In [1]:
```python
import pandas as pd
import numpy as np
```

In [2]:
```python
courses = pd.read_csv("courses.csv")
students = pd.read_csv("students.csv")
may = pd.read_csv("reg-month1.csv")
june = pd.read_csv("reg-month2.csv")
matches = pd.read_csv("matches.csv")
deliveries = pd.read_csv("deliveries.csv")
```

In [3]:
```python
courses.head(2)
```

Out[3]:

	course_id	course_name	price
0	1	python	2499
1	2	sql	3499

In [4]:
```python
students.head(2)
```

Out[4]:

	student_id	name	partner
0	1	Kailash Harjo	23
1	2	Esha Butala	1

In [5]:
```python
may.head(2)
```

Out[5]:

	student_id	course_id
0	23	1
1	15	5

In [6]:
```python
june.head(2)
```

Out[6]:

	student_id	course_id
0	3	5
1	16	7

In [7]:
```python
matches.head(2)
```

Out[7]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_ma
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yuvraj Si
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	SPD Sr

Concat

it is a powerful function that allows you to concatenate two or more DataFrames along a particular axis (row-wise or column-wise). You can control how the data is concatenated by specifying several parameters, such as axis, join, ignore_index, and keys.

```
In [8]: regs = pd.concat([may,june],ignore_index=True) # Vertically merged
        regs
```

Out[8]:

	student_id	course_id
0	23	1
1	15	5
2	18	6
3	23	4
4	16	9
5	18	1
6	1	1
7	7	8
8	22	3
9	15	1
10	19	4
11	1	6
12	7	10
13	11	7
14	13	3
15	24	4
16	21	1
17	16	5
18	23	3
19	17	7
20	23	6
21	25	1
22	19	2
23	25	10
24	3	3
25	3	5
26	16	7
27	12	10
28	12	1
29	14	9
30	7	7
31	7	2
32	16	3
33	17	10
34	11	8
35	14	6
36	12	5
37	12	7
38	18	8
39	1	10
40	1	9
41	2	5
42	7	6
43	22	5
44	22	6
45	23	9
46	23	5
47	14	4
48	14	1
49	11	10
50	42	9
51	50	8
52	38	1

In [9]:
```
# Multi_index DataFrame

multi = pd.concat([may,june],keys=['may','june'])
multi
```

Out[9]:

		student_id	course_id
may	**0**	23	1
	1	15	5
	2	18	6
	3	23	4
	4	16	9
	5	18	1
	6	1	1
	7	7	8
	8	22	3
	9	15	1
	10	19	4
	11	1	6
	12	7	10
	13	11	7
	14	13	3
	15	24	4
	16	21	1
	17	16	5
	18	23	3
	19	17	7
	20	23	6
	21	25	1
	22	19	2
	23	25	10
	24	3	3
june	**0**	3	5
	1	16	7
	2	12	10
	3	12	1
	4	14	9
	5	7	7
	6	7	2
	7	16	3
	8	17	10
	9	11	8
	10	14	6
	11	12	5
	12	12	7
	13	18	8
	14	1	10
	15	1	9
	16	2	5
	17	7	6
	18	22	5
	19	22	6
	20	23	9
	21	23	5
	22	14	4
	23	14	1
	24	11	10
	25	42	9
	26	50	8
	27	38	1

In [10]:
```
multi.loc['may']
```

Out[10]:

	student_id	course_id
0	23	1
1	15	5
2	18	6
3	23	4
4	16	9
5	18	1
6	1	1
7	7	8
8	22	3
9	15	1
10	19	4
11	1	6
12	7	10
13	11	7
14	13	3
15	24	4
16	21	1
17	16	5
18	23	3
19	17	7
20	23	6
21	25	1
22	19	2
23	25	10
24	3	3

