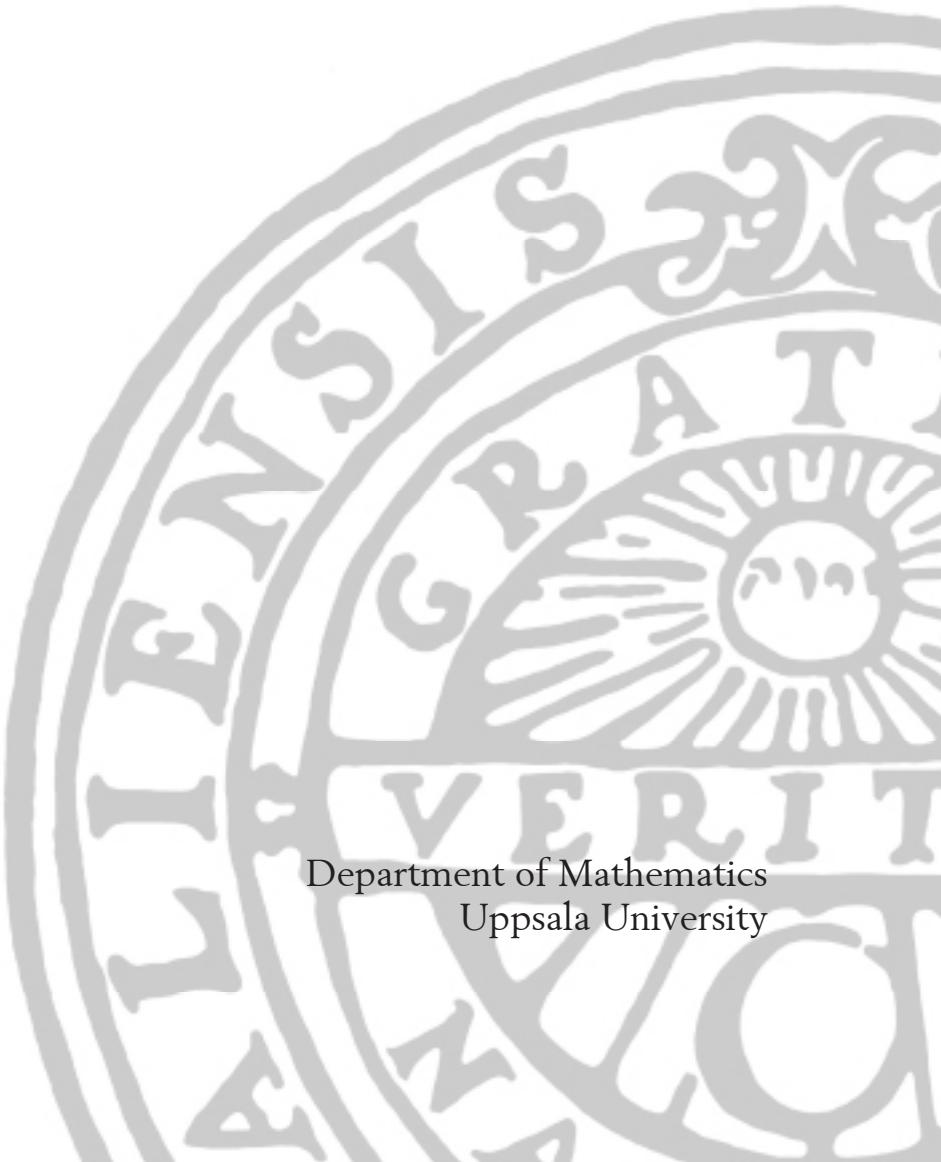
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2022:63

Classifying football players using cluster analysis

Jimmy Jansson

Examensarbete i matematik, 15 hp
Handledare: Silvelyn Zwanzig
Examinator: Martin Herschend
Maj 2022



Department of Mathematics
Uppsala University

Uppsala university

1MA193

Degree Project C in Mathematics

Classifying football players using cluster analysis

Author:
Jimmy Jansson

Supervisor:
Silvelyn Zwanzig

May 19, 2022

Contents

Abstract

This thesis attempts to apply cluster analysis to the game of football where a set of football players were clustered into subsets based on data gathered over a season. Two different types of clustering algorithms were used and the results from both these methods were then looked at discussed. A couple of methods of choosing the number of clusters for the input in the k-means algorithm was also compared with each other to determine a suitable number of clusters for the data set. Looking at the result from the clustering algorithms and also the methods of choosing the number of clusters its obvious that the used data set it homogeneous and no distinct clusters could be found (using the methods in the thesis). It is still possible to draw conclusions from the result but as cluster analysis is often used initially in data exploration there is no need for a real conclusion and the result can still be interpreted differently based on how the result is looked at.

Introduction

Exploring data is becoming more and more relevant today where a lot of data driven decisions are being made. Being able to summarize and group data sets in a meaningful way to be able to make the most rational decision is then more important then ever. Data exploration is one of the initial approaches when understanding a data set and one way of exploring the data is by dividing the data into clusters to see how it is organized. In this thesis we are going to explore some methods used to cluster data. Different methods are going to be used and the result will then be evaluated against each other to find pros and cons. We will apply these methods on data from the German football league during the season 2020/21 where we will attempt to classify players in different playing styles.

Short on Cluster analysis

Cluster analysis[3] is a multivariate analysis method where the aim to divide a set of observations into subsets where the observations in each set have similar features. Before applying it to our data we'll go through the basic concepts and our assumptions.

In order to be able divide the set based in similarity, a distance measure will have to be defined.
The choice of this measure will define the shape of the clusters as the similarities will be calculated differently. This measure of similarity will be done using the distance between each observation. We recall the Euclidean distance (straight-line) which is defined as the following:

$$d_{euc}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Here d_{euc} is the distance between the points x and y in our data set where i $\in$ I and n the total amount of variables for each point. We call I our index set.

The Euclidean distance is what will be used as the distance measure as the variables which will be used here are all continuous.

In an attempt to be able to compare all our different variables and for them to contribute equally we will have to normalize our data. Since we do not have any major outliers we will be doing this using the Z score which is defined as following:

$$Z = \frac{x - \overline{x}}{\sigma}$$

The Z-score is calculated individually for each value and player where x is the initial value we want to convert, μ and σ is the sample mean respectively the sample standard deviation for the sample in the data set.

