**Third Project Report**

**Introduction**

Problem definition : Our objective is to grouping player based on some of their features like overall, potential, their value and ages and so on.

**Algorithms implemented**

* **KMeans**

This code block is using the K-Means clustering algorithm from the `sklearn.cluster` module to group similar data points together. Here's a step-by-step explanation:

1. The necessary libraries are imported. `KMeans` for clustering, `LabelEncoder` for encoding non-numeric data, and `matplotlib.pyplot` for plotting.

2. The DataFrame `df` is cleaned by dropping rows with missing values using `df.dropna(inplace=True)`.

3. The non-numeric columns in the DataFrame are identified using `df.select\_dtypes(exclude=['float64', 'int64']).columns`.

4. The non-numeric columns are then transformed into numeric form using a `LabelEncoder`. This is done because K-Means algorithm requires numeric data.

5. The K-Means clustering algorithm is applied to the DataFrame. The number of clusters is set to 4 (`n\_clusters=4`), and the random state is set to 42 for reproducibility.

6. The `fit` method is called on the DataFrame `df` to compute K-Means clustering.

7. The labels of each point are stored in `labels`, and the coordinates of cluster centers are stored in `centroids`.

* **AgglomerativeClustering**

This code block is using the Agglomerative Clustering algorithm from the `sklearn.cluster` module to group similar data points together. Here's a step-by-step explanation:

1. The necessary libraries are imported. `AgglomerativeClustering` for clustering, `LabelEncoder` for encoding non-numeric data, and `matplotlib.pyplot` for plotting.

2. The DataFrame `df` is cleaned by dropping rows with missing values using `df.dropna(inplace=True)`.

3. The non-numeric columns in the DataFrame are identified using `df.select\_dtypes(exclude=['float64', 'int64']).columns`.

4. The non-numeric columns are then transformed into numeric form using a `LabelEncoder`. This is done because the Agglomerative Clustering algorithm requires numeric data.

5. The Agglomerative Clustering algorithm is applied to the DataFrame. The number of clusters is set to 4 (`n\_clusters=4`).

6. The `fit\_predict` method is called on the DataFrame `df` to compute Agglomerative Clustering and predict the cluster labels for each data point. These labels are stored in `labels\_agg`.

7. The clusters are visualized using a scatter plot. The x and y coordinates correspond to the first and second features of the data points (i.e., `df.iloc[:, 0]` and `df.iloc[:, 1]`), the color of the points is determined by their cluster label (`c=labels\_agg`), and the color map is set to 'viridis'.

* **GaussianMixture**

This code block is using the Gaussian Mixture Model (GMM) from the `sklearn.mixture` module to group similar data points together. Here's a step-by-step explanation:

1. The necessary libraries are imported. `GaussianMixture` for clustering, `LabelEncoder` for encoding non-numeric data, and `matplotlib.pyplot` for plotting.

2. The DataFrame `df` is cleaned by dropping rows with missing values using `df.dropna(inplace=True)`.

3. The non-numeric columns in the DataFrame are identified using `df.select\_dtypes(exclude=['float64', 'int64']).columns`.

4. The non-numeric columns are then transformed into numeric form using a `LabelEncoder`. This is done because the GMM algorithm requires numeric data.

5. The Gaussian Mixture Model is applied to the DataFrame. The number of components (clusters) is set to 4 (`n\_components=4`), and the random state is set to 42 for reproducibility.

6. The `fit\_predict` method is called on the DataFrame `df` to compute the GMM and predict the cluster labels for each data point. These labels are stored in `labels\_gmm`.

7. The clusters are visualized using a scatter plot. The x and y coordinates correspond to the first and second features of the data points (i.e., `df.iloc[:, 0]` and `df.iloc[:, 1]`), the color of the points is determined by their cluster label (`c=labels\_gmm`), and the color map is set to 'viridis'.

* **DBSCAN**

This code block is using the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm from the `sklearn.cluster` module to group similar data points together. Here's a step-by-step explanation:

1. The necessary libraries are imported. `DBSCAN` for clustering, `LabelEncoder` for encoding non-numeric data, and `matplotlib.pyplot` for plotting.

2. The DataFrame `df` is cleaned by dropping rows with missing values using `df.dropna(inplace=True)`.

3. The non-numeric columns in the DataFrame are identified using `df.select\_dtypes(exclude=['float64', 'int64']).columns`.

4. The non-numeric columns are then transformed into numeric form using a `LabelEncoder`. This is done because the DBSCAN algorithm requires numeric data.

5. The DBSCAN clustering algorithm is applied to the DataFrame. The radius of the neighborhoods is set to 0.5 (`eps=0.5`), and the minimum number of points in a neighborhood to define a cluster is set to 5 (`min\_samples=5`).

6. The `fit\_predict` method is called on the DataFrame `df` to compute DBSCAN clustering and predict the cluster labels for each data point. These labels are stored in `labels\_dbscan`.

7. The clusters are visualized using a scatter plot. The x and y coordinates correspond to the first and second features of the data points (i.e., `df.iloc[:, 0]` and `df.iloc[:, 1]`), the color of the points is determined by their cluster label (`c=labels\_dbscan`), and the color map is set to 'viridis'.

* **SpectralClustering**

This code block is using the Spectral Clustering algorithm from the `sklearn.cluster` module to group similar data points together. Here's a step-by-step explanation:

1. The necessary libraries are imported. `SpectralClustering` for clustering, `LabelEncoder` for encoding non-numeric data, and `matplotlib.pyplot` for plotting.

2. The DataFrame `df` is cleaned by dropping rows with missing values using `df.dropna(inplace=True)`.

3. The non-numeric columns in the DataFrame are identified using `df.select\_dtypes(exclude=['float64', 'int64']).columns`.

4. The non-numeric columns are then transformed into numeric form using a `LabelEncoder`. This is done because the Spectral Clustering algorithm requires numeric data.

5. The Spectral Clustering algorithm is applied to the DataFrame. The number of clusters is set to 4 (`n\_clusters=4`), and the random state is set to 42 for reproducibility.

6. The `fit\_predict` method is called on the DataFrame `df` to compute Spectral Clustering and predict the cluster labels for each data point. These labels are stored in `labels\_spectral`.

7. The clusters are visualized using a scatter plot. The x and y coordinates correspond to the first and second features of the data points (i.e., `df.iloc[:, 0]` and `df.iloc[:, 1]`), the color of the points is determined by their cluster label (`c=labels\_spectral`), and the color map is set to 'viridis'.

**Conclusion**

Finally we are successfully able to grouping the players after applying some preprocessing and machine learning algorithms