

# AI/ML BOOTCAMP

---

William Phelps

Christopher Newport University/Jefferson Lab

# 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\*

I 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



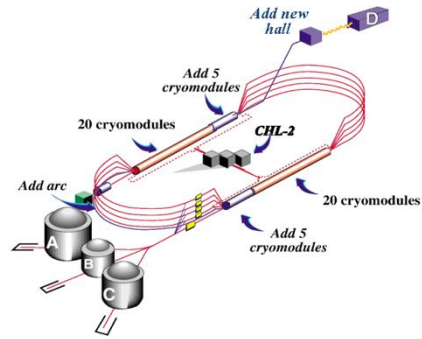
---

**THE GEORGE  
WASHINGTON  
UNIVERSITY**

WASHINGTON, DC



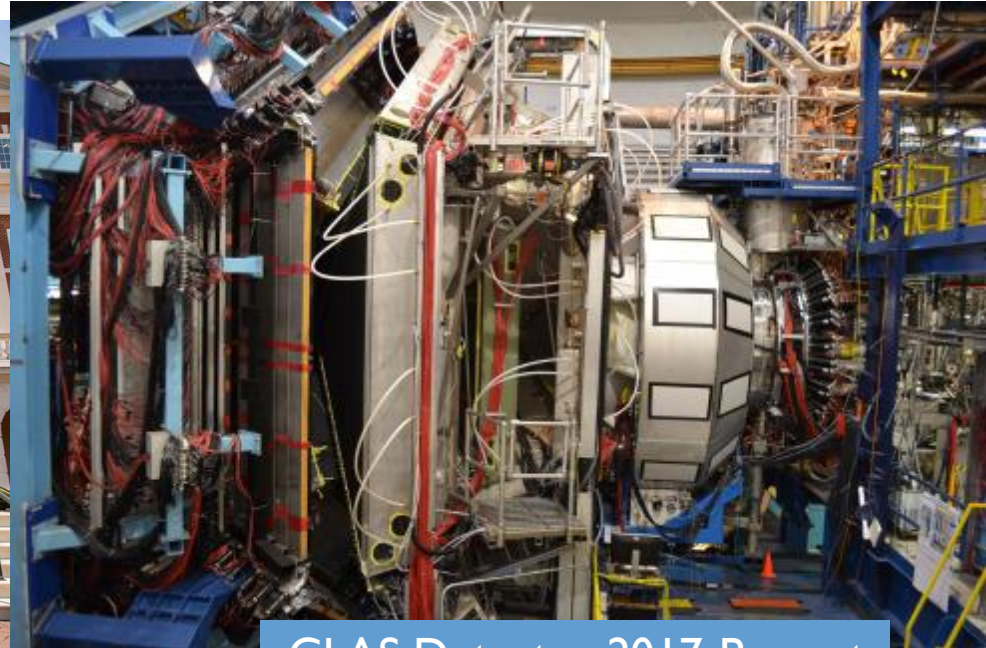
# What I do at Jefferson Lab



# CLAS Collaboration



CLAS Collaboration July 2023

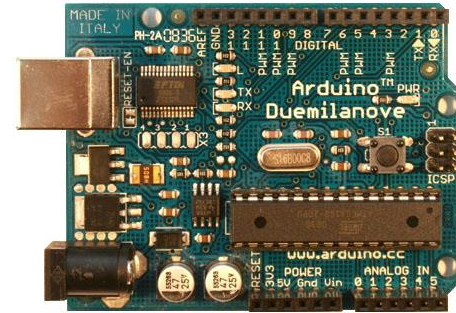


CLAS Detector 2017-Present

Elected Collaboration Chair  
from 2023-2025

# Software work

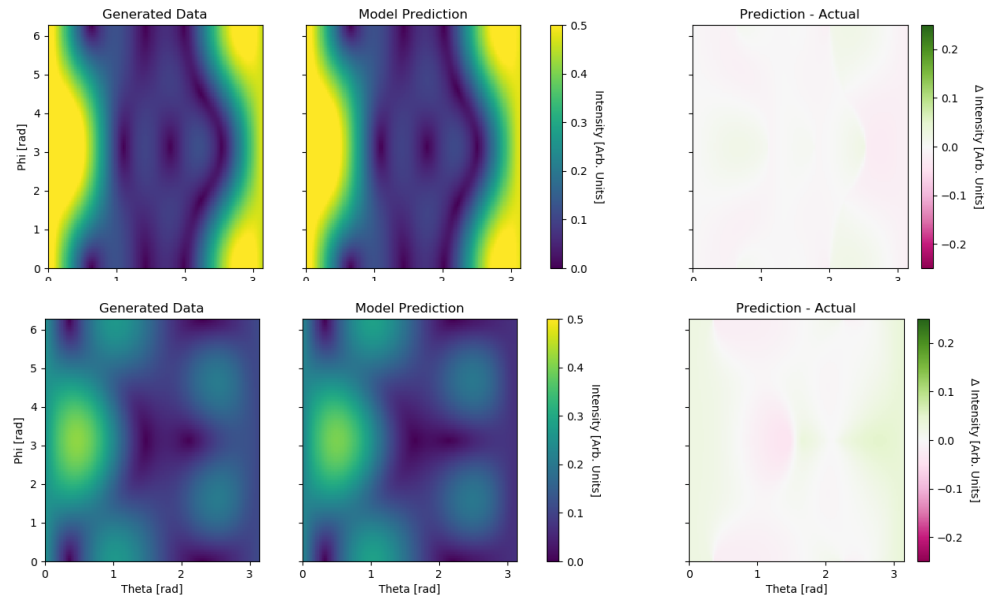
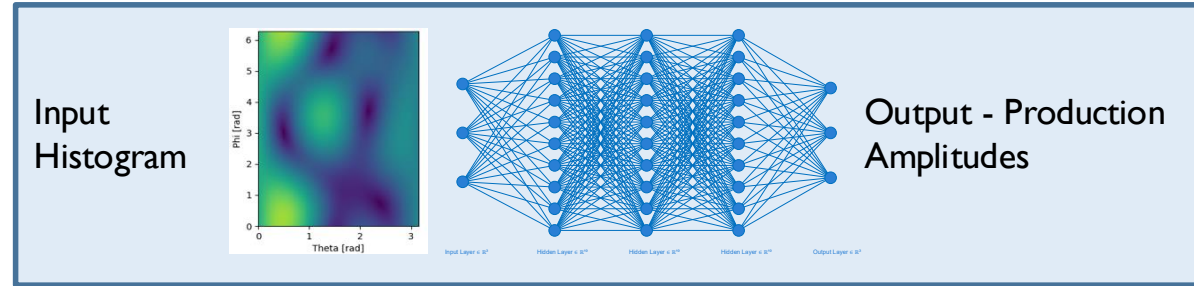
- Lots of data analysis in physics!
- Multithreaded analysis is needed for large datasets
- Petabytes of data per run
- IGB/s DAQ rate!



# 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

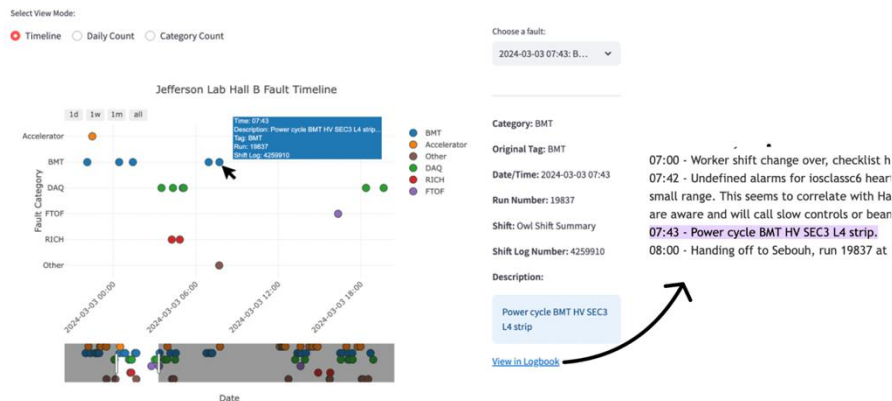


Figure 4: Interactive timeline view of faults, also showing specific fault details and the link to the original logbook entry.

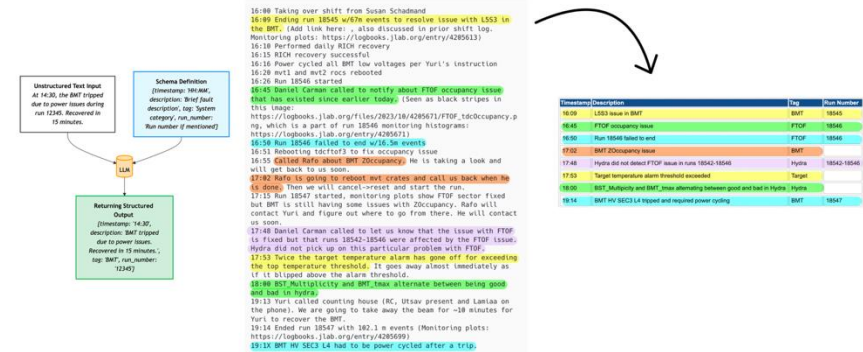
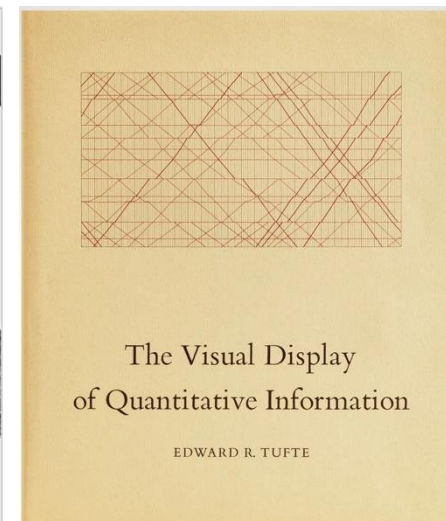
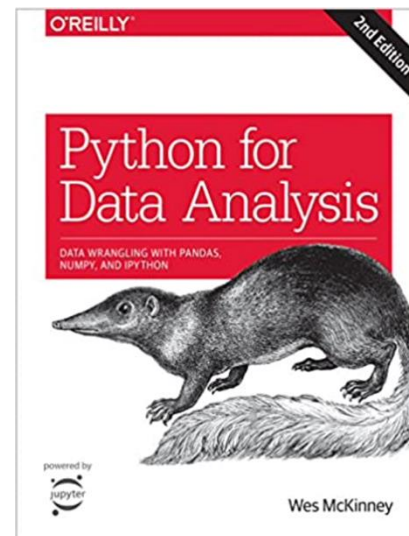


Figure 2: Left: Fault schema example. Right: Example Shift Summary with the LLM's tabulated output

