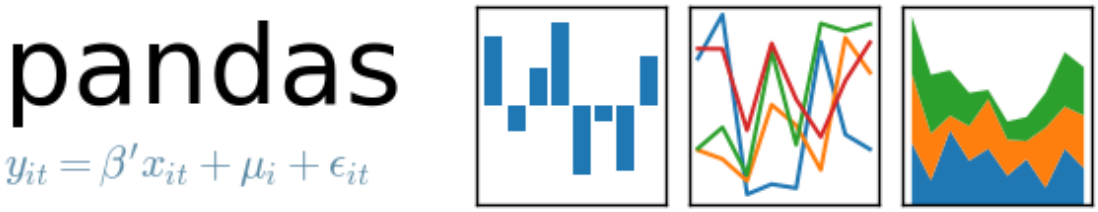
Pandas groupby

groupby in pandas is a function that lets you group data in a DataFrame based on specific criteria, and then apply aggregate functions to each group. It's a powerful tool for data analysis that allows you to quickly and easily calculate summary statistics for your data.



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

Groupby (Applies on Categorical data)

In [2]:
```
movies = pd.read_csv(" imdb-top-1000.csv")
movies.head(3)
```

Out[2]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	234311(
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	1620367
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	2303232

In [3]:
```
movies.groupby('Genre')
```

Out[3]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000024A502D5550>
```

In [4]:
```
generes =movies.groupby('Genre')
```

In [5]:
```
# Applying builtin aggregation fuctions on groupby objects
generes.sum().head(2)
```

Out[5]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	22196	1367.3	72282412	3.263226e+10	10499.0
Adventure	9656	571.5	22576163	9.496922e+09	5020.0

In [6]:
```
#Second Approach
movies.groupby('Genre')['Gross'].sum().sort_values()
```

Out[6]:
```
Genre
Thriller     1.755074e+07
Western      5.822151e+07
Film-Noir    1.259105e+08
Family       4.391106e+08
Fantasy      7.827267e+08
Horror       1.034649e+09
Mystery      1.256417e+09
Biography    8.276358e+09
Crime        8.452632e+09
Adventure    9.496922e+09
Animation    1.463147e+10
Comedy       1.566387e+10
Action       3.263226e+10
Drama        3.540997e+10
Name: Gross, dtype: float64
```

In [7]:
```
# find the genre with highest avg IMDB rating
movies.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False).hea
```

Out[7]:
```
Genre
Western    8.35
Name: IMDB_Rating, dtype: float64
```

In [8]:
```
# find director with most popularity
movies.groupby('Director')['No_of_Votes'].sum().sort_values(ascending=False).h
```

Out[8]:
```
Director
Christopher Nolan    11578345
Name: No_of_Votes, dtype: int64
```

In [9]:
```
# find the highest rated movie of each genre
movies.head(1)
```

Out[9]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	2343110

In [10]:
```
# find number of movies done by each actor
#movies['Star1'].value_counts() # first method
movies.groupby('Star1')['Series_Title'].count().sort_values(ascending=False)
```

Out[10]:
```
Star1
Tom Hanks             12
Robert De Niro        11
Clint Eastwood        10
Al Pacino             10
Leonardo DiCaprio      9
                      ..
Glen Hansard           1
Giuseppe Battiston     1
Giulietta Masina       1
Gerardo Taracena       1
Ömer Faruk Sorak       1
Name: Series_Title, Length: 660, dtype: int64
```

GroupBy Attributes and Methods

In [12]:
```
# find total number of groups -> len
len(movies.groupby('Genre')) # 14 different groups
```

Out[12]: 14

In [13]:
```
movies['Genre'].nunique() # second method
```

Out[13]: 14

In [14]:
```
# find items in each group -> size
movies.groupby('Genre').size() # index based
```

Out[14]:
```
Genre
Action       172
Adventure     72
Animation     82
Biography     88
Comedy       155
Crime        107
Drama        289
Family         2
Fantasy        2
Film-Noir      3
Horror        11
Mystery       12
Thriller       1
Western        4
dtype: int64
```

In [15]:
```
movies['Genre'].value_counts() # second method
```