In [11]:
```
multi.loc[('june',0)]
```

Out[11]:
```
student_id    3
course_id     5
Name: (june, 0), dtype: int64
```

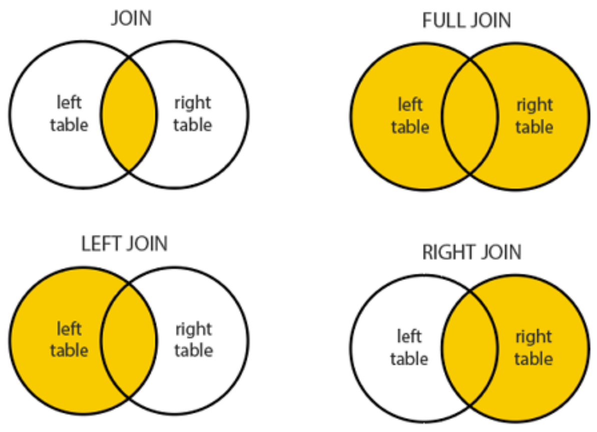
In [12]:
```
# Horizontally placed

pd.concat([may,june],axis=1)
```

Out[12]:

	student_id	course_id	student_id	course_id
0	23.0	1.0	3	5
1	15.0	5.0	16	7
2	18.0	6.0	12	10
3	23.0	4.0	12	1
4	16.0	9.0	14	9
5	18.0	1.0	7	7
6	1.0	1.0	7	2
7	7.0	8.0	16	3
8	22.0	3.0	17	10
9	15.0	1.0	11	8
10	19.0	4.0	14	6
11	1.0	6.0	12	5
12	7.0	10.0	12	7
13	11.0	7.0	18	8
14	13.0	3.0	1	10
15	24.0	4.0	1	9
16	21.0	1.0	2	5
17	16.0	5.0	7	6
18	23.0	3.0	22	5
19	17.0	7.0	22	6
20	23.0	6.0	23	9
21	25.0	1.0	23	5
22	19.0	2.0	14	4
23	25.0	10.0	14	1
24	3.0	3.0	11	10
25	NaN	NaN	42	9
26	NaN	NaN	50	8
27	NaN	NaN	38	1

Merge

On Joins



Inner Join

For joining any data ,

In each set of data, there should to be a "common" column. Students[student_id] and regs[student_id] are listed here. We combine based on the student_id, however the inner join only displays the data that is "Common" across the two dataframes.

In [13]:
```
students.merge(regs, how= 'inner' , on = 'student_id').tail()
```

Out[13]:

	student_id	name	partner	course_id
45	23	Chhavi Lachman	18	9
46	23	Chhavi Lachman	18	5
47	24	Radhika Suri	17	4
48	25	Shashank D'Alia	2	1
49	25	Shashank D'Alia	2	10

Left Join

Here we have same column --- > course_id

on basis on this we can merge using left join.

Regardless of whether or not the right side data leaves, it prints all of the left side data. so , we can see left data (Numpy , c++) but we cannot see any right side data which is student_id here, courses reflect = Left and regs reflect = right

In [14]:
```
courses.merge(regs,how='left',on='course_id').tail(5)
```

Out[14]:

	course_id	course_name	price	student_id
50	10	pyspark	2499	17.0
51	10	pyspark	2499	1.0
52	10	pyspark	2499	11.0
53	11	Numpy	699	NaN
54	12	C++	1299	NaN

Right join

In [15]:
```
temp_df = pd.DataFrame({
    'student_id':[26,27,28],
    'name':['Nitish','Ankit','Rahul'],
    'partner':[28,26,17]
})

students = pd.concat([students,temp_df],ignore_index=True)
```

In [16]:
```
students.tail()
```

Out[16]:

	student_id	name	partner
23	24	Radhika Suri	17
24	25	Shashank D'Alia	2
25	26	Nitish	28
26	27	Ankit	26
27	28	Rahul	17

Regs data(50,51,52) in the current case does not contain students data, however even this, data is printed since the join was done right.

why.?
because when using a right join, all right side data is printed regardless of whether the left side data exits or not.

here right reflects = regs , Left reflects = students

In [17]:
```
students.merge(regs, how='right',on='student_id').tail(5)
```

Out[17]:

	student_id	name	partner	course_id
48	14	Pranab Natarajan	22.0	1
49	11	David Mukhopadhyay	20.0	10
50	42	NaN	NaN	9
51	50	NaN	NaN	8
52	38	NaN	NaN	1

Since there is no course_id in the student data in the current case, "Nan" data is displayed.

