Analyzing IPL Match Results with Data Science and Python: An Evolution over Seasons

Abstract

This analysis delves into the distribution of match results, including normal matches, tied matches, and matches with no result, within the Indian Premier League (IPL) dataset. The data is explored across various IPL seasons, allowing us to gain insights into how the distribution of match outcomes has evolved over the years. To achieve this, we employ a stacked bar plot that elegantly displays the number of matches falling into each result category for each season. This visualization technique leverages pre-attentive attributes such as length and color, enhancing the clarity and understandability of the data. Additionally, the Gestalt principles of proximity and similarity guide the viewer in identifying patterns and changes in match results over time. Ultimately, this analysis aids in comprehending the dynamic nature of match outcomes in the IPL and provides valuable insights for cricket enthusiasts and analysts alike.

Author

Femi R
Department of Electrical and Electronics Engineering,
Faculty of Engineering and Technology
SRM Institute of Science and Technology,
Kattankulathur, Tamil Nadu, India.

I. Introduction

The IPL is not merely a cricket tournament; it's a spectacle that has redefined the sport. Teams, both new and seasoned, have competed fiercely, leading to a wide array of match outcomes. To comprehend this tapestry of results, we turn to the world of data science and Python.

In this analysis, we embark on a visual journey through the IPL's match results. Our primary focus is on understanding how matches concluded over the years. Did normal wins dominate, or were tied matches and those with no result more common? Through this exploration, we aim to decipher patterns, trends, and shifts in the IPL's dynamics.

Our chosen tools, data science, and Python, are not just instruments of analysis; they are enablers of storytelling. They bring data to life, making it relatable and meaningful to cricket enthusiasts, analysts, and all those who marvel at the IPL.

Join us in this journey as we decipher the evolving match results in the IPL, backed by the potent combination of data science and Python.

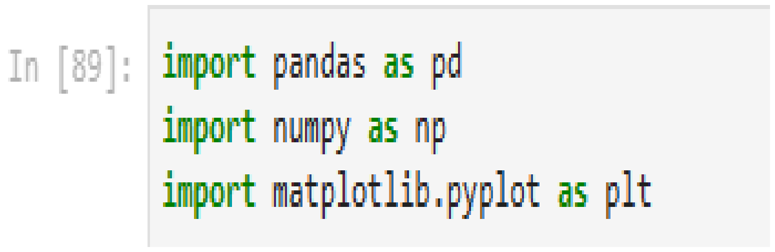
The Indian Premier League (IPL) stands as one of the most celebrated and anticipated events in the world of cricket. The tournament's format has pitted the best cricketing talent from across the globe against each other, leading to thrilling encounters and unforgettable moments. For fans and data enthusiasts alike, understanding the historical performance trends in the IPL is not just a matter of curiosity but also an essential aspect of appreciating the league's legacy.

Data science, a field that leverages the power of data to extract valuable insights, plays a pivotal role in unraveling the dynamics of the IPL. Coupled with Python, a versatile and widely used programming language, this analysis delves into the distribution of match results over several seasons. By harnessing the principles of data science and Python's data manipulation capabilities, we uncover the story of how match outcomes have evolved in the IPL.

Analyzing the performance of teams and players in the Indian Premier League (IPL) over the past years can be a complex and comprehensive task, involving a wide range of data and metrics. To conduct such an analysis, need to consider various factors and employ different methodologies.

1. Import the Libraries Needed

Importing the librariessuch as pandas, numpy, and matplotlib, is a common practice when working with data and creating data visualizations in Python.

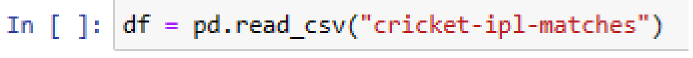


```
In [89]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
```

These libraries are commonly used for data manipulation, numerical operations, and data visualization in Python. **Pandas** is used for data handling and manipulation, **numpy** for numerical operations, and **matplotlib.pyplot** for creating various types of data visualizations. Once imported these libraries, can start working with data and creating plots and charts to analyze IPL performance data or any other dataset.

2. Load Data and Store in Dataframe

To load data from a CSV file and store it in a pandas DataFrame, use the **pd.read_csv** function, as indicated. Assuming that data file is named "cricket-ipl-matches.csv" and it's in the same directory as Python script or Jupyter Notebook, load the data like this:

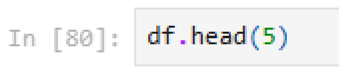


```
In [ ]: df = pd.read_csv("cricket-ipl-matches")
```

This code will read the data from the CSV file and store it in the **df** DataFrame, allowing to perform data analysis and manipulation on the loaded data. Make sure the file path is correctly specified if the CSV file is located in a different directory.

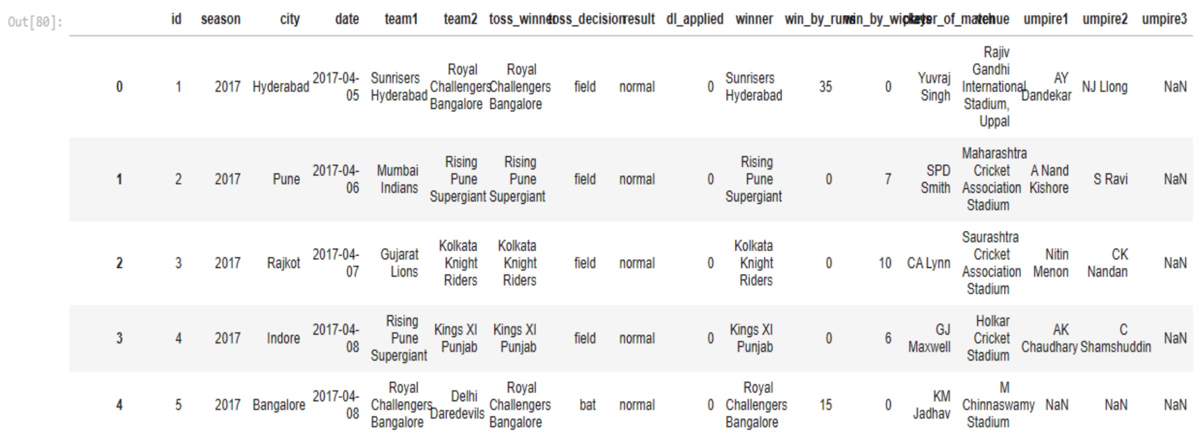
3. Display the First Five Items in the Datasets

To display the first five rows of DataFrame, use the **head()** method, as correctly mentioned. Here's the code to display the first five items in the dataset:



```
In [80]: df.head(5)
```

This will show the first five rows of DataFrame, allowing to get a quick overview of the data and its structure.