Out[15]:
```
Drama        289
Action       172
Comedy       155
Crime        107
Biography     88
Animation     82
Adventure     72
Mystery       12
Horror        11
Western        4
Film-Noir      3
Fantasy        2
Family         2
Thriller       1
Name: Genre, dtype: int64
```

In [18]:
```
# first()/last() -> nth item
#movies.groupby('Genre').first()
#movies.groupby('Genre').last()
movies.groupby('Genre').nth(6) ---> #gives 7th movie
```

Out[18]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Vote
Action	Star Wars: Episode V - The Empire Strikes Back	1980	124	8.7	Irvin Kershner	Mark Hamill	115931
Adventure	North by Northwest	1959	136	8.3	Alfred Hitchcock	Cary Grant	29919
Animation	WALL·E	2008	98	8.4	Andrew Stanton	Ben Burtt	99979
Biography	Braveheart	1995	178	8.3	Mel Gibson	Mel Gibson	95918
Comedy	The Great Dictator	1940	125	8.4	Charles Chaplin	Charles Chaplin	20315
Crime	Se7en	1995	127	8.6	David Fincher	Morgan Freeman	144509
Drama	It's a Wonderful Life	1946	130	8.6	Frank Capra	James Stewart	40580
Horror	Get Out	2017	104	7.7	Jordan Peele	Daniel Kaluuya	49285
Mystery	Sleuth	1972	138	8.0	Joseph L. Mankiewicz	Laurence Olivier	4474

In [19]:
```
# get_group -> vs filtering
movies.groupby('Genre').get_group('Horror')
```

Out[19]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Vo
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins	6042
75	Alien	1979	117	Horror	8.4	Ridley Scott	Sigourney Weaver	7878
271	The Thing	1982	109	Horror	8.1	John Carpenter	Kurt Russell	3712
419	The Exorcist	1973	122	Horror	8.0	William Friedkin	Ellen Burstyn	3623
544	Night of the Living Dead	1968	96	Horror	7.9	George A. Romero	Duane Jones	1165
707	The Innocents	1961	100	Horror	7.8	Jack Clayton	Deborah Kerr	270
724	Get Out	2017	104	Horror	7.7	Jordan Peele	Daniel Kaluuya	4928
844	Halloween	1978	91	Horror	7.7	John Carpenter	Donald Pleasence	2331
876	The Invisible Man	1933	71	Horror	7.7	James Whale	Claude Rains	306
932	Saw	2004	103	Horror	7.6	James Wan	Cary Elwes	3790
948	The Others	2001	101	Horror	7.6	Alejandro Amenábar	Nicole Kidman	3376

In [20]:
```
movies.groupby('Genre').get_group('Fantasy')
```

Out[20]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	57428
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794

In [21]:
```
movies[movies['Genre']=='Fantasy'] #Second Method
```

Out[21]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	57428
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794

In [24]:
```
# groups
movies.groupby('Genre').groups ---> #Dictionary= gives index postion
```

Out[24]:
```
{'Action': [2, 5, 8, 10, 13, 14, 16, 29, 30, 31, 39, 42, 44, 55, 57, 59, 60,
63, 68, 72, 106, 109, 129, 130, 134, 140, 142, 144, 152, 155, 160, 161, 166,
168, 171, 172, 177, 181, 194, 201, 202, 216, 217, 223, 224, 236, 241, 262, 27
5, 294, 308, 320, 325, 326, 331, 337, 339, 340, 343, 345, 348, 351, 353, 356,
357, 362, 368, 369, 375, 376, 390, 410, 431, 436, 473, 477, 479, 482, 488, 49
3, 496, 502, 507, 511, 532, 535, 540, 543, 564, 569, 570, 573, 577, 582, 583,
602, 605, 608, 615, 623, ...], 'Adventure': [21, 47, 93, 110, 114, 116, 118,
137, 178, 179, 191, 193, 209, 226, 231, 247, 267, 273, 281, 300, 301, 304, 30
6, 323, 329, 361, 366, 377, 402, 406, 415, 426, 458, 470, 497, 498, 506, 513,
514, 537, 549, 552, 553, 566, 576, 604, 609, 618, 638, 647, 675, 681, 686, 69
2, 711, 713, 739, 755, 781, 797, 798, 851, 873, 884, 912, 919, 947, 957, 964,
