CS 260: Foundations of Data Science

Prof. Sara Mathieson

Spring 2025

HAVERFORD
COLLEGE

Admin

- **EVERYONE**: (enrolled and waitlist)
  - Sign in
  - Pick up two handouts (one half-sheet, one full-sheet)
  - Pick up a construction paper sheet
- We still have an imbalance in the labs – if you can come to **Lab A** please let me know!
- Reserve chairs for enrolled students
- Let me know if you can't access Piazza

Outline

- Preliminaries
- Examples of Data Science and learning from data
- Syllabus highlights
- Python for this course
  - Numpy
  - Matplotlib

Outline

- Preliminaries
- Examples of Data Science and learning from data
- Syllabus highlights
- Python for this course
  - Numpy
  - Matplotlib

Course Staff

- **Instructor**: Sara Mathieson

- **TAs**:
  - Edgar Leon
  - Rachel November
  - Danylo (Dan) Shudrenko

Handout and Name card

- **Half-sheet handout**

- **Name card** (“tent”)
  - Preferred first name
  - Pronouns (optional)
  - (sharpies going around!)

Discuss with a Partner

- Introduce yourselves
  - Be ready to introduce your partner to the class!
- Come up with your own definition of “Data Science”
- What are some examples of Data Science that you have encountered or would like to encounter?

Outline

- Preliminaries
- Examples of Data Science and learning from data
- Syllabus highlights
- Python for this course
  - Numpy
  - Matplotlib

Information is what we want, but data are what we've got. The techniques for transforming data into information go back hundreds of years. A good starting marker is 1592 with the publication of weekly "bills of mortality" in London. These bills were tabulations: a condensing of the data into a form more readily assimilated by the human reader. Constructing such tabulations was a manual operation.

As data became larger, machines were introduced to speed up the tabulations. A major step was Hollerith's development of punched cards and an electrical tabulating system for the US Census of 1890. This was so successful that Hollerith started a company, IBM, that came to play an important role in the development of today's electronic computers.

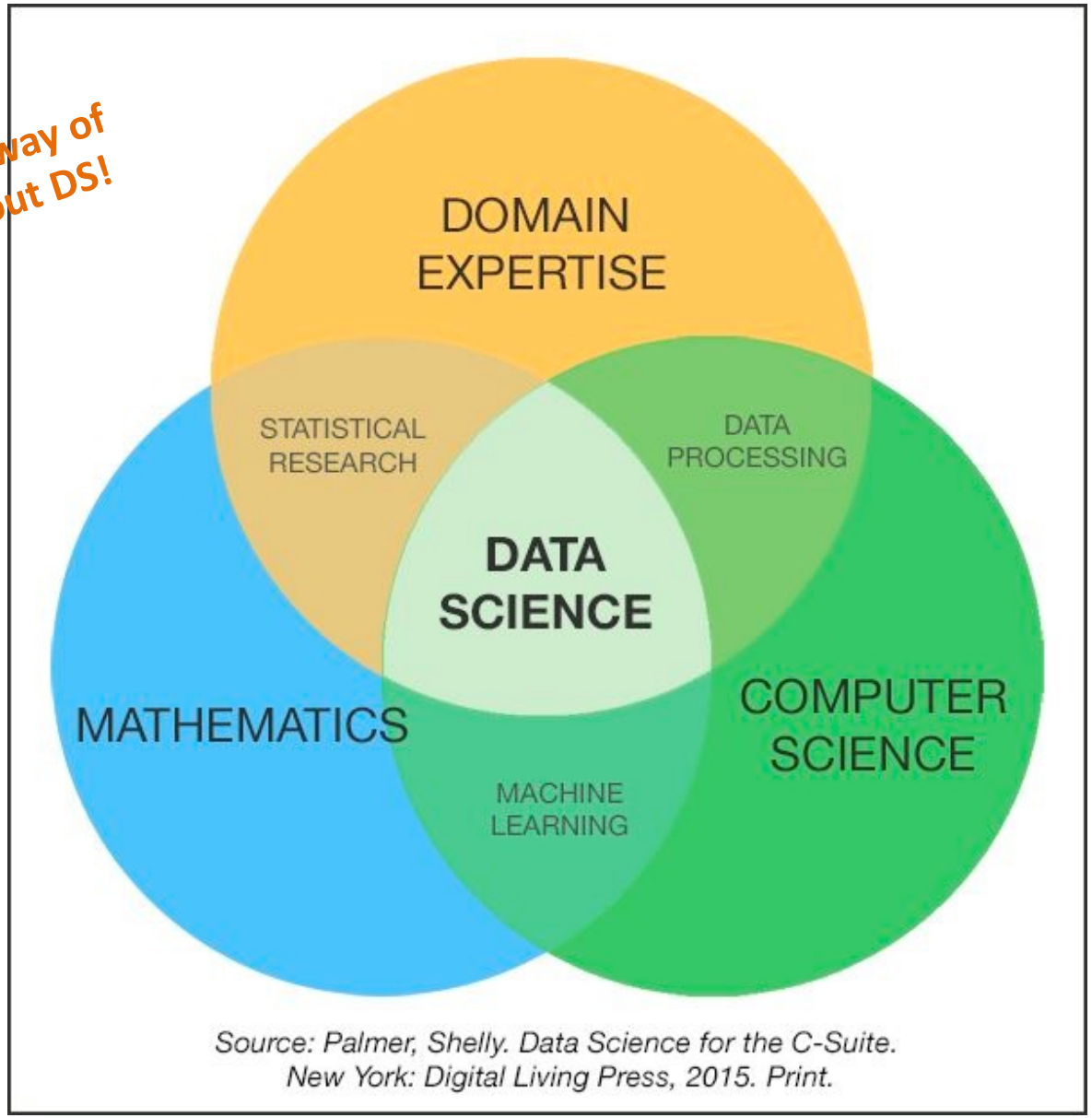
Also in the late 19th century, statistical methods began to develop rapidly. These methods have been tremendously important in interpreting data, but they were not intrinsically tied to mechanical data processing. Generations of students have learned to carry out statistical operations by hand on small sets of data, typically from a laboratory experiment.

Nowadays, it's common to have datasets that are so large they can be processed only by machine. In this era of "big data," often data are amassed by networks of instruments and computers. The settings for this are diverse: the genome, satellite observations of Earth, entries by web users, sales transactions, etc. With such data, there are new opportunities for finding and characterizing patterns using techniques such as data mining, machine learning, data visualization, and so on. As a result, everyday uses of data involve computer processing of data. This includes data cleaning, combining data from

