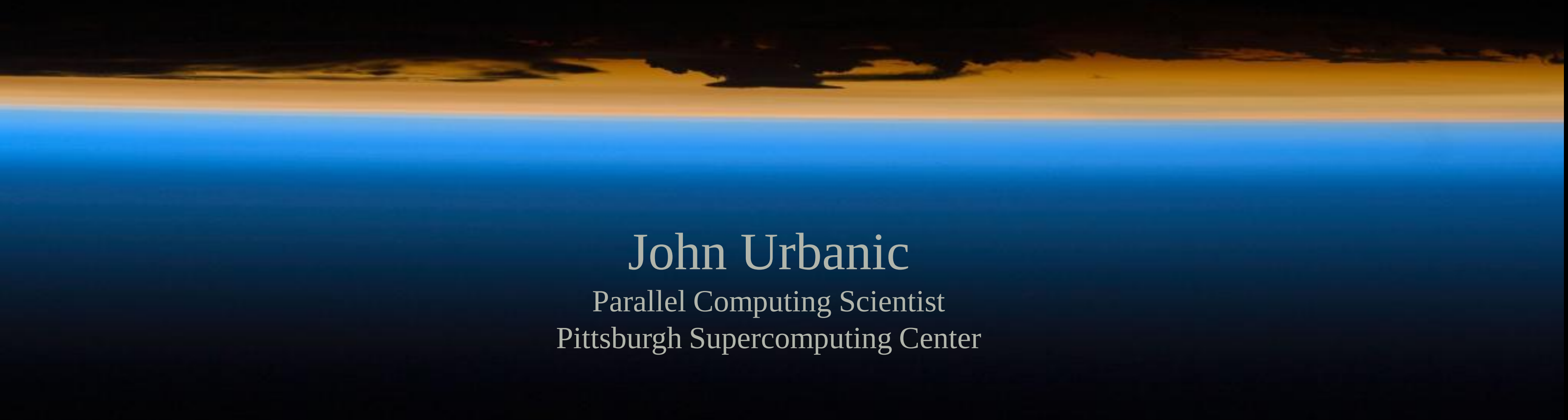
Clustering and Dimensionality Reduction



John Urbanic
Parallel Computing Scientist
Pittsburgh Supercomputing Center

Homework #1: Using Spark to mine astro signals

Q: Can you find the repeating cosmological signal (pulsar?) in the captured data?

In `~urbanic/LargeScaleComputing/Spark/pulsar.dat` on Bridges you will find the data with a series of signals stored as:

```
        ascension (degrees), declination (degrees), time (seconds), frequency (MHz)
```

These are radiofrequency signals captured by an array of instruments scanning a solid angle of the sky.

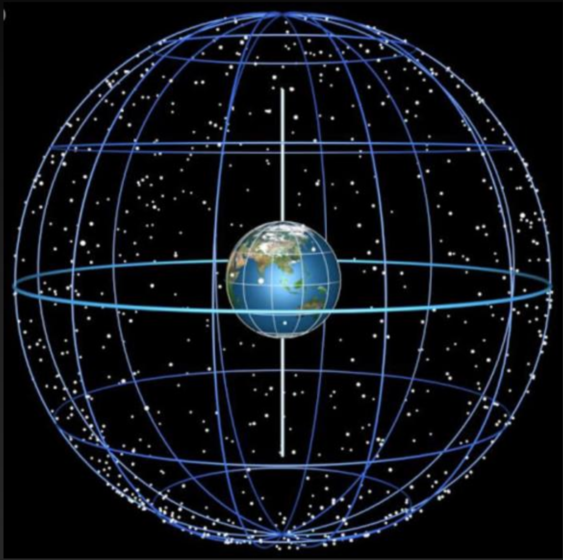
The data is, like almost all real data, a little noisy and has sources of errors. In our case the angular coordinates have 0.1 degrees error, the signal frequency has 0.1 MHZ error and the timebase/period error is <0.1s (that is one STD or standard deviation).

Your job is to find the most regular temporarily repeating RF source.

Your target will be found in the same location (within error) of the sky, on the same frequency (within error) chirping for the most blips, regularly spaced in time during that active period. So...

...........blip...blip...blip...blip...blip...blip..

Again, you are looking for the signal with the *most* blips.



Homework #1: This is very close to serious research.

The LIGO gravitational wave detector was able to confirm the collision of two neutron stars with both a gamma ray satellite and optical and other electromagnetic spectrum telescopes. For these transient events, it requires rapid real-time signal analysis to steer other instruments to the proper celestial coordinates. The 2 second gamma-ray burst was detected 1.7 seconds after the GW merger signal. 70 observatories were able to mine signatures in the following days. Even so, the refined location alert took a long time, and much improvement lies ahead.

This rapid processing requirement will only become more extreme as the Square Kilometer Array comes fully on-line. It will generate over an Exabyte of data a day. It will require extreme real-time processing to classify and compress this data down to an archivable size.

Strange, repeating radio signal near the center of the Milky Way has scientists stumped

This article (www.livescience.com/strange-radio-source-milky-way-center) is the summary of the paper (arxiv.org/pdf/2109.00652.pdf) that looks an awful lot like what we are doing.

Homework #1: Using Spark to mine astro signals

There are multiple approaches to this problem that will work. I can think of several that would be dead painful (hints in lecture).

This is doable interactively. You can explore the data within PySpark using operations that are reasonably quick. You do not need to create long running scripts to get to the answer.

To recap:

1. log on
2. cp the datafile to where you want to work with your pyspark session
3. get an interactive node
4. load the spark module
5. start a pyspark session
6. load the datafile
7. and use transforms to wring out the answer: the coordinates, the frequency and the period (rounded to integers is fine).

Use Spark commands (RDDs) to derive your answer. Yes, with a dataset this size you could use python directly, but a real dataset would be far too large. Use Spark transforms to boil down your data until you have a modest amount (screenful) of data to inspect. Use python at that point, if you wish, or just observe your answer. Ask if you don't understand this point. You could lose credit otherwise.

Submit your answer along with the exact sequence of spark commands that got you there. *This means a single email*. An attachment of the spark commands with your answer in the body of the mail, and any explanation you care to provide, is the preferred format.

This assignment is 20% of your grade. Due when we review solutions in 1 week. *Must be on time!*

Homework #1: Approach Hints

Don't overthink this problem, particularly the errors. While it is good to be diligent about these types of issues, it can also lead you into the weeds. Here are a few ways you might "reset" and approach the problem:

- One of the great things about having Spark at your disposal is that you can interactively explore large data. Because of the speed of the parallel system, you can ask questions in real time. With a dataset like this you can just start poking around and looking for large signals and interesting points. Then you can ask yourself how to find the "biggest" signal.

- But, if you are more comfortable starting algorithmically, it is often best to think of a tiny version of the problem that you have to solve on paper with only a pencil - no computer. Ask yourself how you would solve that. Then turn that into an algorithm.

- Or, start by ignoring the errors. Just find the answer as if the errors didn't matter. Then, since there are errors, what data might you be overlooking? And how might you find it?

These are three different approaches, but none of them should take more than a few steps to get to the answer. And none of them involve complex type conversion issues or routines from MLLIB, although you are welcome to use those too.

Homework #1: Hints

- Comments in your code are helpful to me - and to you too. If you make a mistake it is easier for me to give you credit if I know your intent. And of course comments are just good professional practice.

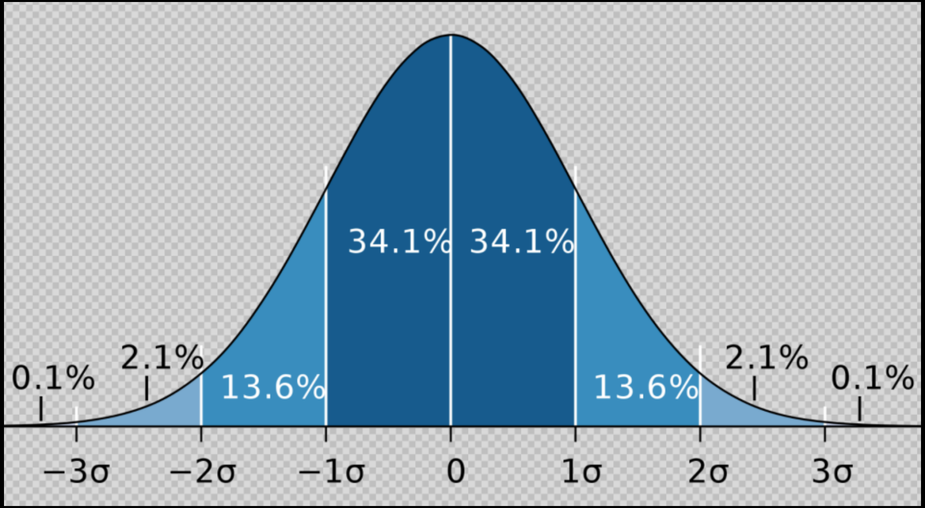
- We'll be exploring clustering just now. *Do not assume that that approach is necessary for Assignment #1*, although you could maybe use it. It will be essential for Assignment #2.

- If you see an occasional "Warning" from Spark (or really any of these machine learning packages) you may want to ignore it. That is terrible general programming advice, but a fact of life in this domain.

- I don't want anyone to struggle with Python accuracy issues, so you can treat the signal repeating time accuracy to be ~0.1s if you want. You can't ignore the precision issues, but you should be able to see by inspection the signal in rounded off data as you home in on the answer. I don't want you to have to squint and compare 5th decimal places.

A brief word about errors...

For purposes of the first assignment realize that errors specified with a "standard deviation" imply that not all the data falls exactly within the error. There is a "bell curve", and some of the data can be slightly outside the 1 deviation error specified. I assumed most of you are familiar with that concept, but not all of you. *I am never trying to trick you with the assignments.*



For a normal or gaussian distribution, the standard deviation indicates the region where about 2/3 of the points will fall. And for small sample sizes, this may be noisy.

Using MLlib

One of the reasons we use spark is for easy access to powerful data analysis tools. The MLlib library gives us a machine learning library that is easy to use and utilizes the scalability of the Spark system.

It has supported APIs for Python (with NumPy), R, Java and Scala.

We will use the Python version in a generic manner that looks very similar to any of the above implementations.