# Course Materials

- **Python for Data Analysis** by Wes McKinney 3<sup>rd</sup> 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!

```
1 from __future__ import print_function
2 import argparse
3 import torch
4 import torch.nn as nn
5 import torch.optim as optim
6 import numpy as np
7 import matplotlib
8 matplotlib.use('Agg')
9 import matplotlib.pyplot as plt
10
11 class Sequence(nn.Module):
12     def __init__(self):
13         super(Sequence, self).__init__()
14         self.lstm1 = nn.LSTMCell(1, 51)
15         self.lstm2 = nn.LSTMCell(51, 51)
16         self.linear = nn.Linear(51, 1)
17
18     def forward(self, input, future = 0):
19         outputs = []
20         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

Finance

Public  
policy

Politics

Healthcare

Urban  
planning

Education

Libraries

...

# Relation of Data Science with other fields



Source: [towardsdatascience.com](https://towardsdatascience.com)



Statistics



Computer Science



Engineering



Business Analytics



(Computational) Social Science

# What skills do we need for Data Science?



Willing to experiment

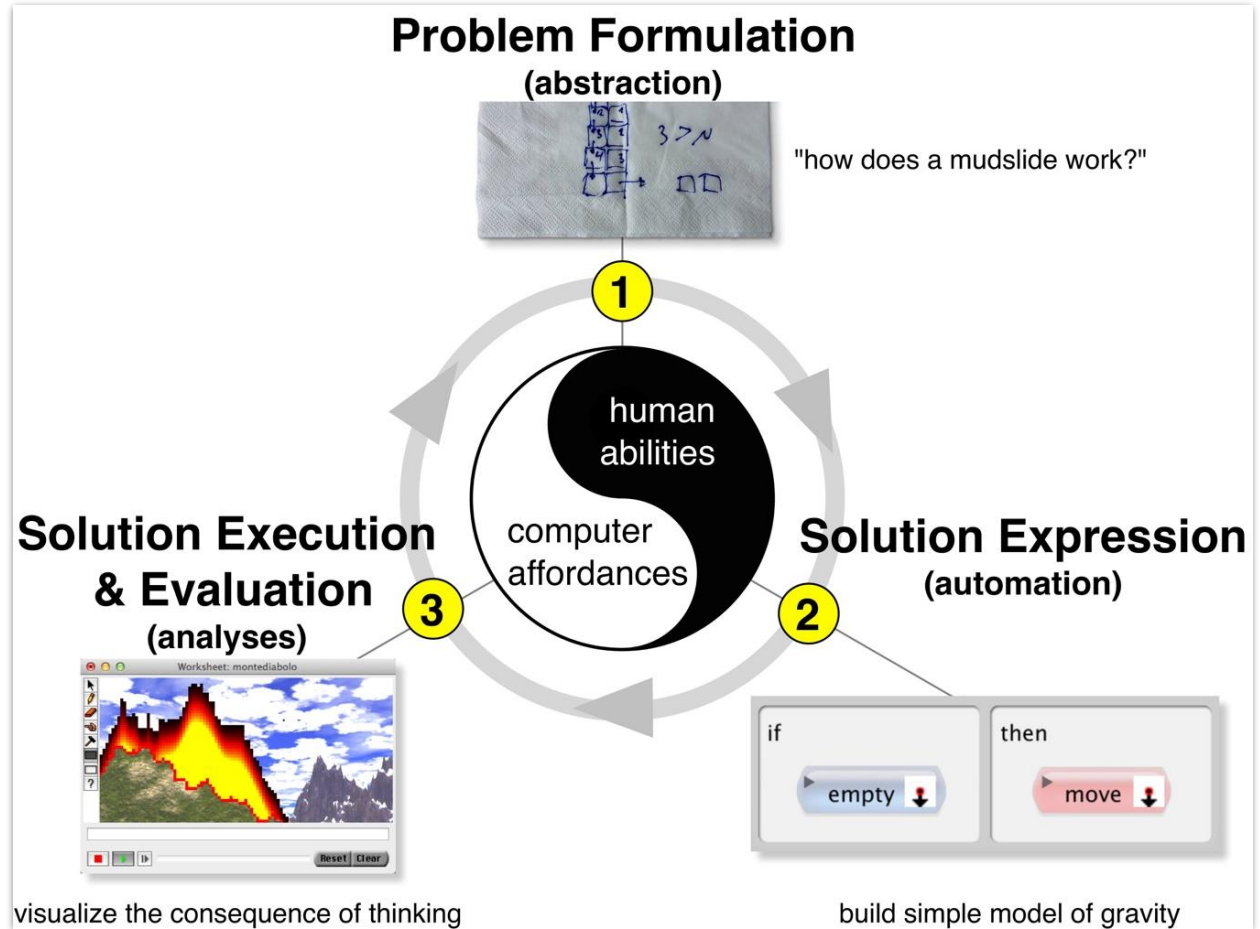


Proficiency in  
mathematical  
reasoning



Data literacy

# Computational thinking



# 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

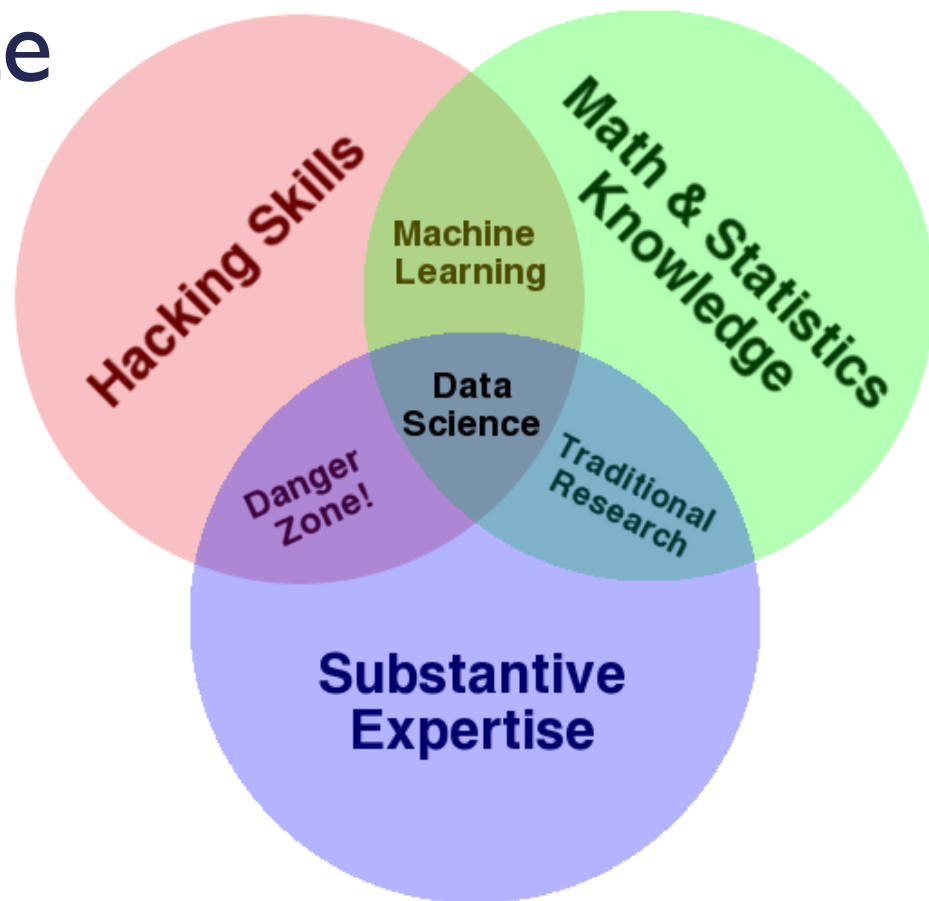


Figure source: Drew Conway

# Data is produced at an increasing rate

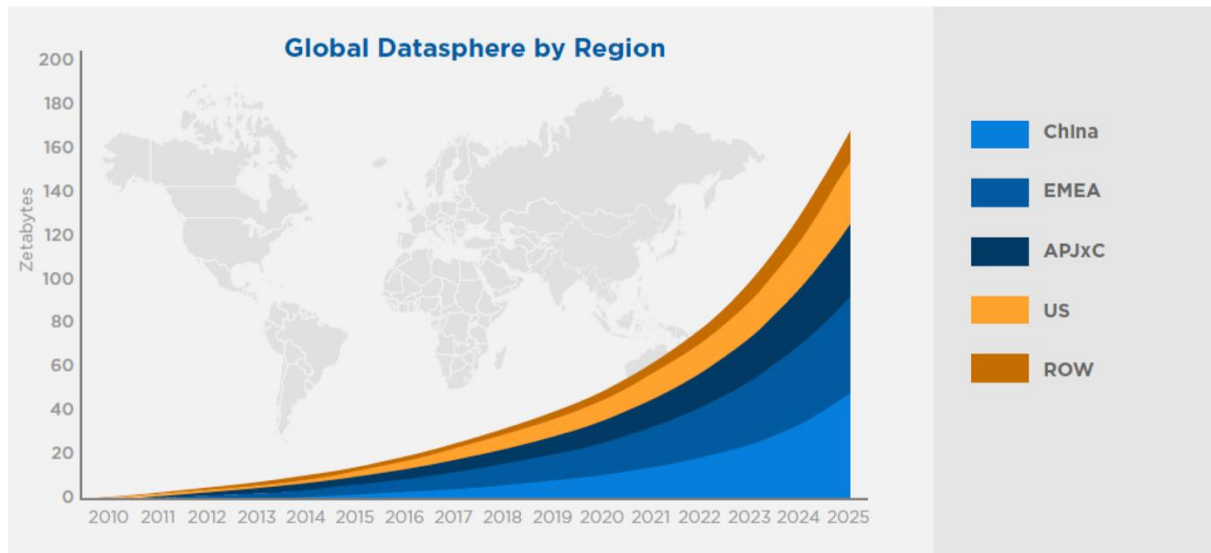
1 ZB is 1E21 Bytes  
 1 ZB is 10<sup>9</sup> TB  
 1 TB is a typical HDD size

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

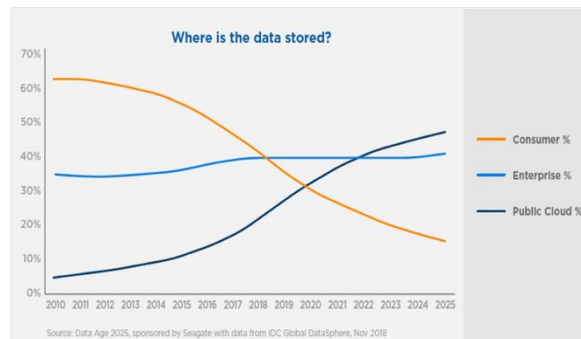
Decimal		Binary		
Value	SI	Value	IEC	JEDEC
1000	k kilo	1024	Ki kibi	K kilo
1000 <sup>2</sup>	M mega	1024 <sup>2</sup>	Mi mebi	M mega
1000 <sup>3</sup>	G giga	1024 <sup>3</sup>	Gi gibi	G giga
1000 <sup>4</sup>	T tera	1024 <sup>4</sup>	Ti tebi	—
1000 <sup>5</sup>	P peta	1024 <sup>5</sup>	Pi pebi	—
1000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

Source: Wikipedia

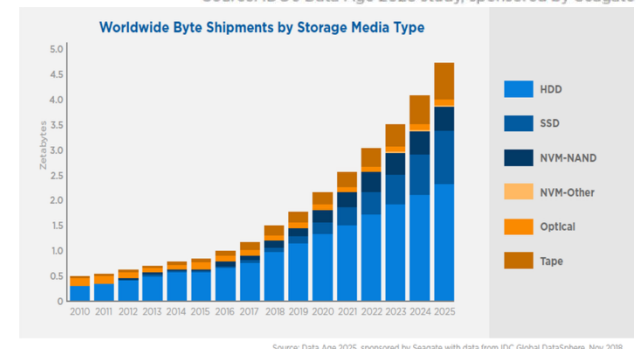
V•T•E



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



Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018



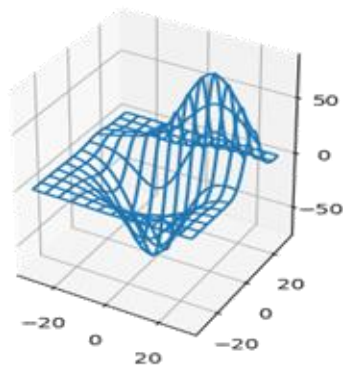
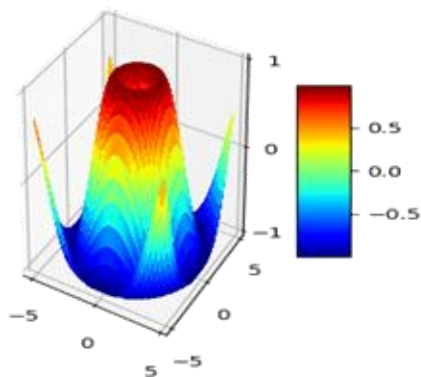
Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

# WEEK 1 TOPICS

---

# Numerical Calculations and Visualization

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



**matplotlib**

# Pandas Dataframes and Dataseries

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

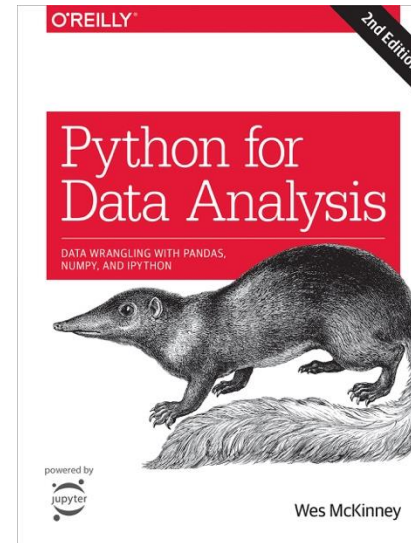


Diagram illustrating the structure of a Pandas DataFrame. The columns are labeled "Name", "Team", "Number", "Position", and "Age". The rows are labeled "Rows". The data is presented in a table format:

	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

The diagram highlights the "Data" structure with a pink box around the values in the "Number" and "Position" columns for rows 2 through 6. A green logo is visible in the bottom right corner of the diagram area.

Source: [geeksforgeeks.com](http://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

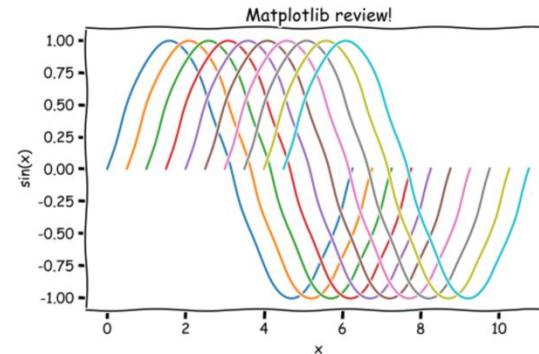
```
In [1]: print("Hello World!")
```

