Group Assignment 3
Presentation and Demo

Team 5

Jessica Pinto | Dayoung Kim | Mark Haddad | William Garcia

Which dataset did we choose?

- **Dataset Chosen: normalized weather attributes**
  - This dataset contains weather attributes such as ‘Air Temperature’, ‘Relative Humidity’, ‘Air Pressure’, ‘Track Temperature’, ‘Wind Speed’
  - Each feature(attribute) was normalized to ensure similar weight when clustering

- **Description of features chosen**
  - Air Temperature: Temperature of the air, which may influence track conditions
  - Relative Humidity: A measure of moisture in air
  - Air Pressure: Can indicate altitude or weather changes
  - Track Temperature: Surface(track) temperature, which could affect traction
  - Wind Speed: Could affect driving stability

- **The dataset contains numerous data points, where each represents a unique weather update with the above attributes**

K-means

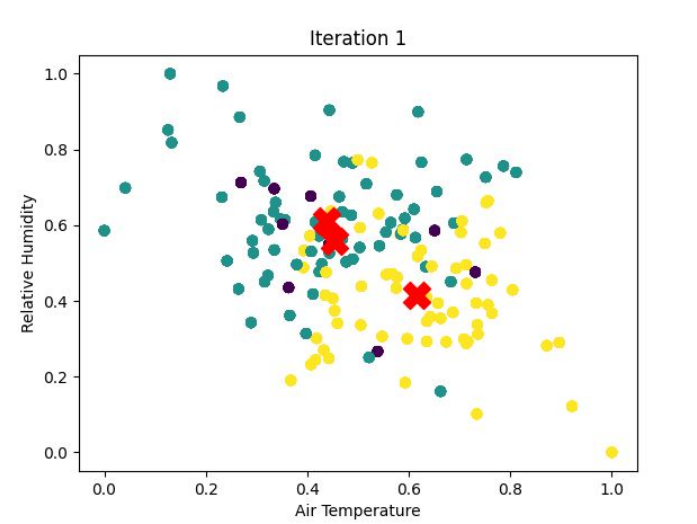
- **Purpose of K-means**
  - K-means is unsupervised learning algorithm organizing data into clusters, each centered around a ‘centroid’
- **Preparation Step**
  - Normalization was performed to ensure that all features are on a similar scale
- **How the code works for K-means**
  - Initialization: Randomly select k data points as the initial centroids, which represent the center of each cluster
  - Assignment: Assign each data point to the nearest centroid based on ***Euclidean distance***
  - Update: Calculate the new centroids by averaging all points assigned to each cluster
  - Convergence check: Repeat the previous two steps until centroids do not significantly change

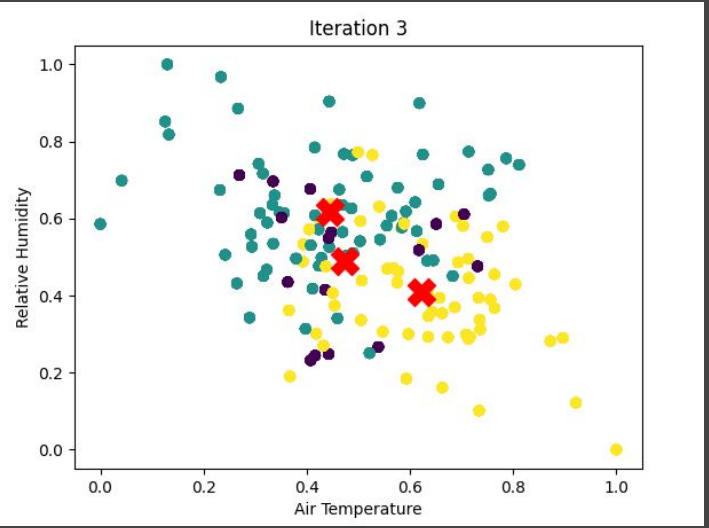
Implementation in `clustering.py`

- Main Functions in `clustering.py`
  - `lloyds`: The main function conducting lloyd's algorithm for K-mans. It arranges cluster assignment and centroid updates
  - `initalizeCentroid`: Initialize centroids by selecting random points from the dataset
  - `assign_cluster`: It calculates the Euclidean distance from each point to each centroid and assassins points to the nearest centroid
  - `Calculate_centroids`: The mean of points in each cluster to update the centroid positions

