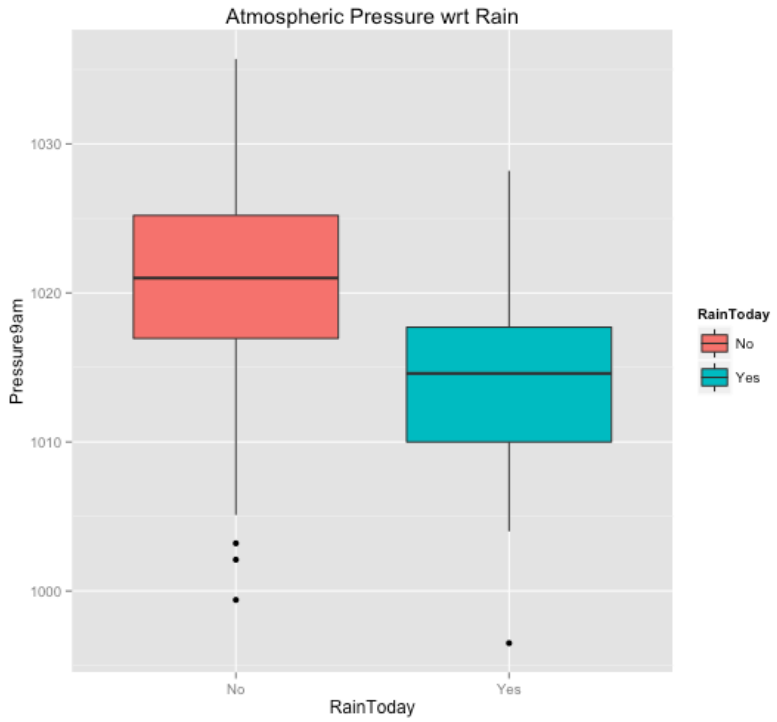


Stacked Bar Chart

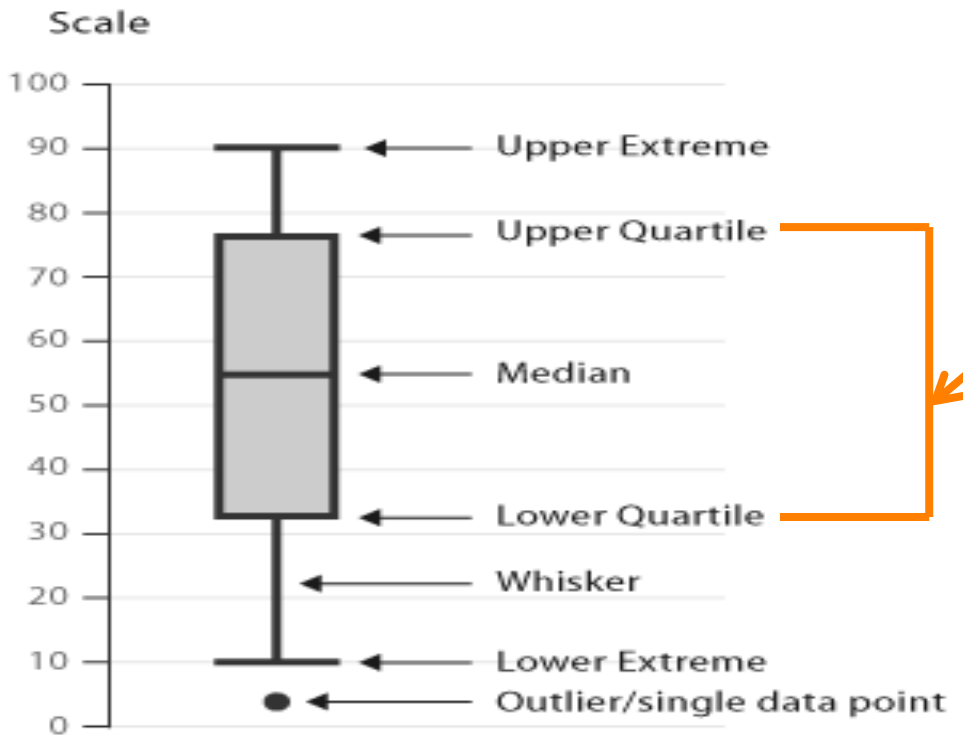


Box Plot

Compares distributions of variables

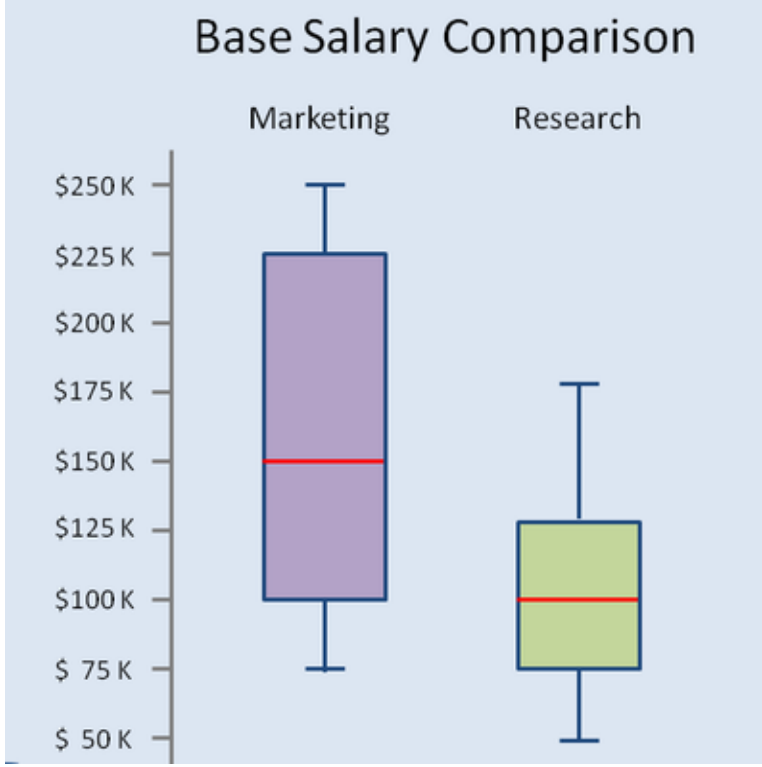


Components of a Box Plot

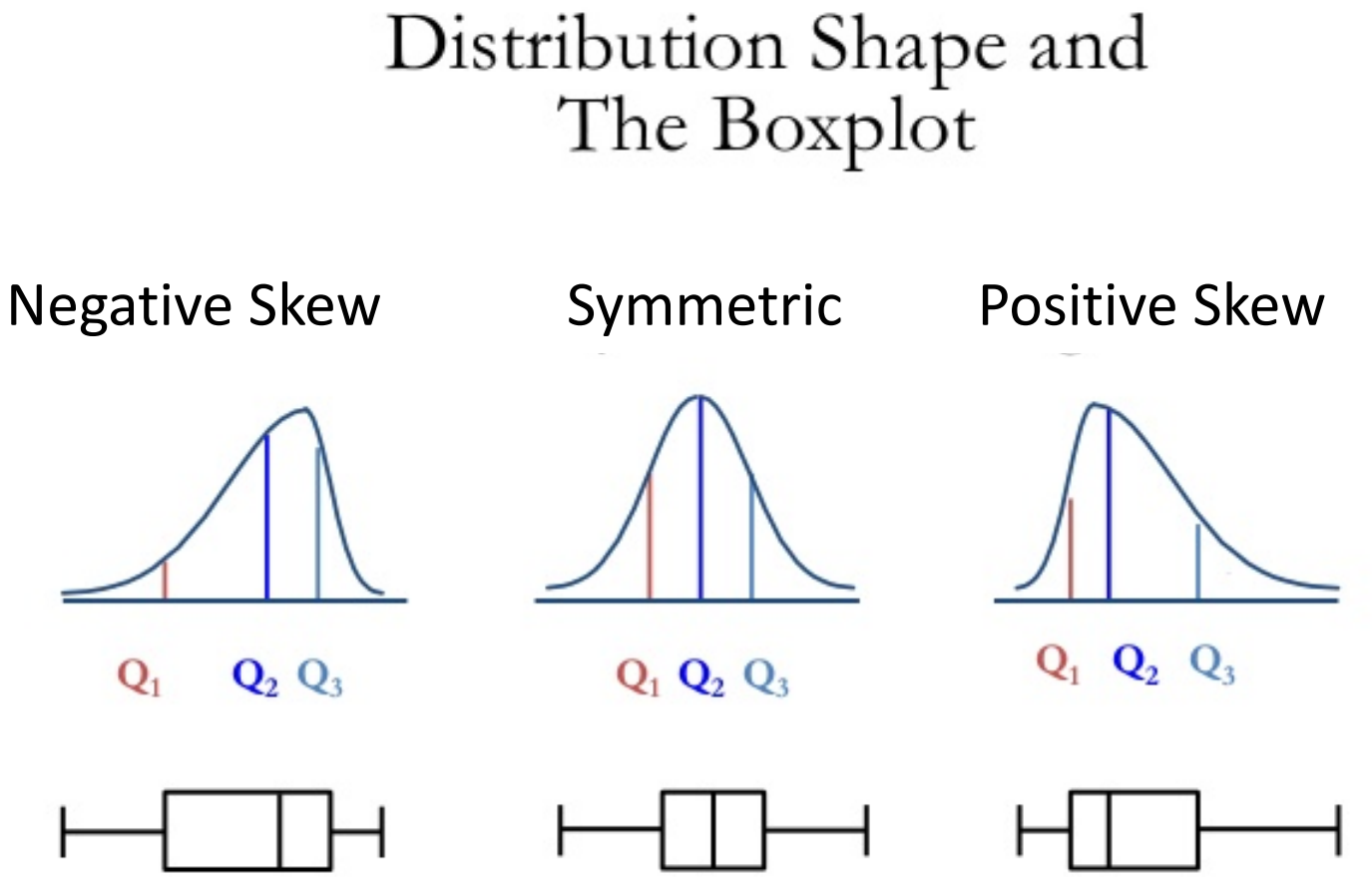


The middle 50% of data are in this region

What a Box Plot Shows



What a Box Plot Shows



Summary of concepts

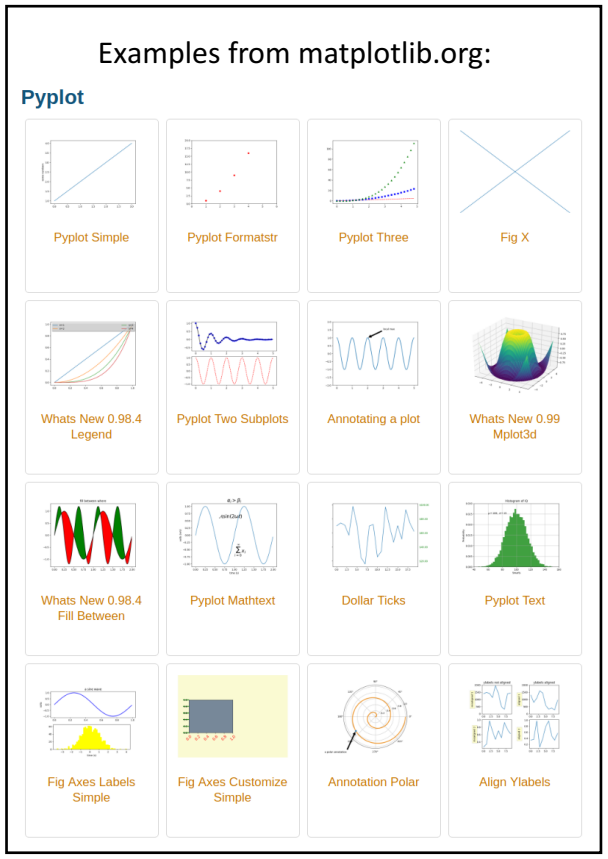
- Provides intuitive way to look at data
- Should be used with summary statistics for data exploration
- Are also useful for communicating results

Python Data Products

Course 1: Basics

Lecture: Introduction to Matplotlib

Learning objectives



In this lecture we will...

- Demonstrate the basics of the **matplotlib** and **pyplot libraries**
- Illustrate the use of matplotlib to **generate simple plots** in python

Matplotlib

Matplotlib is a powerful library that can be used to generate both quick visualizations, as well as publication-quality graphics

- This lecture will introduce some of its most basic functionality (via pyplot), such as bar and line plots
- Examples (with code!) of the types of plots that can be generated are available on https://matplotlib.org/

Code: generating some simple statistics

First, let's quickly compile some statistics from our Yelp data

(see the lecture on Time and Date data)

In [1]:
```
import json
import time
path = "datasets/yelp_data/review.json"
f = open(path, 'r')
```

In [2]:
```
dataset = []
for i in range(50000):
    dataset.append(json.loads(f.readline()))
```

In [3]:
```
datasetWithTimeValues = []
```

In [4]:
```
for d in dataset:
    d['date']
    d['timeStruct'] = time.strptime(d['date'], "%Y-%m-%d")
    d['timeInt'] = time.mktime(d['timeStruct'])
    datasetWithTimeValues.append(d)
```

