Algorithmic Fairness in Clustering: A Study in Replication

Sophie Boileau, Avery Hall, Victor Huang, Jeremiah Mensah, Armira Nance, Muno Siyakurima, and Brie Sloves

1 Introduction

When we consider algorithms in computing, it's easy to assume that computers are objective, and therefore capable of doing work and making decisions in a way that is not susceptible to human error and implicit bias. Real-world applications of these algorithms, however, can reveal hidden bias and patterns of unfairness in our results. We consider this in the context of *k*-means clustering.

k-means clustering is a commonly used data science algorithm that partitions a dataset with *n* instances into fixed integer *k* clusters, in which each data point is closer to the mean of all points in its cluster than the mean of all points in any other cluster. The resulting clusters are a useful tool for data analysis, providing insight into internally similar subgroups that exist in the broader dataset. Certain applications of *k*-means clustering, for instance grouping pixels in a photo for compression or analyzing natural disaster data, do not necessitate protecting sensitive data. However, when clustering data containing sensitive demographic information such as racial identity and sex, the issue of maintaining fairness within individual clusters arises.

Fairness can mean different things depending on the context. In this paper, we explore two different definitions of fairness based on the papers "Fair Clustering Through Fairlets" by Chierichetti et al. and "Socially Fair *k* Means Clustering" by Ghadiri, Mehrdad et al. detailed in Section 2.2.

We chose to apply *k*-means clustering on data surrounding student academic performance and demographic information, outlined in more detail in Section 3. Due to the sensitive information in the data, it becomes crucial to consider the distribution of protected attributes within individual clusters in order to prevent inequity for marginalized groups (i.e. people of color and females for race and sex respectively). For our project, we completed an in-depth exploration of four variants of *k*-means clustering in relation to our data and explored differences in the fairness of the results. In particular, we explore Lloyd's algorithm (Dabbura et al. 2022), *k*-means++ (GeeksforGeeks), clustering via fairlets (Chierichetti et al. 2017), and socially fair Lloyd's algorithm (Ghadiri et al. 2021).

2 Conceptual Background

Succinctly, clustering algorithms identify groups in data that share similar features. *k*-means clustering partitions a dataset of size *n* into fixed-integer *k* clusters, each consisting of a subset of the data points and a centroid, which is the center of mass of all points in the cluster. The goal is to cluster points such that the distance from the farthest point to its centroid is minimized. We formally define the problem of *k*-means clustering in Section 2.2.1, using the definition from "Fair Clustering Through Fairlets" by Chierichetti et al. In this section, we describe the four different variants of *k*-means clustering that we explored. We outline the algorithms and discuss the goals and shortcomings of each.

2.1 Clustering Algorithms

The most basic approach to *k*-means clustering is Lloyd's algorithm which relies on random initialization of centroids. The algorithm is described below, accompanied by an illustration using the toy dataset below. Note that the data are denoted by circles, the cluster centers are denoted by stars, and the clusterings are denoted by colors.

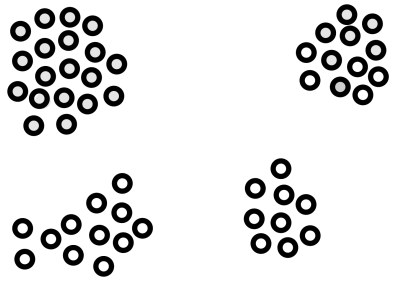

Figure 2A: Toy Dataset

We begin by choosing *k* random data points across the dataset's domain to be centroids. In this example, we consider $k = 4$.

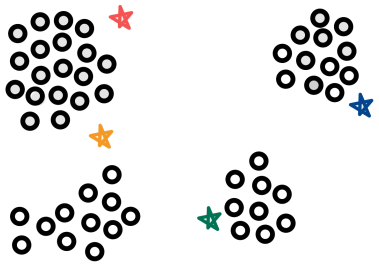

Figure 2B: Toy Data Centroid Initialization

For each of the *n* data points, calculate the distance between that point and each of the *k* centroids, and assign the data point to the cluster with the nearest centroid.

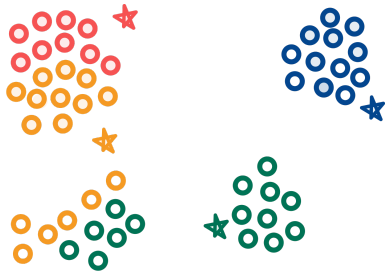

Figure 2C: Toy Data Initial Clustering

Then, for each cluster, find the mean of all data points in that cluster and update the location of the centroid.

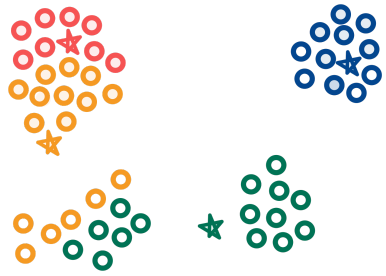

Figure 2D: Toy Data Second Iteration

Repeat the data point assignment and centroid update step until convergence is reached (i.e. the clusters are unchanging).

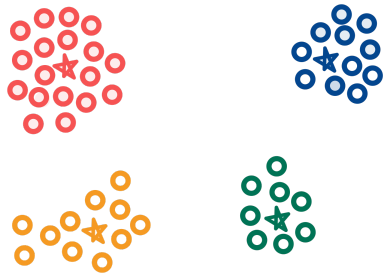

Figure 2E: Toy Data Final Clustering

One big shortcoming of Lloyd's algorithm is that it can get stuck in local optima depending on the random initialization of the centroids. Consider the following two clusterings.

