

|  |  |
| --- | --- |
| Project Title | **Analyzing IPL Match Data (2008 - 2023)** |
| Languages | Machine learning, python, MYSQL, Excel |
| Tools | Visual Studio code / Jupyter notebook |
| Domain | Data Analyst |
| Project Difficulties level | Advance |

Dataset: Dataset is available in the given link. You can download it at your convenience.

[Click here to download data set](https://drive.google.com/file/d/1sN5FONb4O73IHc3h4fCEyY1lsxvwQP49/view?usp=drivesdk)

**About Dataset**

This dataset provides an extensive analysis of Indian Premier League (IPL) matches, offering valuable insights into team performance, match outcomes, and tournament trends from 2008 to 2023. It captures key attributes such as match season, participating teams, venue, toss details, match result, winning margin, and the player of the match. By analyzing over 1000 matches, the project uncovers patterns related to the most successful teams, dominant venues, impact of toss decisions, and performance consistency across seasons. This analysis not only aids cricket enthusiasts in understanding the dynamics of the IPL but also serves as a strong foundation for predictive modeling and data-driven sports strategies.

**Columns include:**

* **id**: A unique identifier for every match played in the IPL.
* **season**: The year in which the IPL match was held, ranging from 2008 to 2023.
* **city**: The city where the match was played, offering geographical context for venue-based performance analysis.
* **date**: The calendar date on which the match occurred.
* **team1**: The first competing team, representing one side of the match-up.
* **team2**: The second competing team, opposing team1.
* **toss\_winner**: The team that won the toss before the start of the match.
* **toss\_decision**: The decision made by the toss winner—either to bat or to field first.
* **result**: The outcome type of the match, such as a normal win, tie, or no result.
* **dl\_applied**: A binary indicator (1 or 0) specifying whether the Duckworth-Lewis (D/L) method was applied due to weather interruptions.
* **winner**: The team that won the match.
* **win\_by\_runs**: Indicates the margin of victory in terms of runs (non-zero if the team batting first won).
* **win\_by\_wickets**: Indicates the margin of victory in terms of wickets (non-zero if the team chasing won).
* **player\_of\_match**: The player awarded as "Man of the Match" for outstanding performance.
* **venue**: The stadium where the match was played, which can impact team performance.
* **umpire1**: The name of the first on-field umpire officiating the match.
* **umpire2**: The name of the second on-field umpire.
* **umpire3**: The name of the third (TV) umpire; may contain missing values for older matches.

**Major Machine Learning Project: Analyzing IPL Match Data (2008 - 2023)**

This project aims to analyze IPL match performance by leveraging historical match data and applying Machine Learning techniques to uncover trends, patterns, and winning strategies. The focus is on Exploratory Data Analysis (EDA), interactive data visualizations, and building a predictive model to forecast match winners based on pre-match features such as teams, venue, toss decisions, and season. Ultimately, the project provides actionable insights into team consistency, the impact of toss and venue, and historical dominance, helping analysts and cricket enthusiasts better understand the dynamics that contribute to success in the Indian Premier League.

# Step-by-Step Workflow

1. **Import Libraries**

python code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.ticker as mticker

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

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.preprocessing import LabelEncoder

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

from sklearn.ensemble import RandomForestClassifier

# Load and Explore the Dataset

python code

# Load the dataset

data = pd.read\_csv('D:/Project/Data Analysis

Project/Analyzing IPL Match Data/IPL\_Match\_(2008-23)\_data.csv')

# Preview the dataset data.head()

# Display basic information about the dataset

print(data.info())

# Display Index

print(data.index)

# Display Columns

print(data.columns)

# Check shape

print(data.shape)

# Check for null values print(data.isnull().sum())

# Data Cleaning Handle Missing Values:

python code

# Drop null values, Drop rows with missing values (for simplicity)

data.dropna(inplace=True)

4. Exploratory Data Analysis (EDA) Analyze relationships:

● Correlation Heatmap:

python code

numeric\_data = data [['dl\_applied', 'win\_by\_runs', 'win\_by\_wickets']]

correlation\_matrix = numeric\_data.corr()

plt.figure(figsize = (15, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='YlGnBu', fmt='.2f', linewidths=0.5, square=True)

plt.title("Heatmap of IPL Numeric Features", fontsize=16)

plt.tight\_layout()

plt.show()

● Toss Decision vs Match Outcome (2008 - 23):

python code

# # Create a new column indicating if toss winner also won the match

# data['toss\_win\_and\_match\_win'] = data['toss\_winner'] == data['winner']

# # Group by toss decision and match outcome (True/False)

toss\_impact = data.groupby('toss\_decision')['toss\_win\_and\_match\_win'].value\_counts().unstack().fillna(0)

# # Rename columns for clarity

toss\_impact.columns = ['Lost Match', 'Won Match']

# toss\_impact.plot(kind='bar', stacked=True, figsize=(15, 8), color=['Red', 'green'])

plt.title("Toss Decision vs Match Outcome (2008-23)", fontsize=16) plt.xlabel("Toss Decision", fontsize=12)

plt.ylabel("Number of Matches", fontsize=12) plt.xticks(rotation=0)

plt.legend(title="Match Outcome", loc="upper right") plt.tight\_layout()

plt.show()

5. Feature Engineering

Create new features:

python code

# # Count number of matches per city (excluding NaNs)

city\_counts = data['city'].value\_counts()

# # Select only numeric columns

numeric\_data = data[['dl\_applied', 'win\_by\_runs', 'win\_by\_wickets']]