As we have a lot of different variables we will need to reduce the dimensions so that the result of the clusters will be understandable to the human eye. To do this we will use Principal Component Analysis and reduce our initial matrices to two dimensions so that we can use a scatter plot to visualize our results.

Hierarchical clustering

Hierarchical clustering is type a method of clustering that either starts with a series of clusters (where each point is their own cluster) or as one cluster which includes all points. The former is called a agglomerative hierarchical method where each point is put in its own cluster at the beginning of the algorithm. The two closest clusters based on the initial distance matrix (using the chosen metric) are then merged together and we then have fewer clusters then before. The distance matrix is then recalculated using the newly created clusters and the last step is then repeated to merge the two closest clusters. This is then repeated until we have one cluster which includes all points.

We will be using a agglomerative method with complete linkage (maximum distance is used before merging each point) to group our data set due to it being a popular method and also fits our data set which contains continuous values. Complete linkage avoids the drawbacks of an alternative method; the single linkage method. This phenomena that we avoid by using complete linkage is the "chaining"-phenomenon. This basically causes groups formed through single linkage to bunch together because of single points being near one another, despite the fact that a large number of the points in each group might very far off to each other.

The other method where we start with one initial cluster with all points included is called a divisive hierarchical clustering method and works in the opposite way of the agglomerative method. Instead of using the distance to find the closest clusters, we use it to find the two clusters which are furthest apart and then divide them based on that distance.

The result of each clustering algorithm can be displayed as a dendrogram and will visualize each merger/division at each level of the algorithm. A dendrogram is what we will use in our result to visualize the clusters formed from our agglomerative method.

Non-hierarchical clustering

Non-hierarchical methods do not follow the same "tree structure" as we do not follow the same hierarchical methods where the clusters are either merged or divided.

A popular non-hierarchical cluster method is called "k-means clustering" which uses a predetermined number K to classify our variables into K clusters. This method adds a number K randomly places variables in our set (called centroids) and then assigns each variable in our set to the closest centroid. We will then have k different clusters when each variable has been assigned to a centroid. In our next iteration of the algorithm, we determine a new set of centroids by calculating the mean of each cluster and redo the first step. This is then repeated until there are no more reassignments between the first and second step.

As no distance matrix is saved between iterations K-means is then often used when computing large data sets as the amount of iterations and is much lower compared to the hierarchical methods.

Methods and data

Data collection

The source of the data comes from FBRef[1] where we initially have a matrix with dimensions 337x92. That is, 337 football players which have played more than 450 minutes during the season and are not goalkeepers. As goalkeepers are not involved

in the same way as other outfield players are, we will not include them in our data set. Players that have played less then 5 games (450 minutes as one game is 90 minutes) during the season will also be excluded as the sample of games is to low.

The actual data consists of different observations during football games in the football league Bundesliga (1st division in Germany). The data is from last season 20/21 where each team will have played 34 games (as players may be injured or not on the pitch during the game everyone will not have 34 games played). The data set includes every available variable from our source of data. This includes all different actions made by the players which is countable by the human eye.

In order to attempt to weight each variable as equal as possible we will also remove the highly correlated variables. This was done using a correlation matrix and then simply looking at highly correlated variables and then removing the most correlated variable from the data set. If two (or more) variables have a rang correlation coefficient of >0.80, we keep the variable with the highest correlation coefficient and remove the others from our data set.

Using the method above we reduce our matrix to 337x51 (instead of 337x92) and instead have 51 different variables which describes each football player in the following table:

Goals	Goal creating action	Pressing made in attacking 3rd
Assists	Dead ball pass led to shot	Blocks made
Yellow Cards	Dribble led to goal	Shots blocked
Red Cards	Shot that led to new goal-scoring shot	Shots on target blocked
Expected number of goals	Foul drawn led to goal	Passes blocked
Expected number of assists	Defensive action led to goal	Interceptions
% of shoots on target	Tackles made	Clearences
Goals per shots on target	Tackles won	Mistakes led to shot
% of passes completed	Tackles made in defensive 1/3	Succesful dribbles
% of short passes completed	Tackles made in middle 1/3	Percentage of succesful dribbles
# of long passes completed	Tackles made in attacking 1/3	Nutmegs completed
In-play passes led to shot	Tackles made on dribble	Distance covered with ball
Dead ball pass led to shot	Tackles attempted on dribble	Distance covered with ball towards goal
Dribble led to shot	Percentage of dribblers tackled	Ball carried into attacking 1/3
Shot that led to new shot	Pressing made	Ball carried into penalty area
Foul drawn led to shot	Percentage of succesful pressing	Ball lost during carry
Defensive action led to shot	Pressing made in defensive 3rd	Target of pass

Note that the table is not sorted in any way and just shows what variables are included in the analysis.

We will then use and compare the results between complete-linkage hierarchical clustering and the non-hierarchical K-means method. These were chosen mainly due to them being the easiest to implement and also for being the most popular methods when using continuous variables.

The data set has also been divided into subsets based on the 3 different positions that each player has been classified by their team that they belong to. These positions are forward, midfielder and defender which is what each team included in the football league have each player registered as. The K-means algorithm and hierarchical clustering will then be used on each subset and also the full set to see if we may get a different result if the set is divided. These results will then be compared to see if the result is better or worse depending on how our clustering algorithm has divided our data sets.

Metrics

We will define the distance between each value as the Euclidean distance. This is the most used distance for similar cluster analyses and is also the easiest to implement as our data set consists of only continuous variables. Sneath and Sokal[5] mentions that the simplest coefficient applicable to the data set should be chosen in order to simplify the task of understanding the result which in our case would be using the euclidean distance.

Methods of choosing number of clusters

We will look at a few different methods of choosing the number of clusters to which we group our initial set and then compare the different results after using the K-means algorithm / Single linkage-method. In order to attempt to quantify the result (even though it will most likely be subjective and hard to put a number on the result) we will look at the variance of each cluster.