Why was a left join performed using the student_id? Regardless of whether or not the right side data leaves, it prints all of the left side data.

here Left reflects = students , right reflects = regs

In [18]:
```
students.merge(regs, how='left',on='student_id').tail(5)
```

Out[18]:

	student_id	name	partner	course_id
55	25	Shashank D'Alia	2	1.0
56	25	Shashank D'Alia	2	10.0
57	26	Nitish	28	NaN
58	27	Ankit	26	NaN
59	28	Rahul	17	NaN

Outer join

Initially the left join data is clearly apparent with (Nitish, Ankit, Rahul) data written,

but the right side data (course id) is blank. like which,

Right join shows Nan even though we don't have any data for (42, 50, 38), but we can see the course's id column because it's a right join.

Finally, we may view both data sets, both common and individual, regardless of whether they have ever been. As seen in the outer join

In [19]:
```
students.merge(regs ,how ='outer', on= 'student_id' ).tail(10)
```

Out[19]:

	student_id	name	partner	course_id
53	23	Chhavi Lachman	18.0	5.0
54	24	Radhika Suri	17.0	4.0
55	25	Shashank D'Alia	2.0	1.0
56	25	Shashank D'Alia	2.0	10.0
57	26	Nitish	28.0	NaN
58	27	Ankit	26.0	NaN
59	28	Rahul	17.0	NaN
60	42	NaN	NaN	9.0
61	50	NaN	NaN	8.0
62	38	NaN	NaN	1.0

In [20]:
```
# 1. find total revenue generated
regs.merge(courses, how = 'inner' , on = 'course_id')['price'].sum()
```

Out[20]:
```
154247
```

In [27]:
```
# 2. find month by month revenue
temp = pd.concat([may,june], keys=['may','june']).reset_index()
temp.merge(courses,on = 'course_id').groupby('level_0')['price'].sum()
```

Out[27]:
```
level_0
june    65072
may     89175
Name: price, dtype: int64
```

In [32]:
```
# 3. Print the registration table
# cols -> name -> course -> price

regs.merge(students, on = 'student_id').merge(courses , on='course_id')
```

Out[32]:

