Final Project

Group 12:

Bader Albaarak, Ta Wei Lo, Shih-Wei Pan, Leela Bharani Teddu, Rose Hemans

Group Lead:

Bader Albaarak

Ask 1 - Search for a dataset

Our team decided to analyze English Premier League data for 21 Seasons between 1997/1998 to 2017/2018 for our final project. The dataset includes the date of games, referee, fixture name, home and away teams, and 21 different measured match events such as Points (Full Time), Corners Played, Shots, Shots on Target etc. The dataset size is 52.4 MB and has 335,160 records and 14 variables. It contains both text and numerical format to present the information.

We can download CSV file from (https://data.world/africananalyst/english-premier-league-data-for-20-seasons-to-20172018)

The data was gathered and cleaned by Ahmed Oyelowo using Microsoft Power Query ensuring the table is as slim as possible. So there is only one single field for "Values" which is described by another field called "Attribute" such as goals scored, corners conceded, yellow cards etc. This makes the data more organized for analysis.

To modify the dataset, we will use csvkit to conduct some preliminary analysis, remove additional columns and create a cleaner dataset which we will use the for the project.

First we will load SQL, create a new database, and retrieve the dataset from a public link where we downloaded the database to. The S3 bucket is open for anyone to download the dataset for to ensure reproducibility of this notebook.

Dataset Source

The dataset above represents the English Premier League, which is the highest level of professional soccer played in England. The dataset itself is derived from official data released the by Premier League. Additional statistics that are more player specific can be found from the Premier League website (https://www.premierleague.com/stats).

Why is this dataset important and what appeals to you about it?

Soccer is a business, with many shareholders, and even more stakeholders around the world. Like in any business, decision making is crucial to the success of teams and nations. With sports analytics, we can interpret the meaning of data, explain our findings, and relate them to overall concerns about how to play and ensure more success or reduce failures.

Currently in soccer, there is a wealth of data available from GPS data, player performance data, injury data, to team performance data. Insights from this data can be delivered to players, fans, coaches, recruitment staff and soccer executives.

In the past 20 years, data analysis has gone from being relatively unheard of in soccer to a crucial element of success. In 2020, liverpool won the Premier League thanks to the perfect recruitment of players who they invested millions in. Their player recruitment was driven by their data analysis team and the team was applauded for their contribution to the title.

Sports data analytics companies such as Opta, Prozone, and StatsSport have provided another dimension to the game as the quality of play improves every year. We are interested in analyzing long-term trends between different match events and their effects on overall success for teams. We think this is a great time to select a soccer dataset during the 2022 World Cup. While domestic leagues like the Premier League take a break for this international competition in Qatar, now is the perfect time to answer some analytical questions using this data for interested businesses.

Is this dataset suitable for dimensional modeling and analysis?

Upon exploring the dataset, we find that it is suitable for dimensional modeling and analytical analysis. The categorical attributes for match event are clearly outlined as well as the values counting each time those events occurred. The data is mostly clean, with some data in older years missing. We will address this during the project.

Analytical questions we can answer with this data

Business Question 1:

What is the best performing team with highest score across the seasons between 2007|2008 and 2017|2018? What strategy earns more points-'Attack' vs 'Defense' attributes.

Business Question 2:

Who is the strictest referee that issued the most yellow and red cards during 2015-2018, that we need to be aware of? Over time, what's the trend of red cards and yellow cards issued?

Business Question 3:

What is the total points of each team across 21 seasons? How does the corner kick effect the team's total points? What is the relationship between these attributes?

Business Question 4:

Which team had received the highest number of yellow cards during the season 2017-2018 and under which referee did they receive them from? From this, we can advise certain teams to play more conservatively in future games when they are assigned these referees.

Concerns with the data and changes we expect to overcome

This dataset has 335,160 rows, we anticipate the processing time for wrangling and data analysis to be faster than the CitiBike dataset which had approximately 3.7 million rows of transactional data. However, at 335k rows, we still consider this Premier League dataset to be big data and require more processing time than smaller datasets that do not require additional computing capacity we will be utilizing Amazon Elastic Compute Cloud for.

Currently, the dataset has the following headers;

date referee year month_no month_name day date_for_season fixture key season side team attribute_1 value

We have observed that date_for_season and date appear to be identical and we may be able to reduce the size of the dataset by removing the date_for_season column.

After some initial data exploration, we can see that the values for the key column are a mix of numerical and text values. We have found with our previous experience of working with large datasets that it is sometimes easier to work with only numerical values and therefore the keys that we add to dimensional tables will help with this.

The final concern we have with the dataset is that it was last updated 4 years ago. Our data analysis would be considered out of date for any analysis to affect decision making if we were to consult with any of the teams affected. We will consider finding additional data to supplement or update our analysis.

Ask 2 - Data Wrangling and Dimensional Modeling

Setup SQL and create the final project database:

In [1]:
```
%load_ext sql
```

In [2]:
```
!dropdb -U student finalproject
```

In [3]:
```
!createdb -U student finalproject
```

In [4]:
```
%sql postgresql://student@/finalproject
```

Retrieve the dataset saved to an AWS S3 bucket:

In [5]:
```
!wget https://myhemansbucket.s3.amazonaws.com/EPL_20_Yrs_Data_1997_to_2018.csv
```

```
--2022-12-10 03:01:07--  https://myhemansbucket.s3.amazonaws.com/EPL_20_Yrs_Data_1997
_to_2018.csv
Resolving myhemansbucket.s3.amazonaws.com (myhemansbucket.s3.amazonaws.com)... 54.23
1.171.49, 54.231.171.9, 52.217.227.65, ...
Connecting to myhemansbucket.s3.amazonaws.com (myhemansbucket.s3.amazonaws.com)|54.23
1.171.49|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 52410576 (50M) [text/csv]
Saving to: ‘EPL_20_Yrs_Data_1997_to_2018.csv’

EPL_20_Yrs_Data_199 100%[===================>]  49.98M  68.1MB/s    in 0.7s

2022-12-10 03:01:07 (68.1 MB/s) - ‘EPL_20_Yrs_Data_1997_to_2018.csv’ saved [52410576/
52410576]
```

In [6]:
```
!mv EPL_20_Yrs_Data_1997_to_2018.csv prem97to18.csv
```

Fix encoding, check column names and data head:

In [7]:
```
import pandas as pd
prem = pd.read_csv('prem97to18.csv',encoding='latin-1')
```

In [8]:
```
prem.head()
```

Out[8]:

	Date	Referee	Year	Month No	Month Name	Day	Date for season	Fixture	Key	Season	
0	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom
1	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom
2	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom
3	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom
4	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom

We will begin with this schema before we develop our complete dimensional model:

In [9]:
```
from IPython.display import Image
Image(url="https://myhemansbucket.s3.amazonaws.com/beginningschema.png")
```

Out[9]:

Now we will rename the columns so they are clear and related to the sport.

In [10]:
```
prem.columns = ['Date', 'Referee', 'Year', 'Month No', 'Month Name', 'Day',
       'Date for season', 'Fixture', 'Game id', 'Season', 'Side', 'Team',
       'match_event', 'match_event_count']
```

In [11]:
```
prem.head(1)
```

Out[11]:

	Date	Referee	Year	Month No	Month Name	Day	Date for season	Fixture	Game id	Season	
0	26-Dec-98	NaN	1998	12	December	Saturday	26-Dec-98	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham	1998 | 1999	Hom

Identify how many unique games there are to continue cleaning the dataset so game_id_key can be used as database's primary key:

In [12]:
```
prem_gameid = prem.groupby("Game id").count().index.tolist()
```

In [13]:
```
len(prem_gameid)
```

Out[13]:
```
7980
```

Now we will only use csvkit to wrangle the data.

There are 7980 unique games in the dataset. We will create a new dataframe using these unique games and corresponding data from the dataset by creating new columns for home and away count of attributes:

In [14]:
```
prem.to_csv(r'/home/ubuntu/notebooks/final_project/prem.csv',index=False, header=True)
```

In [15]:
```
!wc -l prem.csv
```

```
wc: prem.csv: No such file or directory
```

In [16]:
```
!csvcut -n prem.csv
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'prem.csv'
```

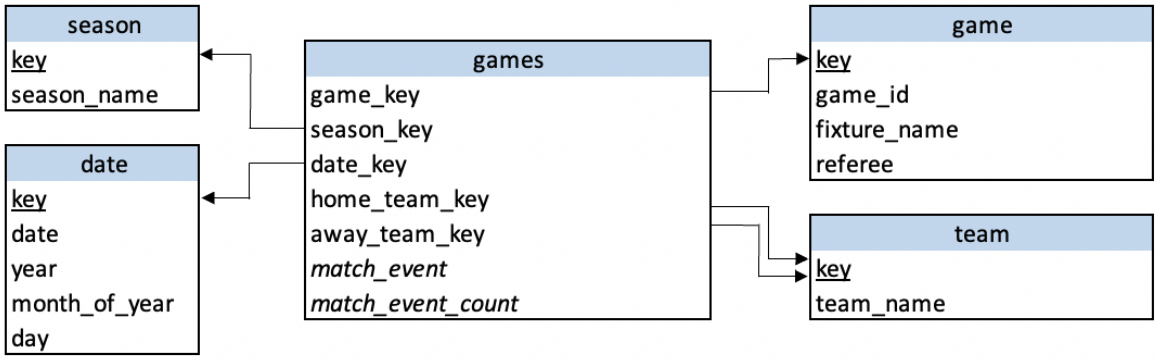
In [17]:
```
!head -n 50000 prem.csv | csvstat
```

```
head: cannot open 'prem.csv' for reading: No such file or directory
/home/ubuntu/.local/lib/python3.8/site-packages/agate/table/from_csv.py:74: RuntimeWa
rning: Error sniffing CSV dialect: Could not determine delimiter
StopIteration:
```

Now we can create a star schema to base our dimensional model on.

In [18]:
```
Image(url="https://myhemansbucket.s3.amazonaws.com/finalprojectstarschema2.png")
```

Out[18]:



Based on these values, we can expect to work with the following:

In [19]:
```
%%sql
DROP TABLE IF EXISTS games;

CREATE TABLE games (
    date TIMESTAMP NOT NULL,
    referee VARCHAR(64),
    year INTEGER NOT NULL,
    month_no INTEGER NOT NULL,
    month_name VARCHAR(64) NOT NULL,
    day VARCHAR(64) NOT NULL,
    date_for_season TIMESTAMP NOT NULL,
    fixture VARCHAR(64) NOT NULL,
    game_id VARCHAR(64) NOT NULL,
    season CHAR(11) NOT NULL,
    side VARCHAR(64) NOT NULL,
    team VARCHAR(64) NOT NULL,
    match_event VARCHAR(64) NOT NULL,
    match_event_count INTEGER NOT NULL
)
```

```
 * postgresql://student@/finalproject
Done.
Done.
```

Out[19]:
```
[]
```

We should look at the data for a single game in a table using xsvkit.

In [20]:
```
!xsv search "18/08/2012Arsenal vs Sunderland" prem.csv| csvsort -c13 |xsv table
```

```
failed to open prem.csv: No such file or directory (os error 2)
/home/ubuntu/.local/lib/python3.8/site-packages/agate/table/from_csv.py:74: RuntimeWa
rning: Error sniffing CSV dialect: Could not determine delimiter
StopIteration:
```

In [21]:
```
!xsv search "18/08/2012Arsenal vs Sunderland" prem.csv | xsv table | wc -l
```

```
failed to open prem.csv: No such file or directory (os error 2)
0
```

In [22]:
```
!pwd
```

```
/home/ubuntu/notebooks
```

Now we are loading the data directly using the `COPY` command and looking at the games table in SQL:

In [23]:
```
%%sql
COPY games FROM '/home/ubuntu/notebooks/final_project/prem.csv'
CSV
HEADER;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[23]: []

We can look at the number of rows of data using the `COUNT` command to see if its the same as the original dataset.

In [24]:
```
%%sql
SELECT COUNT(*) FROM games;
```

```
 * postgresql://student@/finalproject
1 rows affected.
```

Out[24]:

count
335160

We can view the first ten rows using the `SELECT` command:

In [25]:
```
%%sql
SELECT * FROM games
LIMIT 10
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[25]:

date	referee	year	month_no	month_name	day	date_for_season	fixture	game_id
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham
1998-12-26 00:00:00	None	1998	12	December	Saturday	1998-12-26 00:00:00	Arsenal vs West Ham	26/12/1998Arsenal vs West Ham

Our database will use game_id to identify unique games and enable us to work on analytics of games. We will look at `distinct` values for some columns:

In [26]:

```
%%sql
SELECT DISTINCT game_id
FROM games
ORDER BY game_id
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[26]:

game_id
01/01/2001Charlton vs Arsenal
01/01/2001Chelsea vs Aston Villa
01/01/2001Coventry vs Man City
01/01/2001Derby vs Everton
01/01/2001Leeds vs Middlesbrough
01/01/2001Leicester vs Bradford
01/01/2001Liverpool vs Southampton
01/01/2001Man United vs West Ham
01/01/2001Sunderland vs Ipswich
01/01/2002Charlton vs Ipswich

In [27]:

```
%%sql
SELECT DISTINCT season
FROM games
ORDER BY season;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[27]:

season
1997 | 1998
1998 | 1999
1999 | 2000
2000 | 2001
2001 | 2002
2002 | 2003
2003 | 2004
2004 | 2005
2005 | 2006
2006 | 2007
2007 | 2008
2008 | 2009
2009 | 2010
2010 | 2011
2011 | 2012
2012 | 2013
2013 | 2014
2014 | 2015
2015 | 2016
2016 | 2017
2017 | 2018

In [28]:
```
%%sql
SELECT DISTINCT date
FROM games
ORDER BY date
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[28]:

date
1997-08-09 00:00:00
1997-08-10 00:00:00
1997-08-11 00:00:00
1997-08-12 00:00:00
1997-08-13 00:00:00
1997-08-23 00:00:00
1997-08-24 00:00:00
1997-08-25 00:00:00
1997-08-26 00:00:00
1997-08-27 00:00:00

In [29]:
```
%%sql
SELECT DISTINCT date_for_season
FROM games
ORDER BY date_for_season
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[29]:

date_for_season
1997-08-09 00:00:00
1997-08-10 00:00:00
1997-08-11 00:00:00
1997-08-12 00:00:00
1997-08-13 00:00:00
1997-08-23 00:00:00
1997-08-24 00:00:00
1997-08-25 00:00:00
1997-08-26 00:00:00
1997-08-27 00:00:00

The columns date and date_for_season are identical. We will remove date_for_season. The columns for year, month, month_no, day, and month_name also will not improve our data analysis later on so we will remove them. Using just the date column, we can extract the day, month, and year of a game instead as more detailed time data. We will use the `ALTER TABLE` and `DROP COLUMN` commands.

In [30]:
```
%%sql
ALTER TABLE games
DROP COLUMN date_for_season,
```

```
DROP COLUMN year,
DROP COLUMN month_no,
DROP COLUMN day,
DROP COLUMN month_name;
```

```
 * postgresql://student@/finalproject
Done.
```

Out[30]: []

We are creating the season, date, game, team dimensions, modifying fact, and linking it to the games table using `DROP TABLE` and `CREATE TABLE`.

