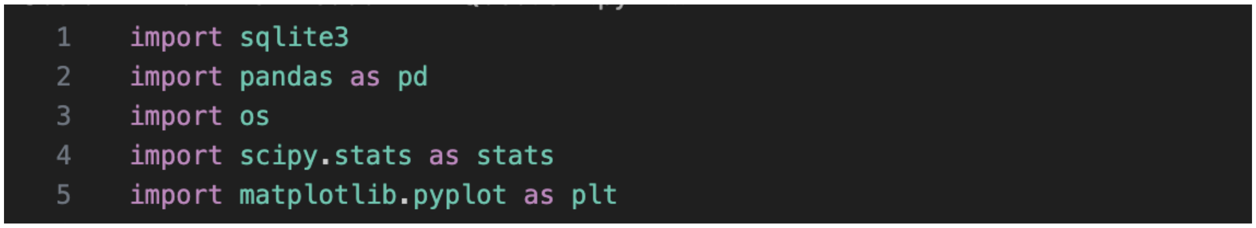
INC261 2/2025

Report Assignment 3: Play around with SQL

Member

66070504002	Kongphob	Suranan
66070504007	Napatsorn	Amadmuntree
66070504011	Petidarat	Vagost
66070504012	Phurinat	Soayyala
66070504017	Sirawit	Hengroong

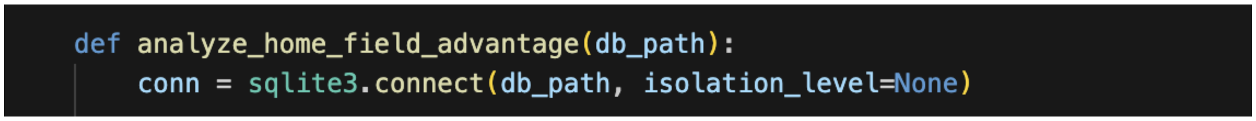
Question 1:



```python
import sqlite3
import pandas as pd
import os
import scipy.stats as stats
import matplotlib.pyplot as plt
```

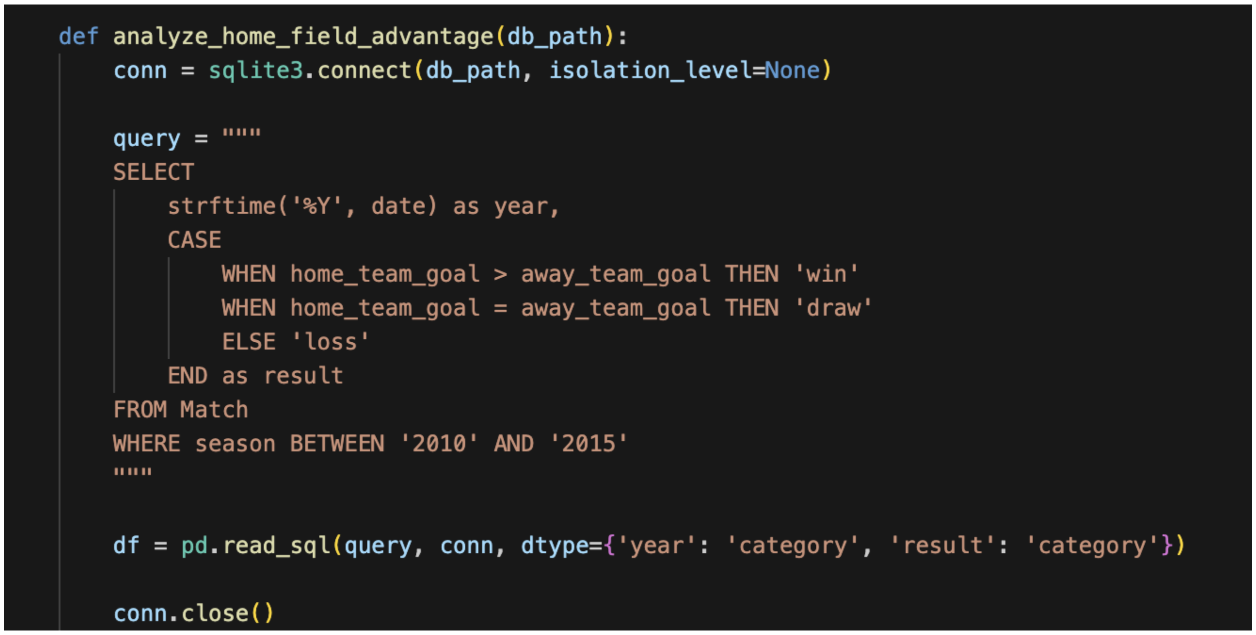
These are libraries used for work.

sqlite3: Used to connect to a .sqlite file database.
pandas: Used to manage data in a tabular format.
os: It is a library that collects various command sets that have basic commands to manage files and folders that we want, whether it is moving, saving, renaming, or deleting files and folders that we want.
scipy.stats: Used to analyze statistics in Python.
matplotlib.pyplot: Used for creating various types of plots, such as bar charts and histograms, to help visualize the data clearly.



```python
def analyze_home_field_advantage(db_path):
    conn = sqlite3.connect(db_path, isolation_level=None)
```

This defines a function named analyze_home_field_advantage that takes a path to a database file and connects to a SQLite database, transaction disabled, read-only access



```python
def analyze_home_field_advantage(db_path):
    conn = sqlite3.connect(db_path, isolation_level=None)

    query = """
    SELECT
        strftime('%Y', date) as year,
        CASE
            WHEN home_team_goal > away_team_goal THEN 'win'
            WHEN home_team_goal = away_team_goal THEN 'draw'
            ELSE 'loss'
        END as result
    FROM Match
    WHERE season BETWEEN '2010' AND '2015'
    """

    df = pd.read_sql(query, conn, dtype={'year': 'category', 'result': 'category'})

    conn.close()
```

This query selects the year and result (win/draw/loss) of each match for seasons 2010 to 2015 and then loads the data into a pandas DataFrame and closes the database connection. The data types are optimized as categories.

```python
total_games = len(df)
result_counts = df['result'].value_counts()
home_wins = result_counts.get('win', 0)
home_draws = result_counts.get('draw', 0)
home_losses = result_counts.get('loss', 0)
```

Counts the total number of games and how many were wins, draws, or losses.

```python
p_value = stats.binomtest(home_wins, total_games, p=1/3, alternative='greater')

print(f"Home wins: {home_wins}")
print(f"Home draws: {home_draws}")
print(f"Home losses: {home_losses}")
print(f"Total games analyzed: {total_games}")
print(f"P-value for home win rate being greater than draw or loss: {p_value.pvalue:.4f}")
```

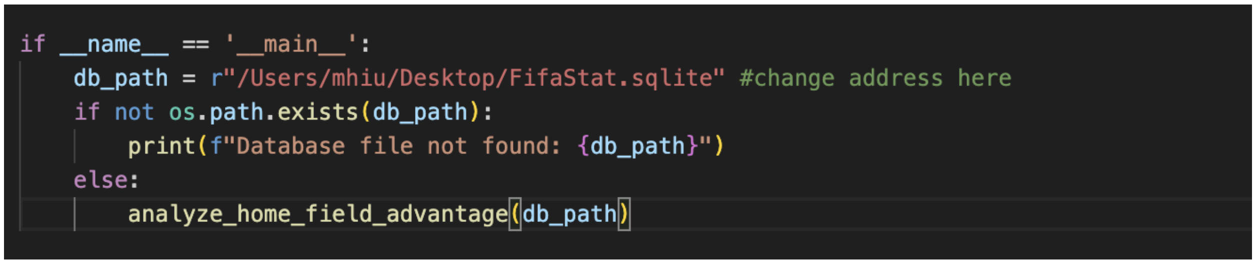
Performs a binomial test to see if the home win rate is significantly higher than the expected 1/3 and displays the counts of each result and the p-value from the statistical test.

```python
yearly_counts = df.groupby(['year', 'result']).size().unstack(fill_value=0)
yearly_totals = yearly_counts.sum(axis=1)
win_rate_by_year = yearly_counts.div(yearly_totals, axis=0)

print("\nHome match outcome rates per year (2010-2015):")
print(win_rate_by_year[['win', 'draw', 'loss']])

print("\nStatistical test: Is home win rate significantly greater than draw or loss?")
print(f"Home wins: {home_wins} out of {total_games} ({home_wins/total_games:.2%})")
print(f"Binomial test p-value: {p_value.pvalue:.4f}")
```

Group the results by year to calculate how win/draw/loss rates vary over time and then display the win/draw/loss percentages per year.

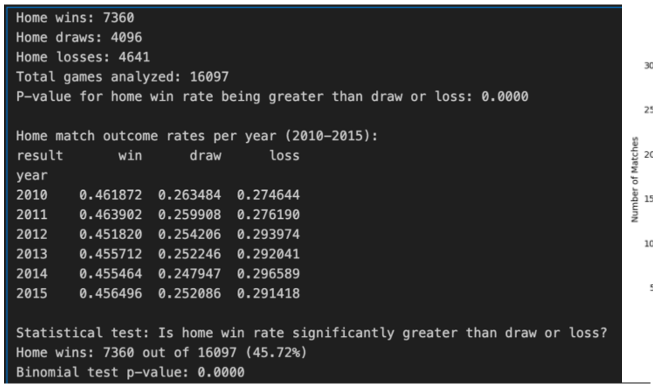
```python
yearly_counts.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title("Home Match Outcomes by Year (2010–2015)")
plt.ylabel("Number of Matches")
plt.xlabel("Year")
plt.legend(title="Result")
plt.tight_layout()
plt.show()
```

Creates a stacked bar chart showing the number of home wins, draws, and losses per year.

