**4.3 Statistical Analysis**

1. **Player Performance Evaluation**

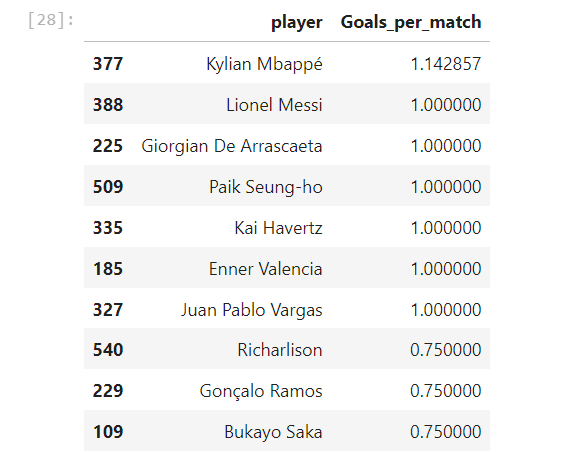
# Calculate goals per match

players['Goals\_per\_match'] = players['goals'] / players['games']

# Display top scorers based on goals per match

top\_scorers = players.sort\_values(by='Goals\_per\_match', ascending=False).head(10)

top\_scorers[['player', 'Goals\_per\_match']]

****

1. **Predicting Match Outcomes**

# Since matches have no result we use a proxy for Match\_Result based on goals

# This is a simplified approach to adjust it as needed for our data.

teams['Match\_Result'] = (teams['goals'] > teams['goals'].mean()).astype(int)

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Calculate Pass Accuracy

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

# Prepare data for training

features = teams[['goals', 'possession', 'Pass\_Accuracy']]

labels = teams['Match\_Result']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Train the model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Test the model

predictions = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, predictions))



1. **Improved Tactical Understanding**

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Calculate Pass Accuracy

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

# Apply clustering (excluding Goals\_Conceded if not available)

kmeans = KMeans(n\_clusters=4, random\_state=0).fit(teams[['possession', 'Pass\_Accuracy']])

teams['Play\_Style'] = kmeans.labels\_

# Visualize clusters

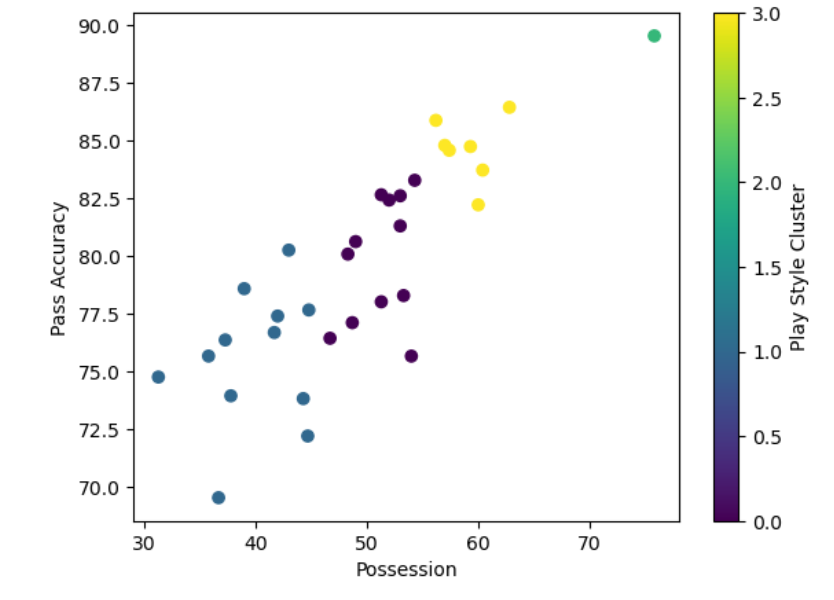
plt.scatter(teams['possession'], teams['Pass\_Accuracy'], c=teams['Play\_Style'], cmap='viridis')

plt.xlabel('Possession')

plt.ylabel('Pass Accuracy')

plt.colorbar(label='Play Style Cluster')

plt.show()

****

1. **Team Management and Player Selection**

# Check players with position as 'Forward'

forwards = players[players['position'] == 'FW']

print("Total forwards:", len(forwards))

# Check players with Goals per Match greater than 0.3

high\_scorers = players[players['Goals\_per\_match'] > 0.3]

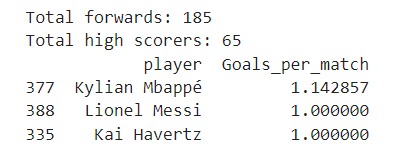
print("Total high scorers:", len(high\_scorers))

# Try a lower threshold for Goals per Match

selected\_players = players[(players['position'] == 'FW') & (players['Goals\_per\_match'] > 0.1)]

selected\_players = selected\_players.sort\_values(by='Goals\_per\_match', ascending=False).head(3)

print(selected\_players[['player', 'Goals\_per\_match']])

****

1. **World Cup Wins Analysis**

# Count the number of World Cups each country has won

world\_cup\_wins = world\_cups['Winner'].value\_counts()

print(world\_cup\_wins)

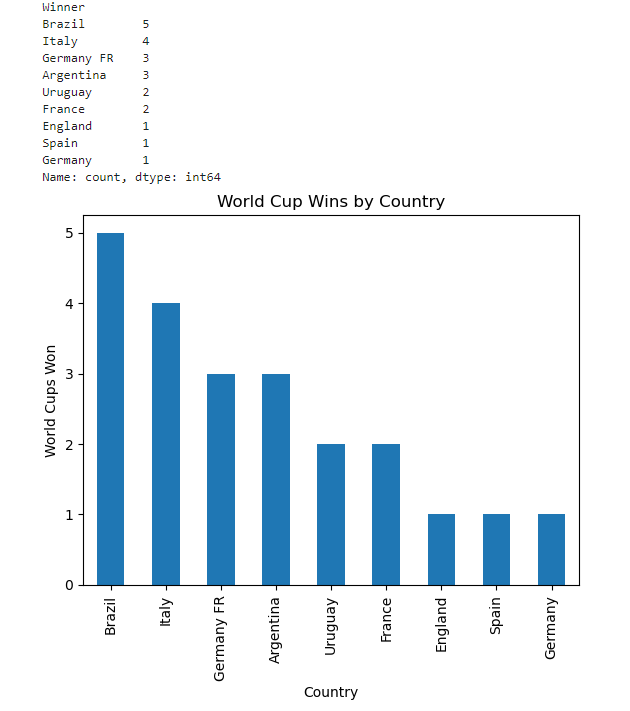
# Plot world cup wins

world\_cup\_wins.plot(kind='bar', title="World Cup Wins by Country")

plt.xlabel("Country")

plt.ylabel("World Cups Won")

plt.show()