Elbow method

The "Elbow"-method[6] is one of the most used methods when determining the number of clusters in non-hiearchical clustering. The idea behind the method is

by choosing the number of clusters K based on when there is a diminishing return of variance after iterating over the number of clusters. Plotting this would give us something that looks like an arm where the number of clusters would be on the X-axis and the variance on the Y-axis. The number of clusters would then be where the elbow is located in the graph. The downside of this method is due to it being heuristic and the result of the method is subjective due to there not being a set definition of what number K is supposed to be. The plot can then have different conclusions depending on who is looking at it.

Silhouette method

When using the silhouette method[4] to choose the number of clusters K we look at the difference between the average distance within the cluster and the minimum distance between the clusters. This is given as following:

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

Here, a(i) is the average distance of all other observations inside a cluster for a observation i and b(i) is the average distance of all other observations in other clusters. This then gives us the silhouette coefficient for one point i. We will instead use the coefficient for the whole data set which is simply the average of all silhouette coefficients for each observation.

Clustering metrics

Before the next two methods we will define two metrics of cluster evaluation. These metrics can be used to measure the compactness both within of a cluster and between clusters. We call these metrics "Within-Cluster scatter matrix" (WCSM) and "Between-Cluster scatter matrix" (BCSM). Before defining these we also define the scatter matrix S_K for the K'th cluster in the set as following:

$$S(K) = \sum_{x \in C_K} (x - \mu_K)(x - \mu_K)^T$$

Where x is a member of the cluster c_K and μ_K is the centroid for each cluster. The Within-cluster scatter matrix is then defined as the summation of the scatter matrix

for all clusters in the set:

$$WCSM = \sum_{k=1}^{K} S(K) \tag{1}$$

We also define the Between-cluster scatter matrix as:

$$BCSM = \sum_{k=1} N_K(\mu_K - \mu)(\mu_K - \mu)^T \tag{2}$$

Here N_k is the total number of points included in the kth cluster and μ is the mean of the whole data set. We will then use both of these metrics when defining the Calinski-Harabasz index and the Gap-statistic.

Calinski-Harabasz Index

The Calinski-Harabasz Index (also known as the Variance ratio Criterion) is both used as a clustering performance index and a method of choosing the number K in a data set. The index is defined as the ratio of the sum between-cluster dispersion and of the inter-cluster dispersion for all clusters;

$$CH(k) = \frac{tr(BCSM)}{k-1} \frac{n-k}{tr(WCSM)}$$

Where n is the number of points in our data set, k the number of clusters, tr(BCSM) (see equation 2) is the trace of the between-cluster scatter matrix and tr(WCSM) (see equation 1) is the trace of the within-cluster scatter matrix. This is then calculated for each value of k and we want to maximize the CH-index to find the optimal number of k.

Gap-statistic method

The Gap-statistic, developed by Tibshirani[7] is a more recent idea compared to the other methods already mentioned. The idea behind the Gap-statistic and using it to find the number of clusters in data set is to compare the cluster compactness with an null reference distribution (generated in Python) of the data. What we want do is to first choose the number K and then look at at the within-cluster distance based on the overall behavior of our null reference distribution. The Gap-statistic is defined as following:

$$G(k) = Log(WCSM^{null}) - Log(WCSM^{data})$$

Here, $WCSM^{data}$ (see equation 1) is the within-cluster scatter matrix of our data set and $WCSM^{null}$ is a the within-cluster scatter matrix of a generated null reference distribution as earlier mentioned.

So the basic idea of the Gap-statistic is to find the number K where it is maximized, e.g. where the biggest jump in the within-cluster distance has occured. If there would be a case where the Gap-statistic keeps increasing and does not have a local maxima, Tibshirani mentions the 1-standard-error method such that (informally) K is instead chosen as the point where the increments of the Gap-statistic is slowing down.

Dendrogram (Hierarchical)

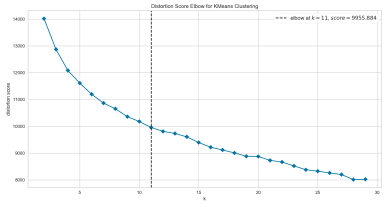
A Dendrogram[2] is specific to hierarchical clustering methods and is a visualization of the joining of each point for each step in the clustering algorithm. The final result is a tree-like structure which shows which points are being joined together.

We already know that the number of clusters is not needed as an input in hierarchical methods and the dendrogram is pretty much a result of the clustering method. We can still use it to determine the number of clusters by looking at the distance from the "initial" step of the algorithm and determining a optimal distance. Note that there is no method of choosing this distance and it depends on the researcher on setting this at an optimal value.

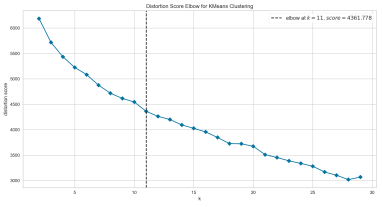
Results

The presented plots below are the results of the python-code shown in the Appendix section. We first normalized the values using the Z-scores and then removed highly correlated values (rank>0.80). All methods regarding choosing the cluster number mentioned earlier where then applied on the data for the K-means algorithm. Using PCA, the dimensions of the data set was then reduced so that it is possible to show the result in a scatter plot. As PCA is just used to visualize the result, we do not look at the cumulative variance. These are the results from the clustering algorithms and the mentioned methods of choosing the number of clusters from our data set of football players. Each section shows the result from the full data set and then the results from the sub sets based on the players position on the field. For the input of the number of clusters in the K-means algorithm, we have used the result from the Gap-statistic plot where we will discuss the reasoning later. The colour of each data point in figure 5 shows which clustering they belong to and the red crosses are the centroids of each cluster.