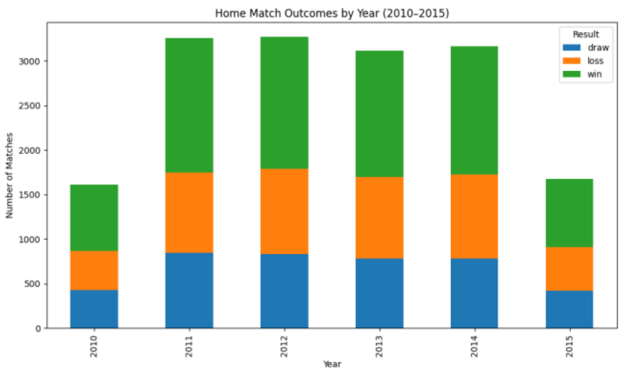


```
if __name__ == '__main__':
    db_path = r"/Users/mhiu/Desktop/FifaStat.sqlite" #change address here
    if not os.path.exists(db_path):
        print(f"Database file not found: {db_path}")
    else:
        analyze_home_field_advantage(db_path)
```

Checks if the database file exists and runs the analysis function if it does.



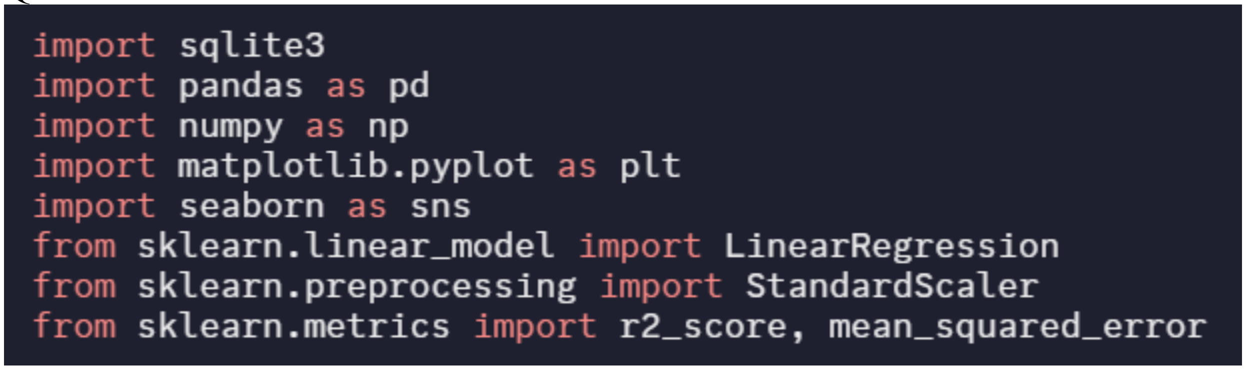
```
Home wins: 7360
Home draws: 4096
Home losses: 4641
Total games analyzed: 16097
P-value for home win rate being greater than draw or loss: 0.0000

Home match outcome rates per year (2010-2015):
result       win      draw      loss
year
2010    0.461872  0.263484  0.274644
2011    0.463902  0.259908  0.276190
2012    0.451820  0.254206  0.293974
2013    0.455712  0.252246  0.292041
2014    0.455464  0.247947  0.296589
2015    0.456496  0.252086  0.291418

Statistical test: Is home win rate significantly greater than draw or loss?
Home wins: 7360 out of 16097 (45.72%)
Binomial test p-value: 0.0000
```



Q1: Analyze home field advantage across different leagues and seasons: Does playing at home provide a statistically significant advantage, and has this advantage changed over time (2010-2015)?

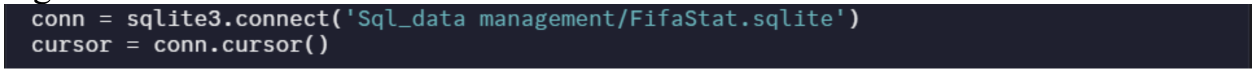
Based on the match data from 2010 to 2015, the analysis shows that playing at home offers a statistically significant advantage (p-value < 0.0000), with home teams winning more often than drawing or losing. Although the home win rate remains significantly higher than draws or losses, the home loss rate appears to be gradually increasing.

Question 2:



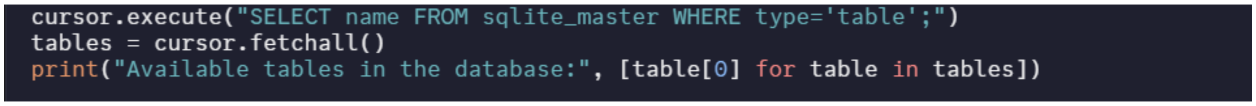
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
```

These are libraries used for work.
sqlite3: Used to connect to a .sqlite file database.
pandas: Used to manage data in a tabular format.
numpy: Used for working with arrays.
matplotlib.pyplot: Used for creating various types of plots, such as bar charts and histograms, to help clear visualisation of the data.
seaborn: a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.
LinearRegression: uses the relationship between the data points to draw a straight line through all of them. This line can be used to predict future values.
StandardScaler: Standardise features by removing the mean and scaling to unit variance. where u is the mean of the training samples or zero if with_mean=False , and s is the standard deviation of the training samples or one if with_std=False.
r2_score, mean_squared_error: regression score function, Mean Squared Error (MSE) is one of the most common metrics used for evaluating the performance of regression models.



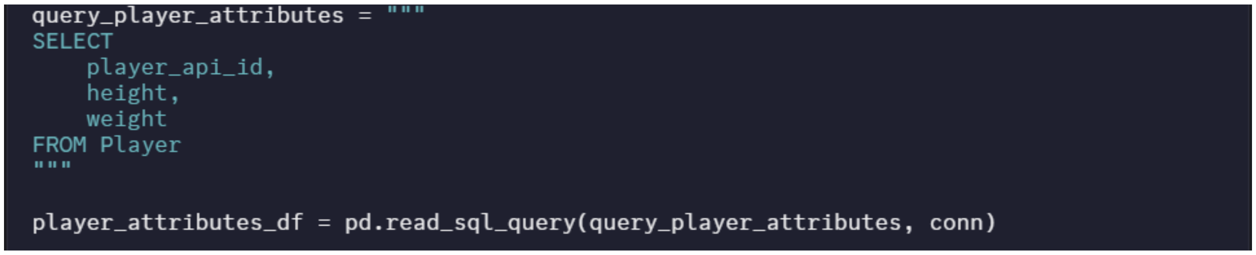
```
conn = sqlite3.connect('Sql_data management/FifaStat.sqlite')
cursor = conn.cursor()
```

This line of code takes a path to a database file and connects to the SQLite database. The cursor is used to execute statements to communicate with the Mysql database.



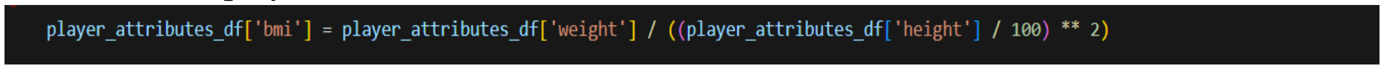
```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()
print("Available tables in the database:", [table[0] for table in tables])
```

This line of code is used to check if the table exists in the data.



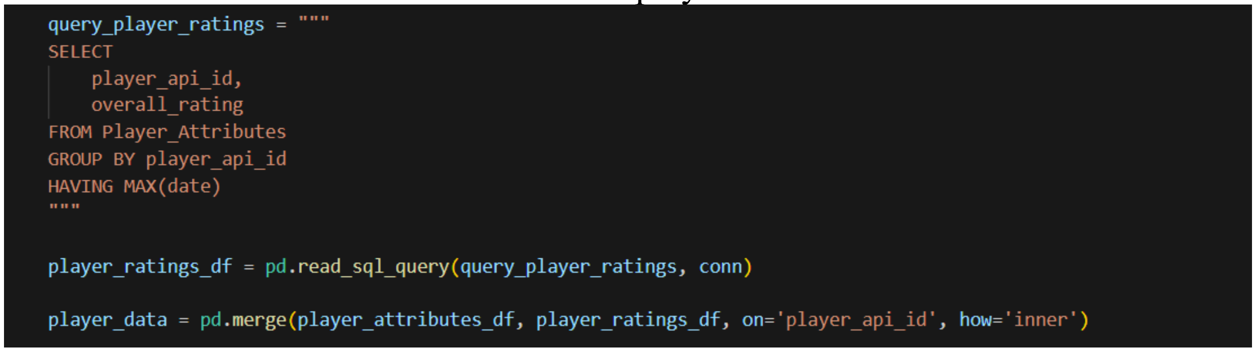
```
query_player_attributes = """
SELECT
    player_api_id,
    height,
    weight
FROM Player
"""