**4.4 Implementation**

**Fetch**

**import statements**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

**Load the data**

players = pd.read\_csv("player\_stats.csv")

teams = pd.read\_csv("team\_data.csv")

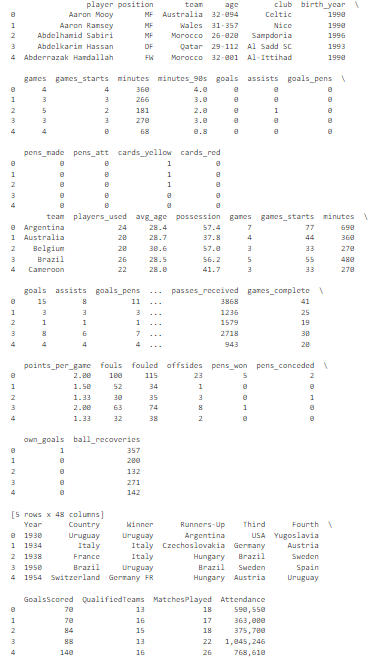
world\_cups = pd.read\_csv("world\_cups.csv")

**Display basic info**

print(players.head())

print(teams.head())

print(world\_cups.head())

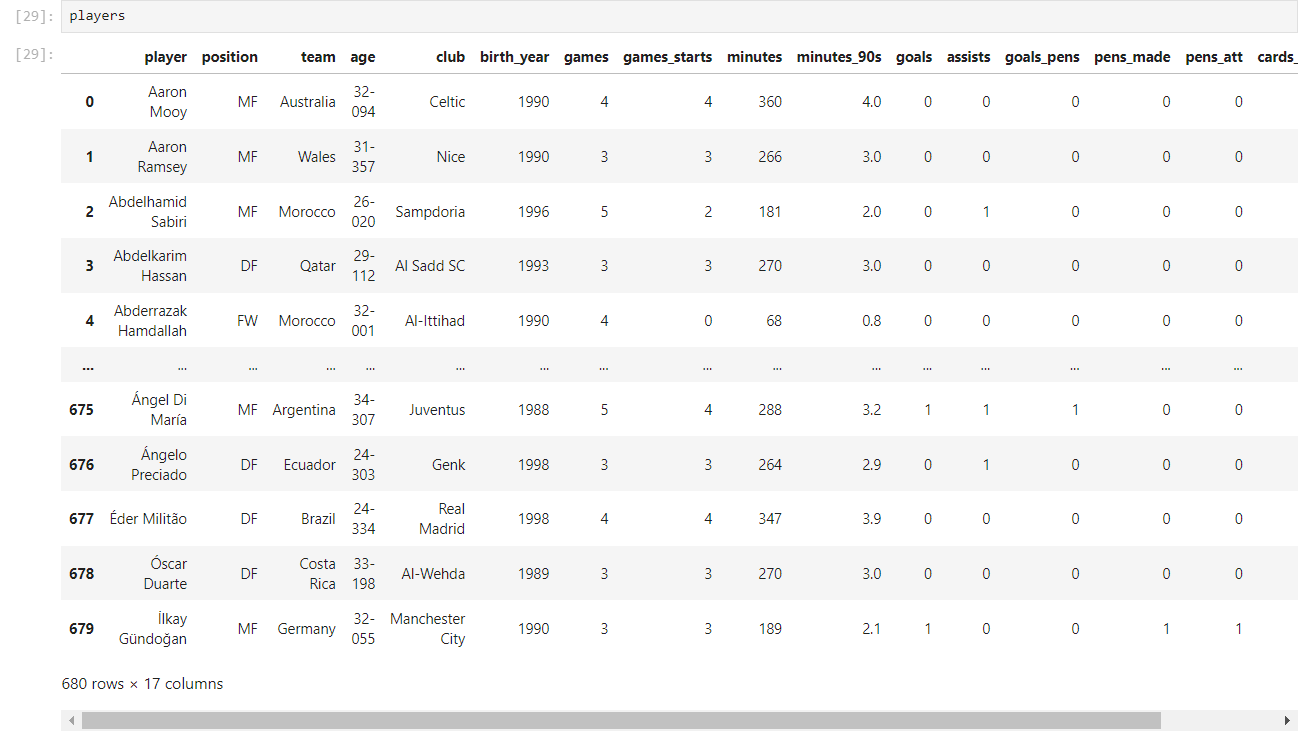


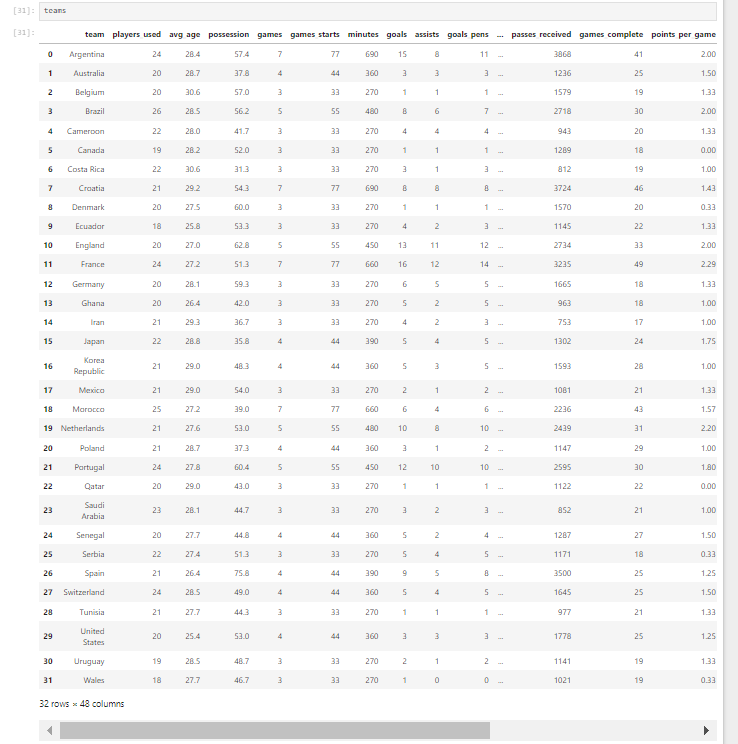
**Check for missing values and handle them**