966, 984, 991], 'Animation': [23, 43, 46, 56, 58, 61, 66, 70, 101, 135, 146,
151, 158, 170, 197, 205, 211, 213, 219, 229, 230, 242, 245, 246, 270, 330, 33
2, 358, 367, 378, 386, 389, 394, 395, 399, 401, 405, 409, 469, 499, 510, 516,
518, 522, 578, 586, 592, 595, 596, 599, 633, 640, 643, 651, 665, 672, 694, 72
8, 740, 741, 744, 756, 758, 761, 771, 783, 796, 799, 822, 828, 843, 875, 891,
892, 902, 906, 920, 956, 971, 976, 986, 992], 'Biography': [7, 15, 18, 35, 3
8, 54, 102, 107, 131, 139, 147, 157, 159, 173, 176, 212, 215, 218, 228, 235,
243, 263, 276, 282, 290, 298, 317, 328, 338, 342, 346, 359, 360, 365, 372, 37
3, 385, 411, 416, 418, 424, 429, 484, 525, 536, 542, 545, 575, 579, 587, 600,
606, 614, 622, 632, 635, 644, 649, 650, 657, 671, 673, 684, 729, 748, 753, 75
7, 759, 766, 770, 779, 809, 810, 815, 820, 831, 849, 858, 877, 882, 897, 910,
915, 923, 940, 949, 952, 987], 'Comedy': [19, 26, 51, 52, 64, 78, 83, 95, 96,
112, 117, 120, 127, 128, 132, 153, 169, 183, 192, 204, 207, 208, 214, 221, 23
3, 238, 240, 250, 251, 252, 256, 261, 266, 277, 284, 311, 313, 316, 318, 322,
327, 374, 379, 381, 392, 396, 403, 413, 414, 417, 427, 435, 445, 446, 449, 45
5, 459, 460, 463, 464, 466, 471, 472, 475, 481, 490, 494, 500, 503, 509, 526,
528, 530, 531, 533, 538, 539, 541, 547, 557, 558, 562, 563, 565, 574, 591, 59
3, 594, 598, 613, 626, 630, 660, 662, 667, 679, 680, 683, 687, 701, ...], 'Cr
ime': [1, 3, 4, 6, 22, 25, 27, 28, 33, 37, 41, 71, 77, 79, 86, 87, 103, 108,
111, 113, 123, 125, 133, 136, 162, 163, 164, 165, 180, 186, 187, 189, 198, 22
2, 232, 239, 255, 257, 287, 288, 299, 305, 335, 363, 364, 380, 384, 397, 437,
438, 441, 442, 444, 450, 451, 465, 474, 480, 485, 487, 505, 512, 519, 520, 52
3, 527, 546, 556, 560, 584, 597, 603, 607, 611, 621, 639, 653, 664, 669, 676,
695, 708, 723, 762, 763, 767, 775, 791, 795, 802, 811, 823, 827, 833, 885, 89
5, 921, 922, 926, 938, ...], 'Drama': [0, 9, 11, 17, 20, 24, 32, 34, 36, 40,
45, 50, 53, 62, 65, 67, 73, 74, 76, 80, 82, 84, 85, 88, 89, 90, 91, 92, 94, 9
7, 98, 99, 100, 104, 105, 121, 122, 124, 126, 138, 141, 143, 148, 149, 150, 1
54, 156, 167, 174, 175, 182, 184, 185, 188, 190, 195, 196, 199, 200, 203, 20
6, 210, 225, 227, 234, 237, 244, 248, 249, 253, 254, 258, 259, 260, 264, 265,
268, 269, 272, 274, 278, 279, 280, 283, 285, 286, 289, 291, 292, 293, 295, 29
6, 297, 302, 303, 307, 310, 312, 314, 315, ...], 'Family': [688, 698], 'Fanta
sy': [321, 568], 'Film-Noir': [309, 456, 712], 'Horror': [49, 75, 271, 419, 5
44, 707, 724, 844, 876, 932, 948], 'Mystery': [69, 81, 119, 145, 220, 393, 42
0, 714, 829, 899, 959, 961], 'Thriller': [700], 'Western': [12, 48, 115, 69
1]}
```

In [25]:
```
# describe
movies.groupby('Genre').describe()
```

Out[25]:

	Runtime								IMDB_Rating	.
	count	mean	std	min	25%	50%	75%	max	count	mean	.
Genre											
Action	172.0	129.046512	28.500706	45.0	110.75	127.5	143.25	321.0	172.0	7.949419	.
Adventure	72.0	134.111111	33.317320	88.0	109.00	127.0	149.00	228.0	72.0	7.937500	.
Animation	82.0	99.585366	14.530471	71.0	90.00	99.5	106.75	137.0	82.0	7.930488	.
Biography	88.0	136.022727	25.514466	93.0	120.00	129.0	146.25	209.0	88.0	7.938636	.
Comedy	155.0	112.129032	22.946213	68.0	96.00	106.0	124.50	188.0	155.0	7.901290	.
Crime	107.0	126.392523	27.689231	80.0	106.50	122.0	141.50	229.0	107.0	8.016822	.
Drama	289.0	124.737024	27.740490	64.0	105.00	121.0	137.00	242.0	289.0	7.957439	.