Code: generating some simple statistics

```
In [5]: from collections import defaultdict

In [6]: weekRatings = defaultdict(list)

In [7]: for d in datasetWithTimeValues:
            day = d['timeStruct'].tm_wday
            weekRatings[day].append(d['stars'])

In [8]: weekAverages = {}

In [9]: for d in weekRatings:
            weekAverages[d] = sum(weekRatings[d]) * 1.0 / len(weekRatings[d])

In [10]: weekAverages

Out[10]: {0: 3.7094594594594597,
          1: 3.715375187253166,
          2: 3.750551876379691,
          3: 3.763665361751486,
          4: 3.7551891653172382,
          5: 3.7231843981953134,
          6: 3.7072147651006713}
```

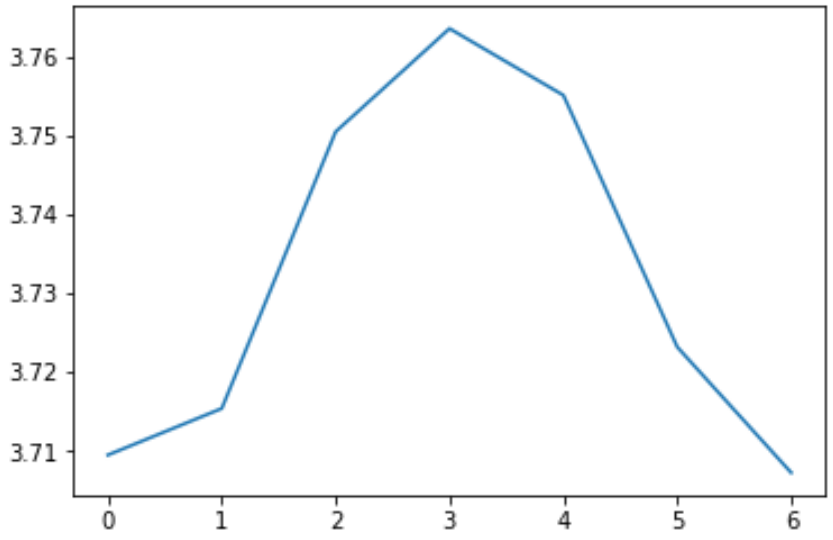
Average ratings per day of week

Code: drawing a simple plot

```
In [11]: X = list(weekAverages.keys())
```

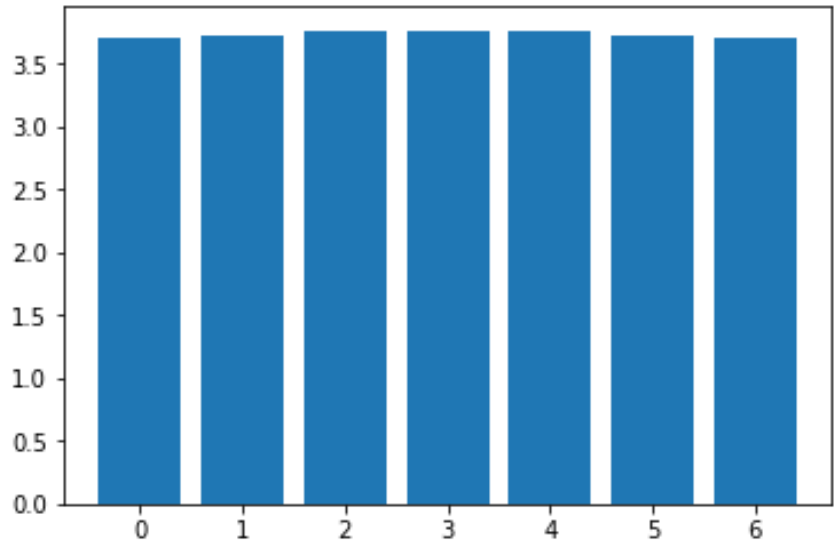
[0,1,2,3,4,5,6]

```
In [12]: Y = [weekAverages[x] for x in X]

In [13]: import matplotlib.pyplot as plt

In [14]: plt.plot(X, Y)
Out[14]: [<matplotlib.lines.Line2D at 0x7fc15a615a20>]
```



Code: bar plots

```
In [15]: plt.bar(X, Y)
Out[15]: <Container object of 7 artists>
```

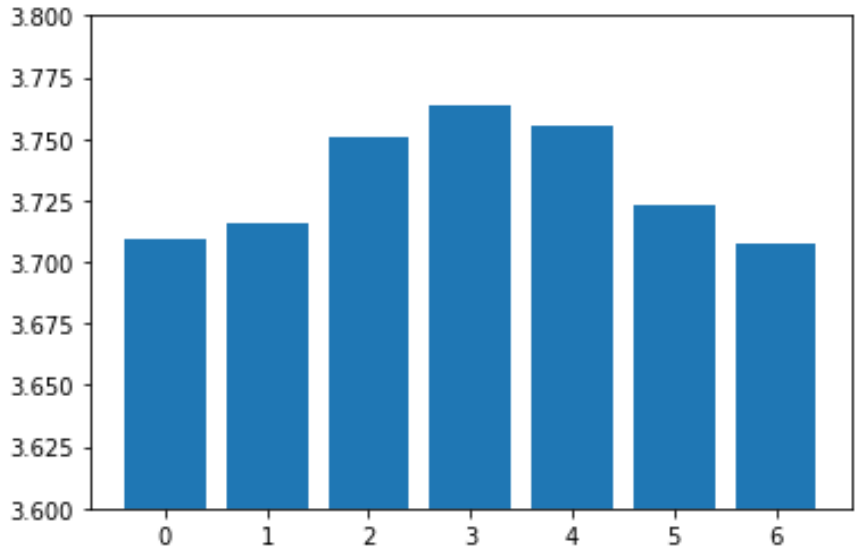


- Looks right, but need to zoom in more to see the detail

Code: bar plots

```
In [16]: plt.ylim(3.6, 3.8)
         plt.bar(X, Y)
```

```
Out[16]: <Container object of 7 artists>
```

