Day_26_171123

January 23, 2024

1 Functions

- Arguments can be given using positional and keyword
- While passing arguments first we have to pass positional first and then keyworded arguments

[8]:
```
model = 'Q8'
name = 'Audi'
def car_spec(name, model, year):
    name = 'Auto'
    print(f"{name}")
    print(f"{model}")
    print(f"{year}")
car_spec(name,model,year=2000)
```

```
Auto
Q8
2000
```

[6]:
```
car_spec(year=2019, name='Benz',model = '800')
```

```
Benz
800
2019
```

[12]:
```
def multi(a,b):
    res = a*b
    print("Multiplication of two numbers:",res)
    return 1
ret = multi(3,10)
```

```
Multiplication of two numbers: 30
```

2 Star Patterns

Right Angle triangle

[34]:
```
n = int(input("No of rows want:"))
for i in range(1,n+1):
    for k in range(i):
```

```
        print("*",end=" ")
    print()
```

```
No of rows want: 4
```

```
*
* *
* * *
* * * *
```

Reverse Mirror right angled triangle

[43]:
```
n = int(input("No of rows want:"))
for i in range(1,n+1):
    print(" "*(i-1),"*"*n,end=" ")
    n -= 1
    print()
```

```
No of rows want: 9
```

```
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *
```

Reversed Equilateral Triangle

[70]:
```
n = int(input("No of rows want:"))
for i in range(1,n+1):
    print(" "*(i-1),"*"*n,end=" ")
    n -= 2
    print()
    if n==0:
        print(" "*(i),"*",end=" ")
```

```
No of rows want: 10
```

```
 **********
  ********
   ******
    ****
     **
      *
```

Equilateral Triangle

[76]:
```
n = int(input("No of rows want:"))
for i in range(1,n+1):
    print(" "*n,end=" ")
    n -= 1
    for k in range(i):
        print("*",end=" ")
    print()
```

```
No of rows want: 10
          *
         * *
        * * *
       * * * *
      * * * * *
     * * * * * *
    * * * * * * *
   * * * * * * * *
  * * * * * * * * *
 * * * * * * * * * *
```

Names in Stars

[91]:
```
letter_patterns = {
    'A': ["  *  ", " * * ", "*   *", "*****", "*   *", "*   *"],
    'B': ["**** ", "*   *", "**** ", "*   *", "**** "],
    'C': [" *** ", "*   *", "*    ", "*   *", " *** "],
    'D': ["**** ", "*   *", "*   *", "*   *", "**** "],
    'E': ["*****", "*    ", "**** ", "*    ", "*****"],
    'F': ["*****", "*    ", "**** ", "*    ", "*    "],
    'G': [" ****", "*    ", "*  **", "*   *", " *** "],
    'H': ["*   *", "*   *", "*****", "*   *", "*   *"],
    'I': ["*****", "  *  ", "  *  ", "  *  ", "*****"],
    'J': ["  ***", "   * ", "   * ", "*  * ", " **  "],
    'K': ["*   *", "*  * ", "***  ", "*  * ", "*   *"],
    'L': ["*    ", "*    ", "*    ", "*    ", "*****"],
    'M': ["*   *", "** **", "* * *", "*   *", "*   *"],
    'N': ["*   *", "**  *", "* * *", "*  **", "*   *"],
    'O': [" *** ", "*   *", "*   *", "*   *", " *** "],
    'P': ["**** ", "*   *", "**** ", "*    ", "*    "],
    'Q': [" *** ", "*   *", "* * *", "*  **", " ** *"],
    'R': ["**** ", "*   *", "**** ", "*  * ", "*   *"],
    'S': [" ****", "*    ", " *** ", "    *", "**** "],
    'T': ["*****", "  *  ", "  *  ", "  *  ", "  *  "],
    'U': ["*   *", "*   *", "*   *", "*   *", " *** "],
```

```
    'V': ["*   *", "*   *", " * * ", " * * ", "  *  "],
    'W': ["*   *", "*   *", "* * *", "** **", "*   *"],
    'X': ["*   *", " * * ", "  *  ", " * * ", "*   *"],
    'Y': ["*   *", " * * ", "  *  ", "  *  ", "  *  "],
    'Z': ["*****", "   * ", "  *  ", " *   ", "*****"],
    ' ': ["     ", "     ", "     ", "     ", "     "]
}

name = input("Enter your name: ")
def star_name(name):
    for row in range(5):
        for letter in name.upper():
            if letter in letter_patterns:
                print(letter_patterns[letter][row], end="  ")
            else:
                print(letter_patterns[' '][row], end="  ")
        print()
star_name(name)
```

```
Enter your name:  KGF
*   *   ****  *****
*  *   *      *
***    *  **  ****
*  *   *   *  *
*   *   ***   *
```

[ ]:

Day_37_021223

January 23, 2024

[3]:
```
import numpy as np
import pandas as pd
!gdown 1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
!gdown 1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
movies = pd.read_csv("movies.csv",index_col=0)
directors = pd.read_csv("directors.csv",index_col=0)
data = pd.merge(movies,directors,left_on="director_id",right_on='id',how='left')
data.drop('id_y',axis=1,inplace=True)
data.rename({"id_x":"movies_id"},axis=1,inplace=True)
data
```

```
Downloading…
From: https://drive.google.com/uc?id=1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\movies.csv

  0%|          | 0.00/112k [00:00<?, ?B/s]
100%|##########| 112k/112k [00:00<00:00, 1.60MB/s]
Downloading…
From: https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\directors.csv

  0%|          | 0.00/65.4k [00:00<?, ?B/s]
100%|##########| 65.4k/65.4k [00:00<00:00, 41.2MB/s]
```

[3]:

	movies_id	budget	popularity	revenue	\
0	43597	237000000	150	2787965087	
1	43598	300000000	139	961000000	
2	43599	245000000	107	880674609	
3	43600	250000000	112	1084939099	
4	43602	258000000	115	890871626	
…	…	…	…	…	
1460	48363	0	3	321952	
1461	48370	27000	19	3151130	
1462	48375	0	7	0	
1463	48376	0	3	0	
1464	48395	220000	14	2040920	

	title	vote_average	vote_count	\

```
0                                     Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                    Spectre           6.3        4466
3                      The Dark Knight Rises           7.6        9106
4                                Spider-Man 3           5.9        3576
…                                         …           …          …
1460                        The Last Waltz           7.9          64
1461                                Clerks           7.4         755
1462                               Rampage           6.0         131
1463                               Slacker           6.4          77
1464                            El Mariachi          6.6         238

      director_id  year month       day      director_name gender
0            4762  2009   Dec  Thursday      James Cameron   Male
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …    …    …       …                …    …
1460         4809  1978   May    Monday    Martin Scorsese   Male
1461         5369  1994   Sep   Tuesday        Kevin Smith   Male
1462         5148  2009   Aug    Friday           Uwe Boll   Male
1463         5535  1990   Jul    Friday  Richard Linklater   Male
1464         5097  1992   Sep    Friday   Robert Rodriguez    NaN

[1465 rows x 13 columns]
```

1 Grouping in Pandas

[4]:
```
data.groupby('director_name').nunique()
```

[4]:
```
                           movies_id  budget  popularity  revenue  title  \
director_name
Adam McKay                         6       6           6        6      6
Adam Shankman                      8       8           7        8      8
Alejandro González Iñárritu        6       6           6        6      6
Alex Proyas                        5       5           5        5      5
Alexander Payne                    5       5           5        5      5
…                                …       …           …        …      …
Wes Craven                        10       7           9       10     10
Wolfgang Petersen                  7       7           7        7      7
Woody Allen                       18       9          13       10     18
Zack Snyder                        7       7           7        7      7
Zhang Yimou                        6       4           6        4      6

                           vote_average  vote_count  director_id  year  \
```

```
director_name
Adam McKay                              6           6           1     6
Adam Shankman                           8           8           1     7
Alejandro González Iñárritu             6           6           1     6
Alex Proyas                             5           5           1     5
Alexander Payne                         3           5           1     5
…                                     …           …           …     …
Wes Craven                              9          10           1     9
Wolfgang Petersen                       6           7           1     7
Woody Allen                            12          18           1    18
Zack Snyder                             5           7           1     7
Zhang Yimou                             5           6           1     6

                              month  day  gender
director_name
Adam McKay                        3    2       1
Adam Shankman                     5    2       1
Alejandro González Iñárritu       5    3       1
Alex Proyas                       4    3       1
Alexander Payne                   4    2       0
…                               …  …    …
Wes Craven                        6    5       1
Wolfgang Petersen                 5    3       1
Woody Allen                       9    6       1
Zack Snyder                       4    4       1
Zhang Yimou                       2    3       1

[199 rows x 12 columns]
```

2 Get how many groups are groupped

[5]:
```
data.groupby('director_name').ngroups
```

[5]:
```
199
```

3 Displaying the groups

[6]:
```
data.groupby('director_name').groups
```

[6]:
```
{'Adam McKay': [176, 323, 366, 505, 839, 916], 'Adam Shankman': [265, 300, 350,
404, 458, 843, 999, 1231], 'Alejandro González Iñárritu': [106, 749, 1015, 1034,
1077, 1405], 'Alex Proyas': [95, 159, 514, 671, 873], 'Alexander Payne': [793,
1006, 1101, 1211, 1281], 'Andrew Adamson': [11, 43, 328, 501, 947], 'Andrew
Niccol': [533, 603, 701, 722, 1439], 'Andrzej Bartkowiak': [349, 549, 754, 911,
924], 'Andy Fickman': [517, 681, 909, 926, 973, 1023], 'Andy Tennant': [314,
320, 464, 593, 676, 885], 'Ang Lee': [99, 134, 748, 840, 1089, 1110, 1132,
```

1184], 'Anne Fletcher': [610, 650, 736, 789, 1206], 'Antoine Fuqua': [310, 338, 424, 467, 576, 808, 818, 1105], 'Atom Egoyan': [946, 1128, 1164, 1194, 1347, 1416], 'Barry Levinson': [313, 319, 471, 594, 878, 898, 1013, 1037, 1082, 1143, 1185, 1345, 1378], 'Barry Sonnenfeld': [13, 48, 90, 205, 591, 778, 783], 'Ben Stiller': [209, 212, 547, 562, 850], 'Bill Condon': [102, 307, 902, 1233, 1381], 'Bobby Farrelly': [352, 356, 481, 498, 624, 630, 654, 806, 928, 972, 1111], 'Brad Anderson': [1163, 1197, 1350, 1419, 1430], 'Brett Ratner': [24, 39, 188, 207, 238, 292, 405, 456, 920], 'Brian De Palma': [228, 255, 318, 439, 747, 905, 919, 1088, 1232, 1261, 1317, 1354], 'Brian Helgeland': [512, 607, 623, 742, 933], 'Brian Levant': [418, 449, 568, 761, 860, 1003], 'Brian Robbins': [416, 441, 669, 962, 988, 1115], 'Bryan Singer': [6, 32, 33, 44, 122, 216, 297, 1326], 'Cameron Crowe': [335, 434, 488, 503, 513, 698], 'Catherine Hardwicke': [602, 695, 724, 937, 1406, 1412], 'Chris Columbus': [117, 167, 204, 218, 229, 509, 656, 897, 996, 1086, 1129], 'Chris Weitz': [17, 500, 794, 869, 1202, 1267], 'Christopher Nolan': [3, 45, 58, 59, 74, 565, 641, 1341], 'Chuck Russell': [177, 410, 657, 1069, 1097, 1339], 'Clint Eastwood': [369, 426, 447, 482, 490, 520, 530, 535, 645, 727, 731, 786, 787, 899, 974, 986, 1167, 1190, 1313], 'Curtis Hanson': [494, 579, 606, 711, 733, 1057, 1310], 'Danny Boyle': [527, 668, 1083, 1085, 1126, 1168, 1287, 1385], 'Darren Aronofsky': [113, 751, 1187, 1328, 1363, 1458], 'Darren Lynn Bousman': [1241, 1243, 1283, 1338, 1440], 'David Ayer': [50, 273, 741, 1024, 1146, 1407], 'David Cronenberg': [541, 767, 994, 1055, 1254, 1268, 1334], 'David Fincher': [62, 213, 253, 383, 398, 478, 522, 555, 618, 785], 'David Gordon Green': [543, 862, 884, 927, 1376, 1418, 1432, 1459], 'David Koepp': [443, 644, 735, 1041, 1209], 'David Lynch': [583, 1161, 1264, 1340, 1456], 'David O. Russell': [422, 556, 609, 896, 982, 989, 1229, 1304], 'David R. Ellis': [582, 634, 756, 888, 934], 'David Zucker': [569, 619, 965, 1052, 1175], 'Dennis Dugan': [217, 260, 267, 293, 303, 718, 780, 977, 1247], 'Donald Petrie': [427, 507, 570, 649, 858, 894, 1106, 1331], 'Doug Liman': [52, 148, 251, 399, 544, 1318, 1451], 'Edward Zwick': [92, 182, 346, 566, 791, 819, 825], 'F. Gary Gray': [308, 402, 491, 523, 697, 833, 1272, 1380], 'Francis Ford Coppola': [487, 559, 622, 646, 772, 1076, 1155, 1253, 1312], 'Francis Lawrence': [63, 72, 109, 120, 679], 'Frank Coraci': [157, 249, 275, 451, 577, 599, 963], 'Frank Oz': [193, 355, 473, 580, 712, 813, 987], 'Garry Marshall': [329, 496, 528, 571, 784, 893, 1029, 1169], 'Gary Fleder': [518, 667, 689, 867, 981, 1165], 'Gary Winick': [258, 797, 798, 804, 1454], 'Gavin O'Connor': [820, 841, 939, 953, 1444], 'George A. Romero': [250, 1066, 1096, 1278, 1367, 1396], 'George Clooney': [343, 450, 831, 966, 1302], 'George Miller': [78, 103, 233, 287, 1250, 1403, 1450], 'Gore Verbinski': [1, 8, 9, 107, 119, 633, 1040], 'Guillermo del Toro': [35, 252, 419, 486, 1118], 'Gus Van Sant': [595, 1018, 1027, 1159, 1240, 1311, 1398], 'Guy Ritchie': [124, 215, 312, 1093, 1225, 1269, 1420], 'Harold Ramis': [425, 431, 558, 586, 788, 1137, 1166, 1325], 'Ivan Reitman': [274, 643, 816, 883, 910, 935, 1134, 1242], 'James Cameron': [0, 19, 170, 173, 344, 1100, 1320], 'James Ivory': [1125, 1152, 1180, 1291, 1293, 1390, 1397], 'James Mangold': [140, 141, 557, 560, 829, 845, 958, 1145], 'James Wan': [30, 617, 1002, 1047, 1337, 1417, 1424], 'Jan de Bont': [155, 224, 231, 270, 781], 'Jason Friedberg': [812, 1010, 1012, 1014, 1036], 'Jason Reitman': [792, 1092, 1213, 1295, 1299], 'Jaume Collet-Serra': [516, 540, 640, 725, 1011, 1189], 'Jay Roach': [195, 359, 389,

```
397, 461, 703, 859, 1072], 'Jean-Pierre Jeunet': [423, 485, 605, 664, 765], 'Joe
Dante': [284, 525, 638, 1226, 1298, 1428], 'Joe Wright': [85, 432, 553, 803,
814, 855], 'Joel Coen': [428, 670, 691, 707, 721, 889, 906, 980, 1157, 1238,
1305], 'Joel Schumacher': [128, 184, 348, 484, 572, 614, 652, 764, 876, 886,
1108, 1230, 1280], 'John Carpenter': [537, 663, 686, 861, 938, 1028, 1080, 1102,
1329, 1371], 'John Glen': [601, 642, 801, 847, 864], 'John Landis': [524, 868,
1276, 1384, 1435], 'John Madden': [457, 882, 1020, 1249, 1257], 'John
McTiernan': [127, 214, 244, 351, 534, 563, 648, 782, 838, 1074], 'John
Singleton': [294, 489, 732, 796, 1120, 1173, 1316], 'John Whitesell': [499, 632,
763, 1119, 1148], 'John Woo': [131, 142, 264, 371, 420, 675, 1182], 'Jon
Favreau': [46, 54, 55, 382, 759, 1346], 'Jon M. Chu': [100, 225, 810, 1099,
1186], 'Jon Turteltaub': [64, 180, 372, 480, 760, 846, 1171], 'Jonathan Demme':
[277, 493, 1000, 1123, 1215], 'Jonathan Liebesman': [81, 143, 339, 1117, 1301],
'Judd Apatow': [321, 710, 717, 865, 881], 'Justin Lin': [38, 123, 246, 1437,
1447], 'Kenneth Branagh': [80, 197, 421, 879, 1094, 1277, 1288], 'Kenny Ortega':
[412, 852, 1228, 1315, 1365], 'Kevin Reynolds': [53, 502, 639, 1019, 1059], …}
```

4 Accessing the grouped elements

[7]:
```
data.groupby('director_name').get_group('James Cameron')
```

[7]:

	movies_id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day	director_name	gender
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday	James Cameron	Male
19	43622	200000000	100	1845034188	Titanic	7.5	7562	4762	1997	Nov	Tuesday	James Cameron	Male
170	43876	100000000	101	520000000	Terminator 2: Judgment Day	7.7	4185	4762	1991	Jul	Monday	James Cameron	Male
173	43879	115000000	38	378882411	True Lies	6.8	1116	4762	1994	Jul	Thursday	James Cameron	Male
344	44184	70000000	24	90000098	The Abyss	7.1	808	4762	1989	Aug	Wednesday	James Cameron	Male
1100	46000	18500000	67	183316455	Aliens	7.7	3220	4762	1986	Jul	Friday	James Cameron	Male
1320	47036	6400000	74	78371200	The Terminator	7.3	4128	4762	1984				

```
1320   Oct     Friday  James Cameron   Male
```

[8]:
```
data.groupby('director_name')['title'].count().sort_values(ascending=False)
```

[8]:
```
director_name
Steven Spielberg    26
Clint Eastwood      19
Martin Scorsese     19
Woody Allen         18
Robert Rodriguez    16
                    ..
Paul Weitz           5
John Madden          5
Paul Verhoeven       5
John Whitesell       5
Kevin Reynolds       5
Name: title, Length: 199, dtype: int64
```

[9]:
```
data.groupby('director_name')['title'].value_counts()
```

[9]:
```
director_name  title
Adam McKay     Anchorman 2: The Legend Continues                1
               Anchorman: The Legend of Ron Burgundy            1
               The Other Guys                                   1
               The Big Short                                    1
               Talladega Nights: The Ballad of Ricky Bobby      1
                                                               ..
Zhang Yimou    Hero                                             1
               Curse of the Golden Flower                       1
               Coming Home                                      1
               A Woman, a Gun and a Noodle Shop                 1
               The Flowers of War                               1
Name: count, Length: 1465, dtype: int64
```

[10]:
```
data.groupby('director_name')['year'].aggregate(['min','max'])
```

[10]:
```
                              min   max
director_name
Adam McKay                   2004  2015
Adam Shankman                2001  2012
Alejandro González Iñárritu  2000  2015
Alex Proyas                  1994  2016
Alexander Payne              1999  2013
…                             …     …
Wes Craven                   1984  2011
Wolfgang Petersen            1981  2006
Woody Allen                  1977  2013
```

```
Zack Snyder                    2004  2016
Zhang Yimou                    2002  2014

[199 rows x 2 columns]
```

5 Get me the list of High budget directors

- Atleast 1 movie with 1 Million Budget

Getting max budget of directors

[11]:
```
data_dir_budget = data.groupby('director_name')['budget'].max().reset_index()
```