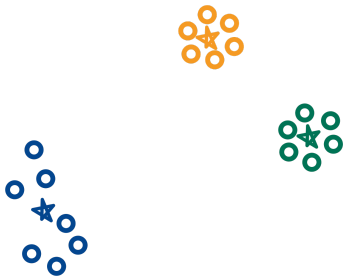

Figure 2F: Globally Optimal Clustering

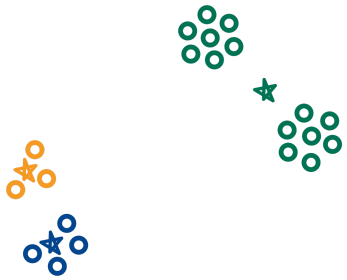

Figure 2G: Globally Suboptimal Clustering

The clustering pictured in Figure 2F shows the clear optimal solution for clustering the data. Each cluster is well-defined, has points close to its respective centroid, and is distant from other clusters. However, the clustering pictured on the right shows a locally optimal but globally suboptimal solution for clustering the data. All data is correctly assigned to the nearest centroid, allowing for convergence, but this solution is not globally optimal. The orange and blue clusters, pictured in the bottom left of Figure 2G, are much closer together than the clusters in the globally optimal solution above. In addition, the green cluster is much larger than any of the clusters seen in the globally optimal solution.

k-means++ clustering is a variation of Lloyd's algorithm which aims to solve the issue of convergence at local optima, improves runtime, and provides more stable clustering. It is much the same as Lloyd's algorithm, only differing in the initial selection of centroids. Like with Lloyd's algorithm, we randomly select a data point to be the first centroid. However, any additional centroids are selected as follows: For each of the n data points, calculate the squared distance between the point and the nearest previously selected centroid. Then, assign probabilities to each of the n points, which are directly proportional to their distance from the nearest centroid, calculated by dividing the distance by the sum of distances from each point to its nearest centroid. Select the next centroid randomly from the n points, with selection probabilities as calculated in the previous step. Once k centroids have been selected, proceed with Lloyd's algorithm as previously described.

Put simply, the probability of a point being selected as a centroid is higher the farther it is from other centroids. This increases the chance that initial centroid positions are evenly across the data, minimizing the occurrence of scenarios similar to the globally suboptimal clustering in Figure 2G where the orange and blue clusters are in close proximity to one another.

2.2 Fair Clustering Algorithms

In certain applications, such as partitioning electoral precincts or resource allocation, groupings should be done in a fair way. To this end, many modifications have been proposed in recent years to make *k*-means clustering algorithms more fair. However, the definition of fairness may vary. Suppose a clustering algorithm is applied to data from a set of individuals in a town with the hopes that the clusters can be used to identify electoral precincts. If the clustering algorithm optimizes for some definition of fairness, then it is possible that downstream decisions that require different definitions of fairness, such as the allocation of resources based on those

electoral precincts, may lead to inequities. For the remainder of this section, we detail two such algorithms that were designed to be fair.

To compare the two algorithms, we will examine the toy dataset given above in Figure 2A. However, we will first make a couple features of the dataset salient, as shown in the diagram below.

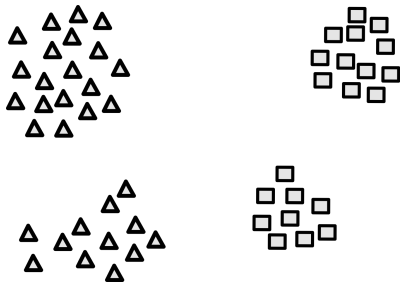

Figure 2H: Toy Dataset with Protected Attributes Emphasized

For demonstration purposes, rather than depicting all of the data with a single shape, we replace the circles with triangles and squares. These shapes exhibit that the data belong to two separate protected attributes. Suppose that this data represents the population of a town, where the triangles represent Asian residents, and the squares represent Black residents. If we were to take clustering results as a determining factor for allocating resources or partitioning voters into electoral precincts, clustering must be done fairly between racial demographic groups.

2.2.1 Fair Clustering through Fairlets

In their 2017 publication *Fair Clustering through Fairlets,* Chierichetti et al. append a preprocessing step to the traditional *k*-means clustering algorithm that ensures a proportionally fair output. We focus on their results for *k*-means, but their findings extend to *k*-median algorithms as well.

Before detailing their algorithm and main findings, we must first address some preliminaries: Let X be a set of points in a metric space, which we cluster into k disjoint subsets $\mathcal{U} = \{U_1, \ldots, U_k\}$ with centers $C = \{c_1, \ldots, c_k\}$. To evaluate the output, our objective function is

$$min_{C=\{c_1, \ldots, c_k\}} \Delta(C, \mathcal{U}_C),$$

where

$$\Delta(C, \mathcal{U}) = \sum_{i=1}^{k} \sum_{p \in U_i} ||p - c_i||^2.$$

Here, p describes a point in the cluster. This is the same cost function as the k-means clustering algorithm described earlier.

As shown in the toy data, we assume that each point in X belongs to exactly one of two protected attributes – protected attribute A or protected attribute B. From this information, we define *balance* in terms of the ratio of the number of points in each protected attribute, expressed as a number between 0 and 1, where 0 denotes drastically imbalanced clusters, and 1 denotes perfectly balanced clusters. More formally, we describe the balance of protected attributes in a nonempty subset $Y \subseteq X$ as

$$balance(Y) = min(\frac{\#\ Attribute\ A(Y)}{\#\ Attribute\ B(Y)}, \frac{\#\ Attribute\ B(Y)}{\#\ Attribute\ A(Y)}) \in [0, 1],$$

and the balance of a clustering $\mathcal{U}$ as

$$balance(\mathcal{U}) = min_{U \in \mathcal{U}} balance(U).$$

In other words, the balance of an overall clustering is simply the ratio of protected attributes in its least evenly balanced cluster.

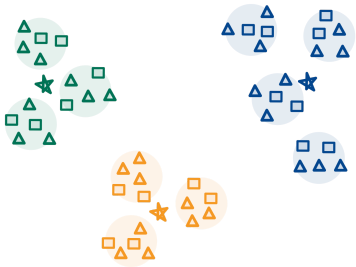

Figure 2I: Clustering via Fairlets

Importantly and without loss of generalization, if $balance(X) = \frac{a}{b}$ for some integers $1 \leq a \leq b$ such that this fraction is simplified, then there exists a clustering $\mathcal{Y} = \{Y_1, \ldots, Y_m\}$ where each cluster is as small as possible and maintains the proportion $\frac{a}{b}$. These clusters of minimal sets that preserve the balance of protected attributes from the overall dataset are dubbed *fairlets*, and the clustering $\mathcal{Y}$ is a (a, b)-*fairlet decomposition* of X. Once we obtain the fairlet decomposition of the dataset, we cluster the fairlets according to a classical clustering algorithm, so the balance of the clusters is maintained throughout the clustering process, as desired. Considering the dataset in Figure 2I above, the balance of square to triangle protected attributes is $\frac{2}{3}$. Thus, we make sets of data that are as small as possible (two squares and three triangles) while preserving this ratio and minimizing the cost of each fairlet. These fairlets are depicted in

the diagram as shaded circles enveloping the minimal sets. Now that these fairlets preserve the balance of the overall dataset, we can cluster them into three clusters (shown in different colors) without fear of disrupting the balance. Therefore, the key to designing an efficient fair clustering algorithm lies in optimizing the preprocessing step.

