Cricket Meets Data Science: Creating an IPL Win Prediction App

BEGINNER DATA SCIENCE GITHUB MACHINE LEARNING PYTHON REGRESSION STREAMLIT

Introduction

IPL (Indian Premier League) is one of the most popular cricket leagues in the world, with millions of fans coming in to watch the games. With the sport's increasing popularity, there has been a growing demand for accurate predictions of the outcome of the games. This article explains one Data Science Project related to Cricket.

To address this, we are going to build a machine learning model that can forecast the winning chances of either of the teams during the middle of the match, based on the first innings performance of a team. Coaches and management of the team can use this model to make data-driven decisions during the game, for example: changing tactics, replacing players, and modifying their plans based on the projected winning likelihood.

Learning Objective

In this article, we will use the IPL dataset from Kaggle, which includes 2 datasets: one is about the matches played between 2008-2019, and the other one is about all the deliveries between 2008-2019. Here is the dataset link

This article was published as a part of the Data Science Blogathon.

Table of Contents

1. Dataset Description
2. Prerequisites
3. Project Pipeline
4. Step-1: Importing the Necessary Libraries
5. Step-2: Read and Load the Dataset

6. Step-3: Exploratory Data Analysis
7. Step-4: Data Preprocessing
8. Step-5: Splitting Data into Train and Test Sets
9. Step-6: Data Science Model Implementation
10. Step-7: Building the App
11. Step-8: Deploying the app on Streamlit Share
12. GitHub Link

Dataset Description

Our **Matches** DataFrame contains the following variables:

- id – Unique id of the matches
- Season – Season of IPL
- city – the city where the match is played
- date – date of the respective match
- team 1 – the name of team 1
- team 2 – the name of team 2
- toss_winner
- toss_decision – field or batting
- result – normal, tie or no result
- dl_applied – related to Duckworth lewis system
- winner – the name of the team
- win by runs
- win by wickets
- player of the match
- venue
- umpire 1, umpire 2, and umpire 3.

There are 21 columns and 179078 observations in our **deliveries** dataset.

21 columns in the deliveries dataset are

[‘match_id,’ ‘inning,’ ‘batting_team,’ ‘bowling_team,’ ‘over,’ ‘ball,’ ‘batsman,’ ‘non_striker,’ ‘bowler,’ ‘is_super_over,’ ‘wide_runs,’ ‘bye_runs,’ ‘legbye_runs,’ ‘noball_runs,’ ‘penalty_runs,’ ‘batsman_runs,’ ‘extra_runs,’ ‘total_runs,’ ‘player_dismissed,’ ‘dismissal_kind,’ ‘fielder’]

Our main focus would be to extract useful information and merge these 2 data frames in order to get all the data in one single DataFrame so that we can build our machine-learning model on top of it.

Prerequisites

- Basic Python
- Jupyter Notebook

- Familiarity with data analysis and machine learning libraries like numpy, Pandas, matplotlib, and Scikit-learn.
- Basic Knowledge of Data Science Projects, EDA, and Data Preprocessing
- Supervised Machine Learning in Data Science Project
- Familiarity with streamlit (optional)
- VS code (here's a quick tutorial: Vscode setup
- Basic Familiarity with GitHub

Note: Even if you lack the knowledge and skills to complete this Data Science Project, you can still learn by following the steps and looking for information whenever you encounter problems. Learning by doing can be an effective and productive way of acquiring new knowledge and skills. So don't worry if you don't know everything immediately; you can learn things by yourself while doing this project.

First things first, follow these steps before starting the Data Science Project:

1. Go to your desktop and create a new folder by right-clicking on your desktop and selecting "New Folder."
2. Name the folder something relevant to the Data Science Project, like IPL Win Predictor System.
3. Move your datasets into the new folder by dragging and dropping them from their current location to the new folder.
4. Open Jupyter Notebook and navigate to the desktop's IPL Win Predictor System folder. There, click on the "New" button and select "Notebook" from the dropdown list. This will open a new python3 notebook tab in your browser.
5. This will create a new Python notebook file in the same directory.
6. You can now start following up with me in your new notebook file and import your datasets using their file paths relative to the notebook file location.

By following these steps, you'll have all your Data Science Project files organized in a single folder, which makes it easier to manage and share your code and datasets.

Project Pipeline

- Importing Necessary Libraries
- Read and Load the Dataset
- Exploratory Data Analysis
- Data Preprocessing
- Splitting our Data into Train and Test sets
- Data Science Model Implementation
- Building the App
- Deploying the App on Streamlit Share

Step-1: Importing the Necessary Libraries

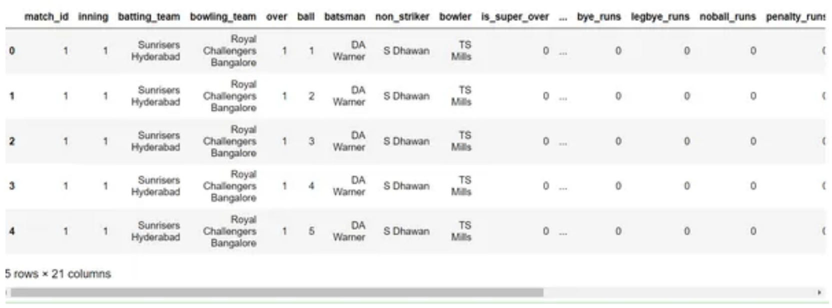
```
#utilities import pandas as pd import numpy as np #plotting import seaborn as sns import matplotlib.pyplot as plt %matplotlib inline #model building from sklearn.model_selection import train_test_split from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder from
```

```
sklearn.linear_model import LogisticRegression from sklearn.ensemble import RandomForestClassifier from sklearn.pipeline import Pipeline from sklearn import metrics
```

Step-2: Read and Load the Dataset

```
matches = pd.read_csv('matches.csv') deliveries = pd.read_csv('deliveries.csv') matches.head()
```

```
deliveries.head()
```



	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	bye_runs	legbye_runs	noball_runs	penalty_run
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	

5 rows × 21 columns

Step-3: Exploratory Data Analysis

3.1: Shape of our data

```
matches.shape,deliveries.shape
```

Output: ((756, 18), (179078, 21))

matches:

There are 756 observations and 18 features in our dataset.

deliveries:

There are 179078 observations and 21 features in our dataset.