Names of directors who have more than 1 million budget movies

[12]:
```
names = data_dir_budget.
  ↪loc[data_dir_budget['budget']>=100000000]['director_name']
```

Checking whether names are present in data

[13]:
```
data.loc[data['director_name'].isin(names)]
```

[13]:
```
      movies_id     budget  popularity     revenue  \
0         43597  237000000         150  2787965087
1         43598  300000000         139   961000000
2         43599  245000000         107   880674609
3         43600  250000000         112  1084939099
4         43602  258000000         115   890871626
…           …          …           …           …
1450      48267     400000          33   100000000
1451      48268     200000          13     4505922
1452      48274          0           5     2611555
1458      48335      60000          27     3221152
1460      48363          0           3      321952

                                      title  vote_average  vote_count  \
0                                    Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                              Spider-Man 3           5.9        3576
…                                        …             …           …
1450                               Mad Max           6.6        1213
1451                              Swingers           6.8         253
1452                                 Three           6.3          31
1458                                    Pi           7.1         586
1460                         The Last Waltz           7.9          64

      director_id  year month       day      director_name gender
```

```
0            4762  2009   Dec  Thursday      James Cameron   Male
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …     …     …       …                …     …
1450         4845  1979   Apr  Thursday      George Miller   Male
1451         4813  1996   Oct    Friday         Doug Liman   Male
1452         4936  2010   Dec  Thursday         Tom Tykwer   Male
1458         4881  1998   Jul    Friday   Darren Aronofsky   Male
1460         4809  1978   May    Monday    Martin Scorsese   Male

[679 rows x 13 columns]
```

[21]:
```
def high_budget(data):
    return data['budget'].max() >= 100000000

data.groupby('director_name').filter(high_budget)
```

[21]:
```
      movies_id     budget  popularity     revenue  \
0         43597  237000000         150  2787965087
1         43598  300000000         139   961000000
2         43599  245000000         107   880674609
3         43600  250000000         112  1084939099
4         43602  258000000         115   890871626
…           …          …           …           …
1450      48267     400000          33   100000000
1451      48268     200000          13     4505922
1452      48274          0           5     2611555
1458      48335      60000          27     3221152
1460      48363          0           3      321952

                                      title  vote_average  vote_count  \
0                                    Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                                Spider-Man 3           5.9        3576
…                                         …             …           …
1450                                Mad Max           6.6        1213
1451                               Swingers           6.8         253
1452                                  Three           6.3          31
1458                                     Pi           7.1         586
1460                          The Last Waltz           7.9          64

      director_id  year month       day      director_name gender
0            4762  2009   Dec  Thursday      James Cameron   Male
```

```
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …     …     …       …                 …     …
1450         4845  1979   Apr  Thursday      George Miller   Male
1451         4813  1996   Oct    Friday         Doug Liman   Male
1452         4936  2010   Dec  Thursday         Tom Tykwer   Male
1458         4881  1998   Jul    Friday   Darren Aronofsky   Male
1460         4809  1978   May    Monday    Martin Scorsese   Male

[679 rows x 13 columns]
```

6 Find out the Risky Movies

- Average Revenue of the Director - 10, 20, 15, 20, 18 - 21M
- Risky - 21M : 25M,30M, 18M, 10M, 50M

[30]:
```
def is_risky(x):
    x['is_risky'] = (x['budget']-x['revenue'].mean())>= 0
    return x

data_risky = data.groupby('director_name').apply(is_risky)
```

[31]:
```
data_risky.head()
```

[31]:
```
                    movies_id     budget  popularity    revenue  \
director_name
Adam McKay    176      43882  100000000          24  170432927
              323      44151   72500000          12  162966177
              366      44236   65000000          22  128107642
              505      44503   50000000          38  173649015
              839      45301   28000000          57  133346506

                                                      title  vote_average  \
director_name
Adam McKay    176                                The Other Guys           6.1
              323  Talladega Nights: The Ballad of Ricky Bobby           6.2
              366                                  Step Brothers           6.5
              505            Anchorman 2: The Legend Continues           6.0
              839                                  The Big Short           7.3

                    vote_count  director_id  year month        day  \
director_name
Adam McKay    176        1383         4925  2010   Aug     Friday
              323         491         4925  2006   Aug     Friday
              366        1062         4925  2008   Jul     Friday
```

```
              505         923         4925  2013   Dec  Wednesday
              839        2607         4925  2015   Dec     Friday

                  director_name gender  is_risky
director_name
Adam McKay    176    Adam McKay   Male     False
              323    Adam McKay   Male     False
              366    Adam McKay   Male     False
              505    Adam McKay   Male     False
              839    Adam McKay   Male     False
```

[32]:
```
data_risky.loc[data_risky['is_risky']==True]
```

[32]:
```
                        movies_id     budget  popularity    revenue  \
director_name
Andrzej Bartkowiak 349       44192   60000000          29   55987321
Atom Egoyan        946       45538   25000000           4          0
                   1194      46370   15000000          26    8459458
                   1347      47224    5000000           7    3263585
Brett Ratner       24        43630  210000000           3  459359555
…                            …         …          …         …
Uwe Boll           944       45536   25000000           7    2405420
                   1058      45834   20000000           9   10442808
                   1383      47453    3500000           4          0
Wayne Wang         468       44419   55000000          19  154906693
Zhang Yimou        192       43914   94000000          12   95311434

                                              title  vote_average  \
director_name
Andrzej Bartkowiak 349                         Doom           5.0
Atom Egoyan        946          Where the Truth Lies           5.9
                   1194                        Chloe           5.9
                   1347          The Sweet Hereafter           6.8
Brett Ratner       24          X-Men: The Last Stand           6.3
…                                                …           …
Uwe Boll           944                   BloodRayne           3.5
                   1058            Alone in the Dark           3.1
                   1383  In the Name of the King III           3.3
Wayne Wang         468             Maid in Manhattan           5.6
Zhang Yimou        192            The Flowers of War           7.1

                        vote_count  director_id  year month        day  \
director_name
Andrzej Bartkowiak 349         609         5061  2005   Oct   Thursday
Atom Egoyan        946          66         5599  2005   Oct     Friday
                   1194        498         5599  2009   Mar  Wednesday
                   1347        103         5599  1997   May  Wednesday
```

```
Brett Ratner       24          3525         4786  2006   May  Wednesday
…                             …          …   …   …        …
Uwe Boll           944          118         5148  2005   Oct   Saturday
                   1058         173         5148  2005   Jan     Friday
                   1383          19         5148  2013   Dec     Friday
Wayne Wang         468          485         5162  2002   Dec     Friday
Zhang Yimou        192          187         4945  2011   Dec   Thursday

                                director_name gender  is_risky
director_name
Andrzej Bartkowiak 349   Andrzej Bartkowiak   Male      True
Atom Egoyan        946          Atom Egoyan   Male      True
                   1194         Atom Egoyan   Male      True
                   1347         Atom Egoyan   Male      True
Brett Ratner       24          Brett Ratner   Male      True
…                                     …    …       …
Uwe Boll           944             Uwe Boll   Male      True
                   1058            Uwe Boll   Male      True
                   1383            Uwe Boll   Male      True
Wayne Wang         468           Wayne Wang    NaN      True
Zhang Yimou        192          Zhang Yimou   Male      True

[131 rows x 14 columns]
```

[ ]:

Day_29_241123

January 23, 2024

Logical Functions

[1]:
```
import numpy as np
```

[2]:
```
a  = np.array([6,4,5,0])
b = np.array([4,3,2,1])
```

[3]:
```
a < b
```

[3]:
```
array([False, False, False,  True])
```

any function will return true if one condition is True otherwise it will return False

[4]:
```
np.any(a<b)
```

[4]:
```
True
```

all function will return False if any one condition is False

[5]:
```
np.all(a>b)
```

[5]:
```
False
```

Defining 2D Arrays

[6]:
```
a = np.array([1,2,3,4,5,6,7,8,9,0,1,1,22,1,22,1])
```

[7]:
```
b = np.array([[1,2,3],[4,5,6]])
b
```

[7]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[8]:
```
b.shape
```

[8]:
```
(2, 3)
```

[9]:
```
a.size
```

[9]:
```
16
```

Reshaping a 1D array into 2D Array

[10]:
```
a = a.reshape(4,4)
```

[11]:
```
a
```

[11]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9,  0,  1,  1],
       [22,  1, 22,  1]])
```

[12]:
```
d = np.arange(12).reshape(2,6)
```

[13]:
```
d
```

[13]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
```

Transpose of a Matrix

[14]:
```
d.T
```

[14]:
```
array([[ 0,  6],
       [ 1,  7],
       [ 2,  8],
       [ 3,  9],
       [ 4, 10],
       [ 5, 11]])
```

[15]:
```
a.T
```

[15]:
```
array([[ 1,  5,  9, 22],
       [ 2,  6,  0,  1],
       [ 3,  7,  1, 22],
       [ 4,  8,  1,  1]])
```

Quiz

[16]:
```
c = np.array([1,2,3,4,5])
mask = (c%2 ==0)
c[mask] = -1
c
```

[16]:
```
array([ 1, -1,  3, -1,  5])
```

Decreasing the Dimensions (Flatten) 2D to 1D

[17]:
```
a
```

[17]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9,  0,  1,  1],
```

```
       [22,  1, 22,  1]])
```

[18]:
```
a.flatten()
```

[18]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9,  0,  1,  1, 22,  1, 22,  1])
```

Indexing/Slicing over arrays

[19]:
```
a[[0,1,2],[0,1,2]]
```

[19]:
```
array([1, 6, 1])
```

[20]:
```
a
```

[20]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9,  0,  1,  1],
       [22,  1, 22,  1]])
```

[21]:
```
a[:]
```

[21]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9,  0,  1,  1],
       [22,  1, 22,  1]])
```

[22]:
```
a[0:2] # a[rows : columns]
```

[22]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

[23]:
```
a[1:3,2:3]
```

[23]:
```
array([[7],
       [1]])
```

Masking (Fancy indexing)

[24]:
```
a[a>2].reshape(3,3)
```

[24]:
```
array([[ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 22, 22]])
```

[25]:
```
np.max(a,axis=1) # When axis = 0 it will consider Column, axis = 1 it will␣
↪consider row
```

[25]:
```
array([ 4,  8,  9, 22])
```

[26]:
```
a
```

[26]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9,  0,  1,  1],
       [22,  1, 22,  1]])
```

[27]:
```
np.sum(a,axis=1)
```

[27]:
```
array([10, 26, 11, 46])
```

[28]:
```
np.sort(a) # Default axis is 1 Which is Row
```

[28]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 0,  1,  1,  9],
       [ 1,  1, 22, 22]])
```

[29]:
```
np.argmin(a) # argmin will return the index of min value
```

[29]:
```
9
```

[32]:
```
a = a.reshape(1,16)
np.argsort(a)
```

[32]:
```
array([[ 9,  0, 10, 11, 13, 15,  1,  2,  3,  4,  5,  6,  7,  8, 12, 14]],
      dtype=int64)
```

[ ]:

Day_32_271123

January 23, 2024

[1]:
```
import numpy as np
```

1 Shallow Copy & Deep Copy

[2]:
```
a = np.arange(1,6)
a
```

[2]:
```
array([1, 2, 3, 4, 5])
```

- Stride is nothing but step size in array

Shallow Copy - The other variable shares the memory address of a variable

[3]:
```
b = a
```

[4]:
```
b
```

[4]:
```
array([1, 2, 3, 4, 5])
```

- Here assigning a to b is a shallow Copy
- But when we made any changes to original array it will create a new adress

[5]:
```
x = a+1
```

[6]:
```
x
```

[6]:
```
array([2, 3, 4, 5, 6])
```

[7]:
```
b = a[::2]
```

[8]:
```
b
```

[8]:
```
array([1, 3, 5])
```

- Here it is shallow copy, The changes will be made in header
- Initially (a) ### Metadata of a | Headers | | |————|————| | Shape | (5,) | | ndim | 1 | | Size | 5 | | Stride | 1 |
- After assigning (b) ### Metadata of b | Headers | | |————|————| | Shape | (5,) | | ndim | 1 | | Size | 5 | | Stride | 2 |

[9]:
```
a = np.array([1,2,3,4,5])
b = a
```

2 Function to check whether Memory Sharing happens or not

[10]:
```
np.shares_memory(a,b)
```

[10]:
```
True
```

[11]:
```
b = a+1
np.shares_memory(a,b)
```

[11]:
```
False
```

3 Splitting in 1D

[12]:
```
a = np.arange(9)
print(a)
b = np.split(a,3) # Equal Splits
```

```
[0 1 2 3 4 5 6 7 8]
```

[13]:
```
b
```

[13]:
```
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]
```

[14]:
```
b = np.split(a,(2,4,7)) # Split using Index
b
```

[14]:
```
[array([0, 1]), array([2, 3]), array([4, 5, 6]), array([7, 8])]
```

4 Splitting in 2D

[15]:
```
x = np.arange(1,17).reshape(4,4)
x
```

[15]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

[16]:
```
np.split(x,2) #Divides into equal rows defaulty with axis = 0
```

[16]:
```
[array([[1, 2, 3, 4],
        [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12],
```

```
        [13, 14, 15, 16]])]
```

[17]:
```
np.hsplit(x,2) #Horizontal split
```

[17]:
```
[array([[ 1,  2],
        [ 5,  6],
        [ 9, 10],
        [13, 14]]),
 array([[ 3,  4],
        [ 7,  8],
        [11, 12],
        [15, 16]])]
```

[18]:
```
np.vsplit(x,2) #Vertical Split
```

[18]:
```
[array([[1, 2, 3, 4],
        [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12],
        [13, 14, 15, 16]])]
```

5 Stacking

[19]:
```
a = np.arange(10)
```

[20]:
```
np.vstack((a,a,a))
```

[20]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

[21]:
```
np.hstack((a,a,a))
```

[21]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1,
       2, 3, 4, 5, 6, 7, 8, 9])
```

[22]:
```
z = np.arange(9).reshape(3,3)
z
```

[22]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[23]:
```
np.vstack((z,z,z))
```

[23]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
```

```
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[24]:
```
np.hstack((z,z,z))
```

[24]:
```
array([[0, 1, 2, 0, 1, 2, 0, 1, 2],
       [3, 4, 5, 3, 4, 5, 3, 4, 5],
       [6, 7, 8, 6, 7, 8, 6, 7, 8]])
```

[25]:
```
np.concatenate((z,z,z),axis=0) #Stacking using concatenate function
```

[25]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8],
       [0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[26]:
```
g = np.arange(0,4)
g
```

[26]:
```
array([0, 1, 2, 3])
```

[27]:
```
g = np.vstack(g)
g
```

[27]:
```
array([[0],
       [1],
       [2],
       [3]])
```

[28]:
```
g = np.concatenate((g,g,g),axis=1)
g
```

[28]:
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

6 Broadcasting

[29]:
```
a = np.arange(0,40,10)
a
```

[29]:
```
array([ 0, 10, 20, 30])
```

[30]:
```
a = np.vstack((a,a,a))
```

[31]:
```
np.hstack((a,a,a))
```

[31]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30]])
```

[32]:
```
np.concatenate((a,a),axis=1)
```

[32]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30]])
```

[33]:
```
np.tile(a,(3,3))
```

[33]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30]])
```

[34]:
```
np.tile(a,(3,2))
```

[34]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30]])
```

6.1 Case 1

[35]:
```
a
```

[35]:
```
array([[ 0, 10, 20, 30],
       [ 0, 10, 20, 30],
       [ 0, 10, 20, 30]])
```

[36]:
```
v = np.vstack(a)
v
```

[36]:
```
array([[ 0, 10, 20, 30],
       [ 0, 10, 20, 30],
       [ 0, 10, 20, 30]])
```

[37]:
```
j = np.concatenate((v,v),axis =1)
j
```

[37]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30]])
```

[38]:
```
i = np.arange(0,3)
i
```

[38]:
```
array([0, 1, 2])
```

[39]:
```
j+i
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[39], line 1
----> 1 j+i

ValueError: operands could not be broadcast together with shapes (3,12) (3,)
```

6.2 Case 2

[40]:
```
j
```

[40]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30]])
```

[41]:
```
k = np.vstack((i,i,i,i))
k
```

[41]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [0, 1, 2],
       [0, 1, 2]])
```

[42]:
```
j+k
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[42], line 1
----> 1 j+k

ValueError: operands could not be broadcast together with shapes (3,12) (4,3)
```

6.3 Case 3

[43]:
```
i
```

[43]:
```
array([0, 1, 2])
```

[44]:
```
v
```

[44]:
```
array([[ 0, 10, 20, 30],
       [ 0, 10, 20, 30],
       [ 0, 10, 20, 30]])
```

[45]:
```
o = np.concatenate((v,v,v),axis = 1)
o
```

[45]:
```
array([[ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30],
       [ 0, 10, 20, 30,  0, 10, 20, 30,  0, 10, 20, 30]])
```

[46]:
```
p = np.vstack((i,i,i,i))
p
```

[46]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [0, 1, 2],
       [0, 1, 2]])
```

[47]:
```
p+o
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[47], line 1
----> 1 p+o
```

```
ValueError: operands could not be broadcast together with shapes (4,3) (3,12)
```

7 Rule 1 : If two array differ in the number of dimesions, the shape of one with fewer dimensions is padded with ones on its leading (Left side)

8 Rule 2 : If the shape of two arrays doesnot match in any dimensions, the array with shape equal to 1 is stretched to match the other shape.

[48]:
```
array = np.arange(16).reshape(4,4)
array
```

[48]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[49]:
```
arr = np.array([0,1,2,3])
```

[50]:
```
array+arr
```

[50]:
```
array([[ 0,  2,  4,  6],
       [ 4,  6,  8, 10],
       [ 8, 10, 12, 14],
       [12, 14, 16, 18]])
```

[51]:
```
a = np.arange(6)
a
```

[51]:
```
array([0, 1, 2, 3, 4, 5])
```

[52]:
```
a.shape
```

[52]:
```
(6,)
```

9 Day 33 28-11-23

10 Other function to increase the dimensions

[58]:
```
b = np.expand_dims(a,axis=0)
```

[60]:
```
print(b)
b.shape
```

```
[[0 1 2 3 4 5]]
```

[60]:
```
(1, 6)
```

[61]:
```
c = a[np.newaxis,:]
c.shape
```

[61]:
```
(1, 6)
```

[62]:
```
c
```

[62]:
```
array([[0, 1, 2, 3, 4, 5]])
```

[63]:
```
d = a[:,np.newaxis]
d
```

[63]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5]])
```

[64]:
```
e = np.arange(6).reshape(2,3)
e
```

[64]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[65]:
```
np.expand_dims(e,axis=0).shape
```

[65]:
```
(1, 2, 3)
```

[68]:
```
f = np.arange(6).reshape(2,3)
np.expand_dims(f,axis=2)
```

[68]:
```
array([[[0],
        [1],
        [2]],

       [[3],
        [4],
        [5]]])
```

11 Removing Dimensions

[78]:
```
a = np.arange(5)
b = np.expand_dims(a,axis=1)
b
```

[78]:
```
array([[0],
       [1],
       [2],
       [3],
       [4]])
```

[79]:
```
np.squeeze(b,axis=1)
```

[79]:
```
array([0, 1, 2, 3, 4])
```

[92]:
```
k = np.arange(12).reshape(1,12)
k
```

[92]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]])
```

[86]:
```
np.squeeze(k).shape
```

[86]:
```
(12,)
```

11.0.1 NumPy cannot remove the original dimension but can remove the fake dimension

[93]:
```
np.squeeze(k,axis=1).shape
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[93], line 1
----> 1 np.squeeze(k,axis=1).shape

