

Unsupervised Learning Practice Project Solution: Fantasy Sports Clustering Analysis

Context

Fantasy sports are online gaming platforms where participants draft and manage virtual teams of real professional sports players. Based on the performance of the players in the real world, players are allotted points in the fantasy sports platform every match. The objective is to create the best possible team with a fixed budget to score maximum fantasy points, and users compete against each other over an entire sports league or season. Some of these fantasy sports require actual financial investments for participation, with chances of winning monetary rewards as well as free matchday tickets on a periodic basis.

The fantasy sports market has seen tremendous growth over the past few years, with a valuation of \ \$18.6 billion in 2019. The soccer (globally - football) segment led in terms of market share in 2019, with over 8 million participants worldwide, and is expected to retain its dominance over the next couple of years. Digitalization is one of the primary factors driving the growth of the fantasy sports market as it allows participants the opportunity to compete on a global level and test their skills. With an increase in smartphone usage and availability of fantasy sports apps, this market is expected to witness a global surge and reach a \ \$48.6 billion valuation by 2027.

Objective

OnSports is a fantasy sports platform that has fantasy leagues for many different sports and has witnessed an increasing number of participants globally over the past 5 years. For each player, a price is set at the start, and the price keeps changing over time based on the performance of the players in the real world. With the new English Premier League season about to start, they have collected data from the past season and want to analyze it to determine the price of each player for the start of the new season. OnSports have hired you as a data scientist and asked you to conduct a cluster analysis to identify players of different potentials based on previous season performances. This will help them understand the patterns in player performances and fantasy returns and decide the exact price to be set for each player for the upcoming football season.


Data Description

- **Player_Name:** Name of the player.
- **Club:** Club in which the player plays.
- **Position:** Position in which the player plays.
- **Goals_Scored:** Number of goals scored by the player in the previous season.
- **Assists:** Number of passes made by the player leading to goals in the previous season.
- **Total_Points:** Total number of fantasy points scored by the player in the previous season.
- **Minutes:** Number of minutes played by the player in the previous season.
- **Goals_Conceded:** Number of goals conceded by the player in the previous season.


- **Creativity:** A score, computed using a range of stats, that assesses player performance in terms of producing goalscoring opportunities for other players.
- **Influence:** A score, computed using a range of stats, that evaluates a player's impact on a match, taking into account actions that could directly or indirectly affect the match outcome.
- **Threat:** A score, computed using a range of stats, that gauges players who are most likely to score goals.
- **Bonus:** Total bonus points received. The three best performing players in each match receive additional bonus points based on a score computed using a range of stats. 3 points are awarded to the highest scoring player, 2 to the second best, and 1 to the third.
- **Clean_Sheets:** Number of matches without conceding a goal in the previous season.

✓ Importing the necessary libraries and overview of the dataset

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
1 !pip install scikit-learn-extra
```

 Requirement already satisfied: scikit-learn-extra in /usr/local/lib/python3.11/dist-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.11/dist-packages (from scikit-learn-extra) (1.26.4)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.11/dist-packages (from scikit-learn-extra) (1.13.1)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn-extra) (1.6.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.5.0)

```
1 # Libraries to help with reading and manipulating data
2 import numpy as np
3 import pandas as pd
4
5 # Libraries to help with data visualization
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8
9 sns.set_theme(style='darkgrid')
10
11 # Removes the limit for the number of displayed columns
12 pd.set_option("display.max_columns", None)
13 # Sets the limit for the number of displayed rows
14 pd.set_option("display.max_rows", 200)
15
16 # To scale the data using z-score
17 from sklearn.preprocessing import StandardScaler
18
19 # To compute distances
20 from scipy.spatial.distance import cdist, pdist
21
22 # To perform K-Means clustering and compute silhouette scores
23 from sklearn.cluster import KMeans
24 from sklearn.metrics import silhouette_score
25
26 # To import K-Medoids
```

```
27 from sklearn_extra.cluster import KMedoids
28
29 # To import DBSCAN and Gaussian Mixture
30 from sklearn.cluster import DBSCAN
31 from sklearn.mixture import GaussianMixture
32
33 # To perform hierarchical clustering, compute cophenetic correlation, and create dendrograms
34 from sklearn.cluster import AgglomerativeClustering
35 from scipy.cluster.hierarchy import dendrogram, linkage, cophenet
```


```
1 data = pd.read_csv("/content/drive/MyDrive/1.PROGETTI/2024.09.Corso Mit /03.Week – Data Analysis & Visualization/Practice Project Analysis & Visualization/fpl_data.csv")
```



```
1 data.shape
```

 (476, 13)

- The dataset has 476 rows and 13 columns.

```
1 # Viewing 10 random rows of the data
2 data.sample(10, random_state = 1)
```



	Player_Name	Club	Position	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets	
441	Mark Noble	West Ham United	Midfielder	0	0	27	701	15	88.6	80.4	7	0	0	
363	Sean Longstaff	Newcastle United	Midfielder	0	1	41	1405	26	182.8	179.2	148	1	2	
31	Anwar El Ghazi	Aston Villa	Midfielder	10	0	111	1604	22	426.1	500.4	726	13	5	
132	Olivier Giroud	Chelsea	Forward	4	0	47	740	5	112.0	161.4	403	6	4	
90	Chris Wood	Burnley	Forward	12	3	138	2741	43	323.2	595.8	1129	16	9	
249	Vontae Daley-Campbell	Leicester City	Defender	0	0	0	0	0	0.0	0.0	0	0	0	
65	Danny Welbeck	Brighton and Hove Albion	Forward	6	4	89	1541	18	269.7	319.8	595	15	6	
445	Ryan Fredericks	West Ham United	Defender	1	1	28	564	9	166.8	155.2	96	0	1	
117	Christian Pulisic	Chelsea	Midfielder	4	3	82	1731	21	378.8	361.4	724	3	7	
415	Ryan Sessegnon	Tottenham Hotspurs	Defender	0	0	0	0	0	0.0	0.0	0	0	0	

Observations:

- The data has players from various clubs.
- There seem to be a lot of players playing in the midfield.
- Some players have played less or zero minutes in the previous season and so they have scored few or zero fantasy points, respectively.

```
1 # Copying the data to another variable to avoid any changes to original data
2 df = data.copy()
```

```
1 # Checking datatypes and number of non-null values for each column
2 df.info()
```

```
>>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 476 entries, 0 to 475
Data columns (total 13 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Player_Name         476 non-null   object
1   Club                476 non-null   object
2   Position            476 non-null   object
3   Goals_Scored        476 non-null   int64
4   Assists             476 non-null   int64
5   Total_Points        476 non-null   int64
6   Minutes             476 non-null   int64
7   Goals_Conceded      476 non-null   int64
8   Creativity          476 non-null   float64
9   Influence            476 non-null   float64
10  Threat              476 non-null   int64
11  Bonus               476 non-null   int64
12  Clean_Sheets        476 non-null   int64
dtypes: float64(2), int64(8), object(3)
memory usage: 48.5+ KB
```

- Player_Name, Club, and Position are categorical variables.
- All the other columns in the data are numeric in nature.

```
1 # Checking for duplicate values
2 df.duplicated().sum()
```

```
>>> 0
```

- There are no duplicate values in the data.

```
1 # Checking for missing values
2 df.isnull().sum()
```



	0
Player_Name	0
Club	0
Position	0
Goals_Scored	0
Assists	0
Total_Points	0
Minutes	0
Goals_Conceded	0
Creativity	0
Influence	0
Threat	0
Bonus	0
Clean_Sheets	0

dtype: int64

- There are no missing values in the data.

✕ Exploratory Data Analysis

Let's check the statistical summary of the data.

```
1 df.describe(include = 'all').T
```



	count	unique	top	freq	mean	std	min	25%	50%	75%	max	
Player_Name	476	476	Alex Runnarsson	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Club	476	17	Arsenal	30	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Position	476	4	Midfielder	195	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Goals_Scored	476.0	NaN	NaN	NaN	1.907563	3.455562	0.0	0.0	0.5	2.0	23.0	
Assists	476.0	NaN	NaN	NaN	1.752101	2.708563	0.0	0.0	0.0	2.0	14.0	
Total_Points	476.0	NaN	NaN	NaN	58.516807	51.293559	0.0	10.0	48.0	94.25	244.0	
Minutes	476.0	NaN	NaN	NaN	1336.909664	1073.773995	0.0	268.75	1269.5	2256.25	3420.0	
Goals_Conceded	476.0	NaN	NaN	NaN	19.157563	15.946171	0.0	4.0	18.0	31.0	68.0	
Creativity	476.0	NaN	NaN	NaN	195.97605	251.478541	0.0	8.3	96.95	296.95	1414.9	
Influence	476.0	NaN	NaN	NaN	294.617647	267.779681	0.0	46.5	233.1	499.5	1318.2	
Threat	476.0	NaN	NaN	NaN	224.962185	318.240377	0.0	5.75	104.5	298.25	1980.0	
Bonus	476.0	NaN	NaN	NaN	4.718487	6.252625	0.0	0.0	2.0	7.0	40.0	
Clean_Sheets	476.0	NaN	NaN	NaN	4.745798	4.394312	0.0	0.0	4.0	8.0	19.0	

Observations:

- There are players from 17 clubs in the data.
- Most of the players in the data play in midfield.
- The average number of minutes played is ~1350.
- Players scored an average of ~60 fantasy points in the previous season.
- More than 50% of the players in the data have scored one or no goals.
- 50% of the players in the data have assisted no goals.

Univariate Analysis

```
1 # Function to plot a boxplot and a histogram along the same scale
2
3
4 def histogram_boxplot(data, feature, figsize = (12, 7), kde = False, bins = None):
5
6     """
7     Boxplot and histogram combined
8
9     data: dataframe
10    feature: dataframe column
11    figsize: size of figure (default (12, 7))
12    kde: whether to the show density curve (default False)
13    bins: number of bins for histogram (default None)
14    """
15
16    f2, (ax_box2, ax_hist2) = plt.subplots(
```

```

17     nrows = 2,      # Number of rows of the subplot grid = 2
18     sharex = True,  # X-axis will be shared among all subplots
19     gridspec_kw = {"height_ratios": (0.25, 0.75)},
20     figsize = figsize,
21 ) # Creating the 2 subplots
22 sns.boxplot(
23     data = data, x = feature, ax = ax_box2, showmeans = True, color = "violet"
24 ) # Boxplot will be created and a star will indicate the mean value of the column
25 sns.histplot(
26     data = data, x = feature, kde = kde, ax = ax_hist2, bins = bins, palette = "winter"
27 ) if bins else sns.histplot(
28     data = data, x = feature, kde = kde, ax = ax_hist2
29 ) # For histogram
30 ax_hist2.axvline(
31     data[feature].mean(), color = "green", linestyle = "--"
32 ) # Add mean to the histogram
33 ax_hist2.axvline(
34     data[feature].median(), color = "black", linestyle = "-"
35 ) # Add median to the histogram

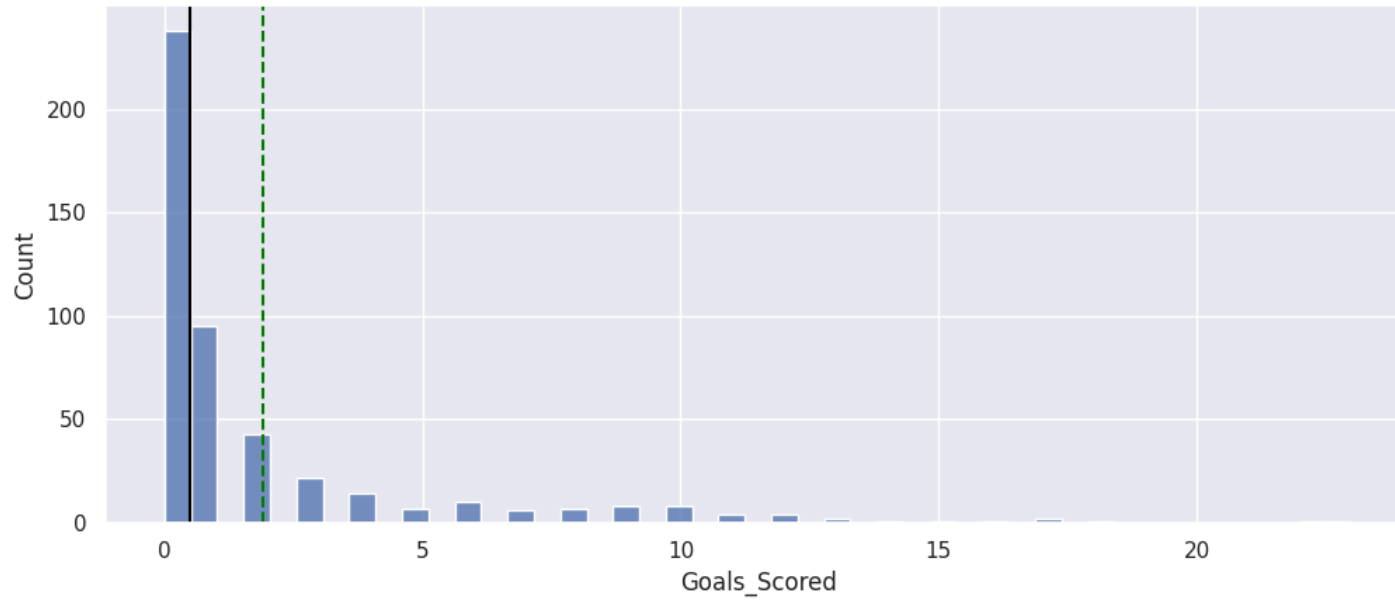
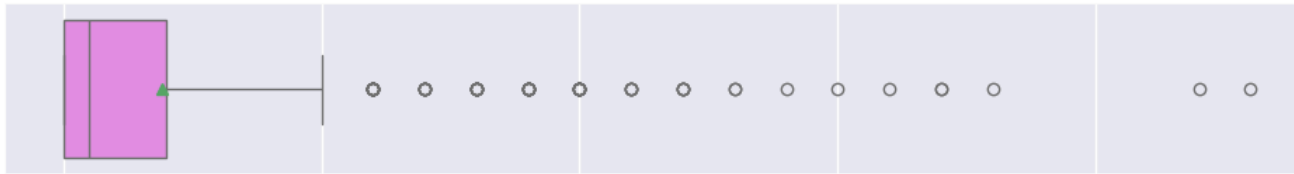
```

Goals_Scored

```

1 histogram_boxplot(df, 'Goals_Scored')

