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

In [1]:
```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme(style='darkgrid')

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# To scale the data using z-score
from sklearn.preprocessing import StandardScaler

# To compute distances
from scipy.spatial.distance import cdist, pdist

# To perform K-Means clustering and compute silhouette scores
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# To import K-Medoids
from sklearn_extra.cluster import KMedoids

# To import DBSCAN and Gaussian Mixture
from sklearn.cluster import DBSCAN
from sklearn.mixture import GaussianMixture

# To perform hierarchical clustering, compute cophenetic correlation, and create dendrograms
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

# To use in defined functions
import matplotlib.patheffects as path_effects
import matplotlib as mpl
from scipy.stats import pearsonr

# Importing PCA
from sklearn.decomposition import PCA
```

In [2]:
```
data = pd.read_csv("data/fpl_data.csv")
```

In [3]:
```
data.shape
```

Out[3]:
```
(476, 13)
```

Observations:

- The dataset has 476 rows and 13 columns.

In [4]:
```
# Viewing 10 random rows of the data
data.sample(10, random_state = 1)
```

Out[4]:

	Player_Name	Club	Position	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence
441	Mark Noble	West Ham United	Midfielder	0	0	27	701	15	88.6	80.4
363	Sean Longstaff	Newcastle United	Midfielder	0	1	41	1405	26	182.8	179.2
31	Anwar El Ghazi	Aston Villa	Midfielder	10	0	111	1604	22	426.1	500.4
132	Olivier Giroud	Chelsea	Forward	4	0	47	740	5	112.0	161.4
90	Chris Wood	Burnley	Forward	12	3	138	2741	43	323.2	595.8
249	Vontae Daley-Campbell	Leicester City	Defender	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
445	Ryan Fredericks	West Ham United	Defender	1	1	28	564	9	166.8	155.2
117	Christian Pulisic	Chelsea	Midfielder	4	3	82	1731	21	378.8	361.4
415	Ryan Sessegnon	Tottenham Hotspurs	Defender	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.

In [5]:
```
# Copying the data to another variable to avoid any changes to original data
df = data.copy()
```

In [6]:
```
# Checking datatypes and number of non-null values for each column
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.

In [7]:
```
# Checking for duplicate values
df.duplicated().sum()
```

Out[7]:
```
0
```

Observations:

- There are no duplicate values in the data.

In [8]:
```
# Checking for missing values
df.isnull().sum()
```

Out[8]:
```
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
```

Observations:

- There are no missing values in the data.

Exploratory Data Analysis

Let's check the statistical summary of the data.

In [9]:
```
df.describe(include = 'all').T
```

Out[9]:

	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.

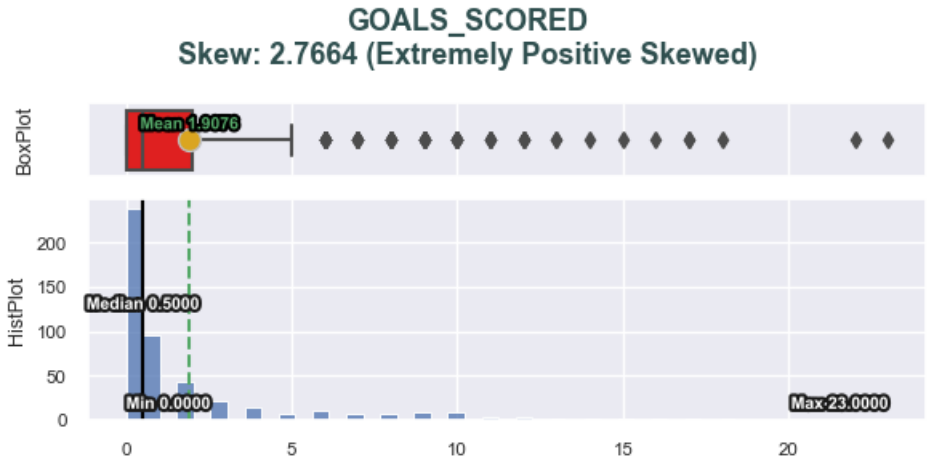
Univariate Analysis

In [10]:
```
# Global variable to use in plots.
suptitle_param = dict(color='darkslategray', weight='bold', fontsize='x-large')

def get_feature_hist_and_boxplot(feature, bins="auto", figsize=(6, 3)):
    """ Boxplot and histogram combined
    feature: pandas.series
    bins: number of bins (default "auto")
    figsize: size of fig (default (6, 3))
    """
    mean = feature.mean()
    median = np.median(feature)
    min_v = feature.min()
    max_v = feature.max()

    sns.set(font_scale=.75)
    f, (ax_box, ax_hist) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid
        sharex=True,  # The X-axis will be shared among all the subplots
        gridspec_kw={"height_ratios": (.25, .75)},
        figsize=figsize
    )

    # Creating the subplots
    # Boxplot will be created and the mean value of the column will be indicated using some symbol
    sns.boxplot(x=feature, ax=ax_box, color='red',
                showmeans=True,
                meanprops={"marker":"o",
                           "markerfacecolor":"goldenrod",
                           "markeredgecolor":"silver",
                           "markersize":"10"})

    text = ax_box.annotate("Mean {:,.4f}".format(mean), fontsize='small',
                           xy=(mean, -0.15), color='g', weight='bold', ha='center')
    text.set_path_effects([path_effects.Stroke(linewidth=3,
                                               foreground='black'), path_effects.Normal()])
    ax_box.set_ylabel('BoxPlot\n')
    ax_box.set_xlabel('')

    # For histogram
    sns.histplot(x=feature, kde=False, bins=bins, ax=ax_hist)
    ax_hist.axvline(mean, color='g', linestyle='--')      # Add mean to the histogram
    ax_hist.axvline(median, color='black', linestyle='-') # Add median to the histogram

    min_max_pos = 0.05 * ax_hist.get_ylim()[1]
    text = ax_hist.annotate("Median {:,.4f}".format(median), fontsize='small',
                            xy=(median, ax_hist.get_ylim()[1]/2),
                            color='w', weight='bold', ha='center')
    text.set_path_effects([path_effects.Stroke(linewidth=3, foreground='k'), path_effects.Normal()])
    text = ax_hist.annotate("Min {:,.4f}".format(min_v), fontsize='small',
                            xy=(min_v, min_max_pos),
                            color='w', weight='bold', ha='left')
    text.set_path_effects([path_effects.Stroke(linewidth=3, foreground='k'), path_effects.Normal()])
    text = ax_hist.annotate("Max {:,.4f}".format(max_v), fontsize='small',
                            xy=(max_v, min_max_pos),
                            color='w', weight='bold', ha='right')
    text.set_path_effects([path_effects.Stroke(linewidth=3, foreground='k'), path_effects.Normal()])

    ax_hist.set_ylabel('HistPlot')
    ax_hist.set_xlabel('')

    # Calculating the skewness
    # If the skewness is between -0.5 & 0.5, the data are nearly symmetrical.
    # If the skewness is between -1 & -0.5 (negative skewed) or between 0.5 & 1(positive skewed), the
    # If the skewness is lower than -1 (negative skewed) or greater than 1 (positive skewed), the dat
    skewness = feature.skew()
    if skewness<-1: skewness_str = 'Extremely Negative Skewed'
    elif skewness<-0.5: skewness_str = 'Negative Skewed'
    elif skewness==0: skewness_str = 'Simetrical Distributed'
    elif skewness<=0.5: skewness_str = 'Nearly Simmetrical'
    elif skewness<=1: skewness_str = 'Positive Skewed'
    elif skewness>1: skewness_str = 'Extremely Positive Skewed'
```

```
    f.suptitle(f'{feature.name.upper()}\nSkew: {skewness:0.4f} ({skewness_str})', **suptitle_param)
    plt.subplots_adjust(hspace=1, top=0.9)
    plt.tight_layout()

    plt.show()
    plt.style.use('default')
```

Goals_Scored

In [11]:
```
get_feature_hist_and_boxplot(df.Goals_Scored)
```

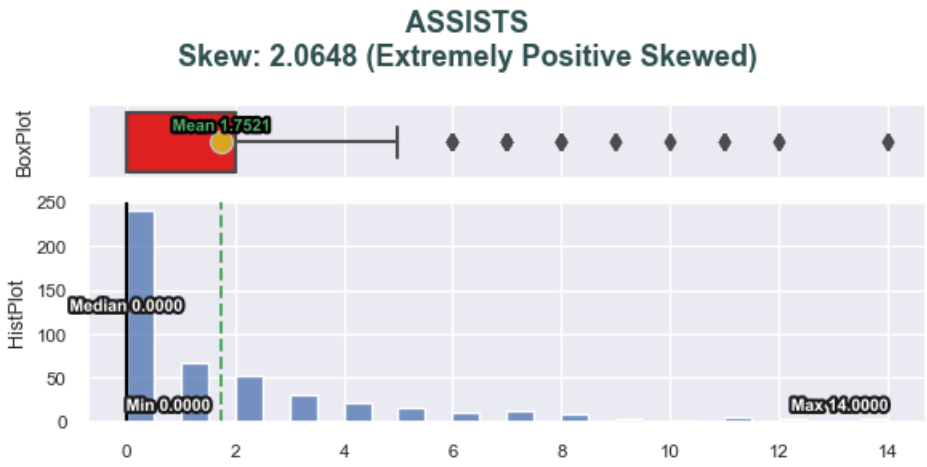


Observations:

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

Assists

In [12]:
```
get_feature_hist_and_boxplot(df.Assists)
```

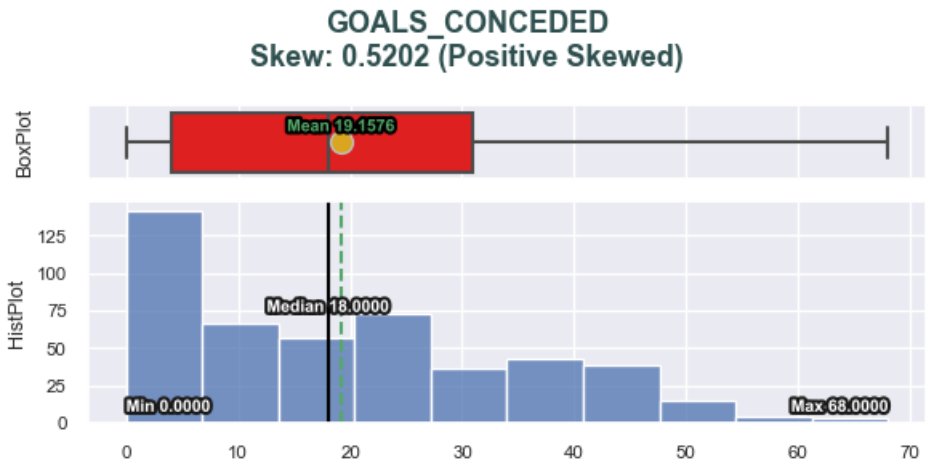


Observations:

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

Goals_Conceded

In [13]:
```
get_feature_hist_and_boxplot(df.Goals_Conceded)
```

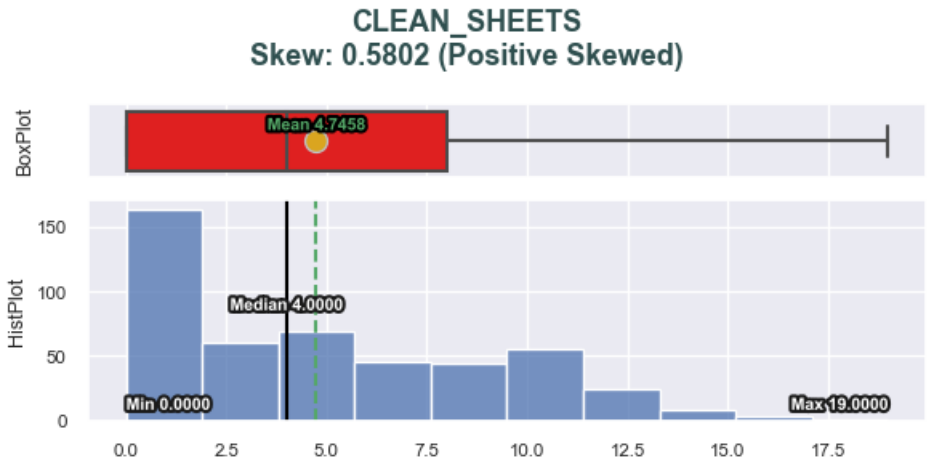


Observations:

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

`Clean_Sheets`

In [14]:
```
get_feature_hist_and_boxplot(df.Clean_Sheets)
```

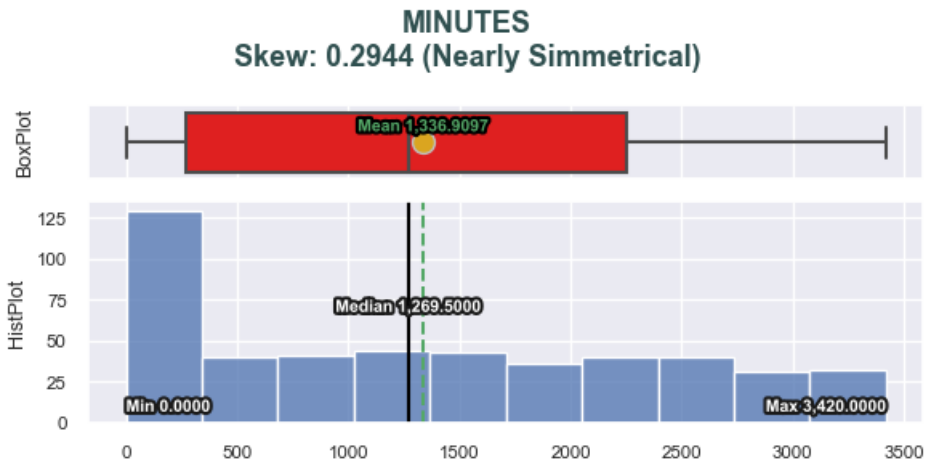


Observations:

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

`Minutes`

In [15]:
```
get_feature_hist_and_boxplot(df.Minutes)
```

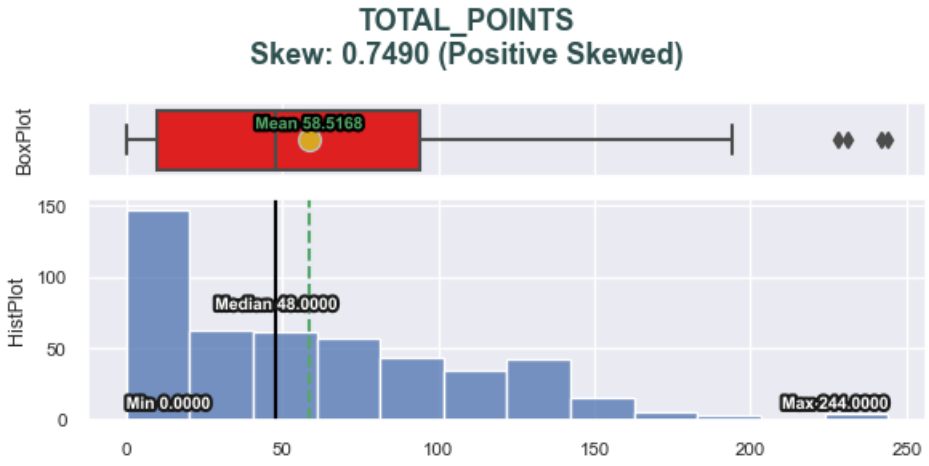


Observations:

- 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`

In [16]:
```
get_feature_hist_and_boxplot(df.Total_Points)
```

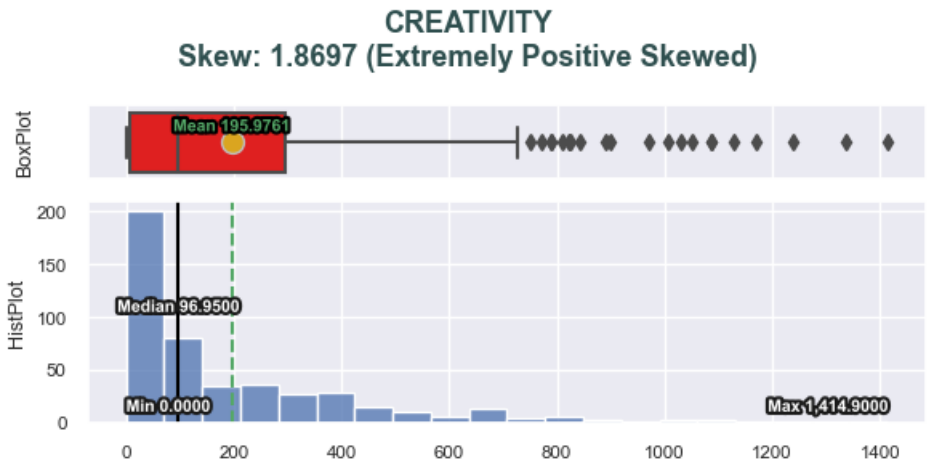


Observations:

- 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`

In [17]:
```
get_feature_hist_and_boxplot(df.Creativity)
```

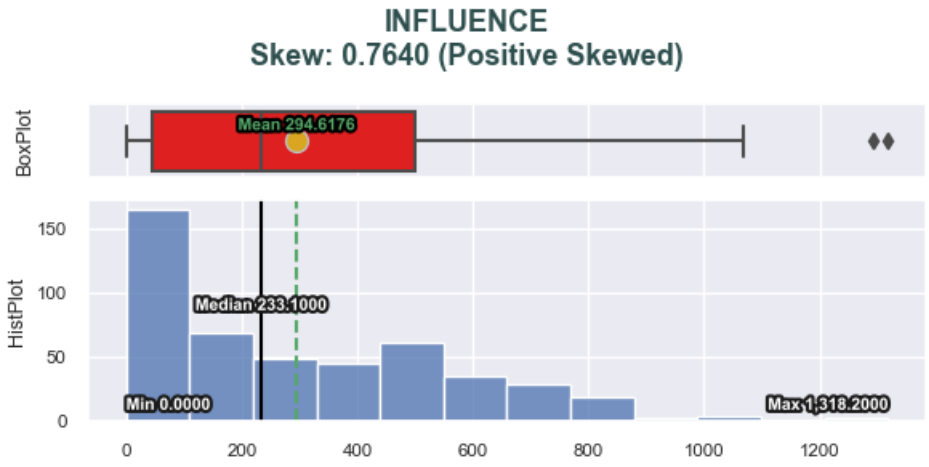


Observations:

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

`Influence`

In [18]:
```
get_feature_hist_and_boxplot(df.Influence)
```

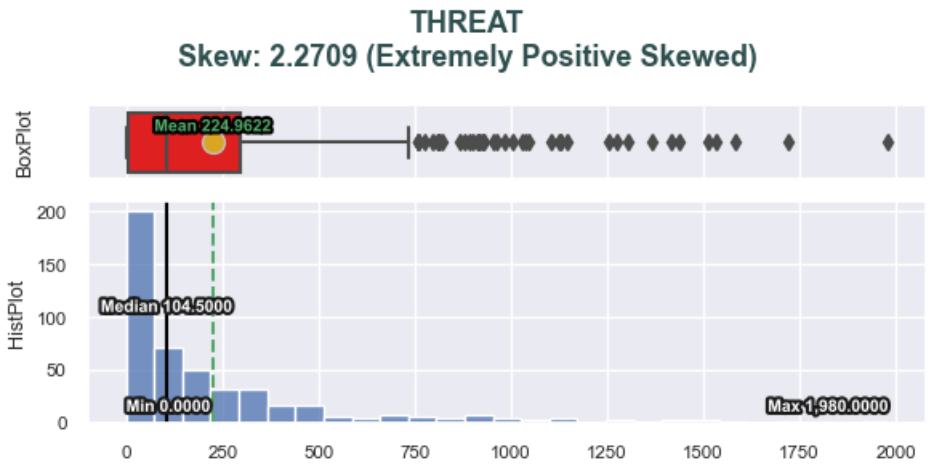


Observations:

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

`Threat`

In [19]:
```
get_feature_hist_and_boxplot(df.Threat)
```

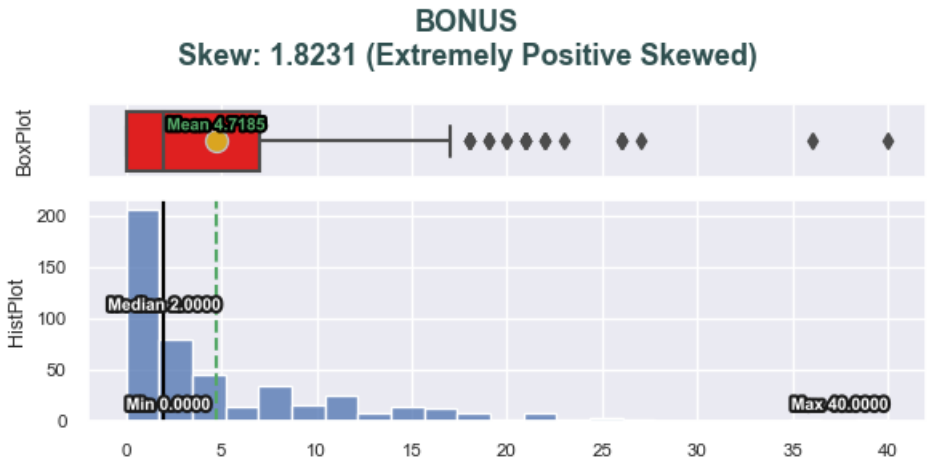


Observations:

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

Bonus

In [20]:
```
get_feature_hist_and_boxplot(df.Bonus)
```



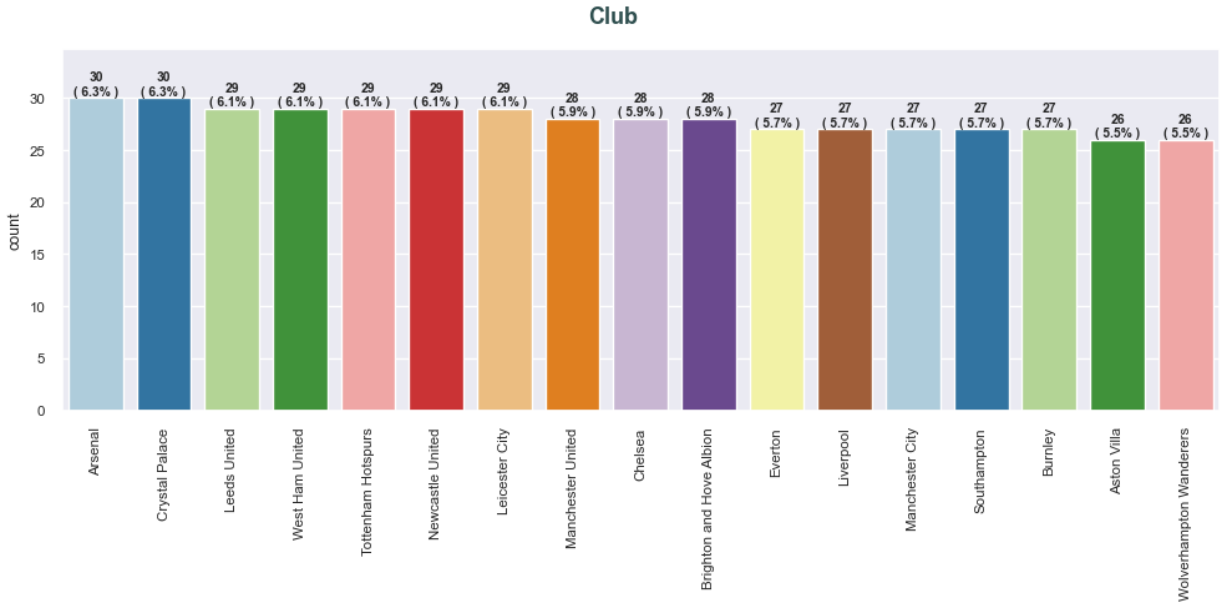
Observations:

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

In [21]:
```
# Function to create labeled barplots
def get_feature_countplot(feature, rotation=0, top=None, title=None, figsize=(10, 5)):
    """Countplot of a categorical variable
    feature: pandas.series
    rotation: rotation of xticks (default 0)
    top: max rows to return. If none is provided all rows are returned. (Default: None)
    title: title of the plot. If none value is provided, feature names are displayed.
    (Default: None)
    figsize: size of fig (default (10, 5))
    """
    title = title if title else feature.name