Family	2.0	107.500000	10.606602	100.0	103.75	107.5	111.25	115.0	2.0	7.800000	.
Fantasy	2.0	85.000000	12.727922	76.0	80.50	85.0	89.50	94.0	2.0	8.000000	.
Film-Noir	3.0	104.000000	4.000000	100.0	102.00	104.0	106.00	108.0	3.0	7.966667	.
Horror	11.0	102.090909	13.604812	71.0	98.00	103.0	109.00	122.0	11.0	7.909091	.
Mystery	12.0	119.083333	14.475423	96.0	110.75	117.5	130.25	138.0	12.0	7.975000	.
Thriller	1.0	108.000000	NaN	108.0	108.00	108.0	108.00	108.0	1.0	7.800000	.
Western	4.0	148.250000	17.153717	132.0	134.25	148.0	162.00	165.0	4.0	8.350000	.

14 rows × 40 columns

In [30]:
```
# sample
movies.groupby('Genre').sample(2,replace=True)
```

Out[30]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
308	White Heat	1949	114	Action	8.1	Raoul Walsh	James Cagney	
171	Die Hard	1988	132	Action	8.2	John McTiernan	Bruce Willis	
300	Ben-Hur	1959	212	Adventure	8.1	William Wyler	Charlton Heston	
47	Back to the Future	1985	116	Adventure	8.5	Robert Zemeckis	Michael J. Fox	
640	Les triplettes de Belleville	2003	80	Animation	7.8	Sylvain Chomet	Michèle Caucheteux	
799	South Park: Bigger, Longer & Uncut	1999	81	Animation	7.7	Trey Parker	Trey Parker	
632	The World's Fastest Indian	2005	127	Biography	7.8	Roger Donaldson	Anthony Hopkins	
372	Mar adentro	2014	126	Biography	8.0	Alejandro Amenábar	Javier Bardem	
846	Love and Death	1975	85	Comedy	7.7	Woody Allen	Woody Allen	
660	The Sandlot	1993	101	Comedy	7.8	David Mickey Evans	Tom Guiry	
397	Bound by Honor	1993	180	Crime	8.0	Taylor Hackford	Damian Chapa	
108	Scarface	1983	170	Crime	8.3	Brian De Palma	Al Pacino	
53	Capharnaüm	2018	126	Drama	8.4	Nadine Labaki	Zain Al Rafeea	
894	Creed	2015	133	Drama	7.6	Ryan Coogler	Michael B. Jordan	
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas	
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas	
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	
309	The Third Man	1949	104	Film-Noir	8.1	Carol Reed	Orson Welles	
712	Shadow of a Doubt	1943	108	Film-Noir	7.8	Alfred Hitchcock	Teresa Wright	
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins	

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
948	The Others	2001	101	Horror	7.6	Alejandro Amenábar	Nicole Kidman	
81	Rear Window	1954	112	Mystery	8.4	Alfred Hitchcock	James Stewart	
959	Dark City	1998	100	Mystery	7.6	Alex Proyas	Rufus Sewell	
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn	
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn	
691	The Outlaw Josey Wales	1976	135	Western	7.8	Clint Eastwood	Clint Eastwood	
48	Once Upon a Time in the West	1968	165	Western	8.5	Sergio Leone	Henry Fonda	

In [31]:
```
# nunique
movies.groupby('Genre').nunique() # unique --> unique items , nunique--> gives
```

Out[31]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Votes	Gr
Action	172	61	78	15	123	121	172	
Adventure	72	49	58	10	59	59	72	
Animation	82	35	41	11	51	77	82	
Biography	88	44	56	13	76	72	88	
Comedy	155	72	70	11	113	133	155	
Crime	106	56	65	14	86	85	107	
Drama	289	83	95	14	211	250	288	
Family	2	2	2	1	2	2	2	
Fantasy	2	2	2	2	2	2	2	
Film-Noir	3	3	3	3	3	3	3	
Horror	11	11	10	8	10	11	11	
Mystery	12	11	10	8	10	11	12	
Thriller	1	1	1	1	1	1	1	
Western	4	4	4	4	2	2	4	

aggregate method

In [33]:
```
# passing dict
movies.groupby('Genre').agg(
    {
        'Runtime':'mean',
        'IMDB_Rating':'mean',
        'No_of_Votes':'sum',
        'Gross':'sum',
        'Metascore':'min'

