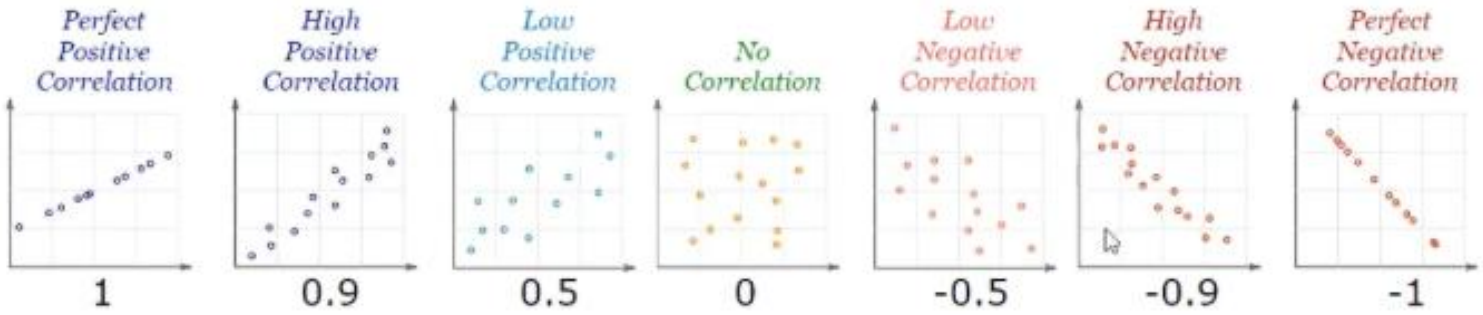
Feature Engineering 2

9 April 2026

Alex Lyman

A few slides we didn’t get to last time

Correlation




```
In [116]: dataset.corr()
```

Out[116]:

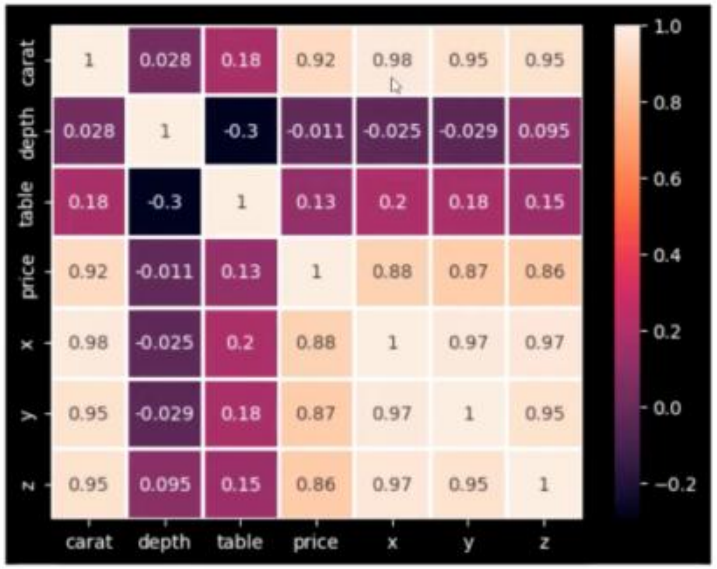
	carat	depth	table	price	x	y	z
carat	1.000000	0.028224	0.181618	0.921591	0.975094	0.951722	0.953387
depth	0.028224	1.000000	-0.295779	-0.010647	-0.025289	-0.029341	0.094924
table	0.181618	-0.295779	1.000000	0.127134	0.195344	0.183760	0.150929
price	0.921591	-0.010647	0.127134	1.000000	0.884435	0.865421	0.861249
x	0.975094	-0.025289	0.195344	0.884435	1.000000	0.974701	0.970772
y	0.951722	-0.029341	0.183760	0.865421	0.974701	1.000000	0.952006
z	0.953387	0.094924	0.150929	0.861249	0.970772	0.952006	1.000000