	student_id	course_id	name	partner	course_name	price
0	23	1	Chhavi Lachman	18	python	2499
1	15	1	Preet Sha	16	python	2499
2	18	1	Fardeen Mahabir	13	python	2499
3	1	1	Kailash Harjo	23	python	2499
4	21	1	Seema Kota	15	python	2499
5	25	1	Shashank D’Alia	2	python	2499
6	12	1	Radha Dutt	19	python	2499
7	14	1	Pranab Natarajan	22	python	2499
8	23	4	Chhavi Lachman	18	machine learning	9999
9	19	4	Qabeel Raman	12	machine learning	9999
10	24	4	Radhika Suri	17	machine learning	9999
11	14	4	Pranab Natarajan	22	machine learning	9999
12	23	3	Chhavi Lachman	18	data analysis	4999
13	16	3	Elias Dodiya	25	data analysis	4999
14	22	3	Yash Sethi	21	data analysis	4999
15	13	3	Munni Varghese	24	data analysis	4999
16	3	3	Parveen Bhalla	3	data analysis	4999
17	23	6	Chhavi Lachman	18	power bi	1899
18	18	6	Fardeen Mahabir	13	power bi	1899
19	1	6	Kailash Harjo	23	power bi	1899
20	7	6	Tarun Thaker	9	power bi	1899
21	22	6	Yash Sethi	21	power bi	1899
22	14	6	Pranab Natarajan	22	power bi	1899
23	23	9	Chhavi Lachman	18	plotly	699
24	16	9	Elias Dodiya	25	plotly	699
25	1	9	Kailash Harjo	23	plotly	699
26	14	9	Pranab Natarajan	22	plotly	699
27	23	5	Chhavi Lachman	18	tableau	2499
28	15	5	Preet Sha	16	tableau	2499
29	16	5	Elias Dodiya	25	tableau	2499
30	22	5	Yash Sethi	21	tableau	2499
31	3	5	Parveen Bhalla	3	tableau	2499
32	12	5	Radha Dutt	19	tableau	2499
33	2	5	Esha Butala	1	tableau	2499
34	18	8	Fardeen Mahabir	13	pandas	1099
35	7	8	Tarun Thaker	9	pandas	1099
36	11	8	David Mukhopadhyay	20	pandas	1099
37	16	7	Elias Dodiya	25	ms sxcel	1599
38	7	7	Tarun Thaker	9	ms sxcel	1599
39	11	7	David Mukhopadhyay	20	ms sxcel	1599
40	17	7	Yasmin Palan	7	ms sxcel	1599
41	12	7	Radha Dutt	19	ms sxcel	1599
42	1	10	Kailash Harjo	23	pyspark	2499
43	7	10	Tarun Thaker	9	pyspark	2499
44	11	10	David Mukhopadhyay	20	pyspark	2499
45	17	10	Yasmin Palan	7	pyspark	2499
46	25	10	Shashank D’Alia	2	pyspark	2499
47	12	10	Radha Dutt	19	pyspark	2499
48	7	2	Tarun Thaker	9	sql	3499
49	19	2	Qabeel Raman	12	sql	3499

In [33]:
```
regs.merge(students, on = 'student_id').merge(courses , on='course_id')[['name','course_name','price']]
```

Out[33]:

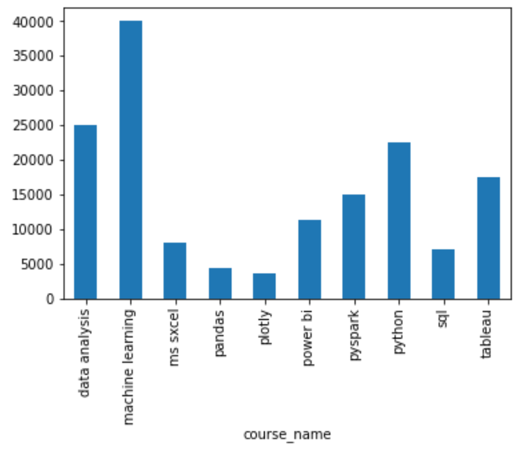
	name	course_name	price
0	Chhavi Lachman	python	2499
1	Preet Sha	python	2499
2	Fardeen Mahabir	python	2499
3	Kailash Harjo	python	2499
4	Seema Kota	python	2499
5	Shashank D’Alia	python	2499
6	Radha Dutt	python	2499
7	Pranab Natarajan	python	2499
8	Chhavi Lachman	machine learning	9999
9	Qabeel Raman	machine learning	9999
10	Radhika Suri	machine learning	9999
11	Pranab Natarajan	machine learning	9999
12	Chhavi Lachman	data analysis	4999
13	Elias Dodiya	data analysis	4999
14	Yash Sethi	data analysis	4999
15	Munni Varghese	data analysis	4999
16	Parveen Bhalla	data analysis	4999
17	Chhavi Lachman	power bi	1899
18	Fardeen Mahabir	power bi	1899
19	Kailash Harjo	power bi	1899
20	Tarun Thaker	power bi	1899
21	Yash Sethi	power bi	1899
22	Pranab Natarajan	power bi	1899
23	Chhavi Lachman	plotly	699
24	Elias Dodiya	plotly	699
25	Kailash Harjo	plotly	699
26	Pranab Natarajan	plotly	699
27	Chhavi Lachman	tableau	2499
28	Preet Sha	tableau	2499
29	Elias Dodiya	tableau	2499
30	Yash Sethi	tableau	2499
31	Parveen Bhalla	tableau	2499
32	Radha Dutt	tableau	2499
33	Esha Butala	tableau	2499
34	Fardeen Mahabir	pandas	1099
35	Tarun Thaker	pandas	1099
36	David Mukhopadhyay	pandas	1099
37	Elias Dodiya	ms sxcel	1599
38	Tarun Thaker	ms sxcel	1599
39	David Mukhopadhyay	ms sxcel	1599
40	Yasmin Palan	ms sxcel	1599
41	Radha Dutt	ms sxcel	1599
42	Kailash Harjo	pyspark	2499
43	Tarun Thaker	pyspark	2499
44	David Mukhopadhyay	pyspark	2499
45	Yasmin Palan	pyspark	2499
46	Shashank D’Alia	pyspark	2499
47	Radha Dutt	pyspark	2499
48	Tarun Thaker	sql	3499
49	Qabeel Raman	sql	3499