- Parameters used in `clustering.py`
  - `K`: Determines the number of Clusters
  - `Columns`: Features selected, representing weather related attributes
  - `Convergence`:
    - (1e-4) will stop the iterations when the movement is minimal
    - n is set to 100 by default, stopping the algorithm when it reaches 100 iterations

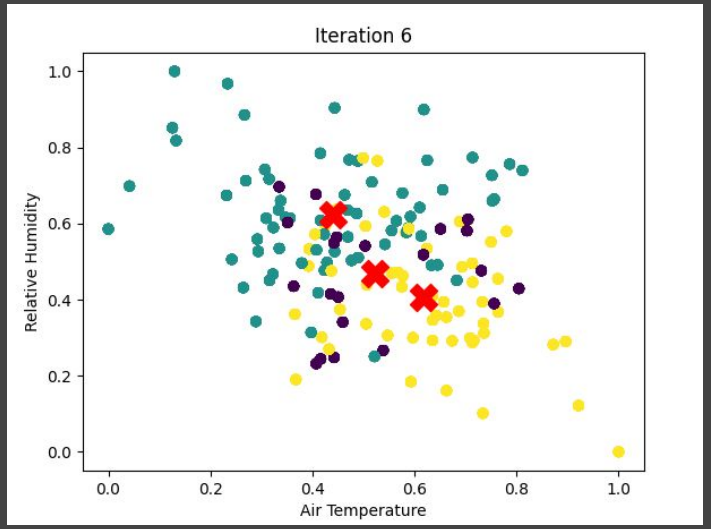
Result of K-means

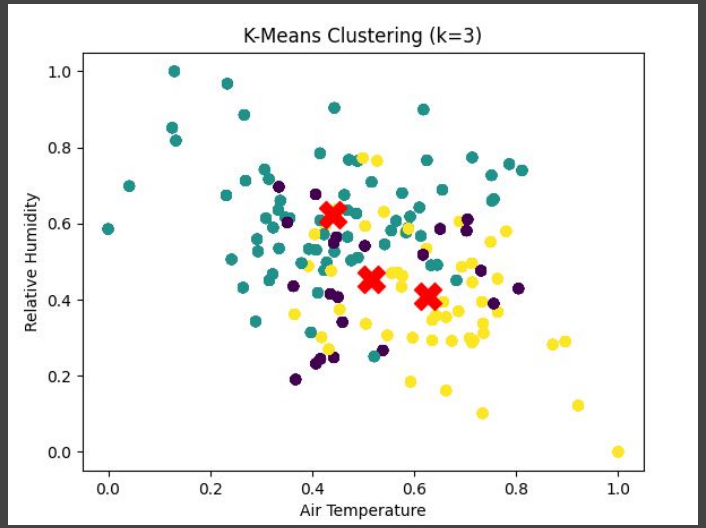




- It shows you the steps where the K-means clustering iterates by reasoning points to clusters and adjusting centroids

Result of K-means





- And stops when the centroid shifts are minimal, showing the stable clusters
  `if max(centroid_shifts) < 1e-4:`

DBSCAN

- **Purpose of DBSCAN**
  - Unsupervised clustering method that can work well with clusters of arbitrary shape, this method model clusters as dense regions separated by sparse regions

- **Preparation Step**
  - Relevant features were normalized
  - Relevant categorical features were mapped to numerical values

- **How the code works for DBSCAN**
  - Input: data points, columns, eps, min_samples
  - Output: Set of density-based clusters
  - Finds core points using n-dimensional Euclidean distance, n is the # of features in a data point, core point if neighbors >= min_samples
  - Iterates through core points, recursively adds neighboring core points to visit, loop breaks when all core points are visited