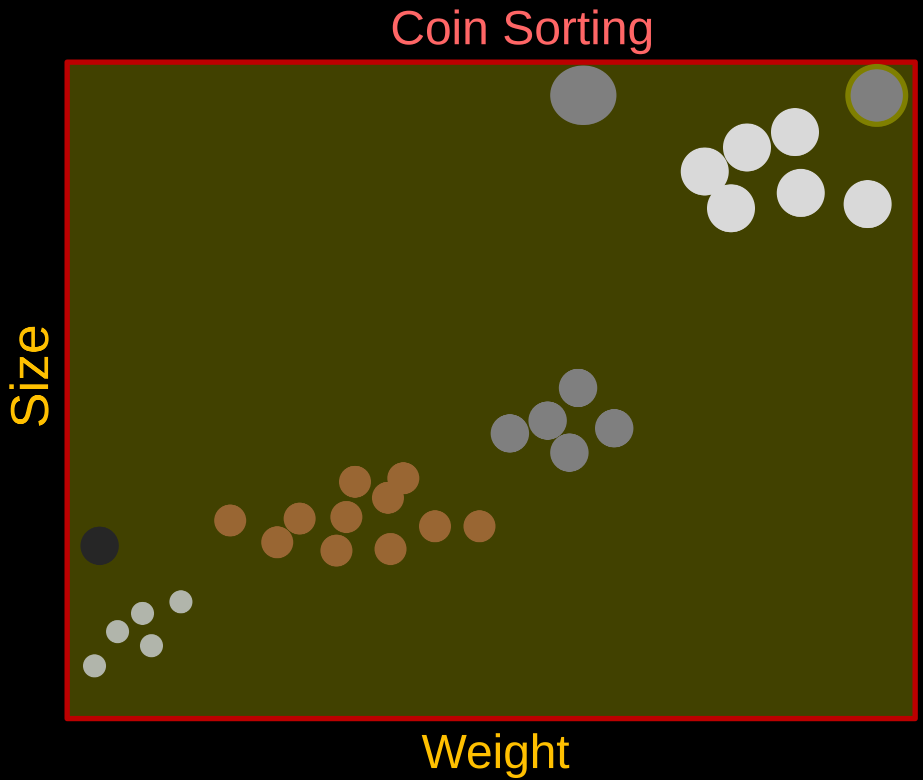
There are good example documents for the clustering routine we are using, as well as alternative clustering algorithms, here:

http://spark.apache.org/docs/latest/mllib-clustering.html

I suggest you use these pages for your Spark work.

Clustering

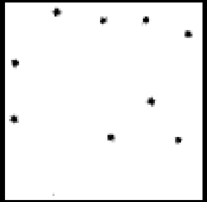
Clustering is a very common operation for finding grouping in data and has countless applications. This is a very simple example, but you will find yourself reaching for a clustering algorithm frequently in pursuing many diverse machine learning objectives, sometimes as one part of a pipeline.



Clustering

As intuitive as clustering is, it presents challenges to implement in an efficient and robust manner.

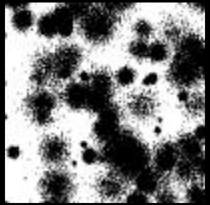
You might thi ensional spaces.



But it can get

From 1900 until 1956 humans were considered to have 48 chromosomes, instead of 46, based upon the interpretation of this camera lucida image.

Sometimes y tart with. Often you don't.
How hard can ere?



We will start with 5000 2D points. We want to figure out how many clusters there are, and their centers. Let's fire up pyspark and get to it…

Finding Clusters

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
>>> rdd1 = sc.textFil
>>>
>>> rdd2 = rdd1.map(l
>>> rdd3 = rdd2.map(l
>>>
```

to RDD

rm to words and integers

```
br06% interact
...
r288%
r288% module load spark
r288% pyspark
```

Finding Our Way

```
>>> rdd1 = sc.textFile("5000_points.txt")
>>> rdd1.count()
5000
>>> rdd1.take(4)
['    664159    550946', '    665845    557965', '    597173    575538', '    618600    551446']
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd2.take(4)
[['664159', '550946'], ['665845', '557965'], ['597173', '575538'], ['618600', '551446']]
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>> rdd3.take(4)
[[664159, 550946], [665845, 557965], [597173, 575538], [618600, 551446]]
>>>
```

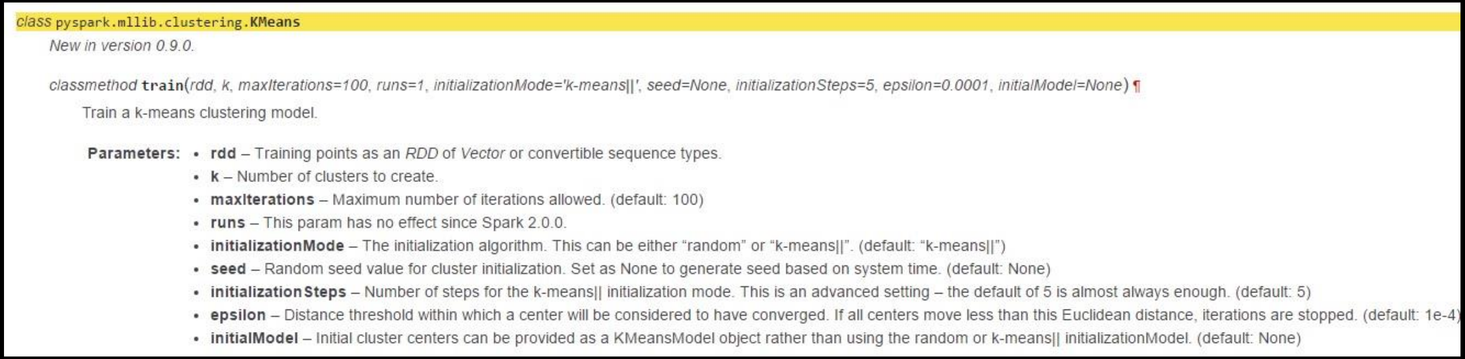
Finding Clusters

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
>>> rdd1 = sc.textFile("5000_points.txt")
>>>
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>>
>>>
>>> from pyspark.mllib.clustering import KMeans
```

Read into RDD

Transform

Import Kmeans



class `pyspark.mllib.clustering.`**`KMeans`**

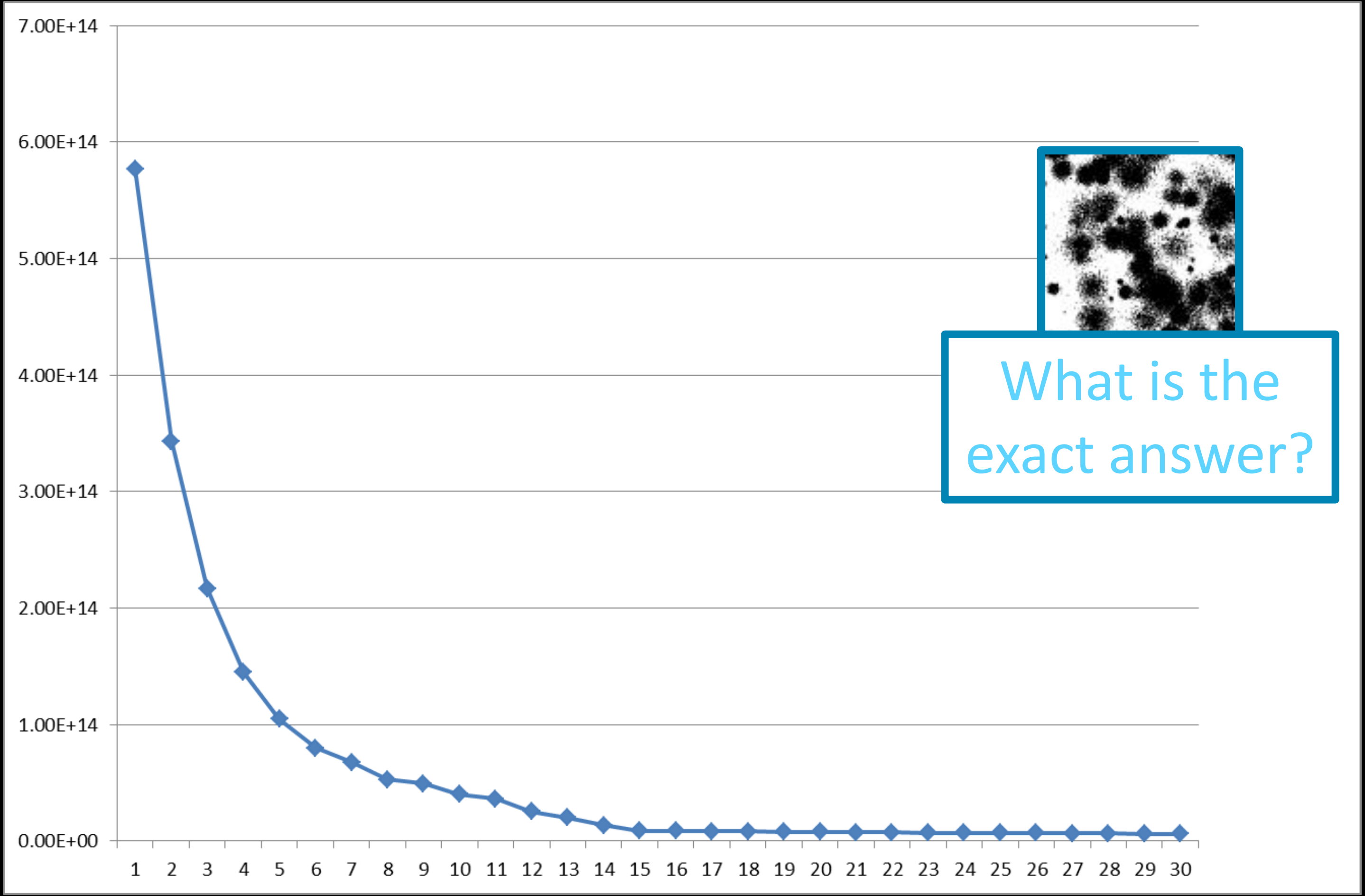
New in version 0.9.0.

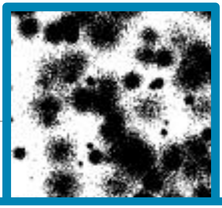
classmethod **`train`**(*rdd, k, maxIterations=100, runs=1, initializationMode='k-means||', seed=None, initializationSteps=5, epsilon=0.0001, initialModel=None*) ¶

Train a k-means clustering model.

Parameters:
- **rdd** – Training points as an *RDD* of *Vector* or convertible sequence types.
- **k** – Number of clusters to create.
- **maxIterations** – Maximum number of iterations allowed. (default: 100)
- **runs** – This param has no effect since Spark 2.0.0.
- **initializationMode** – The initialization algorithm. This can be either “random” or “k-means||”. (default: “k-means||”)
- **seed** – Random seed value for cluster initialization. Set as None to generate seed based on system time. (default: None)
- **initializationSteps** – Number of steps for the k-means|| initialization mode. This is an advanced setting – the default of 5 is almost always enough. (default: 5)
- **epsilon** – Distance threshold within which a center will be considered to have converged. If all centers move less than this Euclidean distance, iterations are stopped. (default: 1e-4)
- **initialModel** – Initial cluster centers can be provided as a KMeansModel object rather than using the random or k-means|| initializationModel. (default: None)

Finding Clusters





What is the exact answer?

Finding Clusters

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Nov 20 2015 02:00:19)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
>>> rdd1 = sc.textFile("5000_points.txt")
>>>
>>> rdd2 = rdd1.map(lambda x:x.split())
>>> rdd3 = rdd2.map(lambda x: [int(x[0]),int(x[1])])
>>>
>>> from pyspark.mllib.clustering import KMeans
>>>
>>> for clusters in range(1,30):
...     model = KMeans.train(rdd3, clusters)
...     print (clusters, model.computeCost(rdd3))
...
```

Let's see results for 1-30 cluster tries

```
1 5.76807041184e+14
2 3.43183673951e+14
3 2.23097486536e+14
4 1.64792608443e+14
5 1.19410028576e+14
6 7.97690150116e+13
7 7.16451594344e+13
8 4.81469246295e+13
9 4.23762700793e+13
10 3.65230706654e+13
11 3.16991867996e+13
12 2.94369408304e+13
13 2.04031903147e+13
14 1.37018893034e+13
15 8.91761561687e+12
16 1.31833652006e+13
17 1.39010717893e+13
18 8.22806178508e+12
19 8.22513516563e+12
20 7.79359299283e+12
21 7.79615059172e+12
22 7.70001662709e+12
23 7.24231610447e+12
24 7.21990743993e+12
25 7.09395133944e+12
26 6.92577789424e+12
27 6.53939015776e+12
28 6.57782690833e+12
29 6.37192522244e+12
```

Right Answer?