```
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)

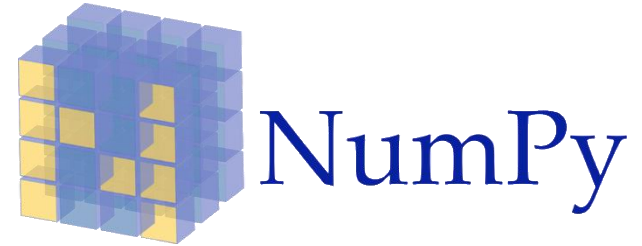
- You can iterate over the lines in a file
  - Lines in a file are delimited by newlines (`\n`)
- Once you have each line you can split by the **comma separated values (.csv files)**
- What is the data type of words?
- What is the data type of temperature?

```
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

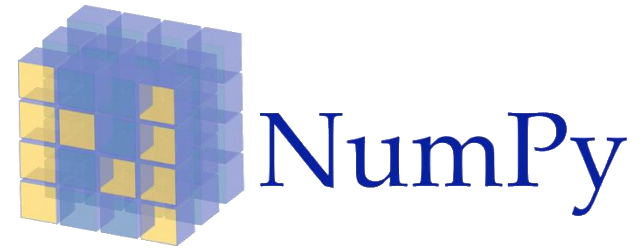


- Modules
  - **numpy**

<http://www.numpy.org>

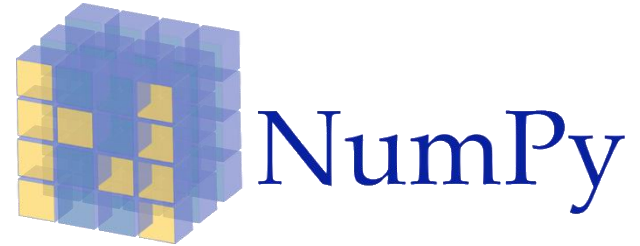
<https://docs.scipy.org/doc/>

# Why do we use Numpy?



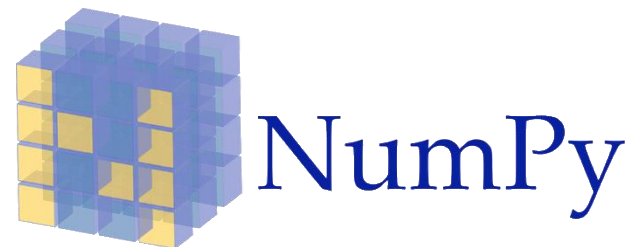
- Python has lists built in
  - Still lists! Not arrays
  - Slow for large datasets
- Numpy is used for:
  - Linear Algebra
  - Matrix computation
  - Numerical analysis
  - Lingua franca of scientific computing and data science libraries (Pandas, scipy, etc.)

# 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 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.zeros()`
- `np.ones()`
- `np.linspace()`
- `np.arange()`
- Or `np.array([python list])`

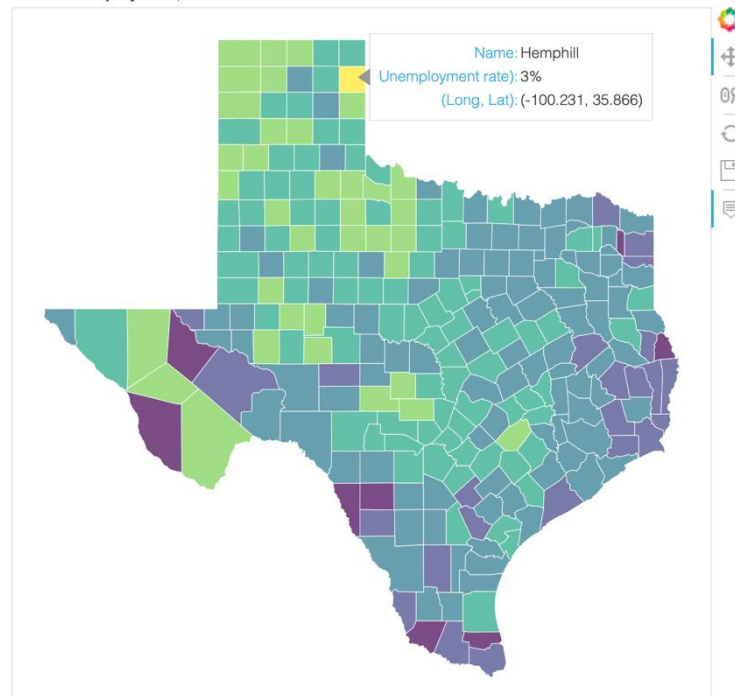
## Operations

- `*/+-` 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

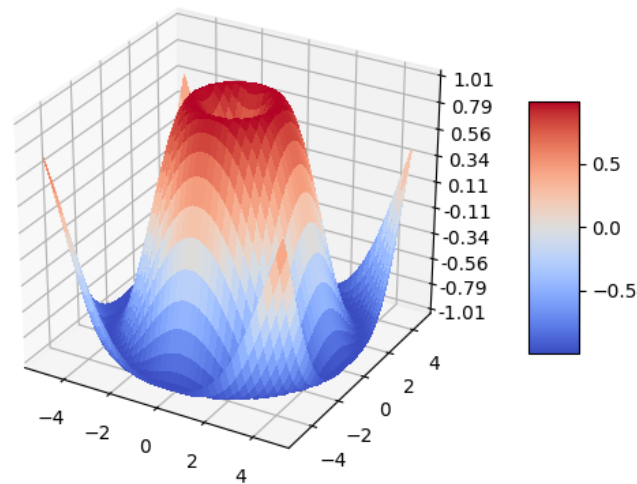
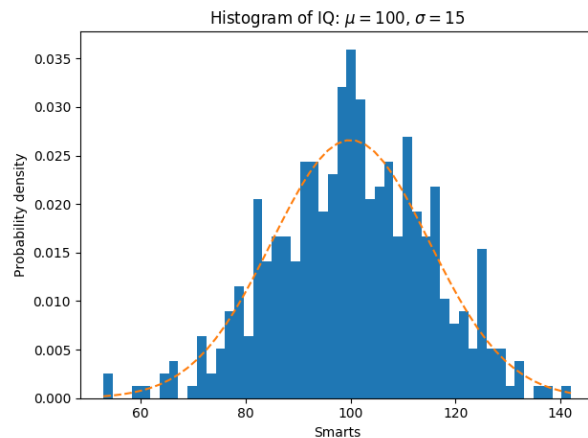
Texas Unemployment, 2009



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

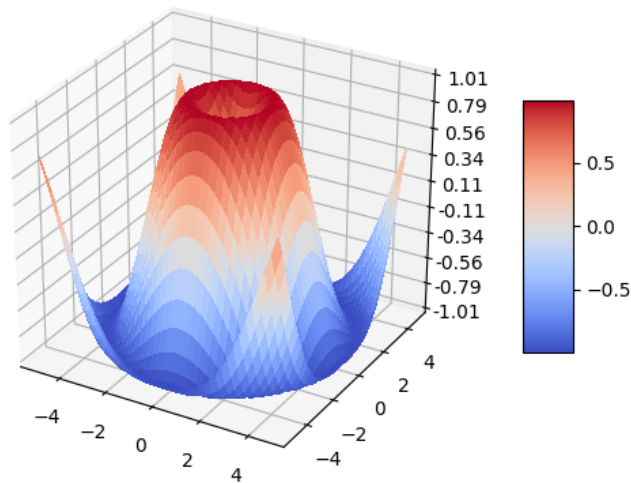
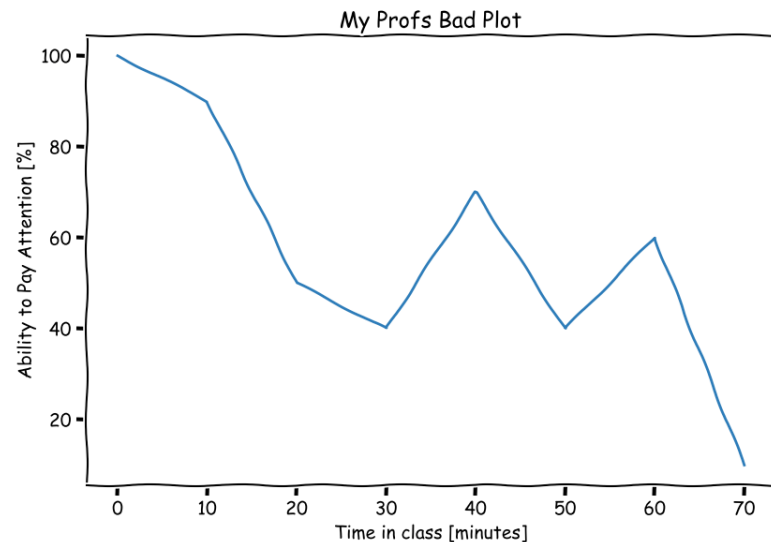
- Import matplotlib.pyplot as plt
- Here we are plotting two lists
- Samples is a list that contains integers [1,2,3,4,5,6...]
- Temperature\_list is a list of temperatures that we have created by appending to an empty list
- plt.plot(x,y) is the simplest way to plot arrays of x and y values
- plt.show(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

3D Visualization



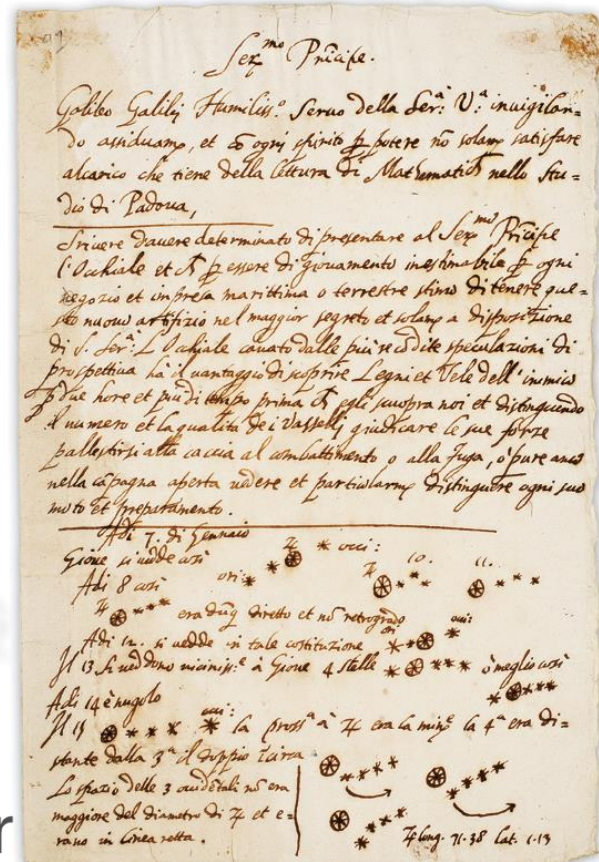
XKCD Theme...

# 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

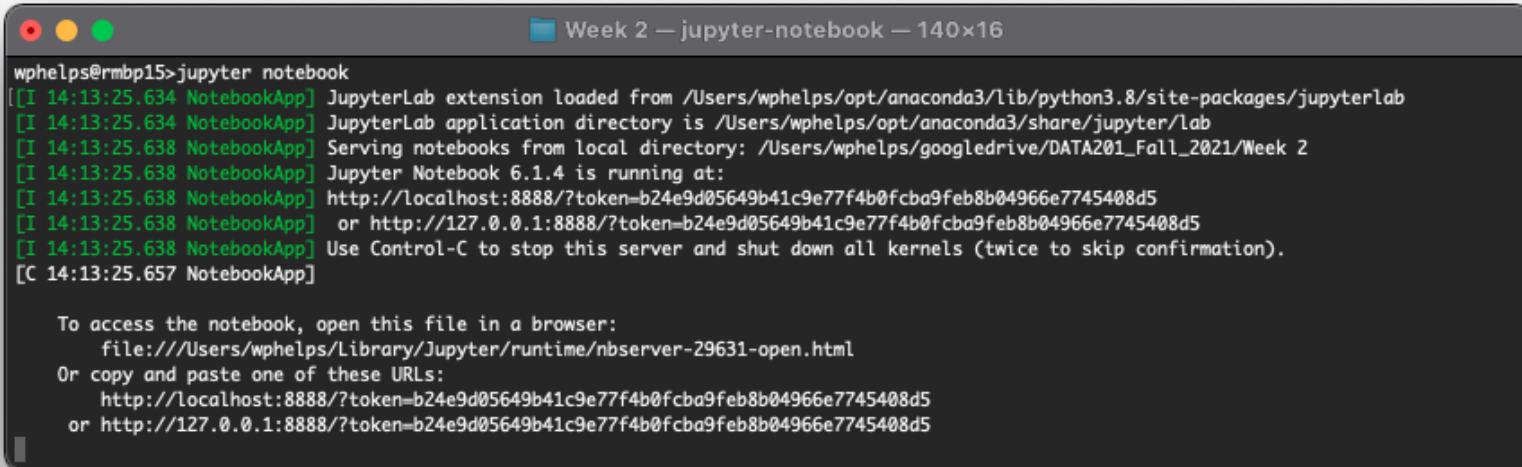


Galileo's Manuscript - 1610

# 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

A terminal window titled "Week 2 — jupyter-notebook — 140x16" showing the output of the command "jupyter notebook". The output includes several informational messages from the NotebookApp, such as the JupyterLab extension path, application directory, serving directory, and the URL to access the notebook. It also provides instructions on how to access the notebook via a browser or copy-paste URLs.