DBSCAN Implementation

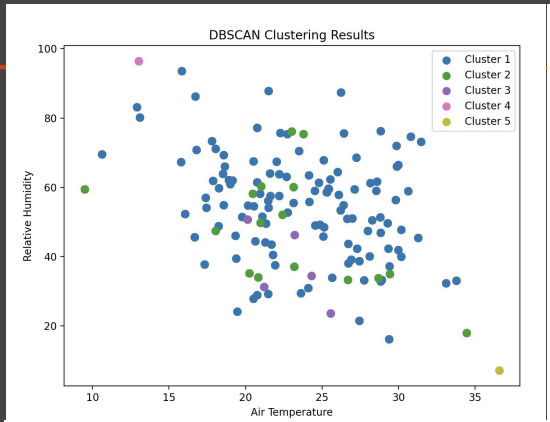
- Helper functions

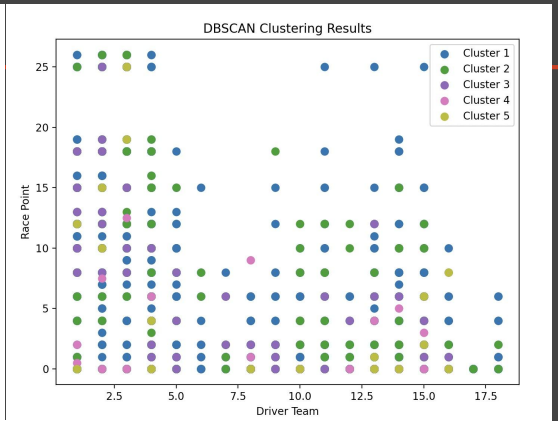
```python
def euclidean_distance(point1, point2):
    return math.sqrt(sum((point1[i] - point2[i]) ** 2 for i in range(len(columns))))


def get_neighbors(point, data):
    neighbors = []
    for other_point in data:
        if euclidean_distance(point, other_point) <= eps:
            neighbors.append(tuple(other_point))
    return neighbors
```

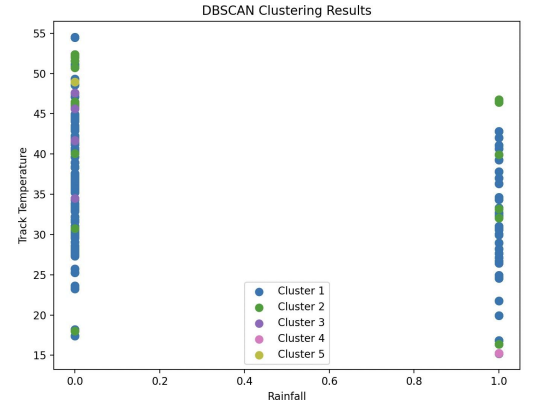
- Recursive function loop

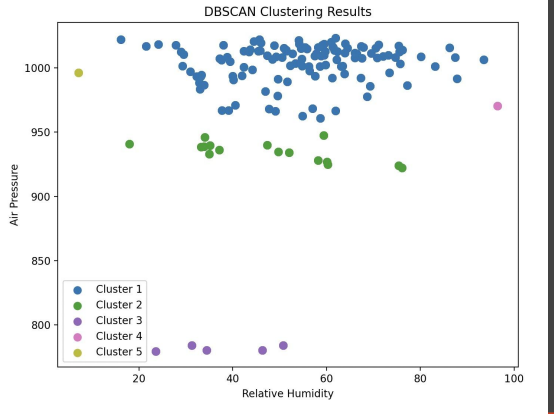
```python
# Identify core points
core_points = [tuple(point) for point in data if len(get_neighbors(point, data)) >= min_samples]

# Expand clusters from core points
visited = set()
clusters = []

for core_point in core_points:
    if core_point in visited:
        continue  # Skip if we've already processed this point

    # Start a new cluster
    cluster = []
    points_to_visit = [core_point]

    while points_to_visit:
        point = points_to_visit.pop()
        if point in visited:
            continue  # Skip points that have already been visited

        visited.add(point)
        cluster.append(point)

        # Get neighbors and expand if they are also core points
        neighbors = get_neighbors(point, data)
        if len(neighbors) >= min_samples:
            # Only expand from core points
            for neighbor in neighbors:
                if neighbor not in visited:
                    points_to_visit.append(neighbor)

    clusters.append(cluster)  # Add the formed cluster to the list of clusters

# Identify and label noise points
noise = [tuple(point) for point in data if tuple(point) not in visited]

return clusters, noise
```

Result of DBSCAN









Result of DBSCAN

- Density-based clusters are formed when distances are calculated in n-dimensions
- For visualization, any two selected features may not fully represent the significance of how the clusters are formed
- Some pairs of features show better separation than others
- If trends are shown in the visuals that are not represented by the actual clusters, we can know further which features we'd want to include or exclude to show a significant separation in the visualization
    - For example, visualizing Race Points over Driver Teams
- Variance in the eps and min_samples parameters can change the silhouette score performance, occurrence of noise points, and number of clusters

Cluster evaluation - code/ approach

- **In evaluating our clusters we utilized an internal method using silhouette coefficients in order to determine the accuracy of our clusters.**

- **The way we implemented it was through separating each part (separation point, cohesion point, and distance) into different functions**
  - After the visualization and DBSCAN the math is done utilizing the formula needed to evaluate this coefficient
    - When using this we used the euclidean distance between points rather than any other distance formula.