In [31]:
```
%%sql
DROP TABLE IF EXISTS season;

CREATE TABLE season (
    key SERIAL PRIMARY KEY,
    season_name CHAR(11)
);

DROP TABLE IF EXISTS game;

CREATE TABLE game (
    key SERIAL PRIMARY KEY,
    game_id VARCHAR(64),
    fixture_name VARCHAR(64),
    referee VARCHAR(64)
);

DROP TABLE IF EXISTS team;

CREATE TABLE team (
    key SERIAL PRIMARY KEY,
    team_name VARCHAR(64)
);

DROP TABLE IF EXISTS side;

CREATE TABLE side (
    key SERIAL PRIMARY KEY,
    home_away VARCHAR(64)
);

DROP TABLE IF EXISTS match_events;

CREATE TABLE match_events (
    key SERIAL PRIMARY KEY,
    match_event VARCHAR(64)
);
```

```
 * postgresql://student@/finalproject
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
```

Out[31]: []

We can extract the day, month, and year from the date column:

In [32]:
```
%%sql
SELECT DISTINCT date,
    TO_CHAR(date, 'YYYY-MM-DD HH24:00:00') AS hour,
    TO_CHAR(date, 'YYYY-MM-DD') AS day,
    TO_CHAR(date, 'YYYY') AS year,
    TO_CHAR(date, 'Month') AS month_of_year_str,
    TO_CHAR(date, 'MM') AS month_of_year,
    TO_CHAR(date, 'DD') AS day_of_month,
    TO_CHAR(date, 'Day') AS day_of_week_str,
    TO_CHAR(date, 'D') AS day_of_week,
    CASE WHEN CAST(TO_CHAR(date, 'D') AS INTEGER) in (7,1)
        THEN 'true'
        ELSE 'false'
    END AS is_weekend,
    CASE WHEN CAST(TO_CHAR(date, 'D') AS INTEGER) Not in (7,1)
        THEN 'true'
        ELSE 'false'
    END AS is_weekday,
    TO_CHAR(date, 'HH24') AS hour_of_day,
    TO_CHAR(date, 'Q') AS quarter_of_year
FROM games
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[32]:

date	hour	day	year	month_of_year_str	month_of_year	day_of_month	day_of_week_str	da
2014-01-18 00:00:00	2014-01-18 00:00:00	2014-01-18	2014	January	01	18	Saturday	
2013-04-29 00:00:00	2013-04-29 00:00:00	2013-04-29	2013	April	04	29	Monday	
2009-03-01 00:00:00	2009-03-01 00:00:00	2009-03-01	2009	March	03	01	Sunday	
2001-09-19 00:00:00	2001-09-19 00:00:00	2001-09-19	2001	September	09	19	Wednesday	
1998-11-23 00:00:00	1998-11-23 00:00:00	1998-11-23	1998	November	11	23	Monday	
1997-11-02 00:00:00	1997-11-02 00:00:00	1997-11-02	1997	November	11	02	Sunday	
2017-01-31 00:00:00	2017-01-31 00:00:00	2017-01-31	2017	January	01	31	Tuesday	
2015-05-16 00:00:00	2015-05-16 00:00:00	2015-05-16	2015	May	05	16	Saturday	
1998-04-26 00:00:00	1998-04-26 00:00:00	1998-04-26	1998	April	04	26	Sunday	
2003-01-28 00:00:00	2003-01-28 00:00:00	2003-01-28	2003	January	01	28	Tuesday	

In [33]:

```
%%sql
DROP TABLE IF EXISTS date;

CREATE TABLE date (
    key SERIAL PRIMARY KEY,
    hour CHAR(19),
    day CHAR(10),
    year INTEGER,
    month_of_year_str VARCHAR(12),
    month_of_year INTEGER,
    day_of_month INTEGER,
    day_of_week_str CHAR(9),
    day_of_week INTEGER,
    is_weekend BOOLEAN,
    is_weekday BOOLEAN,
    hour_of_day INTEGER,
    quarter_of_year INTEGER
);
```

```
 * postgresql://student@/finalproject
Done.
Done.
```

Out[33]: []

In [34]:
```
%%sql
INSERT INTO date (hour, day, year, month_of_year_str, month_of_year, day_of_month,
                  day_of_week_str, day_of_week, is_weekend, is_weekday,
                  hour_of_day, quarter_of_year)
SELECT DISTINCT TO_CHAR(date, 'YYYY-MM-DD HH24:00:00') AS hour,
    TO_CHAR(date, 'YYYY-MM-DD') AS day,
    CAST(TO_CHAR(date, 'YYYY') AS INTEGER) AS year,
    TO_CHAR(date, 'Month') AS month_of_year_str,
    CAST(TO_CHAR(date, 'MM') AS INTEGER) AS month_of_year,
    CAST(TO_CHAR(date, 'DD') AS INTEGER) AS day_of_month,
    TO_CHAR(date, 'Day') AS day_of_week_str,
    CAST(TO_CHAR(date, 'D') AS INTEGER) AS day_of_week,
    CASE WHEN CAST(TO_CHAR(date, 'D') AS INTEGER) IN (1, 7)
        THEN TRUE
        ELSE FALSE
    END AS is_weekend,
    CASE WHEN CAST(TO_CHAR(date, 'D') AS INTEGER) NOT IN (1, 7)
        THEN TRUE
        ELSE FALSE
    END AS is_weekday,
    CAST(TO_CHAR(date, 'HH24') AS INTEGER) AS hour_of_day,
    CAST(TO_CHAR(date, 'Q') AS INTEGER) AS quarter_of_year
FROM games;
```

```
 * postgresql://student@/finalproject
2117 rows affected.
```

Out[34]: []

In [35]:
```
%%sql
SELECT * FROM date
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[35]:

key	hour	day	year	month_of_year_str	month_of_year	day_of_month	day_of_week_str	day_of_
1	2013-12-26 00:00:00	2013-12-26	2013	December	12	26	Thursday	
2	2007-11-04 00:00:00	2007-11-04	2007	November	11	4	Sunday	
3	1998-02-07 00:00:00	1998-02-07	1998	February	2	7	Saturday	
4	2003-12-22 00:00:00	2003-12-22	2003	December	12	22	Monday	
5	2014-04-27 00:00:00	2014-04-27	2014	April	4	27	Sunday	
6	2002-10-21 00:00:00	2002-10-21	2002	October	10	21	Monday	
7	2014-12-22 00:00:00	2014-12-22	2014	December	12	22	Monday	
8	2013-08-24 00:00:00	2013-08-24	2013	August	8	24	Saturday	
9	1998-04-11 00:00:00	1998-04-11	1998	April	4	11	Saturday	
10	1998-01-10 00:00:00	1998-01-10	1998	January	1	10	Saturday	

In [36]:
```
%%sql
ALTER TABLE games
ADD COLUMN date_key INTEGER,
ADD CONSTRAINT fk_date
    FOREIGN KEY (date_key)
    REFERENCES date (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[36]: []

In [37]:
```
%%sql
UPDATE games
SET date_key = date.key
FROM date
WHERE TO_CHAR(games.date, 'YYYY-MM-DD HH24:00:00') = date.hour;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[37]: []

We are working on the team dimension table, inserting all unique teams from the dataset into the team dimension. Then we will add a home_team_key and away_team_key to the games fact table:

In [38]:
```
%%sql
SELECT * FROM team
LIMIT 10;
```

```
 * postgresql://student@/finalproject
0 rows affected.
```

Out[38]:

key	team_name

In [39]:
```
%%sql
INSERT INTO team (team_name)
SELECT DISTINCT team AS team_name
FROM games;
```

```
 * postgresql://student@/finalproject
48 rows affected.
```

Out[39]: []

In [40]:
```
%%sql
SELECT * FROM team
ORDER BY team_name
LIMIT 5;
```

```
 * postgresql://student@/finalproject
5 rows affected.
```

Out[40]:

key	team_name
35	Arsenal
30	Aston Villa
45	Barnsley
3	Birmingham
46	Blackburn

In [41]:
```
%%sql
ALTER TABLE games
ADD COLUMN team_key INTEGER,
ADD CONSTRAINT fk_team
    FOREIGN KEY (team_key)
    REFERENCES team (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[41]: []

In [42]:
```
%%sql
SELECT * FROM games LIMIT 1;
```

```
 * postgresql://student@/finalproject
1 rows affected.
```

Out[42]:

date	referee	fixture	game_id	season	side	team	match_event	match_
1998-12-26 00:00:00	None	Newcastle vs Leeds	26/12/1998Newcastle vs Leeds	1998 | 1999	HomeTeam	Newcastle	GF(Full Time)	

In [43]:

```
%%sql
UPDATE games
SET team_key = team.key
FROM team
WHERE games.team = team.team_name;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[43]: []

In [44]:

```
%%sql
SELECT * FROM games
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[44]:

date	referee	fixture	game_id	season	side	team	match_event	match_
1998-12-26 00:00:00	None	Newcastle vs Leeds	26/12/1998Newcastle vs Leeds	1998 | 1999	HomeTeam	Newcastle	GF(Full Time)	
1998-10-31 00:00:00	None	Newcastle vs West Ham	31/10/1998Newcastle vs West Ham	1998 | 1999	AwayTeam	West Ham	Shots Conceded	
1998-10-24 00:00:00	None	Charlton vs West Ham	24/10/1998Charlton vs West Ham	1998 | 1999	AwayTeam	West Ham	GF (2nd Half)	

We are working on the side dimension table, where we will use the unique key and insert this into the games fact table. This will reference the 'side' in the game table.

In [45]:

```
%%sql
SELECT * FROM side
LIMIT 10;
```

```
 * postgresql://student@/finalproject
0 rows affected.
```

Out[45]:

key	home_away

In [46]:

```
%%sql
INSERT INTO side (home_away)
SELECT DISTINCT side AS home_away
FROM games;
```

```
 * postgresql://student@/finalproject
2 rows affected.
```

Out[46]: []

In [47]:
```
%%sql
SELECT * FROM side
ORDER BY home_away;
```

```
 * postgresql://student@/finalproject
2 rows affected.
```

Out[47]:

key	home_away
2	AwayTeam
1	HomeTeam

In [48]:
```
%%sql
ALTER TABLE games
ADD COLUMN side_key INTEGER,
ADD CONSTRAINT fk_side
    FOREIGN KEY (side_key)
    REFERENCES side (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[48]: []

In [49]:
```
%%sql
SELECT * FROM games LIMIT 1;
```

```
 * postgresql://student@/finalproject
1 rows affected.
```

Out[49]:

date	referee	fixture	game_id	season	side	team	match_event	match_event_cc
1998-10-24 00:00:00	None	Charlton vs West Ham	24/10/1998Charlton vs West Ham	1998 | 1999	AwayTeam	West Ham	GF (2nd Half)	

In [50]:
```
%%sql
UPDATE games
SET side_key = side.key
FROM side
WHERE games.side = side.home_away;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[50]: []

In [51]:
```
%%sql
SELECT * FROM games
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[51]:

date	referee	fixture	game_id	season	side	team	match_event	match_ev
1998-10-24 00:00:00	None	Charlton vs West Ham	24/10/1998Charlton vs West Ham	1998 | 1999	AwayTeam	West Ham	GF (2nd Half)	
1998-10-17 00:00:00	None	Chelsea vs Charlton	17/10/1998Chelsea vs Charlton	1998 | 1999	AwayTeam	Charlton	Fouls Committed	
1998-11-14 00:00:00	None	Chelsea vs Wimbledon	14/11/1998Chelsea vs Wimbledon	1998 | 1999	HomeTeam	Chelsea	GA (Full Time)	

We are working on the match_events dimension table, where we will use the unique key and insert this into the games fact table. This will reference the 'match_event' in the game table.

In [52]:
```
%%sql
SELECT * FROM match_events
LIMIT 10;
```

```
 * postgresql://student@/finalproject
0 rows affected.
```

Out[52]:

key	match_event

In [53]:
```
%%sql
INSERT INTO match_events (match_event)
SELECT DISTINCT match_event AS match_event
FROM games;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[53]: []

In [54]:
```
%%sql
SELECT * FROM match_events
ORDER BY match_event;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[54]:

key	match_event
19	Corners Conceded
4	Corners Played
11	Fouls Committed
10	Fouls Suffered
18	GA (2nd half)
8	GA (Full Time)
13	GA (Half Time)
5	GF (2nd Half)
20	GF(Full Time)
7	GF(Half Time)
21	Points (2nd Half)
16	Points (Full Time)
17	Points (Half Time)
1	Red Card (advantage)
9	Red Card (disadvantage)
15	Shots Conceded
3	Shots Conceded on target
2	Shots on target
6	Total Shots
14	Yellow Card (advantage)
12	Yellow Card (disadvantage)

In [55]:
```
%%sql
ALTER TABLE games
ADD COLUMN match_event_key INTEGER,
ADD CONSTRAINT fk_match_event
    FOREIGN KEY (match_event_key)
    REFERENCES match_events (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[55]: []

In [56]:
```
%%sql
SELECT * FROM games LIMIT 1;
```

```
 * postgresql://student@/finalproject
1 rows affected.
```

Out[56]:

date	referee	fixture	game_id	season	side	team	match_event	match_eve
1998-11-14 00:00:00	None	Chelsea vs Wimbledon	14/11/1998Chelsea vs Wimbledon	1998 | 1999	HomeTeam	Chelsea	GA (Full Time)	

In [57]:
```
%%sql
UPDATE games
SET match_event_key = match_events.key
FROM match_events
WHERE games.match_event = match_events.match_event;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[57]: []

In [58]:
```
%%sql
SELECT * FROM games
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[58]:

date	referee	fixture	game_id	season	side	team	match_event	matcl
1998-11-14 00:00:00	None	Chelsea vs Wimbledon	14/11/1998Chelsea vs Wimbledon	1998 | 1999	HomeTeam	Chelsea	GA (Full Time)	
1998-09-26 00:00:00	None	Everton vs Blackburn	26/09/1998Everton vs Blackburn	1998 | 1999	AwayTeam	Blackburn	GF (2nd Half)	
1998-09-26 00:00:00	None	Tottenham vs Leeds	26/09/1998Tottenham vs Leeds	1998 | 1999	AwayTeam	Leeds	Total Shots	

We are working on the game dimension table, where we will use the unique key and insert this into the games fact table. This will reference the fixture and referee in the game table.

In [59]:
```
%%sql
SELECT * FROM game
LIMIT 3;
```

```
 * postgresql://student@/finalproject
0 rows affected.
```

Out[59]:

key	game_id	fixture_name	referee

In [60]:
```
%%sql
INSERT INTO game (game_id)
SELECT DISTINCT game_id AS game_id
FROM games;
```

```
 * postgresql://student@/finalproject
7980 rows affected.
```

Out[60]: []

In [61]:
```
%%sql
SELECT * FROM game
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[61]:

key	game_id	fixture_name	referee
1	26/10/2013Norwich vs Cardiff	None	None
2	22/02/2014Chelsea vs Everton	None	None
3	08/04/2017Stoke vs Liverpool	None	None

In [62]:
```
%%sql
UPDATE game
SET fixture_name = games.fixture
FROM games
WHERE game.game_id = games.game_id;
```

```
 * postgresql://student@/finalproject
7980 rows affected.
```

Out[62]: []

In [63]:
```
%%sql
SELECT * FROM game
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[63]:

key	game_id	fixture_name	referee
6523	26/09/1998Everton vs Blackburn	Everton vs Blackburn	None
5731	26/09/1998Tottenham vs Leeds	Tottenham vs Leeds	None
3469	06/02/1999Chelsea vs Southampton	Chelsea vs Southampton	None

In [64]:
```
%%sql
UPDATE game
SET referee = games.referee
FROM games
WHERE game.game_id = games.game_id;
```

```
 * postgresql://student@/finalproject
7980 rows affected.
```

Out[64]: []

In [65]:
```
%%sql
SELECT * FROM game
ORDER BY referee
LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[65]:

key	game_id	fixture_name	referee
6105	28/12/2003Wolves vs Leeds	Wolves vs Leeds	A D'Urso
2025	15/12/2002Fulham vs Birmingham	Fulham vs Birmingham	A D'Urso
4050	09/03/2004Middlesbrough vs Tottenham	Middlesbrough vs Tottenham	A D'Urso

In [66]:
```
%%sql
SELECT DISTINCT referee
FROM game
ORDER BY referee
LIMIT 5;
```

```
 * postgresql://student@/finalproject
5 rows affected.
```

Out[66]:

referee
A D'Urso
A Marriner
A Taylor
A Wiley
A. G. Wiley

In [67]:
```
%%sql
ALTER TABLE games
ADD COLUMN game_key INTEGER,
ADD CONSTRAINT fk_game_key
    FOREIGN KEY (game_key)
    REFERENCES game (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[67]: []

In [68]:
```
%%sql
SELECT * FROM games LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[68]:

date	referee	fixture	game_id	season	side	team	match_event	r
1998-09-26 00:00:00	None	Everton vs Blackburn	26/09/1998Everton vs Blackburn	1998 | 1999	AwayTeam	Blackburn	GF (2nd Half)	
1998-09-26 00:00:00	None	Tottenham vs Leeds	26/09/1998Tottenham vs Leeds	1998 | 1999	AwayTeam	Leeds	Total Shots	
1999-02-06 00:00:00	None	Chelsea vs Southampton	06/02/1999Chelsea vs Southampton	1998 | 1999	AwayTeam	Southampton	GF(Full Time)	

In [69]:
```
%%sql
UPDATE games
SET game_key = game.key
FROM game
WHERE games.game_id = game.game_id;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[69]: []

In [70]:
```
%%sql
SELECT * FROM games LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[70]:

date	referee	fixture	game_id	season	side	team	match_event
1999-02-06 00:00:00	None	Chelsea vs Southampton	06/02/1999Chelsea vs Southampton	1998 | 1999	AwayTeam	Southampton	GF(Full Time)
1999-02-06 00:00:00	None	Tottenham vs Coventry	06/02/1999Tottenham vs Coventry	1998 | 1999	HomeTeam	Tottenham	Shots Conceded on target
1999-01-16 00:00:00	None	Nott'm Forest vs Arsenal	16/01/1999Nott'm Forest vs Arsenal	1998 | 1999	AwayTeam	Arsenal	GA (Full Time)

We are working on the season dimension table, inserting all unique seasons from the dataset into the season dimension. Then we will add a season_key games fact table:

In [71]:
```
%%sql
INSERT INTO season (season_name)
SELECT DISTINCT season AS season_name
FROM games;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[71]: []

In [72]:
```
%%sql
SELECT * FROM season
ORDER BY season_name;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[72]:

key	season_name
3	1997 | 1998
2	1998 | 1999
10	1999 | 2000
6	2000 | 2001
19	2001 | 2002
9	2002 | 2003
15	2003 | 2004
11	2004 | 2005
4	2005 | 2006
5	2006 | 2007
1	2007 | 2008
17	2008 | 2009
13	2009 | 2010
21	2010 | 2011
16	2011 | 2012
7	2012 | 2013
12	2013 | 2014
8	2014 | 2015
18	2015 | 2016
14	2016 | 2017
20	2017 | 2018