**6. Data Visualization** win\_by\_runs Vs win\_by\_wickets**:**

python code

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# # Plot 1: Distribution of win\_by\_runs (Defending teams)

sns.histplot(data[data['win\_by\_runs'] > 0]['win\_by\_runs'], bins=25, kde=True, ax=axes[0], color='brown') axes[0].set\_title("Distribution of Win Margins (By Runs)", fontsize=16) axes[0].set\_xlabel("Win By Runs", fontsize=12) axes[0].set\_ylabel("Number of Matches", fontsize=12)

# # Plot 2: Distribution of win\_by\_wickets (Chasing teams)

sns.histplot(data[data['win\_by\_wickets'] > 0]['win\_by\_wickets'], bins=25, kde=True, ax=axes[1], color='orange') axes[1].set\_title("Distribution of Win Margins (By Wickets)", fontsize=16)

axes[1].set\_xlabel("Win By Wickets", fontsize=12) axes[1].set\_ylabel("Number of Matches", fontsize=12)

plt.tight\_layout()

plt.show()

● Number of Wins by Teams (2008 - 23):

python code

# # Count number of wins per team in decending order

# team\_wins = data['winner'].value\_counts().sort\_values(ascending=False)

# # Plotting

plt.figure(figsize=(15, 8))

ax = sns.barplot(x=team\_wins.values, y=team\_wins.index, palette="viridis")

plt.title("Number of Wins by Teams (2008-23)", fontsize=16) plt.xlabel("Number of Wins", fontsize=12)

plt.ylabel("Teams", fontsize=12)

# # Add value labels on each bar

for i, value in enumerate(team\_wins.values): ax.text(value + 1, i, str(value), va='center')

plt.tight\_layout()

plt.show()

● IPL Matches Played Per City (2008 - 23):

python code

# # Count number of matches per city (excluding NaNs)

city\_counts = data['city'].value\_counts()

# # Plotting line graph

plt.figure(figsize=(15, 6)) plt.plot(city\_counts.index, city\_counts.values, marker='o', linestyle='-', color='teal')

# # Add data labels to each point

for i, value in enumerate(city\_counts.values):

plt.text(i, value + 1, str(int(value)), ha='center', fontsize=10)

# # Formatting

plt.title("IPL Matches Played Per City (2008-23)", fontsize=16) plt.xlabel("City", fontsize=12) plt.ylabel("Number of Matches", fontsize=12)

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

plt.tight\_layout()

plt.show()

● Top 10 Players of the Match in IPL (2008 - 23):

python code

# # Count Player of the Match awards

top\_players = data['player\_of\_match'].value\_counts().head(10)

# # Plotting

plt.figure(figsize=(15, 7))

ax = sns.barplot(x=top\_players.values, y=top\_players.index, palette="magma", alpha=0.9)

# # Add count labels

for i, value in enumerate(top\_players.values):

plt.text(value + 0.5, i, str(int(value)), va='center')

# # Titles and labels

plt.title("Top 10 Players of the Match in IPL (2008-23)", fontsize=16)

plt.xlabel("Number of Awards", fontsize=12)

plt.ylabel("Player", fontsize=12)

plt.tight\_layout()

plt.show()

● IPL Matches Distribution Per Season (2008-23):

python code

# # Count matches per season

season\_counts = data['season'].value\_counts().sort\_index()

# # Plotting the pie chart

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

plt.pie(season\_counts, labels=season\_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.tab20.colors, wedgeprops={'edgecolor': 'white'})

# # Title

plt.title("IPL Matches Distribution Per Season (2008-23)", fontsize=16)

plt.tight\_layout()

plt.show()

# 7. Predictive Model: Estimate Winning Prepare Data:

python code

# Drop unnecessary columns

data.drop(columns=['id', 'umpire1', 'umpire2', 'umpire3', 'player\_of\_match', 'win\_by\_runs', 'win\_by\_wickets'], inplace=True)

# Drop rows with missing values

data.dropna(subset=['winner', 'team1', 'team2', 'toss\_winner', 'venue', 'city'], inplace=True)

# Convert 'date' to datetime and extract date parts

data['date'] = pd.to\_datetime(data['date'], format='%d-%m-%Y', errors='coerce')

data['year'] = data['date'].dt.year

data['month'] = data['date'].dt.month

data['day'] = data['date'].dt.day

data.drop(columns=['date'], inplace=True)

Encode categorical features label\_cols = ['season', 'city', 'team1', 'team2', 'toss\_winner', 'toss\_decision', 'result', 'venue', 'winner'] label\_encoders = {}

for col in label\_cols:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le

# Define features and target

X = data.drop(columns=['winner'])

y = data['winner']

Split into train and test sets

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

● Train Random Forest Regressor:

python code

model = RandomForestClassifier(n\_estimators=100, random\_state=42) model.fit(X\_train, y\_train)

# Predict on test data

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

● Evaluate the Model:

python code

# Calculate performance metrics

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

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

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R² Score: {r2}")

# 8. Insights and Recommendations

Use visualizations and feature importance to derive insights: python

code

# Feature Importance

importances = model.feature\_importances\_

feature\_importance\_df = pd.DataFrame({'Feature': features, 'Importance': importances})

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance',ascending = False)

# Plotting feature importance

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

sns.barplot(x='Importance', y='Feature', data=feature\_importance\_df)

plt.title("Feature Importance for Predicting Winning Team")

plt.show()

9. Deployment and Presentation

* + Summarize findings:
    - Highlight Top Performing Teams.
    - Identify Key Influencing Factors(e.g., venue, toss decisions, and batting second).
    - Spot Winning Patterns.