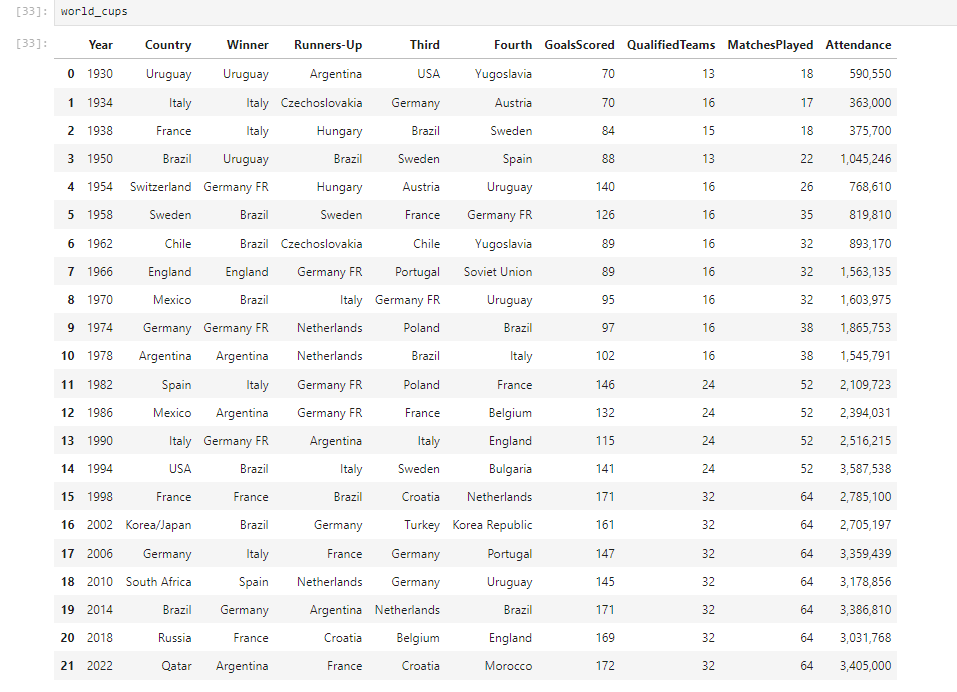
players.fillna(0, inplace=True)

teams.fillna(0, inplace=True)

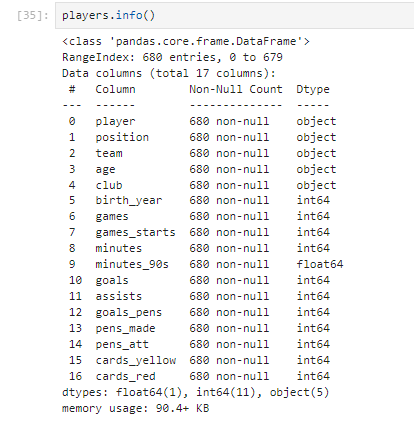
world\_cups.fillna(0, inplace=True)

