AI/ML BOOTCAMP

William Phelps

Christopher Newport University/Jefferson Lab

CHRISTOPHER NEWPORT UNIVERSITY

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Announcements

- Welcome to the AI/ML Bootcamp for CNUGS
- We will not be using git today
- If you do not have anaconda or python with matplotlib, numpy, and pandas installed we will be using google colab

Install Python 3, PyCharm on your computer

- ~~https://www.anaconda.com/distribution/ with Python version 3.12 for new installs!~~
  - ~~This includes many packages that we will need (Numpy, Scipy, Pandas, Matplotlib)~~
  - Instructions for miniforge coming soon
- (Future!) PyCharm IDE https://www.jetbrains.com/pycharm/
  - Professional edition is preferred!
  - Professional version is available free for students/faculty
    - Needs verification via email
- (Future!) Git https://git-scm.org/downloads (Windows)
  - When it asks you whether or not you want to use VIM, select the "Use system file editor" option.
  - When it asks you about which command line interface to use, select the "Git Bash Command Line Interface".
  - You should leave all other options as default when finishing the installation.
    - **Note:** Mac and Linux usually have git built in or it will install on first use

Course Topics*

1 Semester DS/AI Course in 4 lectures
45 hours in 8 hours

- **Lectures 1 & 2**
  - **Today**
    - Introduction to workflow and tools (git/anaconda/pycharm)
    - Introduction to Data Science
    - Python Review with jupyter notebooks
    - Pandas introduction
  - **Lecture 2**
    - Data Visualization
    - Principles of data visualization from Edward Tufte
    - Interactive plots with plotly and folium

- **Lectures 3 & 4**
  - **Lecture 3**
    - Regression/Curvefit
    - Overfitting/Underfitting
    - kNN classifier
  - **Lecture 4**
    - MNIST/Handwritten digit classifier
    - YoloV11 Demo with classification/pose detection/image segmentation
    - Background blue
    - VDOT traffic counter

*Tentative

About your instructor

- Experimental Nuclear Physicist and Computer Science Professor
- BS Applied Physics: CNU '11
- PhD Nuclear Physics: FIU '17
- Postdoctoral Scientist at GWU 2017 – 2019
- Working at JLab since 2009
  - CLAS 2009-Present
  - GlueX 2015-2019
  - PrimEx-II, Klong Facility, ePIC
- Assistant Professor at CNU and Staff Scientist at Jefferson Lab – 2019
- Honorary Research Fellow at University of Glasgow - 2022-Present
- Associate Professor – Fall 2025

CNU

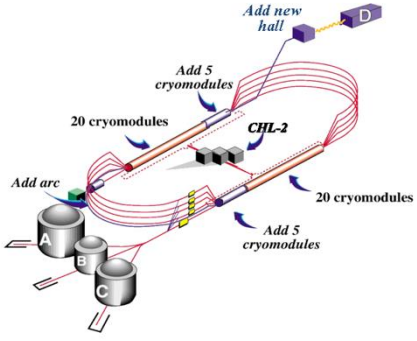
FIU
Florida International University
Miami's public research university

The George Washington University
Washington, DC

Jefferson Lab
Thomas Jefferson National Accelerator Facility

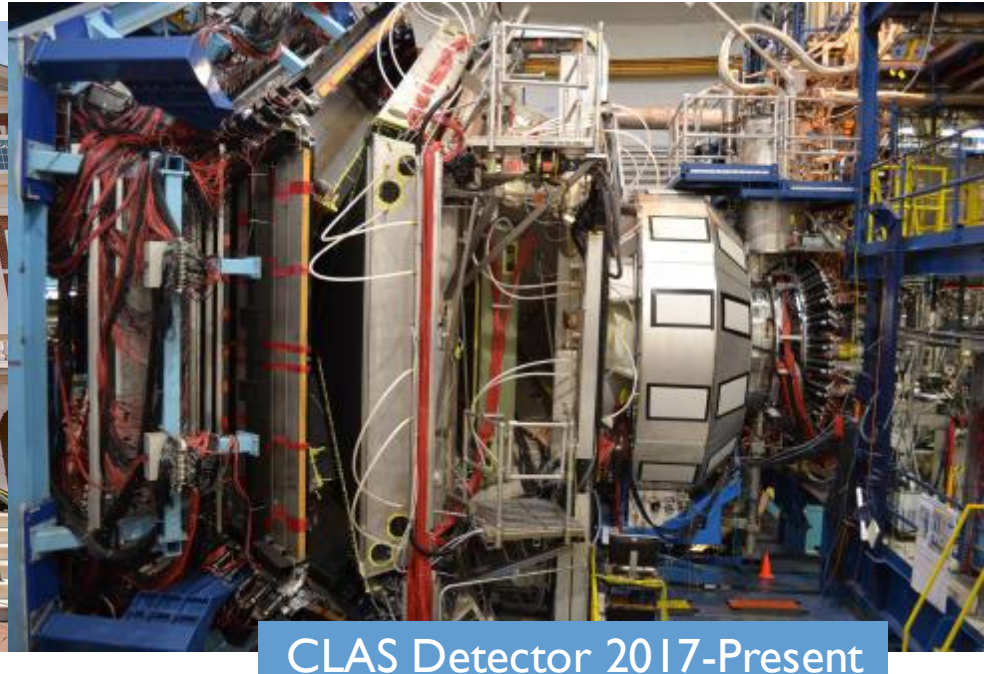
University of Glasgow

What I do at Jefferson Lab



CLAS Collaboration

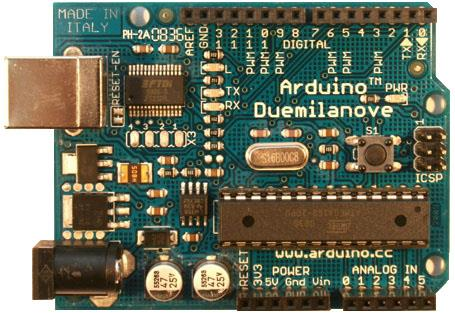
CLAS Collaboration July 2023



CLAS Detector 2017-Present

Elected Collaboration Chair from 2023-2025

Software work



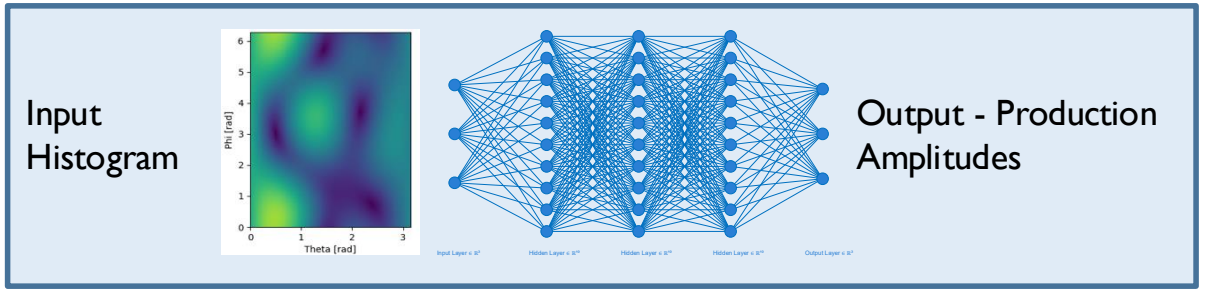
- Lots of data analysis in physics!
- Multithreaded analysis is needed for large datasets
- Petabytes of data per run
- 1GB/s DAQ rate!

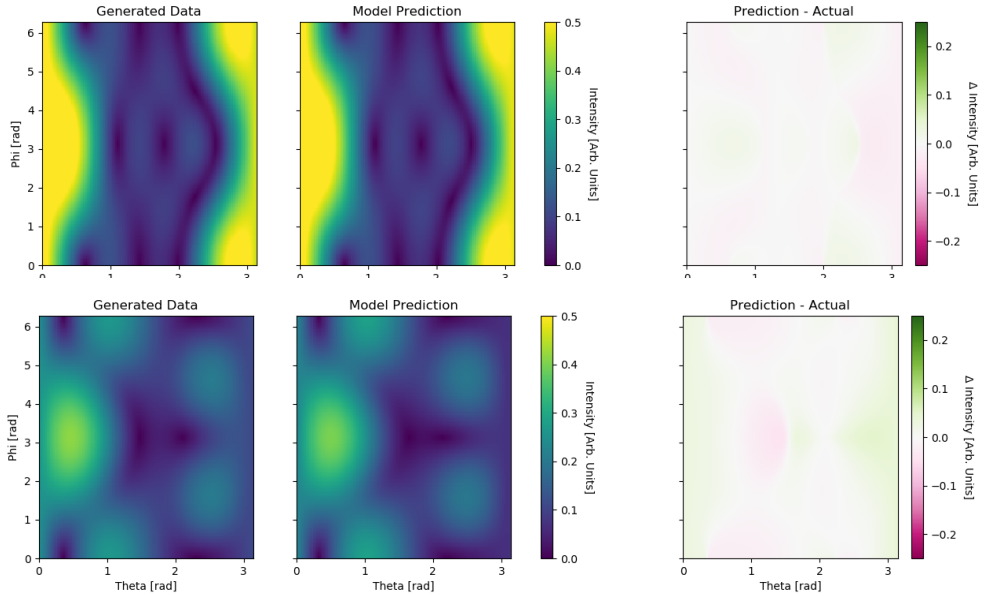
PyPWA

FIRST

AI PWA

- We compare the intensity function and compare it to the model prediction
- Model Architecture:
  - 128x128 2D histogram as input
  - 9x128 Dense Layers – Relu activation
  - 9 production amplitudes as output
- In order to deal with the vast amounts of data we used generators to generate data for each epoch on the fly





Interesting Tools: Generators, Complex Valued Deep Learning

Large Language Model Research

- Summer 2023-Present
- Working with LLM-based agents such as AutoGPT for applications in Nuclear Physics
- Retrieval Augmented Generation with LLMs
  - Four students accepted into the Department of Energy SULI program
  - Three students accepted through CNU Summer Scholars

GPU Research Server

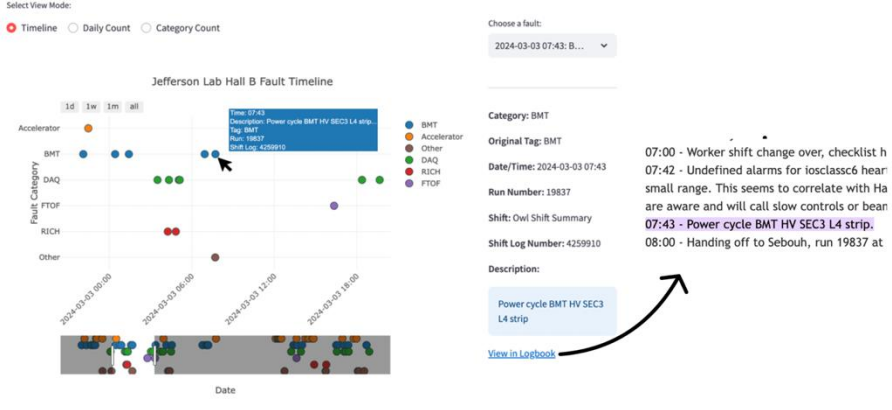


Figure 4: *Interactive timeline view of faults, also showing specific fault details and the link to the original logbook entry.*

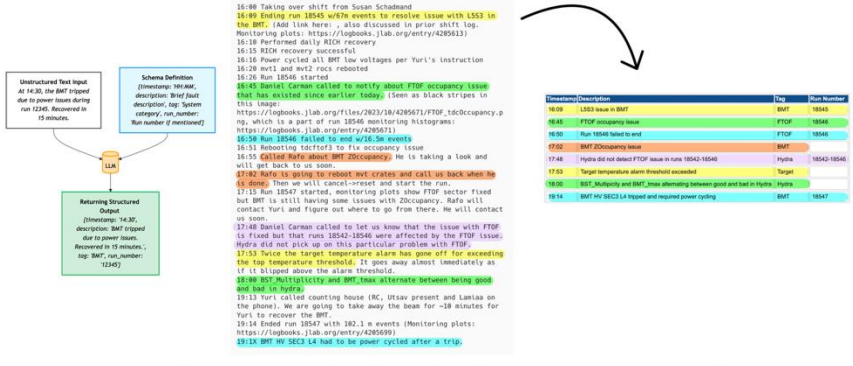
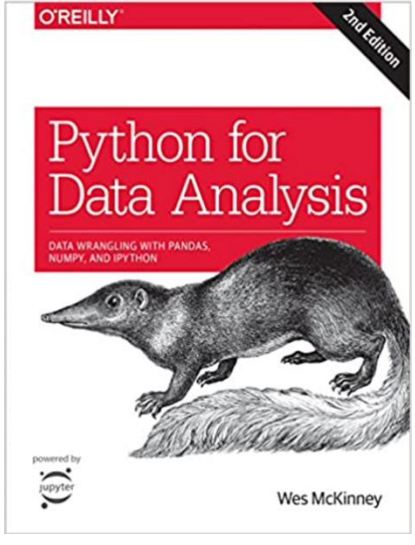
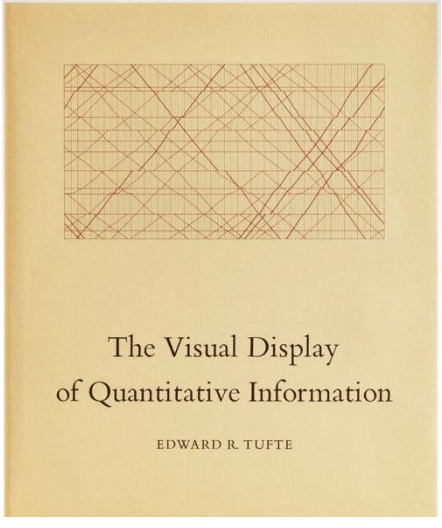


Figure 2: *Left: Fault schema example. Right: Example Shift Summary with the LLM's tabulated output*

Course Materials

- **Python for Data Analysis** by Wes McKinney 3rd edition
  - Freely available on O'Reilly Tech Books
  - New* https://wesmckinney.com/book/
    - Full book available in HTML
    - Github page with jupyter notebooks for each chapter
- **The Visual Display of Quantitative Information** by Edward Tufte
  - Optional but we will draw from material in this book





How to access O'Reilly Books

- https://www.oreilly.com
  - Access granted through JLab
- The pandas book is online and freely available: https://wesmckinney.com/book/

AI Tools

- PyCharm Professional has a local LLM assistant built in
  - Runs jupyter notebook server
- Github Copilot is strongly recommended
  - Freely available for faculty
  - Google "Github Copilot Educator/Student"
  - Requires some documentation to sign up but it's worth it!