Export model:

python code

import joblib

joblib.dump(model, 'ipl\_match\_winner\_predictor.pkl')

Sample code with output

In the world of YouTube, where number of views, uploads and category are matter to increase the subscribers. This dataset offers a treasure trove of insights into YouTube channel analytics. Let's dive into the data and see what stories it has to tell.

In [1]:

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.ticker as mticker

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

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.preprocessing import LabelEncoder

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

from sklearn.ensemble import RandomForestClassifier

In [2]:

*# Load the dataset*

data = pd.read\_csv(' D:/Project/Data Analysis

Project/Analyzing IPL Match Data/IPL\_Match\_(2008-23)\_data.csv')

data.head()

Out [2]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 1370353 | 2023 | Ahmedabad | 29-05-2023 | Gujarat Titans | Chennai Super Kings | Chennai Super Kings | field | D/L | 1 | Chennai Super Kings | 0 | 5 | DP Conway | Narendra Modi Stadium, Ahmedabad | Nitin Menon | RJ Tucker | KN Ananthapadmanabhan |
| 1370352 | 2023 | Ahmedabad | 26-05-2023 | Gujarat Titans | Mumbai Indians | Mumbai Indians | field | normal | 0 | Gujarat Titans | 62 | 0 | Shubman Gill | Narendra Modi Stadium, Ahmedabad | Nitin Menon | RJ Tucker | J Madanagopal |
| 1370351 | 2023 | Chennai | 24-05-2023 | Mumbai Indians | Lucknow Super Giants | Mumbai Indians | bat | normal | 0 | Mumbai Indians | 81 | 0 | Akash Madhwal | MA Chidambaram Stadium, Chepauk, Chennai | BNJ Oxenford | VK Sharma | CB Gaffaney |
| 1370350 | 2023 | Chennai | 23-05-2023 | Chennai Super Kings | Gujarat Titans | Gujarat Titans | field | normal | 0 | Chennai Super Kings | 15 | 0 | RD Gaikwad | MA Chidambaram Stadium, Chepauk, Chennai | AK Chaudhary | CB Gaffaney | BNJ Oxenford |
| 1359543 | 2023 | Mumbai | 21-05-2023 | Sunrisers Hyderabad | Mumbai Indians | Mumbai Indians | field | normal | 0 | Mumbai Indians | 0 | 8 | C Green | Wankhede Stadium, Mumbai | KN Ananthapadmanabhan | RJ Tucker | R Pandit |

Data Overview

Let's look at the basic information about the dataset to understand its structure and contents.

In [3]:

data.info()

Out [3]:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1024 entries, 0 to 1023

Data columns (total 18 columns):

# # Column Non-Null Count Dtype

0 id 1024 non-null int64

1 season 1024 non-null object

2 city 973 non-null object

3 date 1024 non-null object

4 team1 1024 non-null object

5 team2 1024 non-null object

6 toss\_winner 1024 non-null object

7 toss\_decision 1024 non-null object

8 result 1024 non-null object

9 dl\_applied 1024 non-null int64

10 winner 1005 non-null object

11 win\_by\_runs 1024 non-null int64

12 win\_by\_wickets 1024 non-null int64

13 player\_of\_match 1019 non-null object

14 venue 1024 non-null object

15 umpire1 1024 non-null object

16 umpire2 1024 non-null object

17 umpire3 1021 non-null object

dtypes: int64(4), object(14) memory usage: 144.1+ KB

Data Cleaning and Preprocessing

Before diving into analysis, let's ensure the data is clean and ready for exploration.

In [4]:

*# Check for missing values*

data. dropna(inplace=True)

**Exploratory Data Analysis**

Let's explore the data to uncover patterns and insights.

In [5]:

*#* Correlation Heatmap:

plt.figure(figsize = (15, 6))

*# Compute the correlation matrix*

numeric\_data = data [['dl\_applied', 'win\_by\_runs', 'win\_by\_wickets']]

correlation\_matrix = numeric\_data.corr()

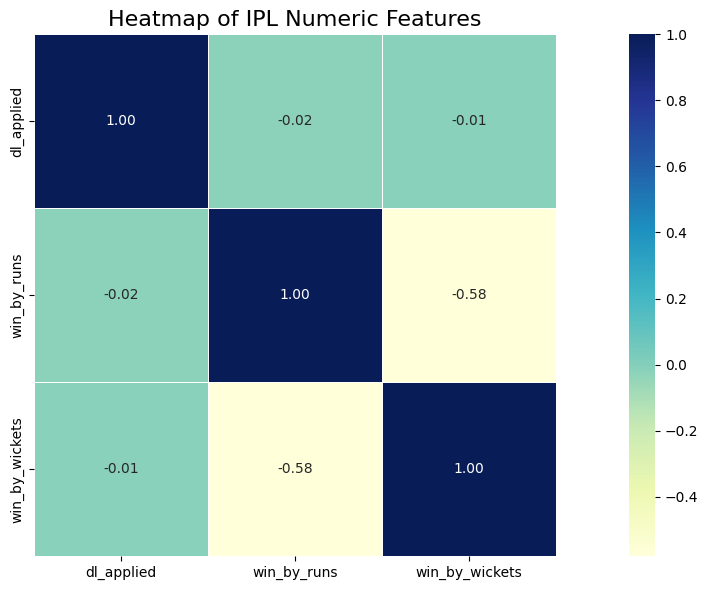
*# Plot the heatmap*

sns.heatmap(correlation\_matrix, annot = True, fmt = ".2f", cmap = "YlGnBu", linewidths = 0.5, square = True,)

plt.title("Heatmap of IPL Numeric Features", fontsize = 16)

plt.show()