File C:\Data\env\Lib\site-packages\numpy\core\fromnumeric.py:1558, in squeeze(a
 ↪axis)
   1556     return squeeze()
   1557 else:
-> 1558     return squeeze(axis=axis)

ValueError: cannot select an axis to squeeze out which has size not equal to on
```

[94]:
```
a = np.arange(12).reshape(12,1,1)
a
```

[94]:
```
array([[[ 0]],

       [[ 1]],

       [[ 2]],

       [[ 3]],

       [[ 4]],

       [[ 5]],

       [[ 6]],

       [[ 7]],

       [[ 8]],

       [[ 9]],

       [[10]],

       [[11]]])
```

[95]:
```
np.squeeze(a,axis=-1)
```

[95]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11]])
```

[98]:
```
np.squeeze(a,axis=-2)
```

[98]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
```

```
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11]])
```

[99]:
```
array = np.arange(10)
```

[100]:
```
array2 = array.view
```

[102]:
```
np.shares_memory(array2,array)
```

[102]: False

[101]:
```
array3 = array.copy
```

[103]:
```
np.shares_memory(array3,array)
```

[103]: False

[ ]:
```

```

Day_24_151123

January 23, 2024

1 Revision

[2]:
```
s = 'Saiteja'
z = s.ljust(15)
print(z)
z1 = z.rjust(20)
print(z1)
```

```
Saiteja
     Saiteja
```

[6]:
```
b = [1,2,3,4,5,6,7,8]
print(b[:7:2])
```

```
[1, 3, 5, 7]
```

[ ]:
```

```

Day_30_251123

January 23, 2024

1 Fitbit data analysis using NumPy 2D arrays

[2]:
```
import numpy as np
```

[1]:
```
!gdown 1vk1Pu0djiYcrdc85yUXZ_Rqq2oZNcohd
```

```
Downloading…
From: https://drive.google.com/uc?id=1vk1Pu0djiYcrdc85yUXZ_Rqq2oZNcohd
To: C:\Data\Data_science\Data Science RIA\3 Python\Codes\fit.txt

  0%|          | 0.00/3.43k [00:00<?, ?B/s]
100%|##########| 3.43k/3.43k [00:00<?, ?B/s]
```

[6]:
```
data = np.loadtxt("fit.txt",dtype='str')
```

[8]:
```
data.ndim
```

[8]:
```
2
```

[9]:
```
data.shape
```

[9]:
```
(96, 6)
```

[11]:
```
date,step_count,mood,calories_burned,hours_of_sleep,activity_status = data.T
```

[13]:
```
step_count = np.array(step_count,dtype='int')
```

[14]:
```
calories_burned = np.array(calories_burned,dtype='int')
```

[15]:
```
hours_of_sleep = np.array(hours_of_sleep,dtype='int')
```

[31]:
```
np.unique(mood,return_counts = 'True')
```

[31]:
```
(array(['Happy', 'Neutral', 'Sad'], dtype='<U10'),
 array([40, 27, 29], dtype=int64))
```

[33]:
```
unique, counts = np.unique(activity_status,return_counts = 'True')
counts
```

[33]:
```
array([42, 54], dtype=int64)
```

Operating with data and getting insights

[18]:
```
step_count.mean()
```

[18]:
```
2935.9375
```

[21]:
```
step_count.max()
```

[21]:
```
7422
```

[23]:
```
step_count.argmax()
```

[23]:
```
69
```

[26]:
```
date[step_count.argmax()]
```

[26]:
```
'14-12-2017'
```

[27]:
```
date[step_count.argmin()]
```

[27]:
```
'08-10-2017'
```

[34]:
```
calories_burned[step_count.argmax()]
```

[34]:
```
243
```

[36]:
```
np.mean(step_count[mood=='Sad'])
```

[36]:
```
2103.0689655172414
```

[38]:
```
np.mean(step_count[mood=='Happy'])
```

[38]:
```
3392.725
```

[39]:
```
np.unique(mood[step_count>4000],return_counts='True')
```

[39]:
```
(array(['Happy', 'Neutral', 'Sad'], dtype='<U10'),
 array([22,  9,  7], dtype=int64))
```

[42]:
```
np.unique(mood[step_count<2000],return_counts='True')
```

[42]:
```
(array(['Happy', 'Neutral', 'Sad'], dtype='<U10'),
 array([13,  8, 18], dtype=int64))
```

[44]:
```
np.mean(hours_of_sleep[activity_status=='Active'])
```

[44]:
```
5.4523809523809526
```

[51]:
```
a = np.arange(9,0,-1).reshape(3,3)
a
```

[51]:
```
array([[9, 8, 7],
       [6, 5, 4],
       [3, 2, 1]])
```

[57]:
```
a.sort(axis = 1)
```

[58]:
```
a
```

[58]:
```
array([[7, 8, 9],
       [4, 5, 6],
       [1, 2, 3]])
```

2 Matrix Multiplications

[12]:
```
a = np.arange(5)
```

To generate the matrix of shape (n*m) with all ones we use np.ones(shape=(n,m))

[70]:
```
b = np.ones(shape=(5)) * 2
a*b
```

[70]:
```
array([0., 2., 4., 6., 8.])
```

[71]:
```
a = np.arange(12).reshape(3,4)
```

[72]:
```
a
```

[72]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

If the matrix shapes are same we do element multiplication a*b or np.dot(a,b) but the shape is Transpose of it we use np.matmul(matrix1,matrix2) or a@b

[85]:
```
a = np.ones(shape=(3,4))
b = np.ones(shape=(4,3))
np.matmul(a,b)
```

[85]:
```
array([[4., 4., 4.],
       [4., 4., 4.],
       [4., 4., 4.]])
```

[86]:
```
c = np.ones(shape=(4,4))
d = np.ones(shape=(4,4))
np.matmul(c,d)
```

[86]:
```
array([[4., 4., 4., 4.],
       [4., 4., 4., 4.],
       [4., 4., 4., 4.],
       [4., 4., 4., 4.]])
```

[87]:
```
c@d
```

[87]:
```
array([[4., 4., 4., 4.],
       [4., 4., 4., 4.],
       [4., 4., 4., 4.],
       [4., 4., 4., 4.]])
```

[101]:
```
f = np.arange(16).reshape(4,4)
g = np.arange(16).reshape(4,4)
np.dot(f,g)
```

[101]:
```
array([[ 56,  62,  68,  74],
       [152, 174, 196, 218],
       [248, 286, 324, 362],
       [344, 398, 452, 506]])
```

[102]:
```
h = np.arange(12).reshape(4,3)
j = np.arange(12).reshape(3,4)
np.matmul(h,j)
```

[102]:
```
array([[ 20,  23,  26,  29],
       [ 56,  68,  80,  92],
       [ 92, 113, 134, 155],
       [128, 158, 188, 218]])
```

3 Vectorization

[3]:
```
z = np.arange(12)
```

[4]:
```
import math
```

[5]:
```
z
```

[5]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

[ ]:
```
x = np.vectorize(math.log)(a)
```

[ ]:
```
x
```

4 3D

[117]:
```
a = np.arange(24).reshape(2,3,4)
```

[118]:
```
a
```

[118]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[114]:
```
a.size
```

[114]:
```
24
```

[120]:
```
a.ndim
```

[120]:
```
3
```

[133]:
```
a = np.arange(12).reshape(3,4)
```

[134]:
```
a
```

[134]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[135]:
```
b = np.arange(16).reshape(4,4)
```

[136]:
```
b
```

[136]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[137]:
```
np.matmul(a,b)
```

[137]:
```
array([[ 56,  62,  68,  74],
       [152, 174, 196, 218],
       [248, 286, 324, 362]])
```

Day_25_161123

January 23, 2024

0.0.1 Functions

[2]:
```
def greeting_people():
    print("Hello")
```

[3]:
```
greeting_people()
```

```
Hello
```

[11]:
```
def greet_one(name):
    print(f"Hello {name}")

name = input("Enter you name:")
greet_one(name)
```

```
Enter you name: Sharan
```

```
Hello Sharan
```

[13]:
```
def personal_details(name,age,phno,mail):
    print(f"Given details are\nName:{name}\nAge:{age}\nPhonenumber:{phno}\nMail␣
 ↪id:{mail}")

name = input("Enter your name:")
age = input("Enter your age:")
phno = input("Enter your phone number:")
mail = input("Enter your mail id:")
personal_details(name,age,phno,mail)
```

```
Enter your name: Sai
Enter your age: 22
Enter your phone number: 56556677
Enter your mail id: sai@gmail.com
```

```
Given details are
Name:Sai
Age:22
Phonenumber:56556677
Mail id:sai@gmail.com
```

[19]:
```
def even_odd():
    n = input("Enter number")
    try:
        if int(n)%2 == 0 or float(n)%2 == 0:
            print(f"{n} is even")
        elif int(n)%2 != 0 or float(n)%2 != 0:
            print(f"{n} is odd")
    except ValueError:
        print("Enter valid integer")

even_odd()
```

```
Enter number 77

77 is odd
```

[27]:
```
def own_type(n):
    try:
        int(n)
        print("It is integer")
    except ValueError:
        try:
            float(n)
            print("It is float")
        except ValueError:
            print("It is string")
n = input()
own_type(n)
```

```
 thrr

It is string
```

Day_28_211123

January 23, 2024

[5]:
```
import numpy as np
```

When we have a different dataTypes in a single array there is a priority followed

- string > Float > int > Boolean

[6]:
```
a = np.array([1,2,3,4,5,6,7,8,9,10,'a',2.55])
```

[7]:
```
a
```

[7]:
```
array(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'a', '2.55'],
      dtype='<U32')
```

Assigning DataTypes when creating array

[8]:
```
a = np.array([[1,2,3,4],[2,3,5,4]],dtype='float')
a
```

[8]:
```
array([[1., 2., 3., 4.],
       [2., 3., 5., 4.]])
```

When ever we want help about any function we can use help

[9]:
```
#help(np.array([1,2,3]))
```

Changing the DataType after creating array

[10]:
```
b = a.astype('int')
b
```

[10]:
```
array([[1, 2, 3, 4],
       [2, 3, 5, 4]])
```

[11]:
```
arr = np.arange(1,40,0.5)
```

[12]:
```
arr
```

[12]:
```
array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,
        6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5, 11. , 11.5,
       12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5, 16. , 16.5, 17. ,
       17.5, 18. , 18.5, 19. , 19.5, 20. , 20.5, 21. , 21.5, 22. , 22.5,
```

```
       23. , 23.5, 24. , 24.5, 25. , 25.5, 26. , 26.5, 27. , 27.5, 28. ,
       28.5, 29. , 29.5, 30. , 30.5, 31. , 31.5, 32. , 32.5, 33. , 33.5,
       34. , 34.5, 35. , 35.5, 36. , 36.5, 37. , 37.5, 38. , 38.5, 39. ,
       39.5])
```

Where will return the particular index value of the element based on condition

[13]:
```
np.where((arr>9) & (arr<39))
```

[13]:
```
(array([17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75], dtype=int64),)
```

[14]:
```
np.where(arr>2)
```

[14]:
```
(array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
        20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
        37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
        54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
        71, 72, 73, 74, 75, 76, 77], dtype=int64),)
```

[15]:
```
a
```

[15]:
```
array([[1., 2., 3., 4.],
       [2., 3., 5., 4.]])
```

Using multiple condition in where

[16]:
```
np.where(a>3,a,a*100)
```

[16]:
```
array([[100., 200., 300.,   4.],
       [200., 300.,   5.,   4.]])
```

[17]:
```
own = np.array([2,5,4,6,-3,-7,5,-3,-9])
np.where(own<0,own*10,own/10)
```

[17]:
```
array([  0.2,   0.5,   0.4,   0.6, -30. , -70. ,   0.5, -30. , -90. ])
```

0.1 Airbnb is a company send us the data in 1D array

[18]:
```
!gdown 1c0ClC8SrPwJq5rrkyMKyPn80nyHcFikK
```

```
Downloading…
From: https://drive.google.com/uc?id=1c0ClC8SrPwJq5rrkyMKyPn80nyHcFikK
To: C:\Data\Data_science\Data Science RIA\3 Python\Codes\survey.txt

  0%|          | 0.00/2.55k [00:00<?, ?B/s]
100%|##########| 2.55k/2.55k [00:00<?, ?B/s]
```

[19]:
```
score = np.loadtxt("survey.txt", dtype="int")
```

[20]:
```
score
```

[20]:
```
array([ 7, 10,  5, …,  5,  9, 10])
```

[21]:
```
score.shape
```

[21]:
```
(1167,)
```

[22]:
```
score.ndim
```

[22]:
```
1
```

[23]:
```
score.size
```

[23]:
```
1167
```

[24]:
```
score.min()
```

[24]:
```
1
```

[25]:
```
score.max()
```

[25]:
```
10
```

[26]:
```
promoters = score[score>=9].shape[0]
```

[27]:
```
detractors = score[score<=6].shape[0]
```

[28]:
```
neutral = score[(score>6)&(score<9)].shape[0]
```

[29]:
```
nps = ( (promoters/(score.shape[0])) - (detractors/(score.shape[0])) ) * 100
```

[30]:
```
print(f"Net Promoter Score: {round(nps,2)}")
```

```
Net Promoter Score: 23.74
```

1 Creating an empty array with shape

[31]:
```
arr = np.empty(shape=score.shape,dtype='U20')
```

[32]:
```
arr.shape
```

[32]:
```
(1167,)
```

[33]:
```
arr # Here U1 indicates that Unicode<lenght of element in array>
```

[33]:
```
array(['', '', '', …, '', '', ''], dtype='<U20')
```

1.0.1 Converting Continous to Categorical data

[34]:
```
arr[score>=9] = "promoters"
arr[(score>=7) & (score<=8)] = "passive"
arr[score<=6] = "detractors"
```

[35]:
```
arr
```

[35]:
```
array(['passive', 'promoters', 'detractors', …, 'detractors',
       'promoters', 'promoters'], dtype='<U20')
```

[36]:
```
arr.shape
```

[36]:
```
(1167,)
```

[37]:
```
arr[arr=='promoters'].size
```

[37]:
```
609
```

[38]:
```
arr[arr=='passive'].size
```

[38]:
```
226
```

[39]:
```
arr[arr=='detractors'].size
```

[39]:
```
332
```

What if there are more no of elements we use unique function to get unique elements

[40]:
```
unique, count = np.unique(arr,return_counts='True')
```

[41]:
```
count
```

[41]:
```
array([332, 226, 609], dtype=int64)
```

[42]:
```
pod = count[0] / count.sum() * 100
```

[43]:
```
pop = count[2] / count.sum() * 100
```

[44]:
```
nps = pop - pod
nps
```

[44]:
```
23.73607540702657
```

Day_35_301123

January 23, 2024

[131]:
```
import pandas as pd
```

[132]:
```
df = pd.read_csv("mckinsey (1).csv")
```

[133]:
```
df.head()
```

[133]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106

[134]:
```
df.shape
```

[134]:
```
(1704, 6)
```

1 Adding duplicates

[135]:
```
df.loc[1704] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1705] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1706] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1707] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1708] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1709] = ['India',1933,89778854,'Asia',86.23,897.956]
df.loc[1710] = ['India',1933,89778854,'Asia',86.23,897.956]
```

[136]:
```
df.tail()
```

[136]:

	country	year	population	continent	life_exp	gdp_cap
1706	India	1933	89778854	Asia	86.23	897.956
1707	India	1933	89778854	Asia	86.23	897.956
1708	India	1933	89778854	Asia	86.23	897.956
1709	India	1933	89778854	Asia	86.23	897.956
1710	India	1933	89778854	Asia	86.23	897.956

[137]:
```
df.duplicated()
```

[137]:
```
0       False
1       False
2       False
3       False
4       False
        …
1706     True
1707     True
1708     True
1709     True
1710     True
Length: 1711, dtype: bool
```

[138]:
```
df.loc[df.duplicated()]
```

[138]:

	country	year	population	continent	life_exp	gdp_cap
1705	India	1933	89778854	Asia	86.23	897.956
1706	India	1933	89778854	Asia	86.23	897.956
1707	India	1933	89778854	Asia	86.23	897.956
1708	India	1933	89778854	Asia	86.23	897.956
1709	India	1933	89778854	Asia	86.23	897.956
1710	India	1933	89778854	Asia	86.23	897.956

2 Removing duplicated

3 Drop duplicated and keep last one

[139]:
```
df.drop_duplicates(keep='last')
```

[139]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298
1710	India	1933	89778854	Asia	86.230	897.956000

[1705 rows x 6 columns]

4 Drop everything which are duplicated

[140]:
```
df.drop_duplicates(keep=False,inplace=True)
```

5 Working with columns and rows using Slicing

[141]:
```
df.iloc[:4,:3]
```

[141]:

	country	year	population
0	Afghanistan	1952	8425333
1	Afghanistan	1957	9240934
2	Afghanistan	1962	10267083
3	Afghanistan	1967	11537966

[142]:
```
df.loc[1:5,['country','life_exp']]
```

[142]:

	country	life_exp
1	Afghanistan	30.332
2	Afghanistan	31.997
3	Afghanistan	34.020
4	Afghanistan	36.088
5	Afghanistan	38.438

[143]:
```
df.loc[1:5,'country':'life_exp']
```

[143]:

	country	year	population	continent	life_exp
1	Afghanistan	1957	9240934	Asia	30.332
2	Afghanistan	1962	10267083	Asia	31.997
3	Afghanistan	1967	11537966	Asia	34.020
4	Afghanistan	1972	13079460	Asia	36.088
5	Afghanistan	1977	14880372	Asia	38.438

[144]:
```
df.iloc[[1,3,5],[2,4,5]]
```

[144]:

	population	life_exp	gdp_cap
1	9240934	30.332	820.853030
3	11537966	34.020	836.197138
5	14880372	38.438	786.113360

[145]:
```
df.loc[1:10:2,'country':'gdp_cap':2]
```

[145]:

	country	population	life_exp
1	Afghanistan	9240934	30.332
3	Afghanistan	11537966	34.020
5	Afghanistan	14880372	38.438
7	Afghanistan	13867957	40.822
9	Afghanistan	22227415	41.763

[146]:
```
df.loc[[3,4,5],'country':'gdp_cap':2]
```

[146]:

	country	population	life_exp
3	Afghanistan	11537966	34.020
4	Afghanistan	13079460	36.088
5	Afghanistan	14880372	38.438

6 Sorting

[147]:
```
df.sort_values(['year','life_exp'],ascending=[False,True])
```

[147]:

	country	year	population	continent	life_exp	gdp_cap
1463	Swaziland	2007	1133066	Africa	39.613	4513.480643
1043	Mozambique	2007	19951656	Africa	42.082	823.685621
1691	Zambia	2007	11746035	Africa	42.384	1271.211593
1355	Sierra Leone	2007	6144562	Africa	42.568	862.540756
887	Lesotho	2007	2012649	Africa	42.592	1569.331442
…	…	…	…	…	…	…
408	Denmark	1952	4334000	Europe	70.780	9692.385245
1464	Sweden	1952	7124673	Europe	71.860	8527.844662
1080	Netherlands	1952	10381988	Europe	72.130	8941.571858
684	Iceland	1952	147962	Europe	72.490	7267.688428
1140	Norway	1952	3327728	Europe	72.670	10095.421720

[1704 rows x 6 columns]

[148]:
```
df.sort_values(['gdp_cap','population']).head()
```

[148]:

	country	year	population	continent	life_exp	gdp_cap
334	Congo, Dem. Rep.	2002	55379852	Africa	44.966	241.165876
335	Congo, Dem. Rep.	2007	64606759	Africa	46.462	277.551859
876	Lesotho	1952	748747	Africa	42.138	298.846212
624	Guinea-Bissau	1952	580653	Africa	32.500	299.850319
333	Congo, Dem. Rep.	1997	47798986	Africa	42.587	312.188423

[149]:
```
df.sort_values(['gdp_cap','population'],ascending=[False,True]).head()
```

[149]:

	country	year	population	continent	life_exp	gdp_cap
853	Kuwait	1957	212846	Asia	58.033	113523.13290
856	Kuwait	1972	841934	Asia	67.712	109347.86700
852	Kuwait	1952	160000	Asia	55.565	108382.35290
854	Kuwait	1962	358266	Asia	60.470	95458.11176
855	Kuwait	1967	575003	Asia	64.624	80894.88326

7 Mathematical Functions

[150]:
```
le = df['life_exp']
```

[151]:
```
le.min()
```

[151]:
```
23.599
```

[152]:
```
le.max()
```

[152]:
```
82.603
```

[153]:
```
le.mean()
```

[153]:
```
59.474439366197174
```

[154]:
```
le.std()
```

[154]:
```
12.917107415241192
```

[155]:
```
le.var()
```

[155]:
```
166.851663976879
```

[156]:
```
le.mode()
```

[156]:
```
0    69.39
Name: life_exp, dtype: float64
```

[157]:
```
le.count()
```

[157]:
```
1704
```

[158]:
```
pop = df['population']
```

[159]:
```
pop.min()
```

[159]:
```
60011
```

[160]:
```
pop.max()
```

[160]:
```
1318683096
```

[161]:
```
pop.mean()
```

[161]:
```
29601212.324530516
```

[162]:
```
pop.sum()
```

[162]:
```
50440465801
```

[163]:
```
gdp = df['gdp_cap']
```

[164]:
```
gdp.min()
```

[164]:
```
241.1658765
```

[165]:
```
gdp.max()
```

[165]:
```
113523.1329
```

[166]:
```
gdp.mean()
```

[166]:
```
7215.327081212149
```

[167]:
```
gdp.sum()
```

[167]:
```
12294917.346385501
```

8 Joining & Merging Tables

[168]:
```
users = pd.DataFrame(
    {
        'user_id':[1,2,3,4,5],
        'name':['Sai','Preethi','Shamika','Veenasree','Sharan']
    }
)
```

[169]:
```
users
```

[169]:

	user_id	name
0	1	Sai
1	2	Preethi
2	3	Shamika
3	4	Veenasree
4	5	Sharan

[170]:
```
msgs = pd.DataFrame(
    {
        'user_id':[1,1,3,4,2],
        'message':['hi','hello','Fine!','How are you ?','Bye']
    }
)
```

[171]:
```
msgs
```

[171]:
```
   user_id        message
0        1             hi
1        1          hello
2        3          Fine!
3        4  How are you ?
4        2            Bye
```

[172]:
```
pd.concat([users,msgs],ignore_index=True)  # Union, vstack, full join
```

[172]:
```
   user_id       name        message
0        1        Sai            NaN
1        2    Preethi            NaN
2        3    Shamika            NaN
3        4  Veenasree            NaN
4        5     Sharan            NaN
5        1        NaN             hi
6        1        NaN          hello
7        3        NaN          Fine!
8        4        NaN  How are you ?
9        2        NaN            Bye
```

[173]:
```
pd.concat([users,msgs],axis=1) #hstack
```

[173]:
```
   user_id       name  user_id        message
0        1        Sai        1             hi
1        2    Preethi        1          hello
2        3    Shamika        3          Fine!
3        4  Veenasree        4  How are you ?
4        5     Sharan        2            Bye
```

9 Joining two tables

- ### 9.0.1 pd.merge(table1, table2, on='comman_column', how='Type_of_join')

- ### 9.0.2 table1.merge(table2, on='comman_column', how='Type_of_join')

[174]:
```
pd.merge(users,msgs,on='user_id')
```

[174]:
```
   user_id       name        message
0        1        Sai             hi
1        1        Sai          hello
2        2    Preethi            Bye
3        3    Shamika          Fine!
4        4  Veenasree  How are you ?
```

[175]:
```
users.merge(msgs,on='user_id',how='outer')
```

[175]:

	user_id	name	message
0	1	Sai	hi
1	1	Sai	hello
2	2	Preethi	Bye
3	3	Shamika	Fine!
4	4	Veenasree	How are you ?
5	5	Sharan	NaN

[176]:
```
users.merge(msgs,on='user_id',how='right')
```

[176]:

	user_id	name	message
0	1	Sai	hi
1	1	Sai	hello
2	3	Shamika	Fine!
3	4	Veenasree	How are you ?
4	2	Preethi	Bye

[179]:
```
users.rename(columns={'user_id':'id'},inplace=True)
```

[180]:
```
users
```

[180]:

	id	name
0	1	Sai
1	2	Preethi
2	3	Shamika
3	4	Veenasree
4	5	Sharan

[186]:
```
users.merge(msgs,left_on='id',right_on='user_id')
```

[186]:

	id	name	user_id	message
0	1	Sai	1	hi
1	1	Sai	1	hello
2	2	Preethi	2	Bye
3	3	Shamika	3	Fine!
4	4	Veenasree	4	How are you ?

[187]:
```
!gdown 1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
```