```
wphelps@rmbp15>jupyter notebook
[I 14:13:25.634 NotebookApp] JupyterLab extension loaded from /Users/wphelps/opt/anaconda3/lib/python3.8/site-packages/jupyterlab
[I 14:13:25.634 NotebookApp] JupyterLab application directory is /Users/wphelps/opt/anaconda3/share/jupyter/lab
[I 14:13:25.638 NotebookApp] Serving notebooks from local directory: /Users/wphelps/googledrive/DATA201_Fall_2021/Week 2
[I 14:13:25.638 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 14:13:25.638 NotebookApp] http://localhost:8888/?token=b24e9d05649b41c9e77f4b0fcb9feb8b04966e7745408d5
[I 14:13:25.638 NotebookApp] or http://127.0.0.1:8888/?token=b24e9d05649b41c9e77f4b0fcb9feb8b04966e7745408d5
[I 14:13:25.638 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:13:25.657 NotebookApp]

To access the notebook, open this file in a browser:
file:///Users/wphelps/Library/Jupyter/runtime/nbserver-29631-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=b24e9d05649b41c9e77f4b0fcb9feb8b04966e7745408d5
or http://127.0.0.1:8888/?token=b24e9d05649b41c9e77f4b0fcb9feb8b04966e7745408d5
```

# File browser

 jupyter

Quit

Logout

Files

Running

Clusters

Select items to perform actions on them.

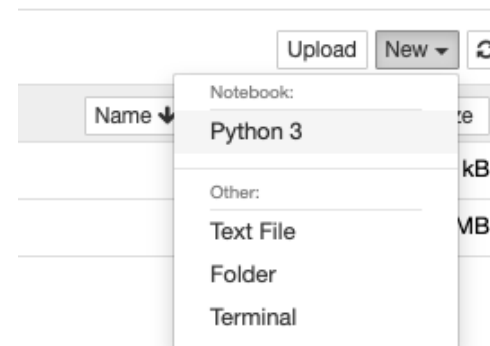
Upload

New ▾



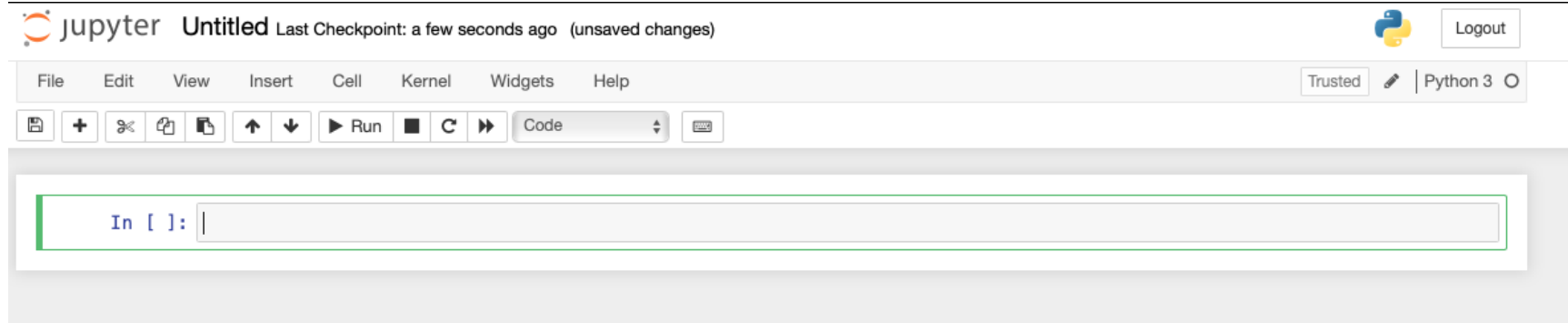
<input type="checkbox"/> 0 ▾	📁 /	Name ▾	Last Modified	File size
<input type="checkbox"/>	 Jupyter_Introduction_and_Review.ipynb		a minute ago	121 kB
<input type="checkbox"/>	 DATA201_Week2.pptx		8 minutes ago	15.9 MB

Interactive file browser



Open a new notebook

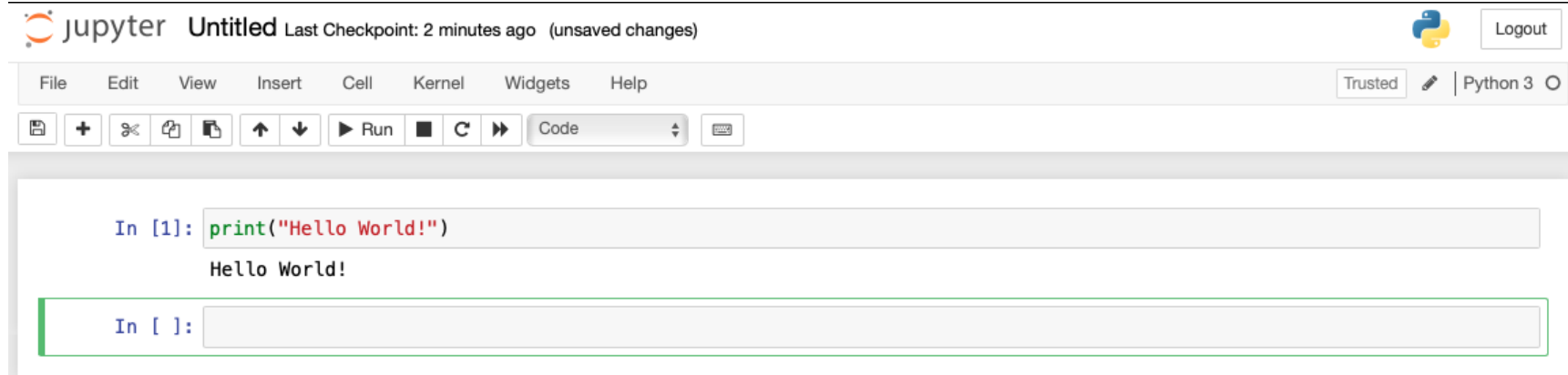
# My first jupyter notebook



The screenshot displays the Jupyter Notebook interface. At the top, the title bar shows "jupyter Untitled" and "Last Checkpoint: a few seconds ago (unsaved changes)". On the right side of the title bar, there is a Python logo, a "Logout" button, and a "Trusted" status indicator. Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar below the menu bar contains icons for file operations (save, new, copy, paste), navigation (up, down), execution (run, stop, refresh), and a dropdown menu currently set to "Code". The main workspace contains a single code cell with the prompt "In [ ]: |" and a cursor.

Cells can be either code or markdown cells

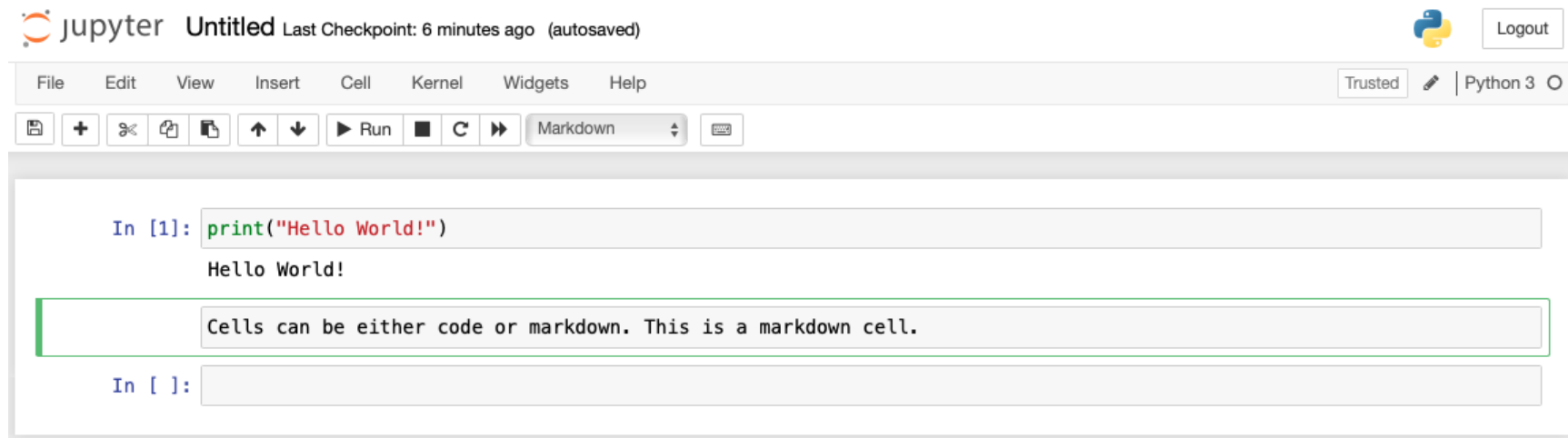
# Hello World!



The screenshot displays the Jupyter Notebook interface. At the top, the title bar shows 'jupyter Untitled' and 'Last Checkpoint: 2 minutes ago (unsaved changes)'. A 'Logout' button is visible in the top right corner. Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar, there is a 'Trusted' status indicator, a pencil icon, and 'Python 3' with a dropdown arrow. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and navigation (up/down arrows, Run, Stop, Refresh, Next). A dropdown menu is set to 'Code'. The main workspace contains two code cells. The first cell has the code `In [1]: print("Hello World!")` and the output `Hello World!`. The second cell is empty and has the prompt `In [ ]:`.

Each cell can be executed independently by pressing shift+enter

# Adding a Markdown Cell



The screenshot displays the Jupyter Notebook interface. At the top left, the Jupyter logo is followed by the text "jupyter Untitled Last Checkpoint: 6 minutes ago (autosaved)". On the top right, there is a Python logo and a "Logout" button. Below this is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". To the right of the menu bar are "Trusted" and "Python 3" indicators. A toolbar below the menu bar contains icons for file operations, navigation, and execution, along with a dropdown menu currently set to "Markdown".

The main workspace contains three cells:

- The first cell is a code cell with the prompt "In [1]:" and the code `print("Hello World!")`. Below the code, the output "Hello World!" is displayed.
- The second cell is a markdown cell, highlighted with a green border, containing the text "Cells can be either code or markdown. This is a markdown cell."
- The third cell is an empty code cell with the prompt "In [ ]:".

# 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)

✓ •  $x = 10$

✓ •  $y = 4$

✓ •  $\text{name} = \text{“Captain Chris”}$

✓ •  $z = x * x + y * y$

✗ •  $x + y = z$

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

✓ •  $z = x + 42$

✗ •  $z = \text{name} + x$

TypeError: can only concatenate str (not "int") to str

✓ •  $z = \text{name} + \text{str}(x)$

✓ •  $x42 = 13$

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:
<code>int()</code>	Creates integers	int, float, strings w/ integers only
<code>float()</code>	Creates floats	int, float, strings w/ integers or fractions
<code>str()</code>	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

Terminology:  
radius is a  
“parameter”

Terminology:  
10. is an  
“argument” assigned  
to radius when  
calling function