    sns.set(font_scale=.75)
    total = len(feature) # Length of the column
    plt.figure(figsize = figsize)

    # Convert the column to a categorical data type
    feature = feature.astype('category')
    origin = feature.copy()

    labels = feature.value_counts().index
    if top:
        labels = labels[:top]
        feature = feature.loc[feature.isin(labels)]

    ax = sns.countplot(x=feature, palette='Paired', order=labels)
    ax.set_xlabel('')

    # custom label calculates percent and add an empty string so 0 value bars don't have a number
    for container in ax.containers:
        labels = [f'{h:.0f}\n( {h/origin.count()*100:0.1f}% )'
                  if (h := v.get_height()) > 0 else '' for v in container]
        ax.bar_label(container, labels=labels, label_type='edge',
                     fontsize='small', weight='bold') # color='white', label_type='center'
    ylim = plt.ylim()
    plt.ylim(ylim[0], ylim[1]*1.1)
    if title is None: title=feature.name
    plt.suptitle(title, **suptitle_param)
    plt.xticks(rotation=rotation)
    plt.tight_layout()
    # plt.autoscale(enable=True, axis='y', tight=False)

    plt.show()
    plt.style.use('default')
```

Club

In [22]:
```
get_feature_countplot(df.Club, rotation=90)
```

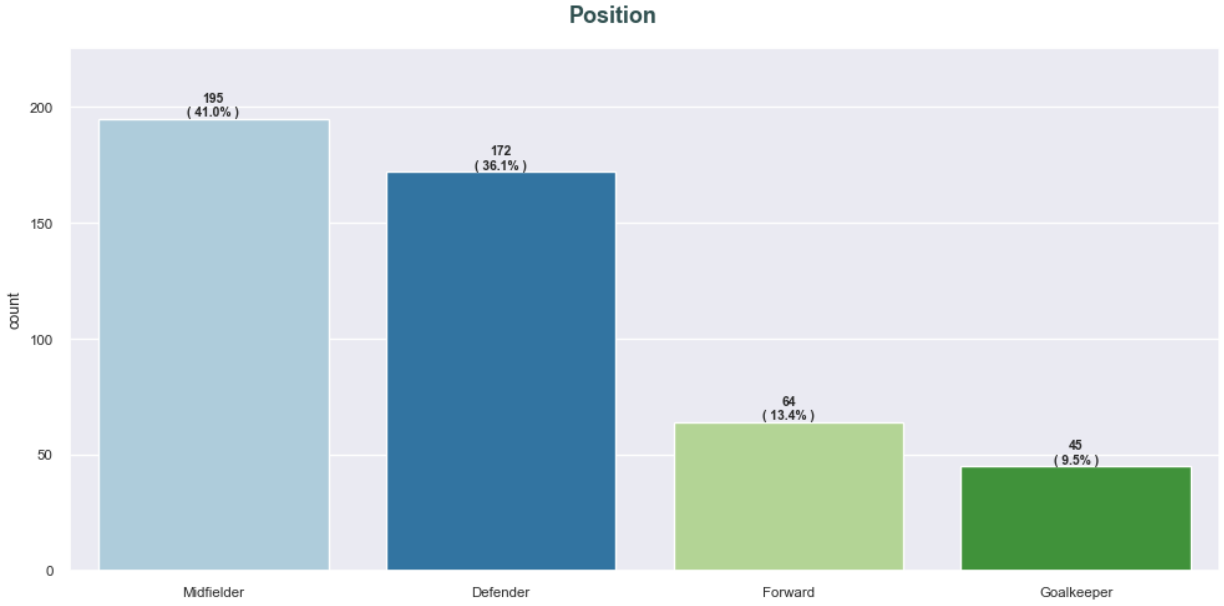


Observations:

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

`Position`

In [23]:
```
get_feature_countplot(df.Position)
```



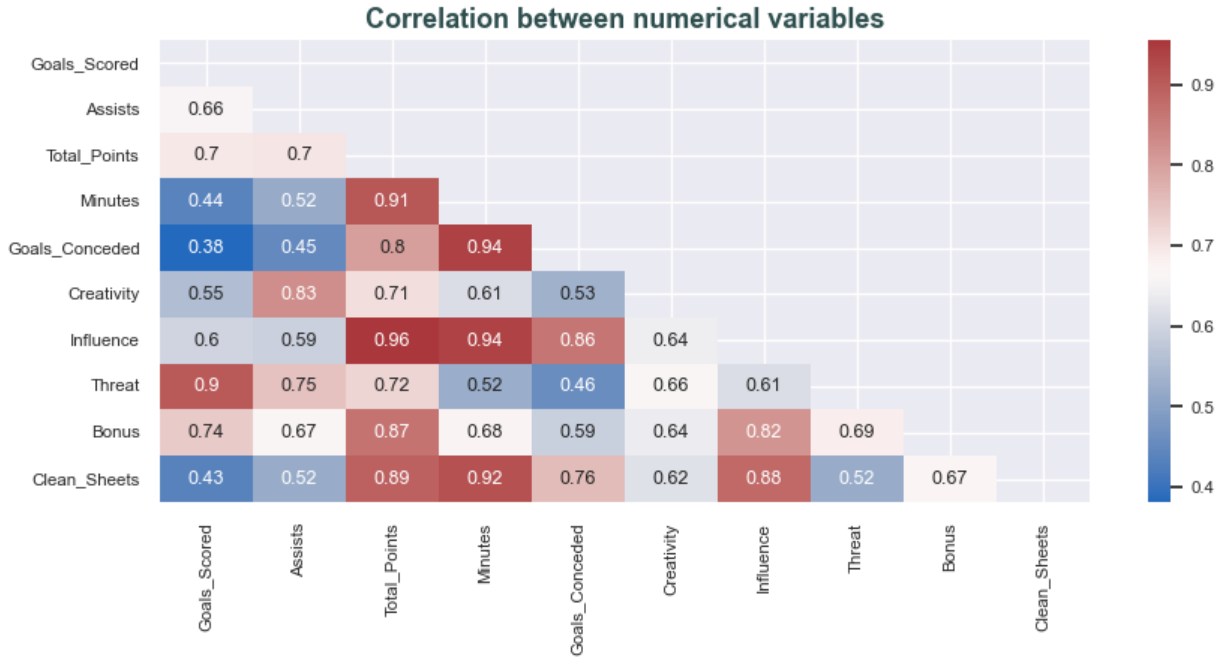
Observations:

- 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

In [24]:
```
def get_df_correlation(df):
    """Plot a heatmap of the correlation between numerical variables in df.
    df: dataframe.
    """
    corr = df.corr(numeric_only=True)

    mask = np.zeros_like(corr, dtype=bool)
    mask[np.triu_indices_from(mask)] = True