In [73]:
```
%%sql
ALTER TABLE games
ADD COLUMN season_key INTEGER,
ADD CONSTRAINT fk_season
    FOREIGN KEY (season_key)
    REFERENCES season (key);
```

```
 * postgresql://student@/finalproject
Done.
```

Out[73]: []

In [74]:
```
%%sql
SELECT * FROM games LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[74]:

date	referee	fixture	game_id	season	side	team	match_event	matc
1999-02-06 00:00:00	None	Tottenham vs Coventry	06/02/1999Tottenham vs Coventry	1998 | 1999	HomeTeam	Tottenham	Shots Conceded on target	
1999-01-16 00:00:00	None	Nott'm Forest vs Arsenal	16/01/1999Nott'm Forest vs Arsenal	1998 | 1999	AwayTeam	Arsenal	GA (Full Time)	
1999-01-16 00:00:00	None	Tottenham vs Wimbledon	16/01/1999Tottenham vs Wimbledon	1998 | 1999	HomeTeam	Tottenham	Shots Conceded	

In [75]:
```
%%sql
UPDATE games
SET season_key = season.key
FROM season
WHERE games.season = season.season_name;
```

```
 * postgresql://student@/finalproject
335160 rows affected.
```

Out[75]: []

In [76]:
```
%%sql
SELECT * FROM games LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[76]:

date	referee	fixture	game_id	season	side	team	match_event	matc
1999-02-06 00:00:00	None	Tottenham vs Coventry	06/02/1999Tottenham vs Coventry	1998 | 1999	HomeTeam	Tottenham	Shots Conceded on target	
1999-01-16 00:00:00	None	Nott'm Forest vs Arsenal	16/01/1999Nott'm Forest vs Arsenal	1998 | 1999	AwayTeam	Arsenal	GA (Full Time)	
1999-01-16 00:00:00	None	Tottenham vs Wimbledon	16/01/1999Tottenham vs Wimbledon	1998 | 1999	HomeTeam	Tottenham	Shots Conceded	

In [77]:
```
%%sql
SELECT * FROM games
ORDER BY game_key
LIMIT 1;
```

```
 * postgresql://student@/finalproject
1 rows affected.
```

Out[77]:

date	referee	fixture	game_id	season	side	team	match_event	match_event_c
2013-10-26 00:00:00	M Jones	Norwich vs Cardiff	26/10/2013Norwich vs Cardiff	2013 | 2014	AwayTeam	Cardiff	GA (Full Time)	

In [78]:
```
%%sql
ALTER TABLE games
DROP COLUMN referee,
DROP COLUMN fixture,
DROP COLUMN game_id,
DROP COLUMN season,
DROP COLUMN side,
DROP COLUMN team,
DROP COLUMN match_event,
DROP COLUMN date;
```

```
 * postgresql://student@/finalproject
Done.
```

Out[78]: []

In [79]:
```
%%sql
SELECT * FROM games LIMIT 3;
```

```
 * postgresql://student@/finalproject
3 rows affected.
```

Out[79]:

match_event_count	date_key	team_key	side_key	match_event_key	game_key	season_key
0	1944	8	1	3	2565	2
0	88	35	2	8	4269	2
0	88	8	1	15	2868	2

Data Exploration - we are exploring some simple queries to explore the data through SQL:

In [80]:
```
%matplotlib inline
```

In [81]:
```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import pandas as pd
import pickle
```

We will create a query to analyze the 2017 | 2018 league table by displaying the season, team, and total points earned:

In [82]:
```
%%sql
SELECT
    s.season_name, t.team_name,
    sum(match_event_count) sum_points
FROM
    Season S, Team T, Games G, Match_Events M
WHERE
    s.season_name = '2017 | 2018' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key and
    m.match_event = 'Points (Full Time)'
```

```
GROUP BY
    s.season_name, t.team_name
ORDER BY s.season_name, sum_points DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[82]:

season_name	team_name	sum_points
2017 | 2018	Man City	100
2017 | 2018	Man United	81
2017 | 2018	Tottenham	77
2017 | 2018	Liverpool	75
2017 | 2018	Chelsea	70
2017 | 2018	Arsenal	63
2017 | 2018	Burnley	54
2017 | 2018	Everton	49
2017 | 2018	Leicester	47
2017 | 2018	Newcastle	44
2017 | 2018	Crystal Palace	44
2017 | 2018	Bournemouth	44
2017 | 2018	West Ham	42
2017 | 2018	Watford	41
2017 | 2018	Brighton	40
2017 | 2018	Huddersfield	37
2017 | 2018	Southampton	36
2017 | 2018	Swansea	33
2017 | 2018	Stoke	33
2017 | 2018	West Brom	31

We will analyze the total number of red cards by season across the 20 seasons:

In [83]:

```
%%sql
SELECT
    s.season_name,
    sum(match_event_count) total_red_cards
FROM
    Season S, Match_Events M, Games G
WHERE
    s.key = g.season_key and
    m.key = g.match_event_key and
    m.match_event = 'Red Card (disadvantage)'
GROUP BY
    s.season_name
ORDER BY s.season_name;
```

```
 * postgresql://student@/finalproject
21 rows affected.
```

Out[83]:

season_name	total_red_cards
1997 | 1998	0
1998 | 1999	0
1999 | 2000	0
2000 | 2001	63
2001 | 2002	72
2002 | 2003	75
2003 | 2004	58
2004 | 2005	59
2005 | 2006	76
2006 | 2007	53
2007 | 2008	61
2008 | 2009	63
2009 | 2010	68
2010 | 2011	63
2011 | 2012	64
2012 | 2013	52
2013 | 2014	53
2014 | 2015	71
2015 | 2016	59
2016 | 2017	41
2017 | 2018	39

Here we can see there was most likely no data for red cards collected in the first three seasons even though in real life there were red cards given in these years. This is interesting to note. If we chose to analyze red cards in our business questions, we may select a shorter number of seasons than all available so the data is not skewed.

We will analyze the total points, total shots on target, and total shots conceded teams had in the most recent season available, '2017 | 2018':

In [84]:

```
%%sql
SELECT A.team_name, sum_points, sum_shots_on_target, sum_shots_conceded_on_target
FROM
(
SELECT t.team_name,
    sum(match_event_count) sum_shots_on_target,
    count(1) as "count"
FROM Team T, Season S, Match_Events M, Games G
```

```
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as A,

(SELECT t.team_name,
    sum(match_event_count) sum_points,
    count(1) as "points"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as B,

(SELECT t.team_name,
    sum(match_event_count) sum_shots_conceded_on_target,
    count(1) as "shots conceded on target"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots Conceded on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as C
WHERE a.team_name=b.team_name and a.team_name=c.team_name
ORDER BY sum_points DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[84]:

team_name	sum_points	sum_shots_on_target	sum_shots_conceded_on_target
Man City	100	261	87
Man United	81	181	146
Tottenham	77	217	126
Liverpool	75	233	103
Chelsea	70	220	120
Arsenal	63	233	148
Burnley	54	127	164
Everton	49	122	183
Leicester	47	148	168
Newcastle	44	150	157
Crystal Palace	44	151	176
Bournemouth	44	157	176
West Ham	42	133	190
Watford	41	132	160
Brighton	40	118	181
Huddersfield	37	111	165
Southampton	36	145	170
Swansea	33	104	191
Stoke	33	131	220
West Brom	31	114	157

Here we see there may be some correlation between points and attacking/defensive match events. This is something we can explore further

We are exploring a similar query with more columns, now with corners played, and condceded, and yellows cards (advantage) and (disadvantage):

In [85]:

```
%%sql
SELECT A.team_name, sum_points, sum_shots_on_target, sum_shots_conceded_on_target, sum
FROM
(
SELECT t.team_name,
    sum(match_event_count) sum_shots_on_target,
    count(1) as "count"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
```

```
GROUP BY t.team_name, g.match_event_key) as A,

(SELECT t.team_name,
    sum(match_event_count) sum_points,
    count(1) as "points"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as B,

(SELECT t.team_name,
    sum(match_event_count) sum_shots_conceded_on_target,
    count(1) as "shots conceded on target"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots Conceded on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as C,

(SELECT t.team_name,
    sum(match_event_count) sum_corners_played,
    count(1) as "Corners played"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Corners Played' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as D,

(SELECT t.team_name,
    sum(match_event_count) sum_corners_conceded,
    count(1) as "Corners Conceded"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Corners Conceded' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as E,

(SELECT t.team_name,
    sum(match_event_count) sum_yellow_cards_advantage,
    count(1) as "Yellow Card (advantage)"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Yellow Card (advantage)' and
```

```
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as F,

(SELECT t.team_name,
    sum(match_event_count) sum_yellow_cards_disadvantage,
    count(1) as "Yellow Card (disadvantage)"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Yellow Card (disadvantage)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as H
WHERE a.team_name=b.team_name and a.team_name=c.team_name and a.team_name=d.team_name
ORDER BY sum_points DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[85]:

team_name	sum_points	sum_shots_on_target	sum_shots_conceded_on_target	sum_corners_played	su
Man City	100	261	87	284	
Man United	81	181	146	220	
Tottenham	77	217	126	246	
Liverpool	75	233	103	230	
Chelsea	70	220	120	230	
Arsenal	63	233	148	223	
Burnley	54	127	164	167	
Everton	49	122	183	150	
Leicester	47	148	168	203	
Newcastle	44	150	157	167	
Crystal Palace	44	151	176	209	
Bournemouth	44	157	176	217	
West Ham	42	133	190	161	
Watford	41	132	160	184	
Brighton	40	118	181	163	
Huddersfield	37	111	165	165	
Southampton	36	145	170	227	
Swansea	33	104	191	150	
Stoke	33	131	220	136	
West Brom	31	114	157	176	

Ask 3: Data Analysis and Visualization

Business Question 1:

Who is the best performing team (in terms of points) between the seasons 2007-2008 and 2017-2018? Do attacking attributes such as shots on target or defensive attributes shots conceded on target and goals allowed lead to higher points earned.

First, let's see who the best performing teams are in the past 10 years. We will explore individual seasons after.

In [86]:
```
%%sql
SELECT A.team_name, sum_points, season_2017_2018, season_2015_2016, season_2014_2015,
FROM
(
SELECT t.team_name,
    sum(match_event_count) sum_points,
    count(1) as "Total Points Over 10 Year Period"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Points (Full Time)' and
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    NOT s.season_name = '2001 | 2002' and
    NOT s.season_name = '2002 | 2003' and
    NOT s.season_name = '2003 | 2004' and
    NOT s.season_name = '2004 | 2005' and
    NOT s.season_name = '2005 | 2006' and
    NOT s.season_name = '2006 | 2007' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as A,

(
SELECT t.team_name,
    sum(match_event_count) season_2017_2018,
    count(1) as "Total Points for Season 2017-2018"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as B,

(
SELECT t.team_name,
    sum(match_event_count) season_2015_2016,
```

```
    count(1) as "Total Points for Season 2015-2016"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2015 | 2016' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as C,

(
SELECT t.team_name,
    sum(match_event_count) season_2014_2015,
    count(1) as "Total Points for Season 2014-2015"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2014 | 2015' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as D,

(
SELECT t.team_name,
    sum(match_event_count) season_2011_2012,
    count(1) as "Total Points for Season 2014-2015"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2011 | 2012' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as E
WHERE a.team_name=b.team_name and a.team_name=c.team_name and a.team_name=d.team_name
ORDER BY sum_points DESC;
```

```
 * postgresql://student@/finalproject
11 rows affected.
```

Out[86]:

team_name	sum_points	season_2017_2018	season_2015_2016	season_2014_2015	season_2011_2012
Man United	870	81	66	70	89
Chelsea	846	70	50	87	64
Man City	819	100	66	79	89
Arsenal	804	63	71	75	70
Liverpool	753	75	60	62	52
Tottenham	736	77	70	64	69
Everton	638	49	47	47	56
Stoke	457	33	51	54	45
Newcastle	398	44	37	39	65
West Brom	374	31	43	44	47
Swansea	312	33	47	56	47

The best performing team in the past 10 years since 2018 is Man United, as they have accumulated 870 points, followed by Chelsea then Man City.

Let's test our hypothesis in the seasons 2017/2018, 2015/2016, and 2011/2012 to assess whether there is a correlation with higher points and certain attributes.

In [87]:

```
%%sql
SELECT t.team_name,
    sum(match_event_count) season_2017_2018
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY season_2017_2018 DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

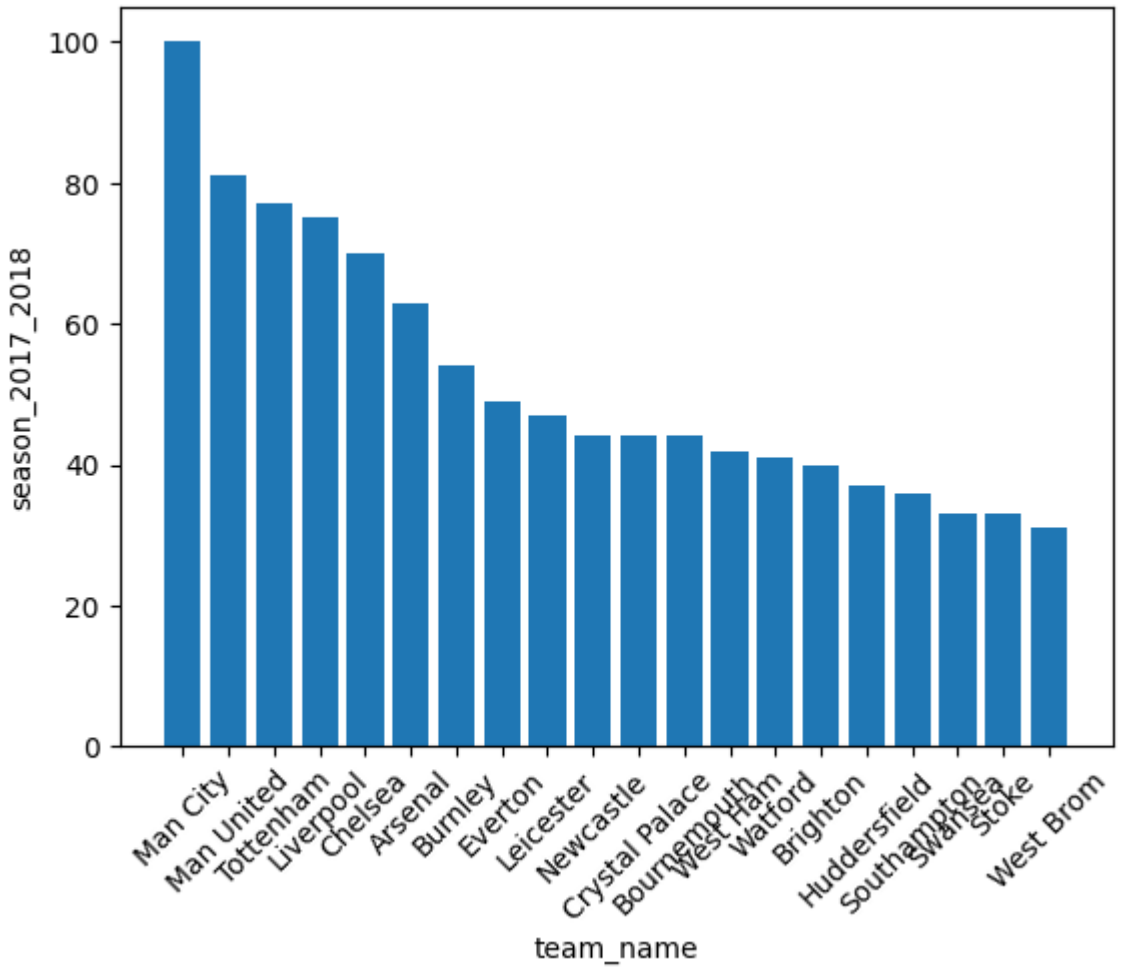
Out[87]:

team_name	season_2017_2018
Man City	100
Man United	81
Tottenham	77
Liverpool	75
Chelsea	70
Arsenal	63
Burnley	54
Everton	49
Leicester	47
Newcastle	44
Crystal Palace	44
Bournemouth	44
West Ham	42
Watford	41
Brighton	40
Huddersfield	37
Southampton	36
Swansea	33
Stoke	33
West Brom	31

In [88]:
```
%matplotlib inline
```

In [89]:
```
_.bar()
```

Out[89]:
```
<BarContainer object of 20 artists>
```



Lets explore shots on target for the same season:

In [90]:
```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_on_target DESC;
```