```
Downloading…
From: https://drive.google.com/uc?id=1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\movies.csv

  0%|          | 0.00/112k [00:00<?, ?B/s]
100%|##########| 112k/112k [00:00<00:00, 1.16MB/s]
```

[188]:
```
!gdown 1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
```

```
Downloading…
From: https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\directors.csv

  0%|          | 0.00/65.4k [00:00<?, ?B/s]
100%|##########| 65.4k/65.4k [00:00<00:00, 1.53MB/s]
```

[223]:
```
movies = pd.read_csv("movies.csv") # to choose index col throw an argument␣
↪index_col = 0
```

[224]:
```
directors = pd.read_csv("directors.csv")
```

[225]:
```
movies.shape
```

[225]:
```
(1465, 12)
```

[226]:
```
directors.shape
```

[226]:
```
(2349, 4)
```

[227]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1465 entries, 0 to 1464
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     1465 non-null   int64
 1   id             1465 non-null   int64
 2   budget         1465 non-null   int64
 3   popularity     1465 non-null   int64
 4   revenue        1465 non-null   int64
 5   title          1465 non-null   object
 6   vote_average   1465 non-null   float64
 7   vote_count     1465 non-null   int64
 8   director_id    1465 non-null   int64
 9   year           1465 non-null   int64
 10  month          1465 non-null   object
 11  day            1465 non-null   object
dtypes: float64(1), int64(8), object(3)
memory usage: 137.5+ KB
```

[228]:
```
directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2349 entries, 0 to 2348
Data columns (total 4 columns):
```

```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     2349 non-null   int64
 1   director_name  2349 non-null   object
 2   id             2349 non-null   int64
 3   gender         1724 non-null   object
dtypes: int64(2), object(2)
memory usage: 73.5+ KB
```

[229]:
```
movies.ndim
```

[229]:
```
2
```

[230]:
```
directors.ndim
```

[230]:
```
2
```

[231]:
```
movies.drop('Unnamed: 0',axis=1,inplace=True)
```

[232]:
```
directors.drop('Unnamed: 0',axis=1,inplace=True)
```

[234]:
```
movies.sort_values('vote_count',ascending=False)
```

[234]:

	id	budget	popularity	revenue	title	vote_average	vote_count	director_id
59	43693	160000000	167	825532764	Inception	8.1	13752	4765
45	43662	185000000	187	1004558444	The Dark Knight	8.2	12002	4765
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762
58	43692	165000000	724	675120017	Interstellar	8.1	10867	4765
178	43884	100000000	82	425368238	Django Unchained	7.8	10099	4927
…	…	…	…	…	…	…	…	…
1431	47962	0	0	0	Walking and Talking	6.6	7	6204
879	45373	0	0	0	The Magic Flute	6.9	6	4847
1438	48145	500000	0	0	Everything Put Together	5.0	2	4773
1440	48155	0	0	0	Alleluia! The Devil's Carnival	6.0	2	6056
1378	47387	0	0	0	An Everlasting Piece	6.0	1	5037

```
      year month        day
59    2010   Jul  Wednesday
45    2008   Jul  Wednesday
0     2009   Dec   Thursday
58    2014   Nov  Wednesday
178   2012   Dec    Tuesday
…     …    …          …
1431  1996   Jul  Wednesday
879   2006   Sep   Thursday
1438  2001   Nov     Friday
1440  2016   Mar    Tuesday
1378  2000   Dec     Friday

[1465 rows x 11 columns]
```

[ ]:

[ ]:

[ ]:

Day_39_051223

January 23, 2024

[85]:
```
import pandas as pd
import numpy as np
data = pd.read_csv("Pfizer_1.csv")
```

[86]:
```
data.head()
```

[86]:

	Date	Drug_Name	Parameter	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:30:00	7:30:00	8:30:00	9:30:00	10:30:00	11:30:00	12:30:00
0	15-10-2020	diltiazem hydrochloride	Temperature	23.0	22.0	NaN	21.0	21.0	22	23.0	21.0	22.0	20	20.0	21
1	15-10-2020	diltiazem hydrochloride	Pressure	12.0	13.0	NaN	11.0	13.0	14	16.0	16.0	24.0	18	19.0	20
2	15-10-2020	docetaxel injection	Temperature	NaN	17.0	18.0	NaN	17.0	18	NaN	NaN	23.0	23	25.0	25
3	15-10-2020	docetaxel injection	Pressure	NaN	22.0	22.0	NaN	22.0	23	NaN	NaN	27.0	26	29.0	28
4	15-10-2020	ketamine hydrochloride	Temperature	24.0	NaN	NaN	27.0	NaN	26	25.0	24.0	23.0	22	21.0	20

[87]:
```
data_melt = pd.melt(data,␣
↪id_vars=['Date','Drug_Name','Parameter'],var_name='Time',value_name='Reading')
```

[88]:
```
data_melt.head()
```

[88]:

	Date	Drug_Name	Parameter	Time	Reading
0	15-10-2020	diltiazem hydrochloride	Temperature	1:30:00	23.0
1	15-10-2020	diltiazem hydrochloride	Pressure	1:30:00	12.0
2	15-10-2020	docetaxel injection	Temperature	1:30:00	NaN
3	15-10-2020	docetaxel injection	Pressure	1:30:00	NaN
4	15-10-2020	ketamine hydrochloride	Temperature	1:30:00	24.0

[89]:
```
data_tidy = data_melt.
↪pivot(index=['Date','Drug_Name','Time'],columns='Parameter',values='Reading').
↪reset_index()
```

[90]:
```
data_tidy.head()
```

[90]:
```
Parameter        Date                Drug_Name      Time  Pressure  \
0          15-10-2020  diltiazem hydrochloride  10:30:00      18.0
1          15-10-2020  diltiazem hydrochloride  11:30:00      19.0
2          15-10-2020  diltiazem hydrochloride  12:30:00      20.0
3          15-10-2020  diltiazem hydrochloride   1:30:00      12.0
4          15-10-2020  diltiazem hydrochloride   2:30:00      13.0

Parameter  Temperature
0                 20.0
1                 20.0
2                 21.0
3                 23.0
4                 22.0
```

1 Groupping using drug name and apply function

[91]:
```
def temp_mean(x):
    x['Average temparature'] = x['Temperature'].mean()
    return x

data_tidy = data_tidy.groupby('Drug_Name').apply(temp_mean)
```

[92]:
```
data_tidy
```

[92]:
```
Parameter                          Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
…                                   …                       …       …
ketamine hydrochloride  103  17-10-2020   ketamine hydrochloride   5:30:00
                        104  17-10-2020   ketamine hydrochloride   6:30:00
                        105  17-10-2020   ketamine hydrochloride   7:30:00
                        106  17-10-2020   ketamine hydrochloride   8:30:00
                        107  17-10-2020   ketamine hydrochloride   9:30:00

Parameter                        Pressure  Temperature  Average temparature
Drug_Name
diltiazem hydrochloride 0            18.0         20.0            24.848485
```

```
                                1        19.0         20.0            24.848485
                                2        20.0         21.0            24.848485
                                3        12.0         23.0            24.848485
                                4        13.0         22.0            24.848485
…                                        …            …                       …
ketamine hydrochloride          103      11.0         17.0            17.709677
                                104      12.0         18.0            17.709677
                                105      12.0         19.0            17.709677
                                106      11.0         20.0            17.709677
                                107      12.0         21.0            17.709677

[108 rows x 6 columns]
```

[93]:
```
data_tidy[:20]
```

[93]:
```
Parameter                              Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
                        5    15-10-2020  diltiazem hydrochloride   3:30:00
                        6    15-10-2020  diltiazem hydrochloride   4:30:00
                        7    15-10-2020  diltiazem hydrochloride   5:30:00
                        8    15-10-2020  diltiazem hydrochloride   6:30:00
                        9    15-10-2020  diltiazem hydrochloride   7:30:00
                        10   15-10-2020  diltiazem hydrochloride   8:30:00
                        11   15-10-2020  diltiazem hydrochloride   9:30:00
                        36   16-10-2020  diltiazem hydrochloride  10:30:00
                        37   16-10-2020  diltiazem hydrochloride  11:30:00
                        38   16-10-2020  diltiazem hydrochloride  12:30:00
                        39   16-10-2020  diltiazem hydrochloride   1:30:00
                        40   16-10-2020  diltiazem hydrochloride   2:30:00
                        41   16-10-2020  diltiazem hydrochloride   3:30:00
                        42   16-10-2020  diltiazem hydrochloride   4:30:00
                        43   16-10-2020  diltiazem hydrochloride   5:30:00

Parameter                          Pressure  Temperature  Average temparature
Drug_Name
diltiazem hydrochloride 0          18.0         20.0            24.848485
                        1          19.0         20.0            24.848485
                        2          20.0         21.0            24.848485
                        3          12.0         23.0            24.848485
                        4          13.0         22.0            24.848485
                        5           NaN          NaN            24.848485
                        6          11.0         21.0            24.848485
```

7	13.0	21.0	24.848485
8	14.0	22.0	24.848485
9	16.0	23.0	24.848485
10	16.0	21.0	24.848485
11	24.0	22.0	24.848485
36	24.0	40.0	24.848485
37	NaN	NaN	24.848485
38	27.0	42.0	24.848485
39	18.0	34.0	24.848485
40	19.0	35.0	24.848485
41	20.0	36.0	24.848485
42	21.0	36.0	24.848485
43	22.0	37.0	24.848485

2 Filling the null values of Temparature and pressure using mean

[94]:
```
data_tidy.Temperature.fillna(data_tidy.Temperature.mean(),inplace=True)
```

[95]:
```
data_tidy.Pressure.fillna(data_tidy.Pressure.mean(),inplace=True)
```

[96]:
```
data_tidy.isna().sum()
```

[96]:
```
Parameter
Date                    0
Drug_Name               0
Time                    0
Pressure                0
Temperature             0
Average temparature     0
dtype: int64
```

3 Binning the data using cut function in pandas

[97]:
```
data_tidy.Temperature.min()
```

[97]:
```
8.0
```

[98]:
```
data_tidy.Temperature.max()
```

[98]:
```
58.0
```

[99]:
```
data_tidy.Pressure.min()
```

[99]:
```
3.0
```

[100]:
```
data_tidy.Pressure.max()
```

[100]:
```
30.0
```

[101]:
```
temp_points = [5,20,35,50,65]
temp_lables = ['low','medium','high','very_high']
data_tidy['Temparature category']= pd.cut(data_tidy.
 ↪Temperature,bins=temp_points,labels=temp_lables)
```

[102]:
```
data_tidy
```

[102]:
```
Parameter                          Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
…                                   …                       …       …
ketamine hydrochloride  103  17-10-2020   ketamine hydrochloride   5:30:00
                        104  17-10-2020   ketamine hydrochloride   6:30:00
                        105  17-10-2020   ketamine hydrochloride   7:30:00
                        106  17-10-2020   ketamine hydrochloride   8:30:00
                        107  17-10-2020   ketamine hydrochloride   9:30:00

Parameter                        Pressure  Temperature  Average temparature  \
Drug_Name
diltiazem hydrochloride 0            18.0         20.0            24.848485
                        1            19.0         20.0            24.848485
                        2            20.0         21.0            24.848485
                        3            12.0         23.0            24.848485
                        4            13.0         22.0            24.848485
…                                       …            …                    …
ketamine hydrochloride  103          11.0         17.0            17.709677
                        104          12.0         18.0            17.709677
                        105          12.0         19.0            17.709677
                        106          11.0         20.0            17.709677
                        107          12.0         21.0            17.709677

Parameter                        Temparature category
Drug_Name
diltiazem hydrochloride 0                         low
                        1                         low
                        2                      medium
                        3                      medium
                        4                      medium
…                                                 …
ketamine hydrochloride  103                       low
                        104                       low
```

```
                                105                  low
                                106                  low
                                107               medium

[108 rows x 7 columns]
```

[103]:
```
press_points = [5,15,16,25]
press_lables = ['Below_average','Average','Above_average']
data_tidy['Pressure category']= pd.cut(data_tidy.
 ↪Pressure,bins=press_points,labels=press_lables)
```

[104]:
```
data_tidy['Pressure category'].value_counts()
```

[104]:
```
Pressure category
Above_average    43
Below_average    40
Average           3
Name: count, dtype: int64
```

[105]:
```
data_tidy['Temparature category'].value_counts()
```

[105]:
```
Temparature category
low          45
medium       43
high         15
very_high     5
Name: count, dtype: int64
```

4 Retrieving the data contains certain string using Contains function

[106]:
```
data_tidy.loc[data_tidy.Drug_Name.str.contains('hydrochloride',case=False)] #␣
 ↪Case will igonre whether it is lower or upper case
```

[106]:
```
Parameter                          Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
…                                  …                      …       …
ketamine hydrochloride  103  17-10-2020   ketamine hydrochloride   5:30:00
                        104  17-10-2020   ketamine hydrochloride   6:30:00
                        105  17-10-2020   ketamine hydrochloride   7:30:00
                        106  17-10-2020   ketamine hydrochloride   8:30:00
                        107  17-10-2020   ketamine hydrochloride   9:30:00
```

```
Parameter                    Pressure  Temperature  Average temparature  \
Drug_Name
diltiazem hydrochloride 0        18.0         20.0            24.848485
                        1        19.0         20.0            24.848485
                        2        20.0         21.0            24.848485
                        3        12.0         23.0            24.848485
                        4        13.0         22.0            24.848485
…                                …            …               …
ketamine hydrochloride  103      11.0         17.0            17.709677
                        104      12.0         18.0            17.709677
                        105      12.0         19.0            17.709677
                        106      11.0         20.0            17.709677
                        107      12.0         21.0            17.709677

Parameter                   Temparature category Pressure category
Drug_Name
diltiazem hydrochloride 0                    low     Above_average
                        1                    low     Above_average
                        2                 medium     Above_average
                        3                 medium     Below_average
                        4                 medium     Below_average
…                                          …              …
ketamine hydrochloride  103                  low     Below_average
                        104                  low     Below_average
                        105                  low     Below_average
                        106                  low     Below_average
                        107               medium     Below_average

[72 rows x 8 columns]
```

