

IPL Data Analysis

March 3, 2021

1 Data analysis for Indian Premier League

IPL is a professional Twenty20 cricket league founded by the Board of Control for Cricket in India (BCCI) in 2008.

The dataset that we use in this notebook is IPL (Indian Premier League) Dataset posted on Kaggle Datasets sourced from cricsheet. (<https://www.kaggle.com/manasgarg/ipl/data>)

The dataset consist of data about IPL matches played from the year 2008 to 2017. It uses two sets of data based from 2008 -2017; match-by-match and ball by ball statistics.

```
[3]: import pandas as pd
```

```
df1 = pd.read_csv (r'matches.csv')  
df1.head()
```

```
[3]:
```

	id	season	city	date	team1 \
0	1	2017	Hyderabad	05-04-2017	Sunrisers Hyderabad
1	2	2017	Pune	06-04-2017	Mumbai Indians
2	3	2017	Rajkot	07-04-2017	Gujarat Lions
3	4	2017	Indore	08-04-2017	Rising Pune Supergiant
4	5	2017	Bangalore	08-04-2017	Royal Challengers Bangalore

		team2	toss_winner	toss_decision \
0	Royal Challengers Bangalore	Royal Challengers Bangalore	field	
1	Rising Pune Supergiant	Rising Pune Supergiant	field	
2	Kolkata Knight Riders	Kolkata Knight Riders	field	
3	Kings XI Punjab	Kings XI Punjab	field	
4	Delhi Daredevils	Royal Challengers Bangalore	bat	

	result	dl_applied	winner	win_by_runs \
0	normal	0	Sunrisers Hyderabad	35
1	normal	0	Rising Pune Supergiant	0
2	normal	0	Kolkata Knight Riders	0
3	normal	0	Kings XI Punjab	0
4	normal	0	Royal Challengers Bangalore	15

	win_by_wickets	player_of_match	venue \
--	----------------	-----------------	---------

0	0	Yuvraj Singh	Rajiv Gandhi International Stadium, Uppal
1	7	SPD Smith	Maharashtra Cricket Association Stadium
2	10	CA Lynn	Saurashtra Cricket Association Stadium
3	6	GJ Maxwell	Holkar Cricket Stadium
4	0	KM Jadhav	M Chinnaswamy Stadium

	umpire1	umpire2	umpire3
0	AY Dandekar	NJ Llong	NaN
1	A Nand Kishore	S Ravi	NaN
2	Nitin Menon	CK Nandan	NaN
3	AK Chaudhary	C Shamsuddin	NaN
4	NaN	NaN	NaN

```
[4]: df1.info()
```

```
<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
```

```
[5]: df1.describe()
```

```
[5]:
```

	id	season	dl_applied	win_by_runs	win_by_wickets	\
count	636.000000	636.000000	636.000000	636.000000	636.000000	
mean	318.500000	2012.490566	0.025157	13.682390	3.372642	
std	183.741666	2.773026	0.156726	23.908877	3.420338	

min	1.000000	2008.000000	0.000000	0.000000	0.000000
25%	159.750000	2010.000000	0.000000	0.000000	0.000000
50%	318.500000	2012.000000	0.000000	0.000000	4.000000
75%	477.250000	2015.000000	0.000000	20.000000	7.000000
max	636.000000	2017.000000	1.000000	146.000000	10.000000

	umpire3
count	0.0
mean	NaN
std	NaN
min	NaN
25%	NaN
50%	NaN
75%	NaN
max	NaN

```
[6]: df1.head(2)
```

```
[6]:   id  season    city    date    team1 \
0   1   2017  Hyderabad  05-04-2017  Sunrisers Hyderabad
1   2   2017    Pune  06-04-2017    Mumbai Indians

      team2    toss_winner toss_decision \
0  Royal Challengers Bangalore  Royal Challengers Bangalore    field
1    Rising Pune Supergiant    Rising Pune Supergiant    field

  result  dl_applied    winner  win_by_runs  win_by_wickets \
0  normal          0  Sunrisers Hyderabad      35          0
1  normal          0  Rising Pune Supergiant      0          7

  player_of_match    venue    umpire1 \
0   Yuvraj Singh  Rajiv Gandhi International Stadium, Uppal    AY Dandekar
1    SPD Smith    Maharashtra Cricket Association Stadium  A Nand Kishore

  umpire2  umpire3
0  NJ Llong    NaN
1   S Ravi    NaN
```

2 How many matches we've got in the dataset?

```
[7]: #matches, _=df1.shape
matches=df1['id'].max()
print("How many matches we've got in the dataset?\nAns:",matches)
```

How many matches we've got in the dataset?

Ans: 636

3 How many seasons we've got in the dataset?

```
[8]: #no_of_seasons=len(set(df1.season))
no_of_seasons=len(df1['season'].unique())
print("How many seasons we've got in the dataset?\nAns:",no_of_seasons)
```

How many seasons we've got in the dataset?

Ans: 10

4 Which Team had won by maximum runs?

```
[48]: # maximum_run_win=max(df1['win_by_runs'])
# for i in range(matches):
#     if df1.win_by_runs[i]==maximum_run_win:
#         print("Which Team had won by maximum runs?\nAns:",df1.winner[i],"won_
→by",maximum_run_win,"runs")

#df1.iloc[df1['win_by_runs'].idxmax()]
#idxmax will return the id of the maximumth value
#iloc takes an index value and returns the row.

df1.iloc[df1['win_by_runs'].idxmax()]['winner']
```

[48]: 'Mumbai Indians'

5 Which Team had won by maximum wicket?

```
[10]: max_wicket_win=max(df1['win_by_wickets'])
for i in range(matches):
    if df1.win_by_wickets[i]==max_wicket_win:
        print("Which Team had won by maximum wickets?\nAns:",df1.winner[i],"won_
→by",max_wicket_win,"wickets\n")

# df1.iloc[df1['win_by_wickets'].idxmax()]['winner']
```

Which Team had won by maximum wickets?

Ans: Kolkata Knight Riders won by 10 wickets

Which Team had won by maximum wickets?

Ans: Kings XI Punjab won by 10 wickets

Which Team had won by maximum wickets?

Ans: Deccan Chargers won by 10 wickets

Which Team had won by maximum wickets?

Ans: Delhi Daredevils won by 10 wickets

Which Team had won by maximum wickets?

Ans: Royal Challengers Bangalore won by 10 wickets

Which Team had won by maximum wickets?

Ans: Rajasthan Royals won by 10 wickets

Which Team had won by maximum wickets?

Ans: Mumbai Indians won by 10 wickets

Which Team had won by maximum wickets?

Ans: Chennai Super Kings won by 10 wickets

Which Team had won by maximum wickets?

Ans: Royal Challengers Bangalore won by 10 wickets

Which Team had won by maximum wickets?

Ans: Sunrisers Hyderabad won by 10 wickets

6 Which Team had won by closest Margin (minimum runs)?

```
[49]: # minimum_run_win=min(df1['win_by_runs']>0)
# for i in range(matches):
#     if df1.win_by_runs[i]==minimum_run_win:
#         print("Which Team had won by maximum runs?\nAns:",df1.winner[i],"won_
→by",minimum_run_win,"runs")

matches=df1
# matches.iloc[matches[matches['win_by_runs'].gt(0)].win_by_runs.
→idxmin()][ 'winner']
```

```
[50]: matches.iloc[matches[matches['win_by_runs'].ge(1)].win_by_runs.
→idxmin()][ 'winner']
#matches[matches['win_by_runs'].gt(0)].win_by_runs.idxmin()
```

```
[50]: 'Mumbai Indians'
```

7 Which Team had won by minimum wicket?

```
[13]: matches.iloc[matches[matches['win_by_wickets'].ge(1)].win_by_wickets.
→idxmin()][ 'winner']
```

```
[13]: 'Kolkata Knight Riders'
```

8 Which season had most number of matches?

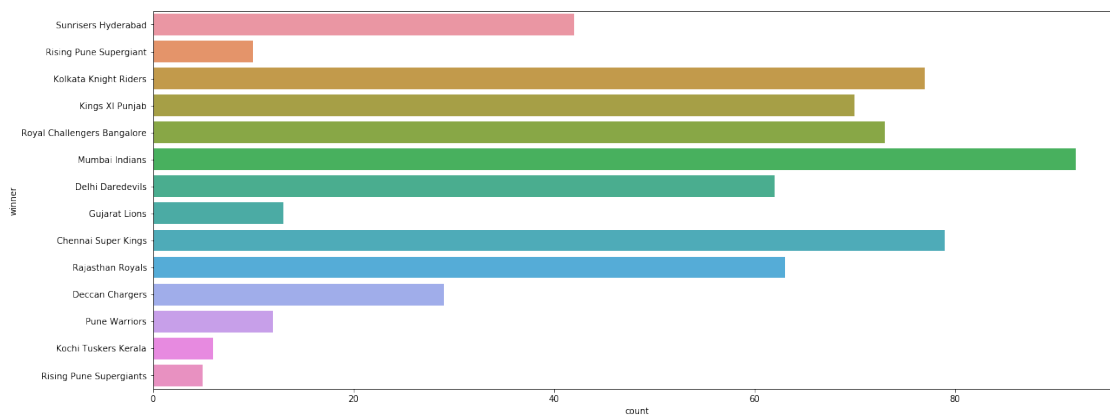
```
[14]: import matplotlib.pyplot as plt #visualization
import seaborn as sns #modern visualization
plt.rcParams['figure.figsize'] = (20, 8)

sns.countplot(x='season',data=matches)
#Show the counts of observations in each categorical bin using bars.
plt.show()
```



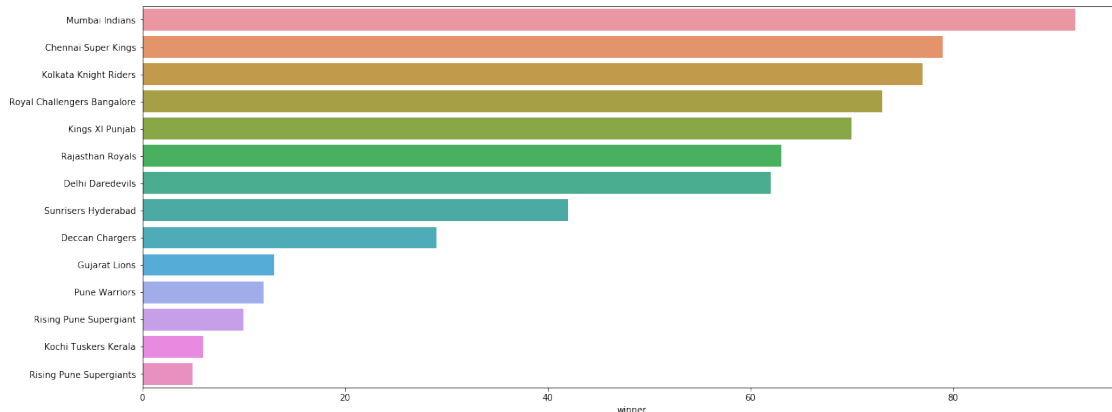
9 The most successful IPL Team

```
[15]: sns.countplot(y='winner',data=matches)
#Show the counts of observations in each categorical bin using bars.
plt.show()
```



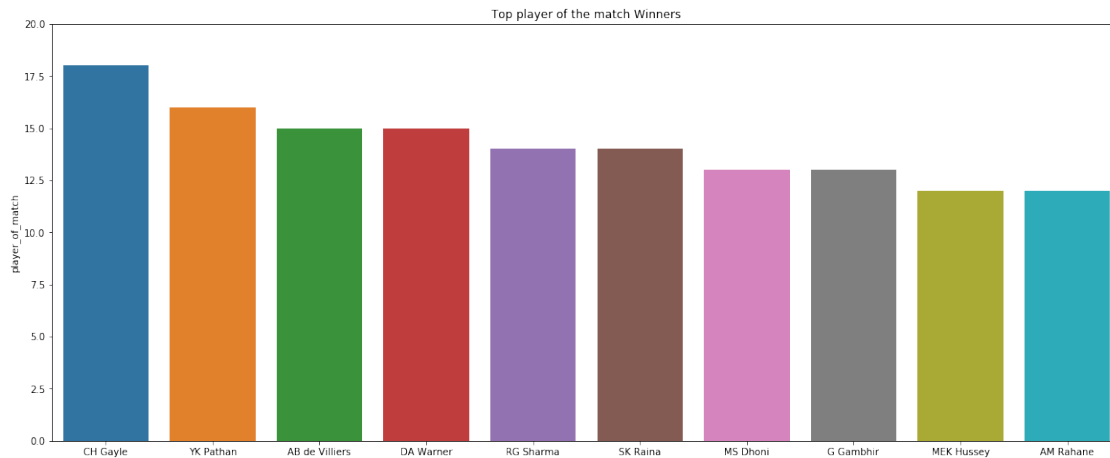
10 The most successful IPL Team (descending order)

```
[16]: # sns.countplot(y='winner', data = matches)
# plt.show()
data = matches.winner.value_counts()
sns.barplot(y = data.index, x = data, orient='h');
```



