Data-Analytics-Coursework-1- Sayed Sohail (220541549)

Clustering of Football Players Similar to 'Kylian Mbappé' using K-means and GMM Algorithms

Introduction

The aim of this notebook is to cluster the aggregated football performance data of Top 5 European Leagues (Premier League in England, the Bundesliga in Germany, La Liga in Spain, Serie A in Italy, and Ligue 1 in France) across 2018-19 to 2022-2023 (5 seasons) to find players similar to 'Kylian Mbappé'. The aggregated player performance data has been obtained from FBref (https://fbref.com/en/). The notebook uses Pandas for data manipulation through DataFrames, Matplotlib abdn Seaborn for data visualtisations, Scikit-Learn for Machine Learning (K-means and GMM).

Index

1. Dependencies

Import Libraries and Modules

In [4]:
```
# Python ≥3.5 (ideally)
import platform
import sys, getopt
assert sys.version_info >= (3, 5)
import csv

!{sys.executable} -m pip install html5lib
# Import Dependencies
%matplotlib inline

# Math Operations
import numpy as np
from math import pi

# Datetime
import datetime
from datetime import date
import time

# Data Preprocessing
import pandas as pd
import ydata_profiling as pp
import os
import re
import random
import glob
from io import BytesIO
from pathlib import Path

# Reading directories
```

```
import glob
import os

# Working with JSON
import json
from pandas.io.json import json_normalize

# Web Scraping
import requests
from bs4 import BeautifulSoup
import re


# Machine Learning
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture as GMM
from sklearn.metrics import silhouette_score,silhouette_samples
from sklearn.metrics import davies_bouldin_score,calinski_harabasz_score

# Data Visualisation
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns
plt.style.use('seaborn-whitegrid')
import missingno as msno

# Progress Bar
from tqdm import tqdm

# Display in Jupyter
from IPython.display import Image, YouTubeVideo
from IPython.core.display import HTML

# Ignore Warnings
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")


from soccerplots.radar_chart import Radar   # for custom radar visuals
print('Setup Complete')
```

```
Requirement already satisfied: html5lib in /Users/sayedsohail/opt/anaconda3/
lib/python3.9/site-packages (1.1)
Requirement already satisfied: webencodings in /Users/sayedsohail/opt/anacon
da3/lib/python3.9/site-packages (from html5lib) (0.5.1)
Requirement already satisfied: six>=1.9 in /Users/sayedsohail/opt/anaconda3/
lib/python3.9/site-packages (from html5lib) (1.16.0)
Setup Complete
```

Defining Filepaths and Variables

In [5]:

```
# Set up initial paths to subfolders
# base_dir = os.path.join('..', '..')
base_dir = os.getcwd()
data_dir = os.path.join(base_dir, 'data')
data_dir_fbref = os.path.join(base_dir, 'data', 'fbref')
img_dir = os.path.join(base_dir, 'img')
fig_dir = os.path.join(base_dir, 'img', 'fig')
video_dir = os.path.join(base_dir, 'video')
```

```
today = datetime.datetime.now().strftime('%d/%m/%Y').replace('/', '')
print(data_dir_fbref)
```

```
/Users/sayedsohail/QMUL/SEM-B/Data Analytics/Project/Coursework1/data/fbref
```

2. Data Scraping

The data needed for analysing aggregated player performance has been scraped from FBref (https://fbref.com/). The FBref dataset has over 5,264 outfield players for the last five seasons for the 'Big 5' European leagues.

Selecting the various performance stats to be scraped from the FBref Web Pages

In [6]:
```
#@title

#standard(stats)
stats = ["player","nationality","position","team","comp_level","age","birth_
stats3 = ["players_used","possession","games","games_starts","minutes","goal
#goalkeeping(keepers)
keepers = ["player","nationality","position","squad","age","birth_year","gam
keepers3 = ["players_used","games_gk","games_starts_gk","minutes_gk","goals_
#advance goalkeeping(keepersadv)
keepersadv = ["player","nationality","position","squad","age","birth_year","
keepersadv2 = ["minutes_90s","goals_against_gk","pens_allowed","free_kick_go
#shooting(shooting)
shooting = ["player","nationality","position","squad","age","birth_year","mi
shooting2 = ["minutes_90s","goals","pens_made","pens_att","shots_total","sho
shooting3 = ["goals","pens_made","pens_att","shots_total","shots_on_target",
#passing(passing)
passing = ["player","nationality","position","squad","age","birth_year","min
passing2 = ["passes_completed","passes","passes_pct","passes_total_distance"
#passtypes(passing_types)
passing_types = ["player","nationality","position","squad","age","birth_year
passing_types2 = ["passes","passes_live","passes_dead","passes_free_kicks","
#goal and shot creation(gca)
gca = ["player","nationality","position","squad","age","birth_year","minutes
gca2 = ["sca","sca_per90","sca_passes_live","sca_passes_dead","sca_dribbles"
#defensive actions(defense)
defense = ["player","nationality","position","squad","age","birth_year","min
defense2 = ["tackles","tackles_won","tackles_def_3rd","tackles_mid_3rd","tac
#possession(possession)
possession = ["player","nationality","position","squad","age","birth_year","
possession2 = ["touches","touches_def_pen_area","touches_def_3rd","touches_m
#playingtime(playingtime)
playingtime = ["player","nationality","position","squad","age","birth_year",
playingtime2 = ["games","minutes","minutes_per_game","minutes_pct","games_st
#miscallaneous(misc)
misc = ["player","nationality","position","squad","age","birth_year","minute
misc2 = ["cards_yellow","cards_red","cards_yellow_red","fouls","fouled","off
```

Functions to get the data in a dataframe

In [68]:
```
#Functions to get the data in a dataframe using BeautifulSoup

def get_tables(url,text):
    print(url)
    res = requests.get(url)
```

```
    ## The next two lines get around the issue with comments breaking the pa
    comm = re.compile("<!--|-->")
    soup = BeautifulSoup(comm.sub("",res.text),'lxml')
    all_tables = soup.findAll("tbody")

#     team_table = all_tables[0]
#     print(len(all_tables))
#     team_vs_table = all_tables[0]
    player_table = all_tables[1]
#     if text == 'for':
#       return player_table, team_table
#     if text == 'vs':
#       return player_table, team_vs_table
    return player_table

def get_frame(features, player_table, season):
    pre_df_player = dict()
    features_wanted_player = features
    rows_player = player_table.find_all('tr')
    for row in rows_player:
        if(row.find('th',{"scope":"row"}) != None):
            for f in features_wanted_player:
                if(row.find("td",{"data-stat": f}) != None):
                    cell = row.find("td",{"data-stat": f})
                    a = cell.text.strip().encode()
                    text=a.decode("utf-8")
                    if(text == ''):
                        text = '0'
                    if((f!='player')&(f!='nationality')&(f!='position')&(f!=
                        text = float(text.replace(',',''))
                    if f in pre_df_player:
                        pre_df_player[f].append(text)
                    else:
                        pre_df_player[f] = [text]
    pre_df_player['Season'] = season
    df_player = pd.DataFrame.from_dict(pre_df_player)
    return df_player

def get_frame_team(features, team_table):
    pre_df_squad = dict()
    #Note: features does not contain squad name, it requires special treatme
    features_wanted_squad = features
    rows_squad = team_table.find_all('tr')
    for row in rows_squad:
        if(row.find('th',{"scope":"row"}) != None):
            name = row.find('th',{"data-stat":"squad"}).text.strip().encode(
            if 'squad' in pre_df_squad:
                pre_df_squad['squad'].append(name)
            else:
                pre_df_squad['squad'] = [name]
            for f in features_wanted_squad:
                cell = row.find("td",{"data-stat": f})
                a = cell.text.strip().encode()
                text=a.decode("utf-8")
                if(text == ''):
                    text = '0'
                if((f!='player')&(f!='nationality')&(f!='position')&(f!='squ
                    text = float(text.replace(',',''))
                if f in pre_df_squad:
                    pre_df_squad[f].append(text)
                else:
                    pre_df_squad[f] = [text]
    df_squad = pd.DataFrame.from_dict(pre_df_squad)
    return df_squad
```

```
def frame_for_category(category,top,end,features,season):
    url = (top + category + end)
    player_table = get_tables(url,'for')
    df_player = get_frame(features, player_table, season)

    return df_player



#Function to get the player data for outfield player, includes all categorie
#passing, passing types, goal and shot creation, defensive actions, possessi
def get_outfield_data(top, end, season):
    df1 = frame_for_category('stats',top,end,stats,season)
    df2 = frame_for_category('shooting',top,end,shooting2,season)
    df3 = frame_for_category('passing',top,end,passing2,season)
    df4 = frame_for_category('passing_types',top,end,passing_types2,season)
    df5 = frame_for_category('gca',top,end,gca2,season)
    df6 = frame_for_category('defense',top,end,defense2,season)
    df7 = frame_for_category('possession',top,end,possession2,season)
    df8 = frame_for_category('misc',top,end,misc2,season)
    df = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8], axis=1)
    df = df.loc[:,~df.columns.duplicated()]
    return df


#Function to get keeping and advance goalkeeping data
def get_keeper_data(top,end):
    df1 = frame_for_category('keepers',top,end,keepers)
    df2 = frame_for_category('keepersadv',top,end,keepersadv2)
    df = pd.concat([df1, df2], axis=1)
    df = df.loc[:,~df.columns.duplicated()]
    return df



#Function to get team-wise data accross all categories as mentioned above
def get_team_data(top,end,text):
    df1 = frame_for_category_team('stats',top,end,stats3,text)
    df2 = frame_for_category_team('keepers',top,end,keepers3,text)
    df3 = frame_for_category_team('keepersadv',top,end,keepersadv2,text)
    df4 = frame_for_category_team('shooting',top,end,shooting3,text)
    df5 = frame_for_category_team('passing',top,end,passing2,text)
    df6 = frame_for_category_team('passing_types',top,end,passing_types2,tex
    df7 = frame_for_category_team('gca',top,end,gca2,text)
    df8 = frame_for_category_team('defense',top,end,defense2,text)
    df9 = frame_for_category_team('possession',top,end,possession2,text)
    df10 = frame_for_category_team('misc',top,end,misc2,text)
    df = pd.concat([df1, df2, df3, df4, df5, df6, df7, df8, df9, df10], axis
    df = df.loc[:,~df.columns.duplicated()]
    return df
```

Web Scraping and creating a Data Frame

lst_seasons = ['2018-2019', '2019-2020', '2020-2021', '2021-2022', '2022-2023'] for season in lst_seasons: df_outfield = get_outfield_data('https://fbref.com/en/comps/Big5/'+season+'/','/players/'+season+'-Big-5-European-Leagues-Stats', season) filename = season+"_Outfield.csv" print("Saving file", filename) df_outfield.to_csv(filename,index=False) print("Done Downloading....")

Unifying individual CSV files into a single DataFrame

In [7]:
```
## Unify individual CSV files as a single DataFrame
### Show files in directory
all_files = glob.glob(os.path.join(data_dir_fbref + f'/raw/*.csv'))
print(all_files)
lst_player_stats_all = []
for filename in all_files:
        df_temp = pd.read_csv(filename, index_col=None, header=0)
        lst_player_stats_all.append(df_temp)
### Concatenate the files into a single DataFrame
df_fbref_player_stats_all = pd.concat(lst_player_stats_all, axis=0, ignore_i
df_fbref_player_stats_all = df_fbref_player_stats_all.sort_values(['comp_lev
df_fbref_player_stats_all.to_csv(data_dir_fbref + f'/raw/fbref_outfield_play
total_players = df_fbref_player_stats_all['player'].nunique()
print(f'Player stats DataFrame contains {total_players} players.')
```

```
['/Users/sayedsohail/QMUL/SEM-B/Data Analytics/Project/Coursework1/data/fbre
f/raw/fbref_outfield_player_stats_combined_latest_10032023.csv', '/Users/say
edsohail/QMUL/SEM-B/Data Analytics/Project/Coursework1/data/fbref/raw/2018-2
019_Outfield.csv', '/Users/sayedsohail/QMUL/SEM-B/Data Analytics/Project/Cou
rsework1/data/fbref/raw/2020-2021_Outfield.csv', '/Users/sayedsohail/QMUL/SE
M-B/Data Analytics/Project/Coursework1/data/fbref/raw/2021-2022_Outfield.cs
v', '/Users/sayedsohail/QMUL/SEM-B/Data Analytics/Project/Coursework1/data/f
bref/raw/2019-2020_Outfield.csv', '/Users/sayedsohail/QMUL/SEM-B/Data Analyt
ics/Project/Coursework1/data/fbref/raw/2022-2023_Outfield.csv']
/var/folders/nv/yhx12vpj1vz672s8q9xw9ylw0000gn/T/ipykernel_1631/505282526.p
y:7: DtypeWarning: Columns (5) have mixed types. Specify dtype option on imp
ort or set low_memory=False.
  df_temp = pd.read_csv(filename, index_col=None, header=0)
Player stats DataFrame contains 5266 players.
```

3. Exploratory Data Analysis

3.1 Creating Profiler Report for the scraped data

Creating a summary report of dataset using Pandas Profiling Report.

In [11]:
```
df_fbref_outfield_raw = df_fbref_player_stats_all
outfield_raw_profile = pp.ProfileReport(df_fbref_player_stats_all, minimal=T
```

In [12]:
```
outfield_raw_profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	117
Number of observations	83280
Missing cells	0
Missing cells (%)	0.0%
Total size in memory	77.0 MiB
Average record size in memory	969.4 B

Variable types

Categorical	6
Unsupported	1
Numeric	110

Alerts

`player` has a high cardinality: 5266 distinct values	**High cardinality**
`nationality` has a high cardinality: 124 distinct values	**High cardinality**
`team` has a high cardinality: 135 distinct values	**High cardinality**
`assists_per90` is highly skewed (γ1 = 35.27273406)	**Skewed**
`xg_per90` is highly skewed (γ1 = 31.07236083)	**Skewed**

Out[12]:

In [13]:
```
outfield_raw_profile.to_file('outfile_raw_profile.html')
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

3.2 Checking Quality of dataset using head() and tail() methods

In [71]:
```
df_fbref_outfield_raw.head()
```

Out[71]:

	player	nationality	position	team	comp_level	age	birth_year	games	games_
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 117 columns

In [72]:
```
df_fbref_outfield_raw.tail()
```

Out[72]:

	player	nationality	position	team	comp_level	age	birth_year	games	gam
12310	Zlatan Ibrahimović	se SWE	FW	Milan	it Serie A	41-158	1981	2.0	
11811	Ángel Di María	ar ARG	FW,MF	Juventus	it Serie A	35-024	1988	16.0	
11917	Éderson	br BRA	MF	Atalanta	it Serie A	23-246	1999	22.0	
12371	Þórir Jóhann Helgason	is ISL	MF	Lecce	it Serie A	22-163	2000	8.0	
13477	Łukasz Skorupski	pl POL	GK	Bologna	it Serie A	31-309	1991	25.0	

5 rows × 117 columns

3.3 Checking Shape returns a tuple representing the dimensionality of the DataFrame

In [73]:
```
# Print the shape of the raw DataFrame, df_fbref_outfield_raw
print(df_fbref_outfield_raw.shape)
```

```
(13880, 117)
```

3.4 Identifying Column Names of the Data Frame

In [74]:
```
# Features (column names) of the raw DataFrame, df_fbref_outfield_raw
df_fbref_outfield_raw.columns
```

Out[74]:
```
Index(['player', 'nationality', 'position', 'team', 'comp_level', 'age',
       'birth_year', 'games', 'games_starts', 'minutes',
       ...
       'fouls', 'fouled', 'offsides', 'pens_won', 'pens_conceded', 'own_goal
s',
       'ball_recoveries', 'aerials_won', 'aerials_lost', 'aerials_won_pct'],
      dtype='object', length=117)
```

In [14]:
```
df_fbref_outfield_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83280 entries, 0 to 82877
Columns: 117 entries, player to aerials_won_pct
dtypes: float64(109), int64(1), object(7)
memory usage: 77.0+ MB
```

