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
- If you do not have anaconda or python with matplotlib, numpy, and pandas installed we will be using google colab

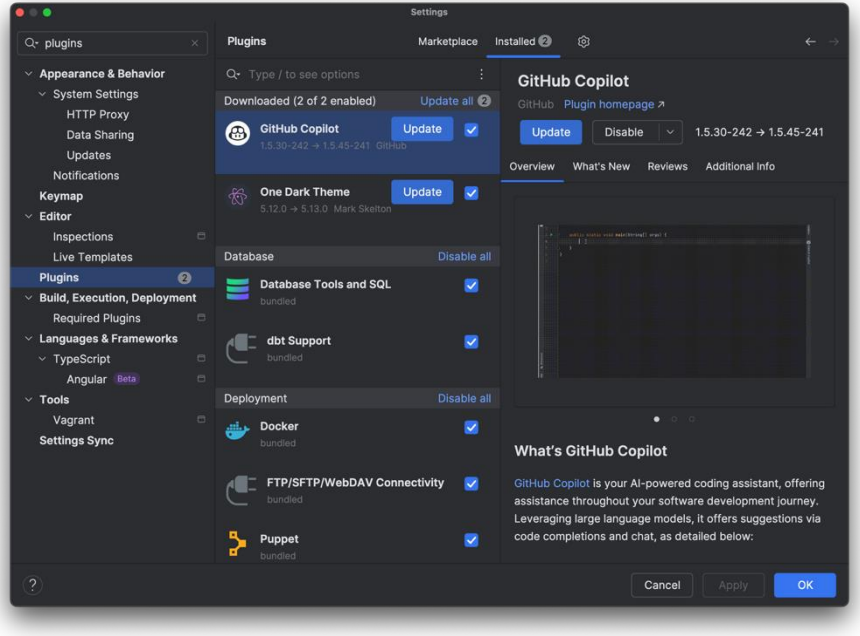
Install Python 3, PyCharm on your computer

- Install Python 3.12
  - numpy, matplotlib, scikit-learn, scipy, tensorflow (preferably with gpu support if available), keras, pandas, folium, and plotly
  - Anaconda is blocked onsite
- PyCharm IDE https://www.jetbrains.com/pycharm/
  - Professional edition is preferred!
  - Professional version is available free for students/faculty
    - Needs verification via email

How to install Github Copilot

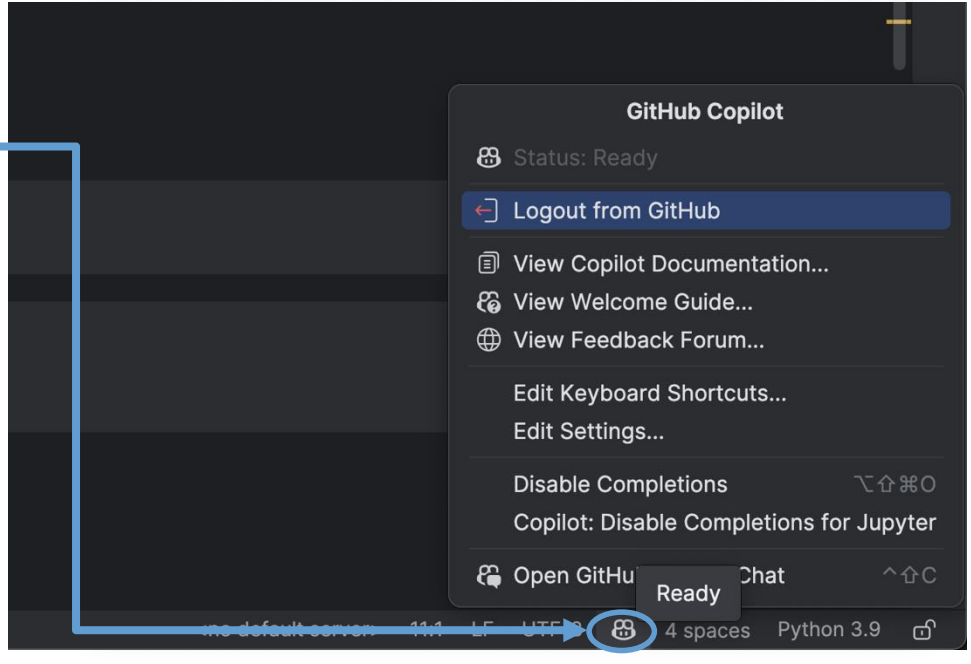
Github has temporarily paused new copilot education accounts as of April 2026

- Create a github account if needed and submit a request to github education and get approved
- Install github copilot plugin
  - Open Pycharm Settings
  - Search plugins
  - Install GitHub Copilot Plugin



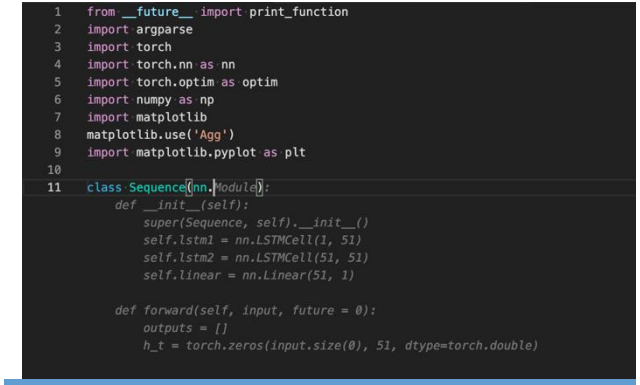
How to install Github Copilot

- Once Installed make sure you log in to github
- Icon visible in bottom tray of pycharm

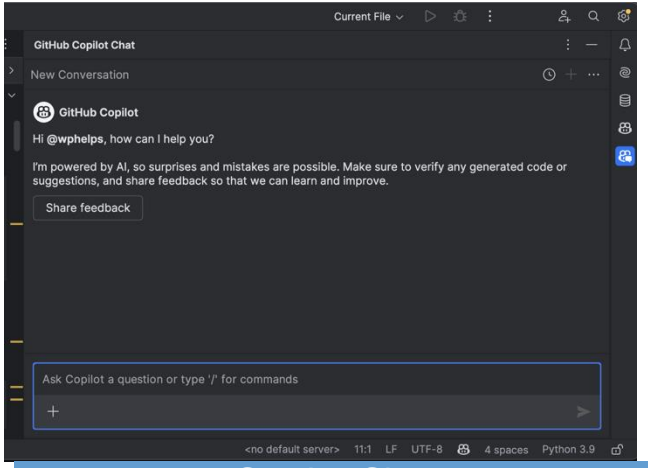


Now you should see PyCharm Completions

- Use comments to help prompt good autocompletions
- Use the chat feature for help debugging and for code suggestions!
  - Honestly: Really great for blindly chucking the traceback into and getting pretty decent results



Copilot with an autocompletion visible



Copilot Chat

Course Topics*

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

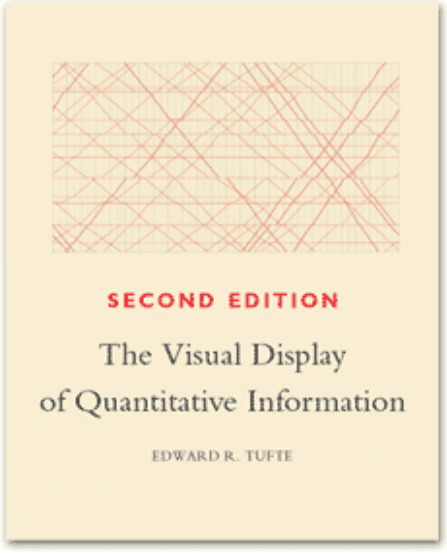
- **Lectures 1 & 2**
  - **Lecture 1**
    - Introduction to workflow and tools (git/anaconda/pycharm)
    - Introduction to Data Science
    - Python Review with jupyter notebooks
  - **Lecture 2**
    - Data Visualization
    - Principles of data visualization from Edward Tufte
    - Pandas introduction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

TUFTE/MINIMALISM

Why do we visualize data?

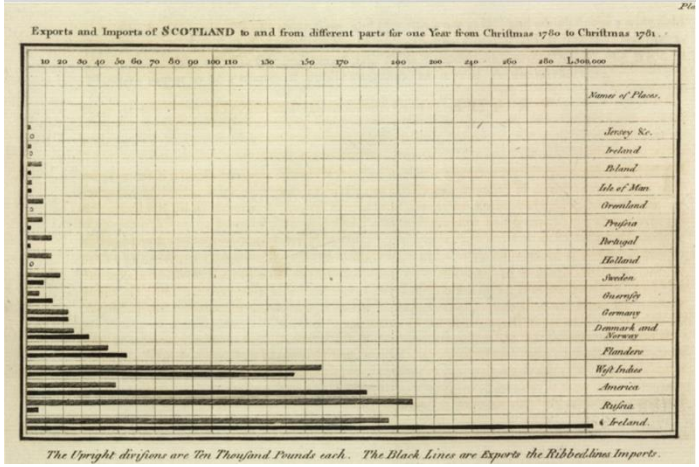
- We visualize data in order to make it easier to see patterns and trends in the data
- By visualizing data we can extrapolate and interpret the data in new ways
- Data visualization is about effective communication

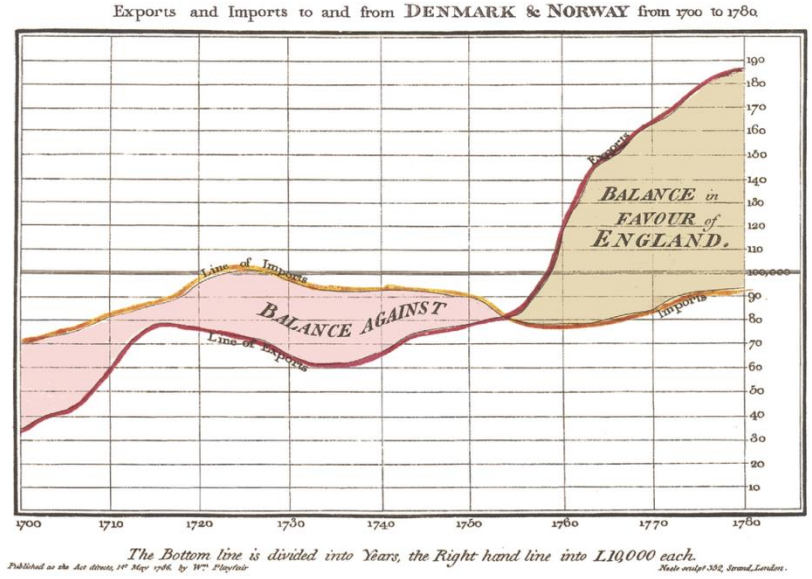


Many topics from Today's lecture come from Chapter 5 and chapter 9

History of Data visualization

- It was not until 1750-1800 that statistical graphics began to appear
- William Playfair (1759-1823)
  - Credited with creating line, bar, area, and pie chart
  - Tables of data < Graphics





W. Playfair The Commercial and Political Atlas 1786

Tips for making figures

- Large font for labels/titles in presentation
  - Slightly too large is better than too small
- Spend the time to revise your figures
  - You can really tell how much thought and care went into the figure
- Proper units!