From:
Data Computing
by Daniel Kaplan

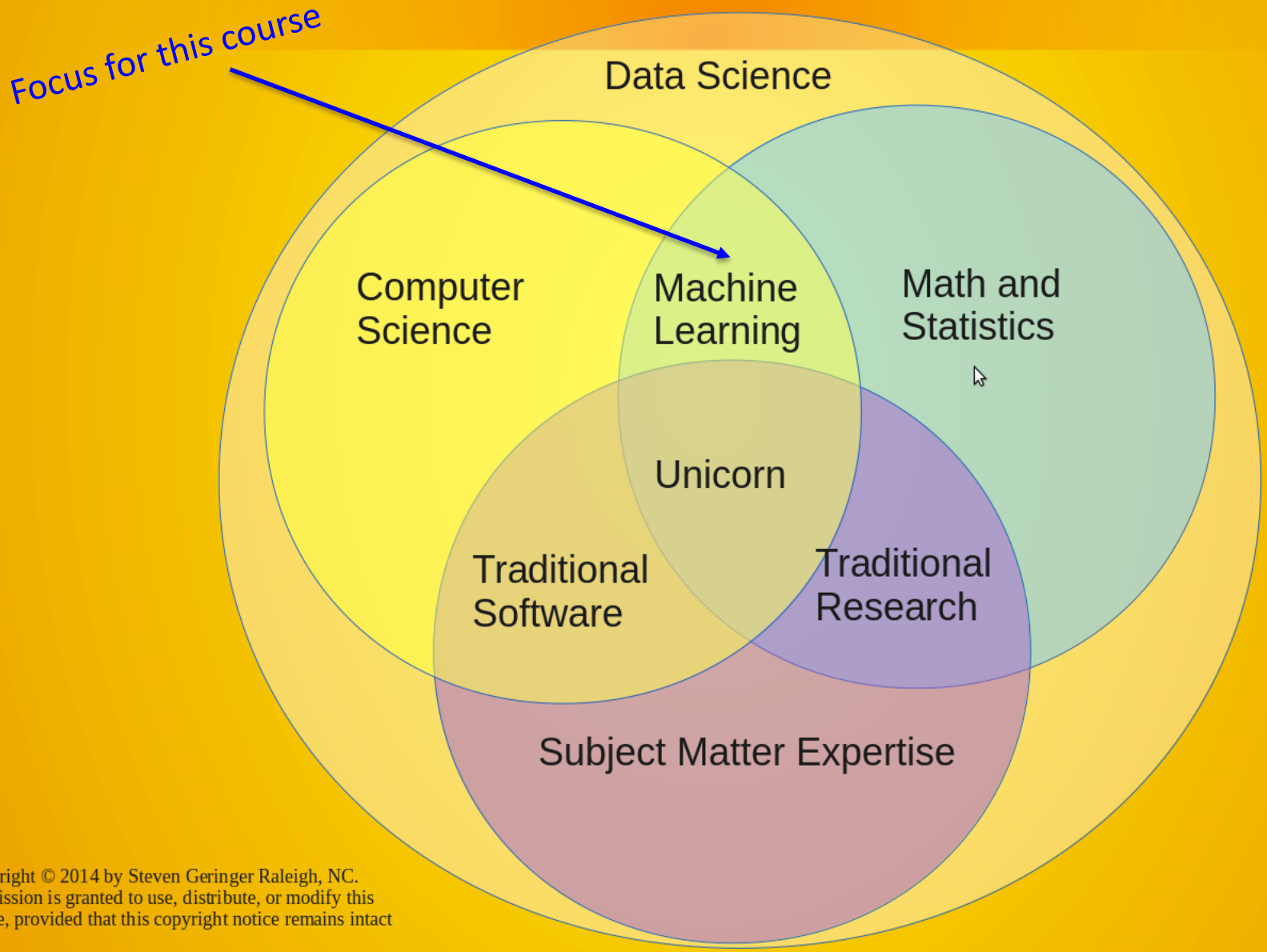
Data Science Venn Diagrams

Not the only way of thinking about DS!



Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

Data Science Venn Diagram v2.0



KEY COMPETENCIES FOR AN UNDERGRADUATE DATA SCIENCE STUDENT

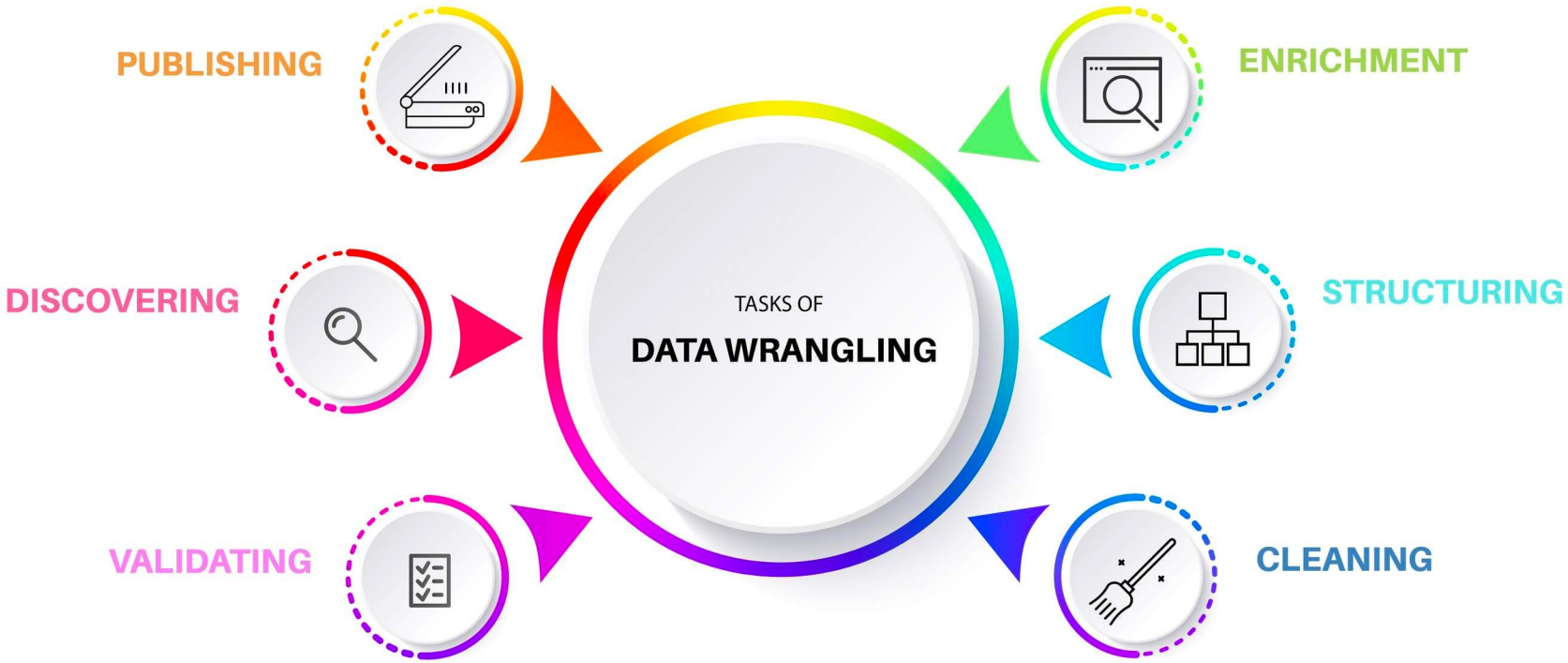
- Computational and statistical thinking
- Mathematical foundations
- Model building and assessment
- Algorithms and software foundation
- Data curation
- Knowledge transference—communication and responsibility

From: De Veaux *et al*, 2017

CS260 is not an entry level DS course

- "Math for Machine Learning"
- Meant to prepare students for 300-level data- oriented courses
  - Machine Learning (CS360)
  - Computational Linguistics (CS325)
  - Computer Vision
  - Computational Biology (CS364)
- Most of these are in Python so we will also cover advanced Python topics and libraries

“Data Wrangling”



https://favtutor.com/blogs/data-wrangling

Featurization

Age	Category
54	
13	
27	
21	
72	
17	

Continuous

Discrete

Featurization

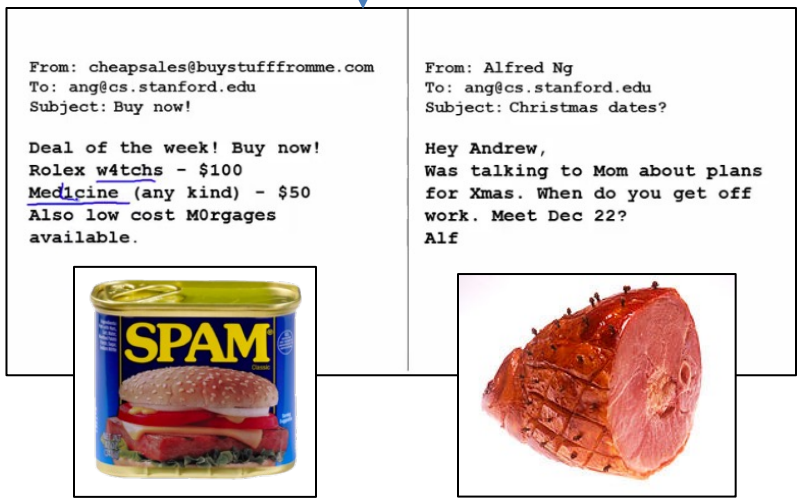
Age	Category
54	Adult
13	Minor
27	Adult
21	Adult
72	Senior
17	Minor