3.5 Identifying Data Types of Columns of the Data Frame

In [75]:
```
# Displays all columns
with pd.option_context('display.max_rows', None, 'display.max_columns', None
    print(df_fbref_outfield_raw.dtypes)
```

```
player                        object
nationality                   object
position                      object
team                          object
comp_level                    object
age                           object
birth_year                     int64
games                       float64
games_starts                float64
minutes                     float64
goals                       float64
assists                     float64
pens_made                   float64
pens_att                    float64
cards_yellow                float64
cards_red                   float64
goals_per90                 float64
assists_per90               float64
goals_assists_per90         float64
goals_pens_per90            float64
goals_assists_pens_per90    float64
xg                          float64
npxg                        float64
xg_per90                    float64
npxg_per90                  float64
Season                       object
minutes_90s                 float64
shots_on_target             float64
shots_free_kicks            float64
shots_on_target_pct         float64
shots_on_target_per90       float64
goals_per_shot              float64
goals_per_shot_on_target    float64
npxg_per_shot               float64
xg_net                      float64
npxg_net                    float64
passes_completed            float64
passes                      float64
passes_pct                  float64
passes_total_distance       float64
passes_progressive_distance float64
passes_completed_short      float64
passes_short                float64
passes_pct_short            float64
passes_completed_medium     float64
passes_medium               float64
passes_pct_medium           float64
passes_completed_long       float64
passes_long                 float64
passes_pct_long             float64
assisted_shots              float64
passes_into_final_third     float64
passes_into_penalty_area    float64
crosses_into_penalty_area   float64
progressive_passes          float64
passes_live                 float64
passes_dead                 float64
passes_free_kicks           float64
through_balls               float64
passes_switches             float64
crosses                     float64
corner_kicks                float64
corner_kicks_in             float64
corner_kicks_out            float64
```

```
corner_kicks_straight          float64
throw_ins                      float64
passes_offsides                float64
passes_blocked                 float64
sca                            float64
sca_per90                      float64
sca_passes_live                float64
sca_passes_dead                float64
sca_shots                      float64
sca_fouled                     float64
gca                            float64
gca_per90                      float64
gca_passes_live                float64
gca_passes_dead                float64
gca_shots                      float64
gca_fouled                     float64
gca_defense                    float64
tackles                        float64
tackles_won                    float64
tackles_def_3rd                float64
tackles_mid_3rd                float64
tackles_att_3rd                float64
blocks                         float64
blocked_shots                  float64
blocked_passes                 float64
interceptions                  float64
clearances                     float64
errors                         float64
touches                        float64
touches_def_pen_area           float64
touches_def_3rd                float64
touches_mid_3rd                float64
touches_att_3rd                float64
touches_att_pen_area           float64
touches_live_ball              float64
carries                        float64
progressive_carries            float64
carries_into_final_third       float64
carries_into_penalty_area      float64
passes_received                float64
miscontrols                    float64
dispossessed                   float64
cards_yellow_red               float64
fouls                          float64
fouled                         float64
offsides                       float64
pens_won                       float64
pens_conceded                  float64
own_goals                      float64
ball_recoveries                float64
aerials_won                    float64
aerials_lost                   float64
aerials_won_pct                float64
dtype: object
```

3.6 Summary Statistics for each column in the Data Set

The describe method is used to show some useful statistics for each numerical column in the DataFrame.

In [77]:
```
# Description of the raw DataFrame, df_fbref_outfield_raw, showing some summ
df_fbref_outfield_raw.describe()
```

Out[77]:

	birth_year	games	games_starts	minutes	goals	assi
count	13880.000000	13880.000000	13880.000000	13880.000000	13880.000000	13880.0000
mean	1994.386671	17.731772	13.331556	1196.824784	1.632637	1.1349
std	4.742461	11.099607	10.933905	944.788889	3.140143	1.9132
min	1977.000000	1.000000	0.000000	1.000000	0.000000	0.0000
25%	1991.000000	7.000000	3.000000	318.750000	0.000000	0.0000
50%	1995.000000	18.000000	12.000000	1064.000000	0.000000	0.0000
75%	1998.000000	27.000000	22.000000	1919.000000	2.000000	2.0000
max	2007.000000	38.000000	38.000000	3420.000000	41.000000	21.0000

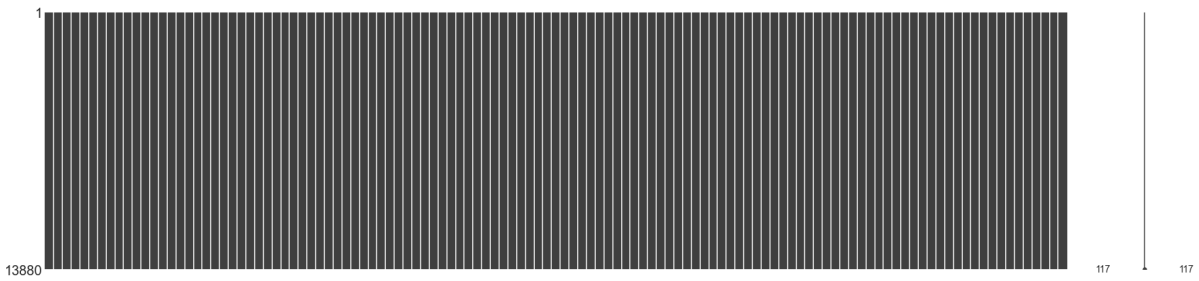
8 rows × 110 columns

3.7 Checking for missing or NULL Values

In [ ]:
```
```

In [78]:
```
# Plot visualisation of the missing values for each feature of the raw DataF
msno.matrix(df_fbref_outfield_raw, figsize = (30, 7))
```

Out[78]:
```
<AxesSubplot:>
```



In [79]:
```
# Counts of missing values
null_value_stats = df_fbref_outfield_raw.isnull().sum(axis=0)
null_value_stats[null_value_stats != 0]
```

Out[79]:
```
Series([], dtype: int64)
```

No values are NULL or missing in the data set.

In [80]:
```
```

4. Data Preprocessing

Assigning Raw DataFrame to New Engineered DataFrame

In [ ]:
```
df_fbref_outfield = df_fbref_outfield_raw
```

4.1 Creating a data frame to store names of all the teams in Top 5 European Leagues

In [ ]:

In [81]:
```
# Create DataFrame of Home and Away teams

## All unique Home and Away teams
lst_teams = list(df_fbref_outfield['team'].unique())


## DataFrames of Home and Away teams
df_teams = pd.DataFrame(lst_teams)


## Export DataFrame
if not os.path.exists(os.path.join(data_dir + '/fbref/reference/team/fbref_t

    ### Save latest version
    df_teams.to_csv(data_dir + '/fbref/reference/team/fbref_teams_big5_lates

    ### Save a copy to archive folder (dated)
    df_teams.to_csv(data_dir + f'/fbref/reference/team/archive/fbref_teams_b

else:
    df_teams = pd.read_csv(data_dir + '/fbref/reference/team/fbref_teams_big
    print('Data already saved previously')
```

In [82]:
```
df_teams.head()
```

Out[82]:

	0
0	Mainz 05
1	Dortmund
2	Düsseldorf
3	Nürnberg
4	Wolfsburg

4.2 Removing Special Characters and Lower Casing Player Names

In [83]:
```
# Remove accents and create lowercase name
df_fbref_outfield['Player Lower'] = (df_fbref_outfield['player']
                                        .str.normalize('NFKD')
                                        .str.encode('ascii', errors='ignore
                                        .str.decode('utf-8')
                                        .str.lower()
                                    )
```

In [84]:
```
# First Name Lower
df_fbref_outfield['First Name Lower'] = df_fbref_outfield['Player Lower'].st

# Last Name Lower
df_fbref_outfield['Last Name Lower'] = df_fbref_outfield['Player Lower'].str
```

```
# First Initial Lower
df_fbref_outfield['First Initial Lower'] = df_fbref_outfield['Player Lower']
```

In [85]:
```
df_fbref_outfield.head()
```

Out[85]:

	player	nationality	position	team	comp_level	age	birth_year	games	games_
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 121 columns

4.3 String Cleaning: Mapping Countries obtained from CSV to Player Nationality in the dataset

In [87]:
```
# Import reference CSV of country names and codes used to map to countries a
df_countries = pd.read_csv(data_dir + '/fbref/reference/countries/countries_

df_countries
```

Out[87]:

	ID	Full Country Name	FIFA Code	IOC Code	ISO Code
0	1	Afghanistan	AFG	AFG	AFG
1	2	Åland Islands	ALA	NaN	ALA
2	3	Albania	ALB	ALB	ALB
3	4	Algeria	ALG	ALG	DZA
4	5	American Samoa	ASA	ASA	ASM
...	...	...	...	...	...
249	250	Wallis and Futuna	WLF	NaN	WLF
250	251	Western Sahara	ESH	NaN	ESH
251	252	Yemen	YEM	YEM	YEM
252	253	Zambia	ZAM	ZAM	ZMB
253	254	Zimbabwe	ZIM	ZIM	ZWE

254 rows × 5 columns

In [90]:
```
df_fbref_outfield['Nationality Code'] = df_fbref_outfield['nationality'].str
df_fbref_outfield.head()
```

Out[90]:

	player	nationality	position	team	comp_level	age	birth_year	games	games_
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 122 columns

In [91]:
```
dict(zip(df_countries['FIFA Code'], df_countries['Full Country Name']))
pd.Series(df_countries['FIFA Code'].values,index=df_countries['Full Country
dict_countries = df_countries.set_index('FIFA Code').to_dict()['Full Country
dict_countries
```

Out[91]:

```
{'AFG': 'Afghanistan',
 'ALA': 'Åland Islands',
 'ALB': 'Albania',
 'ALG': 'Algeria',
 'ASA': 'American Samoa',
 'AND': 'Andorra',
 'ANG': 'Angola',
 'AIA': 'Anguilla',
 'ATA': 'Antarctica',
 'ATG': 'Antigua and Barbuda',
 'ARG': 'Argentina',
 'ARM': 'Armenia',
 'ARU': 'Aruba',
 'AUS': 'Australia',
 'AUT': 'Austria',
 'AZE': 'Azerbaijan',
 'BAH': 'The Bahamas',
 'BHR': 'Bahrain',
 'BAN': 'Bangladesh',
 'BRB': 'Barbados',
 'BLR': 'Belarus',
 'BEL': 'Belgium',
 'BLZ': 'Belize',
 'BEN': 'Benin',
 'BER': 'Bermuda',
 'BHU': 'Bhutan',
 'BOL': 'Bolivia',
 'BES': 'Caribbean Netherlands: Bonaire, Sint Eustatius and Saba',
 'BIH': 'Bosnia and Herzegovina',
 'BOT': 'Botswana',
 'BVT': 'Bouvet Island',
 'BRA': 'Brazil',
 'IOT': 'British Indian Ocean Territory',
 'VGB': 'British Virgin Islands',
 'BRU': 'Brunei',
 'BUL': 'Bulgaria',
 'BFA': 'Burkina Faso',
 'BDI': 'Burundi',
 'CAM': 'Cambodia',
 'CMR': 'Cameroon',
 'CAN': 'Canada',
 'CPV': 'Cape Verde',
 'CAY': 'Cayman Islands',
 'CTA': 'Central African Republic',
 'CHA': 'Chad',
 'CHI': 'Chile',
 'CHN': 'China',
 'CXR': 'Christmas Island',
 'CCK': 'Cocos (Keeling) Islands',
 'COL': 'Colombia',
 'COM': 'Comoros',
 'COD': 'Democratic Republic of the Congo',
 'CGO': 'Republic of the Congo',
 'COK': 'Cook Islands',
 'CRC': 'Costa Rica',
 'CIV': "Côte d'Ivoire",
 'CRO': 'Croatia',
 'CUB': 'Cuba',
 'CUW': 'Curaçao',
 'CYP': 'Cyprus',
 'CZE': 'Czech Republic',
 'DEN': 'Denmark',
 'DJI': 'Djibouti',
 'DMA': 'Dominica',
```

```
'DOM': 'Dominican Republic',
'ECU': 'Ecuador',
'EGY': 'Egypt',
'SLV': 'El Salvador',
'ENG': 'England',
'EQG': 'Equatorial Guinea',
'ERI': 'Eritrea',
'EST': 'Estonia',
'SWZ': 'Eswatini',
'ETH': 'Ethiopia',
'FLK': 'Falkland Islands',
'FRO': 'Faroe Islands',
'FIJ': 'Fiji',
'FIN': 'Finland',
'FRA': 'France',
'GUF': 'French Guiana',
'TAH': 'French Polynesia',
'ATF': 'French Southern and Antarctic Lands',
'GAB': 'Gabon',
'GAM': 'Gambia',
'GEO': 'Georgia',
'GER': 'Germany',
'GHA': 'Ghana',
'GIB': 'Gibraltar',
'GRE': 'Greece',
'GRL': 'Greenland',
'GRN': 'Grenada',
'GPE': 'Guadeloupe',
'GUM': 'Guam',
'GUA': 'Guatemala',
'GGY': 'Guernsey',
'GUI': 'Guinea',
'GNB': 'Guinea-Bissau',
'GUY': 'Guyana',
'HAI': 'Haiti',
'HMD': 'Heard Island and McDonald Islands',
'HON': 'Honduras',
'HKG': 'Hong Kong',
'HUN': 'Hungary',
'ISL': 'Iceland',
'IND': 'India',
'IDN': 'Indonesia',
'IRN': 'Iran',
'IRQ': 'Iraq',
'IRL': 'Ireland',
'IMN': 'Isle of Man',
'ISR': 'Israel',
'ITA': 'Italy',
'JAM': 'Jamaica',
'JPN': 'Japan',
'JEY': 'Jersey',
'JOR': 'Jordan',
'KAZ': 'Kazakhstan',
'KEN': 'Kenya',
'KIR': 'Kiribati',
'PRK': 'North Korea',
'KOR': 'South Korea',
'KVX': 'Kosovo',
'KUW': 'Kuwait',
'KGZ': 'Kyrgyzstan',
'LAO': 'Laos',
'LVA': 'Latvia',
'LBN': 'Lebanon',
'LES': 'Lesotho',
```

```
'LBR': 'Liberia',
'LBY': 'Libya',
'LIE': 'Liechtenstein',
'LTU': 'Lithuania',
'LUX': 'Luxembourg',
'MAC': 'Macau',
'MAD': 'Madagascar',
'MWI': 'Malawi',
'MAS': 'Malaysia',
'MDV': 'Maldives',
'MLI': 'Mali',
'MLT': 'Malta',
'MHL': 'Marshall Islands',
'MTQ': 'Martinique',
'MTN': 'Mauritania',
'MRI': 'Mauritius',
'MYT': 'Mayotte',
'MEX': 'Mexico',
'FSM': 'Micronesia, Federated States of',
'MDA': 'Moldova',
'MON': 'Monaco',
'MNG': 'Mongolia',
'MNE': 'Montenegro',
'MSR': 'Montserrat',
'MAR': 'Morocco',
'MOZ': 'Mozambique',
'MYA': 'Myanmar',
'NAM': 'Namibia',
'NRU': 'Nauru',
'NEP': 'Nepal',
'NED': 'Netherlands',
'NCL': 'New Caledonia',
'NZL': 'New Zealand',
'NCA': 'Nicaragua',
'NIG': 'Niger',
'NGA': 'Nigeria',
'NIU': 'Niue',
'NFK': 'Norfolk Island',
'NIR': 'Northern Ireland',
'MNP': 'Northern Mariana Islands',
'MKD': 'North Macedonia',
'NOR': 'Norway',
'OMA': 'Oman',
'PAK': 'Pakistan',
'PLW': 'Palau',
'PLE': 'State of Palestine',
'PAN': 'Panama',
'PNG': 'Papua New Guinea',
'PAR': 'Paraguay',
'PER': 'Peru',
'PHI': 'Philippines',
'PCN': 'Pitcairn Islands',
'POL': 'Poland',
'POR': 'Portugal',
'PUR': 'Puerto Rico',
'QAT': 'Qatar',
'REU': 'Réunion',
'ROU': 'Romania',
'RUS': 'Russian Federation',
'RWA': 'Rwanda',
'BLM': 'Saint Barthélemy',
'SHN': 'Saint Helena, Ascension and Tristan da Cunha',
'SKN': 'Saint Kitts and Nevis',
'LCA': 'Saint Lucia',
```

```
'MAF': 'Saint Martin (French part)',
'SPM': 'Saint Pierre and Miquelon',
'VIN': 'Saint Vincent and the Grenadines',
'SAM': 'Samoa',
'SMR': 'San Marino',
'STP': 'São Tomé and Príncipe',
'KSA': 'Saudi Arabia',
'SCO': 'Scotland',
'SEN': 'Senegal',
'SRB': 'Serbia',
'SEY': 'Seychelles',
'SLE': 'Sierra Leone',
'SIN': 'Singapore',
'SXM': 'Sint Maarten (Dutch part)',
'SVK': 'Slovakia',
'SVN': 'Slovenia',
'SOL': 'Solomon Islands',
'SOM': 'Somalia',
'RSA': 'South Africa',
'SGS': 'South Georgia and the South Sandwich Islands',
'SSD': 'South Sudan',
'ESP': 'Spain',
'SRI': 'Sri Lanka',
'SDN': 'Sudan',
'SUR': 'Suriname',
'SJM': 'Svalbard and Jan Mayen',
'SWE': 'Sweden',
'SUI': 'Switzerland',
'SYR': 'Syria',
'TPE': 'Taiwan',
'TJK': 'Tajikistan',
'TAN': 'Tanzania',
'THA': 'Thailand',
'TLS': 'Timor-Leste',
'TOG': 'Togo',
'TKL': 'Tokelau',
'TGA': 'Tonga',
'TRI': 'Trinidad and Tobago',
'TUN': 'Tunisia',
'TUR': 'Turkey',
'TKM': 'Turkmenistan',
'TCA': 'Turks and Caicos Islands',
'TUV': 'Tuvalu',
'UGA': 'Uganda',
'UKR': 'Ukraine',
'UAE': 'United Arab Emirates',
'GBR': 'United Kingdom',
'USA': 'United States',
'UMI': 'United States Minor Outlying Islands',
'VIR': 'United States Virgin Islands',
'URU': 'Uruguay',
'UZB': 'Uzbekistan',
'VAN': 'Vanuatu',
'VAT': 'Vatican City State',
'VEN': 'Venezuela',
'VIE': 'Vietnam',
'WAL': 'Wales',
'WLF': 'Wallis and Futuna',
'ESH': 'Western Sahara',
'YEM': 'Yemen',
'ZAM': 'Zambia',
'ZIM': 'Zimbabwe'}
```

