	DBSCAN	HDBSCAN	Spectral	K-means	Hierarchical	Mixture Models	Fuzzy c-means	Mean Shift	Affinity Propagation	Latent Class Analysis (LCA)	Bisecting k-means	Self-Organizing Map (SOM)
Description	Density-Based Spatial Clustering of Applications with Noise, a non-parametric density-based clustering algorithm More Info: https://en.wikipedia.org/wiki/DBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise, performs DBSCAN with varying epsilon values allowing for varying densities (DBSCAN does not allow for varying densities) More Info: https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html#hdbscan	- Spectral clustering uses the eigenvalues of the similarity matrix of the data to perform dimensionality reduction prior to clustering - Spectral clustering takes the similarity matrix as an input - The similarity matrix is a matrix of quantities representing the relative similarity of each pair of points in the dataset More Info: https://en.wikipedia.org/wiki/Spectral_clustering	- Starts with k random centroids (k is user defined) and assigns each observation to the centroid closest to it - After the initial assignments the algorithm iteratively calculates the new k centroids and then assignes each observation based on the new centroids - This process conitnues until convergence is achieved (or a minimum amount of changing) More Info: https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm_(naive_k-means)	- There are two forms of hierarchical clustering - Agglomerative: When each observation starts as its own cluster and clusters are merged one by one by joining the closest clusters - Divisive: When all observations start in one cluster and clusters are formed through recursive splitting of the observations More Info: https://en.wikipedia.org/wiki/Hierarchical_clustering	- A probabilistic method of clustering observations by probability distributions - Most common is using Gaussian Mixture models to group observations according to different Gaussian distributions More Info: https://en.wikipedia.org/wiki/Mixture_model	- Fuzzy clustering is like k-means clustering except it allows observations to belong to more than one cluster at a time More Info: https://en.wikipedia.org/wiki/Fuzzy_clustering	"The underlying idea of the Mean Shift algorithm is that there exists some probability density function from which the data is drawn, and tries to place centroids of clusters at the maxima of that density function [4]."	- Finds 'exemplars' by esentially letting each observation 'vote' for their prefered 'exemplar' [11] - The 'exemplars' are members of the input set that are representative of clusters [12] More Info: https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html#affinity-propagation https://en.wikipedia.org/wiki/Affinity_propagation	- This isn't actually clustering, it would be an alternative (more research required into if this is viable for our problem) More Info: https://docs.displayr.com/wiki/The_Relationship_Between_Cluster_Analysis,_Latent_Class_Analysis_and_Self-Organizing_Maps https://en.wikipedia.org/wiki/Latent_class_model	- A hybrid of k-means and Divisive Hierarchical clustering - Starts with one cluster and breaks it into 2 sub clusters at each step until k-clusters have been achieved More Info: https://medium.com/@afrizalfirbisecting-kmeans-clustering-5bc176030b8a2	- An ANN trained using unsupervised learning (essentially organizes observations based on the similarities of their values, is very easy to generate a plot that shows the breakouts) - uses competative learning instead of error-correction learning
Pros	- Does not require the user to specify the number of clusters [3] - Can find arbitrarily shaped clusters [3] - Takes into account noise and is robust to outliers [3] - Few input parameters and mostly insensitive to order of points in the dataset (edge cases between clusters can sometimes have issues) [3] - Designed for use with databases that can accelerate region queries (not sure what that means but could be relevant for us???) [3] - If the data is well understood the parameters minPts and epsilon can be set by a domain expert [3]	- Same as DBSCAN (but with fewer cons) - Can cluster datasets that have different densities	- Cluster structure better than k-means but still not ideal [4]	- Simple to implement [2] - Scales to large data sets [2] - Guarantees convergence [2] - Can warm-start the positions of centroids (I think this means that you can tell the - algorithm where to set the initial centroids?) [2] - Easily adapts to new examples [2] - Generalizes to clusters of different shapes [2] - Computationally efficient - Often provides clearer groups than HC-	- Easy to understand and implement [1] - Has a method of empirical method of identifying the number of groups	- Mathematically and statistically sound - Can handle data with outliers/skewness through non-Gaussian densities - Parameter estimate/model-fitting built into many clustering algorithms - Can look at statistical inferences like hypothesis testing - Provides a probabilistic answer to whether an observation belongs in a given group	- Gives good results for overlapped data sets [5] - Allows data points to belong to more than one cluster [5] - Performance can be improved with tuning parameters (at the cost of computation time) [5]	- Identifies the number of clusters on its own [4] - Accounts for noise observations [4] - Input parameters are more intuitive than k-means (meaning you can figure it out rather than just having to try a bunch of different ones) [4]	- Dont need to choose k [11] - Allows for non-metric dissimilarities [11]	- Can work with many different data types (doesn't all need to be numeric) [13] - Can deal with missing data (allocates observations based on the available data) [13] - Can work with weights [13] - Is an actual statistical model, and therefore is backed up by statistical theory [13] - "Can be modified to incorporate lots of varied phenomena" [13]	- Random centroid generation doesn't occure (like k-means) [10] - Deterministic optimizer (will get same results every time you run it) - Similar time and space complexity to k-means [10]	- Can be used for classification and variable reduction (output is 2D by design as it is intended for plotting) [14] - Doesn't need an estimate of the number of classes to be passed in [15] - Can deal with ambiguity and assignment of points to multiple clusters [15]
Cons	- Not entirely deterministic (border points between clusters can depend on the order the data was processed) [3] - The quality of the clustering depends on the distance measure used (Euclidean distance can have issues in high dimensional data) [3] - Does not cluster datasets that have large differences in densities well [3] - If the data and scale are not well understood, choosing a meaningful distance threshold epsilon can be difficult [3]	- The minimum cluster size is difficult to determine (but the algorithm is robust to this parameter so not a huge deal) [4] - The quality of the clustering depends on the distance measure used (Euclidean distance can have issues in high dimensional data) [3] (inherited from DBSCAN)	- Includes noise observations in clusters [4] - Number of clusters is user defined [4] - Different starting values can give different groups (similar stability issues as k-means, but slightly better) [4] - Slow computation [4] - Not great for 'long thin' clusters [4]	- Different k's can give different groups (local optimizer not global) - Groups will be found no matter what (even if there are no groups) - Number of groups is user defined (need to test various different k's) - Has troubles with data of varying sizes and densities (clusters will be uniformly sized) [2] - Outliers can cause problems (move centroids or assigned own clusters) [2] - Has issues with data with high dimensionality [2]	- Can be very time consuming for very large samples - Results are often sensitive to what distance type and what linkage method are used - Rarely gives best solution [1] - Can give poor solutions without it being apparent (particularly with high dimensional data) [1] - Doesn't work well with large datasets or mixed data types [1] - This is why we used distance measures in Data 573 - Does not work with missing data [1]	- Is a deterministic monotonic optimization algorithm - Susceptible to local maxima - Hard to find optimal configuration - Often overconfident in clustering probabilities - Running with different seeds is likely to change results - Computationally intensive - Model selection method isn't ideal (currently use BIC) - Not much info available online (at least not on the pros and cons)	- Requires number of clusters to be specified by user [5] - The method of improving results (reducing the beta value) requires more iterations (can improve performance at the cost of time) [5] - Euclidean distance measures can unequally weight underlying factors [5]	- The sklearn implementation is quite slow (but in theory can be fast) [4] - I couldn't find any other libraries, so unless we want to code it ourselves this is a slow algorithm as far as I am concerned - Without visual validation it is hard to know if your results are reasonable [4] - Random initialization meaning that results can change with different starting seeds [4] - Not robust to the "bandwidth" parameter (meaning difficult to tune and results can vary significantly) [4]	- Is slow (source says that it is "essentially impossible without a carefully crafted and optimized implementation" when used on large datasets) [11] - Picking tuning parameters can be difficult [11] - Observations will be put into clusters even if they don't belong to any cluster (ex. noise values) [11] - Assumes a globular structure (won't capture irregularly shaped clusters appropriately) [11]	- Slow relative to other methods (especially k-means) [13] - Not very easy to generate visualizations from (many of the clustering algorithms have this issue to it working in high dimensional data) [13]	- User defined k is still required [10] - Could probably determine k from testing different numbers and looking at the WCSS like determining k for k-means	- Can take a long time to train (particulary when there is a large amount of data) [15]
Packages	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html - ddbscan (MIT License): https://pypi.org/project/ddbscan/	- HDBSCAN (3-clause BSD licensed): https://pypi.org/project/hdbscan/	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering - Scipy (BSD License): https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/mixture.html# - Gaussian and Bayesian	- Scikit-fuzzy (3-clause BSD license): https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html	- scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html	- Scikit-learn (BSD License): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html	- lccm (Modified BSD (3-clause) license): https://pypi.org/project/lccm/ - I think latent class choice models are the same as latent class analysis (the descriptions sound close to me at least)	- pyclust (GPL-2.0 License): https://pypi.org/project/pyclust/	- MiniSom (Creative Commons Attribution 3.0 Unported License): https://pypi.org/project/MiniSom/ - Can do anything we want with this package, just need to attribute it to the author is some reasonable way (I think we could just reference it in our GitHub readme)
In MLlib?	No	No	No	Yes	No - Has Bisecting k-means which is similar to Hierarchical	Yes (Gaussian only)	No	No	No	No	Yes	No
Notes	- Started coding	- Started coding	Can be up to O(n^3) so probably not a good idea, from what I have read DBSCAN provides similar results but is much faster and is therefore generally prefered - Not going to code unless we decide we need to (everything I have read says that this method is extremely slow, so probably not worth investigating unless we need it)	- Started coding	- Function in the scikit-learn library is 'AgglomerativeClustering()' but the default connectivity is for hierarchical clustering - Started coding	- All pros and cons (except for the last one) are from my notes from Data 573, couldn't find any pros and cons online - Started coding (both Gaussian and Bayesian)	- Allows a data point to belong to more than one cluster (could be useful for working with items that may have the same end use but different UOMs, ex. Temperature and pressure could actually be providing the same end use) - Looks like it is pretty much just k-means but allows observations to belong to multiple clusters - Started coding	- Started coding	- This is probably not right for our data due to scalability concerns, but if the other methods aren't giving us good results, the fact that it allows for non-metric dissimilarities could help us - Not going to code unless we decide we need to (everything I have read says that this method is extremely slow, so probably not worth investigating unless we need it)	- Doesn't seem to be a well supported algorithm in Python but looks like there are several implementations in R, if we wanted to go with this algorithm we could look at the possibility of calling an R script from our dataframes file (not sure if this is a realistic option though) - Not going to code up, likely not beneficial for us to consider (almost seems more like a FA alternative, it is not a clustering method, so this probably should have been included in the "Feature Mapping" tab)	- The pyclust library has a copyleft license and would require that everything we do is published under the same license so i don't think we should use it - The above issue doesn't apply if we choose to go with the MLlib implementation - Not going to code unless we really want to use this one (has a copyleft license which is less than ideal for us)	- I have used the MiniSom package and it was extremely simple to implement in regular Python (haven't tried in databricks though) - Not going to code right now. Doesn't give us cluster values, likely only usefull if we want to visualize similarities between observations in a 2-d space

References

[1]	https://www.displayr.com/strengths-weaknesses-hierarchical-clustering/
[2]	https://developers.google.com/machine-learning
[3]	https://en.wikipedia.org/wiki/DBSCAN#Advanta
[4]	https://hdbscan.readthedocs.io/en/latest/compa
[5]	https://sites.google.com/site/dataclusteringalgor
[10]	https://shodhganga.inflibnet.ac.in/bitstream/10603/209167/12/12_%20chapter%204.pdf
[11]	https://hdbscan.readthedocs.io/en/latest/compa
[12]	https://en.wikipedia.org/wiki/Affinity_propagation
[13]	https://docs.displayr.com/wiki/The_Relationship_Between_Cluster_Analysis,_Latent_Class_Analysis_and_Self-Organizing_Maps
[14]	https://www.sciencedirect.com/science/article/pii/S235291481930098X
[15]	https://homepage.cs.uri.edu/faculty/hamel/pubs/advances-som.pdf