11 Top player of the match Winners

```
[17]: top_players = matches.player_of_match.value_counts()[:10]
#sns.barplot(x="day", y="total_bill", data=top_playersps)
fig, ax = plt.subplots()
ax.set_ylim([0,20])
ax.set_ylabel("Count")
ax.set_title("Top player of the match Winners")
#top_players.plot.bar()
sns.barplot(x = top_players.index, y = top_players, orient='v');
→#palette="Blues");
plt.show()
```

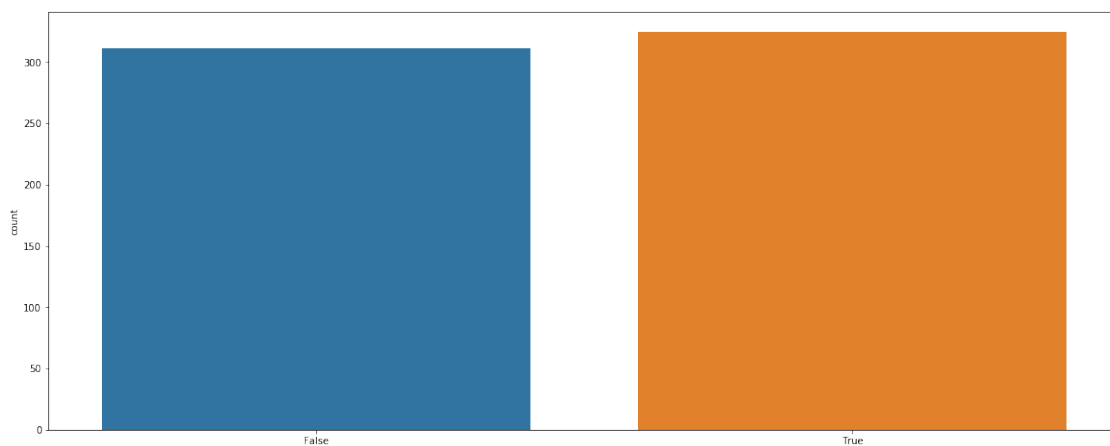


12 Has Toss-winning helped in Match-winning?

```
[18]: # (df1[df1['toss_winner']==df1['winner']].count())/636
sns.countplot(df1['toss_winner']==df1['winner'])

ss = matches['toss_winner'] == matches['winner']
#print(ss)
print(ss.groupby(ss).size()/636)
```

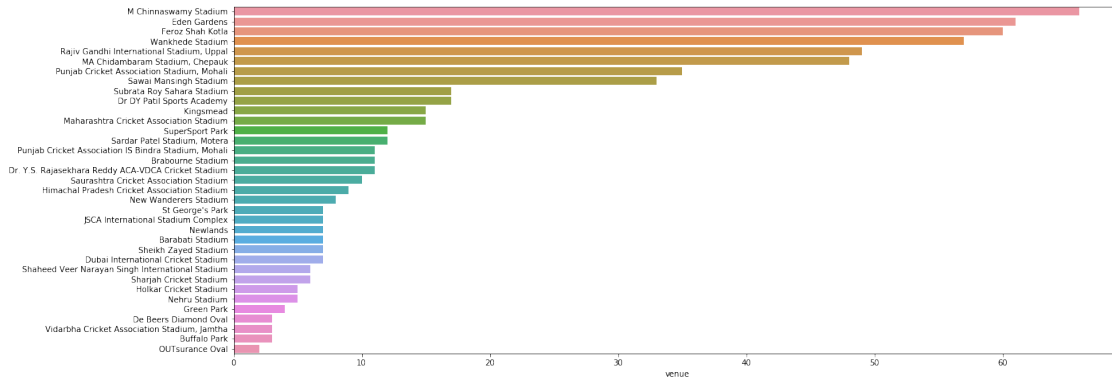
```
False    0.488994
True     0.511006
dtype: float64
```



13 Number of matches in each venue

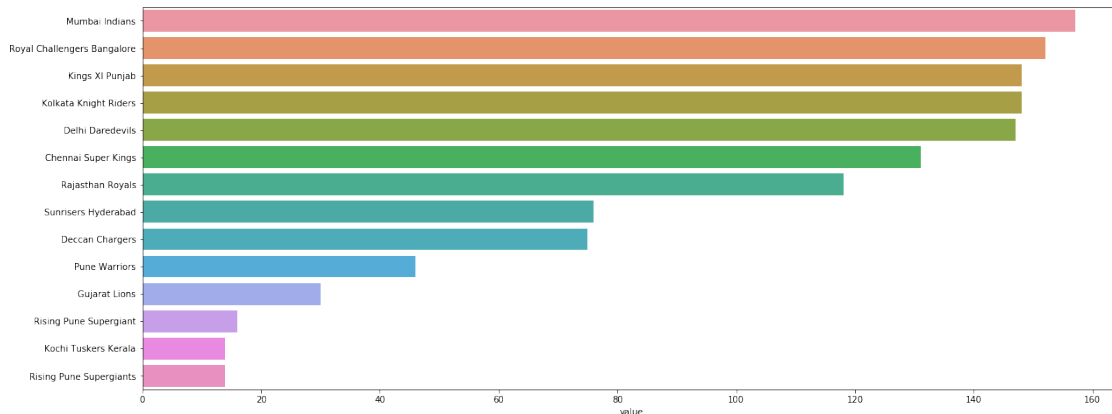
```
[19]: #1.      Number of matches in each venue
data = matches.venue.value_counts()
print(data)
sns.barplot(y = data.index, x = data, orient='h');
```

M Chinnaswamy Stadium	66
Eden Gardens	61
Feroz Shah Kotla	60
Wankhede Stadium	57
Rajiv Gandhi International Stadium, Uppal	49
MA Chidambaram Stadium, Chepauk	48
Punjab Cricket Association Stadium, Mohali	35
Sawai Mansingh Stadium	33
Subrata Roy Sahara Stadium	17
Dr DY Patil Sports Academy	17
Kingsmead	15
Maharashtra Cricket Association Stadium	15
SuperSport Park	12
Sardar Patel Stadium, Motera	12
Punjab Cricket Association IS Bindra Stadium, Mohali	11
Brabourne Stadium	11
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium	11
Saurashtra Cricket Association Stadium	10
Himachal Pradesh Cricket Association Stadium	9
New Wanderers Stadium	8
St George's Park	7
JSCA International Stadium Complex	7
Newlands	7
Barabati Stadium	7
Sheikh Zayed Stadium	7
Dubai International Cricket Stadium	7
Shaheed Veer Narayan Singh International Stadium	6
Sharjah Cricket Stadium	6
Holkar Cricket Stadium	5
Nehru Stadium	5
Green Park	4
De Beers Diamond Oval	3
Vidarbha Cricket Association Stadium, Jamtha	3
Buffalo Park	3
OUTsurance Oval	2
Name: venue, dtype: int64	



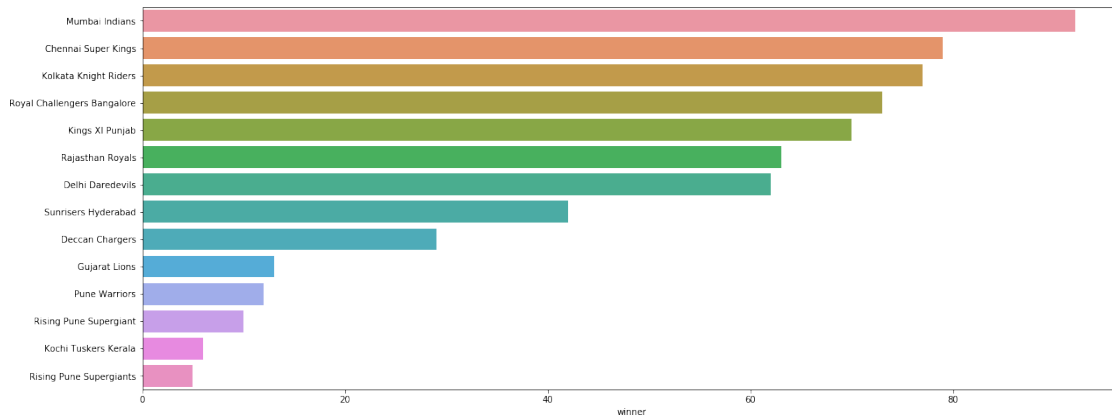
14 Number of matches played by each team

```
[20]: temp_df = pd.melt(df1, id_vars=['id', 'season'], value_vars=['team1', 'team2'])
data = temp_df.value.value_counts()
sns.barplot(y = data.index, x = data, orient='h');
```



15 Number of wins per team

```
[21]: #Number of wins per team
data=df1.winner.value_counts()
sns.barplot(y = data.index, x = data, orient='h');
```



16 Champions each season

[22]: *#Champions each season*

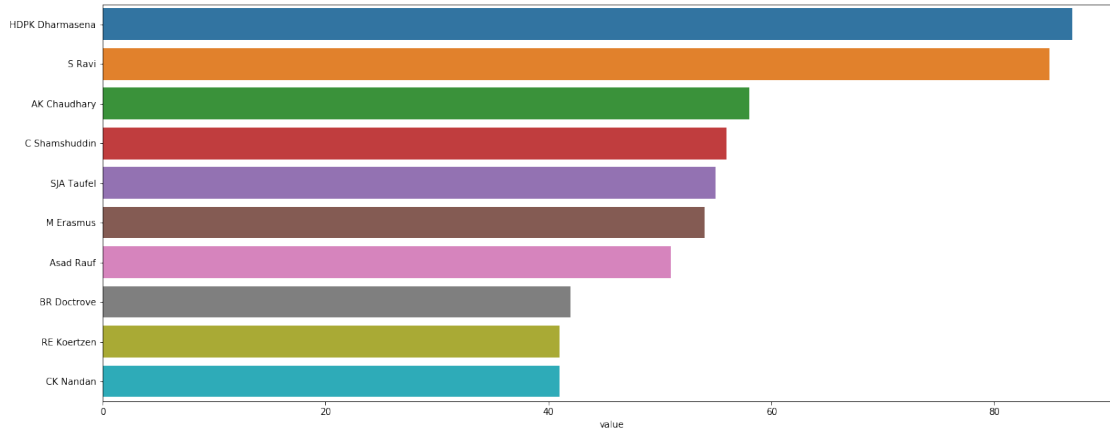
```
#DataFrame.drop_duplicates(subset=None, keep='first', inplace=False,
    → ignore_index=False)

temp_df = df1.drop_duplicates(subset=['season'], keep='last')[['season',
    → 'winner']].reset_index(drop=True)
print((temp_df))
```

	season	winner
0	2017	Mumbai Indians
1	2008	Rajasthan Royals
2	2009	Deccan Chargers
3	2010	Chennai Super Kings
4	2011	Chennai Super Kings
5	2012	Kolkata Knight Riders
6	2013	Mumbai Indians
7	2014	Kolkata Knight Riders
8	2015	Mumbai Indians
9	2016	Sunrisers Hyderabad

17 Top Umpires

```
[23]: temp_df = pd.melt(df1, value_vars=['umpire1', 'umpire2', 'umpire3'])
    #print(temp_df)
    data = (temp_df.value.value_counts())[10]
    sns.barplot(y = data.index, x = data, orient='h');
```



—> Batsman analysis

```
[24]: import pandas as pd
import matplotlib.pyplot as plt #visualization
import seaborn as sns #modern visualization
score_df = pd.read_csv(r'deliveries.csv')
score_df.head()
```

```
[24]: match_id  inning  batting_team  bowling_team  over  \
0          1        1  Sunrisers Hyderabad  Royal Challengers Bangalore  1
1          1        1  Sunrisers Hyderabad  Royal Challengers Bangalore  1
2          1        1  Sunrisers Hyderabad  Royal Challengers Bangalore  1
3          1        1  Sunrisers Hyderabad  Royal Challengers Bangalore  1
4          1        1  Sunrisers Hyderabad  Royal Challengers Bangalore  1
```

```
ball  batsman non_striker  bowler  is_super_over  ...  bye_runs  \
0     1  DA Warner      S Dhawan  TS Mills          0  ...      0
1     2  DA Warner      S Dhawan  TS Mills          0  ...      0
2     3  DA Warner      S Dhawan  TS Mills          0  ...      0
3     4  DA Warner      S Dhawan  TS Mills          0  ...      0
4     5  DA Warner      S Dhawan  TS Mills          0  ...      0
```

```
legbye_runs  noball_runs  penalty_runs  batsman_runs  extra_runs  \
0           0           0           0           0           0
1           0           0           0           0           0
2           0           0           0           4           0
3           0           0           0           0           0
4           0           0           0           0           2
```

```
total_runs  player_dismissed  dismissal_kind  fielder
0           0                NaN            NaN    NaN
1           0                NaN            NaN    NaN
```

2	4	NaN	NaN	NaN
3	0	NaN	NaN	NaN
4	2	NaN	NaN	NaN

[5 rows x 21 columns]