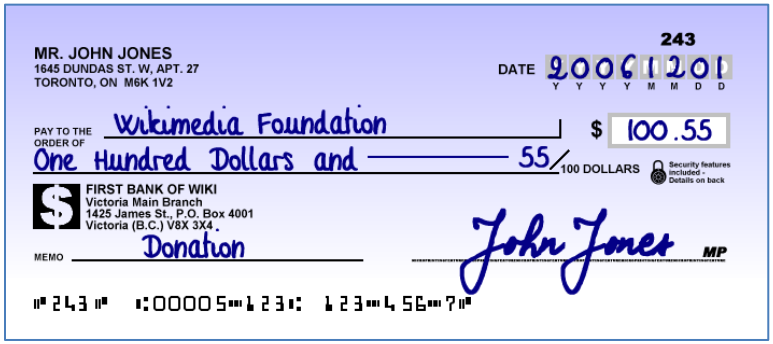
Continuous

Discrete

Learning from data: classification

- Email filtering (spam vs. not-spam)

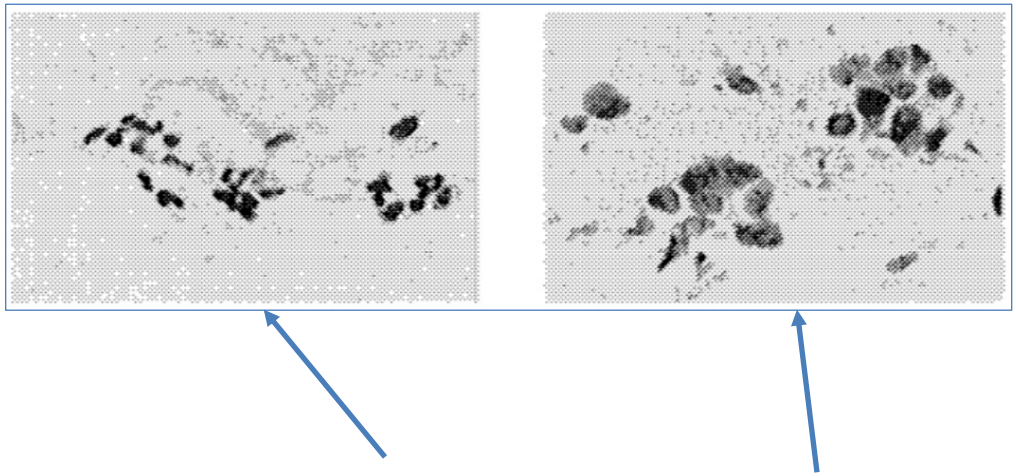




- Handwriting recognition (digits in a check)

Images: Wikipedia, Matouš Havlena

Learning from data: classification



- Tumor detection (benign vs. malignant)

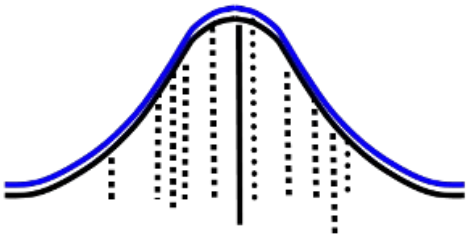
"On Breast Cancer Detection: An Application of Machine Learning Algorithms on the Wisconsin Diagnostic Dataset"

Statistical Tests

Do smokers weigh the same as non-smokers?

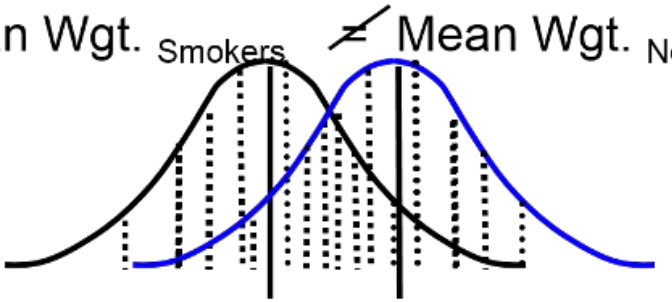
Null Hypothesis (H_0): the average weight does *not* differ

H_0: Mean Wgt. $_{\text{smokers}}$ = Mean Wgt. $_{\text{Non-smokers}}$



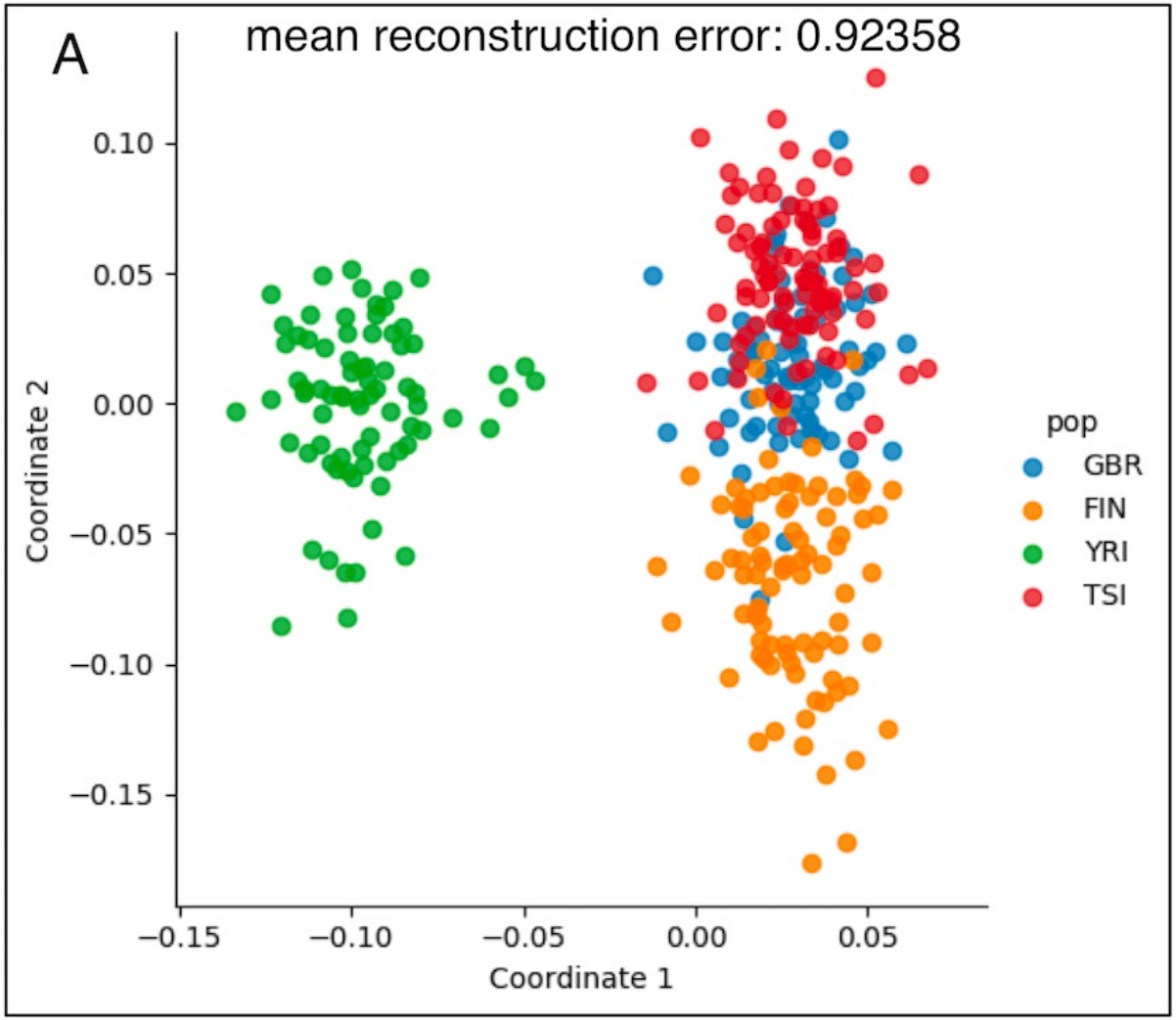
Alternative Hypothesis (H_A): the average weights differ