```
>>> for trials in range(10):
...     print
...     for clusters in range(12,18):
...         model = KMeans.train(rdd3,clusters)
...         print (clusters, model.computeCost(rdd3))
```

```
12 2.45472346524e+13
13 2.00175423869e+13
14 1.90313863726e+13
15 1.52746006962e+13
16 8.67526114029e+12
17 8.49571894386e+12

12 2.62619056924e+13
13 2.90031673822e+13
14 1.52308079405e+13
15 8.91765957989e+12
16 8.70736515113e+12
17 8.49616440477e+12

12 2.5524719797e+13
13 2.14332949698e+13
14 2.11070395905e+13
15 1.47792736325e+13
16 1.85736955725e+13
17 8.42795740134e+12

12 2.31466242693e+13
13 2.10129797745e+13
14 1.45400177021e+13
15 1.52115329071e+13
16 1.41347332901e+13
17 1.31314086577e+13

12 2.47927778784e+13
13 2.43404436887e+13
14 2.1522702068e+13
15 8.91765000665e+12
16 1.4580927737e+13
17 8.57823507015e+12
```

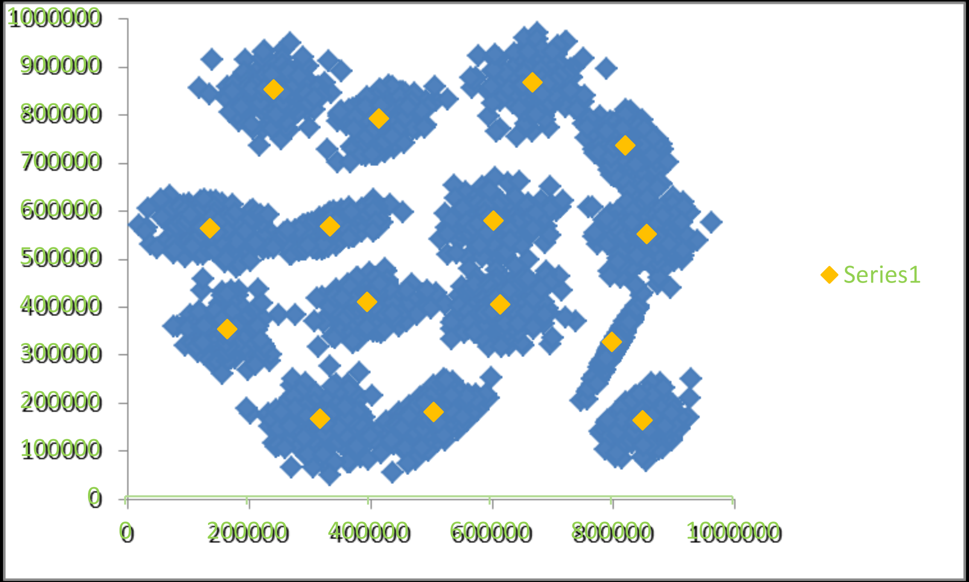
```
12 2.31466520037e+13
13 1.91856542103e+13
14 1.49332023312e+13
15 1.3506302755e+13
16 8.7757678836e+12
17 1.60075548613e+13

12 2.5187054064e+13
13 1.83498739266e+13
14 1.96076943156e+13
15 1.41725666214e+13
16 1.41986217172e+13
17 8.46755159547e+12

12 2.38234539188e+13
13 1.85101922046e+13
14 1.91732620477e+13
15 8.91769396968e+12
16 8.64876051004e+12
17 8.54677681587e+12

12 2.5187054064e+13
13 2.04031903147e+13
14 1.95213876047e+13
15 1.93000628589e+13
16 2.07670831868e+13
17 8.47797102908e+12

12 2.39830397362e+13
13 2.00248378195e+13
14 1.34867337672e+13
15 2.09299321238e+13
16 1.32266735736e+13
17 8.50857884943e+12
```

Find the Centers

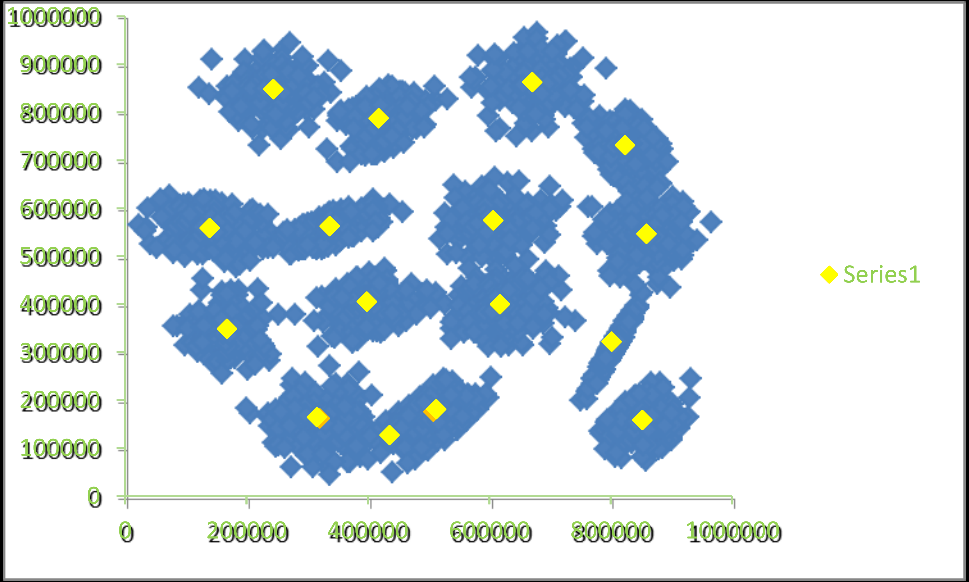
```
>>> for trials in range(10):                          #Try ten times to find best result
...    for clusters in range(12, 16):                 #Only look in interesting range
...        model = KMeans.train(rdd3, clusters)
...        cost = model.computeCost(rdd3)
...        centers = model.clusterCenters             #Let's grab cluster centers
...        if cost<1e+13:                             #If result is good, print it out
...           print (clusters, cost)
...           for coords in centers:
...             print (int(coords[0]), int(coords[1]))
...           break
...
```