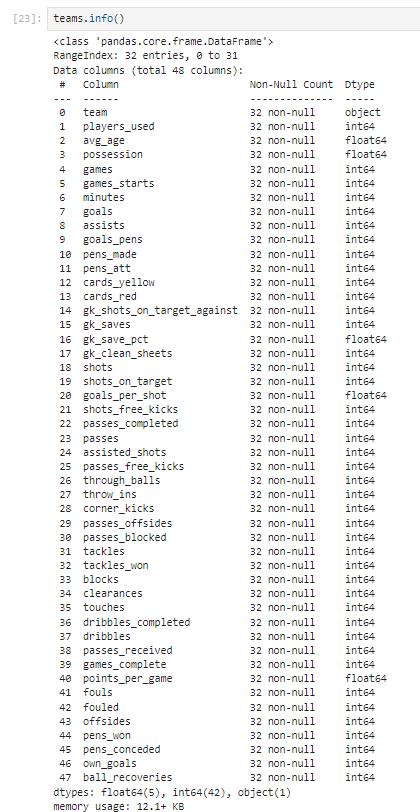
****

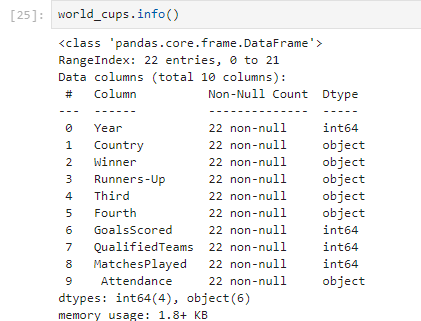
****

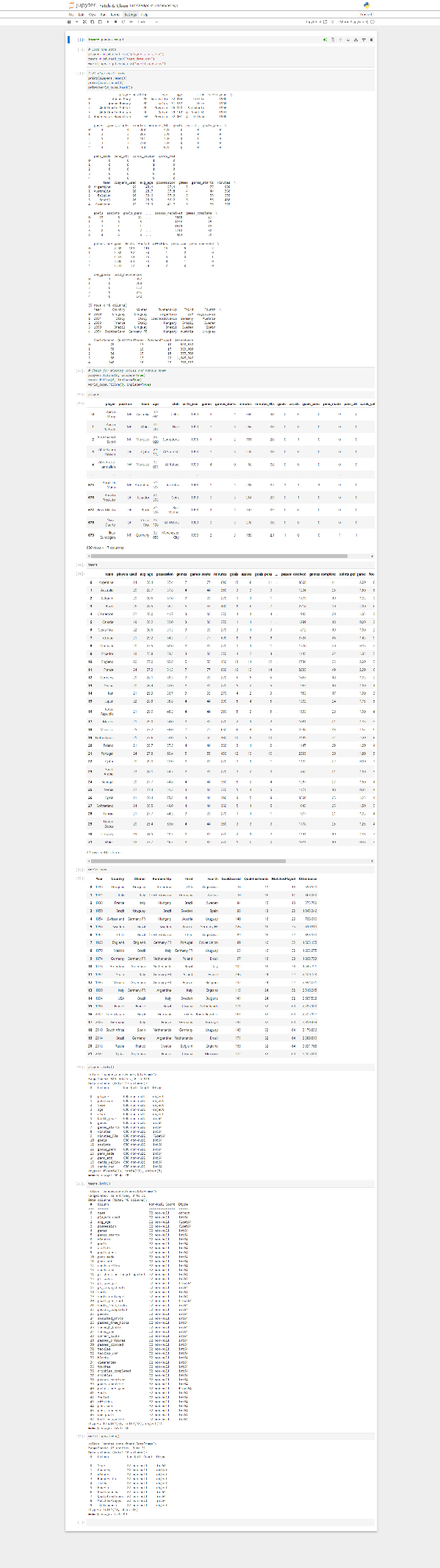
****

**Information of Each Data Sets**

****

****

****

**Fetch & Clean – Full Page Screenshot 🡪4.5 Data Visualization**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**1. Player Performance Comparison**

# Plot for Top Players by Goals per Match

plt.figure(figsize=(12, 6))

sns.barplot(data=top\_scorers, x='player', y='Goals\_per\_match', color='skyblue')

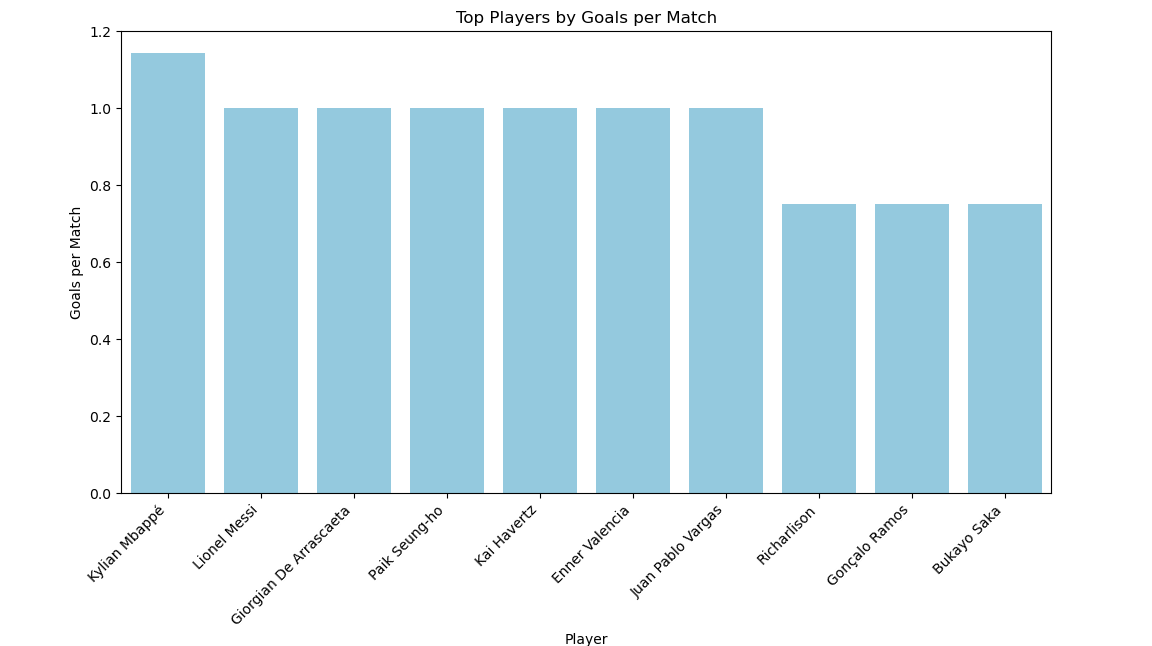
plt.xticks(rotation=45, ha="right")

plt.title("Top Players by Goals per Match")

plt.xlabel("Player")

plt.ylabel("Goals per Match")

plt.show()

****

**2. Team Performance**

top\_teams = teams[['team', 'goals']].sort\_values(by='goals', ascending=False).head(10)

plt.figure(figsize=(12, 6))

sns.barplot(data=top\_teams, x='team', y='goals', palette='viridis')

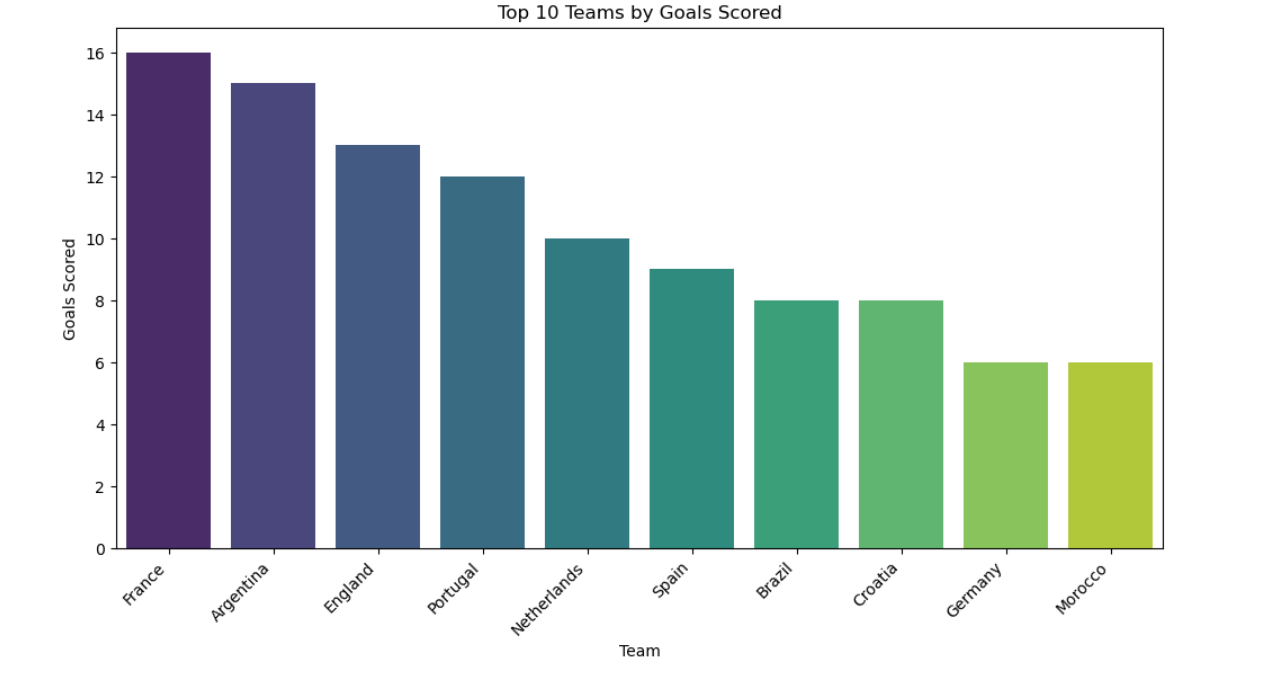
plt.xticks(rotation=45, ha="right")