5 Date and Time Functions in Pandas

[107]:
```
data_tidy[['Date','Time']]
```

[107]:
```
Parameter                        Date      Time
Drug_Name
diltiazem hydrochloride 0    15-10-2020  10:30:00
                        1    15-10-2020  11:30:00
                        2    15-10-2020  12:30:00
                        3    15-10-2020   1:30:00
                        4    15-10-2020   2:30:00
…                                …       …
ketamine hydrochloride  103  17-10-2020   5:30:00
                        104  17-10-2020   6:30:00
                        105  17-10-2020   7:30:00
```

```
                          106  17-10-2020   8:30:00
                          107  17-10-2020   9:30:00

[108 rows x 2 columns]
```

6 Getting year from Date column

[108]:
```
def get_year(x):
    return x[2]
data_tidy['Year'] = data_tidy['Date'].str.split('-').apply(get_year)
```

[109]:
```
data_tidy
```

[109]:
```
Parameter                          Date                 Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0     15-10-2020  diltiazem hydrochloride  10:30:00
                        1     15-10-2020  diltiazem hydrochloride  11:30:00
                        2     15-10-2020  diltiazem hydrochloride  12:30:00
                        3     15-10-2020  diltiazem hydrochloride   1:30:00
                        4     15-10-2020  diltiazem hydrochloride   2:30:00
…                                    …                        …       …
ketamine hydrochloride  103   17-10-2020   ketamine hydrochloride   5:30:00
                        104   17-10-2020   ketamine hydrochloride   6:30:00
                        105   17-10-2020   ketamine hydrochloride   7:30:00
                        106   17-10-2020   ketamine hydrochloride   8:30:00
                        107   17-10-2020   ketamine hydrochloride   9:30:00

Parameter                        Pressure  Temperature  Average temparature  \
Drug_Name
diltiazem hydrochloride 0            18.0         20.0            24.848485
                        1            19.0         20.0            24.848485
                        2            20.0         21.0            24.848485
                        3            12.0         23.0            24.848485
                        4            13.0         22.0            24.848485
…                                       …            …                    …
ketamine hydrochloride  103          11.0         17.0            17.709677
                        104          12.0         18.0            17.709677
                        105          12.0         19.0            17.709677
                        106          11.0         20.0            17.709677
                        107          12.0         21.0            17.709677

Parameter                    Temparature category Pressure category  Year
Drug_Name
diltiazem hydrochloride 0                     low     Above_average  2020
                        1                     low     Above_average  2020
                        2                  medium     Above_average  2020
```

```
                          3                medium     Below_average  2020
                          4                medium     Below_average  2020
…                                           …              …   …
ketamine hydrochloride    103                 low     Below_average  2020
                          104                 low     Below_average  2020
                          105                 low     Below_average  2020
                          106                 low     Below_average  2020
                          107              medium     Below_average  2020

[108 rows x 9 columns]
```

[110]:
```
data_tidy['Time stamp'] = data_tidy['Date'] + ' ' + data_tidy['Time']
```

[111]:
```
data_tidy
```

[111]:
```
Parameter                          Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
…                                  …                      …       …
ketamine hydrochloride  103  17-10-2020   ketamine hydrochloride   5:30:00
                        104  17-10-2020   ketamine hydrochloride   6:30:00
                        105  17-10-2020   ketamine hydrochloride   7:30:00
                        106  17-10-2020   ketamine hydrochloride   8:30:00
                        107  17-10-2020   ketamine hydrochloride   9:30:00

Parameter                        Pressure  Temperature  Average temparature  \
Drug_Name
diltiazem hydrochloride 0            18.0         20.0            24.848485
                        1            19.0         20.0            24.848485
                        2            20.0         21.0            24.848485
                        3            12.0         23.0            24.848485
                        4            13.0         22.0            24.848485
…                                      …            …                    …
ketamine hydrochloride  103          11.0         17.0            17.709677
                        104          12.0         18.0            17.709677
                        105          12.0         19.0            17.709677
                        106          11.0         20.0            17.709677
                        107          12.0         21.0            17.709677

Parameter                      Temparature category Pressure category  Year  \
Drug_Name
diltiazem hydrochloride 0                       low     Above_average  2020
                        1                       low     Above_average  2020
```

```
                          2                 medium     Above_average  2020
                          3                 medium     Below_average  2020
                          4                 medium     Below_average  2020
…                                            …              …    …
ketamine hydrochloride    103                  low     Below_average  2020
                          104                  low     Below_average  2020
                          105                  low     Below_average  2020
                          106                  low     Below_average  2020
                          107               medium     Below_average  2020

Parameter                                Time stamp
Drug_Name
diltiazem hydrochloride 0    15-10-2020 10:30:00
                        1    15-10-2020 11:30:00
                        2    15-10-2020 12:30:00
                        3     15-10-2020 1:30:00
                        4     15-10-2020 2:30:00
…                                      …
ketamine hydrochloride  103   17-10-2020 5:30:00
                        104   17-10-2020 6:30:00
                        105   17-10-2020 7:30:00
                        106   17-10-2020 8:30:00
                        107   17-10-2020 9:30:00

[108 rows x 10 columns]
```

7 Converting string into day format

[112]:
```
data_tidy['Time stamp'] = pd.to_datetime(data_tidy['Time stamp'])
```

```
C:\Users\saite\AppData\Local\Temp\ipykernel_2660\3180829823.py:1: UserWarning:
Parsing dates in %d-%m-%Y %H:%M:%S format when dayfirst=False (the default) was
specified. Pass `dayfirst=True` or specify a format to silence this warning.
  data_tidy['Time stamp'] = pd.to_datetime(data_tidy['Time stamp'])
```

[113]:
```
data_tidy
```

[113]:
```
Parameter                          Date                Drug_Name      Time  \
Drug_Name
diltiazem hydrochloride 0    15-10-2020  diltiazem hydrochloride  10:30:00
                        1    15-10-2020  diltiazem hydrochloride  11:30:00
                        2    15-10-2020  diltiazem hydrochloride  12:30:00
                        3    15-10-2020  diltiazem hydrochloride   1:30:00
                        4    15-10-2020  diltiazem hydrochloride   2:30:00
…                                  …                       …       …
ketamine hydrochloride  103  17-10-2020   ketamine hydrochloride   5:30:00
                        104  17-10-2020   ketamine hydrochloride   6:30:00
```

```
                        105  17-10-2020   ketamine hydrochloride   7:30:00
                        106  17-10-2020   ketamine hydrochloride   8:30:00
                        107  17-10-2020   ketamine hydrochloride   9:30:00

Parameter                     Pressure  Temperature  Average temparature  \
Drug_Name
diltiazem hydrochloride 0         18.0         20.0             24.848485
                        1         19.0         20.0             24.848485
                        2         20.0         21.0             24.848485
                        3         12.0         23.0             24.848485
                        4         13.0         22.0             24.848485
…                               …            …                 …
ketamine hydrochloride  103       11.0         17.0             17.709677
                        104       12.0         18.0             17.709677
                        105       12.0         19.0             17.709677
                        106       11.0         20.0             17.709677
                        107       12.0         21.0             17.709677

Parameter                    Temparature category Pressure category  Year  \
Drug_Name
diltiazem hydrochloride 0                     low     Above_average  2020
                        1                     low     Above_average  2020
                        2                  medium     Above_average  2020
                        3                  medium     Below_average  2020
                        4                  medium     Below_average  2020
…                                           …               …   …
ketamine hydrochloride  103                   low     Below_average  2020
                        104                   low     Below_average  2020
                        105                   low     Below_average  2020
                        106                   low     Below_average  2020
                        107                medium     Below_average  2020

Parameter                              Time stamp
Drug_Name
diltiazem hydrochloride 0   2020-10-15 10:30:00
                        1   2020-10-15 11:30:00
                        2   2020-10-15 12:30:00
                        3   2020-10-15 01:30:00
                        4   2020-10-15 02:30:00
…                                   …
ketamine hydrochloride  103 2020-10-17 05:30:00
                        104 2020-10-17 06:30:00
                        105 2020-10-17 07:30:00
                        106 2020-10-17 08:30:00
                        107 2020-10-17 09:30:00

[108 rows x 10 columns]
```

[114]:
```
type(data_tidy['Time stamp'][1])
```

```
C:\Users\saite\AppData\Local\Temp\ipykernel_2660\352291693.py:1: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  type(data_tidy['Time stamp'][1])
```

[114]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

8 Accessing the Day, Month, Year from a time stamp

[115]:
```
Date = data_tidy['Time stamp'][0]
```

```
C:\Users\saite\AppData\Local\Temp\ipykernel_2660\2824729332.py:1: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  Date = data_tidy['Time stamp'][0]
```

[116]:
```
Date.year
```

[116]:
```
2020
```

[117]:
```
Date.day
```

[117]:
```
15
```

[118]:
```
Date.month
```

[118]:
```
10
```

[119]:
```
Date.month_name()
```

[119]:
```
'October'
```

[120]:
```
Date.day_name()
```

[120]:
```
'Thursday'
```

[121]:
```
data_tidy['Time stamp'].dt.month_name()
```

[121]:
```
Drug_Name
diltiazem hydrochloride  0      October
                         1      October
                         2      October
                         3      October
                         4      October
```

```
                                ...
ketamine hydrochloride   103    October
                         104    October
                         105    October
                         106    October
                         107    October
Name: Time stamp, Length: 108, dtype: object
```

9 If you want to get the date in specified format

[122]:
```
Date
```

[122]:
```
Timestamp('2020-10-15 10:30:00')
```

Printing only year

[125]:
```
Date.strftime('%Y') #Or we can use lower y which give year in this format YY
```

[125]:
```
'2020'
```

[127]:
```
Date.strftime('%M')
```

[127]:
```
'30'
```

[128]:
```
Date.strftime('%h')
```

[128]:
```
'Oct'
```

[129]:
```
Date.strftime("%d-%m-%y")
```

[129]:
```
'15-10-20'
```

10 Saving the CSV File

[130]:
```
data_tidy.to_csv("Pfizer_tidy.csv")
```

[ ]:

Day_33_281123

January 23, 2024

1 Pandas

[97]:
```
import pandas as pd
```

[98]:
```
!gdown 1M82e0zOMJgV1ISQe3auBFnglYvMyZyBh
```

```
Downloading…
From: https://drive.google.com/uc?id=1M82e0zOMJgV1ISQe3auBFnglYvMyZyBh
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\mckinsey (1).csv

  0%|          | 0.00/83.8k [00:00<?, ?B/s]
100%|##########| 83.8k/83.8k [00:00<00:00, 41.6MB/s]
```

- Heterogenous data
- Visualization
- Manipulate the dataframe
- Complex analysis
- Easy

2 Importing data

[99]:
```
df = pd.read_csv("mckinsey (1).csv")
df
```

[99]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…
1699	Zimbabwe	1987	9216418	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298

[1704 rows x 6 columns]

[100]:
```
type(df)
```

[100]:
```
pandas.core.frame.DataFrame
```

[101]:
```
type(df[['country']])
```

[101]:
```
pandas.core.frame.DataFrame
```

[102]:
```
type(df['country'])
```

[102]:
```
pandas.core.series.Series
```

[103]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   country     1704 non-null   object
 1   year        1704 non-null   int64
 2   population  1704 non-null   int64
 3   continent   1704 non-null   object
 4   life_exp    1704 non-null   float64
 5   gdp_cap     1704 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
```

Memory	
1 KB	1024B
1 MB	1024 KB
1 GB	1024 MB
1 TB	1024 GB

3 Getting top n records

[104]:
```
df.head(4)
```

[104]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138

4 Getting bottom n records

[105]:
```
df.tail(4)
```

[105]:

	country	year	population	continent	life_exp	gdp_cap
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298

[106]:
```
df.shape
```

[106]: (1704, 6)

5 Creating a dataframe

Using dictionaries

[107]:
```
new_df = pd.DataFrame(
    {
     "Name":["Sai","Sharan","Bunny","Shiva","Sagar"],
     "Age":[22,27,20,29,25],
     "City":["HYD","BEN","KNL","BEN","BEN"],
     "Phno":[72880,94954,98981,88771,90890]
    }
)
```

[108]:
```
new_df
```

[108]:

	Name	Age	City	Phno
0	Sai	22	HYD	72880
1	Sharan	27	BEN	94954
2	Bunny	20	KNL	98981
3	Shiva	29	BEN	88771
4	Sagar	25	BEN	90890

[109]:
```
new_df.shape
```

[109]: (5, 4)

[110]:
```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    5 non-null      object
 1   Age     5 non-null      int64
```

```
 2   City    5 non-null      object
 3   Phno    5 non-null      int64
dtypes: int64(2), object(2)
memory usage: 292.0+ bytes
```

5.0.1 Using Lists

[111]:
```
new_df2 = pd.DataFrame([['Sai',22,'HYD',9900],
                        ['Ria',23,'MAR',7788],
                        ['Mahesh',45,'HYD',4005]],
                       columns=['Name','Age','City','Phno'])
```

[112]:
```
new_df2
```

[112]:

	Name	Age	City	Phno
0	Sai	22	HYD	9900
1	Ria	23	MAR	7788
2	Mahesh	45	HYD	4005

[113]:
```
new_df2[['City','Phno','Name','Age']]
```

[113]:

	City	Phno	Name	Age
0	HYD	9900	Sai	22
1	MAR	7788	Ria	23
2	HYD	4005	Mahesh	45

5.1 Unique values

[114]:
```
df.country.unique()
```

[114]:
```
array(['Afghanistan', 'Albania', 'Algeria', 'Angola', 'Argentina',
       'Australia', 'Austria', 'Bahrain', 'Bangladesh', 'Belgium',
       'Benin', 'Bolivia', 'Bosnia and Herzegovina', 'Botswana', 'Brazil',
       'Bulgaria', 'Burkina Faso', 'Burundi', 'Cambodia', 'Cameroon',
       'Canada', 'Central African Republic', 'Chad', 'Chile', 'China',
       'Colombia', 'Comoros', 'Congo, Dem. Rep.', 'Congo, Rep.',
       'Costa Rica', "Cote d'Ivoire", 'Croatia', 'Cuba', 'Czech Republic',
       'Denmark', 'Djibouti', 'Dominican Republic', 'Ecuador', 'Egypt',
       'El Salvador', 'Equatorial Guinea', 'Eritrea', 'Ethiopia',
       'Finland', 'France', 'Gabon', 'Gambia', 'Germany', 'Ghana',
       'Greece', 'Guatemala', 'Guinea', 'Guinea-Bissau', 'Haiti',
       'Honduras', 'Hong Kong, China', 'Hungary', 'Iceland', 'India',
       'Indonesia', 'Iran', 'Iraq', 'Ireland', 'Israel', 'Italy',
       'Jamaica', 'Japan', 'Jordan', 'Kenya', 'Korea, Dem. Rep.',
       'Korea, Rep.', 'Kuwait', 'Lebanon', 'Lesotho', 'Liberia', 'Libya',
       'Madagascar', 'Malawi', 'Malaysia', 'Mali', 'Mauritania',
       'Mauritius', 'Mexico', 'Mongolia', 'Montenegro', 'Morocco',
       'Mozambique', 'Myanmar', 'Namibia', 'Nepal', 'Netherlands',
```

```
'New Zealand', 'Nicaragua', 'Niger', 'Nigeria', 'Norway', 'Oman',
'Pakistan', 'Panama', 'Paraguay', 'Peru', 'Philippines', 'Poland',
'Portugal', 'Puerto Rico', 'Reunion', 'Romania', 'Rwanda',
'Sao Tome and Principe', 'Saudi Arabia', 'Senegal', 'Serbia',
'Sierra Leone', 'Singapore', 'Slovak Republic', 'Slovenia',
'Somalia', 'South Africa', 'Spain', 'Sri Lanka', 'Sudan',
'Swaziland', 'Sweden', 'Switzerland', 'Syria', 'Taiwan',
'Tanzania', 'Thailand', 'Togo', 'Trinidad and Tobago', 'Tunisia',
'Turkey', 'Uganda', 'United Kingdom', 'United States', 'Uruguay',
'Venezuela', 'Vietnam', 'West Bank and Gaza', 'Yemen, Rep.',
'Zambia', 'Zimbabwe'], dtype=object)
```

5.2 Get the count

[115]:
```
df['continent'].value_counts()
```

[115]:
```
continent
Africa      624
Asia        396
Europe      360
Americas    300
Oceania      24
Name: count, dtype: int64
```

5.3 Rename the column

[116]:
```
df.rename(
    {
        'country':'COUNTRY',
        'population':'POPULATION'
    },axis=1,inplace=True  #Inplace will save the changes to original dataframe
)
```

[117]:
```
df
```

[117]:

	COUNTRY	year	POPULATION	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…
1699	Zimbabwe	1987	9216418	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298

```
[1704 rows x 6 columns]
```

[118]:
```
df_1 = df.T
```

[119]:
```
df_1.head()
```

[119]:
```
                   0            1            2            3            4     \
COUNTRY     Afghanistan  Afghanistan  Afghanistan  Afghanistan  Afghanistan
year               1952         1957         1962         1967         1972
POPULATION      8425333      9240934     10267083     11537966     13079460
continent          Asia         Asia         Asia         Asia         Asia
life_exp         28.801       30.332       31.997        34.02       36.088

                   5            6            7            8            9     \
COUNTRY     Afghanistan  Afghanistan  Afghanistan  Afghanistan  Afghanistan
year               1977         1982         1987         1992         1997
POPULATION     14880372     12881816     13867957     16317921     22227415
continent          Asia         Asia         Asia         Asia         Asia
life_exp         38.438       39.854       40.822       41.674       41.763

               …      1694      1695      1696      1697      1698      1699  \
COUNTRY        …  Zimbabwe  Zimbabwe  Zimbabwe  Zimbabwe  Zimbabwe  Zimbabwe
year           …      1962      1967      1972      1977      1982      1987
POPULATION     …   4277736   4995432   5861135   6642107   7636524   9216418
continent      …    Africa    Africa    Africa    Africa    Africa    Africa
life_exp       …    52.358    53.995    55.635    57.674    60.363    62.351

                  1700      1701      1702      1703
COUNTRY     Zimbabwe  Zimbabwe  Zimbabwe  Zimbabwe
year            1992      1997      2002      2007
POPULATION  10704340  11404948  11926563  12311143
continent     Africa    Africa    Africa    Africa
life_exp      60.377    46.809    39.989    43.487

[5 rows x 1704 columns]
```

[120]:
```
df_1.rename(
    {
        'country':'COUNTRY',
        'population':'POPULATION'
    }
)
```

[120]:
```
                   0            1            2            3            4     \
COUNTRY     Afghanistan  Afghanistan  Afghanistan  Afghanistan  Afghanistan
year               1952         1957         1962         1967         1972
```

```
POPULATION      8425333      9240934     10267083     11537966     13079460
continent          Asia         Asia         Asia         Asia         Asia
life_exp         28.801       30.332       31.997       34.02       36.088
gdp_cap      779.445314    820.85303    853.10071   836.197138   739.981106

                     5            6            7            8            9     \
COUNTRY      Afghanistan  Afghanistan  Afghanistan  Afghanistan  Afghanistan
year                1977         1982         1987         1992         1997
POPULATION      14880372     12881816     13867957     16317921     22227415
continent          Asia         Asia         Asia         Asia         Asia
life_exp         38.438       39.854       40.822       41.674       41.763
gdp_cap       786.11336   978.011439   852.395945   649.341395   635.341351

               …         1694         1695         1696         1697         1698  \
COUNTRY        …     Zimbabwe     Zimbabwe     Zimbabwe     Zimbabwe     Zimbabwe
year           …         1962         1967         1972         1977         1982
POPULATION     …      4277736      4995432      5861135      6642107      7636524
continent      …       Africa       Africa       Africa       Africa       Africa
life_exp       …       52.358       53.995       55.635       57.674       60.363
gdp_cap        …   527.272182   569.795071   799.362176   685.587682   788.855041