    sns.set(font_scale=0.75)
    plt.figure(figsize = (10, 4))
    sns.heatmap(corr, annot=True, mask=mask, cmap='vlag')
    plt.title('Correlation between numerical variables', **suptitle_param)
    plt.show()
    plt.style.use('default')
```

In [25]:
```
get_df_correlation(df)
```



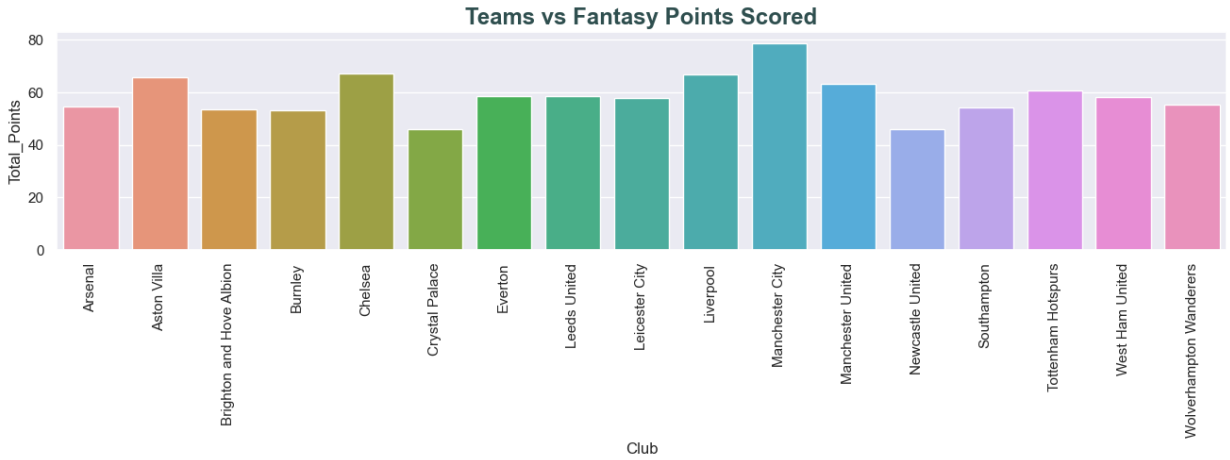
Observations:

- 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.

In [26]:
```
def get_2features_barplot(x, y, title, df=df, rotation=0):
    """Barplot of 2 variables relation.
    x: name of the feature.
    y: name of the feature.
    title: title of the graph.
    df: dataframe.
    rotation: rotation of x-ticks, default:0.
    """
    plt.figure(figsize = (16,3))
    sns.set()
    sns.barplot(data=df, x=x, y=y, errorbar=('ci', False))
    plt.xticks(rotation=rotation)
    plt.title(title, **suptitle_param)
    plt.show()
    plt.style.use('default')
```

In [27]:
```
get_2features_barplot('Club', 'Total_Points', 'Teams vs Fantasy Points Scored', rotation=90)
```

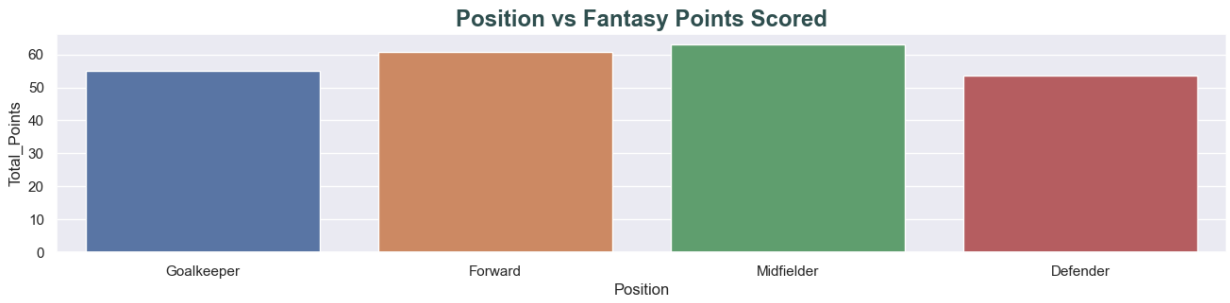


Observations:

- 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.

In [28]:
```
get_2features_barplot('Position', 'Total_Points', 'Position vs Fantasy Points Scored')
```

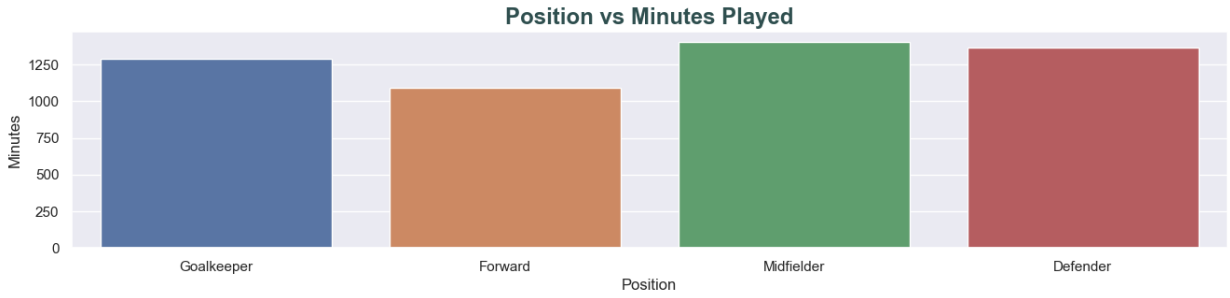


Observations:

- 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.

In [29]:
```
get_2features_barplot('Position', 'Minutes', 'Position vs Minutes Played')
```

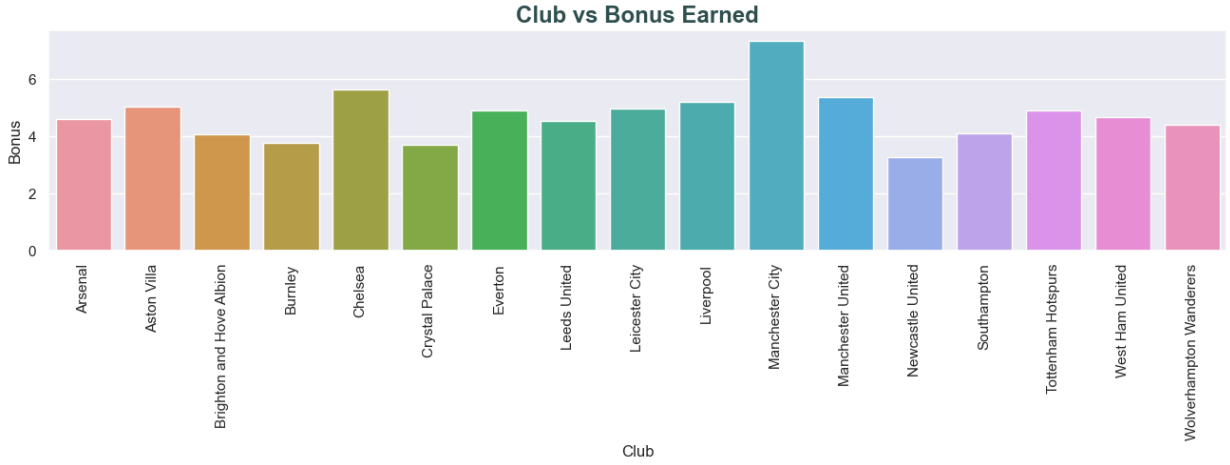


Observations:

- 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.

In [30]:
```
get_2features_barplot('Club', 'Bonus', 'Club vs Bonus Earned', rotation=90)
```



Observations:

- 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.

In [31]:
```
filter = df.groupby(['Position']).Total_Points.idxmax()
df.loc[df.index.isin(filter)][['Player_Name', 'Club', 'Position', 'Total_Points']]
```

Out[31]:

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

Observations:

- 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.

In [32]:
```
(df.sort_values(by=['Position', 'Total_Points'], ascending=[True, False])
   .groupby('Position')[['Player_Name', 'Position', 'Club', 'Total_Points']]
   .head(10))
```

Out[32]:

	Player_Name	Position	Club	Total_Points
223	Stuart Dallas	Defender	Leeds United	171
257	Andrew Robertson	Defender	Liverpool	161
278	Trent Alexander-Arnold	Defender	Liverpool	160
421	Aaron Cresswell	Defender	West Ham United	153
308	Aaron Wan-Bissaka	Defender	Manchester United	144
303	Ruben Dias	Defender	Manchester City	142
113	Benjamin Chilwell	Defender	Chelsea	139
48	Matt Targett	Defender	Aston Villa	138
290	Joao Cancelo	Defender	Manchester City	138
73	Lewis Dunk	Defender	Brighton and Hove Albion	130
403	Harry Kane	Forward	Tottenham Hotspurs	242
219	Patrick Bamford	Forward	Leeds United	194
238	Jamie Vardy	Forward	Leicester City	187
51	Ollie Watkins	Forward	Aston Villa	168
177	Dominic Calvert-Lewin	Forward	Everton	165
275	Roberto Firmino	Forward	Liverpool	141
90	Chris Wood	Forward	Burnley	138
367	Che Adams	Forward	Southampton	137
338	Callum Wilson	Forward	Newcastle United	134
369	Danny Ings	Forward	Southampton	131
36	Emiliano Martinez	Goalkeeper	Aston Villa	186
284	Ederson Moares	Goalkeeper	Manchester City	160
203	Illan Meslier	Goalkeeper	Leeds United	154
406	Hugo Lloris	Goalkeeper	Tottenham Hotspurs	149
107	Nick Pope	Goalkeeper	Burnley	144
118	Edouard Mendy	Goalkeeper	Chelsea	140
256	Alisson Becker	Goalkeeper	Liverpool	140
439	Lukasz Fabianski	Goalkeeper	West Ham United	133
472	Rui Pedro Patricio	Goalkeeper	Wolverhampton Wanderers	132
2	Bernd Leno	Goalkeeper	Arsenal	131
315	Bruno Fernandes	Midfielder	Manchester United	244
269	Mohamed Salah	Midfielder	Liverpool	231
405	Heung-Min Son	Midfielder	Tottenham Hotspurs	228
276	Sadio Mane	Midfielder	Liverpool	176
326	Marcus Rashford	Midfielder	Manchester United	174
204	Jack Harrison	Midfielder	Leeds United	160
289	Ilkay Gundogan	Midfielder	Manchester City	157
374	James Ward-Prowse	Midfielder	Southampton	156
300	Raheem Sterling	Midfielder	Manchester City	154
129	Mason Mount	Midfielder	Chelsea	147

Observations:

- 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.

In [33]:
```
df.select_dtypes(include=np.number)
```

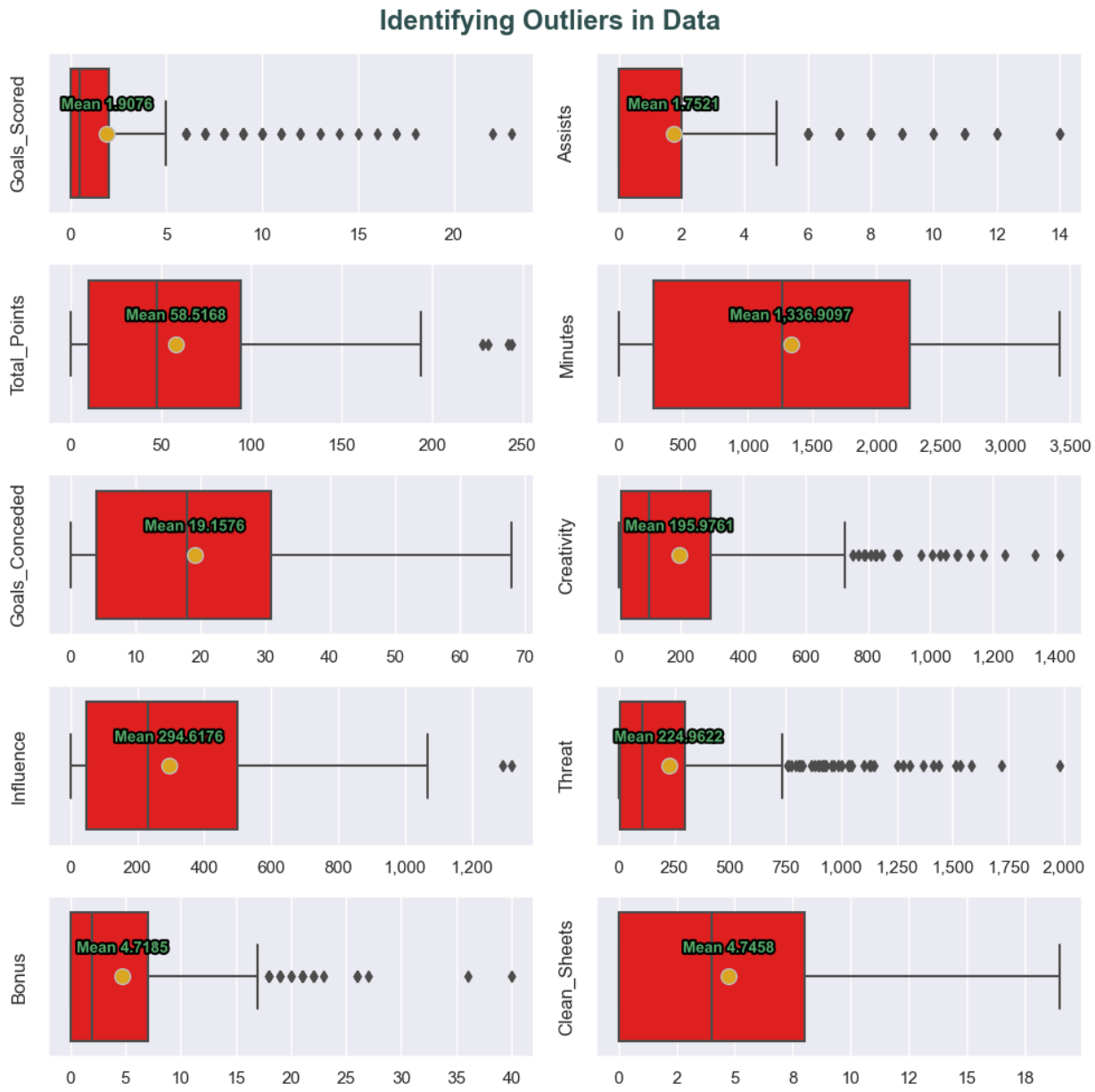
Out[33]:

	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets
0	0	0	1	15	0	0.0	16.6	0	0	0
1	13	3	129	1916	21	307.4	602.4	797	21	7
2	0	0	131	3131	37	0.0	702.2	2	11	11
3	5	5	114	2554	31	650.6	493.0	984	8	8
4	0	3	36	751	10	169.4	171.8	77	3	2
...	...	...	...	...	...	...	...	...	...	...
471	0	0	1	172	7	25.3	31.2	0	0	0
472	0	0	132	3330	51	0.0	763.6	0	12	10
473	0	1	26	509	8	179.0	80.0	72	0	0
474	1	1	40	1106	13	165.9	89.6	256	3	5
475	1	1	68	1879	29	46.1	414.0	182	4	6

476 rows × 10 columns

In [34]:
```
def get_df_boxplot(df, title, cols, rows, figsize):
    """Visualize the five-number summary of all numerical variables in dataset: minimum,
    first quartile (Q1), median (Q2), third quartile (Q3), and maximum.
    df: dataframe.
    title: title of the plot

    """
    df_t = df.select_dtypes(include=np.number)

    sns.set()
    fig, axes = plt.subplots(nrows=rows, ncols=cols, sharex=False, sharey=False, figsize=figsize)
    axes=axes.ravel()

    # Creating the subplots
    # Boxplot will be created and the mean value of the column will be indicated using some symbol
    for i, col in enumerate(df_t.columns):
        feature = df_t[col]
        ax = axes[i]

        mean = feature.mean()
        median = np.median(feature)
        min_v = feature.min()
        max_v = feature.max()

        sns.boxplot(x=feature, ax=ax, color='red', showmeans=True,
                    meanprops={"marker":"o", "markerfacecolor":"goldenrod", "markeredgecolor":"silver

        text = ax.annotate("Mean {:,.4f}".format(mean), fontsize='small',
                           xy=(mean, -0.15), color='g', weight='bold', ha='center')
        text.set_path_effects([path_effects.Stroke(linewidth=3,
                                                   foreground='black'), path_effects.Normal()])
        ax.set_ylabel(col)
        ax.set_xlabel('')
        ax.xaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))
    if i<=len(axes):
        for j in range(i+1,len(axes)):
            axes[j].axis('off')
    plt.subplots_adjust(hspace=0.5)
    fig.suptitle(title, **suptitle_param)
    plt.tight_layout()
    plt.show()
    plt.style.use('default')
```

In [35]:
```
get_df_boxplot(df, 'Identifying Outliers in Data', cols=2, rows=5, figsize=(10, 10))
```



Observations:

- 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.

In [36]:
```
df.iloc[:, 3:].head()
```

Out[36]:

	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheets
0	0	0	1	15	0	0.0	16.6	0	0	0
1	13	3	129	1916	21	307.4	602.4	797	21	7
2	0	0	131	3131	37	0.0	702.2	2	11	11
3	5	5	114	2554	31	650.6	493.0	984	8	8
4	0	3	36	751	10	169.4	171.8	77	3	2

In [37]:
```
# Scaling the data before clustering
scaler = StandardScaler()
subset = df.iloc[:, 3:].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [38]:
```
# Creating a dataframe of the scaled data
subset_scaled_df = pd.DataFrame(subset_scaled, columns = subset.columns)
subset_scaled_df.head()
```

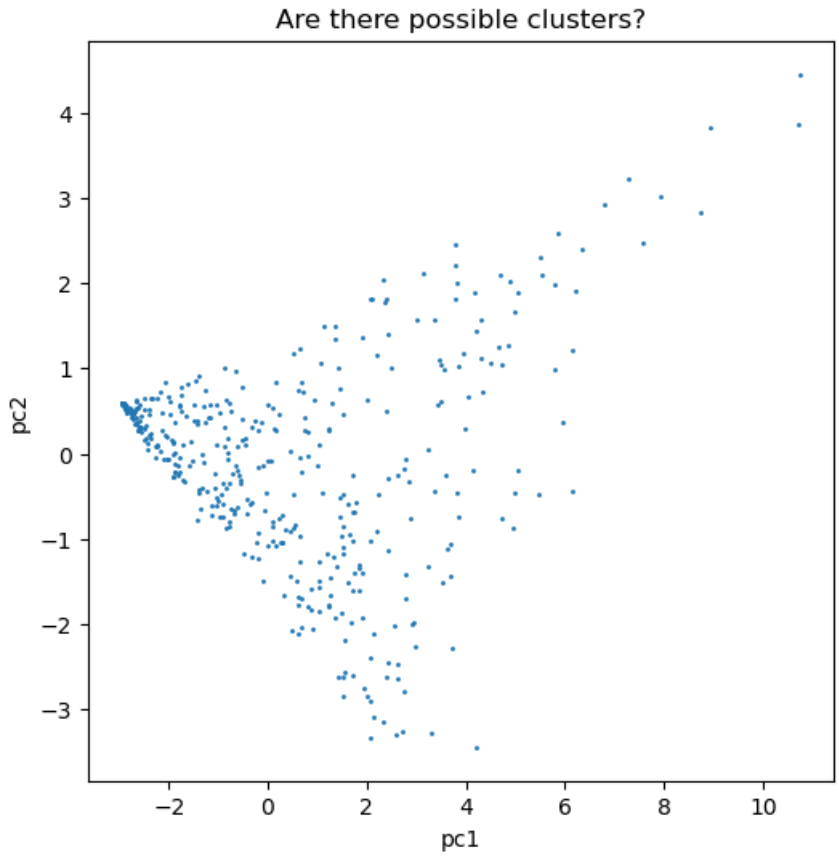
Out[38]:

	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	Bonus	Clean_Sheet
0	-0.552608	-0.647555	-1.122506	-1.232383	-1.202653	-0.780115	-1.039325	-0.707638	-0.755435	-1.08112
1	3.213401	0.461208	1.375560	0.539871	0.115663	0.443542	1.150596	1.799393	2.606688	0.51352
2	-0.552608	-0.647555	1.414592	1.672585	1.120094	-0.780115	1.523682	-0.701346	1.005677	1.42474
3	0.895857	1.200384	1.082818	1.134662	0.743432	1.809706	0.741621	2.387618	0.525374	0.74132
4	-0.552608	0.461208	-0.439441	-0.546229	-0.574884	-0.105790	-0.459134	-0.465428	-0.275132	-0.62551

Taking a look on possible clusters using PCA

In [39]:
```
# Applying PCA
pca = PCA(n_components = 2)
pca2_df = pca.fit_transform(subset_scaled_df)
pca2_df = pd.DataFrame(pca2_df, columns = ['pc1', 'pc2'])

# Printing the percentage of variance explained by each of the selected components (n_components = 2)
print(f'Explained variance: {pca.explained_variance_ratio_.sum()}')

# Now, we finally need to plot to try to find correlations visually
plt.figure(figsize=(6, 6))
plt.scatter(pca2_df.pc1, pca2_df.pc2, s=1)
plt.xlabel('pc1')
plt.ylabel('pc2')
plt.title('Are there possible clusters?')
plt.show()
```

```
Explained variance: 0.8580828019339619
```



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.