player_attributes_df = pd.read_sql_query(query_player_attributes, conn)
```

This line of code is for player physical attributes (height, weight) data, and we get these from the player table.



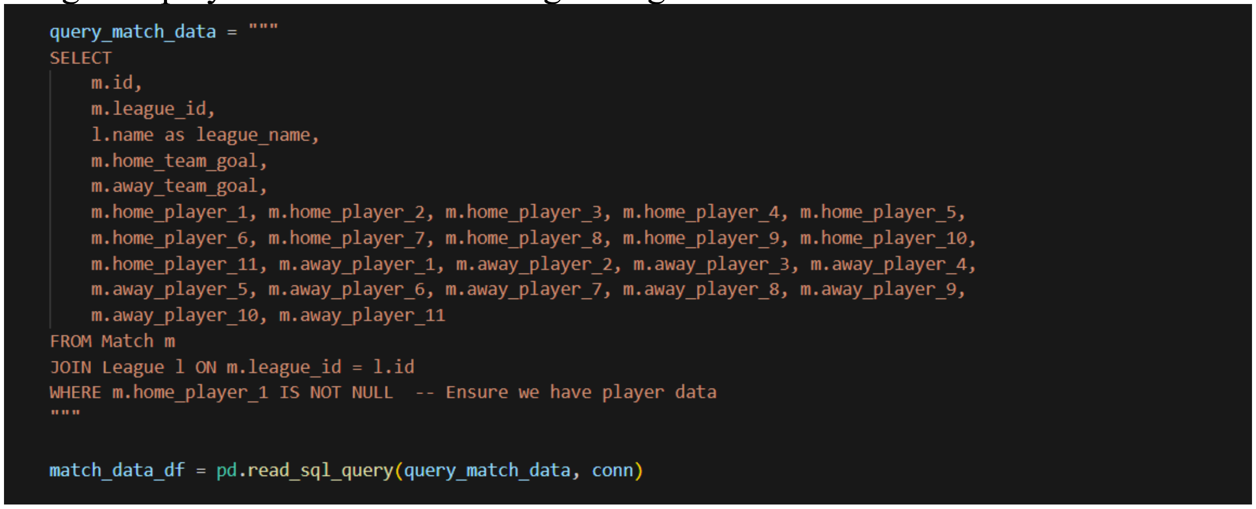
```
player_attributes_df['bmi'] = player_attributes_df['weight'] / ((player_attributes_df['height'] / 100) ** 2)
```

This line of code is used to calculate each player's BMI.



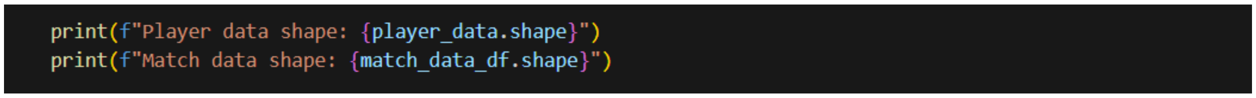
```
query_player_ratings = """
SELECT
    player_api_id,
    overall_rating
FROM Player_Attributes
GROUP BY player_api_id
HAVING MAX(date)
"""

player_ratings_df = pd.read_sql_query(query_player_ratings, conn)

player_data = pd.merge(player_attributes_df, player_ratings_df, on='player_api_id', how='inner')
```

This line of code is to get each player an overall rating from the file. And then merge the player attributes and rating all together.



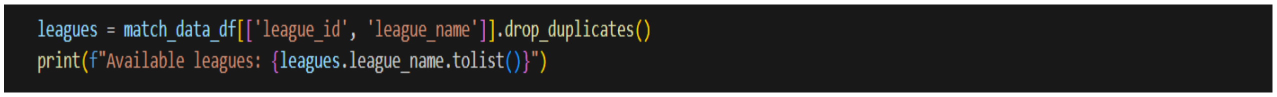
```
query_match_data = """
SELECT
    m.id,
    m.league_id,
    l.name as league_name,
    m.home_team_goal,
    m.away_team_goal,
    m.home_player_1, m.home_player_2, m.home_player_3, m.home_player_4, m.home_player_5,
    m.home_player_6, m.home_player_7, m.home_player_8, m.home_player_9, m.home_player_10,
    m.home_player_11, m.away_player_1, m.away_player_2, m.away_player_3, m.away_player_4,
    m.away_player_5, m.away_player_6, m.away_player_7, m.away_player_8, m.away_player_9,
    m.away_player_10, m.away_player_11
FROM Match m
JOIN League l ON m.league_id = l.id
WHERE m.home_player_1 IS NOT NULL  -- Ensure we have player data
"""

match_data_df = pd.read_sql_query(query_match_data, conn)
```

This line of code is used for getting match data with player IDs and league information.



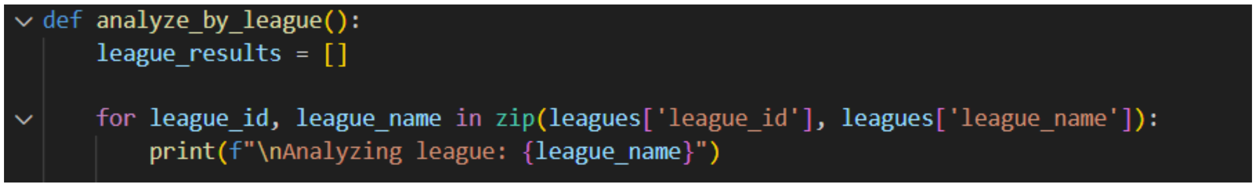
```
print(f"Player data shape: {player_data.shape}")
print(f"Match data shape: {match_data_df.shape}")
```

This line of code is used to check the structure of data.



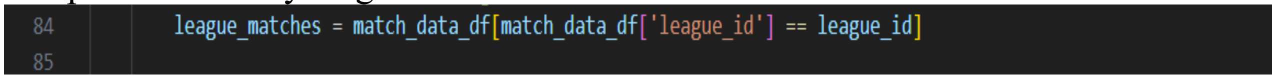
```
leagues = match_data_df[['league_id', 'league_name']].drop_duplicates()
print(f"Available leagues: {leagues.league_name.tolist()}")
```

This line of code is used to get a unique league.



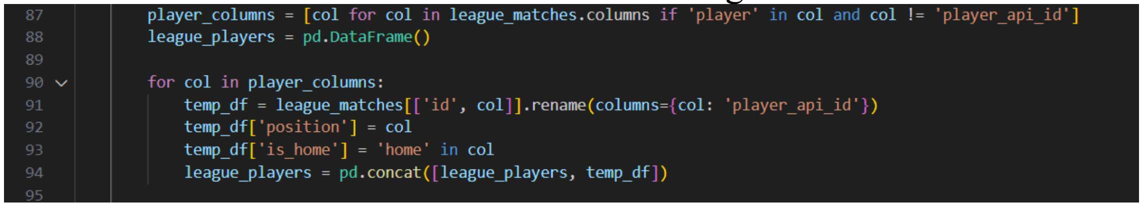
```python
def analyze_by_league():
    league_results = []

    for league_id, league_name in zip(leagues['league_id'], leagues['league_name']):
        print(f"\nAnalyzing league: {league_name}")
```

This is to create a function to analyze the relationship between physical attributes and performance by league.



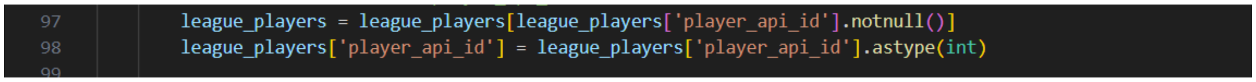
```python
        league_matches = match_data_df[match_data_df['league_id'] == league_id]
```

This line of code is for filter matches for this league.



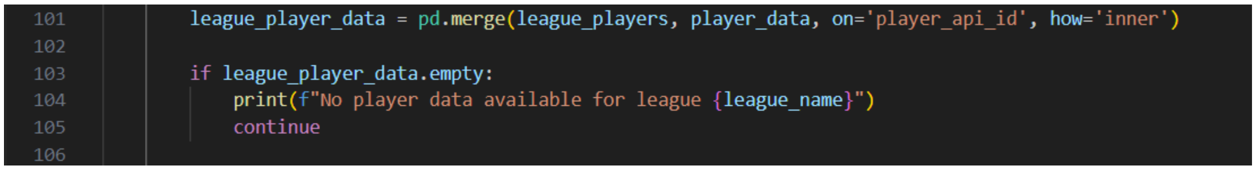
```python
        player_columns = [col for col in league_matches.columns if 'player' in col and col != 'player_api_id']
        league_players = pd.DataFrame()

        for col in player_columns:
            temp_df = league_matches[['id', col]].rename(columns={col: 'player_api_id'})
            temp_df['position'] = col
            temp_df['is_home'] = 'home' in col
            league_players = pd.concat([league_players, temp_df])
```

This line of code is to get all players who played in this league.



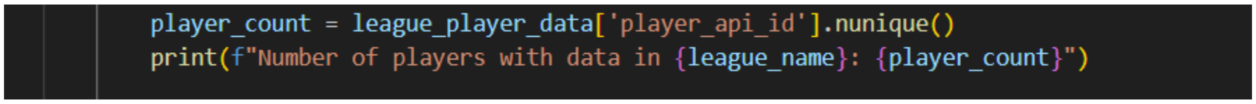
```python
        league_players = league_players[league_players['player_api_id'].notnull()]
        league_players['player_api_id'] = league_players['player_api_id'].astype(int)
```

This line of code is to remove rows with a null player_api_id.



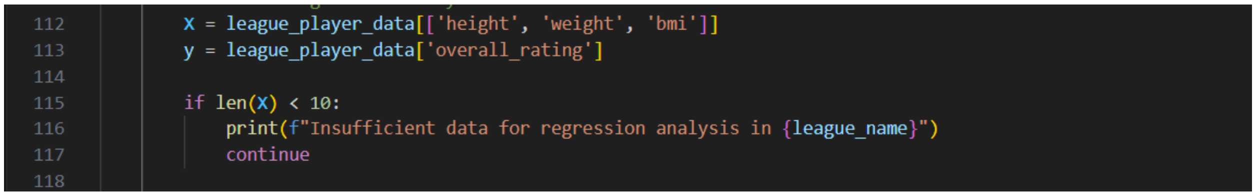
```python
        league_player_data = pd.merge(league_players, player_data, on='player_api_id', how='inner')

        if league_player_data.empty:
            print(f"No player data available for league {league_name}")
            continue
```

This line of code is to merge with player data.



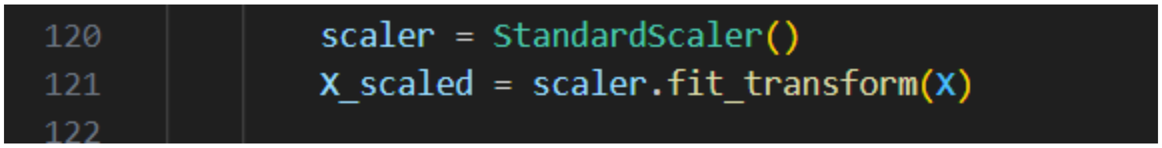
```python
        player_count = league_player_data['player_api_id'].nunique()
        print(f"Number of players with data in {league_name}: {player_count}")
```

This line is to the number of players in this match.



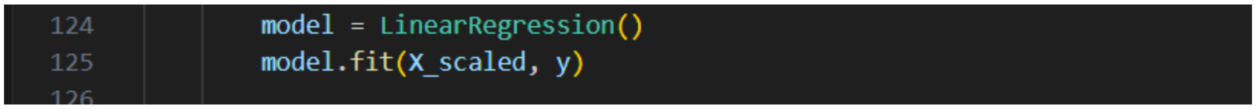
```python
        X = league_player_data[['height', 'weight', 'bmi']]
        y = league_player_data['overall_rating']

