# Introduction to Clustering and its Applications

Dongfeng Li

Instructor: Yikun Zhang

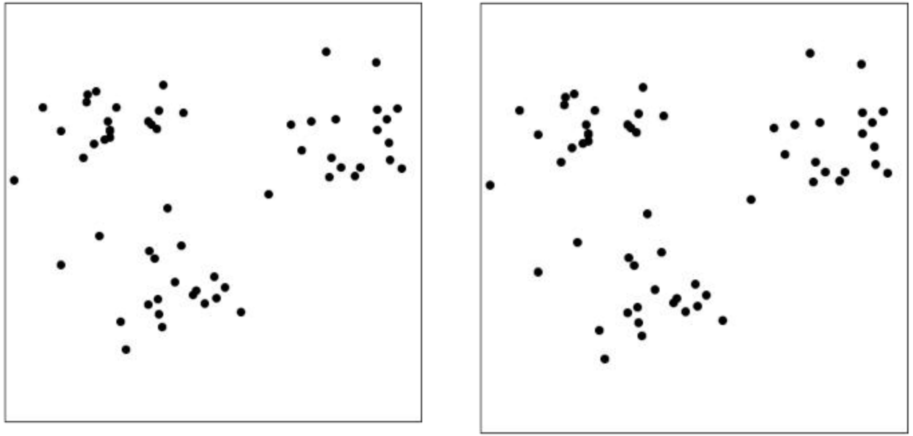
# Index

# Introduction

## Clustering

- In many real world contexts, there aren't clearly defined labels so we won't be able to do classification
- We will need to come up with methods that uncover structure from the (unlabeled) input data $X$.
- **Clustering** is an automatic process of trying to find related groups within the given dataset.

$Input: x_1,\ x_2, \dots, x_n$ $\quad$ $Output: z_1,\ z_2, \dots, z_n$



# Materials and Methods

## Data

First-hand data from hometown powerplant.

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | TIME | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | |
| 2 | | BurdenOfPlant | | PowerOfEngine | | TotalAirFlow | | TotalCoal | | BInduceFanExtend | | AInduceFanExtend | |
| 3 | 2019/5/16 0:00 | -99.96077 | OK | 236.30933 | OK | 912.93738 | OK | 97.152702 | OK | 52.201302 | OK | 51.619404 | OK |
| 4 | 2019/5/16 0:05 | -99.96077 | OK | 236.64929 | OK | 906.98834 | OK | 97.122398 | OK | 52.167515 | OK | 51.510666 | OK |
| 5 | 2019/5/16 0:10 | -99.96077 | OK | 236.66893 | OK | 908.45911 | OK | 97.3022 | OK | 52.206203 | OK | 51.514648 | OK |
| 6 | 2019/5/16 0:15 | -99.96077 | OK | 235.4534 | OK | 910.43323 | OK | 97.482002 | OK | 52.244888 | OK | 51.520927 | OK |
| 7 | 2019/5/16 0:20 | -99.96077 | OK | 234.36554 | OK | 904.2218 | OK | 96.28569 | OK | 52.283577 | OK | 51.52721 | OK |
| 8 | 2019/5/16 0:25 | -99.96077 | OK | 234.68204 | OK | 903.49042 | OK | 97.021927 | OK | 52.322266 | OK | 51.533489 | OK |
| 9 | 2019/5/16 0:30 | -99.96077 | OK | 234.99852 | OK | 900.04553 | OK | 97.758163 | OK | 52.36095 | OK | 51.539772 | OK |
| 10 | 2019/5/16 0:35 | -99.96077 | OK | 235.315 | OK | 909.71777 | OK | 98.4944 | OK | 52.399639 | OK | 51.546051 | OK |
| 11 | 2019/5/16 0:40 | -99.96077 | OK | 235.98448 | OK | 907.79218 | OK | 99.230637 | OK | 52.438328 | OK | 51.552334 | OK |
| 12 | 2019/5/16 0:45 | -99.96077 | OK | 235.12865 | OK | 918.35657 | OK | 99.661842 | OK | 52.529381 | OK | 51.558613 | OK |
| 13 | 2019/5/16 0:50 | -99.96077 | OK | 234.17522 | OK | 917.34961 | OK | 99.295532 | OK | 52.627354 | OK | 51.564896 | OK |
| 14 | 2019/5/16 0:55 | -99.96077 | OK | 236.40817 | OK | 908.47229 | OK | 98.92923 | OK | 52.725323 | OK | 51.571175 | OK |
| 15 | 2019/5/16 1:00 | -99.96077 | OK | 235.90971 | OK | 902.5365 | OK | 98.56292 | OK | 52.823292 | OK | 51.577457 | OK |
| 16 | 2019/5/16 1:05 | -99.96077 | OK | 235.37727 | OK | 899.09155 | OK | 98.31562 | OK | 52.921265 | OK | 51.98439 | OK |
| 17 | 2019/5/16 1:10 | -99.96077 | OK | 235.11009 | OK | 908.47229 | OK | 99.100967 | OK | 53.282002 | OK | 52.952244 | OK |
| 18 | 2019/5/16 1:15 | -99.96077 | OK | 234.84291 | OK | 934.15015 | OK | 99.886307 | OK | 54.417202 | OK | 53.920094 | OK |
| 19 | 2019/5/16 1:20 | -99.96077 | OK | 234.57573 | OK | 913.90466 | OK | 97.412666 | OK | 54.999222 | OK | 54.79063 | OK |
| 20 | 2019/5/16 1:25 | -99.96077 | OK | 234.30855 | OK | 912.89771 | OK | 97.931602 | OK | 53.657471 | OK | 53.812138 | OK |
| 21 | 2019/5/16 1:30 | -99.96077 | OK | 234.04137 | OK | 909.43958 | OK | 98.450539 | OK | 52.762287 | OK | 52.833649 | OK |
| 22 | 2019/5/16 1:35 | -99.96077 | OK | 233.7742 | OK | 915.36212 | OK | 98.150261 | OK | 53.94828 | OK | 53.024765 | OK |
| 23 | 2019/5/16 1:40 | -99.96077 | OK | 233.5414 | OK | 908.4458 | OK | 96.96051 | OK | 54.261009 | OK | 53.337887 | OK |
| 24 | 2019/5/16 1:45 | -99.96077 | OK | 233.3168 | OK | 905.98138 | OK | 96.83345 | OK | 53.734726 | OK | 53.096699 | OK |
| 25 | 2019/5/16 1:50 | -99.96077 | OK | 233.26668 | OK | 899.06506 | OK | 96.706398 | OK | 53.208443 | OK | 52.855511 | OK |
| 26 | 2019/5/16 1:55 | -99.96077 | OK | 233.21657 | OK | 900.54901 | OK | 96.579346 | OK | 52.682156 | OK | 52.614323 | OK |
| 27 | 2019/5/16 2:00 | -99.96077 | OK | 233.16644 | OK | 906.00787 | OK | 96.452286 | OK | 52.174076 | OK | 52.373135 | OK |
| 28 | 2019/5/16 2:05 | -99.96077 | OK | 233.11633 | OK | 909.45276 | OK | 96.325233 | OK | 51.969036 | OK | 52.131947 | OK |
| 29 | 2019/5/16 2:10 | -99.96077 | OK | 233.06621 | OK | 905.50439 | OK | 97.442825 | OK | 51.763996 | OK | 51.890759 | OK |
| 30 | 2019/5/16 2:15 | -99.96077 | OK | 233.10202 | OK | 905.98138 | OK | 98.334267 | OK | 51.558956 | OK | 51.64957 | OK |
| 31 | 2019/5/16 2:20 | -99.96077 | OK | 233.41496 | OK | 906.96185 | OK | 97.820923 | OK | 51.353912 | OK | 51.408382 | OK |
| 32 | 2019/5/16 2:25 | -99.96077 | OK | 233.72791 | OK | 905.00092 | OK | 97.307579 | OK | 51.148872 | OK | 51.167194 | OK |
| 33 | 2019/5/16 2:30 | -99.96077 | OK | 234.04086 | OK | 900.19568 | OK | 96.794235 | OK | 50.943832 | OK | 50.926006 | OK |
| 34 | 2019/5/16 2:35 | -99.96077 | OK | 234.35381 | OK | 900.05878 | OK | 96.280891 | OK | 50.738792 | OK | 50.684818 | OK |
| 35 | 2019/5/16 2:40 | -99.96077 | OK | 233.52443 | OK | 899.88654 | OK | 95.767548 | OK | 50.533752 | OK | 50.44363 | OK |
| 36 | 2019/5/16 2:45 | -99.96077 | OK | 231.8333 | OK | 904.02039 | OK | 95.254204 | OK | 50.328709 | OK | 50.202442 | OK |
| 37 | 2019/5/16 2:50 | -99.96077 | OK | 229.12521 | OK | 879.03156 | OK | 94.048073 | OK | 50.123669 | OK | 49.961254 | OK |
| 38 | 2019/5/16 2:55 | -99.96077 | OK | 228.5052 | OK | 883.72192 | OK | 94.431305 | OK | 50.499115 | OK | 50.364414 | OK |
| 39 | 2019/5/16 3:00 | -99.96077 | OK | 227.88519 | OK | 878.28076 | OK | 94.446594 | OK | 51.082069 | OK | 51.001881 | OK |
| 40 | 2019/5/16 3:05 | -99.96077 | OK | 227.68805 | OK | 884.70239 | OK | 93.350403 | OK | 51.66502 | OK | 51.639347 | OK |
| 41 | 2019/5/16 3:10 | -99.96077 | OK | 227.82768 | OK | 886.42487 | OK | 95.387619 | OK | 52.247971 | OK | 52.276817 | OK |
| 42 | 2019/5/16 3:15 | -99.96077 | OK | 227.789 | OK | 907.49182 | OK | 97.657578 | OK | 52.683952 | OK | 52.914284 | OK |
| 43 | 2019/5/16 3:20 | -99.96077 | OK | 227.50406 | OK | 884.23602 | OK | 95.660812 | OK | 51.746529 | OK | 52.007488 | OK |
| 44 | 2019/5/16 3:25 | -99.96077 | OK | 227.42073 | OK | 877.29584 | OK | 94.697952 | OK | 50.809105 | OK | 51.047447 | OK |
| 45 | 2019/5/16 3:30 | -99.96077 | OK | 227.70638 | OK | 878.79309 | OK | 93.10981 | OK | 49.871685 | OK | 50.087402 | OK |

| | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL9]TI | | PITOPI. 910TL. [TL | |
| 2 | AInduceFanCurrent | | BInduceFanCurrent | | 9AborstionExitTemp1 | | 9AborstionExitTemp2 | | 9AborstionExitTemp3 | | 9DesulfurizationEffi | | 9InputAirFlow | |
| 3 | 160.0752 | OK | 162.20375 | OK | 48.62722 | OK | 48.647926 | OK | 48.042564 | OK | 98.972153 | OK | 1082.8059 | OK |
| 4 | 157.11647 | OK | 158.50661 | OK | 48.645256 | OK | 48.67001 | OK | 48.051273 | OK | 99.130425 | OK | 1059.5935 | OK |
| 5 | 157.24234 | OK | 157.13161 | OK | 48.663292 | OK | 48.692093 | OK | 48.059986 | OK | 99.155052 | OK | 1081.2317 | OK |
| 6 | 157.44067 | OK | 159.63234 | OK | 48.681324 | OK | 48.714172 | OK | 48.068695 | OK | 99.179672 | OK | 1065.9972 | OK |
| 7 | 156.98769 | OK | 158.80907 | OK | 48.69936 | OK | 48.736256 | OK | 48.077404 | OK | 99.2043 | OK | 1077.6847 | OK |
| 8 | 156.83818 | OK | 155.97313 | OK | 48.717392 | OK | 48.758339 | OK | 48.086117 | OK | 99.228928 | OK | 1081.1243 | OK |
| 9 | 156.71387 | OK | 156.46387 | OK | 48.735428 | OK | 48.780418 | OK | 48.094826 | OK | 99.253548 | OK | 1083.5823 | OK |
| 10 | 156.58955 | OK | 157.79459 | OK | 48.753464 | OK | 48.802502 | OK | 48.103539 | OK | 99.278175 | OK | 1082.1748 | OK |
| 11 | 157.80341 | OK | 159.76068 | OK | 48.771496 | OK | 48.824585 | OK | 48.112247 | OK | 99.302803 | OK | 1090.0674 | OK |
| 12 | 158.93718 | OK | 161.37271 | OK | 48.787025 | OK | 48.846664 | OK | 48.120956 | OK | 99.327423 | OK | 1083.0566 | OK |
| 13 | 158.71207 | OK | 161.25337 | OK | 48.801228 | OK | 48.868748 | OK | 48.129669 | OK | 99.352051 | OK | 1100.569 | OK |
| 14 | 158.13989 | OK | 159.34662 | OK | 48.815434 | OK | 48.890831 | OK | 48.138378 | OK | 99.376678 | OK | 1092.4124 | OK |
| 15 | 157.1711 | OK | 158.39644 | OK | 48.829636 | OK | 48.91291 | OK | 48.147087 | OK | 99.401299 | OK | 1077.9198 | OK |
| 16 | 157.28561 | OK | 159.36942 | OK | 48.843838 | OK | 48.934994 | OK | 48.1558 | OK | 99.425926 | OK | 1085.5874 | OK |
| 17 | 159.36604 | OK | 162.34364 | OK | 48.85804 | OK | 48.957077 | OK | 48.164509 | OK | 99.450554 | OK | 1082.7114 | OK |
| 18 | 168.49081 | OK | 172.20274 | OK | 48.872242 | OK | 48.979156 | OK | 48.173222 | OK | 99.475174 | OK | 1123.0363 | OK |
| 19 | 165.86543 | OK | 167.4613 | OK | 48.886444 | OK | 49.00124 | OK | 48.181931 | OK | 99.499802 | OK | 1103.4413 | OK |
| 20 | 159.56488 | OK | 160.53636 | OK | 48.900646 | OK | 49.023323 | OK | 48.19064 | OK | 99.524429 | OK | 1104.7762 | OK |
| 21 | 160.45299 | OK | 161.82588 | OK | 48.914848 | OK | 49.045403 | OK | 48.199352 | OK | 99.549049 | OK | 1085.2242 | OK |
| 22 | 163.91995 | OK | 168.43022 | OK | 48.92905 | OK | 49.067486 | OK | 48.208061 | OK | 99.573677 | OK | 1111.927 | OK |
| 23 | 162.20056 | OK | 163.17329 | OK | 48.943253 | OK | 49.089569 | OK | 48.216774 | OK | 99.598305 | OK | 1099.8215 | OK |
| 24 | 158.59279 | OK | 159.87076 | OK | 48.957455 | OK | 49.111649 | OK | 48.225483 | OK | 99.622925 | OK | 1077.3386 | OK |
| 25 | 158.59401 | OK | 159.38022 | OK | 48.971657 | OK | 49.133732 | OK | 48.234192 | OK | 99.647552 | OK | 1103.9589 | OK |
| 26 | 158.59523 | OK | 158.20158 | OK | 48.985859 | OK | 49.155815 | OK | 48.242905 | OK | 99.67218 | OK | 1078.8448 | OK |
| 27 | 158.59645 | OK | 160.09877 | OK | 49.000061 | OK | 49.177895 | OK | 48.251614 | OK | 99.2183 | OK | 1090.8875 | OK |
| 28 | 158.59767 | OK | 161.32814 | OK | 49.014263 | OK | 49.199978 | OK | 48.260326 | OK | 99.235291 | OK | 1114.0399 | OK |
| 29 | 158.59889 | OK | 162.06055 | OK | 49.028469 | OK | 49.222061 | OK | 48.269035 | OK | 99.253098 | OK | 1090.9503 | OK |
| 30 | 158.60011 | OK | 160.75931 | OK | 49.042671 | OK | 49.244141 | OK | 48.277744 | OK | 99.270905 | OK | 1076.5092 | OK |
| 31 | 158.60133 | OK | 160.47404 | OK | 49.056873 | OK | 49.266224 | OK | 48.286457 | OK | 99.288704 | OK | 1100.0784 | OK |
| 32 | 158.40552 | OK | 160.39709 | OK | 49.071075 | OK | 49.288307 | OK | 48.295166 | OK | 99.306511 | OK | 1070.8879 | OK |
| 33 | 158.0872 | OK | 160.22743 | OK | 49.085278 | OK | 49.310387 | OK | 48.303879 | OK | 99.324318 | OK | 1079.0116 | OK |
| 34 | 157.76889 | OK | 159.95848 | OK | 49.09948 | OK | 49.33247 | OK | 48.312588 | OK | 99.342125 | OK | 1075.2229 | OK |
| 35 | 157.45058 | OK | 159.33435 | OK | 49.113682 | OK | 49.354553 | OK | 48.321297 | OK | 99.359924 | OK | 1077.1824 | OK |
| 36 | 157.13228 | OK | 158.31682 | OK | 49.127884 | OK | 49.376633 | OK | 48.330009 | OK | 99.377731 | OK | 1066.2306 | OK |
| 37 | 152.69093 | OK | 153.00484 | OK | 49.142086 | OK | 49.398716 | OK | 48.338718 | OK | 99.395538 | OK | 1078.0537 | OK |
| 38 | 151.80992 | OK | 153.00789 | OK | 49.156288 | OK | 49.420799 | OK | 48.347431 | OK | 99.413345 | OK | 1071.2308 | OK |
| 39 | 152.24785 | OK | 153.7235 | OK | 49.17049 | OK | 49.442879 | OK | 48.35614 | OK | 99.431145 | OK | 1064.782 | OK |
| 40 | 152.94183 | OK | 154.30724 | OK | 49.184692 | OK | 49.450676 | OK | 48.364849 | OK | 99.448952 | OK | 1055.6884 | OK |
| 41 | 153.79955 | OK | 154.87686 | OK | 49.198895 | OK | 49.452915 | OK | 48.373562 | OK | 99.466759 | OK | 1054.2303 | OK |
| 42 | 157.45749 | OK | 159.58978 | OK | 49.213097 | OK | 49.455154 | OK | 48.382271 | OK | 99.484566 | OK | 1074.145 | OK |
| 43 | 153.66884 | OK | 155.63359 | OK | 49.227303 | OK | 49.457394 | OK | 48.390984 | OK | 99.502373 | OK | 1076.3883 | OK |
| 44 | 151.73872 | OK | 153.47995 | OK | 49.241505 | OK | 49.459637 | OK | 48.399693 | OK | 99.520172 | OK | 1090.759 | OK |
| 45 | 149.29523 | OK | 149.46278 | OK | 49.255707 | OK | 49.461876 | OK | 48.408401 | OK | 99.537979 | OK | 1067.5957 | OK |