In [40]:
```
# Defining the number of principal components to generate
n = subset.shape[1]                                          # Storing the number of variables

pca = PCA(n_components = n, random_state = 1)                # Storing PCA function with n com
data_pca = pd.DataFrame(pca.fit_transform(subset_scaled_df ))  # Applying PCA on scaled data

# The percentage of variance explained by each principal component is stored
exp_var = (pca.explained_variance_ratio_)
exp_var, exp_var.cumsum()
```

Out[40]:
```
(array([0.72148934, 0.13659346, 0.05880028, 0.03138379, 0.02274545,
        0.01522727, 0.00700781, 0.00393725, 0.00194529, 0.00087005]),
 array([0.72148934, 0.8580828 , 0.91688309, 0.94826687, 0.97101232,
        0.9862396 , 0.99324741, 0.99718466, 0.99912995, 1.        ]))
```

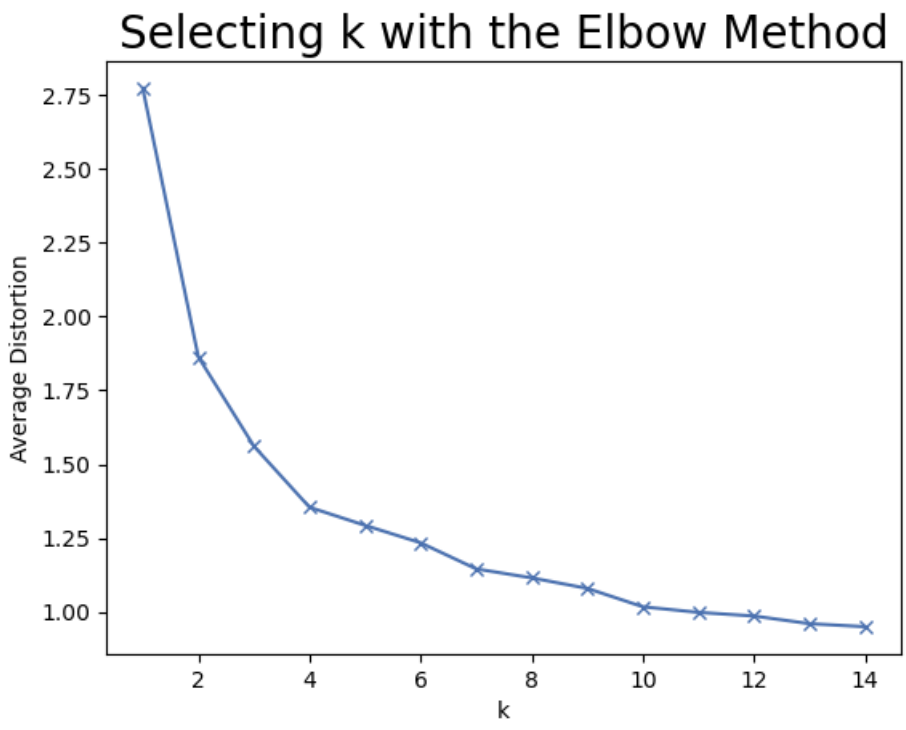
K-Means Clustering

- 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.

In [41]:
```
k_means_df = data_pca.copy()
```

In [42]:
```
clusters = range(1, 15)
meanDistortions = []

for k in clusters:
    model = KMeans(n_clusters = k, random_state = 1, n_init='auto')
    model.fit(data_pca)
    prediction = model.predict(k_means_df)
    distortion = (
        sum(np.min(cdist(k_means_df, model.cluster_centers_, "euclidean"), axis = 1))
        / k_means_df.shape[0]
    )
    meanDistortions.append(distortion)
    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)

plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average Distortion")
plt.title("Selecting k with the Elbow Method", fontsize = 20)
plt.show()
```

```
Number of Clusters: 1   Average Distortion: 2.7730371100978024
Number of Clusters: 2   Average Distortion: 1.8635736785898263
Number of Clusters: 3   Average Distortion: 1.561277403810161
Number of Clusters: 4   Average Distortion: 1.3545171820838149
Number of Clusters: 5   Average Distortion: 1.2931541699741684
Number of Clusters: 6   Average Distortion: 1.2341231453420078
Number of Clusters: 7   Average Distortion: 1.1457934035634147
Number of Clusters: 8   Average Distortion: 1.1153076568890792
Number of Clusters: 9   Average Distortion: 1.0797310475776056
Number of Clusters: 10  Average Distortion: 1.017436992641064
Number of Clusters: 11  Average Distortion: 0.9986112688354993
Number of Clusters: 12  Average Distortion: 0.9862831494790063
Number of Clusters: 13  Average Distortion: 0.9602766985773127
Number of Clusters: 14  Average Distortion: 0.95014531679089
```



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.

In [43]:
```
kmeans = KMeans(n_clusters = 4, random_state = 1, n_init='auto')
kmeans.fit(k_means_df)
```

Out[43]:
```
KMeans(n_clusters=4, n_init='auto', random_state=1)
```

In [44]:
```
# Creating a copy of the original data
df1 = df.copy()

# Adding K-Means cluster labels to the K-Means and original dataframes
k_means_df["KM_segments"] = kmeans.labels_
df1["KM_segments"] = kmeans.labels_
```

Cluster Profiles

In [45]:
```
km_cluster_profile = df1.groupby("KM_segments").mean(numeric_only=True)
km_cluster_profile
```

Out[45]:

KM_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	E
0	1.371134	1.783505	103.000000	2674.288660	37.649485	256.951546	579.144330	198.288660	7.4
1	0.148936	0.202128	9.824468	238.750000	3.930851	28.171809	43.164894	30.244681	0.4
2	8.919355	6.709677	141.725806	2458.306452	33.451613	625.253226	661.458065	860.677419	16.3
3	1.503876	1.604651	56.038760	1392.736434	20.573643	188.358915	270.818605	223.255814	3.3

In [46]:
```
# Getting how many players are in eqch segment
df1.groupby("KM_segments")["Total_Points"].count()
```

Out[46]:
```
KM_segments
0     97
1    188
2     62
3    129
Name: Total_Points, dtype: int64
```

In [47]:
```
# Creating the "count_in_each_segment" feature in K-Means cluster profile
km_cluster_profile["count_in_each_segment"] = df1.groupby("KM_segments")["Total_Points"].count().valu
```

In [48]:
```
km_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[48]:

KM_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	E
0	1.371134	1.783505	103.000000	2674.288660	37.649485	256.951546	579.144330	198.288660	7.4
1	0.148936	0.202128	9.824468	238.750000	3.930851	28.171809	43.164894	30.244681	0.4
2	8.919355	6.709677	141.725806	2458.306452	33.451613	625.253226	661.458065	860.677419	16.3
3	1.503876	1.604651	56.038760	1392.736434	20.573643	188.358915	270.818605	223.255814	3.3

In [49]:
```
# Let's see the names of the players in each cluster
for cl in df1["KM_segments"].unique():
    print("In cluster {}, the following players are present:".format(cl))
    print(df1[df1["KM_segments"] == cl]["Player_Name"].unique())
    print()
```

```
In cluster 1, the following players are present:
['Alex Runnarsson' 'Cedric Soares' 'Edward Nketiah'
 'Gabriel Teodoro Martinelli Silva' 'Matt Macey' 'Miguel Azeez'
 'Pablo Mari' 'Reiss Nelson' 'Sead Kolasinac' 'Shkodran Mustafi'
 'Sokratis Papastathopoulos' 'William Saliba' 'Ahmed El Mohamady'
 'Carney Chukwuemeka' 'Conor Hourihane' 'Henri Lansbury' 'Jacob Ramsey'
 'Jaden Philogene-Bidace' 'Jose Peleteiro Romallo' 'Keinan Davis'
 'Kortney Hause' 'Marvelous Nakamba' 'Morgan Sanson' 'Orjan Nyland'
 'Wesley Moraes' 'Alireza Jahanbakhsh' 'Andi Zeqiri'
 'Bernardo Fernandes da Silva Junior' 'Davy Propper' 'Jakub Moder'
 'Jason Steele' 'Jayson Molumby' 'Jose Izquierdo' 'Percy Tau'
 'Reda Khadra' 'Anthony Driscoll-Glennon' 'Bailey Peacock-Farrell'
 'Dale Stephens' 'Jimmy Dunne' 'Joel Mumbongo' 'Josh Benson' 'Kevin Long'
 'Lewis Richardson' 'Phil Bardsley' 'Will Norris' 'Billy Gilmour'
 'Emerson Palmieri dos Santos' 'Faustino Anjorin' 'Fikayo Tomori'
 'Karlo Ziger' 'Kepa Arrizabalaga' 'Valentino Livramento'
 'Willy Caballero' 'Connor Wickham' 'Jack Butland' 'James McCarthy'
 'James Tomkins' 'Jean-Philippe Mateta' 'Mamadou Sakho' 'Martin Kelly'
 'Nathan Ferguson' 'Reece Hannam' 'Ryan Inniss' 'Sam Woods'
 'Stephen Henderson' 'Anthony Gordon' 'Bernard Caldeira Duarte'
 'Cenk Tosun' 'Fabian Delph' 'Joao Virginia' 'Jonjoe Kenny' 'Joshua King'
 'Moise Kean' 'Nathan Broadhead' 'Niels Nkounkou' 'Robin Olsen'
 'Adam Forshaw' 'Francisco Casilla' 'Gaetano Berardi'
 'Ian Carlo Poveda-Ocampo' 'Jack Jenkins' 'Jamie Shackleton'
 'Jay-Roy Grot' 'Jordan Stevens' 'Kamil Miazek' 'Leif Davis'
 'Mateusz Bogusz' 'Niall Huggins' 'Pablo Hernandez' 'Cengiz Under'
 'Christian Fuchs' 'Daniel Amartey' 'Demarai Gray' 'Filip Benkovic'
 'Hamza Choudhury' 'Islam Slimani' 'Luke Thomas' 'Sidnei Tavares'
 'Thakgalo Leshabela' 'Vontae Daley-Campbell' 'Wes Morgan'
 'Adrian Castillo' 'Alex Oxlade-Chamberlain' 'Caoimhin Kelleher'
 'Divock Origi' 'Joel Matip' 'Joseph Gomez' 'Naby Keita' 'Neco Williams'
 'Ozan Kabak' 'Rhys Williams' 'Virgil van Dijk' 'Xherdan Shaqiri'
 'Eric Garcia' 'Liam Delap' 'Luke Mbete' 'Nathan Ake' 'Nicolas Otamendi'
 'Scott Carson' 'Taylor Harwood-Bellis' 'Zack Steffen'
 'Alex Nicolao Telles' 'Amad Diallo' 'Anthony Elanga' 'Axel Tuanzebe'
 'Brandon Williams' 'Donny van de Beek' 'Hannibal Mejbri' 'Juan Mata'
 'Nathan Bishop' 'Odion Ighalo' 'Shola Shoretire' 'William Fish'
 'Andy Carroll' 'DeAndre Yedlin' 'Dwight Gayle' 'Elliot Anderson'
 'Florian Lejeune' 'Javier Manquillo' 'Kelland Watts' 'Matthew Longstaff'
 'Yoshinori Muto' 'Caleb Watts' "Daniel N'Lundulu" 'Fraser Forster'
 'Jake Vokins' 'Kgaogelo Chauke' 'Michael Obafemi' 'Mohammed Salisu'
 'Nathan Tella' 'Shane Long' 'William Smallbone' 'Yan Valery'
 'Bamidele Alli' 'Cameron Carter-Vickers' 'Carlos Vinicius Alves Morais'
 'Dane Scarlett' 'Danny Rose' 'Erik Lamela' 'Harry Winks'
 'Japhet Tanganga' 'Joe Rodon' 'Juan Foyth' 'Paulo Gazzaniga'
 'Ryan Sessegnon' 'Ademipo Odubeko' 'Albian Ajeti' 'Andriy Yarmolenko'
 'Ben Johnson' 'Darren Randolph' 'Felipe Anderson Pereira Gomes'
 'Frederik Alves' 'Jamal Baptiste' 'Jordan Hugill' 'Manuel Lanzini'
 'Mark Noble' 'Roberto Jimenez Gago' 'Ryan Fredericks' 'Fernando Marcal'
 'John Ruddy' 'Jonathan Castro Otto' 'Ki-Jana Hoever' 'Morgan Gibbs-White'
 'Oskar Buur' 'Owen Otasowie' 'Patrick Cutrone' 'Ruben Vinagre'
 'Vitor Ferreira']