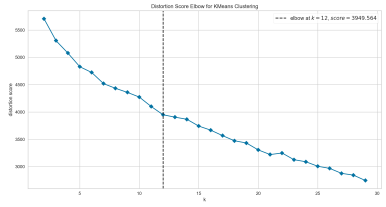
Methods of choosing k



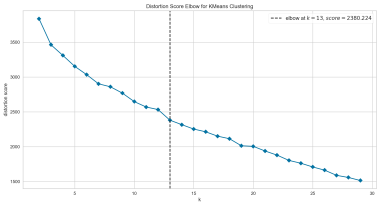
(a) Full data set



(b) Defenders

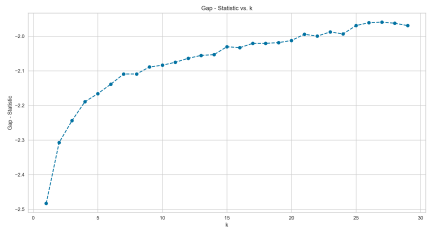


(c) Midfielders

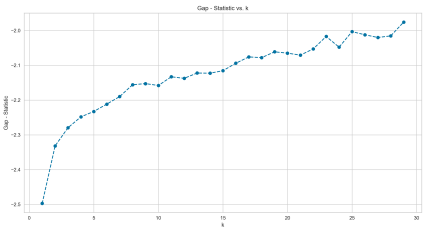


(d) Forwards

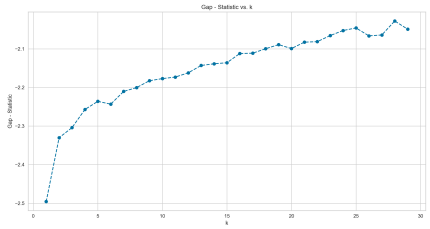
Figure 1: Elbow plot



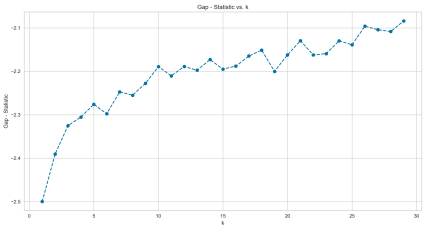
(a) Full data set



(b) Defenders

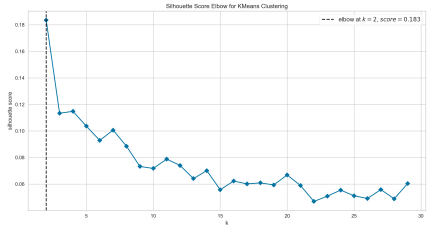


(c) Midfielders

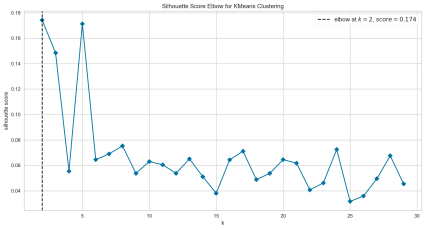


(d) Forwards

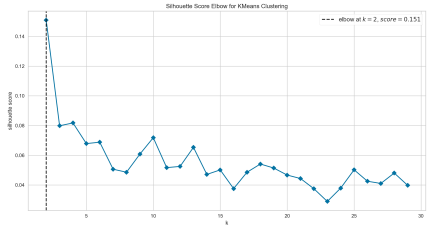
Figure 2: Gap-statistic



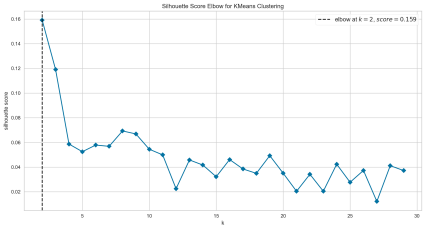