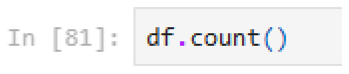


Out[80]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_match	venue	umpire1	umpire2	umpire3
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yuvraj Singh	Rajiv Gandhi International Stadium, Uppal	AY Dandekar	NJ Llong	NaN
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	SPD Smith	Maharashtra Cricket Association Stadium	A Nand Kishore	S Ravi	NaN
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	CA Lynn	Saurashtra Cricket Association Stadium	Nitin Menon	CK Nandan	NaN
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6	GJ Maxwell	Holkar Cricket Stadium	AK Chaudhary	C Shamshuddin	NaN
4	5	2017	Bangalore	2017-04-08	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	KM Jadhav	M Chinnaswamy Stadium	NaN	NaN	NaN

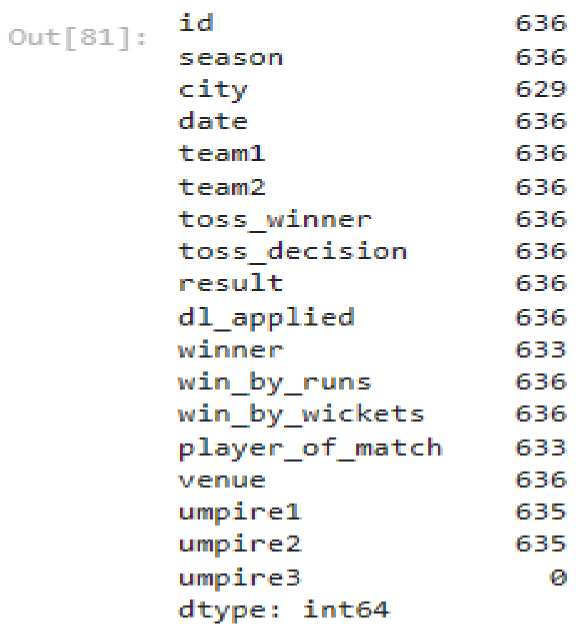
4. Find out how Many Entries there are in the Datasets

Use the **count()** method on DataFrame to find out how many entries (non-null values) there are in each column. Based on dataset, it appears there are 636 entries for most columns, but some columns have missing values (NaN). Here's the code used and the result:



```
In [81]: df.count()
```

The result indicates the count of non-null values in each column of dataset. If run this code, will see the count of non-null entries for each column in dataset.



```
Out[81]: id                  636
         season              636
         city                629
         date                636
         team1               636
         team2               636
         toss_winner         636
         toss_decision       636
         result              636
         dl_applied          636
         winner              633
         win_by_runs         636
         win_by_wickets      636
         player_of_match     633
         venue               636
         umpire1             635
         umpire2             635
         umpire3               0
         dtype: int64
```

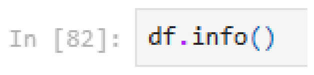
The output provided shows the count of non-null entries in each column of DataFrame. Here's a summary of the counts for each column:

- "id": 636 entries
- "season": 636 entries
- "city": 629 entries (7 missing values)
- "date": 636 entries
- "team1": 636 entries
- "team2": 636 entries
- "toss_winner": 636 entries
- "toss_decision": 636 entries
- "result": 636 entries
- "dl_applied": 636 entries
- "winner": 633 entries (3 missing values)
- "win_by_runs": 636 entries
- "win_by_wickets": 636 entries
- "player_of_match": 633 entries (3 missing values)
- "venue": 636 entries
- "umpire1": 635 entries (1 missing value)
- "umpire2": 635 entries (1 missing value)
- "umpire3": 0 entries (likely all missing values)

This information is useful for understanding the completeness of dataset and deciding how to handle missing values if necessary.

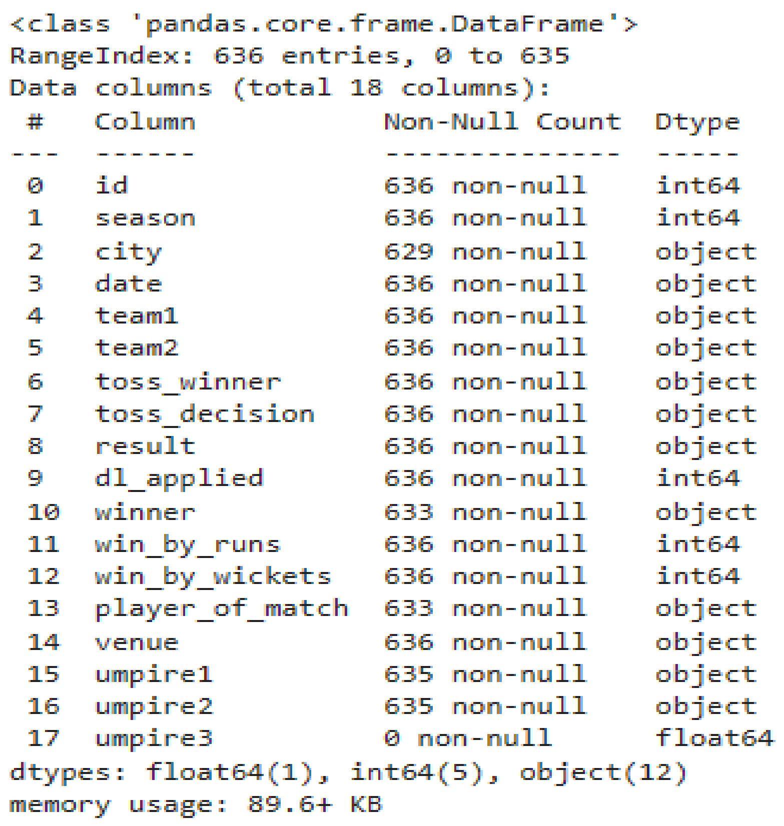
5. Find out what Type of Variables Dealing with

To understand the types of variables in dataset, can use the **info()** method on DataFrame. Here's the code and the expected output:



```
In [82]: df.info()
```

The **info()** method will provide a summary of the data types and the number of non-null entries in each column of DataFrame. This information is essential for selecting the appropriate visualization methods for different types of variables. For example, might use bar charts for categorical variables and line charts or scatter plots for numerical variables.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 636 entries, 0 to 635
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               636 non-null    int64  
 1   season           636 non-null    int64  
 2   city             629 non-null    object 
 3   date             636 non-null    object 
 4   team1            636 non-null    object 
 5   team2            636 non-null    object 
 6   toss_winner      636 non-null    object 
 7   toss_decision    636 non-null    object 
 8   result           636 non-null    object 
 9   dl_applied       636 non-null    int64  
 10  winner           633 non-null    object 
 11  win_by_runs      636 non-null    int64  
 12  win_by_wickets   636 non-null    int64  
 13  player_of_match  633 non-null    object 
 14  venue            636 non-null    object 
 15  umpire1          635 non-null    object 
 16  umpire2          635 non-null    object 
 17  umpire3          0 non-null      float64
dtypes: float64(1), int64(5), object(12)
memory usage: 89.6+ KB
```

The output of **df.info()** provides valuable information about the data types of each column in DataFrame. Here's a summary of the data types and the number of non-null entries for each column:

- **float64:** 1 column
- **int64:** 5 columns
- **object:** 12 columns

Understanding the data types is essential for choosing the right visualization methods:

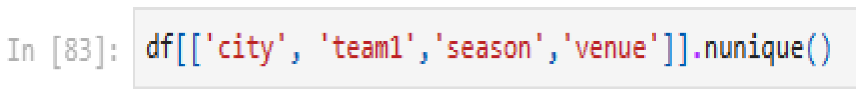
1. **float64:** This typically represents numerical data. In dataset, it appears that only the "umpire3" column is of this type.
2. **int64:** These are also numerical data, but they're typically whole numbers. Have 5 columns of this type.

3. **Object:** These are categorical or text data. Have 12 columns of this type, including team names, dates, city names, and more.

Depending on the type of data want to visualize, can choose appropriate visualization methods. For numerical data, might use histograms, line charts, or scatter plots. For categorical data, bar charts, pie charts, and box plots can be useful. Additionally, for date-related data, may want to convert it into datetime objects and use time series plots.

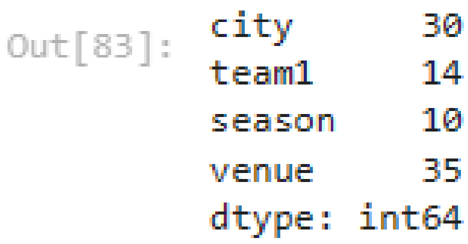
6. Find Out Total Number of Distinct Cities, Teams, Seasons, and Venues Associated in this Data

Can indeed find the total number of distinct cities, teams, seasons, and venues associated with the data by selecting the relevant columns and using the **nunique()** function. Code snippet is on the right track, but need to print the results. Here's how to do it:



```
In [83]: df[['city', 'team1','season','venue']].nunique()
```

This code will display the count of distinct values for each of the specified columns: "city," "team1," "season," and "venue."