                    1699         1700         1701         1702         1703
COUNTRY         Zimbabwe     Zimbabwe     Zimbabwe     Zimbabwe     Zimbabwe
year                1987         1992         1997         2002         2007
POPULATION      9216418     10704340     11404948     11926563     12311143
continent        Africa       Africa       Africa       Africa       Africa
life_exp         62.351       60.377       46.809       39.989       43.487
gdp_cap      706.157306   693.420786   792.44996   672.038623   469.709298

[6 rows x 1704 columns]
```

5.4 Deleting a column‘

[121]:
```
df.drop('continent',axis=1,inplace=True)
```

[122]:
```
df
```

[122]:
```
           COUNTRY  year  POPULATION  life_exp     gdp_cap
0      Afghanistan  1952     8425333    28.801  779.445314
1      Afghanistan  1957     9240934    30.332  820.853030
2      Afghanistan  1962    10267083    31.997  853.100710
3      Afghanistan  1967    11537966    34.020  836.197138
4      Afghanistan  1972    13079460    36.088  739.981106
…              …     …           …         …           …
1699      Zimbabwe  1987     9216418    62.351  706.157306
1700      Zimbabwe  1992    10704340    60.377  693.420786
1701      Zimbabwe  1997    11404948    46.809  792.449960
```

```
1702     Zimbabwe  2002    11926563    39.989  672.038623
1703     Zimbabwe  2007    12311143    43.487  469.709298

[1704 rows x 5 columns]
```

[123]:
```
df.drop(columns=['year','life_exp'],inplace=True)
```

[124]:
```
df
```

[124]:
```
           COUNTRY  POPULATION     gdp_cap
0      Afghanistan     8425333  779.445314
1      Afghanistan     9240934  820.853030
2      Afghanistan    10267083  853.100710
3      Afghanistan    11537966  836.197138
4      Afghanistan    13079460  739.981106
…              …           …          …
1699     Zimbabwe     9216418  706.157306
1700     Zimbabwe    10704340  693.420786
1701     Zimbabwe    11404948  792.449960
1702     Zimbabwe    11926563  672.038623
1703     Zimbabwe    12311143  469.709298

[1704 rows x 3 columns]
```

5.5 Adding a new column into data frame

[125]:
```
df['gdp_cap_per'] = df['gdp_cap']/100
```

[126]:
```
df
```

[126]:
```
           COUNTRY  POPULATION     gdp_cap  gdp_cap_per
0      Afghanistan     8425333  779.445314     7.794453
1      Afghanistan     9240934  820.853030     8.208530
2      Afghanistan    10267083  853.100710     8.531007
3      Afghanistan    11537966  836.197138     8.361971
4      Afghanistan    13079460  739.981106     7.399811
…              …           …          …             …
1699     Zimbabwe     9216418  706.157306     7.061573
1700     Zimbabwe    10704340  693.420786     6.934208
1701     Zimbabwe    11404948  792.449960     7.924500
1702     Zimbabwe    11926563  672.038623     6.720386
1703     Zimbabwe    12311143  469.709298     4.697093

[1704 rows x 4 columns]
```

Day_27_201123

January 23, 2024

1 NumPy

Numerical Python

- It is used for complex calculation
- It is faster than array
- We can perform calculation on 2D, 3D, 4D ……, nDm

Initialize the NumPy and Importing NumPy

[50]:
```
# Installing NumPy in system
!pip install numpy
import numpy as np # Importing Numpy
```

```
Requirement already satisfied: numpy in c:\data\env\lib\site-packages (1.26.1)
```

- How likely do you suggest our product to your friends and family? What would be your preference in this scenario? It ranges from one to nine. One being the least likely and nine being highly likely. Which option do you believe is not at all likely? -

0 1 2 3 4 5	6 7	8 9
Detractors	Neutral	Promoter

- NPS Net promoter score = % of Promoter - % of Detractors
- NPS will be in the range of -100 to 100

Difference of array and list

Array	List
Homogenous [Same type of data]	Heterogenous [Different type of data]
Fast to Generate and Extract Data	Slower Comparitively
Complex Calculation	Simple Calculation

Accessing elements of array using memory address

- a = a + i * memory
- a = 100 + 2 * 8 [We are going to access the index of 2 element]
- a = 116 [Memory location is 116 were we can find a]

When it comes to list it will get the address and using that address it will again search of element store

To check the time complexity Python has a magic function %timeit

1.0.1 Time taken to execute the arrays and list

[4]:
```
list = range(1000)
%timeit [i**2 for i in list]
```

```
194 µs ± 2.58 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[5]:
```
a = np.array(range(1000))
%timeit a**2
```

```
3.41 µs ± 131 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

1.0.2 Creating an NumPy array

[12]:
```
a = np.array([[1,2,3,4,5],[3,5,6,4,7]])
```

1.0.3 Dimension of the Array

[13]:
```
a.ndim
```

[13]: 2

1.0.4 Shape of the Array

[14]:
```
a.shape
```

[14]: (2, 5)

1.0.5 Total values in the Array

[17]:
```
a.size
```

[17]: 10

[18]:
```
len(a)
```

[18]: 2

1.0.6 Masking

[19]:
```
b = np.array([1,2,3,4,5,6,7,8])
b[b>3]
```

[19]: array([4, 5, 6, 7, 8])

1.0.7 Exercise 1: Create a cubes in range 1000 with both array and list get the time difference

[20]:
```
#List
%timeit [x**3 for x in range(1000)]

#Array
c = np.array(1000)
%timeit c**3

d2 = np.array([[1,2,9,8],[5,6,7,4]])
print(d2.ndim)
print(d2.shape)
print(len(d2))
```

```
263 µs ± 16.6 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
2.37 µs ± 113 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
2
(2, 4)
2
```

1.0.8 If we want to get the step size in float numpy comes handy

[26]:
```
[x for x in range(0,100,0.5)] # It will throw an error for list
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 [x for x in range(0,100,0.5)] # It will throw an error for list

TypeError: 'float' object cannot be interpreted as an integer
```

[33]:
```
x = np.arange(0,100,0.5)
x
```

[33]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. , 10.5,
       11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. , 15.5, 16. ,
       16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5, 20. , 20.5, 21. , 21.5,
       22. , 22.5, 23. , 23.5, 24. , 24.5, 25. , 25.5, 26. , 26.5, 27. ,
       27.5, 28. , 28.5, 29. , 29.5, 30. , 30.5, 31. , 31.5, 32. , 32.5,
       33. , 33.5, 34. , 34.5, 35. , 35.5, 36. , 36.5, 37. , 37.5, 38. ,
       38.5, 39. , 39.5, 40. , 40.5, 41. , 41.5, 42. , 42.5, 43. , 43.5,
       44. , 44.5, 45. , 45.5, 46. , 46.5, 47. , 47.5, 48. , 48.5, 49. ,
       49.5, 50. , 50.5, 51. , 51.5, 52. , 52.5, 53. , 53.5, 54. , 54.5,
       55. , 55.5, 56. , 56.5, 57. , 57.5, 58. , 58.5, 59. , 59.5, 60. ,
       60.5, 61. , 61.5, 62. , 62.5, 63. , 63.5, 64. , 64.5, 65. , 65.5,
```

```
66. , 66.5, 67. , 67.5, 68. , 68.5, 69. , 69.5, 70. , 70.5, 71. ,
71.5, 72. , 72.5, 73. , 73.5, 74. , 74.5, 75. , 75.5, 76. , 76.5,
77. , 77.5, 78. , 78.5, 79. , 79.5, 80. , 80.5, 81. , 81.5, 82. ,
82.5, 83. , 83.5, 84. , 84.5, 85. , 85.5, 86. , 86.5, 87. , 87.5,
88. , 88.5, 89. , 89.5, 90. , 90.5, 91. , 91.5, 92. , 92.5, 93. ,
93.5, 94. , 94.5, 95. , 95.5, 96. , 96.5, 97. , 97.5, 98. , 98.5,
99. , 99.5])
```

We have to use '&' when we are checking two different condition to single element, 'and' can be used for comparaing two elements

[48]:
```
x = np.arange(0,100,0.5)
x[(x%2 == 0) & (x%5 ==0)]
```

[48]:
```
array([ 0., 10., 20., 30., 40., 50., 60., 70., 80., 90.])
```

Day_34_291123

January 23, 2024

[110]:
```
import pandas as pd
```

[111]:
```
import numpy as np
```

1 There are two types of indexs

- Explicit index - User can see
- Implicit index - Users cannot see but the computer will assign indices

[112]:
```
temp = pd.DataFrame([['a','b',1,3.0,4]],columns = ['a','b','c','d','e'])
temp
```

[112]:

	a	b	c	d	e
0	a	b	1	3.0	4

[113]:
```
df = pd.read_csv("mckinsey (1).csv")
df.head(4)
```

[113]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138

2 Index values of a dataframe

[114]:
```
df.index.values
```

[114]:
```
array([   0,    1,    2, …, 1701, 1702, 1703], dtype=int64)
```

3 Changing the index values of a dataframe

[115]:
```
df.index = np.arange(1,1705,dtype='int')
```

[116]:
```
df.index.values
```

[116]:
```
array([   1,    2,    3, …, 1702, 1703, 1704])
```

[117]:
```
df
```

[117]:
```
          country  year  population continent  life_exp     gdp_cap
1     Afghanistan  1952     8425333      Asia    28.801  779.445314
2     Afghanistan  1957     9240934      Asia    30.332  820.853030
3     Afghanistan  1962    10267083      Asia    31.997  853.100710
4     Afghanistan  1967    11537966      Asia    34.020  836.197138
5     Afghanistan  1972    13079460      Asia    36.088  739.981106
…             …     …           …         …         …           …
1700     Zimbabwe  1987     9216418    Africa    62.351  706.157306
1701     Zimbabwe  1992    10704340    Africa    60.377  693.420786
1702     Zimbabwe  1997    11404948    Africa    46.809  792.449960
1703     Zimbabwe  2002    11926563    Africa    39.989  672.038623
1704     Zimbabwe  2007    12311143    Africa    43.487  469.709298

[1704 rows x 6 columns]
```

[118]:
```
df.index[1]
```

[118]:
```
2
```

4 loc (location -> Explicit) and iloc (integer location -> Implicit)

[119]:
```
df.iloc[1]
```

[119]:
```
country       Afghanistan
year                 1957
population        9240934
continent            Asia
life_exp           30.332
gdp_cap         820.85303
Name: 2, dtype: object
```

[120]:
```
df.loc[5]
```

[120]:
```
country       Afghanistan
year                 1972
population       13079460
continent            Asia
life_exp           36.088
gdp_cap        739.981106
Name: 5, dtype: object
```

[121]:
```
df.loc[889]
```

[121]:
```
country          Liberia
year                1952
population         863308
continent          Africa
life_exp            38.48
gdp_cap        575.572996
Name: 889, dtype: object
```

[122]:
```
df.iloc[888]
```

[122]:
```
country          Liberia
year                1952
population         863308
continent          Africa
life_exp            38.48
gdp_cap        575.572996
Name: 889, dtype: object
```

[123]:
```
df.iloc[[2,3,1,7]]
```

[123]:

	country	year	population	continent	life_exp	gdp_cap
3	Afghanistan	1962	10267083	Asia	31.997	853.100710
4	Afghanistan	1967	11537966	Asia	34.020	836.197138
2	Afghanistan	1957	9240934	Asia	30.332	820.853030
8	Afghanistan	1987	13867957	Asia	40.822	852.395945

[124]:
```
df.loc[[10,18,1056]]
```

[124]:

	country	year	population	continent	life_exp	gdp_cap
10	Afghanistan	1997	22227415	Asia	41.763	635.341351
18	Albania	1977	2509048	Europe	68.930	3533.003910
1056	Myanmar	2007	47761980	Asia	62.069	944.000000

[125]:
```
df.iloc[-1]
```

[125]:
```
country         Zimbabwe
year                2007
population      12311143
continent         Africa
life_exp           43.487
gdp_cap       469.709298
Name: 1704, dtype: object
```

[126]:
```
df.iloc[df.iloc[0:10:2].index]
```

[126]:

	country	year	population	continent	life_exp	gdp_cap
2	Afghanistan	1957	9240934	Asia	30.332	820.853030
4	Afghanistan	1967	11537966	Asia	34.020	836.197138

```
6   Afghanistan  1977    14880372      Asia    38.438  786.113360
8   Afghanistan  1987    13867957      Asia    40.822  852.395945
10  Afghanistan  1997    22227415      Asia    41.763  635.341351
```

5 Change the index of a dataframe

[127]:
```
temp = df.set_index('country')
```

[128]:
```
temp
```

[128]:
```
             year  population continent  life_exp     gdp_cap
country
Afghanistan  1952     8425333      Asia    28.801  779.445314
Afghanistan  1957     9240934      Asia    30.332  820.853030
Afghanistan  1962    10267083      Asia    31.997  853.100710
Afghanistan  1967    11537966      Asia    34.020  836.197138
Afghanistan  1972    13079460      Asia    36.088  739.981106
…            …          …       …        …          …
Zimbabwe     1987     9216418    Africa    62.351  706.157306
Zimbabwe     1992    10704340    Africa    60.377  693.420786
Zimbabwe     1997    11404948    Africa    46.809  792.449960
Zimbabwe     2002    11926563    Africa    39.989  672.038623
Zimbabwe     2007    12311143    Africa    43.487  469.709298

[1704 rows x 5 columns]
```

[129]:
```
temp2 = temp.set_index('continent')
```

[130]:
```
temp2.loc['Asia']
```

[130]:
```
            year  population  life_exp      gdp_cap
continent
Asia        1952     8425333    28.801   779.445314
Asia        1957     9240934    30.332   820.853030
Asia        1962    10267083    31.997   853.100710
Asia        1967    11537966    34.020   836.197138
Asia        1972    13079460    36.088   739.981106
…           …          …        …          …
Asia        1987    11219340    52.922  1971.741538
Asia        1992    13367997    55.599  1879.496673
Asia        1997    15826497    58.020  2117.484526
Asia        2002    18701257    60.308  2234.820827
Asia        2007    22211743    62.698  2280.769906

[396 rows x 4 columns]
```

6 Reset index to original index in a dataframe

[131]:
```
temp.reset_index(inplace=True)
```

[132]:
```
temp
```

[132]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…
1699	Zimbabwe	1987	9216418	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298

[1704 rows x 6 columns]

7 Add a new row

[227]:
```
df = pd.read_csv("mckinsey (1).csv")
```

[228]:
```
df.head()
```

[228]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106

Creating a dict of new values

[229]:
```
new_row = {'country':'India','year':1988,'population':94343233,'continent':
 ↪'Asia','life_exp':89.99,'gdp_cap':679.90}
```

[230]:
```
new_row
```

[230]:
```
{'country': 'India',
 'year': 1988,
 'population': 94343233,
 'continent': 'Asia',
 'life_exp': 89.99,
 'gdp_cap': 679.9}
```

[189]:
```
df.loc[1704] = new_row
```

[190]:
```
df.tail(3)
```

[190]:

	country	year	population	continent	life_exp	gdp_cap
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298
1704	India	1988	94343233	Asia	89.990	679.900000

[224]:
```
df
```

[224]:

	country	year	population	continent	life_exp	gdp_cap
0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…
1699	Zimbabwe	1987	9216418	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143	Africa	43.487	469.709298

[1704 rows x 6 columns]

8 Update the single cell value

data_frame.at[row_number, column_name] = value_to_update

[ ]:
```
df.at[1703,'population'] = 29601212.324530516
```

[231]:
```
n = int(input('Enter how many extra row you want to add:'))
for i in range(0,n):
    df.loc[len(df.index)+i] = new_row
    print(len(df.index)+i)
df.reset_index(inplace=True)
```

```
Enter how many extra row you want to add: 10

1705
1707
1709
1711
1713
1715
1717
1719
```

1721
1723

9 Dropping the rows

[233]:
```
df.drop(1704,axis=0)
```

[233]:

	index	country	year	population	continent	life_exp	gdp_cap
0	0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…	…
1709	1714	India	1988	94343233	Asia	89.990	679.900000
1710	1716	India	1988	94343233	Asia	89.990	679.900000
1711	1718	India	1988	94343233	Asia	89.990	679.900000
1712	1720	India	1988	94343233	Asia	89.990	679.900000
1713	1722	India	1988	94343233	Asia	89.990	679.900000

[1713 rows x 7 columns]

[237]:
```
df.drop([1709,1710,1711,1712,1713],axis=0)
```

[237]:

	index	country	year	population	continent	life_exp	gdp_cap
0	0	Afghanistan	1952	8425333	Asia	28.801	779.445314
1	1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	4	Afghanistan	1972	13079460	Asia	36.088	739.981106
…	…	…	…	…	…	…	…
1704	1704	India	1988	94343233	Asia	89.990	679.900000
1705	1706	India	1988	94343233	Asia	89.990	679.900000
1706	1708	India	1988	94343233	Asia	89.990	679.900000
1707	1710	India	1988	94343233	Asia	89.990	679.900000
1708	1712	India	1988	94343233	Asia	89.990	679.900000

[1709 rows x 7 columns]

[234]:
```
df.iloc[1:5]
```

[234]:

	index	country	year	population	continent	life_exp	gdp_cap
1	1	Afghanistan	1957	9240934	Asia	30.332	820.853030
2	2	Afghanistan	1962	10267083	Asia	31.997	853.100710
3	3	Afghanistan	1967	11537966	Asia	34.020	836.197138
4	4	Afghanistan	1972	13079460	Asia	36.088	739.981106

10 In loc the end range becomes inclusive

[235]:
```
df.loc[10:20]
```

[235]:

	index	country	year	population	continent	life_exp	gdp_cap
10	10	Afghanistan	2002	25268405	Asia	42.129	726.734055
11	11	Afghanistan	2007	31889923	Asia	43.828	974.580338
12	12	Albania	1952	1282697	Europe	55.230	1601.056136
13	13	Albania	1957	1476505	Europe	59.280	1942.284244
14	14	Albania	1962	1728137	Europe	64.820	2312.888958
15	15	Albania	1967	1984060	Europe	66.220	2760.196931
16	16	Albania	1972	2263554	Europe	67.690	3313.422188
17	17	Albania	1977	2509048	Europe	68.930	3533.003910
18	18	Albania	1982	2780097	Europe	70.420	3630.880722
19	19	Albania	1987	3075321	Europe	72.000	3738.932735
20	20	Albania	1992	3326498	Europe	71.581	2497.437901

[244]:
```
df2 = df.tail(10)
```

[251]:
```
df2
```

[251]:

	country	year	population	continent	life_exp	gdp_cap
1704	India	1988	94343233	Asia	89.99	679.9
1705	India	1988	94343233	Asia	89.99	679.9
1706	India	1988	94343233	Asia	89.99	679.9
1707	India	1988	94343233	Asia	89.99	679.9
1708	India	1988	94343233	Asia	89.99	679.9
1709	India	1988	94343233	Asia	89.99	679.9
1710	India	1988	94343233	Asia	89.99	679.9
1711	India	1988	94343233	Asia	89.99	679.9
1712	India	1988	94343233	Asia	89.99	679.9
1713	India	1988	94343233	Asia	89.99	679.9

11 To check how many duplicates are there in data

[252]:
```
df2.duplicated()
```

[252]:
```
1704    False
1705     True
1706     True
1707     True
1708     True
1709     True
1710     True
1711     True
1712     True
1713     True
```

```
dtype: bool
```