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

class Sequence(nn.Module):
    def __init__(self):
        super(Sequence, self).__init__()
        self.lstm1 = nn.LSTMCell(1, 51)
        self.lstm2 = nn.LSTMCell(51, 51)
        self.linear = nn.Linear(51, 1)

    def forward(self, input, future = 0):
        outputs = []
        h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
```

Copilot with an autocompletion visible

GitHub Copilot

Github Copilot released: October 29, 2021

INTRO TO INTRO DATA SCIENCE

Datum, data, and science

DATA = PLURAL OF DATUM, BUT WE WILL USE 'DATA' FOR BOTH SINGULAR AND PLURAL VERSIONS

INFORMATION = MEANINGFUL DATA

SCIENCE = SYSTEMATIC STUDY OF THE STRUCTURE AND BEHAVIOR OF THE PHYSICAL AND NATURAL WORLD THROUGH OBSERVATION AND EXPERIMENTATION (OXFORD ENGLISH DICTIONARY)

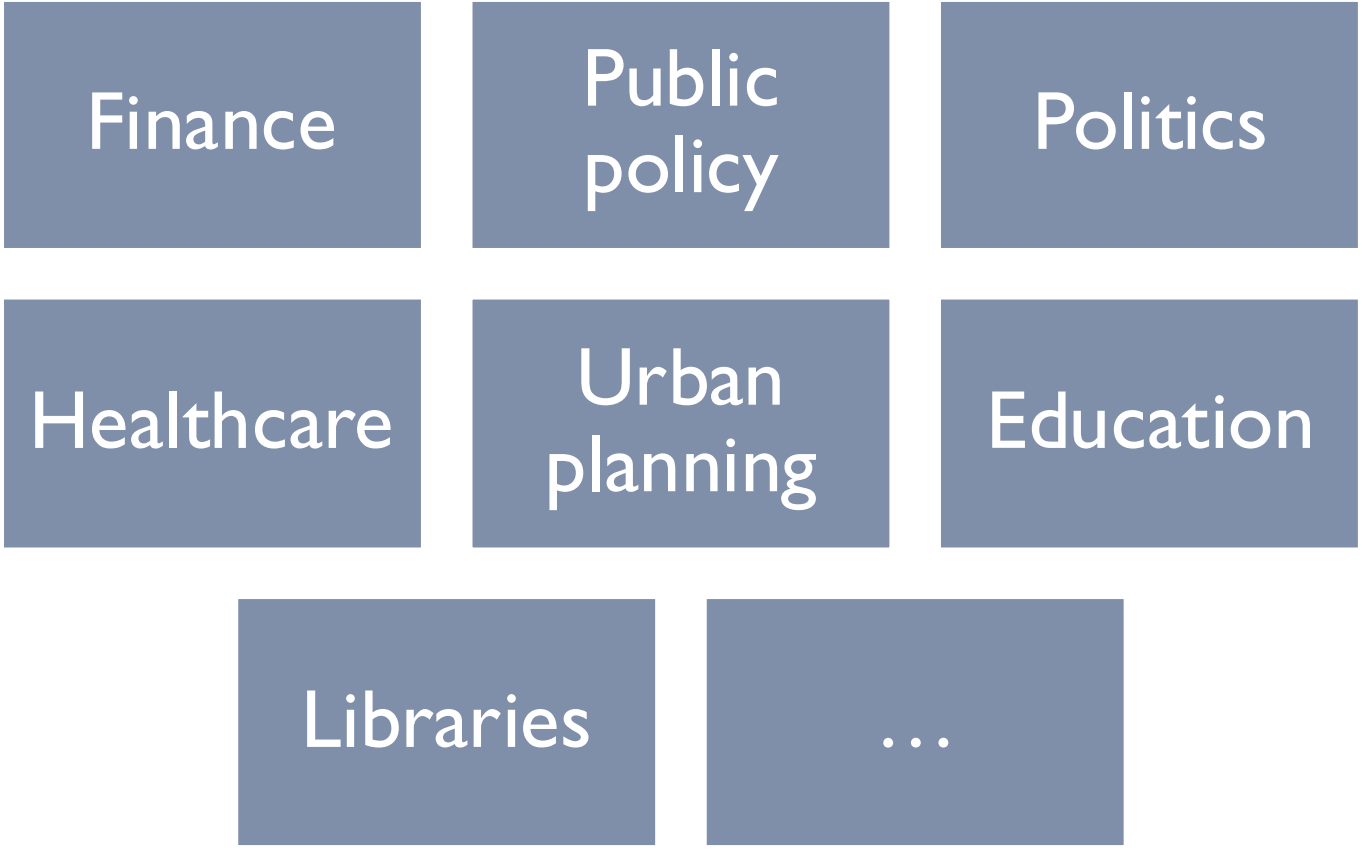
What is Data Science?

A field of study and practice that involves collection, storage, and processing of data in order to derive important insights into a problem or a phenomenon.

Data may be generated by **humans** (surveys, logs, etc.) or **machines** (weather data, road vision, etc.).

Data may be in different **formats** (text, audio, video, augmented or virtual reality, etc.).

Data Science applications in different domains



Relation of Data Science with other fields

Source: towardsdatascience.com

- Statistics
- Computer Science
- Engineering
- Business Analytics
- (Computational) Social Science

What skills do we need for Data Science?

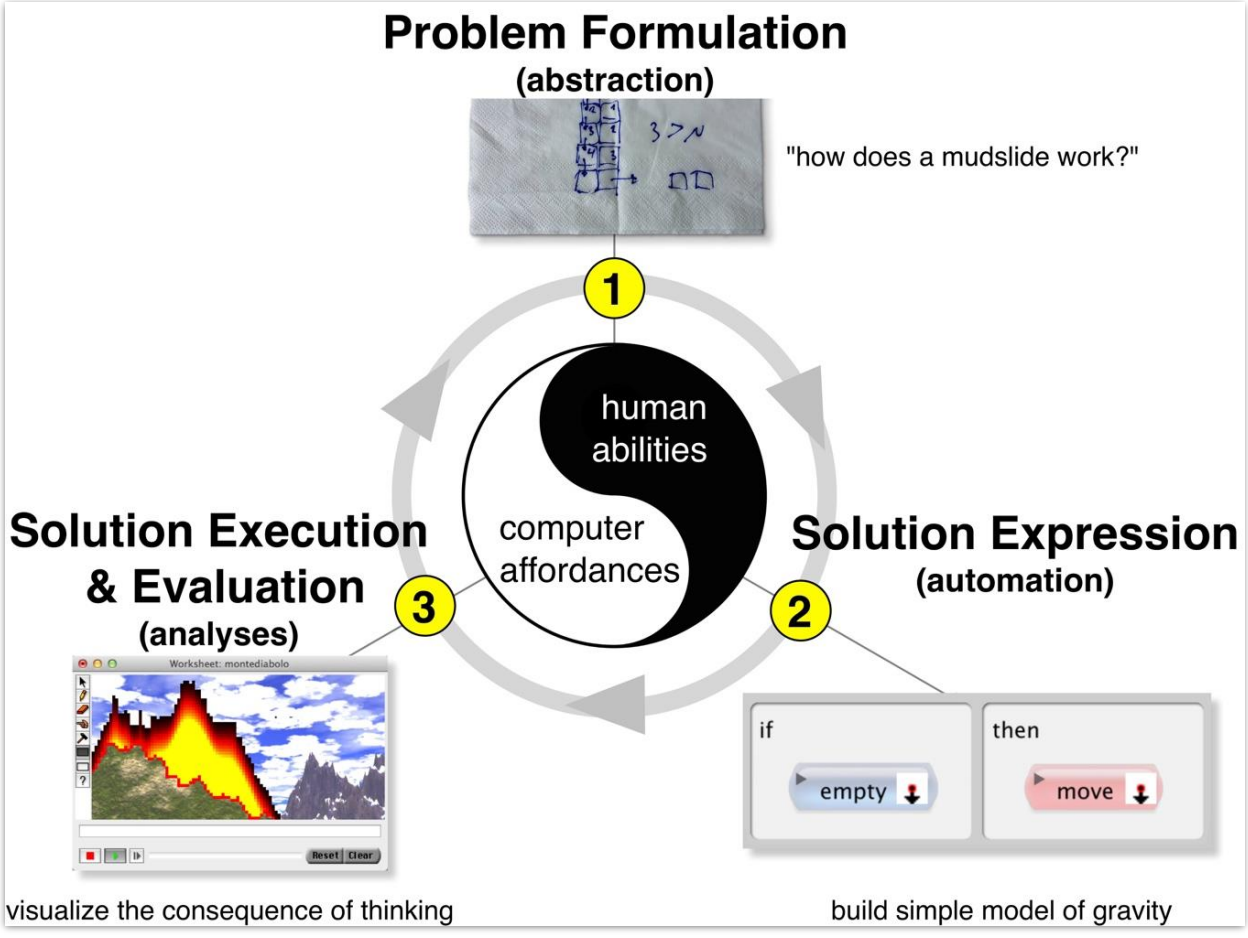
pandas

Willing to experiment

Proficiency in mathematical reasoning

Data literacy

Computational thinking



Source: WikiMedia: A. Repenning, A. Basawapatna, and N. Escherle, "Computational Thinking Tools"

What *is* Data Science

- Data Science is a new field that is not well defined and is a combination of other fields
- Heavily involves
  - Data
  - Statistics
  - Machine Learning

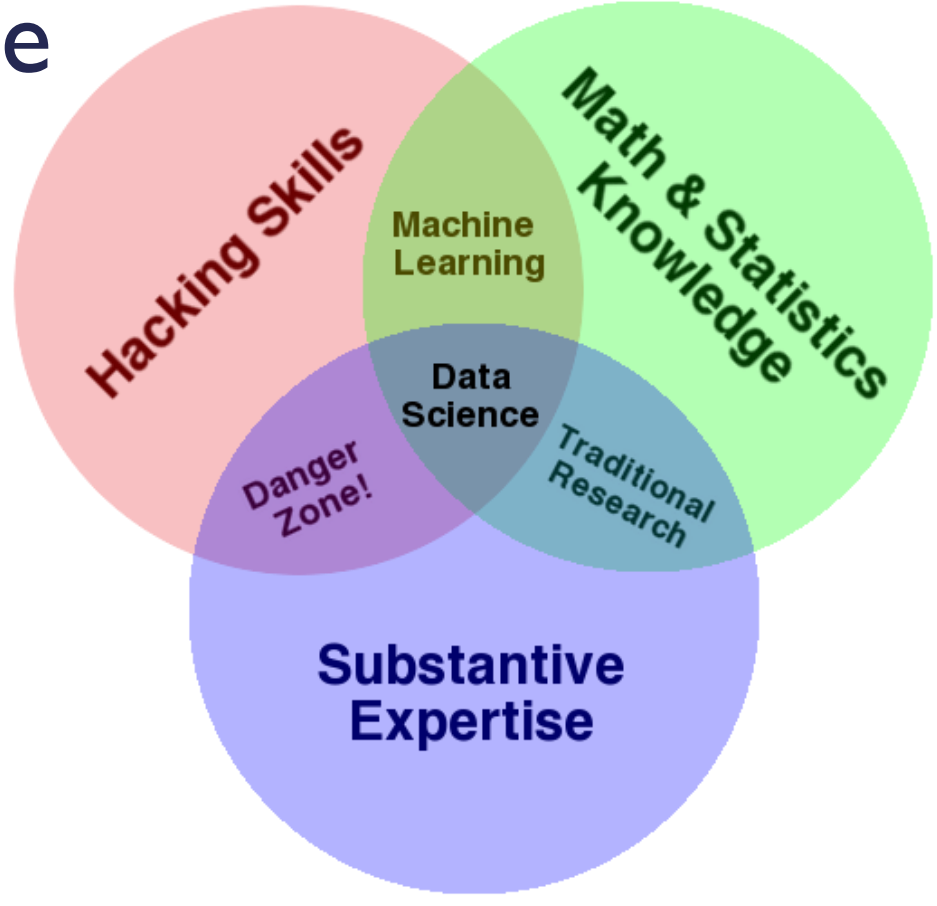


Figure source: Drew Conway

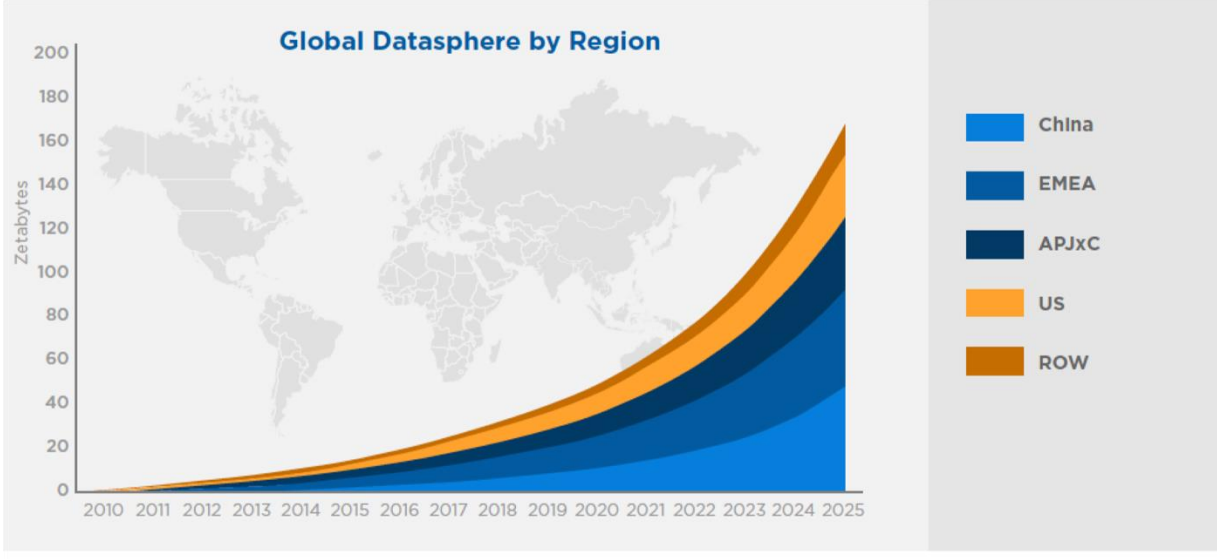
Data is produced at an increasing rate

1ZB is 1E21 Bytes
1ZB is 10^9 TB
1TB is a typical HDD size

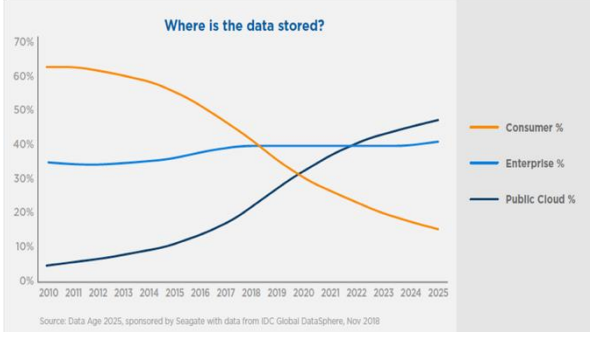
Prefixes for multiples of bits (bit) or bytes (B)

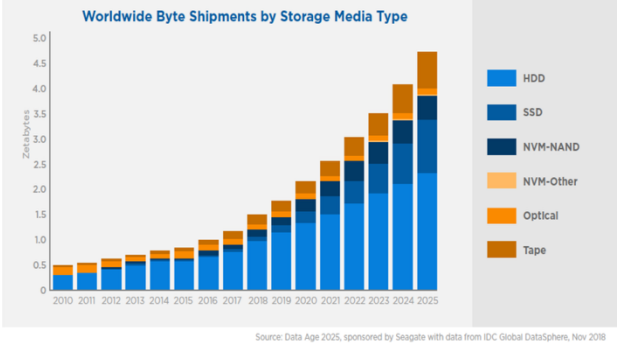
Decimal			Binary			
Value	SI		Value	IEC		JEDEC
1000	k	kilo	1024	Ki	kibi	K kilo
1000^2	M	mega	1024^2	Mi	mebi	M mega
1000^3	G	giga	1024^3	Gi	gibi	G giga
1000^4	T	tera	1024^4	Ti	tebi	–
1000^5	P	peta	1024^5	Pi	pebi	–
1000^6	E	exa	1024^6	Ei	exbi	–
1000^7	Z	zetta	1024^7	Zi	zebi	–
1000^8	Y	yotta	1024^8	Yi	yobi	–

Source:Wikipedia V • T • E



Source: IDC's Data Age 2025 study, sponsored by Seagate



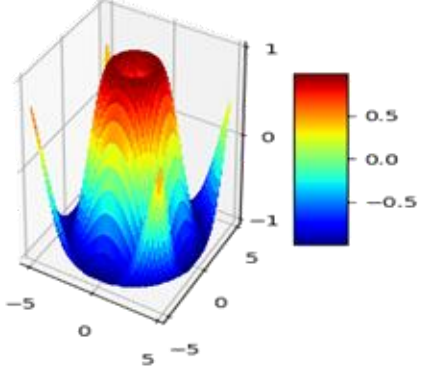


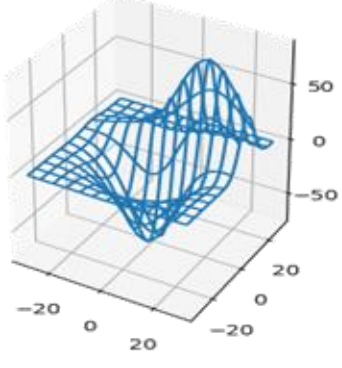
WEEK 1 TOPICS

Numerical Calculations and Visualization

- NumPy
  - Vectors and matrices (Math 235 – Linear Algebra)
  - Numerical calculations on large data sets

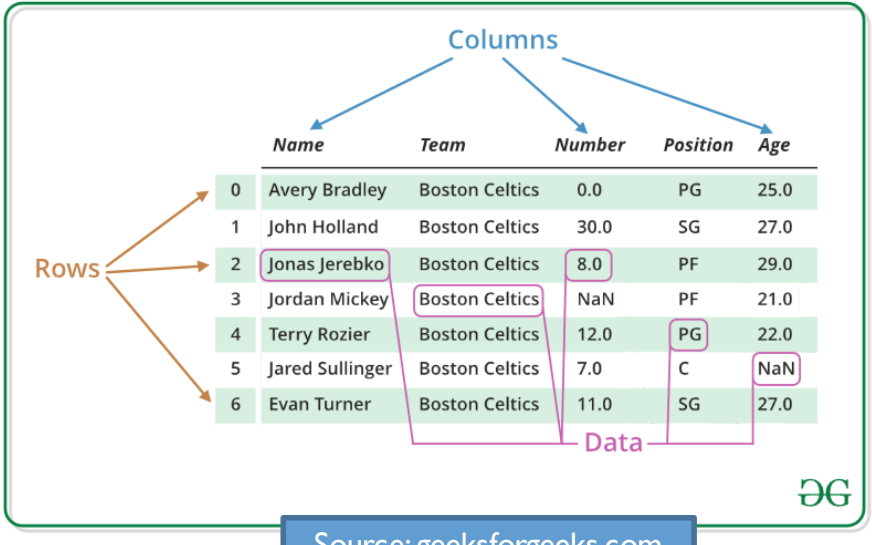
- Matplotlib



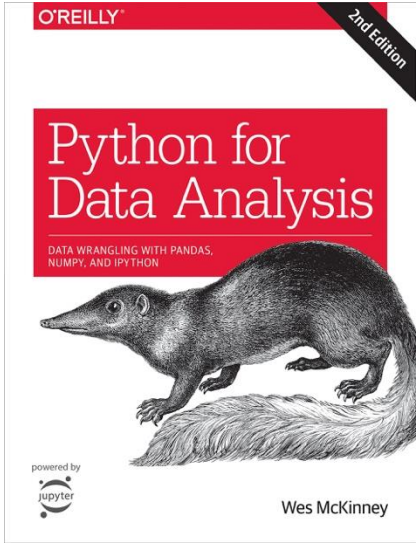


Pandas Dataframes and Dataseries

- Software library written by Wes McKinney ~2008
- Versatile datastructure integrated with NumPy and Matplotlib



Source: geeksforgeeks.com



Data Wrangling

- A large portion of a data scientists time will be spent processing, cleaning, and tranforming datasets
- Missing values, noisy data, etc.

- Stages of data pre-processing:
  - Cleaning
  - Integration
  - Transformation
  - Reduction
  - Discretization

#	Country	Alcohol	Deaths	Heart	Liver
1	Australia	2.5	785	211	15.30000019
2	Austria	3.000000095	863	167	45.59999847
3	Belg/Lux	2.900000095	883	131	20.70000076
4	Canada	2.400000095	793	NA	16.39999962
5	Denmark	2.900000095	971	220	23.89999962
6	Finland	0.800000012	970	297	19
7	France	9.100000381	751	11	37.90000153
8	Iceland	-0.800000012	743	211	11.19999981
9	Ireland	0.699999988	1000	300	6.5
10	Israel	0.600000024	-834	183	13.69999981
11	Italy	27.900000095	775	107	42.20000076
12	Japan	1.5	680	36	23.20000076
13	Netherlands	1.799999952	773	167	9.199999809
14	New Zealand	1.899999976	916	266	7.699999809
15	Norway	0.0800000012	806	227	12.19999981
16	Spain	6.5	724	NA	NA
17	Sweden	1.600000024	743	207	11.19999981
18	Switzerland	5.800000191	693	115	20.29999924
19	UK	1.299999952	941	285	10.30000019
20	US	1.200000048	926	199	22.10000038
21	West Germany	2.700000048	861	172	36.70000076

Jupyter Notebooks

- Jupyter notebooks will be used for our projects
- Allows for code and formatted text to be interwoven providing a readable document

Jupyter Notebook Introduction

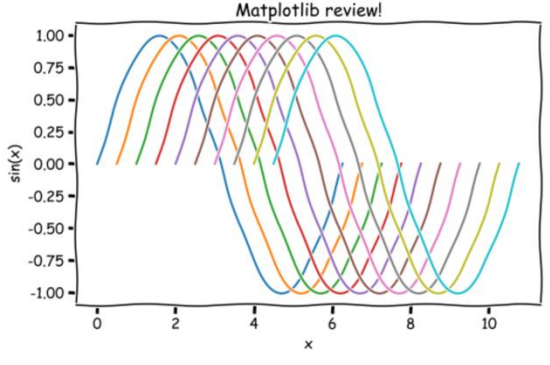
In [1]:
```
print("Hello World!")
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
x = np.linspace(0, np.pi*2, 100)
y = np.sin(x) # Be aware of what would happen if you reran just this cell (Nothing, in
```

In [5]:
```
plt.figure(figsize=(8,5))
plt.xkcd() # Built in XKCD theme!
for i in range(10):
    plt.plot(x+i*.5,y)
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.title("Matplotlib review!")
```

Out[5]:
```
Text(0.5, 1.0, 'Matplotlib review!')
```



Opening files

- You can use the built-in function open() to open files
  - Provide the filename and arguments that specify read mode
- Or use with open() as
  - Automatically closes filehandle