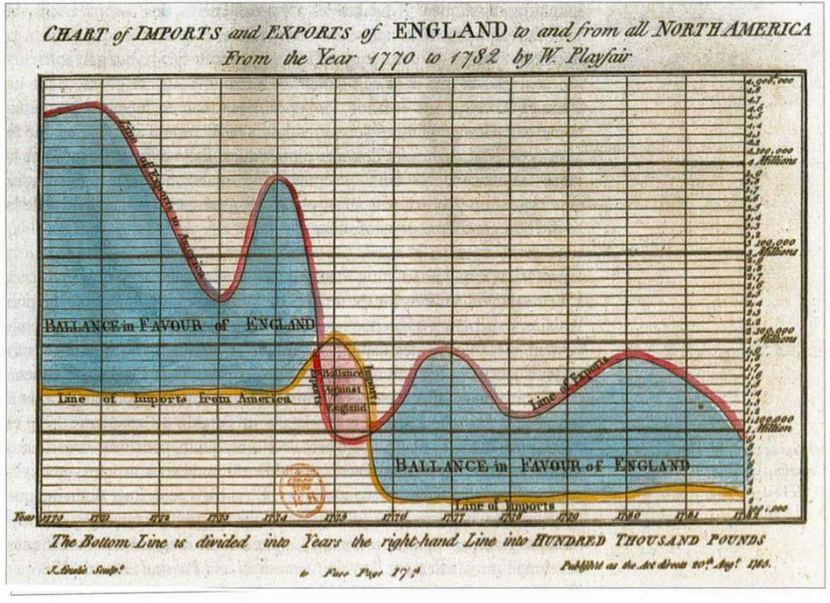
Tufte's recommendations

- Above all else, show data
- Maximize the data-ink ratio
- Erase non-data ink
  - Within reason!
- Erase redundant data ink
- Revise and edit

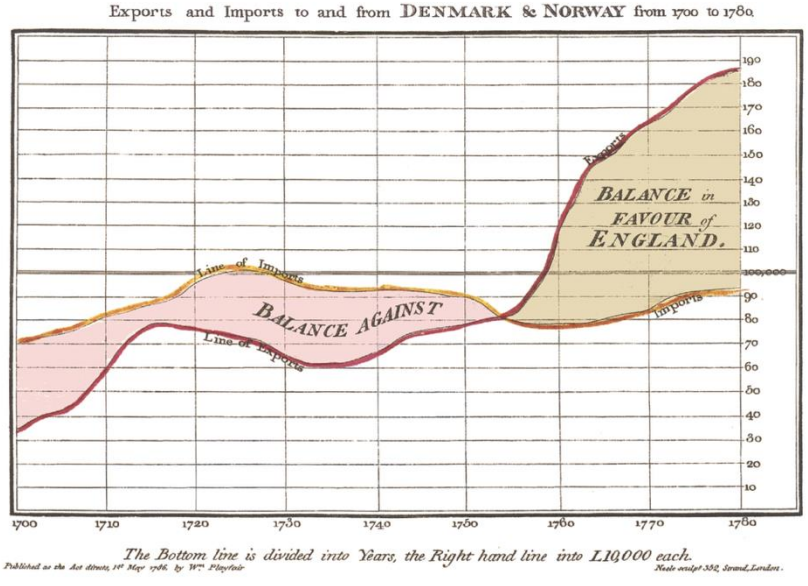
$$\text{Data-ink ratio} = \frac{\text{data-ink}}{\text{total ink used to print the graphic}}$$

Goal is to approach a data-ink ratio of 1.0

Example: William Playfair

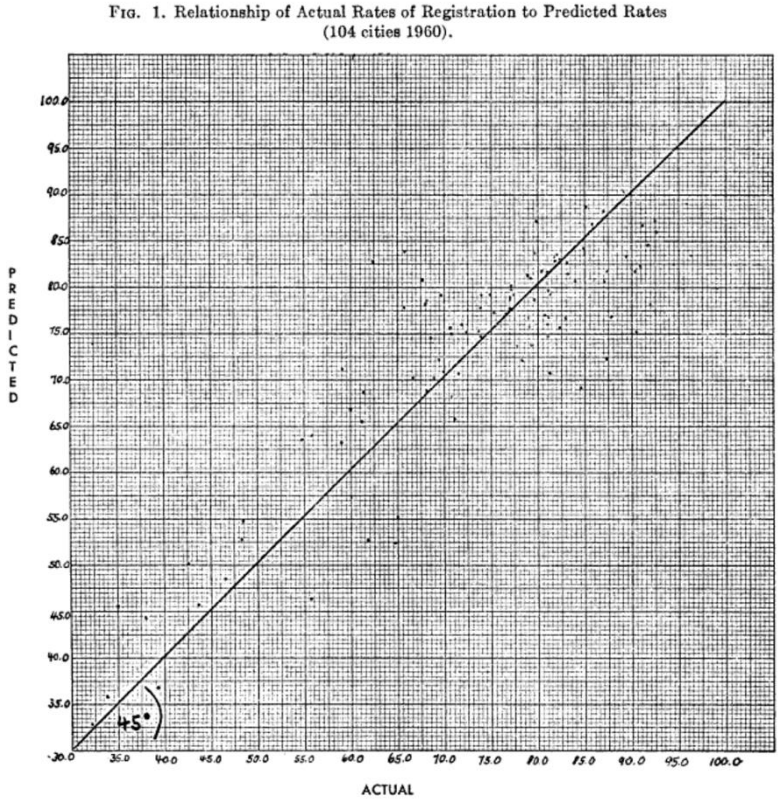


W Playfair 1785

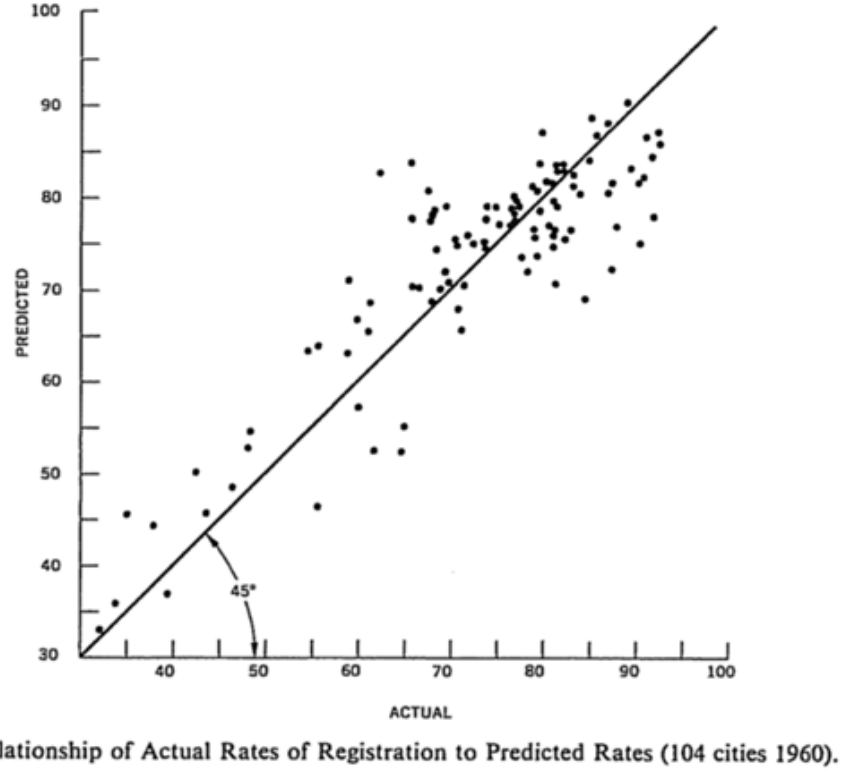


W. Playfair The Commercial and Political Atlas 1786

Example – Voting Registration Study



Kelley, Ayres, and Bowen – American Political Science Rev, 61 (1967)



E. Tufte - The Quantitative Analysis of Social Problems (1970)

Aesthetics

- Directly from Tufte:
  - Have a properly chosen format and design
  - Use words, numbers, and drawing together
  - Reflect a balance, a proportion, a sense of relevant scale
  - Display an accessible complexity of detail
  - Have a narrative
  - Drawn in a professional manner
  - Avoid chartjunk

Accessible Complexity

Friendly	Unfriendly
words are spelled out, mysterious and elaborate encoding avoided	abbreviations abound, requiring the viewer to sort through text to decode abbreviations
words run from left to right, the usual direction for reading occidental languages	words run vertically, particularly along the Y-axis; words run in several different directions
little messages help explain data	graphic is cryptic, requires repeated references to scattered text
elaborately encoded shadings, cross-hatching, and colors are avoided; instead, labels are placed on the graphic itself; no legend is required	obscure codings require going back and forth between legend and graphic
graphic attracts viewer, provokes curiosity	graphic is repellent, filled with chartjunk
colors, if used, are chosen so that the color-deficient and color-blind (5 to 10 percent of viewers) can make sense of the graphic (blue can be distinguished from other colors by most color-deficient people)	design insensitive to color-deficient viewers; red and green used for essential contrasts
type is clear, precise, modest; lettering may be done by hand	type is clotted, overbearing
type is upper-and-lower case, with serifs	type is all capitals, sans serif

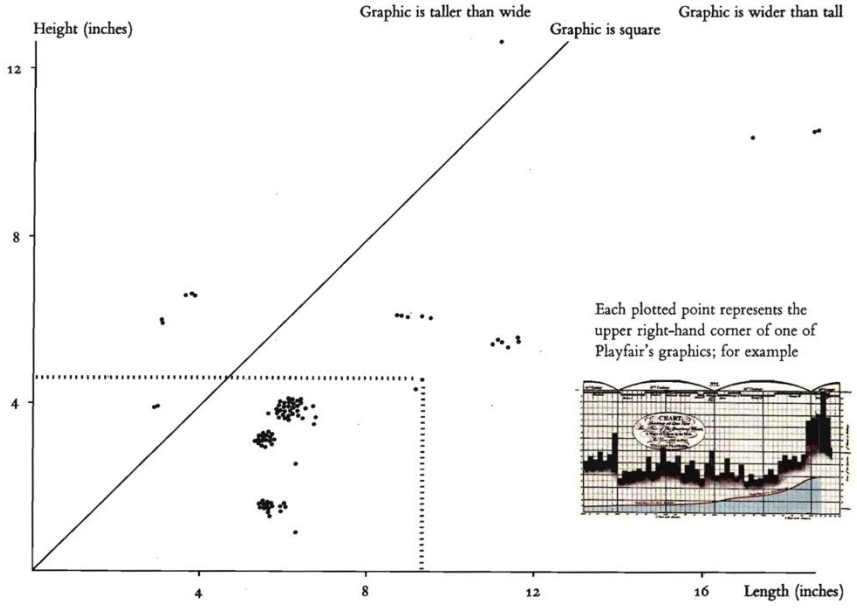
Visual Display of Quantitative Information page 183

There is no consensus about which is more readable, serif or sans serif, and there are many studies finding no difference at all.