[ ]:

Day_38_041223

January 23, 2024

[150]:
```
import numpy as np
import pandas as pd
movies = pd.read_csv("movies.csv",index_col=0)
directors = pd.read_csv("directors.csv",index_col=0)
data = pd.merge(movies,directors,left_on="director_id",right_on='id',how='left')
data.drop('id_y',axis=1,inplace=True)
data.rename({"id_x":"movies_id"},axis=1,inplace=True)
data
```

[150]:

	movies_id	budget	popularity	revenue	\
0	43597	237000000	150	2787965087	
1	43598	300000000	139	961000000	
2	43599	245000000	107	880674609	
3	43600	250000000	112	1084939099	
4	43602	258000000	115	890871626	
…	…	…	…	…	
1460	48363	0	3	321952	
1461	48370	27000	19	3151130	
1462	48375	0	7	0	
1463	48376	0	3	0	
1464	48395	220000	14	2040920	

	title	vote_average	vote_count	\
0	Avatar	7.2	11800	
1	Pirates of the Caribbean: At World's End	6.9	4500	
2	Spectre	6.3	4466	
3	The Dark Knight Rises	7.6	9106	
4	Spider-Man 3	5.9	3576	
…	…	…	…	
1460	The Last Waltz	7.9	64	
1461	Clerks	7.4	755	
1462	Rampage	6.0	131	
1463	Slacker	6.4	77	
1464	El Mariachi	6.6	238	

	director_id	year	month	day	director_name	gender
0	4762	2009	Dec	Thursday	James Cameron	Male
1	4763	2007	May	Saturday	Gore Verbinski	Male

```
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …     …     …       …                 …     …
1460         4809  1978   May    Monday    Martin Scorsese   Male
1461         5369  1994   Sep   Tuesday        Kevin Smith   Male
1462         5148  2009   Aug    Friday           Uwe Boll   Male
1463         5535  1990   Jul    Friday  Richard Linklater   Male
1464         5097  1992   Sep    Friday   Robert Rodriguez    NaN

[1465 rows x 13 columns]
```

1 How the multi indexing works

[151]:
```
data_agg = data.groupby('director_name')[['title','year']].aggregate({'title':
 ↪'count','year':['min','max']})
data_agg
```

[151]:
```
                           title  year
                           count   min   max
director_name
Adam McKay                     6  2004  2015
Adam Shankman                  8  2001  2012
Alejandro González Iñárritu    6  2000  2015
Alex Proyas                    5  1994  2016
Alexander Payne                5  1999  2013
…                            …    …     …
Wes Craven                    10  1984  2011
Wolfgang Petersen              7  1981  2006
Woody Allen                   18  1977  2013
Zack Snyder                    7  2004  2016
Zhang Yimou                    6  2002  2014

[199 rows x 3 columns]
```

[152]:
```
data.columns
```

[152]:
```
Index(['movies_id', 'budget', 'popularity', 'revenue', 'title', 'vote_average',
       'vote_count', 'director_id', 'year', 'month', 'day', 'director_name',
       'gender'],
      dtype='object')
```

[153]:
```
data_agg.columns
```

[153]:
```
MultiIndex([('title', 'count'),
            ( 'year',   'min'),
            ( 'year',   'max')],
```

)

2 Changing the Multi index to Single index

[154]:
```
data_agg.columns = ['_'.join(tuple) for tuple in data_agg.columns]
```

[155]:
```
data_agg
```

[155]:

director_name	title_count	year_min	year_max
Adam McKay	6	2004	2015
Adam Shankman	8	2001	2012
Alejandro González Iñárritu	6	2000	2015
Alex Proyas	5	1994	2016
Alexander Payne	5	1999	2013
…	…	…	…
Wes Craven	10	1984	2011
Wolfgang Petersen	7	1981	2006
Woody Allen	18	1977	2013
Zack Snyder	7	2004	2016
Zhang Yimou	6	2002	2014

[199 rows x 3 columns]

3 Cleaning the Data using Pandas

- When we have more columns and less rows it is called Fat Data
- When we have more rows and less columns it is called Thin Data

[156]:
```
!gdown 173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ
```

```
Downloading…
From: https://drive.google.com/uc?id=173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ
To: C:\Data\Data_science\Data Science RIA\3 Python\Pandas\Codes\Pfizer_1.csv

  0%|          | 0.00/1.51k [00:00<?, ?B/s]
100%|##########| 1.51k/1.51k [00:00<?, ?B/s]
```

[157]:
```
data = pd.read_csv("Pfizer_1.csv")
```

[158]:
```
data.columns
```

[158]:
```
Index(['Date', 'Drug_Name', 'Parameter', '1:30:00', '2:30:00', '3:30:00',
       '4:30:00', '5:30:00', '6:30:00', '7:30:00', '8:30:00', '9:30:00',
       '10:30:00', '11:30:00', '12:30:00'],
      dtype='object')
```

4 Example of Fat data

[159]:
```
data.head()
```

[159]:
```
         Date                Drug_Name    Parameter  1:30:00  2:30:00  \
0  15-10-2020  diltiazem hydrochloride  Temperature     23.0     22.0
1  15-10-2020  diltiazem hydrochloride     Pressure     12.0     13.0
2  15-10-2020      docetaxel injection  Temperature      NaN     17.0
3  15-10-2020      docetaxel injection     Pressure      NaN     22.0
4  15-10-2020   ketamine hydrochloride  Temperature     24.0      NaN

   3:30:00  4:30:00  5:30:00  6:30:00  7:30:00  8:30:00  9:30:00  10:30:00  \
0      NaN     21.0     21.0       22     23.0     21.0     22.0        20
1      NaN     11.0     13.0       14     16.0     16.0     24.0        18
2     18.0      NaN     17.0       18      NaN      NaN     23.0        23
3     22.0      NaN     22.0       23      NaN      NaN     27.0        26
4      NaN     27.0      NaN       26     25.0     24.0     23.0        22

   11:30:00  12:30:00
0      20.0        21
1      19.0        20
2      25.0        25
3      29.0        28
4      21.0        20
```

[160]:
```
data.shape
```

[160]:
```
(18, 15)
```

[161]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        18 non-null     object
 1   Drug_Name   18 non-null     object
 2   Parameter   18 non-null     object
 3   1:30:00     16 non-null     float64
 4   2:30:00     16 non-null     float64
 5   3:30:00     12 non-null     float64
 6   4:30:00     14 non-null     float64
 7   5:30:00     16 non-null     float64
 8   6:30:00     18 non-null     int64
 9   7:30:00     16 non-null     float64
 10  8:30:00     14 non-null     float64
 11  9:30:00     16 non-null     float64
```

```
 12  10:30:00    18 non-null     int64
 13  11:30:00    16 non-null     float64
 14  12:30:00    18 non-null     int64
dtypes: float64(9), int64(3), object(3)
memory usage: 2.2+ KB
```

5 To convert the fat data into thin data

- Pandas has a function named melt

[162]:
```
data_melt = pd.melt(data,␣
↪id_vars=['Date','Drug_Name','Parameter'],var_name='Time',value_name='Reading')
```

[163]:
```
data_melt.shape
```

[163]:
```
(216, 5)
```

[164]:
```
data_melt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 216 entries, 0 to 215
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        216 non-null    object
 1   Drug_Name   216 non-null    object
 2   Parameter   216 non-null    object
 3   Time        216 non-null    object
 4   Reading     190 non-null    float64
dtypes: float64(1), object(4)
memory usage: 8.6+ KB
```

6 Change thin data into fat data

- Pandas has a function called Pivot

[165]:
```
data_melt.pivot(index=['Date','Drug_Name','Parameter'],columns = 'Time',values␣
↪= 'Reading').reset_index()
```

[165]:

Time	Date	Drug_Name	Parameter	10:30:00	11:30:00	\
0	15-10-2020	diltiazem hydrochloride	Pressure	18.0	19.0	
1	15-10-2020	diltiazem hydrochloride	Temperature	20.0	20.0	
2	15-10-2020	docetaxel injection	Pressure	26.0	29.0	
3	15-10-2020	docetaxel injection	Temperature	23.0	25.0	
4	15-10-2020	ketamine hydrochloride	Pressure	9.0	9.0	
5	15-10-2020	ketamine hydrochloride	Temperature	22.0	21.0	
6	16-10-2020	diltiazem hydrochloride	Pressure	24.0	NaN	
7	16-10-2020	diltiazem hydrochloride	Temperature	40.0	NaN	

8	16-10-2020	docetaxel injection	Pressure	28.0	29.0	
9	16-10-2020	docetaxel injection	Temperature	56.0	57.0	
10	16-10-2020	ketamine hydrochloride	Pressure	16.0	17.0	
11	16-10-2020	ketamine hydrochloride	Temperature	13.0	14.0	
12	17-10-2020	diltiazem hydrochloride	Pressure	11.0	13.0	
13	17-10-2020	diltiazem hydrochloride	Temperature	14.0	11.0	
14	17-10-2020	docetaxel injection	Pressure	28.0	29.0	
15	17-10-2020	docetaxel injection	Temperature	21.0	22.0	
16	17-10-2020	ketamine hydrochloride	Pressure	13.0	14.0	
17	17-10-2020	ketamine hydrochloride	Temperature	22.0	23.0	

Time	12:30:00	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:30:00	7:30:00	\
0	20.0	12.0	13.0	NaN	11.0	13.0	14.0	16.0	
1	21.0	23.0	22.0	NaN	21.0	21.0	22.0	23.0	
2	28.0	NaN	22.0	22.0	NaN	22.0	23.0	NaN	
3	25.0	NaN	17.0	18.0	NaN	17.0	18.0	NaN	
4	11.0	8.0	NaN	NaN	7.0	NaN	9.0	10.0	
5	20.0	24.0	NaN	NaN	27.0	NaN	26.0	25.0	
6	27.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	
7	42.0	34.0	35.0	36.0	36.0	37.0	38.0	37.0	
8	30.0	23.0	24.0	NaN	25.0	26.0	27.0	28.0	
9	58.0	46.0	47.0	NaN	48.0	48.0	49.0	50.0	
10	18.0	12.0	12.0	13.0	NaN	15.0	15.0	15.0	
11	15.0	8.0	9.0	10.0	NaN	11.0	12.0	12.0	
12	14.0	3.0	4.0	4.0	4.0	6.0	8.0	9.0	
13	10.0	20.0	19.0	19.0	18.0	17.0	16.0	15.0	
14	28.0	20.0	22.0	22.0	22.0	22.0	23.0	25.0	
15	23.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	
16	15.0	8.0	9.0	10.0	11.0	11.0	12.0	12.0	
17	24.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	

Time	8:30:00	9:30:00
0	16.0	24.0
1	21.0	22.0
2	NaN	27.0
3	NaN	23.0
4	11.0	10.0
5	24.0	23.0
6	25.0	25.0
7	38.0	39.0
8	29.0	28.0
9	52.0	55.0
10	15.0	NaN
11	11.0	NaN
12	NaN	9.0
13	NaN	13.0
14	26.0	27.0

```
15       19.0     20.0
16       11.0     12.0
17       20.0     21.0
```

7 Removing the NULL Values

[166]:
```
data_melt.head()
```

[166]:
```
         Date                Drug_Name    Parameter     Time  Reading
0  15-10-2020  diltiazem hydrochloride  Temperature  1:30:00     23.0
1  15-10-2020  diltiazem hydrochloride     Pressure  1:30:00     12.0
2  15-10-2020      docetaxel injection  Temperature  1:30:00      NaN
3  15-10-2020      docetaxel injection     Pressure  1:30:00      NaN
4  15-10-2020   ketamine hydrochloride  Temperature  1:30:00     24.0
```

[167]:
```
data_tidy = data_melt.
↪pivot(index=['Date','Drug_Name','Time'],columns='Parameter',values='Reading').
↪reset_index()
```

[168]:
```
data_tidy.head()
```

[168]:
```
Parameter        Date                Drug_Name      Time  Pressure  \
0          15-10-2020  diltiazem hydrochloride  10:30:00      18.0
1          15-10-2020  diltiazem hydrochloride  11:30:00      19.0
2          15-10-2020  diltiazem hydrochloride  12:30:00      20.0
3          15-10-2020  diltiazem hydrochloride   1:30:00      12.0
4          15-10-2020  diltiazem hydrochloride   2:30:00      13.0

Parameter  Temperature
0                 20.0
1                 20.0
2                 21.0
3                 23.0
4                 22.0
```

8 Understanding the NULL and None values

[169]:
```
type(None)
```

[169]:
```
NoneType
```

[170]:
```
type(np.nan)
```

[170]:
```
float
```

[171]:
```
pd.Series([1,np.nan,2])
```

[171]:
```
0    1.0
1    NaN
2    2.0
dtype: float64
```

[172]:
```
a = pd.Series(['1','np.nan',2,None])
type(a[2])
```

[172]:
```
int
```

[173]:
```
pd.Series([1,2,3,4,5,np.nan])
```

[173]:
```
0    1.0
1    2.0
2    3.0
3    4.0
4    5.0
5    NaN
dtype: float64
```

[174]:
```
pd.Series([1,2,3,None])
```

[174]:
```
0    1.0
1    2.0
2    3.0
3    NaN
dtype: float64
```

9 How to deal with NULL values

9.0.1 Check whether there are null values

[175]:
```
data.isnull().sum(axis=1)
```

[175]:
```
0     1
1     1
2     4
3     4
4     3
5     3
6     1
7     1
8     1
9     1
10    2
11    2
12    1
13    1
```

```
14    0
15    0
16    0
17    0
dtype: int64
```

9.0.2 Dropping the null values

[176]:
```
data.dropna(axis=0)
```

[176]:
```
          Date               Drug_Name    Parameter  1:30:00  2:30:00  \
14  17-10-2020     docetaxel injection  Temperature     12.0     13.0
15  17-10-2020     docetaxel injection     Pressure     20.0     22.0
16  17-10-2020  ketamine hydrochloride  Temperature     13.0     14.0
17  17-10-2020  ketamine hydrochloride     Pressure      8.0      9.0

    3:30:00  4:30:00  5:30:00  6:30:00  7:30:00  8:30:00  9:30:00  10:30:00  \
14     14.0     15.0     16.0       17     18.0     19.0     20.0        21
15     22.0     22.0     22.0       23     25.0     26.0     27.0        28
16     15.0     16.0     17.0       18     19.0     20.0     21.0        22
17     10.0     11.0     11.0       12     12.0     11.0     12.0        13

    11:30:00  12:30:00
14      22.0        23
15      29.0        28
16      23.0        24
17      14.0        15
```

9.0.3 Filling the null values with 0

[179]:
```
data.fillna(0)
```

[179]:
```
          Date               Drug_Name    Parameter  1:30:00  2:30:00  \
0   15-10-2020  diltiazem hydrochloride  Temperature     23.0     22.0
1   15-10-2020  diltiazem hydrochloride     Pressure     12.0     13.0
2   15-10-2020     docetaxel injection  Temperature      0.0     17.0
3   15-10-2020     docetaxel injection     Pressure      0.0     22.0
4   15-10-2020   ketamine hydrochloride  Temperature     24.0      0.0
5   15-10-2020   ketamine hydrochloride     Pressure      8.0      0.0
6   16-10-2020  diltiazem hydrochloride  Temperature     34.0     35.0
7   16-10-2020  diltiazem hydrochloride     Pressure     18.0     19.0
8   16-10-2020     docetaxel injection  Temperature     46.0     47.0
9   16-10-2020     docetaxel injection     Pressure     23.0     24.0
10  16-10-2020   ketamine hydrochloride  Temperature      8.0      9.0
11  16-10-2020   ketamine hydrochloride     Pressure     12.0     12.0
12  17-10-2020  diltiazem hydrochloride  Temperature     20.0     19.0
13  17-10-2020  diltiazem hydrochloride     Pressure      3.0      4.0
```

```
14  17-10-2020      docetaxel injection  Temperature     12.0     13.0
15  17-10-2020      docetaxel injection     Pressure     20.0     22.0
16  17-10-2020   ketamine hydrochloride  Temperature     13.0     14.0
17  17-10-2020   ketamine hydrochloride     Pressure      8.0      9.0

    3:30:00  4:30:00  5:30:00  6:30:00  7:30:00  8:30:00  9:30:00  10:30:00  \
0       0.0     21.0     21.0       22     23.0     21.0     22.0        20
1       0.0     11.0     13.0       14     16.0     16.0     24.0        18
2      18.0      0.0     17.0       18      0.0      0.0     23.0        23
3      22.0      0.0     22.0       23      0.0      0.0     27.0        26
4       0.0     27.0      0.0       26     25.0     24.0     23.0        22
5       0.0      7.0      0.0        9     10.0     11.0     10.0         9
6      36.0     36.0     37.0       38     37.0     38.0     39.0        40
7      20.0     21.0     22.0       23     24.0     25.0     25.0        24
8       0.0     48.0     48.0       49     50.0     52.0     55.0        56
9       0.0     25.0     26.0       27     28.0     29.0     28.0        28
10     10.0      0.0     11.0       12     12.0     11.0      0.0        13
11     13.0      0.0     15.0       15     15.0     15.0      0.0        16
12     19.0     18.0     17.0       16     15.0      0.0     13.0        14
13      4.0      4.0      6.0        8      9.0      0.0      9.0        11
14     14.0     15.0     16.0       17     18.0     19.0     20.0        21
15     22.0     22.0     22.0       23     25.0     26.0     27.0        28
16     15.0     16.0     17.0       18     19.0     20.0     21.0        22
17     10.0     11.0     11.0       12     12.0     11.0     12.0        13

    11:30:00  12:30:00
0       20.0        21
1       19.0        20
2       25.0        25
3       29.0        28
4       21.0        20
5        9.0        11
6        0.0        42
7        0.0        27
8       57.0        58
9       29.0        30
10      14.0        15
11      17.0        18
12      11.0        10
13      13.0        14
14      22.0        23
15      29.0        28
16      23.0        24
17      14.0        15
```

9.0.4 Fill the NULL Values with Average

[181]:
```
data['2:30:00'].fillna(data['2:30:00'].mean())
```

[181]:
```
0     22.0000
1     13.0000
2     17.0000
3     22.0000
4     18.8125
5     18.8125
6     35.0000
7     19.0000
8     47.0000
9     24.0000
10     9.0000
11    12.0000
12    19.0000
13     4.0000
14    13.0000
15    22.0000
16    14.0000
17     9.0000
Name: 2:30:00, dtype: float64
```

[ ]:
```
# def replace_nan(x):
#     return x['Drug_Name']['.mean()
```

[ ]:
```

```

Day_40_061223

January 23, 2024

1 Data Visualization

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

2 Importing matplotlib and seaborn libraries

[2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

3 Downloading the CSV File using URL

[3]:
```
!gdown 15I3g3TBZvN6-WxLWMwFi1_h8oeT6gA7G
```

```
Downloading…
From: https://drive.google.com/uc?id=15I3g3TBZvN6-WxLWMwFi1_h8oeT6gA7G
To: C:\Data\Data_science\Data Science RIA\3 Python\3 Data Visualization -
Matplotlib and Seaborn\Codes\final_vg.csv