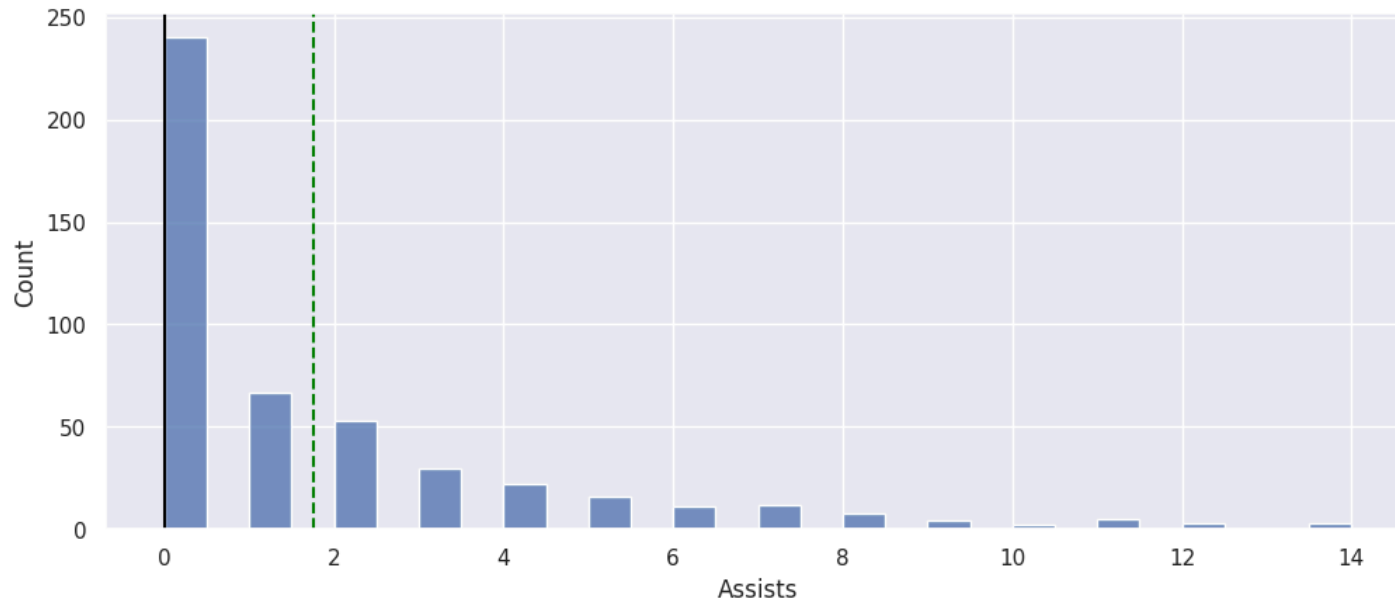
```



- The distribution is right-skewed and very few players have scored more than 15 goals.

Assists

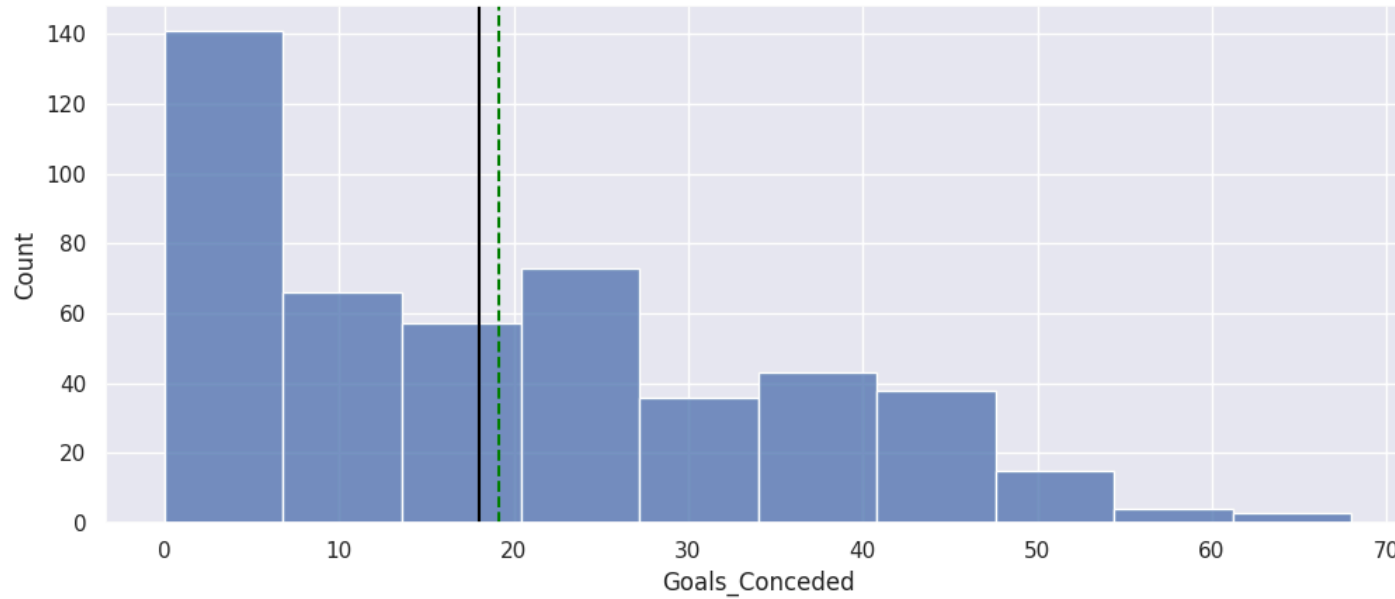
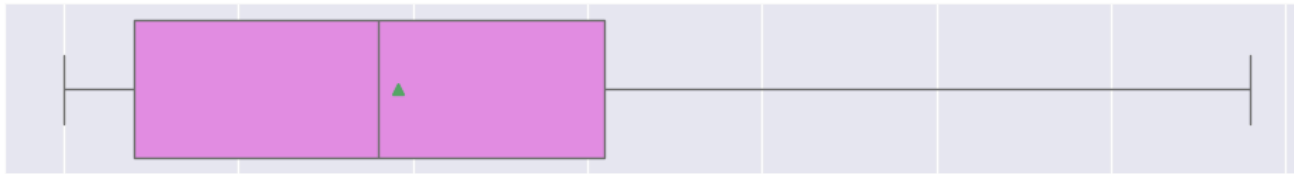
```
1 histogram_boxplot(df, 'Assists')
```

- The distribution is right-skewed and very few players have assisted more than 8 goals.

Goals_Conceded

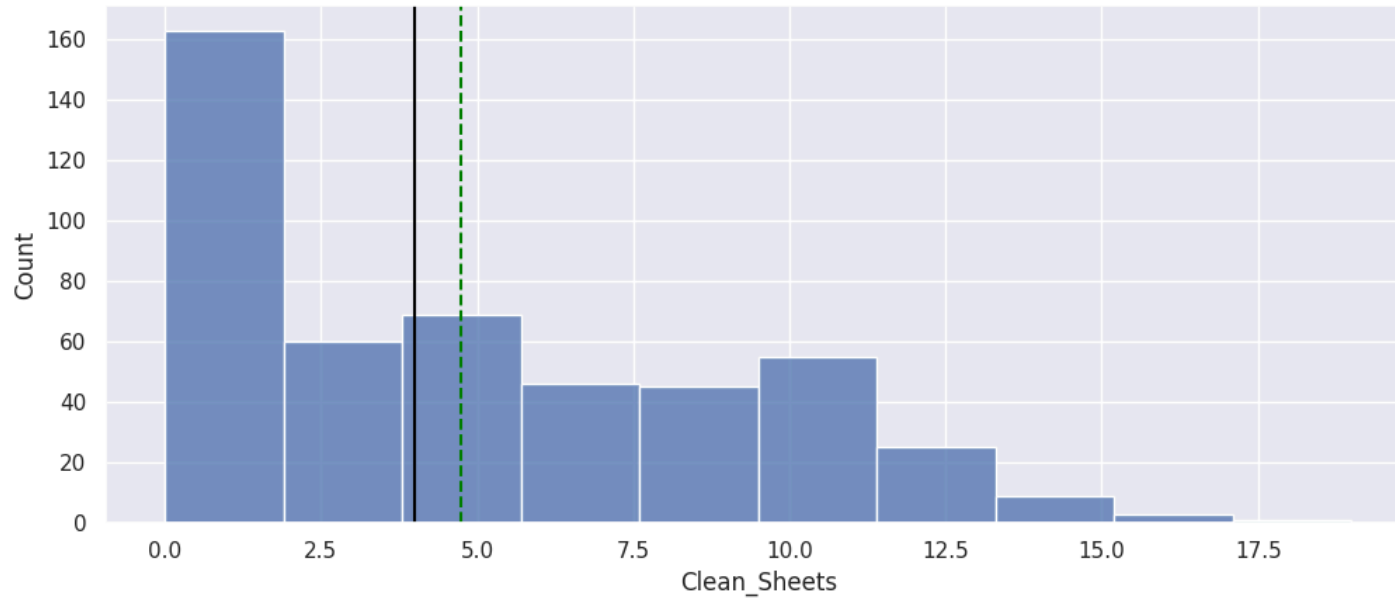
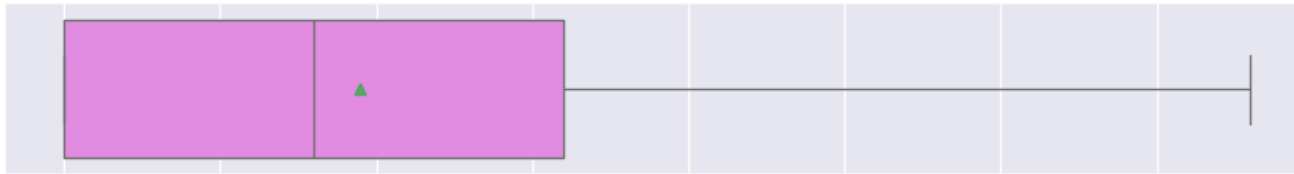
```
1 histogram_boxplot(df, 'Goals_Conceded')
```



- The distribution is slightly right-skewed and ~50% of the players have conceded 20 or less goals.

Clean_Sheets

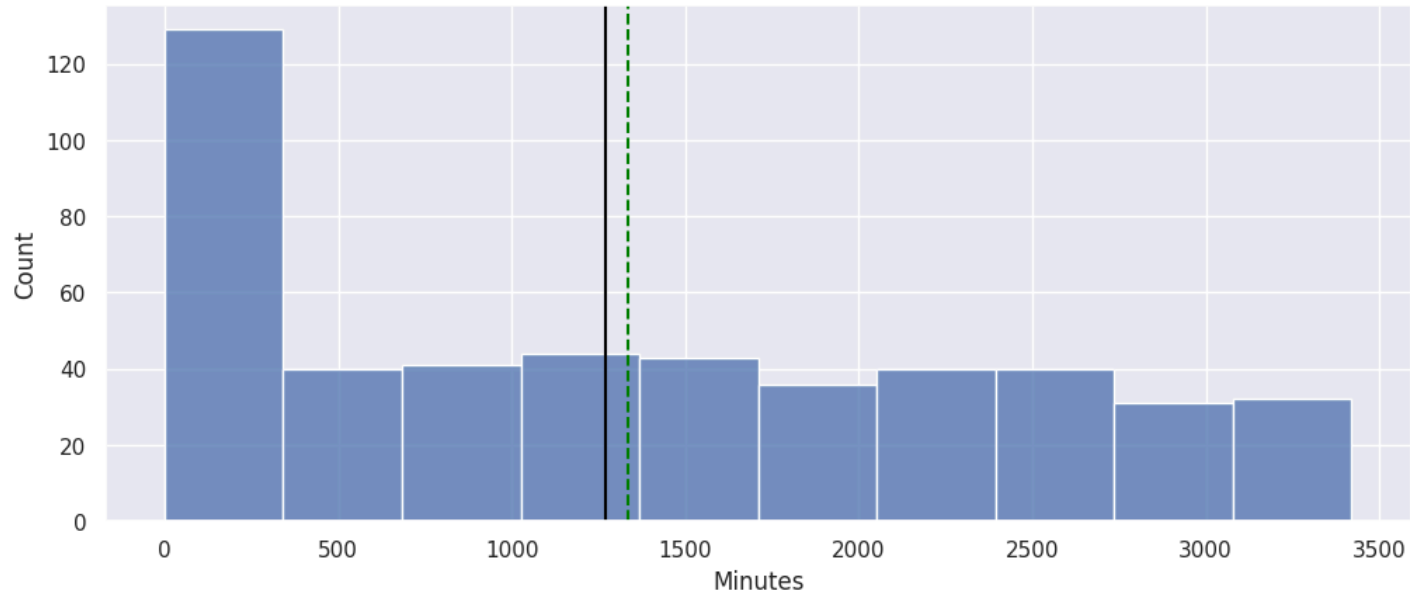
```
1 histogram_boxplot(df, 'Clean_Sheets')
```



- The distribution is slightly right-skewed and 50% of the players have kept 4 or less clean sheets.

Minutes

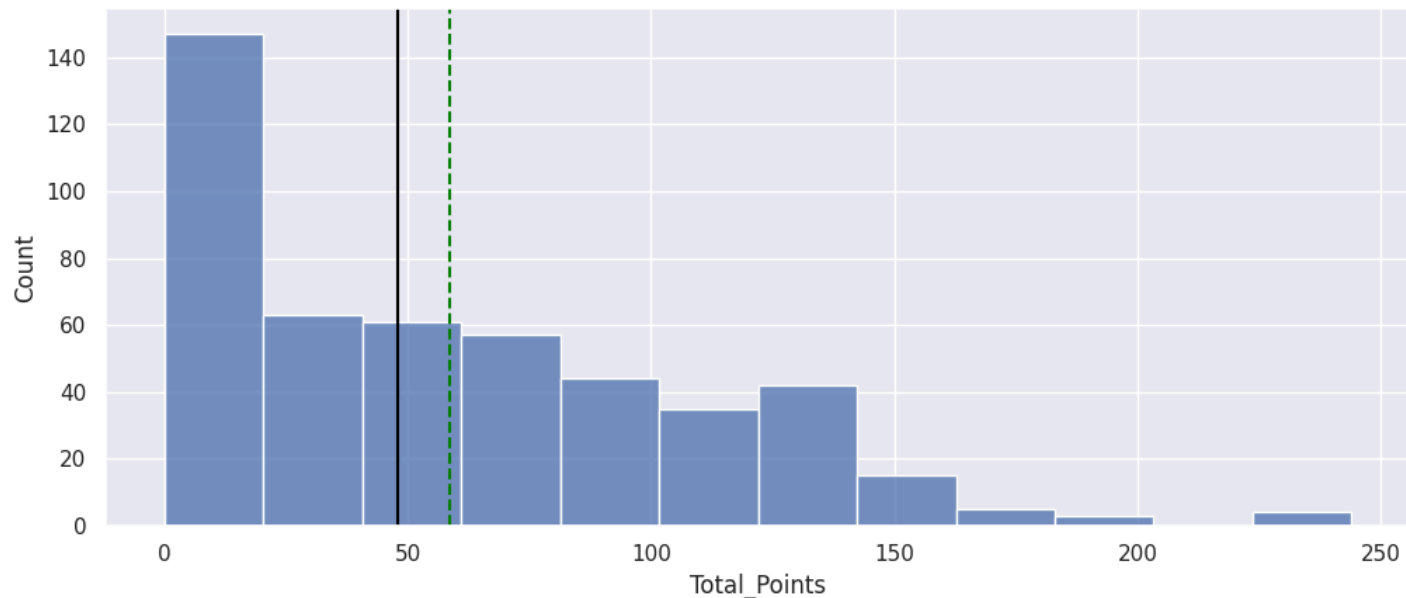
```
1 histogram_boxplot(df, 'Minutes')
```



- The distribution looks close to uniform, and 50% of the players have played ~1250 or more minutes.
- Many players did not play even a single minute of football last season.

Total_Points

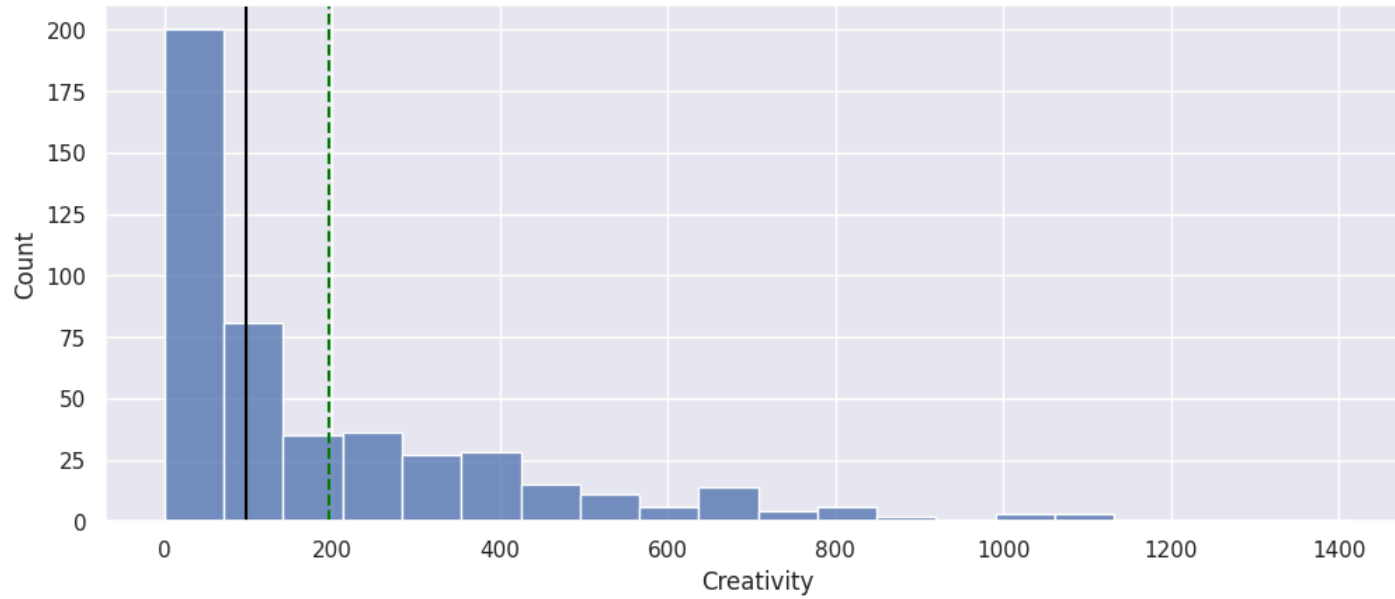
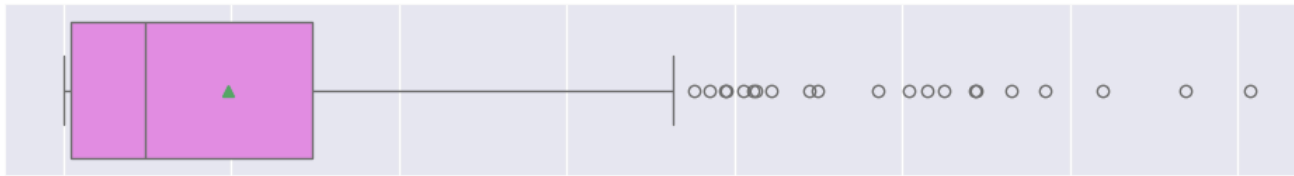
```
1 histogram_boxplot(df, 'Total_Points')
```



- The distribution is right-skewed, and more than 50% of the players have scored more than 50 fantasy points.
- Many players scored no fantasy points last season.
- There are a few outliers, suggesting that these players scored a lot more fantasy points than the others.

Creativity

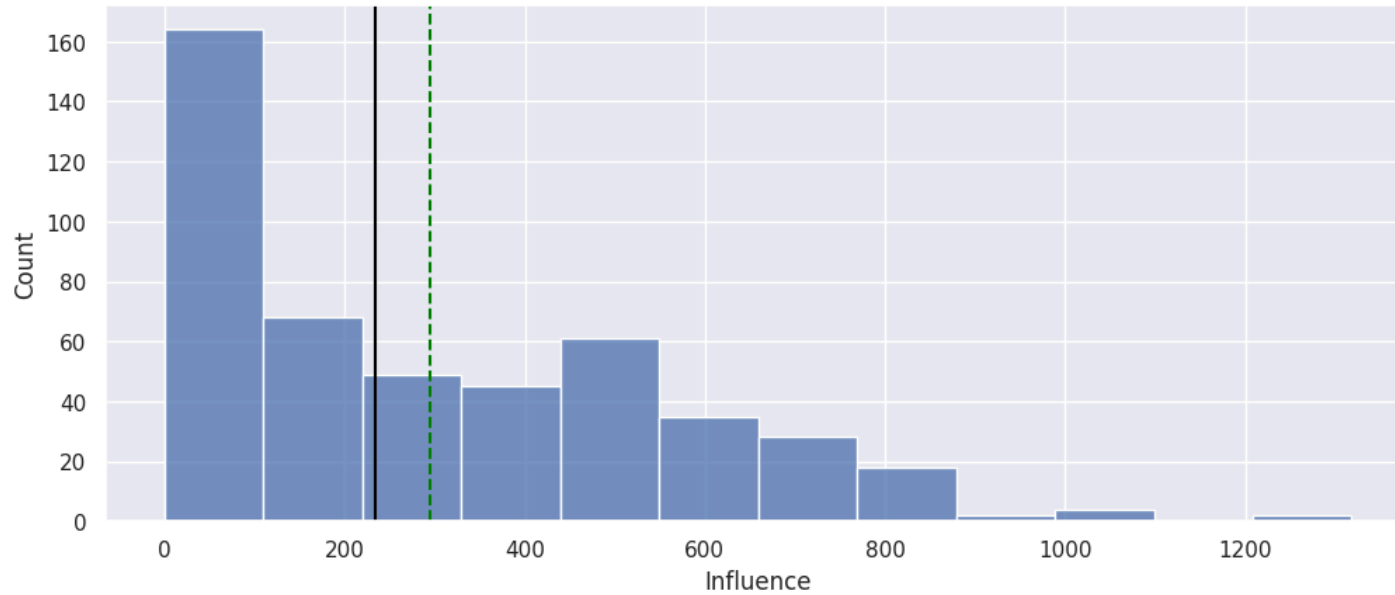
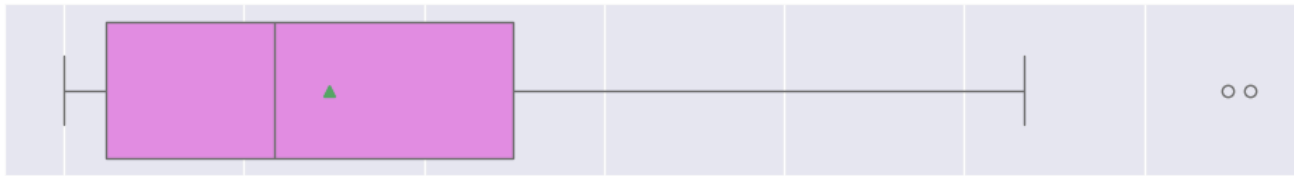
```
1 histogram_boxplot(df, 'Creativity')
```



- The distribution is right-skewed and few players have a creativity score of more than 500.

Influence

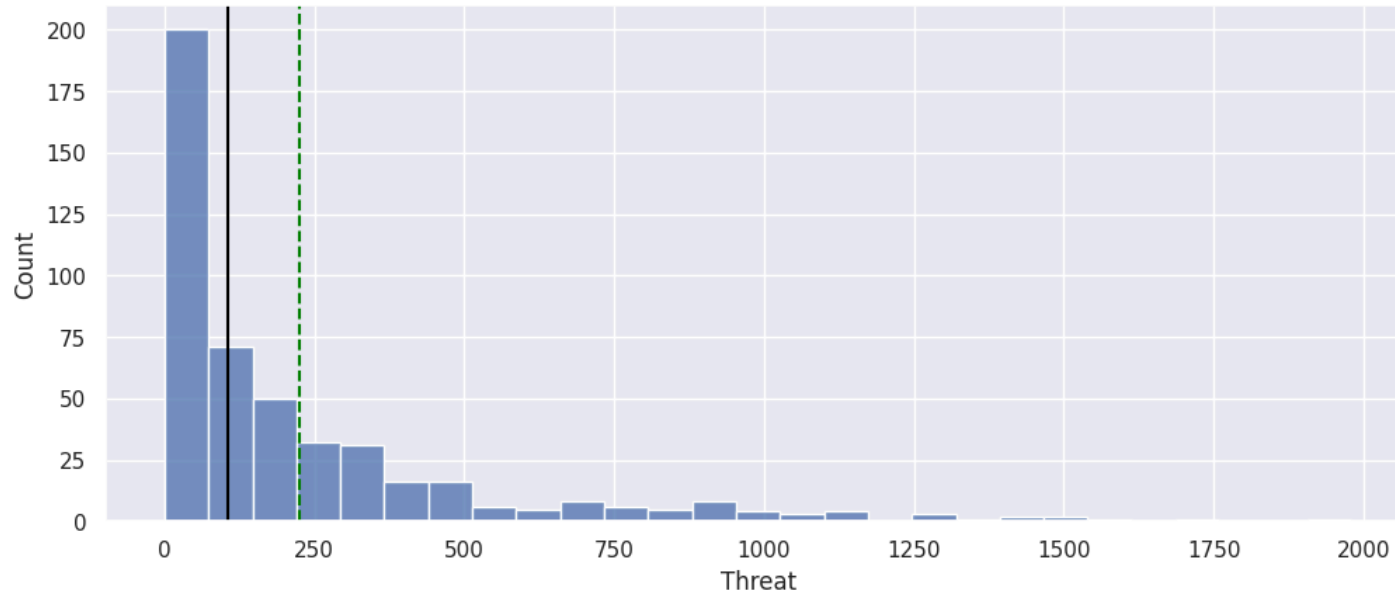
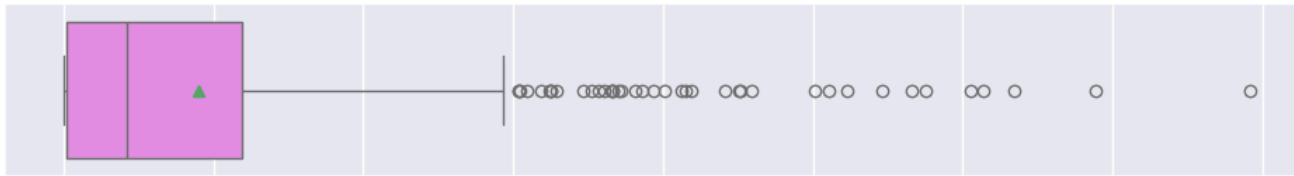
```
1 histogram_boxplot(df, 'Influence')
```



- The distribution is right-skewed and few players have a influence score of more than 800.

Threat

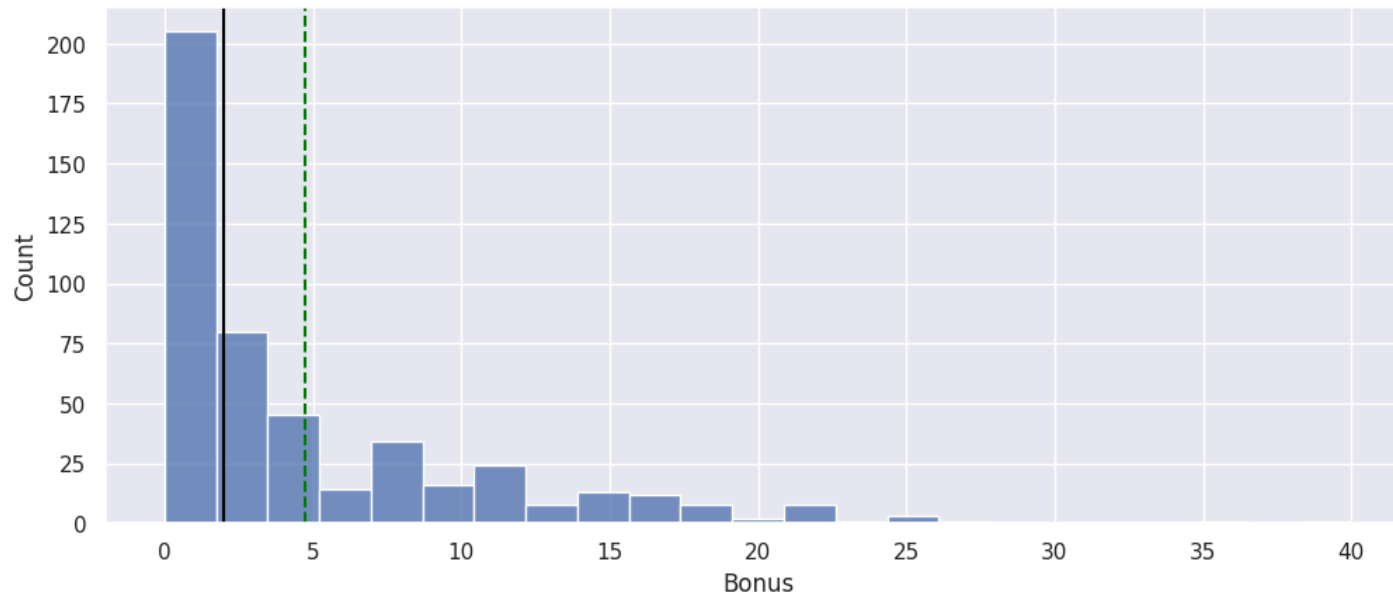
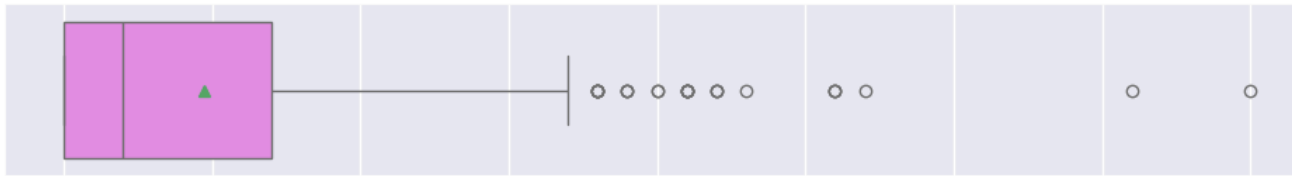
```
1 histogram_boxplot(df, 'Threat')
```



- The distribution is right-skewed and few players have a threat score of more than 500.

Bonus