Visualizing Correlation

- Use seaborn's heatmap function

```
import seaborn as sns
# visualize correlation with heatmap
sns.heatmap(df.corr(), linewidths=1, annot=True)
```

```
Out[119]: <AxesSubplot:>
```



Output Class Skew

- Important output class may be rare
  - Consider nuclear reactor data – Meltdowns vs non-meltdown prediction
- Undersampling or Oversampling
  - Undersampling – if you have 100,000 instances and only 1,000 in minority class, use a high percentage of the minority class and sample from the majority class until you get your desired distribution for training (50/50?)
  - Oversampling – make duplicates of the minority class and add it to the training data to get a better distribution (might cause memorization)
    - Could add copies with some jitter (be careful!)
  - SMOTE (Synthetic Minority Oversampling Technique)
    - pip install imbalanced-learn
- Change your loss function to weight the minority class more heavily
- Use Precision/Recall/F1 or ROC curve rather than just accuracy

Feature Engineering

Feature engineering

From Wikipedia, the free encyclopedia

Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. These features can be used to improve the performance of machine learning algorithms. Feature engineering can be considered as applied machine learning itself [1].

Importance [edit]

Features are important to predictive models and influence results [3].

It is asserted that feature engineering plays an important part of Kaggle competitions [4] and machine learning project's success or failure [5]. Moreover, Python also have so much importance in Machine learning and developers find it so easy to work through Python. [6].

Process [edit]

The feature engineering process is:[7]

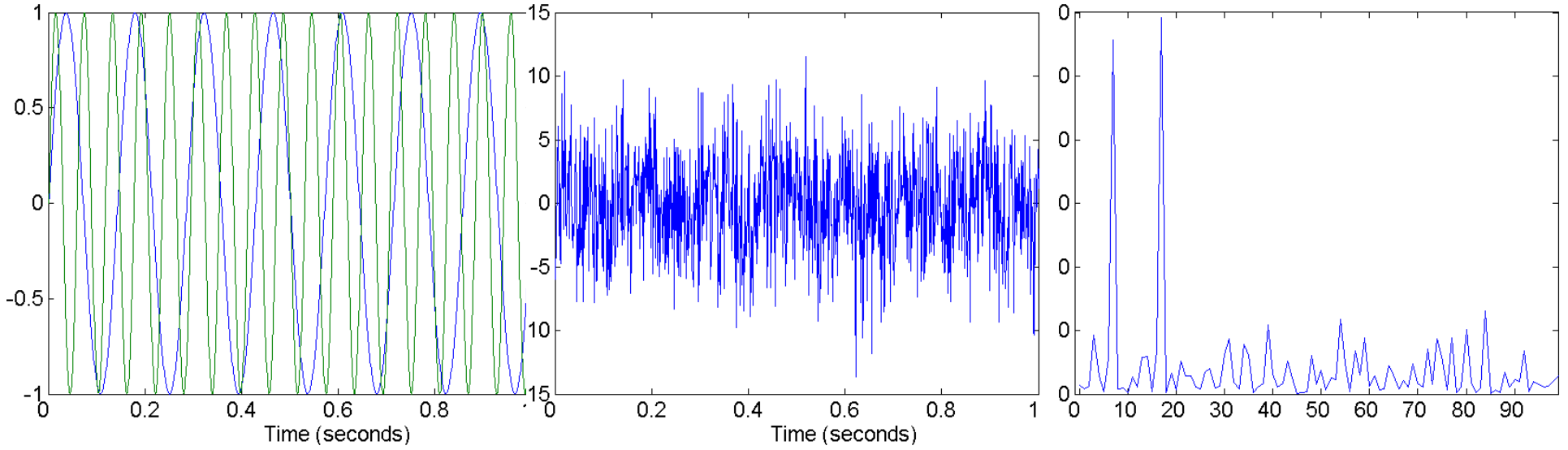
- Brainstorming or testing features;[8]
- Deciding what features to create;
- Creating features;
- Checking how the features work with your model;
- Improving your features if needed;
- Go back to brainstorming/creating more features until the work is done.

