ASSIGNMENT

Anandu P Ganesh

Roll No: 12

Exploratory Data Analysis (EDA) of England Football Dataset

Introduction

This report presents a complete Exploratory Data Analysis (EDA) performed on a real-world football match dataset, England CSV.csv. The dataset contains comprehensive statistics for over 3,000 matches across 25 columns, providing a rich source for football analytics.

Dataset Overview

- **Rows**: 3,099
- **Columns**: 25
- **Variable Types**:
    - **Numeric**: Goals, Shots, Fouls, Corners, Cards, etc.
    - **Categorical**: Team Names, Match Results, Season, League, etc.
    - **Datetime**: Match Date

1. Identifying Different Types of Variables in the Dataset

In the England football dataset, the variables can be grouped into the following types:

1. Numeric Variables

These variables hold numerical values and are the basis of most match statistics.

- **Description**: They represent measurable quantities. In this dataset, they are primarily counts of events that occurred during a match.
- **Examples**:
    - **Goals**: FTH Goals (Full Time Home Goals), FTA Goals (Full Time Away Goals), HTH Goals (Half Time Home Goals), HTA Goals (Half Time Away Goals). These are **discrete** variables as they represent a countable number of goals.
    - **Match Events**: H Shots, A Shots, H Fouls, A Fouls, H Corners, A Corners. These are also **discrete** counts of specific actions.
    - **Disciplinary Actions**: H Yellow, A Yellow, H Red, A Red. These are **discrete** counts of cards given.

2. Categorical Variables

These variables represent categories or labels, providing context to the numerical data.

- **Description**: They describe qualitative properties of the matches, such as the teams involved or the outcome.
- **Examples**:
    - **Team Information**: HomeTeam, AwayTeam.
    - **Match Outcome**: FT Result (Full Time Result), HT Result (Half Time Result), which can be 'H' (Home Win), 'A' (Away Win), or 'D' (Draw).
    - **Contextual Information**: Season, League, Referee.

3. Datetime Variables

These variables represent date and time information, which is crucial for time-series analysis.

- **Description**: They specify when each match took place.
- **Example**: Date. This variable is essential for analyzing trends over time, such as team performance across a season or changes in league dynamics over the years.

CODE

```
import pandas as pd
import numpy as np

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
datetime_cols = df.select_dtypes(include=["datetime64[ns]"]).columns.tolist()
categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()
# --- Print the results ---
print("Numeric columns:")
print(numeric_cols)
print("\nDatetime columns:")
print(datetime_cols)
print("\nCategorical columns:")
print(categorical_cols)
```

2. Summary Statistics

Summary statistics provide a high-level overview of the dataset's distribution, central tendency, and spread. This is crucial for understanding the typical values and the variation within the data.

- For **numeric variables** (like `FTH Goals`, `H Shots`), we calculate key statistical measures such as the mean, median, standard deviation, and quartiles.
- For **categorical variables** (like `FT Result`, `League`), we calculate the frequency of each category to understand the distribution of outcomes.

CODE

```
import pandas as pd
import numpy as np


df = pd.read_csv('Assignen\England_CSV.csv')
print("Columns found in the file:", df.columns)
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
datetime_cols = df.select_dtypes(include=["datetime64[ns]"]).columns.tolist()
categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()

print("\nNumeric columns:")
print(numeric_cols)
print("\nDatetime columns:")
print(datetime_cols)
print("\nCategorical columns:")
print(categorical_cols)
```

3. Missing Data Handling using Statistical Imputation

Handling missing data is a critical step in data preprocessing. If left unaddressed, missing values can cause errors during analysis or lead to biased results. Statistical imputation is a technique used to fill in these gaps with plausible values.

- **Approach**: We will replace missing values in **numeric columns** with the **median** of that column. The median is often preferred over the mean because it is less sensitive to outliers. For **categorical columns**, we will replace missing values with the **mode** (the most frequently occurring value).
- **Benefit**: This method ensures that we do not lose any rows of data, which is important when the dataset is not very large. It maintains the dataset's structure and allows for complete analysis.