```
1 histogram_boxplot(df, 'Bonus')
```

- The distribution is right-skewed and very few players received more than 20 bonus fantasy points last season.

```
1 # Function to create labeled barplots
2
3
4 def labeled_barplot(data, feature, perc = False, n = None):
5     """
6     Barplot with percentage at the top
7
8     data: dataframe
9     feature: dataframe column
10    perc: whether to display percentages instead of count (default is False)
11    n: displays the top n category levels (default is None, i.e., display all levels)
12    """
13
14    total = len(data[feature]) # Length of the column
15    count = data[feature].nunique()
16    if n is None:
17        plt.figure(figsize = (count + 1, 5))
18    else:
19        plt.figure(figsize = (n + 1, 5))
```

```

20
21 plt.xticks(rotation = 90, fontsize = 15)
22 ax = sns.countplot(
23     data = data,
24     x = feature,
25     palette = "Paired",
26     order = data[feature].value_counts().index[:n].sort_values(),
27 )
28
29 for p in ax.patches:
30     if perc == True:
31         label = "{:.1f}%".format(
32             100 * p.get_height() / total
33         ) # Percentage of each class of the category
34     else:
35         label = p.get_height() # Count of each level of the category
36
37 x = p.get_x() + p.get_width() / 2 # Width of the plot
38 y = p.get_height() # Height of the plot
39
40 ax.annotate(
41     label,
42     (x, y),
43     ha = "center",
44     va = "center",
45     size = 12,
46     xytext = (0, 5),
47     textcoords = "offset points",
48 ) # Annotate the percentage
49
50 plt.show() # Show the plot

```

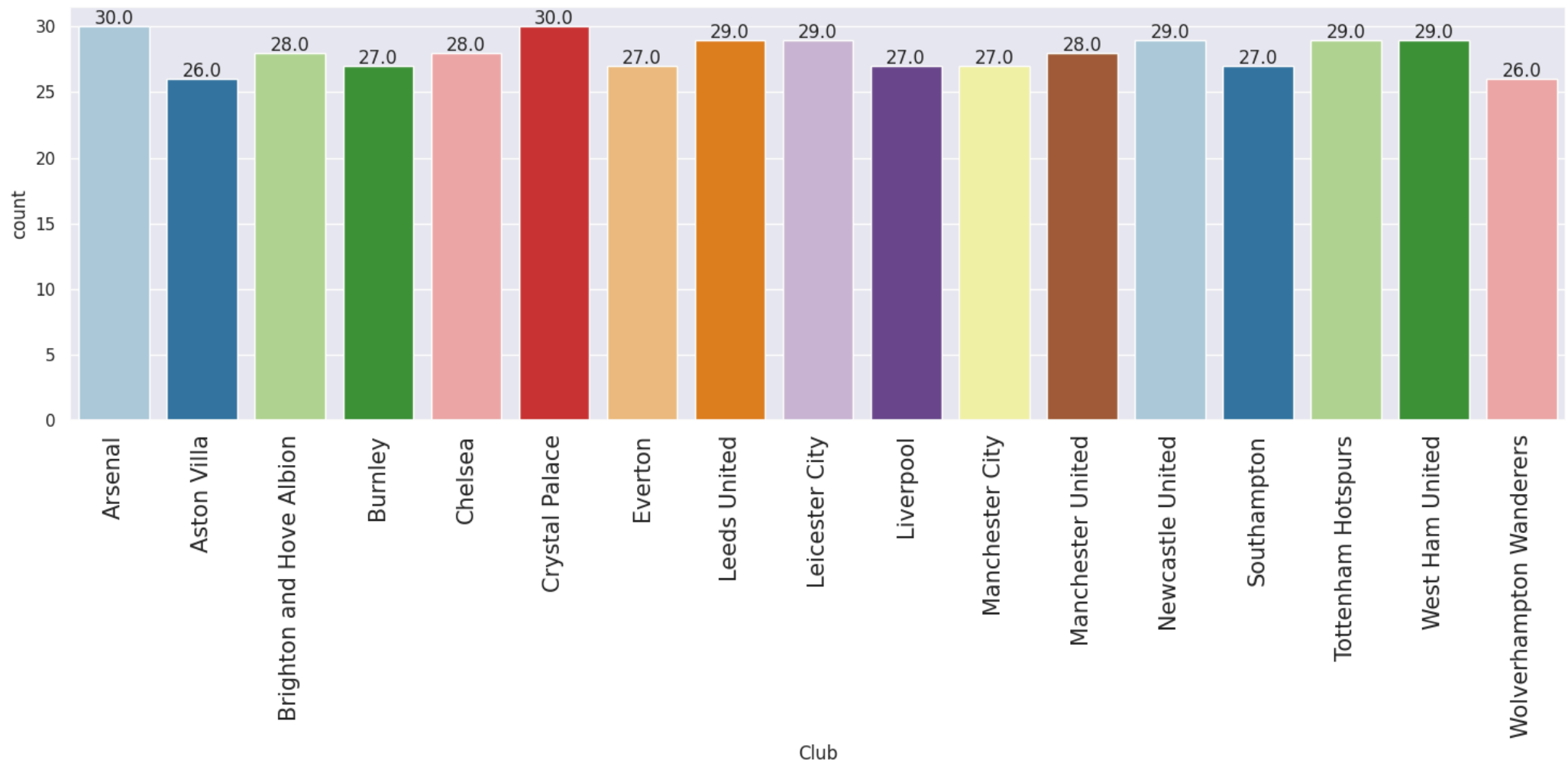
Club

```
1 labeled_barplot(df, 'Club')
```

<ipython-input-29-d0a88f9148b7>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
ax = sns.countplot(
```



- The number of players in each club is almost uniformly distributed.
- All the clubs have at least 26 players.

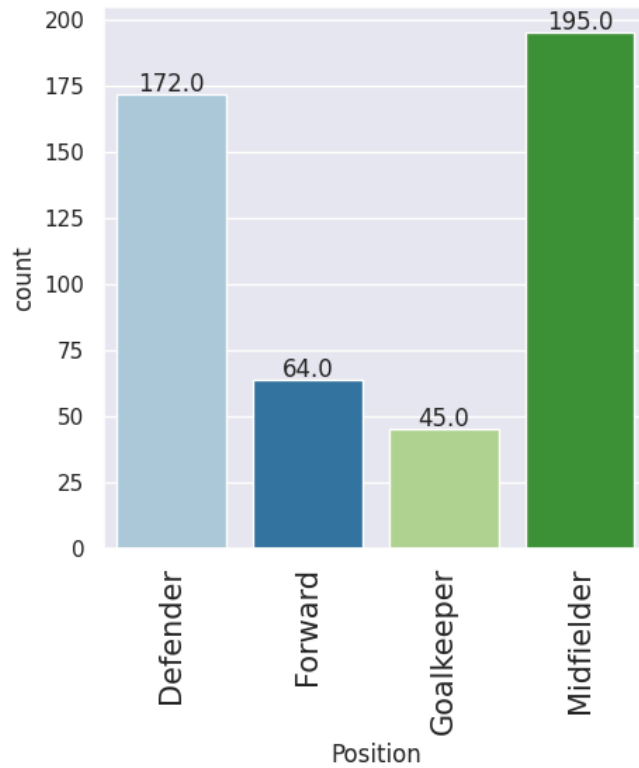
Position

```
1 labeled_barplot(df, 'Position')
```

<ipython-input-29-d0a88f9148b7>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
ax = sns.countplot(
```

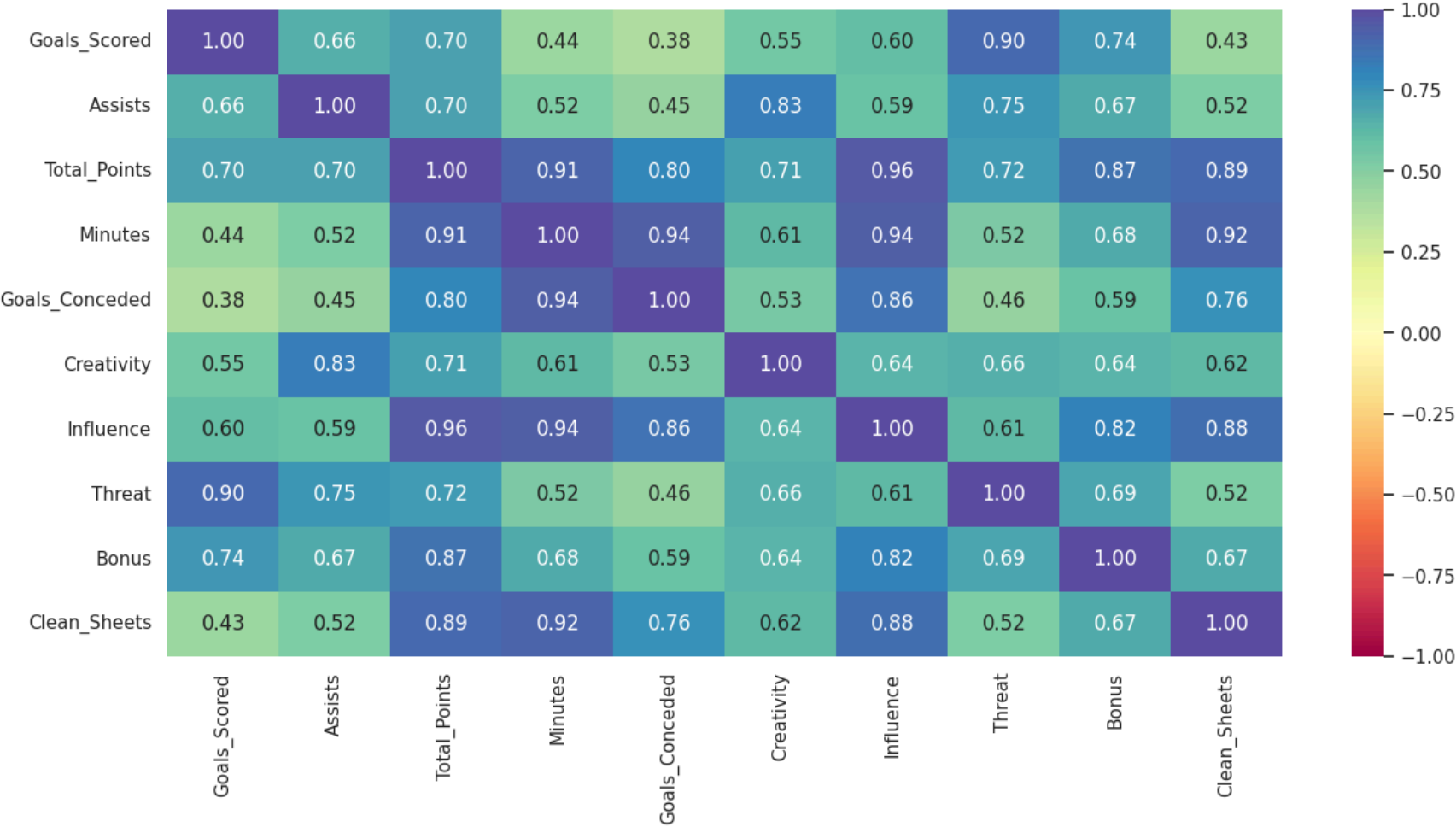


- The number of midfielders in the data is more than four times the number of goalkeepers.
 - This makes sense as a team can only play one goalkeeper in a match, so it doesn't make sense to have too many goalkeepers in the squad.
- The number of defenders in the data is nearly 3 times the number of forwards.
 - This has more to do with the formation in which the teams prefer to play nowadays.
 - Most teams tend to have 1 or 2 forwards only.

▼ Bivariate Analysis

```
1 # Correlation check
2 cols_list = df.select_dtypes(include = np.number).columns.tolist()
3
4 plt.figure(figsize = (15, 7))
5
6 sns.heatmap(
```