In cluster 2, the following players are present:
['Alexandre Lacazette' 'Bukayo Saka' 'Nicolas Pepe'
 'Pierre-Emerick Aubameyang' 'Anwar El Ghazi' 'Bertrand Traore'
 'Jack Grealish' 'Ollie Watkins' 'Leandro Trossard' 'Neal Maupay'
 'Pascal Gross' 'Chris Wood' 'Mason Mount' 'Timo Werner'
 'Christian Benteke' 'Eberechi Eze' 'Wilfried Zaha'
 'Dominic Calvert-Lewin' 'Gylfi Sigurdsson' 'James Rodriguez'
 'Lucas Digne' 'Richarlison de Andrade' 'Jack Harrison' 'Patrick Bamford'
 'Raphael Dias Belloli' 'Rodrigo Moreno' 'Stuart Dallas' 'Harvey Barnes'
 'James Maddison' 'Jamie Vardy' 'Kelechi Iheanacho' 'Youri Tielemans'
 'Andrew Robertson' 'Mohamed Salah' 'Roberto Firmino' 'Sadio Mane'
 'Trent Alexander-Arnold' 'Gabriel Fernando de Jesus' 'Ilkay Gundogan'
 'Joao Cancelo' 'Kevin De Bruyne' 'Phil Foden' 'Raheem Sterling'
 'Riyad Mahrez' 'Bruno Fernandes' 'Edinson Cavani' 'Luke Shaw'
 'Marcus Rashford' 'Callum Wilson' 'Che Adams' 'Danny Ings'
 'James Ward-Prowse' 'Gareth Bale' 'Harry Kane' 'Heung-Min Son'
 'Aaron Cresswell' 'Jarrod Bowen' 'Jesse Lingard' 'Michail Antonio'
 'Pablo Fornals' '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' 'Mason Holgate' 'Michael Keane' 'Ezgjan 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' '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 Djenepo' 'Nathan Redmond' 'Oriol Romeu Vidal' 'Takumi Minamino'
 'Theo Walcott' 'Ben Davies' 'Davinson Sanchez' 'Giovani Lo Celso'
 'Lucas Moura' 'Matt Doherty' 'Moussa Sissoko' 'Serge Aurier'
 'Steven Bergwijn' 'Arthur Masuaku' 'Craig Dawson' 'Fabian Balbuena'
 'Issa Diop' 'Said Benrahma' 'Sebastian Haller' 'Daniel Castelo Podence'
 'Fabio Silva' 'Max Kilman' 'Raul Jimenez ' 'Rayan Ait Nouri'
 'Willian Jose' 'Willy Boly']
```

In [50]:
```
df1.groupby(["KM_segments", "Position"])['Player_Name'].count()
```

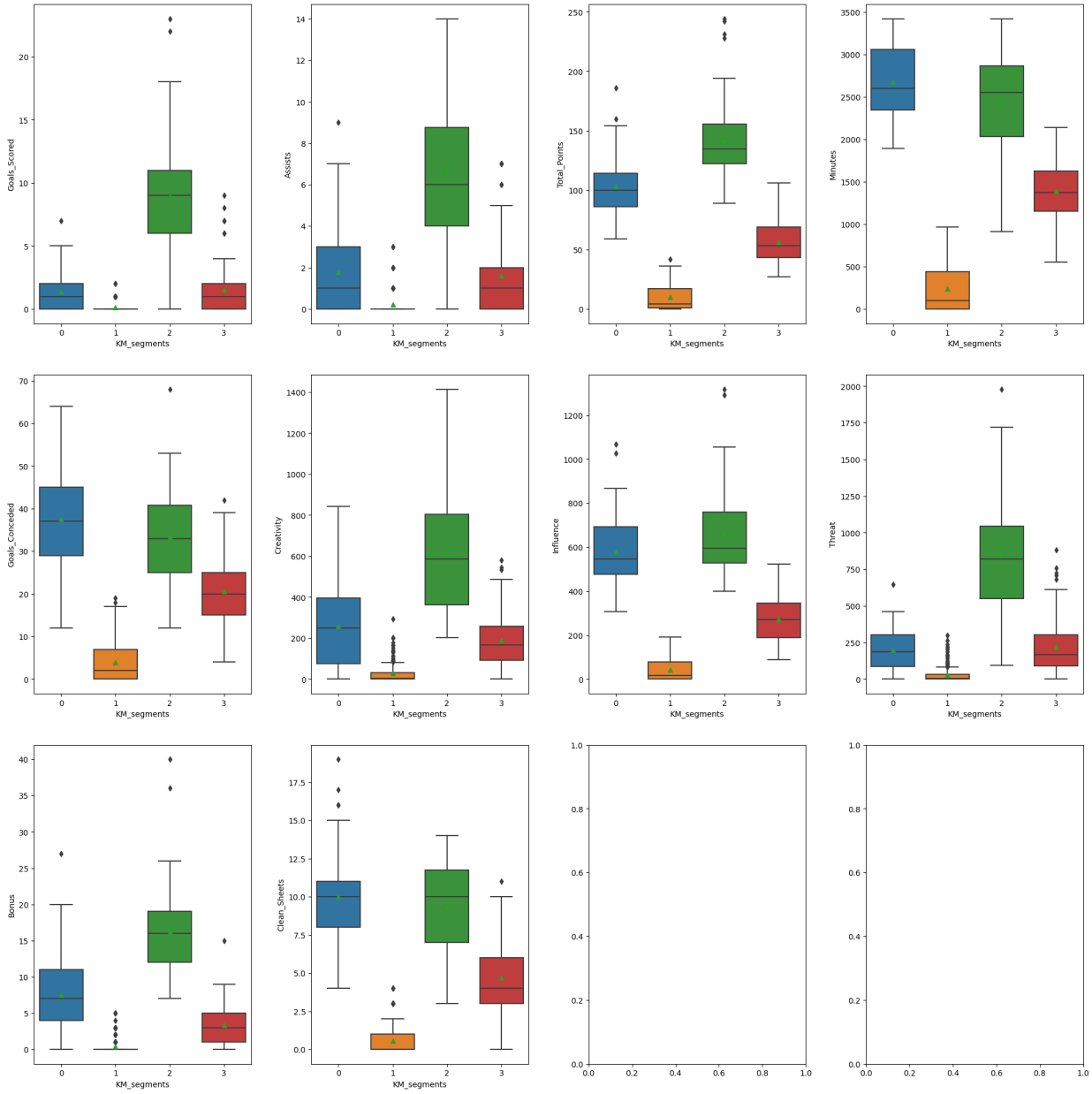
Out[50]:
```
KM_segments  Position
0            Defender      48
             Goalkeeper    17
             Midfielder    32
1            Defender      70
             Forward       28
             Goalkeeper    25
             Midfielder    65
2            Defender       7
             Forward       20
             Midfielder    35
3            Defender      47
             Forward       16
             Goalkeeper     3
             Midfielder    63
Name: Player_Name, dtype: int64
```

Observations:

- Cluster 0 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

In [51]:
```
fig, axes = plt.subplots(3, 4, figsize = (20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 10:
            sns.boxplot(
                ax = axes[ii][jj],
                data = df1,
                y = df1.columns[3 + counter],
                x = "KM_segments", showmeans = True
            )
            counter = counter + 1

fig.tight_layout(pad = 3.0)
```



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.

In [52]:
```
k_med_df = data_pca.copy()
```

In [53]:
```
kmed = KMedoids(n_clusters = 4, random_state = 1) # Create K-Medoids with nclusters = 4
kmed.fit(k_med_df)
```

Out[53]:
```
KMedoids
KMedoids(n_clusters=4, random_state=1)
```

In [54]:
```
# Creating a copy of the original data
df2 = df.copy()

# Add K-Medoids cluster labels to K-Medoids data
k_med_df["KMed_segments"] = kmed.labels_
# Add K-Medoids cluster labels to the whole data
df2["KMed_segments"] =  kmed.labels_
```

Cluster Profiling

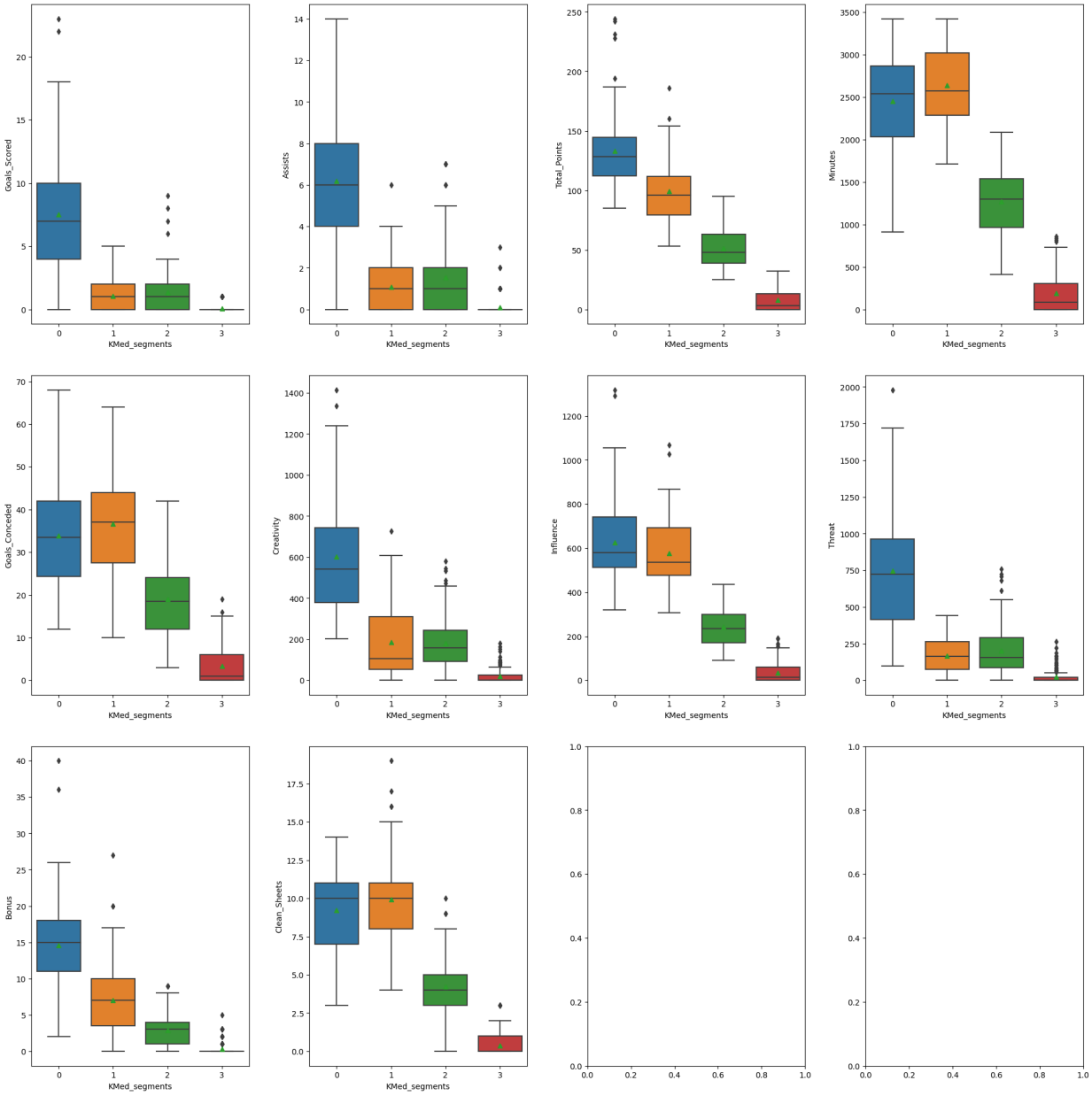
In [55]:
```
kmed_cluster_profile = df2.groupby("KMed_segments").mean(numeric_only=True)
```

In [56]:
```
kmed_cluster_profile["count_in_each_segment"] = df2.groupby("KMed_segments")["Total_Points"].count().
kmed_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[56]:

KMed_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	
0	7.512195	6.195122	133.243902	2452.243902	33.853659	602.902439	625.653659	745.402439	1
1	1.068966	1.091954	99.528736	2638.195402	36.632184	184.582759	575.818391	166.333333	
2	1.338235	1.558824	51.073529	1270.051471	18.977941	180.458824	242.588235	203.102941	
3	0.099415	0.111111	7.736842	193.187135	3.362573	18.979532	34.188304	22.608187	

In [57]:
```
fig, axes = plt.subplots(3, 4, figsize = (20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 10:
            sns.boxplot(
                ax = axes[ii][jj],
                data = df2,
                y = df2.columns[3 + counter],
                x = "KMed_segments", showmeans = True
            )
            counter = counter + 1

fig.tight_layout(pad = 3.0)
```



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.

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

In [58]:
```
hc_df = data_pca.copy()
```

In [59]:
```
# List of distance metrics
distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"]

# List of linkage methods
linkage_methods = ["single", "complete", "average", "weighted"]

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(hc_df, metric = dm, method = lm)
        c, coph_dists = cophenet(Z, pdist(hc_df))
        print(
            "Cophenetic correlation for {} distance and {} linkage is {}.".format(
                dm.capitalize(), lm, c
            )
        )
        if high_cophenet_corr < c:
            high_cophenet_corr = c
            high_dm_lm[0] = dm
            high_dm_lm[1] = lm

# Printing the combination of distance metric and linkage method with the highest cophenetic correlat
print('*'*100)
print(
    "Highest cophenetic correlation is {}, which is obtained with {} distance and {} linkage.".format
        high_cophenet_corr, high_dm_lm[0].capitalize(), high_dm_lm[1]
    )
)
```

```
Cophenetic correlation for Euclidean distance and single linkage is 0.8430175514228708.
Cophenetic correlation for Euclidean distance and complete linkage is 0.741204129226176.
Cophenetic correlation for Euclidean distance and average linkage is 0.8476499945585417.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.8624581351067481.
Cophenetic correlation for Chebyshev distance and single linkage is 0.8381223141111798.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.8028394390632132.
Cophenetic correlation for Chebyshev distance and average linkage is 0.8167064931302255.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.844849787663964.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.8065008904132245.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.6583135946489013.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.7747800632434059.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.6486408054242748.
Cophenetic correlation for Cityblock distance and single linkage is 0.840183551166332.
Cophenetic correlation for Cityblock distance and complete linkage is 0.8241586035407029.
Cophenetic correlation for Cityblock distance and average linkage is 0.8564523087071935.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.8395672301050403.
****************************************************************************************************
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.

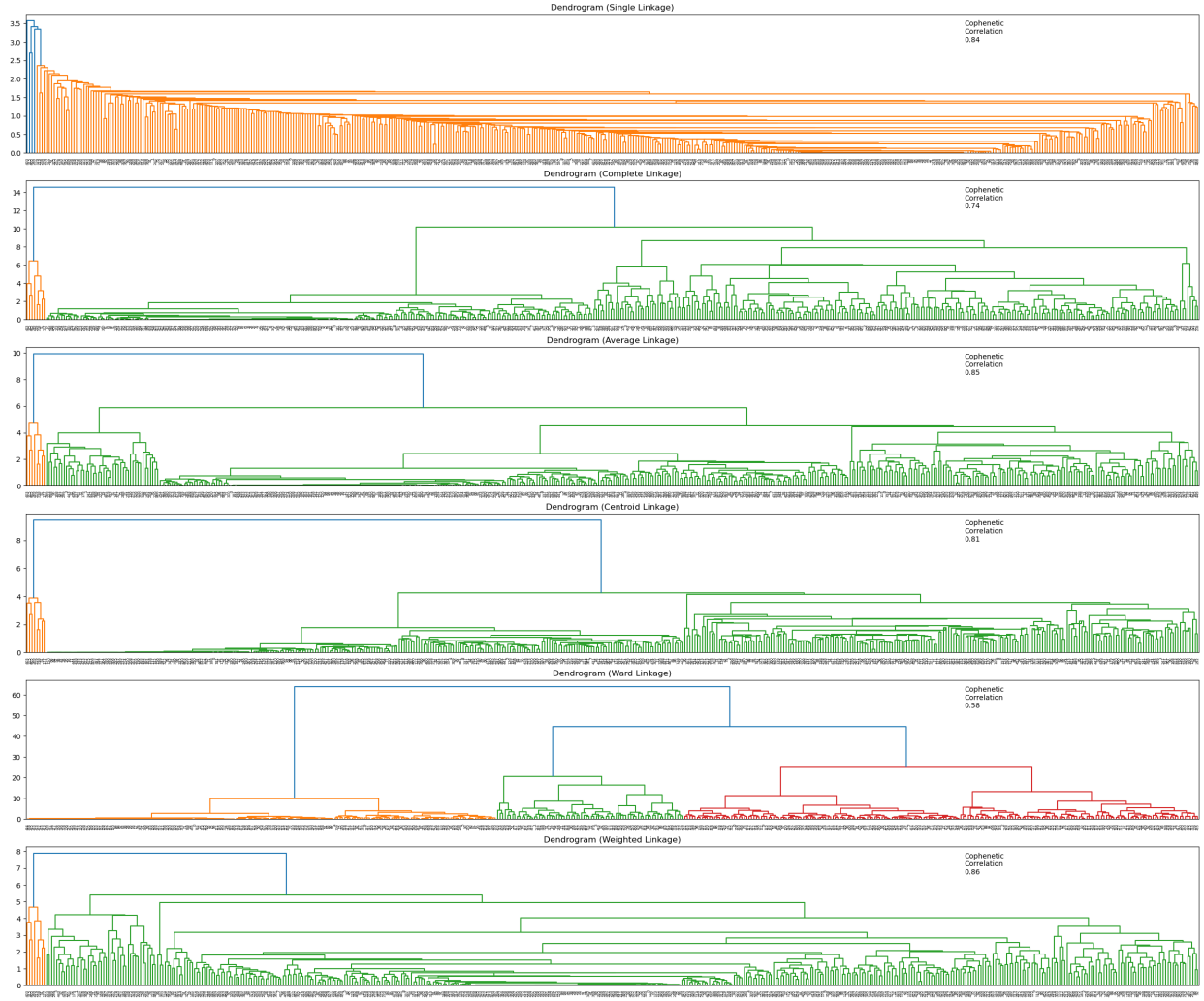
In [60]:
```
# List of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(hc_df, metric = "euclidean", method = lm)
    c, coph_dists = cophenet(Z, pdist(hc_df))
    print("Cophenetic correlation for {} linkage is {}.".format(lm, c))
    if high_cophenet_corr < c:
        high_cophenet_corr = c
        high_dm_lm[0] = "euclidean"
        high_dm_lm[1] = lm

# Printing the combination of distance metric and linkage method with the highest cophenetic correlat
print('*'*100)
print(
    "Highest cophenetic correlation is {}, which is obtained with {} linkage.".format(
        high_cophenet_corr, high_dm_lm[1]
    )
)
```

```
Cophenetic correlation for single linkage is 0.8430175514228708.
Cophenetic correlation for complete linkage is 0.741204129226176.
Cophenetic correlation for average linkage is 0.8476499945585417.
Cophenetic correlation for centroid linkage is 0.8068296032280465.
Cophenetic correlation for ward linkage is 0.577773844586155.
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.