In [92]:
```
df_fbref_outfield['Nationality Cleaned'] = df_fbref_outfield['Nationality Co
df_fbref_outfield.head()
```

Out[92]:

	player	nationality	position	team	comp_level	age	birth_year	games	games_
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 123 columns

4.4 String Cleaning: Cleaning League names in the dataset

In [97]:
```
## Define league names and cleaned league names
dict_league_names = {'eng Premier League': 'Premier League',
                     'fr Ligue 1': 'Ligue 1',
                     'de Bundesliga': 'Bundeliga',
                     'it Serie A': 'Serie A',
                     'es La Liga': 'La Liga'
                    }
df_fbref_outfield['Comp'] = df_fbref_outfield['comp_level'].map(dict_league_
```

In [98]:
```
df_fbref_outfield.tail()
```

Out[98]:

	player	nationality	position	team	comp_level	age	birth_year	games	gam
12310	Zlatan Ibrahimović	se SWE	FW	Milan	it Serie A	41-158	1981	2.0	
11811	Ángel Di María	ar ARG	FW,MF	Juventus	it Serie A	35-024	1988	16.0	
11917	Éderson	br BRA	MF	Atalanta	it Serie A	23-246	1999	22.0	
12371	Þórir Jóhann Helgason	is ISL	MF	Lecce	it Serie A	22-163	2000	8.0	
13477	Łukasz Skorupski	pl POL	GK	Bologna	it Serie A	31-309	1991	25.0	

5 rows × 124 columns

4.5 Grouping Player Positions and Defining a grouped position

In [100…
```
## Define positions and their grouped position names
dict_positions_grouped = {'DF': 'Defender',
                          'DF,FW': 'Defender',
                          'DF,GK': 'Defender',
                          'DF,MF': 'Defender',
                          'FW': 'Forward',
                          'FW,DF': 'Forward',
                          'FW,MF': 'Forward',
                          'GK': 'Goalkeeper',
                          'GK,FW': 'Goalkeeper',
                          'MF': 'Midfielder',
                          'MF,DF': 'Midfielder',
                          'MF,FW': 'Midfielder',
                          'MF,GK': 'Midfielder',
                         }
df_fbref_outfield['Primary Pos'] = df_fbref_outfield['position'].str[:2]
# Map grouped positions to DataFrame
df_fbref_outfield['Position Grouped'] = df_fbref_outfield['position'].map(di
df_fbref_outfield.head()
```

Out[100]:

	player	nationality	position	team	comp_level	age	birth_year	games	games
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 126 columns

4.6 Converting Data Types

Age column converted from string to integer

In [105…
```
## Converting Data Types
## Convert age to string and then to int64
df_fbref_outfield['age'] = df_fbref_outfield['age'].astype(str)
df_fbref_outfield['age'] = df_fbref_outfield['age'].str[:2]
df_fbref_outfield['age'] = pd.to_numeric(df_fbref_outfield['age'], errors='c
df_fbref_outfield['age'] = np.nan_to_num(df_fbref_outfield['age']).astype(in
```

Converting Birth year from String to integer

In [107…
```
df_fbref_outfield['birth_year'] = pd.to_numeric(df_fbref_outfield['birth_yea
```

In [109…
```
df_fbref_outfield.head()
```

Out[109]:

	player	nationality	position	team	comp_level	age	birth_year	games	games
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 126 columns

Export DataFrame

In [113…
```
# Export DataFrame as a CSV file

## Export a copy to the FBref Engineered Outfield folder called 'latest' (ca
df_fbref_outfield.to_csv(data_dir_fbref + '/engineered/outfield/fbref_outfie

## Export a copy to the 'archive' subfolder, including the date
df_fbref_outfield.to_csv(data_dir_fbref + f'/engineered/outfield/archive/fbr
```

Number of Unique Players

In [122…
```
#Number of unique players
len(df_fbref_outfield['Player Lower'].unique().tolist())
```

Out[122]:
```
5264
```

The Number of Forwards

In [124…
```
#Number of players with primary position FW
df_players_raw_fw = df_fbref_outfield[df_fbref_outfield['Primary Pos'] == 'F

len(df_players_raw_fw['Player Lower'].unique().tolist())
```

Out[124]:
```
1590
```

In [125…
```
# Print the shape of the raw DataFrame, df_fbref_outfield
print(df_fbref_outfield.shape)
```

```
(13880, 126)
```

In [128…
```
# Description of the raw DataFrame, df_fbref_outfield, showing some summary
df_fbref_outfield.describe()
```

Out[128]:

	age	birth_year	games	games_starts	minutes	g
count	13880.000000	13880.000000	13880.000000	13880.000000	13880.000000	13880.000
mean	25.414193	1994.386671	17.731772	13.331556	1196.824784	1.632
std	4.551281	4.742461	11.099607	10.933905	944.788889	3.140
min	14.000000	1977.000000	1.000000	0.000000	1.000000	0.000
25%	22.000000	1991.000000	7.000000	3.000000	318.750000	0.000
50%	25.000000	1995.000000	18.000000	12.000000	1064.000000	0.000
75%	29.000000	1998.000000	27.000000	22.000000	1919.000000	2.000
max	42.000000	2007.000000	38.000000	38.000000	3420.000000	41.000

8 rows × 111 columns

In [131…

```
print(len(df_fbref_outfield.columns))
for col in df_fbref_outfield.columns:
    print(col)
```

```
126
player
nationality
position
team
comp_level
age
birth_year
games
games_starts
minutes
goals
assists
pens_made
pens_att
cards_yellow
cards_red
goals_per90
assists_per90
goals_assists_per90
goals_pens_per90
goals_assists_pens_per90
xg
npxg
xg_per90
npxg_per90
Season
minutes_90s
shots_on_target
shots_free_kicks
shots_on_target_pct
shots_on_target_per90
goals_per_shot
goals_per_shot_on_target
npxg_per_shot
xg_net
npxg_net
passes_completed
passes
passes_pct
passes_total_distance
passes_progressive_distance
passes_completed_short
passes_short
passes_pct_short
passes_completed_medium
passes_medium
passes_pct_medium
passes_completed_long
passes_long
passes_pct_long
assisted_shots
passes_into_final_third
passes_into_penalty_area
crosses_into_penalty_area
progressive_passes
passes_live
passes_dead
passes_free_kicks
through_balls
passes_switches
crosses
corner_kicks
corner_kicks_in
```

```
corner_kicks_out
corner_kicks_straight
throw_ins
passes_offsides
passes_blocked
sca
sca_per90
sca_passes_live
sca_passes_dead
sca_shots
sca_fouled
gca
gca_per90
gca_passes_live
gca_passes_dead
gca_shots
gca_fouled
gca_defense
tackles
tackles_won
tackles_def_3rd
tackles_mid_3rd
tackles_att_3rd
blocks
blocked_shots
blocked_passes
interceptions
clearances
errors
touches
touches_def_pen_area
touches_def_3rd
touches_mid_3rd
touches_att_3rd
touches_att_pen_area
touches_live_ball
carries
progressive_carries
carries_into_final_third
carries_into_penalty_area
passes_received
miscontrols
dispossessed
cards_yellow_red
fouls
fouled
offsides
pens_won
pens_conceded
own_goals
ball_recoveries
aerials_won
aerials_lost
aerials_won_pct
Player Lower
First Name Lower
Last Name Lower
First Initial Lower
Nationality Code
Nationality Cleaned
Comp
Primary Pos
Position Grouped
```

4.7 Creating Per 90 Stats

Standardising stats to per 90 to allow players that have had less minutes on the field or had less time on the ball to be compared to firt team regular players.

The following Stats from the original Dataset have been converted to per 90 stats:

- **Goals Per Shot**
- **Goals Per Shot on Target**
- **Non Goals XG per Shot**
- **XG NET**
- **Passes Into Final Third**
- **Passes Into Penalty Area**
- **Crosses Into Penalty Area**
- **Progressive Passes**
- **Through Balls**
- **Carries Into Penalty Area**

In [132…

```
## Creating per 90 stats for later use
df_players = df_fbref_outfield
df_players['goals_per_shot_per90'] = (df_players['goals_per_shot'] / df_play
df_players['goals_per_shot_on_target_per90'] = (df_players['goals_per_shot_o
df_players['npxg_per_shot_per90'] = (df_players['npxg_per_shot'] / df_player
df_players['xg_net_per90'] = (df_players['xg_net'] / df_players['minutes'])
df_players['passes_into_final_third_per90'] = (df_players['passes_into_final
df_players['passes_into_penalty_area_per90'] = (df_players['passes_into_pena
df_players['crosses_into_penalty_area_per90'] = (df_players['crosses_into_pe
df_players['progressive_passes_per90'] = (df_players['progressive_passes'] /
df_players['through_balls_per90'] = (df_players['through_balls'] / df_player
df_players['carries_into_penalty_area_per90'] = (df_players['carries_into_pe
df_players.head()
```

Out[132]:

	player	nationality	position	team	comp_level	age	birth_year	games	games
1533	Aarón Martín	es ESP	DF	Mainz 05	de Bundesliga	21	1997	33.0	
639	Abdou Diallo	sn SEN	DF	Dortmund	de Bundesliga	22	1996	28.0	
1013	Achraf Hakimi	ma MAR	DF	Dortmund	de Bundesliga	19	1998	21.0	
305	Adam Bodzek	de GER	MF,DF	Düsseldorf	de Bundesliga	32	1985	21.0	
2646	Adam Zreľák	sk SVK	FW	Nürnberg	de Bundesliga	24	1994	14.0	

5 rows × 136 columns

4.8 Subset Data of only Forwards with at least 10 Matches

Filter for Forwards that have played at least 10 Matches or 900 minutes during a season

In [134…
```
## Subset Data
# Filter for Forwards that have played at least 10 matches (900 minutes)
df_players_fw = df_players[
                           (df_players['Primary Pos'] ==  'FW') &
                           (df_players['Season'].isin(['2018-2019', '2019-2
                           (df_players['minutes'] >= 90 * 10)
                          ]
```

In [136…
```
# Check the variance in the dataset before and after filtration
print('No. rows in Players DataFrame BEFORE filtration: {}'.format(len(df_pl
print('No. rows AFTER filtration: {}'.format(len(df_players_fw)))
print('-'*10+'\n')
```

```
No. rows in Players DataFrame BEFORE filtration: 13880
No. rows AFTER filtration: 1606
----------
```

4.9 Selecting Numerical and String Columns of Interest

As part of the analysis, we require a mix of player performance stats and player bio

The player performance metrics that are to go into the clustering are metrics for a forward. Irrelevant metrics for defending and goalkeeping are not included as this will distrort the player profiles to be identified in the clustering. The columns of interest from the dataset are the following...

In [516…
```
## Selecting colums of interest
## String Columns:
cols_str = ['Player Lower',
            'Season',
            'Comp',
            'Primary Pos',
            'position',
            'age',
            'birth_year',
            'Nationality Cleaned',
            'team'
]
cols_stats = ['games',
              'games_starts',
              'minutes',
              'goals',
              'assists',
              'goals_per90',
              'assists_per90',
              'goals_assists_per90',
              'goals_pens_per90',
              'goals_assists_pens_per90',
              'xg_per90',
              'npxg_per90',
              'minutes_90s',
              'shots_on_target_pct',
              'shots_on_target_per90',
```

```
                'goals_per_shot_per90',
                'goals_per_shot_on_target_per90',
                'npxg_per_shot_per90',
                'xg_net_per90',
                'passes_into_final_third_per90',
                'passes_into_penalty_area_per90',
                'crosses_into_penalty_area_per90',
                'progressive_passes_per90',
                'through_balls_per90',
                'carries_into_penalty_area_per90',
                'progressive_carries'
]
```

Create a DataFrame of just Numerical values

In [472…

```
# Create DataFrame of numerical values

## Select columns of interest
df_players_fw_stats = df_players_fw[cols_stats]

## Display DataFrame
df_players_fw_stats.head()
```

Out[472]:

	goals	assists	goals_per90	assists_per90	goals_assists_per90	goals_pens_per90
2646	2.0	1.0	0.19	0.10	0.29	0.19
1592	6.0	5.0	0.32	0.27	0.59	0.32
1962	12.0	4.0	0.43	0.14	0.58	0.43
803	10.0	1.0	0.67	0.07	0.74	0.40
683	5.0	2.0	0.47	0.19	0.65	0.47

5 rows × 23 columns

In [473…

```
df_players_fw_stats.shape
```

Out[473]:

```
(1606, 23)
```

Replacing all Null Values if any.

In [474…

```
# Replace all the NULL values with zero.
df_players_fw_stats.fillna(0, inplace=True)
```

```
/var/folders/nv/yhx12vpj1vz672s8q9xw9ylw0000gn/T/ipykernel_73741/3759243583.
py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_players_fw_stats.fillna(0, inplace=True)
```

4.10 Data Normalisation

Normalising the data to scale the values of the features into a similar range.

The MinMaxScaler is used to standardise all the numerical values in the data to have a mean of approx 0 and variance of 1.

In [584…
```
## Standardise Data
x = df_players_fw_stats.values     # NumPy array
scaler = preprocessing.MinMaxScaler()
x_scaled = scaler.fit_transform(x)
X_norm = pd.DataFrame(x_scaled)
```

4.11 Dimentionality Reduction using PCA

PCA is used to reduce the dimensionality of a dataset while retaining most of its important information, by tranforming large set of variables into a smaller one that still contains most of the information.

Here PCA is used to reduce the dimentionality for only Visualisations, For traing the clustering algorithms all the features are being used.

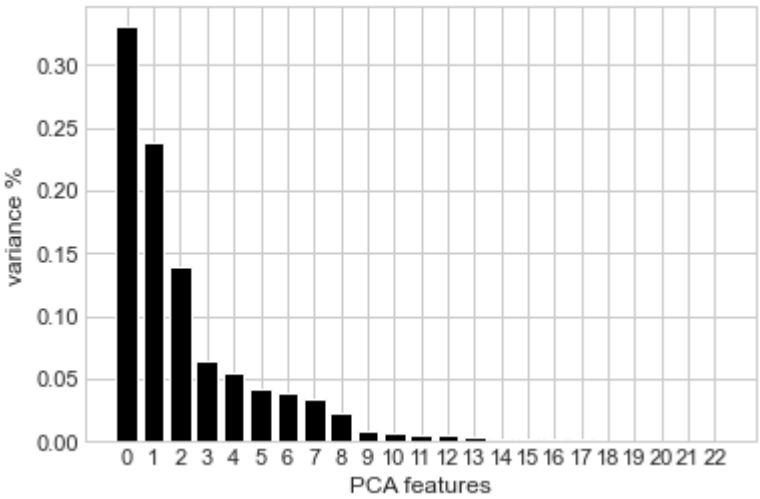
In [ ]:
```
```

In [585…
```
# Create a PCA Instance
pca = PCA()
pca.fit(X_norm)
```

Out[585]:
```
PCA()
```

Determine the optimal number of components which capture the greatest amount of variance in the data.