(a) Full data set



(b) Defenders

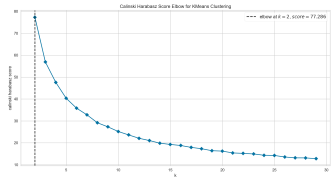


(c) Midfielders

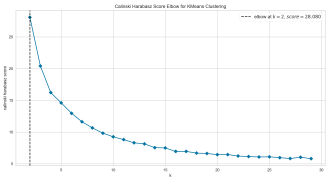


(d) Forwards

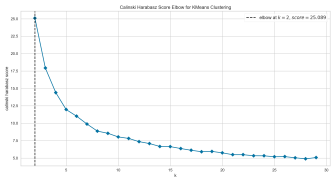
Figure 3: Silhouette plot



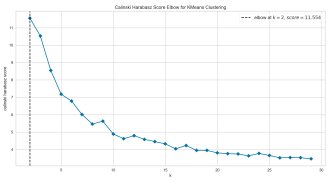
(a) Full data set



(b) Defenders



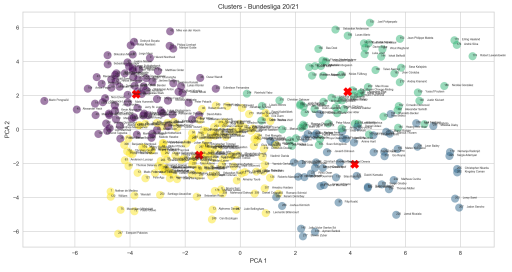
(c) Midfielders



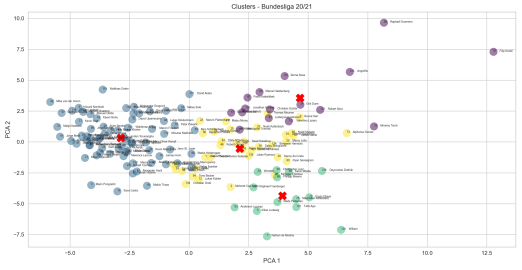
(d) Forwards

Figure 4: Calinski-Harabasz index

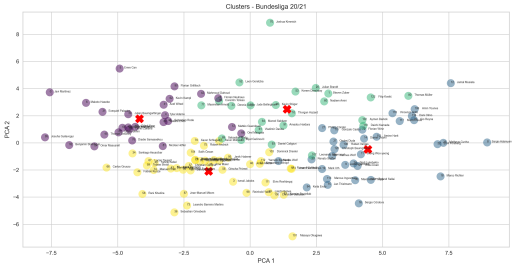
Clustering algorithms



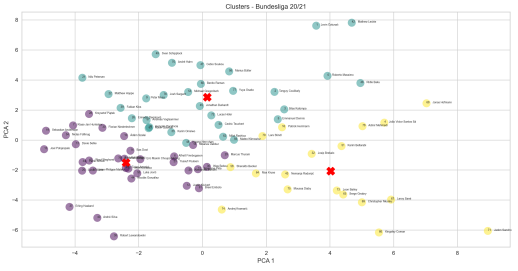
(a) Result for full data set



(b) Result for defenders