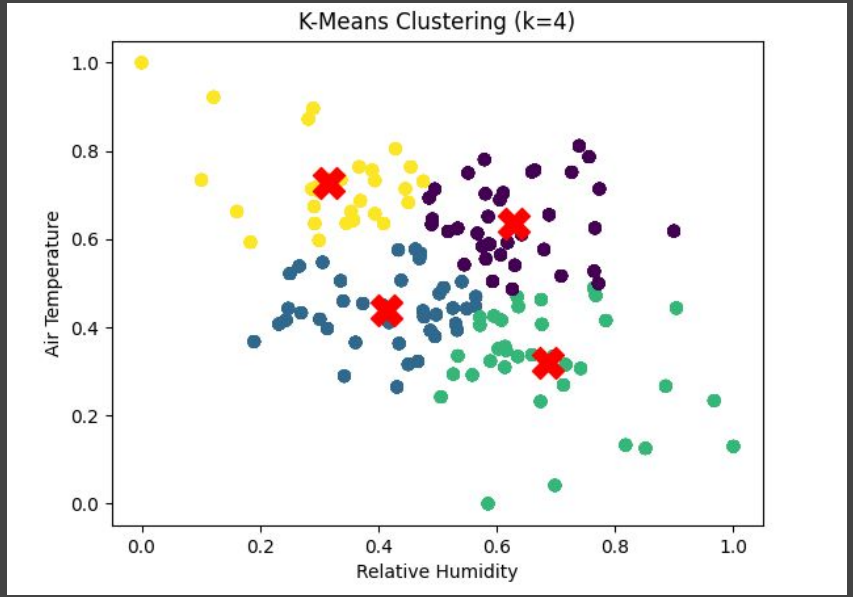
Cluster evaluation - Silhouette Score

- For this dataset, none of our group members felt confident enough with our Formula 1 knowledge to be able to produce the groundwork required for extrinsic evaluation.
- **Instead, we chose to utilize the intrinsic evaluation method of the Silhouette Score.**
  - The silhouette score calculates the distances of each point to the points in its cluster, as well as points in the nearest cluster.
  - For each point, the silhouette coefficient is calculated and stored.
  - At the end, the Silhouette Score is returned as the mean of the Silhouette Coefficients
- **Since our K-means and DBSCAN used varying data types and methods of implementation, we decided to calculate the Silhouette scores of both these algorithms separately.**
- The Silhouette Score should be between -1 and 1, where the higher the score means that points in a cluster are well matched to other points in its own cluster and poorly matched to points in other clusters.

Silhouette Score - K-means

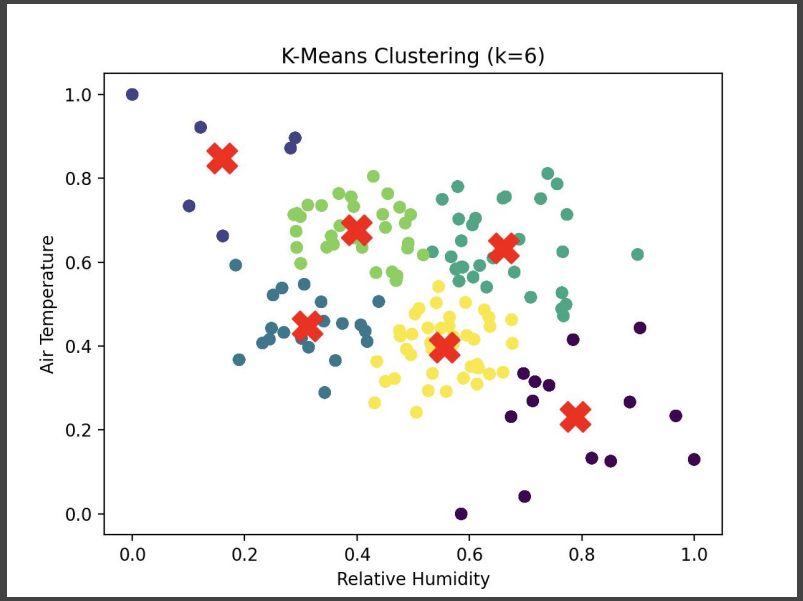
- The ideal **silhouette score for k-means** with our data is around **0.7074016694527617**
- This score is pretty good considering the scale of -1 to 1, so we can say that most of the points in a closer are similar to points in its own cluster, yet different from the points in different clusters
- **What could be the reason for this score?**
  - Looking over this there are a lot of points that do seem to skew the data away from ideal clusters
    - Since k-means uses the mean, it would make sense that outliers highly affect the clusters
  - Perhaps a better k value would increase the score
    - We'll take a look at these next