18 Which batsmen has scored maximum runs?

```
[26]: temp_df = score_df.groupby('batsman')['batsman_runs'].agg('sum').reset_index().
      →sort_values(by='batsman_runs', ascending=False).reset_index(drop=True)

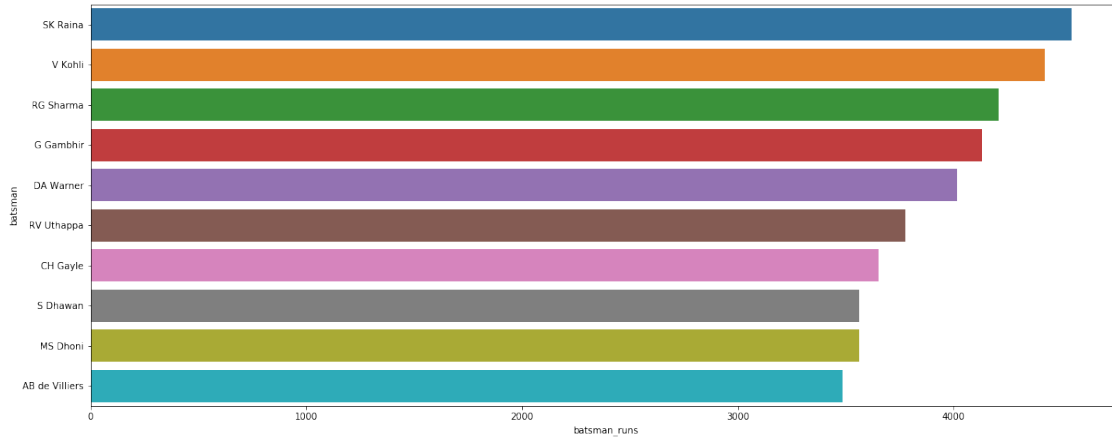
temp_df = temp_df.iloc[:10,:]
print(temp_df)

# labels = np.array(temp_df['batsman'])
# ind = np.arange(len(labels))
# width = 0.9
# fig, ax = plt.subplots()
# rects = ax.bar(ind, np.array(temp_df['batsman_runs']), width=width,
→color='blue')
# ax.set_xticks(ind+((width)/2.))
# ax.set_xticklabels(labels, rotation='vertical')
# ax.set_ylabel("Count")
# ax.set_title("Top run scorers in IPL")
# autolabel(rects)
# plt.show()

plt.rcParams['figure.figsize'] = (20, 8)
g=sns.barplot(y = 'batsman', x = 'batsman_runs', data=temp_df);

# for index, row in score_df.iterrows():
#     g.text(row.name,row.batsman_runs, round(row.batsman_runs,2),
→color='black', ha="center")
```

	batsman	batsman_runs
0	SK Raina	4548
1	V Kohli	4423
2	RG Sharma	4207
3	G Gambhir	4132
4	DA Warner	4014
5	RV Uthappa	3778
6	CH Gayle	3651
7	S Dhawan	3561
8	MS Dhoni	3560
9	AB de Villiers	3486

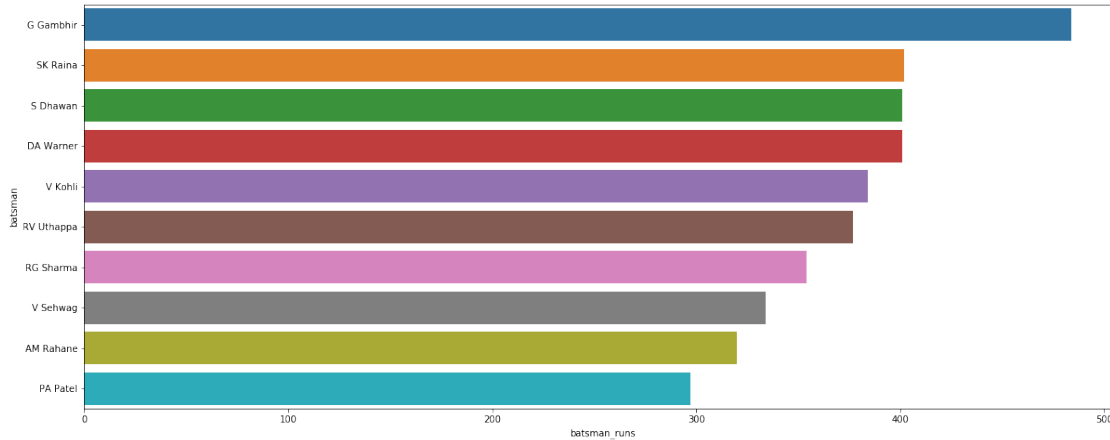


19 Players with more number of boundaries (Fours) in IPL.

```
[27]: temp_df = score_df.groupby('batsman')['batsman_runs'].agg(lambda x: (x==4) .
    ↳sum()).reset_index().sort_values(by='batsman_runs', ascending=False).
    ↳reset_index(drop=True)
temp_df = temp_df.iloc[:10,:]
print(temp_df)

g=sns.barplot(y = 'batsman', x = 'batsman_runs', data=temp_df);
```

	batsman	batsman_runs
0	G Gambhir	484
1	SK Raina	402
2	S Dhawan	401
3	DA Warner	401
4	V Kohli	384
5	RV Uthappa	377
6	RG Sharma	354
7	V Sehwag	334
8	AM Rahane	320
9	PA Patel	297

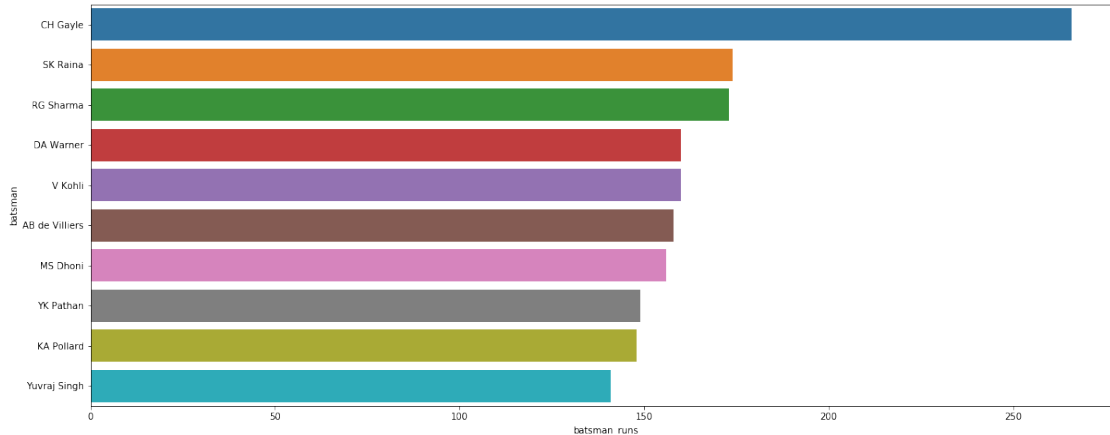


20 Players with more number of boundaries (Sixes) in IPL.

```
[28]: temp_df = score_df.groupby('batsman')['batsman_runs'].agg(lambda x: (x==6) .
    ↳sum()).reset_index().sort_values(by='batsman_runs', ascending=False).
    ↳reset_index(drop=True)
temp_df = temp_df.iloc[:10,:]
print(temp_df)

g=sns.barplot(y = 'batsman', x = 'batsman_runs', data=temp_df);
```

	batsman	batsman_runs
0	CH Gayle	266
1	SK Raina	174
2	RG Sharma	173
3	DA Warner	160
4	V Kohli	160
5	AB de Villiers	158
6	MS Dhoni	156
7	YK Pathan	149
8	KA Pollard	148
9	Yuvraj Singh	141



Helper function to write value above the bar in barplot.

```
[29]: def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.02*height,
                '%d' % int(height),
                ha='center', va='bottom')
```

21 Batsmen played most number of dot balls, singles, doubles, triples, Fours, fives and Sixes.

```
[30]: for i in range(7):
    temp_df = score_df.groupby('batsman')['batsman_runs'].agg(lambda x: (x==i).
    →sum()).reset_index().sort_values(by='batsman_runs', ascending=False).
    →reset_index(drop=True)
    temp_df = temp_df.iloc[:20,:]

    labels = np.array(temp_df['batsman'])
    ind = np.arange(len(labels))
    width = 0.2

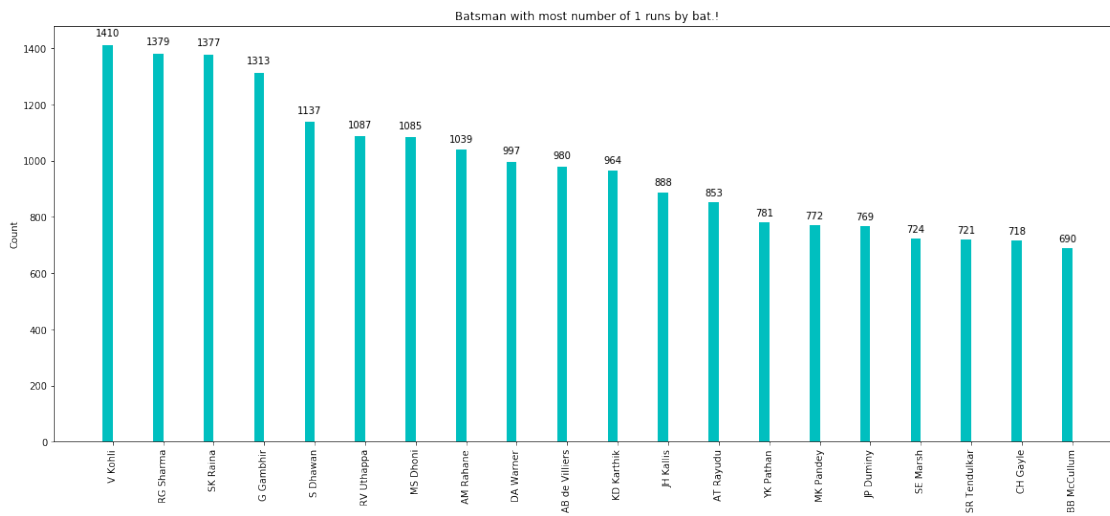
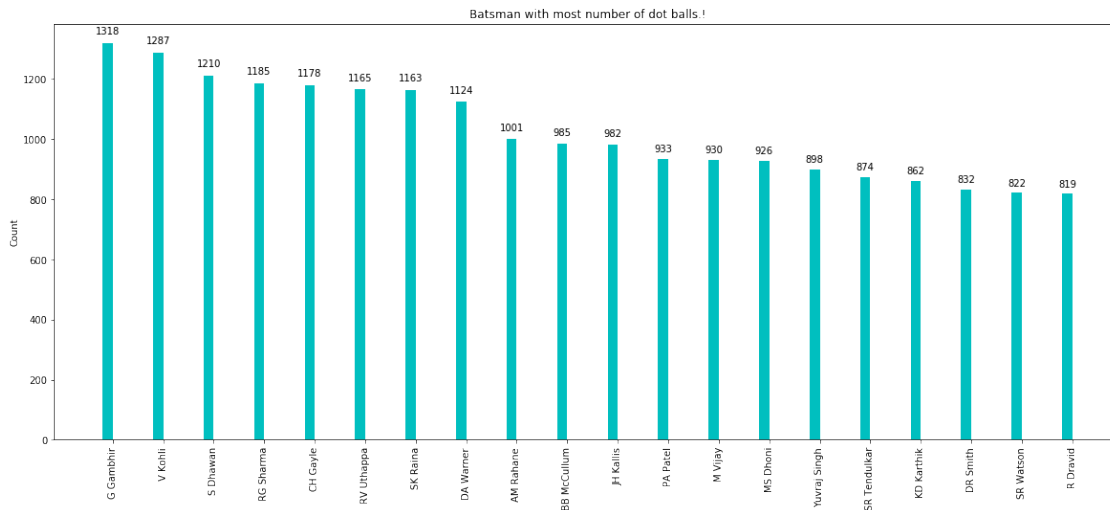
    fig, ax = plt.subplots()
    rects = ax.bar(ind, np.array(temp_df['batsman_runs']), width=width,
    →color='c')
    ax.set_xticks(ind+((width)/2.))
    ax.set_xticklabels(labels, rotation='vertical')
    ax.set_ylabel("Count")
    if i==0:
        ax.set_title("Batsman with most number of dot balls.!")
```