    }

)
```

Out[33]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	129.046512	7.949419	72282412	3.263226e+10	33.0
Adventure	134.111111	7.937500	22576163	9.496922e+09	41.0
Animation	99.585366	7.930488	21978630	1.463147e+10	61.0
Biography	136.022727	7.938636	24006844	8.276358e+09	48.0
Comedy	112.129032	7.901290	27620327	1.566387e+10	45.0
Crime	126.392523	8.016822	33533615	8.452632e+09	47.0
Drama	124.737024	7.957439	61367304	3.540997e+10	28.0
Family	107.500000	7.800000	551221	4.391106e+08	67.0
Fantasy	85.000000	8.000000	146222	7.827267e+08	NaN
Film-Noir	104.000000	7.966667	367215	1.259105e+08	94.0
Horror	102.090909	7.909091	3742556	1.034649e+09	46.0
Mystery	119.083333	7.975000	4203004	1.256417e+09	52.0
Thriller	108.000000	7.800000	27733	1.755074e+07	81.0
Western	148.250000	8.350000	1289665	5.822151e+07	69.0

In [37]:
```
#Passsing List

movies.groupby('Genre').agg(['min','max','mean','sum','median'])
```

Out[37]:

	Runtime					IMDB_Rating					...
	min	max	mean	sum	median	min	max	mean	sum	median	...
Genre											
Action	45	321	129.046512	22196	127.5	7.6	9.0	7.949419	1367.3	7.9	...
Adventure	88	228	134.111111	9656	127.0	7.6	8.6	7.937500	571.5	7.9	...
Animation	71	137	99.585366	8166	99.5	7.6	8.6	7.930488	650.3	7.9	...
Biography	93	209	136.022727	11970	129.0	7.6	8.9	7.938636	698.6	7.9	...
Comedy	68	188	112.129032	17380	106.0	7.6	8.6	7.901290	1224.7	7.9	...
Crime	80	229	126.392523	13524	122.0	7.6	9.2	8.016822	857.8	8.0	...
Drama	64	242	124.737024	36049	121.0	7.6	9.3	7.957439	2299.7	8.0	...
Family	100	115	107.500000	215	107.5	7.8	7.8	7.800000	15.6	7.8	...
Fantasy	76	94	85.000000	170	85.0	7.9	8.1	8.000000	16.0	8.0	...
Film-Noir	100	108	104.000000	312	104.0	7.8	8.1	7.966667	23.9	8.0	...
Horror	71	122	102.090909	1123	103.0	7.6	8.5	7.909091	87.0	7.8	...
Mystery	96	138	119.083333	1429	117.5	7.6	8.4	7.975000	95.7	8.0	...
Thriller	108	108	108.000000	108	108.0	7.8	7.8	7.800000	7.8	7.8	...
Western	132	165	148.250000	593	148.0	7.8	8.8	8.350000	33.4	8.4	...

14 rows × 25 columns

In [39]:
```
# Adding both the syntax
movies.groupby('Genre').agg(
    {
        'Runtime':['min','mean'],
        'IMDB_Rating':'mean',
        'No_of_Votes':['sum','max'],
        'Gross':'sum',
        'Metascore':'min'
    }
)
```

Out[39]:

	Runtime		IMDB_Rating	No_of_Votes		Gross	Metascore
	min	mean	mean	sum	max	sum	min
Genre							
Action	45	129.046512	7.949419	72282412	2303232	3.263226e+10	33.0
Adventure	88	134.111111	7.937500	22576163	1512360	9.496922e+09	41.0
Animation	71	99.585366	7.930488	21978630	999790	1.463147e+10	61.0
Biography	93	136.022727	7.938636	24006844	1213505	8.276358e+09	48.0
Comedy	68	112.129032	7.901290	27620327	939631	1.566387e+10	45.0
Crime	80	126.392523	8.016822	33533615	1826188	8.452632e+09	47.0
Drama	64	124.737024	7.957439	61367304	2343110	3.540997e+10	28.0
Family	100	107.500000	7.800000	551221	372490	4.391106e+08	67.0
Fantasy	76	85.000000	8.000000	146222	88794	7.827267e+08	NaN
Film-Noir	100	104.000000	7.966667	367215	158731	1.259105e+08	94.0
Horror	71	102.090909	7.909091	3742556	787806	1.034649e+09	46.0
Mystery	96	119.083333	7.975000	4203004	1129894	1.256417e+09	52.0
Thriller	108	108.000000	7.800000	27733	27733	1.755074e+07	81.0
Western	132	148.250000	8.350000	1289665	688390	5.822151e+07	69.0

In [40]:
```
# looping on groups
for group , data in movies.groupby('Genre'):
    print(data)
```

```
                                        Series_Title Released_Year  Runt
ime  \
2                                   The Dark Knight          2008
152
5       The Lord of the Rings: The Return of the King          2003
201
8                                         Inception          2010
148
10   The Lord of the Rings: The Fellowship of the Ring          2001
178
13              The Lord of the Rings: The Two Towers          2002
179
..                                               ...           ...
...
968                                  Falling Down          1993
113
979                                  Lethal Weapon          1987
109
982                                      Mad Max 2          1981
```

In [41]:
```
# find the highest rated movie of each genre

df = pd.DataFrame(columns=movies.columns)