```
file = open("data/weather_04_21-28.csv", "r")
#or
with open("data/weather_04_21-28.csv", "r") as file:
```

Iterating over the file (Basic)

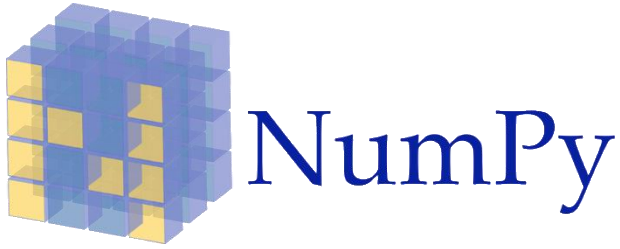
- You can iterate over the lines in a file
  - Lines in a file are delimited by newlines (\n)
- Once you have each line you can split by the **comma separated values (.csv files)**
- What is the data type of words?
- What is the data type of temperature?

```
file = open("data/weather_04_21-28.csv", "r")
for line in file:
    line = line.rstrip()
    words = line.split(",")
    date = words[0]
    temperature = words[1]
    humidity = words[2]
```

```
1  2019-04-21T22:10,58.800,61.700
2  2019-04-21T22:20,58.800,63.700
3  2019-04-21T22:30,58.700,64.700
4  2019-04-21T22:40,58.600,64.700
5  2019-04-21T22:50,58.400,64.700
```

First 5 lines of weather_04_21-28.csv

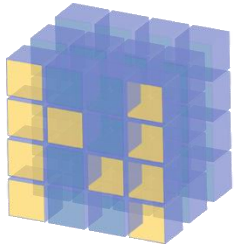
Numpy Package



- Modules
  - **`numpy`** http://www.numpy.org

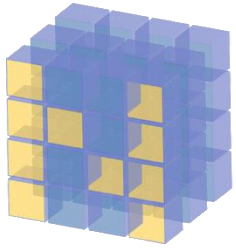
    https://docs.scipy.org/doc/

Why do we use Numpy?



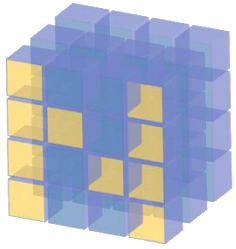
- Python has lists built in
  - Still lists! Not arrays
  - Slow for large datasets
- Numpy is used for:
  - Linear Algebra
  - Matrix computation
  - Numerical analysis
  - Lingua franca of scientific computing and data science libraries (Pandas, scipy, etc.)

Why do we use Numpy?



- Still not convinced?
- Work with large enough datasets and you will be!
  - Numpy array operations are done in compiled C/fortran libraries
  - Working with lists in python are significantly slower for large datasets
    - Benchmark it if you don't believe me!

NumPy Arrays



NumPy

- NumPy arrays have operators overloaded to perform element-wise functions
- */+- work with scalar values (float/int) and arrays of the same length

```
>>> import numpy as np
>>> array_a = np.array([1,2,3])
>>> array_b = np.array([4,5,6])
>>> array_a*5
array([ 5, 10, 15])
>>> array_a+5
array([6, 7, 8])
>>> array_a*array_b
array([ 4, 10, 18])
>>> array_a
array([1, 2, 3])
>>> array_a = array_a*5
>>> array_a
array([ 5, 10, 15])
>>>
```

Helpful NumPy Stuff

Initialization

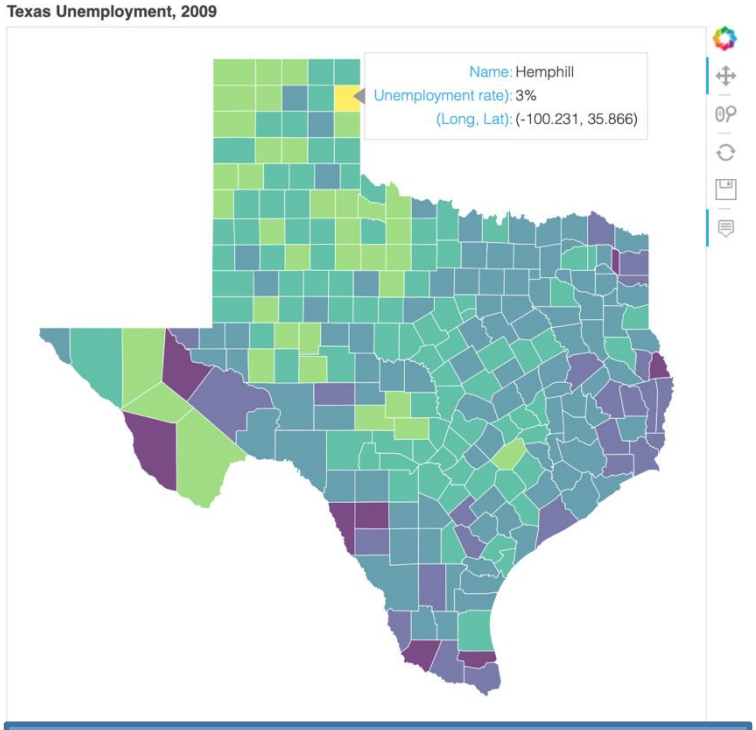
- np.zeroes()
- np.ones()
- np.linspace()
- np.arange()
- Or np.array([python list])

Operations

- */+- operations work with scalars and vectors (arrays)
- np.sum(), np.sqrt(), np.sin(), np.cos(), np.exp(), np.log()
- Boolean logical indices
- Array < scalar returns a True/False array
- Can use Boolean array to select values and return a new array

Data Visualization in Python

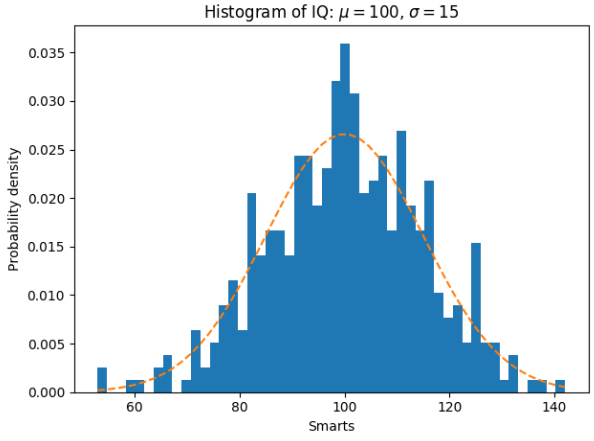
- There are many plotting packages available in python which makes it an excellent choice for any data scientist
- Packages include Matplotlib, Bokeh, ggplot, plotly, Seaborn, etc.
- We will focus on matplotlib for today's example
  - De facto standard scientific plotting library

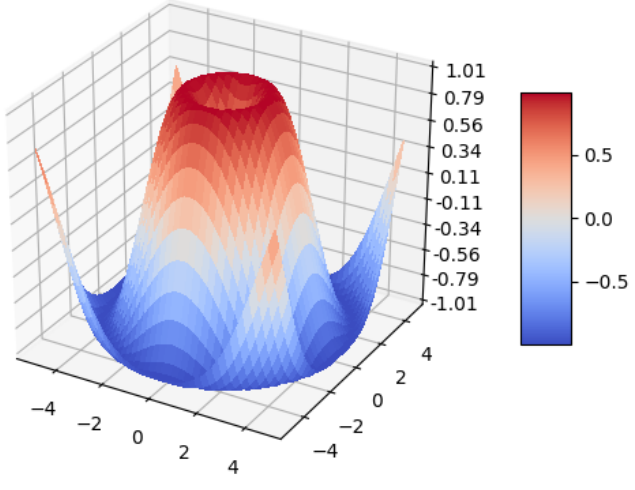


Packages like Bokeh can produce stunning interactive plots

Matplotlib

- Scientific plotting library for Python
- Based on matlab syntax
- Many options for data visualization: Histograms, line charts, scatter plots, 3d plots, etc.
- Originally released in 2003 and was written by John D. Hunter





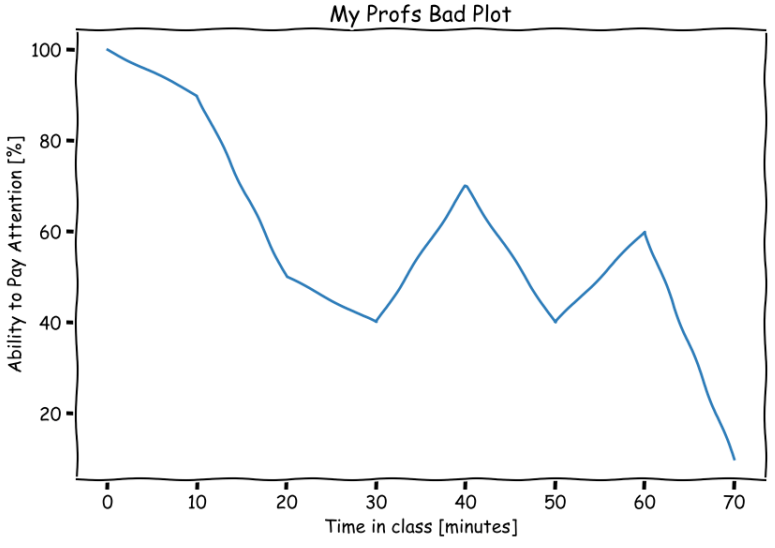
How to make a simple plot in matplotlib

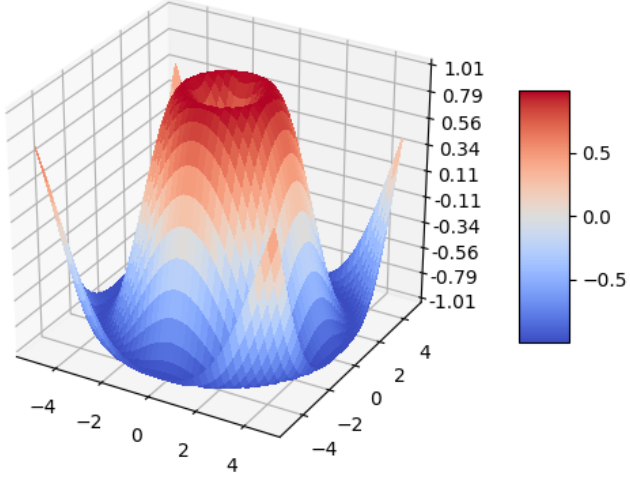
- Import matplotlib.pyplot as plt
- Here we are plotting two lists
- Samples is a list that contains integers [1,2,3,4,5,6…]
- Temperature_list is a list of temperatures that we have created by appending to an empty list
- plt.plot(x,y) is the simplest way to plot arrays of x and y values
- plt.show( ) makes the plot appear