```
>>> def area_circle(radius):
...     area = 3.1415927*radius*radius
...     return area
...
>>> print(area_circle(10.))
314.15927
>>> area = area_circle(10.)
>>> print(area)
314.15927
```

t multiple) or

Don't do this!  
Use `math.pi` instead.

Can use return value in assignment statements

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 condition

- 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
<b>a and b</b>	<b>Boolean AND:</b> True when both operands are True.
<b>a or b</b>	<b>Boolean OR:</b> True when at least one operand is True.
<b>not a</b>	<b>Boolean NOT</b> (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

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

not  $(x < 0)$  **True**  
 False

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

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

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 indentation
  - 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.
<b>unary -</b>	- 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.)
<b>+ -</b>	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.
<b>left-to-right</b>	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 $((y == 10) \text{ and } (z != 10))$ 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

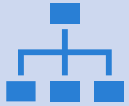
You cannot change the item in list this way.

Use range() and index if you need to modify the list.

# DATA TYPES AND SOURCES

---

# Data types



Structured

Example: tables

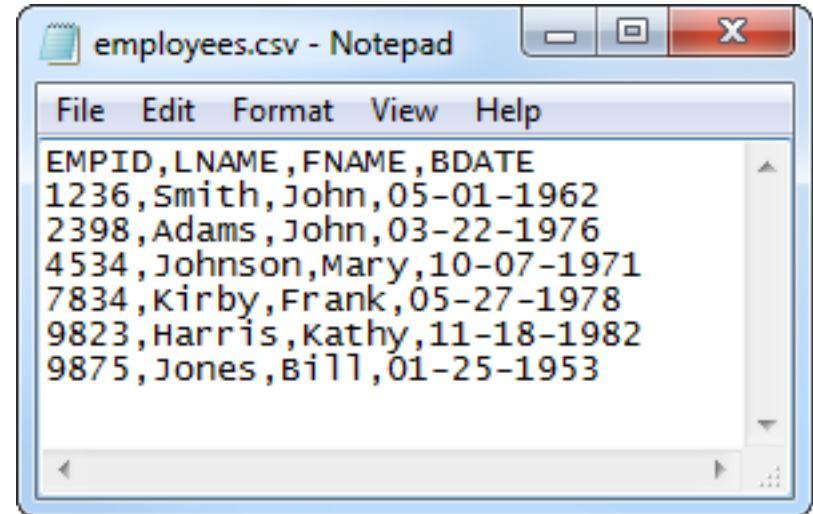


Unstructured

Example: free text

# Structured Data

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