In [61]:
```
# List of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

# Lists to save results of cophenetic correlation calculation
compare_cols = ["Linkage", "Cophenetic Coefficient"]
compare = []

# To create a subplot image
fig, axs = plt.subplots(len(linkage_methods), 1, figsize = (30, 25))

# We will enumerate through the list of linkage methods above
# For each linkage method, we will plot the dendrogram and calculate the cophenetic correlation
for i, method in enumerate(linkage_methods):
    Z = linkage(hc_df, metric = "euclidean", method = method)

    dendrogram(Z, ax = axs[i])
    axs[i].set_title(f"Dendrogram ({method.capitalize()} Linkage)")

    coph_corr, coph_dist = cophenet(Z, pdist(hc_df))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.80, 0.80),
        xycoords="axes fraction",
    )

    compare.append([method, coph_corr])
```



In [62]:
```
# Create and print a dataframe to compare cophenetic correlations for different linkage methods
df_cc = pd.DataFrame(compare, columns = compare_cols)

df_cc = df_cc.sort_values(by = "Cophenetic Coefficient")
df_cc
```

Out[62]:

	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

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

In [63]:
```
HCmodel = AgglomerativeClustering(n_clusters = 4, metric = "euclidean", linkage = "average")
HCmodel.fit(hc_df)
```

Out[63]:
```
AgglomerativeClustering(linkage='average', metric='euclidean', n_clusters=4)
```

In [64]:
```
# Creating a copy of the original data
df3 = df.copy()

# Adding hierarchical cluster labels to the original and whole dataframes
df3["HC_segments"] = HCmodel.labels_
```

Cluster Profiling

In [65]:
```
hc_cluster_profile = df3.groupby("HC_segments").mean(numeric_only=True)
```

In [66]:
```
hc_cluster_profile["count_in_each_segment"] = df3.groupby("HC_segments")["Total_Points"].count().valu
```

In [67]:
```
hc_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[67]:

HC_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat
0	0.881517	1.139810	47.969194	1205.945498	17.580569	148.574408	249.536967	131.753555
1	16.800000	9.200000	189.000000	3033.200000	44.000000	494.340000	860.720000	1591.600000
2	8.565217	5.826087	129.391304	2238.934783	29.760870	543.273913	586.234783	861.739130
3	19.333333	13.000000	238.000000	3101.000000	37.000000	1041.300000	1221.000000	1294.666667

In [68]:
```
# Let's see the names of the players in each cluster
for cl in df3["HC_segments"].unique():
    print("In cluster {}, the following players are present:".format(cl))
    print(df3[df3["HC_segments"] == cl]["Player_Name"].unique())
    print()
```

```
In cluster 0, the following players are present:
['Alex Runnarsson' 'Bernd Leno' 'Calum Chambers' 'Cedric Soares'
 'Daniel Ceballos' 'David Luiz' 'Edward Nketiah' 'Emile Smith Rowe'
 'Gabriel Maghalaes' 'Gabriel Teodoro Martinelli Silva' 'Granit Xhaka'
 'Hector Bellerin' 'Kieran Tierney' 'Martin Odegaard' 'Matt Macey'
 'Miguel Azeez' 'Mohamed Naser El Sayed Elneny' 'Pablo Mari'
 'Reiss Nelson' 'Rob Holding' 'Sead Kolasinac' 'Shkodran Mustafi'
 'Sokratis Papastathopoulos' 'Thomas Partey' 'William Saliba'
 'Willian Borges Da Silva' 'Ahmed El Mohamady' 'Carney Chukwuemeka'
 'Conor Hourihane' 'Douglas Luiz Soares de Paulo' 'Emiliano Martinez'
 'Ezri Konsa Ngoyo' 'Henri Lansbury' 'Jacob Ramsey'
 'Jaden Philogene-Bidace' 'John McGinn' 'Jose Peleteiro Romallo'
 'Keinan Davis' 'Kortney Hause' 'Mahmoud Ahmed Ibrahim Hassan'
 'Marvelous Nakamba' 'Matt Targett' 'Matthew Cash' 'Morgan Sanson'
 'Orjan Nyland' 'Ross Barkley' 'Tyrone Mings' 'Wesley Moraes'
 'Aaron Connolly' 'Adam Lallana' 'Adam Webster' 'Alexis Mac Allister'
 'Alireza Jahanbakhsh' 'Andi Zeqiri' 'Ben White'
 'Bernardo Fernandes da Silva Junior' 'Dan Burn' 'Davy Propper'
 'Jakub Moder' 'Jason Steele' 'Jayson Molumby' 'Joel Veltman'
 'Jose Izquierdo' 'Lewis Dunk' 'Mathew Ryan' 'Percy Tau' 'Reda Khadra'
 'Robert Sanchez' 'Solomon March' 'Steven Alzate' 'Tariq Lamptey'
 'Yves Bissouma' 'Anthony Driscoll-Glennon' 'Ashley Barnes'
 'Ashley Westwood' 'Bailey Peacock-Farrell' 'Ben Mee' 'Charlie Taylor'
 'Dale Stephens' 'Dwight McNeil' 'Erik Pieters' 'Jack Cork'
 'James Tarkowski' 'Jay Rodriguez' 'Jeff Hendrick' 'Jimmy Dunne'
 'Joel Mumbongo' 'Johann Berg Gudmundsson' 'Josh Benson' 'Josh Brownhill'
 'Kevin Long' 'Lewis Richardson' 'Matej Vydra' 'Matthew Lowton'
 'Nick Pope' 'Phil Bardsley' 'Robbie Brady' 'Will Norris'
 'Andreas Christensen' 'Antonio Rudiger' 'Benjamin Chilwell'
 'Billy Gilmour' 'Callum Hudson-Odoi' 'Cesar Azpilicueta'
 'Christian Pulisic' 'Edouard Mendy' 'Emerson Palmieri dos Santos'
 'Faustino Anjorin' 'Fikayo Tomori' 'Hakim Ziyech'
 'Jorge Luiz Frello Filho' 'Kai Havertz' 'Karlo Ziger' 'Kepa Arrizabalaga'
 'Kurt Zouma' 'Marcos Alonso' 'Mateo Kovacic' "N'Golo Kante"
 'Olivier Giroud' 'Reece James' 'Tammy Abraham' 'Thiago Silva'
 'Valentino Livramento' 'Willy Caballero' 'Andros Townsend'
 'Cheikhou Kouyate' 'Connor Wickham' 'Eberechi Eze' 'Gary Cahill'
 'Jack Butland' 'Jairo Riedewald' 'James McArthur' 'James McCarthy'
 'James Tomkins' 'Jean-Philippe Mateta' 'Jeffrey Schlupp' 'Joel Ward'
 'Jordan Ayew' 'Luka Milivojevic' 'Mamadou Sakho' 'Martin Kelly'
 'Michy Batshuayi' 'Nathan Ferguson' 'Nathaniel Clyne'
 'Patrick van Aanholt' 'Reece Hannam' 'Ryan Inniss' 'Sam Woods'
 'Scott Dann' 'Stephen Henderson' 'Tyrick Mitchell' 'Vicente Guaita'
 'Abdoulaye Doucoure' 'Alex Iwobi' 'Allan Marques Loureiro'
 'Andre Tavares Gomes' 'Anthony Gordon' 'Ben Godfrey'
 'Bernard Caldeira Duarte' 'Cenk Tosun' 'Fabian Delph' 'James Rodriguez'
 'Joao Virginia' 'Jonjoe Kenny' 'Jordan Pickford' 'Joshua King'
 'Lucas Digne' 'Mason Holgate' 'Michael Keane' 'Moise Kean'
 'Nathan Broadhead' 'Niels Nkounkou' 'Robin Olsen' 'Seamus Coleman'
 'Tom Davies' 'Yerry Mina' 'Adam Forshaw' 'Diego Llorente'
 'Ezgjan Alioski' 'Francisco Casilla' 'Gaetano Berardi' 'Helder Costa'
 'Ian Carlo Poveda-Ocampo' 'Illan Meslier' 'Jack Jenkins'
 'Jamie Shackleton' 'Jay-Roy Grot' 'Jordan Stevens' 'Kalvin Phillips'
 'Kamil Miazek' 'Leif Davis' 'Liam Cooper' 'Luke Ayling' 'Mateusz Bogusz'
 'Mateusz Klich' 'Niall Huggins' 'Pablo Hernandez' 'Pascal Struijk'
 'Robin Koch' 'Tyler Roberts' 'Ayoze Perez' 'Calgar Soyuncu'
 'Cengiz Under' 'Christian Fuchs' 'Daniel Amartey' 'Demarai Gray'
 'Dennis Praet' 'Filip Benkovic' 'Hamza Choudhury' 'Islam Slimani'
 'James Justin' 'Jonny Evans' 'Kasper Schmeichel' 'Luke Thomas'
 'Marc Albrighton' 'Nampalys Mendy' 'Ricardo Domingos Barbosa Pereira'
 'Sidnei Tavares' 'Thakgalo Leshabela' 'Timothy Castagne'
 'Vontae Daley-Campbell' 'Wes Morgan' 'Wesley Fofana' 'Wilfred Ndidi'
 'Youri Tielemans' 'Adrian Castillo' 'Alex Oxlade-Chamberlain'
 'Alisson Becker' 'Andrew Robertson' 'Caoimhin Kelleher' 'Curtis Jones'
 'Dean Henderson' 'Diogo Jota' 'Divock Origi' 'Fabio Henrique Tavares'
 'Georginio Wijnaldum' 'James Milner' 'Joel Matip' 'Jordan Henderson'
 'Joseph Gomez' 'Naby Keita' 'Nathaniel Phillips' 'Neco Williams'
 'Ozan Kabak' 'Rhys Williams' 'Thiago Alcantara' 'Trent Alexander-Arnold'
 'Virgil van Dijk' 'Xherdan Shaqiri' 'Aymeric Laporte' 'Benjamin Mendy'
 'Bernardo Silva' 'Ederson Moares' 'Eric Garcia' 'Fernando Luiz Rosa'
 'Ferran Torres' 'Joao Cancelo' 'John Stones' 'Kyle Walker' 'Liam Delap'
```

```
 'Luke Mbete' 'Nathan Ake' 'Nicolas Otamendi' 'Oleksandr Zinchenko'
 'Rodrigo Hernandez' 'Ruben Dias' 'Scott Carson' 'Sergio Aguero'
 'Taylor Harwood-Bellis' 'Zack Steffen' 'Aaron Wan-Bissaka'
 'Alex Nicolao Telles' 'Amad Diallo' 'Anthony Elanga' 'Anthony Martial'
 'Axel Tuanzebe' 'Brandon Williams' 'Daniel James' 'David de Gea'
 'Donny van de Beek' 'Eric Bailly' 'Frederico Rodrigues de Paula Santos'
 'Hannibal Mejbri' 'Harry Maguire' 'Juan Mata' 'Luke Shaw' 'Nathan Bishop'
 'Nemanja Matic' 'Odion Ighalo' 'Paul Pogba' 'Scott McTominay'
 'Shola Shoretire' 'Victor Lindelof' 'William Fish' 'Allan Saint-Maximin'
 'Andy Carroll' 'Ciaran Clark' 'DeAndre Yedlin' 'Dwight Gayle'
 'Elliot Anderson' 'Emil Krafth' 'Fabian Schar' 'Federico Fernandez'
 'Florian Lejeune' 'Isaac Hayden' 'Jacob Murphy' 'Jamaal Lascelles'
 'Jamal Lewis' 'Javier Manquillo' 'Joelinton de Lira' 'Jonjo Shelvey'
 'Joseph Willock' 'Karl Darlow' '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']

In cluster 1, the following players are present:
['Ollie Watkins' 'Dominic Calvert-Lewin' 'Patrick Bamford' 'Jamie Vardy'
 'Mohamed Salah']

In cluster 3, the following players are present:
['Bruno Fernandes' 'Harry Kane' 'Heung-Min Son']
```

In [69]:
```
df3.groupby(["HC_segments", "Position"])['Player_Name'].count()
```

Out[69]:
```
HC_segments  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
Name: Player_Name, 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.

In [70]:
```
HCmodel = AgglomerativeClustering(n_clusters = 4, affinity = "euclidean", linkage = "ward")
HCmodel.fit(hc_df)
```

```
C:\Users\jaces\.conda\envs\base_mit\lib\site-packages\sklearn\cluster\_agglomerative.py:983: FutureW
arning: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric`
instead
  warnings.warn(
```

Out[70]:
```
AgglomerativeClustering(affinity='euclidean', n_clusters=4)
```

In [71]:
```
# Creating a copy of the original data
df3 = df.copy()

# Adding hierarchical cluster labels to the HC algorithm and original dataframes
df3["HC_segments"] = HCmodel.labels_
```

Cluster Profiling

In [72]:
```
hc_cluster_profile = df3.groupby("HC_segments").mean(numeric_only=True)
```

In [73]:
```
hc_cluster_profile["count_in_each_segment"] = df3.groupby("HC_segments")["Total_Points"].count().valu
```

In [74]:
```
hc_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[74]:

HC_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	E
0	7.960526	6.342105	135.592105	2467.605263	33.802632	594.343421	638.431579	772.302632	14.7
1	1.357143	1.669643	53.812500	1367.767857	20.866071	186.037500	259.967857	218.866071	3.0
2	1.247423	1.206186	98.453608	2557.814433	35.371134	220.125773	556.723711	185.505155	7.2
3	0.157068	0.251309	10.324607	248.863874	4.094241	31.026702	45.018848	30.785340	0.4

In [75]:
```
# Let's see the names of the players in each cluster
for cl in df3["HC_segments"].unique():
    print("In cluster {}, the following players are present:".format(cl))
    print(df3[df3["HC_segments"] == cl]["Player_Name"].unique())
    print()
```

```
In cluster 3, the following players are present:
['Alex Runnarsson' 'Calum Chambers' 'Cedric Soares' 'Edward Nketiah'
 'Martin Odegaard' 'Matt Macey' 'Miguel Azeez' 'Pablo Mari' 'Reiss Nelson'
 'Sead Kolasinac' 'Shkodran Mustafi' 'Sokratis Papastathopoulos'
 'William Saliba' 'Ahmed El Mohamady' 'Carney Chukwuemeka'
 'Conor Hourihane' 'Henri Lansbury' 'Jacob Ramsey'
 'Jaden Philogene-Bidace' 'Jose Peleteiro Romallo' 'Keinan Davis'
 'Kortney Hause' 'Marvelous Nakamba' 'Morgan Sanson' 'Orjan Nyland'
 'Wesley Moraes' 'Alireza Jahanbakhsh' 'Andi Zeqiri'
 'Bernardo Fernandes da Silva Junior' 'Davy Propper' 'Jakub Moder'
 'Jason Steele' 'Jayson Molumby' 'Jose Izquierdo' 'Percy Tau'
 'Reda Khadra' 'Tariq Lamptey' 'Anthony Driscoll-Glennon'
 'Bailey Peacock-Farrell' 'Dale Stephens' 'Jimmy Dunne' 'Joel Mumbongo'
 'Josh Benson' 'Kevin Long' 'Lewis Richardson' 'Phil Bardsley'
 'Will Norris' 'Billy Gilmour' 'Emerson Palmieri dos Santos'
 'Faustino Anjorin' 'Fikayo Tomori' 'Karlo Ziger' 'Kepa Arrizabalaga'
 'Valentino Livramento' 'Willy Caballero' 'Connor Wickham' 'Jack Butland'
 'James McCarthy' 'James Tomkins' 'Jean-Philippe Mateta' 'Mamadou Sakho'
 'Martin Kelly' 'Michy Batshuayi' 'Nathan Ferguson' 'Reece Hannam'
 'Ryan Inniss' 'Sam Woods' 'Stephen Henderson' 'Anthony Gordon'
 'Bernard Caldeira Duarte' 'Cenk Tosun' 'Fabian Delph' 'Joao Virginia'
 'Jonjoe Kenny' 'Joshua King' 'Moise Kean' 'Nathan Broadhead'
 'Niels Nkounkou' 'Robin Olsen' 'Adam Forshaw' 'Francisco Casilla'
 'Gaetano Berardi' 'Ian Carlo Poveda-Ocampo' 'Jack Jenkins'
 'Jamie Shackleton' 'Jay-Roy Grot' 'Jordan Stevens' 'Kamil Miazek'
 'Leif Davis' 'Mateusz Bogusz' 'Niall Huggins' 'Pablo Hernandez'
 'Cengiz Under' 'Christian Fuchs' 'Daniel Amartey' 'Demarai Gray'
 'Dennis Praet' 'Filip Benkovic' 'Hamza Choudhury' 'Islam Slimani'
 'Luke Thomas' 'Sidnei Tavares' 'Thakgalo Leshabela'
 'Vontae Daley-Campbell' 'Wes Morgan' 'Adrian Castillo'
 'Alex Oxlade-Chamberlain' 'Caoimhin Kelleher' 'Divock Origi' 'Joel Matip'
 'Joseph Gomez' 'Naby Keita' 'Neco Williams' 'Ozan Kabak' 'Rhys Williams'
 'Virgil van Dijk' 'Xherdan Shaqiri' 'Eric Garcia' 'Liam Delap'
 'Luke Mbete' 'Nathan Ake' 'Nicolas Otamendi' 'Scott Carson'
 'Taylor Harwood-Bellis' 'Zack Steffen' 'Alex Nicolao Telles'
 'Amad Diallo' 'Anthony Elanga' 'Axel Tuanzebe' 'Brandon Williams'
 'Donny van de Beek' 'Hannibal Mejbri' 'Juan Mata' 'Nathan Bishop'
 'Odion Ighalo' 'Shola Shoretire' 'William Fish' 'Andy Carroll'
 'DeAndre Yedlin' 'Dwight Gayle' 'Elliot Anderson' 'Florian Lejeune'
 'Javier Manquillo' 'Kelland Watts' 'Matthew Longstaff' 'Yoshinori Muto'
 'Caleb Watts' "Daniel N'Lundulu" 'Fraser Forster' 'Jake Vokins'
 'Kgaogelo Chauke' 'Michael Obafemi' 'Mohammed Salisu' 'Nathan Tella'
 'Shane Long' 'William Smallbone' 'Yan Valery' 'Bamidele Alli'
 'Cameron Carter-Vickers' 'Carlos Vinicius Alves Morais' 'Dane Scarlett'
 'Danny Rose' 'Harry Winks' 'Japhet Tanganga' 'Joe Rodon' 'Juan Foyth'
 'Paulo Gazzaniga' 'Ryan Sessegnon' 'Ademipo Odubeko' 'Albian Ajeti'
 'Andriy Yarmolenko' 'Ben Johnson' 'Darren Randolph'
 'Felipe Anderson Pereira Gomes' 'Frederik Alves' 'Jamal Baptiste'
 'Jordan Hugill' 'Manuel Lanzini' 'Mark Noble' 'Roberto Jimenez Gago'
 'Ryan Fredericks' 'Fernando Marcal' 'John Ruddy' 'Jonathan Castro Otto'
 'Ki-Jana Hoever' 'Morgan Gibbs-White' 'Oskar Buur' 'Owen Otasowie'
 'Patrick Cutrone' 'Ruben Vinagre' 'Vitor Ferreira']