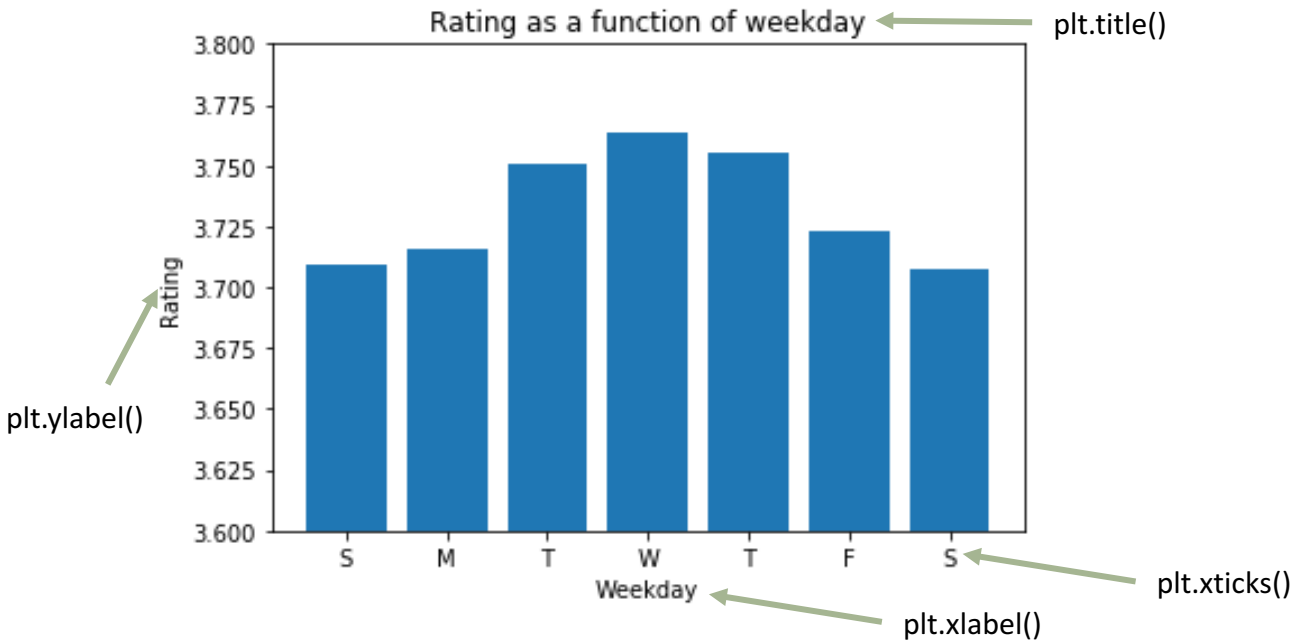


- Next let's add some details

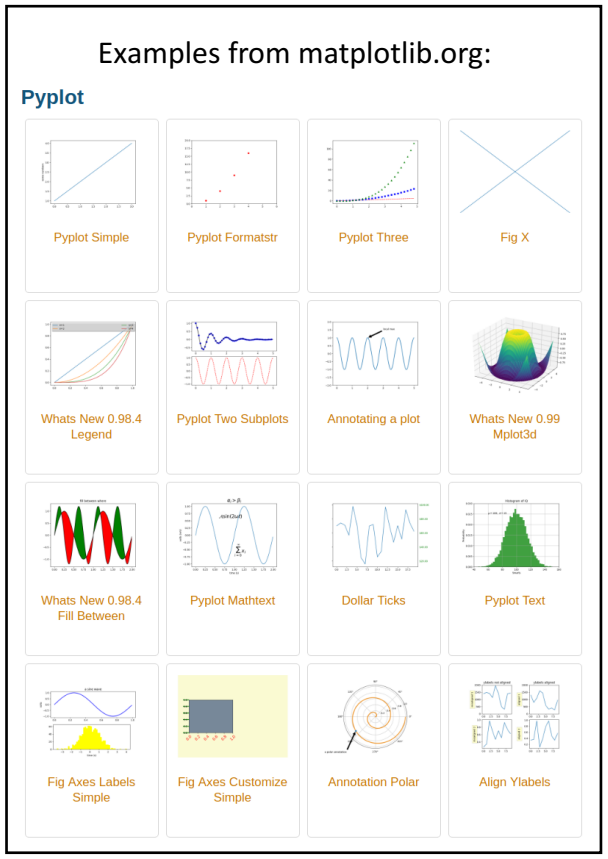
Code: bar plots

```
In [17]: plt.ylim(3.6, 3.8)
         plt.xlabel("Weekday")
         plt.ylabel("Rating")
         plt.xticks([0,1,2,3,4,5,6],['S', 'M', 'T', 'W', 'T', 'F', 'S'])
         plt.title("Rating as a function of weekday")
         plt.bar(X, Y)
```

```
Out[17]: <Container object of 7 artists>
```



Summary of concepts



- Matplotlib is an easy-to-use library to generate plots in Python
- Can be used for both quick visualizations, or to generate publication-quality plots
- Has a lot more functionality than what's covered in this lecture!

On your own...

- Try adding other features to our plot, such as multiple bars, colors, or a legend
- Try generating other types of plots, such as scatterplots

Python Data Products

Course 1: Basics

Lecture: Collecting and parsing Web data with urllib and BeautifulSoup

Learning objectives

In this lecture we will...

- Introduce libraries that can be used to collect data from the Web
- Demonstrate libraries and simple string processing routines that can be used to parse and organize this data

Collecting our own datasets

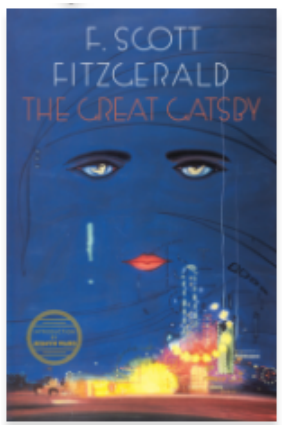
Suppose that we wanted to collect data from a website, but didn't yet have CSV or JSON formatted data

- How could we collect new datasets in machine-readable format?
- What Python libraries could we use to collect data from webpages?
- Once we'd collected (e.g.) raw html data, how could we extract structured information from it?

Collecting our own datasets

E.g. suppose we wanted to collect reviews of "The Great Gatsby" from goodreads.com:

(https://www.goodreads.com/book/show/4671.The_Great_Gatsby)



Want to Read

Rate this book

Open Preview

The Great Gatsby

by F. Scott Fitzgerald, Jake Gyllenhaal (Narrator)

3.9 · Rating details · 3,090,834 Ratings · 56,204 Reviews

A true classic of twentieth-century literature, this edition has been updated by Fitzgerald scholar James L.W. West III to include the author's final revisions and features a note on the composition and text, a personal foreword by Fitzgerald's granddaughter, Eleanor Lanahan—and a new introduction by two-time National Book Award winner Jesmyn Ward.