3.2: Columns/features in data

For matches:

```
matches.columns
```

Output:

```
Index(['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'], dtype='object')
```

For Deliveries:

```
deliveries.columns
```

Output:

```
Index(['match_id', 'inning', 'batting_team', 'bowling_team', 'over', 'ball', 'batsman', 'non_striker',
'bowler', 'is_super_over', 'wide_runs', 'bye_runs', 'legbye_runs', 'noball_runs', 'penalty_runs',
'batsman_runs', 'extra_runs', 'total_runs', 'player_dismissed', 'dismissal_kind', 'fielder'], dtype='object')
```

3.3: Length of the dataset

For matches:

```
len(matches)
```

756 is the length of our matches data

For deliveries:

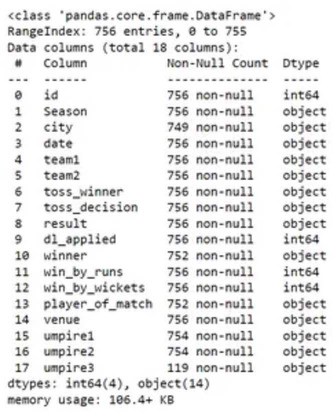
```
len(deliveries)
```

1,79,078 is the length of our deliveries data

3.4: Data information

For matches:

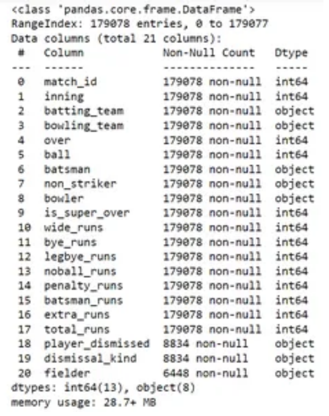
```
matches.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               756 non-null    int64
 1   Season           756 non-null    object
 2   city             749 non-null    object
 3   date             756 non-null    object
 4   team1            756 non-null    object
 5   team2            756 non-null    object
 6   toss_winner      756 non-null    object
 7   toss_decision    756 non-null    object
 8   result           756 non-null    object
 9   dl_applied       756 non-null    int64
 10  winner           752 non-null    object
 11  win_by_runs      756 non-null    int64
 12  win_by_wickets   756 non-null    int64
 13  player_of_match  752 non-null    object
 14  venue            756 non-null    object
 15  umpire1          754 non-null    object
 16  umpire2          754 non-null    object
 17  umpire3          119 non-null    object
dtypes: int64(4), object(14)
memory usage: 106.4+ KB
```

For deliveries:

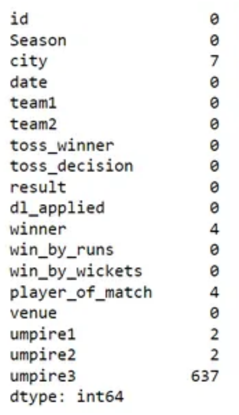
```
deliveries.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 179078 entries, 0 to 179077
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   match_id          179078 non-null  int64
 1   inning            179078 non-null  int64
 2   batting_team      179078 non-null  object
 3   bowling_team      179078 non-null  object
 4   over              179078 non-null  int64
 5   ball              179078 non-null  int64
 6   batsman           179078 non-null  object
 7   non_striker       179078 non-null  object
 8   bowler            179078 non-null  object
 9   is_super_over     179078 non-null  int64
 10  wide_runs         179078 non-null  int64
 11  bye_runs          179078 non-null  int64
 12  legbye_runs       179078 non-null  int64
 13  noball_runs       179078 non-null  int64
 14  penalty_runs      179078 non-null  int64
 15  batsman_runs      179078 non-null  int64
 16  extra_runs        179078 non-null  int64
 17  total_runs        179078 non-null  int64
 18  player_dismissed  8834 non-null    object
 19  dismissal_kind    8834 non-null    object
 20  fielder           6448 non-null    object
dtypes: int64(13), object(8)
memory usage: 28.7+ MB
```

3.5: Checking for null values matches:

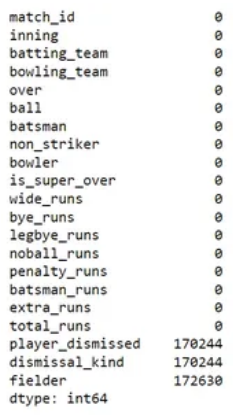
```
matches.isnull().sum()
```



```
id                   0
Season               0
city                 7
date                 0
team1                0
team2                0
toss_winner          0
toss_decision        0
result               0
dl_applied           0
winner               4
win_by_runs          0
win_by_wickets       0
player_of_match      4
venue                0
umpire1              2
umpire2              2
umpire3            637
dtype: int64
```