plt.title('Top 10 Teams by Goals Scored')

plt.xlabel('Team')

plt.ylabel('Goals Scored')

plt.show()



**3. World Cup Wins by Country (Bar Chart)**

plt.figure(figsize=(10, 6))

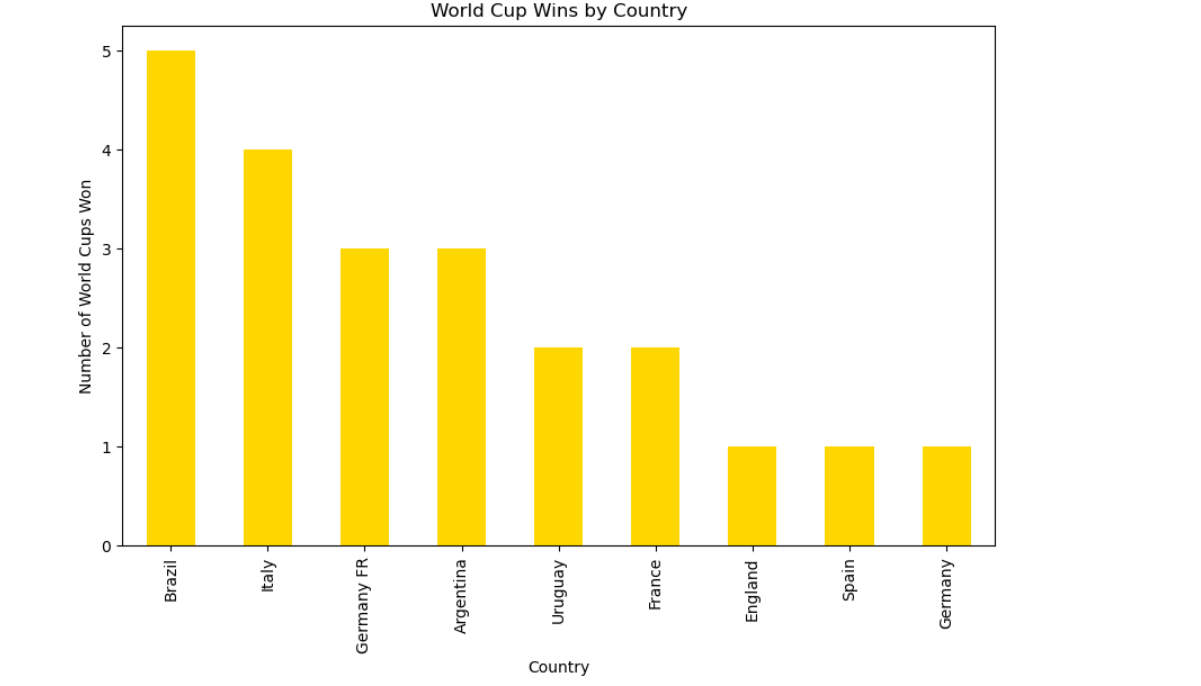
world\_cup\_wins.plot(kind='bar', color='gold')

plt.title("World Cup Wins by Country")

plt.xlabel("Country")

plt.ylabel("Number of World Cups Won")

plt.show()



**4. Team Play Styles (Clustering Result)**

# Calculate Pass Accuracy if not already calculated

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

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

sns.scatterplot(data=teams, x='possession', y='Pass\_Accuracy', hue='Play\_Style', palette='viridis', s=100)

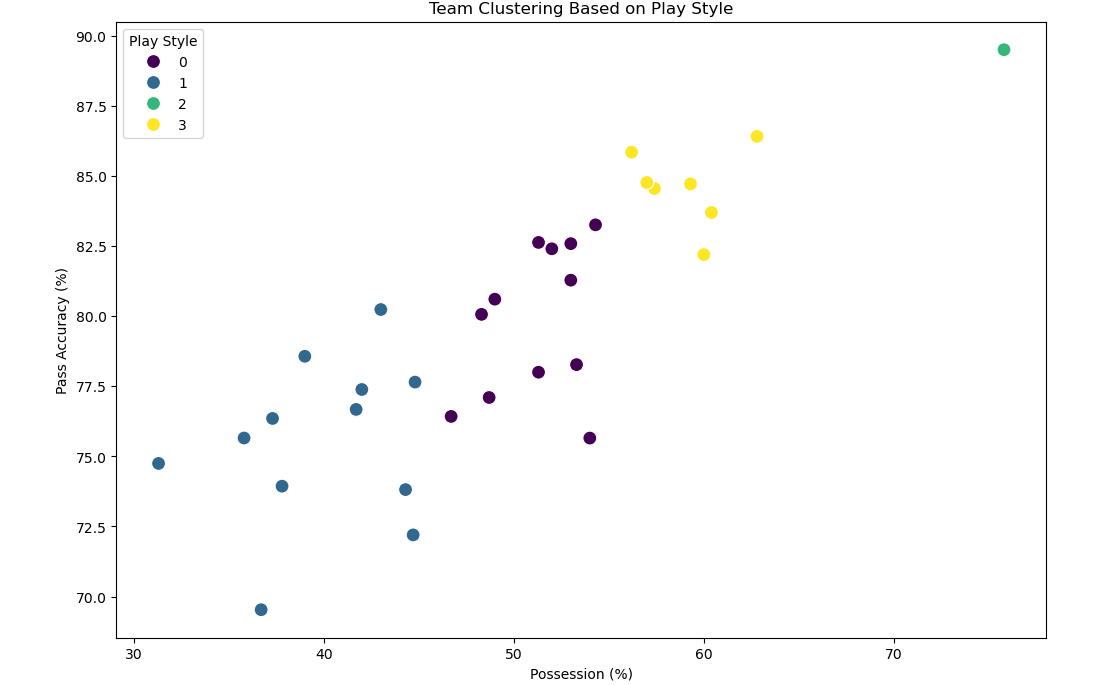
plt.title("Team Clustering Based on Play Style")