Feature Engineering

- Transform existing features (e.g., use area code to approximate location of home)
- Create new features (e.g., compute time to event from dates in dataset)
- Extract numeric data from non-numeric sources
  - Images
  - Video
  - Text
  - Audio

Mapping Data to a New Space

- Fourier Transform
- Wavelet Transform
- Problem domain dependent → Understand the domain

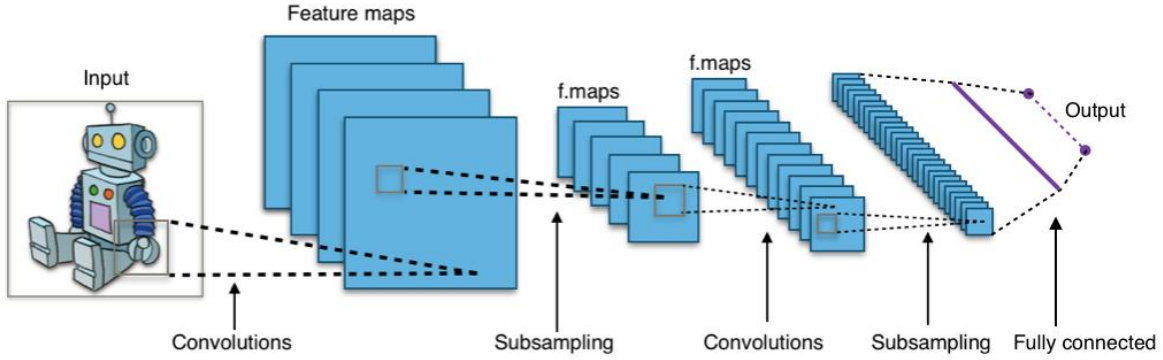


Two Sine Waves **Two Sine Waves + Noise** **Frequency**

Data Mapping

Audio to classify

Image Classifier



Data Mapping

Audio to classify

Transform to image

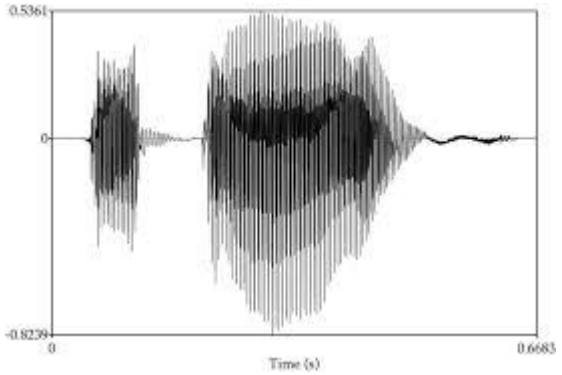
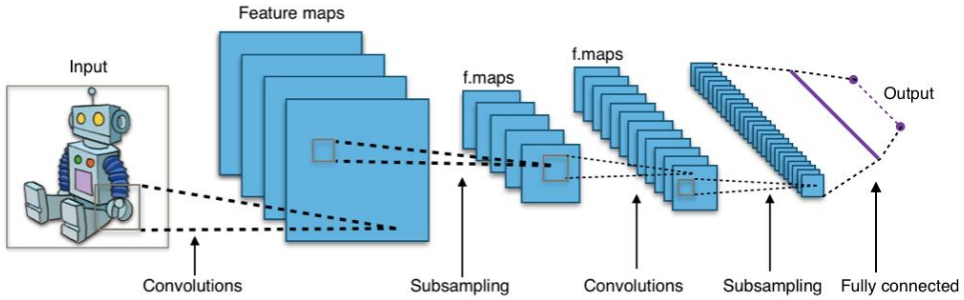