        if len(X) < 10:
            print(f"Insufficient data for regression analysis in {league_name}")
            continue
```

This line is to perform regression analysis.



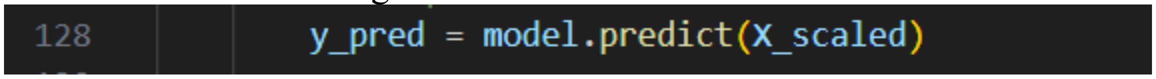
```python
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
```

This line is a scale feature for scaling.



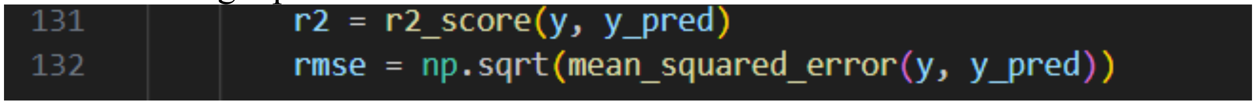
```
        model = LinearRegression()
        model.fit(X_scaled, y)
```

This line is to create a regression model.



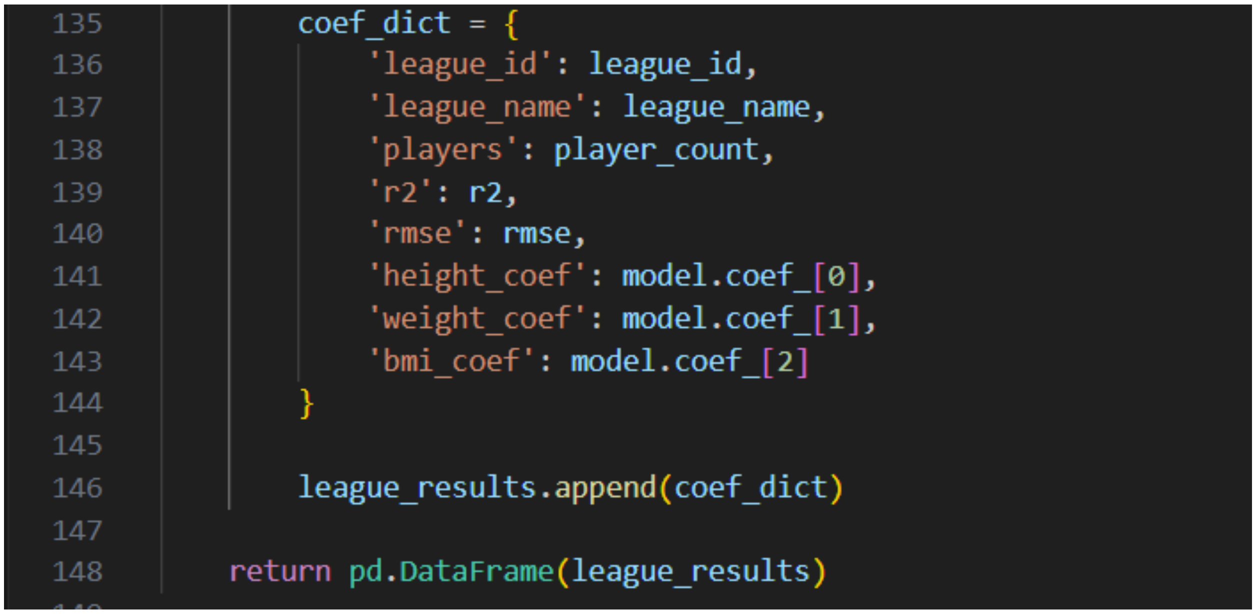
```
        y_pred = model.predict(X_scaled)
```

This line is to get predictions.



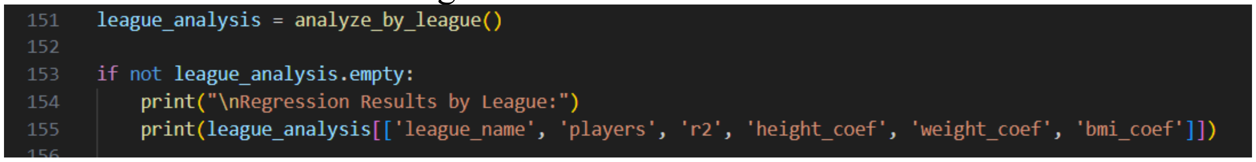
```
        r2 = r2_score(y, y_pred)
        rmse = np.sqrt(mean_squared_error(y, y_pred))
```

This line of code is for the metrics calculations.



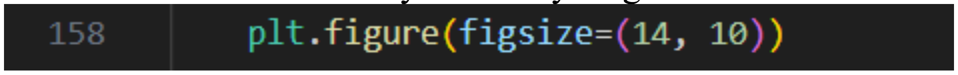
```
        coef_dict = {
            'league_id': league_id,
            'league_name': league_name,
            'players': player_count,
            'r2': r2,
            'rmse': rmse,
            'height_coef': model.coef_[0],
            'weight_coef': model.coef_[1],
            'bmi_coef': model.coef_[2]
        }

        league_results.append(coef_dict)

    return pd.DataFrame(league_results)
```

This line of code is for storing a coefficient.



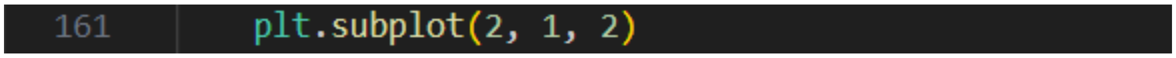
```
league_analysis = analyze_by_league()

if not league_analysis.empty:
    print("\nRegression Results by League:")
    print(league_analysis[['league_name', 'players', 'r2', 'height_coef', 'weight_coef', 'bmi_coef']])
```

This line of code is to analyze data by league.



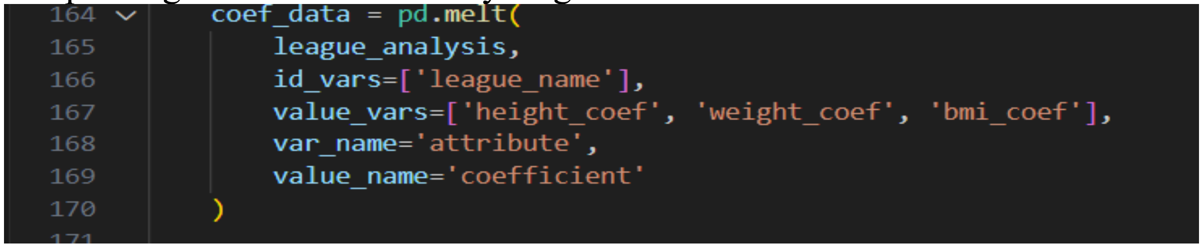
```
    plt.figure(figsize=(14, 10))
```

This line is for visualisation.



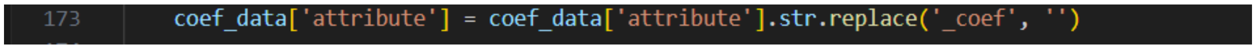
```
    plt.subplot(2, 1, 2)
```

For plotting coefficient values by league.



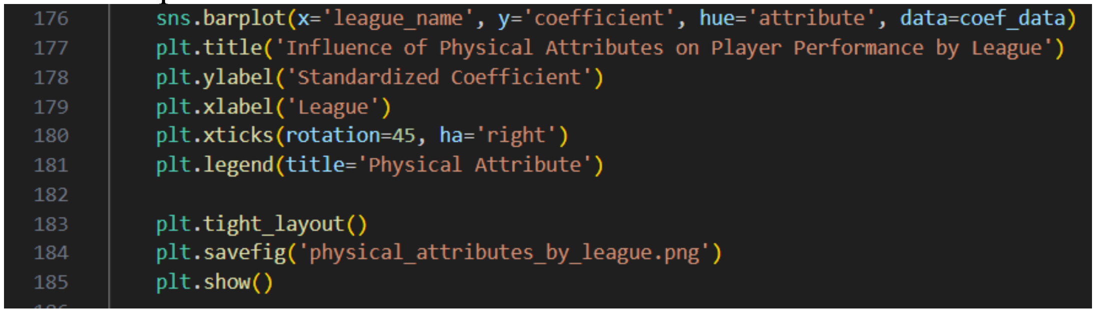
```
    coef_data = pd.melt(
        league_analysis,
        id_vars=['league_name'],
        value_vars=['height_coef', 'weight_coef', 'bmi_coef'],
        var_name='attribute',
        value_name='coefficient'
    )
```

This line is for reshaping data for plotting.



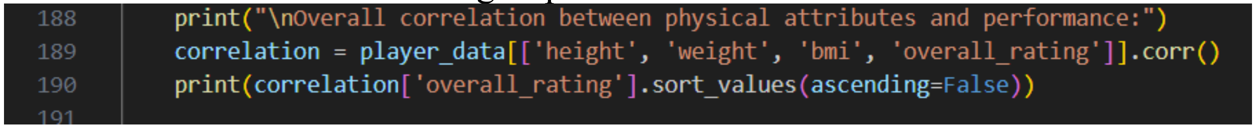
```python
coef_data['attribute'] = coef_data['attribute'].str.replace('_coef', '')
```

To clean up the attribute names.



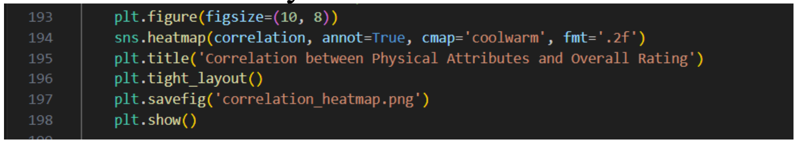
```python
sns.barplot(x='league_name', y='coefficient', hue='attribute', data=coef_data)
plt.title('Influence of Physical Attributes on Player Performance by League')
plt.ylabel('Standardized Coefficient')
plt.xlabel('League')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Physical Attribute')