Out [5]:



In [6]:

***# Toss Decision vs Match Outcome (2008-23):***# Create a new column indicating if toss winner also won the match

data['toss\_win\_and\_match\_win'] = data['toss\_winner'] == data['winner']

# # Group by toss decision and match outcome (True/False)

toss\_impact = data.groupby('toss\_decision')['toss\_win\_and\_match\_win'].value\_counts().unstack().fillna(0)

toss\_impact.columns = ['Lost Match', 'Won Match']

# # Plotting

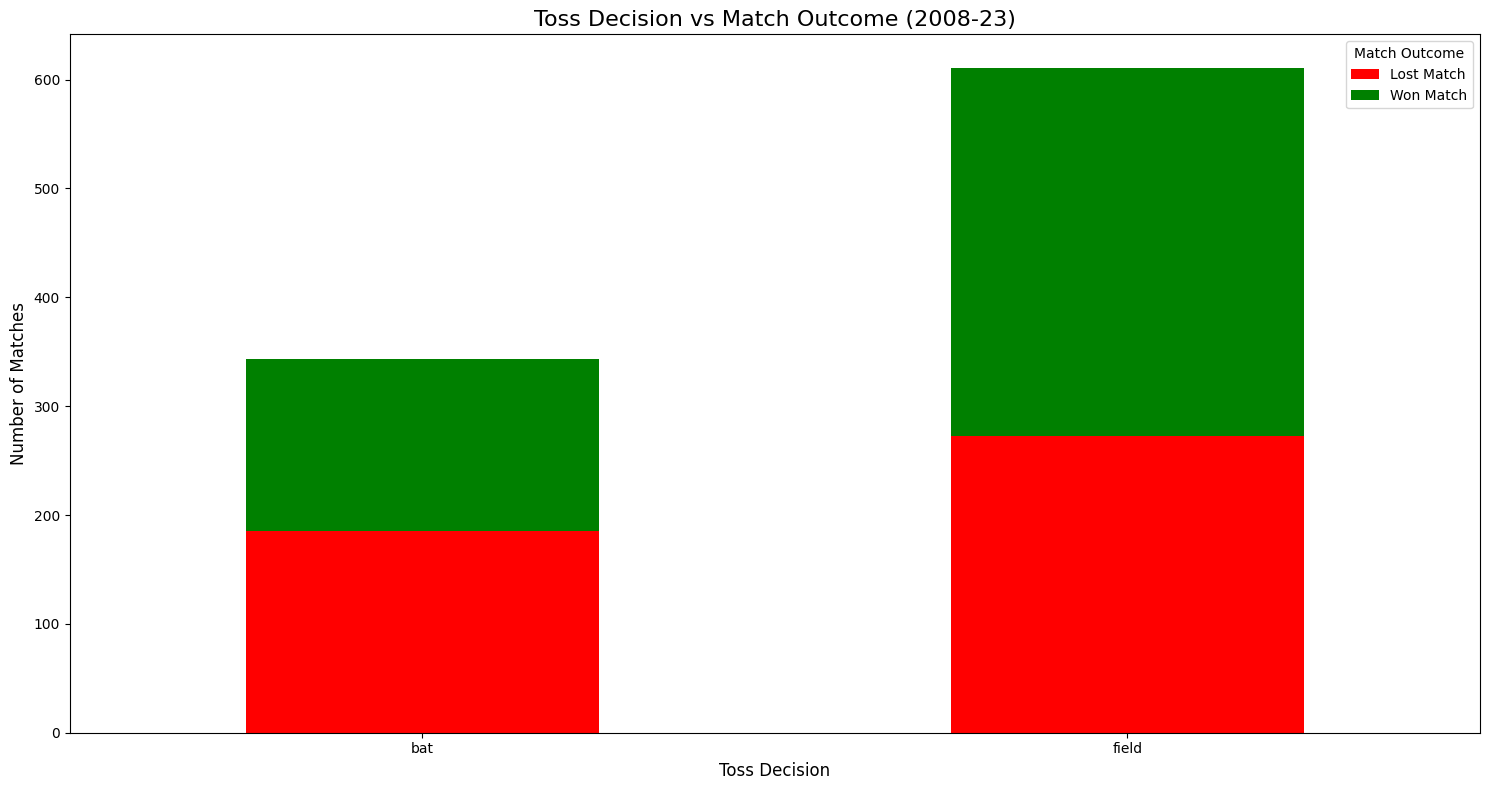
toss\_impact.plot(kind='bar', stacked=True, figsize=(15, 8), color=['Red', 'green'])

plt.title("Toss Decision vs Match Outcome (2008-23)", fontsize=16) plt.xlabel("Toss Decision", fontsize=12) plt.ylabel("Number of Matches", fontsize=12) plt.xticks(rotation=0)

plt.legend(title="Match Outcome", loc="upper right") plt.tight\_layout()

plt.show()

Out [6]:



In [7]:

***# Distribution of win\_by\_runs and win\_by\_wickets:***

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

sns.histplot(data[data['win\_by\_runs'] > 0]['win\_by\_runs'], bins=25, kde=True, ax=axes[0], color='brown') axes[0].set\_title("Distribution of Win Margins (By Runs)", fontsize=16)

axes[0].set\_xlabel("Win By Runs", fontsize=12) axes[0].set\_ylabel("Number of Matches", fontsize=12)