In [586…
```
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
plt.show()
```



5. Learning Methods

5.1 K-Means

K-Means clustering is a clustering algorithm that involves the following steps:

- First, a user specifies the number of clusters (K) and randomly selects initial centroids.
- Then, the algorithm iteratively assigns each observation to one of the K clusters until the assignments stop changing.
- Next, the mean is calculated for each of the K clusters. Finally, each observation is assigned to the cluster whose mean is closest to it.
- The objective is to form clusters in a way that maximizes the similarity of observations within the same cluster.

K-Means clustering measures similarity using the Squared Euclidean distance.

In [ ]:
```
kmeans_pca_2d = PCA(n_components=2)

# Save components to a DataFrame
df_reduced = pd.DataFrame(kmeans_pca_2d.fit_transform(X_norm))
```

In [588…
```
df_reduced.head(10)
```

Out[588]:

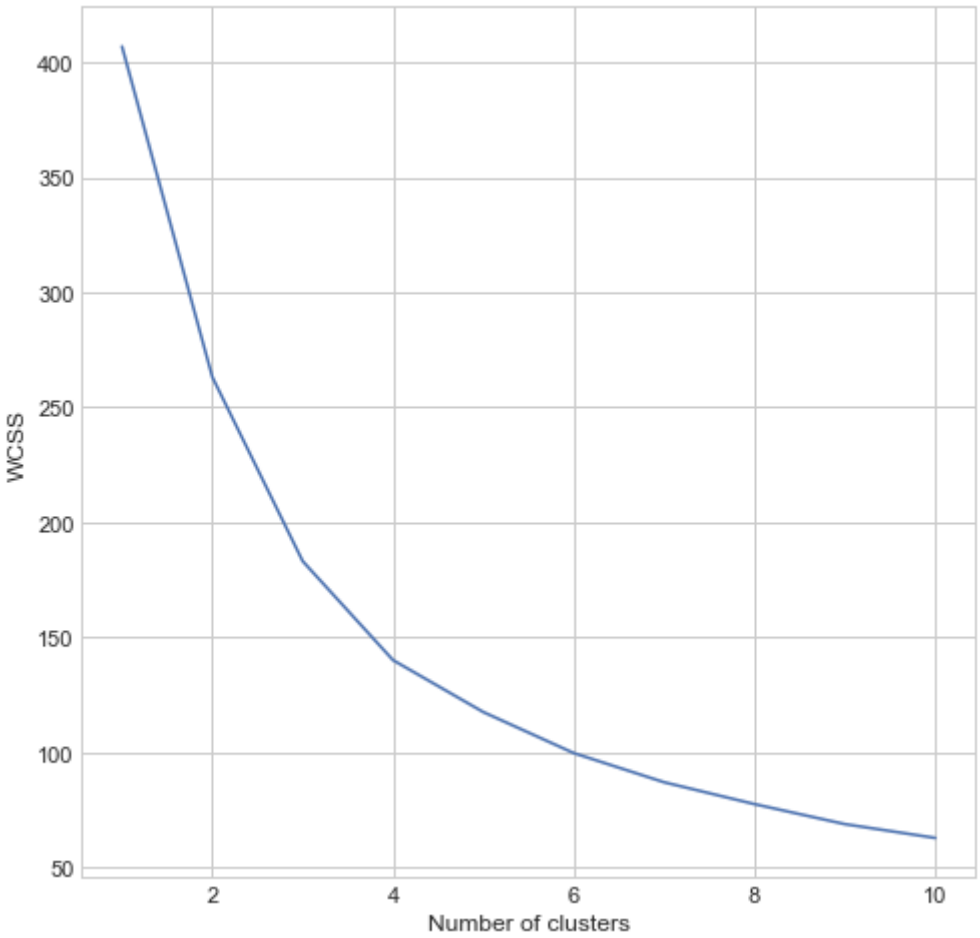
	0	1
0	-0.364910	0.603039
1	-0.005329	-0.079077
2	0.320816	-0.164016
3	0.246158	0.477178
4	0.028878	0.517735
5	-0.295003	-0.262616
6	0.265565	-0.095312
7	-0.311442	0.317614
8	-0.083366	0.335727
9	0.234359	0.404751

5.1.1 Determining Elbow

To find the appropriate number of clusters, one can calculate the Within Cluster Sum of Squares (WCSS) for various cluster solutions. The WCSS values can be plotted in an Elbow plot, and the number of clusters can be determined by selecting the elbow point on the curve. The elbow point is the number of clusters where the rate of WCSS reduction begins to level off, indicating that additional clusters do not significantly improve the clustering performance.

In [589…

```
## Determining elbow
wcss = []

for i in range(1,11):
    model = KMeans(n_clusters = i, init = "k-means++", max_iter = 600, randc
    model.fit(df_reduced)
    wcss.append(model.inertia_)

plt.figure(figsize=(8, 8))
plt.plot(range(1,11), wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



The plot shows that there is an elbow near 4, hence we will be considering 4 clusters.

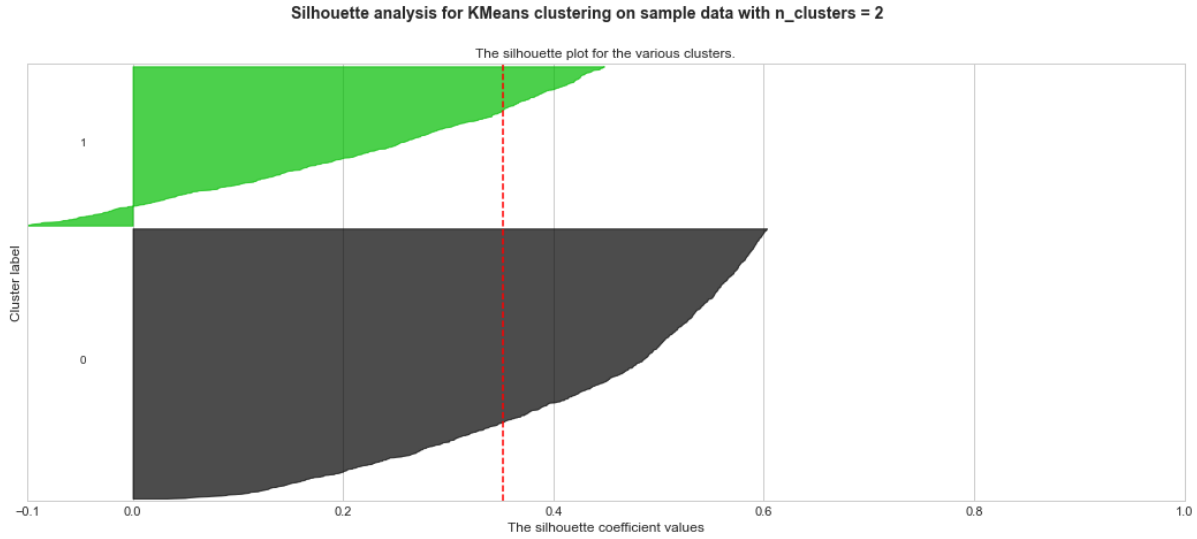
5.1.2 Silhouette Score to determine optimal number of clusters

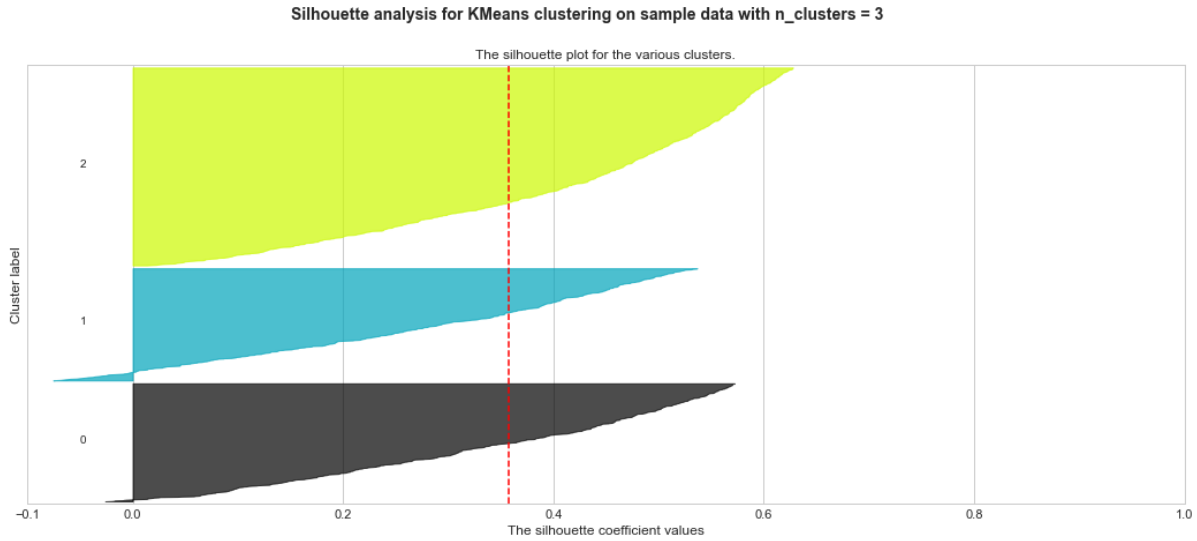
In [591…

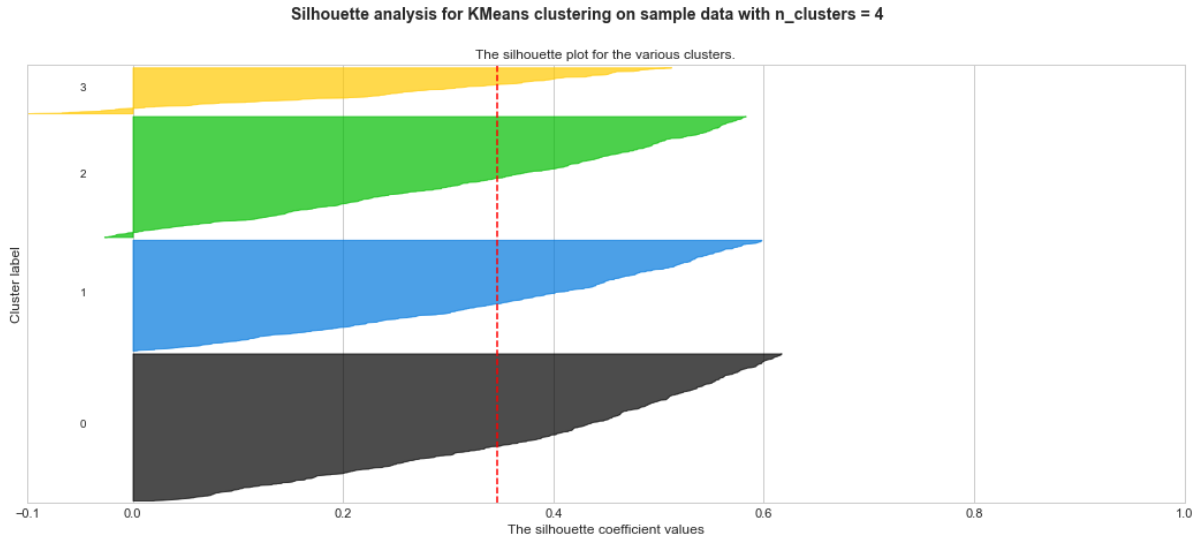
```
range_n_clusters = [2,3,4,5,6,7,8,9,10]
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1) = plt.subplots(1)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example al
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(df_reduced) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters,init='k-means++', max_iter=600,
    cluster_labels = clusterer.fit_predict(df_reduced)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(df_reduced, cluster_labels)
    print(
        "For n_clusters =",
```

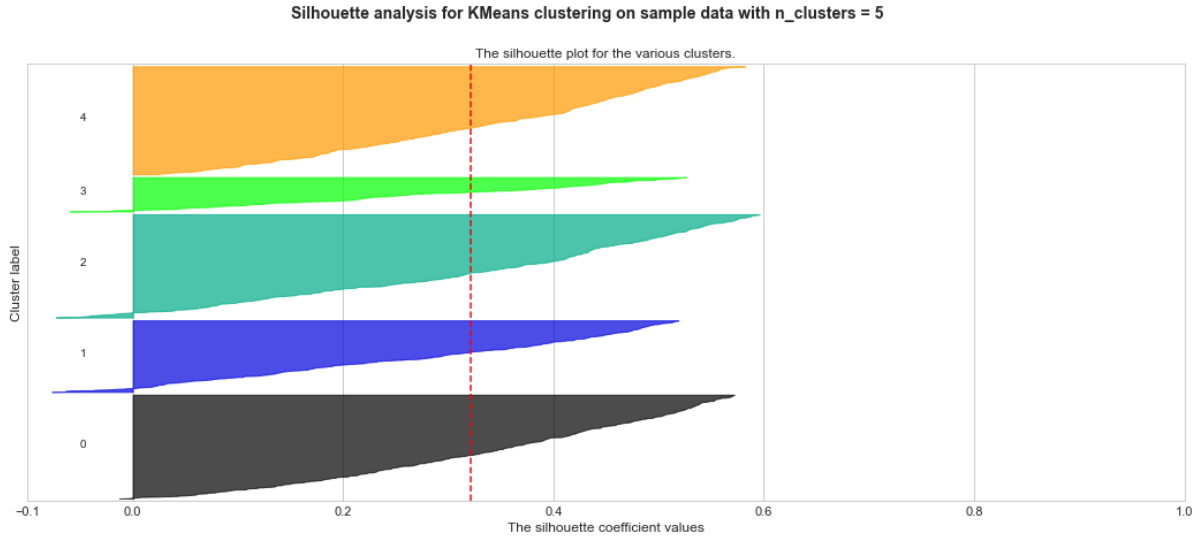
```
        n_clusters,
        "The average silhouette_score is :",
        silhouette_avg,
    )

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(df_reduced, cluster_labels

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_lab

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the middl
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1]