deliveries:

```
deliveries.isnull().sum()
```



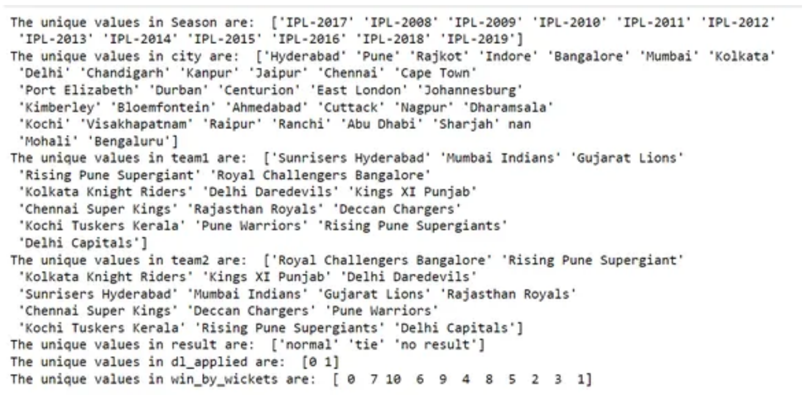
```
match_id                 0
inning                   0
batting_team             0
bowling_team             0
over                     0
ball                     0
batsman                  0
non_striker              0
bowler                   0
is_super_over            0
wide_runs                0
bye_runs                 0
legbye_runs              0
noball_runs              0
penalty_runs             0
batsman_runs             0
extra_runs               0
total_runs               0
player_dismissed    170244
dismissal_kind      170244
fielder             172630
dtype: int64
```

We will see later how to handle these null values.

3.6: Checking unique values in required columns:
For matches:

```
#creating a list of columns and removing unnecessary columns from the list list1 = matches.columns.to_list() remove_from_list = ['id', 'date', 'toss_winner', 'toss_decision', 'winner', 'win_by_runs', 'player_of_match', 'venue', 'umpire1', 'umpire2', 'umpire3'] for i in range(len(remove_from_list)): list1.remove(remove_from_list[i])
```

```
for i in list1: print('The unique values in', i, 'are: ', matches[i].unique())
```



```
The unique values in Season are:  ['IPL-2017' 'IPL-2008' 'IPL-2009' 'IPL-2010' 'IPL-2011' 'IPL-2012'
 'IPL-2013' 'IPL-2014' 'IPL-2015' 'IPL-2016' 'IPL-2018' 'IPL-2019']
The unique values in city are:  ['Hyderabad' 'Pune' 'Rajkot' 'Indore' 'Bangalore' 'Mumbai' 'Kolkata'
 'Delhi' 'Chandigarh' 'Kanpur' 'Jaipur' 'Chennai' 'Cape Town'
 'Port Elizabeth' 'Durban' 'Centurion' 'East London' 'Johannesburg'
 'Kimberley' 'Bloemfontein' 'Ahmedabad' 'Cuttack' 'Nagpur' 'Dharamsala'
 'Kochi' 'Visakhapatnam' 'Raipur' 'Ranchi' 'Abu Dhabi' 'Sharjah' nan
 'Mohali' 'Bengaluru']
The unique values in team1 are:  ['Sunrisers Hyderabad' 'Mumbai Indians' 'Gujarat Lions'
 'Rising Pune Supergiant' 'Royal Challengers Bangalore'
 'Kolkata Knight Riders' 'Delhi Daredevils' 'Kings XI Punjab'
 'Chennai Super Kings' 'Rajasthan Royals' 'Deccan Chargers'
 'Kochi Tuskers Kerala' 'Pune Warriors' 'Rising Pune Supergiants'
 'Delhi Capitals']
The unique values in team2 are:  ['Royal Challengers Bangalore' 'Rising Pune Supergiant'
 'Kolkata Knight Riders' 'Kings XI Punjab' 'Delhi Daredevils'
 'Sunrisers Hyderabad' 'Mumbai Indians' 'Gujarat Lions' 'Rajasthan Royals'
 'Chennai Super Kings' 'Deccan Chargers' 'Pune Warriors'
 'Kochi Tuskers Kerala' 'Rising Pune Supergiants' 'Delhi Capitals']
The unique values in result are:  ['normal' 'tie' 'no result']
The unique values in dl_applied are:  [0 1]
The unique values in win_by_wickets are:  [ 0  7 10  6  9  4  8  5  2  3  1]
```

For deliveries:

```
list2 = deliveries.columns.to_list() remove_from_list2 = ['match_id', 'batsman','inning', 'non_striker', 'bowler', 'player_dismissed', 'fielder'] for i in range(len(remove_from_list2)): list2.remove(remove_from_list2[i])
```

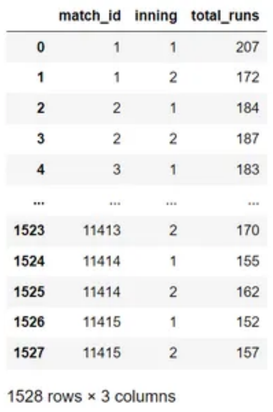
```
for i in list2: print('The unique values in', i, 'are: ', deliveries[i].unique())
```

We are looking for every ball that is thrown during an over. This allows us to see how the batsman score runs, runs awarded to the teams, penalty runs, or other factors that may impact the delivery outcome. It's important to note that the total number of runs scored on a particular delivery can vary based on these factors. For example, it's possible for a batsman to score 10 runs on a single delivery if it is a no-ball and the batsman hits a six and then takes advantage of the no-ball to score 3 more runs for the team. This is how the unique value in the total_runs column includes 10.