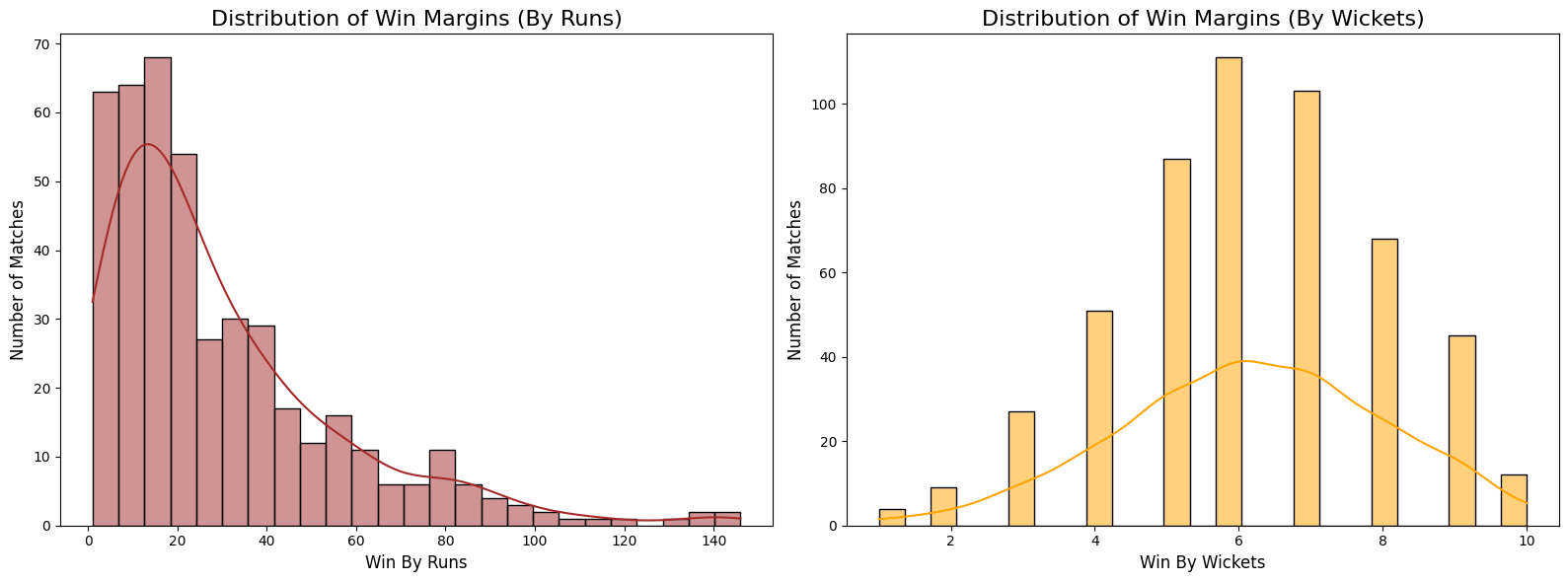
sns.histplot(data[data['win\_by\_wickets'] > 0]['win\_by\_wickets'], bins=25, kde=True, ax=axes[1], color='orange') axes[1].set\_title("Distribution of Win Margins (By Wickets)", fontsize=16)

axes[1].set\_xlabel("Win By Wickets", fontsize=12) axes[1].set\_ylabel("Number of Matches", fontsize=12)

plt.tight\_layout()

plt.show()

Out [7]:



In [8]:

***# Number of Wins by Teams (2008-23):***

team\_wins = data['winner'].value\_counts().sort\_values(ascending=False)

plt.figure(figsize=(15, 8)) ax = sns.barplot(x=team\_wins.values, y=team\_wins.index, palette="viridis")

plt.title("Number of Wins by Teams (2008-23)", fontsize=16) plt.xlabel("Number of Wins", fontsize=12) plt.ylabel("Teams", fontsize=12)

