Clustering - Football

- Aim of the Project
- Data Collection & Cleaning
  - Data Source
  - Data Cleaning
  - Flattening MultiIndex Columns
  - Normalizing Column Names
  - Cleaning the squad Column
- Feature Engineering
  - Defensive Features
  - Attacking Features
  - Possession & Passing Features
- Applying K-means Clustering
  - Finding the Optimal K (Elbow Method)
  - Visualizing Clusters (PCA)
- Example Findings

Aim of the Project

🔗 **App link**: [Streamlit App]

- This project aims to cluster football **teams** and **players** based on playing styles and performance metrics.
  - For **teams**: The goal is to group clubs with similar tactical styles, helping analyse their strengths, weaknesses, and unique identities.
  - For **players**: The aim is to identify players with similar profiles, spot standout performers, and understand their strengths and weaknesses in context.

Data Collection & Cleaning

Data Source

Data was scraped from [FBRef] using `pandas.read_html()`.

Example:

```
# Example Query to scrape Champions league data table
standard_stats = pd.read_html(
    'https://fbref.com/en/comps/8/stats/Champions-League-Stats',
    attrs={'id': 'stats_squads_standard_for'}
)[0]
```

- The same process was applied to player stats.

Data Cleaning

- Scraped data often contains messy formatting, such as:
  - MultiIndex column headers
  - "Unnamed" columns
  - Irregular naming conventions

  A generic cleaning script was used across all tables to ensure consistency.

Flattening MultiIndex Columns

```
standard_stats.columns = [
    '_'.join(
        [lvl if not str(lvl).startswith('Unnamed') else '' for lvl in col]
    ).strip('_')
    for col in standard_stats.columns.values
]
```

This joins multi-level column headers with underscores and removes "Unnamed" levels.

Normalizing Column Names

```
def clean_column(col):
    col = col.strip().lower()
    col = re.sub(r'\s+', '_', col)
    col = re.sub(r'_+', '_', col)
    col = col.strip('_')
    return col

standard_stats.columns = [clean_column(col) for col in standard_stats.columns]
```

This standardizes all column names by:

- Lowercasing
- Replacing spaces with underscores
- Removing duplicate/trailing underscores

Cleaning the `squad` Column

```
standard_stats['squad'] = standard_stats['squad'].apply(
    lambda x: x.split(' ', 1)[1] if isinstance(x, str) and ' ' in x else x
)
```

This removes country prefixes (e.g. `"ENG Manchester City"` → `"Manchester City"`).

Feature Engineering

- Most scraped stats are raw totals. To normalize for playing time, I calculated **per 90-minute** metrics, making comparisons fair across teams or players with different minutes played.

$$\text{Per 90 Stat} = \left(\frac{\text{Raw Stat}}{\text{Minutes Played}}\right) \times 90$$

```
squad_goalkeeping['performance_sota90'] = squad_goalkeeping['performance_sota']/squad_goalkeeping['playing_time_min']*90

squad_defensive_actions['blocks_blocks90'] = squad_defensive_actions['blocks_blocks']/squad_defensive_actions['playing_time_min']
squad_defensive_actions['tkl+int90'] = squad_defensive_actions['tkl+int']/squad_defensive_actions['playing_time_min']
squad_defensive_actions['clr90'] = squad_defensive_actions['clr']/squad_defensive_actions['playing_time_min']

defensive_features_df = (squad_goalkeeping[['squad','performance_ga90','performance_sota90']]
```

```
.merge(squad_defensive_actions[['squad','blocks_blocks90','tkl+int90','clr90']],on='squad' ,how='left'))
```

- Additonally, I grouped relevent attributes together based on the feature set I was creating, so example above is defensive features
- Similar features created for players

Defensive Features

`Index(['squad', 'performance_ga90', 'performance_sota90', 'blocks_blocks90','tkl+int90', 'clr90']`

- Goals conceeded, Shots conceeded, Blocks, Tackles, Clearances

Attacking Features

`['squad', 'per_90_minutes_gls', 'per_90_minutes_xg', 'standard_sh/90','standard_sot/90']`

- Goals, xG, Shots, Shots on target

Possession & Passing Features

`Index(['squad', 'poss', 'touches_att_3rd90', 'total_cmp90', 'total_cmp%','prgp90']`

Possession, Touches in attacking 3rd, Pass completion, Pass completion %, progressive passes