H_A: Mean Wgt. $_{\text{Smokers}}$ $\neq$ Mean Wgt. $_{\text{Non-smokers}}$



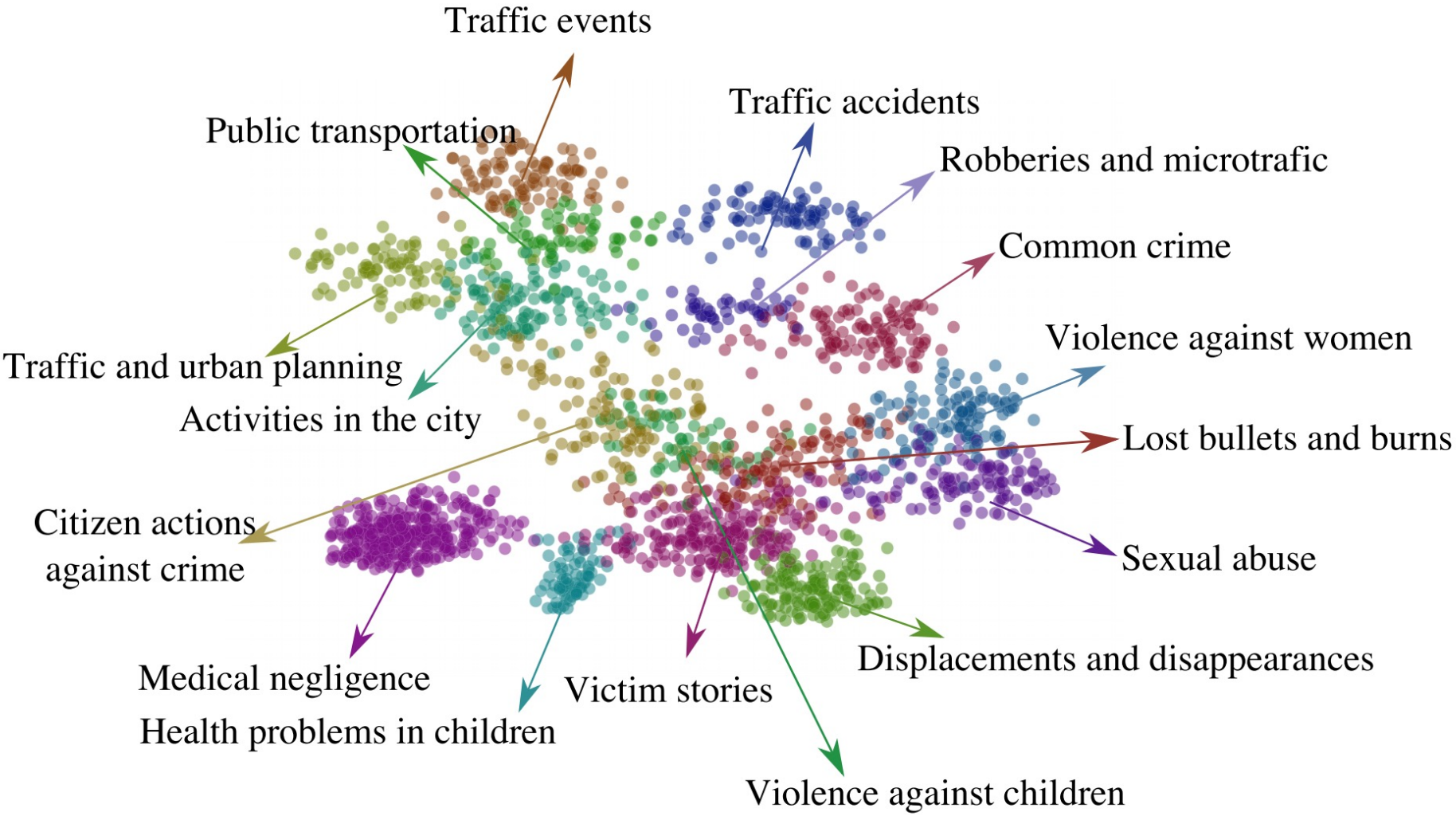
Wayne W. LaMorte, MD, PhD, MPH
Boston University School of Public Health

Data Visualization



"Expression reflects population structure", Brown *et al* (2018)

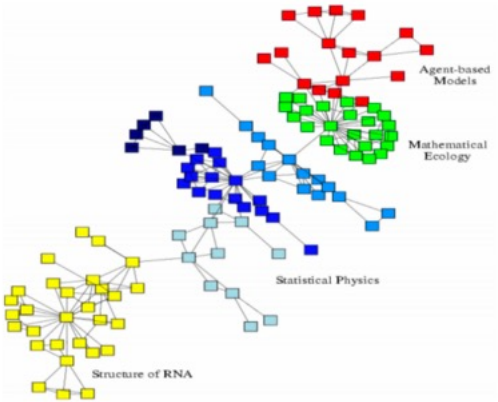
Clustering Data



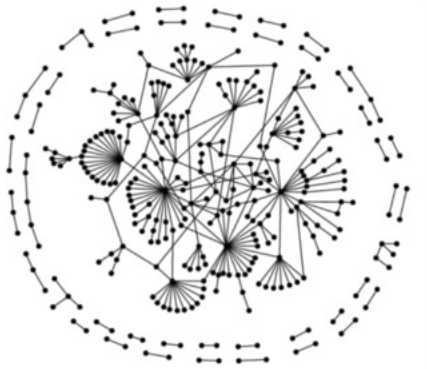
"Event detection in Colombian security Twitter news using fine-grained latent topic analysis", Vargas-Calderon *et al* (2019)

Learning from data: networks

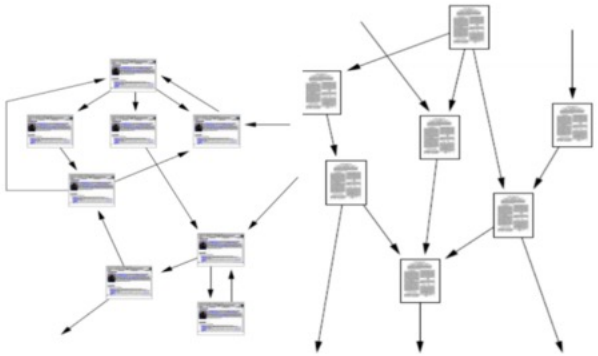
Social networks



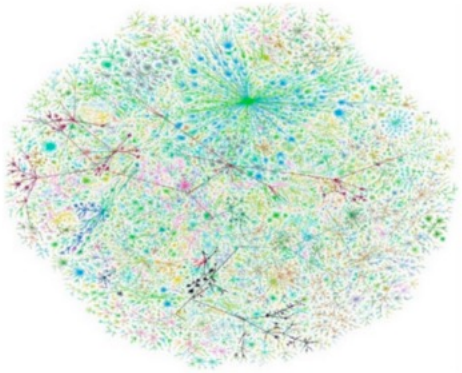
Economic networks



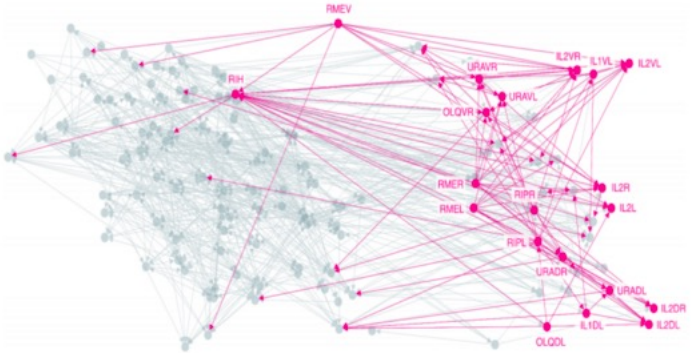
Biomedical networks



Information networks:
Web & citations



Internet



Networks of neurons

"A Gentle Introduction to Graph theory", by Xavier Sumba

Natural Language Examples

- Text Classification
- Language Modeling
- Speech Recognition
- Caption Generation
- Machine Translation
- Document Summarization
- Question Answering
- Text Generation

Machine reading comprehension

Q: What was the theme?
A: "one world, one dream".