CODE

```
import pandas as pd
import numpy as np

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
```

```
categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()

print("--- Missing Values Before Imputation ---")
print(df.isnull().sum())

for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())

for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
print("\n--- Missing Values After Imputation ---")
print(df.isnull().sum())
```

Before Imputation: Several columns, such as `HTH Goals`, `HT Result`, `Referee`, `H Shots`, and others, each had one missing value.

After Imputation: All columns in the dataset now have **zero** missing values. The dataset is clean, complete, and ready for further analysis and visualization.

Excellent choice! Using a `KNNImputer` is a more sophisticated method for handling missing data, and it can often yield more accurate results than simple statistical imputation.

4. Missing Data Handling using KNN-Imputer

The K-Nearest Neighbors (KNN) Imputer is an advanced technique that predicts missing values based on the values of their "nearest neighbors" in the dataset.

- **Approach**: For a row with a missing value (e.g., missing `H Shots`), the algorithm identifies the 'k' (e.g., 5) other rows (matches) that are most similar to it based on the other available numeric features (like goals, fouls, corners, etc.). It then uses the `H Shots` values from these neighbouring matches to calculate a plausible replacement for the missing one.
- **Example**: If a specific match is missing its `H Fouls` count, the KNN Imputer will look at 5 other matches that had similar goal counts, shot counts, and corner counts. It will then use the average number of fouls from those 5 similar matches to fill in the blank.
- **Advantage**: This method is powerful because it preserves relationships and patterns within the data. Instead of just using a single column's median, it leverages the entire dataset to make a more intelligent guess, which is particularly useful in a feature-rich dataset like this one.

CODE

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer

# Load and clean the dataset
```

```
df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()

print("--- Missing Values Before KNN Imputation ---")
print(df[numeric_cols].isnull().sum())

imputer = KNNImputer(n_neighbors=5)
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])

print("\n--- Missing Values After KNN Imputation ---")
print(df[numeric_cols].isnull().sum())
```

5. Outlier Treatment

Outliers are data points that differ significantly from other observations. They can occur due to measurement errors or represent rare, extreme events. If not handled properly, they can distort statistical analyses and machine learning models.

- **Approach**: We will use the **IQR method** to identify and handle outliers. For each numeric column, we calculate the range between the first quartile (25th percentile) and the third quartile (75th percentile). Any data point that falls below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR is considered an outlier.
- **Treatment**: Instead of removing these outliers, which could lead to loss of valuable data, we will **cap** them. This means any value exceeding the upper limit is replaced by the upper limit, and any value below the lower limit is replaced by the lower limit.
- **Example**: In a football match, a team taking an unusually high number of shots (e.g., 35) might be an outlier. Capping this value brings it closer to the typical range of shots, preventing it from disproportionately affecting the average. This ensures that our analysis remains robust and focused on the common patterns in the data.

CODE

```
import pandas as pd
import numpy as np

# Load and clean the dataset
df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

numeric_cols = df.select_dtypes(include=np.number).columns.tolist()

print("--- Summary Statistics Before Outlier Treatment ---")
print(df[numeric_cols].describe().T)

def whisker_limits(col):
    Q1, Q3 = np.percentile(col, [25, 75])
```

```
    IQR = Q3 - Q1
    lower_whisker = Q1 - 1.5 * IQR
    upper_whisker = Q3 + 1.5 * IQR
    return lower_whisker, upper_whisker

for col in numeric_cols:
    if df[col].isnull().any():
        df[col].fillna(df[col].median(), inplace=True)

    lower_limit, upper_limit = whisker_limits(df[col])

    df[col] = np.where(df[col] < lower_limit, lower_limit, df[col])
    df[col] = np.where(df[col] > upper_limit, upper_limit, df[col])