```
fig = plt.figure()
plt.plot(samples, temperature_list)
plt.ylabel("Temperature [Deg. F]")
plt.xlabel("Date")
plt.show()
```

Why use matplotlib?

- I can just use excel to make plots
  - Try making 1000 of them!
- De facto standard scientific plotting library
- Used by nearly all data scientists (at least to get started)





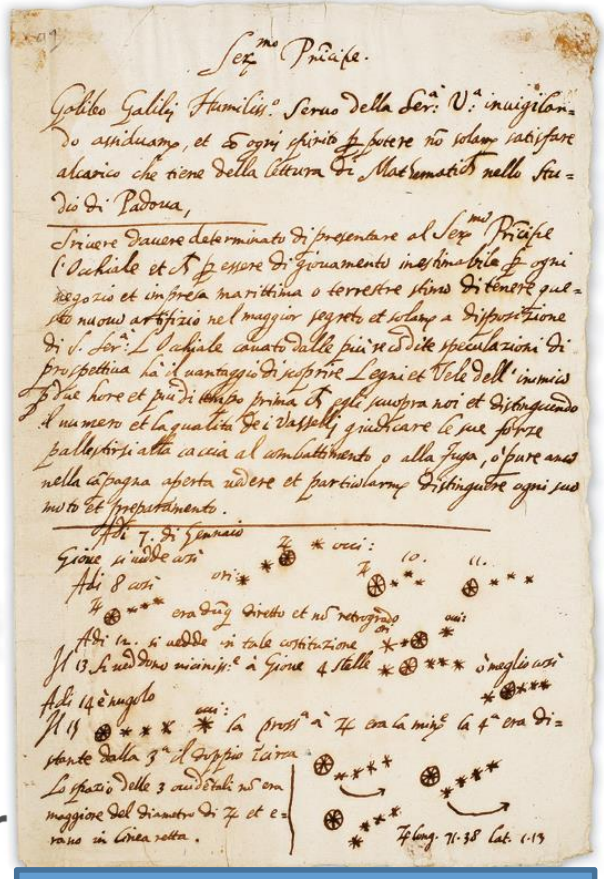
XKCD Theme…

3D Visualization

JUPYTER NOTEBOOKS

Jupyter Notebooks

- Jupyter notebooks allow interactive manipulation of data in which code and notes can be interspersed
- The Jupyter project was founded in 2015 and the name was derived for Julia, Python, and R
  - Many more languages are now supported even allowing custom "Kernels"
  - Founded by Fernando Pérez (physicist)
- Originates from the IPython (2001) interpreter
- Earlier origins are less clear but likely go back to Mathematica

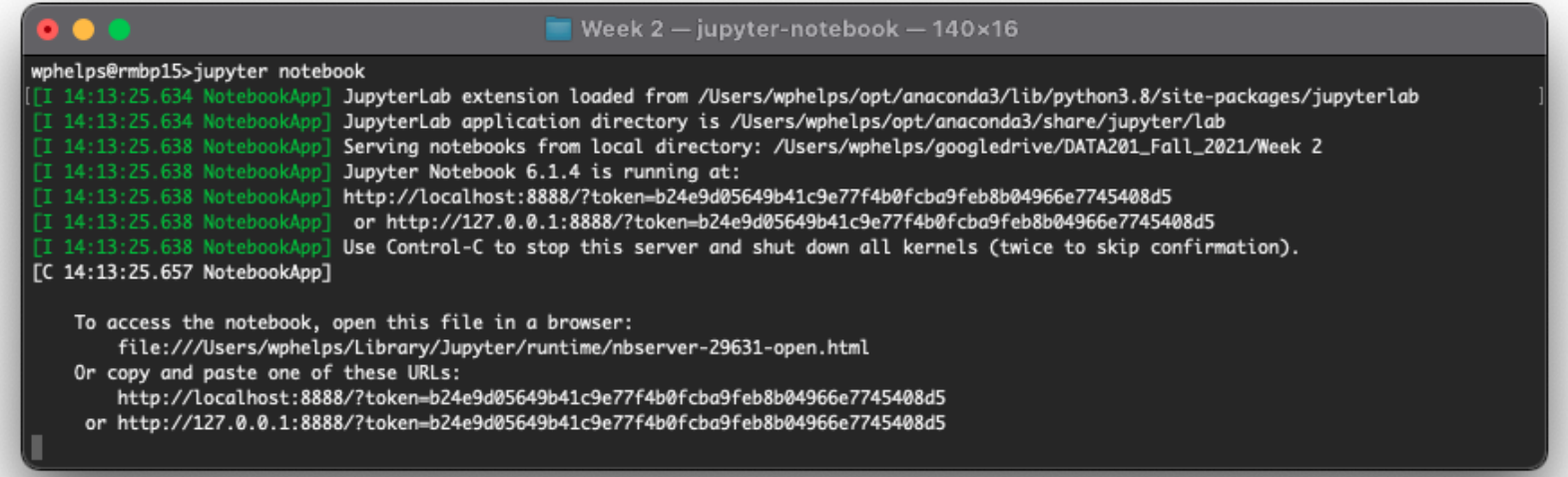


Galileo's Manuscript - 1610

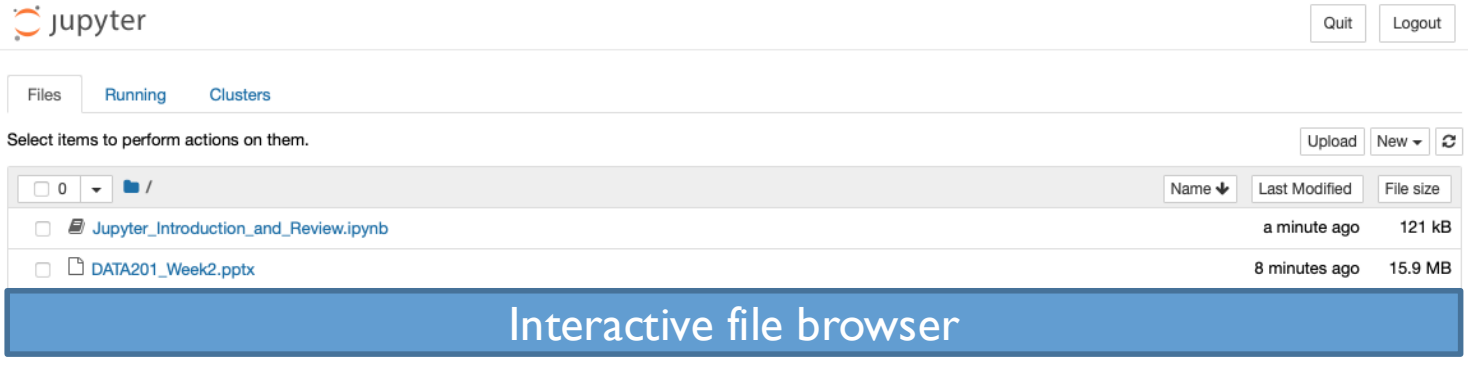
Jupyter Notebooks

- You can run them locally
  - If you execute "jupyter notebook" in the command line you will start a jupyter notebook server
  - Your default web browser should open up to a file browser
  - Pre-bundled with anaconda
  - You can also use Jupyter Lab or use PyCharm for your notebooks
- Google Colab
  - Excellent for free/cheap access to GPUs and even TPUs (may be used later for our Machine Learning projects)
- Amazon Web Services (AWS)

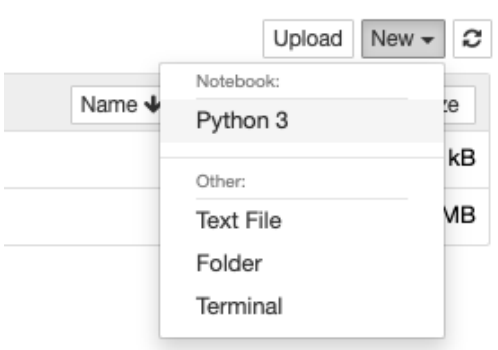
Start Jupyter Notebook



File browser

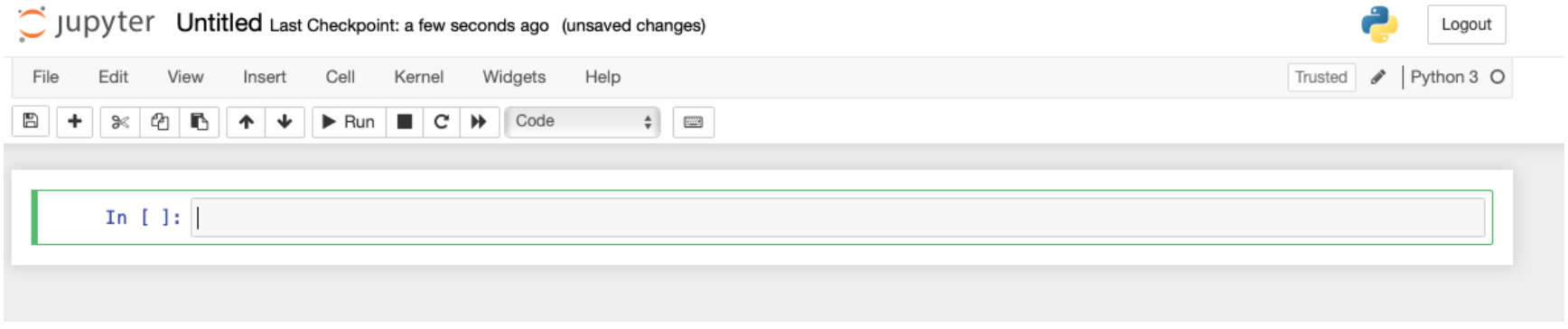


Interactive file browser



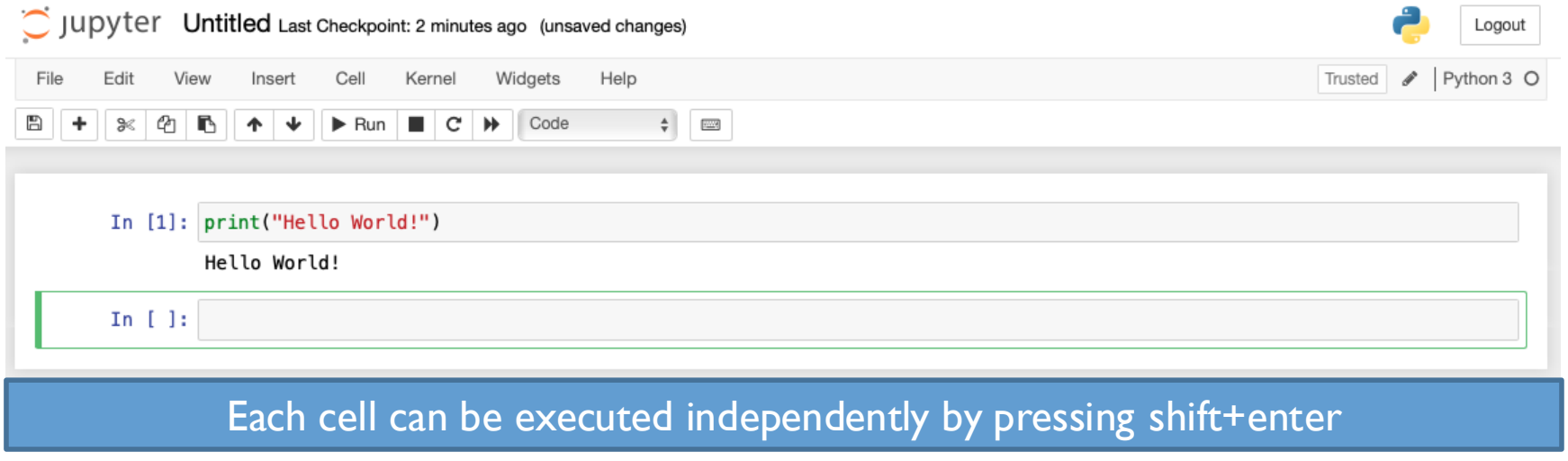
Open a new notebook

My first jupyter notebook



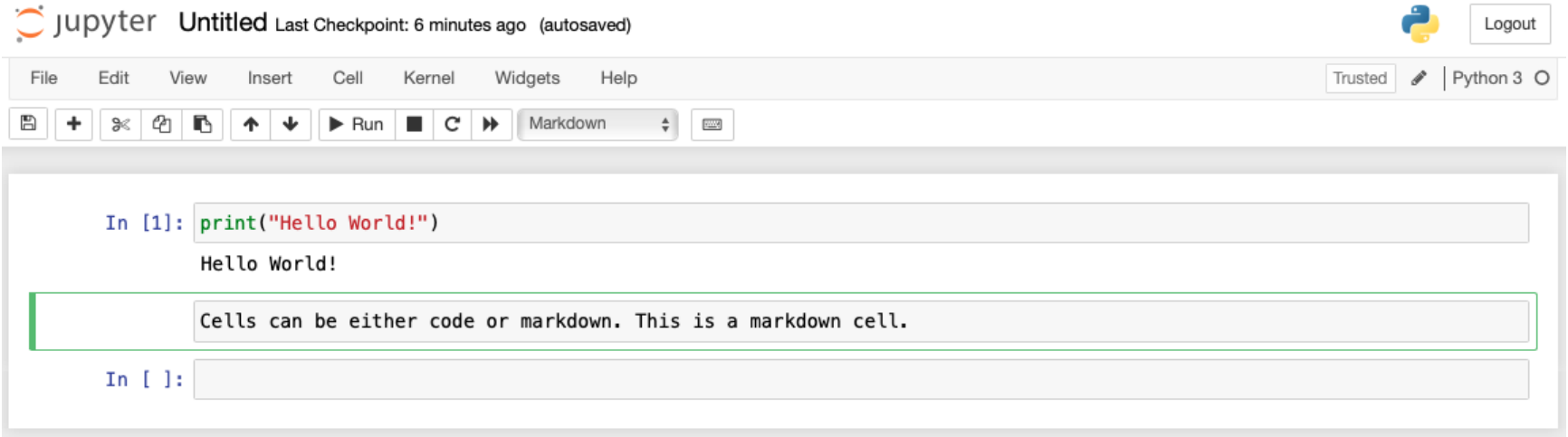
Cells can be either code or markdown cells

Hello World!



Each cell can be executed independently by pressing shift+enter

Adding a Markdown Cell



Wait. What is markdown?

- Markdown is a lightweight markup language for creating formatted text
- Created in 2004 by John Gruber and Aaron Swartz
- Headings, lists, links, embed images, code snippets w/highlighting

Demo Time

- Let us try a few things out interactively!
- If you do not have anaconda on your machine you should use a lab machine!
- Use NumPy to generate a dataset
  - Boolean logical indices
  - Array < scalar returns a True/False array
  - Can use Boolean array to select values and return a new array
- Matplotlib plot!
- Markdown!

Questions?

PYTHON REVIEW

Variables and assignment statements

Assume each statement is executed in Python following prior statements
For valid assignments, what is shown if we "print(x)"? (write on your paper)

- ✓ x = 10
- ✓ y = 4
- ✓ name = "Captain Chris"
- ✓ z = x*x + y*y
- ✗ x + y = z — Assign the right-hand side to a **single** variable on the left-hand side
- ✗ 42 = x — SyntaxError: can't assign to literal --> only variable names on left side
- ✓ z = x + 42
- ✗ z = name + x — TypeError: can only concatenate str (not "int") to str
- ✓ z = name + str(x)
- ✓ x42 = 13 — x42 is a perfectly valid variable name
- ✗ 42x = 13 — 42x is a NOT valid variable name; must start with character or underscore

Objects

- Object = Instances of "things" created by the Python interpreter
- Each object has:
  - Value
  - Type
  - Identity (***Unique*** identifier)
    - "loosely speaking" a memory address, "reference" in Java; pointer in C++

```
>>> num = 2019
>>> num_as_string=str(num)
>>> type(num)
<class 'int'>
>>> type(num_as_string)
<class 'str'>
>>> id(num)
4464478832
>>> id(num_as_string)
4465191432
```

- Object instances can be bound to names (variables)
- Objects can be ***mutable*** or ***immutable***

Data Types

- **int** -1, 0, 42
- **float** 0.0, -1.0, 3.1415927
- **str** (String)
  - Use " or ' to delimit

- **Lists**: list0 = [1, 2, 3, 3, 2] ordered list
  - list0[1] = "hello" sequential access by index
  - [1, "hello", 3, 3, 2] we can mix types!
- **Sets**: a= {1, "hello", 2, 3} or set(list0) = {1, "hello", 2, 3}
  - - Unordered, unique elements
- **Dictionaries** contains key : value pairs