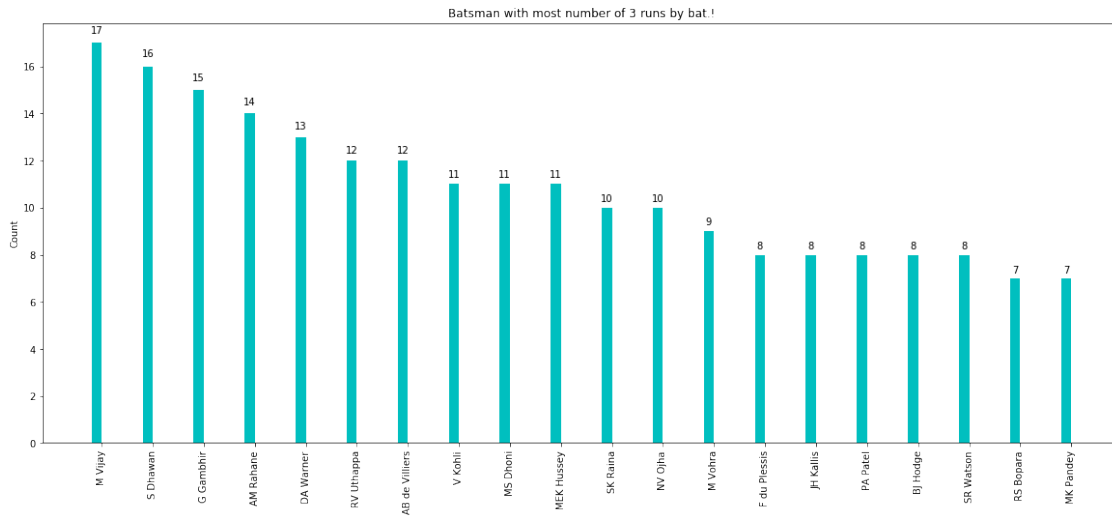
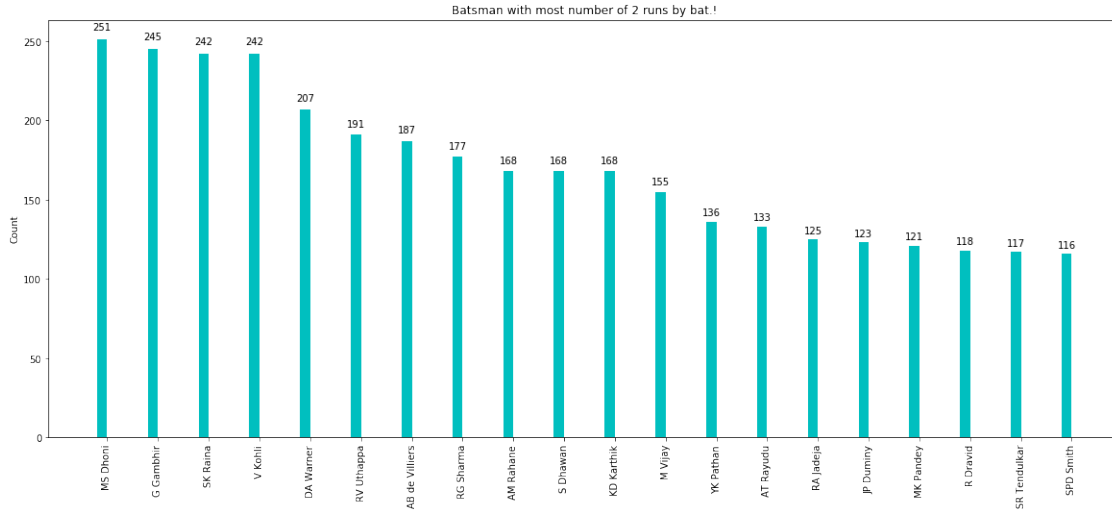


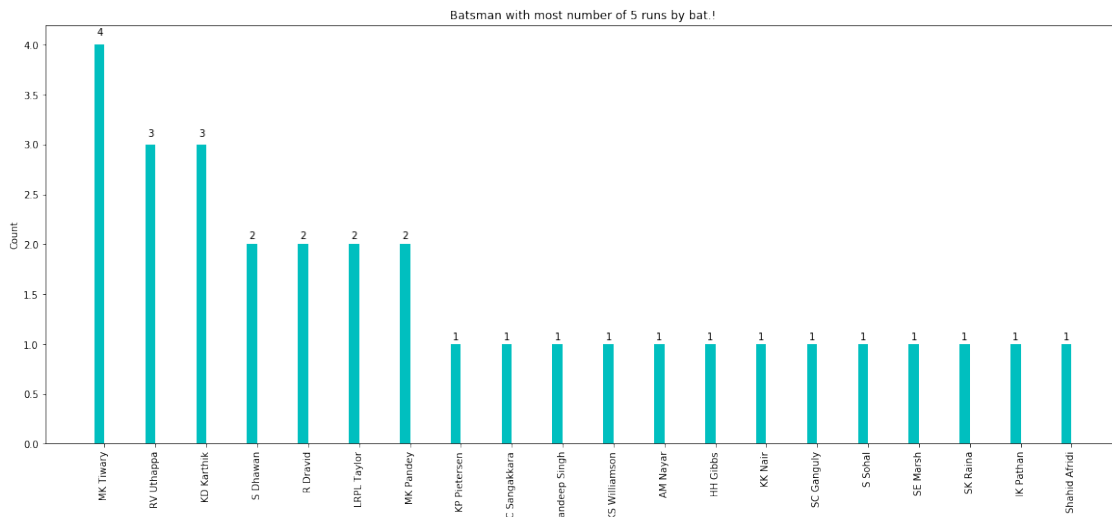
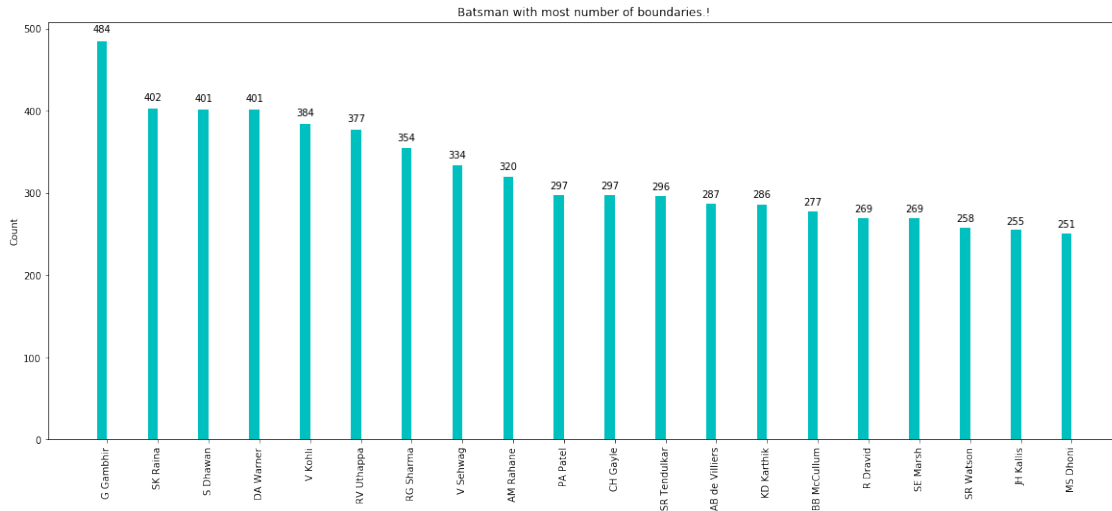
```

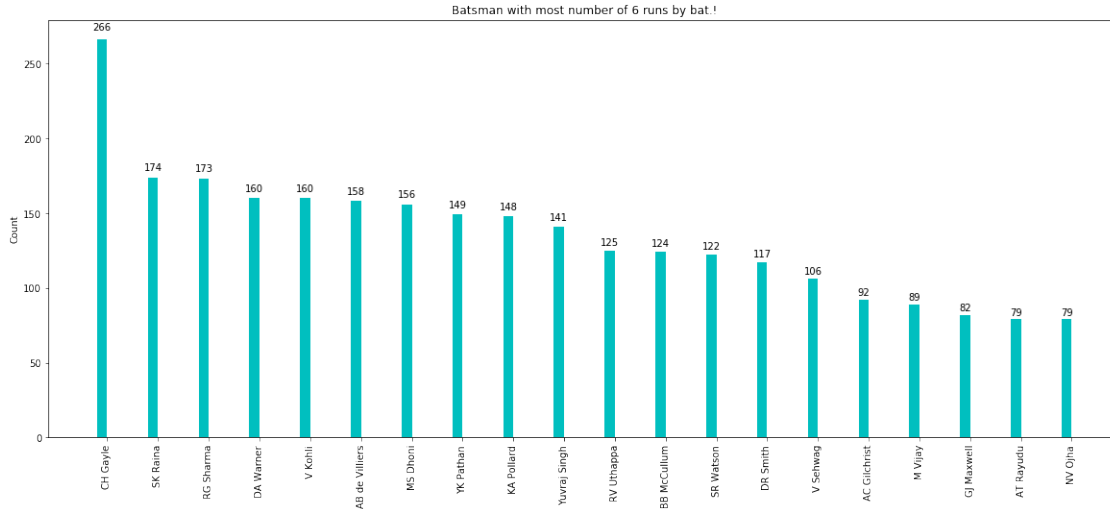
elif i==4:
    ax.set_title("Batsman with most number of boundaries.!")
else:
    ax.set_title("Batsman with most number of "+str(i)+" runs by bat.!")
autolabel(rects)
plt.show()

```



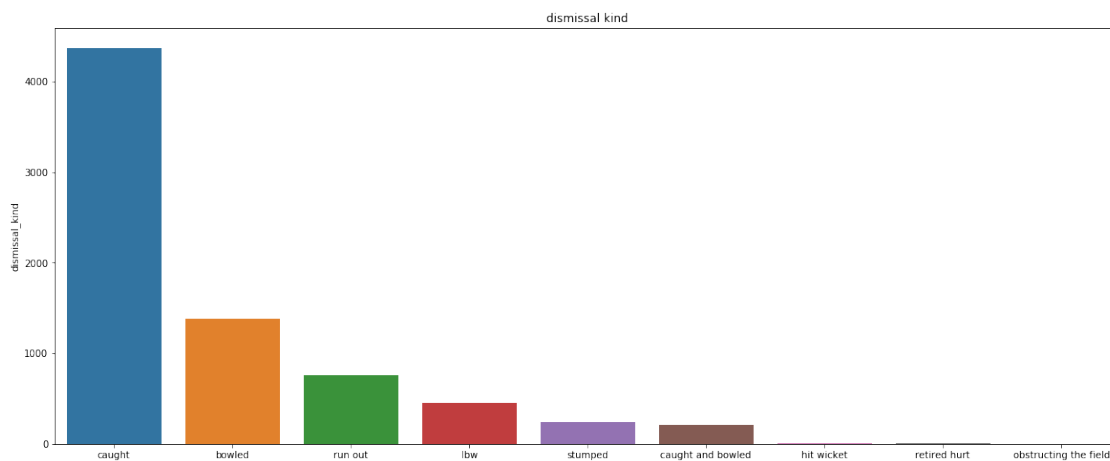






22 Most common dismissal types in IPL

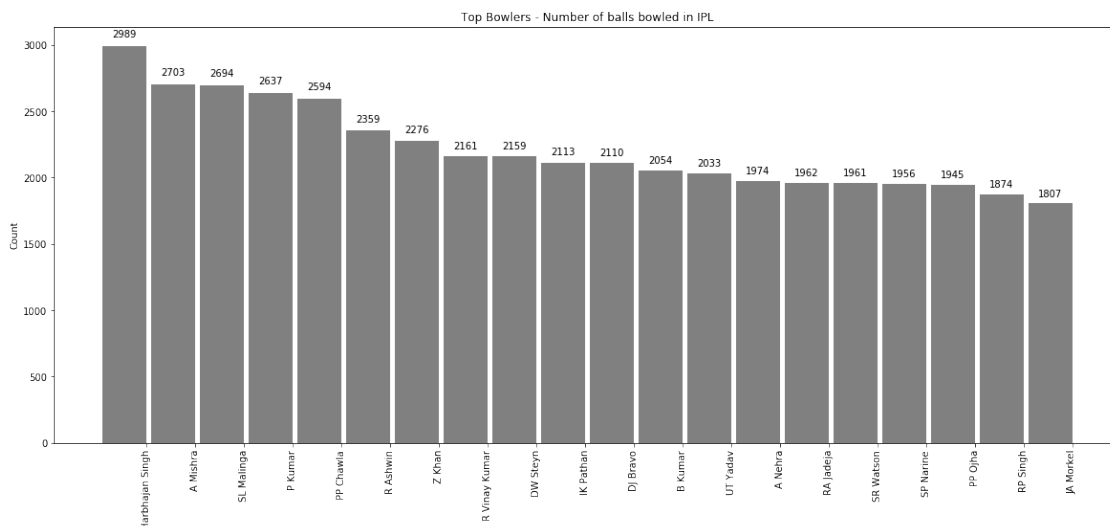
```
[31]: dismissal = score_df.dismissal_kind.value_counts()[:10]
#sns.barplot(x="day", y="total_bill", datatop_playersps)
fig, ax = plt.subplots()
#ax.set_ylim([0,20])
ax.set_ylabel("Count")
ax.set_title("dismissal kind")
#top_players.plot.bar()
sns.barplot(x = dismissal.index, y = dismissal, orient='v'); #palette="Blues");
plt.show()
```



23 Top Bowlers - By number of balls bowled in IPL

```
[32]: temp_df = score_df.groupby('bowler')['ball'].agg('count').reset_index().
      →sort_values(by='ball', ascending=False).reset_index(drop=True)
temp_df = temp_df.iloc[:20,:]
#print(temp_df)

labels = np.array(temp_df['bowler'])
ind = np.arange(len(labels))
width = 0.9
fig, ax = plt.subplots()
rects = ax.bar(ind, np.array(temp_df['ball']), width=width, color='grey')
ax.set_xticks(ind+((width)/2.))
ax.set_xticklabels(labels, rotation='vertical')
ax.set_ylabel("Count")
ax.set_title("Top Bowlers - Number of balls bowled in IPL")
autolabel(rects)
plt.show()
```