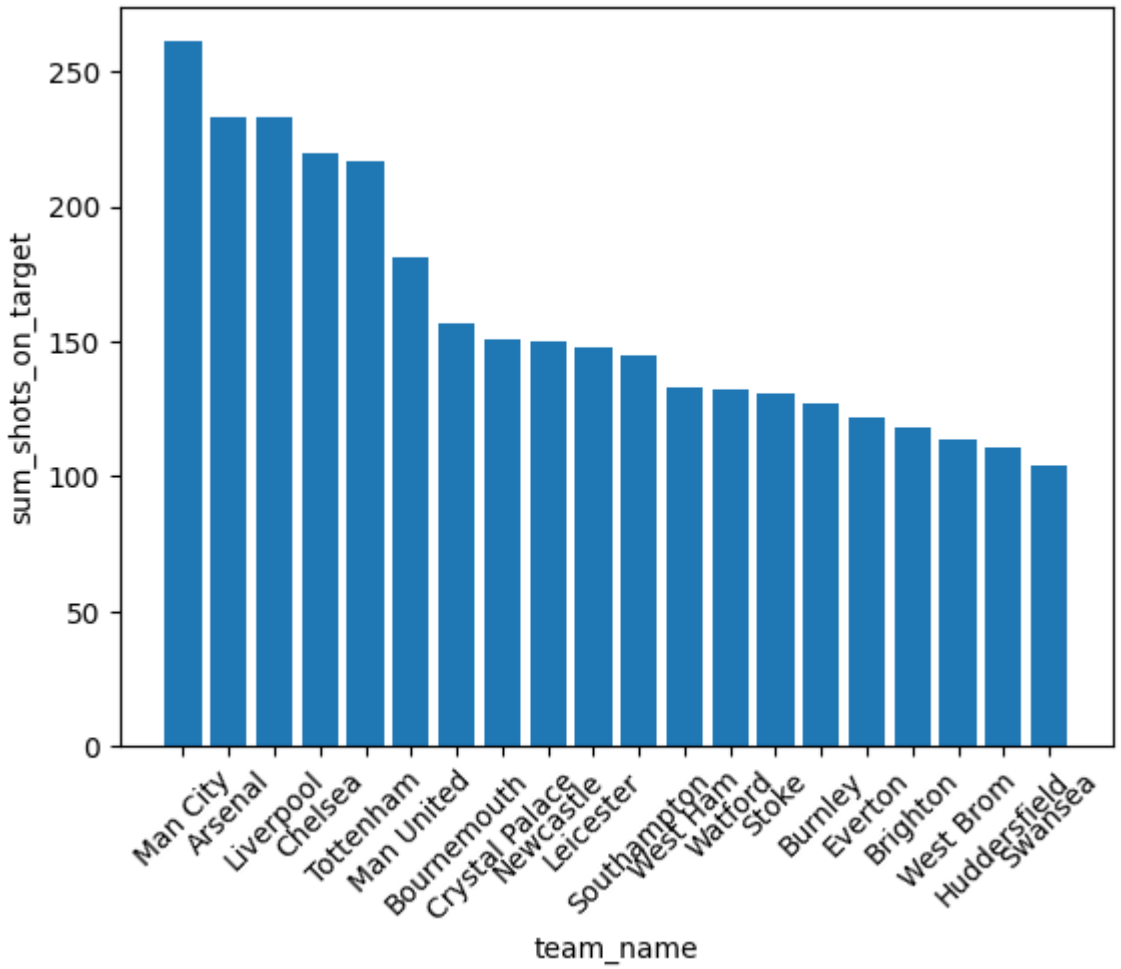
```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[90]:

team_name	sum_shots_on_target
Man City	261
Arsenal	233
Liverpool	233
Chelsea	220
Tottenham	217
Man United	181
Bournemouth	157
Crystal Palace	151
Newcastle	150
Leicester	148
Southampton	145
West Ham	133
Watford	132
Stoke	131
Burnley	127
Everton	122
Brighton	118
West Brom	114
Huddersfield	111
Swansea	104

In [91]:
```
_.bar()
```

Out[91]:
```
<BarContainer object of 20 artists>
```



We see an identical trend of the bars in the graph above. Clearly, there is a correlation as Man City have had more shots on targets and also finished with the highest points.

But we can also see that Man United rank 6th in terms of 'shots on target' but were the second highest team with points. So are there other attributes that resulted to this? Let's look at 'Shots Conceded On Target'.

In [92]:
```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_conceded_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Shots Conceded on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_conceded_on_target;
```

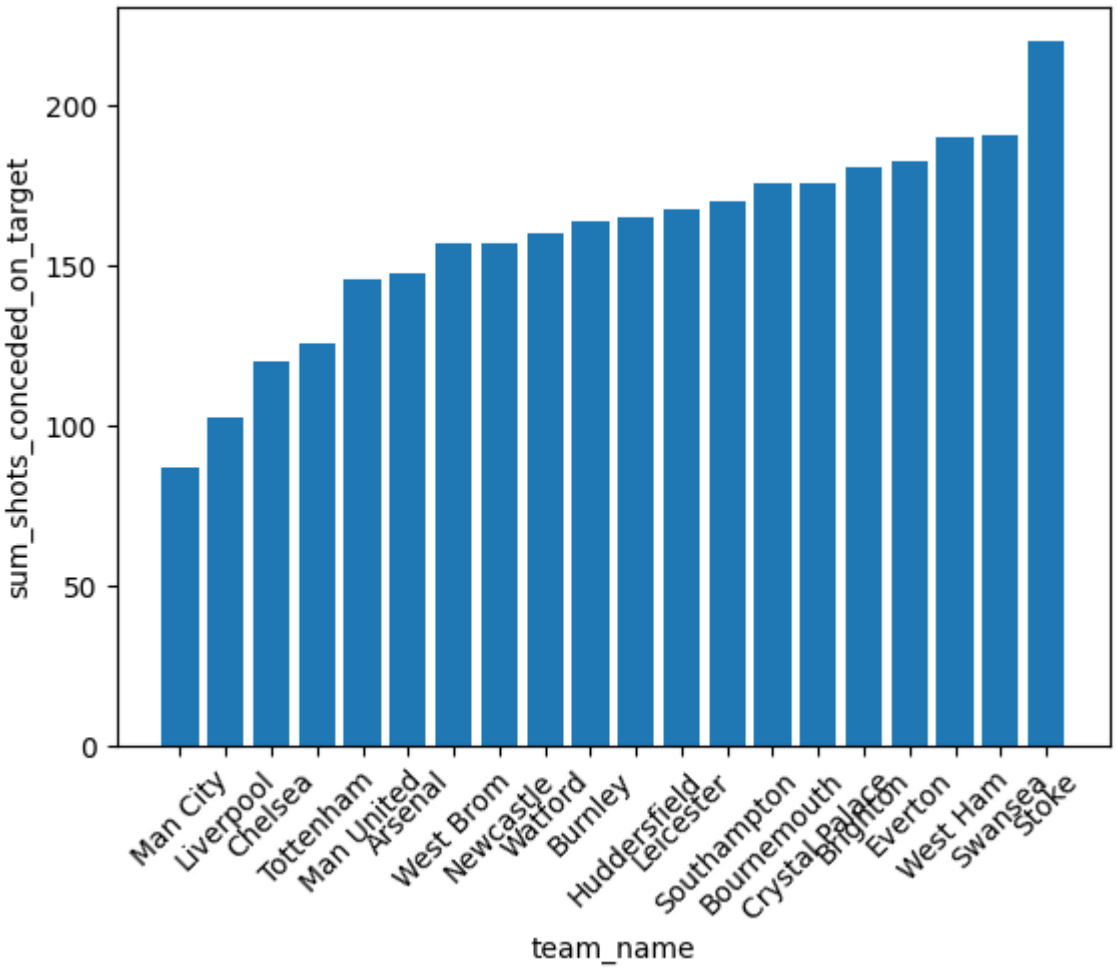
```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[92]:

team_name	sum_shots_conceded_on_target
Man City	87
Liverpool	103
Chelsea	120
Tottenham	126
Man United	146
Arsenal	148
West Brom	157
Newcastle	157
Watford	160
Burnley	164
Huddersfield	165
Leicester	168
Southampton	170
Bournemouth	176
Crystal Palace	176
Brighton	181
Everton	183
West Ham	190
Swansea	191
Stoke	220

In [93]:
```
_.bar()
```

Out[93]:
```
<BarContainer object of 20 artists>
```



We can see an inverse trend here, since conceding more shots implies poorer defending. We can see that has led to few points as Brighton and Everton finished amongst the lowest in total points and highest in shots condeded on target.

Let's take a look at goals allowed next.

In [94]:
```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_goals_allowed
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'GA (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_goals_allowed;
```

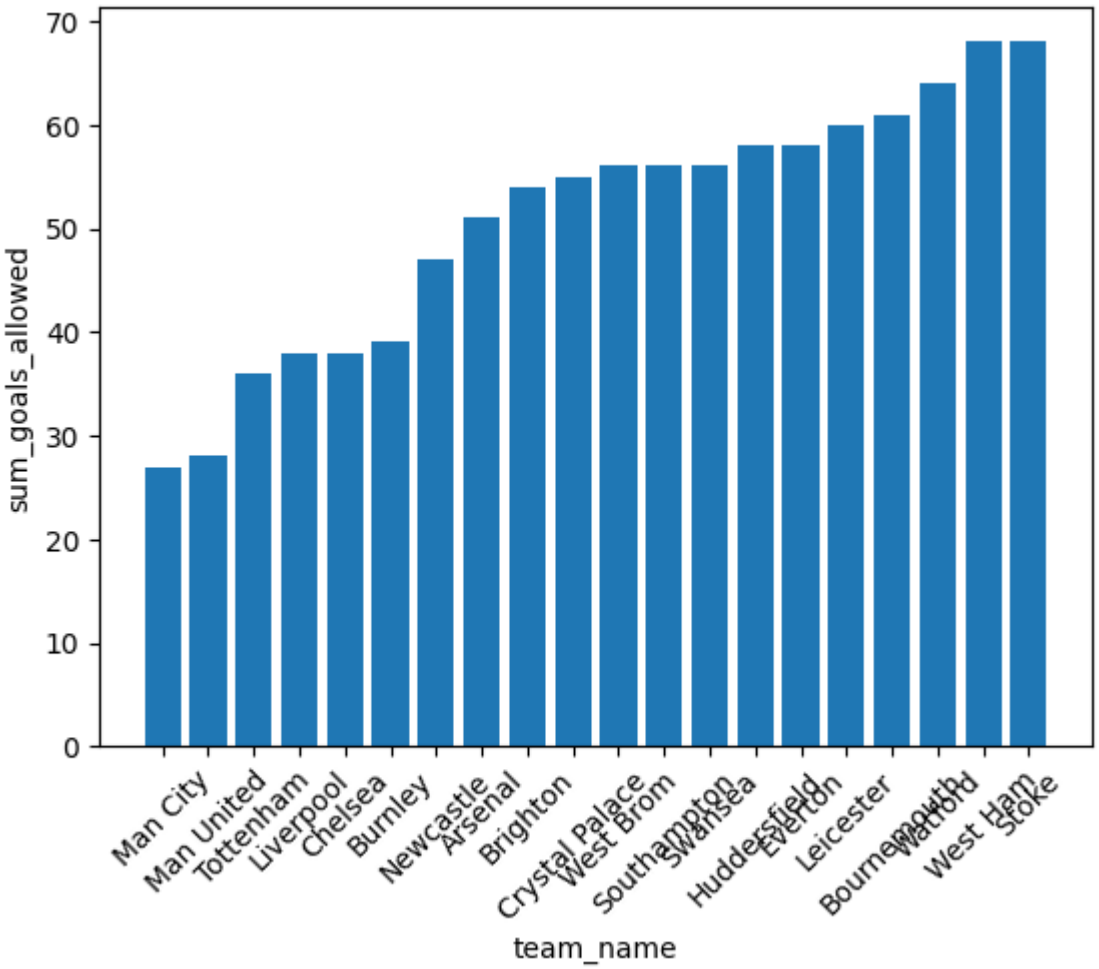
```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[94]:

team_name	sum_goals_allowed
Man City	27
Man United	28
Tottenham	36
Liverpool	38
Chelsea	38
Burnley	39
Newcastle	47
Arsenal	51
Brighton	54
Crystal Palace	55
West Brom	56
Southampton	56
Swansea	56
Huddersfield	58
Everton	58
Leicester	60
Bournemouth	61
Watford	64
West Ham	68
Stoke	68

In [95]:
```
_.bar()
```

Out[95]:
```
<BarContainer object of 20 artists>
```



After running multiple analytical queries on the 2017-2018 season, we now know that while attacking is important so that the team gets higher points, but defense is the decisive factor. The better teams to defend, the better their chances of finishing in the top. Let's test our analysis in other seasons!

2014-2015 Season

In [96]:
```
%%sql
SELECT t.team_name,
    sum(match_event_count) season_2014_2015
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2014 | 2015' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY season_2014_2015 DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

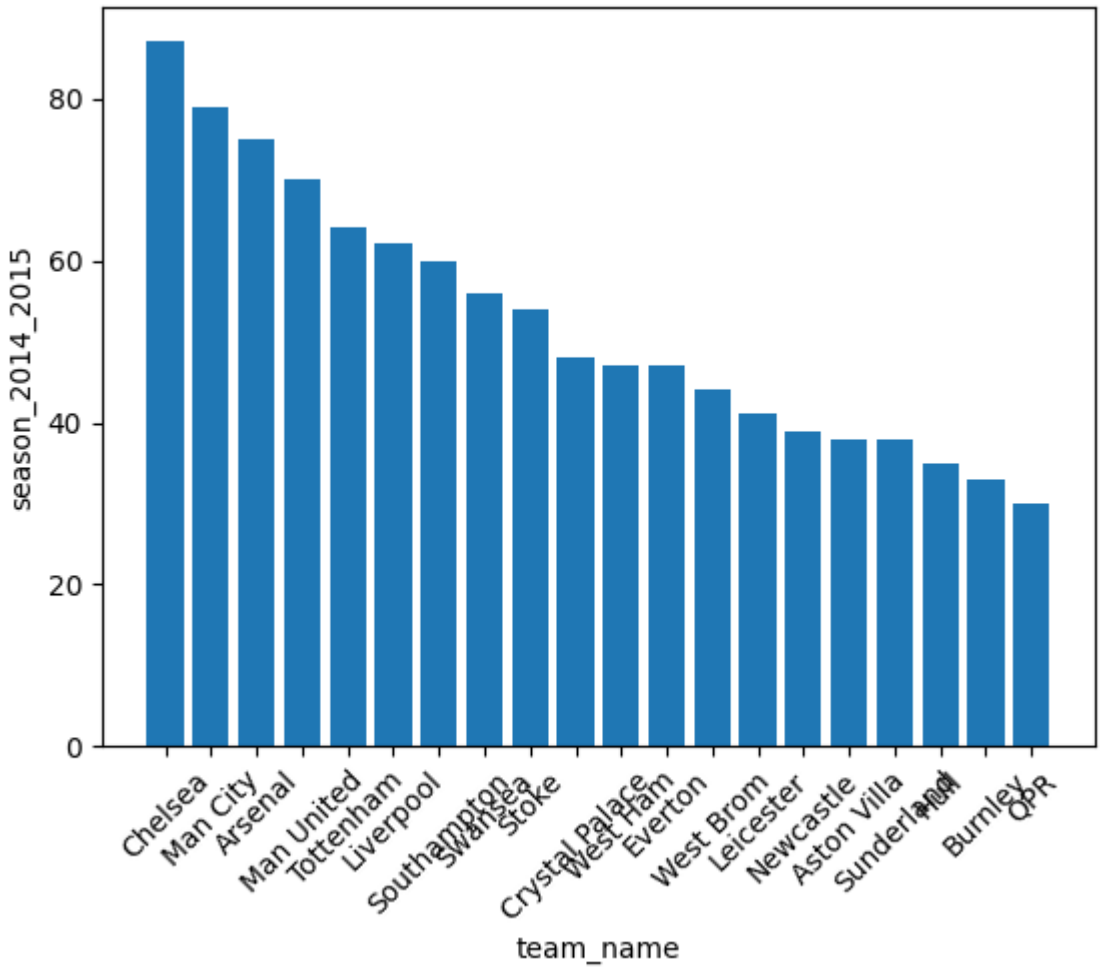
Out[96]:

team_name	season_2014_2015
Chelsea	87
Man City	79
Arsenal	75
Man United	70
Tottenham	64
Liverpool	62
Southampton	60
Swansea	56
Stoke	54
Crystal Palace	48
West Ham	47
Everton	47
West Brom	44
Leicester	41
Newcastle	39
Aston Villa	38
Sunderland	38
Hull	35
Burnley	33
QPR	30

In [97]:
```
%matplotlib inline
```

In [98]:
```
_.bar()
```

Out[98]:
```
<BarContainer object of 20 artists>
```



Lets explore shots on target for the same season:

In [99]:
```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2014 | 2015' and
    m.match_event = 'Shots on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_on_target DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[99]:

team_name	sum_shots_on_target
Arsenal	228
Man City	228
Chelsea	210
Liverpool	194
Man United	179
Tottenham	172
Southampton	170
Everton	166
West Ham	152
QPR	151
Swansea	147
Newcastle	143
Crystal Palace	139
Leicester	137
Stoke	136
Hull	131
West Brom	130
Sunderland	129
Aston Villa	127
Burnley	125

Chelsea had few shots on target, 210, than Arsenal and Man City, who both had 228 shots on target. Yet Chelsea finished top of the league with 87 points and Arsenal and Man City finished 3rd and 2nd, respectively. Lets look at defensive statistics like shots conceded on target.

In [100…