plt.tight_layout()
plt.savefig('physical_attributes_by_league.png')
plt.show()
```

This lines of code is to create a grouped bar chart.



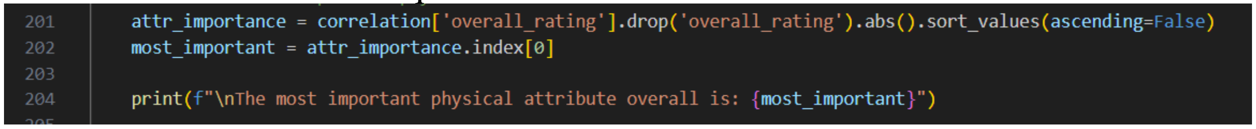
```python
print("\nOverall correlation between physical attributes and performance:")
correlation = player_data[['height', 'weight', 'bmi', 'overall_rating']].corr()
print(correlation['overall_rating'].sort_values(ascending=False))
```

For additional analysis: overall correlation across all leagues.



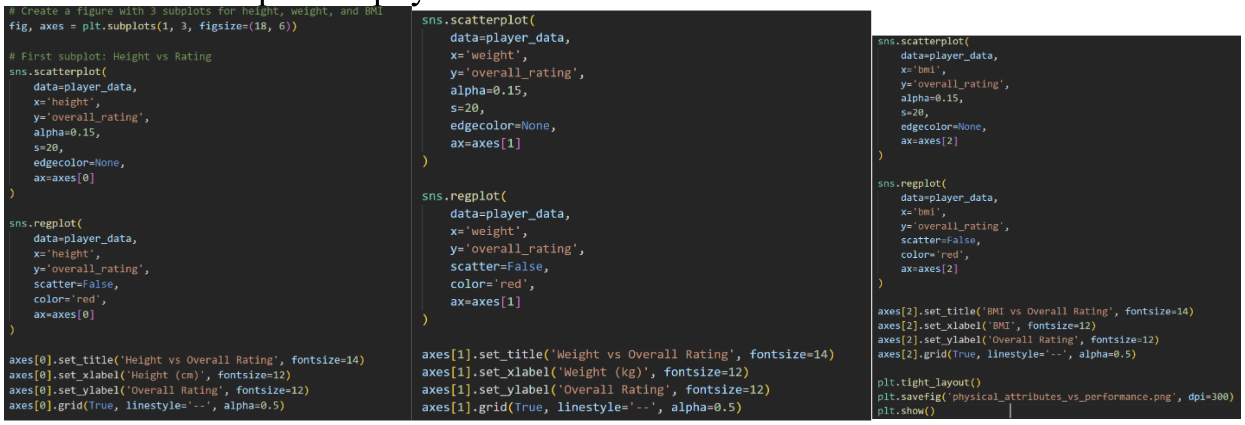
```python
plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation between Physical Attributes and Overall Rating')
plt.tight_layout()
plt.savefig('correlation_heatmap.png')
plt.show()
```

Create a correlation heatmap.



```python
attr_importance = correlation['overall_rating'].drop('overall_rating').abs().sort_values(ascending=False)
most_important = attr_importance.index[0]

print(f"\nThe most important physical attribute overall is: {most_important}")
```

Find the most important physical attribute overall.

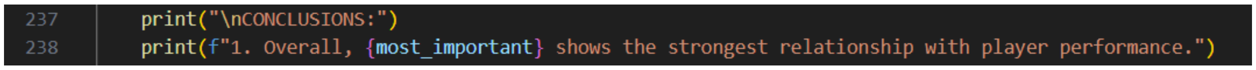


```python
# Create a figure with 3 subplots for height, weight, and BMI
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# First subplot: Height vs Rating
sns.scatterplot(
    data=player_data,
    x='height',
    y='overall_rating',
    alpha=0.15,
    s=20,
    edgecolor=None,
    ax=axes[0]
)

sns.regplot(
    data=player_data,
    x='height',
    y='overall_rating',
    scatter=False,
    color='red',
    ax=axes[0]
)

axes[0].set_title('Height vs Overall Rating', fontsize=14)
axes[0].set_xlabel('Height (cm)', fontsize=12)
axes[0].set_ylabel('Overall Rating', fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.5)
```

```python
sns.scatterplot(
    data=player_data,
    x='weight',
    y='overall_rating',
    alpha=0.15,
    s=20,
    edgecolor=None,
    ax=axes[1]
)

sns.regplot(
    data=player_data,
    x='weight',
    y='overall_rating',
    scatter=False,
    color='red',
    ax=axes[1]
)

axes[1].set_title('Weight vs Overall Rating', fontsize=14)
axes[1].set_xlabel('Weight (kg)', fontsize=12)
axes[1].set_ylabel('Overall Rating', fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.5)
```

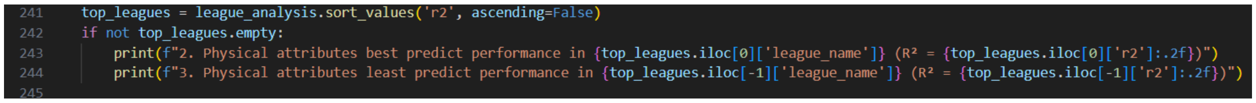
```python
sns.scatterplot(
    data=player_data,
    x='bmi',
    y='overall_rating',
    alpha=0.15,
    s=20,
    edgecolor=None,
    ax=axes[2]
)

sns.regplot(
    data=player_data,
    x='bmi',
    y='overall_rating',
    scatter=False,
    color='red',
    ax=axes[2]
)

axes[2].set_title('BMI vs Overall Rating', fontsize=14)
axes[2].set_xlabel('BMI', fontsize=12)
axes[2].set_ylabel('Overall Rating', fontsize=12)
axes[2].grid(True, linestyle='--', alpha=0.5)

plt.tight_layout()
plt.savefig('physical_attributes_vs_performance.png', dpi=300)
plt.show()
```

Improve the scatter plot of BMI vs overall rating.
These lines of code are for Seaborn, Seaborn's scatterplot is for better style and transparency.



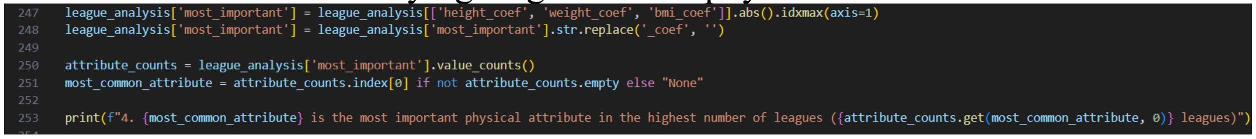
```python
print("\nCONCLUSIONS:")
print(f"1. Overall, {most_important} shows the strongest relationship with player performance.")
```

This line is for the conclusion.



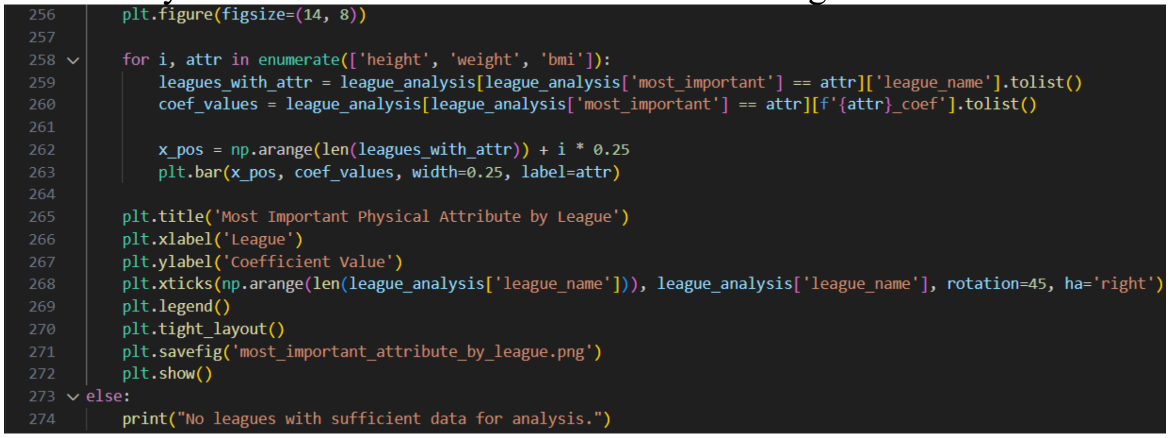
```python
    top_leagues = league_analysis.sort_values('r2', ascending=False)
    if not top_leagues.empty:
        print(f"2. Physical attributes best predict performance in {top_leagues.iloc[0]['league_name']} (R² = {top_leagues.iloc[0]['r2']:.2f})")
        print(f"3. Physical attributes least predict performance in {top_leagues.iloc[-1]['league_name']} (R² = {top_leagues.iloc[-1]['r2']:.2f})")
```

This line is used for identifying leagues where physical attributes matter most.



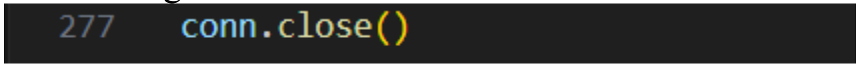
```python
league_analysis['most_important'] = league_analysis[['height_coef', 'weight_coef', 'bmi_coef']].abs().idxmax(axis=1)
league_analysis['most_important'] = league_analysis['most_important'].str.replace('_coef', '')

attribute_counts = league_analysis['most_important'].value_counts()
most_common_attribute = attribute_counts.index[0] if not attribute_counts.empty else "None"

print(f"4. {most_common_attribute} is the most important physical attribute in the highest number of leagues ({attribute_counts.get(most_common_attribute, 0)} leagues)")
```

For identify which attribute matters most in which league.