24 Top Bowlers - By number of dot balls bowled in IPL

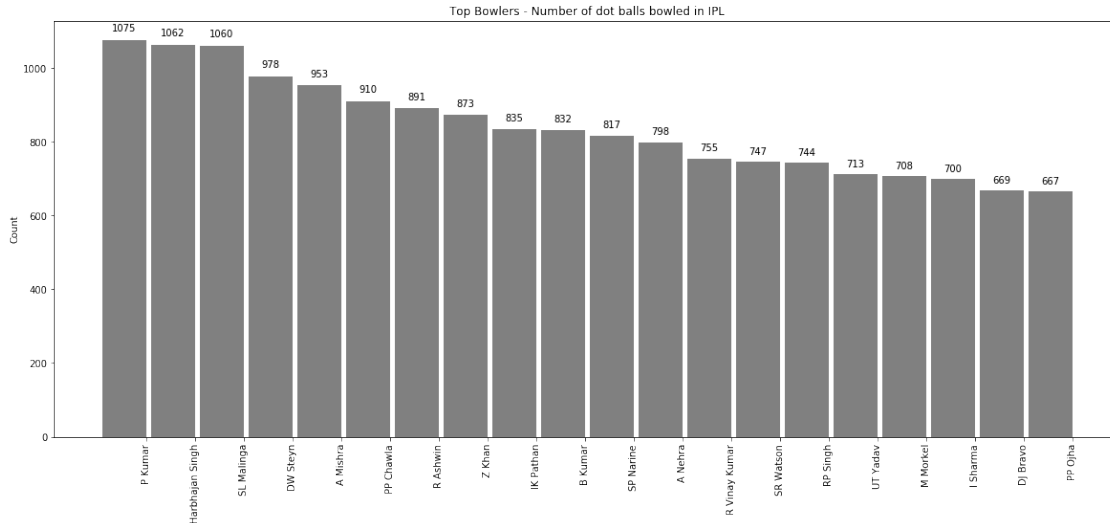
```
[33]: temp_df = score_df.groupby('bowler')['total_runs'].agg(lambda x: (x==0).sum()).
      →reset_index().sort_values(by='total_runs', ascending=False).
      →reset_index(drop=True)
temp_df = temp_df.iloc[:20,:]

labels = np.array(temp_df['bowler'])
ind = np.arange(len(labels))
```

```

width = 0.9
fig, ax = plt.subplots()
rects = ax.bar(ind, np.array(temp_df['total_runs']), width=width, color='grey')
ax.set_xticks(ind+((width)/2.))
ax.set_xticklabels(labels, rotation='vertical')
ax.set_ylabel("Count")
ax.set_title("Top Bowlers - Number of dot balls bowled in IPL")
autolabel(rects)
plt.show()

```



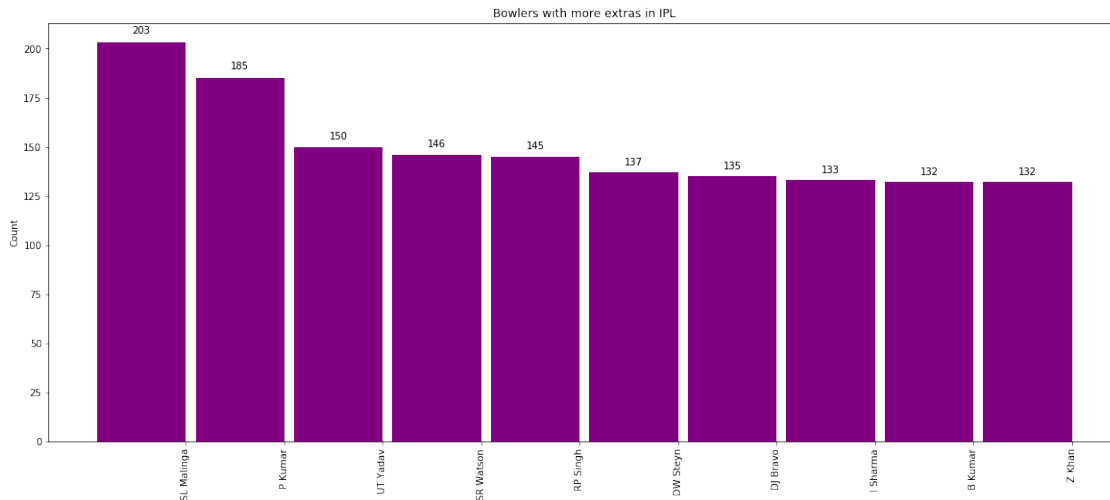
25 Bowlers who gave more extras in IPL

```

[34]: temp_df = score_df.groupby('bowler')['extra_runs'].agg(lambda x: (x>0).sum()).
      →reset_index().sort_values(by='extra_runs', ascending=False).
      →reset_index(drop=True)
temp_df = temp_df.iloc[:10,:]

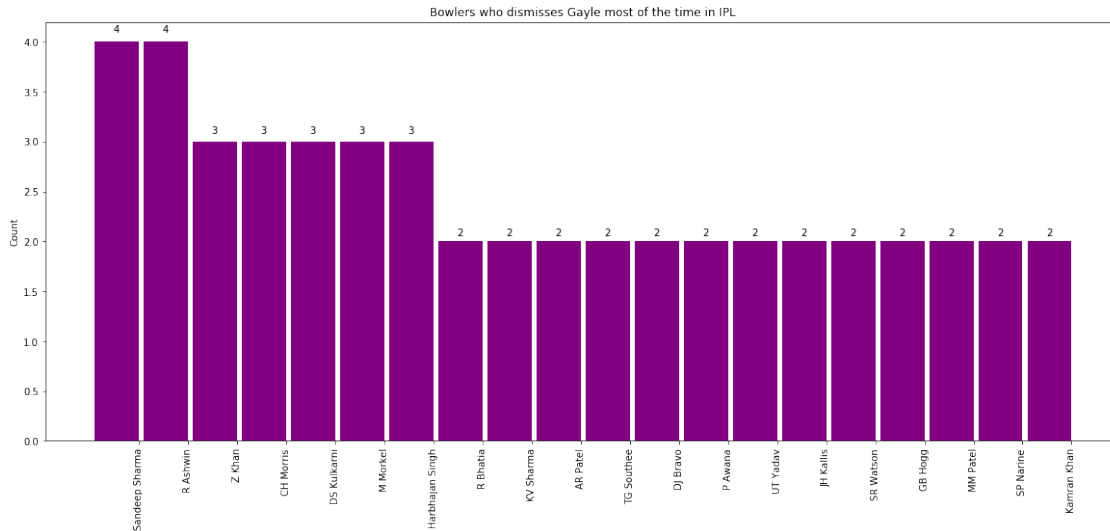
labels = np.array(temp_df['bowler'])
ind = np.arange(len(labels))
width = 0.9
fig, ax = plt.subplots()
rects = ax.bar(ind, np.array(temp_df['extra_runs']), width=width, color='purple')
ax.set_xticks(ind+((width)/2.))
ax.set_xticklabels(labels, rotation='vertical')
ax.set_ylabel("Count")
ax.set_title("Bowlers with more extras in IPL")
autolabel(rects)
plt.show()

```



```
[35]: temp_df = score_df.groupby('bowler')['player_dismissed']. \
agg(lambda x: (x=='CH Gayle').sum()).reset_index().
    ↳sort_values(by='player_dismissed', ascending=False).reset_index(drop=True)
temp_df = temp_df.iloc[:20,:]

labels = np.array(temp_df['bowler'])
ind = np.arange(len(labels))
width = 0.9
fig, ax = plt.subplots()
rects = ax.bar(ind, np.array(temp_df['player_dismissed']), width=width,
    ↳color='purple')
ax.set_xticks(ind+((width)/2.))
ax.set_xticklabels(labels, rotation='vertical')
ax.set_ylabel("Count")
ax.set_title("Bowlers who dismisses Gayle most of the time in IPL")
autolabel(rects)
plt.show()
```

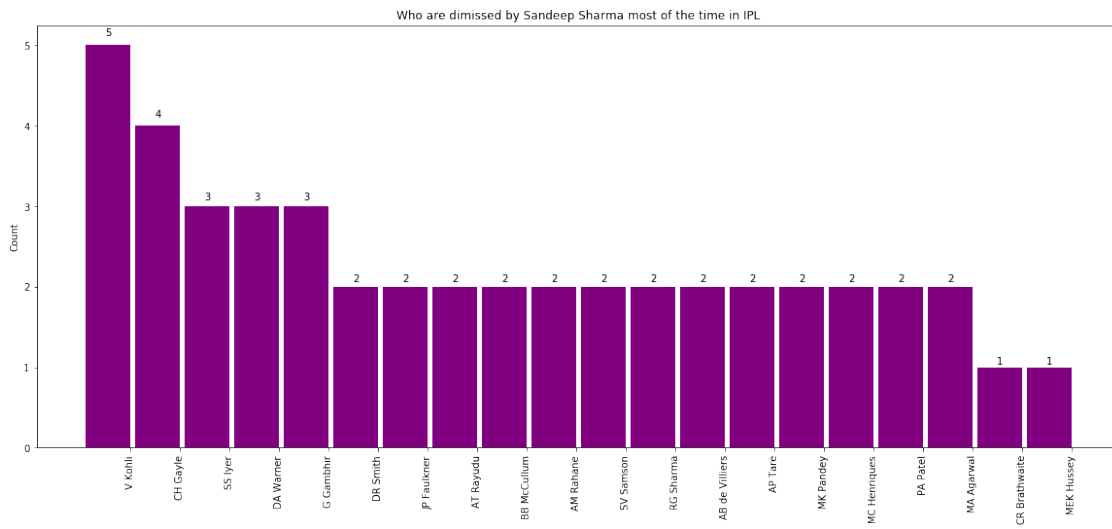
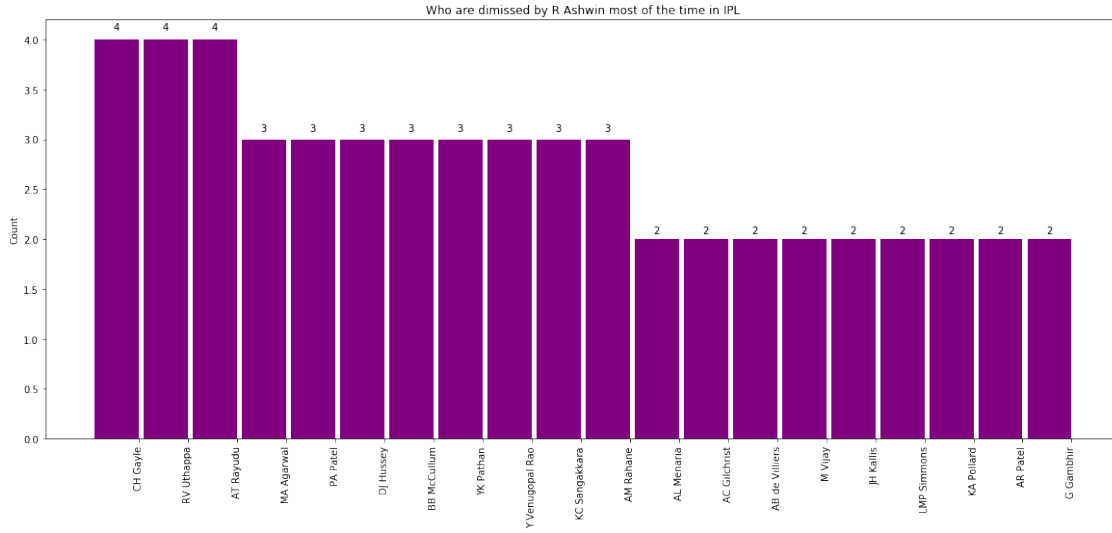