```
7 df[cols_list].corr(numeric_only = True), annot = True, vmin = -1, vmax = 1, fmt = ".2f", cmap = "Spectral"
8 )
9
10 plt.show()
```

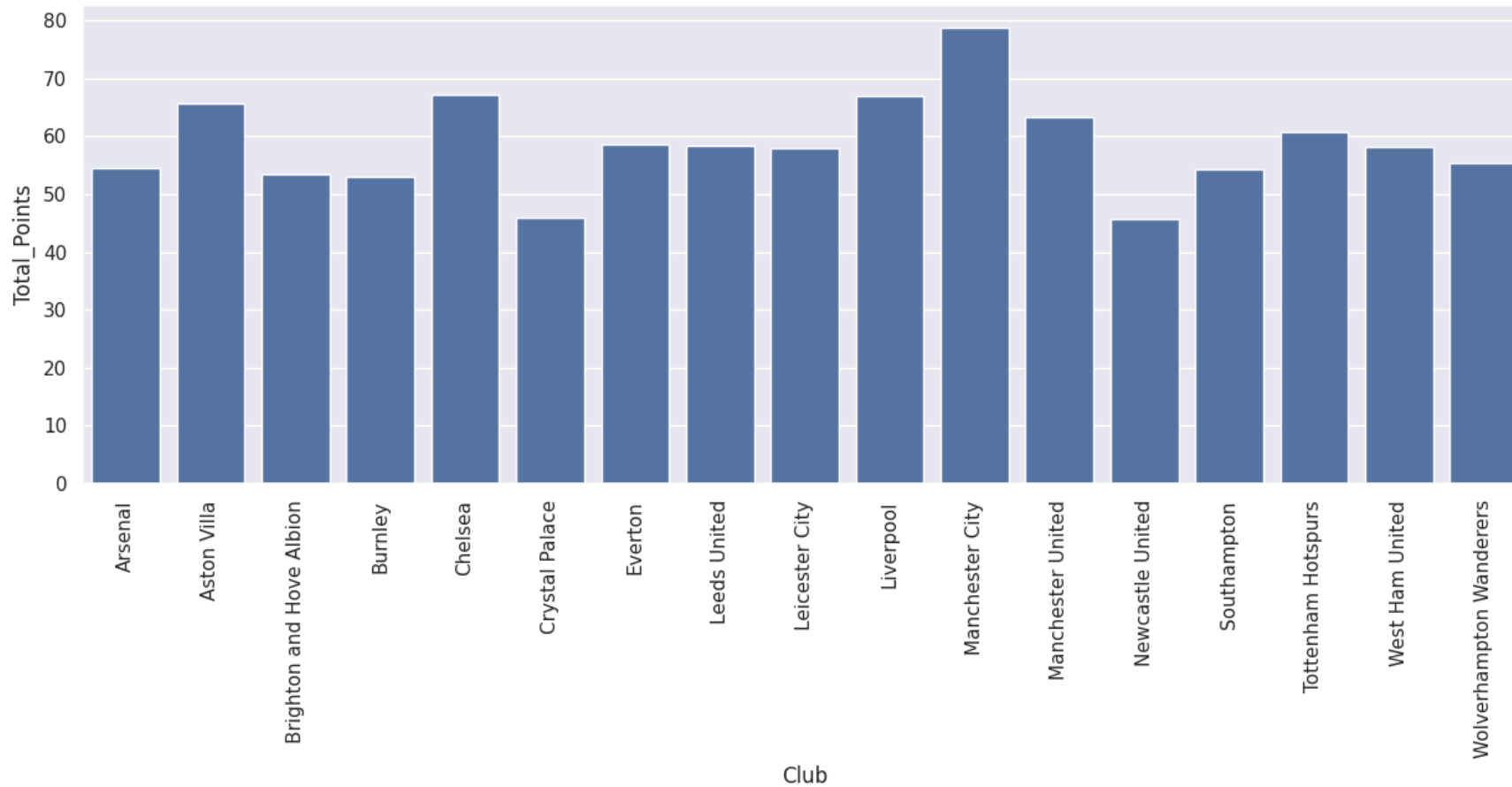


Observations:

- Many variables show a high correlation with each other.
- The number of goals scored by a player and the threat score of the player is highly correlated.
 - This makes sense as the threat score gauges a player's goalscoring potential.
- Influence score is highly correlated with the total fantasy points scored and the number of minutes played by a player.
 - This makes sense as these players have a higher impact on the game's outcome, so they tend to play for long each game and score more fantasy points.

Let's check players from which team have scored the most fantasy points on average.

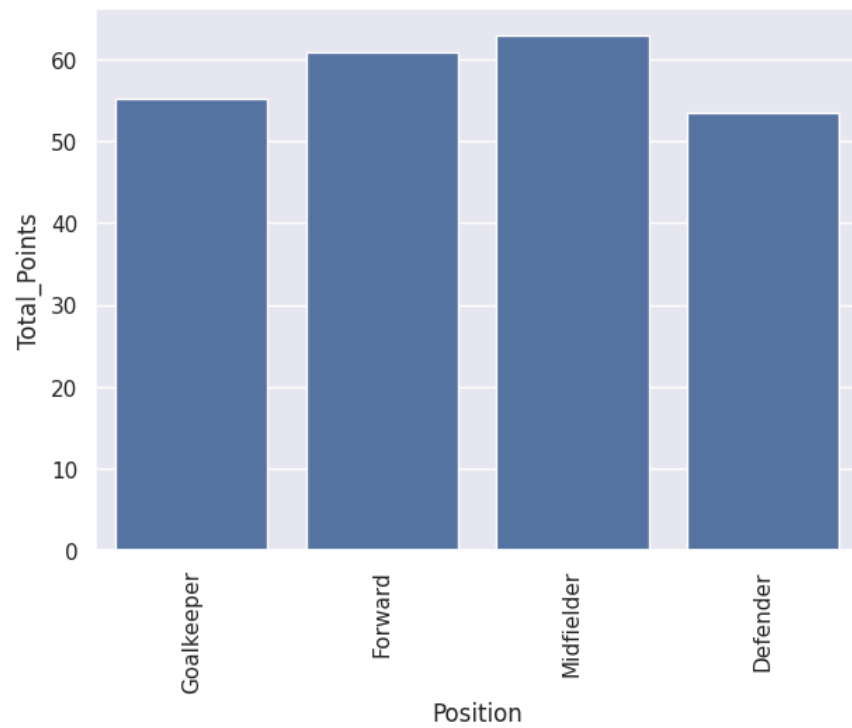
```
1 plt.figure(figsize = (15,5))
2 sns.barplot(data = df, x = 'Club', y = 'Total_Points', errorbar=('ci', False))
3 plt.xticks(rotation = 90)
4 plt.show()
```



- Looks like it is favorable to keep players from Manchester City in a fantasy team as they tend to score more fantasy points on average.

We know that players in different positions have specific roles to play in a team. Let's check players in which positions tend to score more fantasy points on average.

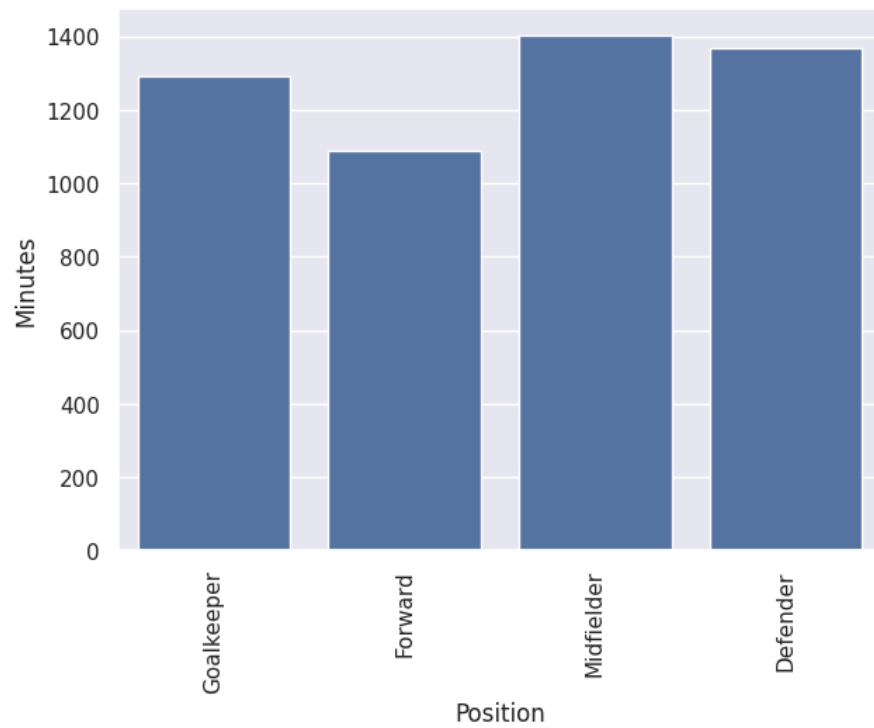
```
1 plt.figure(figsize = (7, 5))
2 sns.barplot(data = df, x = 'Position', y = 'Total_Points', errorbar=('ci', False))
3 plt.xticks(rotation = 90)
4 plt.show()
```



- Midfielders tend to fetch the most number of points for fantasy managers on average.

To effectively utilize their squad depth, managers often rotate the squad to keep key players in shape for tougher games. Let's check the total number of minutes played, on average, across different positions.

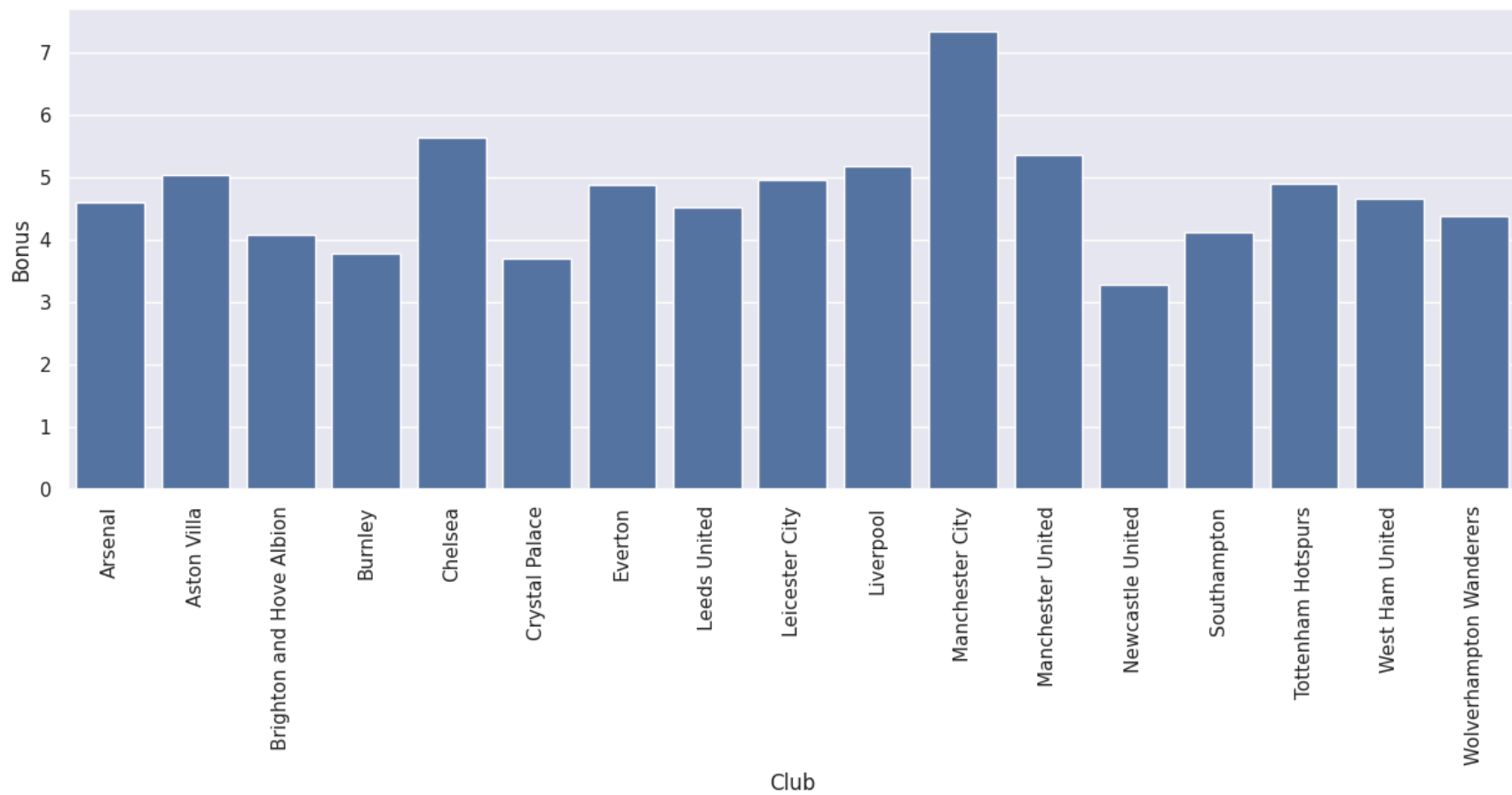
```
1 plt.figure(figsize = (7, 5))
2 sns.barplot(data = df, x = 'Position', y = 'Minutes', errorbar=('ci', False))
3 plt.xticks(rotation = 90)
4 plt.show()
```



- Players in the midfield and defense tend to play more minutes on average than forwards and goalkeepers.

Every point counts in fantasy sports and getting bonus points for a player is always a treat. Let's check which team's players have secured the most bonus points, on average, last season.

```
1 plt.figure(figsize = (15, 5))
2 sns.barplot(data = df, x = 'Club', y = 'Bonus', errorbar=('ci', False))
3 plt.xticks(rotation = 90)
4 plt.show()
```

- It's Manchester City again! The players of this club seem to be great fantasy picks.

Let's see which players scored the most fantasy points last season for different positions of play.

```
1 pos_list = df.Position.unique().tolist()
2 best_df = pd.DataFrame()
3
4 for pos in pos_list:
5     df_aux = df[df.Position == pos]
6     best_df = pd.concat([best_df, df_aux[df_aux.Total_Points == df_aux.Total_Points.max()]['Player_Name', 'Club', 'Position', 'Total_Points']])
7
8 best_df
```



	Player_Name	Club	Position	Total_Points	
36	Emiliano Martinez	Aston Villa	Goalkeeper	186	
403	Harry Kane	Tottenham Hotspurs	Forward	242	
315	Bruno Fernandes	Manchester United	Midfielder	244	
223	Stuart Dallas	Leeds United	Defender	171	

Passaggi successivi:

[Genera codice con best_df](#)

[Visualizza grafici consigliati](#)

[New interactive sheet](#)

- No Manchester City players here! That's surprising.

Let's see the top 10 players with the most fantasy points last season for different positions of play.

```
1 best10_df = pd.DataFrame()
2
3 for pos in pos_list:
4     df_aux = df[df.Position == pos]
5     best10_df = pd.concat([best10_df, df_aux.sort_values('Total_Points', ascending=False).reset_index(drop=True).loc[:10, ['Player_Name', 'Club', 'Position', 'Total_Points']]])
6
7 best10_df
```



	Player_Name	Club	Position	Total_Points
0	Emiliano Martinez	Aston Villa	Goalkeeper	186
1	Ederson Moares	Manchester City	Goalkeeper	160
2	Illan Meslier	Leeds United	Goalkeeper	154
3	Hugo Lloris	Tottenham Hotspurs	Goalkeeper	149
4	Nick Pope	Burnley	Goalkeeper	144
5	Alisson Becker	Liverpool	Goalkeeper	140
6	Edouard Mendy	Chelsea	Goalkeeper	140
7	Lukasz Fabianski	West Ham United	Goalkeeper	133
8	Rui Pedro Patricio	Wolverhampton Wanderers	Goalkeeper	132
9	Bernd Leno	Arsenal	Goalkeeper	131
10	Kasper Schmeichel	Leicester City	Goalkeeper	128
0	Harry Kane	Tottenham Hotspurs	Forward	242
1	Patrick Bamford	Leeds United	Forward	194
2	Jamie Vardy	Leicester City	Forward	187
3	Ollie Watkins	Aston Villa	Forward	168
4	Dominic Calvert-Lewin	Everton	Forward	165
5	Roberto Firmino	Liverpool	Forward	141
6	Chris Wood	Burnley	Forward	138
7	Che Adams	Southampton	Forward	137
8	Callum Wilson	Newcastle United	Forward	134
9	Danny Ings	Southampton	Forward	131
10	Alexandre Lacazette	Arsenal	Forward	129
0	Bruno Fernandes	Manchester United	Midfielder	244
1	Mohamed Salah	Liverpool	Midfielder	231
2	Heung-Min Son	Tottenham Hotspurs	Midfielder	228
3	Sadio Mane	Liverpool	Midfielder	176
4	Marcus Rashford	Manchester United	Midfielder	174
5	Jack Harrison	Leeds United	Midfielder	160
6	Ilkay Gundogan	Manchester City	Midfielder	157
7	James Ward-Prowse	Southampton	Midfielder	156
8	Raheem Sterling	Manchester City	Midfielder	154
9	Tomas Soucek	West Ham United	Midfielder	147
10	Mason Mount	Chelsea	Midfielder	147



0	Stuart Dallas	Leeds United	Defender	171
1	Andrew Robertson	Liverpool	Defender	161
2	Trent Alexander-Arnold	Liverpool	Defender	160
3	Aaron Cresswell	West Ham United	Defender	153
4	Aaron Wan-Bissaka	Manchester United	Defender	144
5	Ruben Dias	Manchester City	Defender	142
6	Benjamin Chilwell	Chelsea	Defender	139
7	Matt Targett	Aston Villa	Defender	138
8	Joao Cancelo	Manchester City	Defender	138
9	Lewis Dunk	Brighton and Hove Albion	Defender	130
10	John Stones	Manchester City	Defender	128

Passaggi successivi:

[Genera codice con best10_df](#)

[Visualizza grafici consigliati](#)

[New interactive sheet](#)

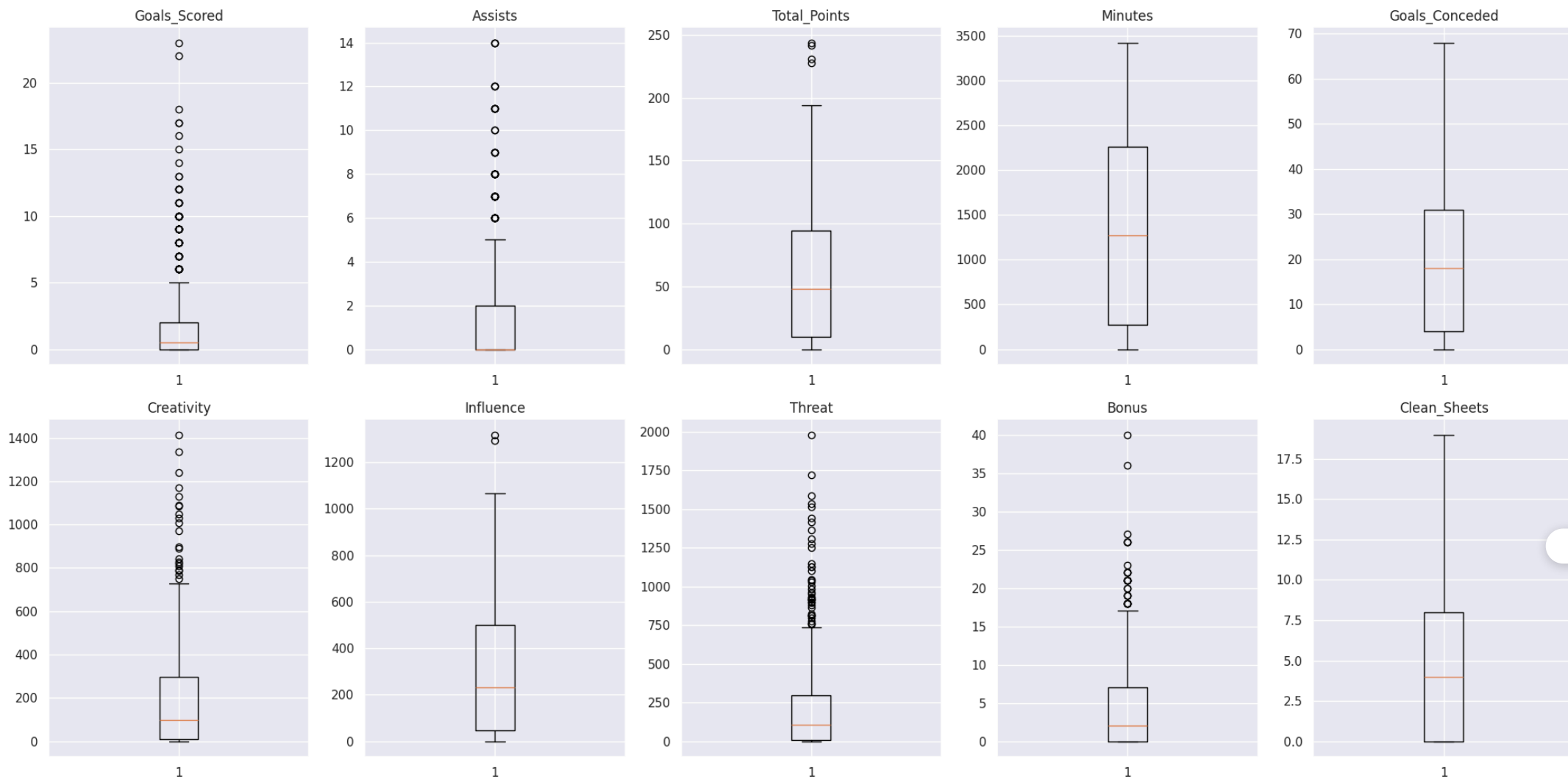
Let's see the distribution of teams now.

- Most of the top 10 players across different positions are from Manchester City and Liverpool.

✓ Checking Outliers

- Let's plot the boxplots of all numerical columns to check for outliers.

```
1 plt.figure(figsize = (20, 10))
2
3 numeric_columns = df.select_dtypes(include = np.number).columns.tolist()
4
5 for i, variable in enumerate(numeric_columns):
6
7     plt.subplot(2, 5, i + 1)
8
9     plt.boxplot(df[variable], whis = 1.5)
10
11     plt.tight_layout()
12
13     plt.title(variable)
14
15 plt.show()
```



- There are some outliers in the data.
- We will not treat them as they are proper values.

✓ Scaling

- Clustering algorithms are distance-based algorithms, and all distance-based algorithms are affected by the scale of the variables. Therefore, we will scale the data before applying clustering.

```
1 # Scaling the data before clustering
2 scaler = StandardScaler()
3 subset = df.iloc[:, 3:].copy()
4 subset_scaled = scaler.fit_transform(subset)
```

```
1 # Creating a dataframe of the scaled data
2 subset_scaled_df = pd.DataFrame(subset_scaled, columns = subset.columns)
```

✓ Applying PCA

- PCA can help to mitigate the effects of collinearity by identifying the most important variables or features that explain the maximum variance in the data. The principal components generated by PCA are uncorrelated with each other, which can reduce the redundancy in the data and can make the clustering more robust.

```
1 # Importing PCA
2 from sklearn.decomposition import PCA
3
4 # Defining the number of principal components to generate
5 n = subset.shape[1]                                # Storing the number of variables in the data
6
7 pca = PCA(n_components = n, random_state = 1)        # Storing PCA function with n components
8
9 data_pca = pd.DataFrame(pca.fit_transform(subset_scaled_df ))    # Applying PCA on scaled data
10
11 # The percentage of variance explained by each principal component is stored
12 exp_var = (pca.explained_variance_ratio_)
```

✓ K-Means Clustering

- K-Means clustering is one of the most popular clustering algorithms used for partitioning a dataset into K clusters. The algorithm works by iteratively assigning each data point to one of the K clusters based on the proximity of the data points to the centroids of the clusters. K-Means clustering is a computationally efficient algorithm that can work well even for datasets with a large number of variables.
- The steps involved in K-Means clustering are as follows:
 - Choose the number of clusters K that you want to partition the data into.
 - Initialize the K centroids randomly.
 - Assign each data point to the nearest centroid.
 - Recalculate the centroids of each cluster as the mean of all the data points assigned to it.
 - Repeat steps 3 and 4 until the centroids no longer change or a maximum number of iterations is reached.

```
1 k_means_df = data_pca.copy()
```

```
1 clusters = range(1, 15)
2 meanDistortions = []
3
```

```

4 for k in clusters:
5     model = KMeans(n_clusters = k, random_state = 1, n_init = "auto")
6     model.fit(data_pca)
7     prediction = model.predict(k_means_df)
8     distortion = (
9         sum(np.min(cdist(k_means_df, model.cluster_centers_, "euclidean"), axis = 1))
10        / k_means_df.shape[0]
11    )
12
13    meanDistortions.append(distortion)
14
15    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)
16
17 plt.plot(clusters, meanDistortions, "bx-")
18 plt.xlabel("k")
19 plt.ylabel("Average Distortion")
20 plt.title("Selecting k with the Elbow Method", fontsize = 20)
21 plt.show()

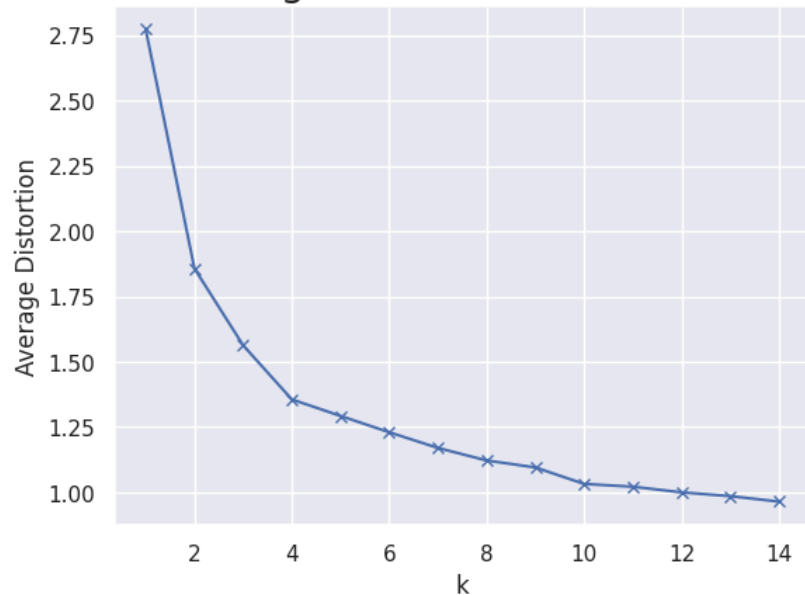
```

```

➦ Number of Clusters: 1   Average Distortion: 2.773037110097803
Number of Clusters: 2   Average Distortion: 1.8571691335599736
Number of Clusters: 3   Average Distortion: 1.5628794778061599
Number of Clusters: 4   Average Distortion: 1.3557368359611872
Number of Clusters: 5   Average Distortion: 1.2924554544010316
Number of Clusters: 6   Average Distortion: 1.2299896613141477
Number of Clusters: 7   Average Distortion: 1.1698205619340916
Number of Clusters: 8   Average Distortion: 1.121916667177523
Number of Clusters: 9   Average Distortion: 1.0952097153818396
Number of Clusters: 10  Average Distortion: 1.0321524046799253
Number of Clusters: 11  Average Distortion: 1.021502640797693
Number of Clusters: 12  Average Distortion: 0.9998153362565623
Number of Clusters: 13  Average Distortion: 0.9853605485319921
Number of Clusters: 14  Average Distortion: 0.9644597167841082

```

Selecting k with the Elbow Method




Observations:



- From the point at 4, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.

We will move ahead with k = 4.