Image Classifier



Feature Selection

The curse of Dimensionality

- Some machine learning problems will have A LOT of features (think thousands or even millions)
- This can make model training very slow (even using some of the big data solutions we've learned)
- It can also make it hard to find a good model (high dimensional spaces are sparse)
- Furthermore, it is extremely hard to visualize high dimensional problems
- The problem of having many many features is sometimes called the curse of dimensionality

Curse of dimensionality

From Wikipedia, the free encyclopedia

The **curse of dimensionality** refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression was coined by Richard E. Bellman when considering problems in dynamic programming.[1][2]

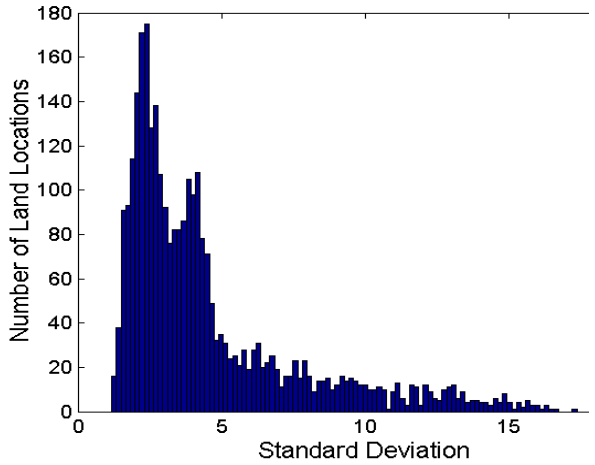
Dimensionally cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

Possible Remedies

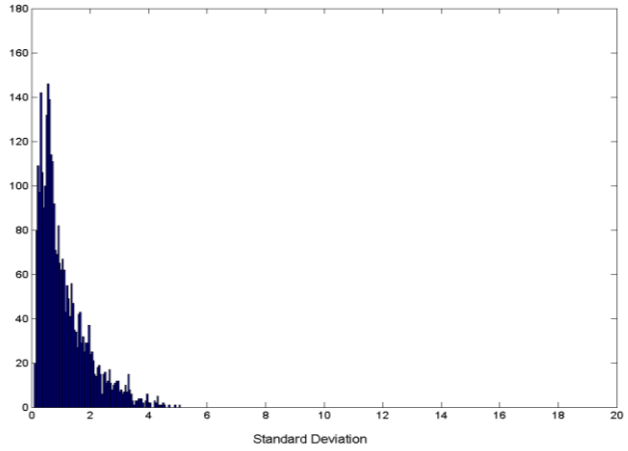
- Feature selection
  - Filter-based methods (choose features based on some metric)
  - Wrapper-based methods
    - Forward, backward, or stepwise selection
- Aggregation
- Dimension reduction
  - Principal Component Analysis (PCA)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Multidimensional Scaling (MDS)
  - Uniform Manifold Approximation and Projection (UMAP)

Aggregation

- Combining two or more attributes (or objects) into a single attribute (or object)
- Purpose
  - Data reduction
  - Change of scale
  - More stability



Standard Deviation of Average Monthly Precipitation



Standard Deviation of Average Yearly Precipitation

PCA

- Find a lower-dimensional hyperplane that preserves most of the variance in the data
- Project the data onto that hyperplane
- Consider the following example where we start with 2 dimensions and will project down to 1 dimension

PCA

Which hyperplane (or in this case line) would you choose the best represent the original data (i.e., has the most variance from the data)?