plt.xlabel("Possession (%)")

plt.ylabel("Pass Accuracy (%)")

plt.legend(title='Play Style')

plt.show()



**5. Correlation Heatmap for Player Stats**

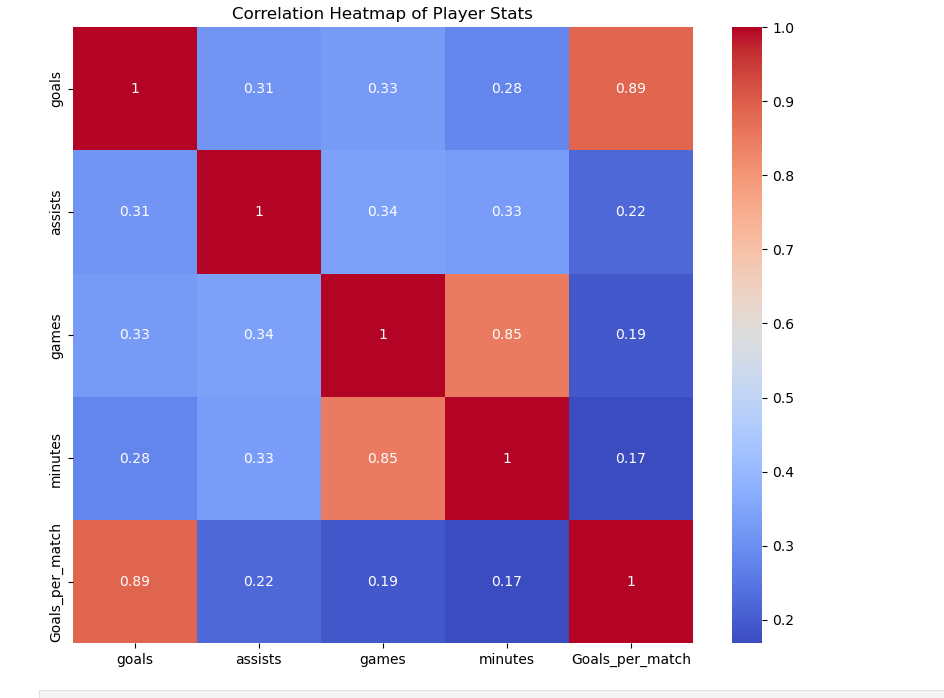
plt.figure(figsize=(10, 8))

correlation\_matrix = players[['goals', 'assists', 'games', 'minutes', 'Goals\_per\_match']].corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap of Player Stats")

plt.show()



**4.6 Model Development and Evolution**

**Step 1: Predicting Player Performance (Goals Prediction)**

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Selecting features

X = players[['minutes', 'games', 'assists']]

y = players['goals']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Model training

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse:.2f}')

print(f'R² Score: {r2:.2f}')

****

**Step 2: Clustering Players by Play Style**

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

# Selecting features for clustering

features = players[['goals', 'assists', 'games', 'minutes']]

kmeans = KMeans(n\_clusters=3, random\_state=42)

players['Play\_Style'] = kmeans.fit\_predict(features)

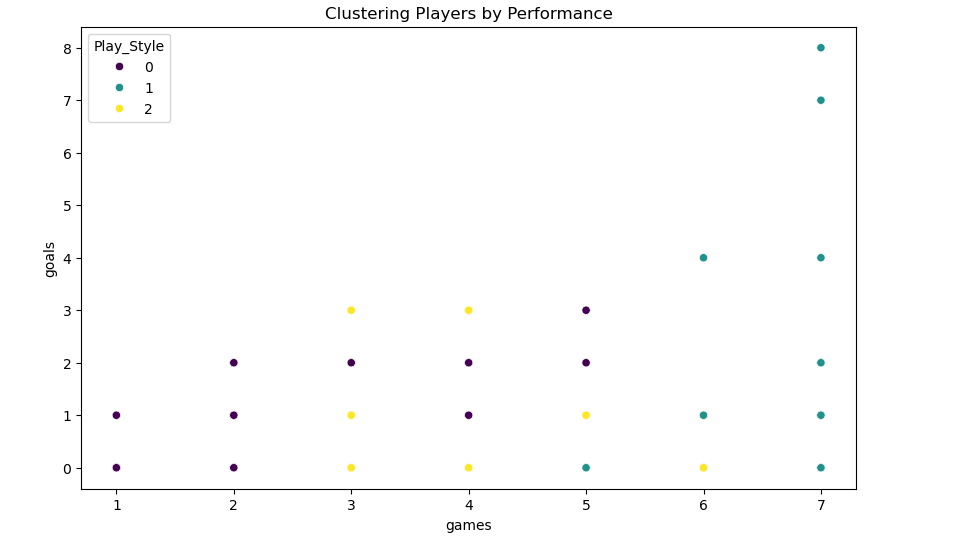
# Visualizing clusters

plt.figure(figsize=(10, 6))

sns.scatterplot(data=players, x='games', y='goals', hue='Play\_Style', palette='viridis')

plt.title("Clustering Players by Performance")

plt.show()