Step-4: Data Preprocessing

4.1: Grouping by deliveries on match id and innings

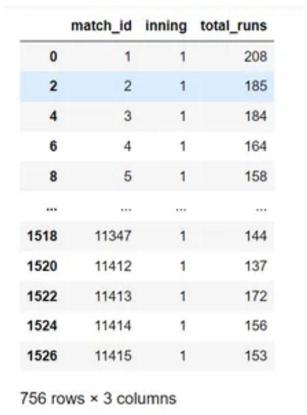
```
# grouping the 1st innings,2nd innings score in a particular matchid # lets say match id = 1,so inning 1 score = 207,inning 2 score = 172 totalrun_df=deliveries.groupby(['match_id','inning']).sum()['total_runs'].reset_index() totalrun_df
```



	match_id	inning	total_runs
0	1	1	207
1	1	2	172
2	2	1	184
3	2	2	187
4	3	1	183
...	...	...	...
1523	11413	2	170
1524	11414	1	155
1525	11414	2	162
1526	11415	1	152
1527	11415	2	157

1528 rows × 3 columns

```
#capturing only the first innings, because we will be predicting the second innings #Also we are changing total runs into targets by using the lambda function totalrun_df = totalrun_df[totalrun_df['inning']==1] totalrun_df['total_runs'] = totalrun_df['total_runs'].apply(lambda x:x+1)#to get target totalrun_df
```

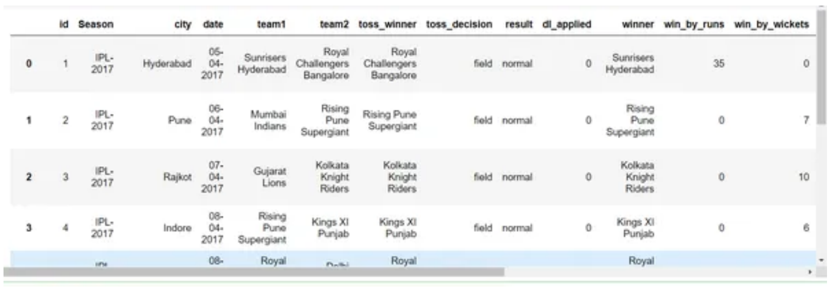


	match_id	inning	total_runs
0	1	1	208
2	2	1	185
4	3	1	184
6	4	1	164
8	5	1	158
...	...	...	...
1518	11347	1	144
1520	11412	1	137
1522	11413	1	172
1524	11414	1	156
1526	11415	1	153

756 rows × 3 columns

4.2: Merging 2 data frames

```
match_df = matches.merge(totalrun_df[['match_id','total_runs']], left_on='id',right_on='match_id') match_df
```



	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets
0	1	IPL-2017	Hyderabad	05-04-2017	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0
1	2	IPL-2017	Pune	06-04-2017	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7
2	3	IPL-2017	Rajkot	07-04-2017	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10
3	4	IPL-2017	Indore	08-04-2017	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6
		IPL-		08-	Royal	Delhi	Royal				Royal		

In this merging process, an inner join is performed between the “matches” DataFrame and the “totalrun_df” DataFrame. So, only the rows that have matching values in both tables will be shown in the result. Specifically, the merge will be done by matching the values in the “**id**” column of the “**matches**” DataFrame with the values in the “**match_id**” column of the “**totalrun_df**” DataFrame. Therefore, the output will contain only the rows where there is a match between these two columns, i.e., id and match_id

4.3: Renaming Teams

```
match_df['team1'].unique()
```

Output:

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

Now we will only keep the most frequent and common teams, which include:

```
teams = [ 'Sunrisers Hyderabad', 'Mumbai Indians', 'Royal Challengers Bangalore', 'Kolkata Knight Riders',
'Kings XI Punjab', 'Chennai Super Kings', 'Rajasthan Royals', 'Delhi Capitals' ]
```

```
# replacing the Delhi Daredevils with Delhi Capitals match_df['team1'] = match_df['team1'].str.replace('Delhi
Daredevils','Delhi Capitals') match_df['team2'] = match_df['team2'].str.replace('Delhi Daredevils','Delhi
Capitals') # replacing the Deccan Chargers with Sunrises Hyderabad match_df['team1'] =
match_df['team1'].str.replace('Deccan Chargers','Sunrisers Hyderabad') match_df['team2'] =
match_df['team2'].str.replace('Deccan Chargers','Sunrisers Hyderabad')
```

```
# considering only frequently occurring teams, # which are mentioned in the team's list match_df =
match_df[match_df['team1'].isin(teams)] match_df = match_df[match_df['team2'].isin(teams)]
match_df['team1'].unique() #again checking unique team names from match_df
```

Output:

```
array(['Sunrisers Hyderabad', 'Royal Challengers Bangalore', 'Kolkata Knight Riders', 'Kings XI Punjab',
'Delhi Capitals', 'Mumbai Indians', 'Chennai Super Kings', 'Rajasthan Royals'], dtype=object)
```

4.4: Handling matches that resulted in the Duckworth lewis system

```
# checking the matches which resulted in the DL method
match_df[match_df['dl_applied']==1].style.background_gradient(cmap = 'plasma')
```

The above code snippet looks into the rows of the “match_df” DataFrame where the value of the “dl_applied” column equals 1, indicating that the Duckworth lewis system was applied during the match.

The “style.background_gradient(cmap=’plasma’)” part of our code applies a color gradient to the resulting DataFrame using the “plasma” color map. This styling feature in pandas allows for visualizing the data more appealingly.

Now we don’t want to confuse our model with this stuff, so we will ignore the rows where the Duckworth-lewis system was applied during the match.

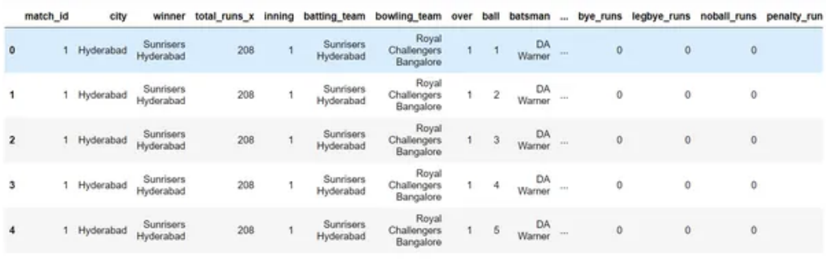
```
# ignoring the rows which were Duckworth lewis system match_df = match_df[match_df['dl_applied']==0] # considering the match_id, city, winner, and total runs match_df = match_df[['match_id','city','winner','total_runs']] match_df
```

4.5: Merging match_df with deliveries on match_id

The reason for merging the "match_df" DataFrame with the "deliveries" DataFrame is to keep track of the current score and the score the team needs to achieve to win the match. While the "match_df" DataFrame provides information on the target score (total_runs), the "deliveries" DataFrame contains details of every ball bowled in the match and the runs scored in each ball.