(c) Result for midfielders



(d) Result for forwards

Figure 5: Result from K-means algorithm with input of k from Gap-statistic

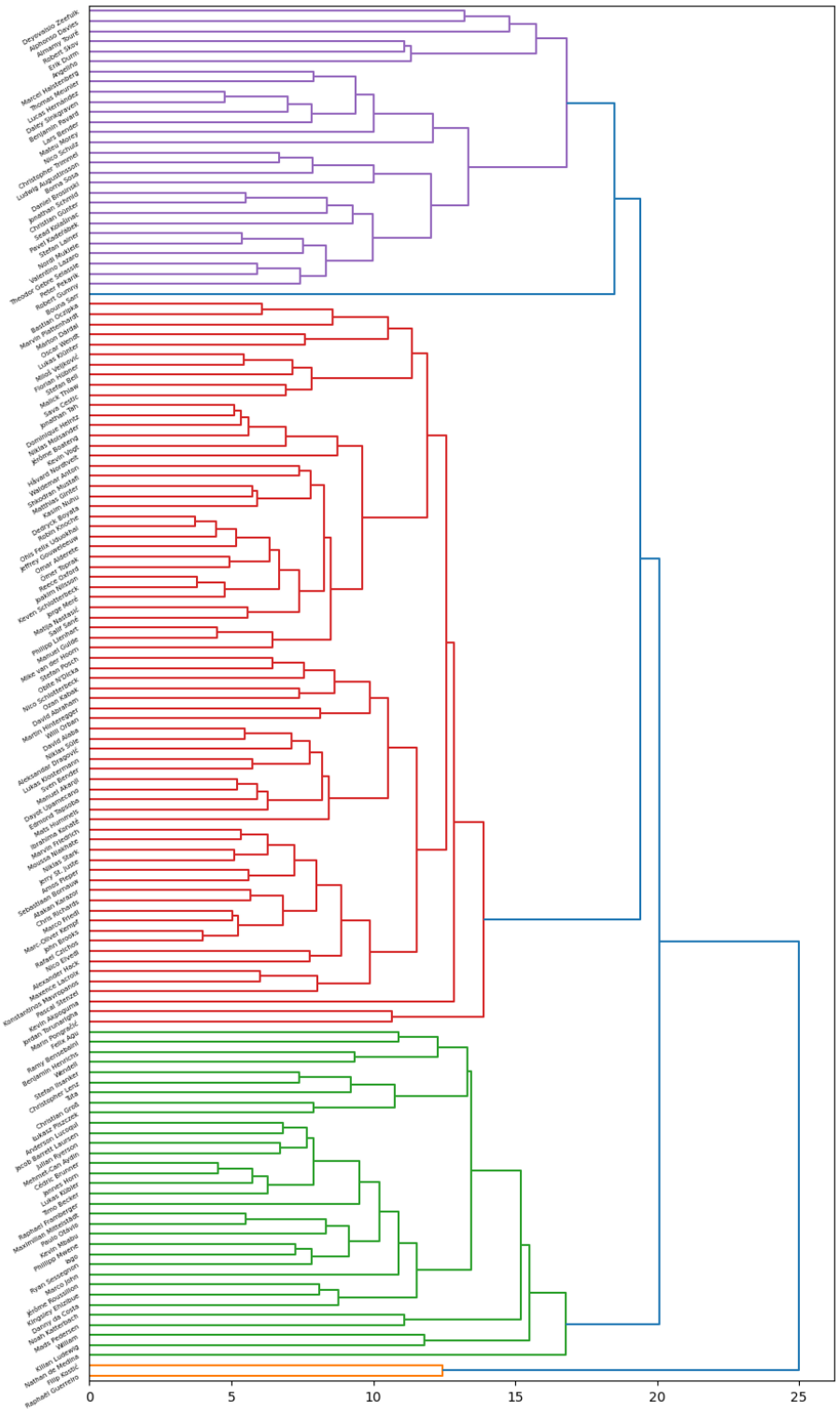


Figure 6: Dendrogram for defenders, Complete-linkage

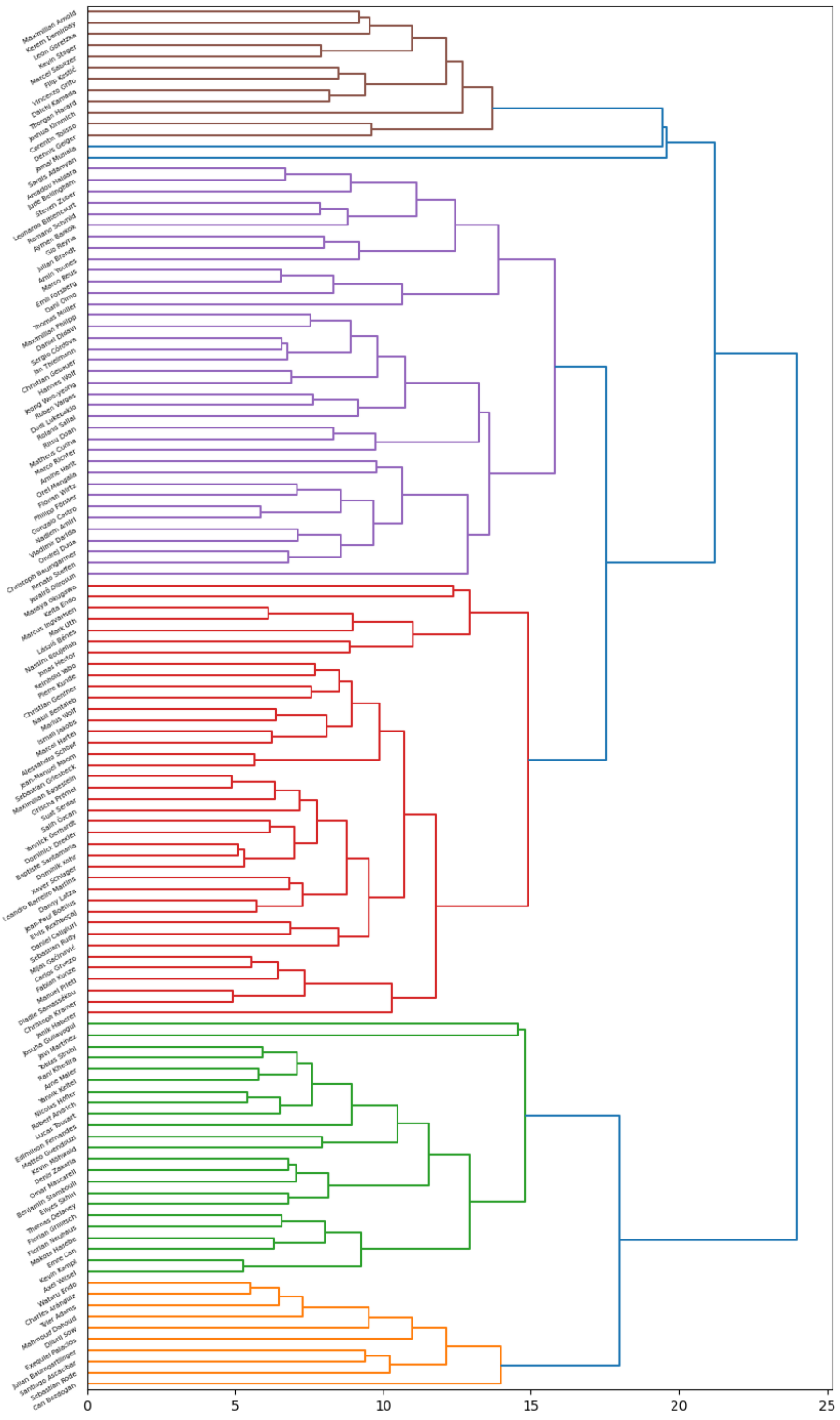


Figure 7: Dendrogram for midfielders, Complete-linkage

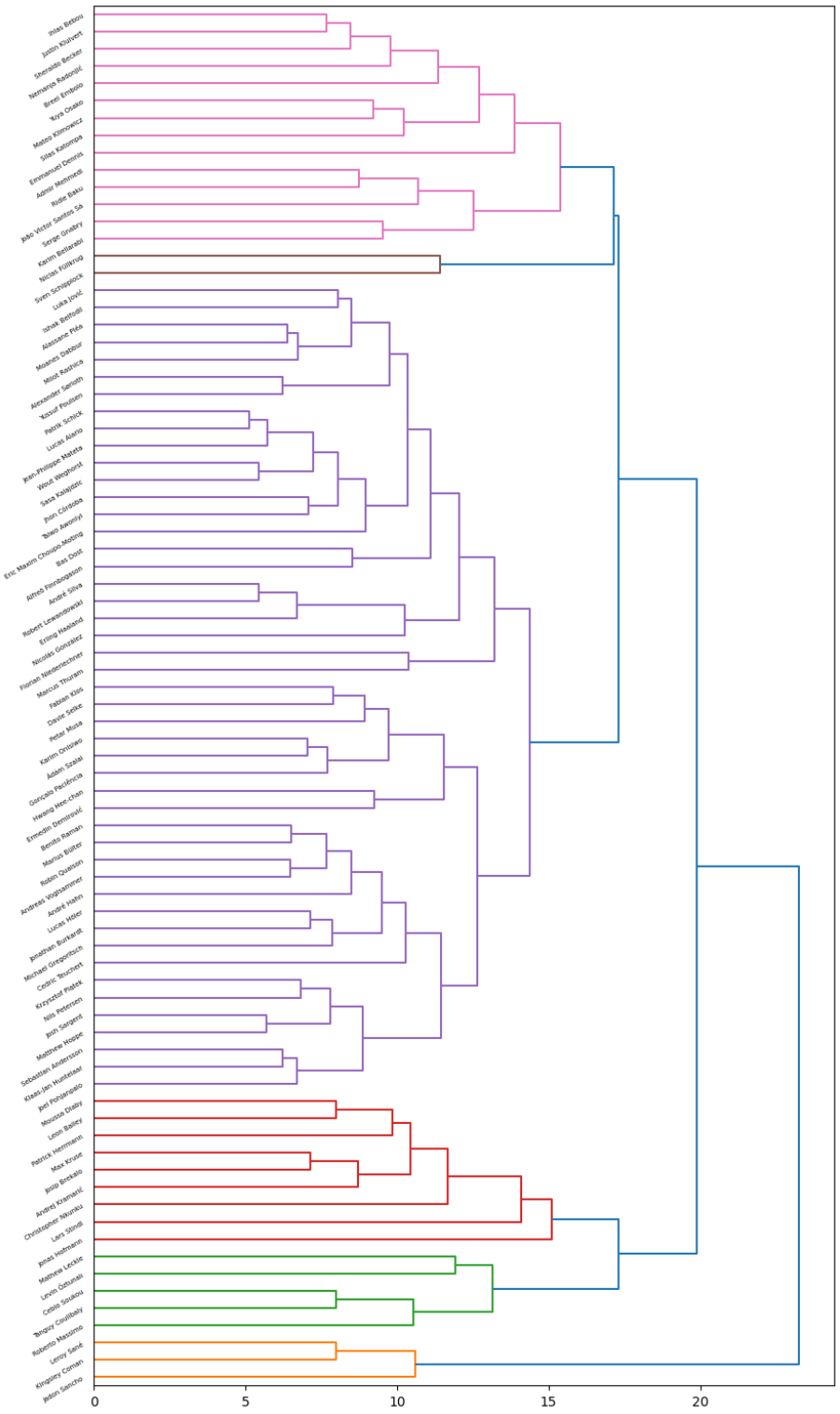


Figure 8: Dendrogram for forwards, Complete-linkage

Discussion

The aim of the cluster analysis was to be able to classify football players into different playing styles and put players who play football similarly in the same cluster.