```
players = {'Lionel Messi': 10, 'Cristiano Ronaldo': 7}
Usage: "players[key] = value"
```

Type conversions

- Numbers
  - 1 + 2 returns an integer type.
  - 1 + 2.0 returns a float type.
  - 1.0 + 2.0 returns a float type
- Conversion methods

Function	Notes	Can convert:
int()	Creates integers	int, float, strings w/ integers only
float()	Creates floats	int, float, strings w/ integers or fractions
str()	Creates strings	Any

e.g. x = float(input("Type number here: ")) type(x) is float

String Formatting

- **f-strings (e.g. `f"I am {age} years old."`)**
  - There are several other formatting methods that I am excluding here for brevity
- String methods will be important (See links below)
  - Ex: find, replace, upper, lower, capitalize

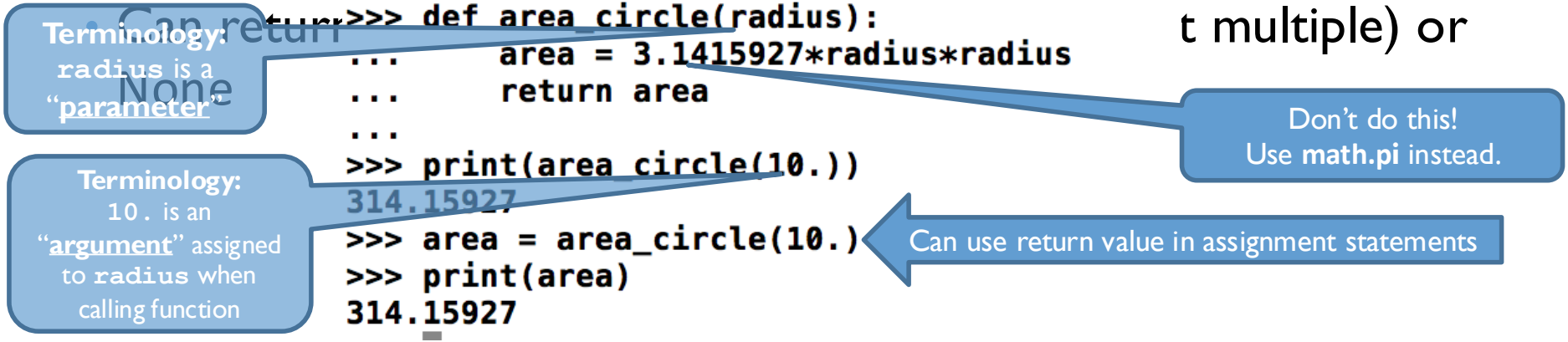
https://docs.python.org/3/library/string.html

https://docs.python.org/3/library/stdtypes.html#str.format

See the "Format examples" section of string

Functions

- Define using def keyword and () :
- Indent the entire body of the function
- May define zero or more input parameters inside the ( ) in definition
- Can return None t multiple) or



Note: "parameter" and "argument" are often informally used interchangeably.

Branching

- If, if-(elif)-else
  - Checks a Boolean (True/False) conditional
  - The else is NOT required
  - Needs a : at end of each condition
  - Indent all code to be executed by a conditic
- Boolean conditionals
  - == (double =) asserts two items are equal; returns True or False
    - Comparing ints and Strings are as expected
    - Don't compare floats this way (due to minor rounding differences)
    - Many other objects ask if they have same identifier (id) not same value!
  - != asserts two items are NOT equal

```
>>> a= 10
>>> if (a == 9):
...     print("Is 9")
... elif (a == 10):
...     print("Is 10")
... else:
...     print("Not 9 or 10")
...
Is 10
>>>
```

Boolean Relational Operators

Relational operators	Description	Example (assume x is 3)
<	a **<** b means a is less than b	x < 4 is true x < 3 is false
>	a **>** b means a is greater than b	x > 2 is true x > 3 is false
<=	a **<=** b means a is less than or equal to b	x <= 4 is true x <= 3 is true x <= 2 is false
>=	a **>=** b means a is greater than or equal to b	x >= 2 is true x >= 3 is true x >= 4 is false

Boolean Operators

Boolean operator	Description
a **and** b	***Boolean AND***: True when both operands are True.
a **or** b	***Boolean OR***: True when at least one operand is True.
not a	***Boolean NOT*** (opposite): True when the single operand is False (and False when operand is True).

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Let x = 7, y = 9

(x > 0) and (y < 10) True
True True

(x > 0) and (y < 5) False
True False

(x < 0) or (y > 10) False
False False

(x < 0) or (y > 5) True
False True

not (x < 0) True
False

not (x > 0) False
True

Python Style - Naming

- https://www.python.org/dev/peps/pep-0008/
- Identifier (variable name)
  - lower_case_alphanumeric_separated_by_underscores (also called "snake_case")
  - Python is case sensitive, but prefer lowercase
  - Must start with letter or underscore (not a number)
  - Double underscore has special meaning (name mangling used with "private" class variables)
- Readability is key
  - "Code is more often read than written." *Guido Van Rossum (inventor of Python)*
  - Use meaningful names
- Cannot use keywords (e.g. if, else, True, False, and, as, in, …)
  - Cheat by adding a trailing underscore (e.g. in_)

Python Style - Indentation

- https://www.python.org/dev/peps/pep-0008/
- Python uses blocks of indented code to specify scope
  - Other languages (e.g. Java, C, C++) use { }
  - Prefer 4 spaces per indention
  - Configure your editor to insert 4 spaces when you hit tab key
  - Never mix tabs and spaces

```
print("Temp: "+str(temp))
if (temp > 87.):
    print("That's hot")
print("done!")
print("-----")
```

```
print("Temp: "+str(temp))
if (temp > 87.):
    print("That's hot")
    print("done!")
print("-----")
```

Python Style - Comments

- https://www.python.org/dev/peps/pep-0008/
- Prefer clearly named and clearly functioning self-documenting code
- But more comments than you think necessary at the time
  - "Code tells you how; Comments tell you ***why***." *Jeff Atwood (aka Coding Horror blog; StackOverflow)*
  - You'll forget in a month what your clever statement was intended to do
- But not too much
  - Comments will need to be corrected and maintained as well
  - Too much is rarely an issue, unless doing in-code documentation that can lead to bloat
- Use # for inline comments
- Use # for each line of block comments, a lonely # on paragraph separation lines
- Use """ for document strings (help and method descriptions)
  - https://www.python.org/dev/peps/pep-0257/

Boolean membership checks

- in, not in for strings, lists, sets, and tuples

```
>>> a=[1,2,3]
>>> 3 in a
True
>>> 4 in a
False
>>> 5 not in a
True
```

- Also checks to see if a KEY is in dictionary
  - Does not check for values!

Boolean Identity Operators

- is, is not for objects
  - Checks if the objects have the same identifier (id) (i.e., same instance)

```
>>> s1="Hello"
>>> s2 = s1
>>> id(s1)
4465191600
>>> id(s2)
4465191600
>>> s1 is s2
True
>>> s1 is not s2
False
```

Order of Operations

- PEMDAS from high school algebra
  - Parenthesis, exponents, multiplication, division, addition, subtraction

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first.	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next.	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	(% is discussed elsewhere.)
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Order of Operations

- PEMDAS from high school algebra
  - Parenthesis, exponents, multiplication, division, addition, subtraction

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In (a * (b + c)) - d, the + is evaluated first, then *, then -.
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	z - 45 * y < 53 evaluates * first, then -, then <.
< <= > >= == !=	Relational, (in)equality, and membership operators	x < 2 or x >= 10 is evaluated as (x < 2) or (x >= 10)because < and >= have precedence over or.
not	not (logical NOT)	not x or y is evaluated as (not x) or y
and	Logical AND	x == 5 or y == 10 and z != 10 is evaluated as (x == 5) or ((y == 10) and (z != 10)) because and has precedence over or.
or	Logical OR	x == 7 or x < 2 is evaluated as (x == 7) or (x < 2) because < and == have precedence over or

Loops

```
while expression: # Loop expression
                  # Loop body: Sub-statements to execute
                  # if the loop expression evaluates to True

# Statements to execute after the expression evaluates to False
```

- While
  - While some condition is true, do the code block
  - Use if you don't know how much/long to process
    - E.g. depends on user input
  - Something inside the block must change the condition; otherwise infinite loop
  - Termination condition must be reachable
- For
  - For specific interval, do something in the code block
  - Commonly used with lists and tuples, or range to get index

Prefer a **`for`** loop if you know how many times something is **expected** to run

for <each> in <Container>:

- Works with list, tuple, or string (or any iterable)