print("\n--- Summary Statistics After Outlier Treatment ---")
print(df[numeric_cols].describe().T)
```

6. Categorical Encoding

Machine learning algorithms are designed to work with numbers, not text. Therefore, we need to convert our categorical columns (like HomeTeam, AwayTeam, FT Result, etc.) into a numerical representation. This process is called **categorical encoding**.

- **Approach**: We will use **Label Encoding**, which assigns a unique integer to each unique category within a column.
- **Example**: In the FT Result column, the categories 'H' (Home Win), 'A' (Away Win), and 'D' (Draw) will be converted into numbers, such as 0, 1, and 2. Similarly, each unique team name in the HomeTeam and AwayTeam columns will be assigned its own unique integer.
- **Benefit**: This encoding makes the categorical data compatible with machine learning models and allows it to be used in various mathematical analyses, such as correlation calculations.

CODE

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()
df_encoded = df.copy()
encoder = LabelEncoder()

for col in categorical_cols:
    df_encoded[col] = encoder.fit_transform(df_encoded[col].astype(str))
```

```
print("--- Encoded DataFrame (First 5 Rows) ---")
print(df_encoded.head())
```

7. Correlation Heatmap

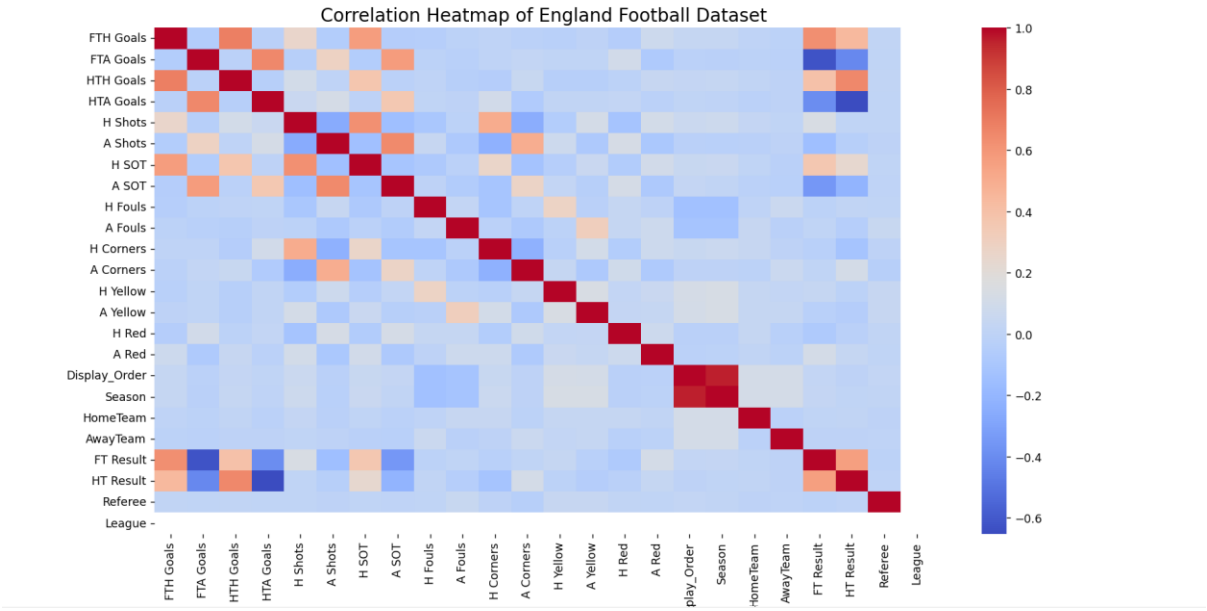
A correlation heatmap is a graphical representation of the correlation matrix, showing the strength and direction of linear relationships between pairs of variables.