    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with n_clu
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )

plt.show()
```

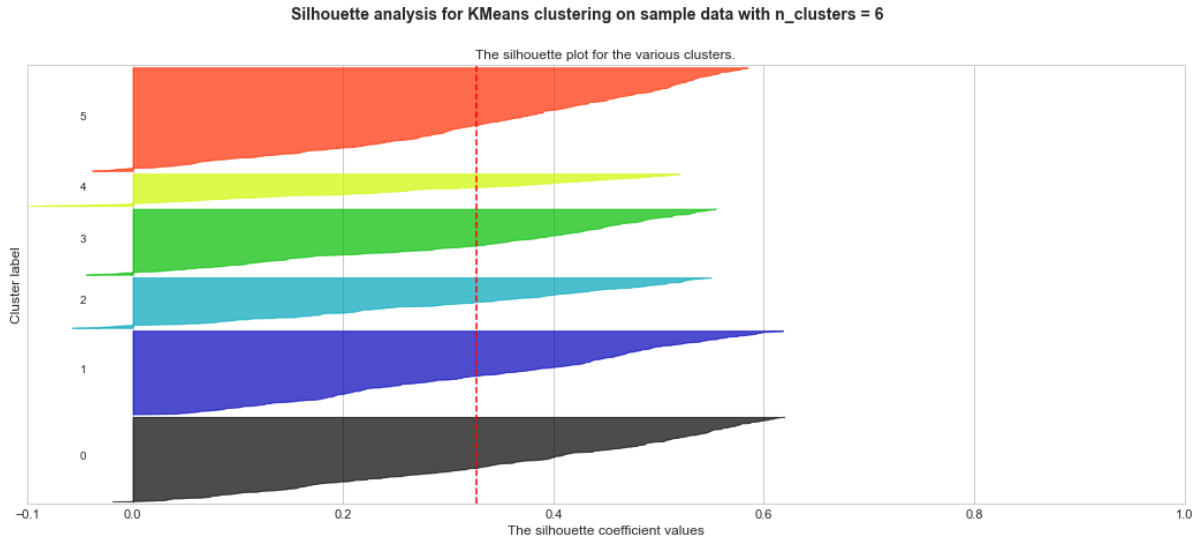
```
For n_clusters = 2 The average silhouette_score is : 0.3517643638160773
For n_clusters = 3 The average silhouette_score is : 0.3579423672179299
For n_clusters = 4 The average silhouette_score is : 0.3465608526411299
For n_clusters = 5 The average silhouette_score is : 0.3210605029601038
For n_clusters = 6 The average silhouette_score is : 0.32695659588683545
For n_clusters = 7 The average silhouette_score is : 0.32950588845850814
For n_clusters = 8 The average silhouette_score is : 0.3206865282670317
For n_clusters = 9 The average silhouette_score is : 0.3282346817062359
For n_clusters = 10 The average silhouette_score is : 0.32452984247592154
```

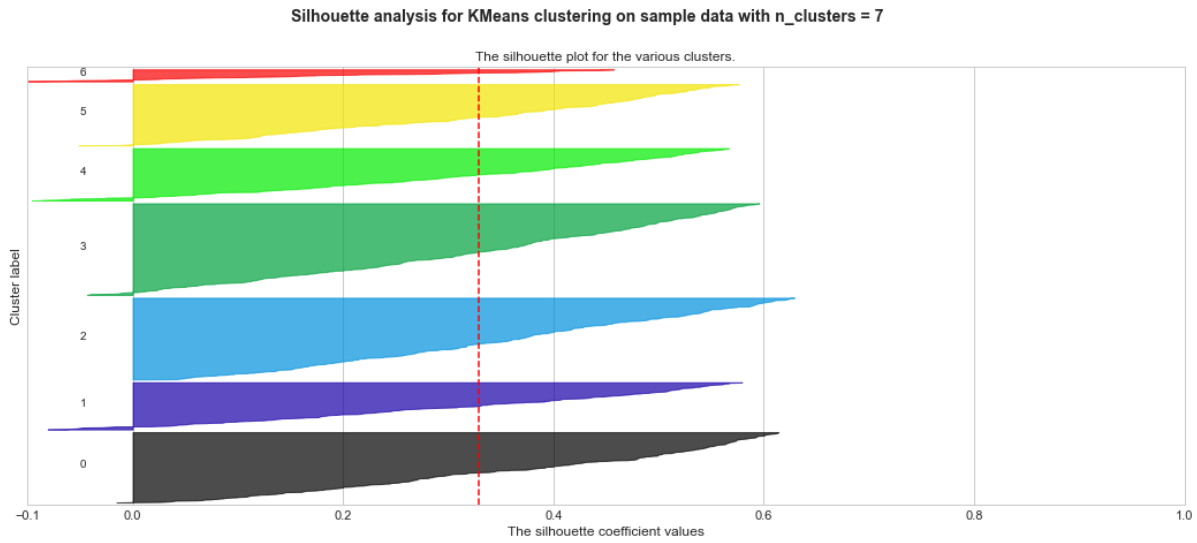

Silhouette analysis for KMeans clustering on sample data with n_clusters = 2
The silhouette plot for the various clusters.
Cluster label
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

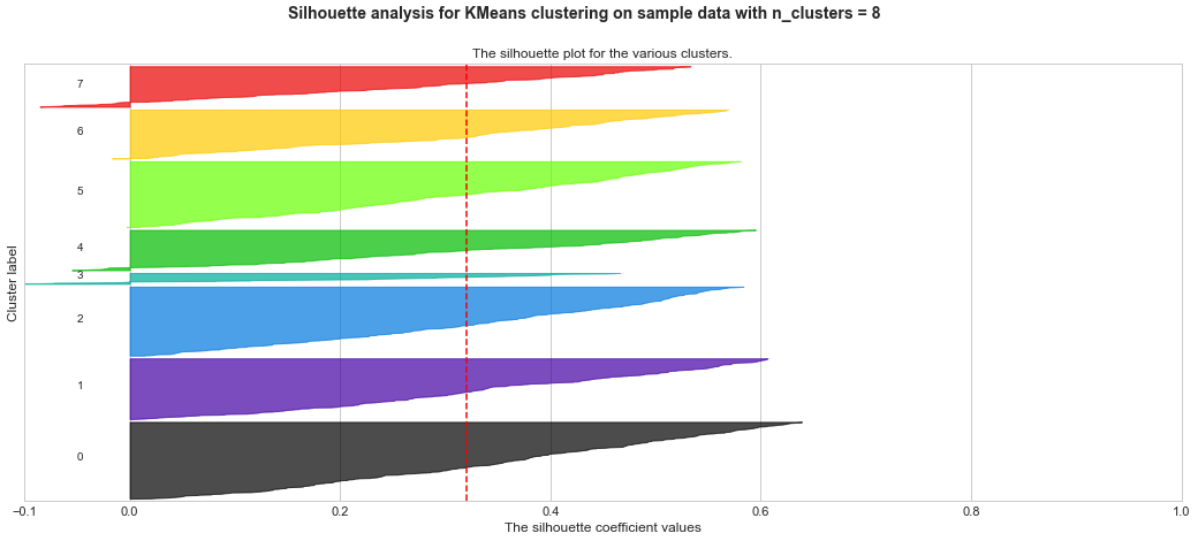

Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

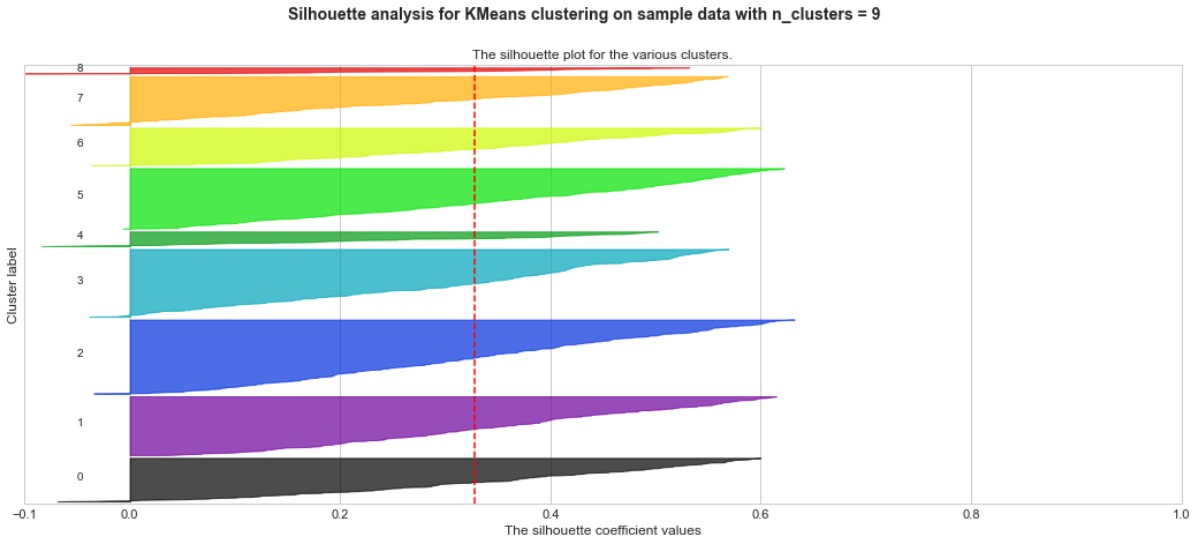

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

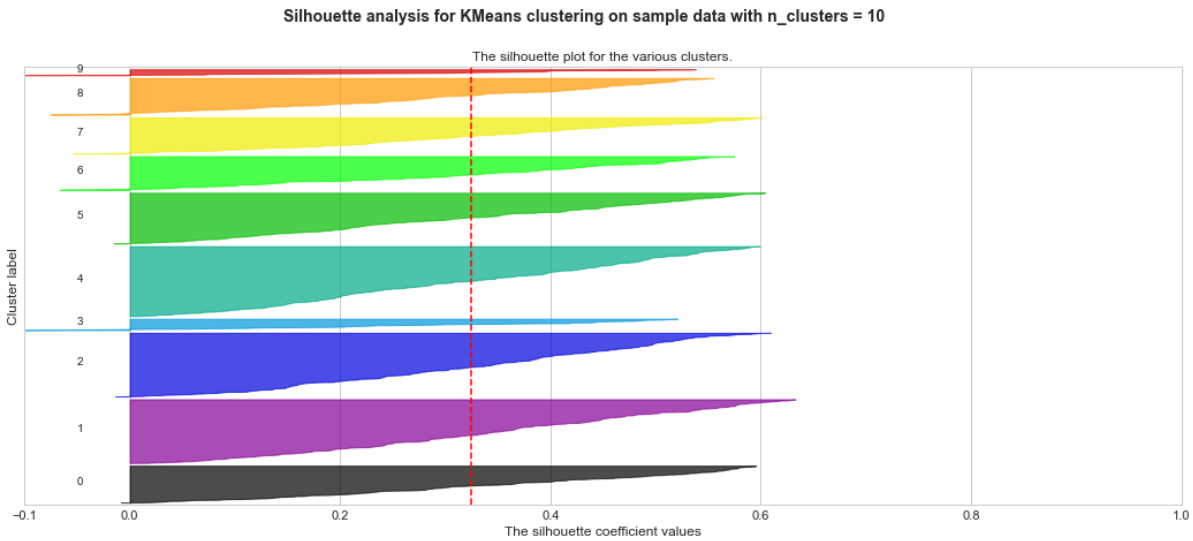

Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 7
The silhouette plot for the various clusters.
Cluster label
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values







The average Silhouette score is being drastically decreased when number of clusters is 4. Hence we will be considering 4 clusters.

5.1.3 Clustering the Data using K-Means

In [592…

```
# K-Means Clustering
```

```
## Specify the number of clusters
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=600, random_state=4

## Fit the input data
kmeans = kmeans.fit(df_reduced)

## Get the cluster labels
labels = kmeans.predict(df_reduced)

## Centroid values
centroid = kmeans.cluster_centers_

## Cluster values
clusters = kmeans.labels_.tolist()
```

In [593…
```
## Silhouette score of Kmeans
km_score = silhouette_score(df_reduced, clusters)
print('Silhouetter Score: %.3f' % km_score)
```

```
Silhouetter Score: 0.347
```

In [594…
```
## Davies Bouldin Score of Kmeans
km_davies_bouldin_score = davies_bouldin_score(df_reduced,clusters)
km_davies_bouldin_score
```

Out[594]:
```
0.8777551503192457
```

In [595…
```
## Calinski Score for Kmeans
km_calinski_score = calinski_harabasz_score(df_reduced, clusters)
km_calinski_score
```

Out[595]:
```
1016.7282138852587
```

In [596…
```
# Convert string attributes to lists to be joined back onto the resulting Da
names = df_players_fw['Player Lower'].tolist()
season = df_players_fw['Season'].tolist()
minutes = df_players_fw['minutes'].tolist()
matches_played = df_players_fw['games'].tolist()
matches_started = df_players_fw['games_starts'].tolist()
team = df_players_fw['team'].tolist()
comp = df_players_fw['Comp'].tolist()
primary_pos = df_players_fw['Primary Pos'].tolist()
pos = df_players_fw['position'].tolist()
age = df_players_fw['age'].tolist()
birth_year = df_players_fw['birth_year'].tolist()
nationality = df_players_fw['Nationality Cleaned'].tolist()
```

In [597…
```
df_reduced['cluster'] = clusters
df_reduced['name'] = names
df_reduced['season'] = season
df_reduced['minutes'] = minutes
df_reduced['matches_played'] = matches_played
df_reduced['matches_started'] = matches_started
df_reduced['comp'] = comp
df_reduced['team'] = team
df_reduced['primary_pos'] = primary_pos
df_reduced['pos'] = pos
df_reduced['age'] = age
df_reduced['birth_year'] = birth_year
df_reduced['nationality'] = nationality
```

In [598…
```
df_reduced.columns = ['x',
                      'y',
                      'cluster',
                      'name',
                      'season',
                      'minutes',
                      'matches_played',
                      'matches_started',
                      'comp',
                      'team',
                      'primary_pos',
                      'pos',
                      'age',
                      'birth_year',
                      'nationality'
]
```

In [599…
```
df_reduced.shape
```

Out[599]:
```
(1606, 15)
```

In [600…
```
df_reduced.head()
```

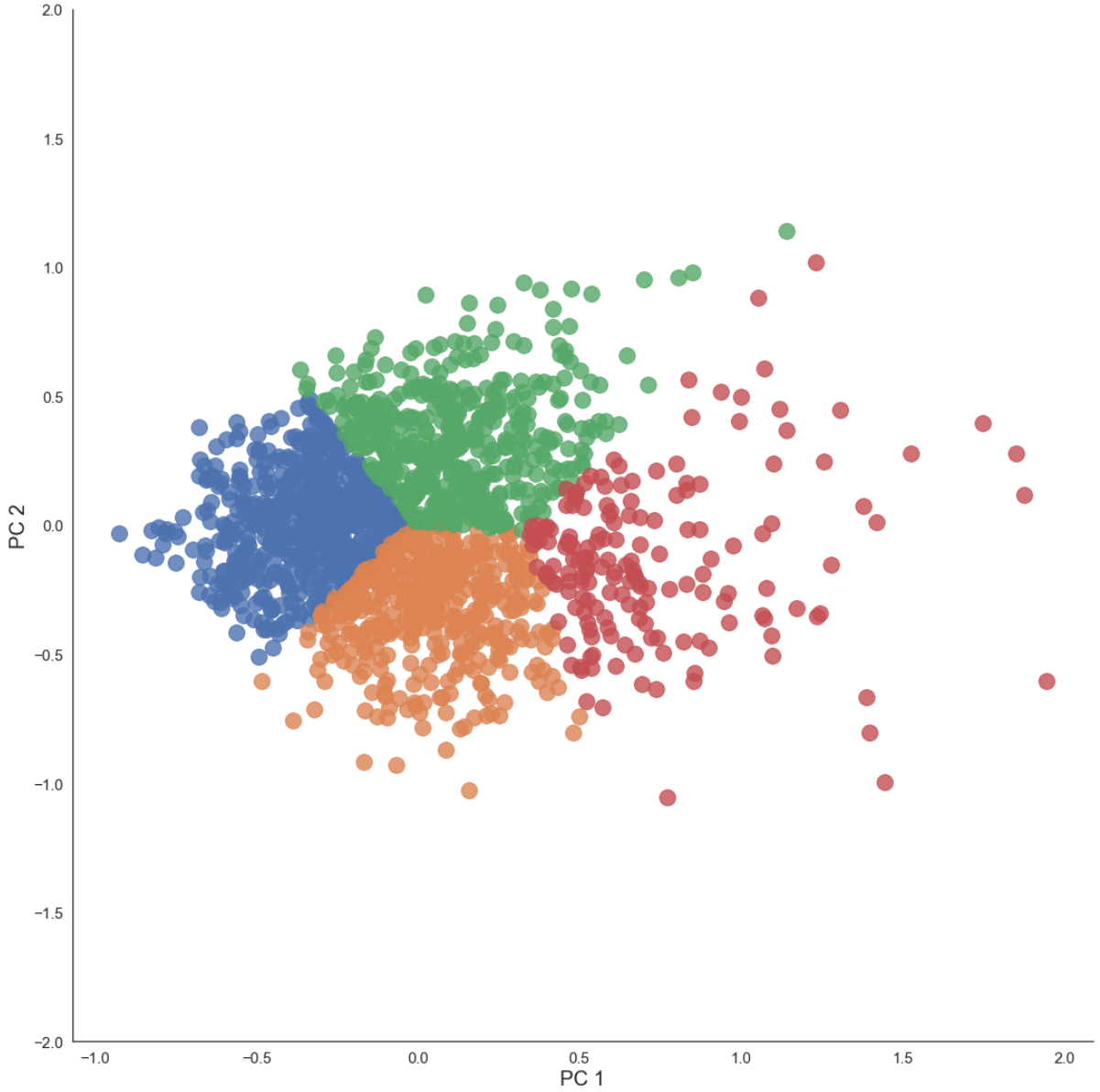
Out[600]:

	x	y	cluster	name	season	minutes	matches_played	matches_
0	-0.364910	0.603039	2	adam zrelak	2018-2019	941.0	14.0	
1	-0.005329	-0.079077	1	admir mehmedi	2018-2019	1692.0	26.0	
2	0.320816	-0.164016	1	alassane plea	2018-2019	2503.0	34.0	
3	0.246158	0.477178	2	alfre finnbogason	2018-2019	1335.0	18.0	
4	0.028878	0.517735	2	anastasios donis	2018-2019	962.0	24.0	

5.1.4 Data Visualisation

In [601…
```
#Data Visualisation
# Visualise the clustering
sns.set(style='white')
ax = sns.lmplot(x='x', y='y', hue='cluster', data=df_reduced, legend=False,
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
#for x, y, s in zip(df_reduced.x, df_reduced.y, df_reduced.name):
    #texts.append(plt.text(x, y, s))
ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel('PC 1', fontsize = 20)
plt.ylabel('PC 2', fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```



5.2 GMM

The Gaussian Mixture Model (GMM) algorithm is a statistical technique used for clustering and density estimation. It assumes that the data is generated by a mixture of several Gaussian distributions, each with its own mean and variance. The GMM algorithm attempts to estimate the parameters of these Gaussians and use them to assign each data point to a particular cluster.