```
1 kmeans = KMeans(n_clusters = 4, random_state = 1, n_init = "auto")
2 kmeans.fit(k_means_df)
```



KMeans

KMeans(n_clusters=4, random_state=1)


```
1 # Creating a copy of the original data
2 df1 = df.copy()
3
4 # Adding K-Means cluster labels to the K-Means and original dataframes
5 k_means_df["KM_segments"] = kmeans.labels_
6 df1["KM_segments"] = kmeans.labels_
```

Cluster Profiles

```
1 km_cluster_profile = df1.groupby("KM_segments").mean(numeric_only = True)
```

```
1 # Creating the "count_in_each_segment" feature in K-Means cluster profile
2
3 km_cluster_profile["count_in_each_segment"] = (
4     df1.groupby("KM_segments")["Total_Points"].count().values
5 )
```

```
1 km_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```



	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets	count_in_each_segment
KM_segments											
0	1.363636	1.878788	103.525253	2670.555556	37.525253	265.671717	579.185859	199.636364	7.676768	10.020202	99
1	0.148936	0.202128	9.824468	238.750000	3.930851	28.171809	43.164894	30.244681	0.409574	0.558511	188
2	9.183333	6.716667	142.150000	2457.266667	33.516667	623.141667	664.133333	880.533333	16.266667	9.250000	60
3	1.503876	1.604651	56.038760	1392.736434	20.573643	188.358915	270.818605	223.255814	3.356589	4.705426	129

```
1 # Let's see the names of the players in each cluster
2 for cl in df1["KM_segments"].unique():
3     print("In cluster {}, the following players are present:".format(cl))
4     print(df1[df1["KM_segments"] == cl]["Player_Name"].unique())
5     print()
```




```
'Tomas Soucek' 'Pedro Lomba Neto']
```

In cluster 0, the following players are present:

```
['Bernd Leno' 'Granit Xhaka' 'Hector Bellerin' 'Kieran Tierney'
 'Rob Holding' 'Douglas Luiz Soares de Paulo' 'Emiliano Martinez'
 'Ezri Konsa Ngoyo' 'John McGinn' 'Matt Targett' 'Matthew Cash'
 'Tyrone Mings' 'Adam Webster' 'Ben White' 'Joel Veltman' 'Lewis Dunk'
 'Robert Sanchez' 'Yves Bissouma' 'Ashley Westwood' 'Ben Mee'
 'Charlie Taylor' 'Dwight McNeil' 'James Tarkowski' 'Josh Brownhill'
 'Matthew Lowton' 'Nick Pope' 'Benjamin Chilwell' 'Cesar Azpilicueta'
 'Edouard Mendy' 'Jorge Luiz Frello Filho' 'Kurt Zouma' 'Reece James'
 'Thiago Silva' 'Andros Townsend' 'Cheikhou Kouyate' 'Joel Ward'
 'Luka Milivojevic' 'Vicente Guaita' 'Abdoulaye Doucoure' 'Ben Godfrey'
 'Jordan Pickford' 'Lucas Digne' 'Mason Holgate' 'Michael Keane'
 'Ezgjjan Alioski' 'Illan Meslier' 'Kalvin Phillips' 'Liam Cooper'
 'Luke Ayling' 'Mateusz Klich' 'James Justin' 'Jonny Evans'
 'Kasper Schmeichel' 'Timothy Castagne' 'Wesley Fofana' 'Wilfred Ndidi'
 'Alisson Becker' 'Fabio Henrique Tavares' 'Georginio Wijnaldum'
 'Bernardo Silva' 'Ederson Moares' 'Joao Cancelo' 'John Stones'
 'Rodrigo Hernandez' 'Ruben Dias' 'Aaron Wan-Bissaka' 'David de Gea'
 'Frederico Rodrigues de Paula Santos' 'Harry Maguire' 'Paul Pogba'
 'Scott McTominay' 'Victor Lindelof' 'Jonjo Shelvey' 'Karl Darlow'
 'Miguel Almiron' 'Alex McCarthy' 'Jan Bednarek' 'Jannik Vestergaard'
 'Kyle Walker-Peters' 'Ryan Bertrand' 'Stuart Armstrong' 'Eric Dier'
 'Hugo Lloris' 'Pierre-Emile Hojbjerg' 'Sergio Reguilon' 'Tanguy Ndombele'
 'Toby Alderweireld' 'Angelo Ogbonna' 'Declan Rice' 'Lukasz Fabianski'
 'Vladimir Coufal' 'Adama Traore' 'Conor Coady' 'Joao Santos Moutinho'
 'Leander Dendoncker' 'Nelson Semedo' 'Romain Saiss' 'Ruben Neves'
 'Rui Pedro Patricio']
```

In cluster 3, the following players are present:

```
['Calum Chambers' 'Daniel Ceballos' 'David Luiz' 'Emile Smith Rowe'
 'Gabriel Maghalaes' 'Martin Odegaard' 'Mohamed Naser El Sayed Elneny'
 'Thomas Partey' 'Willian Borges Da Silva' 'Mahmoud Ahmed Ibrahim Hassan'
 'Ross Barkley' 'Aaron Connolly' 'Adam Lallana' 'Alexis Mac Allister'
 'Dan Burn' 'Danny Welbeck' 'Mathew Ryan' 'Solomon March' 'Steven Alzate'
 'Tariq Lamptey' 'Ashley Barnes' 'Erik Pieters' 'Jack Cork'
 'Jay Rodriguez' 'Jeff Hendrick' 'Johann Berg Gudmundsson' 'Matej Vydra'
 'Robbie Brady' 'Andreas Christensen' 'Antonio Rudiger'
 'Callum Hudson-Odoi' 'Christian Pulisic' 'Hakim Ziyech' 'Kai Havertz'
 'Marcos Alonso' 'Mateo Kovacic' 'N'Golo Kante' 'Olivier Giroud'
 'Tammy Abraham' 'Gary Cahill' 'Jairo Riedewald' 'James McArthur'
 'Jeffrey Schlupp' 'Jordan Ayew' 'Michy Batshuayi' 'Nathaniel Clyne'
 'Patrick van Aanholt' 'Scott Dann' 'Tyrick Mitchell' 'Alex Iwobi'
 'Allan Marques Loureiro' 'Andre Tavares Gomes' 'Seamus Coleman'
 'Tom Davies' 'Yerry Mina' 'Diego Llorente' 'Helder Costa'
 'Pascal Struijk' 'Robin Koch' 'Tyler Roberts' 'Ayoze Perez'
 'Calgar Soyuncu' 'Dennis Praet' 'Marc Albrighton' 'Nampalys Mendy'
 'Ricardo Domingos Barbosa Pereira' 'Curtis Jones' 'Dean Henderson'
 'Diogo Jota' 'James Milner' 'Jordan Henderson' 'Nathaniel Phillips'
 'Thiago Alcantara' 'Aymeric Laporte' 'Benjamin Mendy'
 'Fernando Luiz Rosa' 'Ferran Torres' 'Kyle Walker' 'Oleksandr Zinchenko'
 'Sergio Aguero' 'Anthony Martial' 'Daniel James' 'Eric Bailly'
 'Mason Greenwood' 'Nemanja Matic' 'Allan Saint-Maximin' 'Ciaran Clark'
 'Emil Krafth' 'Fabian Schar' 'Federico Fernandez' 'Isaac Hayden'
 'Jacob Murphy' 'Jamaal Lascelles' 'Jamal Lewis' 'Joelinton de Lira'
 'Joseph Willock' 'Martin Dubravka' 'Matt Ritchie' 'Paul Dummett'
 'Ryan Fraser' 'Sean Longstaff' 'Ibrahima Diallo' 'Jack Stephens'
 'Moussa Diene' 'Nathan Redmond' 'Orion Romeu Vidal' 'Takumi Minamino']
```

```
1 df1.groupby(["KM_segments", "Position"])['Player_Name'].count()
```



		Player_Name
KM_segments	Position	
0	Defender	50
	Goalkeeper	17
	Midfielder	32
1	Defender	70
	Forward	28
	Goalkeeper	25
	Midfielder	65
2	Defender	5
	Forward	20
	Midfielder	35
3	Defender	47
	Forward	16
	Goalkeeper	3
	Midfielder	63

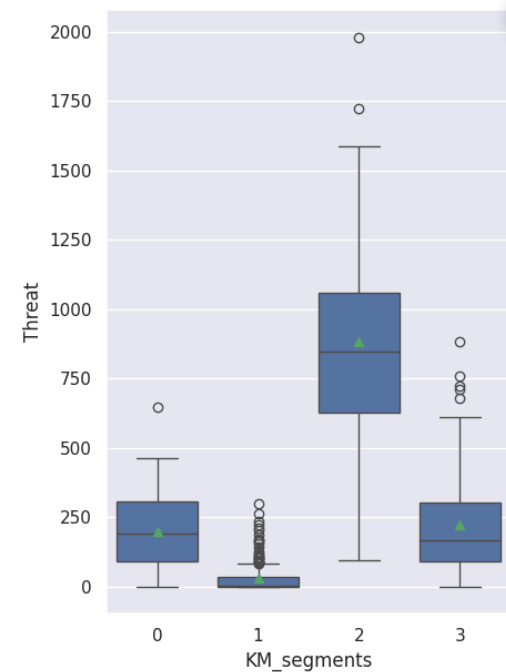
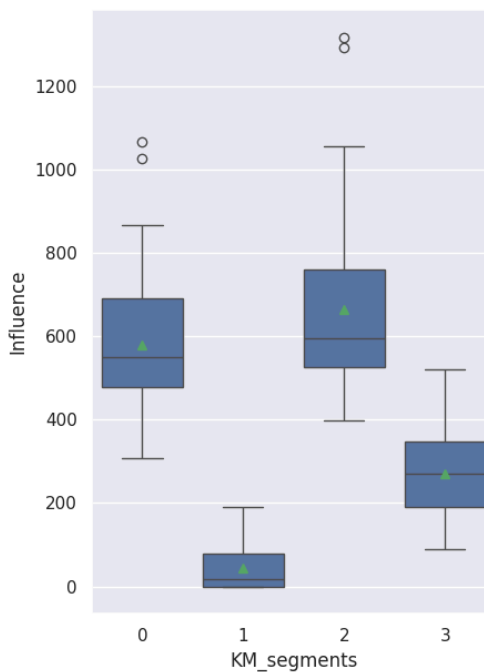
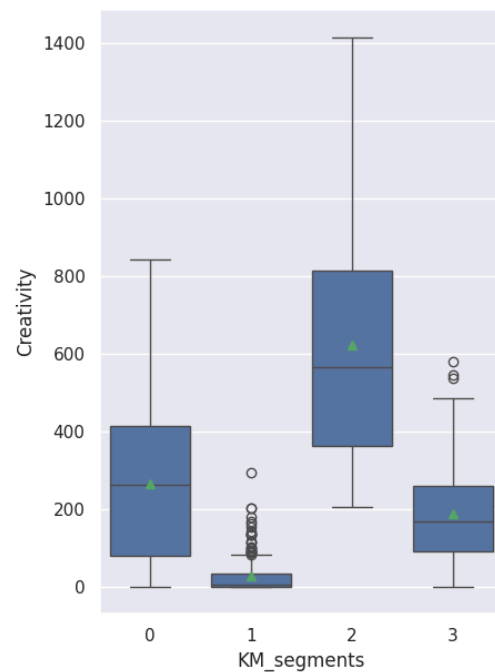
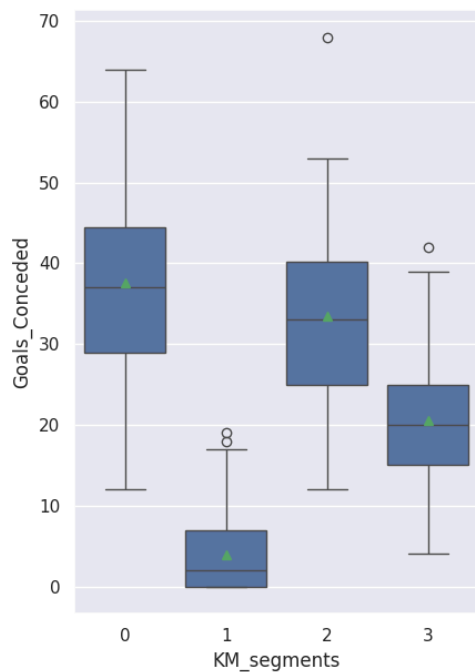
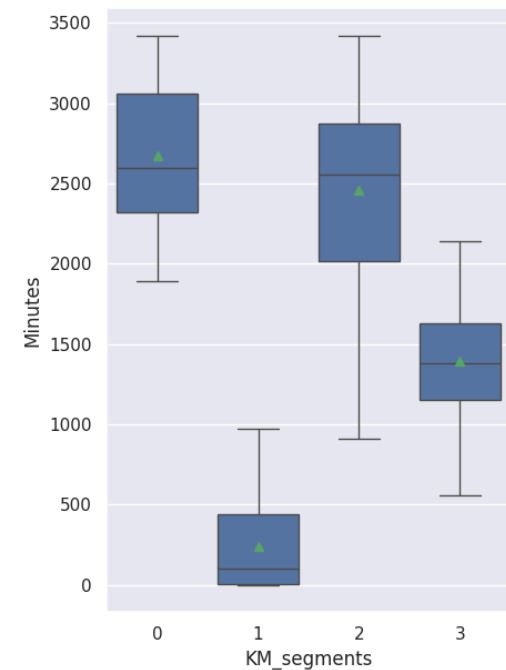
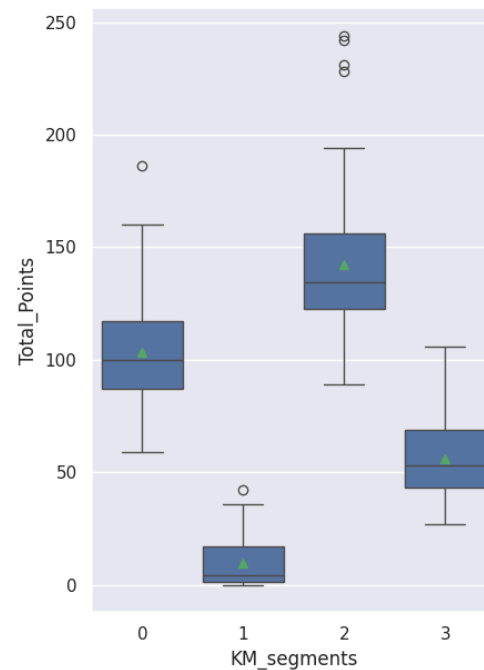
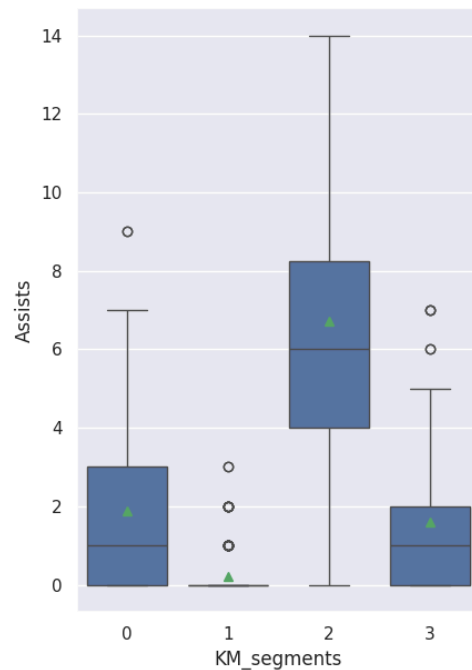
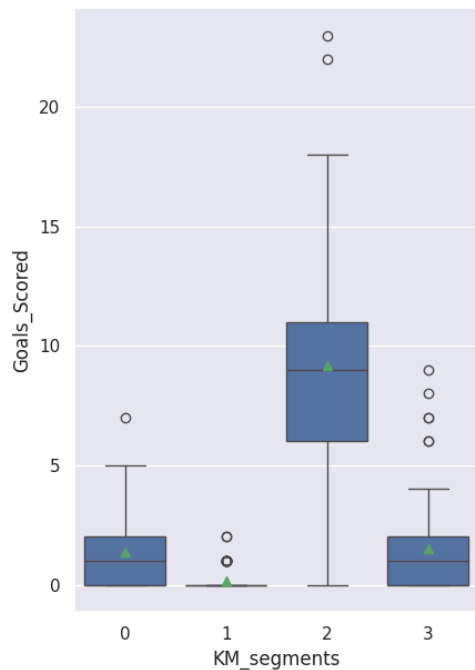
dtype: int64

- Cluster 1 has no forwards, so it is likely to have players with more defensive duties in the team.
- Cluster 2 has no goalkeepers, so it is likely to have players with more offensive duties in the team.

Let's plot the boxplot

```
1 fig, axes = plt.subplots(3, 4, figsize = (20, 20))
2 counter = 0
3
4 for ii in range(3):
5     for jj in range(4):
6         if counter < 10:
7             sns.boxplot(
8                 ax = axes[ii][jj],
9                 data = df1,
10                y = df1.columns[3 + counter],
11                x = "KM_segments", showmeans = True
12            )
13            counter = counter + 1
14
15 fig.tight_layout(pad = 3.0)
```

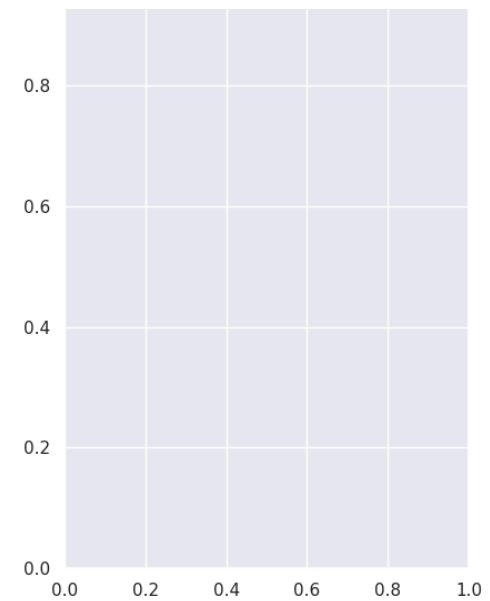
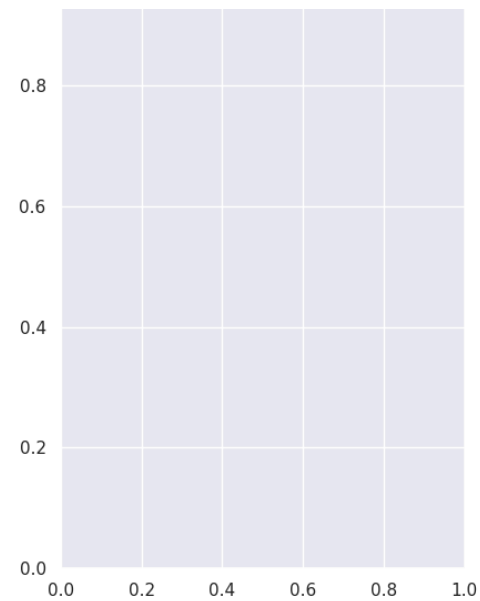
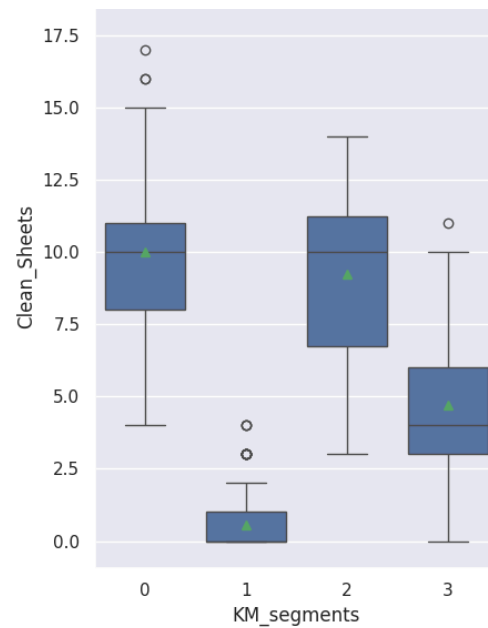
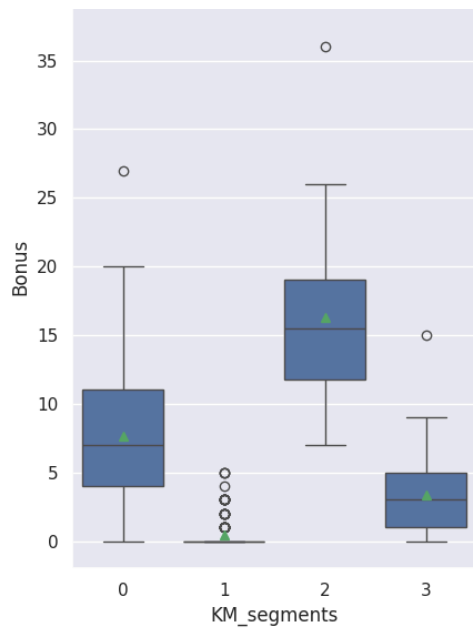
17



40

1.0

1.0



Characteristics of each cluster:

• Cluster 0

- There are 128 players in this cluster.
- Most of the players in this cluster have a few goals and assists, and the total fantasy points scored in the previous season are low.
- Most of the players in this cluster had a moderate game time, a low creativity score, a low influence score, and a moderate threat score.
- Most of the players in this cluster received low bonus points.