for group , data in movies.groupby('Genre'):
    df= df.append(data[data['IMDB_Rating'] == data['IMDB_Rating'].max()])
df
```

Out[41]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale
21	Interstellar	2014	169	Adventure	8.6	Christopher Nolan	Matthew McConaughey
23	Sen to Chihiro no kamikakushi	2001	125	Animation	8.6	Hayao Miyazaki	Daveigh Chase
7	Schindler's List	1993	195	Biography	8.9	Steven Spielberg	Liam Neeson
19	Gisaengchung	2019	132	Comedy	8.6	Bong Joon Ho	Kang-ho Song
26	La vita è bella	1997	116	Comedy	8.6	Roberto Benigni	Roberto Benigni
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas
698	Willy Wonka & the Chocolate Factory	1971	100	Family	7.8	Mel Stuart	Gene Wilder
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss
309	The Third Man	1949	104	Film-Noir	8.1	Carol Reed	Orson Welles
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins
69	Memento	2000	113	Mystery	8.4	Christopher Nolan	Guy Pearce
81	Rear Window	1954	112	Mystery	8.4	Alfred Hitchcock	James Stewart
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn
12	Il buono, il brutto, il cattivo	1966	161	Western	8.8	Sergio Leone	Clint Eastwood

split (apply) combine

In [42]:
```
# apply -> builtin function
movies.groupby('Genre').apply(min)
```

Out[42]:

Genre	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1
Action	300	1924	45	Action	7.6	Abhishek Chaubey	Aamir Khan
Adventure	2001: A Space Odyssey	1925	88	Adventure	7.6	Akira Kurosawa	Aamir Khan
Animation	Akira	1940	71	Animation	7.6	Adam Elliot	Adrian Molina
Biography	12 Years a Slave	1928	93	Biography	7.6	Adam McKay	Adrien Brody
Comedy	(500) Days of Summer	1921	68	Comedy	7.6	Alejandro G. Iñárritu	Aamir Khan
Crime	12 Angry Men	1931	80	Crime	7.6	Akira Kurosawa	Ajay Devgn
Drama	1917	1925	64	Drama	7.6	Aamir Khan	Abhay Deol
Family	E.T. the Extra-Terrestrial	1971	100	Family	7.8	Mel Stuart	Gene Wilder
Fantasy	Das Cabinet des Dr. Caligari	1920	76	Fantasy	7.9	F.W. Murnau	Max Schreck
Film-Noir	Shadow of a Doubt	1941	100	Film-Noir	7.8	Alfred Hitchcock	Humphrey Bogart
Horror	Alien	1933	71	Horror	7.6	Alejandro Amenábar	Anthony Perkins
Mystery	Dark City	1938	96	Mystery	7.6	Alex Proyas	Bernard-Pierre Donnadieu
Thriller	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn
Western	Il buono, il brutto, il cattivo	1965	132	Western	7.8	Clint Eastwood	Clint Eastwood

In [43]:
```
# find number of movies starting with A for each group
def foo(group):
    print(group) # type = Dataframe
    return group
```

In [44]:
```
movies.groupby('Genre').apply(foo)
```

```
                                          Series_Title Released_Year  Runt
ime  \
2                                    The Dark Knight          2008
152
5       The Lord of the Rings: The Return of the King          2003
201
8                                         Inception          2010
148
10   The Lord of the Rings: The Fellowship of the Ring          2001
178
13              The Lord of the Rings: The Two Towers          2002
179
..                                               ...           ...
...
968                                  Falling Down          1993
113
979                                  Lethal Weapon          1987
109
```

In [49]:
```
#Custom Logic
def foo(group):
     return group['Series_Title'].str.startswith('A').sum()
```

In [50]:
```
movies.groupby('Genre').apply(foo)
```

Out[50]:
```
Genre
Action       10
Adventure     2
Animation     2
Biography     9
Comedy       14
Crime         4
Drama        21
Family        0
Fantasy       0
Film-Noir     0
Horror        1
Mystery       0
Thriller      0
Western       0
dtype: int64
```

In [51]:
```
# find ranking of each movie in the group according to IMDB score
def rank_movie(group):
    group['genre_rank']=group['IMDB_Rating'].rank(ascending=False)
    return group
```

In [52]:
```
movies.groupby('Genre').apply(rank_movie)
```

Out[52]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	23
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	162
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	230
3	The Godfather: Part II	1974	202	Crime	9.0	Francis Ford Coppola	Al Pacino	112
4	12 Angry Men	1957	96	Crime	9.0	Sidney Lumet	Henry Fonda	68
...	...	...	...	...	...	...	...	
995	Breakfast at Tiffany's	1961	115	Comedy	7.6	Blake Edwards	Audrey Hepburn	16
996	Giant	1956	201	Drama	7.6	George Stevens	Elizabeth Taylor	3
997	From Here to Eternity	1953	118	Drama	7.6	Fred Zinnemann	Burt Lancaster	4
998	Lifeboat	1944	97	Drama	7.6	Alfred Hitchcock	Tallulah Bankhead	2
999	The 39 Steps	1935	86	Crime	7.6	Alfred Hitchcock	Robert Donat	5

1000 rows × 11 columns

In [55]:
```
 # find normalized IMDB rating group wise
#x normalized = (x - x minimum) / (x maximum - x minimum)
def normal(group):
    group['normal_rating'] = (group['IMDB_Rating']- group['IMDB_Rating'].min()
    return group