In the simplest case in which clusters are perfectly balanced, a $(1, 1)$-fairlet decomposition can be found in polynomial time. In general, however, finding an optimal (a, b) -fairlet decomposition is NP-hard, and we refer to this as the *vanilla* fairlets. However, after converting this problem into a minimum-cost flow (MCF) instance, it is possible to construct a feasible fairlet decomposition while bounding the cost by a factor of 2. We refer the interested reader to the original paper for more of the details. Therefore, we can efficiently cluster any data fairly by first decomposing it into fairlets, then clustering with a traditional algorithm, like k-means.

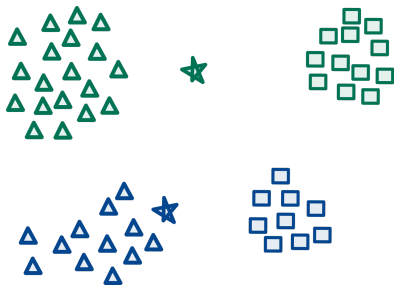

Figure 2J: Toy Data Fairlet Clustering

To illustrate the outcome of this framework, we return to our toy dataset. Suppose that this town of Asian and Black residents must be partitioned into electoral precincts. Given that different marginalized communities might have different shared interests from those of other marginalized communities, creating districts must be done in a way that ensures that proportional representation is maintained within clusters. In other words, we do not want to cluster in a way that gerrymanders this town. With this goal in mind, we consider the clustering in Figure 2J to be fair, as the green cluster and the blue cluster each maintain the proportion of triangles and squares. This partitioning allows for each voter's voice to have as much impact as possible while maintaining the integrity of proportionality within clusters, especially in situations where a certain group of people with the same demographic broadly vote in a different way than a group of people with another demographic attribute may vote.

While this simple example demonstrates the promise of clustering though fairlets, its particular definition of fairness is not applicable for every fair clustering problem. In this way,

the scope of the algorithm is limited. Furthermore, the theory is restricted, as it only permits the dataset to contain two protected attributes. There are plenty of situations in which data cannot be partitioned into binary attributes – i.e., it is unlikely that all residents of a particular town are either Asian or Black and have no other racial identity. In the next section, we describe an algorithm that supports clustering of any number of protected attributes.

2.2.2 Socially Fair *k*-Means Clustering

In 2021, Mehrdad Ghadiri, Samira Samadi, and Santosh Vempala published a paper entitled *Socially Fair k-Means Clustering*. They propose a more "human-centric" algorithm that is at odds with algorithms that prioritize proportionality (Ghadiri et al, 438). Rather than minimizing the average clustering cost over an entire dataset, they minimize the average clustering cost across different demographic groups in the dataset. Their findings extend to datasets with more than two demographic groups, but for the sake of explaining their findings, we limit our description to two demographic groups.

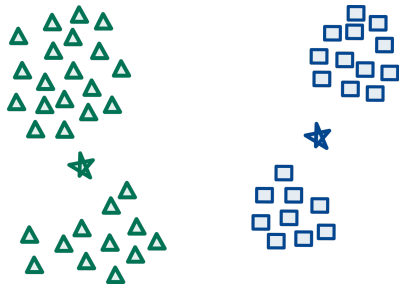

Figure 2K: Toy Data Socially Fair Clustering

Returning to our example dataset of a town, suppose that it has no grocery stores. Luckily, the town has enough money to build exactly two identical grocery stores. We must place them equitably between the Asian and Black households in such a way that does not force Asian shoppers to drive, on average, significantly further (or less far) than the Black shoppers. Therefore, we use the socially fair Lloyd's algorithm, which outputs the clusters above, to inform our decision on where to place stores. In this scenario, the stars denote where we choose to build grocery stores. Importantly, we do not require that each grocery store serves the same amount of Asian and Black shoppers, just that they are conveniently located within the town. When allocating resources in this way, fairness matters for minimizing the travel time to accessing those resources.

To make this notion more rigorous and using the same notation as in the previous algorithm, we again assume that $X = A \cup B$; the dataset exclusively contains points from group A and group B. Then, we define the fair k-means objective as minimizing the larger average cost

$$\phi(C, \mathcal{U}) = max\{\frac{\Delta(C, \mathcal{U} \cap A)}{|A|}, \frac{\Delta(C, \mathcal{U} \cap B)}{|B|}\}.$$

In this case, $|A|$ is the number of points in X from protected attribute A, and $\mathcal{U} \cap A = \{U_1 \cap A, \ldots, U_k \cap A\}$. The same definitions hold for group B. This optimization problem is inserted into Lloyd's algorithm, as the centers are chosen such that $\phi(C, \mathcal{U})$ is minimized. In this way, the only change made from a classical k-means clustering algorithm to this fair algorithm lies in the update step of the cluster centers.

To implement this modification efficiently, we use a line search algorithm, as outlined below. The intuition for this is explained by analyzing the structure of fair centers: Let μ_i^A be the mean of $A \cap U_i$, and let μ_i^B be the mean of $B \cap U_i$. Then, the optimal fair center c_i of cluster U_i is on the line segment between μ_i^A and μ_i^B, as depicted below.

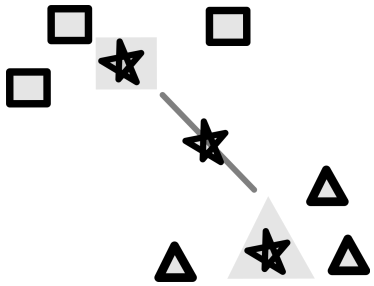

Figure 2L: Fair Centers

To find the fair centers, therefore, we only need to search the points on that line segment.