In cluster 0, the following players are present:
['Alexandre Lacazette' 'Bukayo Saka' 'Nicolas Pepe'
 'Pierre-Emerick Aubameyang' 'Anwar El Ghazi' 'Bertrand Traore'
 'Jack Grealish' 'John McGinn' 'Matt Targett' 'Ollie Watkins'
 'Danny Welbeck' 'Leandro Trossard' 'Neal Maupay' 'Pascal Gross'
 'Ashley Westwood' 'Chris Wood' 'Dwight McNeil' 'Benjamin Chilwell'
 'Mason Mount' 'Timo Werner' 'Christian Benteke' 'Eberechi Eze'
 'Wilfried Zaha' 'Dominic Calvert-Lewin' 'Gylfi Sigurdsson'
 'James Rodriguez' 'Lucas Digne' 'Richarlison de Andrade' 'Jack Harrison'
 'Patrick Bamford' 'Raphael Dias Belloli' 'Rodrigo Moreno' 'Stuart Dallas'
 'Harvey Barnes' 'James Maddison' 'Jamie Vardy' 'Kelechi Iheanacho'
 'Youri Tielemans' 'Andrew Robertson' 'Diogo Jota' 'Mohamed Salah'
 'Roberto Firmino' 'Sadio Mane' 'Trent Alexander-Arnold' 'Bernardo Silva'
 'Gabriel Fernando de Jesus' 'Ilkay Gundogan' 'Kevin De Bruyne'
 'Phil Foden' 'Raheem Sterling' 'Riyad Mahrez' 'Aaron Wan-Bissaka'
 'Bruno Fernandes' 'Edinson Cavani' 'Luke Shaw' 'Marcus Rashford'
 'Mason Greenwood' 'Paul Pogba' 'Callum Wilson' 'Joseph Willock'
 'Che Adams' 'Danny Ings' 'James Ward-Prowse' 'Stuart Armstrong'
```

```
 'Gareth Bale' 'Harry Kane' 'Heung-Min Son' 'Pierre-Emile Hojbjerg'
 'Aaron Cresswell' 'Jarrod Bowen' 'Jesse Lingard' 'Michail Antonio'
 'Pablo Fornals' 'Tomas Soucek' 'Vladimir Coufal' 'Pedro Lomba Neto']

In cluster 2, the following players are present:
['Bernd Leno' 'Gabriel Maghalaes' 'Granit Xhaka' 'Hector Bellerin'
 'Kieran Tierney' 'Rob Holding' 'Douglas Luiz Soares de Paulo'
 'Emiliano Martinez' 'Ezri Konsa Ngoyo' 'Matthew Cash' 'Tyrone Mings'
 'Adam Webster' 'Ben White' 'Joel Veltman' 'Lewis Dunk' 'Robert Sanchez'
 'Yves Bissouma' 'Ben Mee' 'Charlie Taylor' 'James Tarkowski'
 'Josh Brownhill' 'Matthew Lowton' 'Nick Pope' 'Andreas Christensen'
 'Antonio Rudiger' 'Cesar Azpilicueta' 'Edouard Mendy'
 'Jorge Luiz Frello Filho' 'Kurt Zouma' 'Mateo Kovacic' "N'Golo Kante"
 'Reece James' 'Thiago Silva' 'Andros Townsend' 'Cheikhou Kouyate'
 'Joel Ward' 'Luka Milivojevic' 'Vicente Guaita' 'Abdoulaye Doucoure'
 'Ben Godfrey' 'Jordan Pickford' 'Mason Holgate' 'Michael Keane'
 'Yerry Mina' 'Ezgjan 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'
 'Thiago Alcantara' 'Ederson Moares' 'Joao Cancelo' 'John Stones'
 'Kyle Walker' 'Oleksandr Zinchenko' 'Rodrigo Hernandez' 'Ruben Dias'
 'David de Gea' 'Frederico Rodrigues de Paula Santos' 'Harry Maguire'
 'Scott McTominay' 'Victor Lindelof' 'Jonjo Shelvey' 'Karl Darlow'
 'Miguel Almiron' 'Alex McCarthy' 'Jan Bednarek' 'Jannik Vestergaard'
 'Kyle Walker-Peters' 'Ryan Bertrand' 'Eric Dier' 'Hugo Lloris'
 'Sergio Reguilon' 'Tanguy Ndombele' 'Toby Alderweireld' 'Angelo Ogbonna'
 'Craig Dawson' 'Declan Rice' 'Lukasz Fabianski' 'Adama Traore'
 'Conor Coady' 'Joao Santos Moutinho' 'Leander Dendoncker' 'Nelson Semedo'
 'Romain Saiss' 'Ruben Neves' 'Rui Pedro Patricio']

In cluster 1, the following players are present:
['Daniel Ceballos' 'David Luiz' 'Emile Smith Rowe'
 'Gabriel Teodoro Martinelli Silva' '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' 'Mathew Ryan' 'Solomon March' 'Steven Alzate' 'Ashley Barnes'
 'Erik Pieters' 'Jack Cork' 'Jay Rodriguez' 'Jeff Hendrick'
 'Johann Berg Gudmundsson' 'Matej Vydra' 'Robbie Brady'
 'Callum Hudson-Odoi' 'Christian Pulisic' 'Hakim Ziyech' 'Kai Havertz'
 'Marcos Alonso' 'Olivier Giroud' 'Tammy Abraham' 'Gary Cahill'
 'Jairo Riedewald' 'James McArthur' 'Jeffrey Schlupp' 'Jordan Ayew'
 'Nathaniel Clyne' 'Patrick van Aanholt' 'Scott Dann' 'Tyrick Mitchell'
 'Alex Iwobi' 'Allan Marques Loureiro' 'Andre Tavares Gomes'
 'Seamus Coleman' 'Tom Davies' 'Diego Llorente' 'Helder Costa'
 'Pascal Struijk' 'Robin Koch' 'Tyler Roberts' 'Ayoze Perez'
 'Calgar Soyuncu' 'Marc Albrighton' 'Nampalys Mendy'
 'Ricardo Domingos Barbosa Pereira' 'Curtis Jones' 'Dean Henderson'
 'James Milner' 'Jordan Henderson' 'Nathaniel Phillips' 'Aymeric Laporte'
 'Benjamin Mendy' 'Fernando Luiz Rosa' 'Ferran Torres' 'Sergio Aguero'
 'Anthony Martial' 'Daniel James' 'Eric Bailly' 'Nemanja Matic'
 'Allan Saint-Maximin' 'Ciaran Clark' 'Emil Krafth' 'Fabian Schar'
 'Federico Fernandez' 'Isaac Hayden' 'Jacob Murphy' 'Jamaal Lascelles'
 'Jamal Lewis' 'Joelinton de Lira' 'Martin Dubravka' 'Matt Ritchie'
 'Paul Dummett' 'Ryan Fraser' 'Sean Longstaff' 'Ibrahima Diallo'
 'Jack Stephens' 'Moussa Djenepo' 'Nathan Redmond' 'Oriol Romeu Vidal'
 'Takumi Minamino' 'Theo Walcott' 'Ben Davies' 'Davinson Sanchez'
 'Erik Lamela' 'Giovani Lo Celso' 'Lucas Moura' 'Matt Doherty'
 'Moussa Sissoko' 'Serge Aurier' 'Steven Bergwijn' 'Arthur Masuaku'
 'Fabian Balbuena' 'Issa Diop' 'Said Benrahma' 'Sebastian Haller'
 'Daniel Castelo Podence' 'Fabio Silva' 'Max Kilman' 'Raul Jimenez '
 'Rayan Ait Nouri' 'Willian Jose' 'Willy Boly']
```

In [76]:
```
df3.groupby(["HC_segments", "Position"])['Player_Name'].count()
```

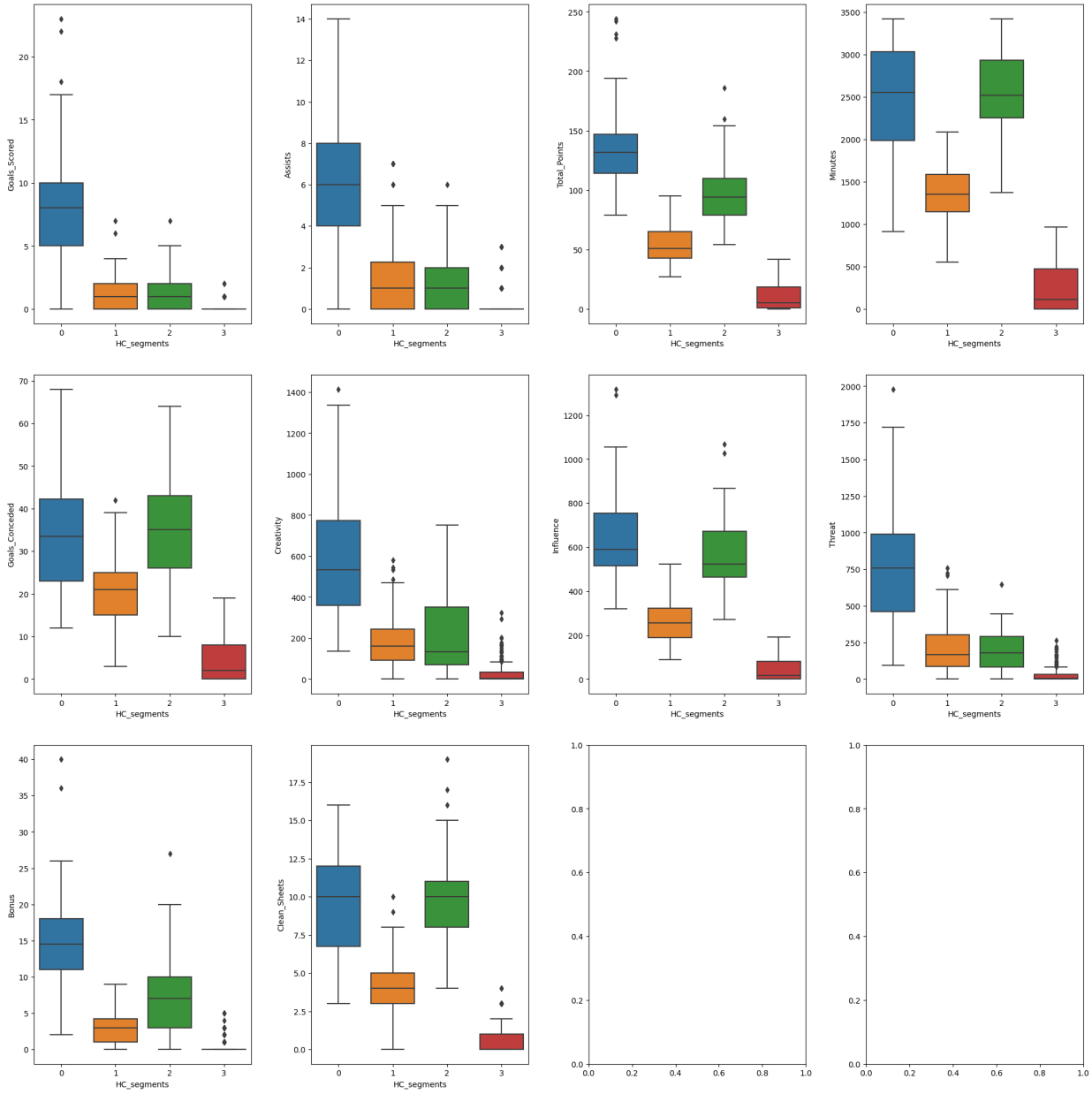
Out[76]:
```
HC_segments  Position
0            Defender      10
             Forward       21
             Midfielder    45
1            Defender      38
             Forward       14
             Goalkeeper     3
             Midfielder    57
2            Defender      52
             Goalkeeper    17
             Midfielder    28
3            Defender      72
             Forward       29
             Goalkeeper    25
             Midfielder    65
Name: Player_Name, dtype: int64
```

Observations:

- 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.

In [77]:
```
fig, axes = plt.subplots(3, 4, figsize = (20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 10:
            sns.boxplot(
                ax = axes[ii][jj],
                data = df3,
                y = df3.columns[3 + counter],
                x = "HC_segments",
            )
            counter = counter + 1

fig.tight_layout(pad = 3.0)
```



Comparison of cluster profiles from Hierarchical and previous algorithms

1. There is a difference in the distribution of each cluster with the algorithms. Two clusters in the Agglomerative algorithm are distributed similarly whereas rest two groups are grouped with a low and high range value.

Characteristics of each cluster

- **Cluster 0**
  - There are 54 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 1**
  - There are 114 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 high influence score, and a moderate threat score.
  - Most of the players in this cluster received low bonus points.
- **Cluster 2**
  - There are 117 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 3**
  - There are 191 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.

Gaussian Mixture Model (GMM) Clustering

- Gaussian Mixture Model (GMM) clustering is a probabilistic clustering method that models the data as a mixture of Gaussian distributions. The algorithm works by estimating the parameters of the Gaussian distributions that best fit the data and then assigning each data point to the Gaussian distribution with the highest probability.
- The steps involved in GMM clustering are as follows:
  - Initialize the parameters of the Gaussian distributions, which include the means, covariances, and mixing coefficients.
  - Compute the probability density function of each data point under each Gaussian distribution.
  - Assign each data point to the Gaussian distribution with the highest probability.
  - Update the parameters of the Gaussian distributions based on the data points assigned to them.
  - Repeat steps 2-4 until the parameters converge.

GMM clustering can be used to identify clusters with arbitrary shapes and sizes, as the Gaussian distributions can model different shapes and orientations. Additionally, GMM clustering can estimate the uncertainty of each data point's assignment to a cluster, which can be useful in applications where the data is noisy or ambiguous.

In [78]:
```
gmm_df = data_pca.copy()
```

In [79]:
```
# Let's apply Gaussian Mixture
gmm = GaussianMixture(n_components = 4, random_state = 1)  # Initializing the Gaussian Mixture algori

gmm.fit(gmm_df)                                            # Fitting the algorithm on the gmm_df data
```

Out[79]:
```
GaussianMixture(n_components=4, random_state=1)
```

Cluster Profiling

In [80]:
```
# Creating a copy of the original data
df4 = df.copy()