```
for name in ['Bill', 'Nicole', 'John']:
        print 'Hi %s!' % name
```

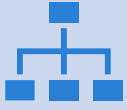
for variable **in** container:
    # Loop body: Sub-statements to execute
    # for each item in the container

Statements to execute after the for loop is complete

Great for accessing data in list, but
You cannot change the item in list this way.
Use range() and index if you need to modify the list.

DATA TYPES AND SOURCES

Data types



Structured

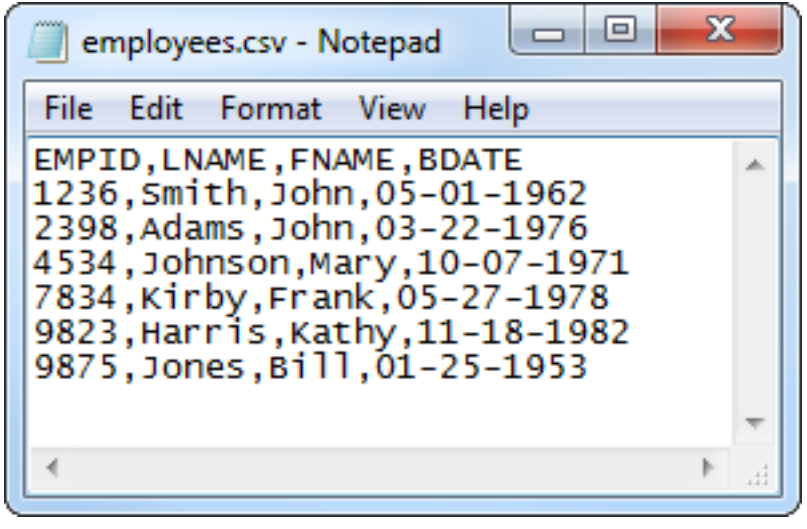
Example: tables

Unstructured

Example: free text

Structured Data

- Organized, defined structure
- Fits into conventional databases
- CSV file is one of the most common formats



Structured data example

custid	sex	is.employed	income	marital.stat	housing.type	num.vehicles	age	state.of.res
2068	F	NA	11300	Married	Homeowner free and clear	2	49	Michigan
2073	F	NA	0	Married	Rented	3	40	Florida
2848	M	TRUE	4500	Never Married	Rented	3	22	Georgia
5641	M	TRUE	20000	Never Married	Occupied with no rent	0	22	New Mexico
6369	F	TRUE	12000	Never Married	Rented	1	31	Florida

Unstructured Data

- How most of the world's data is!
- Videos, images, tweets, natural language
- No one right answer for how to process!

Unstructured Data Examples

Comparison

E. Ford - 10.1136/medethics-2019-105472

	Examples	
Data item	**Structured***	**Unstructured**
Time	dd/mm/yyyy 14 September 2017 Prescription start date: dd/mm/yy	'Earlier today Mr X experienced chest pain' 'Operation scheduled on Tuesday'.
Symptoms	N242300 Neuropathic pain 1B1B.00 Cannot sleep—insomnia	'…c/o shooting pain in upper right leg during the night, disturbing her sleep'
Diagnosis	C109912 Type 2 diabetes without complication	'The patient has diabetes without complications'
Prescription	01040200 (BNF code for codeine phosphate 60 mg tablets)	'Px codeine 60 mg PO qid×7 days'
Referral	8H4D.00 Referral to psychogeriatrician	Rev 4w ?refer pyscho ger
Test	43F1.00 Rheumatoid factor positive	Rheumatoid factor was 421U/mL which is a positive result

*These codes represent Read codes, a UK-based, alphanumeric clinical coding system for general practice, and British National Formulary (BNF) codes, which represent the full list of medications available in the UK.

Structured and Unstructured Data

Structured Data

- Pros
  - Easily analyzed using ML algorithms
  - Works nicely with conventional tools: pandas data frames, excel, etc.
- Cons
  - Restrictive schema
  - Any change to the format may require updating all entries

Unstructured Data

- Pros
  - You can use original file format: video, natural language, etc. You retain more information
  - No need to refine data to to a structured format
- Cons
  - Requires special tools and analysis techniques. E.g. YOLO Models, LLM to structured output

Data collections

Lots of places that host/share data online, or you can collect them yourself.

Open data collections

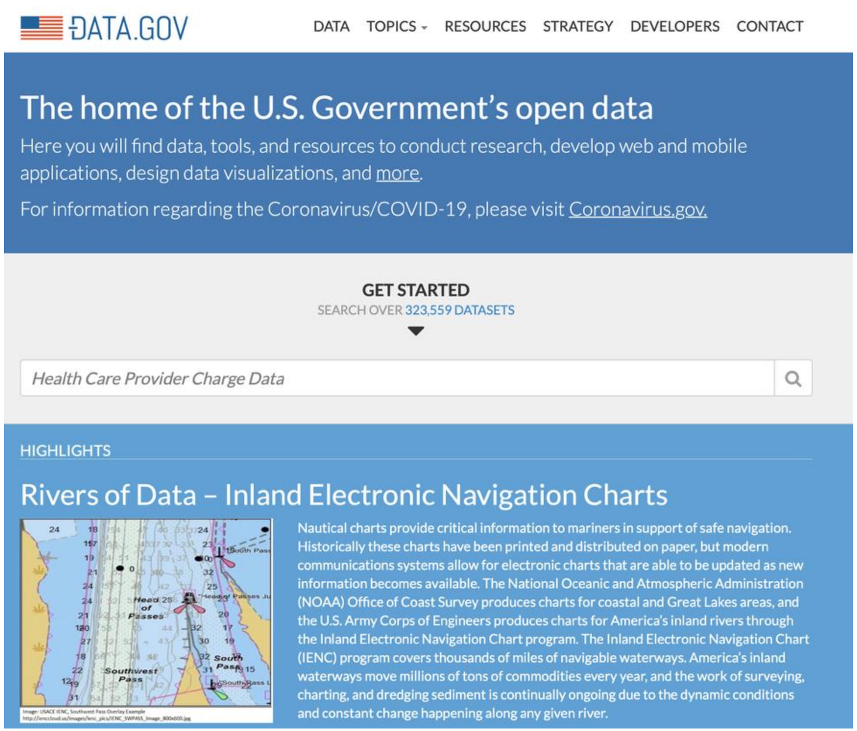
Social media data

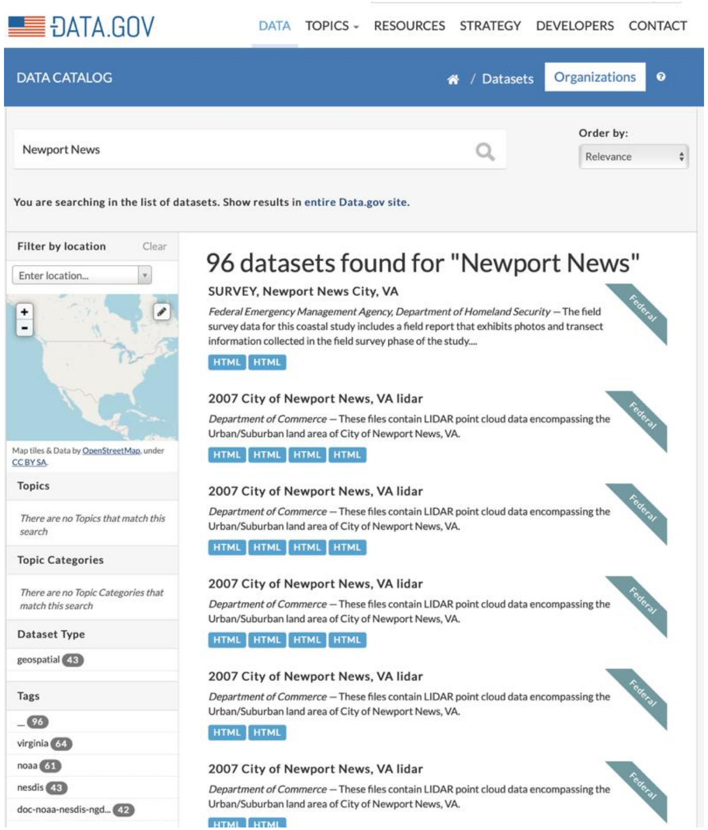
Multimodal data

Open Data Initiative

- In 2013 there was an executive order signed that *"made open and machine-readable data the new default for government information. Making information about government operations more readily available and useful is also core to the promise of a more efficient and transparent government."*

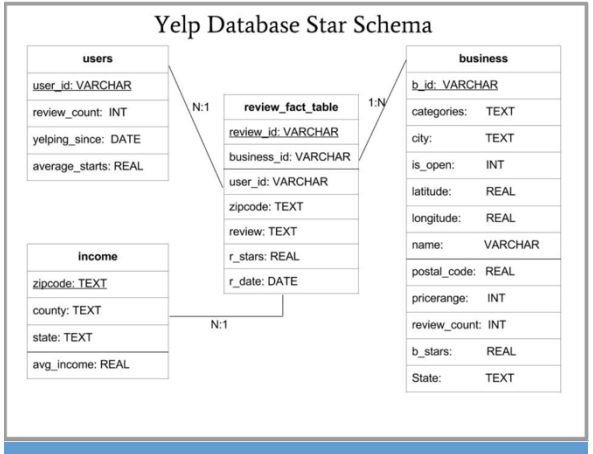
US Government Data – More examples



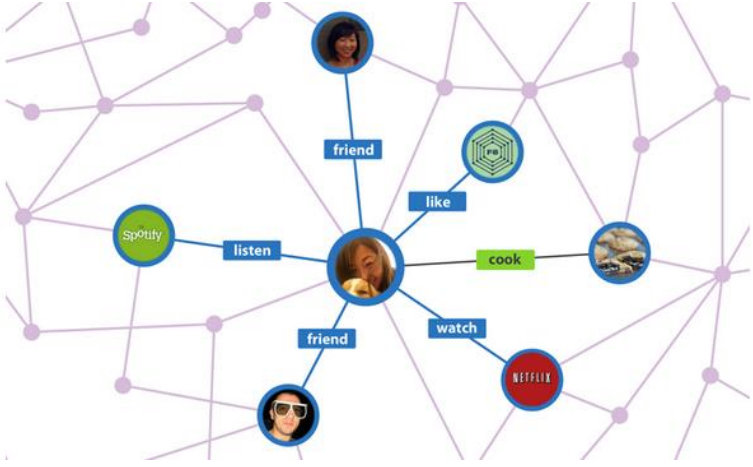


Examples of Data Sources

- Social Media – Facebook Graph API
- Yelp API



Source: medium.com



Source: techcrunch.com

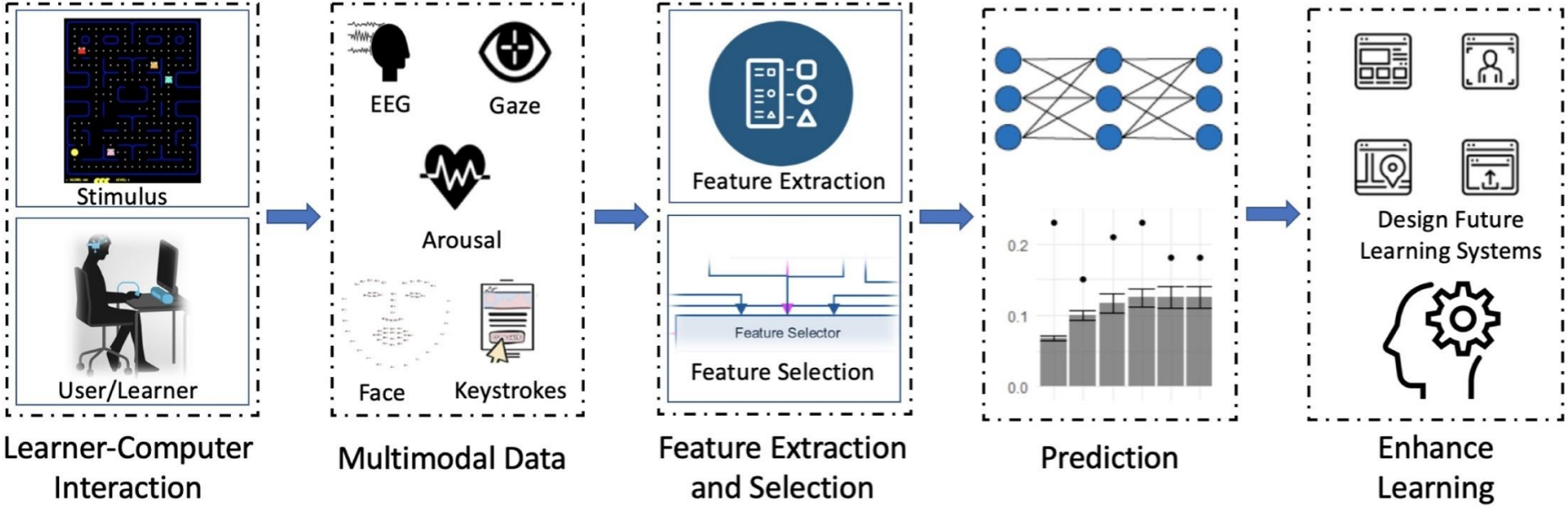
twitter

facebook

LinkedIn

reddit

Multimodal Data



https://doi.org/10.1016/j.ijinfomgt.2019.02.003

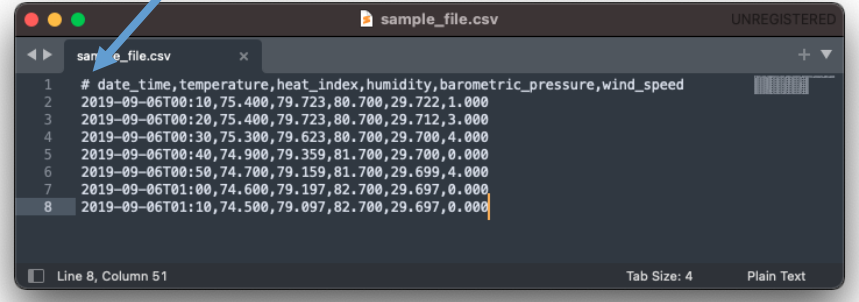
Structured data file types

- CSV (Comma Separated Values)
- TSV (Tab Separated Values)
- XML (eXtensible Markup Language)
- RSS (Really Simple Syndication)
- JSON (JavaScript Object Notation)

CSV and TSV Files

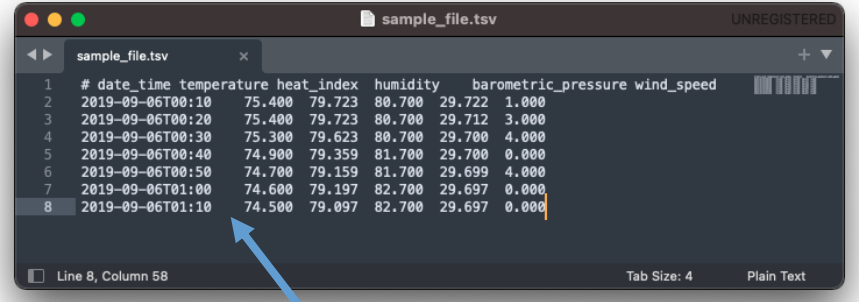
CSV File with Header

Headers may be preceded by "#"



- Standard - RFC 4180 (~2005)
- However, in practice the standard may not be followed and is application dependent

TSV File with Header



\t tab symbol rendered as spaces

- Same as CSV but with tabs!
- Tabs tend to have fewer collisions, easier to read, etc.

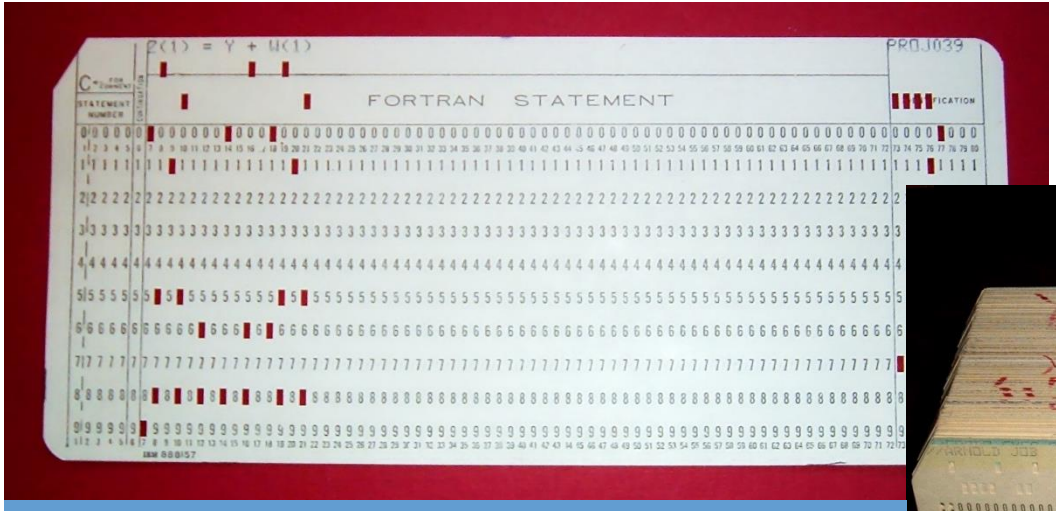
History Moment

- CSV has been around (essentially) since the beginning of time
  - Pre-dates personal computers
  - PCs were created around in the 1970s (Altair 8800, Apple I, Commodore PET, TRS-80)
    - First computers that we would recognize were from ~1980
- IBM Fortran first supported CSV in 1972
- Origin of the name Comma separated values and CSV abbreviation became common in early 1980's

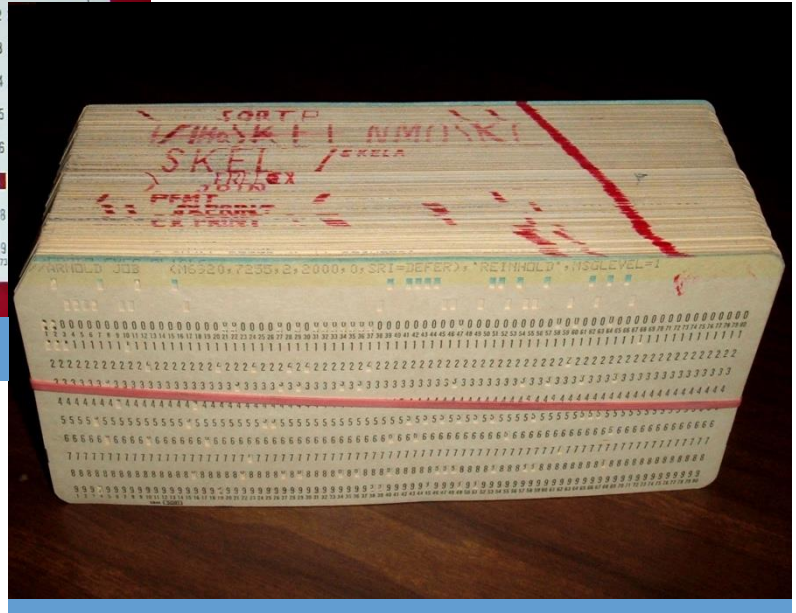
Altair 8800

TSA Approved Apple I

CSV Existed when these were used!



A Fortran Punch Card



Your Capstone Project?

Back to Files: XML

- Extensible Markup Language (XML)
- Published as a W3C recommendation in 1998
- Provides a human and machine-readable format
- Format looks similar to HTML but uses custom tags
- Ubiquitous across many sources

72 lines (64 sloc) | 2.29 KB

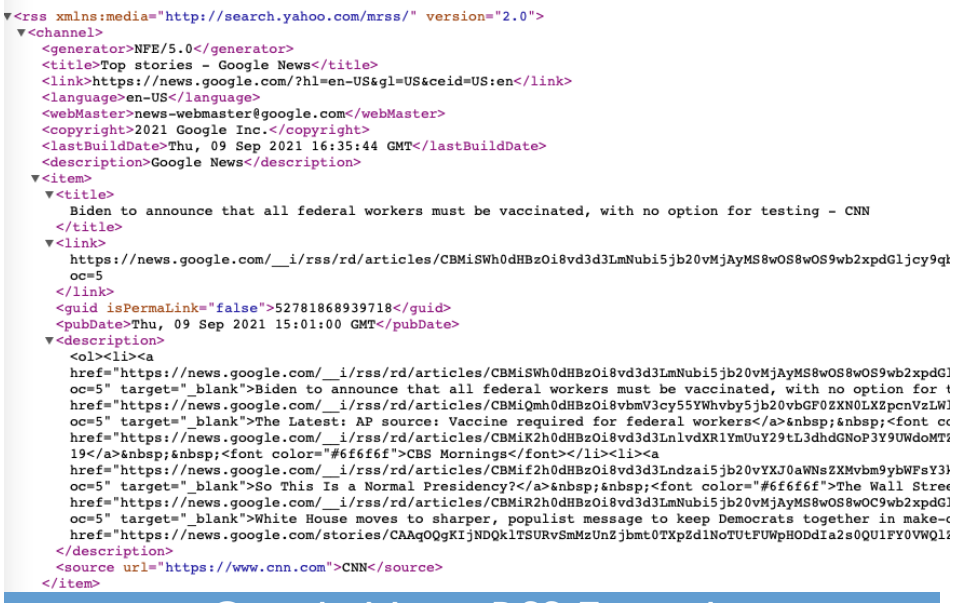
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.jlab.clas12</groupId>
    <artifactId>clas12-analysis</artifactId>
    <version>0.3-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
    <repositories>
        <repository>
            <id>jitpack.io</id>
            <url>https://jitpack.io</url>
        </repository>
        <repository>
            <id>freehep</id>
            <url>https://java.freehep.org/maven2</url>
        </repository>
    </repositories>
```

Custom Tags

Maven POM Build file for clas12-analysis library

RSS

- Really Simple Syndication
- Used to share data between services
  - News sites
  - Companies
  - Calendars
  - Site changes
- Follows the XML standard but follows a standard with required/optional tags
- See for example: https://news.google.com/rss



Google News RSS Example

JSON

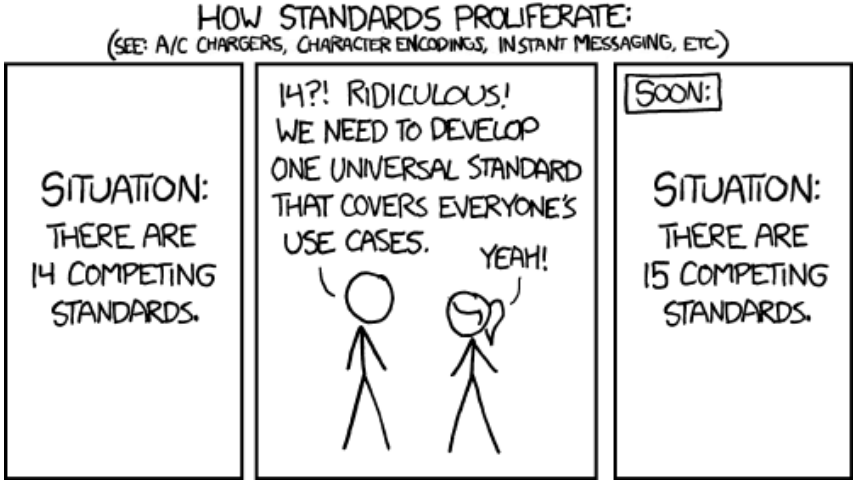
- Javascript Object Notation
- ECMA-262 standard (~1999)
- Format is based on two structures:
  - A collection of name – value pairs
  - Ordered list of values such as: Arrays, vectors, lists

```
{
    "name": "REC::Event",
    "group": 300,
    "item" : 30,
    "info": "Event Header Bank",
    "entries": [
            {"name":"category",   "type":"L", "info":"Undefined"},
            {"name":"topology",   "type":"L", "info":"Undefined"},
            {"name":"beamCharge", "type":"F", "info":"Beam charge, gated (nano-Coulomb)"},
            {"name":"liveTime",   "type":"D", "info":"Lifetime"},
            {"name":"startTime",  "type":"F", "info":"Event Start Time (ns)"},
            {"name":"RFTime",     "type":"F", "info":"RF Time (ns)"},
            {"name":"helicity",   "type":"B", "info":"Helicity of Event (+/-1, else undefined), with HWP-correction"},
            {"name":"helicityRaw","type":"B", "info":"Helicity of Event (+/-1, else undefined)"},
            {"name":"procTime",   "type":"F", "info":"Event Processing Time (UNIX Time = seconds)"}
        ]
},
```