5dd

-Full record on 05/16/2019.
-Size of 4608*79.

# Materials and Methods
# Data

Mixed slurry($H_2O$)

$H_2SO_3 \rightarrow 2H^{+}+CaCO_3$

Rising flue gas($SO_2$)

## Huangtai power plant desulfurization process

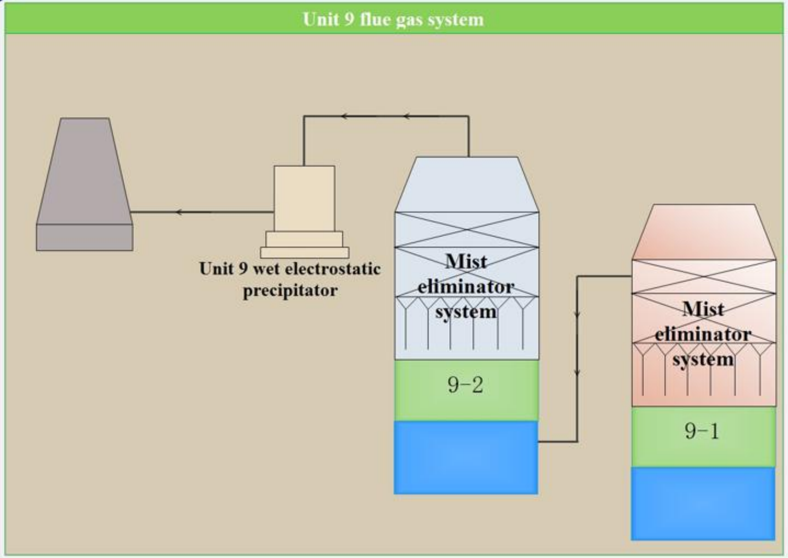



Figure 6. The framework of system

Research object is Unit 9 of Huangtai with double absorption towers.

The system includes:
- $SO_2$ absorption sys,
- flue-gas sys,
- absorbent preparation sys,
- gypsum treatment sys,
- processed water sys,
- accident slurry sys, etc

# Methods: Variable Selection

Variable selection by variance

Why?

- Too many data points, some of which do not provide useful information.

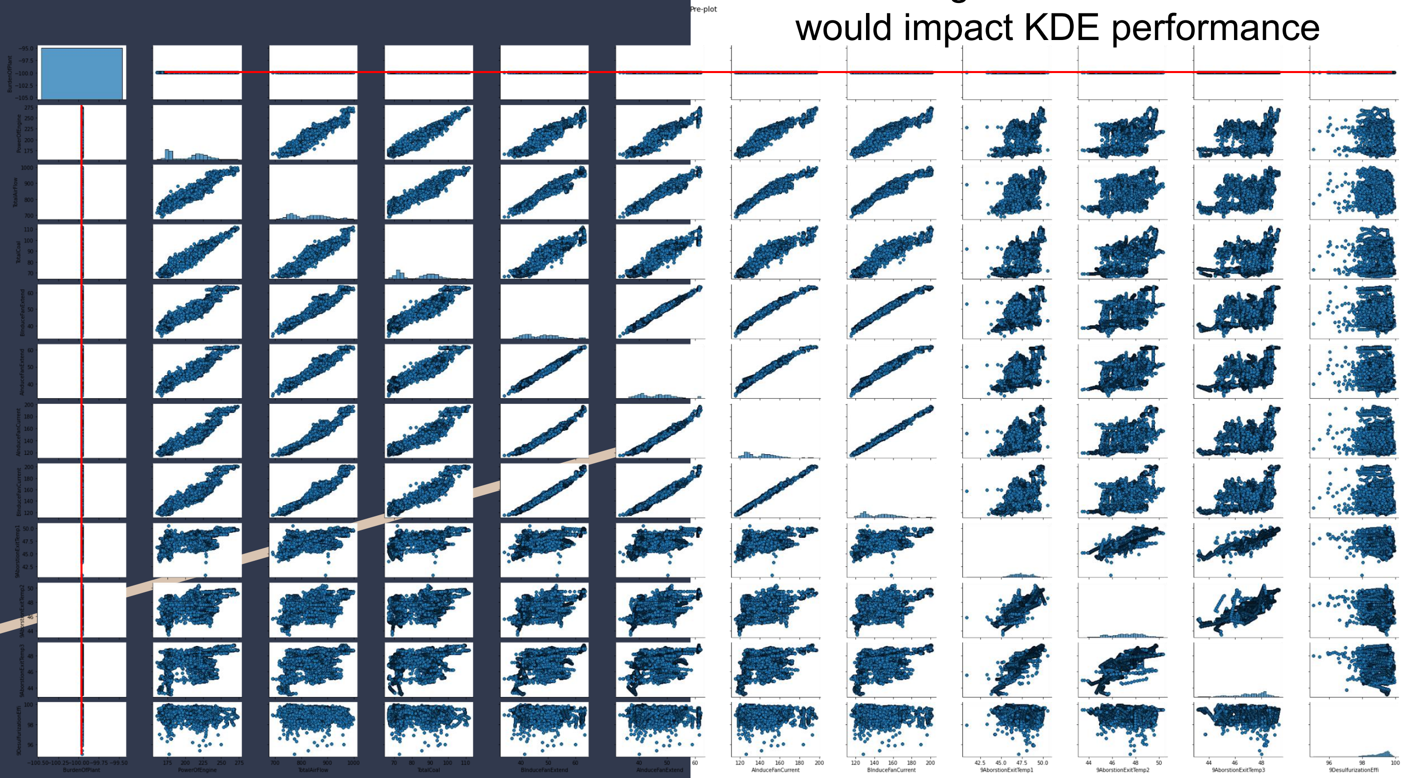
- High dimensions bring curse of dimensionality.