```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_conceded_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2014 | 2015' and
    m.match_event = 'Shots Conceded on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_conceded_on_target;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[100]:

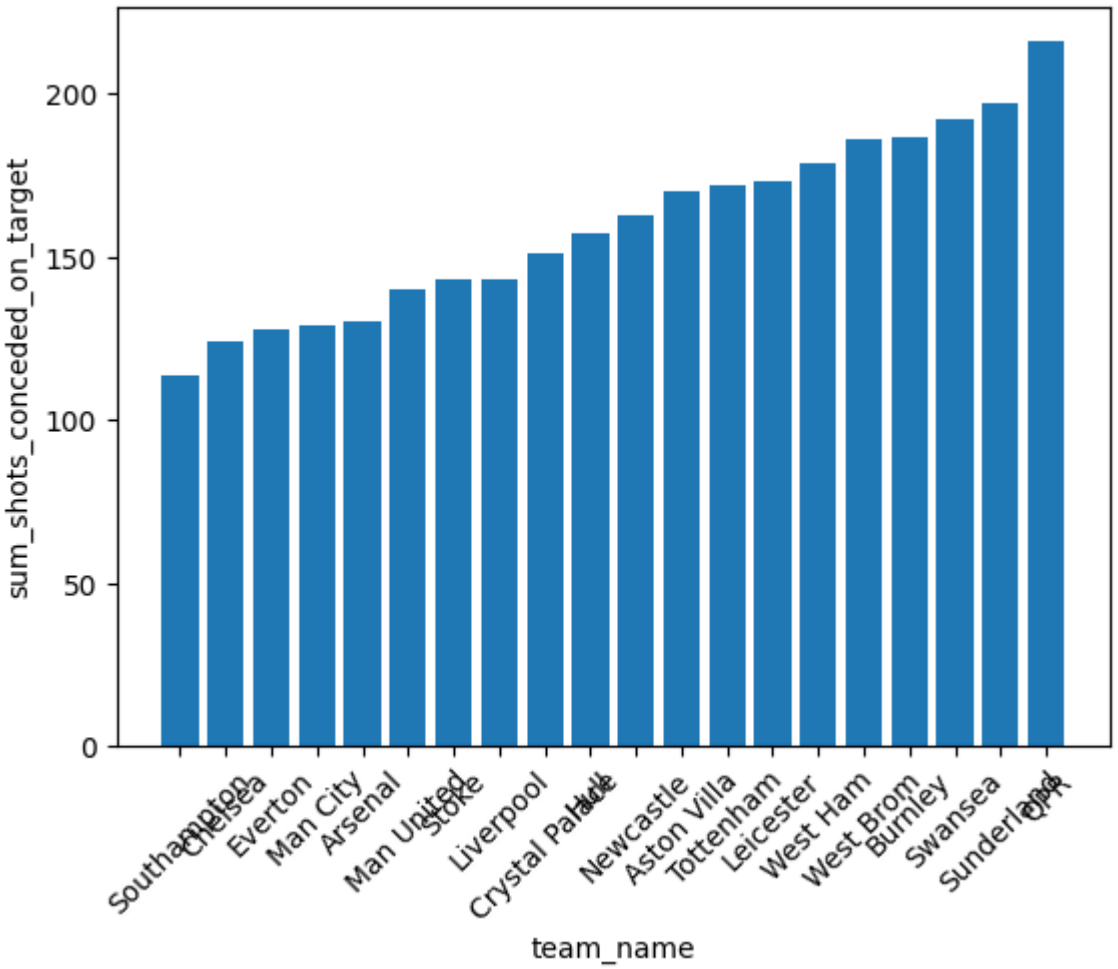
team_name	sum_shots_conceded_on_target
Southampton	114
Chelsea	124
Everton	128
Man City	129
Arsenal	130
Man United	140
Stoke	143
Liverpool	143
Crystal Palace	151
Hull	157
Newcastle	163
Aston Villa	170
Tottenham	172
Leicester	173
West Ham	179
West Brom	186
Burnley	187
Swansea	192
Sunderland	197
QPR	216

In [101…

```
_.bar()
```

Out[101]:

```
<BarContainer object of 20 artists>
```



We can see an inverse trend here, since conceding more shots implies poorer defending. We can see that has led to few points as Brighton and Everton finished amongst the lowest in total points and highest in shots condeded on target.

Let's take a look at goals allowed next.

In [102…

```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_goals_allowed
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2014 | 2015' and
    m.match_event = 'GA (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_goals_allowed;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[102]:

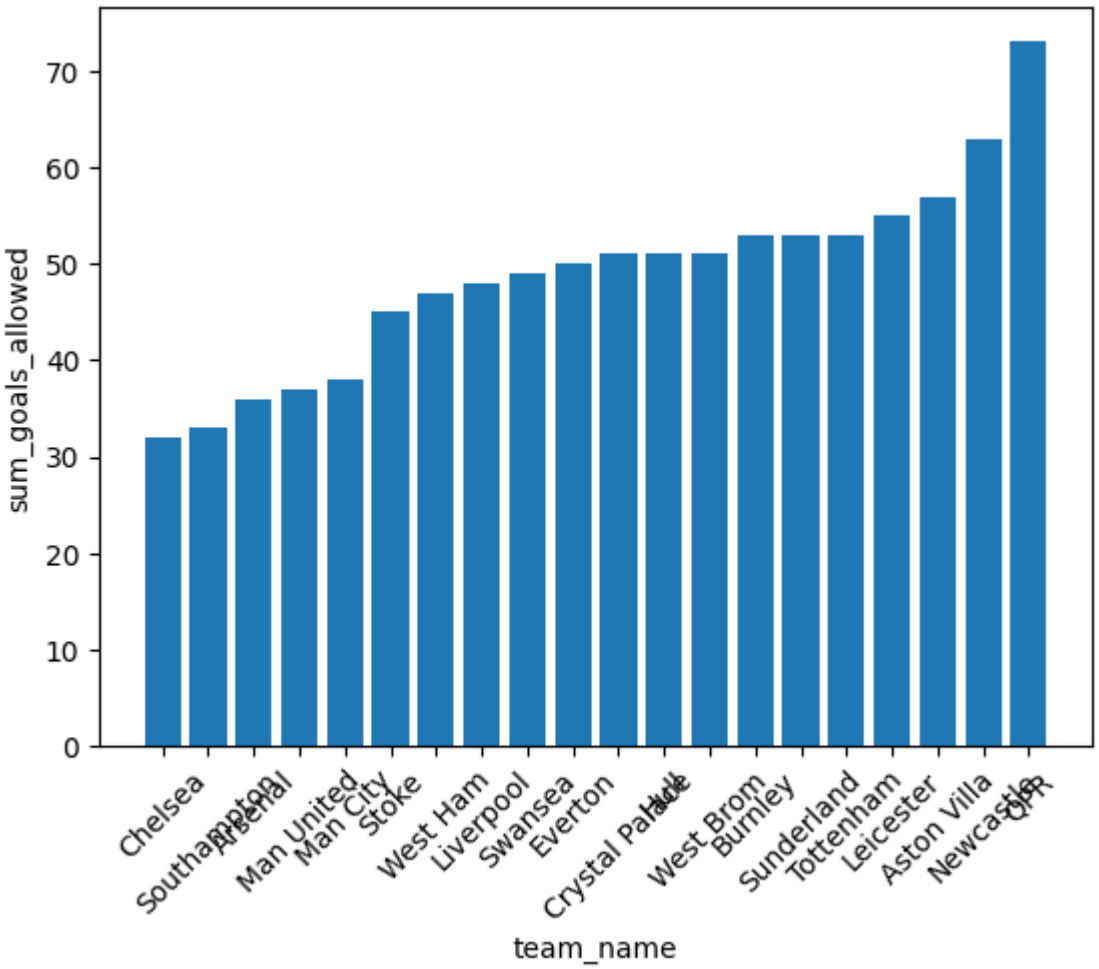
team_name	sum_goals_allowed
Chelsea	32
Southampton	33
Arsenal	36
Man United	37
Man City	38
Stoke	45
West Ham	47
Liverpool	48
Swansea	49
Everton	50
Crystal Palace	51
Hull	51
West Brom	51
Burnley	53
Sunderland	53
Tottenham	53
Leicester	55
Aston Villa	57
Newcastle	63
QPR	73

In [103…

```
_.bar()
```

Out[103]:

```
<BarContainer object of 20 artists>
```



It looks like our preliminary analysis is holding up! The better you defend, the higher your chances to finish in the top.

2011-2012 Season

In [104…

```
%%sql
SELECT t.team_name,
    sum(match_event_count) season_2011_2012
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2011 | 2012' and
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY season_2011_2012 DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[104]:

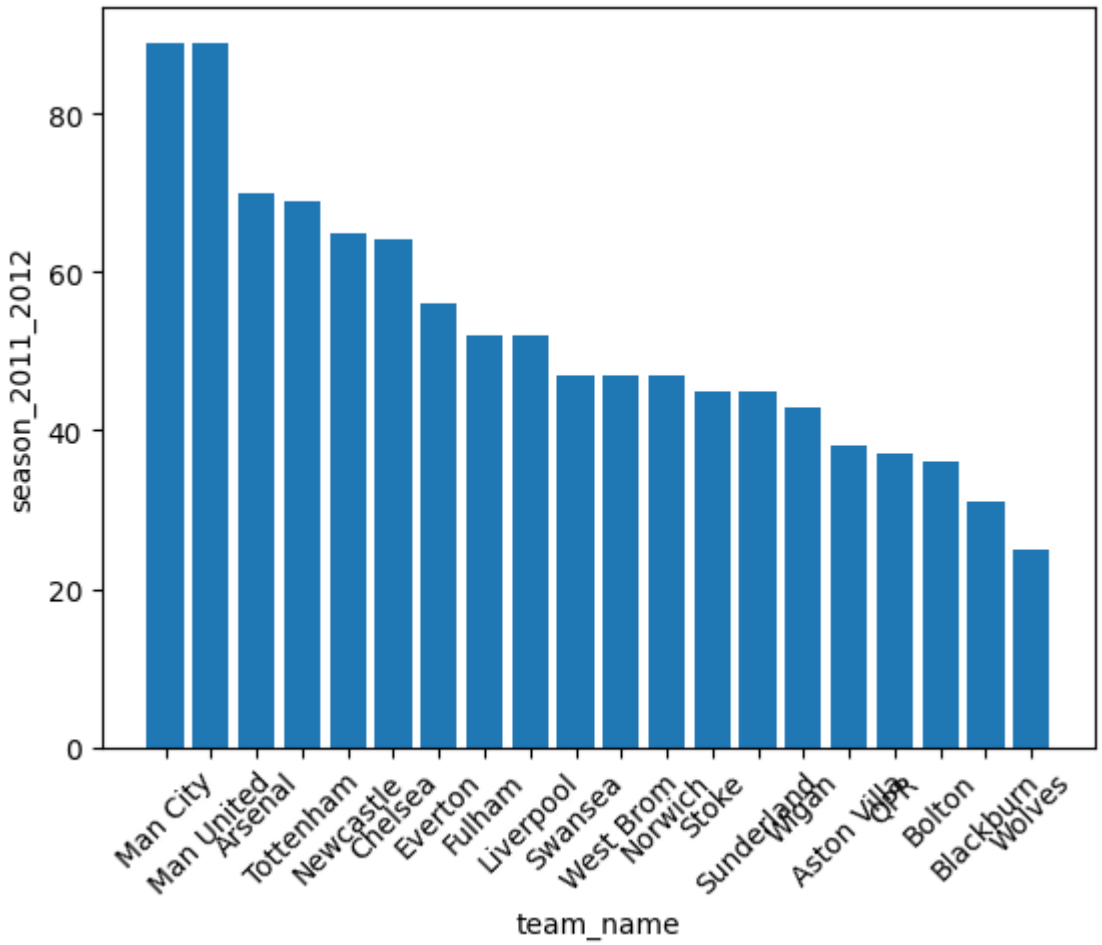
team_name	season_2011_2012
Man City	89
Man United	89
Arsenal	70
Tottenham	69
Newcastle	65
Chelsea	64
Everton	56
Fulham	52
Liverpool	52
Swansea	47
West Brom	47
Norwich	47
Stoke	45
Sunderland	45
Wigan	43
Aston Villa	38
QPR	37
Bolton	36
Blackburn	31
Wolves	25

In [105…

```
_.bar()
```

Out[105]:

```
<BarContainer object of 20 artists>
```



Lets explore shots on target for the same season:

In [106…

```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2011 | 2012' and
    m.match_event = 'Shots on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_on_target DESC;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[106]:

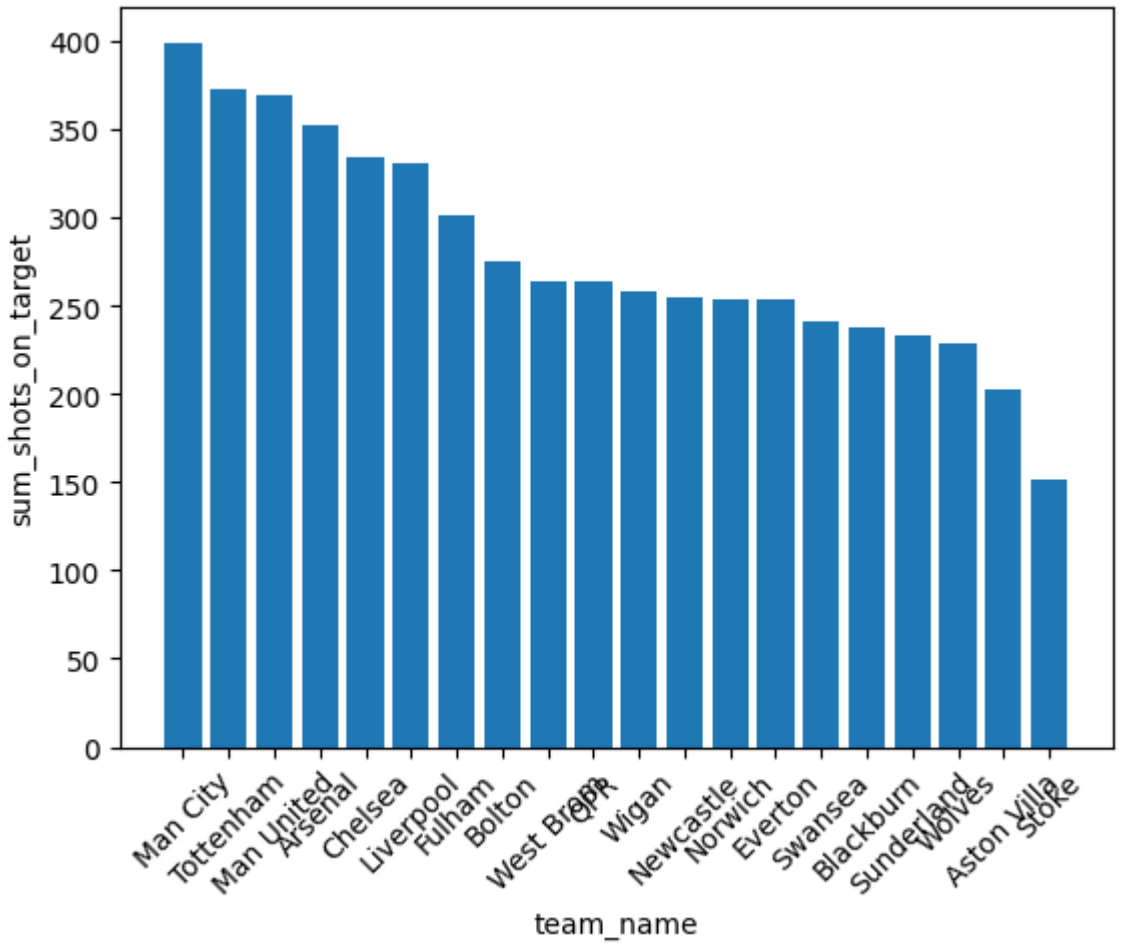
team_name	sum_shots_on_target
Man City	399
Tottenham	373
Man United	369
Arsenal	352
Chelsea	334
Liverpool	330
Fulham	301
Bolton	275
West Brom	264
QPR	264
Wigan	258
Newcastle	255
Norwich	254
Everton	253
Swansea	241
Blackburn	238
Sunderland	233
Wolves	229
Aston Villa	203
Stoke	151

In [107…

```
_.bar()
```

Out[107]:

```
<BarContainer object of 20 artists>
```



In [108…

```
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_shots_conceded_on_target
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2011 | 2012' and
    m.match_event = 'Shots Conceded on target' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_shots_conceded_on_target;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[108]:

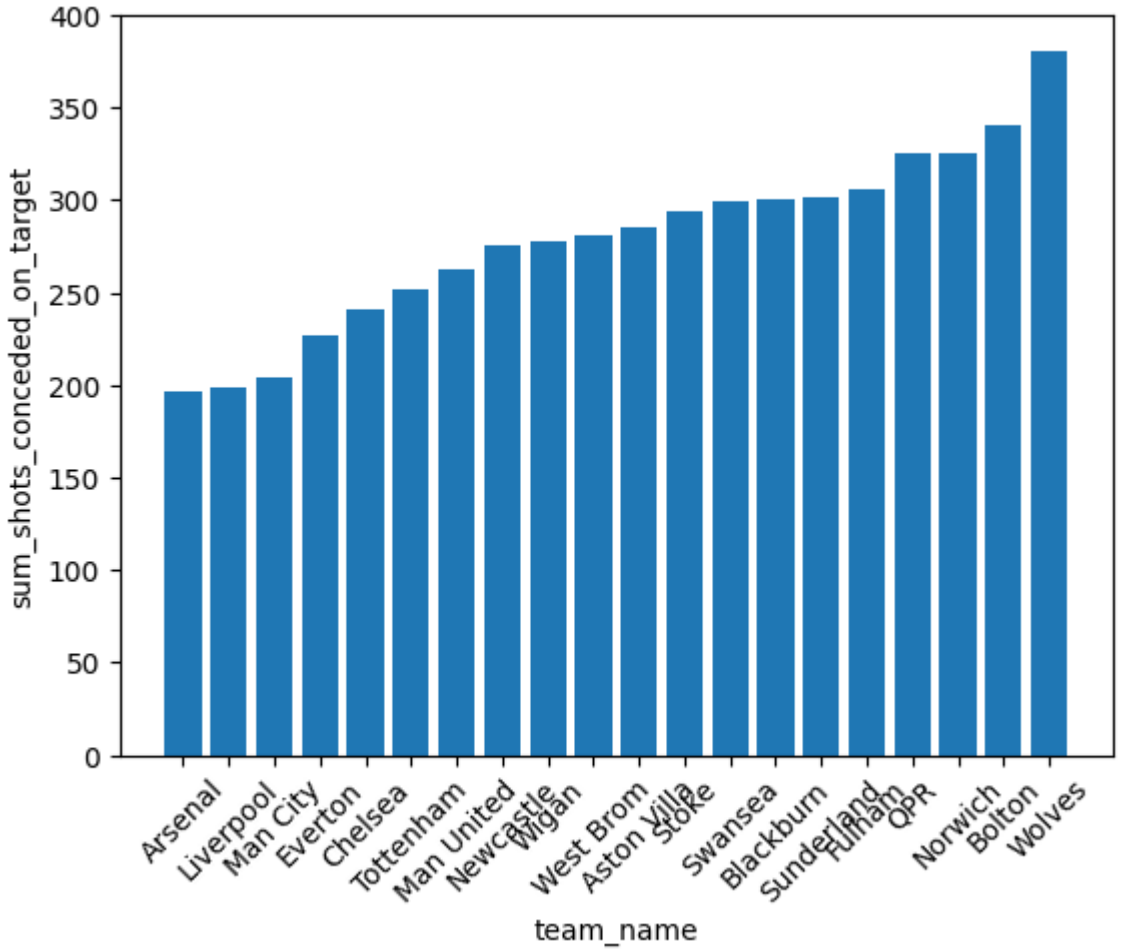
team_name	sum_shots_conceded_on_target
Arsenal	197
Liverpool	199
Man City	204
Everton	227
Chelsea	241
Tottenham	252
Man United	263
Newcastle	276
Wigan	278
West Brom	281
Aston Villa	285
Stoke	294
Swansea	299
Blackburn	300
Sunderland	302
Fulham	306
QPR	325
Norwich	325
Bolton	341
Wolves	381

In [109…

```
_.bar()
```

Out[109]:

```
<BarContainer object of 20 artists>
```



In [110…

```sql
%%sql
SELECT t.team_name,
    sum(match_event_count) sum_goals_allowed
FROM Team T, Season S, Match_Events M, Games G
WHERE
    s.season_name = '2011 | 2012' and
    m.match_event = 'GA (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key
ORDER BY sum_goals_allowed;
```

```
 * postgresql://student@/finalproject
20 rows affected.
```

Out[110]:

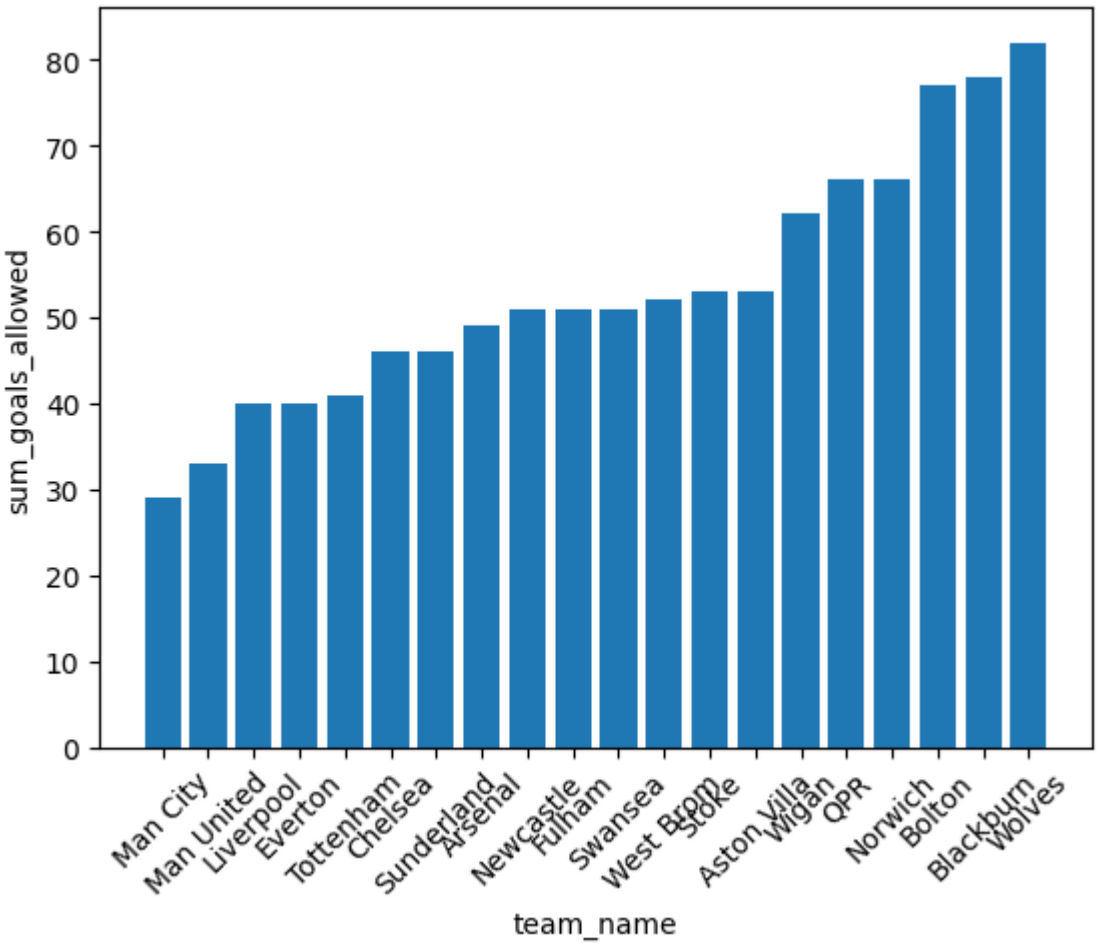
team_name	sum_goals_allowed
Man City	29
Man United	33
Liverpool	40
Everton	40
Tottenham	41
Chelsea	46
Sunderland	46
Arsenal	49
Newcastle	51
Fulham	51
Swansea	51
West Brom	52
Stoke	53
Aston Villa	53
Wigan	62
QPR	66
Norwich	66
Bolton	77
Blackburn	78
Wolves	82

In [111…

```
_.bar()
```

Out[111]:

```
<BarContainer object of 20 artists>
```



Finally, after conducting the analysis on the 2011-2021 season, the data shows that defense is the decisive factor in finishing the season with the high points. However, while the data also shows that defense is the most crucial factor, but to finish in the top, you have to also perform highly on the attack.

Business Question 2:

To help coaches and players be considerate of strict and softer referees, which can change the tactics of play for each game (depending on who the referee is, or what the trend of sticness is) we will be answering the following questions:

- Who are the top 10 strictest Premier League referees between 2015-2016 and 2017-2018?
- Are the number of yellow cards issued trending up or down since 2001?
- Are the number of red cards issued trending up or down since 2001?
- Season with yellow card and red card

First, we want to identify who are strict referees from 2015-2017 season in terms of red cards & yellow cards. So, we sum up the red cards and yellow cards of each referees. After that, rank these two columns descendingly.

In [112…

```
%%sql
SELECT A.referee, sum_yellow_cards, sum_red_cards
```

```
FROM
(
SELECT r.referee,
    sum(match_event_count) sum_yellow_cards,
    count(1) as "Total Yellow Cards"
FROM Game R, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Yellow Card (disadvantage)' and
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    NOT s.season_name = '2001 | 2002' and
    NOT s.season_name = '2002 | 2003' and
    NOT s.season_name = '2003 | 2004' and
    NOT s.season_name = '2004 | 2005' and
    NOT s.season_name = '2005 | 2006' and
    NOT s.season_name = '2006 | 2007' and
    NOT s.season_name = '2007 | 2008' and
    NOT s.season_name = '2008 | 2009' and
    NOT s.season_name = '2009 | 2010' and
    NOT s.season_name = '2010 | 2011' and
    NOT s.season_name = '2011 | 2012' and
    NOT s.season_name = '2012 | 2013' and
    NOT s.season_name = '2013 | 2014' and
    NOT s.season_name = '2014 | 2015' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY r.referee, g.match_event_key) as A,

(
SELECT r.referee,
    sum(match_event_count) sum_red_cards,
    count(1) as "Total Yellow Cards"
FROM Game R, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Red Card (disadvantage)' and
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    NOT s.season_name = '2001 | 2002' and
    NOT s.season_name = '2002 | 2003' and
    NOT s.season_name = '2003 | 2004' and
    NOT s.season_name = '2004 | 2005' and
    NOT s.season_name = '2005 | 2006' and
    NOT s.season_name = '2006 | 2007' and
    NOT s.season_name = '2007 | 2008' and
    NOT s.season_name = '2008 | 2009' and
    NOT s.season_name = '2009 | 2010' and
    NOT s.season_name = '2010 | 2011' and
    NOT s.season_name = '2011 | 2012' and
    NOT s.season_name = '2012 | 2013' and
    NOT s.season_name = '2013 | 2014' and
    NOT s.season_name = '2014 | 2015' and
    s.key = g.season_key and
```

```
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY r.referee, g.match_event_key) as B
WHERE a.referee=b.referee
ORDER BY sum_red_cards DESC, sum_yellow_cards DESC
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[112]:

referee	sum_yellow_cards	sum_red_cards
M Dean	309	17
M Oliver	296	13
J Moss	296	11
A Taylor	318	10
C Pawson	262	10
A Marriner	256	10
M Atkinson	288	9
R Madley	245	9
M Clattenburg	167	9
L Mason	171	8

Consider both yellow and red cards issues, the top 5 strict referees are M Dean, M Oliver, J Moss, A Taylor, and C Pawson. If the referees have the same yellow cards, the code use red card number to rank and make the sequence. When playing against these referees, players need to be careful and avoid the disadvantages.

Next step, we seperated two cards. This part, we identify which referee issued the most yellow cards from 2015-2017 season. Appying the the same method, sum up the yellow cards of each referee and rank by the number of yellow cards.

In [113…

```
%%sql
SELECT r.referee,
    sum(match_event_count) sum_yellow_cards
FROM Game R, Season S, Match_Events M, Games G
WHERE
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    NOT s.season_name = '2001 | 2002' and
    NOT s.season_name = '2002 | 2003' and
    NOT s.season_name = '2003 | 2004' and
    NOT s.season_name = '2004 | 2005' and
    NOT s.season_name = '2005 | 2006' and
    NOT s.season_name = '2006 | 2007' and
    NOT s.season_name = '2007 | 2008' and
    NOT s.season_name = '2008 | 2009' and
    NOT s.season_name = '2009 | 2010' and
```

```
    NOT s.season_name = '2010 | 2011' and
    NOT s.season_name = '2011 | 2012' and
    NOT s.season_name = '2012 | 2013' and
    NOT s.season_name = '2013 | 2014' and
    NOT s.season_name = '2014 | 2015' and
    m.match_event = 'Yellow Card (disadvantage)' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY r.referee, g.match_event_key
ORDER BY sum_yellow_cards DESC
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[113]:

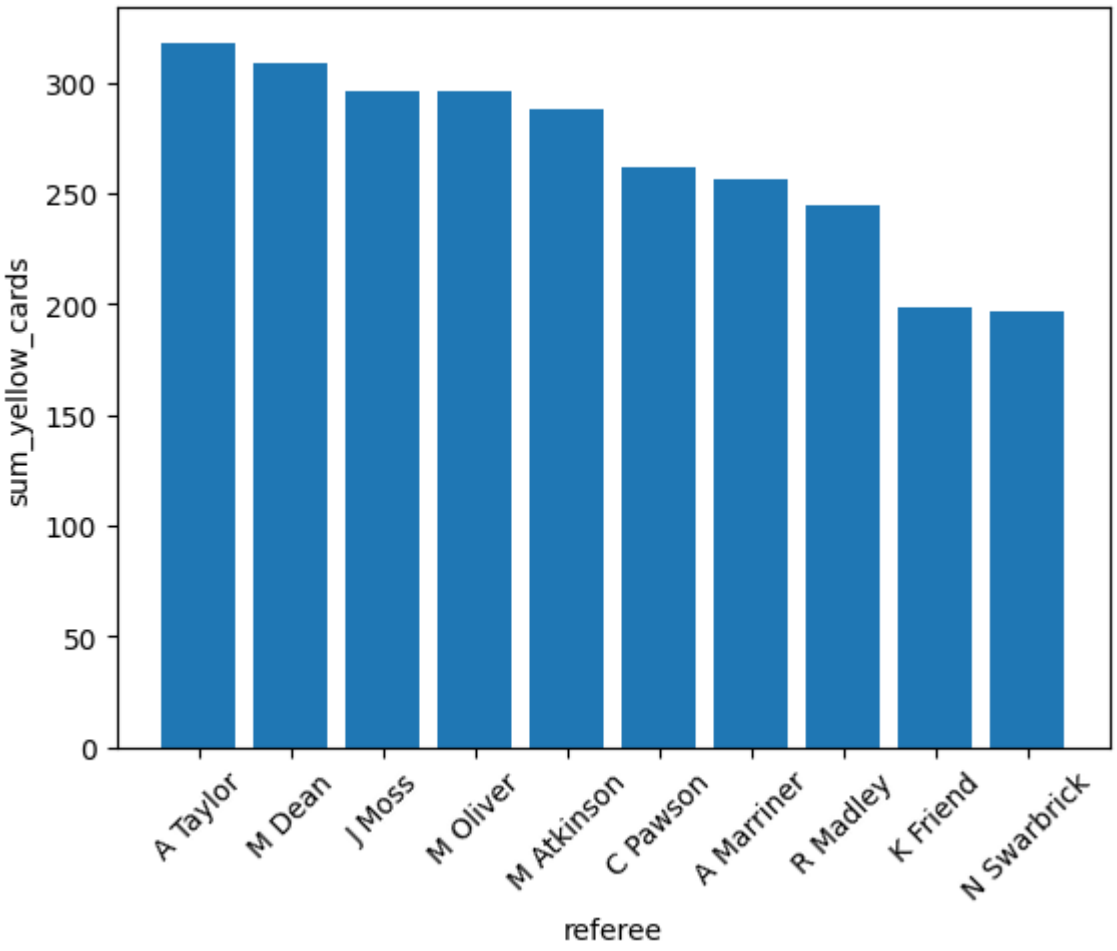
referee	sum_yellow_cards
A Taylor	318
M Dean	309
J Moss	296
M Oliver	296
M Atkinson	288
C Pawson	262
A Marriner	256
R Madley	245
K Friend	199
N Swarbrick	197

In [114…

```
_.bar()
```

Out[114]:

```
<BarContainer object of 10 artists>
```



In terms of the yellow card issued, A Taylor issued more than other referees (318). Following, M Dean is the second and the J Moss the third. These are the top three referees who issue more yellow cards. Thus, player can be aware of them in advance.

Furthermore, we want to identify which referee issued more red cards. The method is similar: sum up the red cards of each referee and rank the number descendingly.