```
15 8.91761561687e+12
852058 157685
606574 574455
320602 161521
139395 558143
858947 546259
337264 562123
244654 847642
398870 404924
670929 862765
823421 731145
507818 175610
801616 321123
617926 399415
417799 787001
167856 347812
15 8.91765957989e+12
670929 862765
139395 558143
244654 847642
852058 157685
617601 399504
801616 321123
507818 175610
337264 562123
858947 546259
823421 731145
606574 574455
167856 347812
398555 404855
417799 787001
320602 161521
```

Fit?


1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1000000
Series1

16 Clusters



1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1000000
Series1

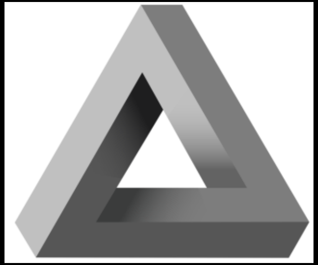
Dimensionality Reduction

We are going to find a recurring theme throughout machine learning:

- Our data naturally resides in higher dimensions
- Reducing the dimensionality makes the problem more tractable
- And simultaneously provides us with insight

This last two bullets highlight the principle that "learning" is often finding an effective compressed representation.

As we return to this theme, we will highlight these slides with our Dimensionality Reduction badge so that you can follow this thread and appreciate how fundamental it is.



Why all these dimensions?

The problems we are going to address, as well as the ones you are likely to encounter, are naturally highly dimensional. If you are new to this concept, lets look at an intuitive example to make it less abstract.

Category	Purchase Total ($)
Children's Clothing	$800
Pet Supplies	$0
Cameras (Dash, Security, Baby)	$450
Containers (Storage)	$350
Romance Book	$0
Remodeling Books	$80
Sporting Goods	$25
Children's Toys	$378
Power Tools	$0
Computers	$0
Garden	$0
Children's Books	$180

< 2900 Categories >

This is a 2900 dimensional vector.

Why all these dimensions?

If we apply our newfound clustering expertise, we might find we have 80 clusters (with an acceptable error).

People spending on “child’s toys “ and “children’s clothing” might cluster with “child’s books” and, less obvious, "cameras (Dashcams, baby monitors and security cams)", because they buy new cars and are safety conscious. We might label this cluster "Young Parents". We also might not feel obligated to label the clusters at all. We can now represent any customer by their distance from these 80 clusters.

80 dimensional vector.

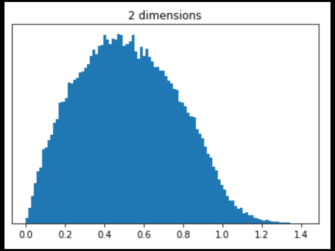
Customer Representation									
Cluster	Young Parents	College Athlete	Auto Enthusiast	Knitter	Steelers Fan	Shakespeare Reader	Sci-Fi Fan	Plumber	...
Distance	0.02	2.3	1.4	8.4	2.2	14.9	3.3	0.8	...

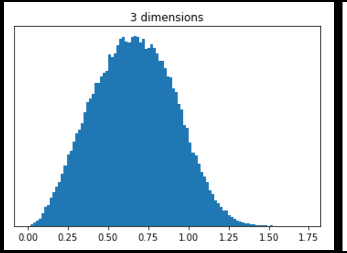
We have now accomplished two things:

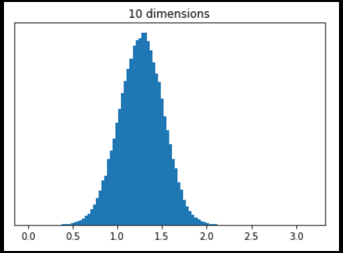
- we have compressed our data
- learned something about our customers (who to send a dashcam promo to).

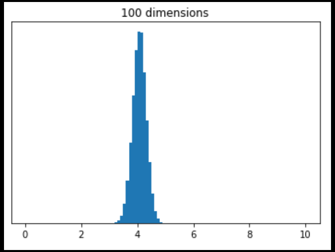
Curse of Dimensionality

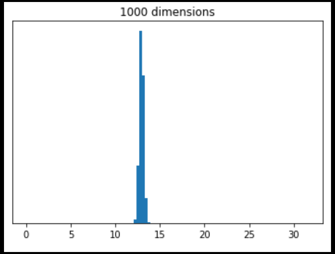
This is a good time to point out how our intuition can lead us astray as we increase the dimensionality of our problems - which we will certainly be doing - and to a great degree. There are several related aspects to this phenomenon, often referred to as the *Curse of Dimensionality*. One root cause of confusion is that our notion of Euclidian distance starts to fail in higher dimensions.

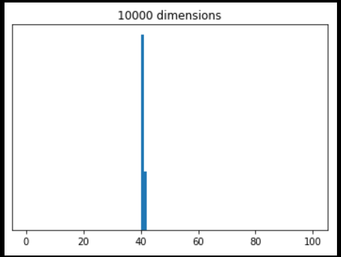












These plots show the distributions of pairwise distances between randomly distributed points within differently dimensioned unit hypercubes. Notice how all the points start to be about the same distance apart.

Once can imagine this makes life harder on a clustering algorithm!

There are other surprising effects: random vectors are almost all orthogonal; the unit sphere takes almost no volume in the unit square. These cause all kinds of problems when generalizing algorithms from our lowly 3D world.

Metrics

Even the definition of distance (the *metric*) can vary based upon application. If you are solving chess problems, you might find the Manhattan distance (or taxicab metric) to be most useful.

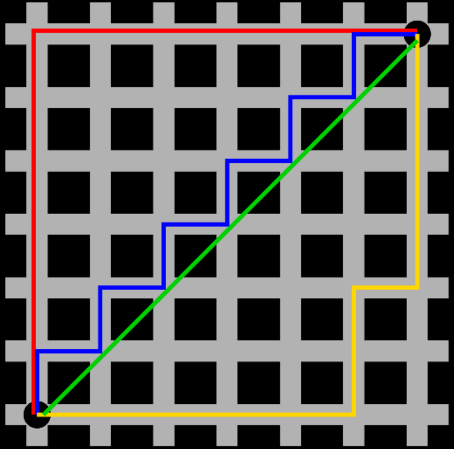


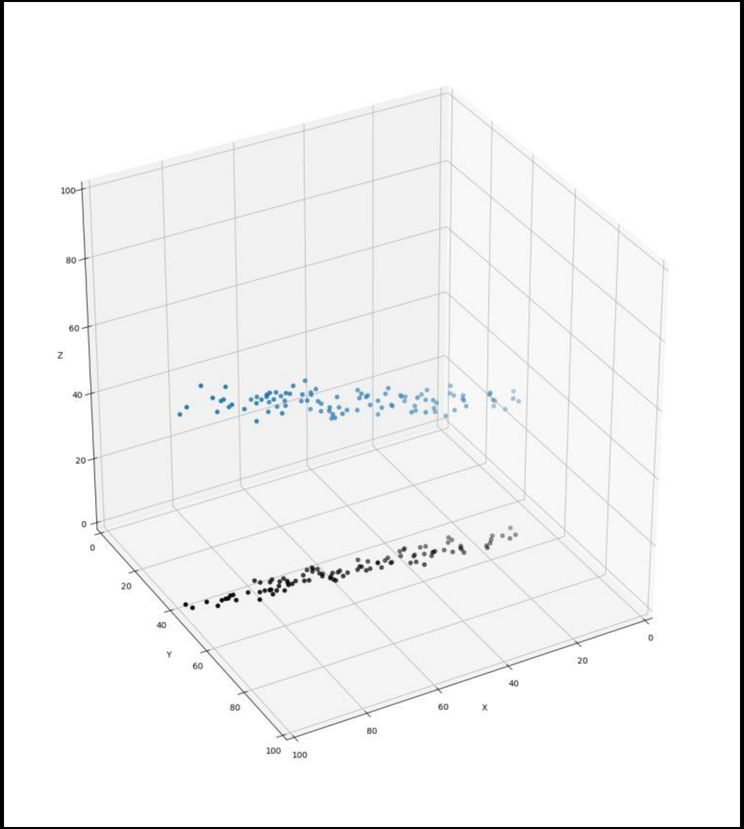
Image Source: Wikipedia

For comparing text strings, we might choose one of dozens of different metrics. For spell checking you might want one that is good for phonetic distance, or maybe edit distance. For natural language processing (NLP), you probably care more about tokens.

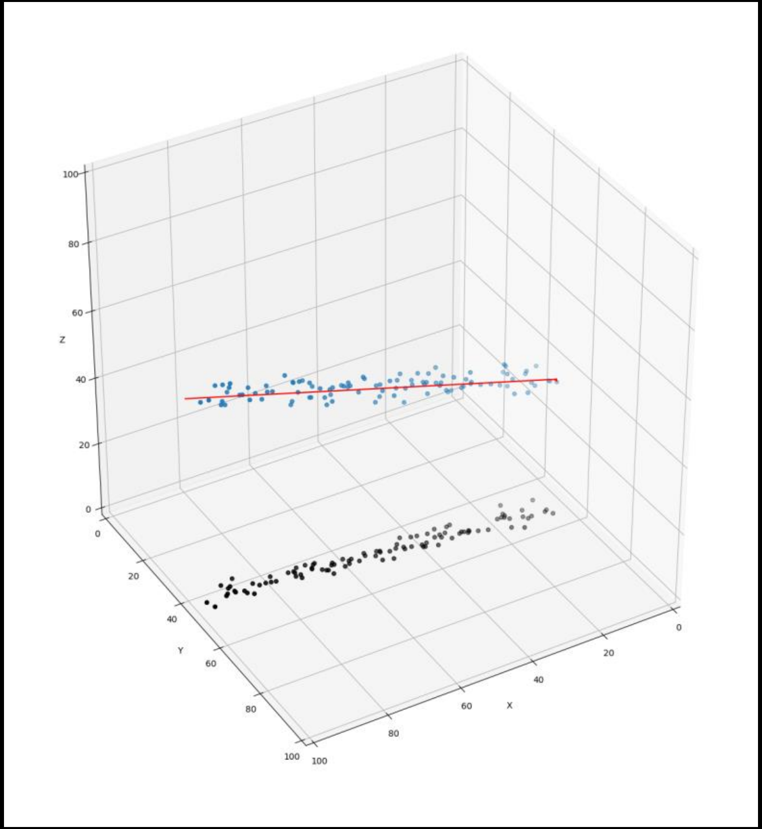
For genomics, you might care more about string sequences.