By merging these two DataFrames, we can get a comprehensive view of the match's progress, including the current score, the target score, and other relevant information we need to analyze the game's performance.

```
#merging matchdf with deliveries on match_id delivery_df = match_df.merge(deliveries,on='match_id') delivery_df.head(5)
```



	match_id	city	winner	total_runs_x	inning	batting_team	bowling_team	over	ball	batsman	...	bye_runs	legbye_runs	noball_runs	penalty_run
0	1	Hyderabad	Sunrisers Hyderabad	208	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	...	0	0	0	
1	1	Hyderabad	Sunrisers Hyderabad	208	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	...	0	0	0	
2	1	Hyderabad	Sunrisers Hyderabad	208	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	...	0	0	0	
3	1	Hyderabad	Sunrisers Hyderabad	208	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	...	0	0	0	
4	1	Hyderabad	Sunrisers Hyderabad	208	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	...	0	0	0	

From the above 2 images, it seems that after merging the relevant DataFrames, there are two columns named "total_runs": "total_runs_x" and "total_runs_y." "total_runs_x" refers to the target score set by the batting team in the first innings, while "total_runs_y" refers to the runs scored on each delivery by the batting team during the second innings. Therefore, by having both columns available in the merged

DataFrame, we can track the progress of the second innings and compare it to the target score set in the first innings.

Now let’s check the different columns present in our delivery_df:

```
delivery_df.columns
```

Output:

```
Index(['match_id', 'city', 'winner', 'total_runs_x', 'inning', 'batting_team', 'bowling_team', 'over', 'ball', 'batsman', 'non_striker', 'bowler', 'is_super_over', 'wide_runs', 'bye_runs', 'legbye_runs', 'noball_runs', 'penalty_runs', 'batsman_runs', 'extra_runs', 'total_runs_y', 'player_dismissed', 'dismissal_kind', 'fielder'], dtype='object')
```

In delivery_df, we are looking into the second innings because we have to keep track of the current score; earlier, we kept the first innings in the totalrun_df DataFrame because we want to get the target that the second team needs to achieve.

Now let’s check the shape of our final data frame – delivery_df:

```
delivery_df.shape
```

Output:

(72413, 24)

This means that after merging the relevant DataFrames, the resulting DataFrame contains 72,413 rows and 24 columns, representing our final dataset with all the relevant information needed to analyze the teams’ performance in the matches.

4.6: Creating necessary columns

```
# to get the current score of a particular match delivery_df['current_score'] = delivery_df.groupby('match_id').cumsum()['total_runs_y'] delivery_df.head()
```

To calculate the chances of either team winning or losing a cricket match, we need to know the current runs scored by the team and the respective RunRate. To get the current runs scored, we can group the data by the “match_id” column and use the “total_runs_y” column, which contains the runs scored by the batting team ball by a ball during the second innings.

By applying the cumulative sum function to the “total_runs_y” column for each match, we can calculate the current score of the batting team at each ball in the innings. Therefore, by using the cumulative sum of “total_runs_y” for each match, we can get the current score of the batting team, which is necessary for calculating their RunRate and predicting the outcome of the match.

As we can see from the above 2 images, the current score keeps on updating after each delivery. This is what we did using the cumulative sum function. Try it out more deeply by observing different rows in your notebook.

```
# creating column for runs left delivery_df['runs_left'] = delivery_df['total_runs_x']-
delivery_df['current_score'] delivery_df[['total_runs_x', 'current_score', 'runs_left']].head()
```

```
# creating a column for balls left delivery_df['balls_left'] = 126-
(delivery_df['over']*6+delivery_df['ball']) delivery_df[['over', 'ball', 'balls_left']].head(10)
```

In the above code, we have used this formula: **126 – (delivery_df['over']*6+delivery_df['ball'])** for calculating the number of balls left, now let's say over is 1, and the ball is 1, then total balls left will be 126 – (1*6+1) = 119. Similarly, when two balls are played, the balls left will be 118. This is similar to when we subtract one ball from total balls 120, i.e., 120-1 = 119. With our formula, we get the same output, which is useful in preventing miscalculations. That's why we used this formula: "balls_left = 126 – (over * 6 + current_ball)"

4.7: Handling null values

```
delivery_df['player_dismissed']
```

Replacing all NaN values with 0 and other ones with 1 in order to make it a binary column.

```
# filling nan values with "0" delivery_df['player_dismissed'] = delivery_df['player_dismissed'].fillna("0") #
now we will convert this player_dismissed col into a boolean col # if the player is not dismissed then it's 0
else it's 1 delivery_df['player_dismissed'] = delivery_df['player_dismissed'].apply(lambda x:x if x=="0" else
"1") # converting string to int delivery_df['player_dismissed'] =
delivery_df['player_dismissed'].astype('int') delivery_df['player_dismissed'].unique()
```

Output:

```
array([0, 1])
```

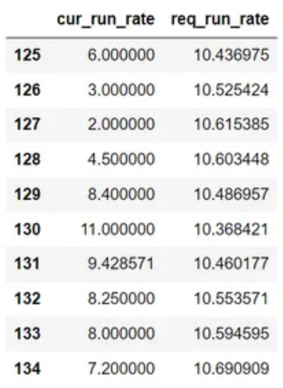
```
# creating a column named wickets left wickets = delivery_df.groupby('match_id').cumsum()
['player_dismissed'].values delivery_df['wickets_left'] = 10-wickets
```

Here, we use cumulative wicket counts for each delivery in each match. For example, if the first delivery in a match results in a wicket, the first element of the array will be 1. If the second ball results in a wicket, the second element will be 2, and so on, and this number will be stored in a variable named wickets.