In [115…

```
%%sql
SELECT r.referee,
    sum(match_event_count) sum_red_cards
FROM Game R, Season S, Match_Events M, Games G
WHERE
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    NOT s.season_name = '2001 | 2002' and
    NOT s.season_name = '2002 | 2003' and
    NOT s.season_name = '2003 | 2004' and
    NOT s.season_name = '2004 | 2005' and
    NOT s.season_name = '2005 | 2006' and
    NOT s.season_name = '2006 | 2007' and
    NOT s.season_name = '2007 | 2008' and
    NOT s.season_name = '2008 | 2009' and
    NOT s.season_name = '2009 | 2010' and
    NOT s.season_name = '2010 | 2011' and
```

```
    NOT s.season_name = '2011 | 2012' and
    NOT s.season_name = '2012 | 2013' and
    NOT s.season_name = '2013 | 2014' and
    NOT s.season_name = '2014 | 2015' and
    m.match_event = 'Red Card (disadvantage)' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY r.referee, g.match_event_key
ORDER BY sum_red_cards DESC
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[115]:

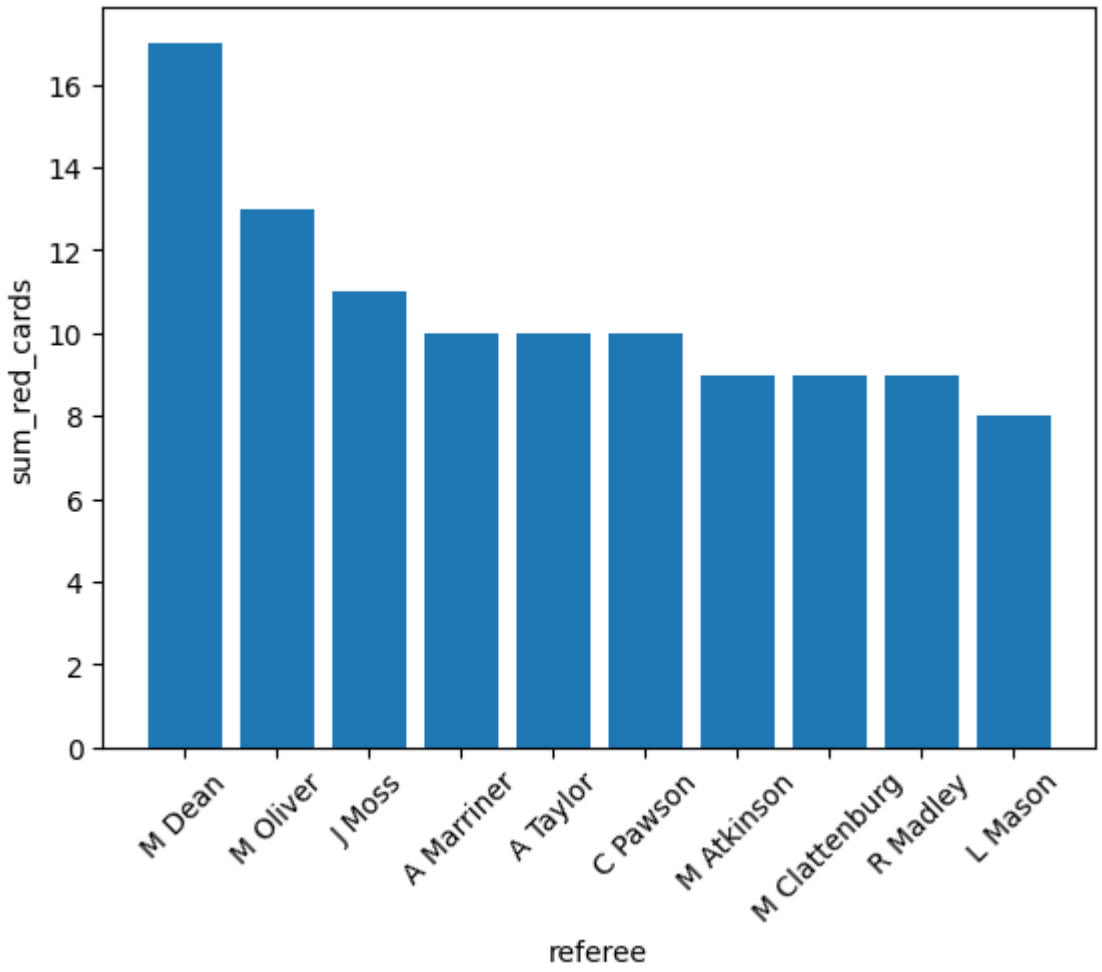
referee	sum_red_cards
M Dean	17
M Oliver	13
J Moss	11
A Marriner	10
A Taylor	10
C Pawson	10
M Atkinson	9
M Clattenburg	9
R Madley	9
L Mason	8

In [116…

```
_.bar()
```

Out[116]:

```
<BarContainer object of 10 artists>
```



In terms of the red card issued, M Dean issued more than other referees (17). Following, M Oliver is the second the J Moss is the third. We notice that M Dean issued more red ticks than the other referees. Apparenly, team players need to be more causious under his watches.

Yellow Cards issued after between 2001-2002 and 2017-2018

We want to study whether the game is playing more competitive as times go by. The way to get the result is by filter the time from 2001-2017, and sum up the yellow cards by each season, and order by the season.

In [117…

```
%%sql
SELECT s.season_name,
    sum(match_event_count) sum_yellow_cards
FROM Game R, Season S, Match_Events M, Games G
WHERE
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    m.match_event = 'Yellow Card (disadvantage)' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
```

```
GROUP BY s.season_name, g.match_event_key
ORDER BY season_name;
```

```
 * postgresql://student@/finalproject
17 rows affected.
```

Out[117]:

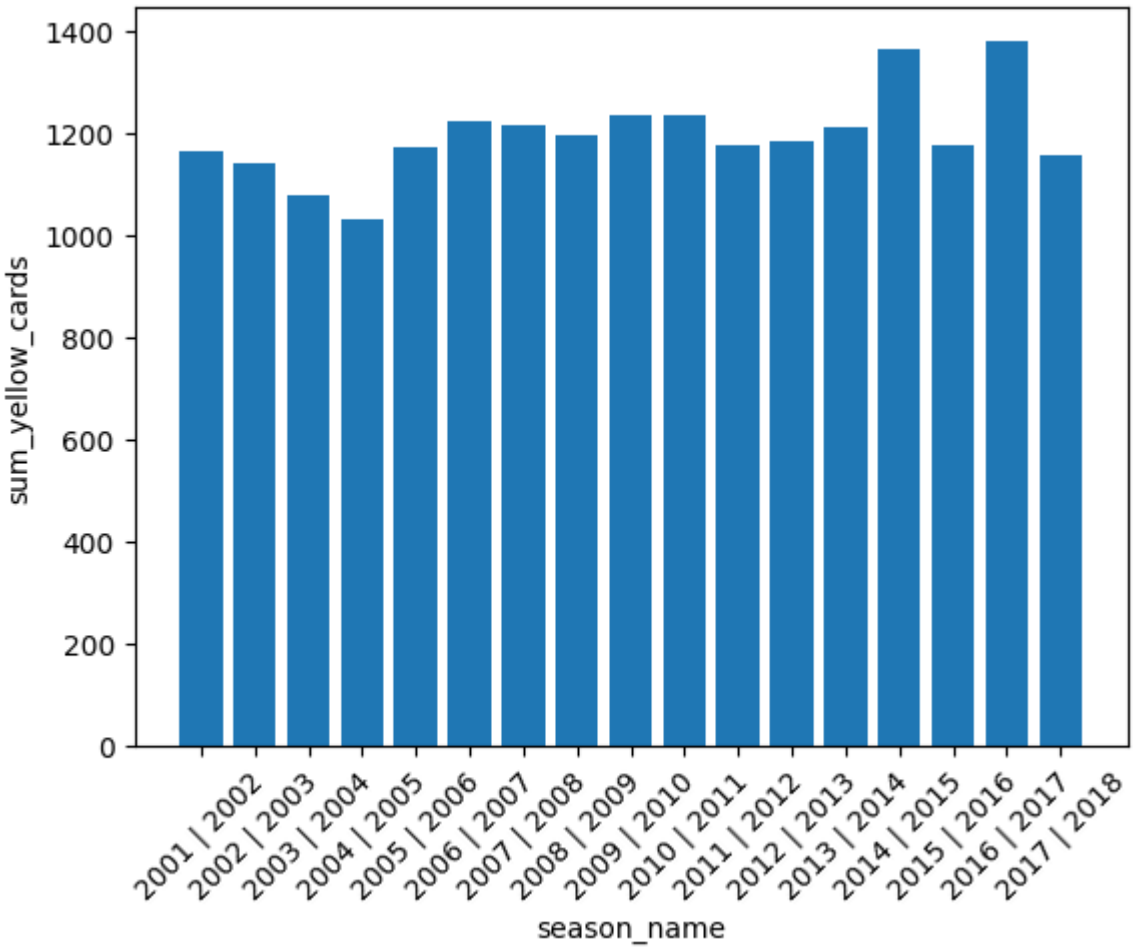
season_name	sum_yellow_cards
2001 | 2002	1165
2002 | 2003	1142
2003 | 2004	1081
2004 | 2005	1031
2005 | 2006	1173
2006 | 2007	1225
2007 | 2008	1216
2008 | 2009	1198
2009 | 2010	1237
2010 | 2011	1236
2011 | 2012	1178
2012 | 2013	1186
2013 | 2014	1212
2014 | 2015	1364
2015 | 2016	1179
2016 | 2017	1380
2017 | 2018	1157

In [118…

```
_.bar()
```

Out[118]:

```
<BarContainer object of 17 artists>
```



It seems that yellow cards issued aren't related with time: there is a up and down with time but no clear trend. From 2001 to 2004 season, the number decreases a bit. From 2004-2017, most of the yellow cards issued are around 1200 yellow cards issued. Only 2014 and 2016 season have higher numbers that does not fit into the trend.

Red Cards issued after between 2001-2002 and 2017-2018

Next, we want to study the red card issued with time. Still use the same method to filter, combine and rank to data.

In [119…

```
%%sql
SELECT s.season_name,
    sum(match_event_count) sum_red_cards
FROM Game R, Season S, Match_Events M, Games G
WHERE
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    m.match_event = 'Red Card (disadvantage)' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
```

```
GROUP BY s.season_name, g.match_event_key
ORDER BY season_name;
```

```
 * postgresql://student@/finalproject
17 rows affected.
```

Out[119]:

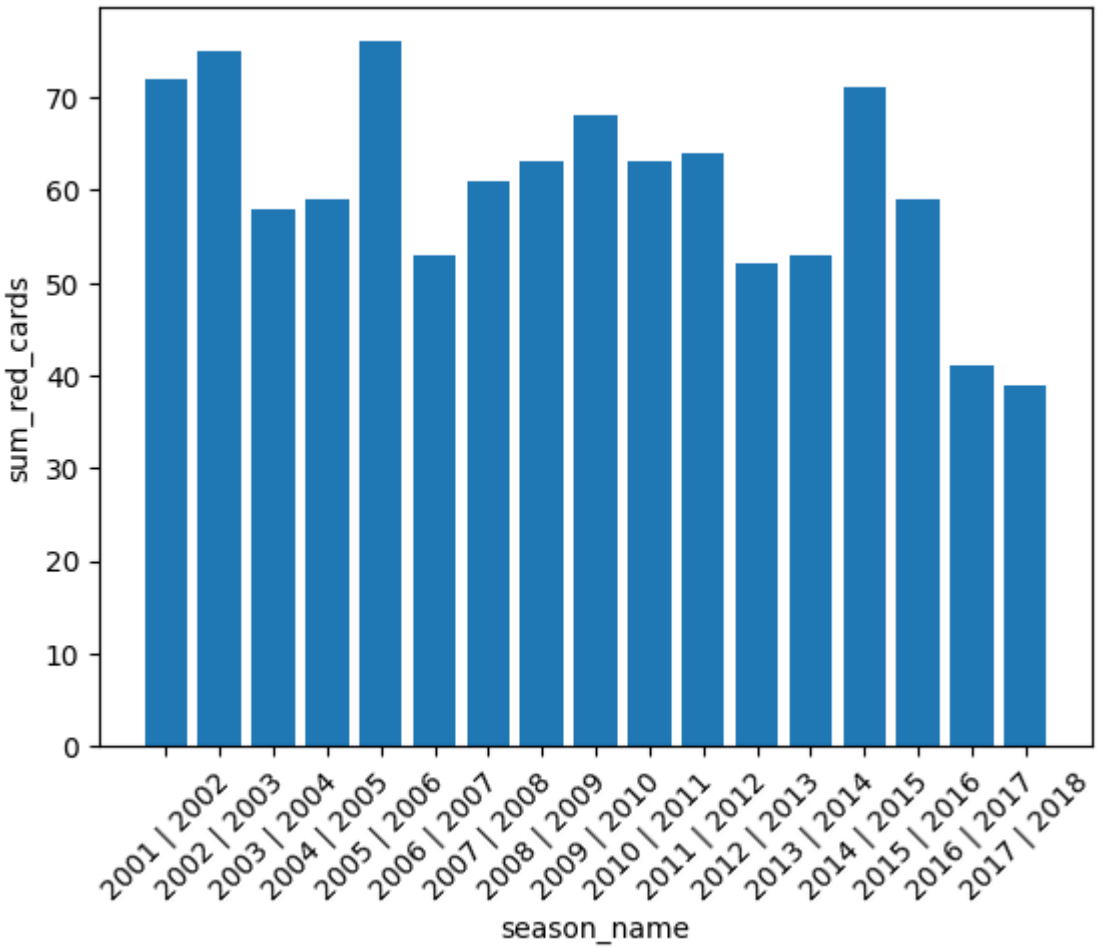
season_name	sum_red_cards
2001 | 2002	72
2002 | 2003	75
2003 | 2004	58
2004 | 2005	59
2005 | 2006	76
2006 | 2007	53
2007 | 2008	61
2008 | 2009	63
2009 | 2010	68
2010 | 2011	63
2011 | 2012	64
2012 | 2013	52
2013 | 2014	53
2014 | 2015	71
2015 | 2016	59
2016 | 2017	41
2017 | 2018	39

In [120…

```
_.bar()
```

Out[120]:

```
<BarContainer object of 17 artists>
```



Side-by-side comparison of yellow cards and red cards issued after between 2001-2002 and 2017-2018

In the table, we can compare time, yellow cards, and red cards together. Coaches will be interested in these two data combined together.

In [121…

```
%%sql
SELECT A.season_name, sum_yellow_cards, sum_red_cards
FROM
(
SELECT s.season_name,
    sum(match_event_count) sum_yellow_cards,
    count(1) as "Total Yellow Cards"
FROM Game R, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Yellow Card (disadvantage)' and
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY s.season_name, g.match_event_key) as A,

(
```

```
SELECT s.season_name,
    sum(match_event_count) sum_red_cards,
    count(1) as "Total Yellow Cards"
FROM Game R, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Red Card (disadvantage)' and
    NOT s.season_name = '1997 | 1998' and
    NOT s.season_name = '1998 | 1999' and
    NOT s.season_name = '1999 | 2000' and
    NOT s.season_name = '2000 | 2001' and
    s.key = g.season_key and
    r.key = g.game_key and
    m.key = g.match_event_key
GROUP BY s.season_name, g.match_event_key) as B
WHERE a.season_name=b.season_name
ORDER BY season_name;
```

```
 * postgresql://student@/finalproject
17 rows affected.
```

Out[121]:

season_name	sum_yellow_cards	sum_red_cards
2001 | 2002	1165	72
2002 | 2003	1142	75
2003 | 2004	1081	58
2004 | 2005	1031	59
2005 | 2006	1173	76
2006 | 2007	1225	53
2007 | 2008	1216	61
2008 | 2009	1198	63
2009 | 2010	1237	68
2010 | 2011	1236	63
2011 | 2012	1178	64
2012 | 2013	1186	52
2013 | 2014	1212	53
2014 | 2015	1364	71
2015 | 2016	1179	59
2016 | 2017	1380	41
2017 | 2018	1157	39

Business Question 3:

What is the total points earned by teams across the 21 seasons recorded? Do corner kicks affect the number of points earned? What is the relationship between these two variables?

To analyze whether corner kicks gives an advantage to teams, below we will create a table to display total points and toal corners played for all teams across the 21 seasons:

In [122…

```
%%sql
SELECT A.team_name, sum_points, sum_corners_played
FROM
(
SELECT t.team_name,
    sum(match_event_count) sum_points,
    count(1) as "count"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as A,

(SELECT t.team_name,
    sum(match_event_count) sum_corners_played,
    count(1) as "points"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Corners Played' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as B
WHERE a.team_name=b.team_name
ORDER BY sum_points DESC;
```