This set of points is defined in the following way. Let

$$\alpha_i = \frac{|A \cap U_i|}{|A|}, \quad \beta_i = \frac{|B \cap U_i|}{|B|}, \quad \text{and } l_i = ||\mu_i^A - \mu_i^B||.$$

To make sense of these expressions, we can think of α_i as the proportion of points from group A in a certain cluster i, β_i as the proportion of points from group B in a certain cluster i, and l_i as the distance between the means μ_i^A and μ_i^B. Also, let M^A and M^B be the set of each cluster's mean from group A and group B, respectively. That is,

$$M^A = \{\mu_1^A, \ldots, \mu_k^A\} \quad \text{and } M^B = \{\mu_1^B, \ldots, \mu_k^B\}.$$

From these terms, we say that the average cost of group A with respect to a partition $\mathcal{U}$ is

$$\frac{\Delta(M^A,\mathcal{U})}{|A|} + \sum_{i=1}^{k} \alpha_i p_i^2.$$

Likewise, the average cost of group B with respect to a partition $\mathcal{U}$ is

$$\frac{\Delta(M^B,\mathcal{U})}{|B|} + \sum_{i=1}^{k} \beta_i (l_i - p_i)^2.$$

Therefore, the goal is to solve the optimization problem

$$max\{\frac{\Delta(M^A,\mathcal{U}\cap A)}{|A|} + \sum_{i=1}^{k} \alpha_i p_i^2,\ \frac{\Delta(M^B,\mathcal{U}\cap B)}{|B|} + \sum_{i=1}^{k} \beta_i (l_i - p_i)^2\},$$

with the basic constraints that $0 \le p_i \le l_i, \forall i \in [k]$. We can express any fair center as $c_i = \frac{(l_i - p_i)\mu_i^A + p_i \mu_i^B}{l_i}$. To synthesize all of this information, we rewrite our objective function as the following program:

$$\begin{aligned} &\min \theta \\ &\text{such that } \frac{\Delta(M^A,\mathcal{U}\cap A)}{|A|} + \sum_{i\in[k]} \alpha_i p_i^2 \le \theta \\ &\qquad \frac{\Delta(M^B,\mathcal{U}\cap B)}{|B|} + \sum_{i\in[k]} \beta_i (l_i - p_i)^2 \le \theta \\ &\qquad 0 \le p_i \le l_i, \quad \forall i \in [k]. \end{aligned}$$

To compute the optimum, we search the set of all possible values for the fair center, which is given by the set

$$Z = \{p: p_i = \frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i + (1-\gamma)\beta_i},\ \gamma \in [0,1]\}.$$

Testing each viable γ in the set Z, we obtain an algorithm that runs in logarithmic time. This algorithm is well behaved, as the optimal solution is found in finite time and can be initialized tactfully with an approximation algorithm.

As before, while this fair algorithm is valuable, it is not the panacea. One major limitation of the literature is that it does not account for instances of intersectionality. In a town like our example, many of the residents may identify themselves as both Asian and Black, which this framework cannot support. This is discussed further in Section 6.1.

2.2.3 Algorithmic Comparison

As noted by Ghadiri et al, “we emphasize that improving the proportionality is at odds with improving the maximum average cost of the groups,” (446). Therefore, the two algorithms that we discussed in this section yield opposite results. For the remainder of this section, we provide some intuition as to why these two definitions of fairness give contrasting output by investigating an instance of the most fair and least fair clustering assignment.

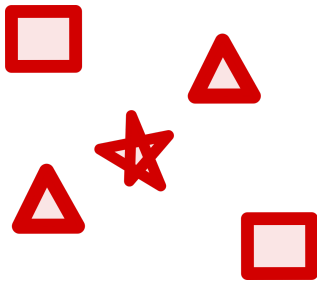

Figure 2M: One Cluster

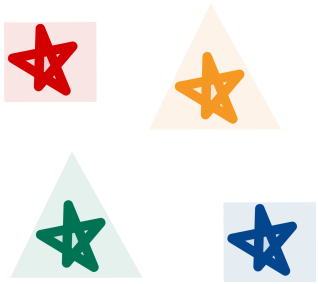

Figure 2N: One Cluster per Observation

First, we examine the fairlet approach, which operates under the goal of maintaining the dataset’s proportion of protected attributes in each cluster. With this objective, an instance of the most fair clustering will be the case in which we only have one cluster (though this defeats the purpose of clustering, as in Figure 2M), and the complete dataset is the cluster output. As the clusterings include more and more clusters, the clustering output is overall less and less likely to preserve the balance of demographics. A worst case scenario occurs when each data point is assigned its own cluster – once again, defeating the point of clustering (see Figure 2N) – as the balance of more than one protected attribute cannot possibly be greater than 0. Thus, a trend may be that as we increase the amount of clusters, we decrease the fairness of the final clustering.

On the other hand, the socially fair Lloyd’s algorithm aims to minimize the average group cost for clusters. Granting each data point its own cluster would give an optimal solution because the group costs would be 0 (Figure 2N). Naturally, it does not make sense to cluster in this way, but this is a useful thought exercise. Reducing the number of clusters and including more points per cluster makes clusters less likely to be fair, and it certainly increases the cost for the clustering. The other extreme, creating just a single cluster from the dataset, not only provides no useful information, but gives us the worst clustering cost, as all points – including the two with the greatest distance between them – are in the same cluster and considered in cost calculations. Thus, we observe that as we decrease the amount of clusters, we decrease the average group cost of the final clustering.

To summarize these findings, these trends point to a reason why the two algorithms and definitions of fairness do in fact give results that are at odds with each other. The least and most fair scenarios manifest at exactly opposite amounts of clusters for each algorithm, This supports the conclusion that the appropriate definition of fairness employed is highly dependent on the context of the problem.

3 Data

Given that we're exploring the fairness of a popular data science tool, we knew very early in the planning process that our project required data with a wide range of numerical and categorical attributes for each data point. The numeric attributes provide different options for dimensions of clustering our data and the categorical attributes offer us insight into what representation looks like within different clusters after we have run the algorithm. In order to implement the *k*-means clustering algorithm in addition to our variants (*k*-means++, clustering through fairlets, and our attempt at a socially fair Lloyd's algorithm), we sought out data with various protected categorical attributes (i.e. culturally and socially sensitive demographic identity information such as race, sex, and neurodevelopmental status, to name a few).

Our original data set was college graduation and dropout data gathered, cleaned, and published by a higher education institution in Portugal. This dataset fit the criteria for our search, and it appealed to us because it was already clean. There were no missing values, and the numerical data didn't have any glaring issues. The first obstacle we faced with this dataset was that the dataset did not contain information about any racial or ethnic identities, because anti-racism laws in the region deem the collection of racial and ethnic data unlawful. We really wanted to analyze race data, because racial and ethnic identities are one of the primary modes of discrimination and bias. Another issue — which was the primary deterministic factor in our decision not to use this dataset — was that we could not, in good faith, attempt to judge the social fairness of this algorithm using data from Portugal. None of our group members have a comprehensive understanding of the socio-political landscape in that country, so we would have struggled to be responsible about the conclusions at which we arrived.