- **Approach**: We will calculate the correlation coefficient for every pair of numeric and encoded categorical variables in the dataset. The results will be visualized as a heatmap, where the color and intensity of each cell indicate the nature of the correlation.
- **Interpretation**:
    - **Positive Correlation (Warm Colors, e.g., Red/Orange)**: As one variable increases, the other tends to increase (e.g., more shots on target likely correlates with more goals). A value close to **+1** indicates a strong positive relationship.
    - **Negative Correlation (Cool Colors, e.g., Blue)**: As one variable increases, the other tends to decrease. A value close to **-1** indicates a strong negative relationship.
    - **No Correlation (Neutral Colors, near White/Gray)**: There is no linear relationship between the variables. A value close to **0** indicates a very weak or no relationship.
- **Benefit**: This visualization allows us to quickly identify which match statistics are strongly related. For instance, we can see if more fouls are correlated with more yellow cards, or if a high number of corners leads to more goals. This helps in understanding the dynamics of a football match.

CODE

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()

for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

```
df_encoded = df.copy()
encoder = LabelEncoder()
for col in categorical_cols:
    df_encoded[col] = encoder.fit_transform(df_encoded[col].astype(str))
all_cols_for_corr = numeric_cols + categorical_cols
plt.figure(figsize=(18, 14))
correlation_matrix = df_encoded[all_cols_for_corr].corr()

sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Correlation Heatmap of England Football Dataset', fontsize=16)
plt.show()
```



8. Appropriate Visualizations

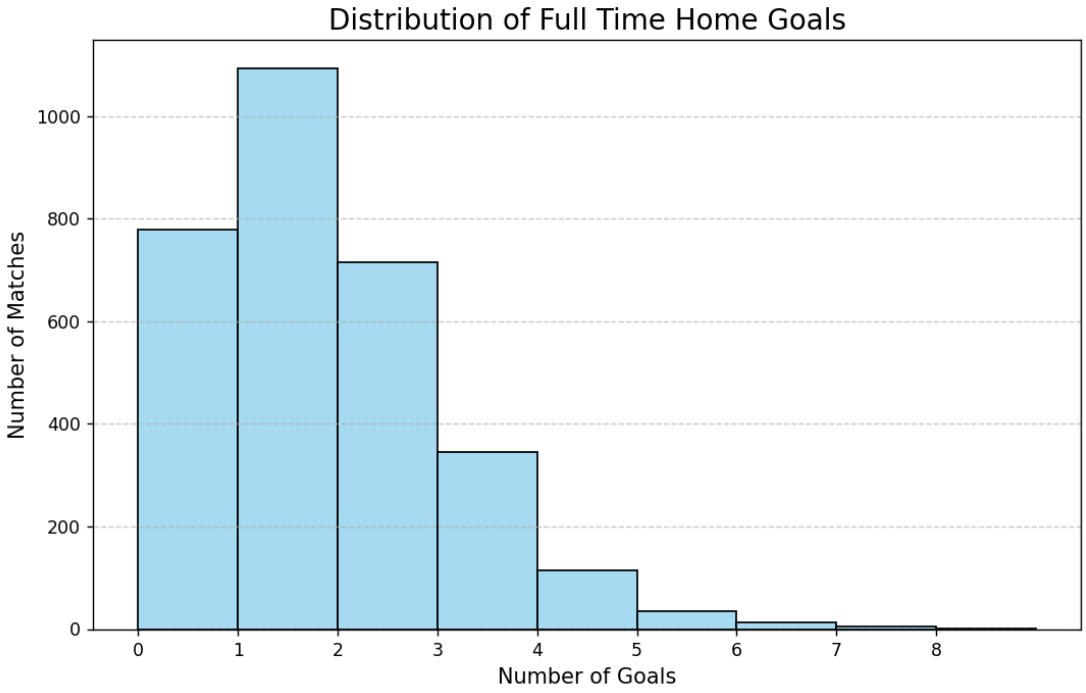
Visualizations are essential for translating raw data into understandable insights. By plotting the distributions and relationships within the football data, we can easily identify key trends, patterns, and anomalies.