The Great Gatsby, F. Scot ...more

GET A COPY

Kindle Store $12.99 | Amazon | Stores ▾ | Libraries

Paperback, US / CAN, 180 pages

Published September 2004 by Scribner (first published April 1925)

More Details... edit details

Collecting our own datasets

How could we extract fields including

- The *ID* of the user,
- The *date* of the review
- The *star rating*
- The *text* of the review itself?
- The *shelves* the book belongs to

Code: urllib

Our first step is to extract the html code of the webpage into a python string. This can be done using **urllib:**

```
In [1]: from urllib.request import urlopen

In [2]: f = urlopen("https://www.goodreads.com/book/show/4671.The_Great_Gatsby")

In [3]: html = str(f.read())

In [4]: html

Out[4]: 'b\'<!DOCTYPE html>\\n<html class="desktop\\n">\\n\\n\\n<head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# good_reads: http://ogp.me/ns/fb/good_reads#">\\n  <title>\\nThe Great Gatsby by F. Scott Fitzgerald\\n</title>\\n\\n\\n    <script type="text/javascript"> var ue_t0=window.ue_t0||+new Date();\\n </script>\\n  <script type="text/javascript">\\n    (function(e){var c=e;var a=c.ue||{};a.main_scope="mainscopecsm";a.q=[];a.t0=c.ue_t0||+new Date();a.d=g;function g(h){return +new Date()-(h?0:a.t0)}function d(h){return function(){a.q.push({n:h,a:arguments,t:a.d()})}}function b(m,l,h,j,i){var k={m:m,f:l,l:h,c:""+j,err:i,fromOnError:1,args:arguments};c.ueLogError(k);return false}b.skipTrace=1;e.onerror=b;function f(){c.uex("ld")}if(e.addEventListener){e.addEventListener("load",f,false)}else{if(e.attachEvent){e.attachEvent("onload",f)}}a.tag=d("tag");a.log=d("log");a.reset=d("rst");c.ue_csm=c;c.ue=a;c.ueLogError=d("err");c.ues=d("ues");c.uet=d("uet");c.uex=d("uex");c.uet("ue")})(window);(function(e,d){var a=e.ue||{};function c(g){if(!g){return}var f=d.head||d.getElementsByTagName("head")[0]||d.documentElement,h=d.createElement("script");h.async="async";h.src=g;f.insertBefore(h,f.firstChild)}function b(){var k=e.ue_cdn||"z-ecx.images-amazon.com",g=e.ue_cdns||"images-na.ssl-images-amazon.com",j="/images/G/01/csminstrumentation/",h=e.ue_file|
```

Note: acts like a file object once opened

Note: url of "The Great Gatsby" reviews

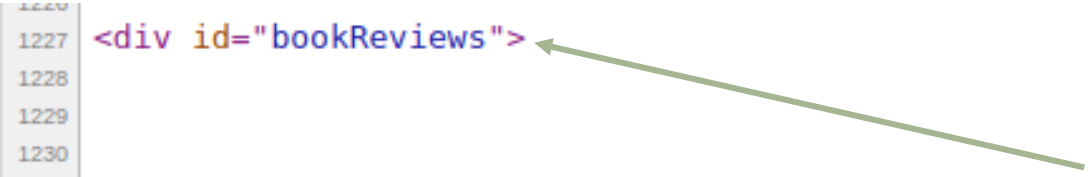
Reading the html data

This isn't very nice to look at, it can be easier to read in a browser or a text editor (which preserves formatting):

```
<!DOCTYPE html>
<html class="desktop
">


<head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# good_reads: http://ogp.me/ns/fb/good_reads#">
  <title>
The Great Gatsby by F. Scott Fitzgerald
</title>


    <script type="text/javascript"> var ue_t0=window.ue_t0||+new Date();
 </script>
  <script type="text/javascript">
    (function(e){var c=e;var a=c.ue||{};a.main_scope="mainscopecsm";a.q=[];a.t0=c.ue_t0||+new Date();a.d=g;fu
{m:m,f:l,l:h,c:""+j,err:i,fromOnError:1,args:arguments};c.ueLogError(k);return false}b.skipTrace=1;e.onerror=
{e.attachEvent("onload",f)}}a.tag=d("tag");a.log=d("log");a.reset=d("rst");c.ue_csm=c;c.ue=a;c.ueLogError=d("
f=d.head||d.getElementsByTagName("head")[0]||d.documentElement,h=d.createElement("script");h.async="async";h.
amazon.com",j="/images/G/01/csminstrumentation/",h=e.ue_file||"ue-full-11e51f253e8ad9d145f4ed644b40f692._V1_.
1:0}i=f?"https://":"http://";i+=f?g:k;i+=j;i+=h;c(i)}if(!e.ue_inline){if(a.loadUEFull){a.loadUEFull()}else{b(
```

Reading the html data

To extract review data, we'll need to look for the part of the html code which contains the reviews:

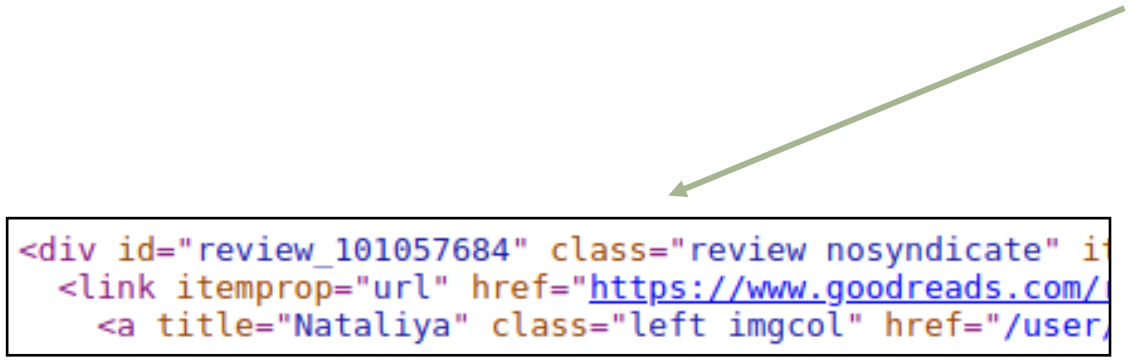


Here it is (over 1000 lines into the page!)

```
1227 <div id="bookReviews">
1228
1229
1230
1231
1232
1233   <div class="friendReviews elementListBrown" >
1234     <div class="section firstReview">
1235
1236
1237 <div id="review_101057684" class="review nosyndicate" itemprop="reviews" itemscope itemtype="http://schema.org/Review">
1238   <link itemprop="url" href="https://www.goodreads.com/review/show/101057684" />
1239     <a title="Nataliya" class="left imgcol" href="/user/show/3672777-nataliya"><img alt="Nataliya" src="https://images.gr-asse
1240
1241   <div class="left bodycol">
1242     <div class="reviewHeader uitext stacked">
1243         <a class="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>
1244
1245       <span itemprop="author" itemscope itemtype="http://schema.org/Person">
1246         <a title="Nataliya" class="user" itemprop="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>
1247       </span>
1248
```