In [38]:
```
# 4. Plot bar chart for revenue/course
regs.merge(courses,on ='course_id').groupby('course_name')['price'].sum()
```

Out[38]:
```
course_name
data analysis       24995
machine learning    39996
ms sxcel             7995
pandas               4396
plotly               3495
power bi            11394
pyspark             14994
python              22491
sql                  6998
tableau             17493
Name: price, dtype: int64
```

In [41]:
```
regs.merge(courses,on ='course_id').groupby('course_name')['price'].sum().plot(kind='bar')
```

Out[41]:
```
<AxesSubplot:xlabel='course_name'>
```



intersect1d

Find the intersection of two arrays. Return the sorted, unique values that are in both of the input arrays.

In [45]:
```
# 5. find students who enrolled in both the months
common_students_id = np.intersect1d(may['student_id'],june['student_id'])
common_students_id
```

Out[45]:
```
array([ 1,  3,  7, 11, 16, 17, 18, 22, 23], dtype=int64)
```

In [47]:
```
students[students['student_id'].isin(common_students_id)]
```

Out[47]:

	student_id	name	partner
0	1	Kailash Harjo	23
2	3	Parveen Bhalla	3
6	7	Tarun Thaker	9
10	11	David Mukhopadhyay	20
15	16	Elias Dodiya	25
16	17	Yasmin Palan	7
17	18	Fardeen Mahabir	13
21	22	Yash Sethi	21
22	23	Chhavi Lachman	18

numpy.setdiff1d()

function find the set difference of two arrays and return the unique values in arr1 that are not in arr2.

In [52]:
```
# 6. find course that got no enrollment
# courses['course_id']
# regs['course_id']

course_id_list = np.setdiff1d(courses['course_id'], regs['course_id'])
courses[courses['course_id'].isin(course_id_list)]
```

Out[52]:

	course_id	course_name	price
10	11	Numpy	699
11	12	C++	1299

In [53]:
```
# 7. find students who did not enroll into any courses

student_id_list = np.setdiff1d(students['student_id'], regs['student_id'])
students[students['student_id'].isin(student_id_list)]
```

Out[53]:

	student_id	name	partner
3	4	Marlo Dugal	14
4	5	Kusum Bahri	6
5	6	Lakshmi Contractor	10
7	8	Radheshyam Dey	5
8	9	Nitika Chatterjee	4
9	10	Aayushman Sant	8
19	20	Hanuman Hegde	11
25	26	Nitish	28
26	27	Ankit	26
27	28	Rahul	17

In [55]:
```
students[students['student_id'].isin(student_id_list)].shape[0]
```

Out[55]:
```
10
```

In [56]:
```
# Percentage of students Enrolled

(10/28)*100
```

Out[56]:
```
35.714285714285715
```

Self Join

A self join is a regular join, but the table is joined with itself.

here, left_on = partner from outside students on left , right_on =student_id from iside students on right .