Q: What was the length of the race?
A: 137,000 km

Q: Was it larger than previous ones?
A: No

Q: Where did the race begin?
A: Olympia, Greece

Q: Is there anything notable about that place?
A: birthplace of Olympic Games

Q: Where did they go after?
A: Athens

Q: How many days was the race?
A: seven

Q: Did they visit any notable landmarks?
A: Panathinaiko Stadium

Q: And did they climb any mountains?
A:

Target answers: *unknown* or *yes*
Model answer: `Everest`

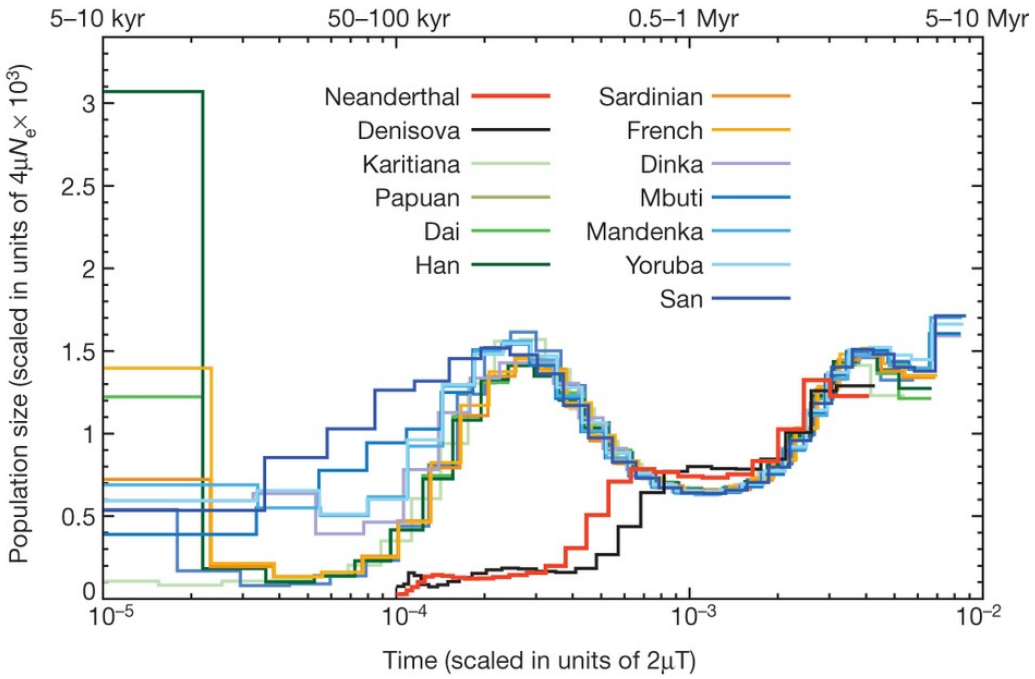
https://openai.com/blog/better-language-models/

https://machinelearningmastery.com/applications-of-deep-learning-for-natural-language-processing/

Biology examples

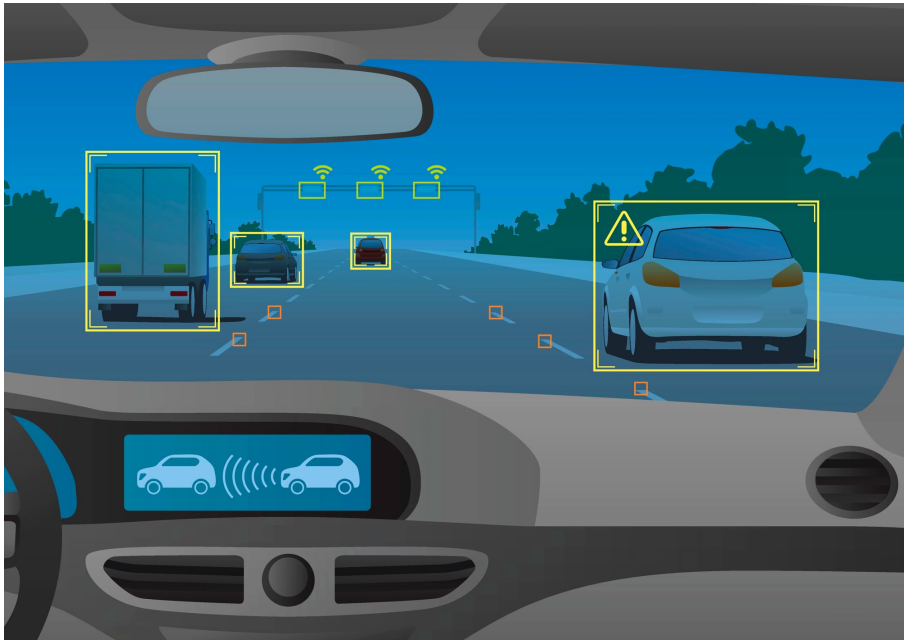
- Identify genes associated with disease
- Protein structure prediction
- Genes under natural selection
- Protein binding site prediction

- Population size changes



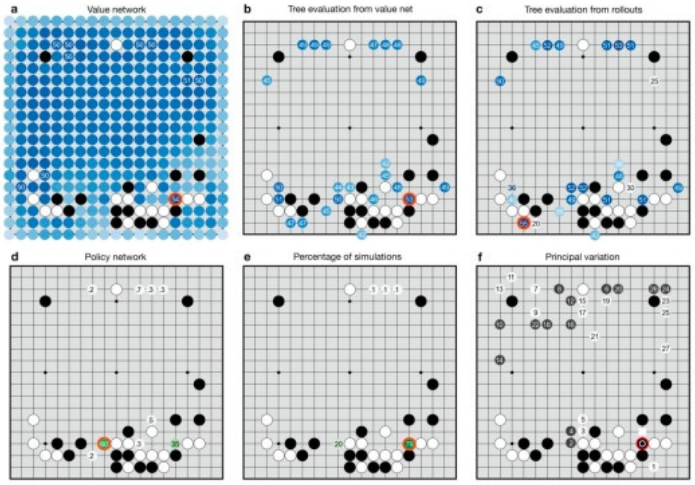
The complete genome sequence of a Neanderthal from the Altai Mountains, Prufer et al (2014)