Reading the html data

To extract review data, we'll need to look for the part of the html code which contains the reviews:



- Note that each individual review starts with a block containing the text "<div id="review_..."
- We can collect all reviews by looking for instances of this text

Code: string.split()

To split the page into individual reviews, we can use the string.split() operator. Recall that we saw this earlier when reading csv files:

```
In [5]: reviews = html.split('<div id="review_')[1:]

In [6]: len(reviews)
Out[6]: 30

In [7]: reviews[0]
Out[7]: '101057684" class="review nosyndicate" itemprop="reviews" itemscope itemtype="http://schema.org/Review">\\n  <link i
temprop="url" href="https://www.goodreads.com/review/show/101057684" />\\n    <a title="Nataliya" class="left imgco
l" href="/user/show/3672777-nataliya"><img alt="Nataliya" src="https://images.gr-assets.com/users/1395089173p2/36727
77.jpg" /></a>\\n\\n  <div class="left bodycol">\\n    <div class="reviewHeader uitext stacked">\\n        <a class
="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>\\n\\n      <span
itemprop="author" itemscope itemtype="http://schema.org/Person">\\n        <a title="Nataliya" class="user" itemprop
="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>\\n      </span>\\n\\n        rated it\\n
   <span class=" staticStars" title="it was amazing"><span size="15x15" class="staticStar p10">it was amazing</span>
<span size="15x15" class="staticStar p10"></span><span size="15x15" class="staticStar p10"></span><span size="15x15"
class="staticStar p10"></span><span size="15x15" class="staticStar p10"></span></span>\\n\\n\\n\\n        \\n\\n\\n
```

Note: Ignore the first block, which contains everything *before* the first review

Note: the page contains 30 reviews total

Code: parsing the review contents

Next we have to write a method to parse individual reviews (i.e., given the text of one review, extract formatted fields into a dictionary)

In [8]:
```
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
        for s in shelfBlock.split('shelf=')[1:]:
            shelves.append(s.split('"')[0])
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">')[1].split('</div')[0]
    d['reviewBlock'] = reviewBlock
    return d
```

Code: parsing the review contents

Let's look at it line-by-line:

```
In [8]: def parseReview(review):
            d = {}
```

- We start by building an empty dictionary
- We'll use this to build a *structured* version of the review

Code: parsing the review contents

Let's look at it line-by-line:

Note: Two splits: everything *after* the first quote, and *before* the second quote

- The next line is more complex:

```
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
```

- We made this line by noticing that the stars appear in the html inside a span with class " staticStars":

```
1248
1249        rated it
1250        <span class=" staticStars" title="it was amazing"><span size="15x15" class="staticStar p10">it was amazing<
p10"></span><span size="15x15" class="staticStar p10"></span></span>
1251
```

- Our "split" command then extracts everything inside the "title" quotes

Code: parsing the review contents

Let's look at it line-by-line:

- The following two lines operate in the same way:

Note: Everything between the two brackets of this "<a" element

```
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
```

- Again we did this by noting that the "date" and "user" fields appear inside certain html elements:

```
1240
1241  <div class="left bodycol">
1242    <div class="reviewHeader uitext stacked">
1243        <a class="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>
1244
1245      <span itemprop="author" itemscope itemtype="http://schema.org/Person">
1246        <a title="Nataliya" class="user" itemprop="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>
1247      </span>
1248
```

Code: parsing the review contents

Let's look at it line-by-line:

- Next we extract the "shelves" the book belongs to
- This follows the same idea, but in a "for" loop since there can be many shelves per book:

```
shelves = []
try:
    shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
    for s in shelfBlock.split('shelf=')[1:]:
        shelves.append(s.split('"')[0])
    d['shelves'] = shelves
except Exception as e:
    pass
```

Note: Everything inside a particular <div

- Here we use a try/except block since this text will be missing for users who didn't add the book to any shelves

Code: parsing the review contents

Next let's extract the review contents:

In [8]:
```
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
        for s in shelfBlock.split('shelf=')[1:]:
            shelves.append(s.split('"')[0])
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">')[1].split('</div')[0]
    d['reviewBlock'] = reviewBlock
    return d
```

Code: parsing the review contents

Now let's look at the results:

```
In [9]: reviewDict = [parseReview(r) for r in reviews]

In [10]: reviewDict[0]

Out[10]: {'date': 'May 02, 2010',
 'reviewBlock': '\\n                <span id="reviewTextContainer101057684" class="readable"\\n                >\\n
      \\n<span id="freeTextContainer7513160808421149349"><br>Oh Gatsby, you old sport, you poor semi-delusionally h
opeful dreamer with \\\'<                                                    i>\\\', focusing your whole self and
soul on that elusive mone                                                    you are *this* close to it. <br><br
><img src="https://i.gr-a                                                    dimages/1380334543i/693798._SX540_.j
pg" width="400" class="gr                                                    eam with the passion and courage few
possess - and the tragedy                                                    hat was even more wrong - and deadl
y.</b> <br><br>Just like                                                     h id="freeText7513160808421149349" s
tyle="display:none"><br>O                                                    eful dreamer with \\\'<i>some height
ened sensitivity to the p                                                    oul on that elusive money-colored gr
een light - a dream that                                                     <img src="https://i.gr-assets.com/im
ages/S/compressed.photo.g                                                    g" width="400" class="gr-hostedUserI
mg"><br><br><b>Jay Gatsby                                                    ossess - and the tragedy was that i
t was a wrong dream colli                                                    .</b> <br><br>Just like the Great Ho
udini - the association t                                                    ed in illusions and escape. Except e
ven the power of most cou                                                    e the world, our past, and ourselve
s, giving rise to one of the most famous closing lines of a novel.<blockquote>\\n  <i>\\\'Gatsby believed in the gre
en light, the orgastic future that year by year recedes before us. It eluded us then, but that\\xe2\\x80\\x99s no ma
tter \\xe2\\x80\\x94 to-morrow we will run faster, stretch out our arms farther...  And one fine morning \\xe2\\x80
```

- Looks okay, but the review block itself still contains embedded html (e.g. images etc.)
- How can we extract just the text part of the review?