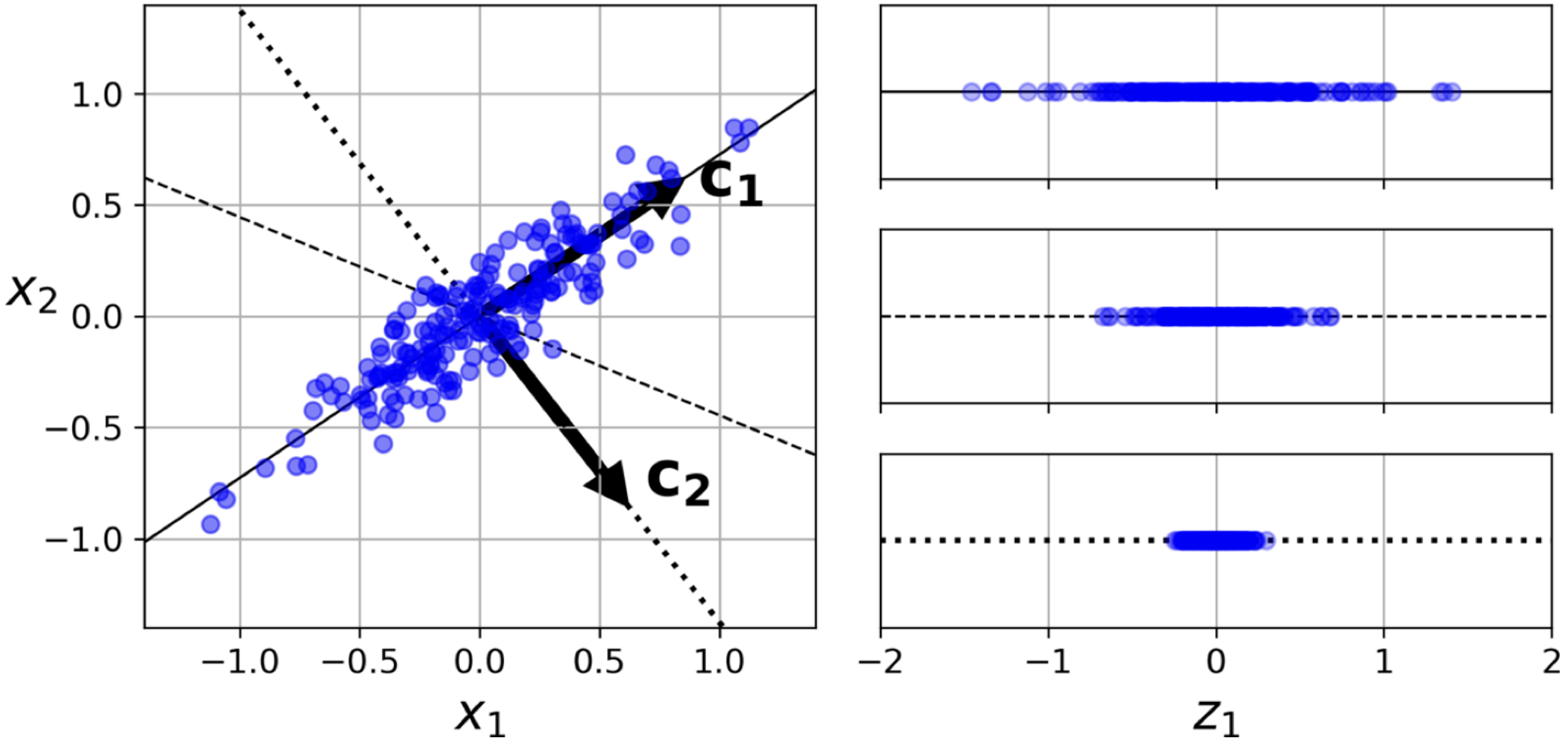


Image from "Hands-On Machine Learning with Scikit-Learn & TensorFlow" by Aurelien Geron

Finding Principal Components

- Principal components can be found by computing eigenvalues and eigenvectors of the covariance matrix of centered data
- They can also be found by computing the singular value decomposition of centered data
- When finding PCs, data should always be centered, and it should usually be scaled too

A few more PCA details (using eigenvalues and eigenvectors)

- Find the covariance matrix of the centered data, *S (d* x *d* matrix*)*
- Compute the eigenvalues and eigenvectors of *S*
- Order the eigenvalues from largest to smallest
- The percent of variation explained by the i^{th} PC is: $\frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j}$