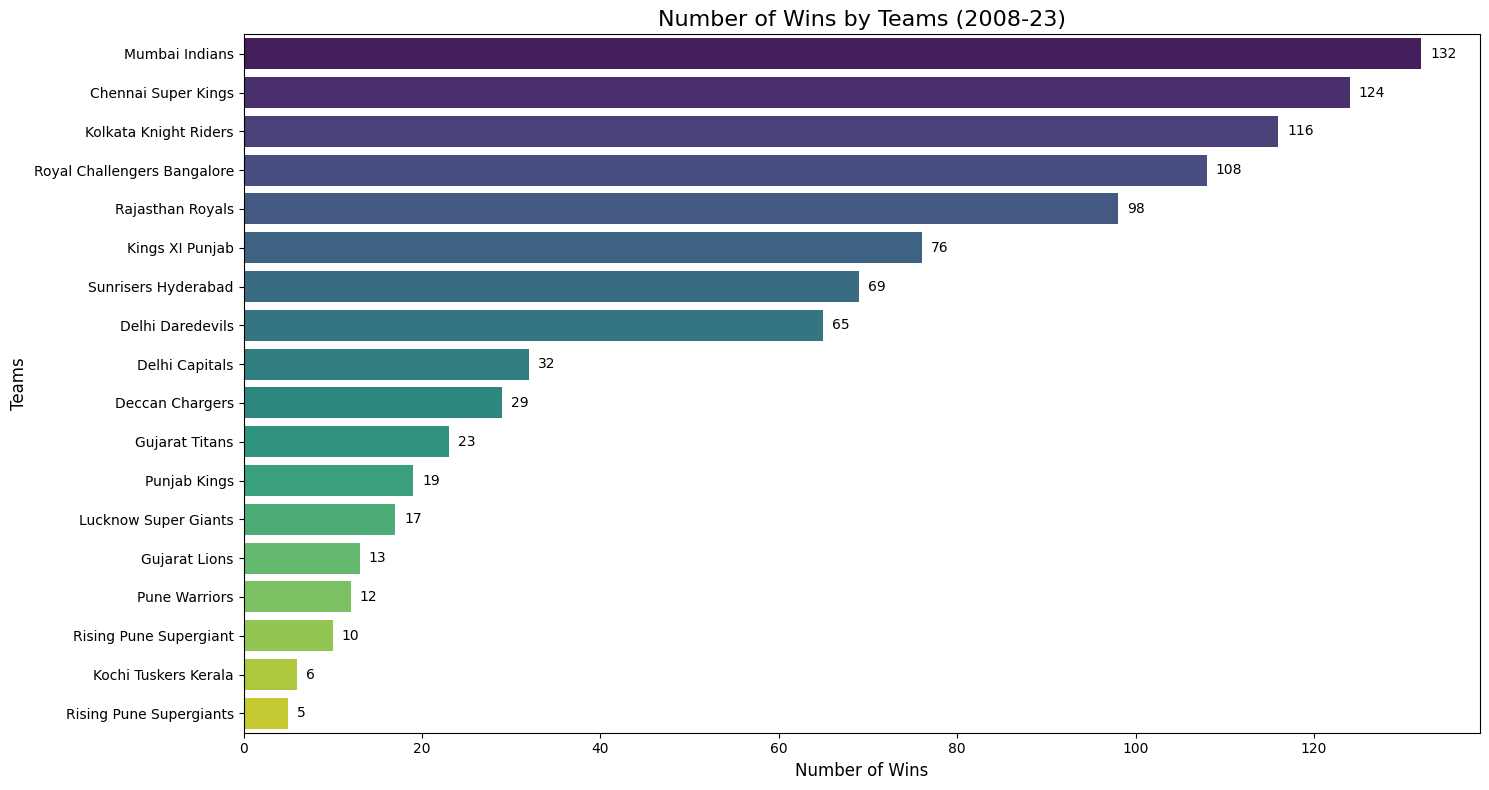
for i, value in enumerate(team\_wins.values):

ax.text(value + 1, i, str(value), va='center')

plt.tight\_layout()

plt.show()

Out [8]:



In [9]:

***# IPL Matches Played Per City (2008-23):***

city\_counts = data['city'].value\_counts()

plt.figure(figsize=(15, 6)) plt.plot(city\_counts.index, city\_counts.values, marker='o', linestyle='-', color='teal')

for i, value in enumerate(city\_counts.values):

plt.text(i, value + 1, str(int(value)), ha='center', fontsize=10)

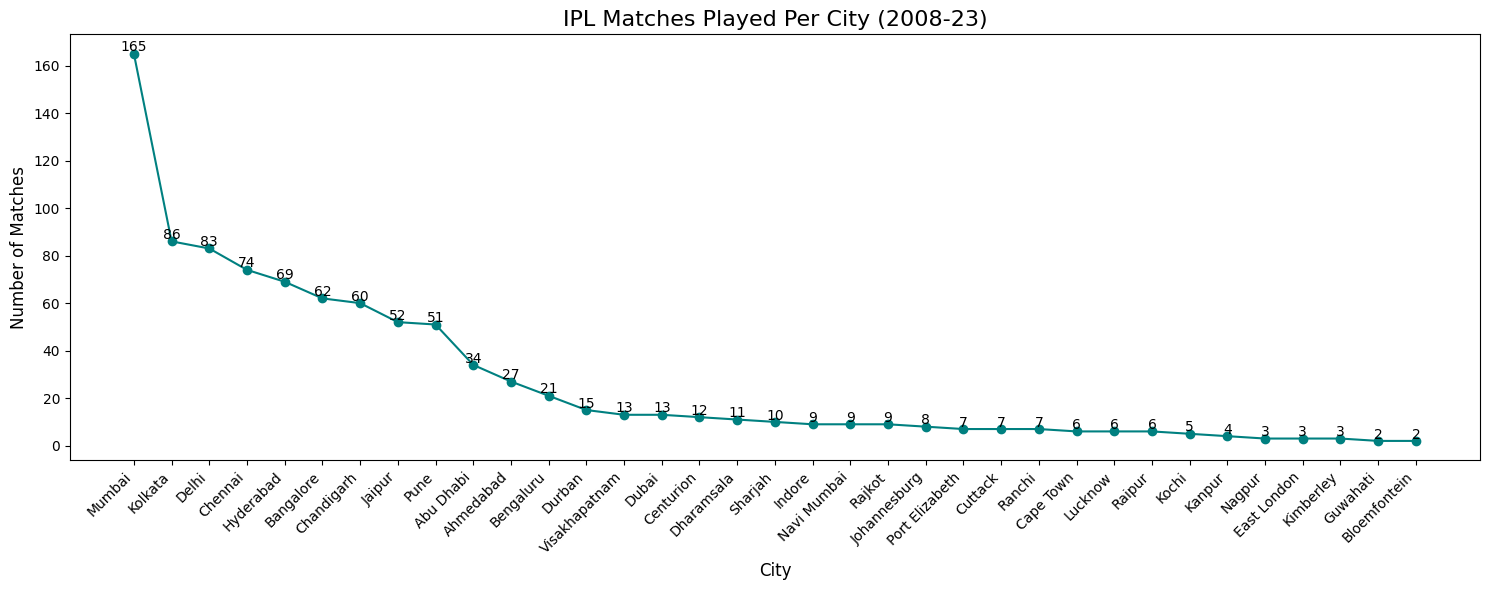
plt.title("IPL Matches Played Per City (2008-23)", fontsize=16) plt.xlabel("City", fontsize=12) plt.ylabel("Number of Matches", fontsize=12)