Upon concluding that this dataset wouldn't be ideal for our project, we searched for similar data gathered from American schools. We came across data gathered by the National Center for Education Statistics' (NCES) High School Longitudinal Study (HSLS), which

gathered about 1900 different data attributes for approximately 23,000 students from 944 schools across the country, selected as part of a seven-year data collection study (2009-2016). Once our group decided to use this data, there was work to do. Unlike our original dataset, this data was not cleaned or standardized, and contained missing values that would've prevented our clustering algorithm from working. We examined the rows containing missing values and found that the vast majority of the rows with missing values in the numerical data also contained missing values for several other chosen categorical attributes, thus it was justifiable to remove these rows, so the algorithm could successfully cluster our data without breaking. Ultimately, of the 1900 data attributes available in the overall dataset, we chose to keep 18 attributes. Out of 18, we ended up directly utilizing 7 attributes that seemed to correspond to issues of fairness and bias: highest parent level of education (ParEdu), socioeconomic status (SES), household income (HHInc) and number of household members (HHMem) (combined to create a new, more comprehensive metric—IncMem), number of hours spent on extracurricular activities per week (HrAct), sex (Sex), and race (Race). Notably, sex had a roughly even split between male and female. SES is an existing numerical metric which considers parent education, occupation, and income.

Then, we proceeded to clean and standardize variables for clustering using R. The first step was converting categorical variables into numerical values to be clustered. For households with greater than 11 members, we encoded the HHMem as 11, as this was the lower bound of the maximum category for this attribute. HHInc consisted primarily of income ranges. For each of the ranges, we encoded the value as the mean of the range presented. For the maximum and minimum values (>$235,000 and <=$15,000), we encoded them to the lower and upper limits of 235 and 15 respectively. We then divided HHInc by HHMem to create the new variable IncMem, representing the annual income per member of the household. HrAct consisted of values from "Less than 1 Hour" to "5 or more hours", with intermediate values being one hour ranges. To encode this, we set values equal to the lower bound of the range. For ParEdu, we encoded each degree type as a number, ordering them based on prerequisite degrees, as follows.

Table 3A: Numerical Encoding for ParEdu

ParEdu	< H.S.	H.S. or GED	Associate's	Bachelor's	Master's	Ph.D., M.D., J.D., other
Numerical	1	2	3	4	5	6

Once the data was cleaned and standardized, we clustered based on the highest parent's level of education, income per number of people in the household, socioeconomic status, and number of hours spent on extracurricular activities per week. Summary statistics of these variables are below.

Table 3B: Summary Statistics for Quantitative Variables

	Min	Q1	Median	Mean	Q3	Max	SD
ParEdu	1.000	2.000	3.000	3.221	4.000	8.000	1.372
SES	-8.000	-0.563	-0.005	-0.055	0.670	2.881	1.362
HrAct	0.000	0.000	1.000	1.444	2.00	5.000	1.452
IncMem	1.364	7.500	15.000	19.218	26.250	117.500	15.280

4 Implementation

We set out to implement four variations of *k*-means clustering, including basic *k*-means as a control and a comparison for the others. *k*-means++, while not a fair clustering algorithm, is a variation proven to reduce runtime and increase average performance. This served as a second dimension of comparison with fair clustering. Our main fair clustering implementation was clustering through fairlets. Each of our clustering implementations drew on previous implementations to varying degrees where necessary or helpful given our time constraints.

4.1 Implementation of *k*-Means and *k*-Means++ Clustering and Elbow Method

This section covers the implementation of the *k*-means and *k*-means++ clustering algorithms, leveraging a variety of software tools and libraries.

4.1.1 *k*-Means Clustering

Our journey through implementation begins with the loading of the dataset from a CSV file using Pandas, a versatile data manipulation library native to Python. The dataset was then preprocessed to extract the relevant features, namely the highest parent education level, socio-economic status, income per household member, and total hours of weekly extracurricular activities. This initial step lays the foundation for subsequent in-depth analysis.

Recognizing the inherent complexity of the dataset, Principal Component Analysis (PCA) is enlisted for dimensionality reduction. PCA is a library offered by Scikit-learn that transforms the dataset into a set of linearly uncorrelated variables, known as principal

components. In this context, the dimensionality is reduced to two principal components, facilitating effective visualization of the underlying patterns.

The heart of the implementation lies in the application of the *k*-means clustering algorithm. A custom `KMeans` class is instantiated, allowing flexibility in specifying the number of clusters and maximum iterations. Recall from Section 2, the algorithm begins by randomly initializing centroids across the dataset's domain. It then iteratively assigns data points to the nearest centroid, updates centroids based on the mean of the points in each cluster, and repeats until convergence or a set maximum number of iterations.

Afterwards, the power of visualization is harnessed through Matplotlib and Seaborn, enabling the creation of a two-dimensional scatter plot. Each data point is color-coded according to its assigned cluster, providing a visually intuitive representation of the dataset's clustering patterns. Centroids are distinctly marked with 'X' yellow symbols, aiding in the interpretation of the algorithm's outcomes.

4.1.2 Elbow Method and Visualization

The function `elbow_method` performs the Elbow Method—a technique used to determine the optimal number of clusters in a *k*-means clustering algorithm. It takes as input a `DataFrame` containing the dataset to be analyzed and an optional parameter `max_clusters`, which represents the maximum number of clusters to evaluate. Within the function, it computes the Within-Cluster Sum of Squares (WCSS), a measure of variability within clusters, for different values of *k*.

The method proceeds by reducing the dataset to 2 principal components using PCA to facilitate visualization. It then iterates through a range of *k* values, from 1 to `max_clusters`, fitting a `KMeans` model for each *k*. For every iteration, the WCSS is calculated and stored in the `wcss` list.

Finally, the function generates an Elbow plot using Matplotlib, displaying the relationship between the number of clusters and the corresponding WCSS values. The point where the plot starts to form an ‘elbow’ or a bend, indicative of diminishing returns in WCSS reduction as *k* increases, helps determine the optimal number of clusters for the given dataset. The function visually assists in identifying the most suitable *k* value for subsequent *k*-means clustering.

4.2 Implementation of *k*-Means ++

Similar to the *k*-means implementation, Principal Component Analysis (PCA) was employed to reduce the dataset's dimensionality. This addresses the challenge of visualizing high-dimensional data. Since the data was in four dimensions, it required a technique for rendering it in two dimensions, and PCA was a helpful tool in accomplishing that.

The implementation further uses Tkinter for Graphical User Interface (GUI) visualization of clustering results. Matplotlib FigureCanvasTkAgg and Figure are utilized to create a canvas and figure for plotting the clusters, offering an interactive and visually intuitive representation. This design choice facilitates easy interpretation of clustering outcomes.