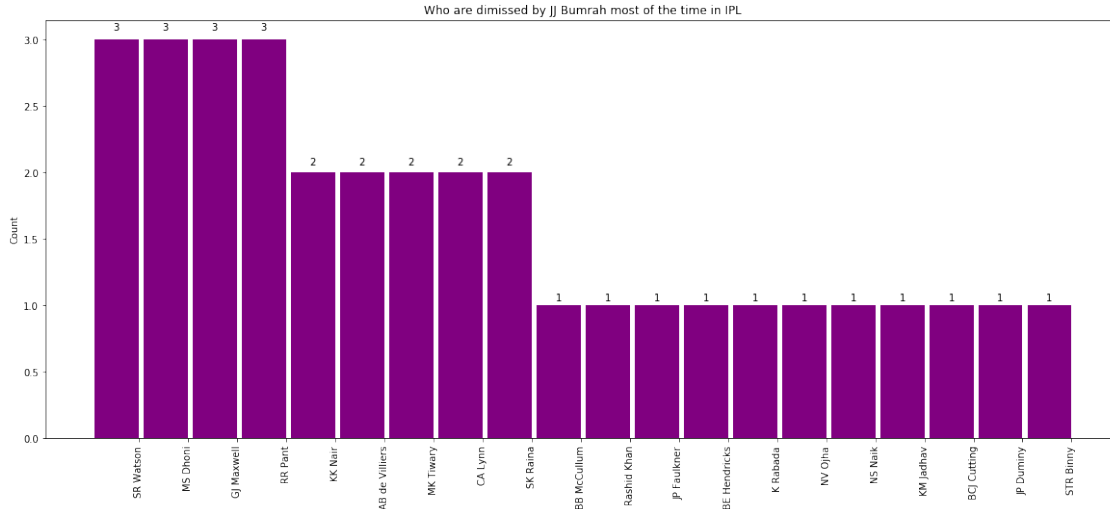
26 Which Bowler has dismissed which batsmen most of the time

```
[36]: for i in ['R Ashwin', 'Sandeep Sharma', 'JJ Bumrah']:
    temp_df = score_df.groupby('player_dismissed')['bowler']. \
        agg(lambda x: (x==i).sum()).reset_index().sort_values(by='bowler',
        ↪ascending=False).reset_index(drop=True)
    temp_df = temp_df.iloc[:20,: ]

    labels = np.array(temp_df['player_dismissed'])
    ind = np.arange(len(labels))
    width = 0.9
    fig, ax = plt.subplots()
    rects = ax.bar(ind, np.array(temp_df['bowler']), width=width, color='purple')

    ax.set_xticks(ind+((width)/2.))
    ax.set_xticklabels(labels, rotation='vertical')
    ax.set_ylabel("Count")
    ax.set_title("Who are dismissed by "+i+" most of the time in IPL")
    autolabel(rects)
    plt.show()
```



27 Predictive Analysis of an IPL Match

1. Prediction using SVM Binary Classifier.
2. Since the output of winner prediction is a categorical value, the problem which we are trying to solve is a Classification problem.

Steps 1. Understand the dataset. 2. Clean the data. 3. Analyze the candidate columns to be Features. 4. Process the features as required by the model/algorithm. 5. Train the model/algorithm on training data. 6. Test the model/algorithm on testing data. 7. Tune the model/algorithm for higher accuracy.

```
[37]: import pandas as pd
import matplotlib.pyplot as plt #visualization
import seaborn as sns #modern visualization
plt.rcParams['figure.figsize'] = (20, 8)
```

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

```
[37]:   id  season   city   date                                team1 \
0   1   2017  Hyderabad  05-04-2017          Sunrisers Hyderabad
1   2   2017    Pune  06-04-2017            Mumbai Indians
2   3   2017   Rajkot  07-04-2017            Gujarat Lions
3   4   2017   Indore  08-04-2017      Rising Pune Supergiant
4   5   2017  Bangalore  08-04-2017  Royal Challengers Bangalore

                                team2      toss_winner toss_decision \
0  Royal Challengers Bangalore  Royal Challengers Bangalore      field
```

1	Rising Pune Supergiant	Rising Pune Supergiant	field
2	Kolkata Knight Riders	Kolkata Knight Riders	field
3	Kings XI Punjab	Kings XI Punjab	field
4	Delhi Daredevils	Royal Challengers Bangalore	bat

	result	dl_applied	winner	win_by_runs	\
0	normal	0	Sunrisers Hyderabad	35	
1	normal	0	Rising Pune Supergiant	0	
2	normal	0	Kolkata Knight Riders	0	
3	normal	0	Kings XI Punjab	0	
4	normal	0	Royal Challengers Bangalore	15	

	win_by_wickets	player_of_match	venue	\
0	0	Yuvraj Singh	Rajiv Gandhi International Stadium, Uppal	
1	7	SPD Smith	Maharashtra Cricket Association Stadium	
2	10	CA Lynn	Saurashtra Cricket Association Stadium	
3	6	GJ Maxwell	Holkar Cricket Stadium	
4	0	KM Jadhav	M Chinnaswamy Stadium	

	umpire1	umpire2	umpire3
0	AY Dandekar	NJ Llong	NaN
1	A Nand Kishore	S Ravi	NaN
2	Nitin Menon	CK Nandan	NaN
3	AK Chaudhary	C Shamshuddin	NaN
4	NaN	NaN	NaN

```
[2]: #matches.isna().any()
null_columns=matches.isnull().sum()
#print(null_columns)
print(null_columns[null_columns > 0])
```

```
city          7
winner        3
player_of_match  3
umpire1       1
umpire2       1
umpire3      636
dtype: int64
```

The data that we have, contains null values in several columns. There are several ways to handle the null values and among them, I will be using Imputation on column city. Imputation is a way to fill the missing values statistically.

```
[38]: #imputing the values in column city based on venue
conditions = [matches["venue"] == "Rajiv Gandhi International Stadium, Uppal",\
              matches["venue"] == "Maharashtra Cricket Association Stadium",\
              matches["venue"] == "Saurashtra Cricket Association Stadium",\
              matches["venue"] == "Holkar Cricket Stadium",\
```

```

matches["venue"] == "M Chinnaswamy Stadium",\
matches["venue"] == "Wankhede Stadium",\
matches["venue"] == "Eden Gardens",matches["venue"] == "Feroz Shah,
→Kotla",\
matches["venue"] == "Punjab Cricket Association IS Bindra Stadium,
→Mohali",\
matches["venue"] == "Green Park",\
matches["venue"] == "Punjab Cricket Association Stadium, Mohali",\
matches["venue"] == "Dr DY Patil Sports Academy",\
matches["venue"] == "Sawai Mansingh Stadium", \
matches["venue"] == "MA Chidambaram Stadium, Chepauk",
matches["venue"] == "Newlands", \
matches["venue"] == "St George's Park",\
matches["venue"] == "Kingsmead", \
matches["venue"] == "SuperSport Park",
matches["venue"] == "Buffalo Park", \
matches["venue"] == "New Wanderers Stadium",\
matches["venue"] == "De Beers Diamond Oval", \
matches["venue"] == "OUTsurance Oval",\
matches["venue"] == "Brabourne Stadium",\
matches["venue"] == "Sardar Patel Stadium",\
matches["venue"] == "Barabati Stadium", \
matches["venue"] == "Vidarbha Cricket Association Stadium,
→Jamtha",\
matches["venue"] == "Himachal Pradesh Cricket Association,
→Stadium",\
matches["venue"] == "Nehru Stadium",\
matches["venue"] == "Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket,
→Stadium",\
matches["venue"] == "Subrata Roy Sahara Stadium",\
matches["venue"] == "Shaheed Veer Narayan Singh International,
→Stadium",\
matches["venue"] == "JSCA International Stadium Complex",\
matches["venue"] == "Sheikh Zayed Stadium",\
matches["venue"] == "Sharjah Cricket Stadium",\
matches["venue"] == "Dubai International Cricket Stadium",\
matches["venue"] == "M. A. Chidambaram Stadium",
matches["venue"] == "Feroz Shah Kotla Ground",\
matches["venue"] == "M. Chinnaswamy Stadium",\
matches["venue"] == "Rajiv Gandhi Intl. Cricket Stadium" ,\
matches["venue"] == "IS Bindra Stadium",
matches["venue"] == "ACA-VDCA Stadium"]

```

```

values =_
→['Hyderabad','Mumbai','Rajkot',"Indore","Bengaluru","Mumbai","Kolkata","Delhi","Mohali","Kanp
→Town","Port Elizabeth","Durban","Centurion",'Eastern_
→Cape','Johannesburg','Northern_
→Cape','Bloemfontein','Mumbai','Ahmedabad','Cuttack','Jamtha','Dharamshala','Chennai','Visakha
→Dhabi','Sharjah','Dubai','Chennai','Delhi','Bengaluru','Hyderabad','Mohali','Visakhapatnam']

matches['city'] = np.where(matches['city'].isnull(),\
                           np.select(conditions, values),\
                           matches['city'])

null_columns=matches.isnull().sum()

print(null_columns[null_columns > 0])

```

```

winner          3
player_of_match 3
umpire1          1
umpire2          1
umpire3         636
dtype: int64

```

```

[40]: #Removing records having null values in "winner" column
matches=matches[matches["winner"].notna()]
matches

```

```

[40]:
   id  season   city   date                team1 \
0    1   2017  Hyderabad  05-04-2017      Sunrisers Hyderabad
1    2   2017    Pune  06-04-2017      Mumbai Indians
2    3   2017   Rajkot  07-04-2017      Gujarat Lions
3    4   2017   Indore  08-04-2017  Rising Pune Supergiant
4    5   2017  Bangalore  08-04-2017  Royal Challengers Bangalore
..  ...   ...   ...   ...   ...
631 632   2016   Raipur  22-05-2016      Delhi Daredevils
632 633   2016  Bangalore  24-05-2016      Gujarat Lions
633 634   2016    Delhi  25-05-2016      Sunrisers Hyderabad
634 635   2016    Delhi  27-05-2016      Gujarat Lions
635 636   2016  Bangalore  29-05-2016      Sunrisers Hyderabad

   team2                toss_winner toss_decision \
0  Royal Challengers Bangalore  Royal Challengers Bangalore      field
1    Rising Pune Supergiant    Rising Pune Supergiant      field
2    Kolkata Knight Riders    Kolkata Knight Riders      field
3    Kings XI Punjab        Kings XI Punjab      field
4    Delhi Daredevils  Royal Challengers Bangalore      bat
..  ...   ...   ...   ...
631  Royal Challengers Bangalore  Royal Challengers Bangalore      field

```

632	Royal Challengers Bangalore	Royal Challengers Bangalore	field
633	Kolkata Knight Riders	Kolkata Knight Riders	field
634	Sunrisers Hyderabad	Sunrisers Hyderabad	field
635	Royal Challengers Bangalore	Sunrisers Hyderabad	bat

	result	dl_applied	winner	win_by_runs	\
0	normal	0	Sunrisers Hyderabad	35	
1	normal	0	Rising Pune Supergiant	0	
2	normal	0	Kolkata Knight Riders	0	
3	normal	0	Kings XI Punjab	0	
4	normal	0	Royal Challengers Bangalore	15	
..	
631	normal	0	Royal Challengers Bangalore	0	
632	normal	0	Royal Challengers Bangalore	0	
633	normal	0	Sunrisers Hyderabad	22	
634	normal	0	Sunrisers Hyderabad	0	
635	normal	0	Sunrisers Hyderabad	8	

	win_by_wickets	player_of_match	\
0	0	Yuvraj Singh	
1	7	SPD Smith	
2	10	CA Lynn	
3	6	GJ Maxwell	
4	0	KM Jadhav	
..	
631	6	V Kohli	
632	4	AB de Villiers	
633	0	MC Henriques	
634	4	DA Warner	
635	0	BCJ Cutting	

	venue	umpire1	\
0	Rajiv Gandhi International Stadium, Uppal	AY Dandekar	
1	Maharashtra Cricket Association Stadium	A Nand Kishore	
2	Saurashtra Cricket Association Stadium	Nitin Menon	
3	Holkar Cricket Stadium	AK Chaudhary	
4	M Chinnaswamy Stadium	NaN	
..	
631	Shaheed Veer Narayan Singh International Stadium	A Nand Kishore	
632	M Chinnaswamy Stadium	AK Chaudhary	
633	Feroz Shah Kotla	M Erasmus	
634	Feroz Shah Kotla	M Erasmus	
635	M Chinnaswamy Stadium	HDPK Dharmasena	

	umpire2	umpire3
0	NJ Llong	NaN
1	S Ravi	NaN

2	CK Nandan	NaN
3	C Shamshuddin	NaN
4	NaN	NaN
..
631	BNJ Oxenford	NaN
632	HDPK Dharmasena	NaN
633	C Shamshuddin	NaN
634	CK Nandan	NaN
635	BNJ Oxenford	NaN

[633 rows x 18 columns]

```
[41]: null_columns=matches.isnull().sum()
      print(null_columns[null_columns > 0])
```

```
umpire1      1
umpire2      1
umpire3    633
dtype: int64
```