http://alexpoole.info/blog/which-are-more-legible-serif-or-sans-serif-typefaces/

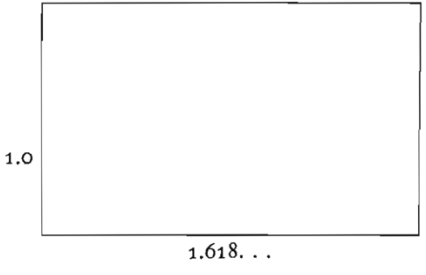
Aspect Ratio

- This is subjective, but generally wider plots are better

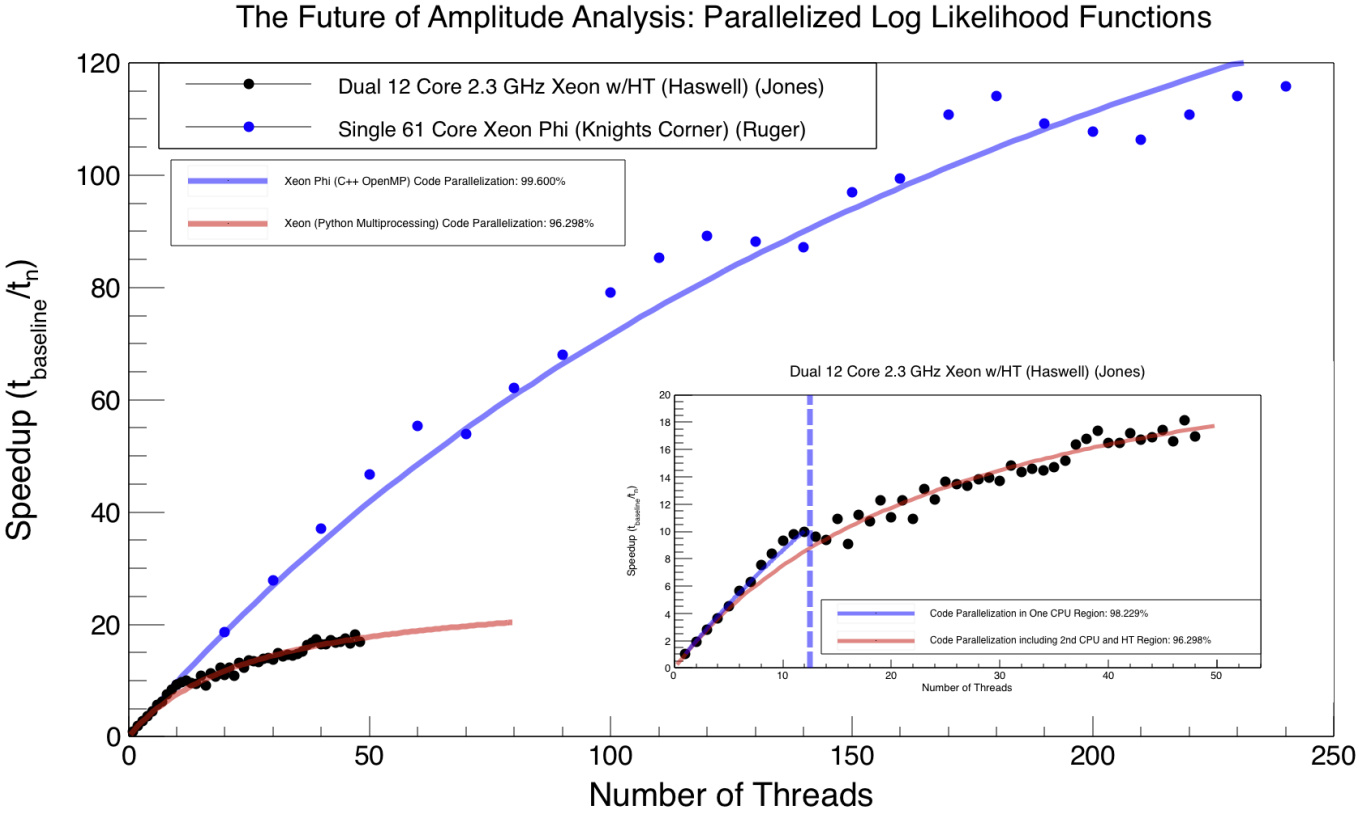


Analysis of William Playfair's figures:

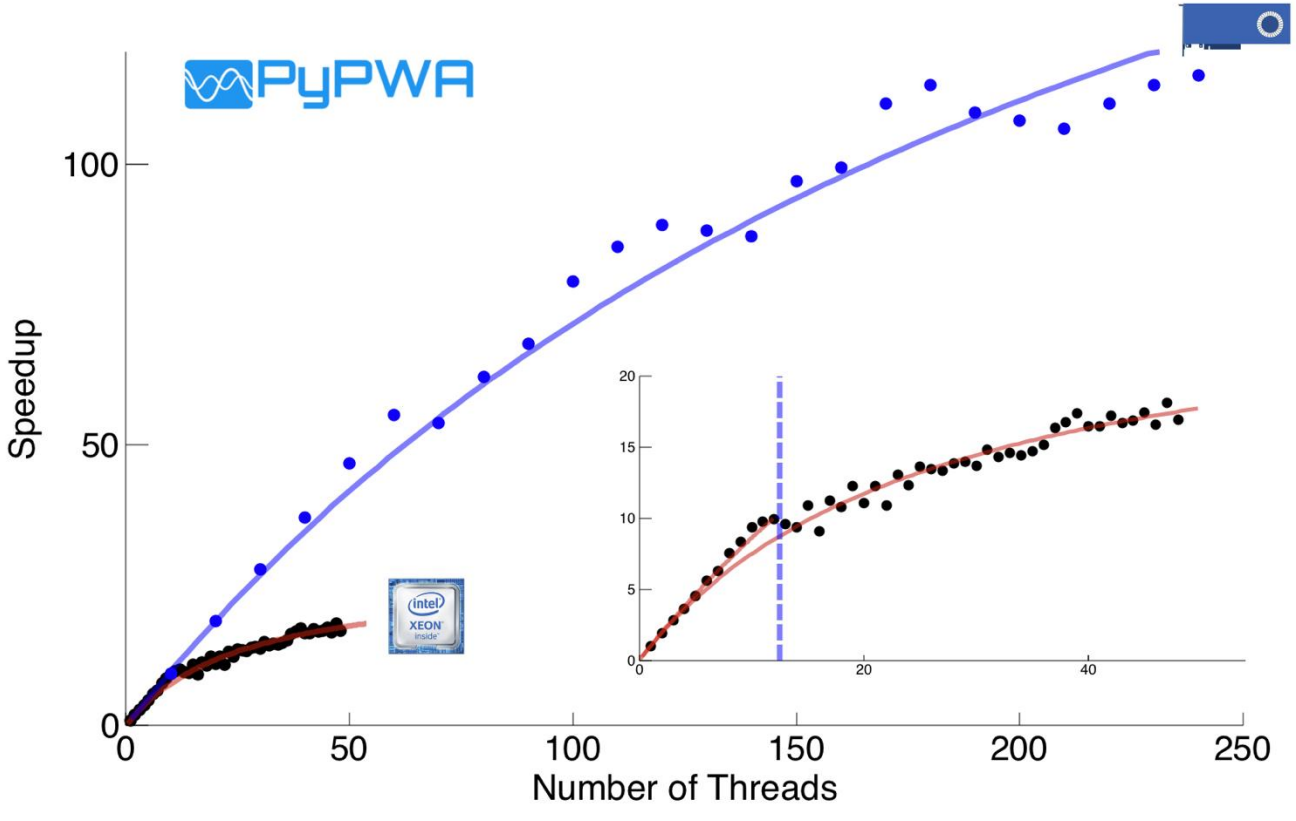
In turn the Golden Rectangle is



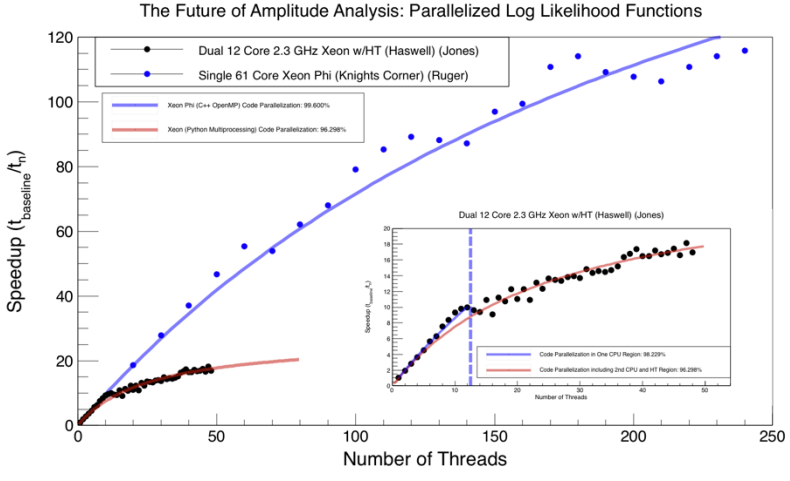
Before


The Future of Amplitude Analysis: Parallelized Log Likelihood Functions
Dual 12 Core 2.3 GHz Xeon w/HT (Haswell) (Jones)
Single 61 Core Xeon Phi (Knights Corner) (Ruger)
Xeon Phi (C++ OpenMP) Code Parallelization: 99.600%
Xeon (Python Multiprocessing) Code Parallelization: 96.298%
Speedup ($t_{baseline}/t_n$)
Number of Threads
Dual 12 Core 2.3 GHz Xeon w/HT (Haswell) (Jones)
Code Parallelization in One CPU Region: 98.229%
Code Parallelization including 2nd CPU and HT Region: 96.298%
Number of Threads

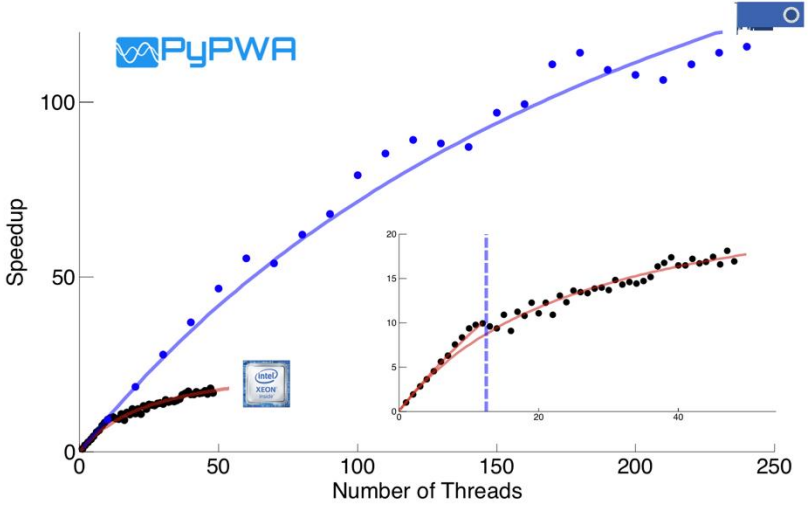
After


PyPWA
Speedup
Number of Threads
0
50
100
150
200
250
0
50
100
0
5
10
15
20
0
20
40
intel
XEON
inside