NumPy was crucial for optimizing the efficiency of *k*-means++ implementation. Its built-in functions such as `sum`, `array`, `random`, and `mean` streamline the process, making the codebase more concise and computationally efficient. NumPy was especially useful when choosing the next centroids in the *k*-means++ algorithm. `Sum()` allowed us to calculate the probabilities, and `Array()` allowed us to store the squared distances calculated.

As we needed both visual and numerical insights, Tkinter was selected for visualization, while the clusters' results were saved to a CSV file for detailed examination. Challenges arose in structuring the function, specifically in linking each cluster to its associated data points. The initial approach involved considering the use of a dictionary to map each data point to its corresponding cluster. However, a more streamlined method emerged during the implementation process.

The method includes extracting the data points from the original dataset that belong to the current cluster. Then creating a list where each element is a string indicating the cluster to which the corresponding data point belongs. Each list has the same length as the number of data points in the current cluster. After that convert cluster points from our data from a pandas dataframe to a numpy array and loop through cluster data and write the resulting clusterings to the CSV file.

Several design choices were made throughout the implementation process. The decision to maintain a unified function for *k*-means++ function, encompassing both the initialization of *k*-means++ and basic *k*-means steps, aims to enhance code readability and cohesion.

Originally, we were only able to read in numerical values from our dataset, which hindered our data analysis because we had no way of telling whether a data point is associated with our chosen protected attribute. However, when switching to pandas for data loading,

non-numerical values were loaded along with numerical values, which allowed for better data analysis.

4.3 Clustering Through Fairlets

With the data creation and sanitation step completed, we start the clustering through fairlets model with the data preparation step. As the clustering through fairlets model was computed on Google Colab due to its GPU processing power, the data was initially copied into a Google Drive folder and accessed through this folder path. Once the data was loaded, a JSON file was created in the configuration file to load the associated fair columns, distance columns, split size, and other required json parameters. Once the data is loaded, it is then parsed using a data loader function to prepare the data for the upcoming fairlets decomposition step. We then proceeded to normalize the dataset and split the data into our binary (male, female) data points represented by the 'blue' and 'red' python lists.

After the data was prepared, we proceeded to create a basic fairlets decomposition class variable that takes the aforementioned prepared data as an input as well as a pair of 'majority' and 'minority' clustering variables. Once the fairlet decomposition outputs basic fairlets, associated fairlet centers, and associated fairlet costs, we proceed with calculating metrics to gauge fairlet clustering efficacy.

The two metrics we will be using to gauge the fairness of clustering through fairlets are *k*-means cost and balance score. The code for calculating balance and *k*-means cost implements the logic mentioned in Section 2.2.1. Once the balance and *k*-means cost metrics were calculated for each *k* value, we proceed with plotting our results to compare how clustering via fairlets fares against conventional *k*-means clustering.

Our two graphs (Figures 5C and 5D) sought to compare the efficacy of all three clustering algorithms when compared subjectively. The first graph is used to compare the k-means cost across multiple *k* values for our clustering algorithms while the second graph is used to compare balance across multiple *k* values for our clustering algorithms. The choice of axis labels and titles for each graph reflects the aforementioned metrics and the line graph displayed shows a comprehensive comparison between all clustering algorithms for each *k* value.

4.4 Socially Fair Lloyd's Algorithm

We initially set out to develop a second fair clustering implementation based on the algorithm described in *Socially Fair k-Means Clustering* by Ghadiri et al. We did this by

adapting a previous implementation to suit our data and methods, similar to our process for implementing the algorithm from *Fair Clustering Through Fairlets*. The first step was to translate the original code from MatLab to Python. Unfortunately, we discovered midway through the process that the base code we were using employed several function calls unique to libraries owned by the original authors, which we were ultimately unable to access. After several attempts to circumvent these issues, we decided to halt development of this implementation, given the time constraints of the project. Nevertheless, we have included the incomplete implementation in our Github repository as documentation of our process.

4.5 Data Parsing for Cluster Analysis

Finally, we wanted a mode of producing non-graphed quantitative results in order to gauge the effectiveness of the various clustering implementations, with the ultimate goal of comparing balance between clustering algorithms. We set out to accomplish this by implementing two Python programs that would parse clustering outputs. The `general-cluster-breakdowns.py` file parses CSV's produced by the basic and *k*++ implementations, while the `fairlets-cluster-breakdowns.py` file takes a series of variables output by the Basic Fairlets Decomposition section of `fairlets/Fairlets.ipynb`. For each cluster within a clustering output, the scripts produce a percentage breakdown of the categories by the protected attribute for which we clustered. For example, if the protected attribute was sex, the breakdown categories would be male and female, each of which would have a given percentage in each cluster.

Once we generate these breakdowns, we can use them to determine how balanced each clustering was, and ultimately compare these results across different algorithms. We make this comparison simply by calculating the offset between the proportion of the dominant group in each algorithm's worst clustering from the proportion of that dominant group in the greater dataset. The further these two proportions are apart, the less balanced the clustering is. We expand on this process in 5.2.

5 Findings

We produced clusterings for each of our three completed implementations, including sample clustering graphs for basic *k*-means and *k*-means++. Additionally, we produced figures

that compare the clustering balance of *k*-means with the clustering through fairlets algorithm implementation, using gender as the protected attribute.

5.1 Comparing Basic *k*-Means and *k*-Means++

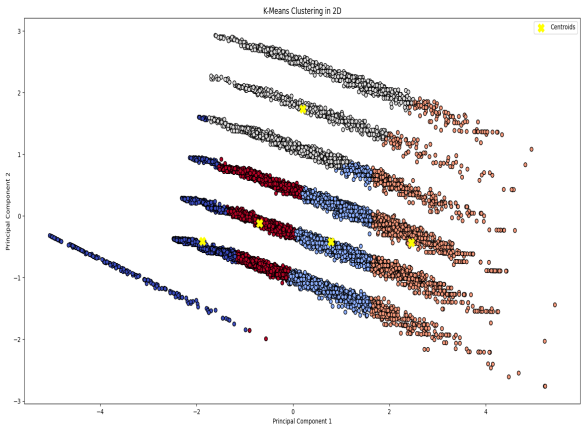

Figure 5A: Visualization of *k*-Means Clustering

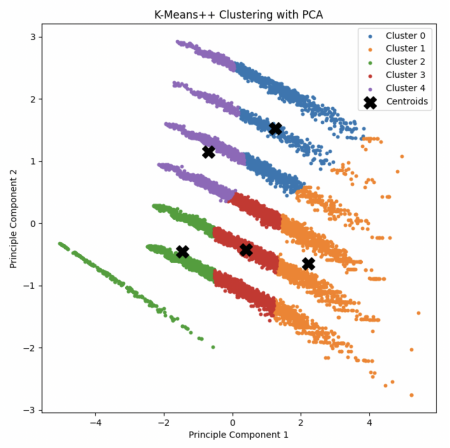

Figure 5B: Visualization of *k*-Means++ Clustering

When comparing these two clusterings, we can see that there is little difference in the final centroid placements (yellow in basic *k*-means and black in *k*-means++) and their resulting clusterings. However we can see that *k*-means++ centroids are a little more spread out across the data space (specifically looking at the top right centroid) making the *k*-means++ clusters more evenly distributed and resulting in the differing clusterings.