****

**1. KMeans Clustering: Grouping Teams by Play Style**

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Calculate Pass Accuracy if not already calculated

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

# Select relevant features for clustering

team\_features = teams[['possession', 'Pass\_Accuracy', 'goals']]

# Initialize and fit KMeans

kmeans = KMeans(n\_clusters=4, random\_state=42)

teams['Play\_Style'] = kmeans.fit\_predict(team\_features)

# Plot the clustering results

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

sns.scatterplot(data=teams, x='possession', y='Pass\_Accuracy', hue='Play\_Style', palette='viridis', s=100)

plt.title("KMeans Clustering: Team Play Style")

plt.xlabel("Possession (%)")

plt.ylabel("Pass Accuracy (%)")

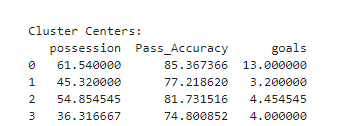
plt.legend(title='Play Style Cluster')

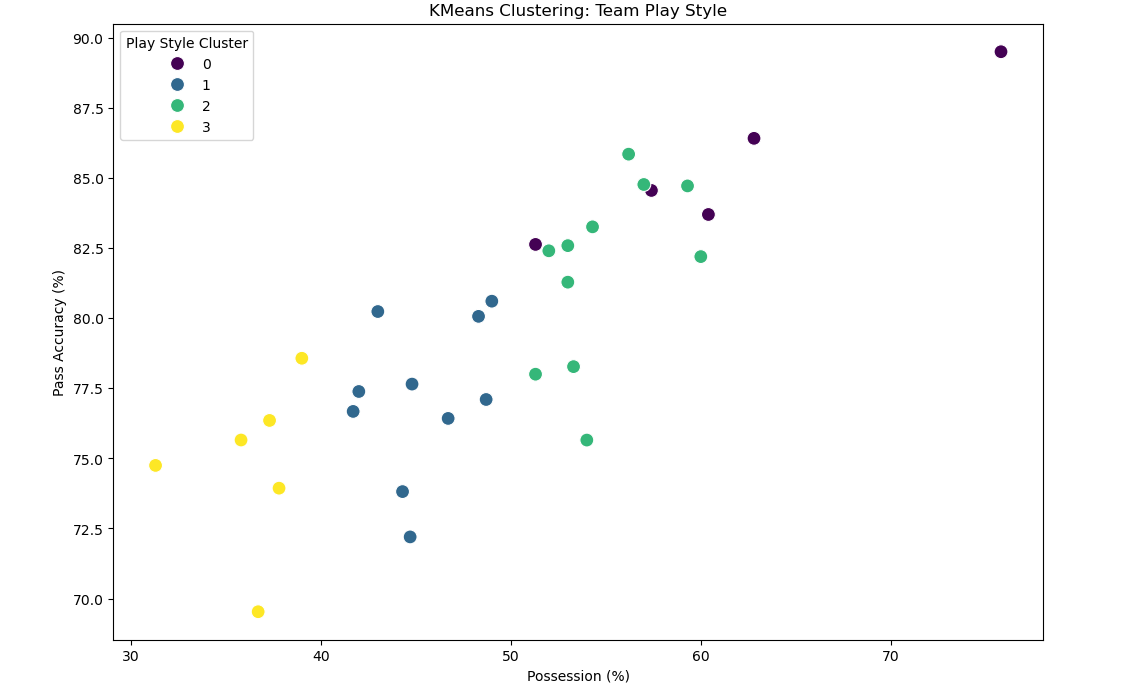
plt.show()

# View cluster centers to analyze the play styles

print("Cluster Centers:")

print(pd.DataFrame(kmeans.cluster\_centers\_, columns=team\_features.columns))

****

****

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

# Perform clustering with KMeans

kmeans = KMeans(n\_clusters=4, random\_state=42)

teams['Play\_Style'] = kmeans.fit\_predict(teams[['possession', 'Pass\_Accuracy', 'goals']])

# Calculate Silhouette Score

silhouette\_avg = silhouette\_score(teams[['possession', 'Pass\_Accuracy', 'goals']], kmeans.labels\_)

print("Silhouette Score for KMeans Clustering:", silhouette\_avg)



**2. K-Nearest Neighbors (KNN): Predicting Match Outcomes**

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

# Calculate Pass Accuracy if not already calculated

teams['Pass\_Accuracy'] = (teams['passes\_completed'] / teams['passes']) \* 100

# Create a basic rule for Match\_Result

teams['Match\_Result'] = teams['goals'].apply(lambda x: 1 if x > 1 else 0)

# Prepare features and labels

features = teams[['goals', 'possession', 'Pass\_Accuracy']]

labels = teams['Match\_Result'] # 1 for Win, 0 for Loss/Draw

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Initialize and train KNN model

knn = KNeighborsClassifier(n\_neighbors=5) # Using k=5

knn.fit(X\_train, y\_train)

# Make predictions and evaluate

y\_pred = knn.predict(X\_test)

print("KNN Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

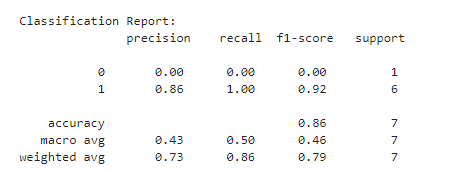
# Plot Confusion Matrix

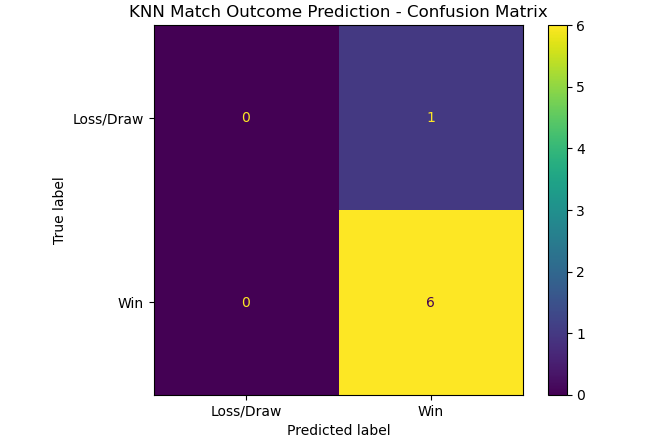
ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred, display\_labels=['Loss/Draw', 'Win'])

plt.title("KNN Match Outcome Prediction - Confusion Matrix")

plt.show()