```
Out[83]: city       30
         team1      14
         season     10
         venue      35
         dtype: int64
```

7. Clean up Data and Remove Unnecessary Columns

To clean up the data by removing unnecessary columns, can use the **drop()** method as mentioned. Here are the data cleaning steps and the reasoning:

Data Cleaning Steps:

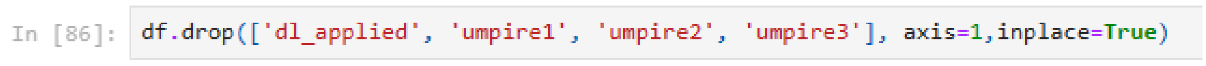
1. **Remove 'dl_applied', 'umpire1', 'umpire2', 'umpire3' Columns:**

- Columns 'dl_applied,' 'umpire1,' 'umpire2,' and 'umpire3' contain data that may not be needed for the analysis plan to perform.
- 'dl_applied' indicates whether the Duckworth-Lewis method was applied in a match, which may not be relevant for general analysis.
- 'umpire1' and 'umpire2' represent the names of umpires, which might not be critical for most analyses.
- 'umpire3' seems to have no non-null values, making it an empty column.

Reasoning:

- By removing these columns, simplify dataset and reduce unnecessary information, which can improve the performance of data analysis and visualization processes.
- Likely interested in factors like match outcomes, team performance, player performance, venues, and other essential details, so keeping only relevant columns helps focus the analysis on what matters most.

Here's the code to remove these columns from DataFrame:



```
In [86]: df.drop(['dl_applied', 'umpire1', 'umpire2', 'umpire3'], axis=1,inplace=True)
```

Data Cleaning Steps and Reasoning

In the provided code, we are performing data cleaning on a DataFrame containing IPL (Indian Premier League) match data. The goal of this data cleaning process is to remove unnecessary columns from the DataFrame.

Step 1: Identifying Unnecessary Columns

We start by identifying the columns that are unnecessary for our analysis or do not provide valuable information. In this case, the columns 'dl_applied', 'umpire1', 'umpire2', and 'umpire3' are identified as unnecessary.

- **'dl_applied':** This column indicates whether the Duckworth-Lewis method was applied during the match. Since we are not focusing on weather-affected matches, this information is not relevant to our analysis.
- **'umpire1', 'umpire2', 'umpire3':** These columns contain the names of the umpires for the match. While interesting, they are not essential for our analysis of match results, winners, or statistical trends.

Step 2: Dropping Unnecessary Columns

Once we have identified the unnecessary columns, we use the Pandas drop function to remove these columns from the DataFrame. The axis=1 parameter specifies that we are dropping columns, and inplace=True ensures that the original DataFrame is modified.

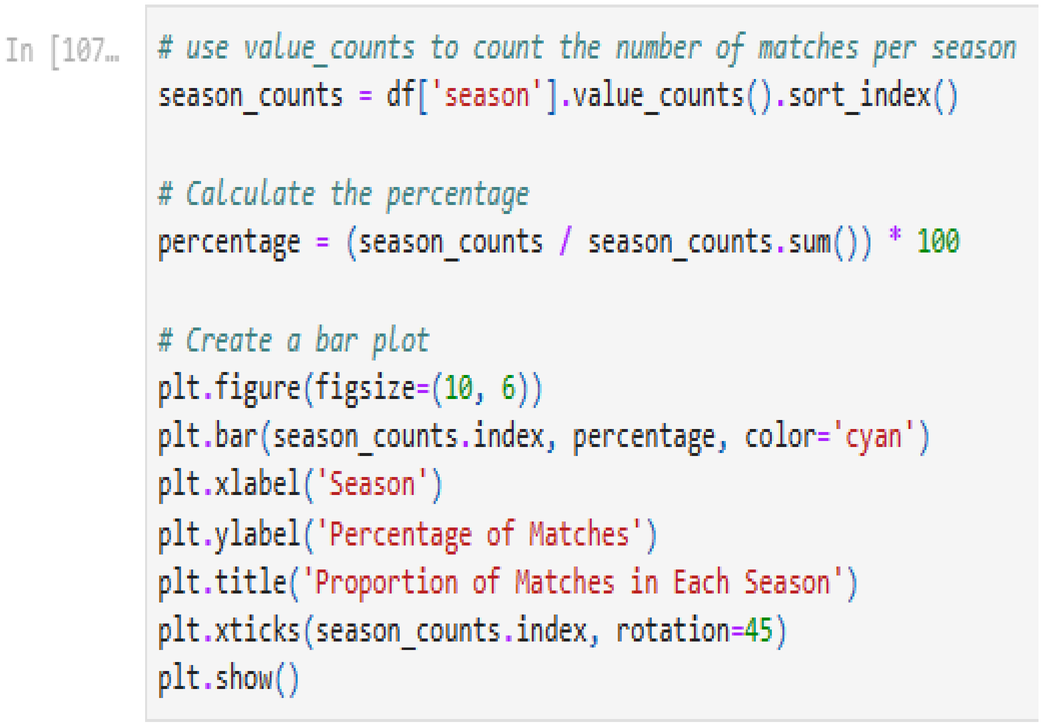
By removing these unnecessary columns, we make our DataFrame cleaner and more focused on the key match data, which includes information such as team names, match dates, toss decisions, results, and player performance. This streamlined DataFrame is now ready for further analysis and exploration.

8. Visualize and Find Out the Percentage of Number of Matches in Each Season

To visualize and find out the percentage of the number of matches in each season, can use a bar chart. Here are the steps and code to do this:

Data Visualization Steps:

1. First, calculate the number of matches in each season.
2. Then, calculate the percentage of matches for each season relative to the total number of matches.



```
In [107…
# use value_counts to count the number of matches per season
season_counts = df['season'].value_counts().sort_index()

