# IPL Match Winning Prediction


## Introduction

The Indian Premier League (IPL) is a popular T20 cricket league that features franchise teams from different cities in India. Predicting the winning probability of an IPL match can be a challenging task, as it depends on several factors such as team strength, players' performance, pitch conditions, weather, and other variables.

In recent years, there has been a growing interest in developing machine learning models that can accurately predict the probability of a team winning an IPL match. These models analyze past match data, team and player statistics, and other variables to estimate the probability of a team winning a match.

This abstract provides an overview of the IPL match winning probability prediction problem and some of the techniques used to address it.We discuss the steps involved in building a winning probability prediction model, including data collection, preprocessing, feature engineering, model selection, training, evaluation, deployment, and maintenance.

The ultimate goal of developing an IPL match winning probability prediction model is to help fans, bettors, and teams make more informed decisions and enhance their IPL experience. With the growing popularity of the IPL and the availability of vast amounts of data, there is a significant opportunity for researchers and developers to further advance the field of IPL match prediction.

## Dataset

The IPL 2008 to 2021 All Match Dataset available on Kaggle is a comprehensive collection of data on all Indian Premier League matches played from the inaugural season in 2008 up to the 2021 season. The dataset includes information on team and player statistics, pitch conditions, weather, and match outcomes.

**IPL_Matches_2008_2022.csv :** This file contains data on each IPL match, including the match ID, season, venue, and the two teams playing in the match.

**IPL_Ball_by_Ball_2008_2022.csv :** This file contains ball-by-ball data for each match, including the bowler, batsman, runs scored, wickets taken, and other relevant information.

## Implementation

In [1]:

```
# Import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Reading in the CSV files using pandas read_csv() function
match_data = pd.read_csv('IPL_Ball_by_Ball_2008_2022.csv')
matches = pd.read_csv('IPL_Matches_2008_2022.csv')
```

In [2]:

```
# This code displays the first 5 rows of the dataframe "match_data"

match_data.head()
```

Out[2]:

| | ID | innings | overs | ballnumber | batter | bowler | non-striker | extra_type | batsman_run | extras_run | total_run | non_boundary | isW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1312200 | 1 | 0 | 1 | YBK Jaiswal | Mohammed Shami | JC Buttler | NaN | 0 | 0 | 0 | 0 | |
| **1** | 1312200 | 1 | 0 | 2 | YBK Jaiswal | Mohammed Shami | JC Buttler | legbyes | 0 | 1 | 1 | 0 | |
| **2** | 1312200 | 1 | 0 | 3 | JC Buttler | Mohammed Shami | YBK Jaiswal | NaN | 1 | 0 | 1 | 0 | |
| **3** | 1312200 | 1 | 0 | 4 | YBK Jaiswal | Mohammed Shami | JC Buttler | NaN | 0 | 0 | 0 | 0 | |
| **4** | 1312200 | 1 | 0 | 5 | YBK Jaiswal | Mohammed Shami | JC Buttler | NaN | 0 | 0 | 0 | 0 | |

In [3]:

```python
import re
# convert all column names to lowercase
match_data.columns = match_data.columns.str.lower()

# Display information about the match_data object
match_data.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225954 entries, 0 to 225953
Data columns (total 17 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   id                 225954 non-null  int64
 1   innings            225954 non-null  int64
 2   overs              225954 non-null  int64
 3   ballnumber         225954 non-null  int64
 4   batter             225954 non-null  object
 5   bowler             225954 non-null  object
 6   non-striker        225954 non-null  object
 7   extra_type         12049 non-null   object
 8   batsman_run        225954 non-null  int64
 9   extras_run         225954 non-null  int64
 10  total_run          225954 non-null  int64
 11  non_boundary       225954 non-null  int64
 12  iswicketdelivery   225954 non-null  int64
 13  player_out         11151 non-null   object
 14  kind               11151 non-null   object
 15  fielders_involved  7988 non-null    object
 16  battingteam        225954 non-null  object
dtypes: int64(9), object(8)
memory usage: 29.3+ MB
```

In [4]:

```
# This code samples 5 random rows from the matches dataframe
matches.sample(5)
```