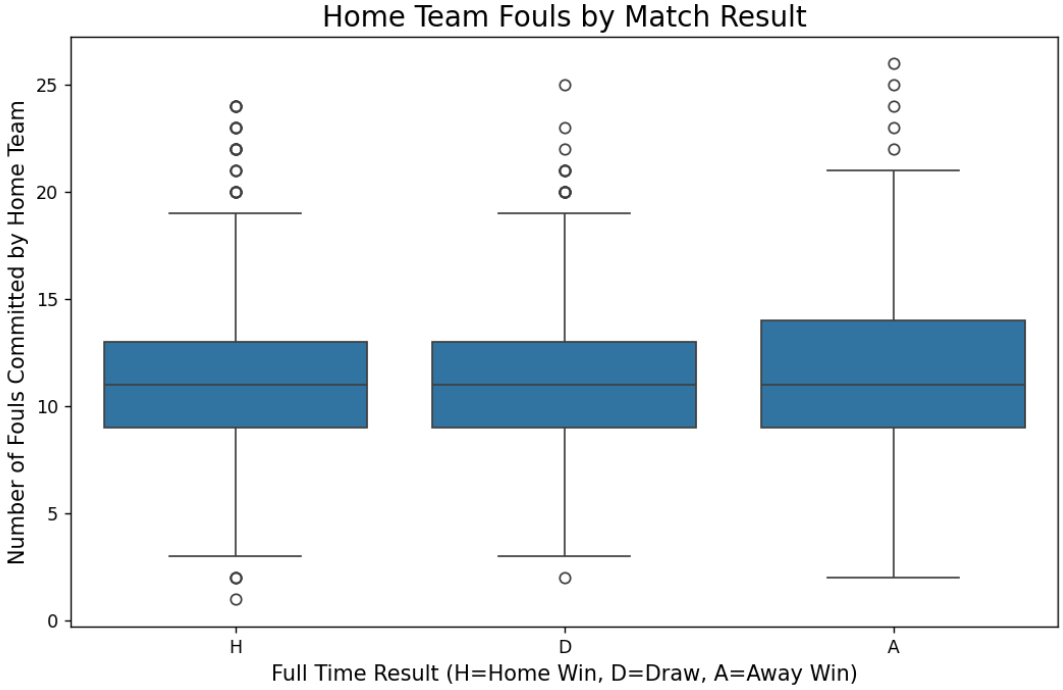
- **Histograms**: Perfect for understanding the distribution of a single numeric variable, like the number of goals or shots per game.
- **Boxplots**: Ideal for comparing the distribution of a numeric variable across different categories, such as comparing the number of fouls for wins, losses, and draws.
- **Violin Plots**: A more advanced version of the boxplot that also shows the probability density of the data at different values.