Silhouette Score - K-means



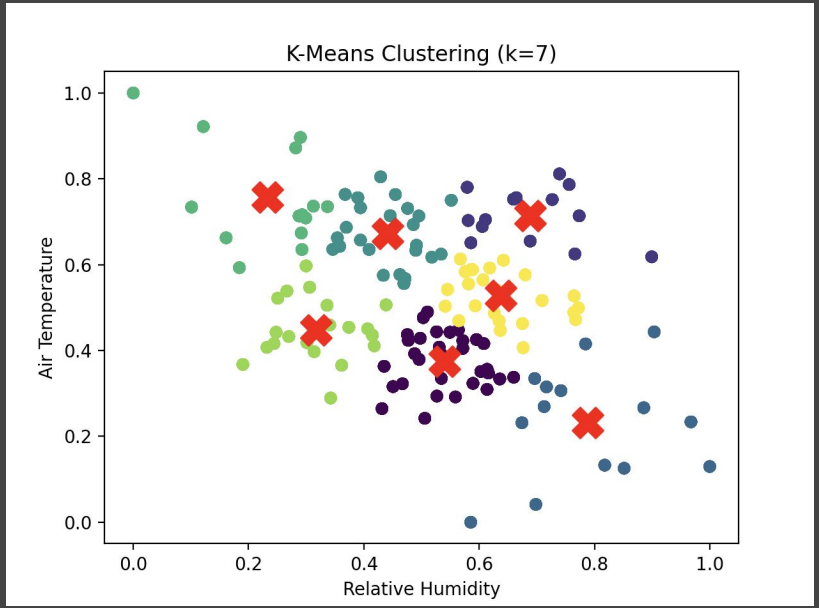
Silhouette Score: 0.4722043588536139

Silhouette Score - K-means



Silhouette Score: 0.6672649884420954

Silhouette Score - K-means

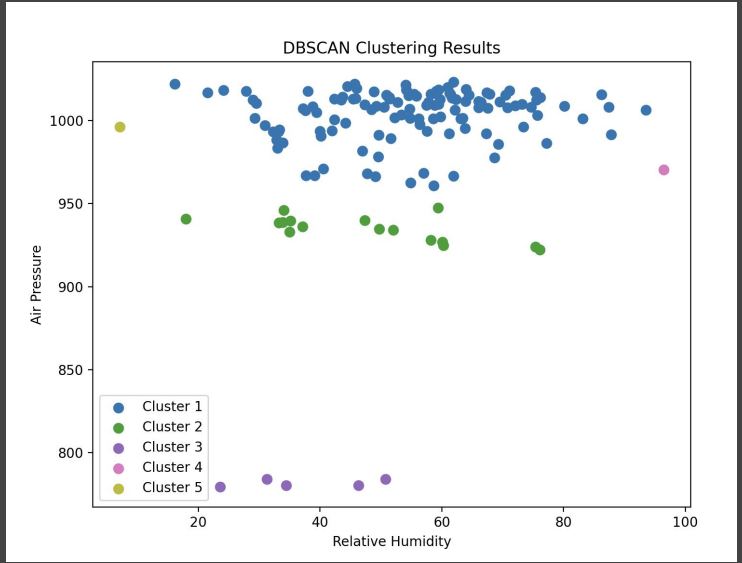


Silhouette Score: 0.515639961143841

Silhouette Score - DBSCAN

- As for DBSCAN, we calculated a Silhouette Score of around **0.8088293062193368**
- This score is pretty high considering the scaling of the Silhouette Score (recall, range of -1 to 1), and we can say that the points in the clusters are well matched
- What could be the reason for this score?
  - The radius for each core point (eps) could be optimized to include more similar points
  - The number of minimum points in each radius could be optimized in the same way
- Altering the epsilon for the DBSCAN algorithm allowed us to find the greatest Silhouette Score for our data
  - We will take a look at some examples of this