• Cluster 1

- There are 99 players in this cluster.
- Most of the players in this cluster have a few goals and assists, and the total fantasy points scored in the previous season are moderate.
- Most of the players in this cluster had a high game time, a moderate creativity score, a high influence score, and a moderate threat score.
- Most of the players in this cluster received moderate bonus points.

• Cluster 2

- There are 61 players in this cluster.
- Most of the players in this cluster have a lot of goals and assists, and the total fantasy points scored in the previous season are high.

- Most of the players in this cluster had a high game time, high creativity, influence, and scores.
- Most of the players in this cluster received high bonus points.
- **Cluster 3**
 - There are 188 players in this cluster.
 - Players in this cluster, except a few, have no goals and assists and did not score any fantasy points scored in the previous season.
 - Most of the players in this cluster had a low game time, and low creativity, influence, and threat scores.
 - Players in this cluster, except a few, received no bonus points.

✓ K-Medoids Clustering

- K-Medoids clustering is a variant of K-Means clustering that uses medoids instead of centroids to define the clusters. Medoids are data points within a cluster that have the minimum average dissimilarity to all the other points in the cluster.
- The steps involved in K-Medoids clustering are as follows:
 - Choose the number of clusters K that you want to partition the data into.
 - Initialize K medoids randomly.
 - Assign each data point to the nearest medoid.
 - For each medoid, compute the average dissimilarity to all the other points in the cluster.
 - For each medoid and non-medoid pair, swap the medoid and non-medoid and compute the new total dissimilarity of the cluster.
 - If the total dissimilarity decreases after the swap, keep the new medoid, otherwise keep the old medoid.
 - Repeat steps 3 to 6 until the medoids no longer change or a maximum number of iterations is reached.

K-Medoids clustering is a robust algorithm that can handle non-linear clusters and is less sensitive to outliers compared to K-Means clustering. However, it can be computationally expensive for large datasets, as it requires computing the pairwise dissimilarities between all the data points.

```
1 k_med_df = data_pca.copy()
```

```
1 kmed = KMedoids(n_clusters = 4, random_state = 1) # Create K-Medoids with nclusters = 4
2 kmed.fit(k_med_df)
```



```
▼ KMedoids ⓘ
KMedoids(n_clusters=4, random_state=1)
```

```
1 # Creating a copy of the original data
2 df2 = df.copy()
3
4 # Add K-Medoids cluster labels to K-Medoids data
5 k_med_df["KMed_segments"] = kmed.labels_
6 # Add K-Medoids cluster labels to the whole data
7 df2["KMed_segments"] = kmed.labels_
```

✓ Cluster Profiling

```
1 kmed_cluster_profile = df2.groupby("KMed_segments").mean(numeric_only = True)
```

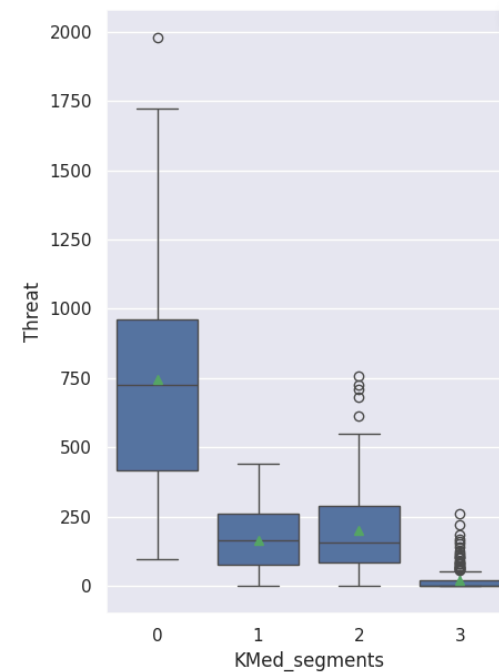
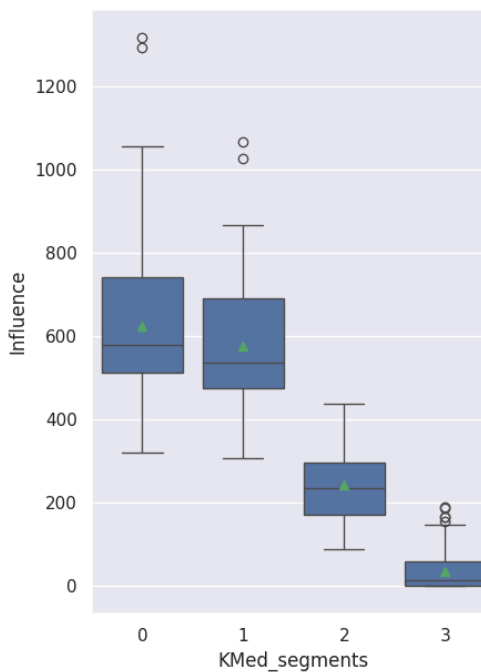
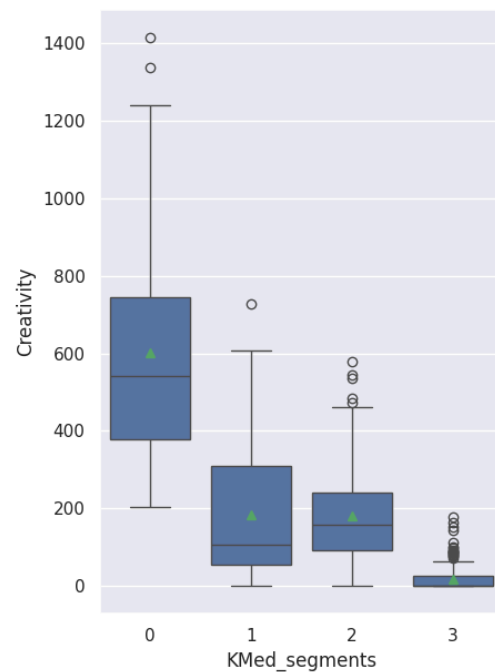
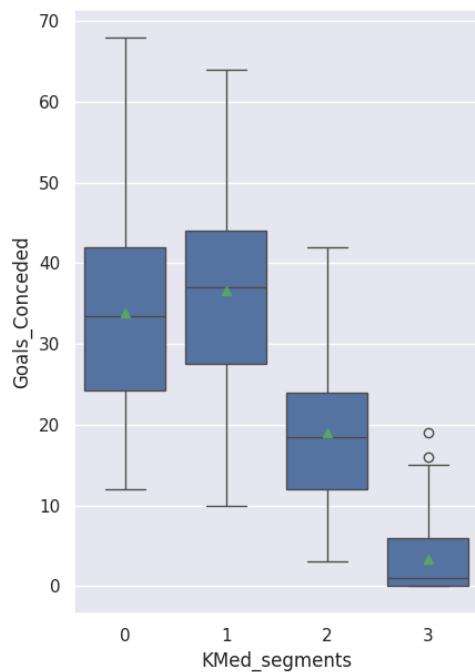
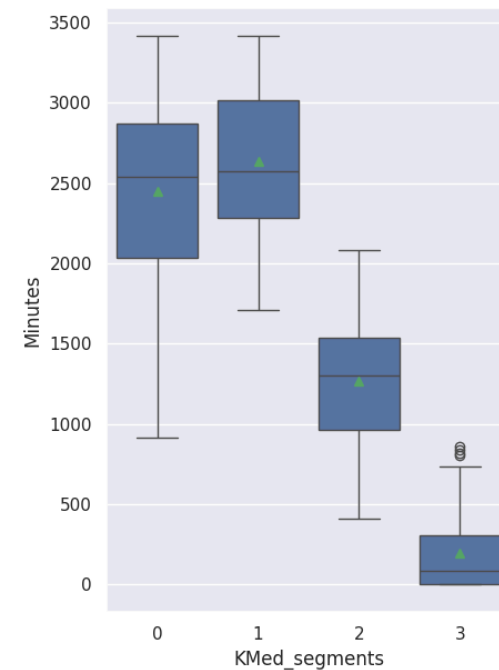
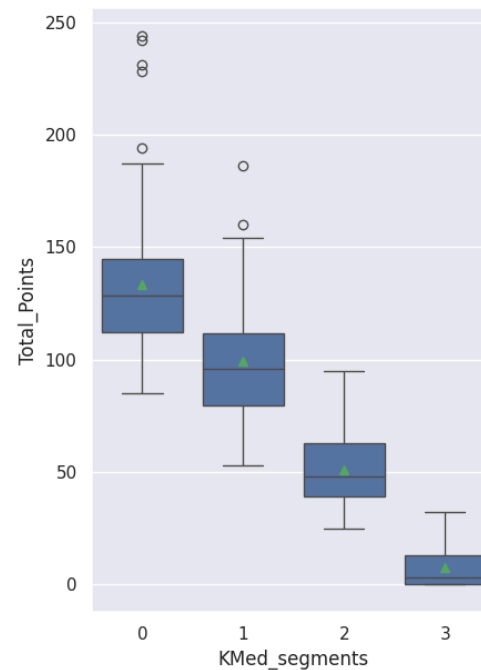
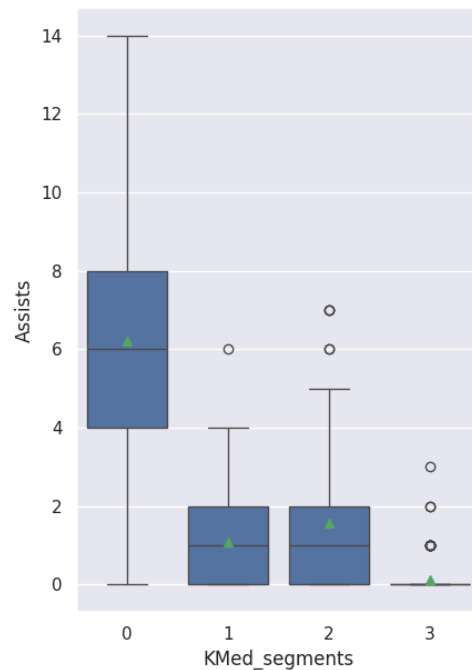
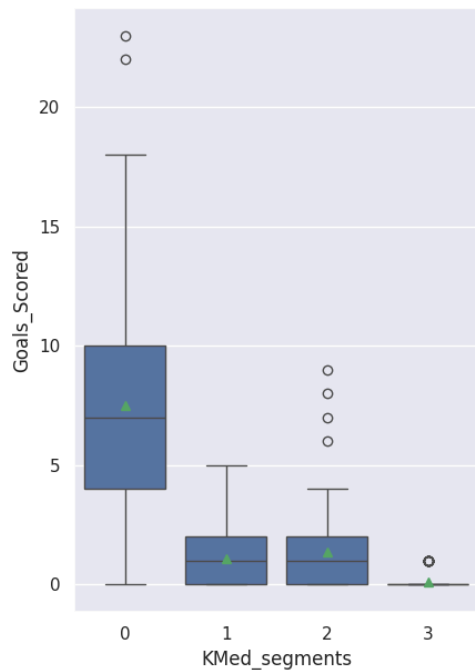
```
1 kmed_cluster_profile["count_in_each_segment"] = (  
2     df2.groupby("KMed_segments")["Total_Points"].count().values  
3 )  
4  
5 kmed_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```



	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets	count_in_each_segment
KMed_segments											
0	7.512195	6.195122	133.243902	2452.243902	33.853659	602.902439	625.653659	745.402439	14.573171	9.231707	82
1	1.068966	1.091954	99.528736	2638.195402	36.632184	184.582759	575.818391	166.333333	6.988506	9.931034	87
2	1.338235	1.558824	51.073529	1270.051471	18.977941	180.458824	242.588235	203.102941	2.904412	4.205882	136
3	0.099415	0.111111	7.736842	193.187135	3.362573	18.979532	34.188304	22.608187	0.280702	0.385965	171

```
1 fig, axes = plt.subplots(3, 4, figsize = (20, 20))  
2 counter = 0  
3  
4 for ii in range(3):  
5     for jj in range(4):  
6         if counter < 10:  
7             sns.boxplot(  
8                 ax = axes[ii][jj],  
9                 data = df2,  
10                y = df2.columns[3 + counter],  
11                x = "KMed_segments", showmeans = True  
12            )  
13            counter = counter + 1  
14  
15 fig.tight_layout(pad = 3.0)
```

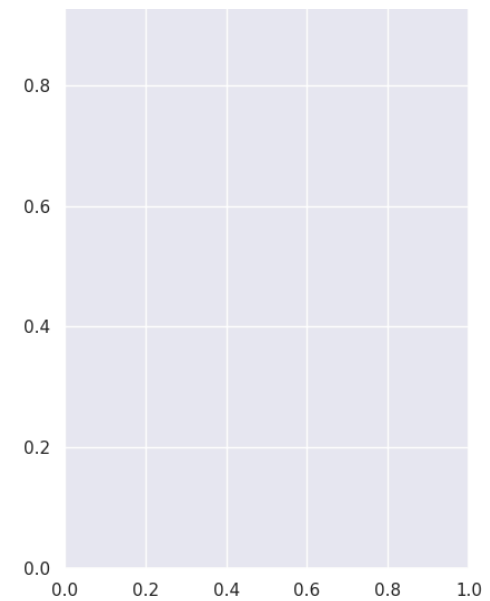
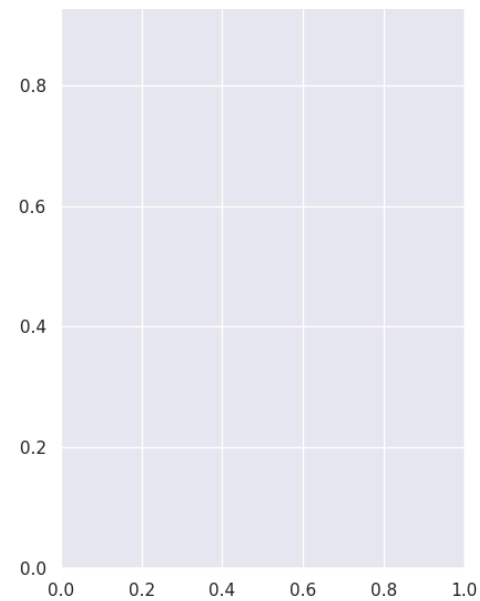
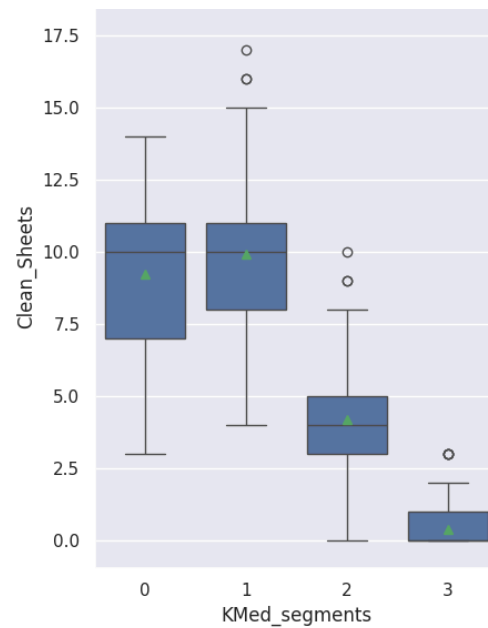
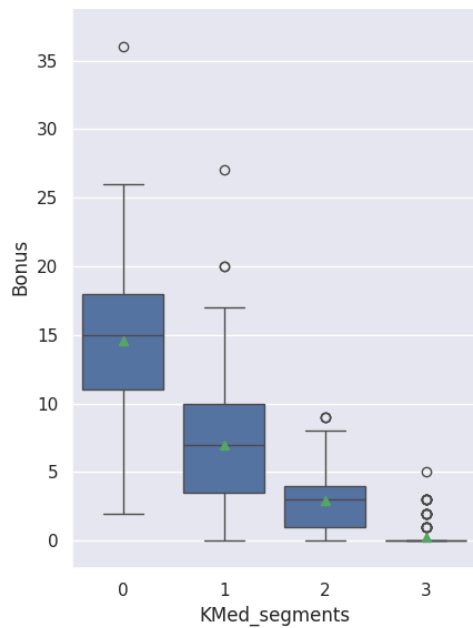
17



40

1.0

1.0



Comparison of cluster profiles from K-Means and K-Medoids

1. There is a difference in the distribution of each cluster in both algorithms. The cluster groups in K-Medoids are more evenly distributed since it uses a median which is less likely to get affected by the external data/outliers.
2. The cluster profiles are the same for both algorithms, Cluster number is changing however the cluster profiles remain the same.

Characteristics of each cluster

• Cluster 0

- There are 82 players in this cluster.
- Most of the players in this cluster have high goals and assists, and the total fantasy points scored in the previous season are high.
- Most of the players in this cluster had a moderate game time with a high creativity score, high influence score, and a moderately high score.
- Most of the players in this cluster received high bonus points.
- Most of the players in this cluster received moderate clean sheets with an average of 9.5.

• Cluster 1

- There are 87 players in this cluster.
- Most of the players in this cluster have a few goals and assists, and the total fantasy points scored in the previous season are high.

- Most of the players in this cluster had a high game time, a moderate creativity score, a high influence score, and a less threat score.
- Most of the players in this cluster received moderate bonus points.
- **Cluster 2**
 - There are 136 players in this cluster.
 - Most of the players in this cluster have moderate goals and assists, and the total fantasy points scored in the previous season are moderate.
 - Most of the players in this cluster had a moderate game time, with low creativity, influence scores, and moderate threat scores.
 - Most of the players in this cluster received fewer bonus points.
- **Cluster 3**
 - There are 171 players in this cluster.
 - Players in this cluster, except a few, have no goals and assists and did not score any fantasy points scored in the previous season.
 - Most of the players in this cluster had a low game time, and low creativity, influence, and threat scores.
 - Players in this cluster, except a few, received no bonus points.

✓ Hierarchical Clustering

- Hierarchical clustering is a popular unsupervised learning algorithm used for grouping similar data points into clusters based on the hierarchical structure of the data. The algorithm works by recursively merging the closest data points or clusters until all the data points belong to a single cluster.
- There are two main types of hierarchical clustering: agglomerative and divisive. Agglomerative clustering starts with each data point as a separate cluster and recursively merges the closest clusters until all the data points belong to a single cluster. Divisive clustering, on the other hand, starts with all the data points in a single cluster and recursively splits the clusters until each data point belongs to a separate cluster. Here, we will implement the agglomerative clustering.
- The steps involved in agglomerative clustering are as follows:
 - Assign each data point to a separate cluster.
 - Compute the dissimilarity between each pair of clusters.
 - Merge the two closest clusters into a single cluster.
 - Update the dissimilarity between the new cluster and the remaining clusters.
 - Repeat steps 3 and 4 until all the data points belong to a single cluster.
- Agglomerative clustering can be used with different linkage criteria to compute the dissimilarity between clusters. The most common linkage criteria are:
 - Single linkage: The dissimilarity between two clusters is the distance between the closest two data points in the clusters.
 - Complete linkage: The dissimilarity between two clusters is the distance between the farthest two data points in the clusters.
 - Average linkage: The dissimilarity between two clusters is the average distance between all the data point pairs in the clusters.

Agglomerative clustering can be computationally expensive for large datasets, as it requires computing the pairwise dissimilarities between all the data points. However, it can be useful for datasets where the underlying structure is hierarchical or where the number of clusters is not known a priori.