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

plt.tight\_layout()

plt.show()

Out [10]:



In [11]:

***# Top 10 Players of the Match in IPL (2008-23):***

top\_players = data['player\_of\_match'].value\_counts().head(10)

plt.figure(figsize=(15, 7)) ax = sns.barplot(x=top\_players.values, y=top\_players.index, palette="magma", alpha=0.9)

for i, value in enumerate(top\_players.values):

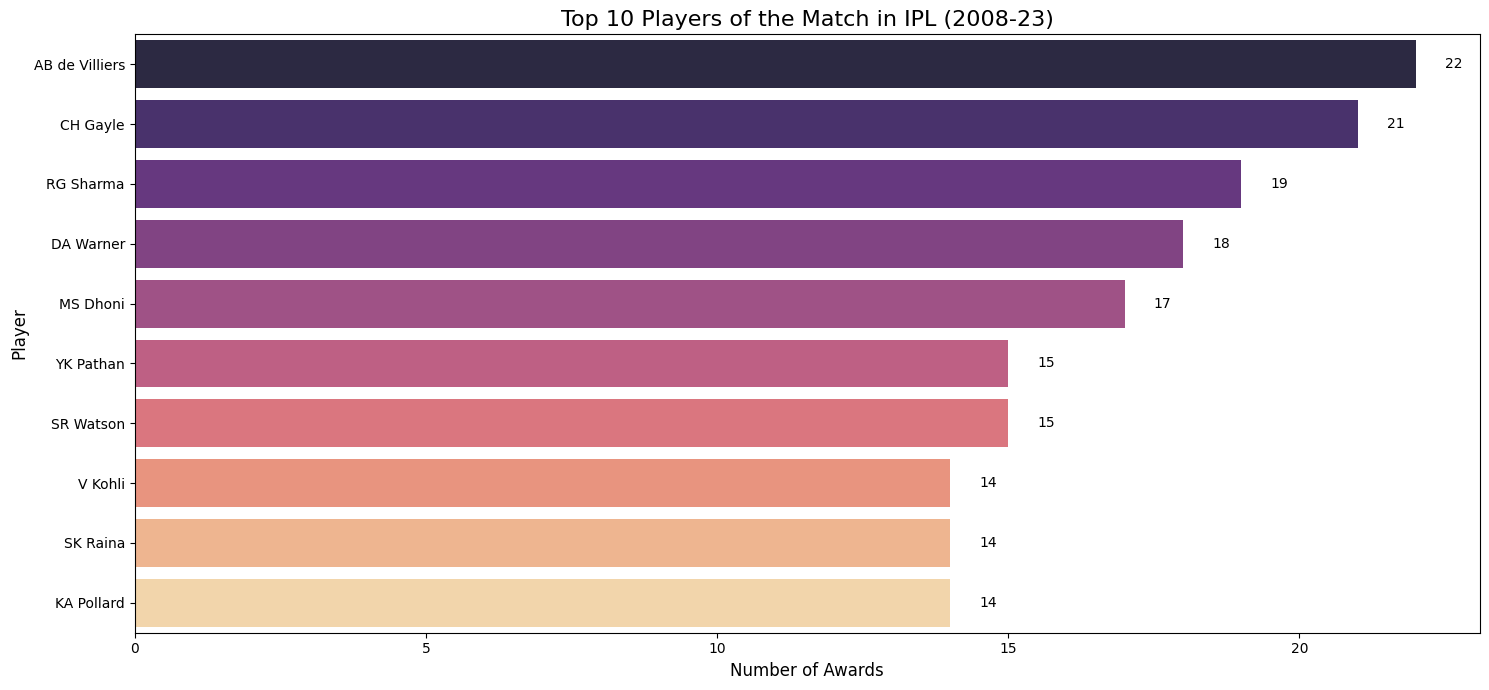
plt.text(value + 0.5, i, str(int(value)), va='center')

plt.title("Top 10 Players of the Match in IPL (2008-23)", fontsize=16) plt.xlabel("Number of Awards", fontsize=12) plt.ylabel("Player", fontsize=12)

plt.tight\_layout()

plt.show()

Out [11]:



In [12]:

***# IPL Matches Distribution Per Season (2008-23):***

season\_counts = data['season'].value\_counts().sort\_index()