And to get wickets_left, subtract wickets from 10.

Now, let’s calculate the current RunRate, and required RunRate:

```
# current RunRate # It is a common practice to express run rates in cricket which means to #express it in
runs per over, # so the score is multiplied by 6. delivery_df['cur_run_rate'] =
(delivery_df['current_score']*6)/(120-delivery_df['balls_left']) # required Run-Rate
delivery_df['req_run_rate'] = (delivery_df['runs_left']*6)/(delivery_df['balls_left']) #Current Run-Rate
delivery_df[['cur_run_rate', 'req_run_rate']].head(10)
```



	cur_run_rate	req_run_rate
125	6.000000	10.436975
126	3.000000	10.525424
127	2.000000	10.615385
128	4.500000	10.603448
129	8.400000	10.486957
130	11.000000	10.368421
131	9.428571	10.460177
132	8.250000	10.553571
133	8.000000	10.594595
134	7.200000	10.690909

4.8: Target Variable analysis

```
def resultofmatch(row): return 1 if row['batting_team'] == row['winner'] else 0 delivery_df['result'] = delivery_df.apply(resultfun,axis=1)
```

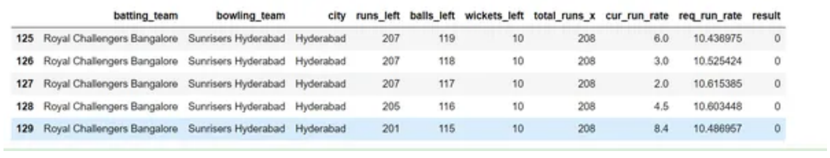
We have created a function named 'resultofmatch' that takes in a row of a DataFrame and returns 1 if the team that batted first is the winner, else it returns 0.

```
sns.countplot(delivery_df['result'])
```

From the above plot, we can see that the batting team won the most number of matches.

4.9: Creating Final Data frame

```
final_df = delivery_df[['batting_team','bowling_team','city','runs_left', 'balls_left','wickets_left','total_runs_x','cur_run_rate', 'req_run_rate','result']] ##we are taking only important columns final_df.head()
```



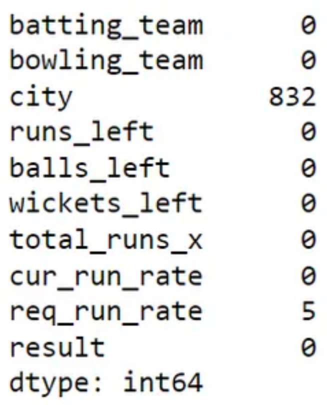
	batting_team	bowling_team	city	runs_left	balls_left	wickets_left	total_runs_x	cur_run_rate	req_run_rate	result
125	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	119	10	208	6.0	10.436975	0
126	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	118	10	208	3.0	10.525424	0
127	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	117	10	208	2.0	10.615385	0
128	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	205	116	10	208	4.5	10.603448	0
129	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	201	115	10	208	8.4	10.486957	0

```
final_df.shape
```

Output:

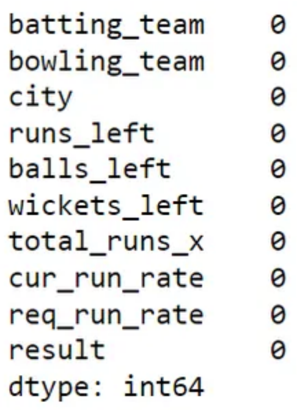
(72413, 10)

```
final_df.isnull().sum()
```



```
batting_team      0
bowling_team      0
city            832
runs_left         0
balls_left        0
wickets_left      0
total_runs_x      0
cur_run_rate      0
req_run_rate      5
result            0
dtype: int64
```

```
# dropping of null values final_df = final_df.dropna() final_df.isnull().sum()
```



```
batting_team    0
bowling_team    0
city            0
runs_left       0
balls_left      0
wickets_left    0
total_runs_x    0
cur_run_rate    0
req_run_rate    0
result          0
dtype: int64
```

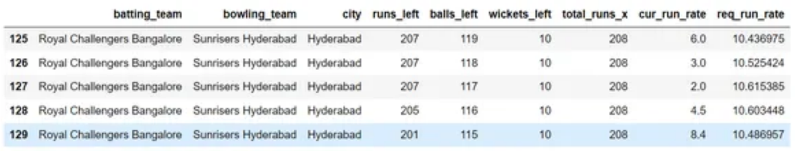
```
final_df = final_df[final_df['balls_left'] != 0]
```

We're removing all the rows where the number of remaining balls is 0, which indicates that the match has ended.

We're doing it because it allows us to focus only on the part of the match that is still in progress and exclude the completed matches. The resulting DataFrame contains only the data from the ongoing matches, which can be further analyzed to make predictions about the outcome.