The basic idea of the GMM algorithm is as follows:

- Initialization: Choose the number of clusters (k) and randomly initialize the mean and covariance of each Gaussian.

- Expectation step: Assign each data point to a cluster by calculating the probability that it belongs to each Gaussian distribution. This is done using Bayes' theorem and the estimated mean and covariance of each Gaussian.
- Maximization step: Update the parameters of each Gaussian based on the assigned data points. This is done by maximizing the likelihood function using the Expectation-Maximization (EM) algorithm.
- Repeat steps 2 and 3 until convergence.

The GMM algorithm can be used for both clustering and density estimation. In clustering, the algorithm assigns each data point to a particular cluster based on the highest probability of belonging to that cluster.

In [602…
```
## GMM
## Standardise Data
gmm_data = df_players_fw_stats.values   # NumPy array
gmm_scaler = preprocessing.MinMaxScaler()
gmm_scaled = gmm_scaler.fit_transform(gmm_data)
gmm_norm = pd.DataFrame(gmm_scaled)
gmm_norm.describe()
```

Out[602]:

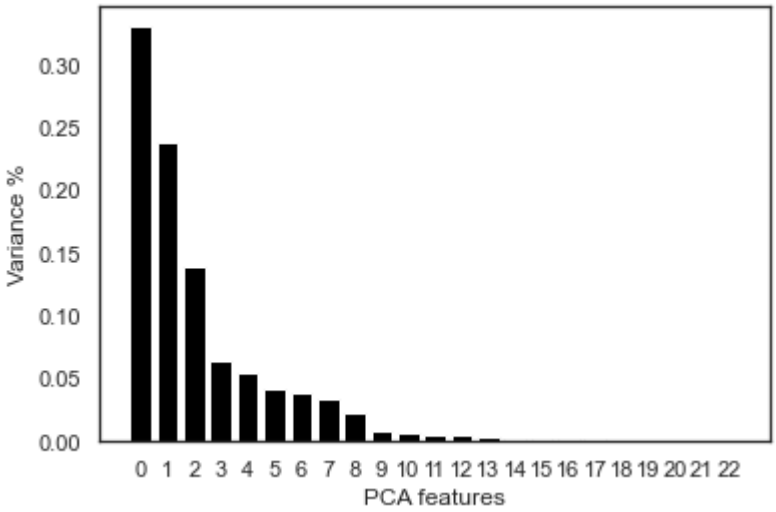
	0	1	2	3	4	5	1
count	1606.000000	1606.000000	1606.000000	1606.000000	1606.000000	1606.000000	
mean	0.174361	0.143005	0.230801	0.177730	0.271256	0.245745	
std	0.132613	0.126268	0.136848	0.138094	0.135400	0.145231	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.073171	0.047619	0.133333	0.084337	0.175824	0.142857	
50%	0.146341	0.095238	0.206667	0.156627	0.252747	0.222222	
75%	0.243902	0.190476	0.306667	0.253012	0.340659	0.325397	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 23 columns

In [603…
```
pca_2 = PCA()
pca_2.fit(gmm_norm)
```

Out[603]:
```
PCA()
```

In [604…
```
features = range(pca_2.n_components_)
plt.bar(features, pca_2.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('Variance %')
plt.xticks(features)
plt.show()
```



In [605…
```
# Create a 2D PCA Instance
pca_2d = PCA(n_components=2)

# Save components to a DataFrame
gmm_df_reduced = pd.DataFrame(pca_2d.fit_transform(gmm_norm))
```

5.2.1 Determining Optimal Number of Clusters using BIC and AIC

The Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) are two statistical metrics used in Gaussian Mixture Model (GMM) clustering to determine the optimal number of clusters.

BIC and AIC are both measures of the quality of the model fit and take into account the number of parameters in the model. They penalize models with a large number of parameters, thus avoiding overfitting.

The AIC is defined as AIC = 2k - 2ln(L), where k is the number of parameters in the model and L is the likelihood function of the data. The BIC is defined as BIC = kln(n) - 2ln(L), where n is the number of data points.

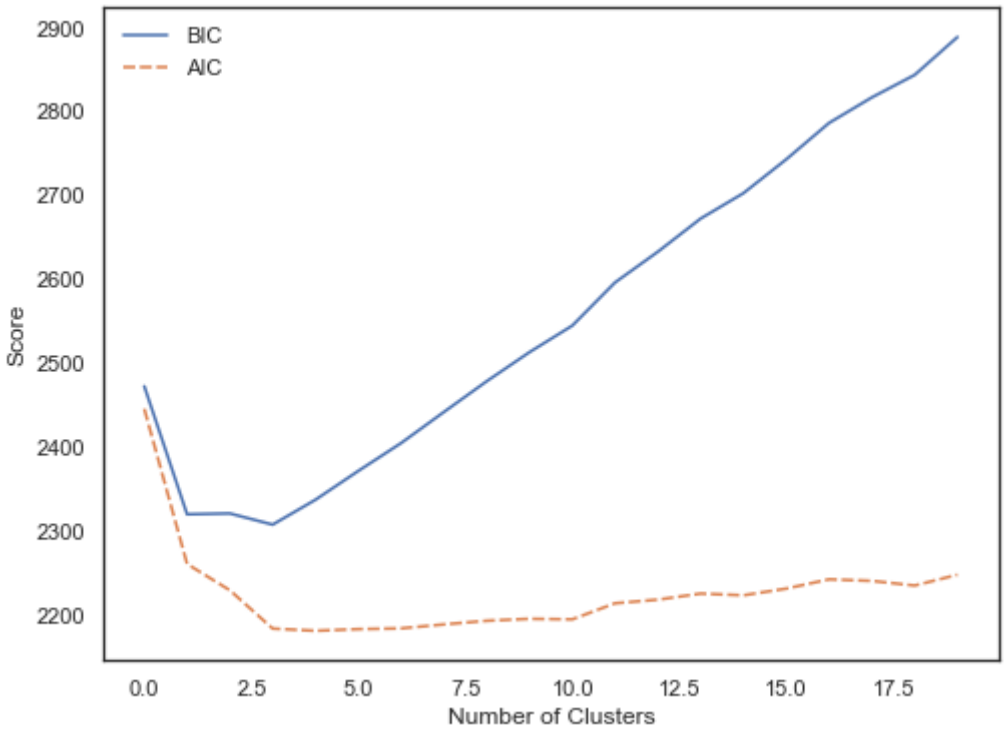
In both cases, the lower the value of the metric, the better the model fit. When comparing GMM models with different numbers of clusters, the model with the lowest AIC or BIC value is typically selected as the optimal model

In [606…
```
n_components = np.arange(1, 21)
models = [GMM(n,covariance_type='full', random_state=0).fit(gmm_df_reduced)
gmm_model_comparisons=pd.DataFrame({"n_components" : n_components,
                                    "BIC" : [m.bic(gmm_df_reduced) for m in mo
                                     "AIC" : [m.aic(gmm_df_reduced) for m in m
gmm_model_comparisons.head()
```

Out[606]:

	n_components	BIC	AIC
0	1	2473.474276	2446.566767
1	2	2320.961238	2261.764717
2	3	2321.916713	2230.431181
3	4	2308.612846	2184.838302
4	5	2338.261262	2182.197707

In [607…
```
plt.figure(figsize=(8,6))
sns.lineplot(data=gmm_model_comparisons[["BIC","AIC"]])
plt.xlabel("Number of Clusters")
plt.ylabel("Score")
plt.savefig("GMM_model_comparison_with_AIC_BIC_Scores_Python.png",
                    format='png',dpi=150)
```



As we can see from the above table and graph the BIC and AIC values are lower when Number of Clusters is 4. Hence Selecting 4 Clusters for GMM Clustering

5.2.2 Clustering Data using GMM

In [608…
```
gmm = GMM(n_components=4)
gmm = gmm.fit(gmm_df_reduced)
gmm_labels = gmm.predict(gmm_df_reduced)
gmm_labels_list = gmm_labels.tolist()
```

In [616…
```
## Silhouette Score for GMM
gmm_score = silhouette_score(gmm_norm, gmm_labels)
print('Silhouette Score: %.3f' % gmm_score)
```

```
Silhouette Score: 0.160
```

In [610…
```
## Davies Bouldin Socre for GMM
gmm_davies_bouldin_score = davies_bouldin_score(gmm_norm,gmm_labels)
gmm_davies_bouldin_score
```

Out[610]:
```
1.6807706185997255
```

In [611…
```
## Calinski Score for GMM
gmm_calinski_score = calinski_harabasz_score(gmm_norm, gmm_labels)
gmm_calinski_score
```

Out[611]:
```
314.36191877045707
```

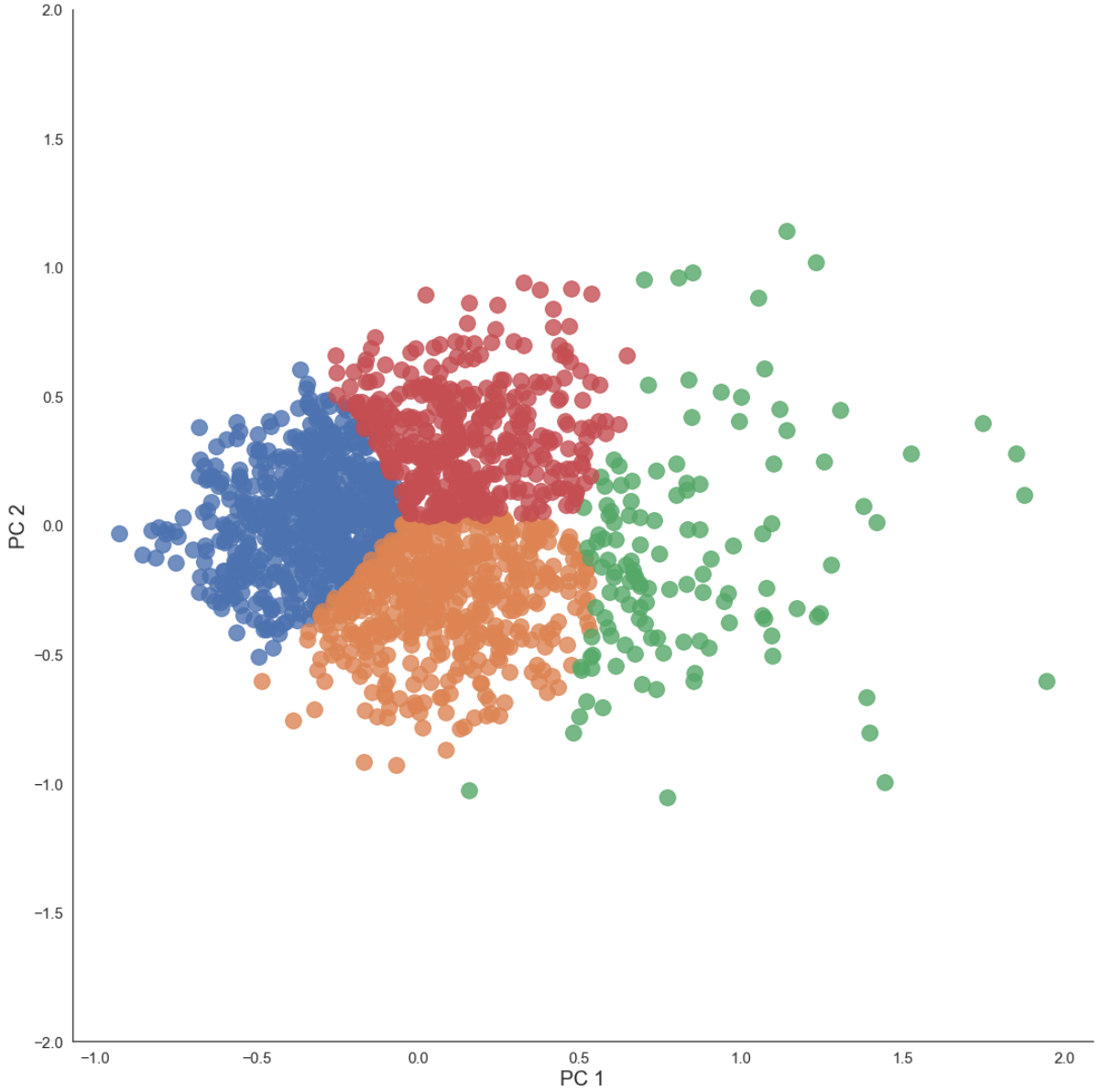
In [612…
```
gmm_df_reduced['cluster'] = gmm_labels_list
gmm_df_reduced['name'] = names
gmm_df_reduced['season'] = season
gmm_df_reduced['minutes'] = minutes
gmm_df_reduced['matches_played'] = matches_played
gmm_df_reduced['matches_started'] = matches_started
gmm_df_reduced['comp'] = comp
gmm_df_reduced['team'] = team
gmm_df_reduced['primary_pos'] = primary_pos
gmm_df_reduced['pos'] = pos
gmm_df_reduced['age'] = age
gmm_df_reduced['birth_year'] = birth_year
gmm_df_reduced['nationality'] = nationality
```

In [613…
```
gmm_df_reduced.columns = ['x',
                          'y',
                          'cluster',
                          'name',
                          'season',
                          'minutes',
                          'matches_played',
                          'matches_started',
                          'comp',
                          'team',
                          'primary_pos',
                          'pos',
                          'age',
                          'birth_year',
                          'nationality'
]
```

5.2.3 Data Visualisation

In [614…
```
sns.set(style='white')
ax = sns.lmplot(x='x', y='y', hue='cluster', data=gmm_df_reduced, legend=Fal
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
# for x, y, s in zip(df_reduced.x, df_reduced.y, df_reduced.name):
#     texts.append(plt.text(x, y, s))
ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel('PC 1', fontsize = 20)
plt.ylabel('PC 2', fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```



6. Comparing K-means and GMM

Silhouette Score, Davies Bouldin Score, and Calinski Harabasz Score are all metrics used to evaluate the quality of clustering results.

- Silhouette Score: measures how well each data point fits into its assigned cluster, and how far it is from the neighbouring clusters. The score ranges from -1 to 1, where a score closer to 1 indicates a better clustering result. Higher scores indicate that the data points are well-clustered and separated from each other.
- Davies Bouldin Score: measures the average similarity between each cluster and its most similar cluster, and compares this to the average distance between each cluster and its most dissimilar cluster. Lower scores indicate better clustering results, with values closer to 0 indicating better separation between the clusters.

- Calinski Harabasz Score: measures the ratio of the between-cluster dispersion to the within-cluster dispersion. Higher scores indicate better clustering results, with larger differences in dispersion between the clusters and smaller dispersion within each cluster.

In [628…

```
# Import matplotlib
import matplotlib.pyplot as plt

# Define the data for the table as a list of lists
cmp_data = [[km_score, km_davies_bouldin_score, km_calinski_score],
        [gmm_score, gmm_davies_bouldin_score, gmm_calinski_score]]

# Define the column labels and row labels for the table
cmp_columns = ["Silhouette Score", "Davies Bouldin Score", "Calinski Harabas
cmp_rows = ["K-Means", "GMM"]

# Create a figure and axes for the table
fig, ax = plt.subplots(figsize=(6, 4))

# Draw the table using matplotlib table function
table = ax.table(cellText=cmp_data,
                 colLabels=cmp_columns,
                 rowLabels=cmp_rows,
                 loc="center")

# Adjust the margins and labels of the table
ax.set_title("Comparing K-means and GMM")
ax.axis("off")
table.auto_set_font_size(False)
table.set_fontsize(14)
table.scale(1.5, 1.5)