Implementing Tips from Edward Tufte

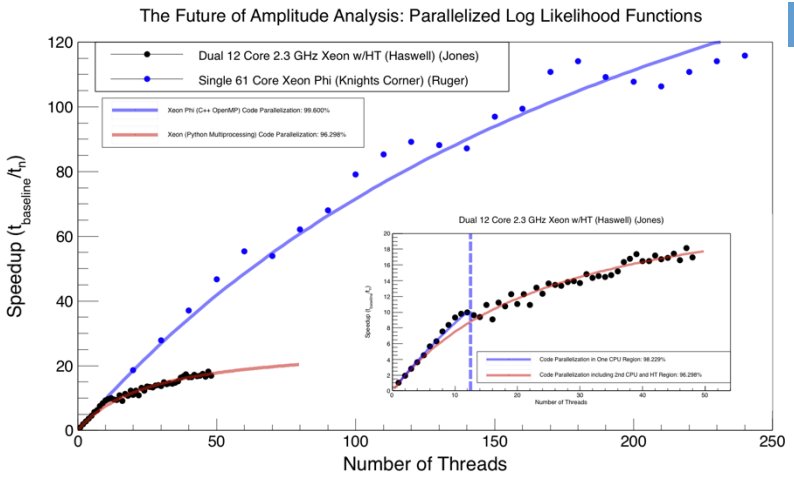


Very Technical, Distracting

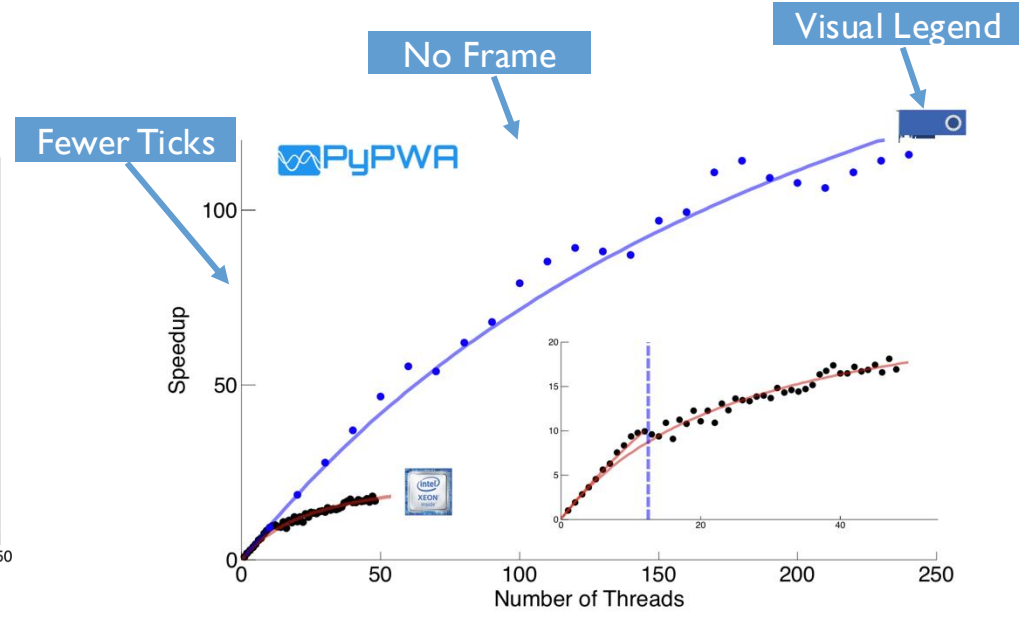


Applying Tufte's Suggestions

Implementing Tips from Edward Tufte



Very Technical, Distracting



Applying Tufte's Suggestions

Tufte's recommendations

- Above all else, show data
- Maximize the data-ink ratio
- Erase non-data ink
  - Within reason!
- Erase redundant data ink
- Revise and edit

$$\text{Data-ink ratio} = \frac{\text{data-ink}}{\text{total ink used to print the graphic}}$$

Goal is to approach a data-ink ratio of 1.0

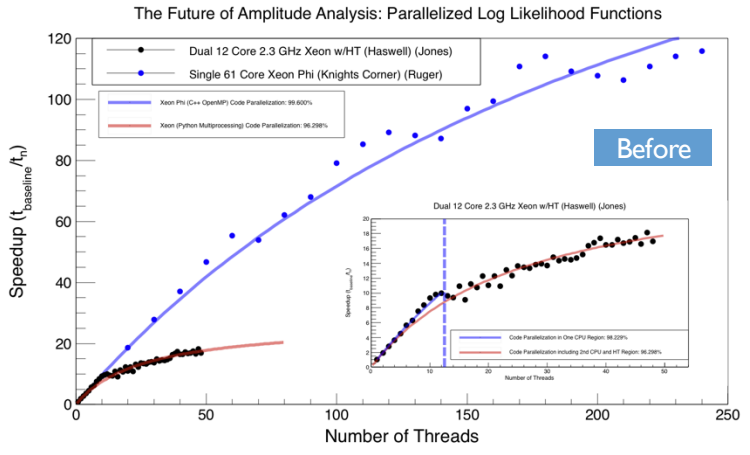
Matplotlib styles

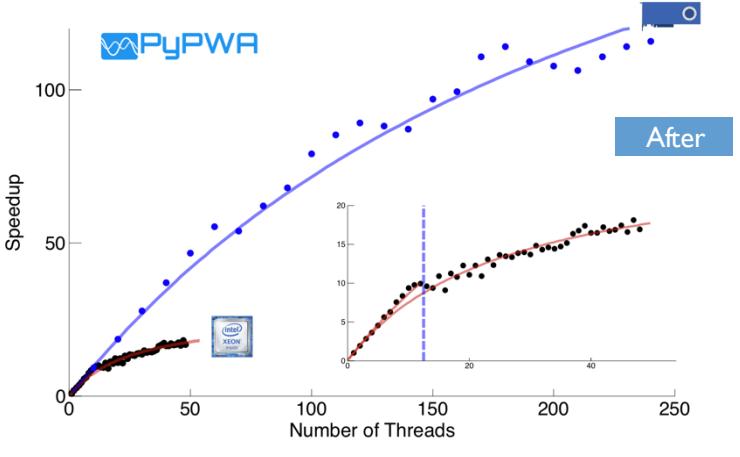
```
In [34]: plt.style.available
```

```
Out[34]: ['Solarize_Light2',
          '_classic_test_patch',
          'bmh',
          'classic',
          'dark_background',
          'fast',
          'fivethirtyeight',
          'ggplot',
          'grayscale',
          'seaborn',
          'seaborn-bright',
          'seaborn-colorblind',
          'seaborn-dark',
          'seaborn-dark-palette',
          'seaborn-darkgrid',
          'seaborn-deep',
          'seaborn-muted',
          'seaborn-notebook',
          'seaborn-paper',
          'seaborn-pastel',
          'seaborn-poster',
          'seaborn-talk',
          'seaborn-ticks',
          'seaborn-white',
          'seaborn-whitegrid',
          'tableau-colorblind10']
```

Data Visualization II

- Two primary uses for data visualization
  - To **explore** data
  - To **communicate** data
- For effective communication we explored Edward Tufte's suggestions for improving our data visualization last week
  - Tufte's style is mainly minimalism and is a good starting place for your own exploration into the field of data visualization (and developing your own style)





Edward Tufte's recommendations

- **Above all else, show data**
- Maximize the data-ink ratio
- Erase non-data ink
  - Within reason!
- Erase redundant data ink
- Revise and edit

$$\text{Data-ink ratio} = \frac{\text{data-ink}}{\text{total ink used to print the graphic}}$$

Goal is to approach a data-ink ratio of 1.0

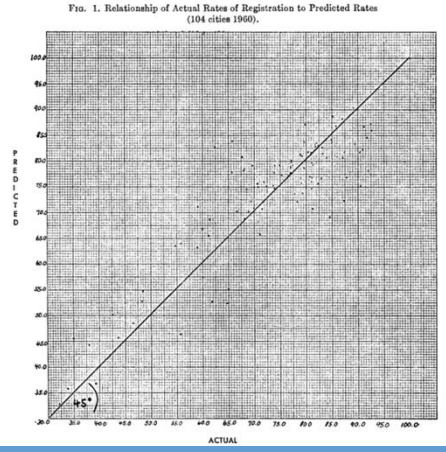
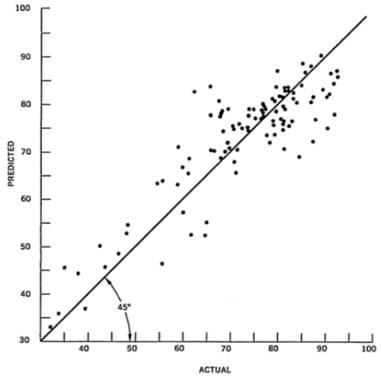


FIG. 1. Relationship of Actual Rates of Registration to Predicted Rates (104 cities 1960).

Kelley, Ayres, and Bowen – American Political Science Rev, 61 (1967)



Relationship of Actual Rates of Registration to Predicted Rates (104 cities 1960).

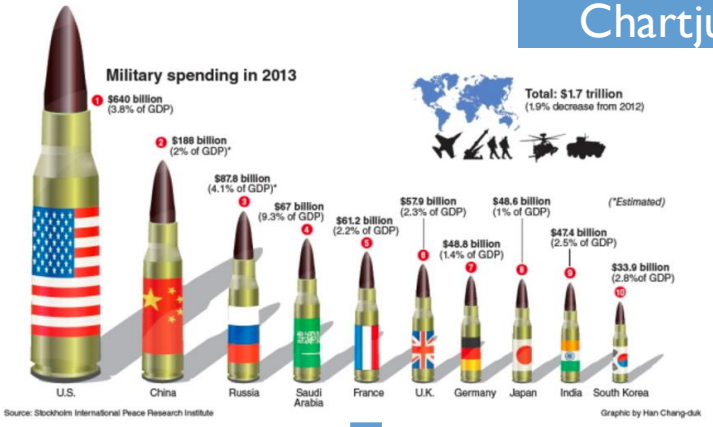
E. Tufte - The Quantitative Analysis of Social Problems (1970)

Aesthetics

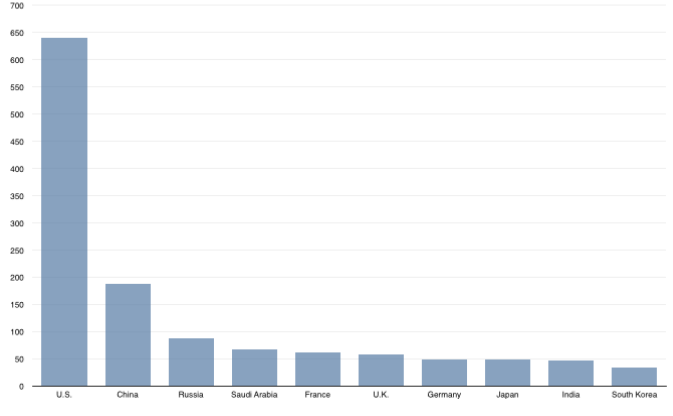
- Directly from Tufte:
  - Have a properly chosen format and design
  - Use words, numbers, and drawing together
  - Reflect a balance, a proportion, a sense of relevant scale
  - Display an accessible complexity of detail
  - Have a narrative
  - Drawn in a professional manner
  - Avoid chartjunk

"Chartjunk" was coined by Tufte in 1983

Chartjunk?