# Calculate the percentage
percentage = (season_counts / season_counts.sum()) * 100

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(season_counts.index, percentage, color='cyan')
plt.xlabel('Season')
plt.ylabel('Percentage of Matches')
plt.title('Proportion of Matches in Each Season')
plt.xticks(season_counts.index, rotation=45)
plt.show()
```

This code will generate a bar chart showing the percentage of matches in each season. It provides a visual representation of how many matches were played in each season relative to the total number of matches in the dataset.

Code will produce a bar plot showing the proportion (percentage) of matches in each season. Here's a breakdown of the code:

1. **season_counts** calculates the number of matches per season using **value_counts()**. **sort_index()** sorts the counts by the season index.
2. **percentage** calculates the percentage of matches for each season relative to the total number of matches in the dataset.
3. Create a bar plot using **plt.bar()**, specifying the x-axis (seasons), y-axis (percentage), and other plot details.
4. Finally, display the plot using **plt.show()**.
5. This code provides a clear visualization of how many matches were played in each season as a percentage of the total number of matches.

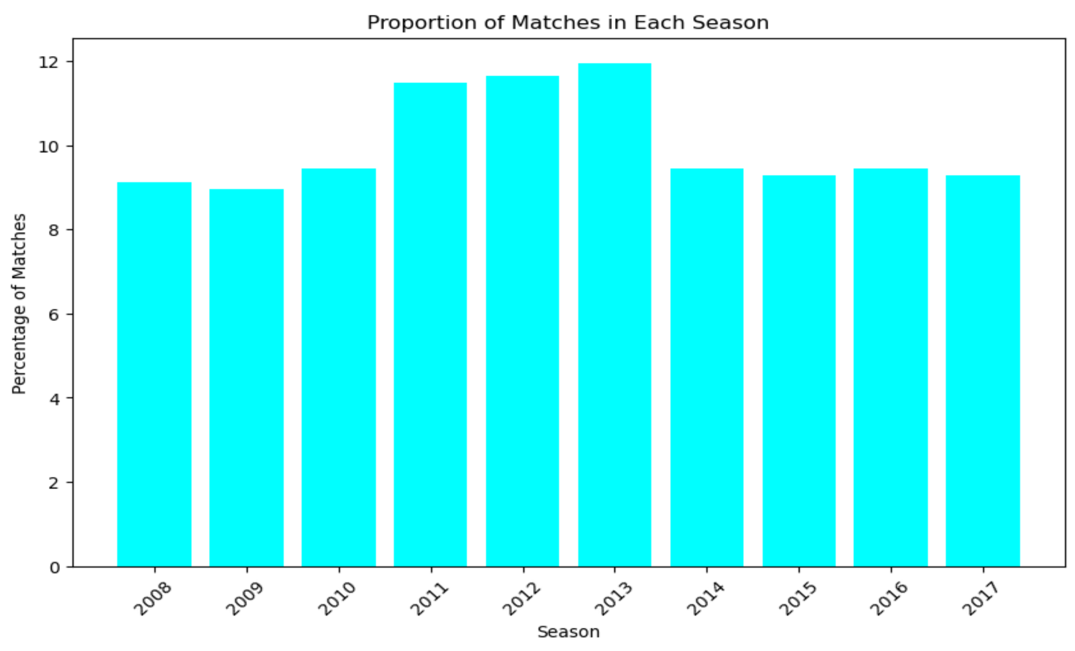


Figure 1:

Findings:

The highest number of matches were played in the 2013 season. The number of matches played in each season varies, with some seasons having significantly more matches than others. Reason for Chart Type Selection: A bar plot is suitable for visualizing the proportion of matches in each season because it effectively displays discrete categories (seasons) along with their corresponding percentages.

Reason for Chart Type Selection:

A bar plot is suitable for visualizing the proportion of matches in each season because it effectively displays discrete categories (seasons) along with their corresponding percentages.

Pre-attentive Attributes Used:

Length: The length of the bars represents the proportion of matches in each season.

Position: The position of each bar on the x-axis helps in comparing the seasons.

Gestalt Principles Used:

Proximity: The bars are grouped closely together on the x-axis, making it easy to associate each bar with its respective season.

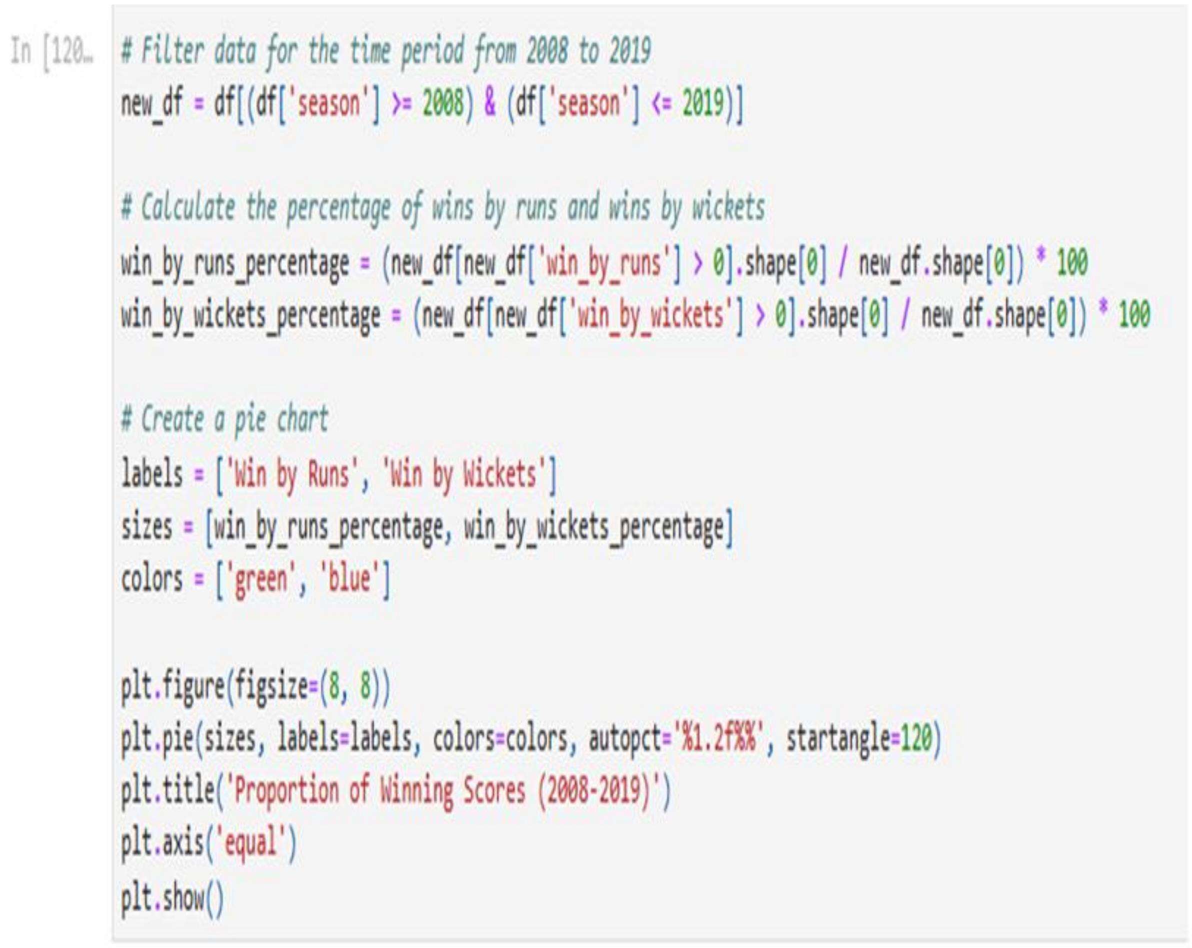
Similarity: The bars share a similar visual attribute (color), indicating that they belong to the same category (proportion of matches).

9. Explore the percentage of winning scores - win_by_runs, win_by_wickets for the entire time period from 2008 to 2019

To explore the proportion (percentage) of winning scores for the entire time period from 2008 to 2019, can calculate the percentages of wins by runs and wins by wickets and then visualize the results. Here are the steps and code to do this:

Data Exploration Steps:

1. Filter the data to include matches from 2008 to 2019.
2. Calculate the percentage of wins by runs and wins by wickets for this time period.