```
employees.csv - Notepad
File Edit Format View Help
EMPID,LNAME,FNAME,BDATE
1236,Smith,John,05-01-1962
2398,Adams,John,03-22-1976
4534,Johnson,Mary,10-07-1971
7834,Kirby,Frank,05-27-1978
9823,Harris,Kathy,11-18-1982
9875,Jones,Bill,01-25-1953
```

# 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



**Elon Musk** ✓  
@elonmusk

Am considering taking Tesla private at \$420. Funding secured.

2:48 AM · Aug 8, 2018 · [Twitter for iPhone](#)

15.9K Retweets 89.9K Likes



**jack** ✓  
@jack

just setting up my twttr

3:50 PM · Mar 21, 2006 · [Twitter Web Client](#)



# Comparison

E. Ford - 10.1136/medethics-2019-105472

Data item	Examples	
	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 qidx7 days'
Referral	8H4D.00 Referral to psychogeriatrician	Rev 4w ?refer psycho ger
Test	43F1.00 Rheumatoid factor positive	Rheumatoid factor was 42 IU/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



DATA TOPICS - RESOURCES STRATEGY DEVELOPERS CONTACT

## The home of the U.S. Government's open data

Here you will find data, tools, and resources to conduct research, develop web and mobile applications, design data visualizations, and [more](#).

For information regarding the Coronavirus/COVID-19, please visit [Coronavirus.gov](#).

GET STARTED

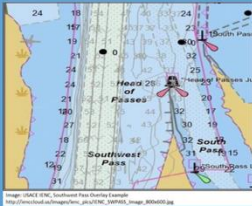
SEARCH OVER 323,559 DATASETS

Health Care Provider Charge Data



### HIGHLIGHTS

## Rivers of Data – Inland Electronic Navigation Charts



Nautical charts provide critical information to mariners in support of safe navigation. Historically these charts have been printed and distributed on paper, but modern communications systems allow for electronic charts that are able to be updated as new information becomes available. The National Oceanic and Atmospheric Administration (NOAA) Office of Coast Survey produces charts for coastal and Great Lakes areas, and the U.S. Army Corps of Engineers produces charts for America's inland rivers through the Inland Electronic Navigation Chart program. The Inland Electronic Navigation Chart (IENC) program covers thousands of miles of navigable waterways. America's inland waterways move millions of tons of commodities every year, and the work of surveying, charting, and dredging sediment is continually ongoing due to the dynamic conditions and constant change happening along any given river.



DATA TOPICS - RESOURCES STRATEGY DEVELOPERS CONTACT

DATA CATALOG

Home / Datasets Organizations

Newport News



Order by:

Relevance

You are searching in the list of datasets. Show results in entire Data.gov site.

Filter by location

Clear

Enter location...



Map files & Data by OpenStreetMap, under CC BY SA

Topics

There are no Topics that match this search

Topic Categories

There are no Topic Categories that match this search

Dataset Type

geospatial 43

Tags

96

virginia 64

noaa 61

neddis 43

doc-noaa-neddis-rgd- 42

## 96 datasets found for "Newport News"

SURVEY, Newport News City, VA

Federal Emergency Management Agency, Department of Homeland Security – The field survey data for this coastal study includes a field report that exhibits photos and transect information collected in the field survey phase of the study...

HTML HTML

2007 City of Newport News, VA lidar

Department of Commerce – These files contain LIDAR point cloud data encompassing the Urban/Suburban land area of City of Newport News, VA.

HTML HTML HTML HTML

2007 City of Newport News, VA lidar

Department of Commerce – These files contain LIDAR point cloud data encompassing the Urban/Suburban land area of City of Newport News, VA.

HTML HTML HTML HTML

2007 City of Newport News, VA lidar

Department of Commerce – These files contain LIDAR point cloud data encompassing the Urban/Suburban land area of City of Newport News, VA.

HTML HTML HTML HTML

2007 City of Newport News, VA lidar

Department of Commerce – These files contain LIDAR point cloud data encompassing the Urban/Suburban land area of City of Newport News, VA.

HTML HTML

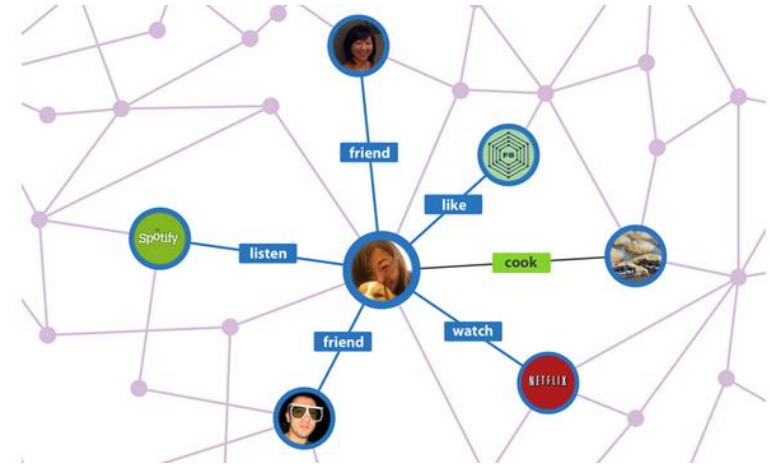
2007 City of Newport News, VA lidar

Department of Commerce – These files contain LIDAR point cloud data encompassing the Urban/Suburban land area of City of Newport News, VA.

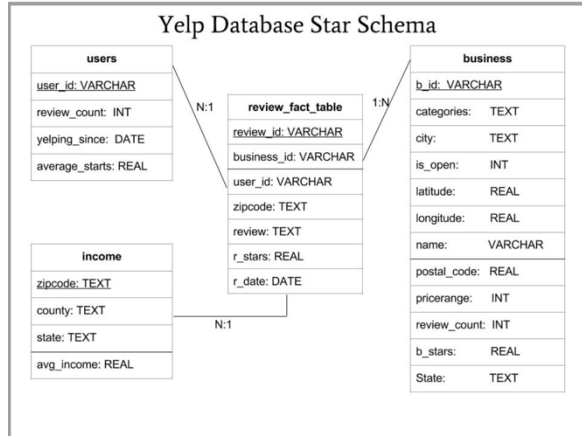
HTML HTML

# Examples of Data Sources

- Social Media – Facebook Graph API
- Yelp API



Source: techcrunch.com



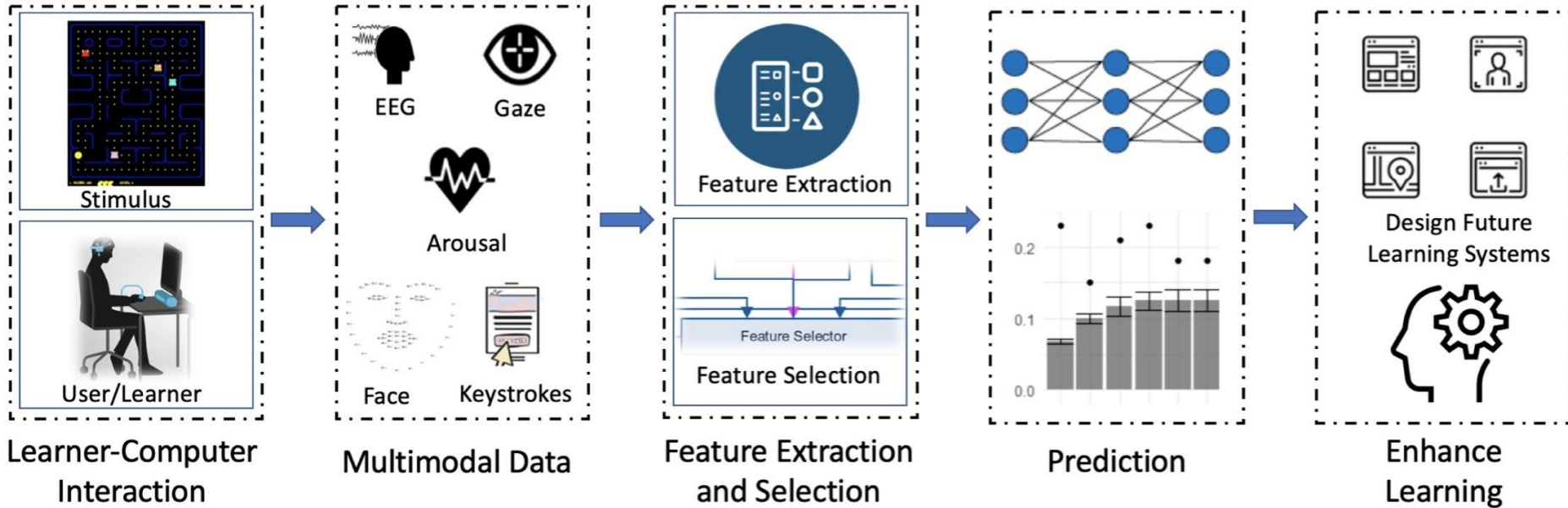
Source: medium.com



facebook



# Multimodal Data



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

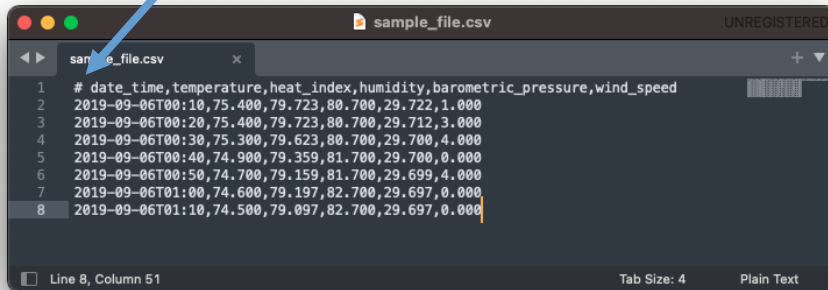
# Structured data file types

- 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 “#”

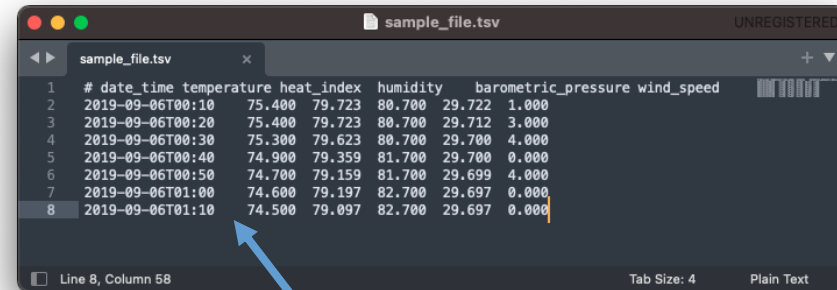


```
sample_file.csv
1 # date_time,temperature,heat_index,humidity,barometric_pressure,wind_speed
2 2019-09-06T00:10,75.400,79.723,80.700,29.722,1.000
3 2019-09-06T00:20,75.400,79.723,80.700,29.712,3.000
4 2019-09-06T00:30,75.300,79.623,80.700,29.700,4.000
5 2019-09-06T00:40,74.900,79.359,81.700,29.700,0.000
6 2019-09-06T00:50,74.700,79.159,81.700,29.699,4.000
7 2019-09-06T01:00,74.600,79.197,82.700,29.697,0.000
8 2019-09-06T01:10,74.500,79.097,82.700,29.697,0.000
```

Line 8, Column 51 Tab Size: 4 Plain Text

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

## TSV File with Header



```
sample_file.tsv
1 # date_time temperature heat_index humidity barometric_pressure wind_speed
2 2019-09-06T00:10 75.400 79.723 80.700 29.722 1.000
3 2019-09-06T00:20 75.400 79.723 80.700 29.712 3.000
4 2019-09-06T00:30 75.300 79.623 80.700 29.700 4.000
5 2019-09-06T00:40 74.900 79.359 81.700 29.700 0.000
6 2019-09-06T00:50 74.700 79.159 81.700 29.699 4.000
7 2019-09-06T01:00 74.600 79.197 82.700 29.697 0.000
8 2019-09-06T01:10 74.500 79.097 82.700 29.697 0.000
```

Line 8, Column 58 Tab Size: 4 Plain Text

\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

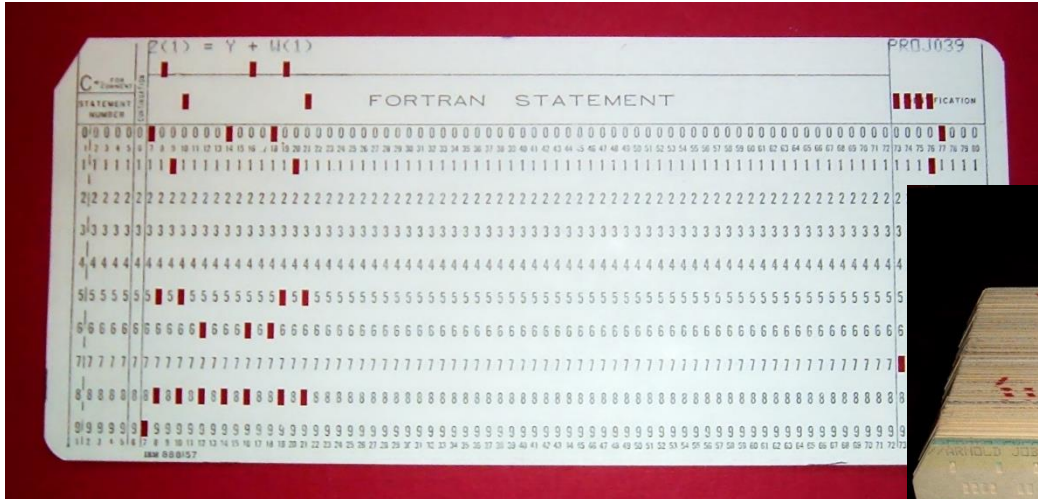


Altair 8800

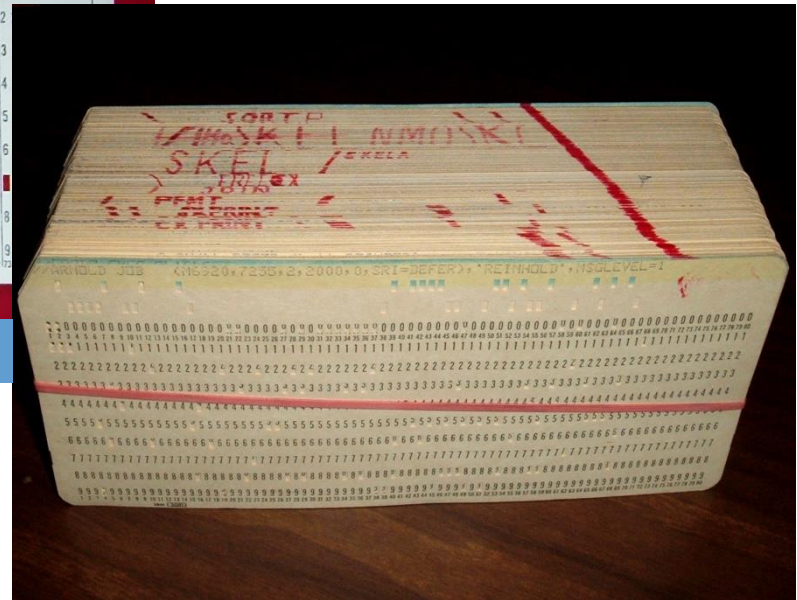


TSA Approved Apple I

# 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
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4
5   <groupId>org.jlab.clas12</groupId>
6   <artifactId>clas12-analysis</artifactId>
7   <version>0.3-SNAPSHOT</version>
8   <packaging>jar</packaging>
9
10  <properties>
11    <maven.compiler.source>11</maven.compiler.source>
12    <maven.compiler.target>11</maven.compiler.target>
13  </properties>
14  <repositories>
15    <repository>
16      <id>jitpack.io</id>
17      <url>https://jitpack.io</url>
18    </repository>
19    <repository>
20      <id>freehep</id>
21      <url>https://java.freehep.org/maven2</url>
22    </repository>
23  </repositories>
```