Modern Machine Learning examples



Self-driving cars are in our present and future

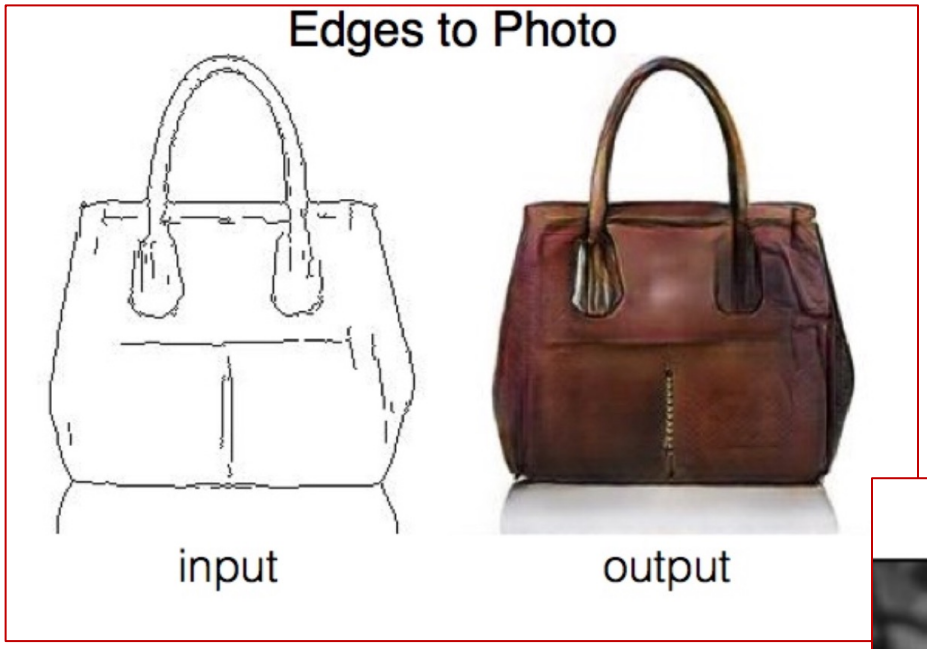
AlphaGo: moves humans never thought of



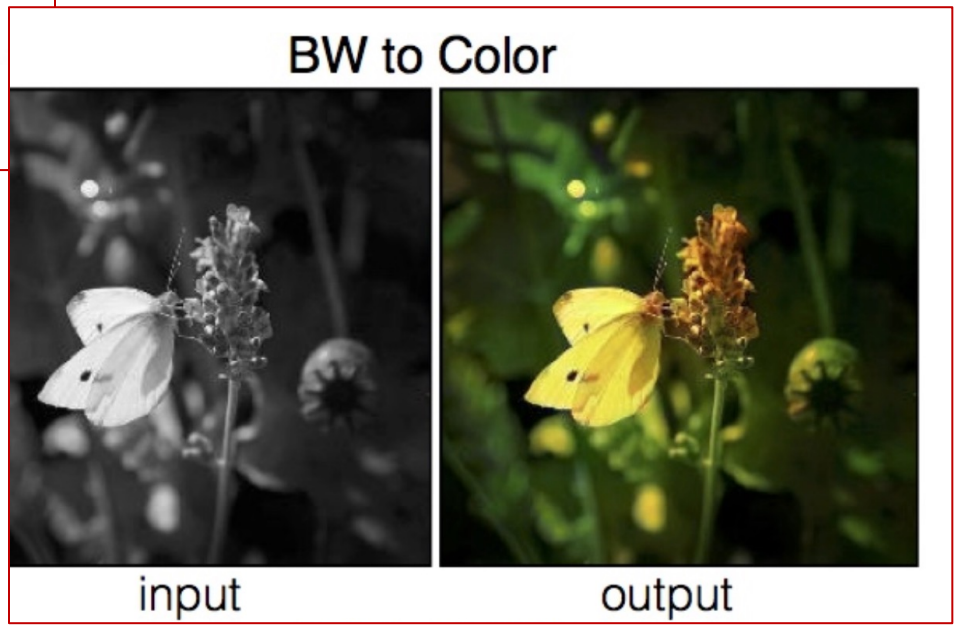
AlphaGo

Images: Scientific American

Modern Machine Learning examples



- Algorithms that learn how to create



[Image-to-Image Translation with Conditional Adversarial Nets (Nov 2016)]

Generative Models

https://github.com/junyanz/CycleGAN

Generative models have also been used to create synthetic genomic data to maintain privacy

Ethics and Responsible use of Data

- Based on huge quantities of data, algorithms decide what you see online
  - Search results
  - Targeted ads
  - Newsfeed content
  - Removal of problematic content

- Even if data is “cleaned” to remove protected features (e.g. race, sex), these can be redundantly encoded

We must join the conversation

"When human beings acquired language, we learned not just how to listen but how to speak. When we gained literacy, we learned not just how to read but how to write. And as we move into an increasingly digital reality, we must learn not just how to use programs but how to make them."

-Douglas Rushkoff

Outline

- Preliminaries
- Examples of Data Science and learning from data
- Syllabus highlights
- Python for this course
  - Numpy
  - Matplotlib

Weekly schedule

Should be most of the way through lab

Lab n posted

Week n	Week n+1
Sun	Sun
Mon	Mon
Tues	**Tues**
Wed	Wed
Thurs	**Thurs**
Fri	Fri
Sat	Sat

Lab n due
Lab n+1 posted

Blue: class meetings

Lab (Tuesdays 2-3pm and 3-4pm)

- Lab attendance is required! Please email me if you will be missing lab
- Usually give short suggestions/tips to everyone
- Occasionally pair-programming exercises
- I will check in with everyone and answer questions individually or in groups

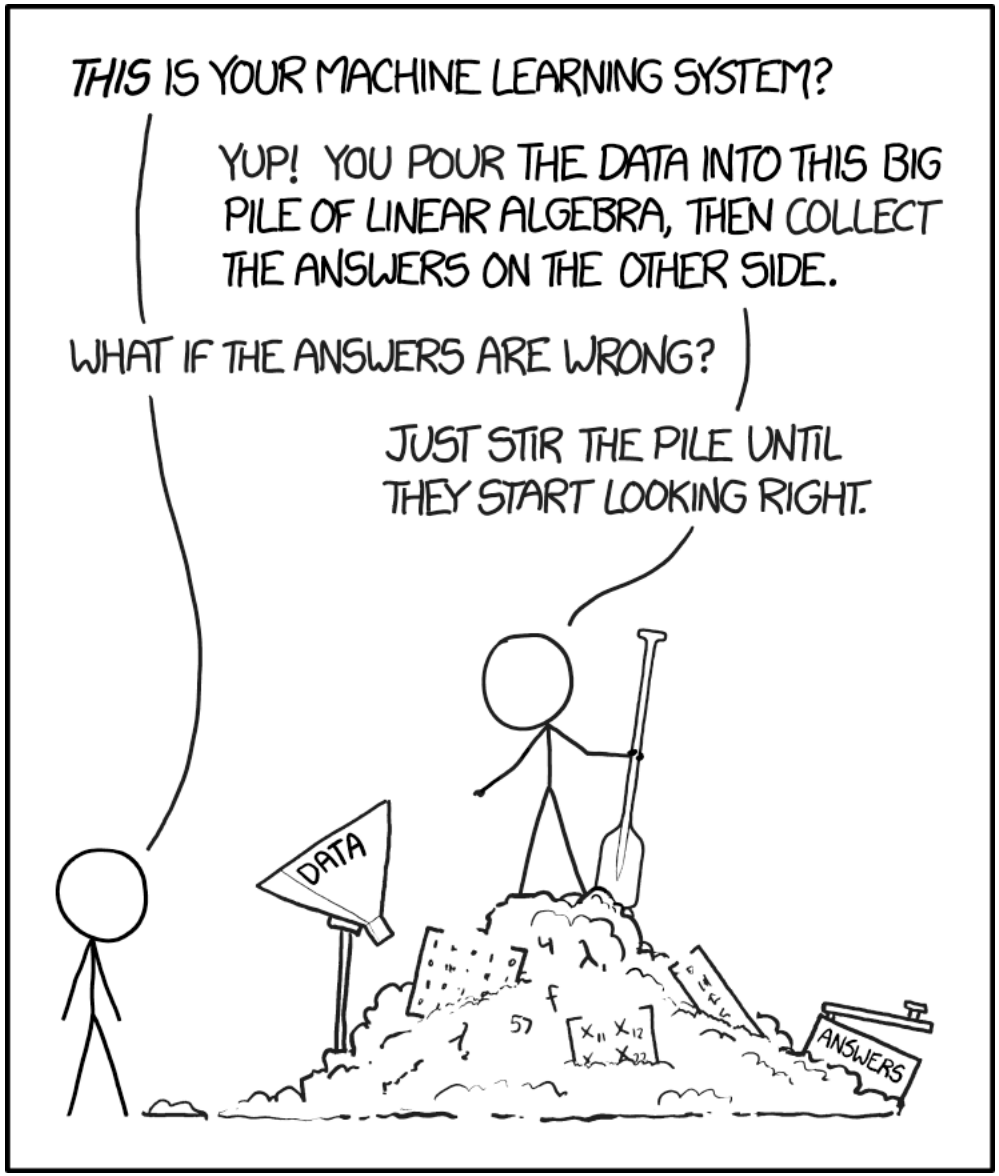
Learning Goals

- Understand how algorithms make decisions based on data
- Understand the theoretical foundations of DS
  - applied linear algebra, probability, statistics, modeling information theory, geometry, and optimization
- Be able to implement the theory and apply it to a variety of domains
- Analyze the ethical use of data and weigh tradeoffs in data collection and usage
- Understand and apply best practices in data visualization
- Throughout and during the project: hypothesis development, featurization, algorithm selection, interpretation of results, iteration, conclusions
- Comfort with using advanced Python topics such as libraries and object-oriented design

Topics (tentative)

- Representing data
- Common Python libraries
- Object-oriented Python
- Modeling
- Linear models
- Applied linear algebra
- Optimization
- Gradient descent
- Confusion matrices
- ROC curves
- Probabilistic modeling
- Naïve Bayes algorithm
- Ethics and protected features
- Information theory
- Data visualization
- Dimensionality reduction
- Clustering
- Permutation testing and bootstrapping
- Hypothesis testing

There will be math!