Some useful measures don't even qualify as metrics (usually because they fail the triangle inequality: $a + b \geq c$).

Alternative DR: Principal Component Analysis

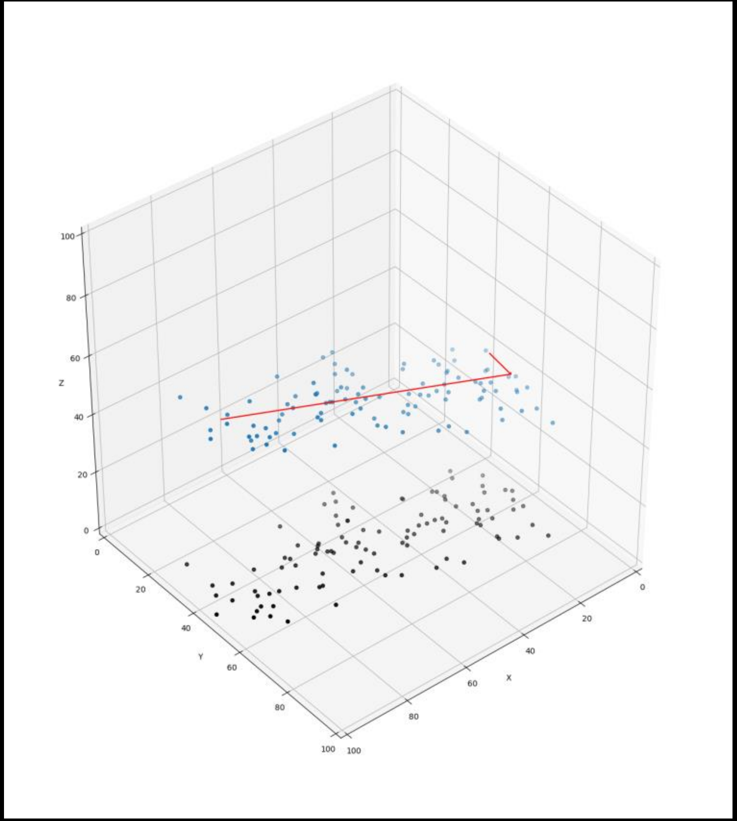


3D Data Set

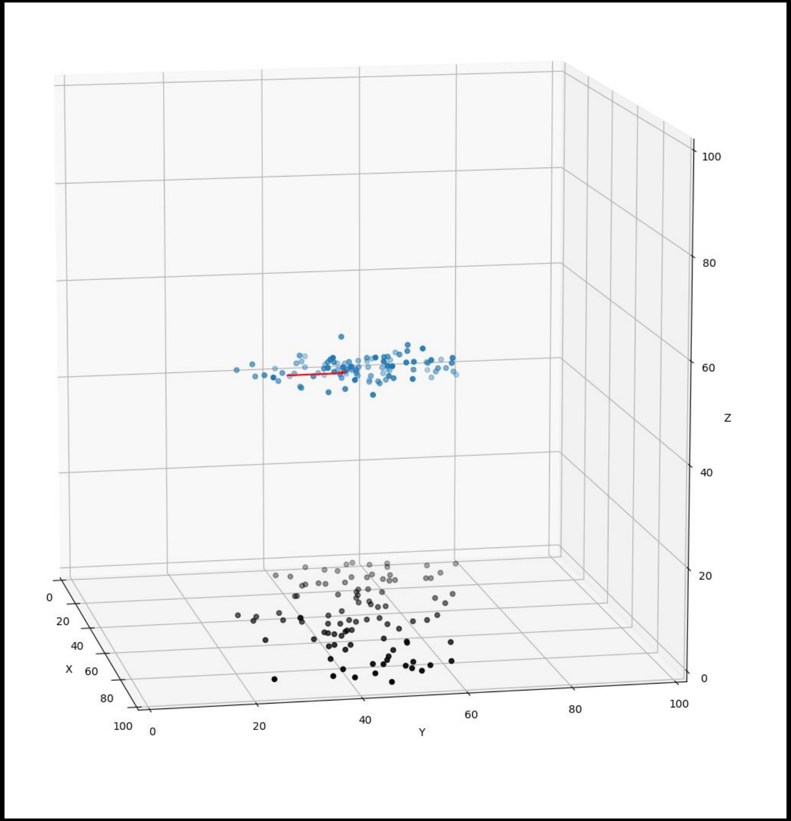


Maybe mostly 1D!

Alternative DR: Principal Component Analysis



Flatter 2D-ish Data Set



View down the 1st Princ. Comp.

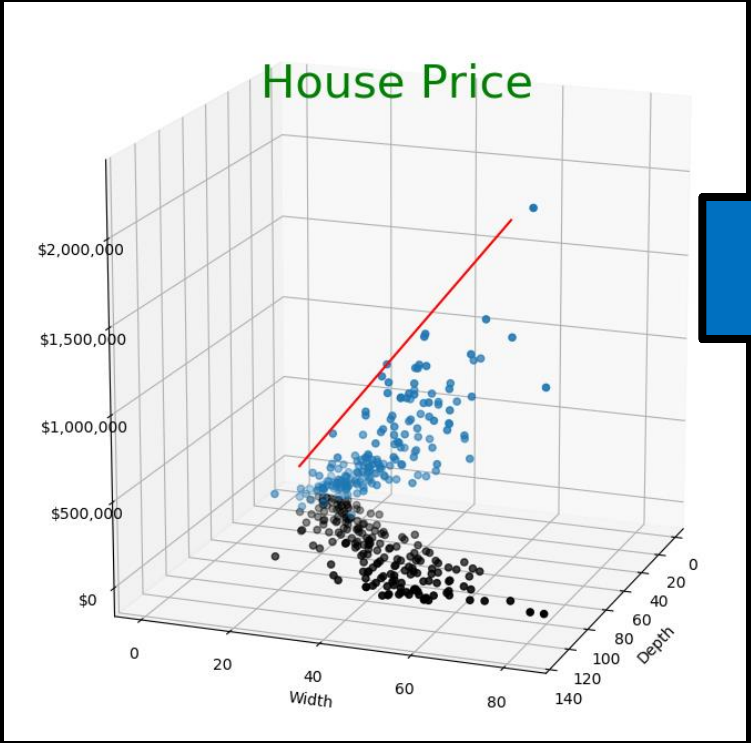
Why So Many Alternatives?

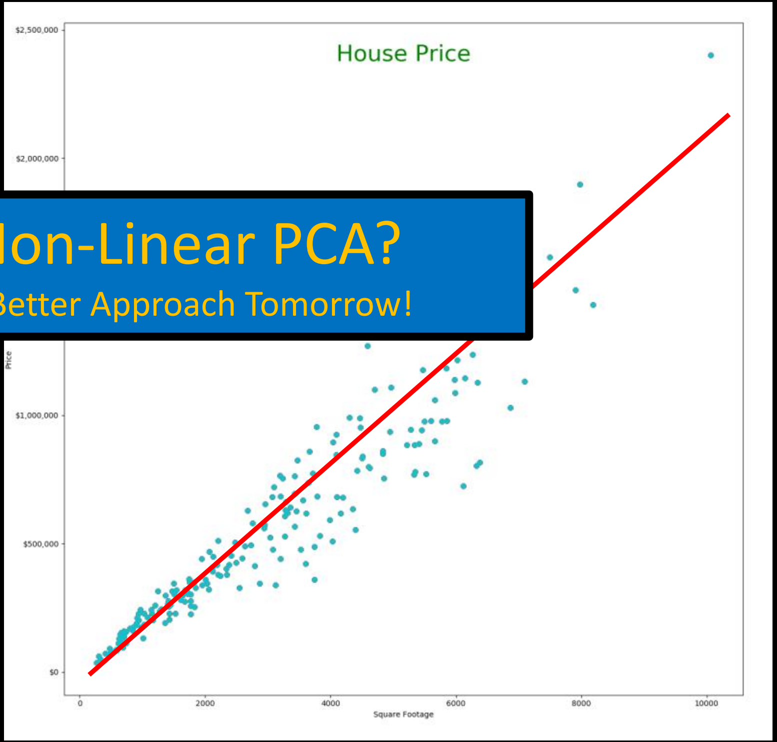
Let's look at one more example today. Suppose we are tying to do a Zillow type of analysis and predict home values based upon available factors. We may have an entry (vector) for each home that captures this kind of data:

Home Data	
Latitude	4833438 north
Longitude	630084 east
Last Sale Price	$ 480,000
Last Sale Year	1998
Width	62
Depth	40
Floors	3
Bedrooms	3
Bathrooms	2
Garage	2
Yard Width	84
Yard Depth	60
...	...

There may be some opportunities to reduce the dimension of the vector here. Perhaps clustering on the geographical coordinates...

Principal Component Analysis Fail





Non-Linear PCA?

A Better Approach Tomorrow!

1st Component Off

Data Not Very Linear

D x W Is Not Linear

But (DxW) Fits Well

Why the fascination with linear techniques?

The Streetlight Effect

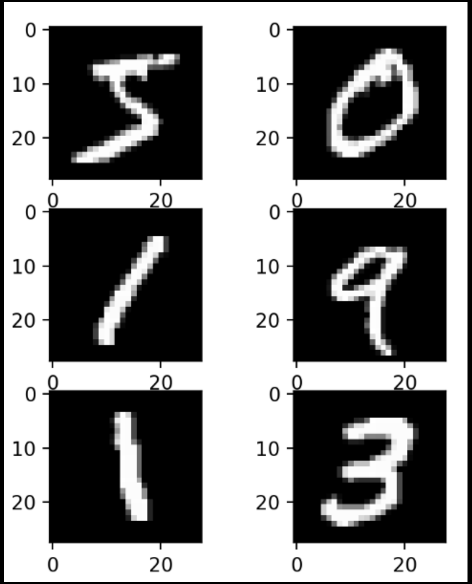
This is a very real and powerful force throughout the sciences.

It is not because practitioners are dumb.

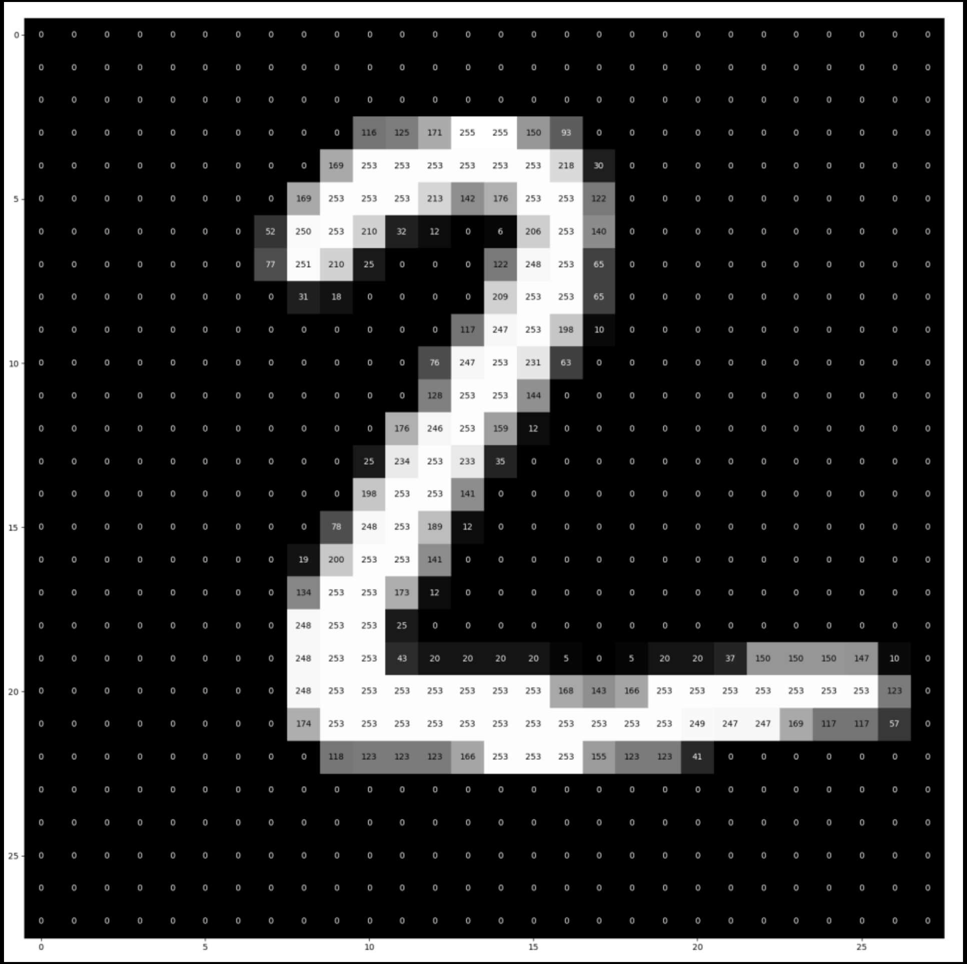
But, it is also very often neither explained nor justified.

Which leads to great confusion.

Why Would An Image Have 784 Dimensions?

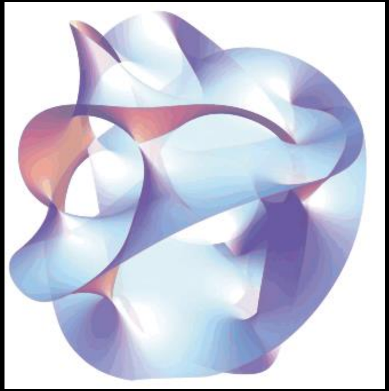


MNIST 28x28 greyscale images

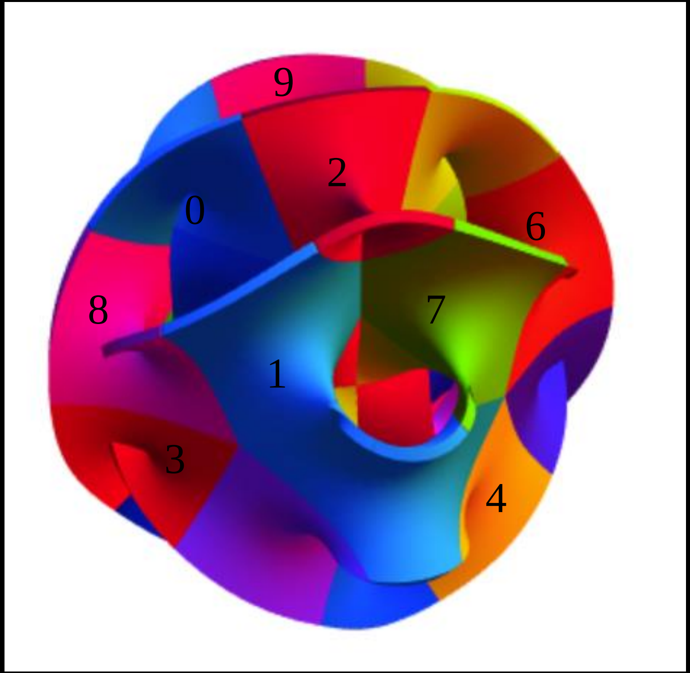


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	116	125	171	255	255	150	93	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	253	253	253	253	253	218	30	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	169	253	253	253	213	142	176	253	253	122	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	52	250	253	210	32	12	0	6	206	253	140	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	77	251	210	25	0	0	0	122	248	253	65	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	31	18	0	0	0	0	209	253	253	65	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	117	247	253	198	10	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	76	247	253	231	63	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	128	253	253	144	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	176	246	253	159	12	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	25	234	253	233	35	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	198	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	78	248	253	189	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	19	200	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	134	253	253	173	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	248	253	253	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	248	253	253	43	20	20	20	20	5	0	5	20	20	37	150	150	150	147	10	0
0	0	0	0	0	0	0	0	248	253	253	253	253	253	253	253	168	143	166	253	253	253	253	253	253	253	123	0
0	0	0	0	0	0	0	0	174	253	253	253	253	253	253	253	253	253	253	253	249	247	247	169	117	117	57	0
0	0	0	0	0	0	0	0	0	118	123	123	123	166	253	253	253	155	123	123	41	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

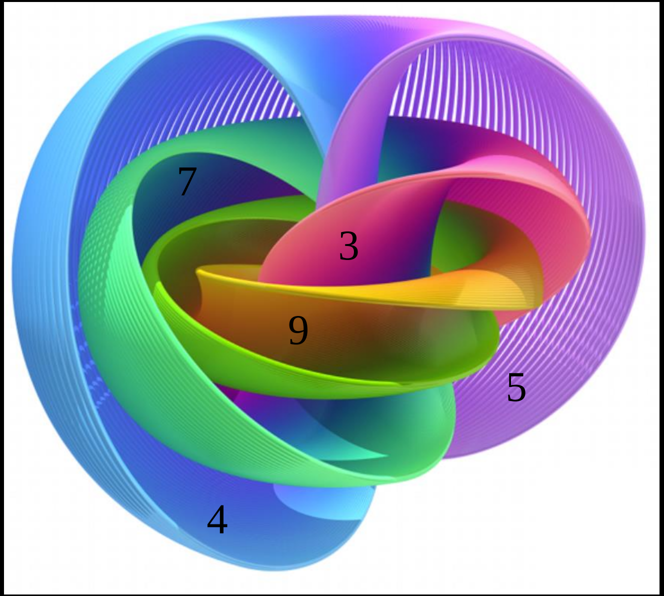
Central Hypothesis of Modern DL



Data Lives On
A Lower Dimensional
Manifold



Maybe Very Contiguous

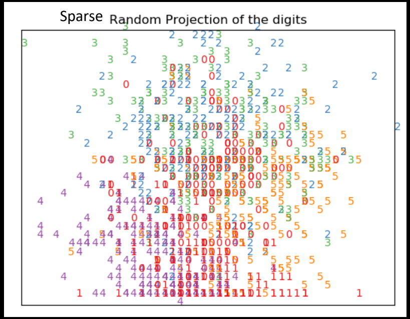


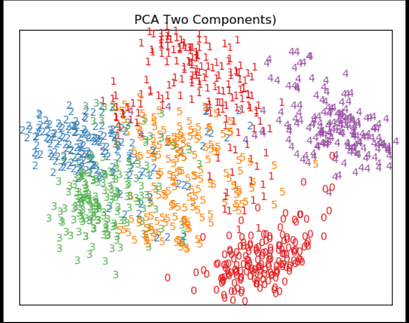
Maybe Less So

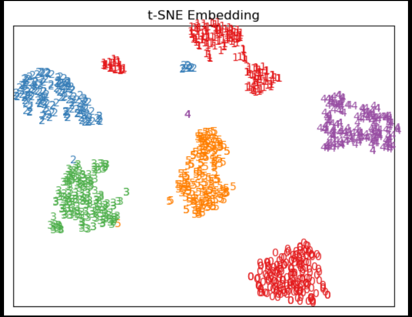
Images from Wikipedia

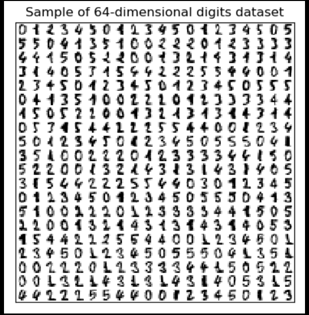
Testing These Ideas With Scikit-learn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, decomposition, manifold, random_projection)

def draw(X, title):
    plt.figure()
    plt.xlim(X.min(0)[0],X.max(0)[0]); plt.ylim(X.min(0)[1],X.max(0)[1])
    plt.xticks([]); plt.yticks([])
    plt.title(title)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(y[i]), color=plt.cm.Set1(y[i] / 10.) )

digits = datasets.load_digits(n_class=6)
X = digits.data
y = digits.target

rp = random_projection.SparseRandomProjection(n_components=2, random_state=42)
X_projected = rp.fit_transform(X)
draw(X_projected, "Sparse Random Projection of the digits")

X_pca = decomposition.PCA(n_components=2).fit_transform(X)
draw(X_pca, "PCA (Two Components)")

tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
X_tsne = tsne.fit_transform(X)
draw(X_tsne, "t-SNE Embedding")

plt.show()
```