Custom Tags

Maven POM Build file for clas12-analysis library



# JSON

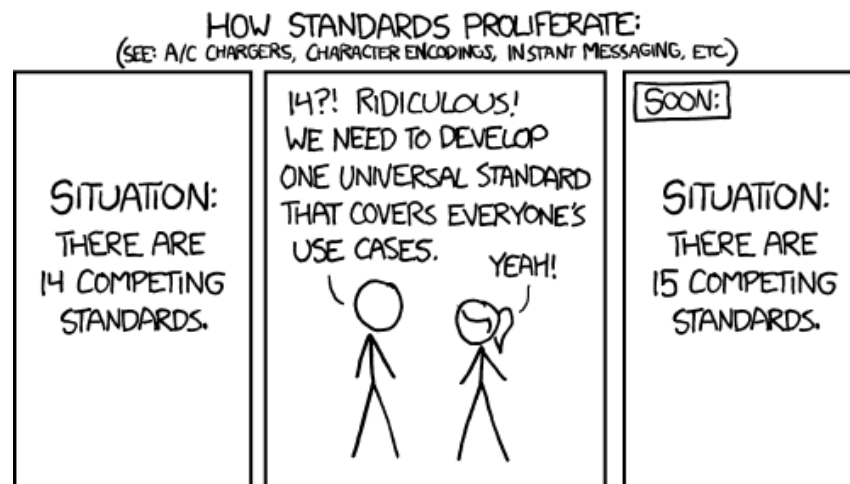
- Javascript Object Notation
- ECMA-262 standard (~1999)
- Format is based on two structures:
  - A collection of name – value pairs
  - Ordered list of values such as: Arrays, vectors, lists

```
{
  "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

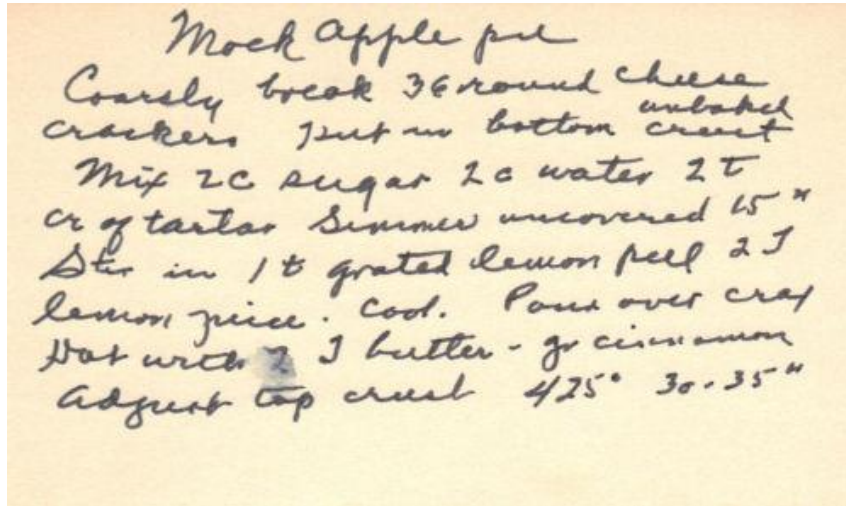


Data reduction



Data discretization

# Data Munging/Wrangling



Hand Written Recipe\*

## INGREDIENTS

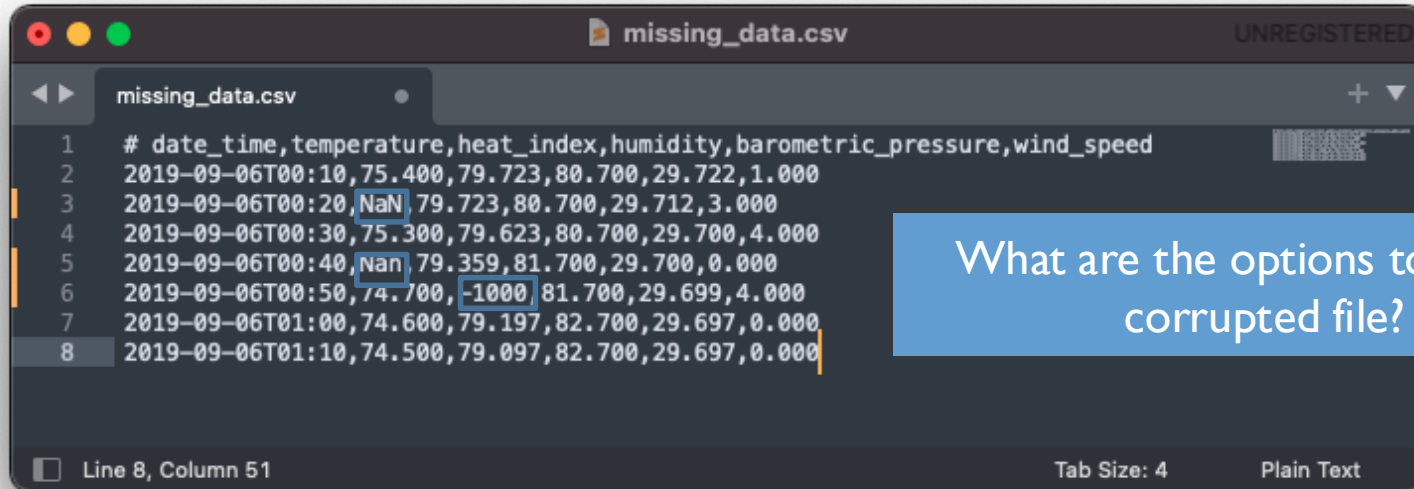
- 1 batch flaky pie crust, divided into 2 unequal halves
- ¼ to ⅓ cup sugar, depending on sweetness of apples
- ½ teaspoon cinnamon
- ¼ teaspoon mace
- 6 large, tart cooking apples, such as Cortlands or pippins, peeled, cored and cut in 6ths
- 1 to 1 ½ tablespoons rosewater, depending on taste
- 4 tablespoons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



The screenshot shows a text editor window titled 'missing\_data.csv' with the following content:

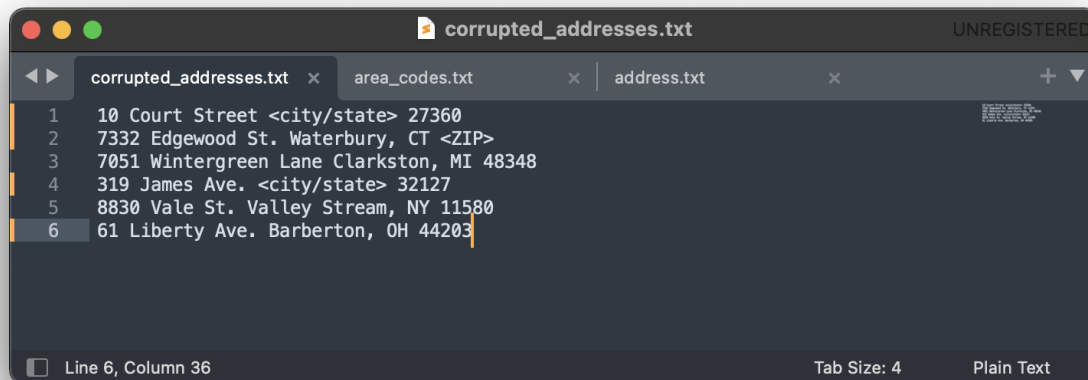
```
1 # date_time,temperature,heat_index,humidity,barometric_pressure,wind_speed
2 2019-09-06T00:10,75.400,79.723,80.700,29.722,1.000
3 2019-09-06T00:20,NaN,79.723,80.700,29.712,3.000
4 2019-09-06T00:30,75.300,79.623,80.700,29.700,4.000
5 2019-09-06T00:40,nan,79.359,81.700,29.700,0.000
6 2019-09-06T00:50,74.700,-1000,81.700,29.699,4.000
7 2019-09-06T01:00,74.600,79.197,82.700,29.697,0.000
8 2019-09-06T01:10,74.500,79.097,82.700,29.697,0.000
```

The status bar at the bottom indicates 'Line 8, Column 51', 'Tab Size: 4', and 'Plain Text'.

What are the options to fix this corrupted file?

# Data Integration / Cleaning

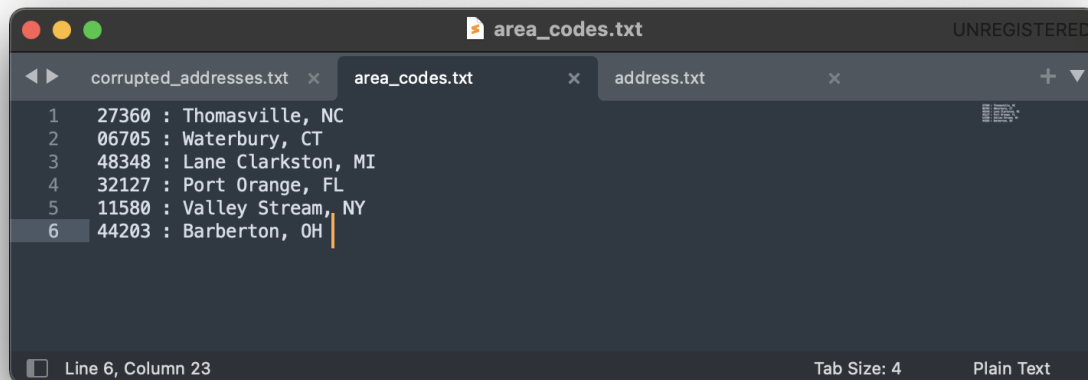
- How can we combine data sources to fix our corrupted address book?



A screenshot of a text editor window titled 'corrupted\_addresses.txt'. The editor shows six lines of text, each representing an address with some fields corrupted by '<city/state>' and '<ZIP>'. The cursor is positioned at the end of the sixth line.

```
1 10 Court Street <city/state> 27360
2 7332 Edgewood St. Waterbury, CT <ZIP>
3 7051 Wintergreen Lane Clarkston, MI 48348
4 319 James Ave. <city/state> 32127
5 8830 Vale St. Valley Stream, NY 11580
6 61 Liberty Ave. Barberton, OH 44203
```

Line 6, Column 36



A screenshot of a text editor window titled 'area\_codes.txt'. The editor shows six lines of text, each representing a ZIP code followed by a colon and the corresponding city and state. The cursor is positioned at the end of the sixth line.

```
1 27360 : Thomasville, NC
2 06705 : Waterbury, CT
3 48348 : Lane Clarkston, MI
4 32127 : Port Orange, FL
5 11580 : Valley Stream, NY
6 44203 : Barberton, OH
```

Line 6, Column 23

# Data Integration / Cleaning

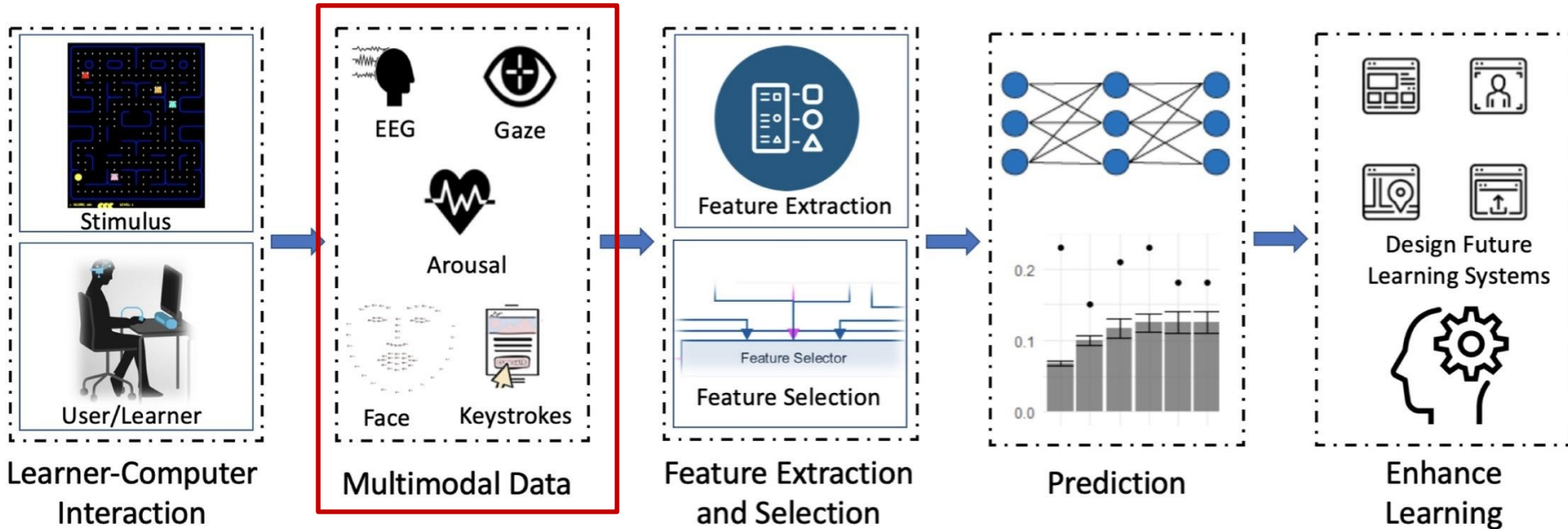
```
corrupted_addresses.txt UNREGISTERED
corrupted_addresses.txt x area_codes.txt x .txt x + v
1 10 Court Street <city/state> 27360
2 7332 Edgewood St. Waterbury, CT <ZIP>
3 7051 Wintergreen Lane Clarkston, MI 48348
4 319 James Ave. <city/state> 32127
5 8830 Vale St. Valley Stream, NY 11580
6 61 Liberty Ave. Barberton, OH 44203
Line 6, Column 36 Tab Size: 4
```