The BeautifulSoup library

Extracting the text contents from the html review block would be extremely difficult, as we'd essentially have to write a html parser to capture all of the edge cases

Instead, we can use an existing library to parse the html contents:
BeautifulSoup

Code: parsing with BeautifulSoup

BeautifulSoup will build an element tree from the html passed to it. For the moment, we'll just use it to extract the text from a html block

```
In [11]: from bs4 import BeautifulSoup

In [12]: soup = BeautifulSoup(reviewDict[0]['reviewBlock'])

In [13]: soup.text

Out[13]: "\\n              \\n            \\nOh Gatsby, you old sport, you poor semi-delusionally hopeful dreamer with \\'som
e heightened sensitivity to the promises of life\\', focusing your whole self and soul on that elusive money-colored
green light - a dream that shatters just when you are *this* close to it. Jay Gatsby, who dreamed a dream with the p
assion and courage few possess - and the tragedy was that it was a wrong dream colliding with reality that was even
more wrong - and deadly. Just like the Great Houdini - the association the\\n  Oh Gatsby, you old sport, you poor se
mi-delusionally hopeful dreamer with \\'some heightened sensitivity to the promises of life\\', focusing your whole
self and soul on that elusive money-colored green light - a dream that shatters just when you are *this* close to i
t. Jay Gatsby, who dreamed a dream with the passion and courage few possess - and the tragedy was that it was a wron
g dream colliding with reality that was even more wrong - and deadly. Just like the Great Houdini - the association
the title of this book so easily invokes - you specialized in illusions and escape. Except even the power of most co
urageous dreamers can be quite helpless to allow us escape the world, our past, and ourselves, giving rise to one of
```

The BeautifulSoup library

In principle we could have used BeautifulSoup to extract *all* of the elements from the webpage

However, for simple page structures, navigating the html elements is not (necessarily) easier than using primitive string operations

Summary of concepts

- Introduced approaches that can be used to collect data from the web, and convert it to a structured format
- Introduced the urllib and BeautifulSoup libraries

On your own...

- Try using the same techniques to extract a page of reviews from another review website
- But, if you want to collect a large number of reviews, **make sure to check the website's terms of service first!**

Python Data Products

Course 1: Basics

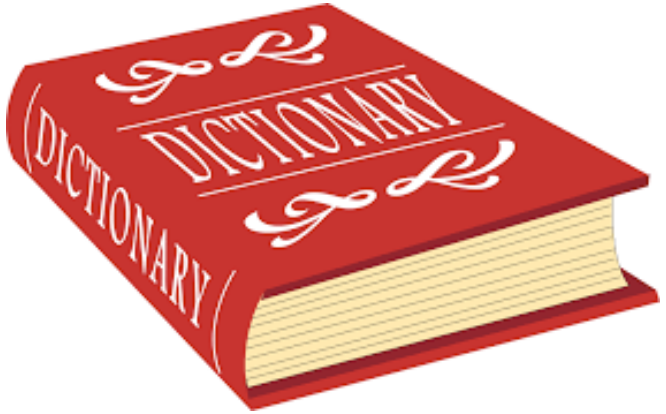
Lecture: Developing a Data Product Strategy

Learning objectives

In this lecture we will...

- Describe main steps to iteratively define a data product strategy that is relevant to you or your organization

What is a strategy?



"a **plan** of **action** or **policy** designed to achieve an overall **aim**"

Parts of a Strategy

- Aim
- Policy
- Plan
- Action

Understand business objectives

- Start with the business objectives
- Educate yourself
- Identify the opportunity
- Analyze gaps
- Prioritize actions

Set Short-term and Long-term Objectives

- Short-term
  - Start with limited data, a small team, and prioritize
  - Get organizational buy-in
    - Commitment
    - Sponsorship
    - Communication

- Long-term
  - Create a roadmap of product features
  - Build your data science team

Open a mini Idea Lab

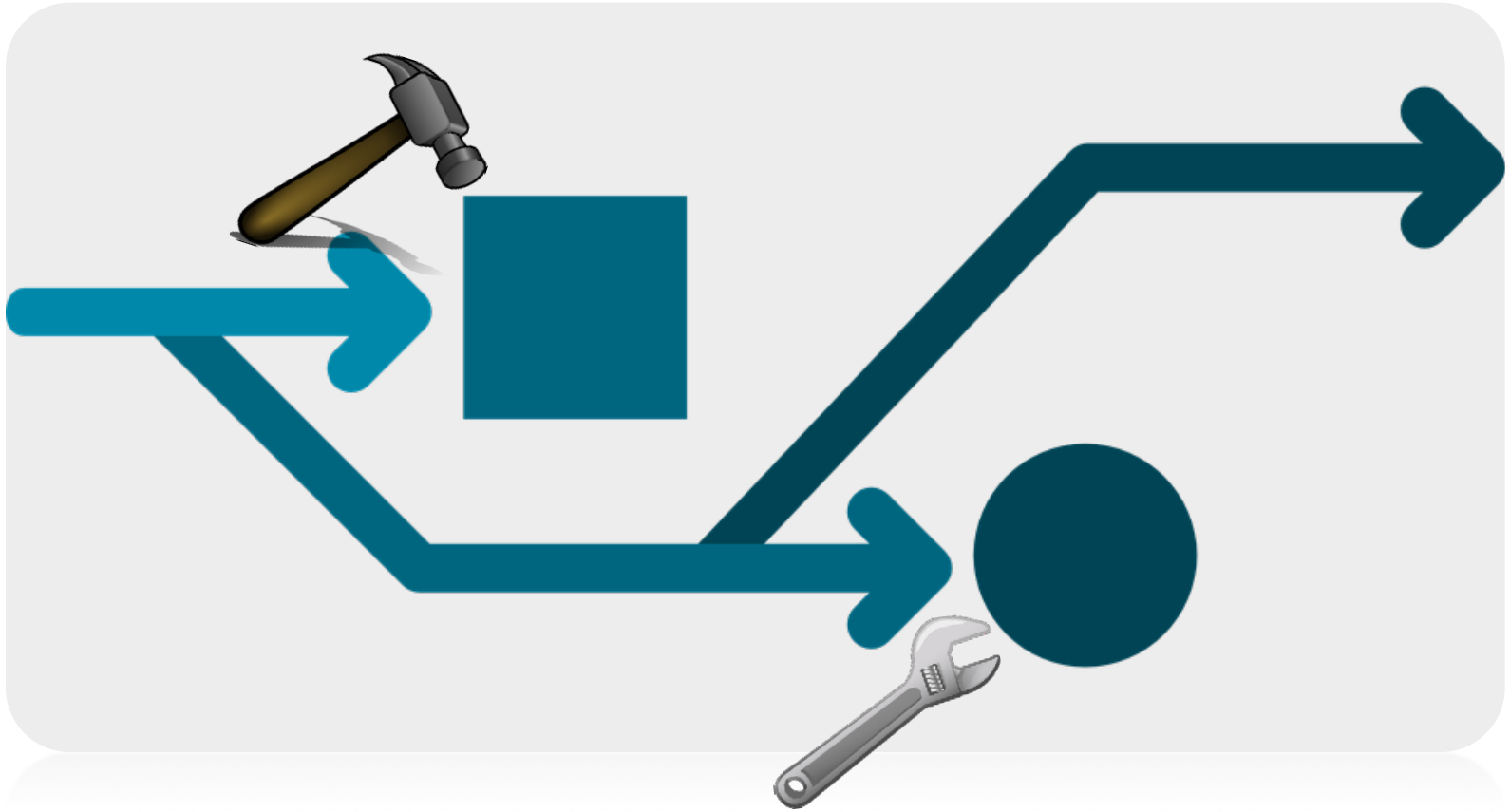
Create a data-oriented culture

- Remove barriers to data access
- No data silos
- Data sharing mindset
- Foster collaborations