- Perform clustering on scattered data instead of too much concentrated data.

# Methods

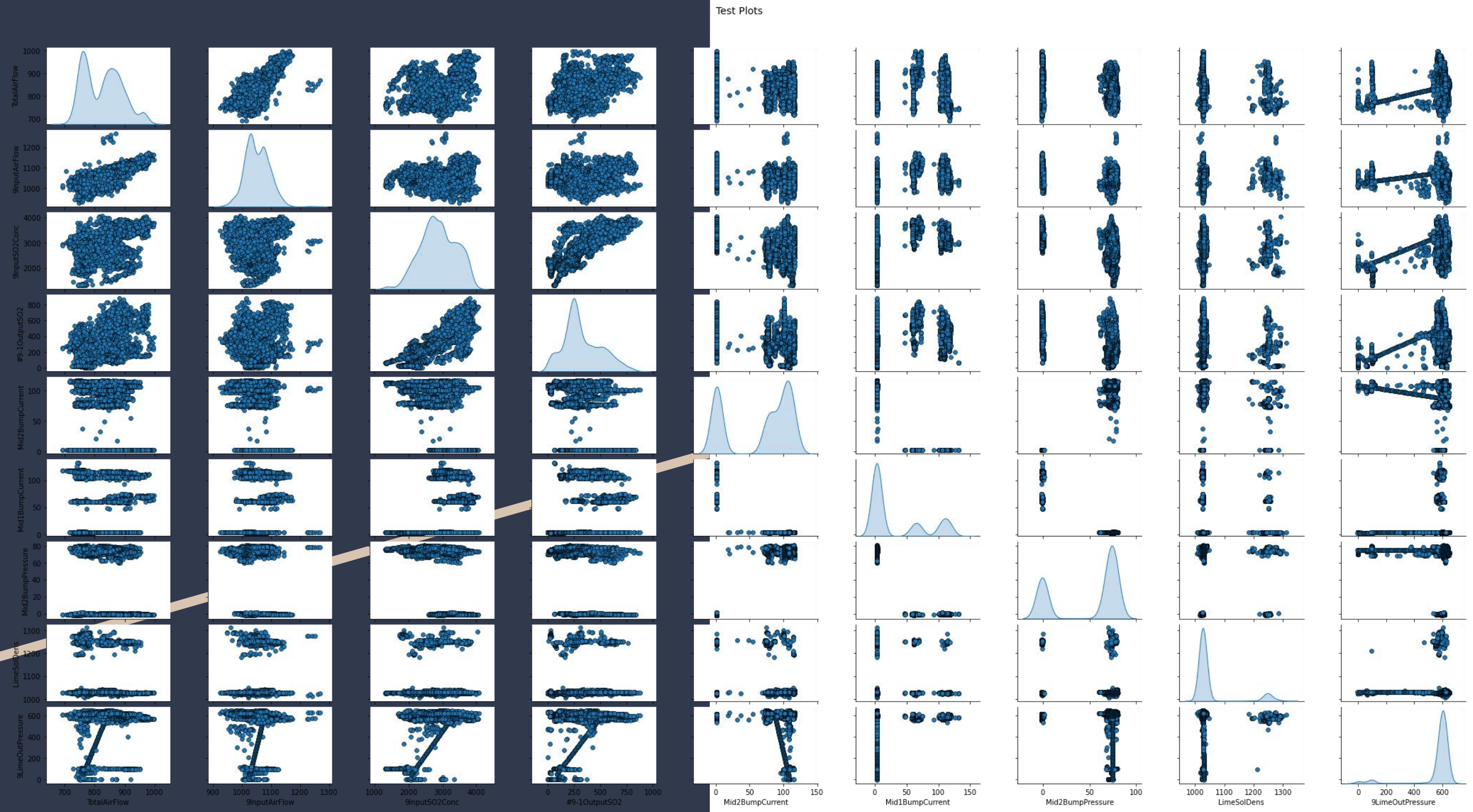
pre-selection

does not give useful information and would impact KDE performance

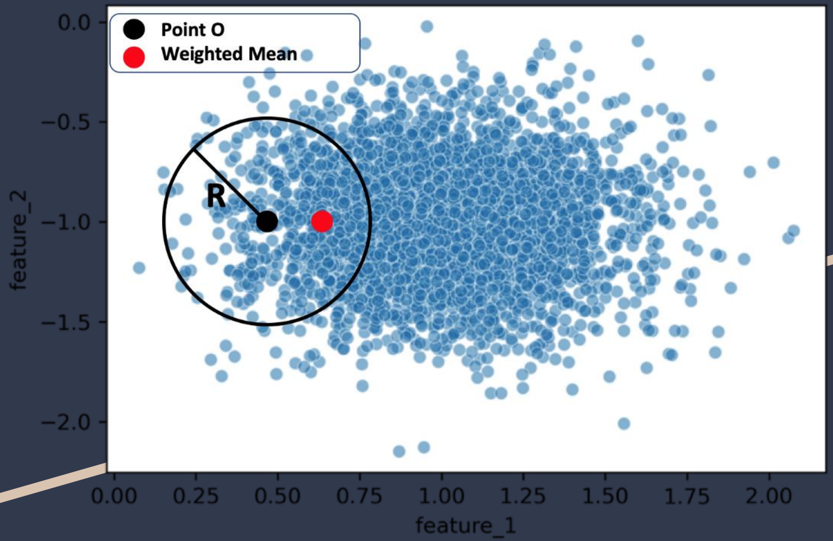


# Methods

post-selection



# Methods: Mean-shift Clustering




Density-based clustering

Weighted mean:

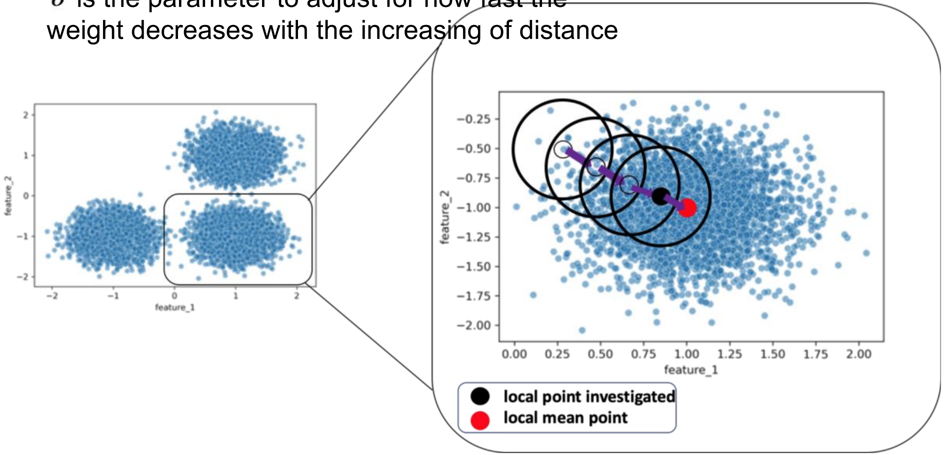
$$M_W = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

Using Gaussian weight function:

$$w(d) = e^{-\frac{d}{2\sigma^2}}$$

d is distance from the current point toward any other points.

$\sigma$ is the parameter to adjust for how fast the weight decreases with the increasing of distance



# Methods: Mean-shift Clustering

KDE is kind of similar to PDF

Then by gradient descent we could get cluster updating

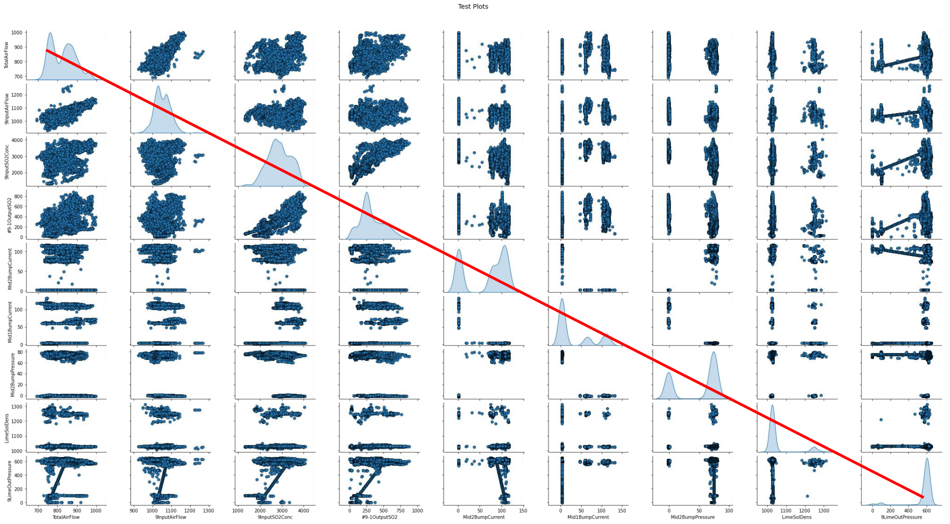
Density gradient estimator:

$$\hat{\nabla} f_{h,K}(\mathbf{x}) \equiv \nabla \hat{f}_{h,K}(\mathbf{x}) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x} - \mathbf{x}_i) k'\left( \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right).$$

- Kernel density estimator(KDE):

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left( \frac{\mathbf{x} - \mathbf{x}_i}{h} \right).$$

n is number of data,
h is bandwidth,
d is dimension of data,
K is the kernel function(we use Gaussian kernel).

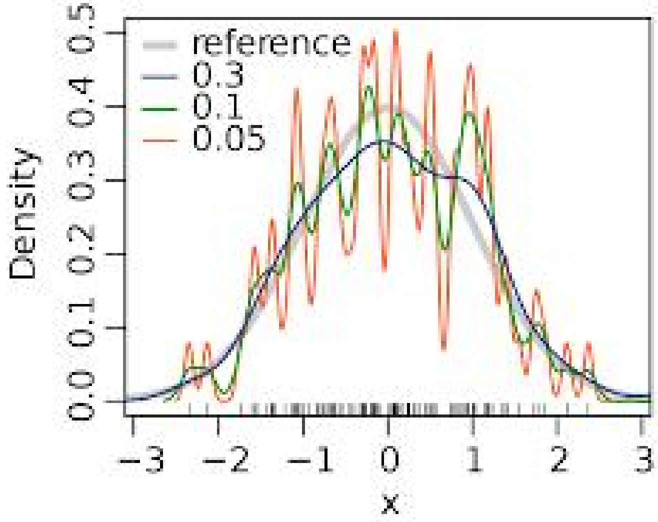


# Methods: Mean-shift Clustering

- Selection of bandwidth

Applying Silverman’s rule of thumb to self-select bandwidth since we are using Gaussian Kernel

$$h = \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} \cdot n^{\frac{-1}{d+4}} \cdot \sum_{0}^{n} \frac{\sigma}{n}$$




Under-smooth
vs
Over-smooth

Examples of different bandwidths on KDE

# Methods: K-means Clustering

From STAT 416 I learned about K-means clustering:

Define the Score for assigning a point to a cluster is

$$Score(x_i, \mu_j) = dist(x_i, \mu_j)$$

Lower score => Better Clustering.

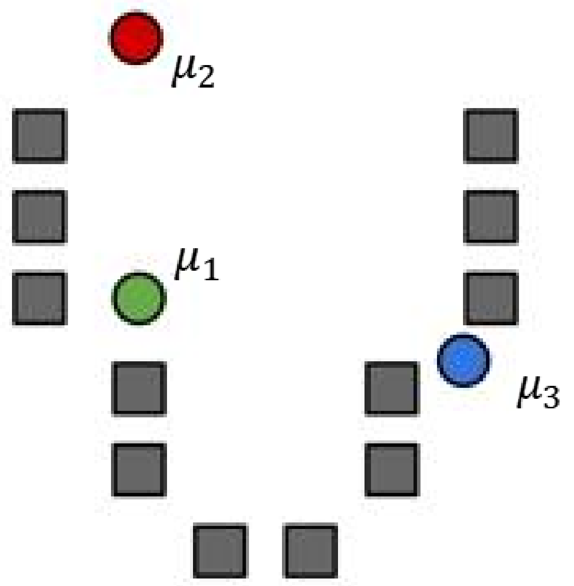
**k-means Algorithm at a glance**
Step 0: Initialize cluster centers

Repeat until convergence:
Step 1: Assign each example to its closest cluster centroid
Step 2: Update the centroids to be the average of all the points assigned to that cluster

# Step 0:

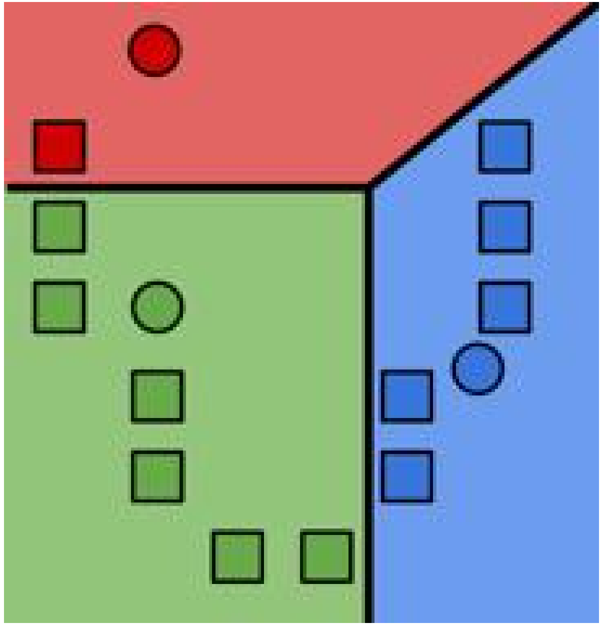
Start by choosing the initial cluster centroids

- A common default choice is to choose centroids at random
- Will see later that there are smarter ways of initializing



# Step 1:

- Assign each example to its closest cluster centroid
- $z_i \leftarrow \underset{j \in [k]}{\operatorname{argmin}} \left|\left|\mu_j - x_i\right|\right|^2$

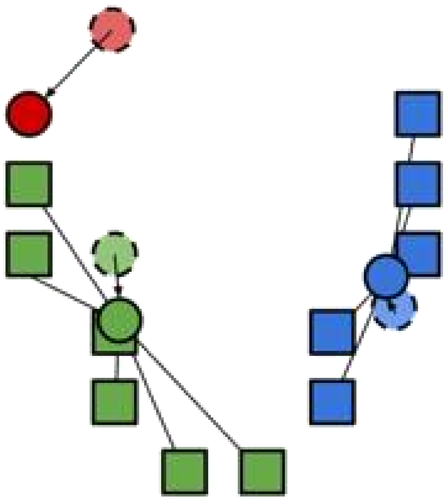


# Step 2:

Update the centroids to be the mean of all the points assigned to that cluster.

$$\mu_j \leftarrow \frac{1}{n_j} \sum_{i:z_i=j} x_i$$

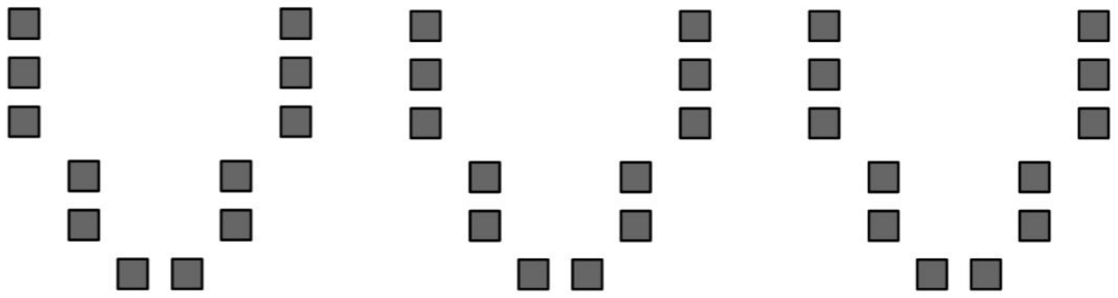
Computes center of mass for cluster!



# K-means++: Smart Initialization

**k-means++** does a slightly smarter random initialization

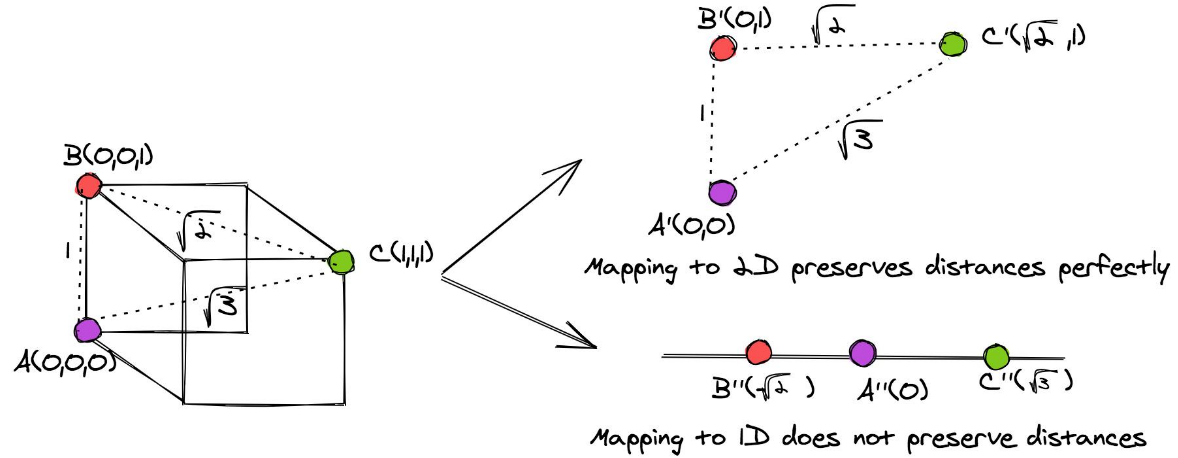
1. Choose first cluster $\mu_1$ from the data uniformly at random
2. For the current set of centroids (starting with just $\mu_1$), compute the distance between each datapoint and its closest centroid
3. Choose a new centroid from the remaining data points with probability of $x_i$ being chosen proportional to $d(x_i)^2$
4. Repeat 2 and 3 until we have selected $k$ centroids



# Methods: Multidimensional Scaling(MDS)

Multidimensional Scaling(MDS) gives a deduction in dimensionality so that we could have a better visualization while still keep the distance or pattern somehow.
We use Euclidean distance and metric MDS which preserve the pairwise distance here.

$$d(p,q) = \sqrt[2]{(q_1 - p_1)^2 + \ldots + (q_n - p_n)^2}$$




A possible mapping of points from 3D to 2D and 1D. The mapping is not optimized.

# Code Implementation

```python
def MS(mesh_0, data, h=None, eps=1e-7, max_iter=1000, wt=None):
    '''
    Mean Shift Algorithm with the Gaussian kernel.

    Parameters
    ----------
        mesh_0: a (m,d)-array
            The coordinates of m initial points in the d-dim Euclidean space.

        data: a (n,d)-array
            The coordinates of n data sample points in the d-dim Euclidean space.

        h: float
            The bandwidth parameter. (Default: h=None. Then the Silverman's
            rule of thumb is applied. See Chen et al.(2016) for details.)

        eps: float
            The precision parameter.

        max_iter: int
            The maximum number of iterations for the SCMS algorithm on each
            initial point.

        wt: (n,)-array
            The weights of kernel density contributions for n random sample
            points. (Default: wt=None, that is, each data point has an equal
            weight "1/n".)

    Return
    ----------
        MS_path: (m,d,T)-array
            The entire iterative MS sequence for each initial point.
    '''

    n = data.shape[0]   ## Number of data points
    d = data.shape[1]   ## Dimension of the data

    if h is None:
        # Apply Silverman's rule of thumb to select the bandwidth parameter
        # (Only works for Gaussian kernel)
        h = (4/(d+2))**(1/(d+4))*(n**(-1/(d+4)))*np.mean(np.std(data, axis=0))
        print("The current bandwidth is "+ str(h) + ".\n")

    MS_path = np.zeros((mesh_0.shape[0], d, max_iter))
    ## Create a vector indicating the convergent status of every mesh point
    conv_sign = np.zeros((mesh_0.shape[0], ))
    if wt is None:
        wt = np.ones((n,))
    else:
        wt = n*wt
    MS_path[:,:,0] = mesh_0
    for t in range(1, max_iter):
        if all(conv_sign == 1):
            print('The MS algorithm converges in ' + str(t-1) + 'steps!')
            break
        for i in range(mesh_0.shape[0]):
            if conv_sign[i] == 0:
                x_pt = MS_path[i,:,t-1]
                ker_w = wt*np.exp(-np.sum(((x_pt-data)/h)**2, axis=1)/2)
                # Mean shift update
                x_new = np.sum(data*ker_w.reshape(n,1), axis=0) / np.sum(ker_w)
                if LA.norm(x_pt - x_new) < eps:
                    conv_sign[i] = 1
                MS_path[i,:,t] = x_new
            else:
                MS_path[i,:,t] = MS_path[i,:,t-1]

    if t >= max_iter-1:
        print('The MS algorithm reaches the maximum number of iterations,'\
              +str(max_iter)+', and has not yet converged.')
    return MS_path[:,:,:t]
```

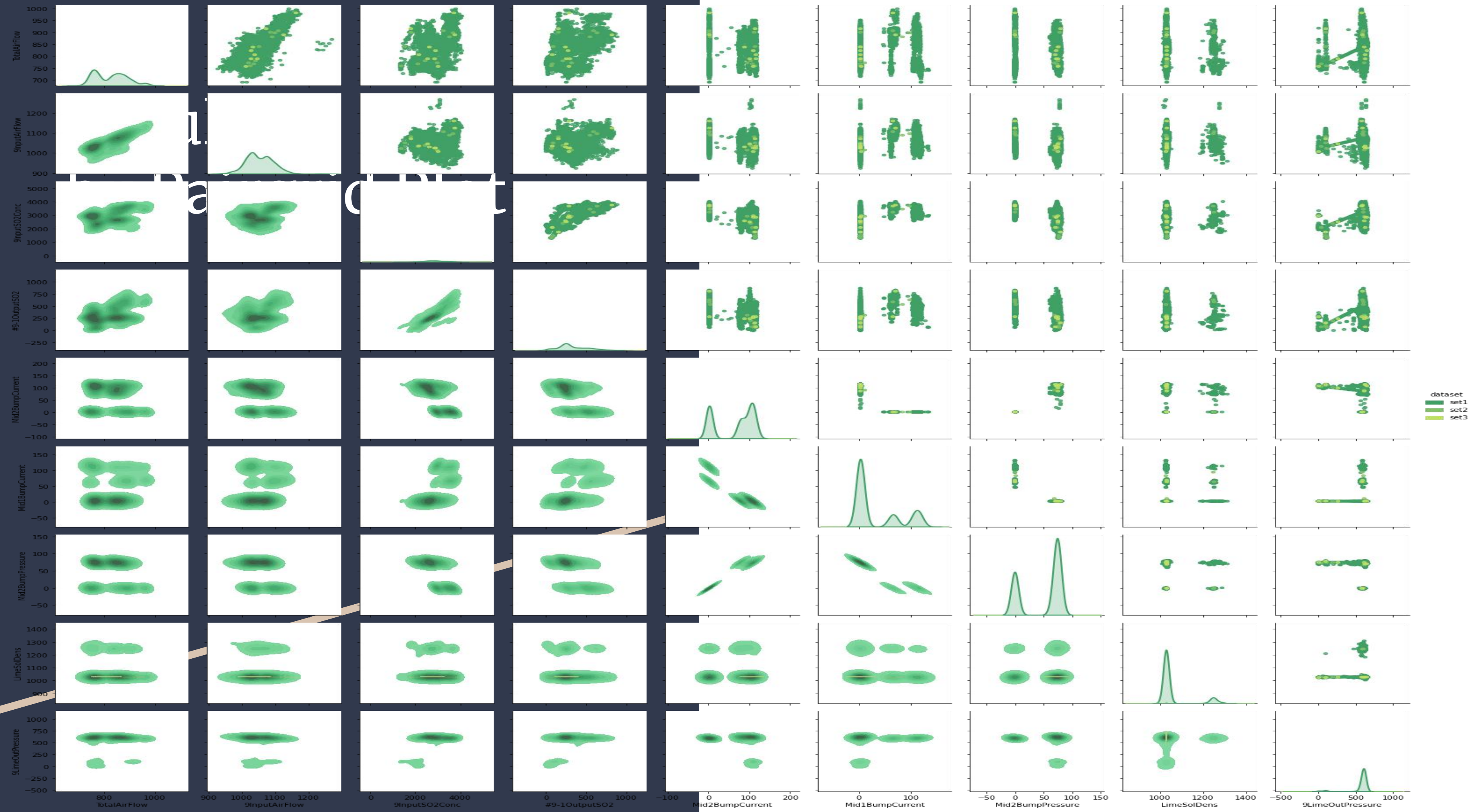
- Variable selection: sklearn.feature_selection.VarianceThreshold

In [7]:
```python
def smart_initialize(data, k, seed=None):
    """
    Use k-means++ to initialize a good set of centroids
    """
    if seed is not None: # useful for obtaining consistent results
        np.random.seed(seed)

    centroids = np.zeros((k, data.shape[1]))

    # Randomly choose the first centroid.
    # Since we have no prior knowledge, choose uniformly at random
    idx = np.random.randint(data.shape[0])
    centroids[0] = data[idx,:]#.toarray()

    # Compute distances from the first centroid chosen to all the other data points
    distances = pairwise_distances(data, centroids[0:1], metric='euclidean').flatten()

    for i in range(1, k):
        # Choose the next centroid randomly, so that the probability for each data point to be chosen
        # is directly proportional to its squared distance from the nearest centroid.
        # Roughly speaking, a new centroid should be as far as from ohter centroids as possible.
        idx = np.random.choice(data.shape[0], 1, p=distances/sum(distances))
        centroids[i] = data[idx,:]#.toarray()

        # Now compute distances from the centroids to all data points
        distances = np.min(pairwise_distances(data, centroids[0:i+1], metric='euclidean'),axis=1)

    return centroids
```

# Code Implementation: Plotting

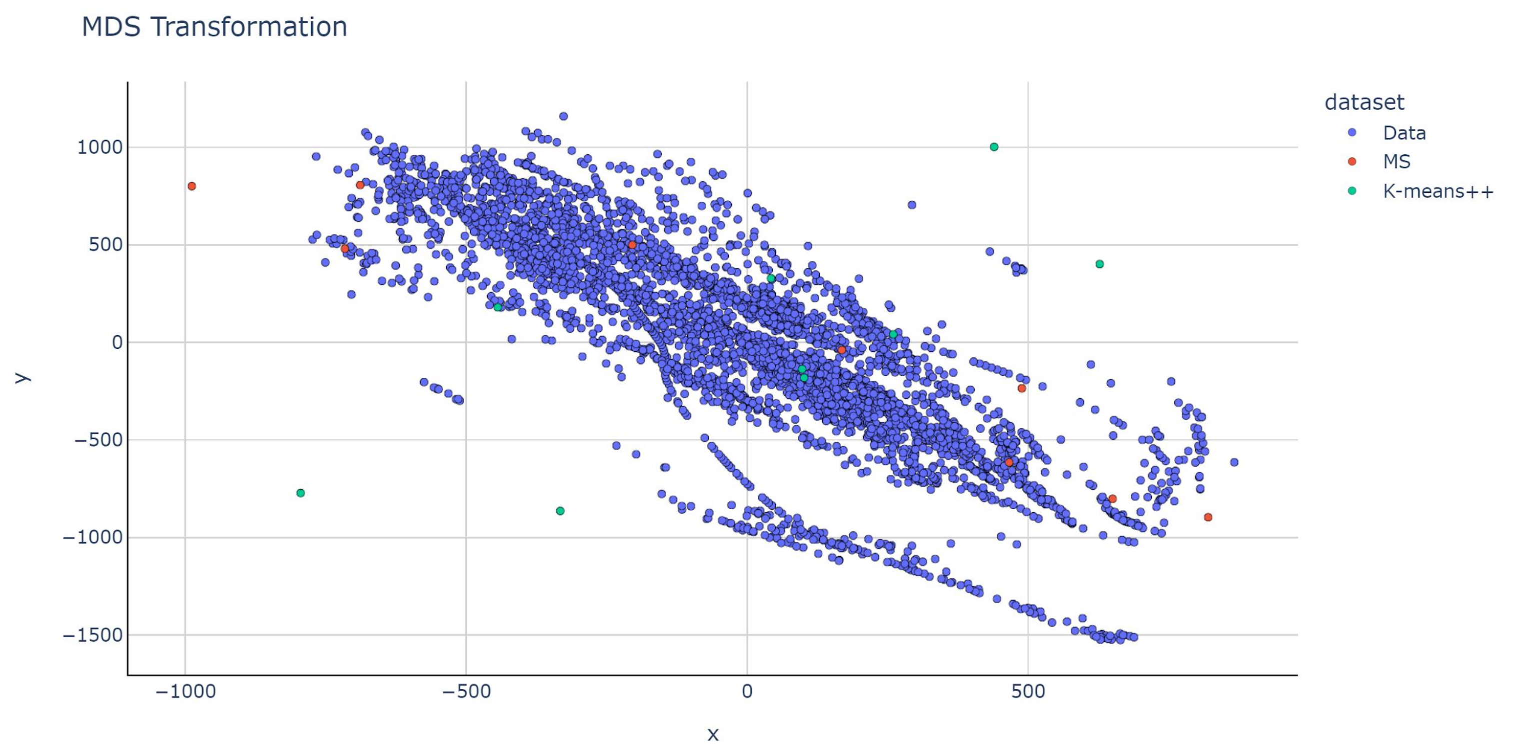
```
In [59]: from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns


dff = pd.DataFrame(transformed, columns = labels)
dfm = pd.DataFrame(oo[0], columns = labels)
dfk = pd.DataFrame(mm, columns = labels)

concatenated = pd.concat([dff.assign(dataset='set1'), dfm.assign(dataset='set2'), dfk.assign(dataset='set3')], ignore_index = Tru


'''pp = sns.pairplot(concatenated, height=1.8, aspect=1.8,
                  plot_kws=dict(edgecolor="k", linewidth=0.5),
                  diag_kind="kde", diag_kws=dict(shade=True),
                  hue='dataset')'''
grid = sns.PairGrid(data=concatenated, hue='dataset', palette = 'summer')
grid = grid.map_upper(plt.scatter, s=20, alpha=0.8)
grid = grid.map_diag(sns.kdeplot, fill=True, lw=2) #Gaussian Kernal
grid = grid.map_lower(sns.kdeplot, fill=True)
grid.add_legend()
#grid = grid.map_lower(sns.kdeplot, levels=4, color=".2")
#pp.map_lower(sns.kdeplot, levels=4, color=".2")


'''fig = pp.fig
fig.subplots_adjust(top=0.93, wspace=0.3)
t = fig.suptitle('Test Plots', fontsize=14)'''
```

```
import plotly.express as px

# Create a scatter plot
fig = px.scatter(cated, x='x', y='y', opacity=1, color="dataset")

# Change chart background color
fig.update_layout(dict(plot_bgcolor = 'white'))

# Update axes lines
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black')

fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black')

# Set figure title
fig.update_layout(title_text="MDS Transformation")

# Update marker size
fig.update_traces(marker=dict(size=5,
                              line=dict(color='black', width=0.2)))

fig.show()
```