# Show the table
plt.show()
```

	Silhouette Score	Davies Bouldin Score	Calinski Harabasz Score
K-Means	0.3465608526411299	0.8777551503192457	1016.7282138852587
GMM	0.15989146361332376	1.6807706185997255	314.36191877045707

From the above Table we can interpret that, K-Means algorithm has better Silhouette Score, Davies Bouldin Score and Calinski Harabasz Score when compared with GMM for the given Data Set. This could be because the data points are well separated and belong to distinct clusters: K-means works well when the clusters are well-separated and have a spherical shape. If the data points belong to distinct clusters with little overlap, K-means can quickly and accurately group them together. Also, K-means works well with uniformly distributed data, where the cluster centers are equidistant from each other. If

the data is uniformly distributed, the K-means algorithm can quickly converge to an optimal solution.

7. Analysis

7.1 Analysing Kylian Mbappe's performance over past 5 seasons

In [629…
```
df_reduced['cluster'].value_counts()
```

Out[629]:
```
0    556
2    457
1    419
3    174
Name: cluster, dtype: int64
```

The results from K-means produces 4 distinct clusters, the most populus being cluster 0 with 556 players, followed by cluster 2 (457), 1 (419) and 3 (174)

In [633…
```
gmm_df_reduced['cluster'].value_counts()
```

Out[633]:
```
0    585
1    500
3    393
2    128
Name: cluster, dtype: int64
```

The results from GMM produces 4 distinct clusters, the most populus being cluster 0 with 585 players, followed by cluster 1 (500), 3 (393) and 2 (128)

The next step is to identify the cluster(s) that Kylian Mbappe is located for the five seasons of data 2018-2019 to 2022-2023

In [568…
```
# Subset the reduced DataFrame for kylian mbappe data points using K-means
df_reduced_mbappe = df_reduced[df_reduced['name'].str.contains('kylian mbapp

## Select columns of interest
df_reduced_mbappe = df_reduced_mbappe[['x', 'y', 'cluster', 'season']]

# Display the DataFrame
df_reduced_mbappe
```

Out[568]:

	x	y	cluster	season
992	1.874440	0.119329	3	2018-2019
1067	1.526649	0.276810	3	2019-2020
1131	1.277519	-0.151959	3	2020-2021
1202	1.387960	-0.666055	3	2021-2022
1261	1.093180	0.008615	3	2022-2023

K-means: From the results we can see that Mbappe is found in the cluster 3 for the past 5 seasons.

Visualising to see where the points for Mbappe lie.

In [634…

```
# Subset the reduced DataFrame for kylian mbappe data points
gmm_df_reduced_mbappe = gmm_df_reduced[gmm_df_reduced['name'].str.contains('

## Select columns of interest
gmm_df_reduced_mbappe = gmm_df_reduced_mbappe[['x', 'y', 'cluster', 'season'

# Display the DataFrame
gmm_df_reduced_mbappe
```

Out[634]:

	x	y	cluster	season
992	1.874440	0.119329	2	2018-2019
1067	1.526649	0.276810	2	2019-2020
1131	1.277519	-0.151959	2	2020-2021
1202	1.387960	-0.666055	2	2021-2022
1261	1.093180	0.008615	2	2022-2023

GMM: From the results we can see that Mbappe is found in the cluster 2 for the past 5 seasons.

Visualising to see where the points for Mbappe lie.

Create variables for each of the x, y coordinates of the kylian mbappe data points, to be plotted on the visualisation

In [569…

```
# Create variables for each of the x, y coordinates of the kylian mbappe dat
mbappe_x_1819 = float(df_reduced['x'][(df_reduced['season'] == '2018-2019')
mbappe_y_1819 = float(df_reduced['y'][(df_reduced['season'] == '2018-2019')

mbappe_x_1920 = float(df_reduced['x'][(df_reduced['season'] == '2019-2020')
mbappe_y_1920 = float(df_reduced['y'][(df_reduced['season'] == '2019-2020')

mbappe_x_2021 = float(df_reduced['x'][(df_reduced['season'] == '2020-2021')
mbappe_y_2021 = float(df_reduced['y'][(df_reduced['season'] == '2020-2021')

mbappe_x_2122 = float(df_reduced['x'][(df_reduced['season'] == '2021-2022')
mbappe_y_2122 = float(df_reduced['y'][(df_reduced['season'] == '2021-2022')

mbappe_x_2223 = float(df_reduced['x'][(df_reduced['season'] == '2022-2023')
mbappe_y_2223 = float(df_reduced['y'][(df_reduced['season'] == '2022-2023')
```

```
# Find the centroid (midpoint) of the 18/19-22/23 mbappe data points triangl
mbappe_x_centroid = (mbappe_x_1819 + mbappe_x_1920 + mbappe_x_2021 + mbappe_
mbhttp://localhost:8888/notebooks/QMUL/SEM-B/Data%20Analytics/Project/Course
```

In [636…

```
# Create variables for each of the x, y coordinates of the kylian mbappe dat
gmm_mbappe_x_1819 = float(gmm_df_reduced['x'][(gmm_df_reduced['season'] == '
gmm_mbappe_y_1819 = float(gmm_df_reduced['y'][(gmm_df_reduced['season'] == '

gmm_mbappe_x_1920 = float(gmm_df_reduced['x'][(gmm_df_reduced['season'] == '
gmm_mbappe_y_1920 = float(gmm_df_reduced['y'][(gmm_df_reduced['season'] == '

gmm_mbappe_x_2021 = float(gmm_df_reduced['x'][(gmm_df_reduced['season'] == '
gmm_mbappe_y_2021 = float(gmm_df_reduced['y'][(gmm_df_reduced['season'] == '

gmm_mbappe_x_2122 = float(gmm_df_reduced['x'][(gmm_df_reduced['season'] == '
gmm_mbappe_y_2122 = float(gmm_df_reduced['y'][(gmm_df_reduced['season'] == '

gmm_mbappe_x_2223 = float(gmm_df_reduced['x'][(gmm_df_reduced['season'] == '
gmm_mbappe_y_2223 = float(gmm_df_reduced['y'][(gmm_df_reduced['season'] == '

# Find the centroid (midpoint) of the 18/19-22/23 mbappe data points triangl
gmm_mbappe_x_centroid = (gmm_mbappe_x_1819 + gmm_mbappe_x_1920 + gmm_mbappe_
gmm_mbappe_y_centroid = (gmm_mbappe_y_1819 + gmm_mbappe_y_1920 + gmm_mbappe_
```

Visualising Mbappe Data Points - K-Means

In [632…

```
# Visualise the clustering, now including kylian mbappe data points
sns.set(style="white")
ax = sns.lmplot(x="x", y="y", hue='cluster', data = df_reduced, legend=False
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
ax.fig.suptitle('kylian mbappe data points K-means')
plt.annotate("kylian mbappe 18/19", (mbappe_x_1819, mbappe_y_1819))
plt.annotate("kylian mbappe 19/120", (mbappe_x_1920, mbappe_y_1920))
plt.annotate("kylian mbappe 20/21", (mbappe_x_2021, mbappe_y_2021))
plt.annotate("kylian mbappe 21/22", (mbappe_x_2122, mbappe_y_2122))
plt.annotate("kylian mbappe 22/23", (mbappe_x_2223, mbappe_y_2223))
plt.annotate("kylian mbappe Centroid", (mbappe_x_centroid, mbappe_y_centroid
ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel("PC 1", fontsize = 20)
plt.ylabel("PC 2", fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```

From the visualisation we can see that pints for Mbappe are in red cluster(3). The visualisation also plots the centroid of the Mbappe points.

Visualising Mbappe Data Points - GMM

In [637…

```
# Visualise the clustering, now including kylian mbappe data points
sns.set(style="white")
ax = sns.lmplot(x="x", y="y", hue='cluster', data = gmm_df_reduced, legend=F
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
ax.fig.suptitle('kylian mbappe data points GMM')
plt.annotate("kylian mbappe 18/19", (gmm_mbappe_x_1819, gmm_mbappe_y_1819))
plt.annotate("kylian mbappe 19/120", (gmm_mbappe_x_1920, gmm_mbappe_y_1920))
plt.annotate("kylian mbappe 20/21", (gmm_mbappe_x_2021, gmm_mbappe_y_2021))
plt.annotate("kylian mbappe 21/22", (gmm_mbappe_x_2122, gmm_mbappe_y_2122))
plt.annotate("kylian mbappe 22/23", (gmm_mbappe_x_2223, gmm_mbappe_y_2223))
plt.annotate("kylian mbappe Centroid", (gmm_mbappe_x_centroid, gmm_mbappe_y_
ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel("PC 1", fontsize = 20)
plt.ylabel("PC 2", fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```

From the visualisation we can see that pints for Mbappe are in Green cluster(2). The visualisation also plots the centroid of the Mbappe points.

Determining the distance of each player to each of the Mbappe points and from the centroid. This will help us to identify the players that are similar to mbappe

In [641…

```
# Determine the distance of each data point from each of the five mbappe dat
df_reduced['dist_mbappe_1819'] = np.sqrt( (mbappe_x_1819 - df_reduced['x'])*
df_reduced['dist_mbappe_1920'] = np.sqrt( (mbappe_x_1920 - df_reduced['x'])*
df_reduced['dist_mbappe_2021'] = np.sqrt( (mbappe_x_2021 - df_reduced['x'])*
df_reduced['dist_mbappe_2122'] = np.sqrt( (mbappe_x_2122 - df_reduced['x'])*
df_reduced['dist_mbappe_2223'] = np.sqrt( (mbappe_x_2223 - df_reduced['x'])*

# Determine the distance of each data point from the 17/18-19/20 mbappe data
df_reduced['dist_mbappe_centroid'] = np.sqrt( (mbappe_x_centroid - df_reduce
```

In [643…

```
# Determine the distance of each data point from each of the five mbappe dat
gmm_df_reduced['dist_mbappe_1819'] = np.sqrt( (gmm_mbappe_x_1819 - gmm_df_re
gmm_df_reduced['dist_mbappe_1920'] = np.sqrt( (gmm_mbappe_x_1920 - gmm_df_re
gmm_df_reduced['dist_mbappe_2021'] = np.sqrt( (gmm_mbappe_x_2021 - gmm_df_re
gmm_df_reduced['dist_mbappe_2122'] = np.sqrt( (gmm_mbappe_x_2122 - gmm_df_re
```

```
gmm_df_reduced['dist_mbappe_2223'] = np.sqrt( (gmm_mbappe_x_2223 - gmm_df_re

# Determine the distance of each data point from the 17/18-19/20 mbappe data
gmm_df_reduced['dist_mbappe_centroid'] = np.sqrt( (gmm_mbappe_x_centroid - g
```

7.2 Closest Performing players in last 5 seasons

In [572…
```
# Display the DataFrame
df_reduced.head()
```

Out[572]:

	x	y	cluster	name	season	minutes	matches_played	matches_
0	-0.364910	0.603039	2	adam zrelak	2018-2019	941.0	14.0	
1	-0.005329	-0.079077	1	admir mehmedi	2018-2019	1692.0	26.0	
2	0.320816	-0.164016	1	alassane plea	2018-2019	2503.0	34.0	
3	0.246158	0.477178	2	alfre finnbogason	2018-2019	1335.0	18.0	
4	0.028878	0.517735	2	anastasios donis	2018-2019	962.0	24.0	

5 rows × 21 columns

Identifying Mbappe's Cluster

Filtering the data for players in the same cluster as Mbappe.

In [639…
```
# Define Mbappe cluster number - K-Means
mbappe_cluster = int(df_reduced['cluster'][(df_reduced.season == '2018-2019'
mbappe_cluster
```

Out[639]: 3

In [644…
```
# Define Mbappe cluster number  - GMM
gmm_mbappe_cluster = int(gmm_df_reduced['cluster'][(gmm_df_reduced.season ==
gmm_mbappe_cluster
```

Out[644]: 2

In [647…
```
# Filter the DataFrame by players in the same cluster as Mbappe - K-means
df_mbappe_cluster = df_reduced[(df_reduced['cluster'] == mbappe_cluster)]

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
df_mbappe_cluster[~df_mbappe_cluster['name'].str.contains('kylian mbappe', c
## Closest performing players in all seasons
```

Out[647]:

	x	y	cluster	name	season	minutes	matches_played	match
108	1.419252	0.012619	3	robert lewandowski	2019-2020	2759.0	31.0	
225	1.377175	0.073961	3	robert lewandowski	2021-2022	2946.0	34.0	
117	1.244581	-0.342463	3	timo werner	2019-2020	2795.0	34.0	
865	1.234378	-0.351267	3	karim benzema	2021-2022	2593.0	32.0	
1365	1.171932	-0.322932	3	ciro immobile	2019-2020	3170.0	37.0	
129	1.066438	-0.031891	3	andre silva	2020-2021	2760.0	32.0	
135	1.257159	0.247114	3	erling haaland	2020-2021	2407.0	28.0	
1431	1.076399	-0.241458	3	cristiano ronaldo	2020-2021	2802.0	33.0	
85	1.066465	-0.347191	3	jadon sancho	2019-2020	2287.0	32.0	
1366	1.071850	-0.360946	3	cristiano ronaldo	2019-2020	2917.0	33.0	

10 rows × 21 columns

Players in Similar Cluster to Mbappe over past 5 seasons using K-means

In [649…
```
df_mbappe_cluster.shape
```

Out[649]:
```
(174, 21)
```

In [705…
```
# Filter the DataFrame by players in the same cluster as Mbappe - GMM
gmm_df_mbappe_cluster = gmm_df_reduced[(gmm_df_reduced['cluster'] == gmm_mba

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
gmm_df_mbappe_cluster[~gmm_df_mbappe_cluster['name'].str.contains('kylian mb
## Closest performing players in all seasons
```

Out[705]:

	x	y	cluster	name	season	minutes	matches_played	match
108	1.419252	0.012619	2	robert lewandowski	2019-2020	2759.0	31.0	
225	1.377175	0.073961	2	robert lewandowski	2021-2022	2946.0	34.0	
117	1.244581	-0.342463	2	timo werner	2019-2020	2795.0	34.0	
865	1.234378	-0.351267	2	karim benzema	2021-2022	2593.0	32.0	
1365	1.171932	-0.322932	2	ciro immobile	2019-2020	3170.0	37.0	
129	1.066438	-0.031891	2	andre silva	2020-2021	2760.0	32.0	
135	1.257159	0.247114	2	erling haaland	2020-2021	2407.0	28.0	
1431	1.076399	-0.241458	2	cristiano ronaldo	2020-2021	2802.0	33.0	
85	1.066465	-0.347191	2	jadon sancho	2019-2020	2287.0	32.0	
1366	1.071850	-0.360946	2	cristiano ronaldo	2019-2020	2917.0	33.0	

10 rows × 21 columns

Players in Similar Cluster to Mbappe over past 5 seasons using GMM

In [650…

```
gmm_df_mbappe_cluster.shape
```

Out[650]:

```
(128, 21)
```

The Kmeans algorithm clustered 174 players and GMM 128 Players into Mbappe's cluster

7.3 Closest Performing players for 22/23 season

Players in Similar Cluster to Mbappe for 22/23 season using Kmeans

In [653…

```
#Closest Performing players in current season 2022-2023
# Filter the DataFrame by players in the same cluster as Mbappe
df_mbappe_cluster_2223 = df_reduced[(df_reduced['cluster'] == mbappe_cluster

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
df_mbappe_cluster_2223[~df_mbappe_cluster_2223['name'].str.contains('kylian
## Closest performing players in all seasons
```

Out[653]:

	x	y	cluster	name	season	minutes	matches_played	match
1263	1.094470	-0.426474	3	lionel messi	2022-2023	1847.0	21.0	
603	1.307279	0.445813	3	erling haaland	2022-2023	2022.0	25.0	
941	0.994545	0.401820	3	robert lewandowski	2022-2023	1587.0	20.0	
1605	1.000987	0.497166	3	victor osimhen	2022-2023	1748.0	21.0	
1282	1.072350	0.606469	3	wissam ben yedder	2022-2023	1333.0	22.0	
271	0.606027	0.013018	3	randal kolo muani	2022-2023	1779.0	22.0	
1584	0.587614	-0.134980	3	khvicha kvaratskhelia	2022-2023	1551.0	21.0	
1256	0.582418	0.080077	3	jonathan david	2022-2023	2112.0	25.0	
606	0.557493	-0.036228	3	harry kane	2022-2023	2325.0	26.0	
275	0.576532	0.153085	3	serge gnabry	2022-2023	1205.0	23.0	

10 rows × 21 columns

Number of Players in Similar Cluster to Mbappe for 22/23 season using Kmeans

In [654…
```
df_mbappe_cluster_2223.shape
```

Out[654]:
```
(18, 21)
```

As per K-means this season there are 18 players that are similar to Mbappe interms of performance

Players in Similar Cluster to Mbappe for 22/23 season using GMM

In [656…
```
#Closest Performing players in current season 2022-2023
# Filter the DataFrame by players in the same cluster as Mbappe
gmm_df_mbappe_cluster_2223 = gmm_df_reduced[(gmm_df_reduced['cluster'] == gm