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

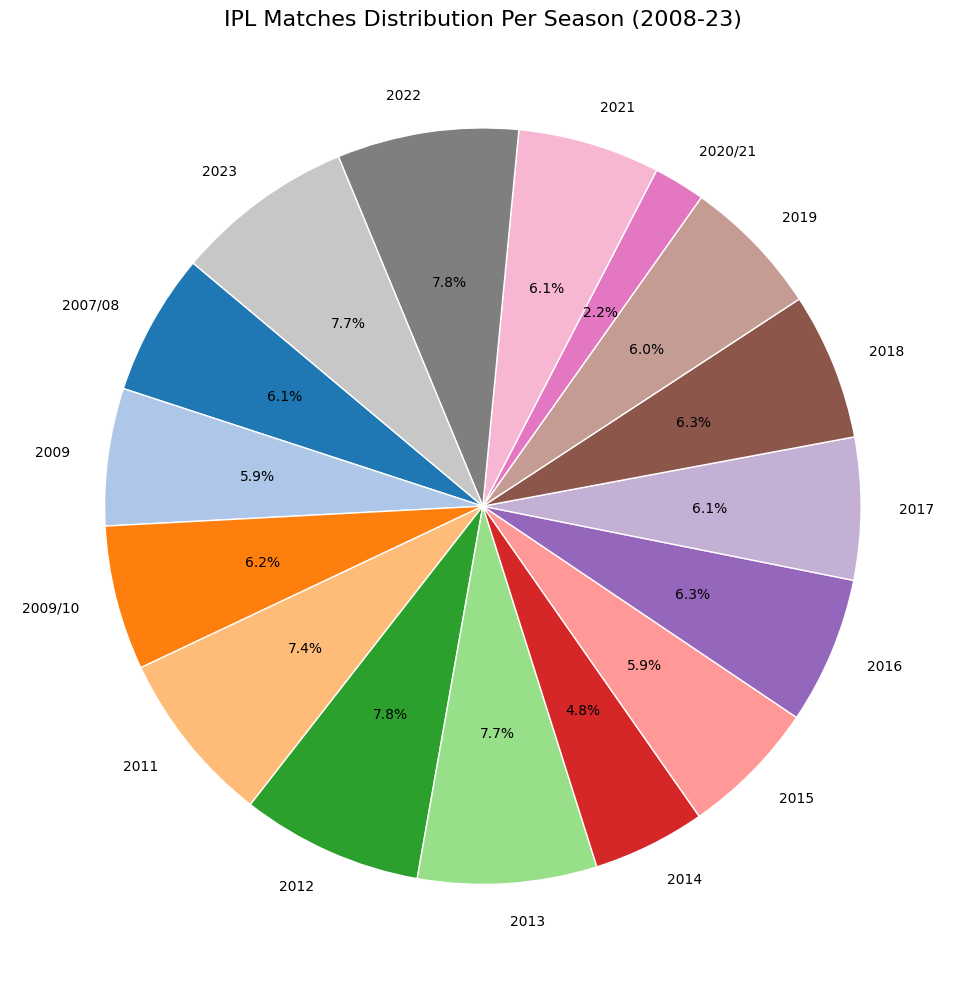
plt.pie(season\_counts, labels=season\_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.tab20.colors, wedgeprops={'edgecolor': 'white'})

plt.title("IPL Matches Distribution Per Season (2008-23)", fontsize=16)

plt.tight\_layout()

plt.show()

Out [12]:



***Predictive Modeling***

In [13]:

# ***# Drop unnecessary columns***

data.drop(columns=['id', 'umpire1', 'umpire2', 'umpire3', 'player\_of\_match', 'win\_by\_runs', 'win\_by\_wickets'], inplace=True)

# ***# Drop rows with missing values***

data.dropna(subset=['winner', 'team1', 'team2', 'toss\_winner', 'venue', 'city'], inplace=True)

# ***# Convert 'date' to datetime and extract date parts***

data['date'] = pd.to\_datetime(data['date'], format='%d-%m-%Y', errors='coerce') data['year'] = data['date'].dt.year data['month'] = data['date'].dt.month data['day'] = data['date'].dt.day data.drop(columns=['date'], inplace=True)

# ***# Encode categorical features***

label\_cols = ['season', 'city', 'team1', 'team2', 'toss\_winner', 'toss\_decision', 'result', 'venue', 'winner'] label\_encoders = {}

for col in label\_cols: le = LabelEncoder() data[col] = le.fit\_transform(data[col]) label\_encoders[col] = le

# ***# Define features and target***

X = data.drop(columns=['winner']) # input features y = data['winner'] # target: winning team

# ***# Split into train and test sets***

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

# ***# Train Random Forest model***

model = RandomForestClassifier(n\_estimators=100, random\_state=42) model.fit(X\_train, y\_train)

# ***# Make predictions and evaluate***

y\_pred = model.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy:.2f}") print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

Out [13]:

Model Accuracy: 0.64

Classification Report:

precision recall f1-score support

0 0.69 1.00 0.81 22

1 0.75 0.50 0.60 6

2 0.50 0.33 0.40 3

3 0.44 0.54 0.48 13

4 1.00 0.33 0.50 3

5 0.57 1.00 0.73 4

6 0.33 0.40 0.36 10

7 0.00 0.00 0.00 0

8 0.85 0.74 0.79 23

9 1.00 0.50 0.67 4

10 0.61 0.79 0.69 28

11 0.00 0.00 0.00 6

12 0.33 0.33 0.33 3

13 0.68 0.62 0.65 21

14 0.00 0.00 0.00 1

15 0.00 0.00 0.00 2

16 0.73 0.57 0.64 28

17 0.69 0.64 0.67 14

accuracy 0.64 191

macro avg 0.51 0.46 0.46 191

weighted avg 0.64 0.64 0.62 191

***Discussion:***

In this notebook, we explored a comprehensive IPL Match (2008-23) dataset. We visualized key metrics, examined correlations, and built a predictive model for estimating revenue. The Random Forest model provided a reasonable prediction accuracy, but there's always room for improvement. Future analysis could explore feature engineering, hyperparameter tuning, or even different modeling approaches to enhance prediction performance. If you found this analysis insightful, please consider upvoting this notebook.

[**Reference link**](https://github.com/Shashidhar2019-DSN/Shashidhar2019-DSN-Analyzing-YouTube-Channel-Statistics.git)