- Select r dimensions based on desired variance explained
- Project onto r-dimensional hyperplane: $X_{proj} = XQ_r$

  (where Q_r is the matrix with the first *r* eigenvectors)

Or use `sklearn`

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
#OR
pca = PCA(n_components = .9)

# if n_components is between 0 and 1, it will be interpreted
# as the desired percent variation explained

pca.fit(X)
X_transform = pca.transform(X)

X_reduced = pca.inverse_transform(X)
```

You should normalize the data first.

UMAP

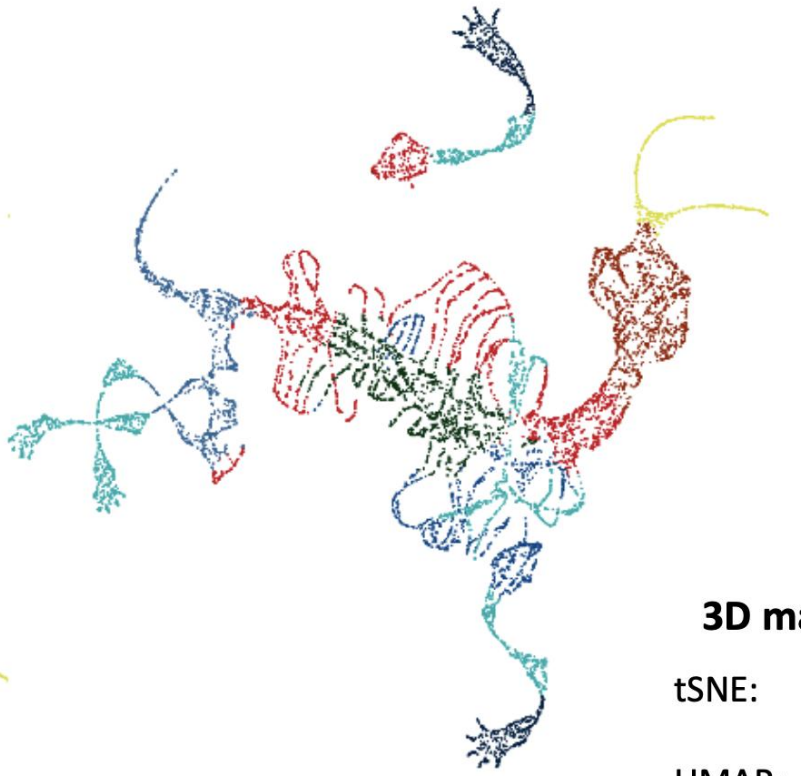
In [55]:
```
from umap import UMAP
from sklearn.preprocessing import MinMaxScaler

# Scale features to [0,1] range
X_scaled = MinMaxScaler().fit_transform(X_train)
# Initialize and fit UMAP
mapper = UMAP(n_components=2, metric="cosine").fit(X_scaled)
# Create a DataFrame of 2D embeddings
df_emb = pd.DataFrame(mapper.embedding_, columns=["X", "Y"])
df_emb["label"] = y_train
df_emb.head()
```

Out[55]:

	X	Y	label
0	4.341662	6.394966	0
1	-3.172323	5.713070	0
2	5.185281	2.880501	3
3	-2.511224	3.013335	2
4	-3.489840	3.741451	3

2D t-SNE projection

2D UMAP projection



3D mammoth skeleton projected into 2D

tSNE:	Perplexity 2000	2h 5min
UMAP:	Nneigh 200, mindist 0.25,	3min

https://pair-code.github.io/understanding-umap/

Take Home Message

1. Know how your data was collected
2. Be thoughtful and transparent when cleaning and transforming
3. Consider how you can add value through the data collection and cleaning process
4. This part can be just as fun as modeling!

Code for PCA, T-SNE, and UMAP