Different Backgrounds

- Prerequisites
  - Data Structures
  - Discrete Math (co-req)
  - Calculus I

- May or may not have had statistics, probability, linear algebra, etc

Readings

- We will draw upon several online textbooks
- As well as supplemental online readings and research papers

Textbook:

You do not need to purchase a textbook for this course. We will draw from several online textbooks, as well as supplemental online readings and research papers.

- **Model-Based Machine Learning** by John Winn ("Winn" in schedule below)
- **Mathematics for Machine Learning** by Deisenroth, Faisal, and Ong ("MML" in schedule below)
- **Doing Bayesian Data Analysis** by John K. Kruschke ("DBD" in schedule below)
- **A Course in Machine Learning** by Hal Duame III ("Duame" in schedule below)
- (optional) **Visualization Analysis and Design** by Tamara Munzner

Course Components

- Labs (8 total): 35%
- Midterms: 40% (20% each)
- Final project: 15%
  - includes an oral presentation and “lab notebook”
- Participation: 10%
  - includes attendance, Piazza, note-taking, and general engagement with the course

My Expectations

- Come to class (Tu/Thurs) and lab (Tu), **ON TIME**, and actively participate
  - Email me if you will be absent from class or lab
- Complete reading before lab on Tuesday (some is technical, do your best)
- Come to office hours and TA hours. **My office hours: Mondays 4-5pm in H110**
- Post questions on **Piazza**

WEEK	DAY	ANNOUNCEMENTS	TOPIC & READING	LABS
1	Jan 21		**Introduction to Data Science and Python** • What can we learn from data? • Representing data • Crash course on Python • Numpy • Matplotlib (plotting in Python) • Classes and objects in Python • Dictionaries Reading: • MML Chap 1	Thurs Jan 23: virtual asynchronous class
	Jan 23			

Syllabus Notes

(Note: you are responsible for reading the entire syllabus on the course webpage)

1. Notes and slides will be posted *after* class on the course webpage
2. Lab is **mandatory** (attendance will be taken)
3. Labs may have an optional **pair programming** component
4. You will get **2 late days** during the semester (max one on any assignment)
5. Extensions beyond these two days must be arranged with your **class dean**
6. **Email**: allow at least 24 hours for a response (more during weekends)
7. **Piazza**: should be used for all content/logistics questions

Participation

What counts as participation?

- Asking and answering questions in class (very important!)
  - Will call on groups, but only after giving you a few minutes to think/discuss
- Actively participating in in-class activities (group work, handouts, polls)
- Collaborating with your lab partner for any pair-programming exercises
- Asking and answering questions on Piazza
  - Avoid long blocks of code and giving away answers
  - Only **non-anonymous public** posts count toward participation grade
- Attending office hours and TA hours

Academic Integrity

Faculty statement on academic integrity

In a community that thrives on relationships between students and faculty that are based on trust and respect, it is crucial that students understand a professor's expectations and what it means to do academic work with integrity. Plagiarism and cheating, even if unintentional, undermine the values of the **Honor Code** and the ability of all students to benefit from the academic freedom and relationships of trust the Code facilitates. Plagiarism is using someone else's work or ideas and presenting them as your own without attribution. Plagiarism can also occur in more subtle forms, such as inadequate paraphrasing, failure to cite another person's idea even if not directly quoted, failure to attribute the synthesis of various sources in a review article to that author, or accidental incorporation of another's words into your own paper as a result of careless note-taking. Cheating is another form of academic dishonesty, and it includes not only copying, but also inappropriate collaboration, exceeding the time allowed, and discussion of the form, content, or degree of difficulty of an exam. Please be conscientious about your work, and check with me if anything is unclear.

Academic Integrity

Note for CS260 in particular

Discussing ideas and approaches to problems with others on a general level is fine (in fact, we encourage you to discuss general strategies with each other), but you should **never read anyone else's code or let anyone else read your code.**

Github copilot (or any other software for automatically generating code) is not allowed for this course, until the final project. The reasoning behind this decision is that code generation tools often create code that is not well understood by the user. Often this code becomes incorrect in the larger context of the program. However, for the final project you are welcome to use Github copilot, and you'll be asked to reflect on your experience.

Academic Accommodations

Faculty statement on accommodations

Haverford College is committed to providing equal access to students with a disability. If you have (or think you have) a learning difference or disability – including mental health, medical, or physical impairment - please contact the Office of Access and Disability Services (ADS) at **hc-ads@haverford.edu**. The Coordinator will confidentially discuss the process to establish reasonable accommodations.

Students who have already been approved to receive academic accommodations and want to use their accommodations in this course should share their verification letter with me and also make arrangements to meet with me as soon as possible to discuss their specific accommodations. Please note that accommodations are **not retroactive** and require advance notice to implement.

It is a state law in Pennsylvania that individuals must be given advance notice if they are to be recorded. Therefore, any student who has a disability-related need to audio record this class must first be approved for this accommodation from the Coordinator of Access and Disability Services and then must speak with me. Other class members will need to be aware that this class may be recorded.

https://www.haverford.edu/access-and-disability-services/accommodations/receiving-accommodations

Outline

- Preliminaries
- Examples of Data Science and learning from data
- Syllabus highlights
- Python for this course
  - Numpy
  - Matplotlib

Python style

- Decompose code into natural functions
- Avoid global variables (sometimes useful)
- Include a file header with purpose, author, and date
- Include headers for each function
- No lines over 80 chars
- Variable names implicitly show type
- Include line breaks and comments!

Python style

- “Snake-case” not “camel-case”
  - ~~linearSearch~~
  - **linear_search**

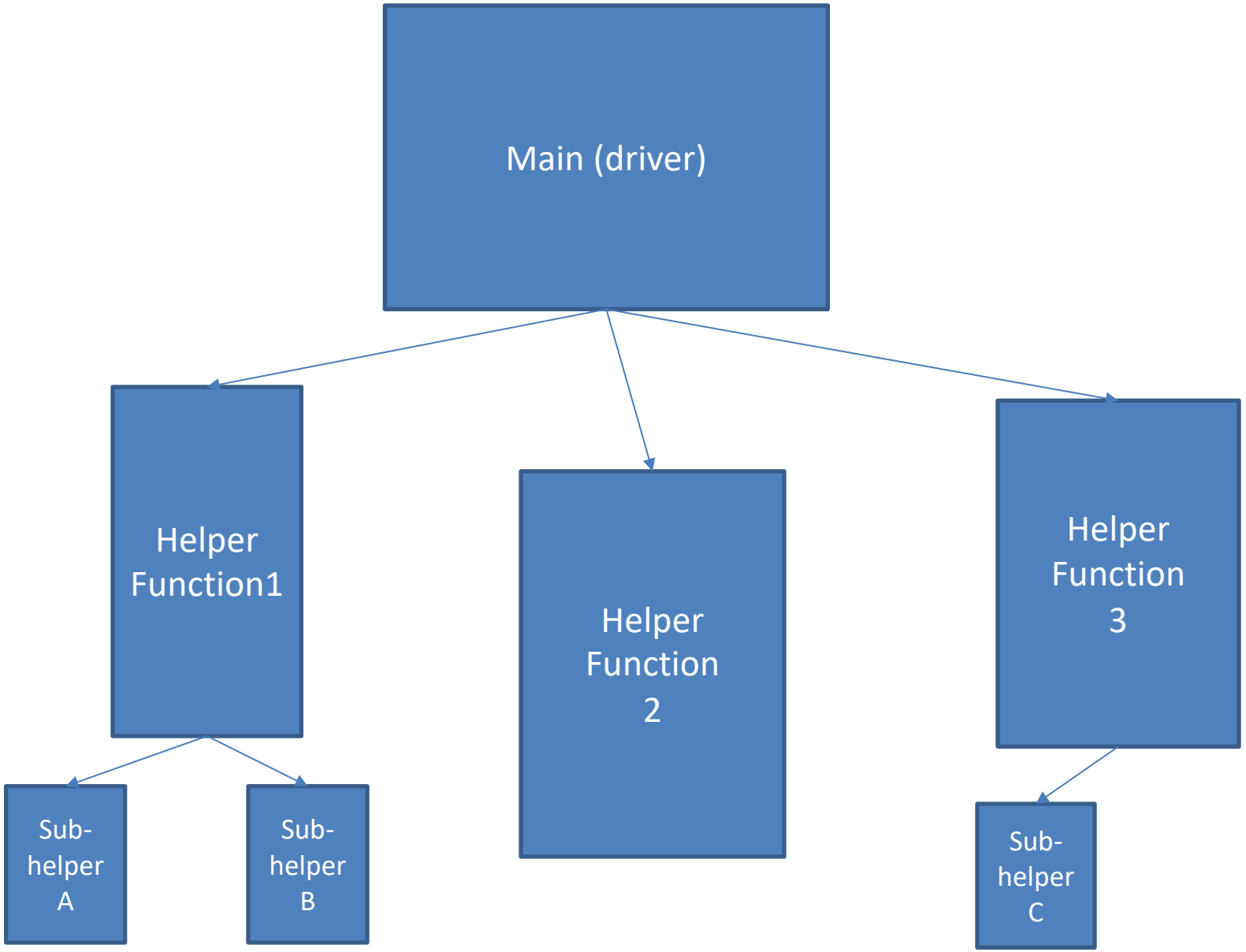
- Alphabetize imports and don’t use “*”
  - ~~from numpy import *~~
  - **import numpy as np**