movies.groupby('Genre').apply(normal)
```

Out[55]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	23
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	16
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	23
3	The Godfather: Part II	1974	202	Crime	9.0	Francis Ford Coppola	Al Pacino	11
4	12 Angry Men	1957	96	Crime	9.0	Sidney Lumet	Henry Fonda	6
...	...	...	...	...	...	...	...	
995	Breakfast at Tiffany's	1961	115	Comedy	7.6	Blake Edwards	Audrey Hepburn	1
996	Giant	1956	201	Drama	7.6	George Stevens	Elizabeth Taylor	
997	From Here to Eternity	1953	118	Drama	7.6	Fred Zinnemann	Burt Lancaster	
998	Lifeboat	1944	97	Drama	7.6	Alfred Hitchcock	Tallulah Bankhead	
999	The 39 Steps	1935	86	Crime	7.6	Alfred Hitchcock	Robert Donat	

1000 rows × 11 columns

groupby on multiple cols

In [57]:
```
combo = movies.groupby(['Director','Star1'])
#size
combo.size()
```

Out[57]:
```
Director             Star1
Aamir Khan           Amole Gupte       1
Aaron Sorkin         Eddie Redmayne    1
Abdellatif Kechiche  Léa Seydoux       1
Abhishek Chaubey     Shahid Kapoor     1
Abhishek Kapoor      Amit Sadh         1
                                      ..
Zaza Urushadze       Lembit Ulfsak     1
Zoya Akhtar          Hrithik Roshan    1
                     Vijay Varma       1
Çagan Irmak          Çetin Tekindor    1
Ömer Faruk Sorak     Cem Yilmaz        1
Length: 898, dtype: int64
```

In [59]:
```
#if we want particular combo---> get_group
combo.get_group(('Aamir Khan','Amole Gupte'))
```

Out[59]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes	
65	Taare Zameen Par	2007	165	Drama	8.4	Aamir Khan	Amole Gupte	168895	1

In [62]:
```
# find the most earning actor-> director combo
combo['Gross'].sum().sort_values(ascending=False).head(2)
```

Out[62]:
```
Director        Star1
Akira Kurosawa  Toshirô Mifune    2.999877e+09
Anthony Russo   Joe Russo         2.205039e+09
Name: Gross, dtype: float64
```

In [75]:
```
# find the best(in-terms of metascore(avg)) actor->genre combo
movies.groupby(['Star1','Genre'])['Metascore'].mean().reset_index().sort_value
```

Out[75]:

	Star1	Genre	Metascore
230	Ellar Coltrane	Drama	100.0

In [77]:
```
# agg on multiple groupby
combo.agg(['min','max','mean'])
```

Out[77]:

		Runtime			IMDB_Rating			No_of_Votes			
		min	max	mean	min	max	mean	min	max	mean	n
Director	Star1										
Aamir Khan	Amole Gupte	165	165	165.0	8.4	8.4	8.4	168895	168895	168895.0	122386
Aaron Sorkin	Eddie Redmayne	129	129	129.0	7.8	7.8	7.8	89896	89896	89896.0	85309041
Abdellatif Kechiche	Léa Seydoux	180	180	180.0	7.7	7.7	7.7	138741	138741	138741.0	219967
Abhishek Chaubey	Shahid Kapoor	148	148	148.0	7.8	7.8	7.8	27175	27175	27175.0	21842830
Abhishek Kapoor	Amit Sadh	130	130	130.0	7.7	7.7	7.7	32628	32628	32628.0	112252
...	...	...	...	...	...	...	...	...	...	...	
Zaza Urushadze	Lembit Ulfsak	87	87	87.0	8.2	8.2	8.2	40382	40382	40382.0	14450
Zoya Akhtar	Hrithik Roshan	155	155	155.0	8.1	8.1	8.1	67927	67927	67927.0	310848
	Vijay Varma	154	154	154.0	8.0	8.0	8.0	31886	31886	31886.0	556653
Çagan Irmak	Çetin Tekindor	112	112	112.0	8.3	8.3	8.3	78925	78925	78925.0	46185536
Ömer Faruk Sorak	Cem Yilmaz	127	127	127.0	8.0	8.0	8.0	56960	56960	56960.0	19620607

898 rows × 15 columns

Excercise

In [78]:
```
ipl = pd.read_csv("deliveries.csv")
ipl.head(2)
```

Out[78]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_s
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	

2 rows × 21 columns

In [80]:
```
ipl.shape # ball by ball dataset
```

Out[80]:
```
(179078, 21)
```

In [87]:
```
# find the top 10 batsman in terms of runs
ipl.groupby('batsman')['batsman_runs'].sum().sort_values(ascending=False).rese
```

Out[87]:

	batsman	batsman_runs
0	V Kohli	5434
1	SK Raina	5415
2	RG Sharma	4914
3	DA Warner	4741
4	S Dhawan	4632
5	CH Gayle	4560
6	MS Dhoni	4477
7	RV Uthappa	4446
8	AB de Villiers	4428
9	G Gambhir	4223

In [94]:
```
# find the batsman with max no of sixes
six = ipl[ipl['batsman_runs']==6]