Applying K-means Clustering

Now we have our data ready, we will apply K-means clustering.

```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply k-means with specified k
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

- First we standardise each feature (mean = 0, std=1) ensures all features contribute equally.
- Then we apply **K-Means** clustering to the scaled data.
- Then assisn each row (team/player) to one of `k` clusters.

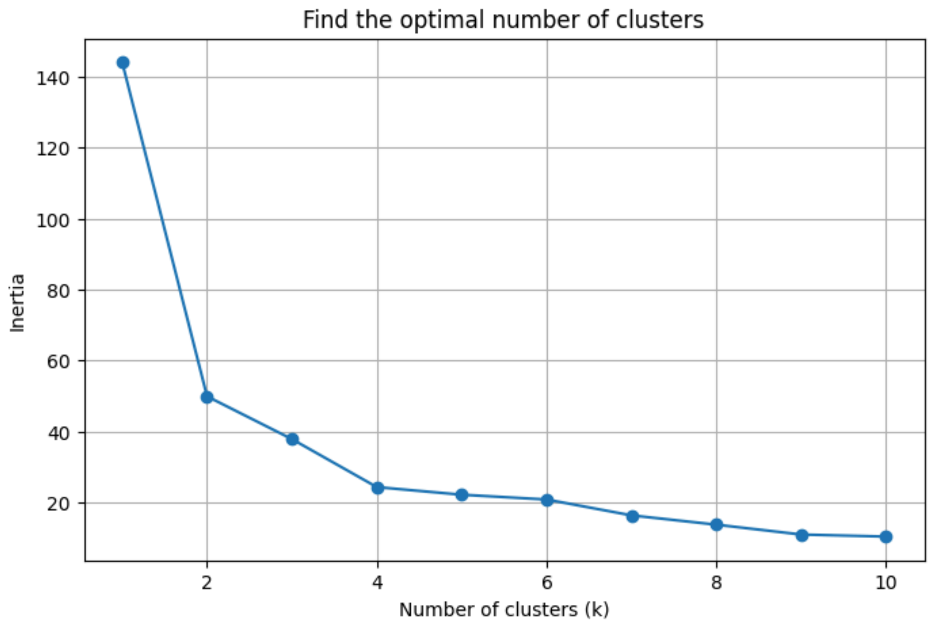
Finding the Optimal K (Elbow Method)

- We can identify the optimal K-Value (Number of Clusters) by using the Elbow Method
- First we need to calculate inertia, this is the sum of squared distances between each data point and its assigned cluster centre

$$\text{Inertia} = \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{c_i}\|^2$$

Where:

- xi is a data point
- µci is the centroid of its assigned cluster
- Lower inertia means points are closer to their cluster centers.



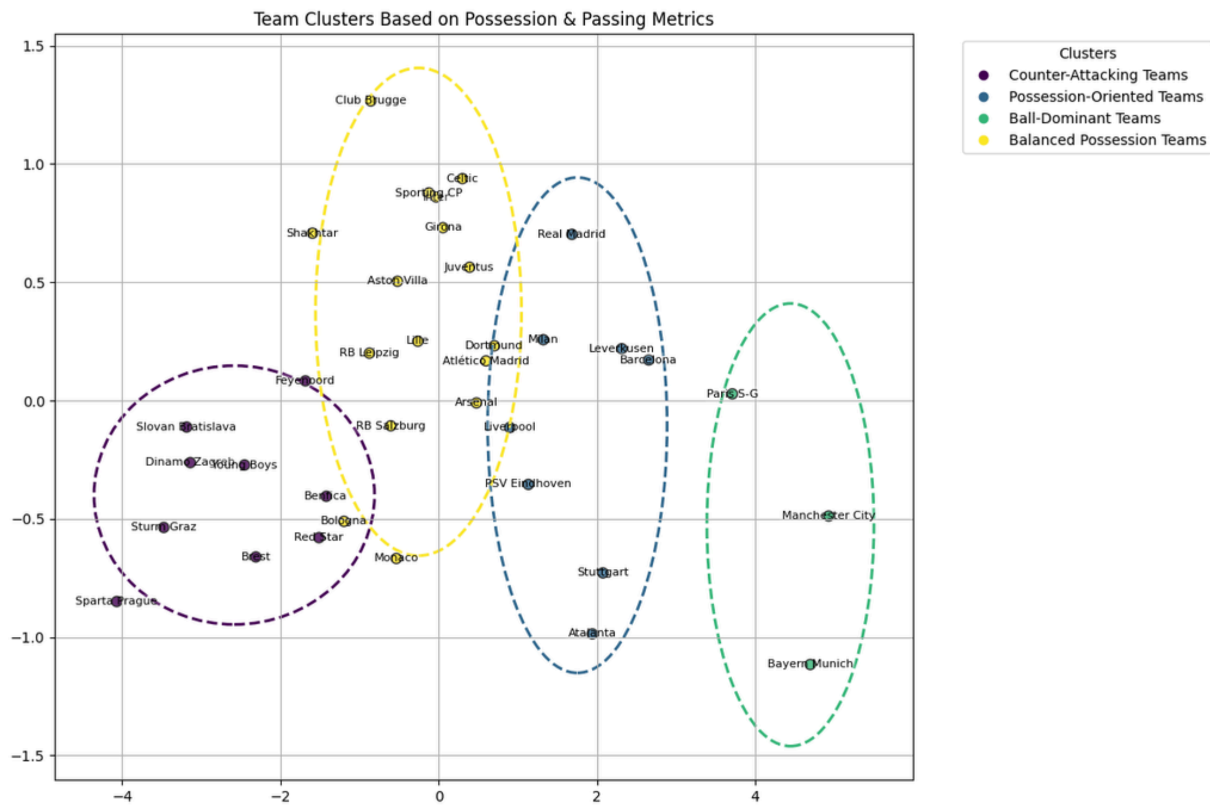
- I plotted `inertia vs k` and look for the **"elbow point"**, this is the optimal K value as any higher K has diminishing returns.
- However I will set up the App so the user can freely choose any cluster amount.

Visualizing Clusters (PCA)

- Since the data has many dimensions, I used **PCA (Principal Component Analysis)** to reduce it to 2 or 3 dimensions for visualization.