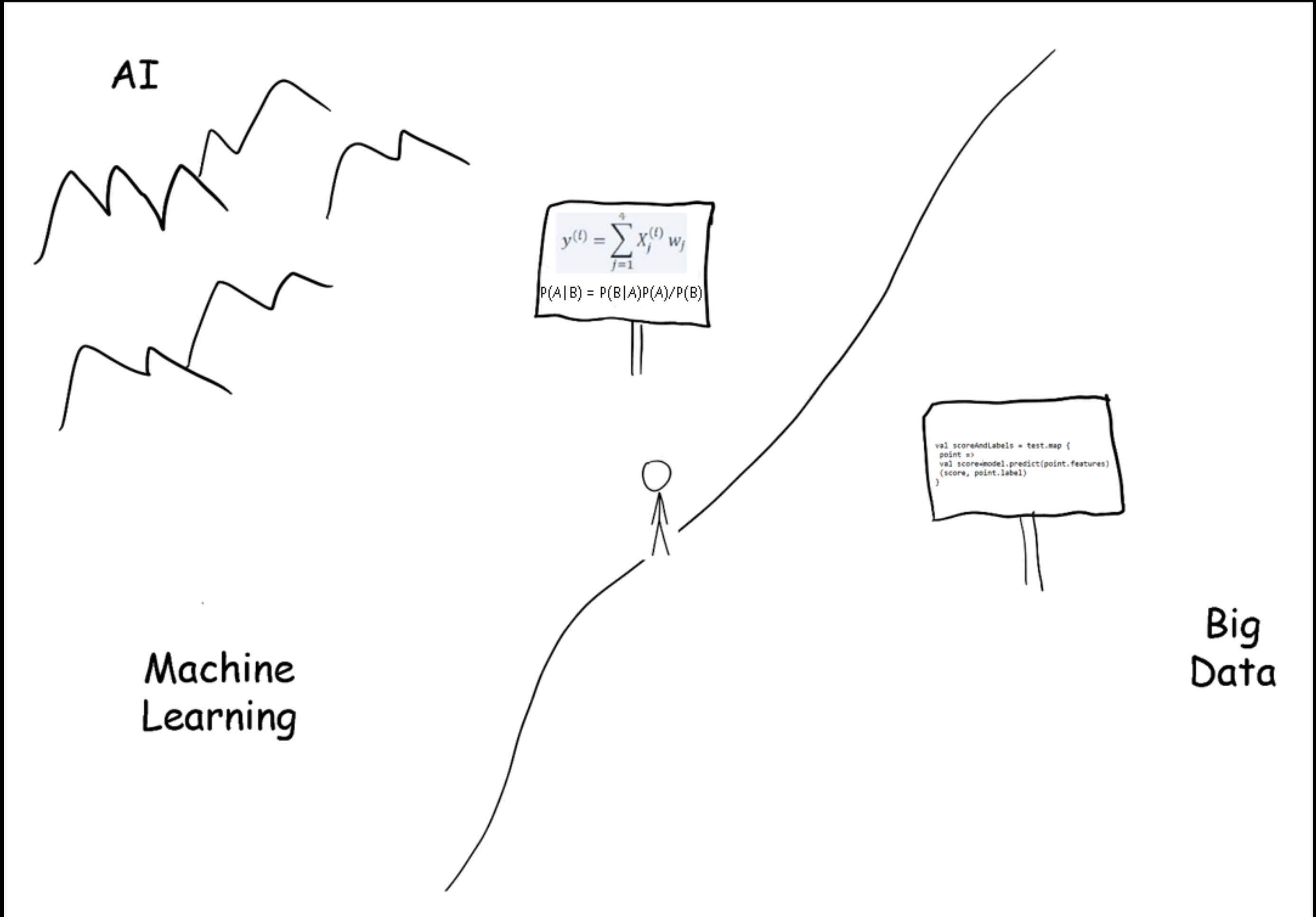








The Journey Ahead



$$y^{(t)} = \sum_{j=1}^{4} X_j^{(t)} w_j$$

P(A|B) = P(B|A)P(A)/P(B)

```
val scoreAndLabels = test.map {
 point =>
 val score=model.predict(point.features)
 (score, point.label)
}
```

As the Data Scientist wanders across the ill-defined boundary between Data Science and Machine Learning, in search of the fabled land of Artificial Intelligence, they find that the language changes from programming to a creole of linear algebra and probablity and statistics.

Homework #2: Submission and Grading

It is possible to extract all of the meaningful structure using the tools you have. Perhaps some of you will. We will certainly grade on a kind curve, and be flexible about what you report.

- You should submit a summary of what you have found. It could be a short paragraph, or a brief table, maybe even some matplotlib. Perhaps the answer is "There was one 2D square, centered at 20,20,20,20,20,20 with edge length 5" and that is all you have to say. Or maybe you found:

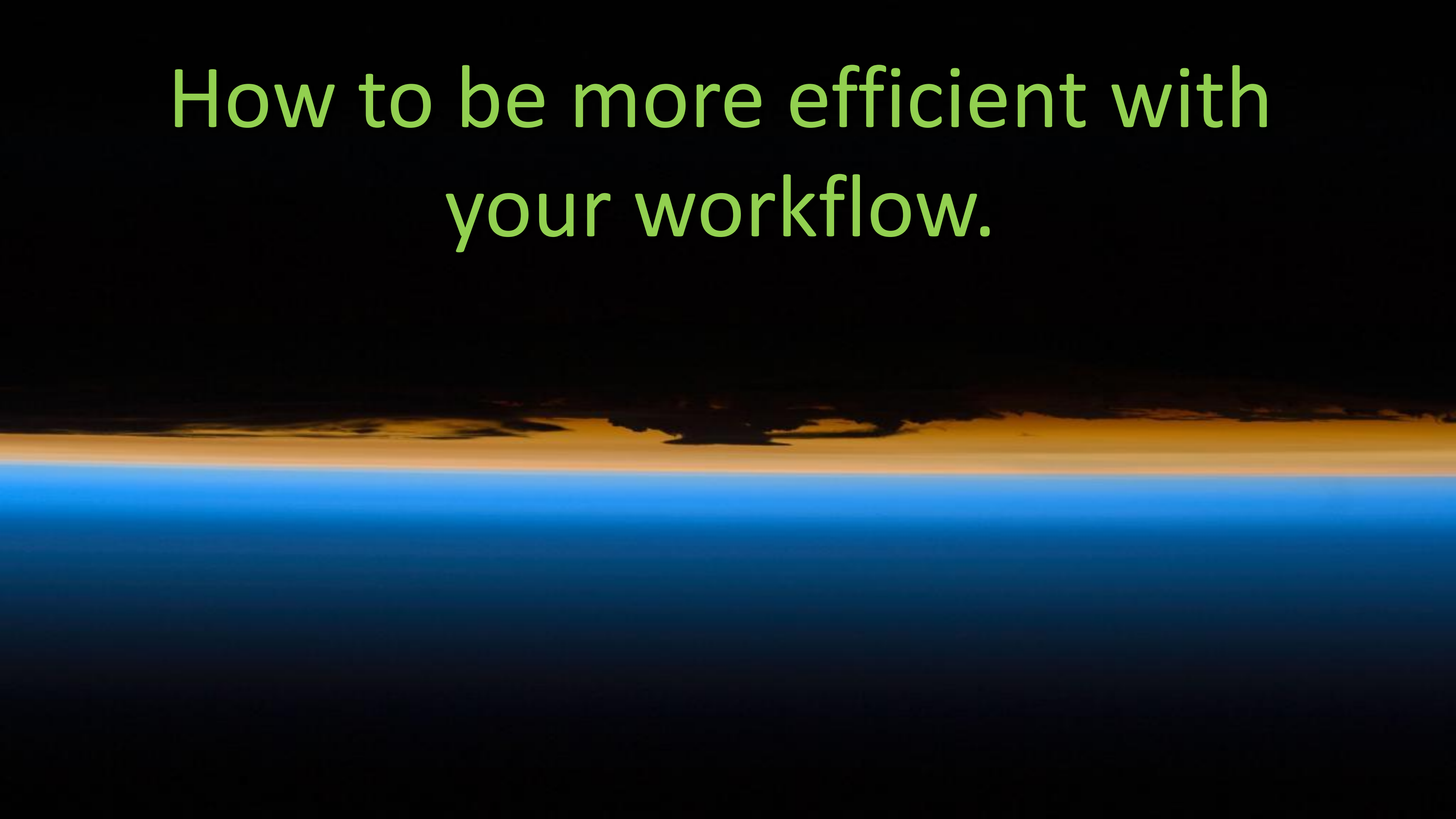
Object	Location	Size	Orientation	Points
Sphere	12,12,23,12,11,16	6	NA	155
Square	...	...	...	...