six.groupby('batsman')['batsman'].count().sort_values(ascending=False).head(2)
```

Out[94]:
```
batsman
CH Gayle          327
AB de Villiers    214
Name: batsman, dtype: int64
```

In [105]:
```
# find batsman with most number of 4's and 6's in last 5 overs

temp = ipl[ipl['over']>15]
temp = temp[(temp['batsman_runs'] == 4) | (temp['batsman_runs'] == 6)]

temp.groupby('batsman')['batsman'].count().sort_values(ascending=False).head(1
```

Out[105]: 'MS Dhoni'

In [110]:
```
# find V Kohli's record against all teams

temp = ipl[ipl['batsman'] == 'V Kohli']

temp.groupby('bowling_team')['batsman_runs'].sum().sort_values(ascending=False
```

Out[110]:

	bowling_team	batsman_runs
0	Delhi Daredevils	763
1	Chennai Super Kings	749

In [118]:
```
# Create a function that can return the highest score of any batsman

temp = ipl[ipl['batsman'] == 'V Kohli']

temp.groupby('match_id')['batsman_runs'].sum().sort_values(ascending=False).he
```

Out[118]: 113

In [122]:
```
def highest(batsman):
    temp = ipl[ipl['batsman'] == batsman]
    return  temp.groupby('match_id')['batsman_runs'].sum().sort_values(ascendi
```

In [123]:
```
highest('DA Warner')
```

Out[123]: 126

In [ ]:
```

```

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

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

Out[3]:

	course_id	course_name	price
0	1	python	2499
1	2	sql	3499

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

Out[4]:

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

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

Out[5]:

	student_id	course_id
0	23	1
1	15	5

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

Out[6]:

	student_id	course_id
0	3	5
1	16	7

In [7]:
```
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
```

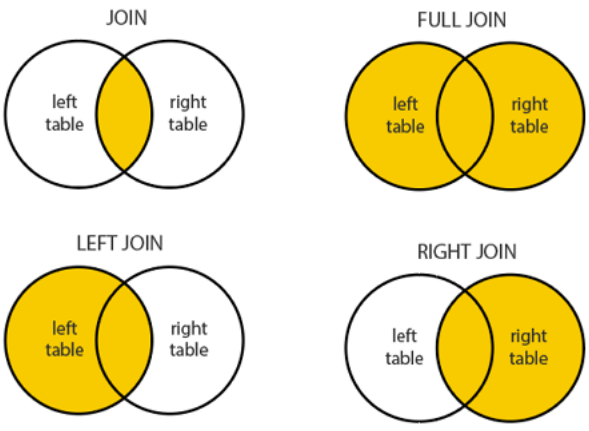
In [12]:
```
# Horizontally placed

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]:

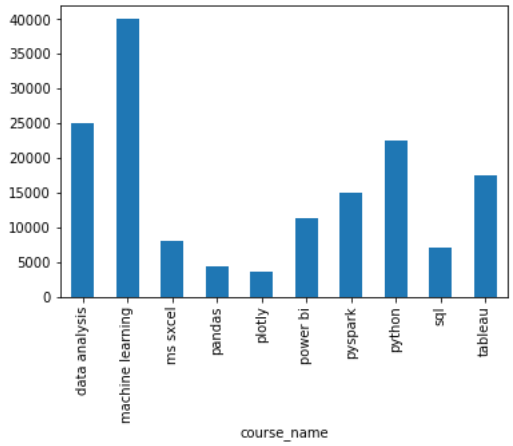
	name	course_name	price
0	Chhavi Lachman	python	2499
1	Preet Sha	python	2499
2	Fardeen Mahabir	python	2499
3	Kailash Harjo	python	2499
4	Seema Kota	python	2499
5	Shashank D'Alia	python	2499
6	Radha Dutt	python	2499
7	Pranab Natarajan	python	2499
8	Chhavi Lachman	machine learning	9999
9	Qabeel Raman	machine 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 [ ]: