Final Project Report

Selected Topic	Topic 3	
Group Membership	**Student ID**	**Name**
Member 1	**17216133**	**Mathew Kai Yin Wong**

Introduction:

Background:

The topic that I chose to analyze for this project was the third topic available, "*K-means and Density Peak Algorithms Implementation with Visualization Interface"*. This topic particularly piqued my interest because of a variety of reasons:

1. As a computer science student, I relished at the opportunity to develop software and improve my skills.

2. Density Peak and K-means are two algorithms I find very interesting, and I liked the opportunity to do more research on the topic.

3. Visualization is an extremely powerful learning tool, and having used them extensively throughout my studies in this topic, the idea of creating one intrigued me.

4. I saw this as a great opportunity to use tools/frameworks that I knew about but never had the opportunity to use.

Tasks:

The high level tasks that I outlined in order to complete this project were fairly straightforward:

1. **Prepare:** In my past experiences in working on software projects, I realized that setting clear expectations and use cases is **essential** in order to make progress. I made sure to read over the assignment description and asked questions.

2. **Research:** Research was mainly concerned with two parts:
   - ➢ Algorithms: Although these algorithms were presented in class, it was important to gain a deeper knowledge of the logic and operations behind them.

   - ➢ Visualization Interfaces: As the project involved building out a visualization interface, much research was done on this topic to gain inspiration and build a direction when it came to creating the design. It appears that many visualization interfaces have been created before for the k-means algorithm [1], [2], and it was very helpful to take inspiration on the design of this application from them.

     Surprisingly, it was difficult to find a popular visualization interface for the density peaks algorithm. This served as extra motivation to create this project, as it was an opportunity to build something that few people had delved into before.

     Although there were many articles delving into comparisons of clustering algorithms [3],[4], there didn't appear to be an easily accessible one available like the aforementioned one by Karanveer Mohan. This was also possibly something else that could be new.

3. **Design:** After the research part of this project, I set out to design the

system with inspiration from the various sources that I mentioned above. The details of this part will be discussed in more detail later on in this report.

4. **Implementation:** The majority of the work of this project was spent on this implementation part. This will be elaborated on more in detail further on in the report.

Related Work

Existing Systems:

In doing the preliminary research for this project, I went through many different visualization systems to learn common practices that I could apply onto my own. In the end, the existing visualization system that I ended up taking the most inspiration from was Karanveer Mohan's visualization which is available online [1]. There are multiple reasons why I decided to use his visualization system as the basis for my design:

1. **Design:** I feel that Mohan's visualization system is very effective due to its simplistic design. It is a one page app that is very organized, without too many parts to distract the user. In a data visualization system, the visualization should be the main item presented to the user, and too much visual clutter in extras can confuse users and hinder effectiveness of the tool. Mohan's design of his user interface is very good, with the scatterplot area taking up a significant amount of screen space, and I chose to emulate the layout that he used in my own design as much as possible.

One particular aspect I enjoyed about his design choices was his use of visual marks and channels to convey information. Mohan used large triangles of distinct colours to represent centroids which was extremely effective at not only differentiating between them and the other points, but

also from the rest of the content as the colours are extremely noticeable on a simplistic white background that he has chosen to use. In regards to the regular data points, Mohan kept them small in size and with an unassuming black colour as they would form clusters which are already very differentiable to the user. Any extra channels used will presumably add more mental load to the user. A bonus effect is that this will draw users' attentions first to the centroids, and not the individual data points.

2. **Implementation:**

   Mohan implemented his visualization with three parameters available for manipulation: Randomness, number of clusters, and number of centroids. These three parameters are definitely important for implementing a k-means visualization interface, as they greatly influence the results that will be generated from one instance. I particularly liked that in his interface, the user is able to select how clustered or random the data is in order to see how effective the algorithm is not just in ideal (clustered) environments, but also random environments with high levels of noise.

   The simplicity of Mohan's design was brought into the implementation of the interface logic as well, as the only interaction required to bring his application to life is clicking a button. The user is expected to continually click the button until the algorithm stops when it hits convergence.

3. **Online Availability:**

   Availability was a major factor for me in choosing Mohan's interface as a basis for my own design. Every software development project starts with a crucial decision of the framework, languages, and tools that will be used to implement the system. Online systems are extremely effective because of ease of access. Javascript may not be as powerful or widely used as Python when it comes to data analysis and statistical

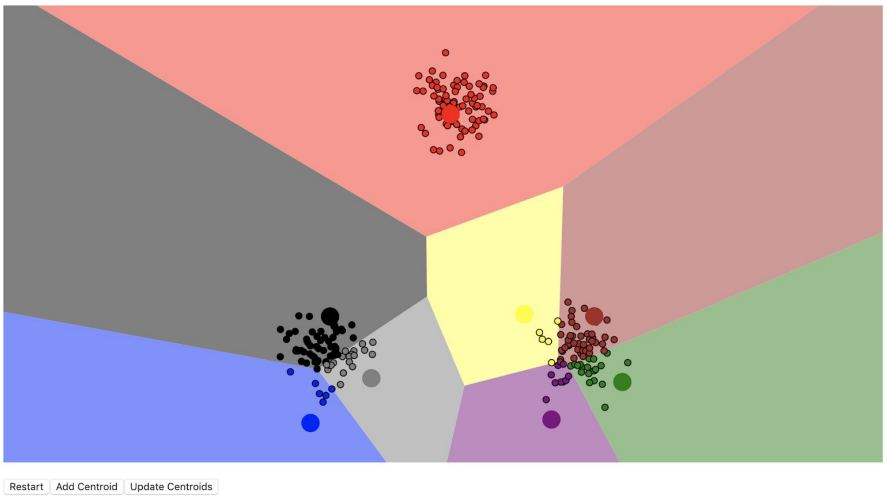
tasks, but it still comes with many great frameworks and tools for working on such problems. Mohan's interface and its ease of access (just through a Google search) convinced me that I should create my interface online. Although there are a plethora of visualization interfaces available, many of whom are much more powerful than Mohan's system, the ease of access online is a very attractive option compared to others where there may be frameworks to download or files to install just to begin using it.

4. **Other Inspirations:**

Aside from Mohan's interface, I also took ideas from other sources when coming up with my design:

Tableau: Tableau is an extremely powerful tool, and it was a great experience using it during this course. Although it is a program that is much more complex and feature-packed than the system I planned to develop, I did take some ideas from their UI design in terms of how to organize control panels with many options in order to not confuse the user.

Naftali Harris's k-Means Visualization [2]*:* Although Harris' page looks more like a blog post on k-Means clustering, it does come with a visualization interface that is much more simple than Mohan's but still gets the job done. What I really liked was the simplicity of her design, and the use of colour-coding when picking centroids. She also provides a variety of different-shaped data that the algorithm can be run on, which provides for interesting analysis.



Designed System

Background:

The system that I chose to develop is a data visualization web-app built with Javascript using mainly the Chart.js framework for graphs and other web tools such as Bootstrap and jQuery to build out my website.

I chose these technologies to carry out my project in order to satisfy three key requirements that I deemed most important during the process:

1. **Ease of development:** As I was working on this project alone and time was limited due to my commitments in other courses, I wanted to make sure that whatever choices I made in this regard would allow me to develop the system rather smoothly without too much effort needed to learn a new technology and without too much time spent playing around with it just to understand enough to get started.

Creating interactive graphs, especially on the web is also a very complex task, so a framework that could help complete that process was very much needed. Therefore, a language that had many tools and options available was of utmost importance.

2. **Interactivity:** As someone who relies a lot on visualization interfaces for learning purposes, I recognize how crucial interactivity and animations are for a good user experience. Because of that, I sought out tools with a good breadth of animation and interactivity support.

3. **Ease of use:** With inspiration from the sources mentioned above, I really tried to focus on creating a user-friendly experience. Therefore, any tool or platform that was web-oriented was much more attractive for me to use.

In the design process, I had a dilemma between making this project Python based or Javascript based. Python was a very attractive choice, as it is currently the most popular programming language when it comes to tasks involving mathematics and statistics due to its numerous frameworks and tools for this field such as Numpy. However, because I wanted to make a web-based application, and because of Chart.js, I decided to set out on this project using Javascript.

Chart.js became a logical choice for me after coming up with the list of requirements that I wanted to satisfy in this project. Chart.js is a robust graphing Javascript framework that I previously was interested in using, but never had a chance to. This appeared to be a logical time to try using it. Not only is Chart.js easy to understand, it also offers great support for animations which I tried to make use of in my visualization interface. Most importantly, it is completely web-based, making it a powerful platform that can create easy to use apps.

Algorithms:

The two algorithms I attempted to visualize in this project were the **k-Means Clustering** algorithm and the **Density Peak Clustering** algorithm.

k-Means Clustering:

k-Means Clustering was the easier of the two algorithms to implement in the system, as it was fairly straightforward with lots of resources available as reference.

k-Means Clustering works by first assigning nodes to random points on a dataset scatterplot, and assigning points to their closest centroid in order to form clusters. The centroids are realigned to the “average” of its cluster, and points are reassigned if a new centroid becomes closer. The process continues until *“convergence”*, which is when the points cease to change clusters [5], [6].

This algorithm is implemented in my visualization interface with 5 parameters: **k**, **n, Randomness,** and **Clusters**.

k represents the amount of starting centroids that the user can start with, which is limited to 2-5.

n represents the amount of nodes if using a randomized dataset. The range is set to 50-1500.

Randomness is between 0.1-1 which controls how randomized/clustered the data is. This was added with inspiration from Karanveer Mohon’s interface [1], as it was very interesting and effective to learn from, allowing the algorithm to be used in different types of datasets. Changing the randomness value will immediately change the dataset to reflect the change.

Clusters sets the amount of clusters the randomized dataset will contain. Users are free to mix and match k and the amount of clusters in the data to see the results.

Centroids can be moved interactively on the screen by the user through dragging. A chart on the right will show the user the location of the chosen

centroid in the plot area.

In my visualization system I have given the centroids distinct colours and a bordered rectangle shape to ensure that they can easily be distinguished by users.

The procedure I have implemented for this algorithm is simple. First, the user can adjust the different parameters I have described above. When the user is ready, clicking the Start button will activate the algorithm. The change is immediately seen, as the clusters will be formed and colour-coded, while the centroids are moved as their coordinates are recalculated to adjust for the changes. In my system, I have decided to have the interface automatically run until convergence. A counter on the right keeps track of the number of iterations required to reach the final result shown.

Density Peak Clustering:

Density Peak Clustering works on two basic principles:

1. Cluster centers usually have high **local density**
2. Cluster centers are usually very far from other high density points [7].

The idea is that once you find the points with the highest local densities in the set, the densest ones furthest away from higher density points should be good candidates to be cluster centers. There is only one iteration of this algorithm, as after the cluster centers are found all points are assigned immediately to their closest cluster. There is no readjustment like in k-means.

The parameters for this algorithm are very similar to those that I implemented for the k-means algorithm. The only difference is the **d** parameter which controls the cutoff for including a point in another point's "cluster radius".

In implementing this algorithm, I first chose to implement the **sweeping line algorithm** in order to get local densities of each point. I came to this idea as I

am currently studying this algorithm in my algorithm design course, and I thought it'd be a good idea to implement it here, as calculating local densities does not require the the interaction with one point and every other point in the set to be calculated. With the sweeping line algorithm I implemented, I set a buffer between two lines with the width being **d** from the "active point" during each iteration of the algorithm. Any points within range in the y-axis of the point is automatically added into a binary search tree. This cuts out unnecessary work, as for each active point, only those left and right of the active point within **d** needs to be checked.

In my implementation, the sweeping line process is shown, and the densities of different points are colour marked as the line sweeps through. After getting through the entire set, the centroids are shown and the clusters are colour-marked.

One major challenge in the implementation of this algorithm was finding a way to deal with multiple points that had the maximum density amount. This would cause centroids to erroneously be placed in the same cluster, throwing off the algorithm. The way I chose to fix this problem was to filter the dataset by density, leaving only the top 50%, and from there making sure that any candidate chosen as a centroid must not be within a distance d from another centroid.

Comparison Mode (Unfinished):

I implemented a comparison mode in the interface with the goal of creating a mode where both algorithms could be visualized side-by-side giving the viewer a direct comparison between the two. Currently, it is activated, and the two graphs are visible but due to a lack of time I have not been able to complete this part.

Data Description:

As part of the visualization interface, I have included three datasets that can be selected:

Random: This is a completely random dataset that can be controlled through the parameters discussed in the previous section. The dataset is generated through a basic normal distribution algorithm (Can be found in the codebase as function *normalDist()*). The algorithm takes in variance as a parameter, and to generate values it simply generates a random number using Javascript's Math.random() function and recursively adds to itself. In order to generate clusters, the algorithm takes in the **randomness** parameter set by the user to divide the different data points into different areas of the plot area.

Spiral: This is a spiral shaped dataset that I borrowed from a Python density peak clustering visualization app created by a Github user named lanbing510 [8].

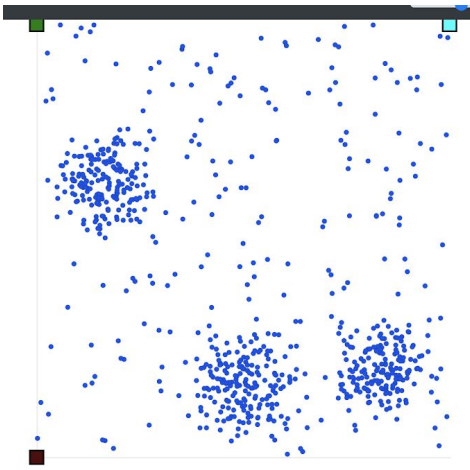
NBA: I borrowed this dataset to use as my real-life dataset from a Kaggle user Omri Goldstein [9] (https://www.kaggle.com/drgilermo/nba-players-stats) who scraped if from Basketball-reference.com, which is a very famous database website for basketball. My interface implements a subset of this file, with all players from 2017, and I used as the two attributes "3-point shooting percentage" and points scored for that year.

Experimental Results:

k-Means

Different initial points:

As part of this experiment, I put the initial points at two extremes: First at the corners of the screens:



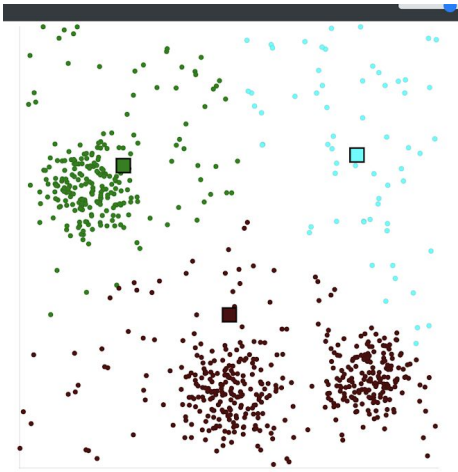
k Means

n = 750

k = 2

Iterations: 0

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0	0	0	0
	2	1	1	0	0
	3	0	1	0	0



k Means

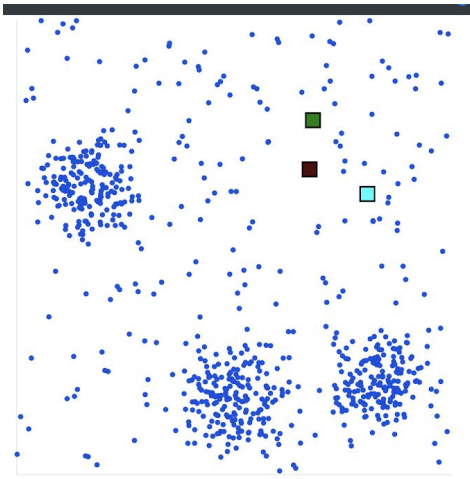
n = 750

k = 2

Iterations: 6

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.5019884140261321	0.34611648998764455	445	0
	2	0.8066658782357833	0.7082341382408912	63	0
	3	0.24879868306938194	0.6841904757718343	242	0

Then I put them closer together...



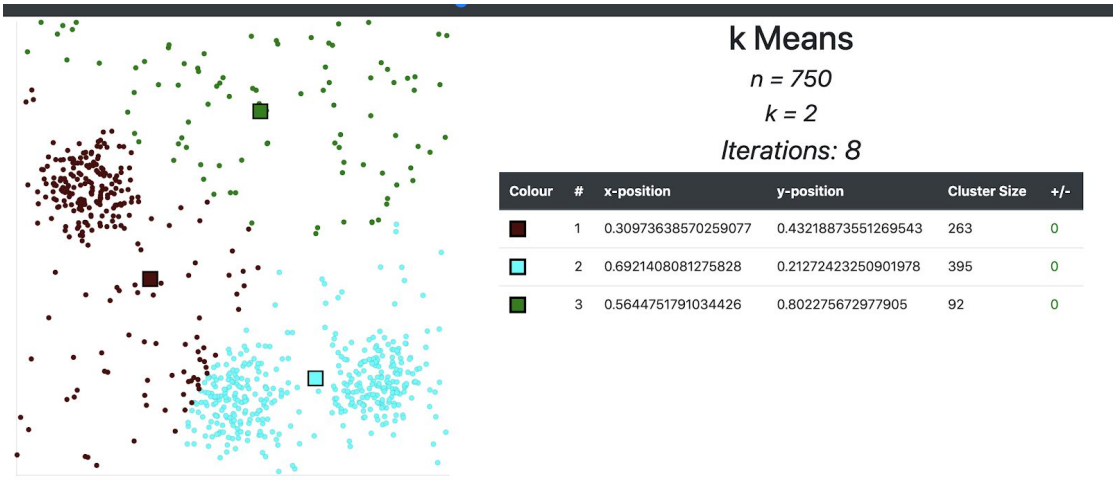
k Means

n = 750

k = 2

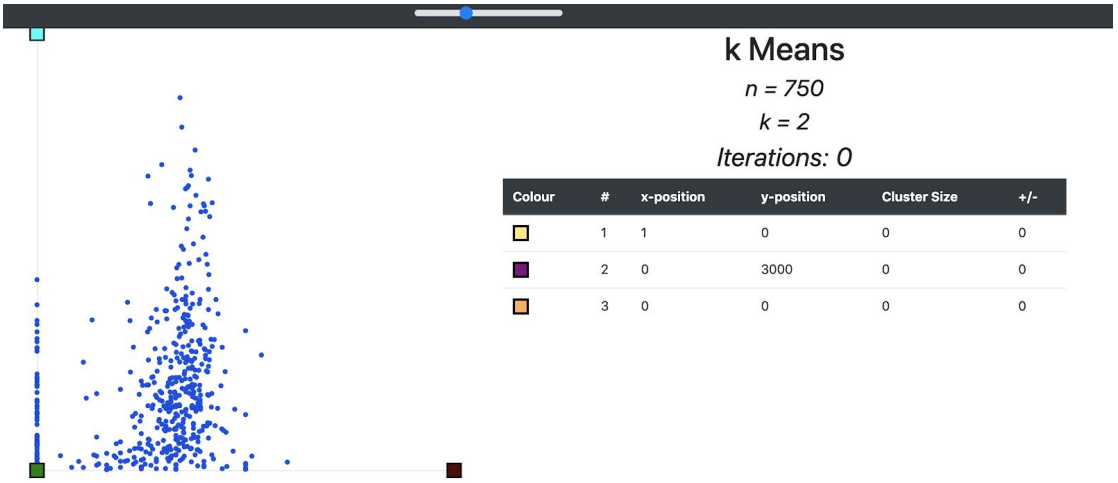
Iterations: 5

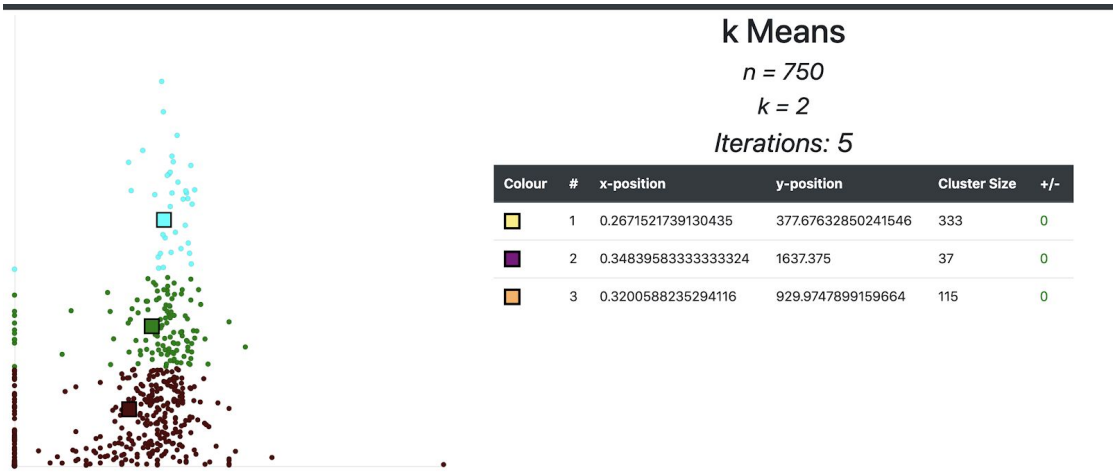
Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.676246631975163	0.6712336753731343	427	0
	2	0.8094107539119527	0.617129197761194	63	0
	3	0.6840798156185035	0.7794426305970149	260	0



With this run, it is very evident the major weakness k-means has over other clustering algorithms, which is that it's clusters are very dependant on the initial positions of it's clusters. As seen in the above two examples, despite the dataset being the same, the final clusters generated were completely different.

Real-life dataset:





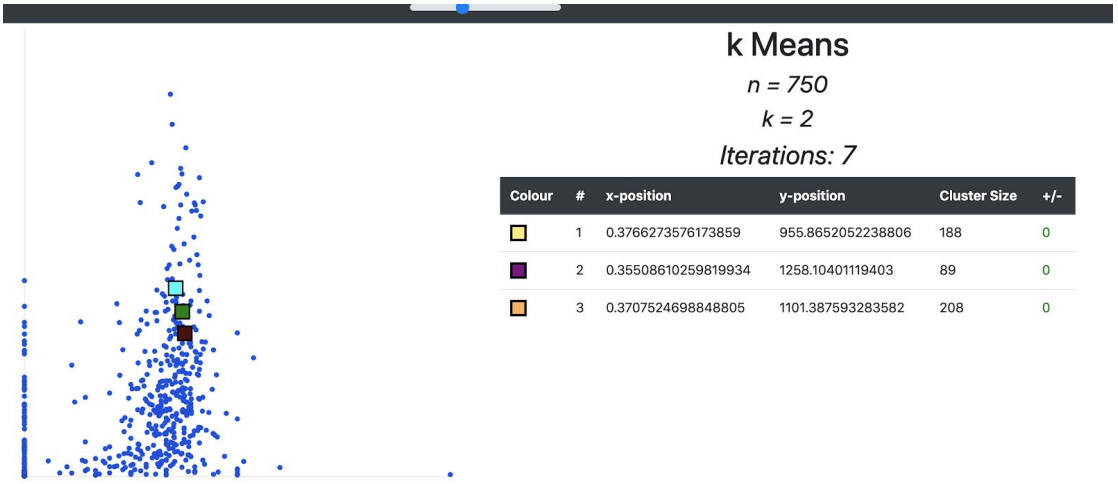
k Means

n = 750

k = 2

Iterations: 5

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.2671521739130435	377.67632850241546	333	0
	2	0.34839583333333324	1637.375	37	0
	3	0.3200588235294116	929.9747899159664	115	0



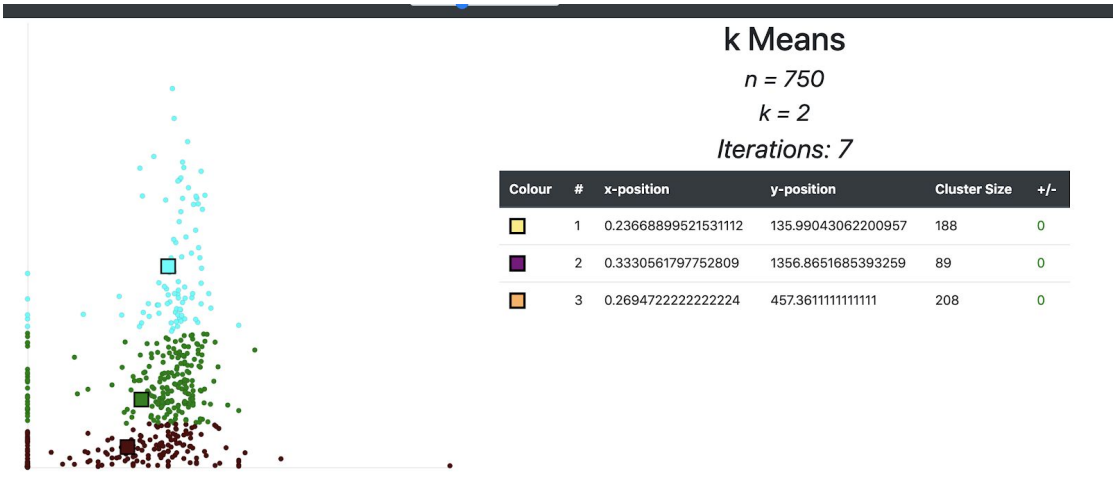
k Means

n = 750

k = 2

Iterations: 7

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.3766273576173859	955.8652052238806	188	0
	2	0.35508610259819934	1258.10401119403	89	0
	3	0.3707524698848805	1101.387593283582	208	0



k Means

n = 750

k = 2

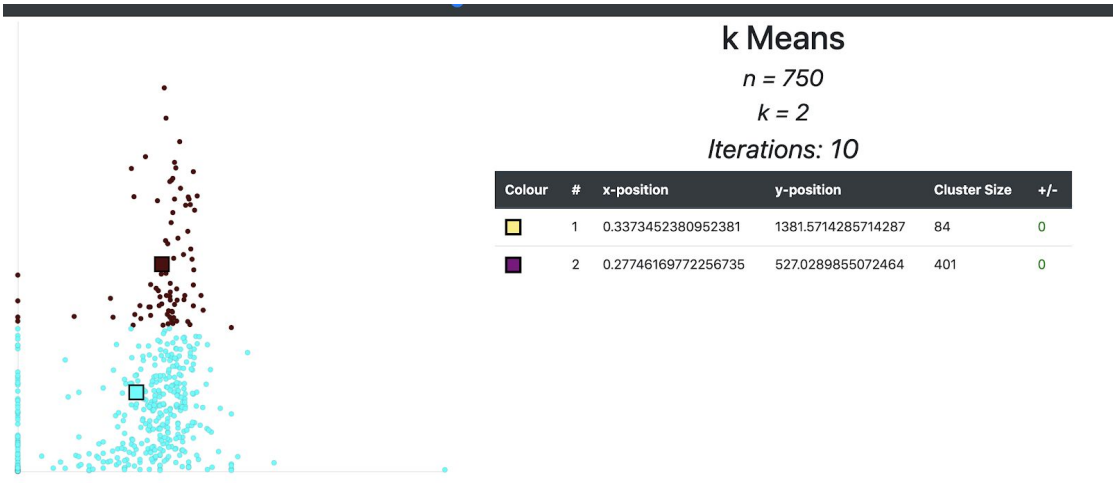
Iterations: 7

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.23668899521531112	135.99043062200957	188	0
	2	0.3330561797752809	1356.8651685393259	89	0
	3	0.2694722222222224	457.3611111111111	208	0

In my real-life dataset, the effect seems to be much smaller than observed with the random dataset. This may be because the data is already relatively clustered, and the algorithm does not have as many options to find other clusters.

Different k:

For this part of the test, I experimented with using different k and seeing the difference in results. For the NBA dataset, I first tried running the algorithm with 2 clusters, and then I ran it again with 5. What's noticeable is that with 5 clusters, a lot more detail can be extracted than with just two:



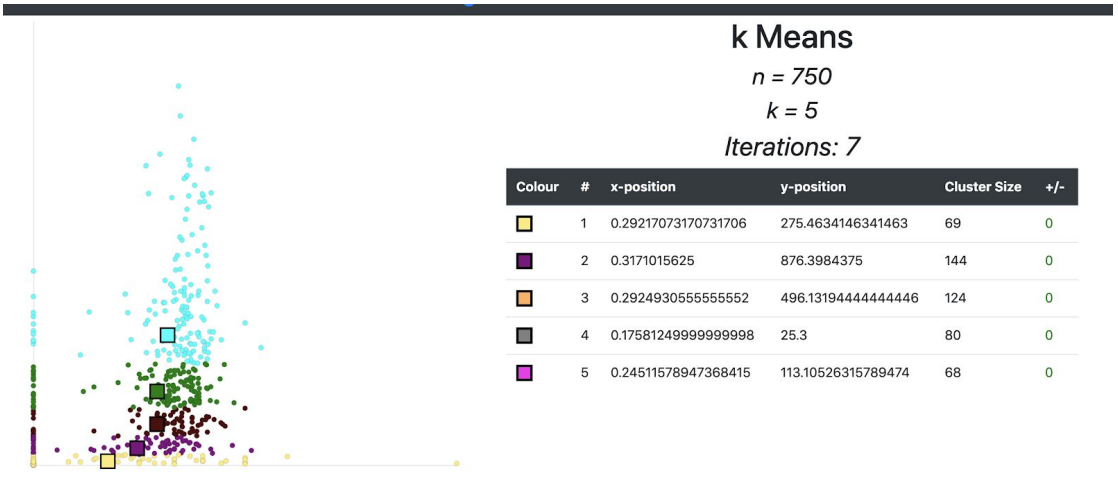
k Means

n = 750

k = 2

Iterations: 10

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.3373452380952381	1381.5714285714287	84	0
	2	0.27746169772256735	527.0289855072464	401	0



k Means

n = 750

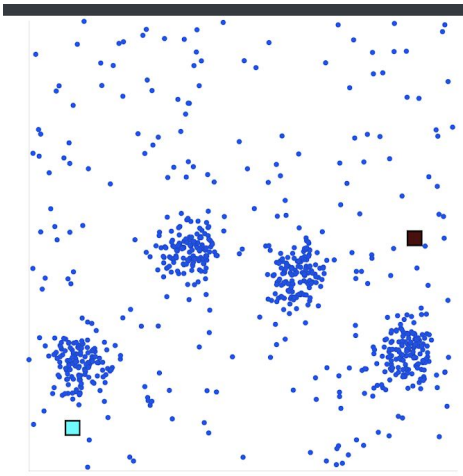
k = 5

Iterations: 7

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.29217073170731706	275.4634146341463	69	0
	2	0.3171015625	876.3984375	144	0
	3	0.2924930555555552	496.13194444444446	124	0
	4	0.17581249999999998	25.3	80	0
	5	0.24511578947368415	113.10526315789474	68	0

As this dataset has as it's x-axis 3 point shooting percentage, and it's y-axis points scored for the 2017 NBA season of that player, the clusters can help to show players and their tiers in terms of being effective at scoring points in the NBA. One interesting thing about this dataset, is that it shows that there are very few players who score highly on the lower end of the spectrum in terms of 3-point shooting percentage.

For the synthetic dataset, I decided to test out different k by first adjusting the amount of clusters. I ran the algorithm with a cluster amount matching k (in this case 4) and I ran it again later with decidedly less k (2). Again, by adjusting k, it's evident that finding an appropriate k for this algorithm is important, as you may miss out on certain details if it's too low.



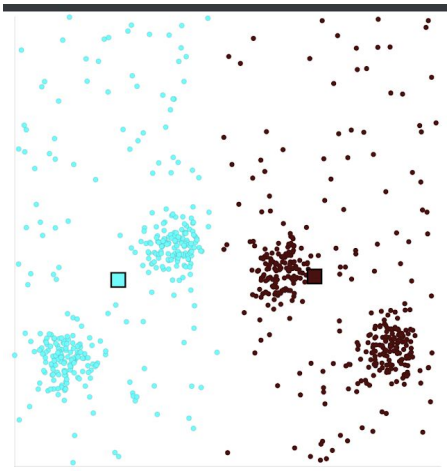
k Means

n = 750

k = 2

Iterations: 8

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.9014506617212046	0.5163829291044776	214	0
	2	0.10246593010046576	0.0947411380597015	174	0



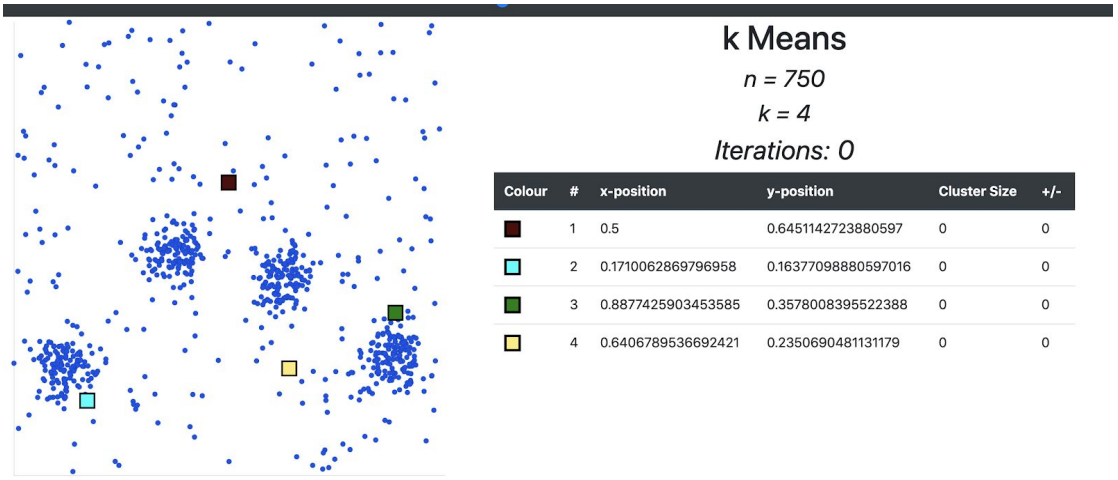
k Means

n = 750

k = 2

Iterations: 4

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.7047750872591924	0.4218496258992876	387	0
	2	0.24391204648538647	0.41401073280464146	366	0



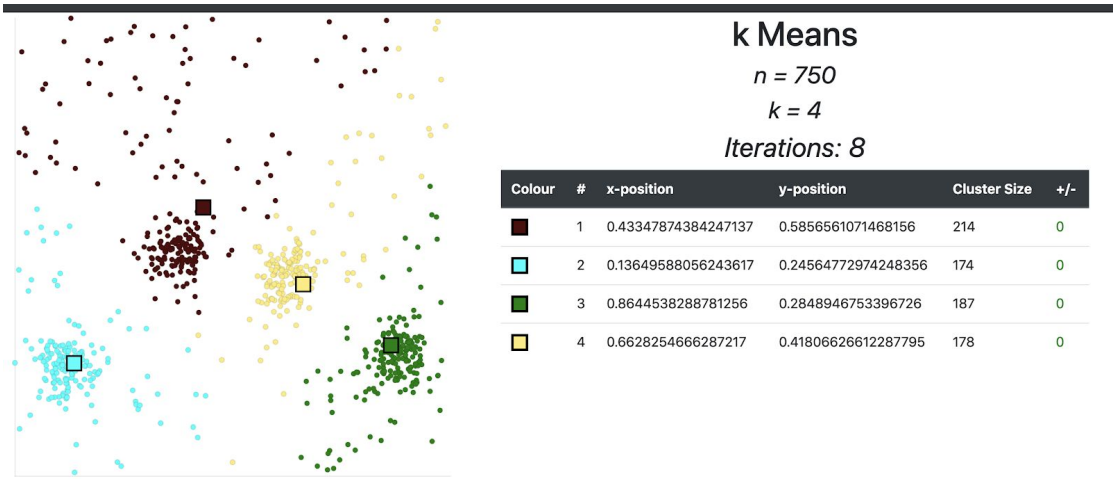
k Means

n = 750

k = 4

Iterations: 0

Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.5	0.6451142723880597	0	0
	2	0.1710062869796958	0.16377098880597016	0	0
	3	0.8877425903453585	0.3578008395522388	0	0
	4	0.6406789536692421	0.2350690481131179	0	0



k Means

n = 750

k = 4

Iterations: 8

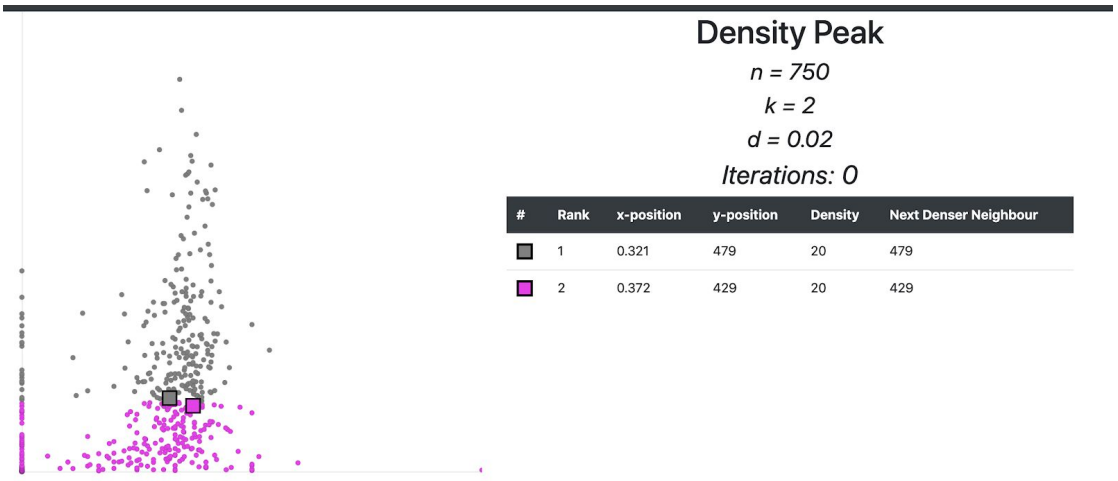
Colour	#	x-position	y-position	Cluster Size	+/-
	1	0.43347874384247137	0.5856561071468156	214	0
	2	0.13649588056243617	0.24564772974248356	174	0
	3	0.8644538288781256	0.2848946753396726	187	0
	4	0.6628254666287217	0.41806626612287795	178	0

Density Peak:

The biggest challenge faced during the development of this algorithm was how to find the “outliers” to represent cluster centers. After toying with several ideas, I decided the best way to go about it would be to first filter out those points that are below a 50% top density threshold, and then check that every centroid picked was not within a **d** amount of distance of another centroid. This helps to ensure centroids are accurately picked.

In experimenting with this algorithm, my main focus was to see how different the results it produces could be compared to k-Means. I was able to produce some noticeably different results:

Real-life dataset:



Density Peak

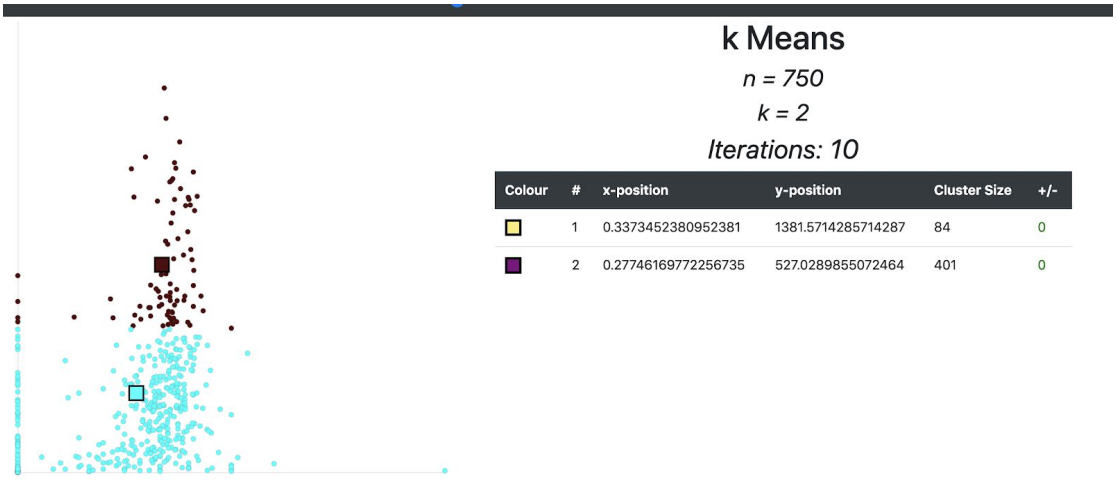
n = 750

k = 2

d = 0.02

Iterations: 0

#	Rank	x-position	y-position	Density	Next Denser Neighbour
■	1	0.321	479	20	479
■	2	0.372	429	20	429



k Means

n = 750

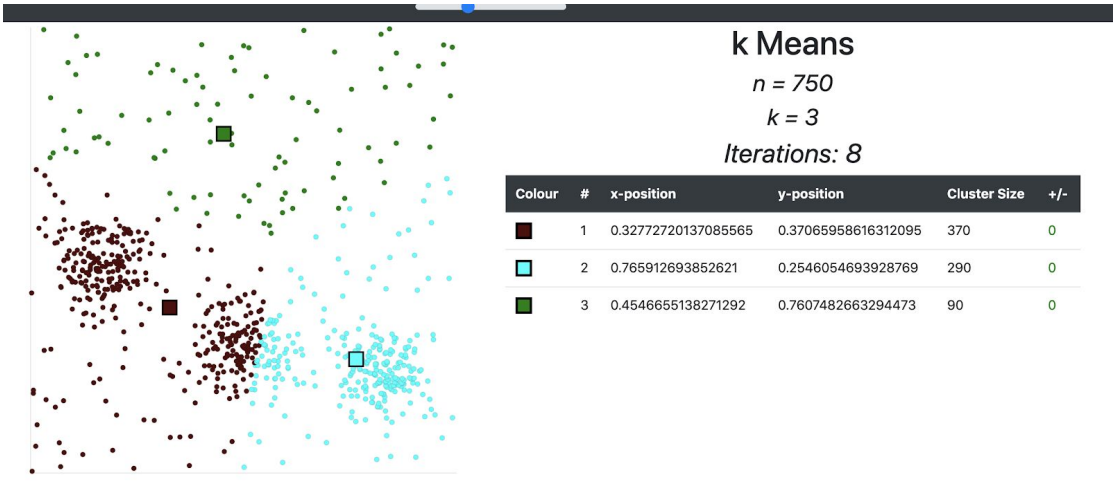
k = 2

Iterations: 10

Colour	#	x-position	y-position	Cluster Size	+/-
■	1	0.3373452380952381	1381.5714285714287	84	0
■	2	0.27746169772256735	527.0289855072464	401	0

Using Density Peak, although the two clusters identified are the same, the top cluster is noticeably larger than in k-Means.

Synthetic dataset:



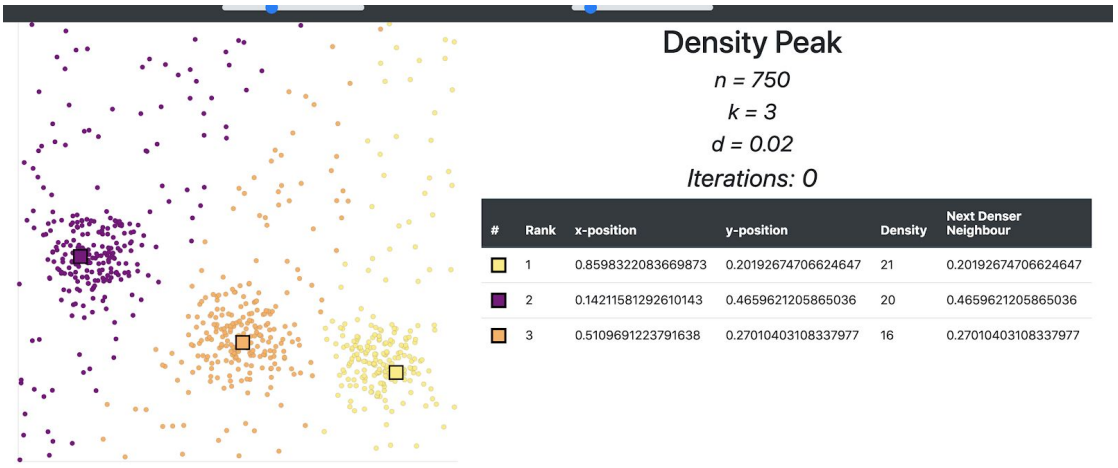
k Means

n = 750

k = 3

Iterations: 8

Colour	#	x-position	y-position	Cluster Size	+/-
■	1	0.32772720137085565	0.37065958616312095	370	0
■	2	0.765912693852621	0.2546054693928769	290	0
■	3	0.4546655138271292	0.7607482663294473	90	0



In this example, not only are the results very different between the two, density peak appears to have generated a much more accurate result as well.

Conclusion:

Through my visualization system, it's very evident where k-Means can produce inaccurate results if their initial points are not set at optimal locations. k Means also requires multiple iterations in order to make adjustments, while Density Peak can be completed in just one run of the algorithm.

Future Work:

This system is far from complete. Due to a lack of time, there are numerous bugs and unfinished components that I would like to finish at a later time. I think the Comparison Mode would be something very interesting and it would be my top focus when picking this project back up later on.

One thing that I would like to implement after the bugs have been fixed is the numerous improved density peak clustering methods that have been proposed since this algorithm was first introduced. One example is a paper written by French students Vincent Courjault-Radé, Ludovic d'Estampes, and Stéphane Puechmorel where they discuss improving the algorithm for larger datasets [10].

References:

[1] Mohan, K. (n.d.). *Visualizing K-Means Clustering*. [online] Stanford University. Available at: http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html [Accessed 24 Apr. 2019].

[2] Harris, N. (n.d.). *Visualizing K-Means Clustering*. [online] Naftaliharris.com. Available at: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/ [Accessed 27 Apr. 2019].

[3] Hdbscan.readthedocs.io. (n.d.). *Comparing Python Clustering Algorithms*. [online] Available at: https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html [Accessed 23 Apr. 2019].

[4] Abu Abbas, O. (2008). Comparisons Between Data Clustering Algorithms. [online] Available at: http://iajit.org/PDF/vol.5,no.3/15-191.pdf [Accessed 22 Apr. 2019].

[5] Stanford University. (n.d.). *CS221*. [online] Available at: http://stanford.edu/~cpiech/cs221/handouts/kmeans.html [Accessed 27 Apr. 2019].

[6] Trevino, A. (n.d.). *Introduction to K-means Clustering*. [online] Datascience.com. Available at: https://www.datascience.com/blog/k-means-clustering [Accessed 27 Apr. 2019].

[7] Rodriguez, A. and Laio, A. (2014). Clustering by fast search and find of density peaks. [online] Available at: http://sites.psu.edu/mcnl/files/2017/03/9-2dhti48.pdf [Accessed 27 Apr. 2019].

[8] GitHub. (n.d.). *Density Peak Cluster*. [online] Available at: https://github.com/lanbing510/DensityPeakCluster [Accessed 27 Apr. 2019].

[9] Goldstein, O. (n.d.). *NBA Players stats since 1950*. [online] Kaggle.com. Available at: https://www.kaggle.com/drgilermo/nba-players-stats [Accessed 27 Apr. 2019].

[10] Courjault-Radé, V., d'Estampes, L. and Puechmorel, S. (2016). Improved density peak clustering for large datasets. [online] HAL. Available at: https://hal.archives-ouvertes.fr/hal-01353574/document [Accessed 23 Apr. 2019].