Out[4]:

| | ID | City | Date | Season | MatchNumber | Team1 | Team2 | Venue | TossWinner | TossDecision | SuperOver |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **435** | 829819 | Pune | 2015-05-20 | 2015 | Eliminator | Royal Challengers Bangalore | Rajasthan Royals | Maharashtra Cricket Association Stadium | Royal Challengers Bangalore | bat | N |
| **198** | 1178430 | Chandigarh | 2019-05-05 | 2019 | 55 | Chennai Super Kings | Kings XI Punjab | Punjab Cricket Association IS Bindra Stadium | Kings XI Punjab | field | N |
| **548** | 729285 | Abu Dhabi | 2014-04-18 | 2014 | 4 | Sunrisers Hyderabad | Rajasthan Royals | Sheikh Zayed Stadium | Rajasthan Royals | field | N |
| **484** | 829719 | Bangalore | 2015-04-13 | 2015 | 8 | Royal Challengers Bangalore | Sunrisers Hyderabad | M Chinnaswamy Stadium | Sunrisers Hyderabad | field | N |

In [5]:

```python
# Lowercase all column names in matches dataframe
matches.columns = matches.columns.str.lower()

# Get information about the 'matches' dataframe
matches.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               950 non-null    int64  
 1   city             899 non-null    object 
 2   date             950 non-null    object 
 3   season           950 non-null    object 
 4   matchnumber      950 non-null    object 
 5   team1            950 non-null    object 
 6   team2            950 non-null    object 
 7   venue            950 non-null    object 
 8   tosswinner       950 non-null    object 
 9   tossdecision     950 non-null    object 
 10  superover        946 non-null    object 
 11  winningteam      946 non-null    object 
 12  wonby            950 non-null    object 
 13  margin           932 non-null    float64 
 14  method           19 non-null     object 
 15  player_of_match  946 non-null    object 
 16  team1players     950 non-null    object 
 17  team2players     950 non-null    object 
 18  umpire1          950 non-null    object 
 19  umpire2          950 non-null    object 
dtypes: float64(1), int64(1), object(18)
memory usage: 148.6+ KB
```

In [6]:

```python
# Group the match data by id and innings, and then calculate the total score for each group
total_score_df = match_data.groupby(['id', 'innings']).sum()['total_run'].reset_index()

# Filter the DataFrame to only include innings with value 1
total_score_df = total_score_df[total_score_df['innings'] == 1]

# This code displays a sample of 5 rows from the total_score_df dataframe.
total_score_df.sample(5)

```

Out[6]:

| | id | innings | total_run |
|---|---|---|---|
| **561** | 548337 | 1 | 146 |
| **707** | 598026 | 1 | 185 |
| **1254** | 1082636 | 1 | 158 |
| **1570** | 1216512 | 1 | 163 |
| **286** | 419132 | 1 | 163 |

In [7]:

```python
# Merge matches dataframe with total_score_df dataframe using the 'id' column
matches_df = matches.merge(total_score_df[['id', 'total_run']], left_on='id', right_on='id')


```

In [8]:

```python
# Get unique values from the 'team1' column of the 'matches_df' DataFrame.
matches_df['team1'].unique()

```

Out[8]:

```
array(['Rajasthan Royals', 'Royal Challengers Bangalore',
       'Sunrisers Hyderabad', 'Delhi Capitals', 'Chennai Super Kings',
       'Gujarat Titans', 'Lucknow Super Giants', 'Kolkata Knight Riders',
       'Punjab Kings', 'Mumbai Indians', 'Kings XI Punjab',
       'Delhi Daredevils', 'Rising Pune Supergiant', 'Gujarat Lions',
       'Rising Pune Supergiants', 'Pune Warriors', 'Deccan Chargers',
       'Kochi Tuskers Kerala'], dtype=object)
```

In [9]:

```python
# A list of cricket teams in the Indian Premier League.
teams = [
    'Chennai Super Kings',
    'Delhi Capitals',
    'Gujarat Titans',
    'Kolkata Knight Riders',
    'Lucknow Super Giants',
    'Mumbai Indians',
    'Punjab Kings',
    'Rajasthan Royals',
    'Royal Challengers Bangalore',
    'Sunrisers Hyderabad'
]

```

In [10]:

```python
# Replace team names in matches_df with updated names

# Replace 'Delhi Daredevils' with 'Delhi Capitals'
matches_df['team1'] = matches_df['team1'].str.replace('Delhi Daredevils', 'Delhi Capitals')
matches_df['team2'] = matches_df['team2'].str.replace('Delhi Daredevils', 'Delhi Capitals')

# Replace 'Deccan Chargers' with 'Sunrisers Hyderabad'
matches_df['team1'] = matches_df['team1'].str.replace('Deccan Chargers', 'Sunrisers Hyderabad')
matches_df['team2'] = matches_df['team2'].str.replace('Deccan Chargers', 'Sunrisers Hyderabad')

# Replace 'Kings XI Punjab' with 'Punjab Kings'
matches_df['team1'] = matches_df['team1'].str.replace('Kings XI Punjab', 'Punjab Kings')
matches_df['team2'] = matches_df['team2'].str.replace('Kings XI Punjab', 'Punjab Kings')