In [60]:
```
# 8. Print student name -> partner name for all enrolled students
# self join
students.merge(students,how ='inner',left_on = 'partner', right_on= 'student_id')[['name_x','name_y']]
```

Out[60]:

	name_x	name_y
0	Kailash Harjo	Chhavi Lachman
1	Esha Butala	Kailash Harjo
2	Parveen Bhalla	Parveen Bhalla
3	Marlo Dugal	Pranab Natarajan
4	Kusum Bahri	Lakshmi Contractor
5	Lakshmi Contractor	Aayushman Sant
6	Tarun Thaker	Nitika Chatterjee
7	Radheshyam Dey	Kusum Bahri
8	Nitika Chatterjee	Marlo Dugal
9	Aayushman Sant	Radheshyam Dey
10	David Mukhopadhyay	Hanuman Hegde
11	Radha Dutt	Qabeel Raman
12	Munni Varghese	Radhika Suri
13	Pranab Natarajan	Yash Sethi
14	Preet Sha	Elias Dodiya
15	Elias Dodiya	Shashank D'Alia
16	Yasmin Palan	Tarun Thaker
17	Fardeen Mahabir	Munni Varghese
18	Qabeel Raman	Radha Dutt
19	Hanuman Hegde	David Mukhopadhyay
20	Seema Kota	Preet Sha
21	Yash Sethi	Seema Kota
22	Chhavi Lachman	Fardeen Mahabir
23	Radhika Suri	Yasmin Palan
24	Rahul	Yasmin Palan
25	Shashank D'Alia	Esha Butala
26	Nitish	Rahul
27	Ankit	Nitish

In [70]:
```
# 9. find top 3 students who did most number enrollments
regs.merge(students, on='student_id').groupby(['student_id','name'])['name'].count().sort_values(ascending=False).head(3)
```

Out[70]:
```
student_id  name          
23          Chhavi Lachman    6
7           Tarun Thaker      5
1           Kailash Harjo     4
Name: name, dtype: int64
```

In [81]:
```
# 10. find top 5 students who spent most amount of money on courses
regs.merge(students , on ='student_id').merge(courses, on= 'course_id').groupby(['student_id','name'])['price'].sum().sort_values
```

Out[81]:
```
student_id  name            
23          Chhavi Lachman      22594
14          Pranab Natarajan    15096
19          Qabeel Raman        13498
7           Tarun Thaker        10595
24          Radhika Suri         9999
Name: price, dtype: int64
```

In [82]:
```
# Alternate syntax for merge
# students.merge(regs)

pd.merge(students,regs , how='inner', on= 'student_id')
```

Out[82]:

	student_id	name	partner	course_id
0	1	Kailash Harjo	23	1
1	1	Kailash Harjo	23	6
2	1	Kailash Harjo	23	10
3	1	Kailash Harjo	23	9
4	2	Esha Butala	1	5
5	3	Parveen Bhalla	3	3
6	3	Parveen Bhalla	3	5
7	7	Tarun Thaker	9	8
8	7	Tarun Thaker	9	10
9	7	Tarun Thaker	9	7
10	7	Tarun Thaker	9	2
11	7	Tarun Thaker	9	6
12	11	David Mukhopadhyay	20	7
13	11	David Mukhopadhyay	20	8
14	11	David Mukhopadhyay	20	10
15	12	Radha Dutt	19	10
16	12	Radha Dutt	19	1
17	12	Radha Dutt	19	5
18	12	Radha Dutt	19	7
19	13	Munni Varghese	24	3
20	14	Pranab Natarajan	22	9
21	14	Pranab Natarajan	22	6
22	14	Pranab Natarajan	22	4
23	14	Pranab Natarajan	22	1
24	15	Preet Sha	16	5
25	15	Preet Sha	16	1
26	16	Elias Dodiya	25	9
27	16	Elias Dodiya	25	5
28	16	Elias Dodiya	25	7
29	16	Elias Dodiya	25	3
30	17	Yasmin Palan	7	7
31	17	Yasmin Palan	7	10
32	18	Fardeen Mahabir	13	6
33	18	Fardeen Mahabir	13	1
34	18	Fardeen Mahabir	13	8
35	19	Qabeel Raman	12	4
36	19	Qabeel Raman	12	2
37	21	Seema Kota	15	1
38	22	Yash Sethi	21	3
39	22	Yash Sethi	21	5
40	22	Yash Sethi	21	6
41	23	Chhavi Lachman	18	1
42	23	Chhavi Lachman	18	4
43	23	Chhavi Lachman	18	3
44	23	Chhavi Lachman	18	6
45	23	Chhavi Lachman	18	9
46	23	Chhavi Lachman	18	5
47	24	Radhika Suri	17	4
48	25	Shashank D’Alia	2	1
49	25	Shashank D’Alia	2	10

In [87]:
```
# IPL Problems

# find top 3 stadiums with highest sixes/match ratio

matches
```

Out[87]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yu
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	S
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6	G
4	5	2017	Bangalore	2017-04-08	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
631	632	2016	Raipur	2016-05-22	Delhi Daredevils	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	6	
632	633	2016	Bangalore	2016-05-24	Gujarat Lions	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	4	AB
633	634	2016	Delhi	2016-05-25	Sunrisers Hyderabad	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Sunrisers Hyderabad	22	0	MC I
634	635	2016	Delhi	2016-05-27	Gujarat Lions	Sunrisers Hyderabad	Sunrisers Hyderabad	field	normal	0	Sunrisers Hyderabad	0	4	D
635	636	2016	Bangalore	2016-05-29	Sunrisers Hyderabad	Royal Challengers Bangalore	Sunrisers Hyderabad	bat	normal	0	Sunrisers Hyderabad	8	0	BC

636 rows × 18 columns

In [89]:
```
deliveries
```
Out[89]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	bye_runs	legbye_runs	noball_runs	penalt
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
179073	11415	2	Chennai Super Kings	Mumbai Indians	20	2	RA Jadeja	SR Watson	SL Malinga	0	...	0	0	0	
179074	11415	2	Chennai Super Kings	Mumbai Indians	20	3	SR Watson	RA Jadeja	SL Malinga	0	...	0	0	0	
179075	11415	2	Chennai Super Kings	Mumbai Indians	20	4	SR Watson	RA Jadeja	SL Malinga	0	...	0	0	0	
179076	11415	2	Chennai Super Kings	Mumbai Indians	20	5	SN Thakur	RA Jadeja	SL Malinga	0	...	0	0	0	
179077	11415	2	Chennai Super Kings	Mumbai Indians	20	6	SN Thakur	RA Jadeja	SL Malinga	0	...	0	0	0	

179078 rows × 21 columns

In [94]:
```
temp = pd.merge(deliveries,matches ,how ='inner',left_on='match_id',right_on='id')

temp.head(2)
```
Out[94]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs	win_
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	

2 rows × 39 columns

In [101]:
```
six_df=temp[temp['batsman_runs']==6]
six_df.head(2)
```
Out[101]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs
10	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	2	4	DA Warner	S Dhawan	A Choudhary	0	...	normal	0	Sunrisers Hyderabad	35
47	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	8	4	MC Henriques	S Dhawan	TM Head	0	...	normal	0	Sunrisers Hyderabad	35