- You must also include the Spark script that you used. Use RDD's. Do not write an unscalable generic python algorithm.
- It should be <u>one</u> script that I can run myself. It should read in the datafile and spit out all of the data that you used for your summary, like a serious production code. Inline comments are welcome.
- I will also accept a Jupyter notebook, but it is your responsibility to get it running on Bridges (using OnDemand).

You may ask questions at any point before this is due. I will not give you spoilers, but I will clarify anything I can, and give you general suggestions.

This is due anytime before we review answers in class in one week. Once we review, I can give no credit. We will go by the email timestamp, so feel free to request a confirmation that it was received. This is 35% of your final grade.

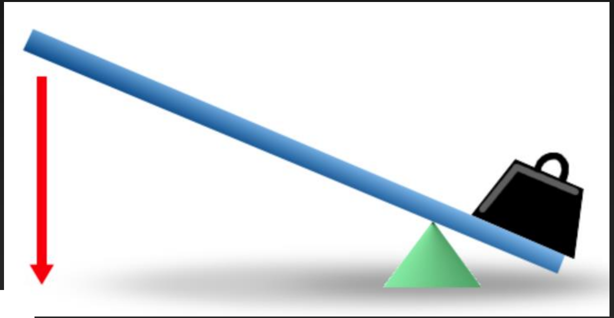
How to be more efficient with your workflow.

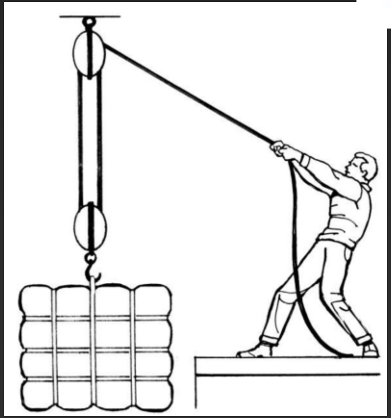


First, *NONE* of this is necessary to be successful in this course. This is mostly to make your future life after this course easier.

My generic solutions to all of our assignments are less than 200 lines of code, total. You could manage that with very poor tools.

Am I Being Condescending?





We are scientists. We start out with first principles.

Philosophy of Course

So that later on we really understand what is happening, when it gets more complex.

And this isn't your project. "Center of gravity, what's that?"

I'm not teaching you to be a code "operator", I'm teaching you to be a code engineer. We will unlearn the quick-and-dirty notebook approach and learn how to build maintainable software from the ground up. You will curse me now and thank me later.

Why not Jupyter Notebooks?

- Jupyter notebooks have their uses
  - great for a report
  - hence instructors love them

- As an IDE it is terrible
  - no debugging
  - no code refactoring
  - no IntelliSense
  - etc.
- Poor code structuring: cells instead of functions encourages anti-patterns:
  - you aren't really going to write functions or classes
  - or package them properly (modules?)
  - you aren't really going to add unit testing without functions
  - whole thing become one big state machine (cells are cached, what order are you running in?)
- Source code control and collaboration becomes useless
  - everything is one big JSON file, good luck with diff/merge

- There is help for many of these issues (nbdime, testbook, etc.)

- Pros use something like PyCharm, or VSCode (with a ton of super useful plugins) or Spyder
  - which can also support Jupyter Notebooks in a more robust fashion
  - why shouldn't you?

Unix commands

You can get very far with a relatively small number of *Unix* (the umbrella term for the family of operating systems that includes *Linux*) commands. Our philosophy will be to introduce them as needed.

You will also find countless help at your fingertips online should you feel the need. Google "how do I do x in linux" if you have any doubts.

The handful of commands that will do most of the work are

`ls`	`list the directory contents`
`cd`	`change the directory you are in`
`cp`	`cp a file`
`rm`	`delete a file`
`pwd`	`show the current directory`
`chmod`	`change file permissions`
`mkdir/rmdir`	`make/remove directory`

If you are put off by the cryptic names, I can't blame you. They do have a somewhat logical etymology, but I won't bore you with it here. As you will ultimately need so few commands, it won't really be a problem.

ls

You will often want to see the contents of your current directory ("folder").

```
[urbanic@bridges2] ls
GPU_Direct_Test                      Test                 Advanced_Computational_Physics
Grading_ACP                          LargeScaleComputing  exec_script.txt
Codes                                MSDAS                proper_C_2D_array_example
Exercises                            Starfield
```

Like most unix commands, it has many options. You can see what they are with *ls --help*. I find a frequently useful combo to be

```
[urbanic@bridges2]$ ls -lt
total 3267940
drwxr-xr-x 2 urbanic pscstaff       4096 Dec 25 20:47 Advanced_Computational_Physics
drwxr-xr-x 2 urbanic pscstaff       4096 Nov 28 23:22 Test
drwxr-x--x 5 urbanic pscstaff       4096 Nov 11 00:26 LargeScaleComputing
-rw-r--r-- 1 urbanic pscstaff         32 Nov  3 23:24 exec_script.txt
drwxr-xr-x 2 urbanic pscstaff       4096 Nov  2 23:33 __pycache__
drwxr-x--- 2 urbanic pscstaff       4096 Apr 26  2021 Starfield
drwxr-x--- 6 urbanic pscstaff       4096 Mar 31  2021 BigData
drwxr-x--- 9 urbanic pscstaff       4096 Mar 10  2021 Codes
lrwxrwxrwx 1 urbanic pscstaff         32 Mar 10  2021 Ocean ->
```

Getting around

You will wish to navigate around the filesystem. It is one big hierarchal tree. You will always start in your home directory. You can move from there with the *cd* command. You can type in a full *path name*

```
[urbanic@bridges2]$ cd /jet/home/urbanic/Test
/jet/home/urbanic
```

or you can just change into any of the directories that happen to be where you are

```
[urbanic@bridges2]$ cd Test
```

or you can move "up" the tree with the .. shorthand

```
[urbanic@bridges2]$ cd ..
```

And doing a *cd* with no argument will teleport you home.

```
[urbanic@bridges2] cd
[urbanic@bridges2] pwd
/jet/home/urbanic/
```

Where am I?

If you ever get confused as to where you are, the *pwd* command will let you know

```
[urbanic@bridges2] pwd
/jet/home/urbanic/Test
```

Feel free to explore. You can't harm anyone else on the system, or go where you aren't welcome.

The reason for this is that Linux has comprehensive permissions on the files and directories. These are controlled with the *chmod* command. It has many details (which are worth knowing), but you can get by for now knowing that everyone is treated as the user/owner (*u*), a member of their group (*g*) or other users (*o*), and they can be allowed to read (*r*), write (*w*) or execute (*x*) a file or directory.

Thus, we can allow all (a) types of users to read a file called open.txt with the command

```
[urbanic@bridges2] chmod a+r open.txt
```

Or retract that permission (and verify with *ls -l*) like so

```
[urbanic@bridges2] chmod a-r open.txt
```

cp

You will often need to copy a file. The unix cp command does that. You specify the source and destination.

```
[urbanic@bridges2] cp test.txt /jet/home/urbanic/subdirectory
```

There are two useful abbreviations that are useful here (as well as elsewhere). First, you can refer to your current directory with the "." shorthand. So, if we want to copy something from somewhere else to "here" we can do

```
[urbanic@bridges2] cp /home/joe/examples/hello.c .
```

And we will now find a copy of hello.c wherever we happen to be.

Often you will want to refer to your home directory directly, without specifying the full path. That is done with ~, and ~*username* can refer to any other user's home directory. For example, no matter where we happen to find ourself in the filesystem, we can always copy the file hello.c from user joe's home directory to our own home directory with the command

```
[urbanic@bridges2] cp ~joe/hello.c ~
```

file path shorthands

Linux *shells* (the name for a particular flavor of command line) have very sophisticated shorthands for the files and filesystem. You can use complex *regular expressions* if you want. The only one I would be remiss not introducing you to here is the `*`, or wildcard. It is used like so

```
[urbanic@bridges2] ls fred*
fred.txt        fred.exe     frederick
```

To recap, the useful shorthands you will want to use are

`~`	user's home directory (yours if no name attached)
`..`	up one level
`.`	this level
`*`	wildcard

And you will find yourself reaching for combinations like

```
[urbanic@bridges2] cd ../../Run_directory
[urbanic@bridges2] cp ~urbanic/data/* ~/data
```

rm and mkdir/rmdir

You will want to delete files. The command is *rm*, and is often used with wildcards to delete multiple files.

```
[urbanic@bridges2] rm output.*
```

This might remove files output.1, output.2 and output.3 at once.

Be careful as there is no undo or recycle bin by default in Linux. Your files are most likely gone forever! You might find that the *cp -r* option to backup a whole directory is a useful hedge against disaster.

Of course you will want to organize your files into directories. You can make directories anywhere that you have permission with *mkdir*, and you can delete them with *rmdir*.

```
[urbanic@bridges2] mkdir Test
[urbanic@bridges2] ls
Test
```

The shell is your friend

There are several very convenient features build into the command line that you will want to use *constantly*. Foremost is command completion and history.

Command completion is where you just hit the tab key, and the shell will attempt to complete whatever you are typing. So, if I with to cp the file data_run10266_batch03.dat up one level, I might just type

```
[urbanic@bridges2] cp dat
```

and hit the tab key. At this point the shell will fill out the full filename and then I can just add the .. on to the line to end up with

```
[urbanic@bridges2] cp data_run10266_batch03.dat ..
```

in just a few keystrokes. Of course this assumes that I don't also have a file named data_run10266_batch04.dat sitting there. If I do, the shell will beep at me and let me know I need to keep typing until I have a unique match. So, there is little risk in just hitting "tab" at any time.

This feature applies to files, paths and commands. Once you start using it, it will become reflexive and save you countless keystrokes.

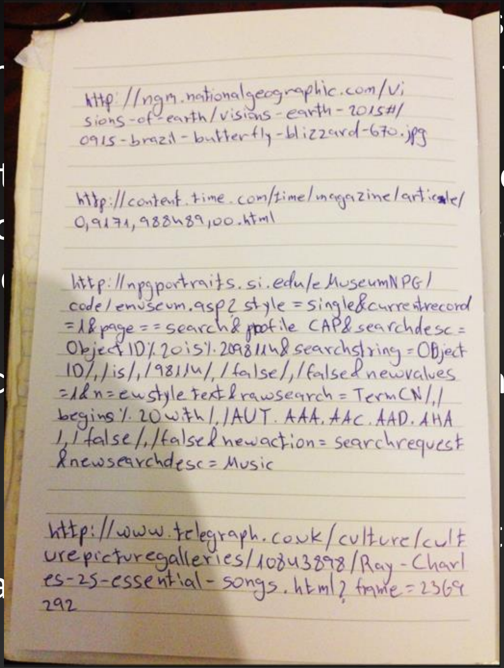
history

A second constantly useful feature is command history. This simply means that the shell keeps track of your previous commands. If you type st, and you can directly access any of them with the ! followed by the line r the most commonly used mode.