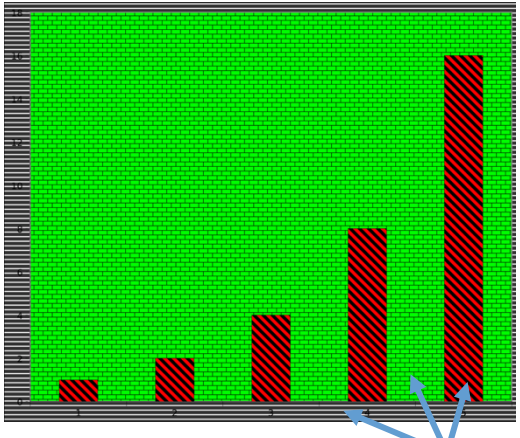
Proportional?



Example: Migle Rusteikaite. - Medium.com

Other Chartjunk Ex.

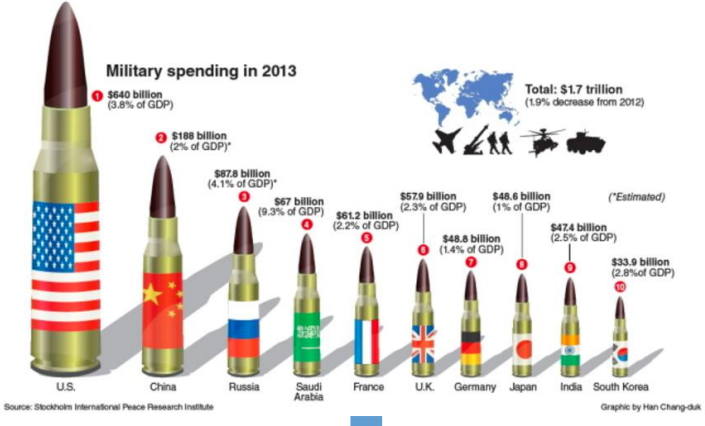
Chartjunk refers to all visual elements in charts and graphs that are not necessary to comprehend the information represented on the graph, or that distract the viewer from this information

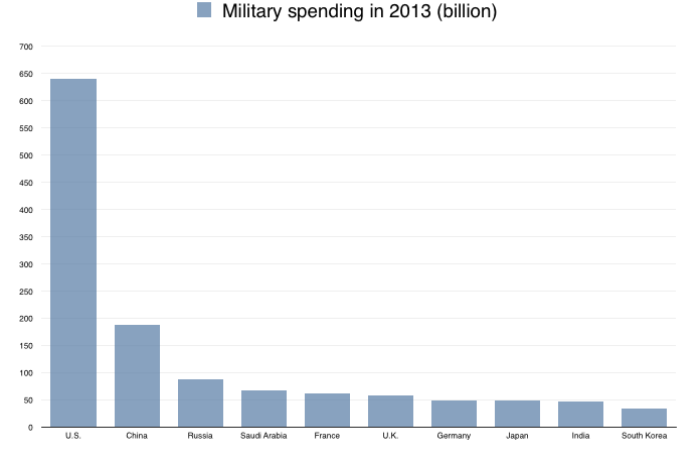


Textures

Distracting Gradient

Examples: https://en.wikipedia.org/wiki/Chartjunk

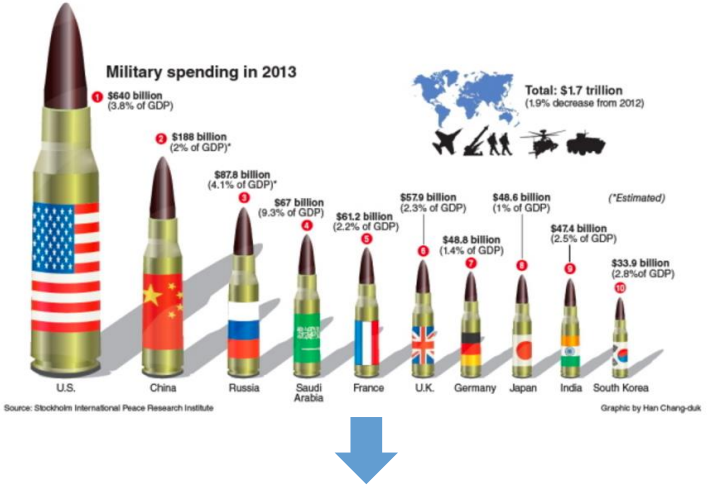


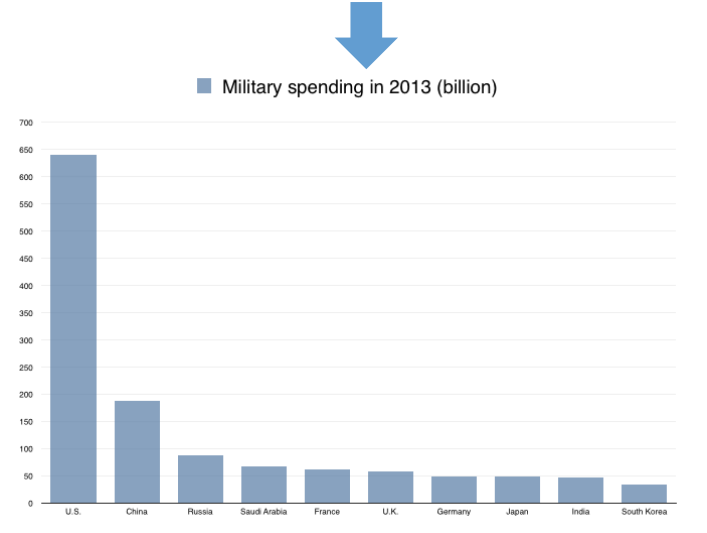


Example: Migle Rusteikaite. - Medium.com

Thoughts on Style

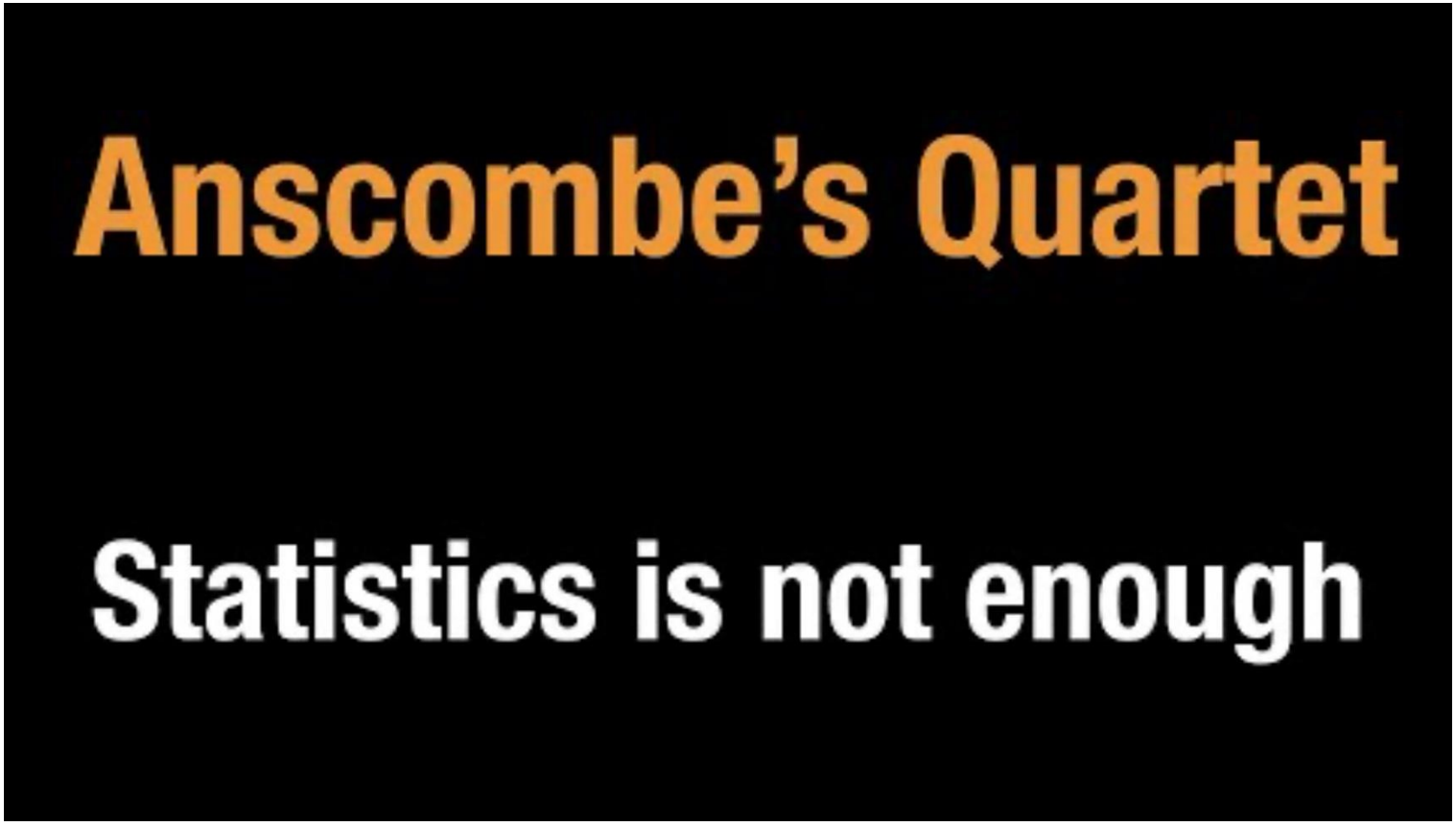
- The top figure lacks statistical fidelity
  - Cartridge length is not proportional to the defense budget
  - But! It is memorable and attractive to the eye.
- Bottom figure shows a minimalistic figure that gives the reader a better representation of the actual budgets
- Minimalism is not always the right choice, but it is a solid choice. Academic reports should typically be minimalistic, but marketing materials may be more like the first figure and stretch the truth a little.





Example: Migle Rusteikaite. - Medium.com

Note on Anscombe's Quartet



From a Data Visualization course SICE @ Indiana University

Tufte Style Demo

- Follow along!

MATPLOTLIB STYLE

Matplotlib styles

- The style package allows for setting styles easily
- More advanced: You can set up the rc file on your computer (sets default settings across your machine)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from cycler import cycler
plt.style.use('ggplot')
data = np.random.randn(50)
```

To list all available styles, use:

```
print(plt.style.available)
```

Out:

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'g
ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palet
te', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-past
el', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-color
blind10']
```

Matplotlib styles

Defining your own style

You can create custom styles and use them by calling `style.use` with the path or URL to the style sheet.