```python
    plt.figure(figsize=(14, 8))

    for i, attr in enumerate(['height', 'weight', 'bmi']):
        leagues_with_attr = league_analysis[league_analysis['most_important'] == attr]['league_name'].tolist()
        coef_values = league_analysis[league_analysis['most_important'] == attr][f'{attr}_coef'].tolist()

        x_pos = np.arange(len(leagues_with_attr)) + i * 0.25
        plt.bar(x_pos, coef_values, width=0.25, label=attr)

    plt.title('Most Important Physical Attribute by League')
    plt.xlabel('League')
    plt.ylabel('Coefficient Value')
    plt.xticks(np.arange(len(league_analysis['league_name'])), league_analysis['league_name'], rotation=45, ha='right')
    plt.legend()
    plt.tight_layout()
    plt.savefig('most_important_attribute_by_league.png')
    plt.show()
else:
    print("No leagues with sufficient data for analysis.")
```

This is for the final bar graph showing which physical attribute is most important in each league.

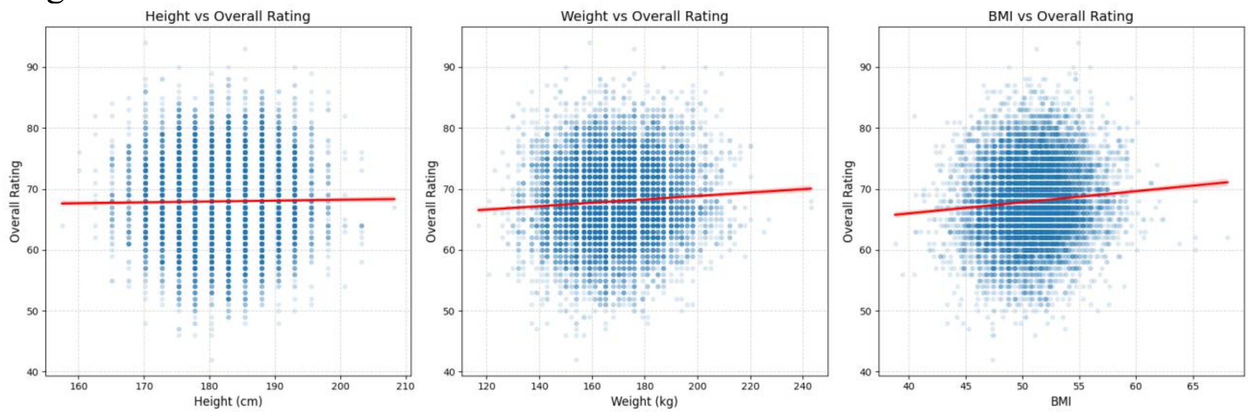


```python
conn.close()
```

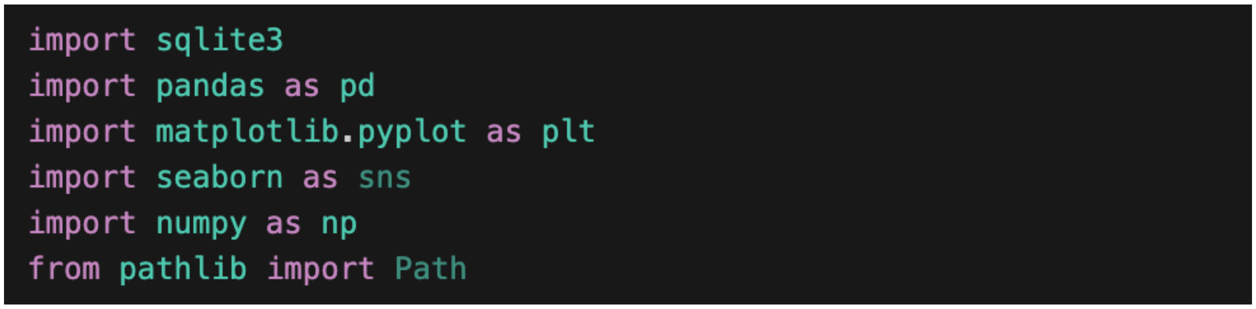
Close a connection to the database.

Q2: Identify which physical attributes (height, weight, BMI) best predict player performance in different positions, and quantify whether these relationships vary by league.?

Based on Seaborn scatter analysis, BMI versus overall rating is the best predictor of player performance in different positions, and it quantifies these relationships by league.performance in different positions and quantify these relationships by league.

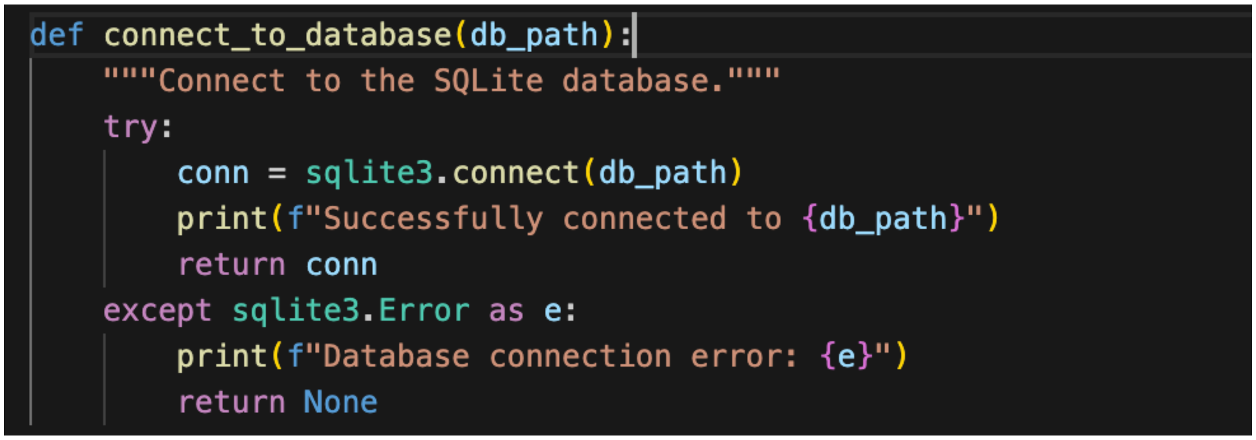


Question3:



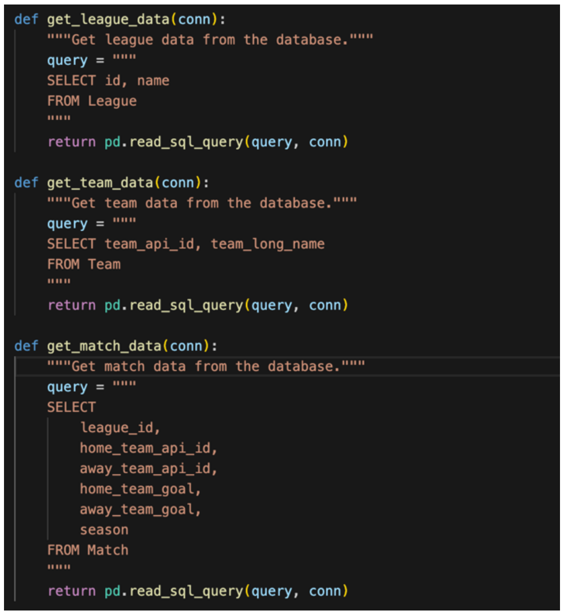
```python
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from pathlib import Path
```

Importing a library similar to Question 1 but with some additions:
numpy: Used to manipulate numbers or arrays, such as creating a set of numbers.
seaborn: Built on top of matplotlib, seaborn is used to make the plots more visually appealing and easier to understand.
path: Used to manage file paths.



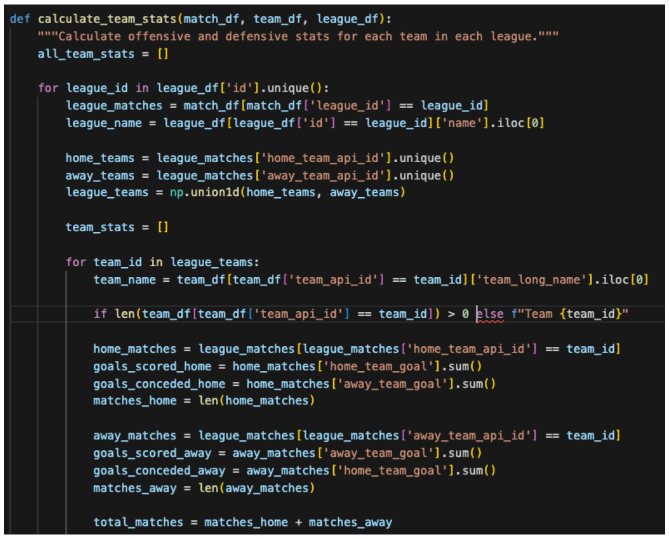
```python
def connect_to_database(db_path):
    """Connect to the SQLite database."""
    try:
        conn = sqlite3.connect(db_path)
        print(f"Successfully connected to {db_path}")
        return conn
    except sqlite3.Error as e:
        print(f"Database connection error: {e}")
        return None
```

Connect to the database from .sqlite file. If the connection is successful, conn will be returned. If it fails, a Database connection error: will be displayed.

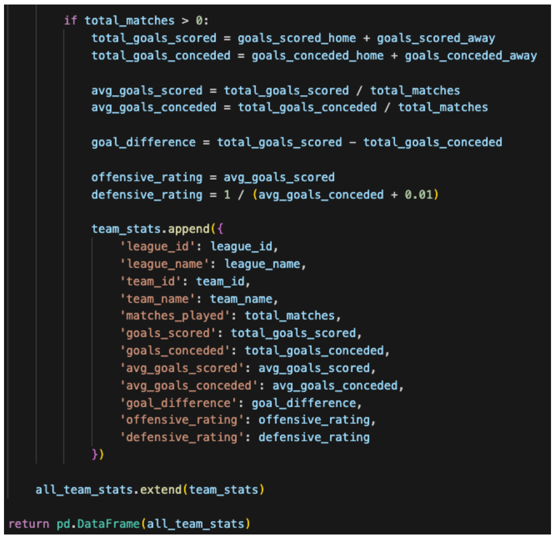