Step-5: Splitting Data into Train and Test Sets

```
data = final_df.copy() test = data['result'] train = data.drop(['result'],axis = 1) train.head()
```



	batting_team	bowling_team	city	runs_left	balls_left	wickets_left	total_runs_x	cur_run_rate	req_run_rate
125	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	119	10	208	6.0	10.436975
126	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	118	10	208	3.0	10.525424
127	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	207	117	10	208	2.0	10.615385
128	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	205	116	10	208	4.5	10.603448
129	Royal Challengers Bangalore	Sunrisers Hyderabad	Hyderabad	201	115	10	208	8.4	10.486957

```
from sklearn.model_selection import train_test_split X_train,X_test,y_train,y_test = train_test_split(train,test,test_size=0.2,random_state=1) X_train.shape,X_test.shape
```

Output:

```
((57073, 9), (14269, 9))
```

```
X_train.columns
```

Output:

```
Index(['batting_team', 'bowling_team', 'city', 'runs_left', 'balls_left', 'wickets', 'total_runs_x', 'cur_run_rate', 'req_run_rate'], dtype='object')
```

Step-6: Data Science Model Implementation

```
# batting team, bowling team, and city are categorical columns # they will be converted to numeric using a one-hot encoder cf = ColumnTransformer(transformers = [ ('tnf1',OneHotEncoder(sparse=False,drop='first'), ['batting_team','bowling_team','city']) ],remainder='passthrough')
```

The transformer we are using here is OneHotEncoder, which is used to encode categorical features into one-hot numeric arrays. The 'sparse' argument is set to False, which means that the output will be a dense array. The 'drop' argument is set to 'first', meaning that each column's first category will be dropped to avoid collinearity.

The column names in the list ['batting_team,' 'bowling_team,' 'city'] are the categorical features that the OneHotEncoder should encode.

5.1: Implementing Logistic Regression Model

```
# creating the pipeline # lr = LogisticRegression(solver='liblinear') pipe = Pipeline(steps=[ ('step1', cf), ('step2',LogisticRegression(solver='liblinear')) ]) # fitting the training data pipe.fit(X_train,y_train)
```

Output:

```
Pipeline(steps=[('step1', ColumnTransformer(remainder='passthrough', transformers=[('tnf1', OneHotEncoder(drop='first', sparse=False), ['batting_team', 'bowling_team', 'city'])])), ('step2', LogisticRegression(solver='liblinear'))])
```

```
y_pred = pipe.predict(X_test) print(metrics.accuracy_score(y_test,y_pred))
```

Output:

0.8063634452309202

0.80 is the accuracy we got from logistic regression. One of the major reasons why we applied logistic regression is because of the binary outcome of our target variable.

Now let's check the performance of our logistic regression model on test data:

```
pipe.predict_proba(X_test)[10]
```

Output:

```
array([0.5443571, 0.4556429])
```

5.2: Implementing Random Forest Classifier

```
# rf = RandomForestClassifier() pipe2 = Pipeline(steps=[ ('step1', cf), ('step2',RandomForestClassifier()) ])
pipe2.fit(X_train,y_train) print(metrics.accuracy_score(y_test,pipe2.predict(X_test)))
```

Output:

0.9990889340528418

Let's check the performance of the random forest classifier on test data:

```
pipe2.predict_proba(X_test)[10]
```

```
array([0.04, 0.96])
```

B0th index represents the likelihood of losing, and the 1st index represents the likelihood of winning

From the above outputs, our model seems 99% accurate. But it might be an indicator of some serious issue or overfitting. So we will move forward with a logistic regression model only.

Also, the Random Forest model produces highly accurate results, which we can see from our output. For example, for the 10th sample, the model shows a 96% likelihood of winning and only a 4% likelihood of losing. This bias can sometimes be unrealistic and too strong, and it is better to use a model that treats both classes equally. We don't know which team will perform better and win the game, so having a fair and unbiased model on both sides is important.

Now let's save our model in a pickle file:

After running this code below, check your folder 'IPL Win Predictor System.' There should be a file named pipe. pkl in that folder.

```
# saving the logistic regression model import pickle pickle.dump(pipe, open('pipe.pkl', 'wb'))
```

Step-7: Building the App

1. Open VS Code on your system.
2. In the VS Code window, click on the "File" menu and select "Open Folder."
3. Navigate to the IPL Win Predictor System folder on your desktop, and select it.
4. Click on the "Open" button to open the folder in VS Code.
5. Once the folder is open in VS Code, create a new file by hovering over the folder name in the left-hand sidebar and selecting "New File."
6. Name the new file app.py, and press Enter to create the file.
7. With the new app.py file selected in the sidebar, start writing your Python code in the file named app.py