```

In [11]:

```python
# This code prints the unique values of the 'team1' column in the 'matches_df' dataframe.
matches_df['team1'].unique()

```

Out[11]:

```
array(['Rajasthan Royals', 'Royal Challengers Bangalore',
       'Sunrisers Hyderabad', 'Delhi Capitals', 'Chennai Super Kings',
       'Gujarat Titans', 'Lucknow Super Giants', 'Kolkata Knight Riders',
       'Punjab Kings', 'Mumbai Indians', 'Rising Pune Supergiant',
       'Gujarat Lions', 'Rising Pune Supergiants', 'Pune Warriors',
       'Kochi Tuskers Kerala'], dtype=object)
```

In [12]:

```
# Filter matches based on teams
matches_df = matches_df[matches_df['team1'].isin(teams)]
matches_df = matches_df[matches_df['team2'].isin(teams)]

```

In [13]:

```
# Select specific columns of matches_df
matches_df = matches_df[['id', 'city', 'winningteam', 'total_run', 'team1', 'team2']]

# Merge the matches_df and match_data dataframes on the 'id' column
total_df = matches_df.merge(match_data, on='id')

# Display the first five rows of the DataFrame
total_df.head()

```

Out[13]:

| | id | city | winningteam | total_run_x | team1 | team2 | innings | overs | ballnumber | batter | ... | extra_type | batsman_run |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 1 | 0 | 1 | YBK Jaiswal | ... | NaN | 0 |
| **1** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 1 | 0 | 2 | YBK Jaiswal | ... | legbyes | 0 |
| **2** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 1 | 0 | 3 | JC Buttler | ... | NaN | 1 |
| **3** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 1 | 0 | 4 | YBK Jaiswal | ... | NaN | 0 |
| **4** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 1 | 0 | 5 | YBK Jaiswal | ... | NaN | 0 |

5 rows × 22 columns

In [14]:

```
# Select only rows where innings column is equal to 2
total_df = total_df[total_df['innings'] == 2]

```

In [15]:

```python
# Calculate the cumulative sum of 'total_run_y' for each 'id' group in 'total_df' and add it as a new colu
total_df['current_score'] = total_df.groupby('id').cumsum()['total_run_y']

# Subtract the current score from the total run
# to find the number of runs left
total_df['runs_left'] = total_df['total_run_x'] - total_df['current_score']

# Calculate the number of balls left in the cricket game
total_df['balls_left'] = 126 - (total_df['overs'] * 6 + total_df['ballnumber'])

```

In [16]:

```python
# Fill in missing values with "0", convert "player_out" column to binary, then to integer
total_df['player_out'] = total_df['player_out'].fillna("0")
total_df['player_out'] = total_df['player_out'].apply(lambda x: x if x == "0" else "1")
total_df['player_out'] = total_df['player_out'].astype('int')

# Calculate number of wickets for each match by subtracting number of player outs from 10
wickets = total_df.groupby('id').cumsum()['player_out'].values
total_df['wickets'] = 10 - wickets

# Display the first few rows of the updated dataframe
total_df.head()

```

Out[16]:

| | id | city | winningteam | total_run_x | team1 | team2 | innings | overs | ballnumber | batter | ... | non_boundary | iswick |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **120** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 2 | 0 | 1 | WP Saha | ... | 0 | |
| **121** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 2 | 0 | 2 | WP Saha | ... | 0 | |
| **122** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 2 | 0 | 3 | WP Saha | ... | 0 | |
| **123** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 2 | 0 | 4 | Shubman Gill | ... | 0 | |
| **124** | 1312200 | Ahmedabad | Gujarat Titans | 130 | Rajasthan Royals | Gujarat Titans | 2 | 0 | 5 | Shubman Gill | ... | 0 | |

5 rows × 26 columns

In [17]:

```python
# Calculate the current required run rate (CRR)
total_df['crr'] = (total_df['current_score'] * 6) / (120 - total_df['balls_left'])

# Calculate the required run rate (RRR)
total_df['rrr'] = (total_df['runs_left'] * 6) / total_df['balls_left']