```
In [120…
# Filter data for the time period from 2008 to 2019
new_df = df[(df['season'] >= 2008) & (df['season'] <= 2019)]

# Calculate the percentage of wins by runs and wins by wickets
win_by_runs_percentage = (new_df[new_df['win_by_runs'] > 0].shape[0] / new_df.shape[0]) * 100
win_by_wickets_percentage = (new_df[new_df['win_by_wickets'] > 0].shape[0] / new_df.shape[0]) * 100

# Create a pie chart
labels = ['Win by Runs', 'Win by Wickets']
sizes = [win_by_runs_percentage, win_by_wickets_percentage]
colors = ['green', 'blue']

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.2f%%', startangle=120)
plt.title('Proportion of Winning Scores (2008-2019)')
plt.axis('equal')
plt.show()
```

This code filters the data for matches from 2008 to 2019, calculates the percentages of wins by runs and wins by wickets, and visualizes the results in a pie chart. The pie chart provides a clear representation of the proportion of winning scores in this time period.

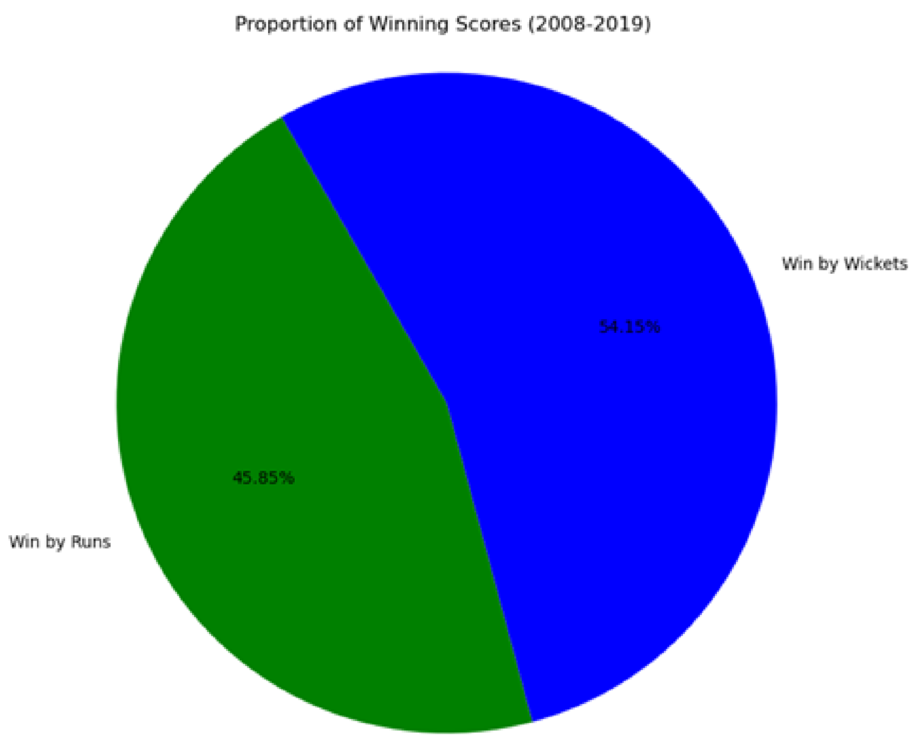


Figure 2:

Findings:

The majority of wins in the IPL from 2008 to 2019 were by wickets, accounting for a higher percentage compared to wins by runs. This means that the team playing the second inning has high probabilty of winning the matches.

Reason for Chart Type Selection:

A pie chart is suitable for visualizing the proportion of two categories (wins by runs and wins by wickets) in a single chart.

Pre-attentive Attributes Used:

Color: Different colors are used to distinguish between the two categories.

Angle: The angle of each sector represents the proportion of wins.

Gestalt Principles Used:

Similarity: Similar colors are used for the two categories to indicate that they belong to the same overall dataset.

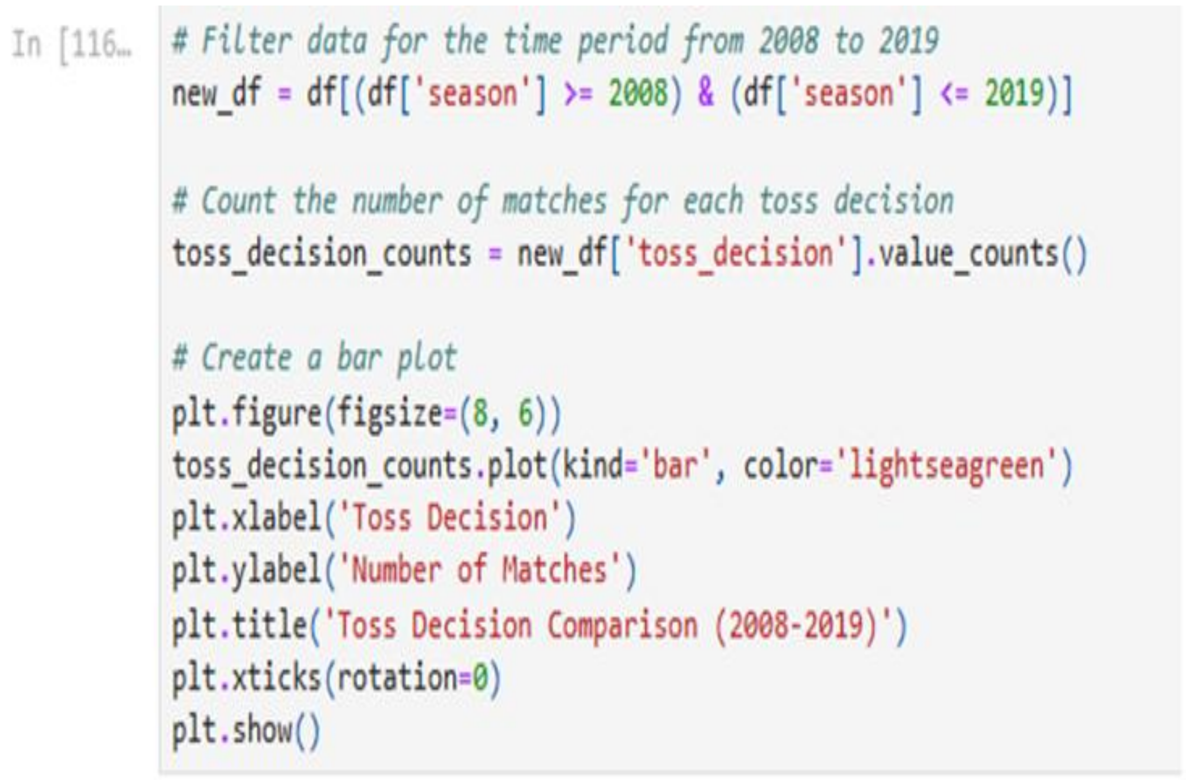
Closure: The pie chart as a whole represents 100% of the data, and the sectors add up to 100%.

10. Perform a Side by Side Visualization for the Toss Decisions, Comparing the Decision to Field or Bat Across the Period 2008-2019

o perform a side-by-side visualization for toss decisions (field or bat) across the period 2008-2019, can create a grouped bar chart. This chart will allow to compare the distribution of toss decisions for each season in that time frame. Here's how can do it:

Data Visualization Steps:

1. Filter the data to include matches from 2008 to 2019.
2. Group the data by season and toss decision to count the occurrences of each decision.
3. Create a grouped bar chart to visualize the distribution of toss decisions for each season.