```
# PCA for dimensionality reduction to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

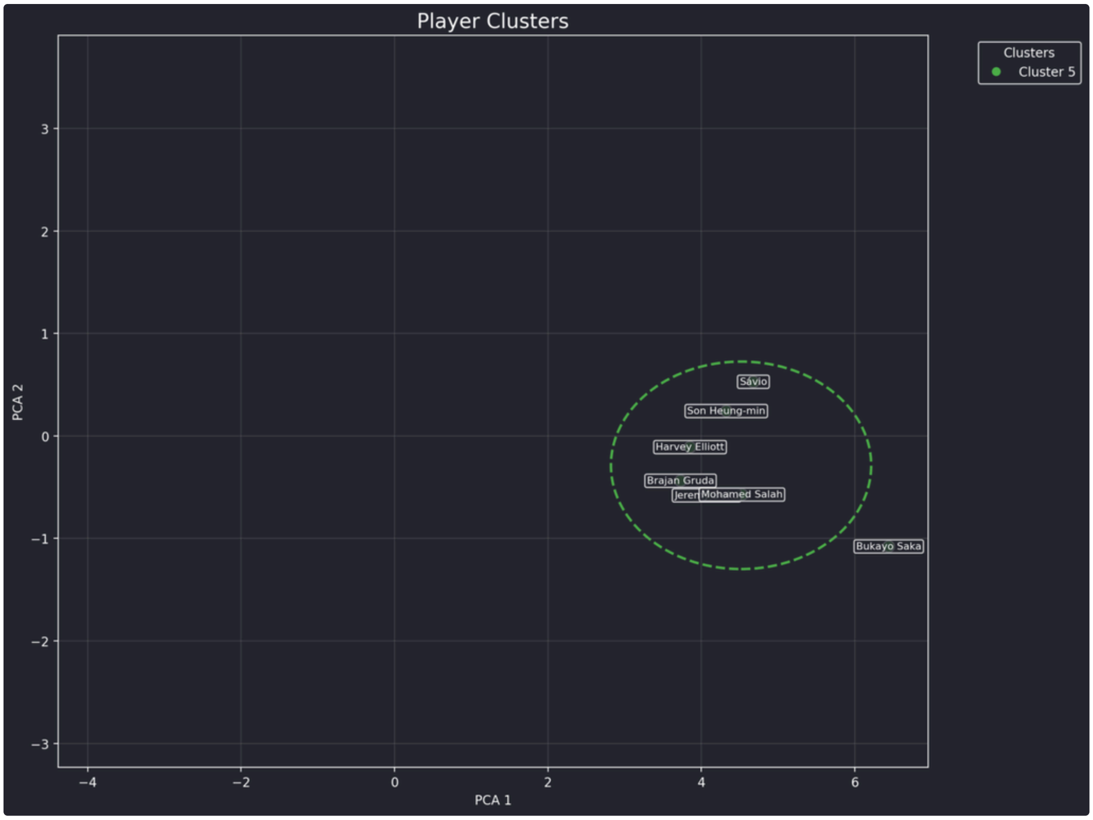
- This preserves structure while simplifying the view.
- The app includes **both 2D and 3D PCA plots** to explore clusters interactively.



- From this PCA plot, we can see the clusters very clearly (K=4), and we can even define playstyles based on the cluster averages.
- Additionally in the app, I also implemented a 3D interactive Visualisation for the user to explore. This somewhat resolves the issue of having overlapping clusters in the 2D space.
- I did a similar process for players, any players in the same cluster as eachother have similar profiles for that specific feature.

Example Findings

- All of this analysis has been deployed on a streamlit App, the user can play around with it.
- An interesting finding is Bukayo Saka, when K=18, features for creativity, we get this output (hide other clusters for clarity)



- The Streamlit app lets users explore clusters dynamically.
- An interesting insight:
  When clustering players by **creativity features** with **K=18**, **Bukayo Saka** appeared at the **edge of his cluster**, showing he's already very unique.
- At **K=19**, Saka was placed in a cluster of his own, suggesting his creative profile is **unmatched** in the dataset.
- Even at K=18, he shares a cluster with elite players like **Salah** and **Son**, indicating elite creativity, but Saka's data shows he’s on another level.