```

In [18]:

```python
def result(row):
    """
    Returns 1 if the 'battingteam' equals the 'winningteam', otherwise returns 0.
    """
    return 1 if row['battingteam'] == row['winningteam'] else 0

# Apply the 'result' function to each row of 'total_df' using 'apply' method.
total_df['result'] = total_df.apply(result, axis=1)

```

In [19]:

```python
# Create a new column named "bowlingteam" based on the existing columns
total_df['bowlingteam'] = total_df['team2'].where(
    total_df['battingteam'] == total_df['team1'],
    total_df['team1']
)

```

In [20]:

```python
# Selecting specific columns from a dataframe
final_df = total_df[['battingteam', 'bowlingteam', 'city', 'runs_left', 'balls_left',
                     'wickets', 'total_run_x', 'crr', 'rrr', 'result']]

```

In [21]:

```python
# Shuffle the rows of final_df in place
final_df = final_df.sample(final_df.shape[0])

# Drop any rows with missing values in final_df
final_df.dropna(inplace=True)

```

In [22]:

```
# Filter out rows where 'balls_left' is 0 or 120
final_df = final_df[final_df['balls_left'] != 0]
final_df = final_df[final_df['balls_left'] != 120]

```

## Model Creation and Evaluation

In [23]:

```
# Split the final_df into feature matrix X and target variable y
X = final_df.iloc[:, :-1]
y = final_df.iloc[:, -1]

# Import train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split

# Split X and y into training and testing sets with test_size=0.2 and random_state=1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

```

In [24]:

```

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Create a transformer to one-hot encode categorical variables
trf = ColumnTransformer([
    ('trf', OneHotEncoder(sparse=False, drop='first'), ['battingteam', 'bowlingteam', 'city'])
], remainder='passthrough')

```

In [25]:

```python
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline


# Define a pipeline with two steps: transformer and logistic regression model
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='liblinear'))
])
```

In [26]:

```python
# Fit the pipeline to the training data
pipe.fit(X_train,y_train)
```

```
C:\Users\HITESH\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse`
was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless
you leave `sparse` to its default value.
  warnings.warn(
```

Out[26]:

```
Pipeline(steps=[('step1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('trf',
                                                  OneHotEncoder(drop='first',
                                                                sparse=False),
                                                  ['battingteam', 'bowlingteam',
                                                   'city'])])),
                ('step2', LogisticRegression(solver='liblinear'))])
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [27]:

```python
# Predict using the pipeline
y_pred = pipe.predict(X_test)

```

In [28]:
```
from sklearn.metrics import accuracy_score

# calculate accuracy score of predicted values compared to actual values
accuracy_score(y_test, y_pred)

```

Out[28]:
```
0.8144763399865441
```

In [29]:
```
# Predict the probability of the 15th sample in the test set
pipe.predict_proba(X_test)[108]

```

Out[29]:
```
array([0.57331794, 0.42668206])
```

In [30]:
```
teams
```

Out[30]:
```
['Chennai Super Kings',
 'Delhi Capitals',
 'Gujarat Titans',
 'Kolkata Knight Riders',
 'Lucknow Super Giants',
 'Mumbai Indians',
 'Punjab Kings',
 'Rajasthan Royals',
 'Royal Challengers Bangalore',
 'Sunrisers Hyderabad']
```

In [31]:
```
 total_df['city'].unique()
```

Out[31]:
```
array(['Ahmedabad', 'Kolkata', 'Mumbai', 'Navi Mumbai', 'Pune', 'Dubai',
       'Sharjah', 'Abu Dhabi', 'Delhi', 'Chennai', nan, 'Hyderabad',
       'Visakhapatnam', 'Chandigarh', 'Bengaluru', 'Jaipur', 'Indore',
       'Bangalore', 'Raipur', 'Ranchi', 'Cuttack', 'Dharamsala', 'Nagpur',
       'Johannesburg', 'Centurion', 'Durban', 'Bloemfontein',
       'Port Elizabeth', 'Kimberley', 'East London', 'Cape Town'],
      dtype=object)
```

In [32]:

```python
import pickle
pickle.dump(pipe,open('pipe1.pkl','wb'))
```

## Conclusion

In conclusion, we have used Python and various libraries such as pandas, numpy, and matplotlib to perform data cleaning, data preprocessing, and data analysis on the Indian Premier League match data. We have also used various techniques such as grouping, merging, and filtering data to extract meaningful insights from the data. Specifically, we have analyzed the scores, wickets, and run rates of the teams in the IPL matches. This information can be used to predict the outcome of future matches and help cricket enthusiasts and analysts make informed decisions. Overall, this exercise showcases the power of data analysis and its applications in the field of sports.