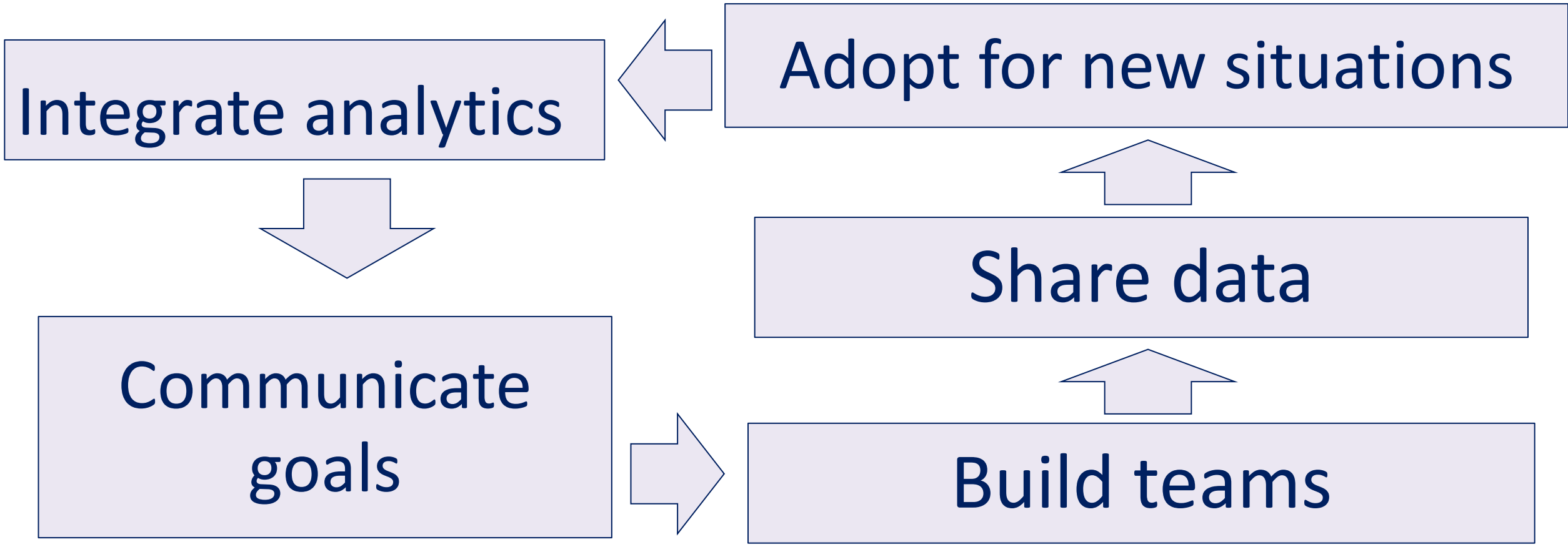
Define the data policies related to your product

- Privacy and lifetime
- Curation and quality
- Interoperability and regulation

Adopt and align through iterations



Summary of concepts



Python Data Products

Course 1: Basics

Lecture: Course Summary

Course summary: **Week 1**

In this course we covered...

- Introductions
- What are data products and where do they appear in our everyday lives?

Course summary: **Week 2**

In this course we covered...

- Basics of Python and Data Ingestion: JSON and CSV
- Described some datasets (Amazon and Yelp) to be used throughout this Specialization


```
{'business_id': 'FYWN1wneV18bWNgQjJ2GNg', 'attributes':
{'BusinessAcceptsCreditCards': True, 'AcceptsInsurance': True,
'ByAppointmentOnly': True}, 'longitude': -111.9785992, 'state':
'AZ', 'address': '4855 E Warner Rd, Ste B9', 'neighborhood': '',
'city': 'Ahwatukee', 'hours': {'Tuesday': '7:30-17:00',
'Wednesday': '7:30-17:00', 'Thursday': '7:30-17:00', 'Friday':
'7:30-17:00', 'Monday': '7:30-17:00'}, 'postal_code': '85044',
'review_count': 22, 'stars': 4.0, 'categories': ['Dentists',
'General Dentistry', 'Health & Medical', 'Oral Surgeons',
'Cosmetic Dentists', 'Orthodontists'], 'is_open': 1, 'name':
'Dental by Design', 'latitude': 33.3306902}
```

Course summary: **Week 3**

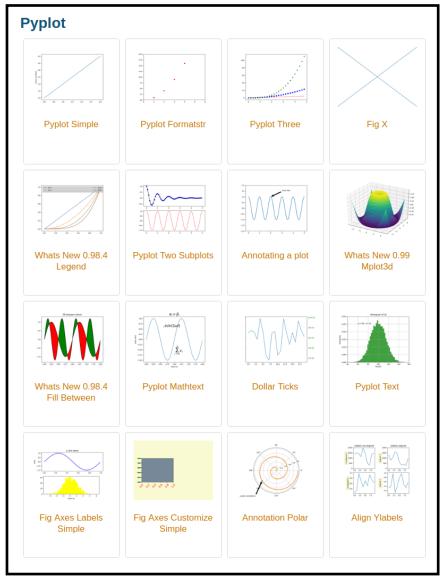
In this course we covered...

- Basics of exploratory data analysis
- Data filtering and cleaning
- Extracting statistics from datasets
- Dealing with structured and semi-structured data: text and strings, time and date processing

Course summary: **Week 4**

In this course we covered...

- Data visualization: Plotting in *matplotlib*
- Data crawling and collection
- Developing data product strategies



Course summary

By now you should be able to...

- Collect and ingest structured datasets (CSV and JSON) in Python
- Extract simple statistics from those datasets, including processing text time/date data
- Explore and visualize the data, and create simple plots from it

Course summary

In the **next course** we will...

- Show how to apply **machine learning** to model the data we have ingested
- Introduce **regression** and **classification** techniques for different prediction tasks
- Show how to extract meaningful **features** from our dataset in order to make predictions effectively