Example of a Bank Definition used at Jefferson Lab

File Formats and Libraries

- There are many, many more formats than what we have covered
  - .yaml, .xls, etc.
- Do not write your own reader/writer unless you have to!
  - Example: We did write our own json reader/writer at Jlab to limit external dependencies. Our experiments run on a ~30 year timespan



https://xkcd.com/927/

Data preprocessing

Data cleaning

Data wrangling
Handling missing data
Smooth noisy data

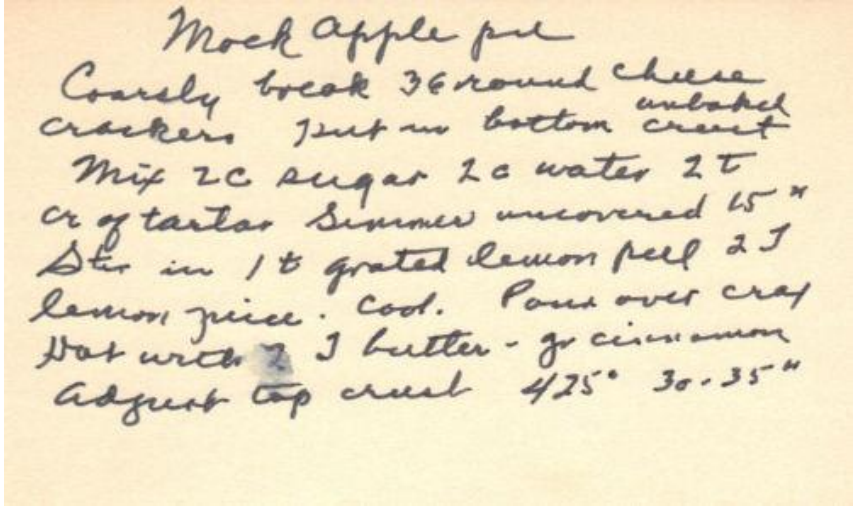
Data integration

Data transformation

Data reduction

Data discretization

Data Munging/Wrangling



Hand Written Recipe*

INGREDIENTS

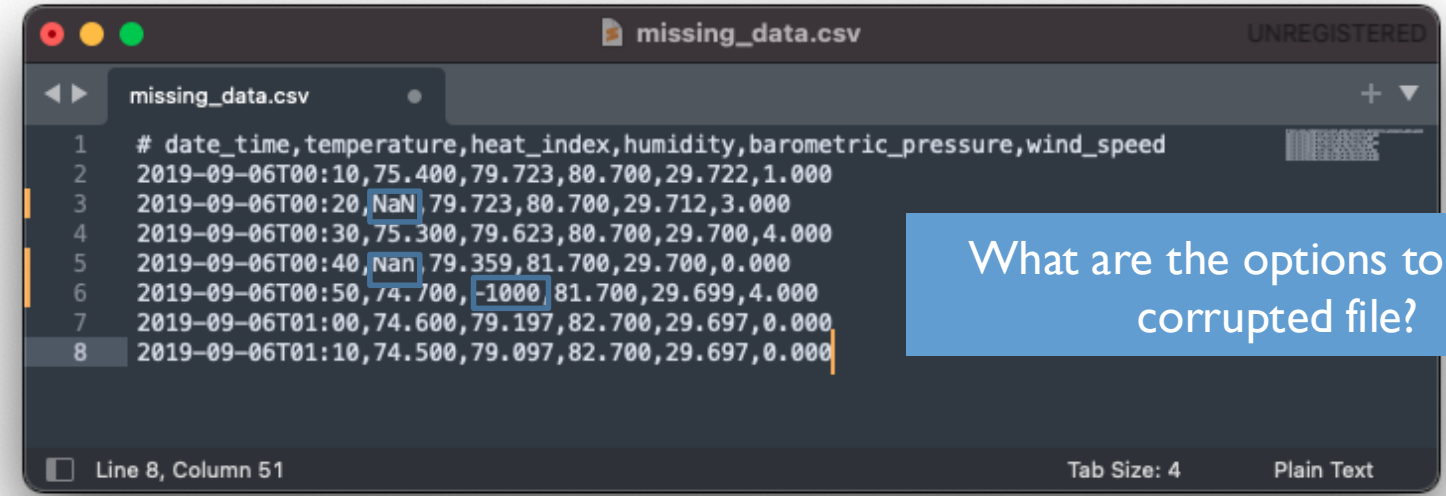
- **1** batch flaky pie crust, divided into 2 unequal halves
- **¼ to ⅓** cup sugar, depending on sweetness of apples
- **½** teaspoon cinnamon
- **¼** teaspoon mace
- **6** large, tart cooking apples, such as Cortlands or pippins, peeled, cored and cut in 6ths
- **1 to 1 ½** tablespoons rosewater, depending on taste
- **4** tablespoons melted butter
- **Cream for brushing the crust**

"Buttered Apple Pie" – Amelia Simmons (1796)

*Pretend it doesn't say "mock apple pie"

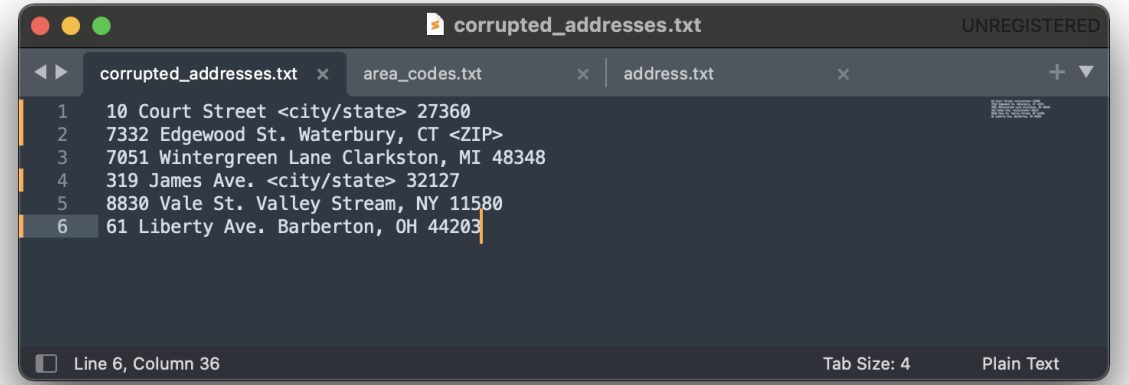
Data Cleaning

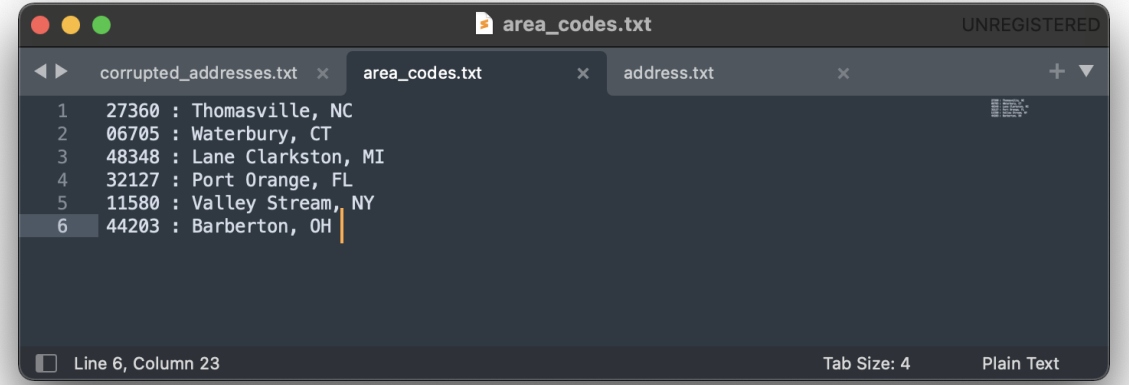
- Missing Data Causes
  - Equipment malfunction (bad sensor reading or DAQ)
  - Human error
    - Ex: Missing phone number or area code
  - Storage system error (corrupted data)



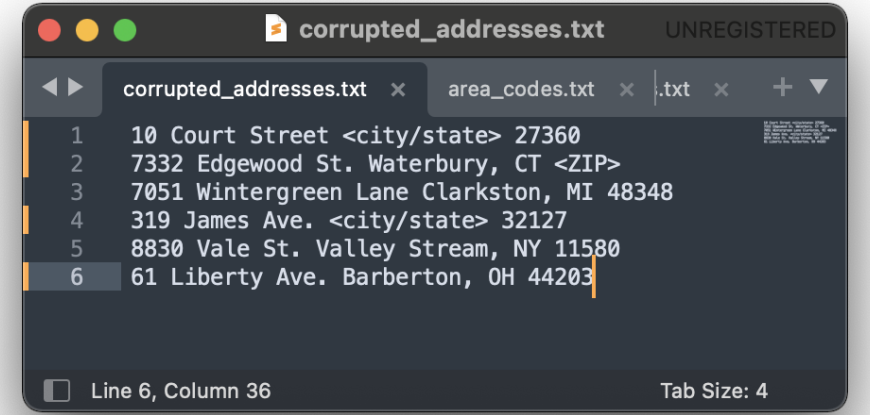
Data Integration / Cleaning

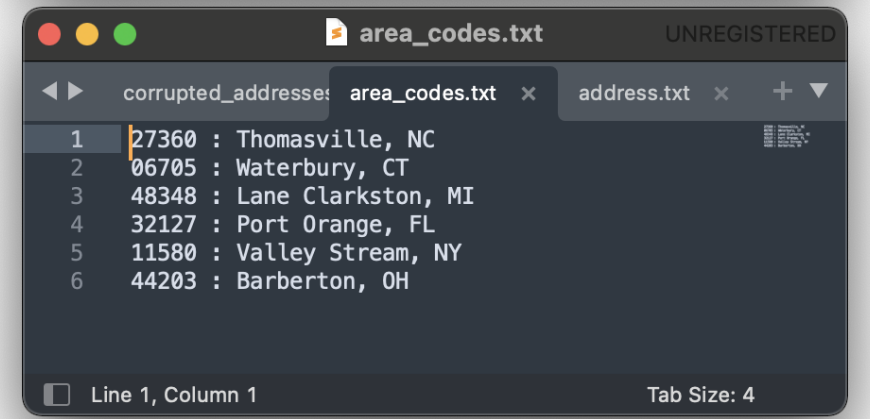
- How can we combine data sources to fix our corrupted address book?

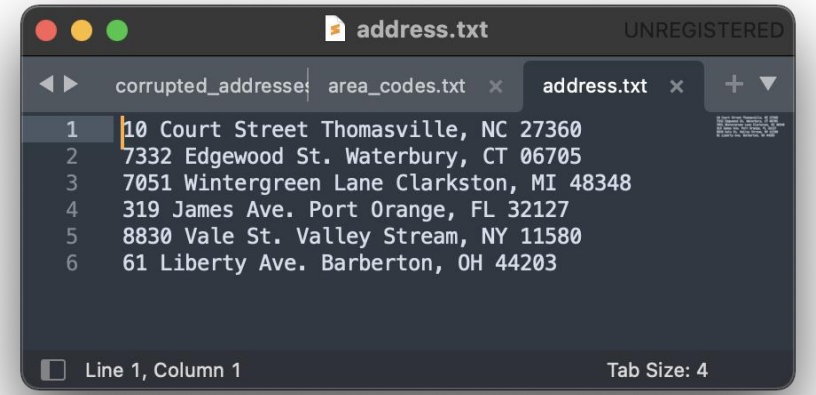




Data Integration / Cleaning

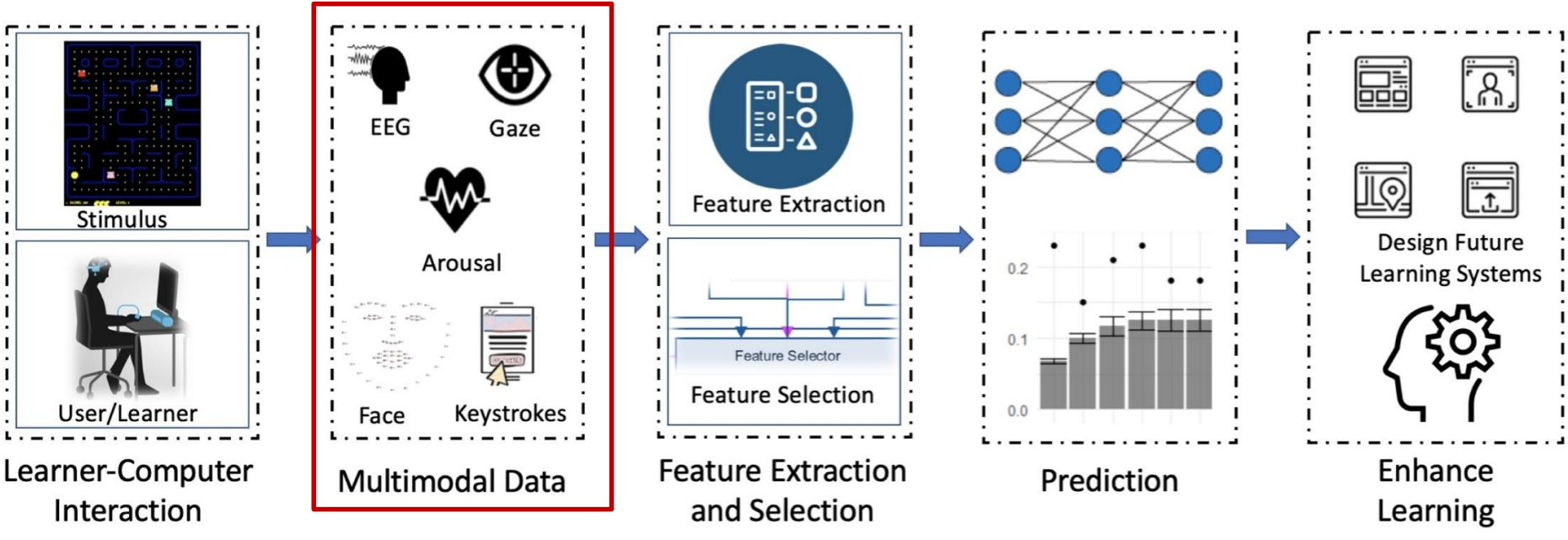






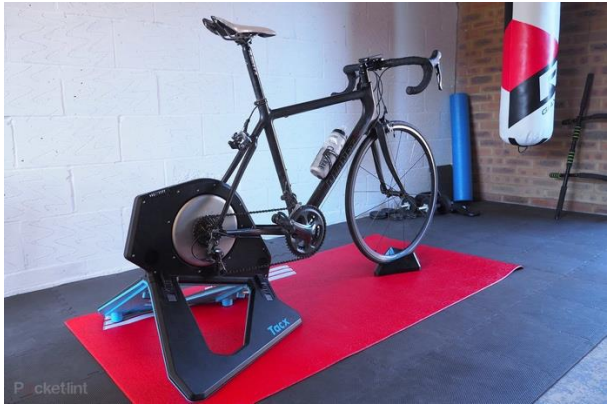
Combining datasets creates a complete address!