```
 * postgresql://student@/finalproject
48 rows affected.
```

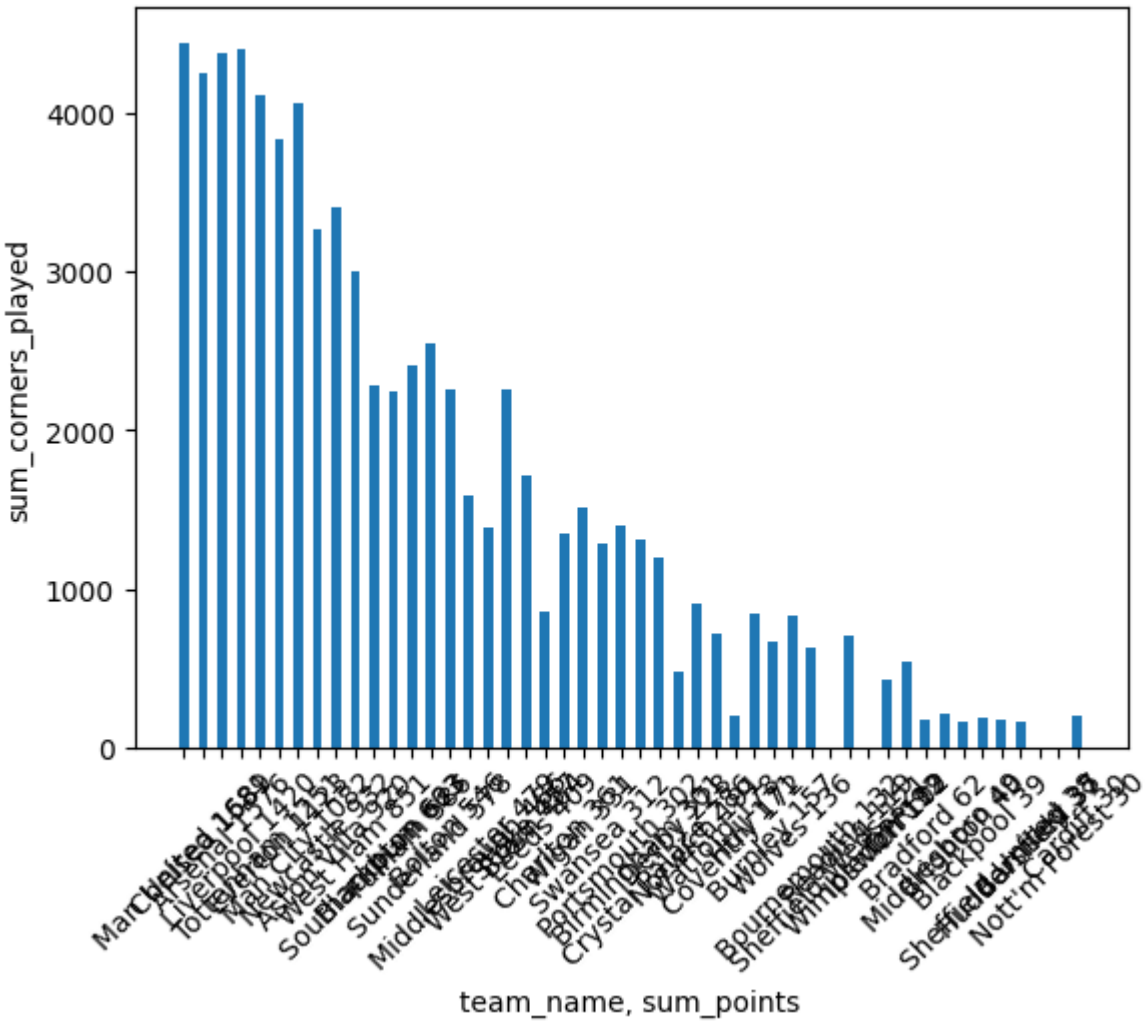
Out[122]:

team_name	sum_points	sum_corners_played
Man United	1681	4440
Chelsea	1589	4250
Arsenal	1576	4381
Liverpool	1420	4400
Tottenham	1251	4111
Everton	1123	3831
Man City	1082	4065
Newcastle	932	3269
Aston Villa	920	3401
West Ham	851	2998
Southampton	663	2280
Blackburn	623	2242
Fulham	586	2408
Sunderland	578	2551
Bolton	546	2258
Middlesbrough	486	1592
Leicester	479	1387
West Brom	464	2255
Stoke	457	1712
Leeds	409	862
Charlton	361	1350
Wigan	331	1508
Swansea	312	1281
Portsmouth	302	1399
Birmingham	301	1313
Crystal Palace	286	1200
Derby	228	477
Norwich	191	912
Watford	178	725
Coventry	172	206
Hull	171	847
Burnley	157	666
Wolves	136	832

team_name	sum_points	sum_corners_played
Bournemouth	132	629
Sheffield Weds	121	0
Reading	119	707
Wimbledon	119	0
Ipswich	102	430
QPR	92	544
Bradford	62	179
Middlesboro	49	211
Brighton	40	163
Blackpool	39	186
Sheffield United	38	180
Huddersfield	37	165
Barnsley	35	0
Nott'm Forest	30	0
Cardiff	30	196

In [123…
```
_.bar(width = 0.5)
```

Out[123]:
```
<BarContainer object of 48 artists>
```



From the bar chart, we can see in general the more corners played, the more points earned across the 20 seasons. However we can see that some teams did not benefit from this correlation in the way that the majority did.

For example, West Brom had 2255 corners played across the 21 seasons but only earned 464 points from games in this time period. To compare, Blackburn had a similar number of corners played, 2242, but earned 623 points.

We can also look at Man City who had 4065 corners played, the 6th most out of all teams, and earned 1082 points, 7th most. Everton, on the other hand, had 3831 corners played, less than Man City, and 1123 total points, more than City overall.

We would like to conduct similar analysis but focusing on the disadvantage of conceding corners and what effect this has on points earned. We will select all 21 seasons.

In [124…

```
%%sql
SELECT A.team_name, sum_points, sum_corners_conceded
FROM
(
SELECT t.team_name,
    sum(match_event_count) sum_points,
    count(1) as "count"
```

```
FROM Team T, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Points (Full Time)' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as A,

(SELECT t.team_name,
    sum(match_event_count) sum_corners_conceded,
    count(1) as "corners conceded"
FROM Team T, Season S, Match_Events M, Games G
WHERE
    m.match_event = 'Corners Conceded' and
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key
GROUP BY t.team_name, g.match_event_key) as B
WHERE a.team_name=b.team_name
ORDER BY sum_points DESC;
```

```
 * postgresql://student@/finalproject
48 rows affected.
```

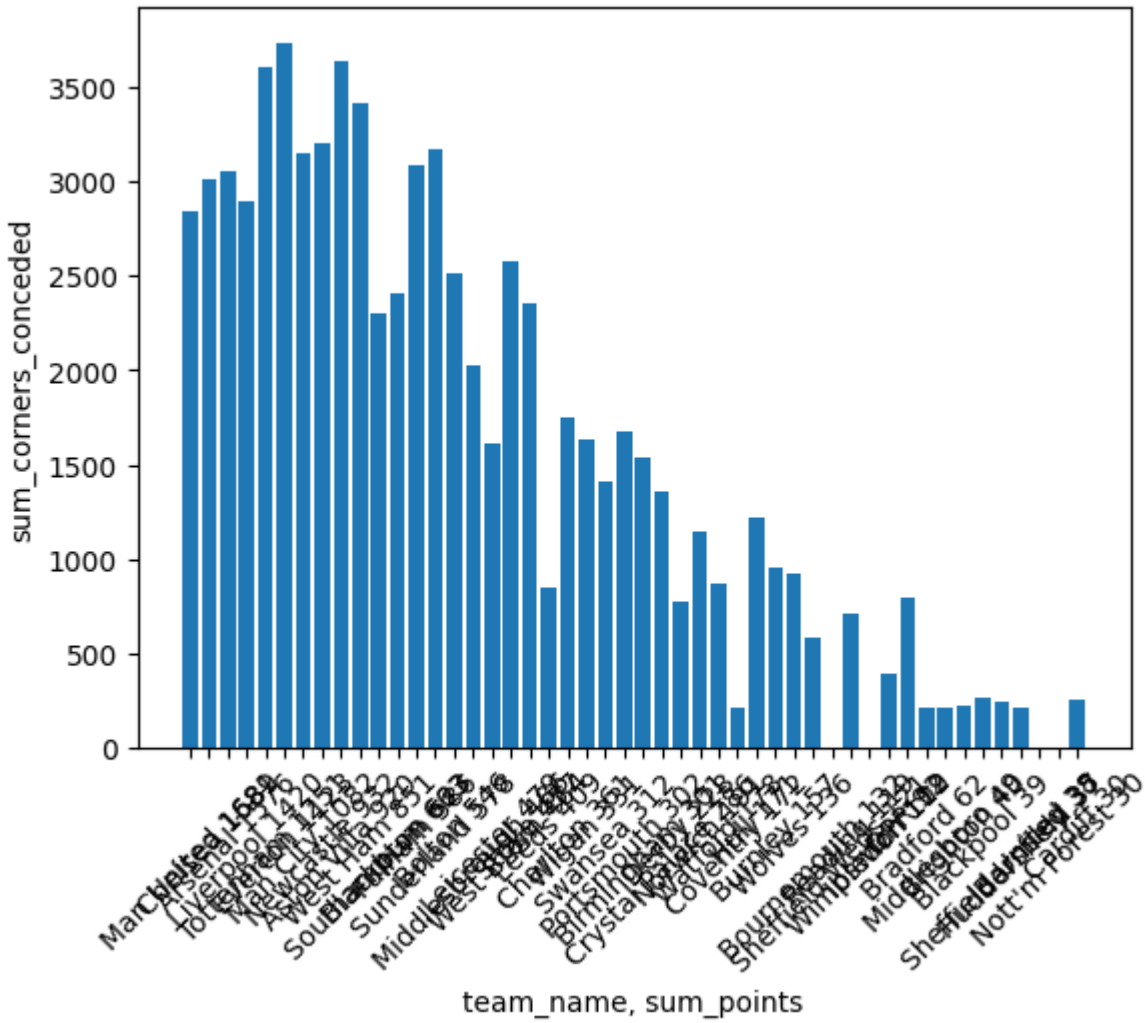
Out[124]:

team_name	sum_points	sum_corners_conceded
Man United	1681	2840
Chelsea	1589	3011
Arsenal	1576	3050
Liverpool	1420	2896
Tottenham	1251	3601
Everton	1123	3733
Man City	1082	3150
Newcastle	932	3205
Aston Villa	920	3637
West Ham	851	3413
Southampton	663	2301
Blackburn	623	2409
Fulham	586	3091
Sunderland	578	3175
Bolton	546	2511
Middlesbrough	486	2023
Leicester	479	1617
West Brom	464	2579
Stoke	457	2357
Leeds	409	846
Charlton	361	1748
Wigan	331	1638
Swansea	312	1406
Portsmouth	302	1679
Birmingham	301	1533
Crystal Palace	286	1353
Derby	228	770
Norwich	191	1149
Watford	178	865
Coventry	172	216
Hull	171	1218
Burnley	157	950
Wolves	136	918

team_name	sum_points	sum_corners_conceded
Bournemouth	132	584
Sheffield Weds	121	0
Reading	119	711
Wimbledon	119	0
Ipswich	102	391
QPR	92	797
Bradford	62	214
Middlesboro	49	211
Brighton	40	228
Blackpool	39	266
Sheffield United	38	241
Huddersfield	37	209
Barnsley	35	0
Nott'm Forest	30	0
Cardiff	30	259

In [125…
```
_.bar()
```

Out[125]:
```
<BarContainer object of 48 artists>
```



From the bar chart, interestingly, we can see the 4 teams with the highest total points, had lower corners conceded than other teams such as Tottenham and Everton. We were expecting to see more corners conceded for teams with lower points but it is important to note that these teams with few points played fewer games overall and this has a big effect on how the correlation looks on the visualization.

In [ ]:

Business Question 4:

Q: Which team had received the highest number of yellow cards during the season 2017-2018 and under which referee did they receive them from? From this, we can advise certain teams to play more conservatively in future games when they are assigned these referees.

In [126…

```
%%sql
SELECT t.team_name, r.referee,
    sum(match_event_count) sum_yellow_cards_received
FROM Team T, Season S, Match_Events M, Games G, Game R
WHERE
    s.season_name = '2017 | 2018' and
    m.match_event = 'Yellow Card (disadvantage)' and
```

```
    s.key = g.season_key and
    t.key = g.team_key and
    m.key = g.match_event_key and
    r.key = g.game_key
GROUP BY r.referee, g.match_event_key,  t.team_name
ORDER BY sum_yellow_cards_received DESC
LIMIT 10;
```

```
 * postgresql://student@/finalproject
10 rows affected.
```

Out[126]:

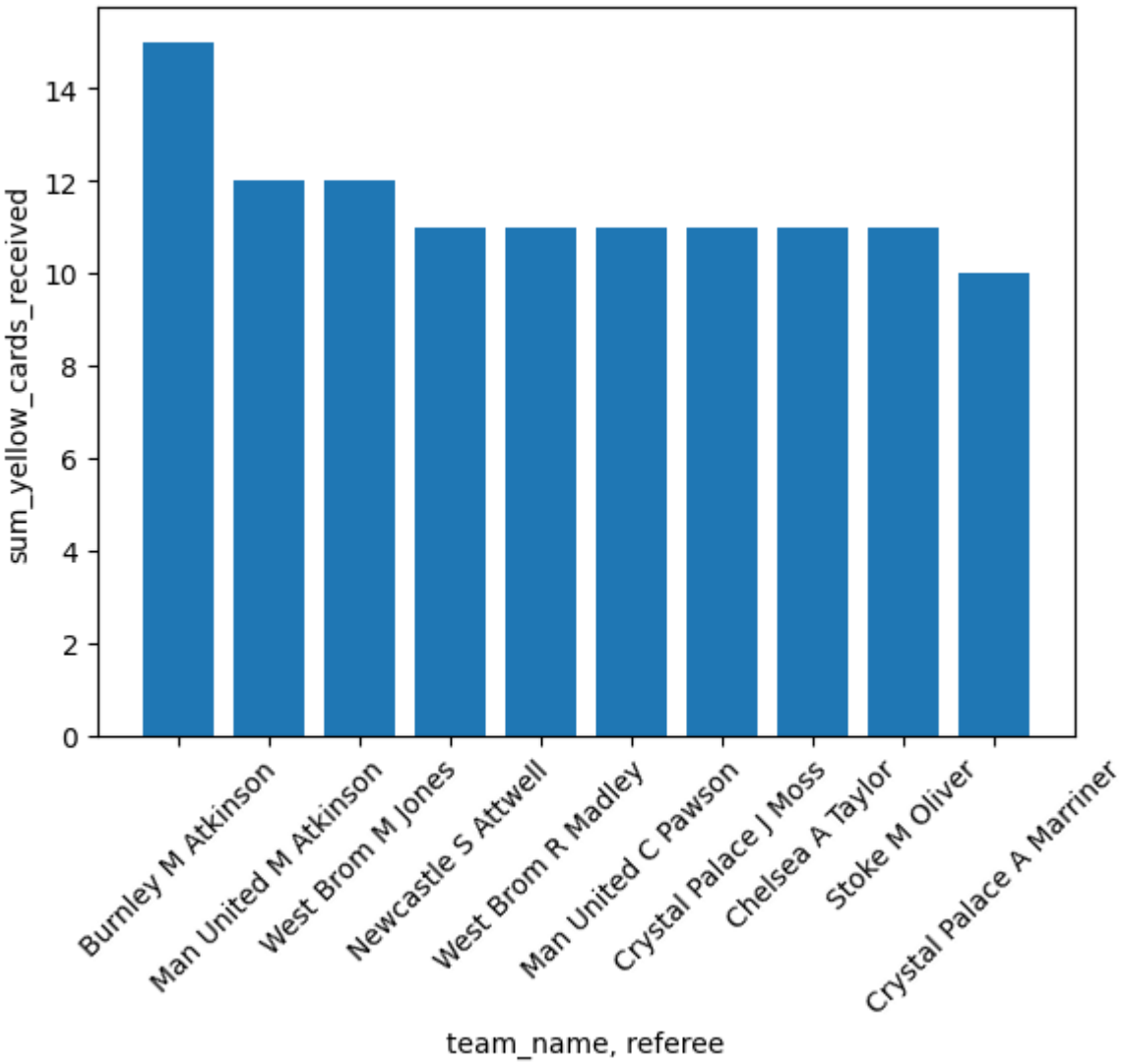
team_name	referee	sum_yellow_cards_received
Burnley	M Atkinson	15
Man United	M Atkinson	12
West Brom	M Jones	12
Newcastle	S Attwell	11
West Brom	R Madley	11
Man United	C Pawson	11
Crystal Palace	J Moss	11
Chelsea	A Taylor	11
Stoke	M Oliver	11
Crystal Palace	A Marriner	10

In [127…

```
_.bar()
```

Out[127]:

```
<BarContainer object of 10 artists>
```



Following these results, we would highly encourage Burnley to play more conservatively and encourage their players not to make strong tackles in games where Martin Atkinson is the referee in the future. We also recommend that Man United spend additional time on defensive tackling drills since they appeared twice in the top ten teams to receive yellow cards from the same referee, with Martin Atkinson handing out 15 yellow cards to them and Craig Pawson handing them 11 yellow cards.

In this same season, Man United finished with 81 points which was 19 fewer than the league champions Man City, who finished with 100 points. Man City did not appear in this table at all so we would argue that these yellow cards may have an influence on points earned overall.