While playing around with the data, I found an interesting redundancy. Team Rising Pune SuperGiants were duplicated in columns team_1, team_2, winner, and toss_winner. Replacing these values with one value is the obvious thing to do next.

```
[42]: #Replacing the Rising Pune Supergiant with Rising Pune Supergiants
matches["team2"]=matches["team2"].replace("Rising Pune Supergiant","Rising Pune_
      ↳Supergiants")
matches["team1"]=matches["team1"].replace("Rising Pune Supergiant","Rising Pune_
      ↳Supergiants")
matches["winner"]=matches["winner"].replace("Rising Pune Supergiant","Rising_
      ↳Pune Supergiants")
matches["toss_winner"]=matches["toss_winner"].replace("Rising Pune_
      ↳Supergiant","Rising Pune Supergiants")
```

```
[46]: #Toss affecting the win dataframe
toss_win_result = matches.groupby(['toss_winner']).winner.value_counts().
      ↳reset_index(name="count")
#print(toss_win_result)
toss_win_result['result']=np.where(toss_win_result.winner==toss_win_result.
      ↳toss_winner, 'won', 'lost')
#numpy.where(condition, x, y)
#print(toss_win_result)

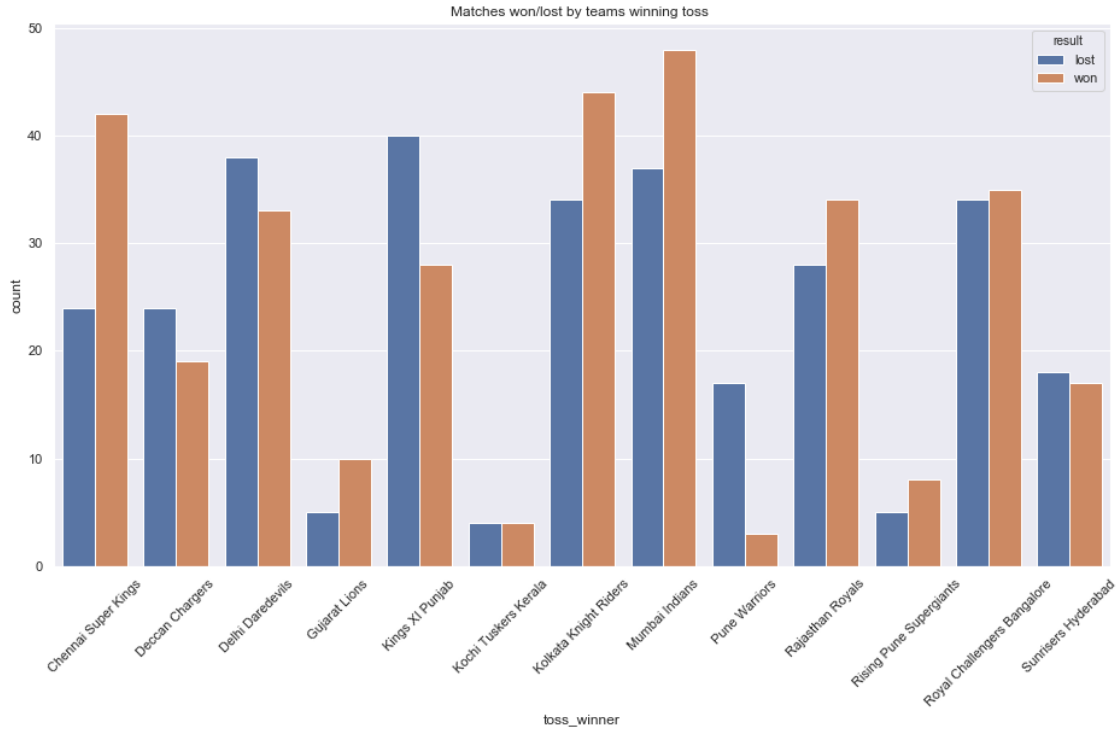
toss_win_result_df = toss_win_result.groupby(['toss_winner', 'result'])['count'].
      ↳sum().reset_index()
print(toss_win_result_df)
#Visualization
```

```

plot = sns.barplot(x="toss_winner", y="count", hue="result",
→data=toss_win_result_df)
plot.set_title('Matches won/lost by teams winning toss ')
plot.set_xticklabels(toss_win_result_df['toss_winner'].unique(),rotation=45)
plt.show()

```

	toss_winner	result	count
0	Chennai Super Kings	lost	24
1	Chennai Super Kings	won	42
2	Deccan Chargers	lost	24
3	Deccan Chargers	won	19
4	Delhi Daredevils	lost	38
5	Delhi Daredevils	won	33
6	Gujarat Lions	lost	5
7	Gujarat Lions	won	10
8	Kings XI Punjab	lost	40
9	Kings XI Punjab	won	28
10	Kochi Tuskers Kerala	lost	4
11	Kochi Tuskers Kerala	won	4
12	Kolkata Knight Riders	lost	34
13	Kolkata Knight Riders	won	44
14	Mumbai Indians	lost	37
15	Mumbai Indians	won	48
16	Pune Warriors	lost	17
17	Pune Warriors	won	3
18	Rajasthan Royals	lost	28
19	Rajasthan Royals	won	34
20	Rising Pune Supergiants	lost	5
21	Rising Pune Supergiants	won	8
22	Royal Challengers Bangalore	lost	34
23	Royal Challengers Bangalore	won	35
24	Sunrisers Hyderabad	lost	18
25	Sunrisers Hyderabad	won	17



```
[44]: #Winning stats of teams bat/field first by venues
venue_toss_decision_result=matches[["toss_winner","toss_decision","winner","venue"]]
#print(venue_toss_decision_result)

venue_toss_decision_result["decision"]=np.where((venue_toss_decision_result.
    ↳toss_winner == venue_toss_decision_result.winner) &_
    ↳(venue_toss_decision_result.toss_decision=="field"),"field_won","bat_won")
#print(venue_toss_decision_result)

venue_result=venue_toss_decision_result.groupby(["venue"]).decision.
    ↳value_counts().reset_index(name="count")
print(venue_result)
#Visualization
sns.set(rc={'figure.figsize':(15.7,8.27)})
plot = sns.barplot(x="venue", y="count", hue="decision", data=venue_result)
plot.set_title('Teams bat/field first results on venues')
plot.set_xticklabels(venue_result['venue'].unique(),rotation=90)
#plt.tight_layout()
plt.show()
```

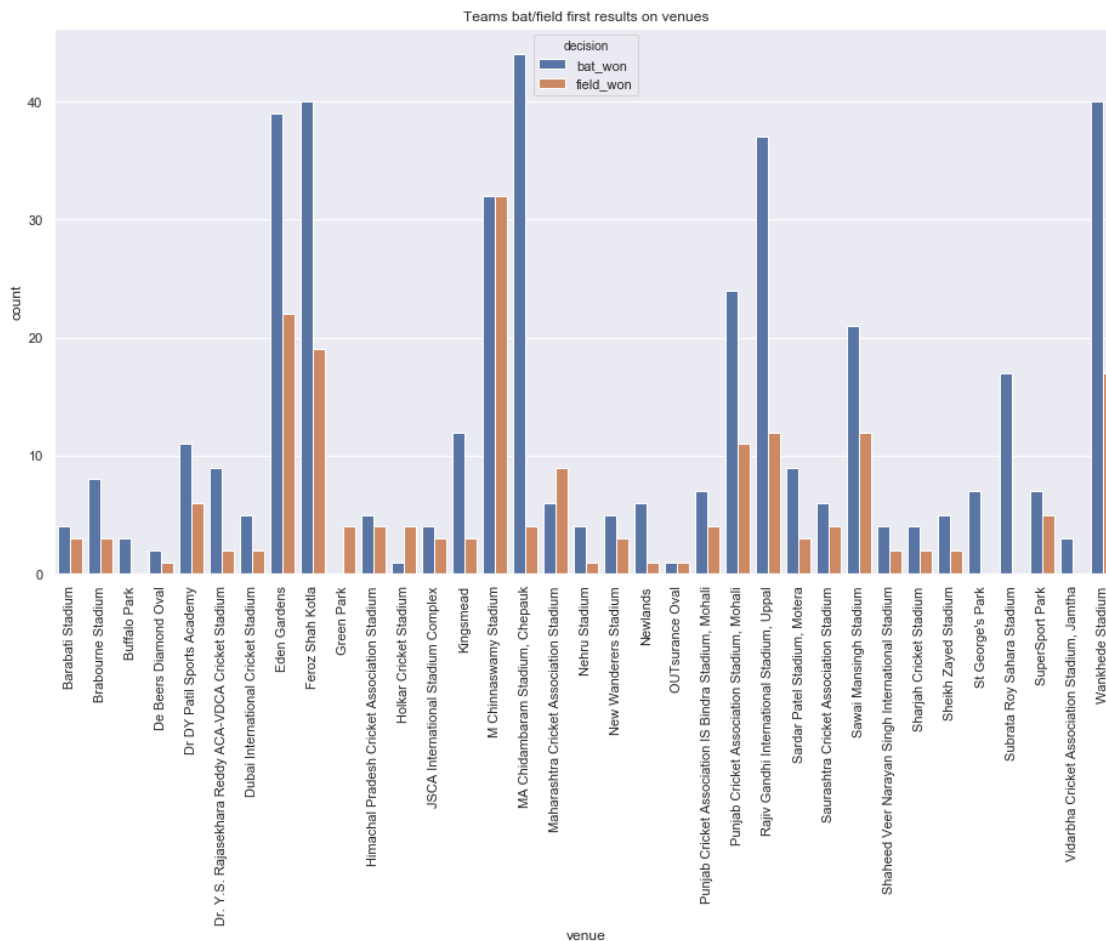
C:\Users\anika\anaconda3\lib\site-packages\ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

	venue	decision	count
0	Barabati Stadium	bat_won	4
1	Barabati Stadium	field_won	3
2	Brabourne Stadium	bat_won	8
3	Brabourne Stadium	field_won	3
4	Buffalo Park	bat_won	3
..
60	SuperSport Park	bat_won	7
61	SuperSport Park	field_won	5
62	Vidarbha Cricket Association Stadium, Jamtha	bat_won	3
63	Wankhede Stadium	bat_won	40
64	Wankhede Stadium	field_won	17

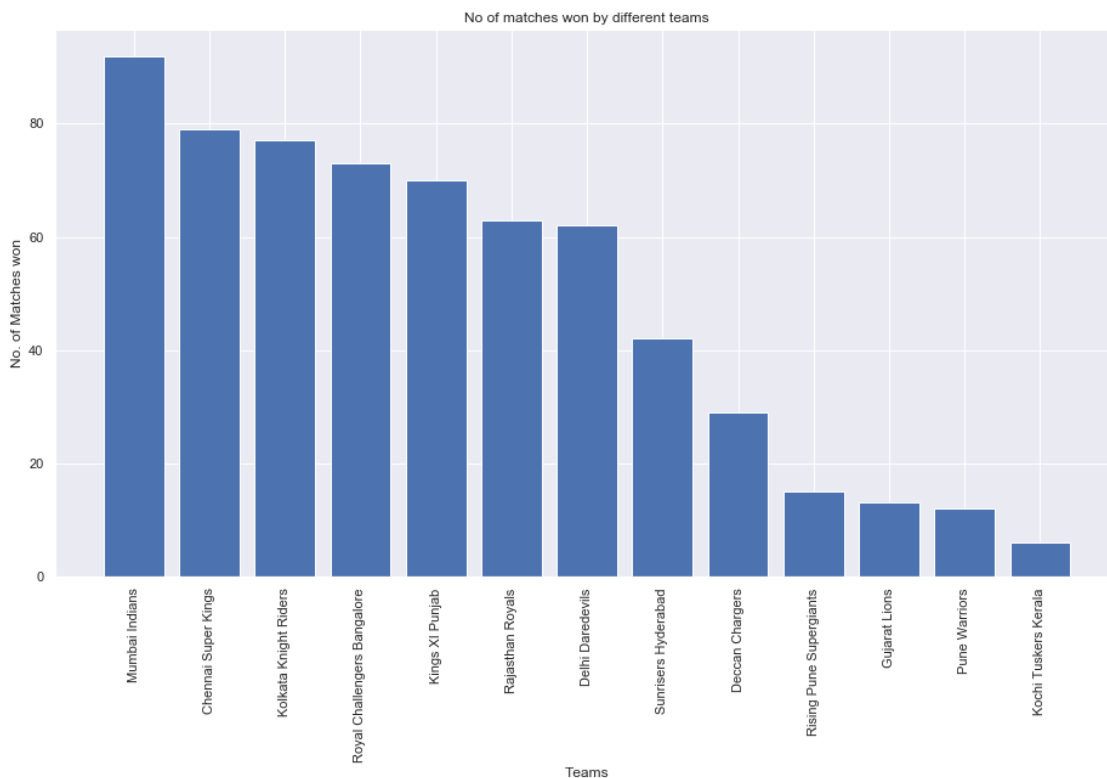
[65 rows x 3 columns]



```
[45]: teams = list(set(matches.loc[:, 'team1']))
matches_won = [len(matches.loc[matches['winner'] == i]) for i in teams]
test_list=(sorted(list(zip(matches_won, teams))))[::-1]
matches_won=[ i for i, _ in test_list ]
teams=[ i for _, i in test_list ]

plt.bar(np.arange(len(teams)), matches_won)
plt.xticks(np.arange(len(teams)), teams, rotation='vertical')
plt.ylabel('No. of Matches won')
plt.xlabel('Teams')
plt.title('No of matches won by different teams')
```

```
[45]: Text(0.5, 1.0, 'No of matches won by different teams')
```



28 Data Preprocessing & Prediction

```
[10]: from sklearn.preprocessing import LabelEncoder
      #Encode target labels with value between 0 and n_classes-1.
      #This transformer should be used to encode target values, i.e. y, and not the
      →input X.
```

For the columns to be able to assist the model in the prediction, the values should make some sense to the computers. Since they (still) don't have the ability to understand and draw inference from the text, we need to encode the strings to numeric categorical values.

```
[11]: #encoder
      encoder= LabelEncoder()
      matches["team1"]=encoder.fit_transform(matches["team1"])
      matches["team2"]=encoder.fit_transform(matches["team2"])
      matches["winner"]=encoder.fit_transform(matches["winner"].astype(str))
      →#astype(str) ??
      matches["toss_winner"]=encoder.fit_transform(matches["toss_winner"])
      matches["venue"]=encoder.fit_transform(matches["venue"])
```

Before we hop on to building models, an important observation has to be acknowledged. Columns like toss_winner, toss_decision, and winner might make sense to us, but what about the machines?