2 rows × 39 columns

In [105]:
```
#stadium --> sixes
number_six = six_df.groupby('venue')['venue'].count()
number_six.head()
```

Out[105]:
```
venue
Barabati Stadium              68
Brabourne Stadium            114
Buffalo Park                  27
De Beers Diamond Oval         34
Dr DY Patil Sports Academy   173
Name: venue, dtype: int64
```

In [108]:
```
# Number of matches
number_matches = matches['venue'].value_counts()
number_matches.head()
```

Out[108]:
```
M Chinnaswamy Stadium                          66
Eden Gardens                                   61
Feroz Shah Kotla                               60
Wankhede Stadium                               57
Rajiv Gandhi International Stadium, Uppal     49
Name: venue, dtype: int64
```

In [112]:
```
(number_six/number_matches).sort_values(ascending=False).head()
```

Out[112]:
```
Holkar Cricket Stadium                              17.600000
M Chinnaswamy Stadium                               13.227273
Sharjah Cricket Stadium                             12.666667
Himachal Pradesh Cricket Association Stadium        12.000000
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium    11.727273
Name: venue, dtype: float64
```

In [113]:
```
# find orange cap holder of all the seasons
```

Out[113]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
150455	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	2	Sachin Baby	CJ Jordan	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150456	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	3	Sachin Baby	CJ Jordan	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150457	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	4	Iqbal Abdulla	Sachin Baby	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150458	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	5	Sachin Baby	Iqbal Abdulla	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150459	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	6	Iqbal Abdulla	Sachin Baby	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8

150460 rows × 39 columns

In [114]:
```
df = pd.merge(deliveries,matches ,how ='inner',left_on='match_id',right_on='id')

df.head(2)
```

Out[114]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs	win_
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	

2 rows × 39 columns

In [117]:
```
df.groupby(['season','batsman'])['batsman_runs'].sum()
```

Out[117]:
```
season  batsman          
2008    A Chopra              42
        A Kumble              13
        A Mishra              37
        A Mukund               0
        A Nehra                3
                            ... 
2017    Washington Sundar      9
        YK Pathan            143
        YS Chahal             13
        Yuvraj Singh         252
        Z Khan                 4
Name: batsman_runs, Length: 1531, dtype: int64
```

In [120]:
```
df.groupby(['season','batsman'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False)
```

Out[120]:

	season	batsman	batsman_runs
1383	2016	V Kohli	973
1278	2016	DA Warner	848
910	2013	MEK Hussey	733
684	2012	CH Gayle	733
852	2013	CH Gayle	720
...	...	...	...
1467	2017	MM Patel	0
658	2012	AC Blizzard	0
475	2011	AB Dinda	0
1394	2017	AD Nath	0
58	2008	L Balaji	0

1531 rows × 3 columns

In [123]:
```
'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False).drop_duplicates(subset='season',keep='first')
```

Out[123]:

	season	batsman	batsman_runs
1383	2016	V Kohli	973
910	2013	MEK Hussey	733
684	2012	CH Gayle	733
1088	2014	RV Uthappa	660
1422	2017	DA Warner	641
446	2010	SR Tendulkar	618
115	2008	SE Marsh	616
502	2011	CH Gayle	608
229	2009	ML Hayden	572
1148	2015	DA Warner	562

```
In [124]: oupby(['season','batsman'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False).sort_values('season')
```

Out[124]:

```
      season        batsman  batsman_runs
  58    2008       L Balaji             0
  45    2008      I Sharma             11
  12    2008       AM Nayar           206
  31    2008      DNT Zoysa            11
  67    2008        M Ntini            11
 ...     ...            ...           ...
1424    2017      DL Chahar            14
1515    2017  Swapnil Singh            12
1516    2017        TA Boult             5
1470    2017      MP Stoinis           17
1400    2017       AR Bawne            12

1531 rows × 3 columns
```

```
In [ ]:
```