Python style examples

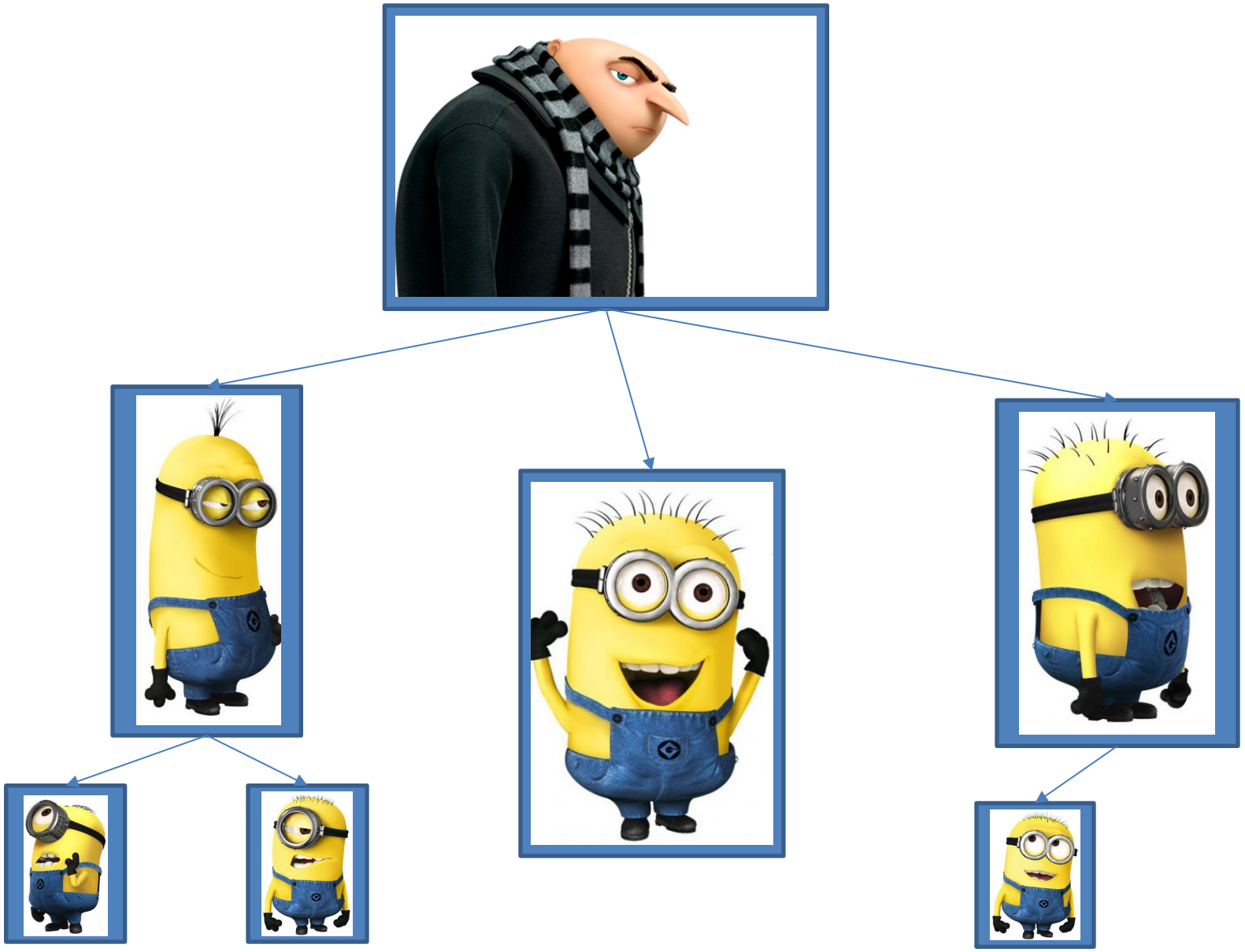
```
"""
Ask the user for their name and welcome them to CS21.
Author: Sara Mathieson
Date: 9/7/18
"""

def main():

    # ask user for their name and print greeting
    name = input("Enter your name: ")
    print("Hello", name, "!")

main()
```

```
def factorial(n):
    """
    Given a non-negative integer n, return n! = n*(n-1)*(n-2)....3*2*1.
    """
    fact = 1 # set up an accumulator variable
    for i in range(n):
        fact = fact * (i+1) # accumulator pattern
    return fact
```

Structure of main and “helper” functions



Structure of main and “helper” functions



Reminder: steps of top-down-design (TDD)

1) Design a high-level main function that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be abstracted into small tasks. Make names for these functions and write their signatures below main.

3) "Stub" out the functions. This means that they should work and return the correct type so that your code runs, but they don't do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

4) Iterate on your design until you have a working main and stubbed out functions. Then start implementing the functions, starting from the "bottom up".

Reasons to use TDD

- Creates code that is easier to implement, debug, modify, and extend
- Avoids going off in the wrong direction (i.e. implementing functions that are not useful or don’t serve the program)
- Creates code that is easier for you or someone else to read and understand later on

DEMO + Handout 1

- Matplotlib
- Numpy
- Dictionaries

positives

* comments are good
* variable names

negatives

* no header
* no main
* I/O problems

☆

num_chars=
count = 0
phrase[i] == letter

for ch in phrase:
    if ch == letter

print(f"Number of {letter}'s: {count}")

Handout 1 notes

```
"""
Given an input phrase and a letter, count how many times that letter appears
in the phrase. For example:

phrase: creative code
letter: e
Number of e's: 3

Author: Jeff Knerr & Sara Mathieson
Date: 9/21/18
"""

def main():

    # ask the user for a phrase and a letter
    phrase = input("phrase: ")
    letter = input("letter: ")
    num_chars = len(phrase)

    # set up accumulator variable count
    count = 0
    for i in range(num_chars):
        # add on 1 each time we see the desired letter
        if phrase[i] == letter:
            count = count + 1

    # example of string formatting (%s for str, %i for int)
    print("Number of %s's: %i" % (letter, count))
```

print(f “Number of {letter}’s: {count}”

```
main()
```

Handout 1
(example solution)

TODO

- Read over **Lab 1**, accept assignment on github
  - (can be done during lab)

- Come to lab TODAY!
  - **Lab A: 2-3pm (H110)**
  - **Lab B: 3-4pm (H110)**

- Reading: **MML Chap 1**

- **Virtual asynchronous class on Thursday!**
  - Watch the video lecture at your own pace
  - Complete the **google form to maintain your spot**
  - Includes answers to Handout 2 questions