```
1 data_pca.head()
```



	0	1	2	3	4	5	6	7	8	9
0	-2.916600	0.569939	-0.041871	0.190663	0.003485	0.008158	-0.042314	0.064757	-0.057486	-0.006269
1	3.815468	1.999554	-2.216345	0.757341	-0.119000	-0.541975	-0.233941	-0.293053	-0.075356	0.238019
2	1.943396	-2.757446	-0.958238	0.816920	0.041188	0.376978	0.005235	0.138392	0.306183	0.136656
3	3.502427	1.043441	0.581995	-0.969567	0.415848	-0.358456	0.661746	0.376272	0.260200	0.155934
4	-1.153639	0.422189	0.642307	0.269271	-0.271583	0.405367	-0.033575	0.098886	0.035234	0.034877



Passaggi successivi:

[Genera codice con data_pca](#)[Visualizza grafici consigliati](#)[New interactive sheet](#)

```
1 hc_df = data_pca.copy()
```

```
1 hc_df1 = hc_df.copy()
```

```
1 # List of distance metrics
2 distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"]
3
4 # List of linkage methods
5 linkage_methods = ["single", "complete", "average", "weighted"]
6
7 high_cophenet_corr = 0
8 high_dm_lm = [0, 0]
9
10 for dm in distance_metrics:
11     for lm in linkage_methods:
12         Z = linkage(hc_df1, metric = dm, method = lm)
13         c, coph_dists = cophenet(Z, pdist(hc_df))
14         print(
15             "Cophenetic correlation for {} distance and {} linkage is {}".format(
16                 dm.capitalize(), lm, c
17             )
18         )
19         if high_cophenet_corr < c:
20             high_cophenet_corr = c
21             high_dm_lm[0] = dm
22             high_dm_lm[1] = lm
23
24 # Printing the combination of distance metric and linkage method with the highest cophenetic correlation
25 print('*'*100)
26 print(
27     "Highest cophenetic correlation is {}, which is obtained with {} distance and {} linkage.".format(
28         high_cophenet_corr, high_dm_lm[0].capitalize(), high_dm_lm[1]
29     )
30 )
```



Cophenetic correlation for Euclidean distance and single linkage is 0.8430175514228706.
Cophenetic correlation for Euclidean distance and complete linkage is 0.7412041292261758.
Cophenetic correlation for Euclidean distance and average linkage is 0.8476499945585416.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.8624581351067481.

```

Cophenetic correlation for Chebyshev distance and single linkage is 0.8381223141111797.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.8028394390632131.
Cophenetic correlation for Chebyshev distance and average linkage is 0.8167064931302253.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.8448497876639632.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.8065008904132246.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.6583135946489012.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.7747800632434058.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.6486408054242747.
Cophenetic correlation for Cityblock distance and single linkage is 0.840183551166332.
Cophenetic correlation for Cityblock distance and complete linkage is 0.8241586035407026.
Cophenetic correlation for Cityblock distance and average linkage is 0.8564523087071934.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.8395672301050401.
*****
Highest cophenetic correlation is 0.8624581351067481, which is obtained with Euclidean distance and weighted linkage.

```

Let's explore different linkage methods with Euclidean distance only.

```

1 # List of linkage methods
2 linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]
3
4 high_cophenet_corr = 0
5 high_dm_lm = [0, 0]
6
7 for lm in linkage_methods:
8     Z = linkage(hc_df1, metric = "euclidean", method = lm)
9     c, coph_dists = cophenet(Z, pdist(hc_df))
10    print("Cophenetic correlation for {} linkage is {}".format(lm, c))
11    if high_cophenet_corr < c:
12        high_cophenet_corr = c
13        high_dm_lm[0] = "euclidean"
14        high_dm_lm[1] = lm
15
16 # Printing the combination of distance metric and linkage method with the highest cophenetic correlation
17 print('*'*100)
18 print(
19     "Highest cophenetic correlation is {}, which is obtained with {} linkage.".format(
20         high_cophenet_corr, high_dm_lm[1]
21     )
22 )

```

```

🔗 Cophenetic correlation for single linkage is 0.8430175514228706.
Cophenetic correlation for complete linkage is 0.7412041292261758.
Cophenetic correlation for average linkage is 0.8476499945585416.
Cophenetic correlation for centroid linkage is 0.8068296032280463.
Cophenetic correlation for ward linkage is 0.57773844586155.
Cophenetic correlation for weighted linkage is 0.8624581351067481.
*****
Highest cophenetic correlation is 0.8624581351067481, which is obtained with weighted linkage.

```

We see that the cophenetic correlation is maximum with Euclidean distance and weighted linkage.

Let's view the dendrograms for the different linkage methods.

```

1 # List of linkage methods
2 linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

```

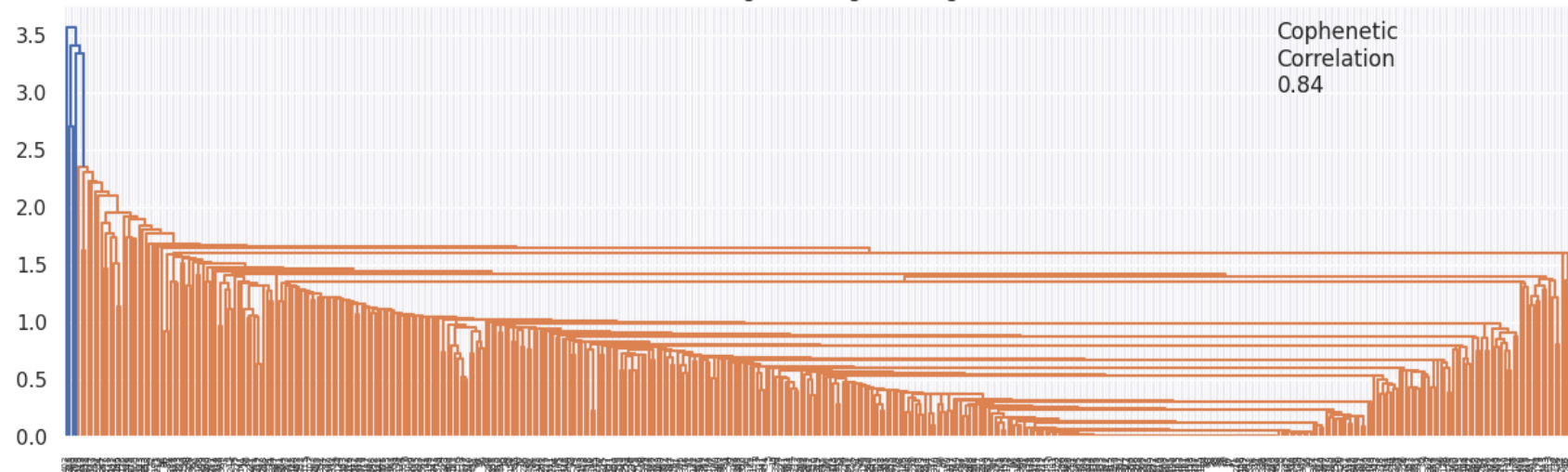
```

3
4 # Lists to save results of cophenetic correlation calculation
5 compare_cols = ["Linkage", "Cophenetic Coefficient"]
6 compare = []
7
8 # To create a subplot image
9 fig, axs = plt.subplots(len(linkage_methods), 1, figsize = (15, 30))
10
11 # We will enumerate through the list of linkage methods above
12 # For each linkage method, we will plot the dendrogram and calculate the cophenetic correlation
13 for i, method in enumerate(linkage_methods):
14     Z = linkage(hc_df1, metric = "euclidean", method = method)
15
16     dendrogram(Z, ax = axs[i])
17     axs[i].set_title(f"Dendrogram ({method.capitalize()} Linkage)")
18
19     coph_corr, coph_dist = cophenet(Z, pdist(hc_df))
20     axs[i].annotate(
21         f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
22         (0.80, 0.80),
23         xycoords="axes fraction",
24     )
25
26     compare.append([method, coph_corr])

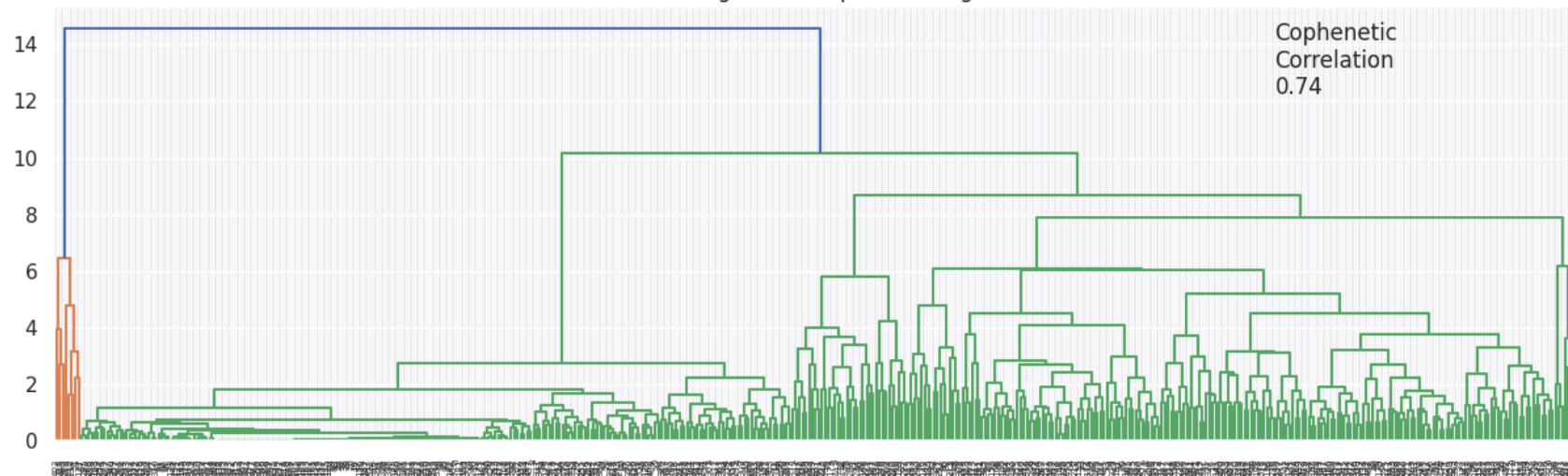
```



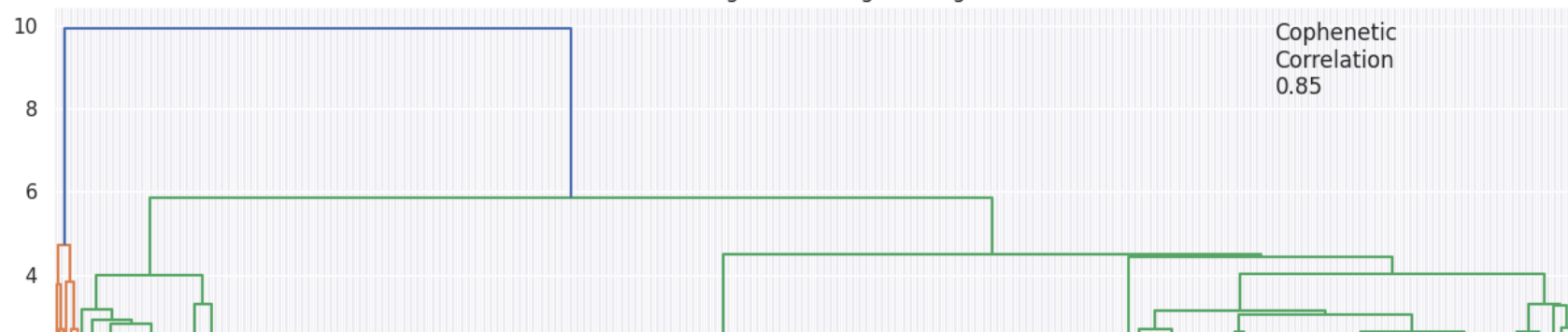
Dendrogram (Single Linkage)

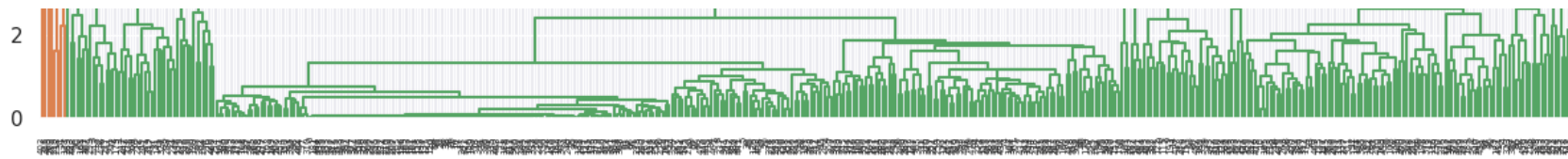


Dendrogram (Complete Linkage)

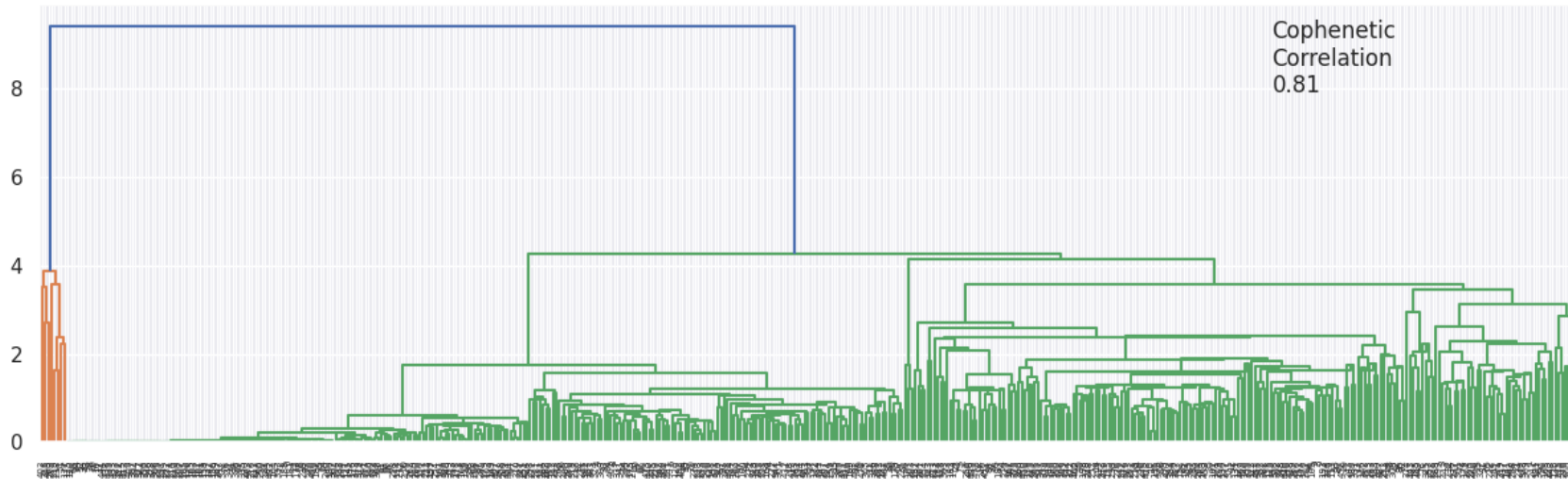


Dendrogram (Average Linkage)

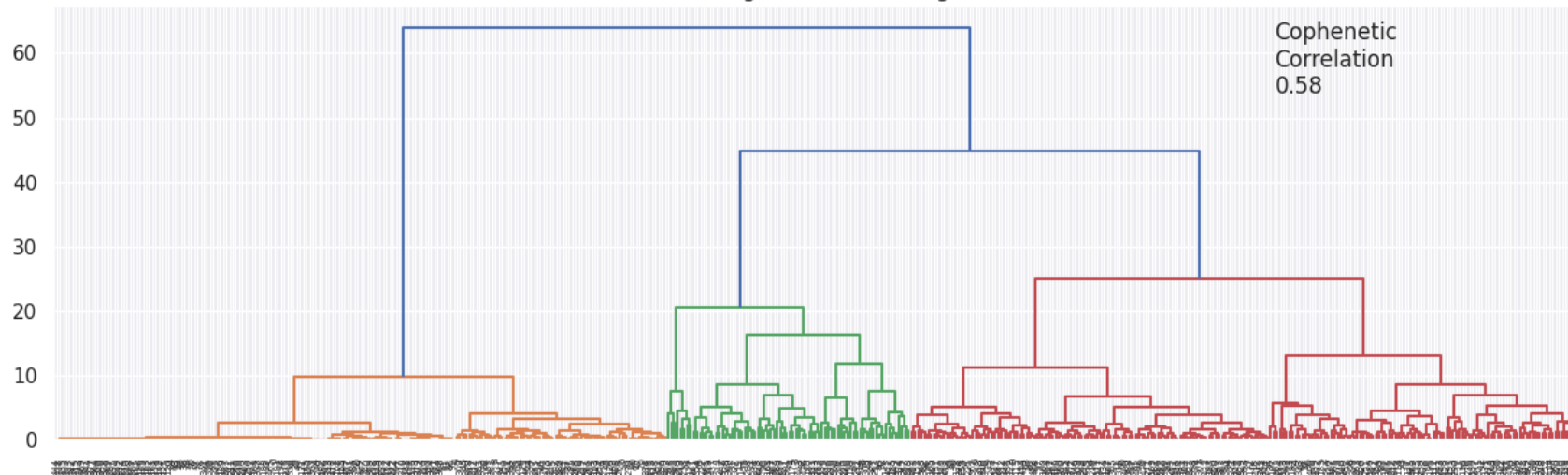




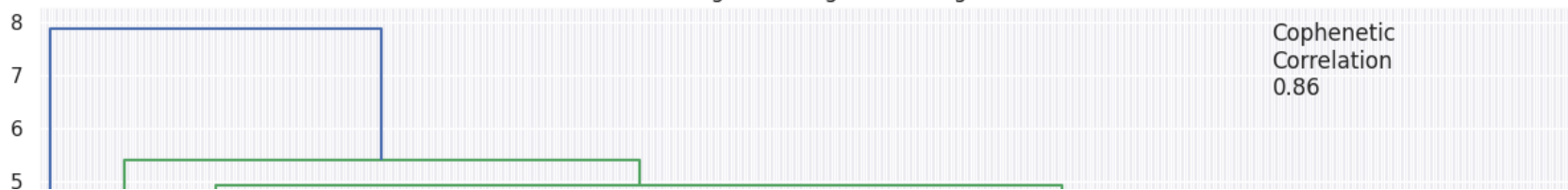
Dendrogram (Centroid Linkage)

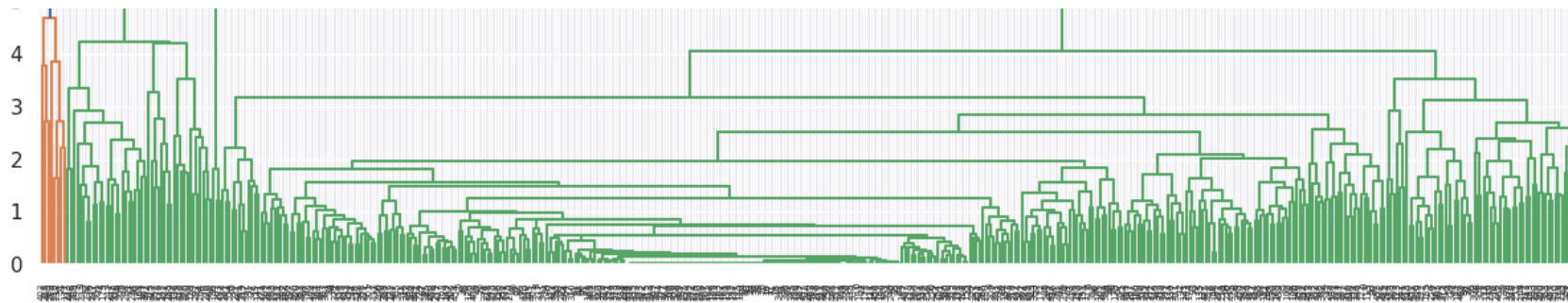


Dendrogram (Ward Linkage)



Dendrogram (Weighted Linkage)





```

1 # Create and print a dataframe to compare cophenetic correlations for different linkage methods
2 df_cc = pd.DataFrame(compare, columns = compare_cols)
3
4 df_cc = df_cc.sort_values(by = "Cophenetic Coefficient")
5 df_cc

```

	Linkage	Cophenetic Coefficient	
4	ward	0.577774	
1	complete	0.741204	
3	centroid	0.806830	
0	single	0.843018	
2	average	0.847650	
5	weighted	0.862458	

Passaggi successivi:

[Genera codice con df_cc](#)
[Visualizza grafici consigliati](#)
[New interactive sheet](#)

Let's move ahead with 4 clusters, Euclidean distance, and average linkage as the sklearn implementation does not support weighted linkage.