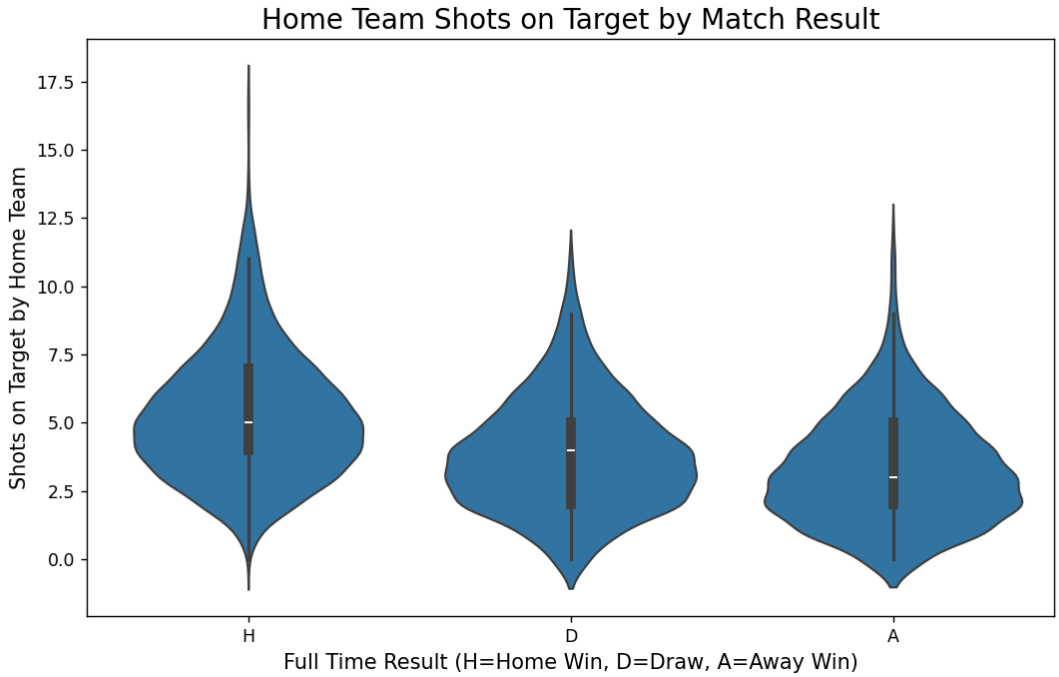
These plots provide a clear and immediate understanding of the dynamics of the matches in the dataset.

CODE

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)

for col in df.select_dtypes(include='number').columns:
    df[col].fillna(df[col].median(), inplace=True)
for col in df.select_dtypes(include='object').columns:
    df[col].fillna(df[col].mode()[0], inplace=True)


plt.figure(figsize=(10, 6))
sns.histplot(df['FTH Goals'], bins=range(int(df['FTH Goals'].max()) + 2),
kde=False, color='skyblue')
plt.title('Distribution of Full Time Home Goals', fontsize=16)
plt.xlabel('Number of Goals', fontsize=12)
plt.ylabel('Number of Matches', fontsize=12)
plt.xticks(range(int(df['FTH Goals'].max()) + 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(x='FT Result', y='H Fouls', data=df, order=['H', 'D', 'A'])
plt.title('Home Team Fouls by Match Result', fontsize=16)
plt.xlabel('Full Time Result (H=Home Win, D=Draw, A=Away Win)', fontsize=12)
plt.ylabel('Number of Fouls Committed by Home Team', fontsize=12)
plt.show()