```
area_codes.txt UNREGISTERED
corrupted_addresses area_codes.txt x address.txt x + v
1 27360 : Thomasville, NC
2 06705 : Waterbury, CT
3 48348 : Lane Clarkston, MI
4 32127 : Port Orange, FL
5 11580 : Valley Stream, NY
6 44203 : Barberton, OH
Line 1, Column 1 Tab Size: 4
```

```
address.txt UNREGISTERED
corrupted_addresses area_codes.txt x address.txt x + v
1 10 Court Street Thomasville, NC 27360
2 7332 Edgewood St. Waterbury, CT 06705
3 7051 Wintergreen Lane Clarkston, MI 48348
4 319 James Ave. Port Orange, FL 32127
5 8830 Vale St. Valley Stream, NY 11580
6 61 Liberty Ave. Barberton, OH 44203
Line 1, Column 1 Tab Size: 4
```

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

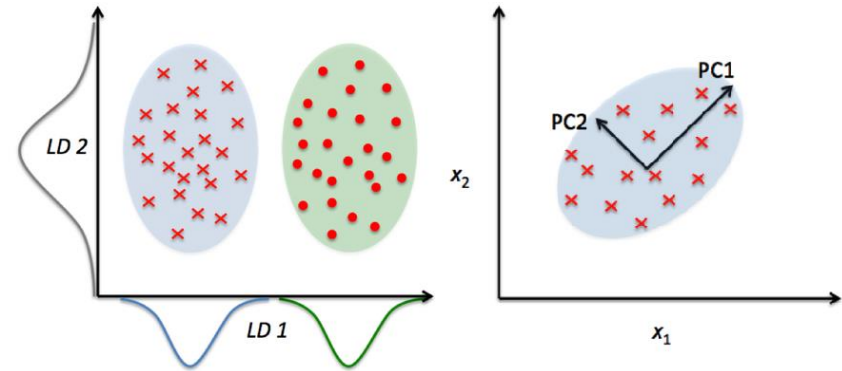


# Data Transformation

- 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](https://towardsdatascience.com)

# Discretization

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

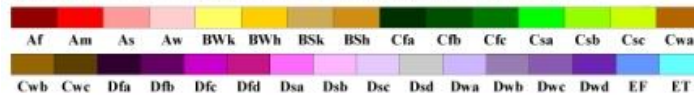
Grade	Letter Grade
94.5	A
78.3	C+
92	A-

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

# Data Discretization

## World Map of Köppen–Geiger Climate Classification

updated with CRU TS 2.1 temperature and VASCLimO v1.1 precipitation data 1951 to 2000



### Main climates

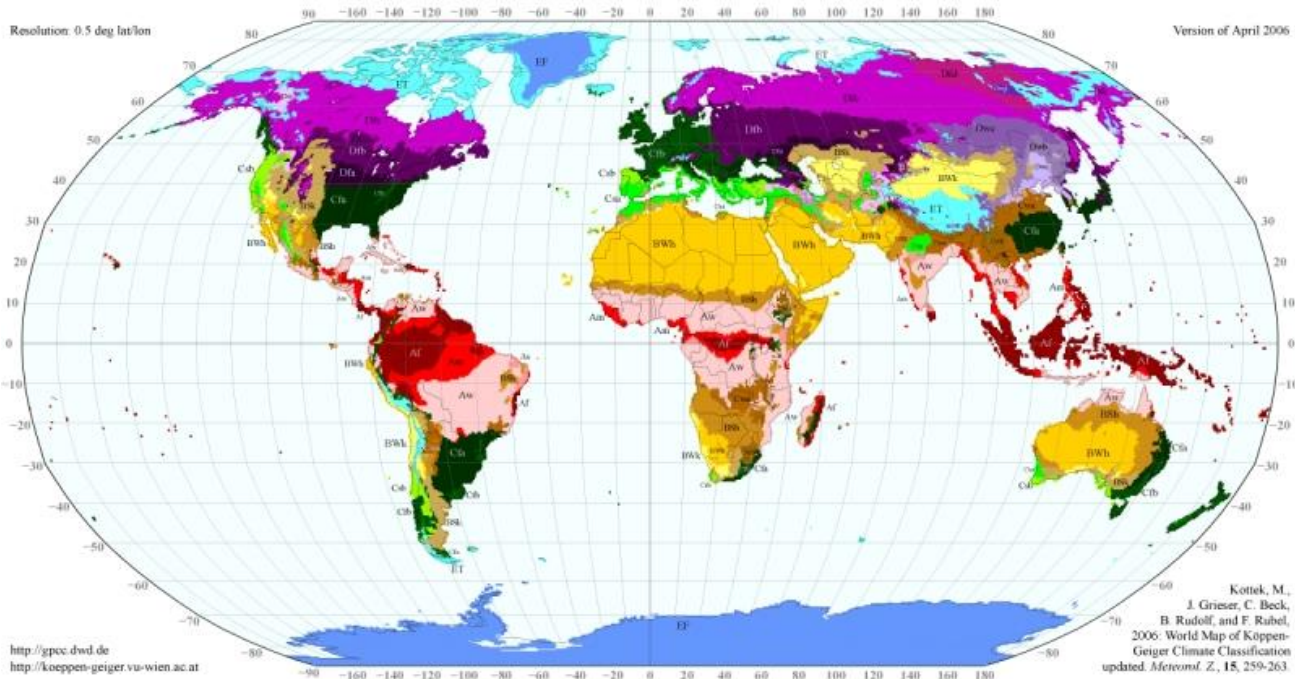
A: equatorial  
B: arid  
C: warm temperate  
D: snow  
E: polar

### Precipitation

W: desert  
S: steppe  
f: fully humid  
s: summer dry  
w: winter dry  
m: monsoonal

### Temperature

h: hot arid  
k: cold arid  
a: hot summer  
b: warm summer  
c: cool summer  
d: extremely continental  
F: polar frost  
T: polar tundra



# PANDAS

---

# Pandas Dataframes and Dataseries

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

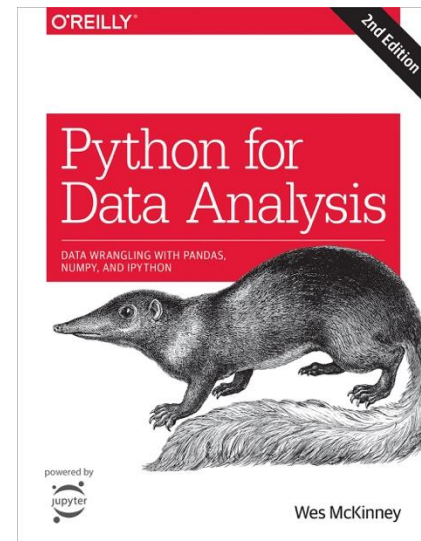


Diagram illustrating the structure of a Pandas DataFrame. The columns are labeled "Name", "Team", "Number", "Position", and "Age". The rows are labeled "Rows". The data is presented in a table format:

	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

The diagram highlights the "Data" structure with a pink box around the "Number" column and a purple box around the "Position" column. The "Rows" label is on the left, and the "Columns" label is at the top. A small logo is visible in the bottom right corner of the diagram area.

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

```
In [4]: obj = pd.Series([4, 7, -5, 3])  
obj
```

```
Out [4]: 0    4  
         1    7  
         2   -5  
         3    3  
         dtype: int64
```

```
In [6]: obj.values
```

```
Out [6]: array([ 4,  7, -5,  3])
```

ndarray

```
In [7]: obj.index
```

```
Out [7]: RangeIndex(start=0, stop=4, step=1)
```

# 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
<b>one</b>	2000	Ohio	1.5	16.5
<b>two</b>	2001	Ohio	1.7	16.5
<b>three</b>	2002	Ohio	3.6	16.5
<b>four</b>	2001	Nevada	2.4	16.5
<b>five</b>	2002	Nevada	2.9	16.5
<b>six</b>	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
<b>one</b>	2000	Ohio	1.5	0.0
<b>two</b>	2001	Ohio	1.7	1.0
<b>three</b>	2002	Ohio	3.6	2.0
<b>four</b>	2001	Nevada	2.4	3.0
<b>five</b>	2002	Nevada	2.9	4.0
<b>six</b>	2003	Nevada	3.2	5.0

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

Out [58]:

	year	state	pop	debt
<b>one</b>	2000	Ohio	1.5	6
<b>two</b>	2001	Ohio	1.7	5
<b>three</b>	2002	Ohio	3.6	4
<b>four</b>	2001	Nevada	2.4	3
<b>five</b>	2002	Nevada	2.9	2
<b>six</b>	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
<b>one</b>	2000	Ohio	1.5	NaN
<b>two</b>	2001	Ohio	1.7	-1.2
<b>three</b>	2002	Ohio	3.6	NaN
<b>four</b>	2001	Nevada	2.4	-1.5
<b>five</b>	2002	Nevada	2.9	-1.7
<b>six</b>	2003	Nevada	3.2	NaN

# Creating and deleting Columns

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

Out[64]:

	year	state	pop	debt	eastern
<b>one</b>	2000	Ohio	1.5	NaN	True
<b>two</b>	2001	Ohio	1.7	-1.2	True
<b>three</b>	2002	Ohio	3.6	NaN	True
<b>four</b>	2001	Nevada	2.4	-1.5	False
<b>five</b>	2002	Nevada	2.9	-1.7	False
<b>six</b>	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='object')

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

Out[66]:

	year	state	pop	debt
<b>one</b>	2000	Ohio	1.5	NaN
<b>two</b>	2001	Ohio	1.7	-1.2
<b>three</b>	2002	Ohio	3.6	NaN
<b>four</b>	2001	Nevada	2.4	-1.5
<b>five</b>	2002	Nevada	2.9	-1.7
<b>six</b>	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]:

	<b>Nevada</b>	<b>Ohio</b>
<b>2001</b>	2.4	1.7
<b>2002</b>	2.9	3.6
<b>2000</b>	NaN	1.5

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

Out [69]:

	<b>2001</b>	<b>2002</b>	<b>2000</b>
<b>Nevada</b>	2.4	2.9	NaN
<b>Ohio</b>	1.7	3.6	1.5

# Demo time

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