```

1 HCmodel = AgglomerativeClustering(n_clusters = 4, metric = "euclidean", linkage = "average")
2 HCmodel.fit(hc_df1)

```

	AgglomerativeClustering	
	AgglomerativeClustering(linkage='average', n_clusters=4)	

```

1 # Creating a copy of the original data
2 df3 = df.copy()
3
4 # Adding hierarchical cluster labels to the original and whole dataframes
5 hc_df["HC_segments_L1"] = HCmodel.labels_
6 df3["HC_segments_L1"] = HCmodel.labels_

```

Cluster Profiling

```

1 hc_cluster_profile = df3.groupby("HC_segments_L1").mean(numeric_only = True)

```

```

1 hc_cluster_profile["count_in_each_segment"] = (
2     df3.groupby("HC_segments_L1")["Total_Points"].count().values
3 )

```

```

1 hc_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)

```




	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets	count_in_each_segment
HC_segments_L1											
0	0.881517	1.139810	47.969194	1205.945498	17.580569	148.574408	249.536967	131.753555	3.293839	4.182464	422
1	16.800000	9.200000	189.000000	3033.200000	44.000000	494.340000	860.720000	1591.600000	21.800000	10.800000	5
2	8.565217	5.826087	129.391304	2238.934783	29.760870	543.273913	586.234783	861.739130	14.021739	8.739130	46
3	19.333333	13.000000	238.000000	3101.000000	37.000000	1041.300000	1221.000000	1294.666667	34.000000	12.666667	3

```
1 # Let's see the names of the players in each cluster
2 for cl in df3["HC_segments_L1"].unique():
3     print("In cluster {}, the following players are present:".format(cl))
4     print(df3[df3["HC_segments_L1"] == cl]["Player_Name"].unique())
5     print()
```



William Jose Willy Doty]

In cluster 2, the following players are present:
['Alexandre Lacazette' 'Bukayo Saka' 'Nicolas Pepe'
'Pierre-Emerick Aubameyang' 'Anwar El Ghazi' 'Bertrand Traore'
'Jack Grealish' 'Danny Welbeck' 'Leandro Trossard' 'Neal Maupay'
'Pascal Gross' 'Chris Wood' 'Mason Mount' 'Timo Werner'
'Christian Benteke' 'Wilfried Zaha' 'Gylfi Sigurdsson'
'Richarlison de Andrade' 'Jack Harrison' 'Raphael Dias Belloli'
'Rodrigo Moreno' 'Stuart Dallas' 'Harvey Barnes' 'James Maddison'
'Kelechi Iheanacho' 'Roberto Firmino' 'Sadio Mane'
'Gabriel Fernando de Jesus' 'Ilkay Gundogan' 'Kevin De Bruyne'
'Phil Foden' 'Raheem Sterling' 'Riyad Mahrez' 'Edinson Cavani'
'Marcus Rashford' 'Mason Greenwood' 'Callum Wilson' 'Che Adams'
'Danny Ings' 'Gareth Bale' 'Jarrod Bowen' 'Jesse Lingard'
'Michail Antonio' 'Pablo Fornals' 'Tomas Soucek' 'Pedro Lomba Neto']

```
1 df3.groupby(["HC_segments_L1", "Position"])['Player_Name'].count()
```



		Player_Name
HC_segments_L1	Position	
0	Defender	171
	Forward	43
	Goalkeeper	45
	Midfielder	163
1	Forward	4
	Midfielder	1
2	Defender	1
	Forward	16
	Midfielder	29
3	Forward	1
	Midfielder	2


dtype: int64

We see that most of the players have been grouped into one cluster, and there are two very sparse clusters. This clustering does not look good as the clusters do not have enough variability.

Let us try using Ward linkage as it has more distinct and separated clusters (as seen from it's dendrogram before). 4 appears to be a good number of clusters from the dendrogram for Ward linkage.

```
1 hc_df2 = data_pca.copy()
```


```
1 HCmodel = AgglomerativeClustering(n_clusters = 4, metric = "euclidean", linkage = "average")
2 HCmodel.fit(hc_df2)
```




▼

AgglomerativeClustering

AgglomerativeClustering(linkage='average', n_clusters=4)






```
1 # Creating a copy of the original data
2 df3 = df.copy()
3
4 # Adding hierarchical cluster labels to the HC algorithm and original dataframes
5 hc_df["HC_segments_L2"] = HCmodel.labels_
6 df3["HC_segments_L2"] = HCmodel.labels_
```

▼ Cluster Profiling

```
1 hc_cluster_profile = df3.groupby("HC_segments_L2").mean(numeric_only = True)

1 hc_cluster_profile["count_in_each_segment"] = (
2     df3.groupby("HC_segments_L2")["Total_Points"].count().values
3 )

1 hc_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```



	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets	count_in_each_segment
HC_segments_L2											
0	0.881517	1.139810	47.969194	1205.945498	17.580569	148.574408	249.536967	131.753555	3.293839	4.182464	422
1	16.800000	9.200000	189.000000	3033.200000	44.000000	494.340000	860.720000	1591.600000	21.800000	10.800000	5
2	8.565217	5.826087	129.391304	2238.934783	29.760870	543.273913	586.234783	861.739130	14.021739	8.739130	46
3	19.333333	13.000000	238.000000	3101.000000	37.000000	1041.300000	1221.000000	1294.666667	34.000000	12.666667	3

```
1 # Let's see the names of the players in each cluster
2 for cl in df3["HC_segments_L2"].unique():
3     print("In cluster {}, the following players are present:".format(cl))
4     print(df3[df3["HC_segments_L2"] == cl]["Player_Name"].unique())
5     print()
```



Joseph Willock 'Ralf Dahlow' 'Kelland Watts' 'Martin Dubravka'
'Matt Ritchie' 'Matthew Longstaff' 'Miguel Almiron' 'Paul Dummett'
'Ryan Fraser' 'Sean Longstaff' 'Yoshinori Muto' 'Alex McCarthy'
'Caleb Watts' 'Daniel N'Lundulu' 'Fraser Forster' 'Ibrahima Diallo'
'Jack Stephens' 'Jake Vokins' 'James Ward-Prowse' 'Jan Bednarek'
'Jannik Vestergaard' 'Kgaogelo Chauke' 'Kyle Walker-Peters'
'Michael Obafemi' 'Mohammed Salisu' 'Moussa Djenepo' 'Nathan Redmond'
'Nathan Tella' 'Oriol Romeu Vidal' 'Ryan Bertrand' 'Shane Long'
'Stuart Armstrong' 'Takumi Minamino' 'Theo Walcott' 'William Smallbone'
'Yan Valery' 'Bamidele Alli' 'Ben Davies' 'Cameron Carter-Vickers'
'Carlos Vinicius Alves Morais' 'Dane Scarlett' 'Danny Rose'
'Davinson Sanchez' 'Eric Dier' 'Erik Lamela' 'Giovani Lo Celso'
'Harry Winks' 'Hugo Lloris' 'Japhet Tanganga' 'Joe Rodon' 'Juan Foyth'
'Lucas Moura' 'Matt Doherty' 'Moussa Sissoko' 'Paulo Gazzaniga'
'Pierre-Emile Hojbjerg' 'Ryan Sessegnon' 'Serge Aurier' 'Sergio Reguilon'
'Steven Bergwijn' 'Tanguy Ndombele' 'Toby Alderweireld' 'Aaron Cresswell'
'Ademipo Odubeko' 'Albian Ajeti' 'Andriy Yarmolenko' 'Angelo Ogbonna'
'Arthur Masuaku' 'Ben Johnson' 'Craig Dawson' 'Darren Randolph'
'Declan Rice' 'Fabian Balbuena' 'Felipe Anderson Pereira Gomes'
'Frederik Alves' 'Issa Diop' 'Jamal Baptiste' 'Jordan Hugill'
'Lukasz Fabianski' 'Manuel Lanzini' 'Mark Noble' 'Roberto Jimenez Gago'
'Ryan Fredericks' 'Said Benrahma' 'Sebastian Haller' 'Vladimir Coufal'
'Adama Traore' 'Conor Coady' 'Daniel Castelo Podence' 'Fabio Silva'
'Fernando Marcal' 'Joao Santos Moutinho' 'John Ruddy'
'Jonathan Castro Otto' 'Ki-Jana Hoever' 'Leander Dendoncker' 'Max Kilman'
'Morgan Gibbs-White' 'Nelson Semedo' 'Oskar Buur' 'Owen Otasowie'
'Patrick Cutrone' 'Raul Jimenez' 'Rayan Ait Nouri' 'Romain Saiss'
'Ruben Neves' 'Ruben Vinagre' 'Rui Pedro Patricio' 'Vitor Ferreira'
'Willian Jose' 'Willy Boly']

In cluster 2, the following players are present:

['Alexandre Lacazette' 'Bukayo Saka' 'Nicolas Pepe'
'Pierre-Emerick Aubameyang' 'Anwar El Ghazi' 'Bertrand Traore'
'Jack Grealish' 'Danny Welbeck' 'Leandro Trossard' 'Neal Maupay'
'Pascal Gross' 'Chris Wood' 'Mason Mount' 'Timo Werner'
'Christian Benteke' 'Wilfried Zaha' 'Gylfi Sigurdsson'
'Richarlison de Andrade' 'Jack Harrison' 'Raphael Dias Belloli'
'Rodrigo Moreno' 'Stuart Dallas' 'Harvey Barnes' 'James Maddison'
'Kelechi Iheanacho' 'Roberto Firmino' 'Sadio Mane'
'Gabriel Fernando de Jesus' 'Ilkay Gundogan' 'Kevin De Bruyne'
'Phil Foden' 'Raheem Sterling' 'Riyad Mahrez' 'Edinson Cavani'
'Marcus Rashford' 'Mason Greenwood' 'Callum Wilson' 'Che Adams'
'Danny Ings' 'Gareth Bale' 'Jarrod Bowen' 'Jesse Lingard'
'Michail Antonio' 'Pablo Fornals' 'Tomas Soucek' 'Pedro Lomba Neto']

```
1 df3.groupby(["HC_segments_L2", "Position"])[ 'Player_Name'].count()
```



		Player_Name
HC_segments_L2	Position	
0	Defender	171
	Forward	43
	Goalkeeper	45
	Midfielder	163
1	Forward	4
	Midfielder	1
2	Defender	1
	Forward	16
	Midfielder	29
3	Forward	1
	Midfielder	2

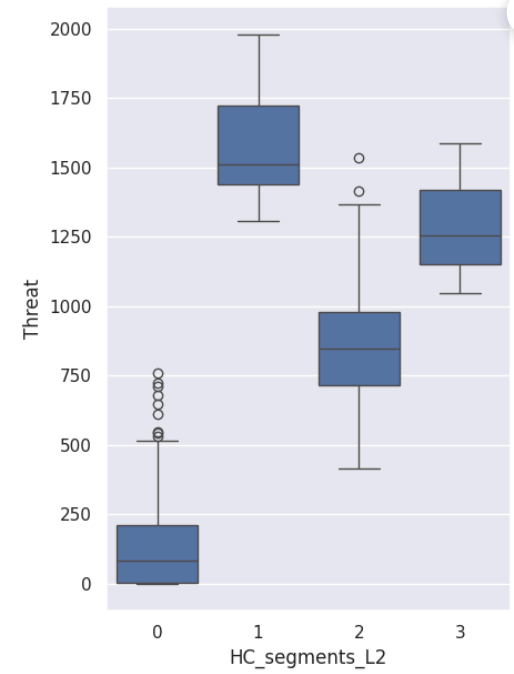
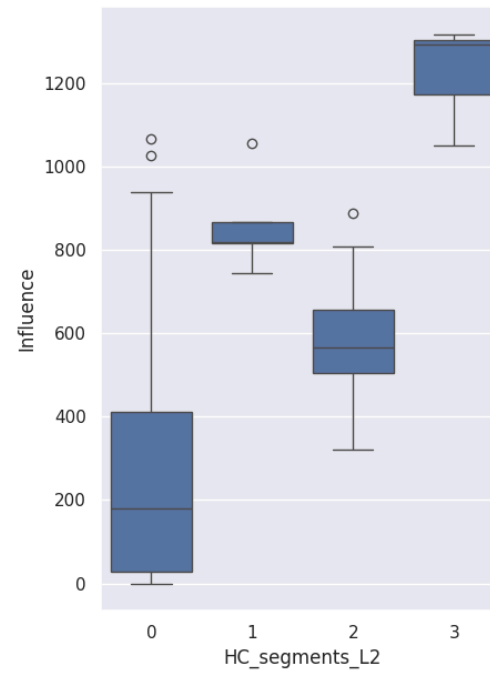
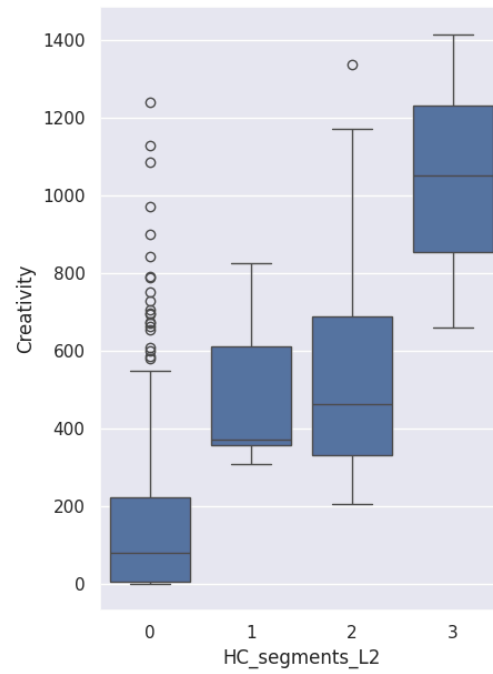
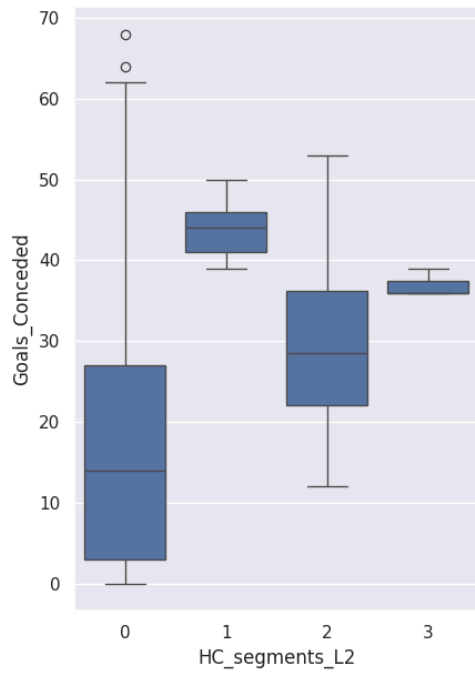
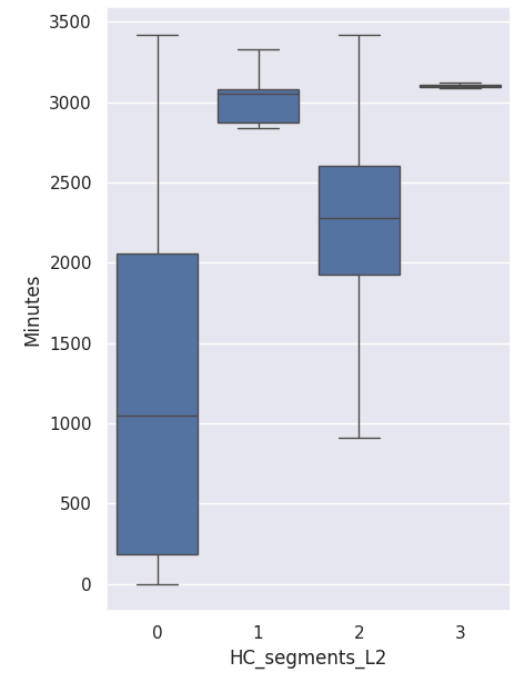
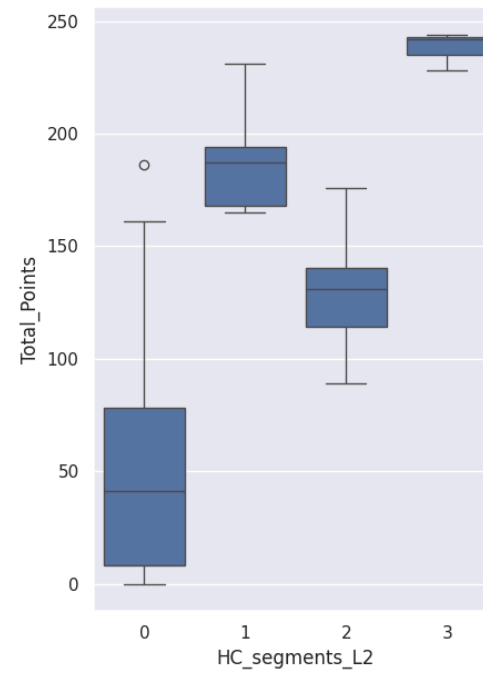
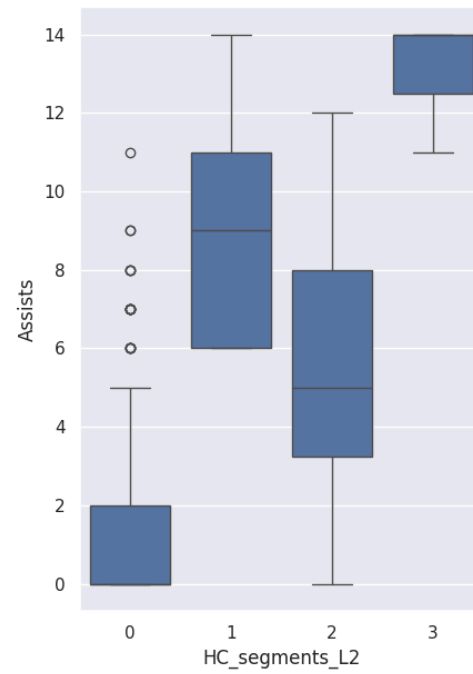
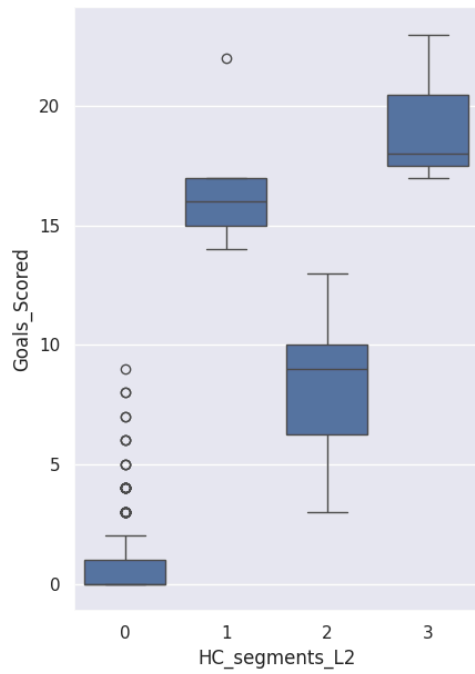
dtype: int64

- Cluster 0 has no goalkeepers, so it is likely to have players with more offensive duties in the team.
- Cluster 2 has no forwards, so it is likely to have players with more defensive duties in the team.

The clusters look better now. Let's check the cluster profiles.

```
1 fig, axes = plt.subplots(3, 4, figsize = (20, 20))
2 counter = 0
3
4 for ii in range(3):
5     for jj in range(4):
6         if counter < 10:
7             sns.boxplot(
8                 ax = axes[ii][jj],
9                 data = df3,
10                y = df3.columns[3 + counter],
11                x = "HC_segments_L2",
12            )
13            counter = counter + 1
14
15 fig.tight_layout(pad = 3.0)
```

17



40

1.0

1.0