Let me elaborate. The values in toss_winner and winner include team names, but what is the relation of these variables with team_1 or team_2? The only thing common between them is that they would share the same value, but that is not enough to be logical. Also, toss_decision might be bat or field, but what team are they referring to? To tackle this problem, we will add new columns team1_win, team1_toss_win, and team1_bat for columns winner, toss_winner, and toss_decision such that they reflect the relationship with column team_1.

```
[12]: #outcome variable as a probability of team1 winning
      matches.loc[matches["winner"]==matches["team1"], "team1_win"]=1
      matches.loc[matches["winner"]!=matches["team1"], "team1_win"]=0

      matches.loc[matches["toss_winner"]==matches["team1"], "team1_toss_win"]=1
      matches.loc[matches["toss_winner"]!=matches["team1"], "team1_toss_win"]=0
```

```
[13]: matches["team1_bat"]=0
      matches.loc[(matches["team1_toss_win"]==1) &
      →(matches["toss_decision"]=="bat"), "team1_bat"]=1
      matches.head()
```

```
[13]:   id  season   city      date  team1  team2  toss_winner  toss_decision  \
0   1   2017  Hyderabad  05-04-2017    12    11           11           field
1   2   2017    Pune    06-04-2017     7    10           10           field
2   3   2017  Rajkot    07-04-2017     3     6            6           field
3   4   2017  Indore    08-04-2017    10     4            4           field
4   5   2017  Bangalore  08-04-2017    11     2           11            bat
```

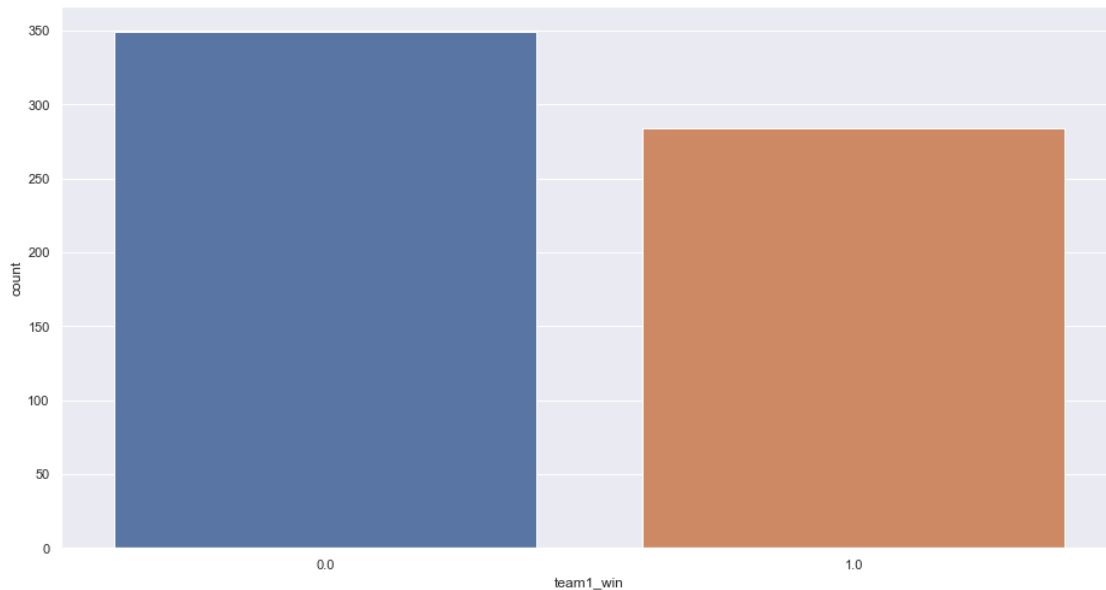
	result	dl_applied	...	win_by_runs	win_by_wickets	player_of_match	\
0	normal	0	...	35	0	Yuvraj Singh	
1	normal	0	...	0	7	SPD Smith	
2	normal	0	...	0	10	CA Lynn	
3	normal	0	...	0	6	GJ Maxwell	
4	normal	0	...	15	0	KM Jadhav	

	venue	umpire1	umpire2	umpire3	team1_win	team1_toss_win	\
0	23	AY Dandekar	NJ Llong	NaN	1.0	0.0	
1	16	A Nand Kishore	S Ravi	NaN	0.0	0.0	
2	25	Nitin Menon	CK Nandan	NaN	0.0	0.0	
3	11	AK Chaudhary	C Shamshuddin	NaN	0.0	0.0	
4	14	NaN	NaN	NaN	1.0	1.0	

	team1_bat
0	0
1	0
2	0
3	0
4	1

[5 rows x 21 columns]

```
[14]: #Checking for the distribution of the dataset
sns.countplot(x="team1_win",data=matches)
plt.show()
# sns.countplot(x="team1_toss_win",data=matches)
# plt.show()
# sns.countplot(x="team1_bat",data=matches)
# plt.show()
```



```
[15]: prediction_df=matches[["team1","team2","team1_toss_win","team1_bat","team1_win","venue"]]
```

```
[16]: prediction_df
```

```
[16]:
```

	team1	team2	team1_toss_win	team1_bat	team1_win	venue
0	12	11	0.0	0	1.0	23
1	7	10	0.0	0	0.0	16
2	3	6	0.0	0	0.0	25
3	10	4	0.0	0	0.0	11
4	11	2	1.0	1	1.0	14
...
631	2	11	0.0	0	0.0	27
632	3	11	0.0	0	0.0	14
633	12	6	0.0	0	1.0	8
634	3	12	0.0	0	0.0	8
635	12	11	1.0	1	1.0	14

[633 rows x 6 columns]

```
[17]: #First, we will check the columns if any of them represent the same values as
      ↳ other columns. For this, we need to create a correlation matrix to find out
      ↳ the relationships between the column. If the absolute value of the correlation
      ↳ between the columns is high enough, we can say that they represent similar
      ↳ values.
```

```
#dropping highly correlated features
```

```
correlated_features = set()
correlation_matrix = prediction_df.drop('team1_win', axis=1).corr()
correlation_matrix
```

*#Here, we see that team1_bat represents the same information as team1_toss_win.
 ↳ Strange, right? It's just how the dataset was built, if team1 wins the toss
 ↳ then they will always bat and if team2 wins the toss then they will always
 ↳ field. So we removed the column team1_bat from our list of features.*

```
[17]:
```

	team1	team2	team1_toss_win	team1_bat	venue
team1	1.000000	-0.098426	-0.087085	-0.087085	0.104659
team2	-0.098426	1.000000	-0.035555	-0.035555	0.077880
team1_toss_win	-0.087085	-0.035555	1.000000	1.000000	0.043885
team1_bat	-0.087085	-0.035555	1.000000	1.000000	0.043885
venue	0.104659	0.077880	0.043885	0.043885	1.000000

```
[18]: for i in range(len(correlation_matrix.columns)):
      for j in range(i):
          if abs(correlation_matrix.iloc[i, j]) > 0.8:
              colname = correlation_matrix.columns[i]
              correlated_features.add(colname)
```

```
[19]: prediction_df.drop(columns=correlated_features)
```

```
[19]:
```

	team1	team2	team1_toss_win	team1_win	venue
0	12	11	0.0	1.0	23
1	7	10	0.0	0.0	16
2	3	6	0.0	0.0	25
3	10	4	0.0	0.0	11
4	11	2	1.0	1.0	14
...
631	2	11	0.0	0.0	27
632	3	11	0.0	0.0	14
633	12	6	0.0	1.0	8
634	3	12	0.0	0.0	8
635	12	11	1.0	1.0	14

[633 rows x 5 columns]

```
[20]: #feature selection
X = prediction_df.drop('team1_win', axis=1)
target = prediction_df['team1_win']
target=target.astype(int)
```

```
[21]: # from sklearn.linear_model import LogisticRegression
```

```

# from sklearn.feature_selection import RFE          #---> Feature ranking with
→recursive feature elimination.

# #Given an external estimator that assigns weights to features (e.g., the
→coefficients of a linear model), the goal of recursive feature elimination
→(RFE) is to select features by recursively considering smaller and smaller
→sets of features. First, the estimator is trained on the initial set of
→features and the importance of each feature is obtained either through any
→specific attribute or callable. Then, the least important features are pruned
→from current set of features. That procedure is recursively repeated on the
→pruned set until the desired number of features to select is eventually
→reached.

# logreg = LogisticRegression(solver='lbfgs')
# # print(logreg)
# rfe = RFE(logReg, 20)
# rfe = rfe.fit(X, target.values.ravel())
# #Checking for the features of they are important
# print(rfe.support_)

```

```

[22]: from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
logReg=LogisticRegression(solver='lbfgs')
print(logReg)
rfe = RFE(estimator=logReg, n_features_to_select=20)
print(rfe)
rfe = rfe.fit(X, target.values.ravel())
#Checking for the features of they are important
print(rfe.support_)
print(rfe.n_features_)
print(rfe.ranking_)
print(rfe.estimator_)

```

```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
RFE(estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                fit_intercept=True, intercept_scaling=1,
                                l1_ratio=None, max_iter=100,
                                multi_class='auto', n_jobs=None, penalty='l2',
                                random_state=None, solver='lbfgs', tol=0.0001,
                                verbose=0, warm_start=False),
    n_features_to_select=20, step=1, verbose=0)
[ True  True  True  True  True]

```

5


```
[1 1 1 1 1]
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False)
```

```
[24]: #Splitting the data into training and testing data and scaling it
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.2,
    →random_state=0)

#Standardize features by removing the mean and scaling to unit variance
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train) #          Fit to data, then transform it.
    →Fits transformer to X and returns a transformed version of X.
X_test = sc.transform(X_test) #          Perform standardization by centering and
    →scaling
```

```
[25]: from sklearn.metrics import confusion_matrix, classification_report
#Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print('Accuracy of logistic regression classifier on test set: {:.4f}'.
    →format(logreg.score(X_test, y_test)))
```

```
[[69  0]
 [58  0]]
```

	precision	recall	f1-score	support
0	0.54	1.00	0.70	69
1	0.00	0.00	0.00	58
accuracy			0.54	127
macro avg	0.27	0.50	0.35	127
weighted avg	0.30	0.54	0.38	127

Accuracy of logistic regression classifier on test set: 0.5433

C:\Users\anika\anaconda3\lib\site-packages\sklearn\metrics_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

```
_warn_prf(average, modifier, msg_start, len(result))
```

```
[26]: from sklearn.svm import SVC

#SVM
svm=SVC()
svm.fit(X_train,y_train)
svm.score(X_test,y_test)
y_pred = svm.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Accuracy of SVM classifier on test set: {:.4f}'.format(svm.score(X_test,
→y_test)))
```

```
[[62  7]
 [46 12]]
```

	precision	recall	f1-score	support
0	0.57	0.90	0.70	69
1	0.63	0.21	0.31	58
accuracy			0.58	127
macro avg	0.60	0.55	0.51	127
weighted avg	0.60	0.58	0.52	127

Accuracy of SVM classifier on test set: 0.5827

```
[27]: from sklearn.tree import DecisionTreeClassifier

#Decision Tree Classifier
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
dtree.score(X_test,y_test)
y_pred = dtree.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Accuracy of decision tree classifier on test set: {:.4f}'.format(dtree.
→score(X_test, y_test)))
```

```
[[38 31]
 [27 31]]
```

	precision	recall	f1-score	support
0	0.58	0.55	0.57	69
1	0.50	0.53	0.52	58
accuracy			0.54	127
macro avg	0.54	0.54	0.54	127
weighted avg	0.55	0.54	0.54	127

Accuracy of decision tree classifier on test set: 0.5433

```
[28]: from sklearn.ensemble import RandomForestClassifier
      #Random Forest Classifier
      randomForest= RandomForestClassifier(n_estimators=100)
      randomForest.fit(X_train,y_train)
      randomForest.score(X_test,y_test)
      y_pred = randomForest.predict(X_test)
      print("Confusion matrix\n",confusion_matrix(y_test,y_pred))
      print(classification_report(y_test,y_pred))
      print('Accuracy of random forest classifier on test set: {:.4f}'.
            →format(randomForest.score(X_test, y_test)))
```

Confusion matrix

[[38 31]

[28 30]]

	precision	recall	f1-score	support
0	0.58	0.55	0.56	69
1	0.49	0.52	0.50	58
accuracy			0.54	127
macro avg	0.53	0.53	0.53	127
weighted avg	0.54	0.54	0.54	127

Accuracy of random forest classifier on test set: 0.5354

It is evident from the results that SVM gives us a higher accuracy than other algorithms for this data distribution.