When we ran *k*-means++ a total of 10 times, we found that the clusterings were unstable. This is not in line with the advantages that *k*-means++ provides. This could be due to a number of different reasons: higher dimensionality, non spherical clusters, unevenly sized clusters, and the random initialization step at the start of *k*-means++. The reason why *k*-means++ does not work well with higher dimensional data is because as the number of dimensions increases, the distance between two points becomes more similar and less meaningful, therefore making the algorithm less effective. *k*-means++ does not work well with non-spherical clusters and unevenly sized clusters because *k*-means++ tends to choose spherical clusters, and *k*-means++ assumes that each cluster occupies the same volume in space. Since the clusters are both non-spherical and uneven, *k*-means++ ia susceptible to unstable clustering. Finally, *k*-means++ still has a bit of randomness when picking its initial centroid, which can lead to instability in the final clustering.

5.2 Comparing Balance and *k*-Means Costs

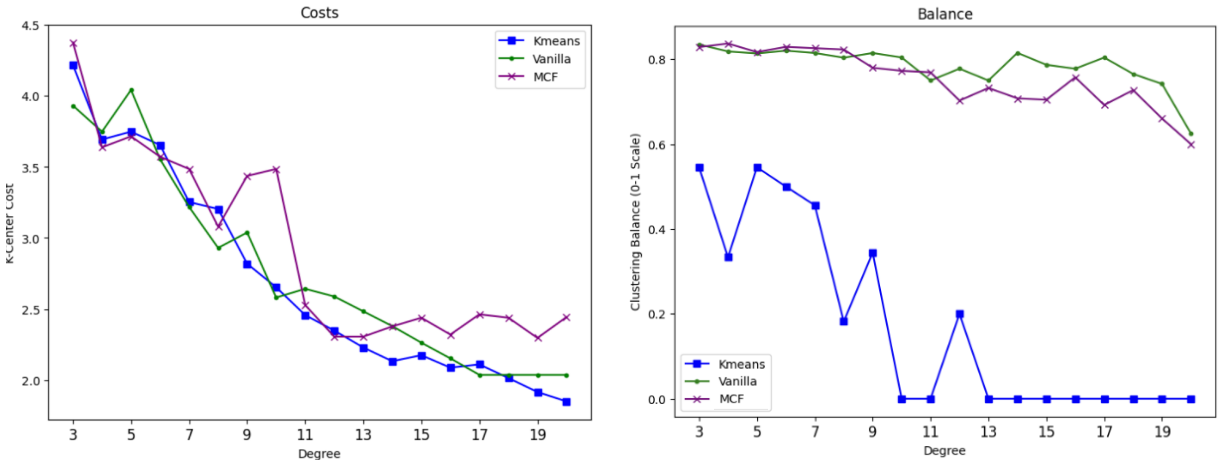

Figure 5C (left): Basic *k*-Means, fairlets, & MCF Compared by Cost and Balance on Binary Fair Clusterings of Sex
Figure 5D (right): Compared by Clustering Balance on Binary Fair Clusterings of Sex

In Figure 5C, we see the *k*-means cost plotted for a singular run per degree, which represents the number of clusters generated. We see that all three variations have a strong negative correlation between these variables, meaning that the more clusters generated, the lower the *k*-means cost (see 2.2.1) was. This outcome was expected, as adding more clusters generally means that each point in the cluster will be closer to its respective center. Thus, points in each cluster will be consistently closer to their respective centers. We also see that MCF had moderately higher costs than the other two algorithms on average, which is sensible given its nature as an approximation of *k*-means clustering that trades off cost performance for runtime.

In Figure 5D, we see clustering balance plotted against degrees, again for a singular run. Clustering balance represents the level to which the original ratios between the protected attributes in the data (in this case male and female) are maintained within the final clusterings. Generally, the two fair algorithms performed much better than the other algorithms, each hovering close to 0.8 and never dropping below 0.6. Meanwhile, the classic *k*-means graph starts below 0.6 and rapidly drops off to 0.0, where it settles after 13 degrees. This was expected, as traditional *k*-means clustering uses no mechanism to enforce balance within clusters. We see that all three algorithms have varying negative correlations between balance and degree, with that of basic *k*-means being the most strongly negative. It is reasonable that a balanced clustering output is generally less likely as degrees increase, given that more clusters causes fewer data points per cluster that are less likely to be representative of the overall data. The two fair algorithms' mild negative slopes (which start to drop off more rapidly in the final region of the domain) indicate

that this effect was softened for fairlet clustering, but not completely. We also see that the MCF fairlets method performed slightly worse than the traditional fairlets method over time, which – again – is expected.

In both plots, but especially in Figure 5D, we see that the graph is non-monotonic. The most striking case appears for the *k*-means algorithm, as there are large deviations from the decreasing pattern when $k = 5$, $k = 9$, and $k = 12$. This is likely due to the random nature of this algorithm's initialization, as well as the fact that each value is obtained from a singular run. If we had taken the average of multiple runs, this graph would probably monotonically decrease and appear smoother.