```
In [116…
# Filter data for the time period from 2008 to 2019
new_df = df[(df['season'] >= 2008) & (df['season'] <= 2019)]

# Count the number of matches for each toss decision
toss_decision_counts = new_df['toss_decision'].value_counts()

# Create a bar plot
plt.figure(figsize=(8, 6))
toss_decision_counts.plot(kind='bar', color='lightseagreen')
plt.xlabel('Toss Decision')
plt.ylabel('Number of Matches')
plt.title('Toss Decision Comparison (2008-2019)')
plt.xticks(rotation=0)
plt.show()
```

This code filters the data, groups it by season and toss decision, and then creates a grouped bar chart that compares the distribution of toss decisions (bat or field) for each season from 2008 to 2019. The chart provides insights into how teams chose to bat or field during that time period.

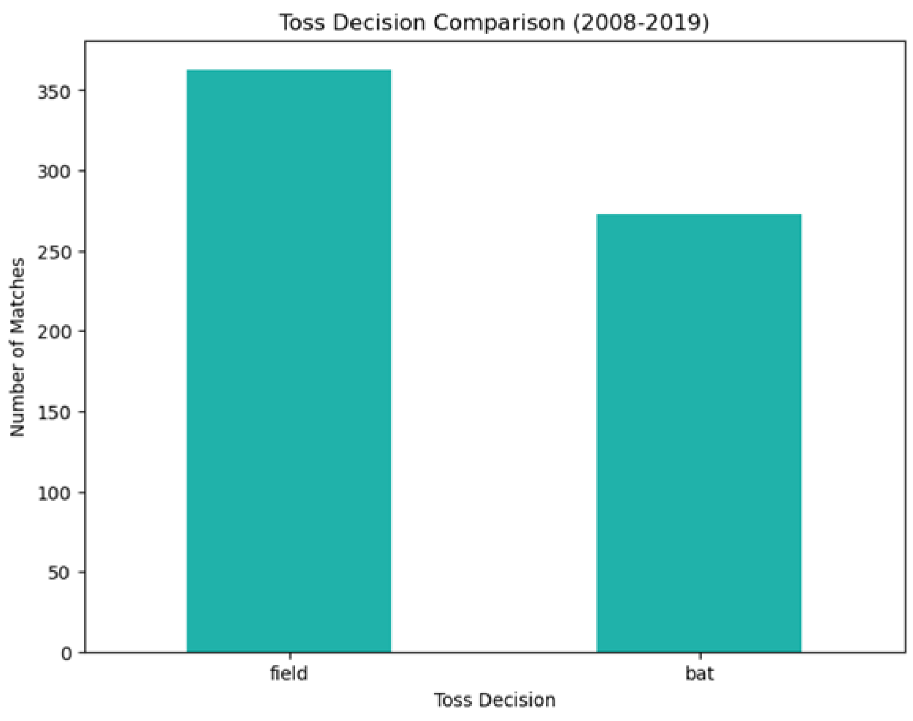


Figure 3:

Findings:

The bar plot compares the number of matches where teams chose to bat or field after winning the toss from 2008 to 2019. It provides information that most time toss winning teams as opted to field first

Reason for Chart Type Selection:

A bar plot is suitable for comparing the counts of different categories (toss decisions) in a side-by-side manner.

Pre-attentive Attributes Used:

Length: The length of the bars represents the number of matches for each toss decision.

Position: The position of each bar on the x-axis helps in comparing the two toss decisions.

Gestalt Principles Used:

Proximity: The two bars are placed next to each other on the x-axis, making it easy to compare them.

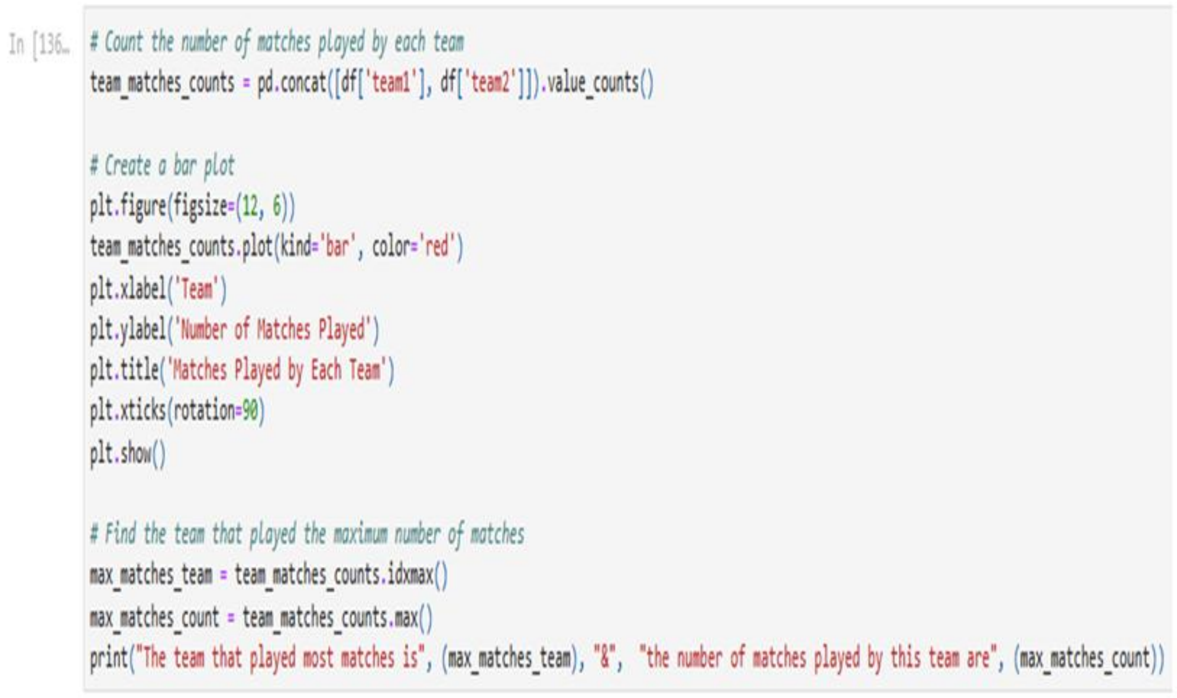
Similarity: The bars share a similar visual attribute (color), indicating

11. Visualize the Matches Played by Each Team and Find Out which Team Played the Maximum Number of Matches in IPL

To visualize the matches played by each team and find out which team played the maximum number of matches in the IPL, can create a bar chart or similar visualization. Here are the steps and code to do this:

Data Visualization Steps:

1. Calculate the total number of matches played by each team.
2. Visualize the number of matches for each team.
3. Identify the team that played the maximum number of matches.



```
In [136…  # Count the number of matches played by each team
team_matches_counts = pd.concat([df['team1'], df['team2']]).value_counts()

# Create a bar plot
plt.figure(figsize=(12, 6))
team_matches_counts.plot(kind='bar', color='red')
plt.xlabel('Team')
plt.ylabel('Number of Matches Played')
plt.title('Matches Played by Each Team')
plt.xticks(rotation=90)
plt.show()