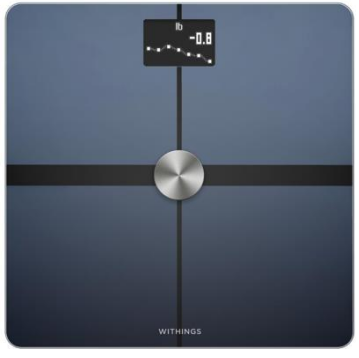
Data integration - Multimodal Data



https://doi.org/10.1016/j.ijinfomgt.2019.02.003

Data Integration – Home edition





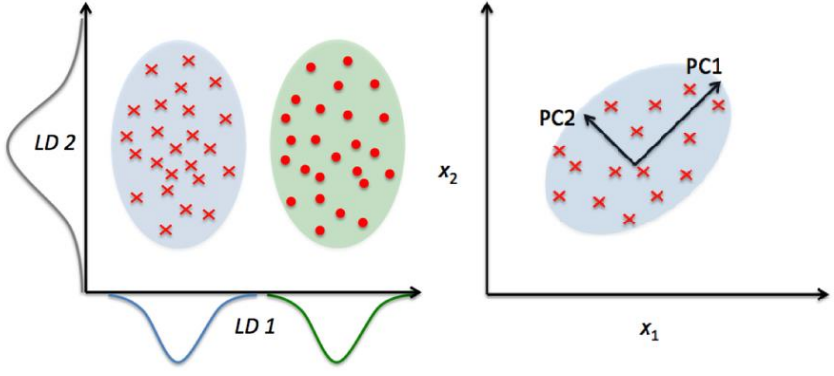
Data Aggregated in the same database from multiple sources – e.g. Apple Health

Data Transformation

- Smoothing – Removing noise from the data
  - Must really be truly noise! Be careful, do not disregard data without serious consideration
  - E.g. Thermometer reading of 11251 °F for a body temperature is unphysical and indicates something is wrong, therefore this point can be disregarded
- Aggregation – Similar to data integration but for high level analysis
- Normalization/Conversion
  - Encoder values 0-1023 -> degrees
  - DAQ voltage -> some meaningful value like MPH or L/min
  - Celsius -> Fahrenheit - > Kelvin
- Attribute/Feature construction

Data Reduction

- Aggregation to the smallest meaningful size
  - Taking Jlab weather station with all of the columns of data and reducing it to the example we recently used
- Dimensionality reduction



Source: towardsdatascience.com

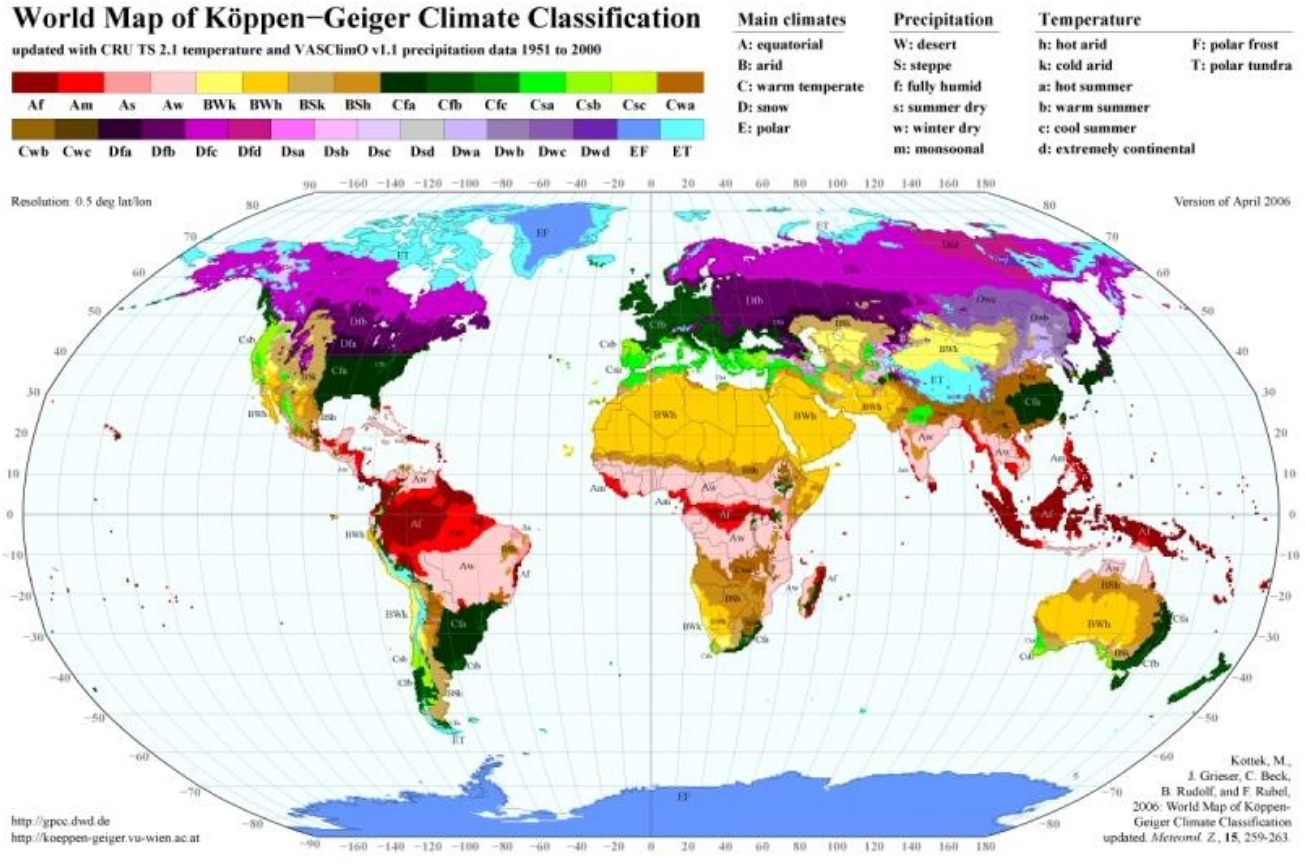
Discretization

- Take continuous values and convert them to discretized values
- Temperatures, climate zones

Grade	Letter Grade
94.5	A
78.3	C+
92	A-

Temp [°F]	Description
87	Hot
75	Warm
57	Cool

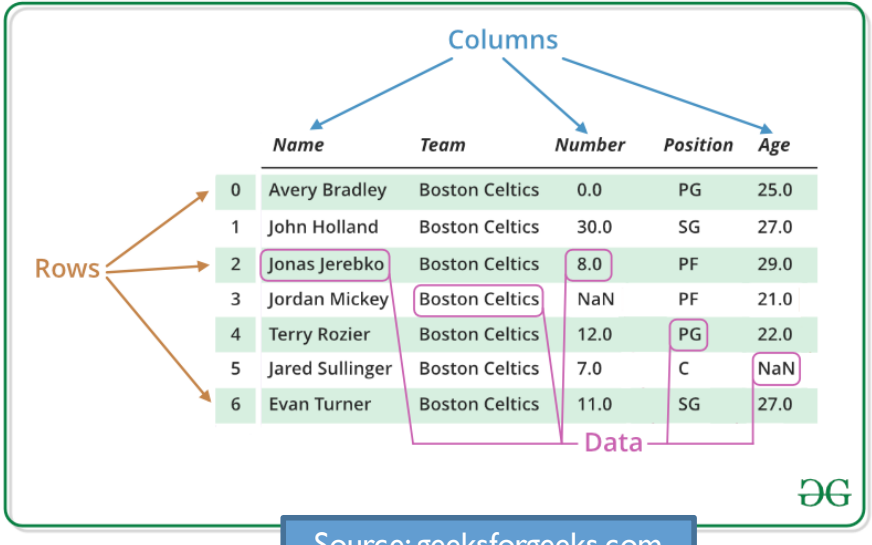
Data Discretization



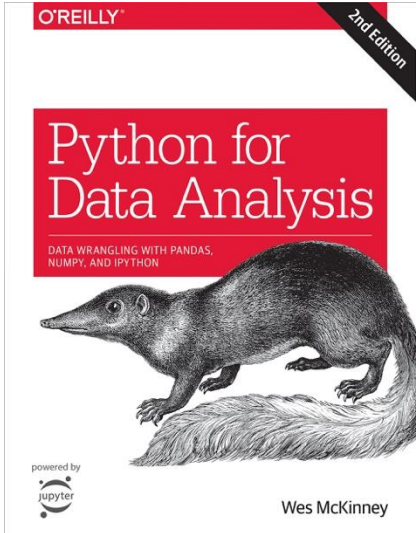
PANDAS

Pandas Dataframes and Dataseries

- Software library written by Wes McKinney ~2008
- Versatile datastructure integrated with NumPy and Matplotlib



Source: geeksforgeeks.com



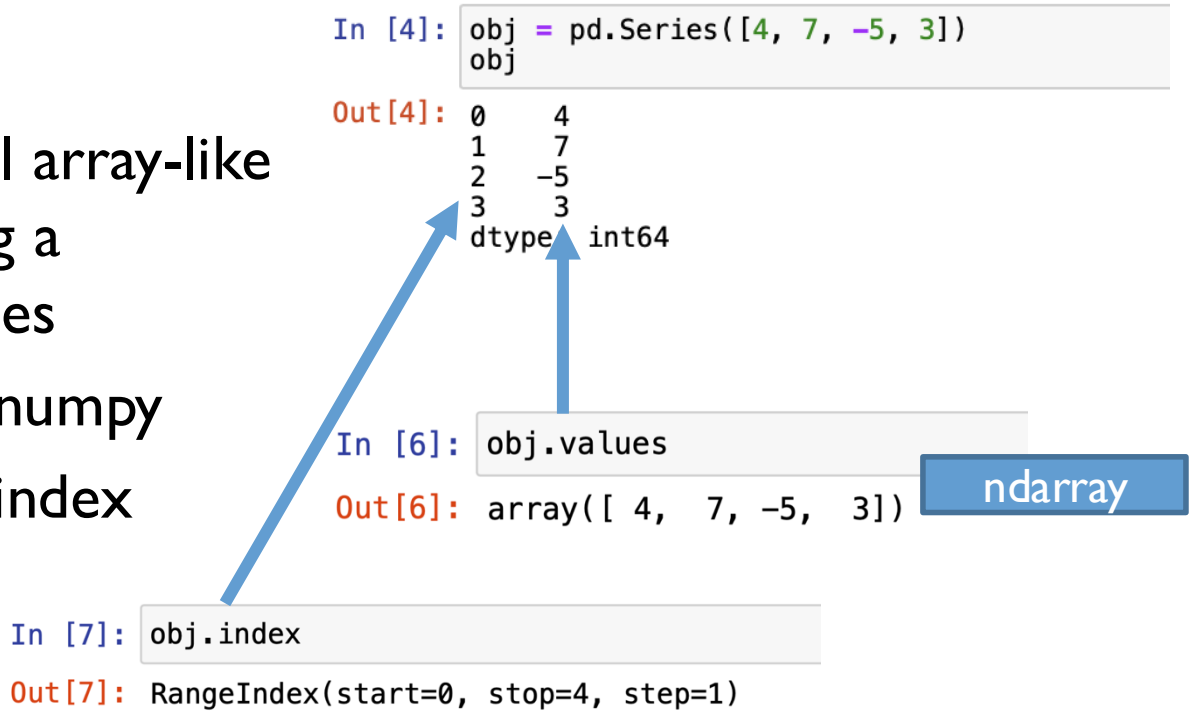
Lecture from Ch5

Pandas Data Structures

- Series
  - One-dimensional array-like object containing a sequence of values
  - Similar types to numpy
  - Labels called an index
- DataFrame
  - Represents a rectangular table of data
  - Ordered collection of columns
    - Types: Numeric, string, Boolean, etc.

Pandas Series

- Series
  - One-dimensional array-like object containing a sequence of values
  - Similar types to numpy
  - Labels called an index



Pandas Series - Indexing

- You can specify the index when creating the series.
  - Yes, you can have duplicate indices.

```
In [8]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
        obj2
Out[8]: d    4
        b    7
        a   -5
        c    3
        dtype: int64
```

```
In [9]: obj2.index
Out[9]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

Selecting by index

- Selection works similar to numpy arrays
- You can also assign values similar to numpy

```
In [15]: obj2['a']
Out[15]: -5
```

```
In [17]: obj2["d"] = 6

In [18]: obj2
Out[18]: d    6
         b    7
         a   -5
         c    3
         dtype: int64
```

Select Multiple Values

- You can select a set of values by passing in a list of indices

```
In [20]: obj2[['c','a','d']]

Out[20]: c    3
         a   -5
         d    6
         dtype: int64
```

Selecting Values

- Boolean conditional statements work as in numpy
- Numpy operations preserve index

```
In [23]: obj2[obj2 > 0]
Out[23]: d    6
         b    7
         c    3
         dtype: int64

In [24]: obj2 * 2
Out[24]: d    12
         b    14
         a   -10
         c     6
         dtype: int64

In [25]: np.exp(obj2)
Out[25]: d     403.428793
         b    1096.633158
         a       0.006738
         c      20.085537
         dtype: float64
```

Other Series Ops

- Series can be created from a dictionary
- One way to think of a Series object is as a fixed length, ordered dictionary

```
In [28]: 'b' in obj2
Out[28]: True

In [29]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
         obj3 = pd.Series(sdata)
         obj3
Out[29]: Ohio      35000
         Texas     71000
         Oregon    16000
         Utah       5000
         dtype: int64
```

Series Part VII

```
In [31]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
         states = ['California', 'Ohio', 'Oregon', 'Texas']
         obj4 = pd.Series(sdata, index=states)
         obj4

Out[31]: California        NaN
         Ohio          35000.0
         Oregon        16000.0
         Texas         71000.0
         dtype: float64
```

Specifying the index will override dictionary keys

```
In [32]: pd.isnull(obj4)

Out[32]: California     True
         Ohio          False
         Oregon        False
         Texas         False
         dtype: bool
```

```
In [35]: obj4[pd.notnull(obj4)]

Out[35]: Ohio      35000.0
         Oregon    16000.0
         Texas     71000.0
         dtype: float64
```

Arithmetic operations

- Automatically aligns by index for arithmetic operations

```
In [36]: obj3

Out[36]: Ohio      35000
         Texas     71000
         Oregon    16000
         Utah       5000
         dtype: int64

In [37]: obj4

Out[37]: California        NaN
         Ohio          35000.0
         Oregon        16000.0
         Texas         71000.0
         dtype: float64

In [38]: obj3 + obj4

Out[38]: California         NaN
         Ohio           70000.0
         Oregon         32000.0
         Texas         142000.0
         Utah               NaN
         dtype: float64
```

Dataframes

- Represents a rectangular table of data
- Ordered collection of columns
  - Types: Numeric, string, Boolean, etc.

```
In [39]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
                 'year': [2000, 2001, 2002, 2001, 2002, 2003],
                 'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
         frame = pd.DataFrame(data)

In [40]: frame
```

Out[40]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

Head and Tail

```
In [41]: frame.head()
```

Out[41]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

```
In [42]: frame.tail()
```

Out[42]:

	state	year	pop
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

Useful to get a quick peak at data

Dataframe - Reindex Columns

```
In [28]: frame
```

Out[28]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

```
In [31]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

Out[31]:

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

Missing Column

- If you pass an additional column that does not exist the missing data will be replaced with NaNs

```
In [33]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                               index=['one', 'two', 'three', 'four',
                                      'five', 'six'])
         frame2
```

Out[33]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

Dataframe - Accessing Columns

Dictionary style

```
In [34]: frame2['state']

Out[34]: one        Ohio
         two        Ohio
         three      Ohio
         four     Nevada
         five     Nevada
         six      Nevada
         Name: state, dtype: object
```

By Attribute

```
In [35]: frame2.year

Out[35]: one      2000
         two      2001
         three    2002
         four     2001
         five     2002
         six      2003
         Name: year, dtype: int64
```

Dataframe - LOC and ILOC

- Loc will retrieve the row given an index
- Iloc will retrieve the row by integer index

```
In [53]: frame2.loc['three']
Out[53]: year     2002
         state    Ohio
         pop       3.6
         debt      NaN
         Name: three, dtype: object
```

```
In [54]: frame2.iloc[2]
Out[54]: year     2002
         state    Ohio
         pop       3.6
         debt      NaN
         Name: three, dtype: object
```

Dataframe - Modification by assignment

In [55]:
```
frame2['debt'] = 16.5
frame2
```

Out[55]:

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

All elements in a column set to the same value

In [56]:
```
frame2['debt'] = np.arange(6.)
frame2
```

Out[56]:

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

In [58]:
```
frame2['debt'] = [6,5,4,3,2,1]
frame2
```

Out[58]:

	year	state	pop	debt
one	2000	Ohio	1.5	6
two	2001	Ohio	1.7	5
three	2002	Ohio	3.6	4
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	2
six	2003	Nevada	3.2	1

Dataframe – Assigning a Series

- When assigning lists or arrays as the to the column they must be exactly the same length
- When you assign a series to a column it will insert NaNs for the missing values

```
In [59]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
         frame2['debt'] = val
         frame2
```

Out[59]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Creating and deleting Columns

```
In [64]: frame2['eastern'] = frame2.state == 'Ohio'
         frame2
```

Out[64]:

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

Creating a new Column*

```
In [65]: del frame2['eastern']
         frame2.columns
```

Out[65]: Index(['year', 'state', 'pop', 'debt'], dtype='obje

```
In [66]: frame2
```

Out[66]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Deleting a Column

*Cannot be done with assignment syntax (frame2.eastern = …)

Dataframes – From nested dictionaries

- Outer dictionary keys will be interpreted as the columns
- Inner keys will be interpreted as the indices

```
In [67]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
                'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

In [68]: frame3 = pd.DataFrame(pop)
         frame3
```

Out[68]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

Transposing a dataframe

In [68]:
```
frame3 = pd.DataFrame(pop)
frame3
```

Out[68]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

In [69]:
```
frame3.T
```

Out[69]:

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

Demo time

- Introduce Dataframe and Series
- If time: Plot stock market data