For example, you might want to create `./images/presentation.mplstyle` with the following:

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

Then, when you want to adapt a plot designed for a paper to one that looks good in a presentation, you can just add:

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use('./images/presentation.mplstyle')
```

Matplotlib styles

Composing styles

Style sheets are designed to be composed together. So you can have a style sheet that customizes colors and a separate style sheet that alters element sizes for presentations. These styles can easily be combined by passing a list of styles:

```
>>> import matplotlib.pyplot as plt
>>> plt.style.use(['dark_background', 'presentation'])
```

Note that styles further to the right will overwrite values that are already defined by styles on the left.

Temporary styling

If you only want to use a style for a specific block of code but don't want to change the global styling, the style package provides a context manager for limiting your changes to a specific scope. To isolate your styling changes, you can write something like the following:

```
with plt.style.context('dark_background'):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
plt.show()
```

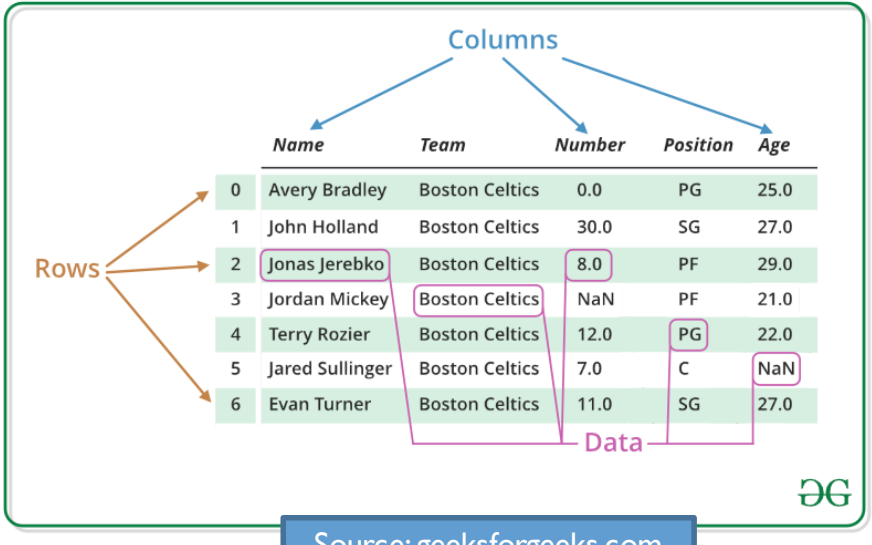
Note: If you are using the XKCD style and do not want to mess up the rest of your plots:

```
with plt.xkcd():
    plt.plot(x, y)
```

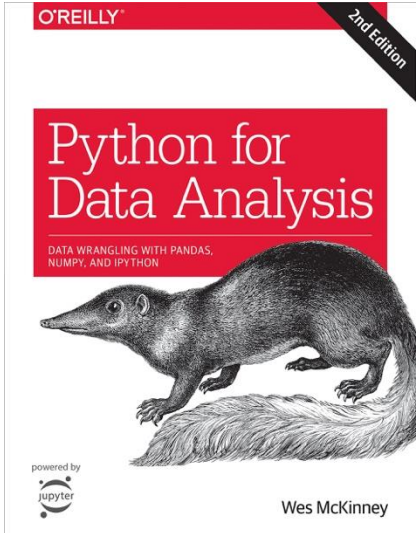
PANDAS

Pandas Dataframes and Dataseries

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



Source: geeksforgeeks.com



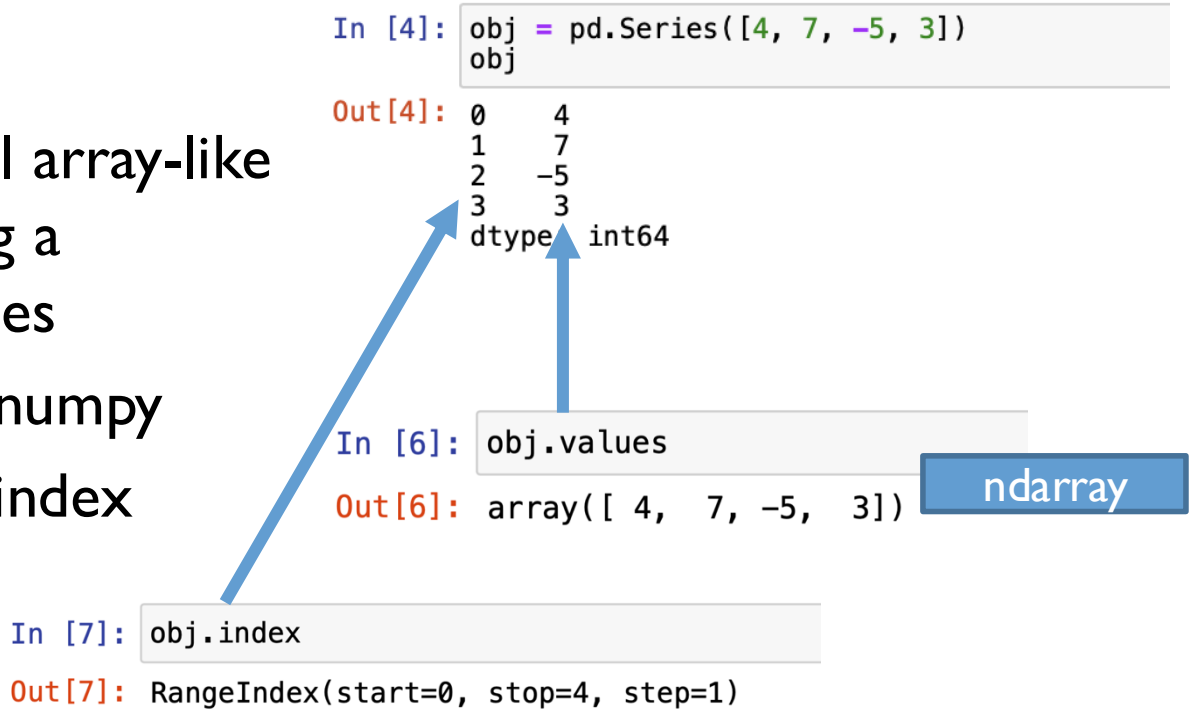
Lecture from Ch5

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 - 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

DATAFRAME OPERATIONS

Dropping Entries

- You may find yourself needing to remove entries from a dataframe or series

Dropping Entries from an Axis

```
In [78]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
         obj

Out[78]: a    0.0
         b    1.0
         c    2.0
         d    3.0
         e    4.0
         dtype: float64

In [79]: new_obj = obj.drop('c')
         new_obj

Out[79]: a    0.0
         b    1.0
         d    3.0
         e    4.0
         dtype: float64

In [80]: obj.drop(["b","c"])

Out[80]: a    0.0
         d    3.0
         e    4.0
         dtype: float64
```

Dropping rows/columns - Dataframes

- By default when you call drop in dataframes it will remove rows
- To remove columns you need to specify axis=1 or axis="columns"
- Like many functions in pandas you can specify inplace=true to do the operation inplace.
  - Very useful for large datasets, but you can lose data if you make a mistake.

```
In [84]: obj.drop('c', inplace=True)
         obj

Out[84]: a    0.0
         b    1.0
         d    3.0
         e    4.0
         dtype: float64
```

inplace drop

Out[81]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [82]: data.drop(['Colorado', 'Ohio'])
```

Out[82]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

```
In [83]: data.drop('two', axis=1)
         data.drop(['two', 'four'], axis='columns')
```

Out[83]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

Slicing - Dataframes

Slicing with Pandas dataframes provides rows. This is not intuitive, but it does provide a convenient way to access rows.

```
In [107]: data
```

Out[107]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [108]: data[:2]
```

Out[108]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

Slicing - Dataframes

Slicing with Pandas dataframes provides rows. This is not intuitive, but it does provide a convenient way to access rows.

```
In [107]: data
```

Out[107]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [108]: data[:2]
```

Out[108]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

```
In [103]: data[data['three'] > 5]
```

Out[103]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Slicing - Dataframes

Operators are overridden in the same was as numpy/Pandas Series

```
In [110]: data < 5
```

Out[110]:

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
In [112]: data[data < 5] = 0
          data
```

Out[112]:

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Reading Tabular Text Files

First line is read in as column names

Reading and Writing Data in Text Format

```
In [2]: !cat examples/ex1.csv
```

```
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
In [3]: df = pd.read_csv('examples/ex1.csv')
        df
```

Out[3]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [4]: pd.read_table('examples/ex1.csv', sep=',')
```

Out[4]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

CSV File without headers

In [5]:
```
!cat examples/ex2.csv
```

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

In [7]:
```
pd.read_csv('examples/ex2.csv', header=None)
```

Out[7]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [8]:
```
pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

Out[8]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Specifying Index Name

```
In [7]: pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

Out[7]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [8]: names = ['a', 'b', 'c', 'd', 'message']
        pd.read_csv('examples/ex2.csv', names=names, index_col='message')
```

Out[8]:

	a	b	c	d
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

Other Pandas File types

- read_csv()
- read_fwf()
- read_clipboard()
- read_excel()
- read_html()
- read_pickle()
- Sas, sql, stata, feather

- df = pd.read_xxxx()
- df.to_xxxx(filename)

Demo and Stock Price Warmup

- https://www.nasdaq.com/market-activity/quotes/historical
- Pick a company and let's see what we can find
  - You can also use marketwatch.com for historical data (<1 year)

Files in Google Colab

```
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Double-click (or enter) to edit

```
file = open('/content/drive/My Drive/aapl.csv')
```

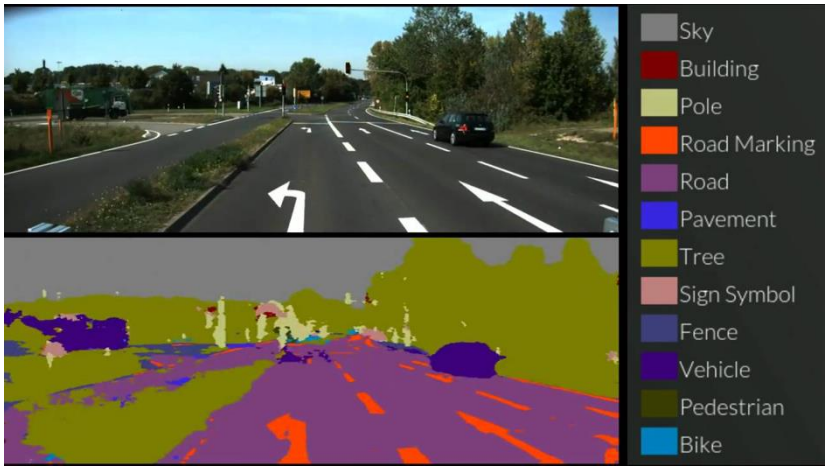
MACHINE LEARNING

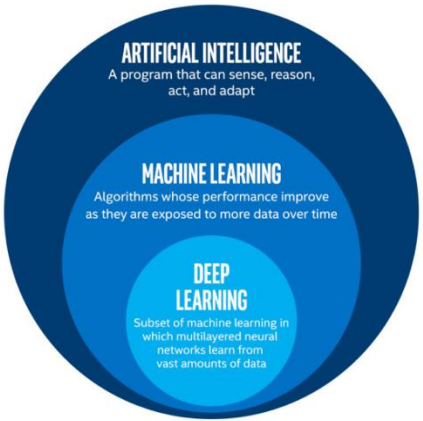
What is next?