One of the first steps made in (before) the analysis was to find the data to analyze. As mentioned in the section regarding data they were sourced from Fbref[1] where all data from outfield players was collected. This data included both very general statistics which have a large impact on a players worth such as goals, assists and also miscellaneous ones such as the amount of throw in's completed successfully. Since the values were normalized it also meant that these values were weighted equally even though they most likely should not as these two actions could affect the outcome of a game very differently. As these "irrelevant" values are still included in our data set it could be what leads to the homogeneous result as there could be a lot of these types of data points which do not bring much value to the actual result.

Weighting more important values could be a solution but since it's hard to say what value is important to the result without having much insight in the data makes the whole analysis kind of useless. This is due to cluster analysis most often being used as an initial way of understanding the data and if there is already a good understanding, is there any need for the analysis in the first place? As our aim in this study is to do an initial analysis of the data I believe that including all data points from the beginning and removing correlated ones is sufficient as an assumption is that we do not know which values are relevant in our use case or what weighting they should have.

Results from methods of choosing K

In figure 1, there is no clear elbow in our elbow plot which suggests that our points are at an (close to) equal distance to each other which gives this result as changes in the number of clusters K does not give a diminishing return of variance as the number K becomes larger.

Our Silhouette and Calinski-Harabasz plot give similar results where we should choose the lowest number possible as the plots are maximized at 2 clusters. This is

due to both methods being distance-based and as we can see in Figure 5, there is no obvious clustering which causes this result.

Looking at the GAP-statistic in figure 2 we can see that the plot is going to keep increasing as we choose a larger k which suggests that our clusters are not very well defined. In these cases where the result is not very useful, Tibshirani suggests in his study[7] that the 1-standard-error method which is finding the number K where the GAP-statistic has the most diminishing return. So according to the plot in figures 2, we see that the GAP-statistic stops growing at around 4 clusters for the full, defender and midfielder data sets and 3 for the forwards data set.

Clustering result

Although our result from the K-means algorithm is very homogeneous it is still possible to see that football players who play similarly/have the same position have been clustered together. For example, if we look at figure 5a and the yellow clustering, we see that most forwards have ended up in this cluster (Lewandowski, Haaland, Poulsen) and we can see that the algorithm have clustered players successfully if we reference the position the players are listed as. Although this looks like a successful clustering, it also does not give us very much information compared to what we already knew from before.

If we instead look at the result from our subsets in figure 5b, 5c and 5d we also see that the data is still very homogeneous and no clear cluster structure is visible. Even though the result is still mostly homogeneous, we can see that players with similar playing styles have ended up in the same cluster (same result as for the full data set but on a sublevel). For example, in figure 5d most wingers which are classified as forwards have ended up in the same cluster (Leroy Sané, Jadon Sancho, Kingsley Coman etc.). Other clusters have similar results but since the clustering is not very clear we can't be sure that the result is of much significance.

The result from the hierarchical clustering in figures 6, 7 and 8 is clearer then in our non-hierarchical cases. Looking at the dendrograms for each position we can

see that the groupings are clear and it is possible to create suitable groups based on the distance from the source of data points. We have set the distance to around 15 lengths for each sub set as this creates both clear and not too many groupings.

Another problem noticed is the lack of labelling of each cluster/group which could cause confusion around the result. This would be the next step of the analysis and could be done using for example a parallel coordinate plot and using the average of each attribute in each cluster. It would then be possible to compare the values between clusters to see differences/similarities between them.

Conclusion

In this study we have looked at the possibility of classifying football players using data in a cluster analysis. The analysis consisted of both hierarchical and non-hierarchical clustering methods and ways of choosing the number of cluster for the input in the k-means clustering. Our data set of 337 players with 51 different attributes was ran in our python-script which output the result that has been discussed.

The result could be valuable but as cluster analysis is meant to be used initially in data exploration there is no correct result and can then be interpreted differently based on who is looking at it.

References

[1] FBref, Football References. Bundesliga statistics season 20/21, Retrieved 2022. Data retrieved from FBref.com, `https://fbref.com/en/comps/20/Bundesliga-Stats`.

[2] Richard Arnold Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*. Upper Saddle River, NJ, 5. ed edition.

[3] Brian S. Everitt; Sabine Landau; Morven Leese. *Cluster analysis. 5th ed.* Hoboken, N.J. : Wiley, 2013.

[4] Peter Rousseeuw. Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987.

[5] P.H.A. Sneath and R.R. Sokal.

[6] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, Dec 1953.

[7] Robert Tibshirani, Walther Guenther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B*, 2001.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.model_selection import ParameterGrid
from sklearn.cluster import KMeans
from sklearn.compose import ColumnTransformer
from yellowbrick.cluster import KElbowVisualizer
import matplotlib.cm as cm
from pandas.plotting import parallel_coordinates
import seaborn as sns

#load values from file
def load_embeddings():
    df_raw = pd.read_csv('bundesliga_20_21.csv')
    df_raw = df_raw.fillna(0)
    df_raw_scaled = df_raw.copy()
    col_names = df_raw.columns.drop(['Player', 'Pos', 'Min'],
    ↪  errors='ignore')
    features = df_raw_scaled[col_names]
    scaler = StandardScaler().fit(features.values)
    features = scaler.transform(features.values)
    df_raw_scaled[col_names] = features
    return df_raw_scaled

#reducing using PCA
def pca_dim_reduction(df_scaled):
    df_scaled = pd.DataFrame(df_scaled.drop(['Player', 'Pos', 'Min'],
    ↪  axis=1, errors='ignore'))
    components = 2
    pca_2 = PCA(components)
```

```
32      pca_2_result = pca_2.fit_transform(df_scaled)
33      print('Explained variance per principal component:
        ↪  {}'.format(pca_2.explained_variance_ratio_))
34      print('Cumulative variance:', str(components), 'principal
        ↪  components: {:.2%}'.format(
35          np.sum(pca_2.explained_variance_ratio_)))
36      PC_values = np.arange(pca_2.n_components_) + 1
37      plt.plot(PC_values, pca_2.explained_variance_ratio_, 'o-',
        ↪  linewidth=2, color='blue')
38      plt.xlabel('Principal Components')
39      plt.ylabel('Explained variance')
40      plt.show()
41      return pca_2_result, pca_2
42
43  #methods of choosing K
44  def gapK(df, nrefs=3, maxClusters=30):
45      df = pd.DataFrame(df.drop(['Player', 'Pos', 'Min'], axis=1,
        ↪  errors='ignore'))
46      gaps = np.zeros((len(range(1, maxClusters)),))
47      resultsdf = pd.DataFrame({'ClusterCount':[], 'gap':[]})
48      for index, k in enumerate(range(1, maxClusters)):
49          refDisps = np.zeros(nrefs)
50          for i in range(nrefs):
51              randomReference = np.random.random_sample(size=df.shape)
52              km = KMeans(k)
53              km.fit(randomReference)
54              refDisp = km.inertia_
55              refDisps[i] = refDisp
56          km = KMeans(k)
57          km.fit(df)
58          origDisp = km.inertia_
59          gap = np.log(np.mean(refDisps)) - np.log(origDisp)
60          gaps[index] = gap
```