plt.figure(figsize=(10, 6))
sns.violinplot(x='FT Result', y='H SOT', data=df, order=['H', 'D', 'A'])
plt.title('Home Team Shots on Target by Match Result', fontsize=16)
plt.xlabel('Full Time Result (H=Home Win, D=Draw, A=Away Win)', fontsize=12)
plt.ylabel('Shots on Target by Home Team', fontsize=12)
plt.show()
```


Distribution of Full Time Home Goals
Number of Matches
Number of Goals
0
200
400
600
800
1000
0
1
2
3
4
5
6
7
8


Home Team Fouls by Match Result
Number of Fouls Committed by Home Team
Full Time Result (H=Home Win, D=Draw, A=Away Win)
0
5
10
15
20
25
H
D
A



9. Bubble Plots

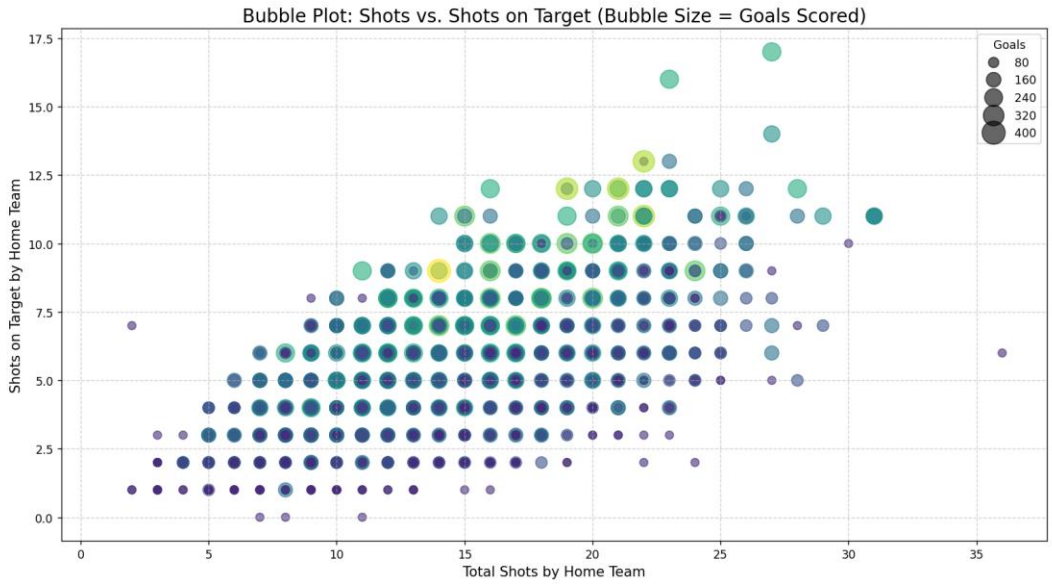
Bubble plots are a powerful variation of the standard scatter plot. They allow us to represent three dimensions of data simultaneously: the x-axis, the y-axis, and the size of the bubble.

- **Approach**: We can explore the relationship between a team's offensive actions (shots and shots on target) and their ultimate outcome (goals).
- **Example Setup**:
  - **X-axis**: `H Shots` (Total shots taken by the home team)
  - **Y-axis**: `H SOT` (Shots on target by the home team)
  - **Bubble Size**: `FTH Goals` (Full-time goals scored by the home team)
- **Interpretation**: This plot will help us visualize the efficiency of a team's shooting. We expect to see larger bubbles (more goals) clustered in the top-right area of the plot, where teams take many shots and have high accuracy (many shots on target). Small bubbles in the bottom-left would represent matches with low offensive output.

CODE

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)

for col in df.select_dtypes(include='number').columns:
    df[col].fillna(df[col].median(), inplace=True)

plt.figure(figsize=(12, 8))
```

```
scatter = plt.scatter(
    df['H Shots'],
    df['H SOT'],
    s=df['FTH Goals'] * 50,
    alpha=0.6,
    c=df['FTH Goals'],
)

plt.xlabel('Total Shots by Home Team', fontsize=12)
plt.ylabel('Shots on Target by Home Team', fontsize=12)
plt.title('Bubble Plot: Shots vs. Shots on Target (Bubble Size = Goals
Scored)', fontsize=16)
plt.grid(True, linestyle='--', alpha=0.6)

handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6, num=5)
legend2 = plt.legend(handles, labels, loc="upper right", title="Goals")

plt.show()
```



10. 2D Density Plots

A 2D Density Plot, or Kernel Density Estimate (KDE) plot, is used to visualize the joint probability distribution of two continuous variables. It essentially creates a contour map that shows where the data points are most densely clustered.