# Find the team that played the maximum number of matches
max_matches_team = team_matches_counts.idxmax()
max_matches_count = team_matches_counts.max()
print("The team that played most matches is", (max_matches_team), "&",  "the number of matches played by this team are", (max_matches_count))
```

This code will calculate and visualize the number of matches played by each team and also identify the team that played the maximum number of matches in the IPL. The bar plot provides a clear overview of team participation in the league.

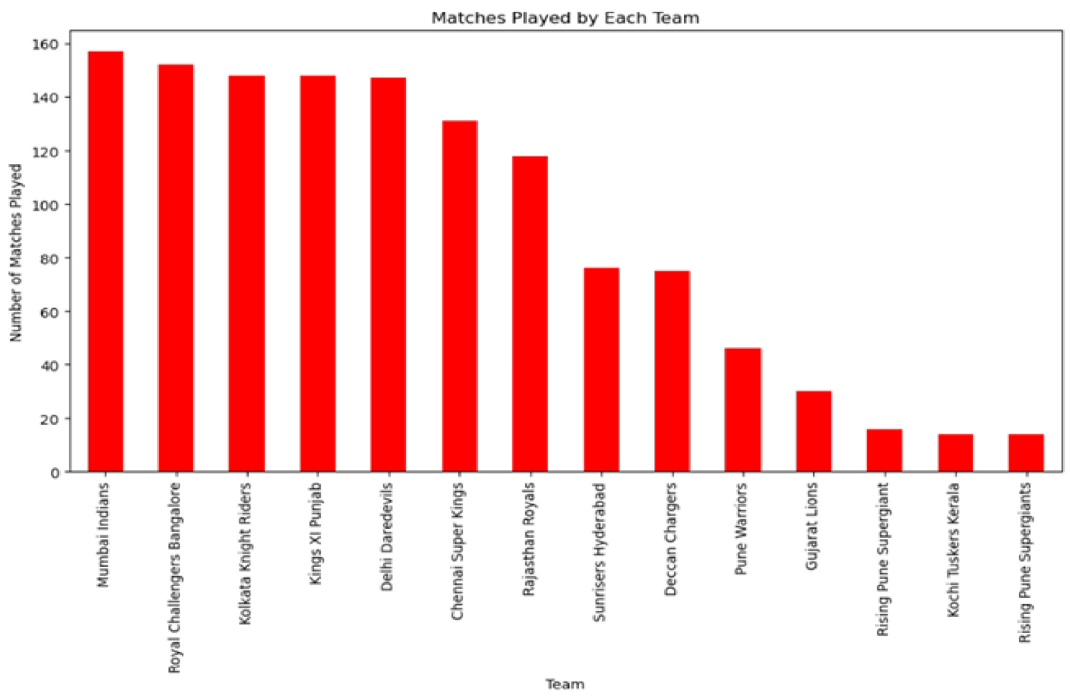


Figure 4:

The team that played most matches is Mumbai Indians & the number of matches played by this team are 157

Findings:

The bar plot shows the number of matches played by each team in the IPL. The team that played the maximum number of matches in the IPL is 'Mumbai Indians' with a count of 157 matches. This indicates that Mumbai India team have enter into final more times as compare to other teams

Reason for Chart Type Selection:

A bar plot is appropriate for comparing the number of matches played by each team as it effectively displays the counts for different categories (teams).

Pre-attentive Attributes Used:

Length: The length of the bars represents the number of matches played by each team.

Position: The position of each bar on the x-axis helps in comparing the teams.

Gestalt Principles Used:

Proximity: The bars are spaced apart, making it easy to distinguish between different teams.

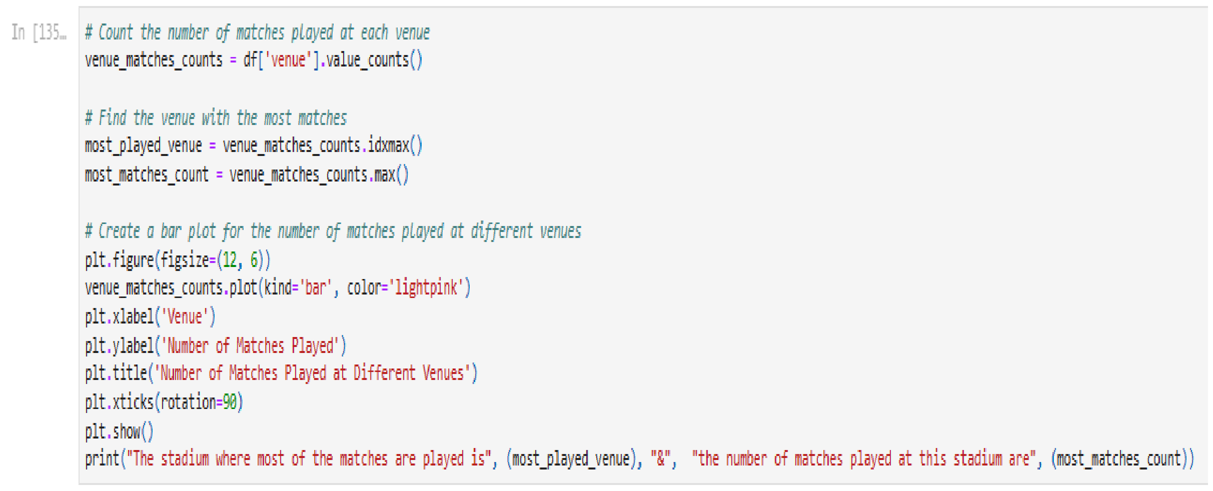
Similarity: The bars share a similar visual attribute (color), indicating that they belong to the same category (number of matches played).

12. Find Out Which Venue had Most Number of Matches Played

To find out which venue had the most number of matches played and to verify this by plotting the number of matches played at different venues, can follow these steps:

Data Exploration Steps:

1. Calculate the total number of matches played at each venue.
2. Find the venue with the most matches played.
3. Create a bar chart to visualize the number of matches played at different venues.



```
In [135…
# Count the number of matches played at each venue
venue_matches_counts = df['venue'].value_counts()

# Find the venue with the most matches
most_played_venue = venue_matches_counts.idxmax()
most_matches_count = venue_matches_counts.max()