TotalAirFlow
9InputAirFlow
9InputSO2Conc
#9-1OutputSO2
Mid2BumpCurrent
Mid1BumpCurrent
Mid2BumpPressure
LimeSolDens
9LimeOutPressure
dataset
set1
set2
set3

# Results: by MDS



# Discussion on Performance

- Time
- Who’s better?

# Performance: Time Complexity

```
Time of MS:240.09044551849365
Time of K-means:0.018936634063720703
```

Mean-shift algorithm:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right).$$

Time complexity of $O(tn^2d)$

For high dimension it is time-consuming.

K-means algorithm:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2$$

is a NP-hard problem that has complexity $O(tnd)$

where t is iteration times, k is number of centroids, n is number of data and d is dimension.

# Performance: Who's better?

- Silhouette Score
  - Separation between different clusters.

$$s = \frac{b - a}{\max(a, b)}$$

Not for DB

- Rand Index/Adjusted Rand Index
  - Calculate difference pairs between ground-truth clusters and model clusters.

$$ARI = \frac{\text{RI - Expected RI}}{\text{Max(RI) - Expected RI}}$$

No true label

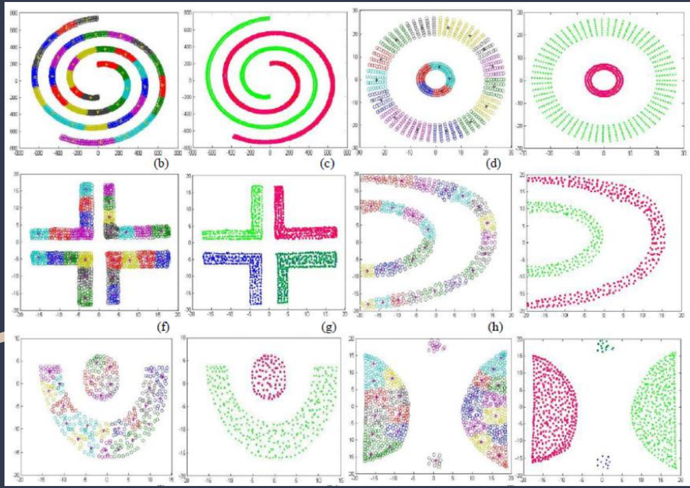
- Calinski-Harabasz Index(Variance Ratio Criterion)

$$s = \frac{\mathrm{tr}(B_k)}{\mathrm{tr}(W_k)} \times \frac{n_E - k}{k - 1}$$

DB has always higher score

# Performance: Who's better?

BUT Generally



- Pros of mean shift(compare to k-means):
  - Mean Shift is quite better at clustering as compared to K Means, mainly due to the fact that we don't need to specify the value of 'K', i.e. the number of clusters.
  - Output of mean shift is not strongly dependent on initialization
  - The algorithm only takes one input, the bandwidth of the window.
  - Has smaller limitation to shape of the data.
- Cons of mean shift:
  - Mean Shift performs a lot of steps, so it can be computationally expensive, with a time complexity of $O(n^2)$ while k-means is O(n)
  - The selection of the bandwidth itself can be non-trivial, although we introduce the silverman's rule.
  - The performance on high dimension would not be good since for MISE we have:

$$\int \mathbb{E}\left((\widehat{p}_{n,\mathrm{opt}}(x) - p(x))^2\right) dx = \inf_{h>0} \int \mathbb{E}\left((\widehat{p}_n(x) - p(x))^2\right) dx = O\left(n^{-\frac{2}{d+4}}\right)$$

# Further Discussion

- Silverman's rule performs well on normal density. Otherwise might be bad-performance.
- Running time for Mean-shift is too long. Introducing parallel programing to be faster.
- Do not have a clear physical explanation of found: what do clusters of data patterns do?
- Performance measurement is not solved. Maybe could try Davies-Bouldin Index for non-convex clusters.
- Do not have ground-truth labels on data set. Introducing labels would be better for quantitative measurement.
- More

# Index

Questions?

- Introduction
- Materials and Methods(data and methods)
- Code Implementation
- Results(graphs by pairgrid and MDS)
- Discussion(Performance)
- References

# References


Sheth, Jash. “Mean Shift Clustering Algorithm.” OpenGenus IQ: Computing Expertise & Legacy, OpenGenus IQ: Computing Expertise & Legacy, 6 Apr. 2019, https://iq.opengenus.org/mean-shift-clustering-algorithm/.

6.869 Advances in Computer Vision: Learning and Interfaces, https://courses.csail.mit.edu/6.869/.

Yufeng. “Understanding Mean Shift Clustering and Implementation with Python.” Medium, Towards Data Science, 22 Feb. 2022, https://towardsdatascience.com/understanding-mean-shift-clustering-and-implementation-with-python-6d5809a2ac40.

Zuccarelli, Eugenio. “Performance Metrics in Machine Learning-Part 3: Clustering.” Medium, Towards Data Science, 31 Jan. 2021, https://towardsdatascience.com/performance-metrics-in-machine-learning-part-3-clustering-d69550662dc6.

Landup, David. “Guide to Multidimensional Scaling in Python with Scikit-Learn.” Stack Abuse, Stack Abuse, 21 July 2022, https://stackabuse.com/guide-to-multidimensional-scaling-in-python-with-scikit-learn/.

Chen, Yen-Chi. “A Tutorial on Kernel Density Estimation and Recent Advances.” ArXiv.org, 12 Sept. 2017, https://arxiv.org/abs/1704.03924.