```python
def get_league_data(conn):
    """Get league data from the database."""
    query = """
    SELECT id, name
    FROM League
    """
    return pd.read_sql_query(query, conn)

def get_team_data(conn):
    """Get team data from the database."""
    query = """
    SELECT team_api_id, team_long_name
    FROM Team
    """
    return pd.read_sql_query(query, conn)

def get_match_data(conn):
    """Get match data from the database."""
    query = """
    SELECT
        league_id,
        home_team_api_id,
        away_team_api_id,
        home_team_goal,
        away_team_goal,
        season
    FROM Match
    """
    return pd.read_sql_query(query, conn)
```

Each function retrieves data from the database, producing three datasets:
one for leagues, one for teams, and one for match information.
League Data: Get the league name and ID
Team Data: Get each team's unique ID and full name.
Match Data: Get match information such as home/away team, number of goals, and season.

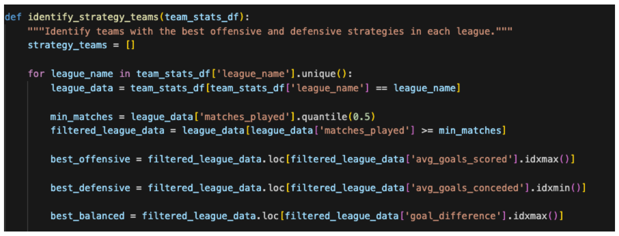


```python
def calculate_team_stats(match_df, team_df, league_df):
    """Calculate offensive and defensive stats for each team in each league."""
    all_team_stats = []

    for league_id in league_df['id'].unique():
        league_matches = match_df[match_df['league_id'] == league_id]
        league_name = league_df[league_df['id'] == league_id]['name'].iloc[0]

        home_teams = league_matches['home_team_api_id'].unique()
        away_teams = league_matches['away_team_api_id'].unique()
        league_teams = np.union1d(home_teams, away_teams)

        team_stats = []

        for team_id in league_teams:
            team_name = team_df[team_df['team_api_id'] == team_id]['team_long_name'].iloc[0]

            if len(team_df[team_df['team_api_id'] == team_id]) > 0 else f"Team {team_id}"

            home_matches = league_matches[league_matches['home_team_api_id'] == team_id]
            goals_scored_home = home_matches['home_team_goal'].sum()
            goals_conceded_home = home_matches['away_team_goal'].sum()
            matches_home = len(home_matches)

            away_matches = league_matches[league_matches['away_team_api_id'] == team_id]
            goals_scored_away = away_matches['away_team_goal'].sum()
            goals_conceded_away = away_matches['home_team_goal'].sum()
            matches_away = len(away_matches)

            total_matches = matches_home + matches_away
```

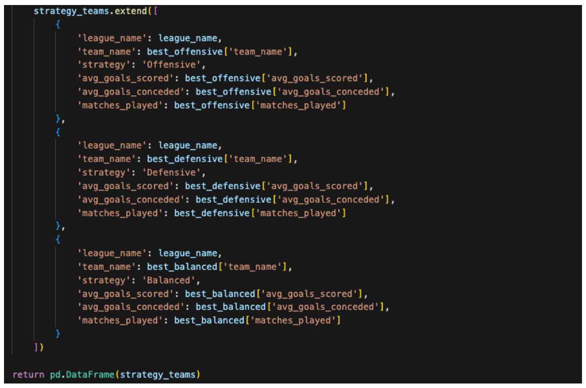


```python
            if total_matches > 0:
                total_goals_scored = goals_scored_home + goals_scored_away
                total_goals_conceded = goals_conceded_home + goals_conceded_away

                avg_goals_scored = total_goals_scored / total_matches
                avg_goals_conceded = total_goals_conceded / total_matches

                goal_difference = total_goals_scored - total_goals_conceded

                offensive_rating = avg_goals_scored
                defensive_rating = 1 / (avg_goals_conceded + 0.01)

                team_stats.append({
                    'league_id': league_id,
                    'league_name': league_name,
                    'team_id': team_id,
                    'team_name': team_name,
                    'matches_played': total_matches,
                    'goals_scored': total_goals_scored,
                    'goals_conceded': total_goals_conceded,
                    'avg_goals_scored': avg_goals_scored,
                    'avg_goals_conceded': avg_goals_conceded,
                    'goal_difference': goal_difference,
                    'offensive_rating': offensive_rating,
                    'defensive_rating': defensive_rating
                })

        all_team_stats.extend(team_stats)

    return pd.DataFrame(all_team_stats)
```

This function is used to calculate the performance statistics of teams in different leagues by pulling data from the match data, team data, and league data for each team in each league. The function computes various statistics, such as the number of matches played, goals scored, goals conceded, average goals scored and conceded, goal difference, offensive rating, and defensive rating, using data from both home and away matches. The results are returned in the form of a DataFrame that contains the statistics for each team in each league.

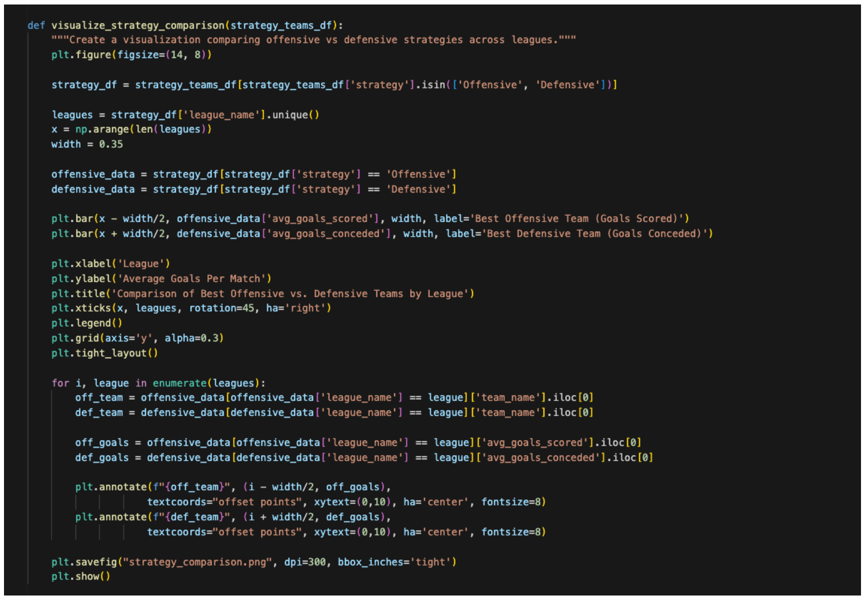


```python
def identify_strategy_teams(team_stats_df):
    """Identify teams with the best offensive and defensive strategies in each league."""
    strategy_teams = []

    for league_name in team_stats_df['league_name'].unique():
        league_data = team_stats_df[team_stats_df['league_name'] == league_name]

        min_matches = league_data['matches_played'].quantile(0.5)
        filtered_league_data = league_data[league_data['matches_played'] >= min_matches]

        best_offensive = filtered_league_data.loc[filtered_league_data['avg_goals_scored'].idxmax()]

        best_defensive = filtered_league_data.loc[filtered_league_data['avg_goals_conceded'].idxmin()]

        best_balanced = filtered_league_data.loc[filtered_league_data['goal_difference'].idxmax()]
```



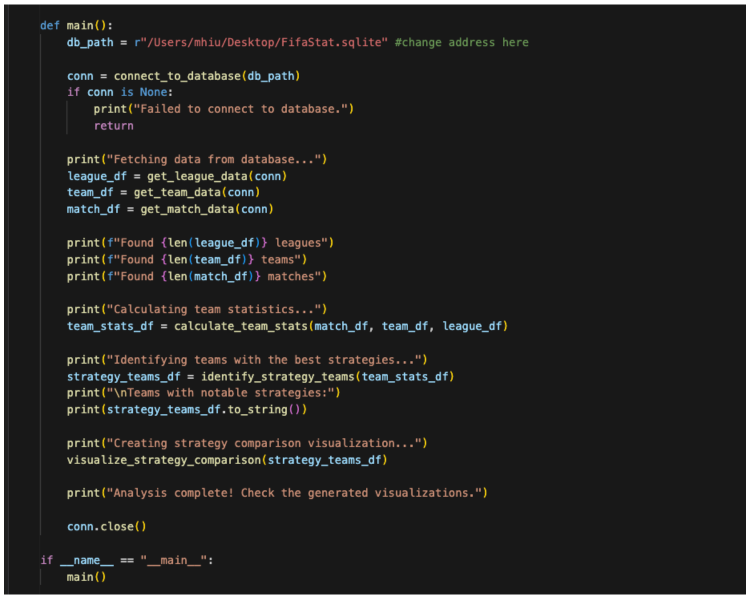
```python
        strategy_teams.extend([
            {
                'league_name': league_name,
                'team_name': best_offensive['team_name'],
                'strategy': 'Offensive',
                'avg_goals_scored': best_offensive['avg_goals_scored'],
                'avg_goals_conceded': best_offensive['avg_goals_conceded'],
                'matches_played': best_offensive['matches_played']
            },
            {
                'league_name': league_name,
                'team_name': best_defensive['team_name'],
                'strategy': 'Defensive',
                'avg_goals_scored': best_defensive['avg_goals_scored'],
                'avg_goals_conceded': best_defensive['avg_goals_conceded'],
                'matches_played': best_defensive['matches_played']
            },
            {
                'league_name': league_name,
                'team_name': best_balanced['team_name'],
                'strategy': 'Balanced',
                'avg_goals_scored': best_balanced['avg_goals_scored'],
                'avg_goals_conceded': best_balanced['avg_goals_conceded'],
                'matches_played': best_balanced['matches_played']
            }
        ])