****

****

**3. Principal Component Analysis (PCA) for Dimensionality Reduction**

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

team\_features\_scaled = scaler.fit\_transform(team\_features)

# Apply PCA

pca = PCA(n\_components=2)

principal\_components = pca.fit\_transform(team\_features\_scaled)

teams['PCA1'], teams['PCA2'] = principal\_components[:, 0], principal\_components[:, 1]

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

sns.scatterplot(data=teams, x='PCA1', y='PCA2', hue='Play\_Style', palette='Set2', s=100)

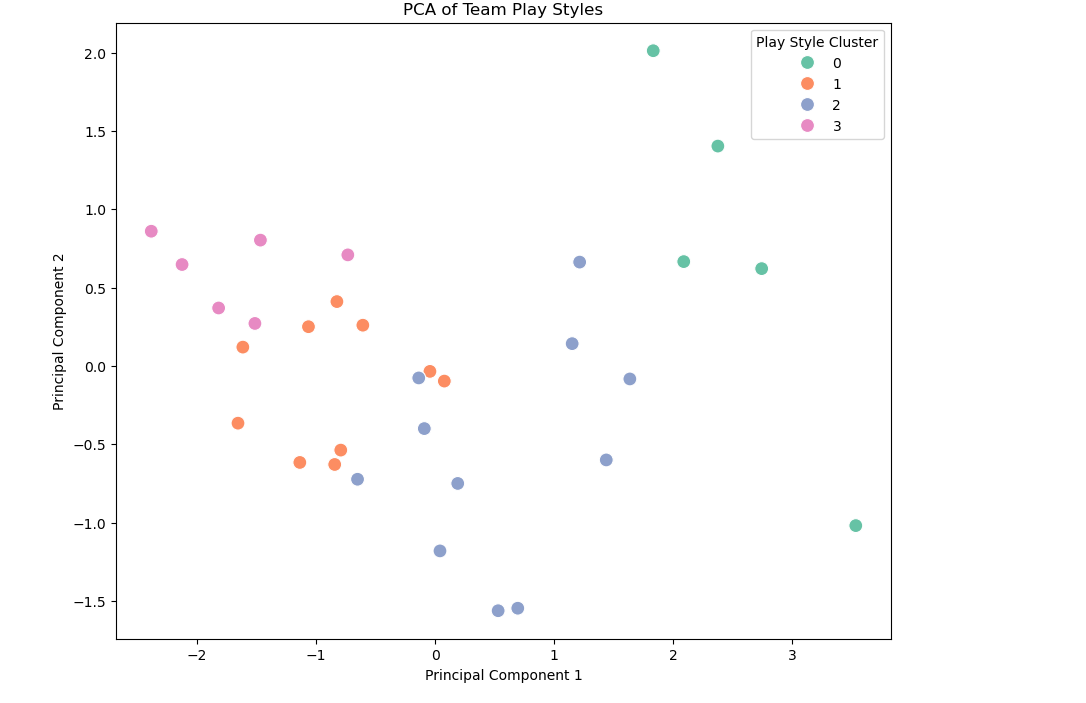
plt.title("PCA of Team Play Styles")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.legend(title='Play Style Cluster')

plt.show()



**4. Predicting Player Performance using Linear Regression**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

# Calculate Goals per Match if not already calculated

players['Goals\_per\_match'] = players['goals'] / players['games']

# Define features and target (without Tackles)

player\_features = players[['minutes', 'assists']] # Adjusting for available columns

target = players['Goals\_per\_match']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(player\_features, target, test\_size=0.2, random\_state=42)

# Train Linear Regression model

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

# Predict and evaluate

y\_pred = regressor.predict(X\_test)

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R^2 Score:", r2\_score(y\_test, y\_pred))

# Plot Actual vs Predicted values

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, alpha=0.7, color="blue")

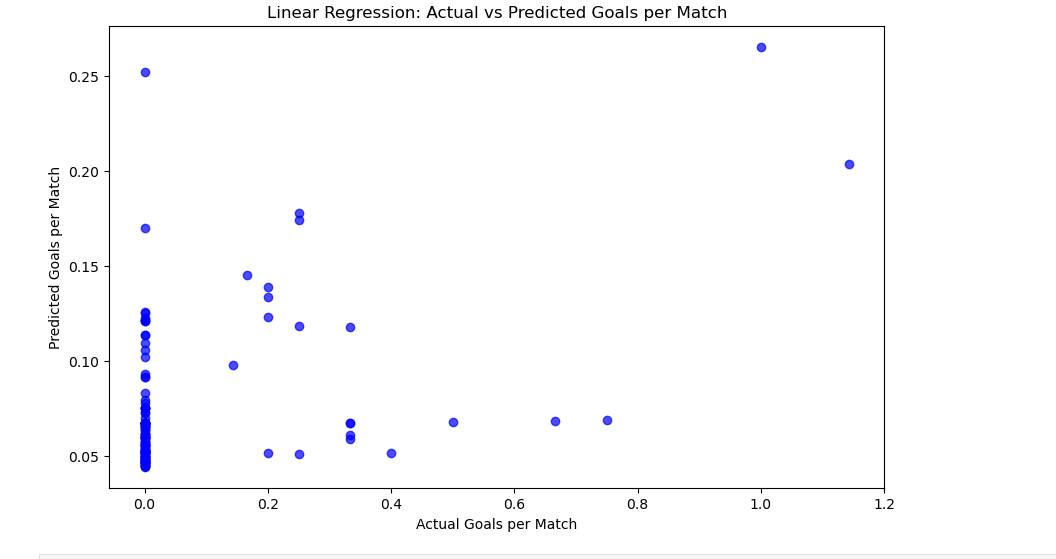
plt.xlabel("Actual Goals per Match")

plt.ylabel("Predicted Goals per Match")

plt.title("Linear Regression: Actual vs Predicted Goals per Match")

plt.show()

****

****