The most convenient mode is simply t cycle up through your most recently used commands. When you find the c an hit return, or you can use the other *cursor keys* to edit it a bit befor

If you make a small syntax error in a c complains, you need only hit the up arrow, fix your typo and hit return.

By the way, the *ipython* shell, used in ted this as well. So, you may already be comfortable with these features, a r of notebook users overlook this capability.



http://ngm.nationalgeographic.com/vi
sions-of-earth/visions-earth-2015#/
0915-brazil-butterfly-blizzard-670.jpg

http://content.time.com/time/magazine/article/
0,9171,988489,00.html

http://npgportraits.si.edu/eMuseumNPG/
code/emuseum.asp?style=single&currentrecord
=1&page==search&profile CAP&searchdesc=
Object ID%20is%2098114&searchstring=Object
ID/,/is/,/98114/,/false/,/false&newvalues
=1&n=ewstyle text&rawsearch=TermCN/,/
begins%20with/,/AUT.AAA.AAC.AAD.AHA
/,/false/,/false&newaction=searchrequest
&newsearchdesc=Music

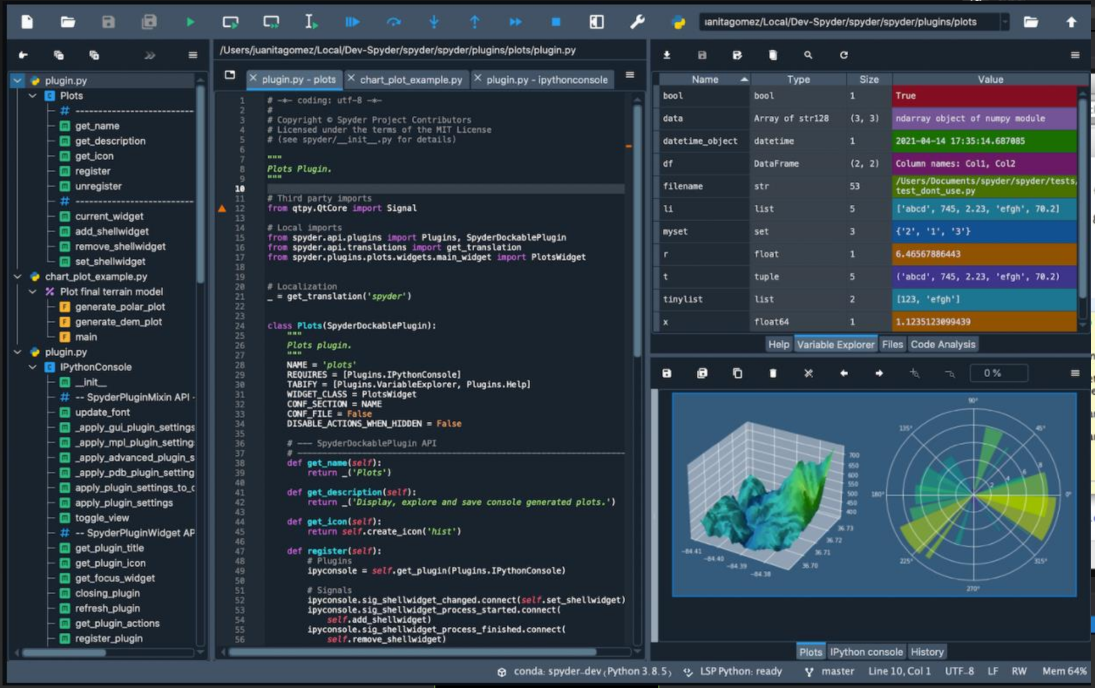
http://www.telegraph.co.uk/culture/cult
urepicturegalleries/10843898/Ray-Charl
es-25-essential-songs.html?frame=2369
292

When I watch someone painfully retype a line or mistype filenames, this is what flashes in my mind...

IDEs

Professionals coders of any sort use Integrated Development Environments. Video game programmers and bank system programmers. Web developers and scientific coders. Everyone.

For various reasons, many of you have not been let in on this secret. We will rectify that before we are done. You may have heard of the most popular ones.

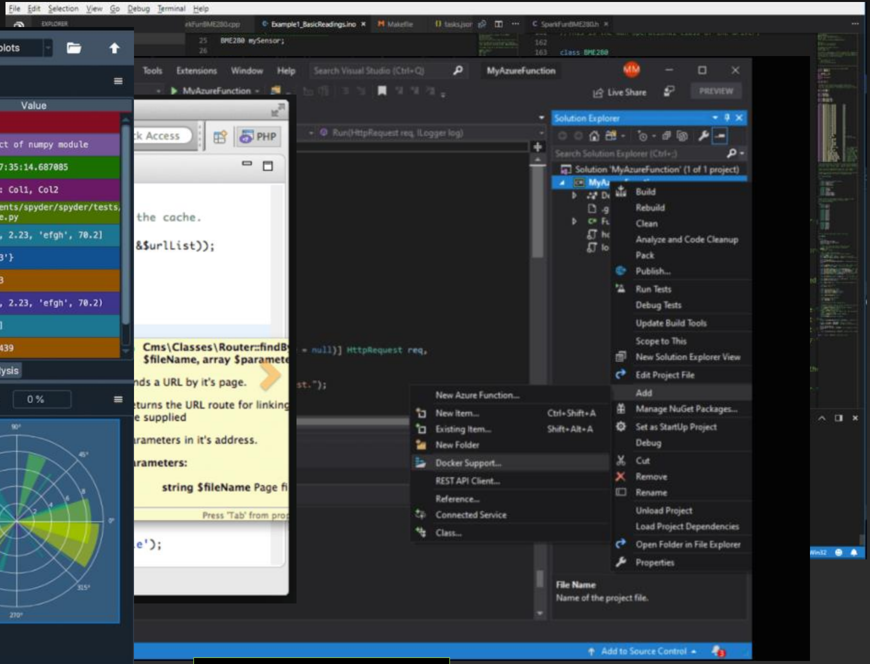


ianitagomez/Local/Dev-Spyder/spyder/spyder/plugins/plots
/Users/juanitagomez/Local/Dev-Spyder/spyder/spyder/plugins/plots/plugin.py
plugin.py - plots
chart_plot_example.py
plugin.py - ipythonconsole
Help
Variable Explorer
Files
Code Analysis
Plots
IPython console
History
conda: spyder-dev (Python 3.8.5)
LSP Python: ready
master
Line 10, Col 1
UTF-8
LF
RW
Mem 64%

```python
# -*- coding: utf-8 -*-
#
# Copyright © Spyder Project Contributors
# Licensed under the terms of the MIT License
# (see spyder/__init__.py for details)

"""
Plots Plugin.
"""


# Third party imports
from qtpy.QtCore import Signal

# Local imports
from spyder.api.plugins import Plugins, SpyderDockablePlugin
from spyder.api.translations import get_translation
from spyder.plugins.plots.widgets.main_widget import PlotsWidget


# Localization
_ = get_translation('spyder')


class Plots(SpyderDockablePlugin):
    """
    Plots plugin.
    """
    NAME = 'plots'
    REQUIRES = [Plugins.IPythonConsole]
    TABIFY = [Plugins.VariableExplorer, Plugins.Help]
    WIDGET_CLASS = PlotsWidget
    CONF_SECTION = NAME
    CONF_FILE = False
    DISABLE_ACTIONS_WHEN_HIDDEN = False

    # --- SpyderDockablePlugin API
    # ------------------------------------------------------------------
    def get_name(self):
        return _('Plots')

    def get_description(self):
        return _('Display, explore and save console generated plots.')

    def get_icon(self):
        return self.create_icon('hist')

    def register(self):
        # Plugins
        ipyconsole = self.get_plugin(Plugins.IPythonConsole)

        # Signals
        ipyconsole.sig_shellwidget_changed.connect(self.set_shellwidget)
        ipyconsole.sig_shellwidget_process_started.connect(
            self.add_shellwidget)
        ipyconsole.sig_shellwidget_process_finished.connect(
            self.remove_shellwidget)
```

Name	Type	Size	Value
bool	bool	1	True
data	Array of str128	(3, 3)	ndarray object of numpy module
datetime_object	datetime	1	2021-04-14 17:35:14.687085
df	DataFrame	(2, 2)	Column names: Col1, Col2
filename	str	53	/Users/Documents/spyder/spyder/tests/test_dont_use.py
li	list	5	['abcd', 745, 2.23, 'efgh', 70.2]
myset	set	3	{'2', '1', '3'}
r	float	1	6.46567886443
t	tuple	5	('abcd', 745, 2.23, 'efgh', 70.2)
tinylist	list	2	[123, 'efgh']
x	float64	1	1.1235123099439

Spyder



Visual Studio

What is so great?

You may have noticed that Jupyter is not included. That is because an IDE is not simply a code editor. It includes powerful tools for working with complex projects.

Editing	syntax highlighting
Autocomplete	intelligent code completion, docs and hovering
Formatting	templates and guideline enforcement
Refactoring	across files/modules
Output Management	plots, graphics, text and shell
Plug-ins	many, many, plug-ins
Personalization	take along across machines and languages
Linting	real-time is nice
Integrated Source Control	git or local
Compiling	and whole build cycle
Remote Use	you are about to find out about this
Debugging	debugging!
Debugging!	debugging!!
Debugging!!	debugging!!!

Long Lifespan Knowledge

In addition, everything we are going to learn in this course is portable, open source and standard. It will be relevant in 10 years, and likely much longer.

I have a lot of code from 20 years ago that is still being actively used. Some of the tools I used to create it are long gone.

The problems I have picked will not penalize us for not using any particular tool.

But, we can incorporate the tools later, once we understand the real foundation. You will not conflate the two. This seems to be a common, and fairly recent, problem.

Editor

You are quite welcome to use a fancy IDE at any point. I will not spend any time helping you configure or debug your installation until we get to that point in the course. Don't distract yourself.

We will start with classic terminal editors. Emacs and vi are both very popular. They are both very powerful. And they both have very nasty learning curves. I use emacs almost everyday. I wouldn't encourage you to learn it. But, either of these are fine choices if you already know the basics.

Otherwise, the nano editor will be quite sufficient. It is simple and friendly. You will understand it almost immediately.

In this demo I'm am going to do a number of tasks a number of different ways, so you can see how painless it is when you have the right tools. These are all free and easy to install.

I like MobaXterm as it packages terminal session management, file transfer and an X server in one simple GUI. However there are many other similar packages available, many for free like this.

I will use Spyder* as my IDE. I have installed it via Anaconda, which I highly recommend as a one-stop-shop for Python packages that are compatible (no weird dependency or compatibility issues, well mostly). It is targeted at scientists and data scientists, so you will find much of what you will want to use here.

* Somebody is going to ask me about running Spyder remotely, which is really cool, but somewhat advanced and not at all necessary for our purposes. Here is a simple how-to: http://docs.spyder-ide.org/current/panes/ipythonconsole.html?highlight=ssh#connect-to-a-remote-kernel . You can do the same with VSCode.