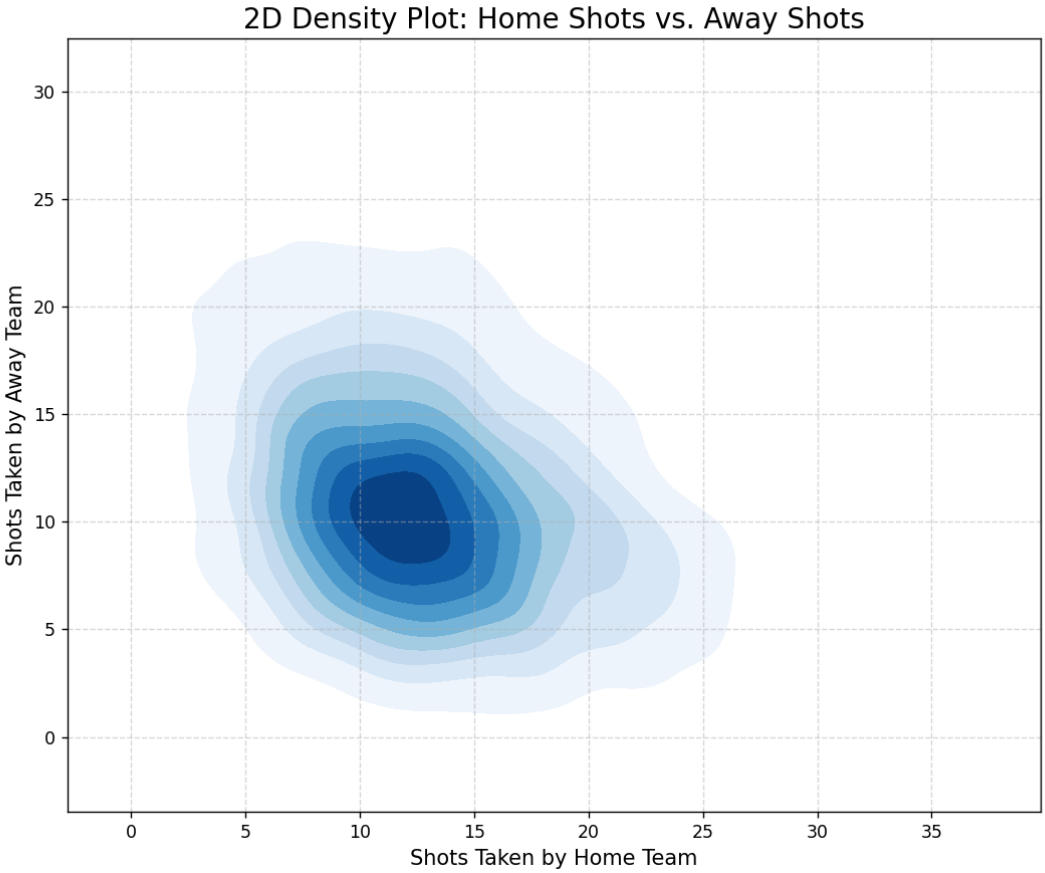
- **Approach**: We can use this plot to examine the relationship between the number of shots taken by the home team (H Shots) and the away team (A Shots).
- **Interpretation**: The "hot spots" or darkest areas on the plot will indicate the most frequent combination of shot counts in a typical match. This helps us understand the

common offensive balance in the games recorded in the dataset. For example, we can see if most matches are low-scoring affairs with few shots from both sides, or if one team typically dominates the shot count.

- **Benefit**: Unlike a simple scatter plot, a density plot is excellent for large datasets where many points might overlap. It provides a clear view of the data's distribution and helps in identifying the most typical game dynamics.

CODE

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('Assignen\England_CSV.csv')
df.dropna(how='all', inplace=True)

df['H Shots'].fillna(df['H Shots'].median(), inplace=True)
df['A Shots'].fillna(df['A Shots'].median(), inplace=True)

plt.figure(figsize=(10, 8))

sns.kdeplot(
    x=df['H Shots'],
    y=df['A Shots'],
    cmap='Blues',
    fill=True,
    thresh=0.05
)

plt.title('2D Density Plot: Home Shots vs. Away Shots', fontsize=16)
plt.xlabel('Shots Taken by Home Team', fontsize=12)
plt.ylabel('Shots Taken by Away Team', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)

plt.show()
```


2D Density Plot: Home Shots vs. Away Shots
Shots Taken by Away Team
Shots Taken by Home Team
0
5
10
15
20
25
30
35