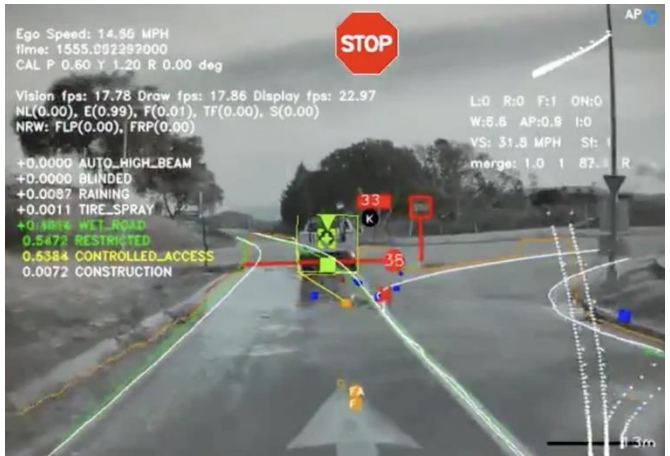
- Machine learning
  - Regression
    - Gradient descent fitting w/Scikit Learn
  - Classification
    - kNN Classification
  - Intro to Neural Networks
    - Keras MNIST dataset
  - Hyperparameter optimization
  - Pretrained neural networks?

What is machine learning?

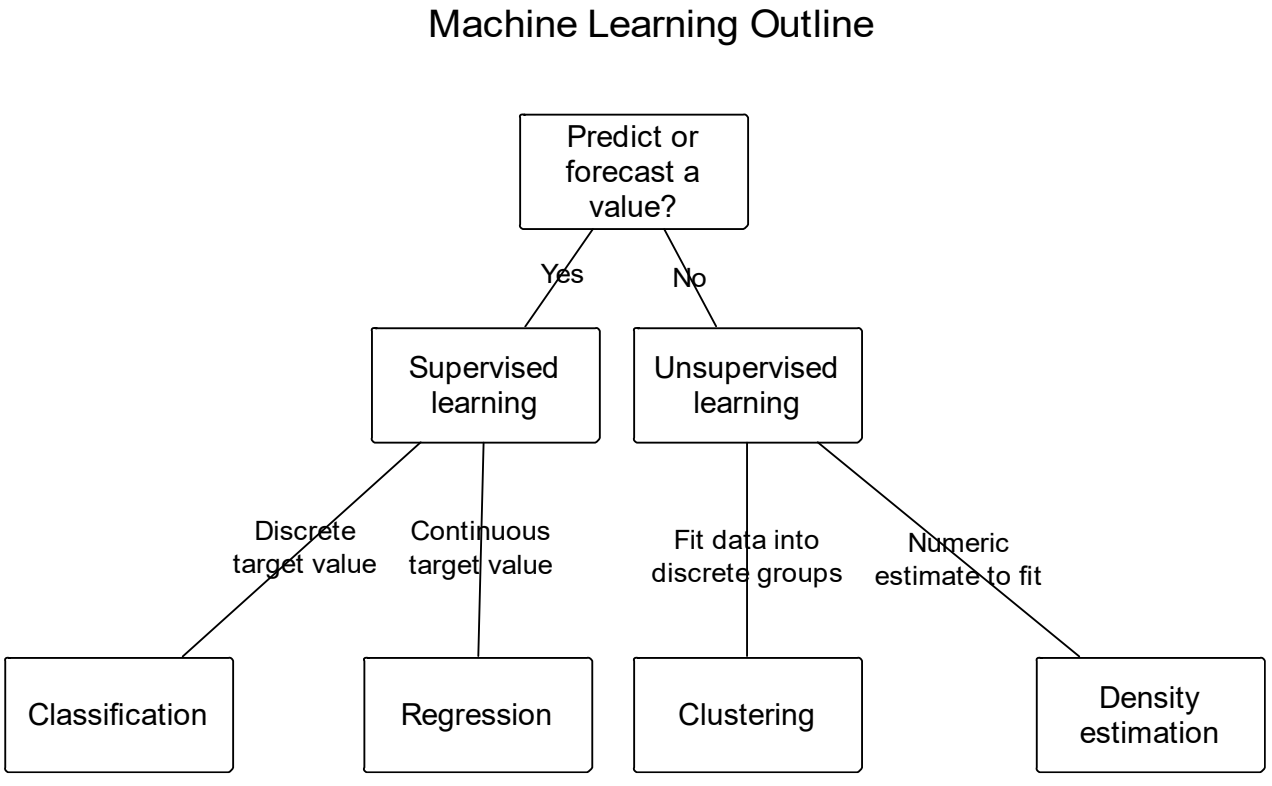
- Tom Mitchell: "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."
- Things learn when they change their behavior in a way that makes them perform better in the future.



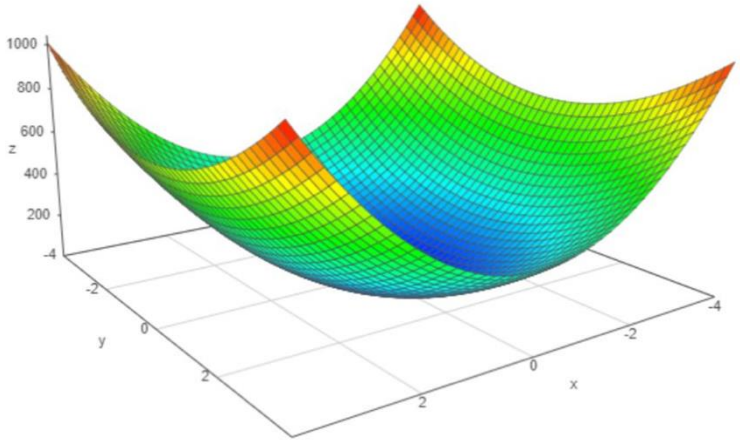


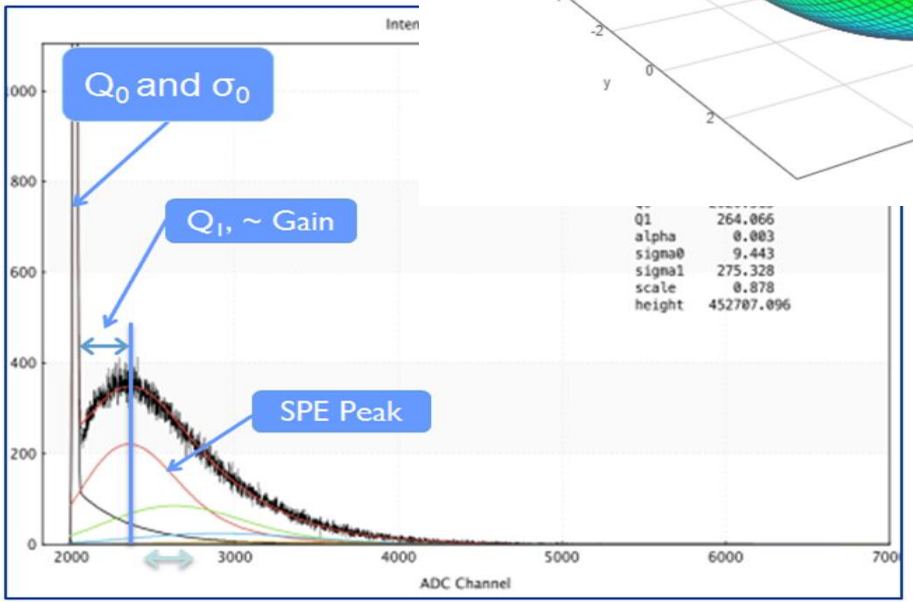


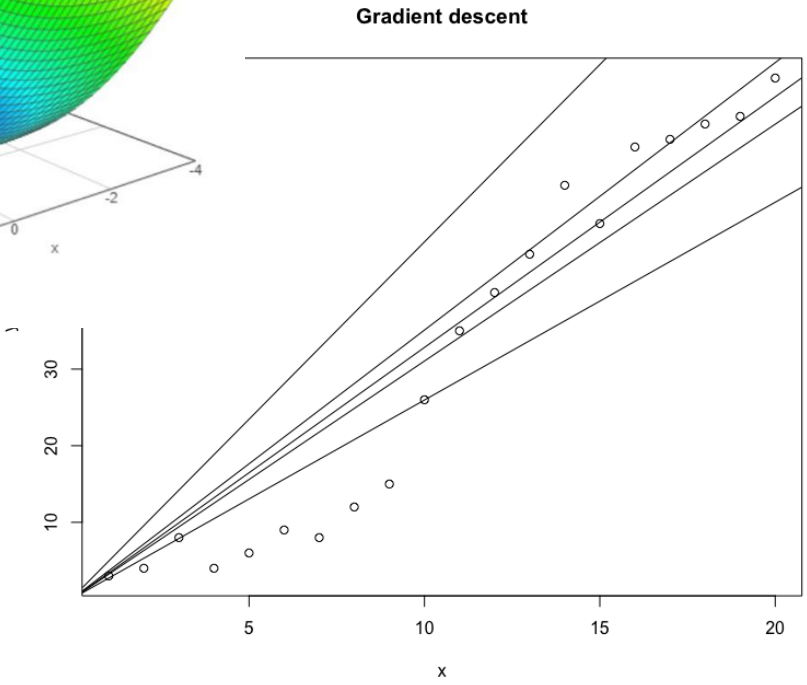
Outline of core ML problems



Machine Learning - Regression







Regression using gradient descent

- A process for modeling the relationship between variables of interest
- Gradient descent is a general approach in machine learning that could be used for solving linear regression

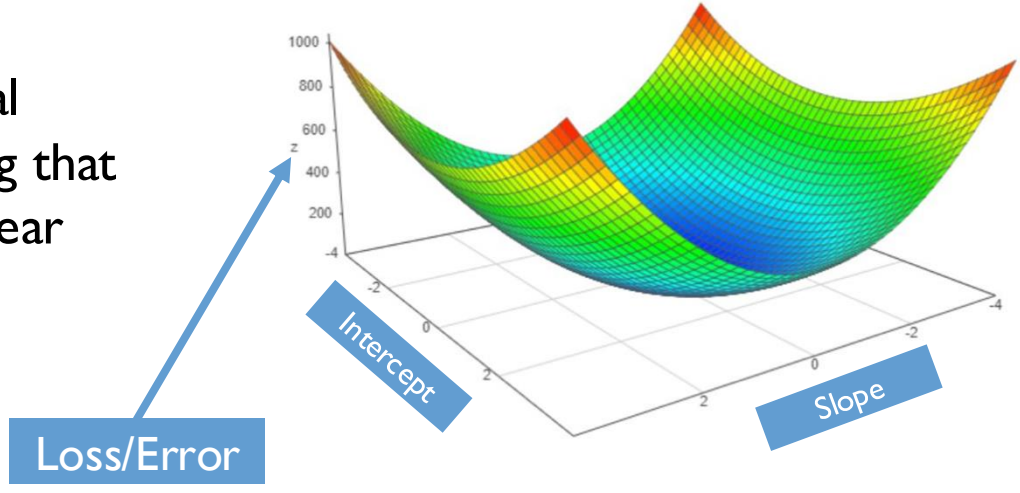
$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where:

$c =$ Degrees of freedom

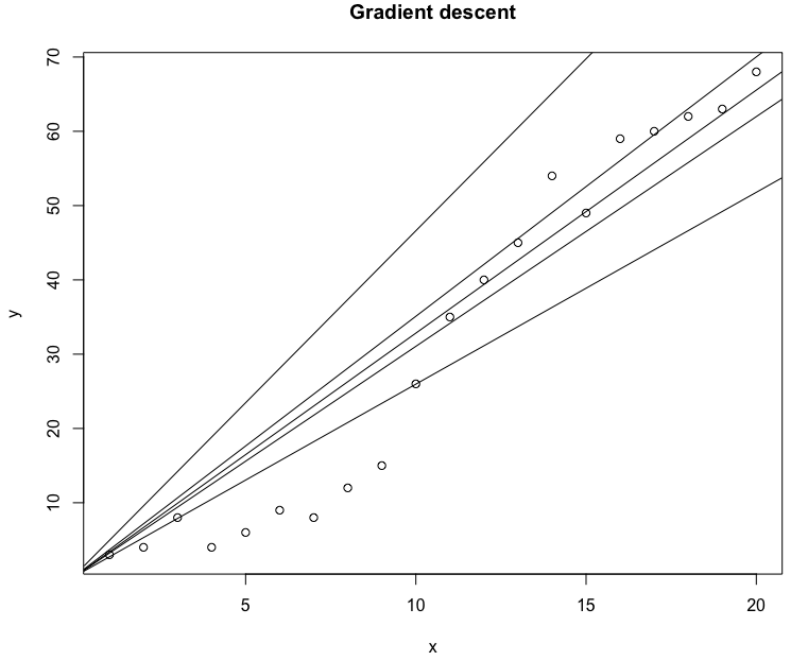
$O =$ Observed value(s)

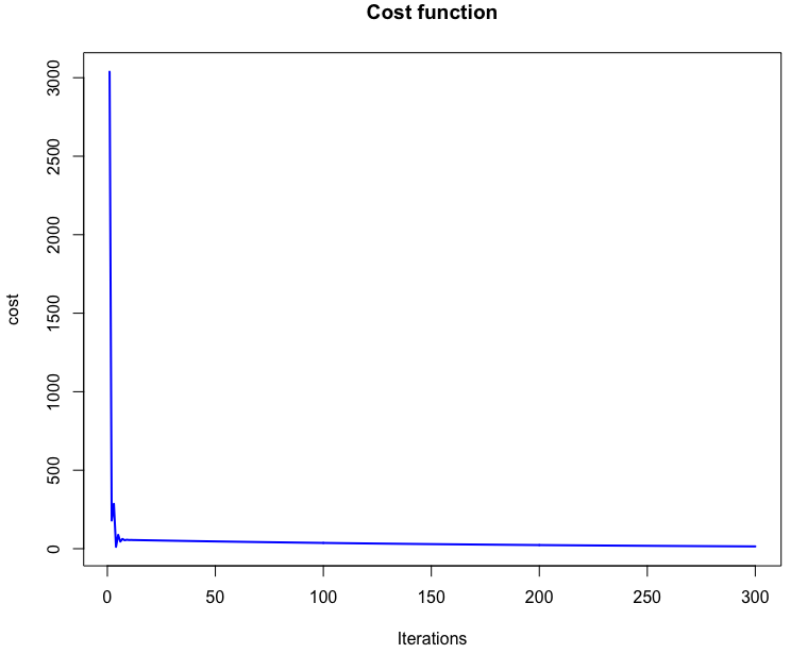
$E =$ Expected value(s)



Error surface for various lines created using linear regression (x represents slope, y represents intercept and z is the error value).

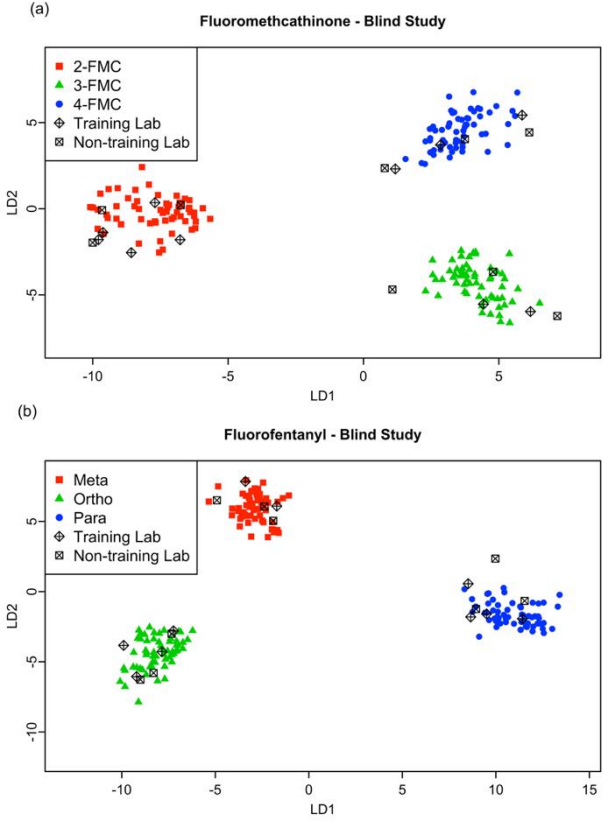
Linear regression using gradient descent





Machine Learning – Classification/Clustering

- Classification is a machine learning technique that is used to categorize data samples/events/items
- Clustering is a way to classify samples/events/items in an unsupervised way!



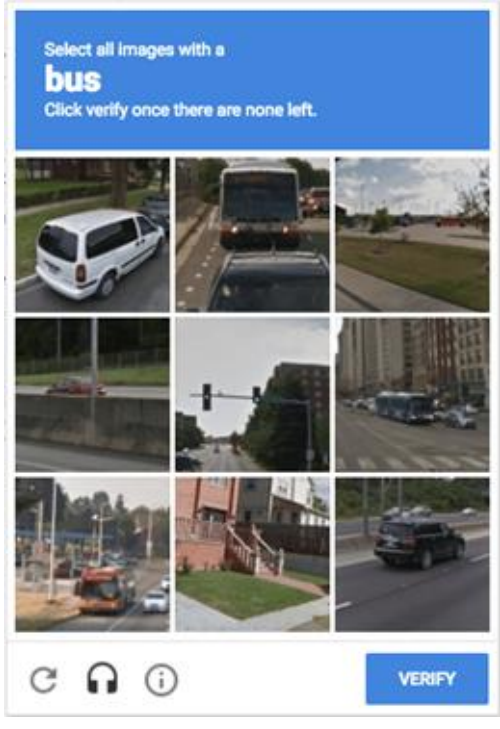
Neural Networks

- MNIST handwritten digit database
- (Modified National Institute of Standards and Technology dataset)
- 60,000 handwritten digits from the census bureau for training
- 10,000 test images
- Each image is 28x28 pixels and has a corresponding label
- Install tensorflow
  - pip install tensorflow

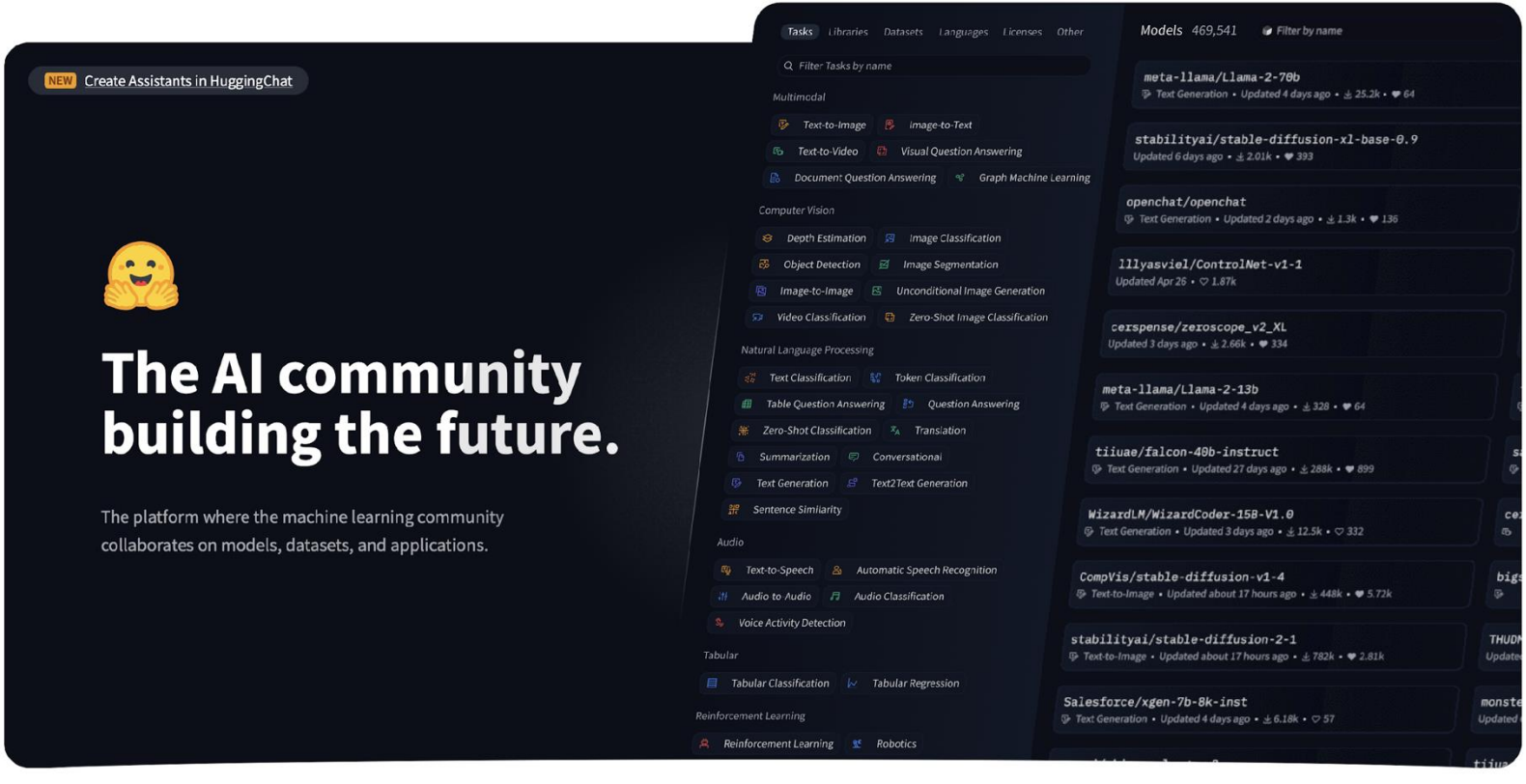
Autonomous Vehicles - Object Detection

You've been creating training datasets!

https://xkcd.com/1897/



Proliferation of Pretrained Models



YOLOv8

- YOLOv8 (You Only Look Once, version 8) is the latest iteration in the YOLO series of real-time object detection algorithms. It is designed to achieve high accuracy and speed in detecting multiple objects in images or video streams.

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
path = model.export(format="onnx")  # export the model to ONNX format
```

https://github.com/ultralytics/ultralytics

Computational advances

An Nvidia RTX 3090 is roughly equivalent to the world's fastest supercomputer in 2002

=

Machine Learning Summary

- **Machine learning**: field that explores the use of algorithms that can learn from the data and use that knowledge to make predictions on the data it has not seen before.
- **Supervised learning**: branch of machine learning that includes problems where a model could be built using the data and true labels or values.
- **Unsupervised learning**: branch of machine learning that includes problems where we do not have true labels for the data to train with. Instead, the goal is to somehow organize the data in some meaningful clusters or densities.
  - Future Topic