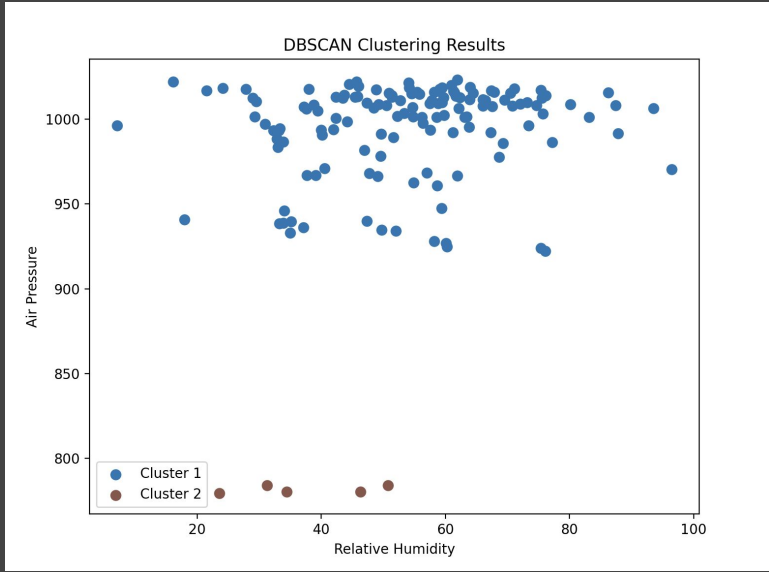
Silhouette Score - DBSCAN



Silhouette Score: 0.36333698959419797

eps: 20

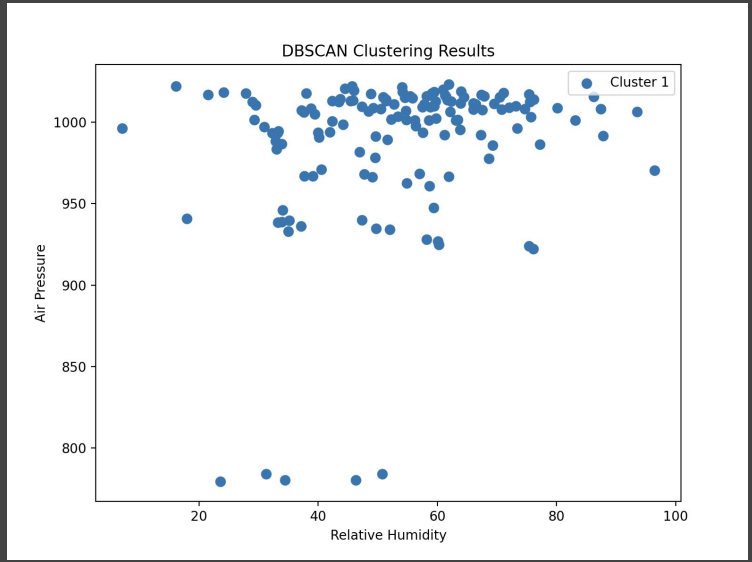
Silhouette Score - DBSCAN



Silhouette Score: 0.8088293062193368

eps: 30

Silhouette Score - DBSCAN



Silhouette Score: -1.0
eps: 200

Expectations vs. Reality

Expectations	Reality
• Initially expecting that normalization may not be necessary and could distort the natural structure of the data	• Applying normalization turned out to be important in preventing distortion, ensuring that all features contributed equally
• When looking at the scatterplot of the k-means data, we expected that the ideal k would be 4	• Our Silhouette Score with k = 4 proved this theory wrong, as we found out k = 6 produced better clusters

Issues/struggles

- The implementation of k-means proved to a lot more difficult than expected
- Issues with the actual silhouette coefficient came down to just getting a high enough score.
  - While the lower score may be errors with data collection it is something that we will have to keep in mind in order to ensure that further clustering is accurate.
- We kept encountering an issue `operand type(s) for -: 'float' and 'list'` with the data types used in dist function within `testcases.py`. It might be because one of values is a float and another one is a list
  - The issue is most likely due to mismatched data types
  - We will fix the code to access the correct level in the data structure

Thank You