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
gmm_df_mbappe_cluster_2223[~gmm_df_mbappe_cluster_2223['name'].str.contains(
## Closest performing players in all seasons
```

Out[656]:

	x	y	cluster	name	season	minutes	matches_played	match
1263	1.094470	-0.426474	2	lionel messi	2022-2023	1847.0	21.0	
603	1.307279	0.445813	2	erling haaland	2022-2023	2022.0	25.0	
941	0.994545	0.401820	2	robert lewandowski	2022-2023	1587.0	20.0	
1605	1.000987	0.497166	2	victor osimhen	2022-2023	1748.0	21.0	
1282	1.072350	0.606469	2	wissam ben yedder	2022-2023	1333.0	22.0	
271	0.606027	0.013018	2	randal kolo muani	2022-2023	1779.0	22.0	
1584	0.587614	-0.134980	2	khvicha kvaratskhelia	2022-2023	1551.0	21.0	
1256	0.582418	0.080077	2	jonathan david	2022-2023	2112.0	25.0	
606	0.557493	-0.036228	2	harry kane	2022-2023	2325.0	26.0	
275	0.576532	0.153085	2	serge gnabry	2022-2023	1205.0	23.0	

10 rows × 21 columns

Number of Players in Similar Cluster to Mbappe for 22/23 season using Kmeans

In [658…

```
gmm_df_mbappe_cluster_2223.shape
```

Out[658]:

```
(14, 21)
```

As per GMM this season there are 14 players that are similar to Mbappe interms of performance

7.4 Closest Performing Young Players(u23) for 22/23 season

Using K-means

In [666…

```
#Closest Performing Young players (Under 25) in current season 2022-2023
# Filter the DataFrame by players in the same cluster as Mbappe
df_mbappe_cluster_u23 = df_reduced[(df_reduced['cluster'] == mbappe_cluster)

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
df_mbappe_cluster_u23[~df_mbappe_cluster_u23['name'].str.contains('kylian mb
## Closest performing players in all seasons
```

Out[666]:

	x	y	cluster	name	season	minutes	matches_played	match
603	1.307279	0.445813	3	erling haaland	2022-2023	2022.0	25.0	
1584	0.587614	-0.134980	3	khvicha kvaratskhelia	2022-2023	1551.0	21.0	
1256	0.582418	0.080077	3	jonathan david	2022-2023	2112.0	25.0	

3 rows × 21 columns

In [663…
```
df_mbappe_cluster_u23.shape
```

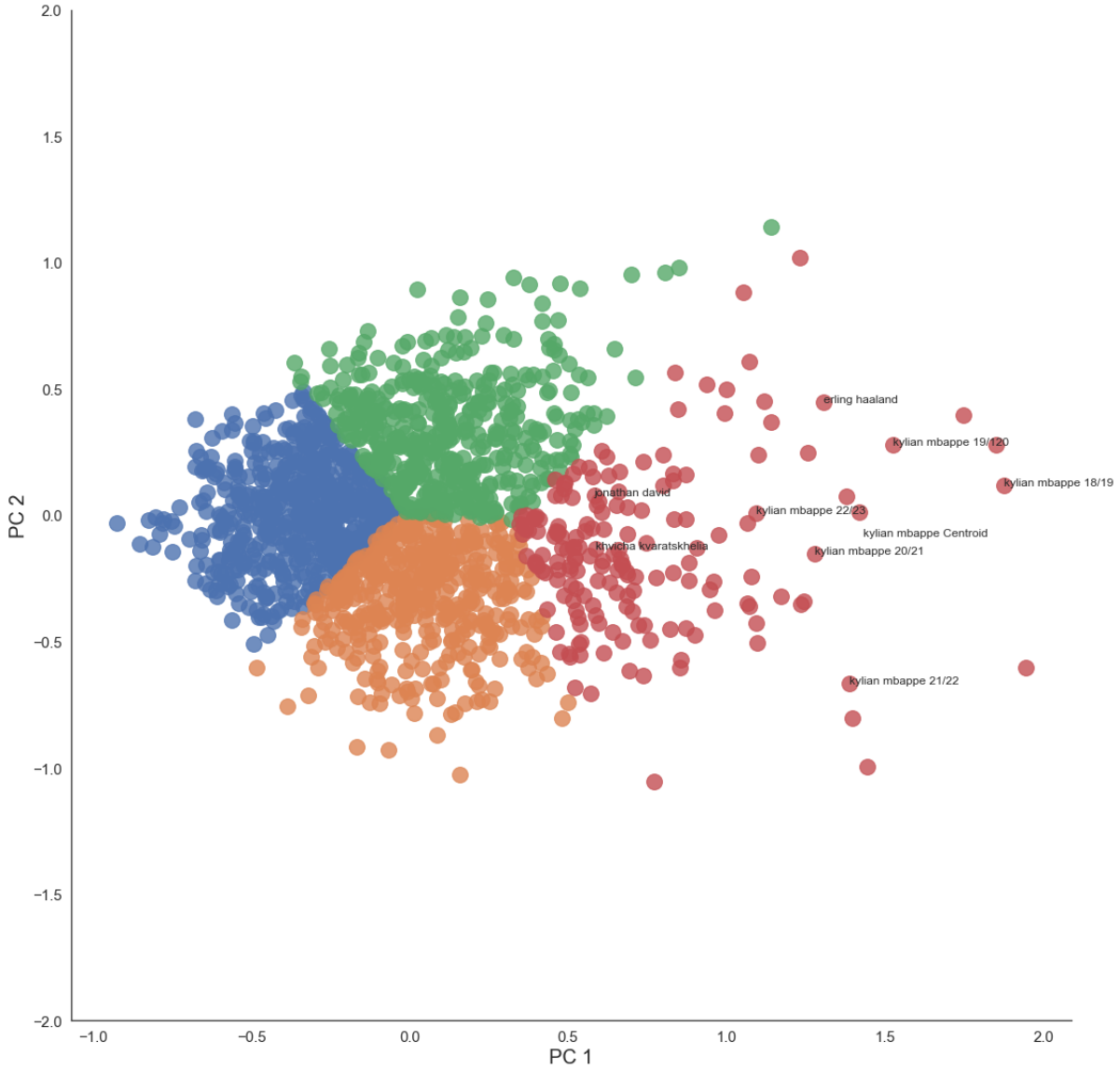
Out[663]: (3, 21)

As per Kmeans there are 3 players Erling Haaland, khvicha kvaratskhelia and Jonathan David, are similar to Mbappe

Visualising the results

In [668…
```
# Visualise the clustering, now including Mbappe data points
sns.set(style="white")
ax = sns.lmplot(x="x", y="y", hue='cluster', data = df_reduced, legend=False
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
for x, y, s in zip(df_mbappe_cluster_u23.head(10).x, df_mbappe_cluster_u23.h
    texts.append(plt.text(x, y, s))

plt.annotate("kylian mbappe 18/19", (mbappe_x_1819, mbappe_y_1819))
plt.annotate("kylian mbappe 19/120", (mbappe_x_1920, mbappe_y_1920))
plt.annotate("kylian mbappe 20/21", (mbappe_x_2021, mbappe_y_2021))
plt.annotate("kylian mbappe 21/22", (mbappe_x_2122, mbappe_y_2122))
plt.annotate("kylian mbappe 22/23", (mbappe_x_2223, mbappe_y_2223))
plt.annotate("kylian mbappe Centroid", (mbappe_x_centroid, mbappe_y_centroid

ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel("PC 1", fontsize = 20)
plt.ylabel("PC 2", fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```



Using GMM

In [664…
```
#Closest Performing Young players (Under 25) in current season 2022-2023
# Filter the DataFrame by players in the same cluster as Mbappe
gmm_df_mbappe_cluster_u23 = gmm_df_reduced[(gmm_df_reduced['cluster'] == gmm

# Display the DataFrame
# df_mbappe_cluster.sort_values(by=['dist_mbappe_centroid'], ascending=True)
gmm_df_mbappe_cluster_u23[~gmm_df_mbappe_cluster_u23['name'].str.contains('k
## Closest performing players in all seasons
```

Out[664]:

	x	y	cluster	name	season	minutes	matches_played	match
603	1.307279	0.445813	2	erling haaland	2022-2023	2022.0	25.0	
1584	0.587614	-0.134980	2	khvicha kvaratskhelia	2022-2023	1551.0	21.0	
1256	0.582418	0.080077	2	jonathan david	2022-2023	2112.0	25.0	

3 rows × 21 columns

In [665…
```
gmm_df_mbappe_cluster_u23.shape
```

Out[665]: (3, 21)

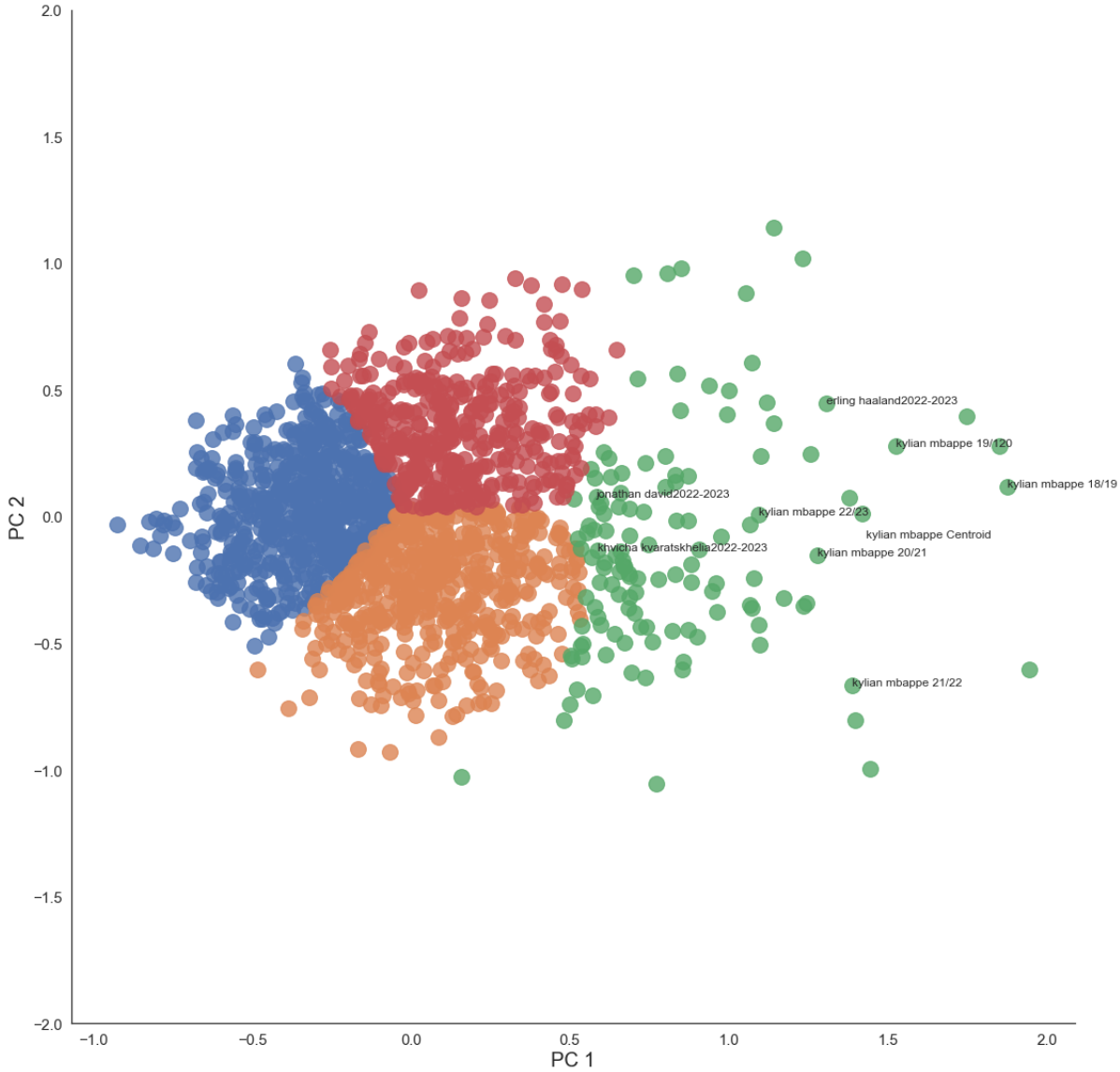
As per GMM there are 3 players Erling Haaland, khvicha kvaratskhelia and Jonathan David, are similar to Mbappe

Visualising the results

In [669…

```
# Visualise the clustering, now including Mbappe data points
sns.set(style="white")
ax = sns.lmplot(x="x", y="y", hue='cluster', data = gmm_df_reduced, legend=F
fit_reg=False, size = 15, scatter_kws={"s": 250})
texts = []
for x, y, s in zip(gmm_df_mbappe_cluster_u23.head(10).x, gmm_df_mbappe_clust
    texts.append(plt.text(x, y, s))

plt.annotate("kylian mbappe 18/19", (gmm_mbappe_x_1819, gmm_mbappe_y_1819))
plt.annotate("kylian mbappe 19/120", (gmm_mbappe_x_1920, gmm_mbappe_y_1920))
plt.annotate("kylian mbappe 20/21", (gmm_mbappe_x_2021, gmm_mbappe_y_2021))
plt.annotate("kylian mbappe 21/22", (gmm_mbappe_x_2122, gmm_mbappe_y_2122))
plt.annotate("kylian mbappe 22/23", (gmm_mbappe_x_2223, gmm_mbappe_y_2223))
plt.annotate("kylian mbappe Centroid", (gmm_mbappe_x_centroid, gmm_mbappe_y_

ax.set(ylim=(-2, 2))
plt.tick_params(labelsize=15)
plt.xlabel("PC 1", fontsize = 20)
plt.ylabel("PC 2", fontsize = 20)
plt.show()
```

```
/Users/sayedsohail/opt/anaconda3/lib/python3.9/site-packages/seaborn/regress
ion.py:581: UserWarning: The `size` parameter has been renamed to `height`;
please update your code.
  warnings.warn(msg, UserWarning)
```



8. Conclusion

The outcome of this analysis was to identify players similar to Kylian Mbappe over past 5 seasons across top 5 European Football Leagues.

- We found that Erling Haaland, khvicha kvaratskhelia and Jonathan David are young players under 23 that have similar performances this season(2022-2023).
- Top 5 Players Lionel Messi, Robert Lewandowski, Erling Haaland, victor osimhen and wissam ben yedder are perfoming smiliar to Mbappe this season(2022-2023).
- Top 5 players over the five seasons 2018-2019 with similar performances like Mbappe are:
  - robert lewandowski (2019-2020)
  - robert lewandowski (2021-2022)
  - timo werner (2019-2020)
  - karim benzema (2021-2022)
  - ciro immobile (2019-2020)
  - andre silva (2020-2021)
- Over Past 5 seasons K-Means algorithm clustered 174 players and GMM 128 Players into Mbappe's cluster.

- In 2022-2023 season, As per GMM 14 players have similar performances to Mbappe.
- In 2022-2023 season, As per K-Means 13 players have similar performances to Mbappe.

9. Future Improvements or Directions

- The notebook could be enhanced to inculde tranfer data and could be used in player recruitment, by comparing stats from similar players and use it to recruit next best players for the club. The player vaulation model could be used to estimate player's market value.
- This code currently mesures proximity to clusters but can be cinverted to percentage score to find how similar a player is to the comparison player
- Implement other clustering algorithms like DBSCAN and Hierarchical Clustering.
- Improve the identification of the characteristics that influence Mbappe's playing style. Currently, the model simplifies the selected performance metrics into two dimensions, which are chosen arbitrarily for visualization purposes. It doesn't explain why a player is achieving results similar to Mbappe's, only that they belong to the same cluster and are in close proximity.