Let's start creating our app now:

```
#Importing the necessary dependencies import streamlit as st import pandas as pd import pickle from
sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder # Declaring the
teams teams = ['Sunrisers Hyderabad', 'Mumbai Indians', 'Royal Challengers Bangalore', 'Kolkata Knight
Riders', 'Kings XI Punjab', 'Chennai Super Kings', 'Rajasthan Royals', 'Delhi Capitals'] # declaring the
venues where the matches are going to take place cities = ['Hyderabad', 'Bangalore', 'Mumbai', 'Indore',
'Kolkata', 'Delhi', 'Chandigarh', 'Jaipur', 'Chennai', 'Cape Town', 'Port Elizabeth', 'Durban', 'Centurion',
'East London', 'Johannesburg', 'Kimberley', 'Bloemfontein', 'Ahmedabad', 'Cuttack', 'Nagpur', 'Dharamsala',
'Visakhapatnam', 'Pune', 'Raipur', 'Ranchi', 'Abu Dhabi', 'Sharjah', 'Mohali', 'Bengaluru'] # Loading our
machine learning model from a saved pickle file pipe = pickle.load(open('pipe.pkl', 'rb')) #remember all
folders including pipe.pkl, # notebook, datasets exist in the same directory # Setting up the app's title
st.title('IPL Win Predictor') # Setting up the layout with two columns col1, col2 = st.columns(2) # Creating
a dropdown selector for the batting team with col1: battingteam = st.selectbox('Select the batting team',
sorted(teams)) # Creating a dropdown selector for the bowling team with col2: bowlingteam =
st.selectbox('Select the bowling team', sorted(teams)) # Creating a dropdown selector for the city where the
match is being played city = st.selectbox( 'Select the city where the match is being played', sorted(cities))
# Creating a numeric input for the target score using number_input method in streamlit target =
int(st.number_input('Target', step=1)) # Setting up the layout with three columns col3, col4, col5 =
st.columns(3) # Creating a numeric input for the current score with col3: score =
int(st.number_input('Score', step=1)) # Creating a numeric input for the number of overs completed with col4:
overs = int(st.number_input('Overs Completed', step=1)) # Creating a numeric input for the number of wickets
fallen with col5: wickets = int(st.number_input('Wickets Fallen', step=1)) # Checking for different match
results based on the input provided if score > target: st.write(battingteam,"won the match") elif score ==
target-1 and overs==20: st.write("Match Drawn") elif wickets==10 and score < target-1: st.write(bowlingteam,
'Won the match') elif wickets==10 and score == target-1: st.write('Match tied') elif
battingteam==bowlingteam: st.write('To proceed, please select different teams because no match can be played
between the same teams') else: # Checking if the input values are valid or not if target >= 0 and target <=
300 and overs >= 0 and overs <=20 and wickets <= 10 and wickets>=0 and score>= 0: try: if st.button('Predict
Probability'): # Calculating the number of runs left for the batting team to win runs_left = target-score #
Calculating the number of balls left balls_left = 120-(overs*6) # Calculating the number of wickets left for
the batting team wickets = 10-wickets # Calculating the current Run-Rate of the batting team currentrunrate =
score/overs # Calculating the Required Run-Rate for the batting team to win requiredrunrate =
(runs_left*6)/balls_left # Creating a pandas DataFrame containing the user inputs input_df = pd.DataFrame(
{'batting_team': [battingteam], 'bowling_team': [bowlingteam], 'city': [city], 'runs_left': [runs_left],
'balls_left': [balls_left], 'wickets': [wickets], 'total_runs_x': [target], 'cur_run_rate': [currentrunrate],
'req_run_rate': [requiredrunrate]}) # Loading the trained machine learning pipeline to make the prediction
result = pipe.predict_proba(input_df) # Extracting the likelihood of loss and win lossprob = result[0][0]
winprob = result[0][1] # Displaying the predicted likelihood of winning and losing in percentage
st.header(battingteam+"-                "+str(round(winprob*100))+"%")                st.header(bowlingteam+"-
"+str(round(lossprob*100))+"%") #Catching ZeroDivisionError except ZeroDivisionError: st.error("Please fill
all the details") #Displaying an error message if the input is incorrect else: st.error('There is something
wrong with the input, please fill the correct details')
```

Above we have implemented IPL win prediction system app using Streamlit. It takes input from the user, including the batting team, bowling team, city where the match is being played, target score, current score, overs completed, and wickets fallen. It then processes the input data and predicts the likelihood of either of the teams winning the match using a pre-trained logistic regression model. The prediction is displayed on the screen. Our code implements several checks to ensure that the input data is valid and appropriate for the IPL T-20 format.

I have used embedded comments to explain the code. You can refer to these comments to understand how we developed the application and how it works.

Finally, we have created our app.

Now let’s check what files are present in our desktop folder till now: app.py, python notebook file, deliveries.csv, matches.csv, pipe. pkl. Now let’s create our requirements.txt file.

How to create a requirements.txt file?

In IPL Win Predictor System, create a new text file named requirements.txt and copy-paste all these dependencies into that folder.

```
streamlit sklearn pandas scikit-learn==0.24
```

There are also traditional techniques to create a requirements.txt file, such as running the “**pip freeze > requirements.txt**” command in the terminal. However, this command can add unnecessary dependencies to the file and may cause version issues. Therefore, instead of relying on the automated process, we manually create the requirements.txt file to ensure that it only includes the necessary dependencies and versions. This way, we can have better control over the dependencies and avoid potential version conflicts.

Finally, we have all the required files in our folder now.

Quick Check: These files should be present in your folder:

- app.py
- deliveries.csv
- matches.csv
- python notebook file (e.g., ipl_win_predictor.ipynb)
- pipe. pkl
- requirement.txt

Step-8: Deploying the app on Streamlit Share

To make our Data Science Project available on GitHub and Streamlit Share, follow these steps:

1. Create a new repository on GitHub with a name related to our project. Make sure to click on “Add README.md” while creating the repository.
2. Upload all the files from your project folder to this repository.
3. Go to the Streamlit Share website and create an account if you don’t already have one.
4. Select “New App” and choose “From Existing Repo” in the dropdown list to create a new Streamlit app.
5. Enter the link to your GitHub repository in the “Repository” section
6. In the “Branch” section, select “main.”
7. In the “Main File Path” section, select the name of your Python file that contains the Streamlit app code (in our case, it is app.py).
8. Click on the “Deploy” button to deploy your app on Streamlit Share.

Finally, we have deployed our app on the streamlit share platform. Congratulations to you; here’s a glimpse at our working app:

Before:

After:

Based on the current score of the match, our app predicts that there is a 47% probability of Chennai Super Kings winning the match and a 53% probability of Mumbai Indians winning the match.

GitHub Link: https://github.com/HarshJain41/IPL-Match-Win-Predictor

Link to the working app: app

Conclusion

We have now ended this exciting Data Science Project, and I hope you have learned something new and valuable. Throughout our journey, we have explored various topics, starting from creating a folder to store our project files, performing exploratory data analysis, data preprocessing, developing a model, and finally deploying it on the Streamlit Share platform.

I would like to thank you for taking the time to complete this Data Science Project, and I congratulate you for making it to the end of this article. Keep exploring and expanding your knowledge in the field of data science. Thank you again for your dedication and the time you have invested in this project.

Article Url - https://www.analyticsvidhya.com/blog/2023/03/cricket-meets-data-science-creating-an-ipl-win-prediction-app/

Harsh Jain