    return pd.DataFrame(strategy_teams)
```

This function identifies teams with the best playing strategies in each league, categorized into three types: Offensive, Defensive, and Balanced. The function filters teams with sufficient matches (greater than the median number of matches) and selects the teams with the best offensive strategy (highest average goals scored), best defensive strategy (lowest average goals conceded), and best-balanced strategy (highest goal difference). The result is a DataFrame displaying the team's name, strategy, average goals scored, average goals conceded, and the number of matches played for the best teams in each strategy.



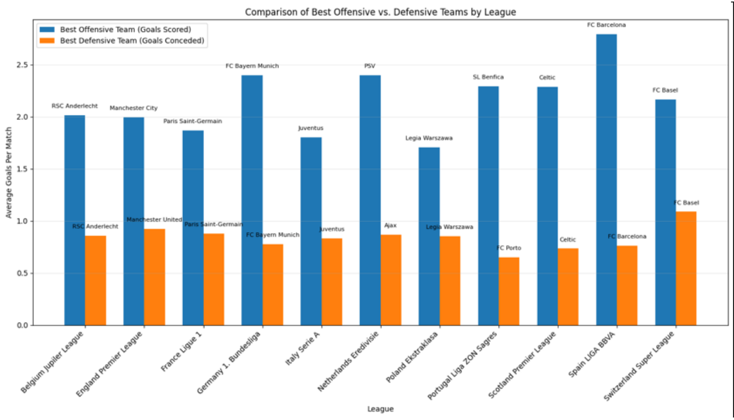
```python
def visualize_strategy_comparison(strategy_teams_df):
    """Create a visualization comparing offensive vs defensive strategies across leagues."""
    plt.figure(figsize=(14, 8))

    strategy_df = strategy_teams_df[strategy_teams_df['strategy'].isin(['Offensive', 'Defensive'])]

    leagues = strategy_df['league_name'].unique()
    x = np.arange(len(leagues))
    width = 0.35

    offensive_data = strategy_df[strategy_df['strategy'] == 'Offensive']
    defensive_data = strategy_df[strategy_df['strategy'] == 'Defensive']

    plt.bar(x - width/2, offensive_data['avg_goals_scored'], width, label='Best Offensive Team (Goals Scored)')
    plt.bar(x + width/2, defensive_data['avg_goals_conceded'], width, label='Best Defensive Team (Goals Conceded)')

    plt.xlabel('League')
    plt.ylabel('Average Goals Per Match')
    plt.title('Comparison of Best Offensive vs. Defensive Teams by League')
    plt.xticks(x, leagues, rotation=45, ha='right')
    plt.legend()
    plt.grid(axis='y', alpha=0.3)
    plt.tight_layout()

    for i, league in enumerate(leagues):
        off_team = offensive_data[offensive_data['league_name'] == league]['team_name'].iloc[0]
        def_team = defensive_data[defensive_data['league_name'] == league]['team_name'].iloc[0]

        off_goals = offensive_data[offensive_data['league_name'] == league]['avg_goals_scored'].iloc[0]
        def_goals = defensive_data[defensive_data['league_name'] == league]['avg_goals_conceded'].iloc[0]

        plt.annotate(f"{off_team}", (i - width/2, off_goals),
                     textcoords="offset points", xytext=(0,10), ha='center', fontsize=8)
        plt.annotate(f"{def_team}", (i + width/2, def_goals),
                     textcoords="offset points", xytext=(0,10), ha='center', fontsize=8)

    plt.savefig("strategy_comparison.png", dpi=300, bbox_inches='tight')
    plt.show()
```

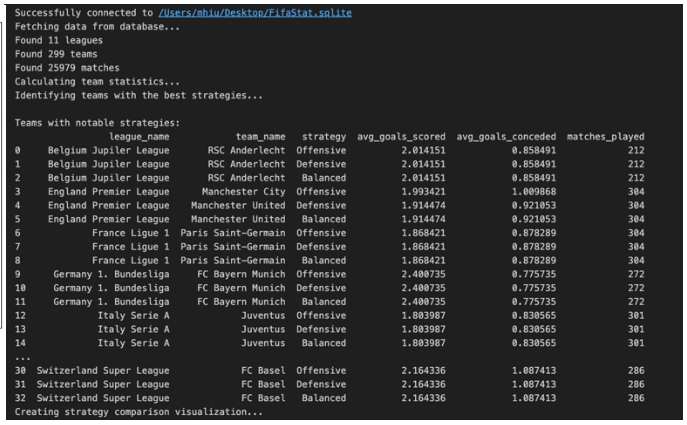
This function creates a bar chart to compare the best offensive and defensive teams across different leagues. It focuses only on teams categorized under the "Offensive" and "Defensive" strategies and visualizes their average goals scored and average goals conceded. The function uses side-by-side bars for each league to display these metrics, adds team names above the bars for clarity, and labels the axes and legend. It saves the plot and displays it.



```python
def main():
    db_path = r"/Users/mhiu/Desktop/FifaStat.sqlite" #change address here

    conn = connect_to_database(db_path)
    if conn is None:
        print("Failed to connect to database.")
        return

    print("Fetching data from database...")
    league_df = get_league_data(conn)
    team_df = get_team_data(conn)
    match_df = get_match_data(conn)

    print(f"Found {len(league_df)} leagues")
    print(f"Found {len(team_df)} teams")
    print(f"Found {len(match_df)} matches")

    print("Calculating team statistics...")
    team_stats_df = calculate_team_stats(match_df, team_df, league_df)

    print("Identifying teams with the best strategies...")
    strategy_teams_df = identify_strategy_teams(team_stats_df)
    print("\nTeams with notable strategies:")
    print(strategy_teams_df.to_string())

    print("Creating strategy comparison visualization...")
    visualize_strategy_comparison(strategy_teams_df)

    print("Analysis complete! Check the generated visualizations.")

    conn.close()

if __name__ == "__main__":
    main()
```

The main() function begins by connecting to a SQLite database using a specified file path. If the connection fails, the program exits. If connected, it retrieves league, team, and match data from the database and displays the number of records found. It then calls calculate_team_stats to compute each team's statistics, such as the number of matches, goals scored and conceded, and performance ratings. Next, it calls identify_strategy_teams to analyze which teams in each league have the best offensive, defensive, and balanced strategies, and prints the results. Finally, it calls visualize_strategy_comparison to generate a chart comparing those strategies and closes the database connection once everything is complete.





```
Successfully connected to /Users/mhiu/Desktop/FifaStat.sqlite
Fetching data from database...
Found 11 leagues
Found 299 teams
Found 25979 matches
Calculating team statistics...
Identifying teams with the best strategies...

Teams with notable strategies:
                  league_name            team_name   strategy  avg_goals_scored  avg_goals_conceded  matches_played
0     Belgium Jupiler League       RSC Anderlecht  Offensive          2.014151            0.858491             212
1     Belgium Jupiler League       RSC Anderlecht  Defensive          2.014151            0.858491             212
2     Belgium Jupiler League       RSC Anderlecht   Balanced          2.014151            0.858491             212
3     England Premier League      Manchester City  Offensive          1.993421            1.009868             304
4     England Premier League    Manchester United  Defensive          1.914474            0.921053             304
5     England Premier League    Manchester United   Balanced          1.914474            0.921053             304
6             France Ligue 1  Paris Saint-Germain  Offensive          1.868421            0.878289             304
7             France Ligue 1  Paris Saint-Germain  Defensive          1.868421            0.878289             304
8             France Ligue 1  Paris Saint-Germain   Balanced          1.868421            0.878289             304
9      Germany 1. Bundesliga     FC Bayern Munich  Offensive          2.400735            0.775735             272
10     Germany 1. Bundesliga     FC Bayern Munich  Defensive          2.400735            0.775735             272
11     Germany 1. Bundesliga     FC Bayern Munich   Balanced          2.400735            0.775735             272
12              Italy Serie A             Juventus  Offensive          1.803987            0.830565             301
13              Italy Serie A             Juventus  Defensive          1.803987            0.830565             301
14              Italy Serie A             Juventus   Balanced          1.803987            0.830565             301
...
30  Switzerland Super League             FC Basel  Offensive          2.164336            1.087413             286
31  Switzerland Super League             FC Basel  Defensive          2.164336            1.087413             286
32  Switzerland Super League             FC Basel   Balanced          2.164336            1.087413             286
Creating strategy comparison visualization...
```

Q3. Analyze the defensive vs. offensive balance: For each league, examine the relationship between goals scored and goals conceded, identifying teams that succeeded with defensive-focused strategies versus offensive-focused approaches.

From the result, we can see how each team's offensive, defensive, and balanced strategies truly perform in terms of average goals scored and conceded:

- **RSC Anderlecht (Belgium)** shows identical numbers across all strategies, suggesting the team's performance is stable regardless of tactical shifts.
- **Manchester City vs. Manchester United (England)**: City excels in offense (1.99 goals), while United performs better defensively (only 0.92 goals conceded), highlighting a division of strengths rather than all-around balance.
- **Paris Saint-Germain (France)** maintains a consistent (though moderate) performance across all strategies, reliable but not standout.
- **Bayern Munich (Germany)** remains top-tier in every strategy (2.4 scored, only 0.77 conceded), proving its tactical setup is dominant and stable no matter the approach.
- **Juventus (Italy)** emphasizes defense (0.83 conceded) with consistent, if modest, offensive output, suggesting a structurally cautious team.
- **FC Basel (Switzerland)** scores well (2.16), but concedes the most (1.08) across all strategies, indicating systemic defensive issues regardless of approach.