# Adding GMM cluster labels to the original and scaled dataframes
gmm_df["GMM_segments"] = gmm.predict(gmm_df)
df4["GMM_segments"] = gmm.predict(data_pca)
```

In [81]:
```
gmm_cluster_profile = df4.groupby("GMM_segments").mean(numeric_only=True)
```

In [82]:
```
gmm_cluster_profile["count_in_each_segment"] = df4.groupby("GMM_segments")["Total_Points"].count().va
```

In [83]:
```
gmm_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[83]:

GMM_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	
0	1.189655	1.344828	94.439655	2453.189655	33.793103	221.881897	535.862069	174.991379	6
1	0.000000	0.000000	3.547619	107.436508	2.055556	8.590476	15.739683	5.746032	0
2	6.552381	5.390476	114.876190	2104.809524	29.619048	501.710476	521.451429	683.466667	11
3	0.635659	0.868217	34.031008	908.968992	14.186047	106.855039	165.444961	110.813953	1

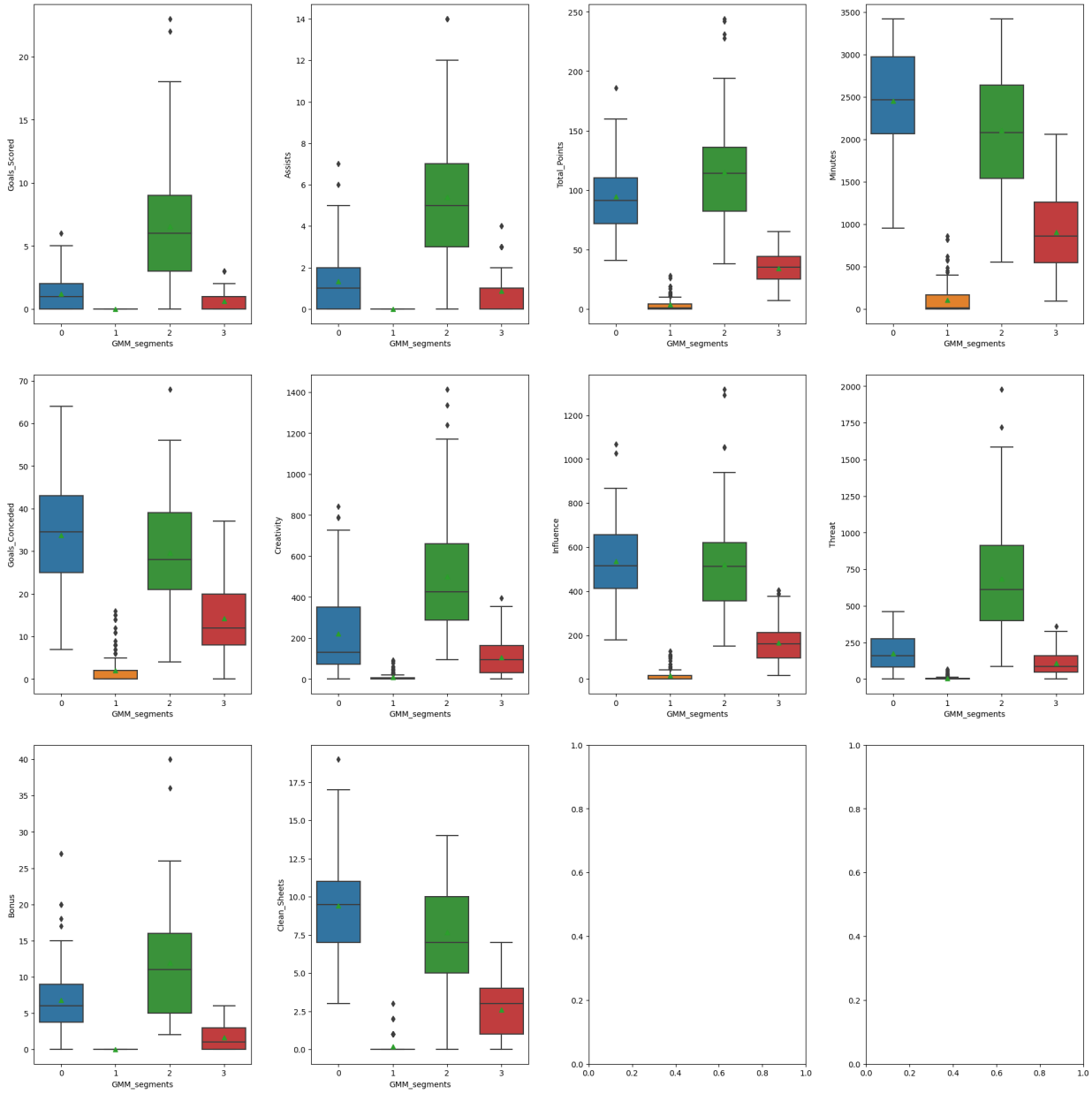
In [84]:
```
# Let's see the names of the players in each cluster
for cl in df4["GMM_segments"].unique():
    print("In cluster {}, the following players are present:".format(cl))
    print(df4[df4["GMM_segments"] == cl]["Player_Name"].unique())
    print()
```

```
In cluster 1, the following players are present:
['Alex Runnarsson' 'Matt Macey' 'Miguel Azeez' 'Reiss Nelson'
 'Sead Kolasinac' 'Shkodran Mustafi' 'Sokratis Papastathopoulos'
 'William Saliba' 'Carney Chukwuemeka' 'Henri Lansbury'
 'Jaden Philogene-Bidace' 'Jose Peleteiro Romallo' 'Morgan Sanson'
 'Orjan Nyland' 'Wesley Moraes' 'Bernardo Fernandes da Silva Junior'
 'Davy Propper' 'Jason Steele' 'Jayson Molumby' 'Jose Izquierdo'
 'Reda Khadra' 'Anthony Driscoll-Glennon' 'Bailey Peacock-Farrell'
 'Dale Stephens' 'Joel Mumbongo' 'Josh Benson' 'Lewis Richardson'
 'Phil Bardsley' 'Will Norris' 'Billy Gilmour'
 'Emerson Palmieri dos Santos' 'Faustino Anjorin' 'Fikayo Tomori'
 'Karlo Ziger' 'Valentino Livramento' 'Willy Caballero' 'Connor Wickham'
 'Jack Butland' 'James McCarthy' 'Mamadou Sakho' 'Martin Kelly'
 'Nathan Ferguson' 'Reece Hannam' 'Ryan Inniss' 'Sam Woods'
 'Stephen Henderson' 'Anthony Gordon' 'Cenk Tosun' 'Fabian Delph'
 'Joao Virginia' 'Jonjoe Kenny' 'Joshua King' 'Moise Kean'
 'Nathan Broadhead' 'Niels Nkounkou' 'Adam Forshaw' 'Jack Jenkins'
 'Jamie Shackleton' 'Jay-Roy Grot' 'Jordan Stevens' 'Kamil Miazek'
 'Leif Davis' 'Mateusz Bogusz' 'Niall Huggins' 'Demarai Gray'
```

In [85]:
```
df4.groupby(["GMM_segments", "Position"])['Player_Name'].count()
```

Out[85]:
```
GMM_segments  Position
0             Defender      64
              Goalkeeper    18
              Midfielder    34
1             Defender      48
              Forward       21
              Goalkeeper    19
              Midfielder    38
2             Defender      10
              Forward       32
              Midfielder    63
3             Defender      50
              Forward       11
              Goalkeeper     8
              Midfielder    60
Name: Player_Name, dtype: int64
```

In [86]:
```
fig, axes = plt.subplots(3, 4, figsize = (20, 20))
counter = 0

for ii in range(3):
    for jj in range(4):
        if counter < 10:
            sns.boxplot(
                ax = axes[ii][jj],
                data = df4,
                y = df4.columns[3 + counter],
                x = "GMM_segments",showmeans = True
            )
            counter = counter + 1

fig.tight_layout(pad = 3.0)
```



Comparison of cluster profiles from GMM and previous algorithms

1. There is a difference in the distribution of each cluster with the other algorithms. Every cluster in GMM is distributed more or less similarly with the same size.

Characteristics of each cluster

- **Cluster 0**
  - There are 116 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 very high game time, high influence, and moderate creativity scores.
  - Most of the players in this cluster received moderate bonus points.
  - Most of the players in this cluster received high clean sheets with an average of 9.43.
- **Cluster 1**
  - There are 126 players in this cluster.
  - The players in this cluster haven't scored goals and assists and the total fantasy points scored in the previous season are very low.
  - Most of the players in this cluster had a very low game time, and very low creativity, influence, and threat scores.
  - The players in this cluster haven't received any bonus points.
- **Cluster 2**
  - There are 105 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 high game time, a moderate influence score, a high creativity score, and a moderate threat score.
  - Most of the players in this cluster received the highest bonus points.
- **Cluster 3**
  - There are 129 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 moderate game time, low creativity, influence, and threat scores.
  - Most of the players in this cluster received low bonus points.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

- DBSCAN is a clustering algorithm that groups together points that are closely packed together while marking points that are not part of any cluster as noise. The algorithm works by defining a neighborhood around each point and then determining whether the point is a core point, a border point, or a noise point based on the density of points within the neighborhood.
- The steps involved in DBSCAN are as follows:
  - Define a neighborhood around each point using a distance metric and a radius.
  - Identify the core points as those that have at least a minimum number of points within their neighborhood.
  - Connect the core points to their directly reachable neighbors to form clusters.
  - Identify the border points as those that are not core points but belong to a cluster.
  - Assign the noise points to their own separate cluster.

DBSCAN can be useful for datasets with irregular shapes or where the number of clusters is not known a priori. It can also handle noisy data by assigning it to its own separate cluster. Additionally, DBSCAN does not require the specification of the number of clusters or assumptions about the shape of the clusters.

It is a very powerful algorithm for finding high-density clusters, but the problem is determining the best set of hyperparameters to use with it. It includes two hyperparameters, eps, and min samples. Since it is an unsupervised algorithm, you have no control over it, unlike a supervised learning algorithm, which allows you to test your algorithm on a validation set. The approach we can follow is trying out a bunch of different combinations of values and finding the silhouette score for each of them.

What is the silhouette score?

Silhouette score is one of the methods for evaluating the quality of clusters created using clustering algorithms such as K-Means. The silhouette score is a measure of how similar an object is to its cluster (cohesion) compared to other clusters (separation). Silhouette score has a range of [-1, 1].

- Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters.
- Silhouette score near -1 indicates that those samples might have been assigned to the wrong cluster.

In [87]:
```
dbscan_df = data_pca.copy()
```

In [88]:
```
# Initializing lists
eps_value = [2,3]           # Taking random eps value
min_sample_values = [6,20]  # Taking random min_sample value

# Creating a dictionary for each of the values in eps_value with min_sample_values
res = {eps_value[i]: min_sample_values for i in range(len(eps_value))}
```

In [89]:
```
# Finding the silhouette_score for each of the combination

high_silhouette_avg = 0                                                 # Assigning 0 to the high_silho
high_i_j = [0, 0]                                                       # Assigning 0's to the high_i_j
key = res.keys()                                                        # Assigning dictionary keys to
for i in key:
    z = res[i]                                                          # Assigning dictionary values o
    for j in z:
        db = DBSCAN(eps = i, min_samples = j).fit(dbscan_df)            # Applying DBSCAN to each of th
        core_samples_mask = np.zeros_like(db.labels_, dtype = bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = db.labels_
        silhouette_avg = silhouette_score(dbscan_df, labels)            # Finding silhouette score
        print(
            "For eps value =" + str(i),
            "For min sample =" + str(j),
            "The average silhoutte_score is :",
            silhouette_avg,                                            # Printing the silhouette score
        )
        if high_silhouette_avg < silhouette_avg:                        # If the silhouette score is gre
            high_i_j[0] = i
            high_i_j[1] = j
```

```
For eps value =2 For min sample =6 The average silhoutte_score is : 0.5283008912823889
For eps value =2 For min sample =20 The average silhoutte_score is : 0.3647818751696756
For eps value =3 For min sample =6 The average silhoutte_score is : 0.624205189855851
For eps value =3 For min sample =20 The average silhoutte_score is : 0.6188492416303977
```

In [90]:
```
# Printing the highest silhouette score
print(
    "Highest_silhoutte_avg is {} for eps = {} and min sample = {}".format(
        high_silhouette_avg, high_i_j[0], high_i_j[1]
    )
)
```

```
Highest_silhoutte_avg is 0 for eps = 3 and min sample = 20
```

In [91]:
```
# Applying DBSCAN with eps as 3 and min sample as 20
dbs = DBSCAN(eps = 3, min_samples = 20)
```

In [92]:
```
# Creating a copy of the original data
df5 = df.copy()

# Add DBSCAN cluster labels to whole data
df5["db_segments"] =  dbs.fit_predict(dbscan_df)
```

In [93]:
```
db_cluster_profile = df5.groupby("db_segments").mean(numeric_only=True)
```

In [94]:
```
db_cluster_profile["count_in_each_segment"] = (
    df5.groupby("db_segments")["Total_Points"].count().values
)
```

In [95]:
```
db_cluster_profile.style.highlight_max(color = "lightgreen", axis = 0)
```

Out[95]:

db_segments	Goals_Scored	Assists	Total_Points	Minutes	Goals_Conceded	Creativity	Influence	Threat	
-1	18.666667	11.666667	221.000000	3045.000000	41.000000	779.533333	1067.566667	1447.000000	2
0	1.693617	1.625532	56.442553	1315.104255	18.878723	188.526383	284.750213	209.361702	

Observations:

- DBSCAN was able to give only two clusters with eps = 3 and min_sample = 20 which is very skewed.
- It is not able to perform well on this dataset.

Choosing the Best Algorithm

- Since cluster profiles are the same for every algorithm except DBSCAN, it is difficult to choose the best algorithm. We can compute the silhouette score to choose the best algorithm among all the algorithms.

In [96]:
```
# Initializing K-Means with number of clusters as 4 and random_state=1
kmeans = KMeans(n_clusters=4, random_state=1, n_init='auto')
preds = kmeans.fit_predict((data_pca))                    # Fitting and predicting K-Means on data_pca
score = silhouette_score(data_pca, preds)                 # Calculating the silhouette score

print(score)
```

```
0.40411092686635713
```

In [97]:
```
kmedoids = KMedoids(n_clusters = 4, random_state = 1)   # Initializing K-Medoids with number of clust
preds = kmedoids.fit_predict((data_pca))                # Fitting and predicting K-Medoids on data_pc
score = silhouette_score(data_pca, preds)               # Calculating the silhouette score
print(score)
```

```
0.393822499693573
```

In [98]:
```
# Initializing Agglomerative Clustering with distance as Euclidean, linkage as ward with clusters = 4
HCmodel = AgglomerativeClustering(n_clusters = 4, metric = "euclidean", linkage = "ward")

# Fitting on PCA data
preds = HCmodel.fit_predict(data_pca)
score = silhouette_score(data_pca, preds)             # Calculating the silhouette score
print(score)
```

```
0.3849709986025467
```

In [99]:
```
# Initializing Gaussian Mixture algorithm with number of clusters as 4 and random_state = 1
gmm = GaussianMixture(n_components=4, random_state=1)

# Fitting and predicting Gaussian Mixture algorithm on data_pca
preds = gmm.fit_predict((data_pca))

# Calculating the silhouette score
score = silhouette_score(data_pca, preds)

# Printing the score
print(score)
```

```
0.28494644297302146
```

Observations:

- Based on the silhouette score, we can see that K-Means algorithm giving the best score on the data. We would proceed K-Means as the best algorithm.

Conclusion:

- The players who have a greater influence on the outcome of the game typically play for a longer duration every game and score more fantasy points.
- The players can be sold for more money who have higher goals scored, creativity, and influence.
- Since there is a drop at K = 4 in the elbow plot, we selected K as 4 for clustering.
- We implemented five algorithms, but we have chosen K-Means algorithm as the final algorithm because it has the highest silhouette score of 0.40.

Recommendations:

- **Cluster 0** players are the top players for fantasy. They fetch more points and have a higher chance of getting bonus points too. These players should be priced higher than the others so that it will be difficult to accommodate too many of them in the same team (because of the fixed budget) and fantasy managers have to make wise choices.
- **Cluster 1** players are players who do not play many minutes, most likely come on as substitutes and fetch lesser fantasy points as a result. These players should be priced low and can be good differential picks.
- **Cluster 2** are the players who are influential in their team's play but do not tend to score or assist much, resulting in lesser fantasy points than the Cluster 0 players. These players should be priced somewhere between Cluster 0 and Cluster 1 players.
- **Cluster 3** has the players who are in the squad to provide backup in case any of the starting 11 players get injured. They get lower game time and barely get any fantasy points. These players should be priced the lowest among the 4 clusters.
- Player performances from previous seasons should be taken into account and fantasy prices from the previous season should be referred to as a benchmark to determine the price for the upcoming season.
- OnSports should conduct cluster analysis separately for each of the playing positions to arrive at a better fantasy pricing strategy.