Table 5A: Worst Case Proportion per Algorithm with Five Clusters

	k-means	*k* ++	Fairlets
Worst Cluster	55.26% Male 44.74% Female	48.6% Male 51.4% Female	54.42% Male 45.58% Female

In Table 5A above, we depict an example output for the *k*-means algorithm, *k*-means ++ algorithm, and clustering via fairlets when $k = 5$. Specifically, we record the proportion of protected attributes belonging to the cluster in the clustering that has the least similar proportion as the original dataset. In this way, we isolate the least fair cluster in the clustering, as this is what determines the output's balance. The original algorithm's proportion of protected attributes was roughly 50.77% male and 49.23% female. Therefore, we see that the *k*-means algorithm unsurprisingly performed the worst. The fairlets algorithm did better at maintaining the balance of the overall dataset, though it was outperformed in this instance by the *k*-means ++ algorithm. This example gives us a wealth of information, as we find that while we expect the *k*-means and *k*-means ++ algorithms to perform the worst – since they were not designed to maintain balance – the randomness of their initializations may lead them to produce more fair clusterings on rare occasions. Nonetheless, we suspect that the fairlet approach would prove to be more reliable when tested many times with multiple *k*-values and on a wide variety of datasets.

6 Conclusion

Through an implementation of Lloyd's algorithm, *k*-means++, clustering through fairlets, and an exploration of socially fair Lloyd's algorithm clustering, we were able to explore the

performance of various algorithms in the context of our dataset about high school student performance and demographics. As mentioned in Section 5, we found that Lloyd's basic algorithm and *k*-means++ yielded very similar clusterings and neither performed consistently across repetitions. We expected more consistency in the results for *k*-means++ due to its theoretical ability to avoid getting stuck in local optima. Some possible explanations for its inconsistent performance are the higher dimensionality clustering space, non-spherical clusters, unevenly sized clusters, and the random initialization step at the start of *k*-means++. As expected, the clustering through fairlets yielded a more balanced clustering when given sex as the protected attribute, aligning with the expected outcome. Despite the unexpected nature of some results and our struggle to complete the implementation of socially fair Lloyd's algorithm, we learned a lot about the various algorithms and their applications in the world of data science.

6.1 Areas for Future Research

There are many areas that we recommend for further research. If given more time, we would certainly prioritize the implementation of the socially fair Lloyd's algorithm in order to analyze its performance on our dataset and compare its performance to the other algorithms we implemented. Moreover, we would like to investigate how fair algorithms deal with datasets containing instances of intersectionality.

So far, we have assumed that each data point can only belong to one demographic group or protected attribute. There are plenty of human-centered applications where this is not the case: To name a few examples, gender is not binary, many people identify with more than one racial group, and the experiences of a disabled, impoverished person cannot be encapsulated by separately analyzing the experiences of a disabled person and an impoverished person; these identities must be examined in conjunction.

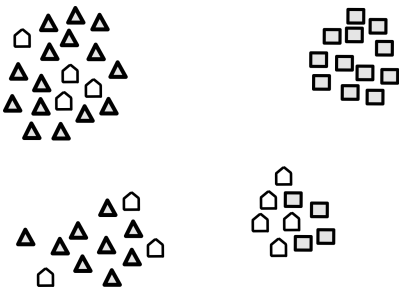

Figure 6A: Toy Data with Intersectional Observations

Investigating the same dataset as in Section 2, it is likely that some of the data points would belong to both the triangle and the square protected attributes (denoted by the pentagons). Thus, a recommendation for future research lies in fair clustering algorithms for datasets with intersectional observations.

7 Acknowledgements

We would like to thank Layla Oesper for guiding us in our journey towards completing this comprehensive exercise and the last requirement for us as seniors. Additionally, we would like to give a special thanks to our families, friends, and other loved ones who supported us through this process.

8 References

Chhabra, Anshuman et al. "An Overview of Fairness in Clustering." *IEEE*, vol. 9, 2021, https://ieeexplore.ieee.org/document/9541160.

Chierichetti, Flavio et al. "Fair Clustering Through Fairlets." *31st Conference on Neural Information Processing Systems*, 2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/978fce5bcc4eccc88ad48ce3914124a2-Paper.pdf.

Chouldechova, Alexandra and Roth, Aaron. "A Snapshot of the Frontiers of Fairness in Machine Learning." *Communications of the ACM*, vol. 63, no. 5, 2020, https://cacm.acm.org/magazines/2020/5/244336-a-snapshot-of-the-frontiers-of-fairness-in-machine-learning/fulltext?mobile=false.

Dabbura, Imad. "K-Means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks." Medium, Towards Data Science, 27 Sept. 2022, https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

Ghadiri, Mehrdad et al. "Socially Fair *k* Means Clustering." *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, https://dl.acm.org/doi/pdf/10.1145/3442188.3445906.

"K-Means Clustering." *Wikipedia*, Wikimedia Foundation, 12 Oct. 2023, https://en.wikipedia.org/wiki/K-means_clustering.

"ML: K-Means++ Algorithm." GeeksforGeeks, GeeksforGeeks, 20 Apr. 2023, www.geeksforgeeks.org/ml-k-means-algorithm/.

Oesper, Layla. *Algorithmic Fairness in Clustering: A Study in Replication*. Carleton College, Fall 2023, https://www.cs.carleton.edu/cs_comps/2324/fairnessInClustering/index.php.

"StatQuest: K-Means Clustering." YouTube, YouTube, 23 May 2018, www.youtube.com/watch?v=4b5d3muPQmA&ab_channel=StatQuestwithJoshStarmer.

Whitfield, Brennan and Pierre, Sadrach. "A Step by Step Explanation of Principal Component Analysis." *builtin*, https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

Wohlenberg, Johannes. "3 Versions of K-Means." *Medium*, Towards Data Science, 2 Apr. 2023, https://towardsdatascience.com/three-versions-of-k-means-cf939b65f4ea.

Mayo, Matthew. "Centroid Initialization Methods for k-means Clustering." KDnuggets, 13 May 2022, https://www.kdnuggets.com/2020/06/centroid-initialization-k-means-clustering.html.

9 Appendix

In this list, we include the libraries that were used in implementing our project.

- NumPy: Used for numerical operations, especially in the calculation of distances and manipulation of data arrays.
- Matplotlib: Employed for creating visualizations, specifically for generating 2D scatter plots.
- Scikit-learn (sklearn): Utilized for preprocessing data and implementing the k-means clustering algorithm.
- Pandas: Used for data manipulation and analysis. Pandas provides data structures like DataFrames, which simplify data handling and manipulation.
- Seaborn: Employed for enhancing the aesthetics of Matplotlib plots.

- CSV: The CSV module is used for reading and writing CSV files, facilitating the loading of the dataset.
- PCA (Principal Component Analysis): Utilized from scikit-learn to perform dimensionality reduction. PCA is employed to reduce the dataset's dimensionality to two principal components for effective visualization.
- Tkinter: Tkinter was utilized to visualize the clustering with a GUI.