```
61          resultsdf = resultsdf.append({'ClusterCount':k, 'gap':gap},
            ↪  ignore_index=True)
62      plt.plot(resultsdf['ClusterCount'], resultsdf['gap'],
        ↪  linestyle='-', marker='o', color='b')
63      plt.xlabel('k')
64      plt.ylabel('Gap - Statistic')
65      plt.title('Gap - Statistic vs K')
66      plt.show()
67      return (gaps.argmax() + 1, resultsdf)
68
69  def elbowK(df):
70      df = pd.DataFrame(df.drop(['Player', 'Pos', 'Min'], axis=1,
        ↪  errors='ignore'))
71      model = KMeans()
72      visualizer = KElbowVisualizer(model, k=(2,30), timings=False)
73      visualizer.fit(df)
74      visualizer.show()
75      return getattr(visualizer, 'elbow_value_')
76
77  def silhouetteK(df):
78      df = pd.DataFrame(df.drop(['Player', 'Pos', 'Min'], axis=1,
        ↪  errors='ignore'))
79      model = KMeans()
80      visualizer = KElbowVisualizer(model, k=(2,30),
        ↪  metric='silhouette', timings=False)
81      visualizer.fit(df)
82      visualizer.show()
83      return getattr(visualizer, 'elbow_value_')
84
85  def calharK(df):
86      df = pd.DataFrame(df.drop(['Player', 'Pos', 'Min'], axis=1,
        ↪  errors='ignore'))
87      model = KMeans()
```

```
88      visualizer = KElbowVisualizer(model,
        ↪  k=(2,30),metric='calinski_harabasz', timings=False)
89      visualizer.fit(df)
90      visualizer.show()
91      return getattr(visualizer, 'elbow_value_')
92
93  #scatter plot of clusters with 2 principal components
94  def visualizing_results(pca_result, label, centroids_pca, data_scaled
    ↪  = None):
95      x = pca_result[:, 0]
96      y = pca_result[:, 1]
97      plt.scatter(x, y, c=label, alpha=0.5, s= 200, cmap='viridis')
98      plt.title('Clusters - Bundesliga 20/21')
99      plt.xlabel('PCA 1')
100     plt.ylabel('PCA 2')
101     plt.scatter(centroids_pca[:, 0], centroids_pca[:, 1], marker='X',
        ↪  s=200, linewidths=1.5, color='red', lw=1.5)
102     for i in range(len(x)):
103       plt.annotate(data_scaled.iloc[[i], 0].to_string(), (x[i], y[i]),
          ↪  fontsize=5)
104     plt.tight_layout()
105     plt.show()
106
107 #plot that displays attributes for each cluster
108 def display_parallel_coordinates_centroids(df, num_clusters):
109     fig = plt.figure(figsize=(12, 5))
110     title = fig.suptitle("Parallel coordinate plot for clusters",
        ↪  fontsize=10)
111     fig.subplots_adjust(top=0.9, wspace=0)
112     parallel_coordinates(df, 'cluster',
        ↪  color=sns.color_palette("bright", 10))
113     ax=plt.gca()
114     for tick in ax.xaxis.get_major_ticks()[1::2]:
115         tick.set_pad(20)
```

```
116     plt.show()
117
118 #creates corr matrix and removes columns which have a corr coeff
    ↪  larger then the threshold
119 def correlation(dataset, threshold):
120     col_corr = set() #
121     corr_matrix = dataset.corr(method='spearman').abs()
122     for i in range(len(corr_matrix.columns)):
123         for j in range(i):
124             if (corr_matrix.iloc[i, j] >= threshold) and
                ↪  (corr_matrix.columns[j] not in col_corr):
125                 colname = corr_matrix.columns[i]
126                 col_corr.add(colname)
127                 if colname in dataset.columns:
128                     del dataset[colname]
129
130     return dataset
131
132
133 def main():
134     print("Loading data set")
135     data_scaled = load_embeddings()
136     print(data_scaled.shape)
137     data_scaled = correlation(data_scaled, 0.80)
138     print(data_scaled.shape)
139
140     print("Reducing dimensions with PCA:")
141     pca_result, pca_2 = pca_dim_reduction(data_scaled)
142
143     print("Plot methods to choose number k:")
144     gap_score = gapK(data_scaled)
145     elbow_score = elbowK(data_scaled)
146     silhouette_score = silhouetteK(data_scaled)
147     calhar_score = calharK(data_scaled)
```

```
148     print("optimum number k=", elbow_score)
149
150     print("Clustering data using K-means:")
151     kmeans = KMeans(n_clusters=4) #look at gap-statistic plot
152     kmeans.fit(pd.DataFrame(data_scaled.drop(['Player', 'Pos', 'Min'],
        ↪  axis=1, errors='ignore')))
153     centroids = kmeans.cluster_centers_
154     centroids_pca = pca_2.transform(centroids)
155
156     print("Plot result:")
157     visualizing_results(pca_result, kmeans.labels_, centroids_pca,
        ↪  data_scaled)
158     centroids = pd.DataFrame(kmeans.cluster_centers_,
        ↪  columns=data_scaled.drop(['Player', 'Pos', 'Min'], axis=1,
        ↪  errors='ignore').columns)
159     centroids['cluster'] = centroids.index
160     display_parallel_coordinates_centroids(centroids, elbow_score)
161
162
163 if __name__ == "__main__":
164     main()
```