# Create a bar plot for the number of matches played at different venues
plt.figure(figsize=(12, 6))
venue_matches_counts.plot(kind='bar', color='lightpink')
plt.xlabel('Venue')
plt.ylabel('Number of Matches Played')
plt.title('Number of Matches Played at Different Venues')
plt.xticks(rotation=90)
plt.show()
print("The stadium where most of the matches are played is", (most_played_venue), "&",  "the number of matches played at this stadium are", (most_matches_count))
```

This code calculates the number of matches played at each venue, identifies the venue with the most matches, and visualizes the number of matches played at different venues using a bar chart. It helps find and confirm the venue with the most matches played in the IPL.

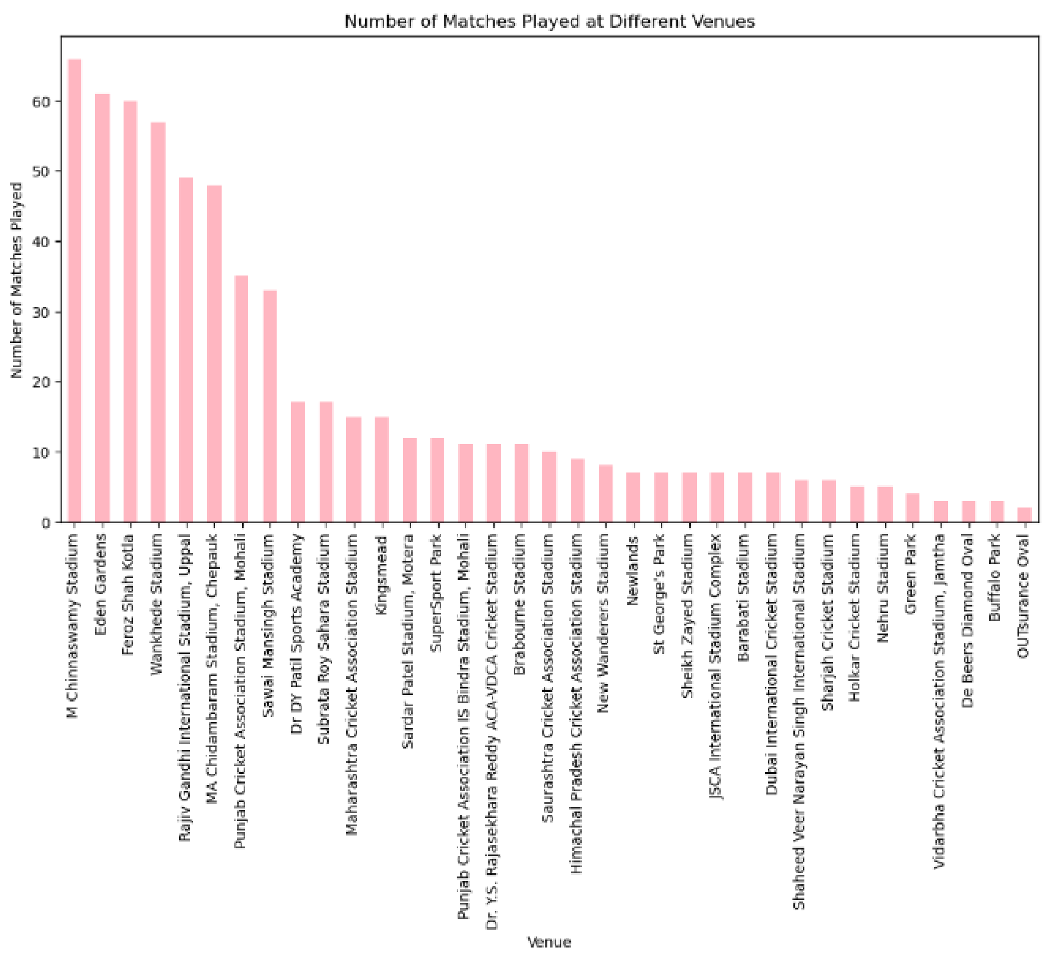


Figure 5:

The stadium where most of the matches are played is M Chinnaswamy Stadium & the number of matches played at this stadium are 66

Findings:

The bar plot displays the number of matches played at different venues in the IPL. The venue that had the most number of matches played is 'M Chinnaswamy' with a count of 66 matches.

Reason for Chart Type Selection:

A bar plot is suitable for comparing the number of matches played at different venues, as it effectively displays the counts for each venue.

Pre-attentive Attributes Used:

Length: The length of the bars represents the number of matches played at each venue.

Position: The position of each bar on the x-axis helps in comparing the venues.

Gestalt Principles Used:

Proximity: The bars are spaced apart, making it easy to distinguish between different venues.

Similarity: The bars share a similar visual attribute (color), indicating that they belong to the same category (number of matches played).

13. Distribution of Match Results (Normal, Tie, No Result) in the IPL Dataset, and how has it Evolved Over the Years

To analyze the distribution of match results (normal, tie, no result) in the IPL dataset and how it has evolved over the years, can follow these steps:

Data Analysis Steps:

1. Calculate the distribution of match results for each season.
2. Visualize the distribution of match results over the years using a line chart.

```
In [15]: result_counts = df.groupby(['season', 'result']).size().unstack(fill_value=0)

         # Create a stacked bar plot
         result_counts.plot(kind='bar', stacked=True, colormap='coolwarm', figsize=(12, 6))
         plt.xlabel('Season')
         plt.ylabel('Number of Matches')
         plt.title('Distribution of Match Results in IPL Over the Years')
         plt.xticks(rotation=45)
         plt.legend(title='Match Result', loc='upper left')
         plt.show()
```

This code calculates and visualizes the distribution of match results (normal, tie, no result) over the years in the IPL dataset. The line chart helps see how the distribution has evolved season by season. Need to adjust the **df['season'] >= 2008** condition based on the range of seasons want to analyze.

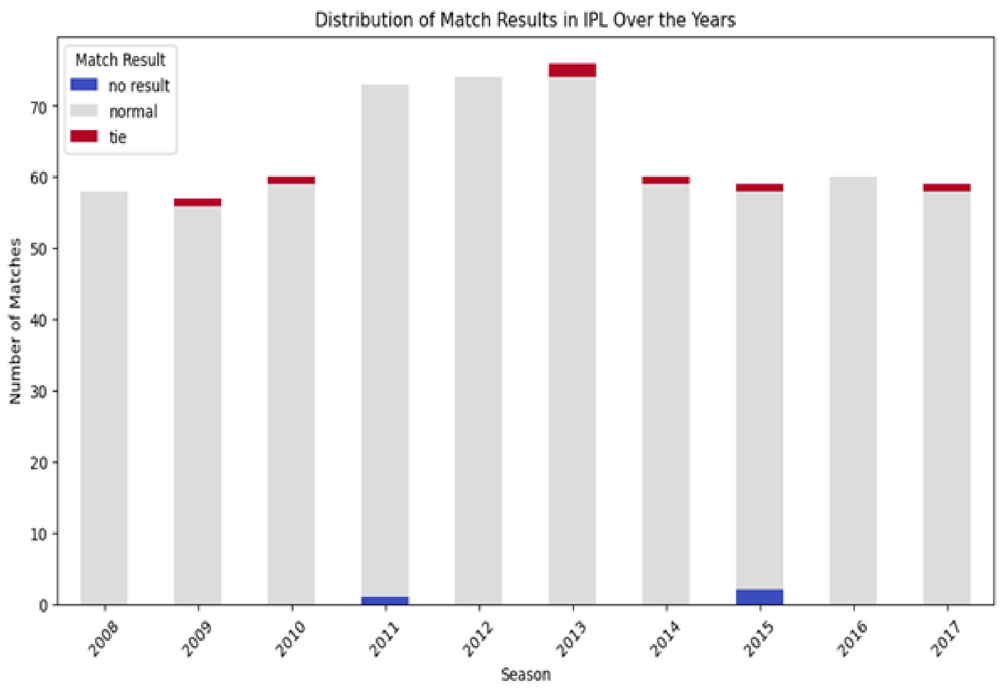


Figure 6:

Findings:

The stacked bar plot visualizes the distribution of match results (normal, tie, no result) in the IPL dataset for each season from the available data. It helps us see how the distribution of match results has changed over the years.

Reason for Chart Type Selection:

A stacked bar plot is suitable for visualizing the distribution of multiple categories (match results) within each season. It allows us to see the total number of matches in each season and how they are divided among different result types.

Pre-Attentive Attributes Used:

Length: The length of each segment of the stacked bars represents the number of matches with a specific result type.

Color: Different colors are used to distinguish between the different result types.

Gestalt Principles Used:

Proximity: Segments of the same stacked bar are close together, indicating that they belong to the same season.

Similarity: Similar colors are used for segments of the same result type, making it easy to identify which result type corresponds to which color.

Summary

This analysis explores the dynamics of match results in the Indian Premier League (IPL) dataset, spanning multiple seasons. The focus is on the distribution of three key match outcomes: normal wins, tied matches, and matches with no result. By leveraging a stacked bar plot, the study presents a visual journey through the evolution of match results in the IPL. The choice of a stacked bar plot as the visualization method proves effective, as it enables a clear representation of the number of matches associated with each result category for every IPL season. The utilization of pre-attentive attributes, such as the length and color of the bars, enhances the visual impact of the chart. Moreover, adherence to Gestalt principles, notably proximity and similarity, helps viewers in recognizing patterns and variations in match outcomes over time.

This analysis provides valuable insights into how match results have shifted over the years in the IPL. The visualization aids in understanding the changing landscape of cricket outcomes, making it a valuable resource for IPL enthusiasts and data analysts seeking to comprehend the tournament's historical performance trends.