  0%|          | 0.00/2.15M [00:00<?, ?B/s]
 24%|##4       | 524k/2.15M [00:00<00:00, 2.27MB/s]
 98%|#########7| 2.10M/2.15M [00:00<00:00, 7.08MB/s]
100%|##########| 2.15M/2.15M [00:00<00:00, 6.13MB/s]
```

4 Reading the CSV File

[4]:
```
data = pd.read_csv("final_vg.csv")
```

[5]:
```
data.head()
```

[5]:

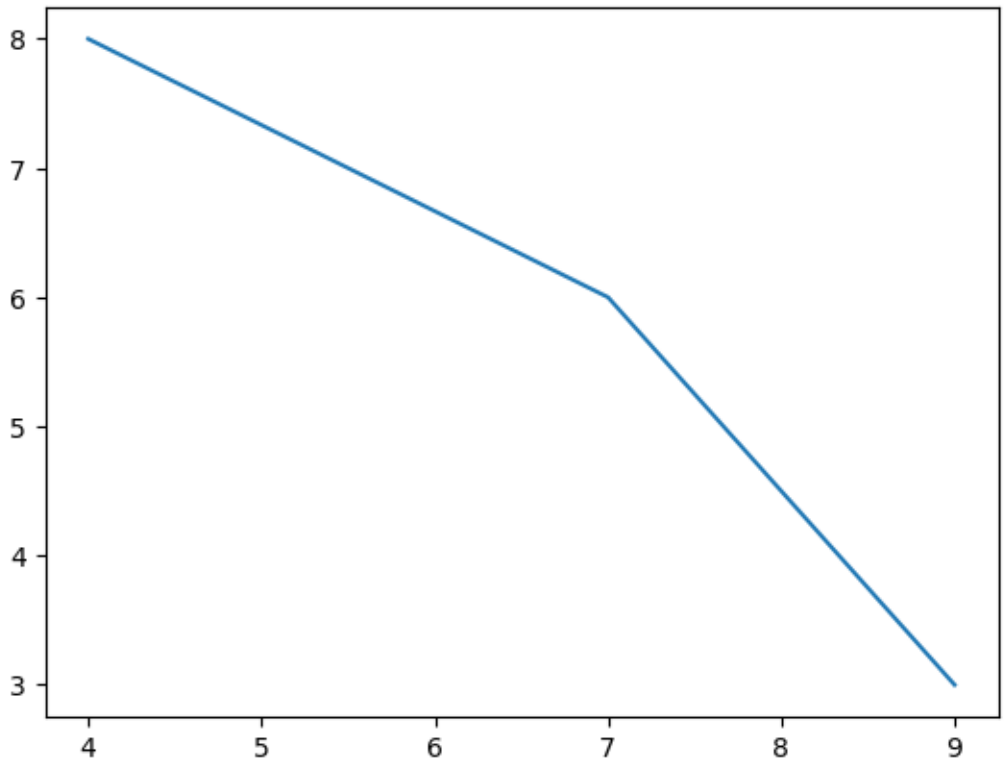
	Unnamed: 0	Rank	Name	Platform	Year	\
0	0	2061	1942	NES	1985.0	
1	1	9137	¡Shin Chan Flipa en colores!	DS	2007.0	
2	2	14279	.hack: Sekai no Mukou ni + Versus	PS3	2012.0	
3	3	8359	.hack//G.U. Vol.1//Rebirth	PS2	2006.0	
4	4	7109	.hack//G.U. Vol.2//Reminisce	PS2	2006.0	

```
          Genre           Publisher  NA_Sales  EU_Sales  JP_Sales  \
0       Shooter              Capcom  4.569217  3.033887  3.439352
1      Platform           505 Games  2.076955  1.493442  3.033887
2        Action  Namco Bandai Games  1.145709  1.762339  1.493442
3  Role-Playing  Namco Bandai Games  2.031986  1.389856  3.228043
4  Role-Playing  Namco Bandai Games  2.792725  2.592054  1.440483

   Other_Sales  Global_Sales
0     1.991671     12.802935
1     0.394830      7.034163
2     0.408693      4.982552
3     0.394830      7.226880
4     1.493442      8.363113
```

5 Plotting general points

[6]:
```
x = [4,7,9]
y = [8,6,3]
plt.plot(x,y)
```

[6]:
```
[<matplotlib.lines.Line2D at 0x25e8e668b00>]
```



6 Find the top 5 geners of video games

7 Univariate Analysis - Categorical

- Distribution of Each category
- What proportion each category has on the total

[7]:
```
data['Genre']
```

[7]:
```
0             Shooter
1            Platform
2              Action
3        Role-Playing
4        Role-Playing
             …
16647          Sports
16648            Misc
16649            Misc
16650    Role-Playing
16651          Action
Name: Genre, Length: 16652, dtype: object
```

[8]:
```
cat_count = data['Genre'].value_counts().sort_values(ascending=False)
cat_count
```

[8]:
```
Genre
Action          3316
Sports          2400
Misc            1739
Role-Playing    1488
Shooter         1310
Adventure       1286
Racing          1249
Platform         886
Simulation       867
Fighting         848
Strategy         681
Puzzle           582
Name: count, dtype: int64
```

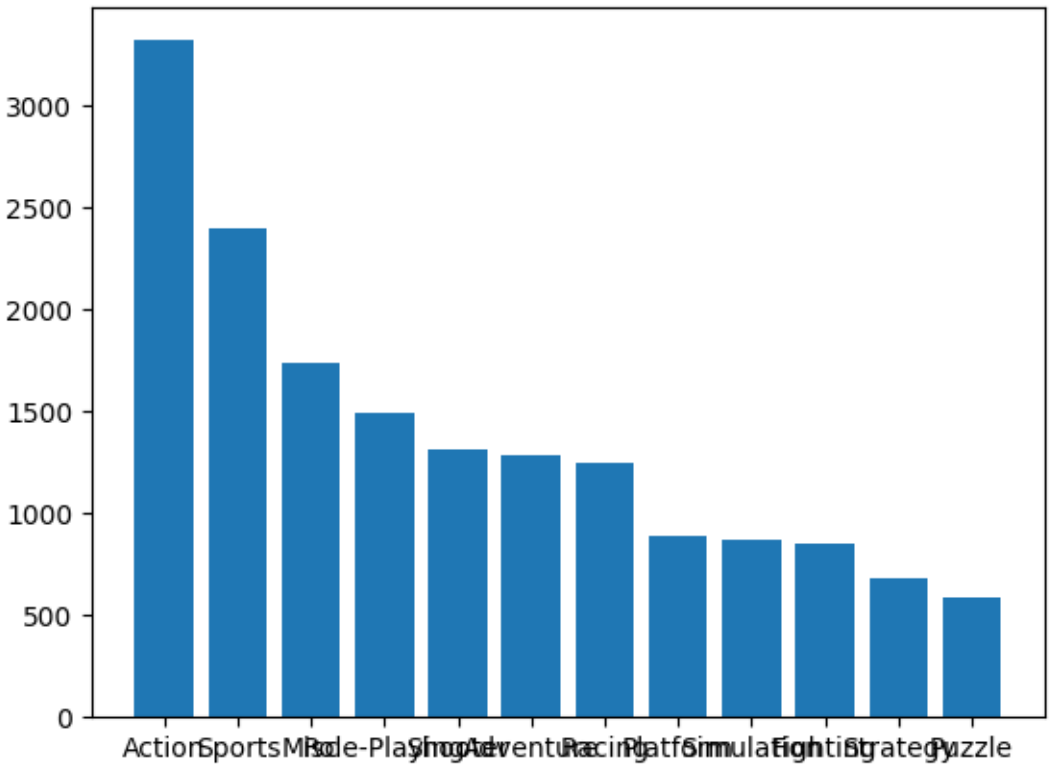
[9]:
```
cat_count.index
```

[9]:
```
Index(['Action', 'Sports', 'Misc', 'Role-Playing', 'Shooter', 'Adventure',
       'Racing', 'Platform', 'Simulation', 'Fighting', 'Strategy', 'Puzzle'],
      dtype='object', name='Genre')
```

7.1 Bar chart to visualize the distribution

[10]:
```
x_bar = cat_count.index
y_bar = cat_count
plt.bar(x_bar,y_bar)
```

[10]:
```
<BarContainer object of 12 artists>
```



7.1.1 Here the names on the x axis is bit messy so

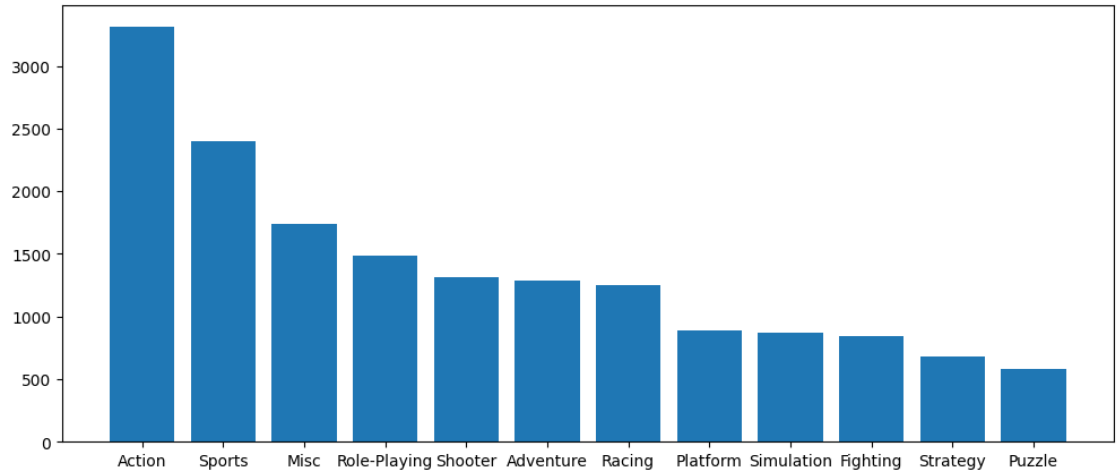
So there are two way

- Increasing the size of the plot
- Rotating the xlabels to certain angles

7.1.2 Figure size

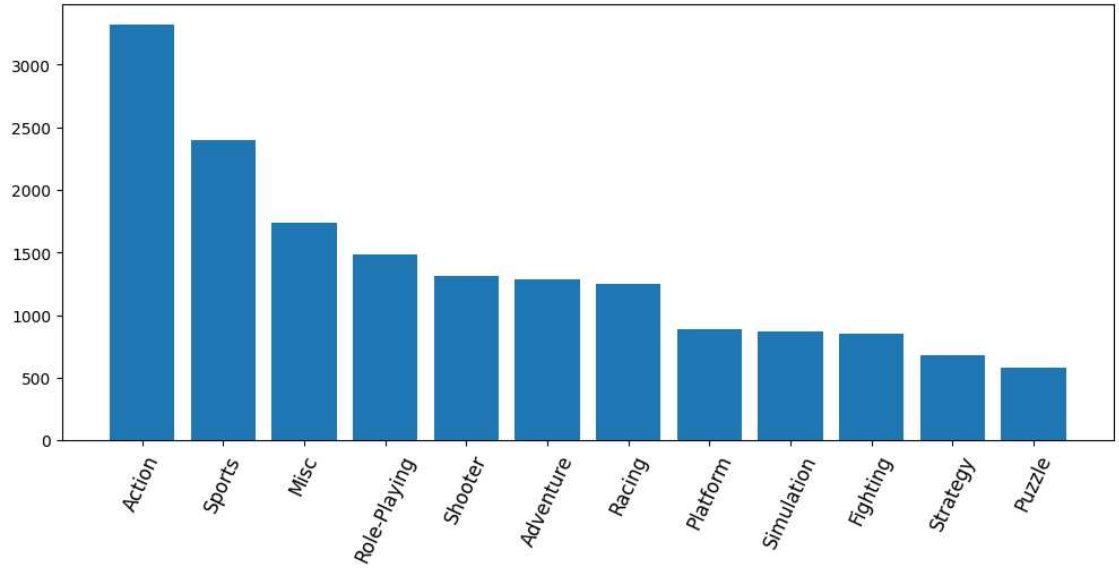
[11]:
```
plt.figure(figsize=(12,5))
x_bar = cat_count.index
y_bar = cat_count
plt.bar(x_bar,y_bar)
```

[11]:
```
<BarContainer object of 12 artists>
```



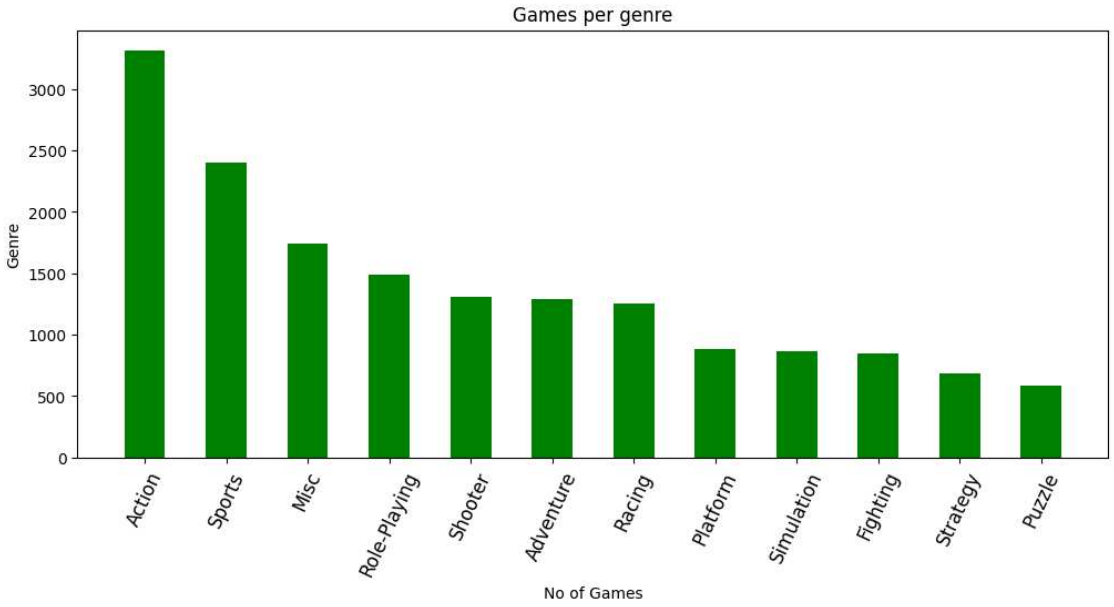
7.1.3 Rotating the label with angle

[12]:
```
plt.figure(figsize=(12,5))
x_bar = cat_count.index
y_bar = cat_count
plt.bar(x_bar,y_bar)
plt.xticks(rotation=65,fontsize=12)
plt.show()
```



7.2 Adding title, x and y lables

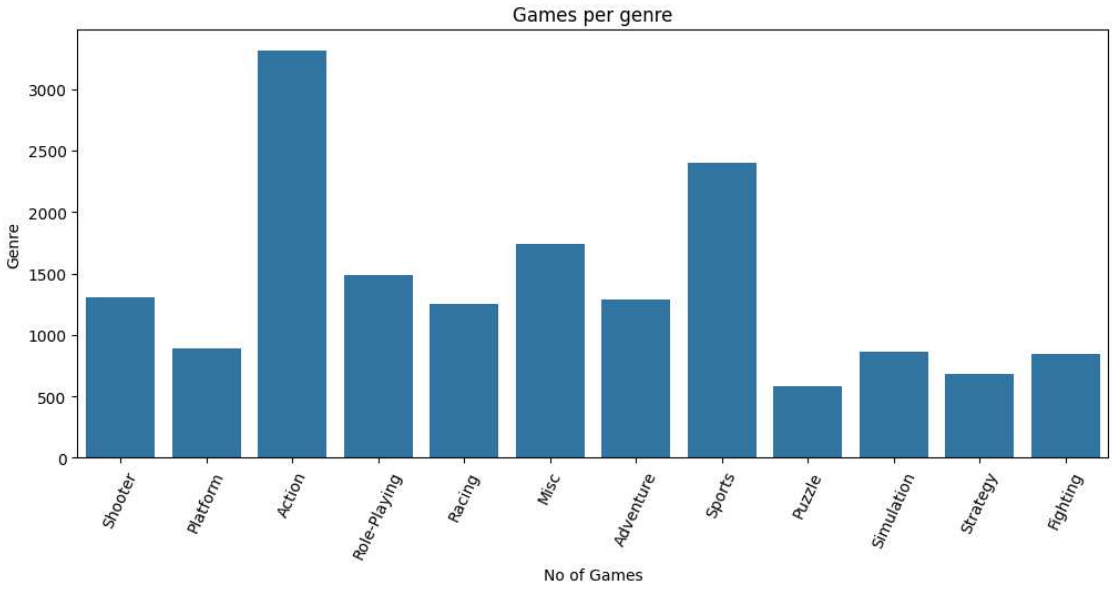
[13]:
```
plt.figure(figsize=(12,5))
x_bar = cat_count.index
y_bar = cat_count
# Here in bar plot we can add color, width of bar
plt.bar(x_bar,y_bar,color='green',width=0.5)
plt.title("Games per genre")
plt.xlabel("No of Games")
plt.ylabel("Genre")
plt.xticks(rotation=65,fontsize=12)
plt.show()
```



Upto here we wrote a lot of code using matplotlib library but using seaborn is much more simple

8 Plotting distribution in seaborn

[14]:
```
plt.figure(figsize=(12,5))
plt.title("Games per genre")
plt.xlabel("No of Games")
plt.ylabel("Genre")
sns.countplot(x='Genre',data=data)
plt.xticks(rotation=65)
plt.savefig("Plots/GamesPerGenre.jpg") # Saving the figure to local directory
plt.show()
```



9 Contribution to the total

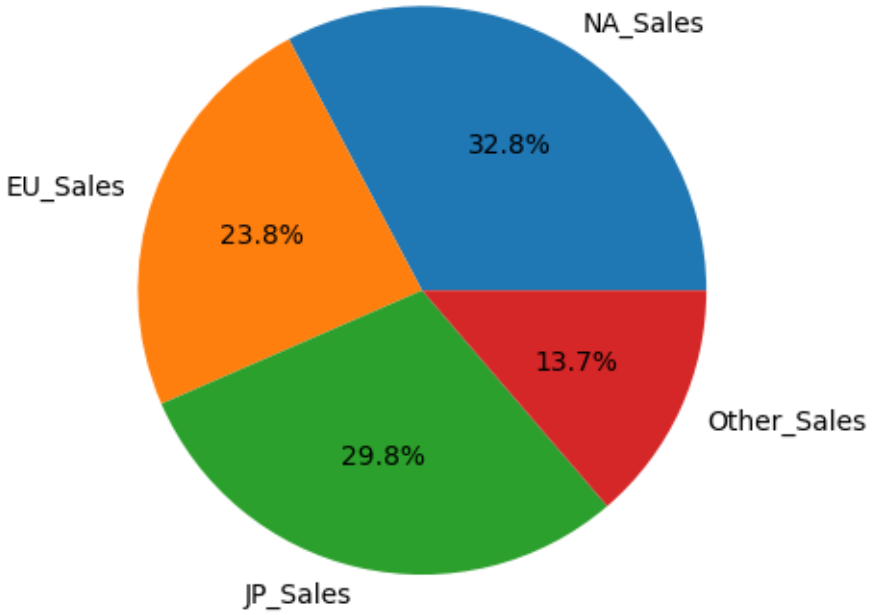
[15]:
```
data.columns
```

[15]:
```
Index(['Unnamed: 0', 'Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher',
       'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales'],
      dtype='object')
```

[16]:
```
sales_data = data[['NA_Sales','EU_Sales','JP_Sales','Other_Sales']]
total_sales = sales_data.sum(axis=0)
total_sales
```

[16]:
```
NA_Sales       45831.525845
EU_Sales       33251.970702
JP_Sales       41624.625635
Other_Sales    19180.256828
dtype: float64
```

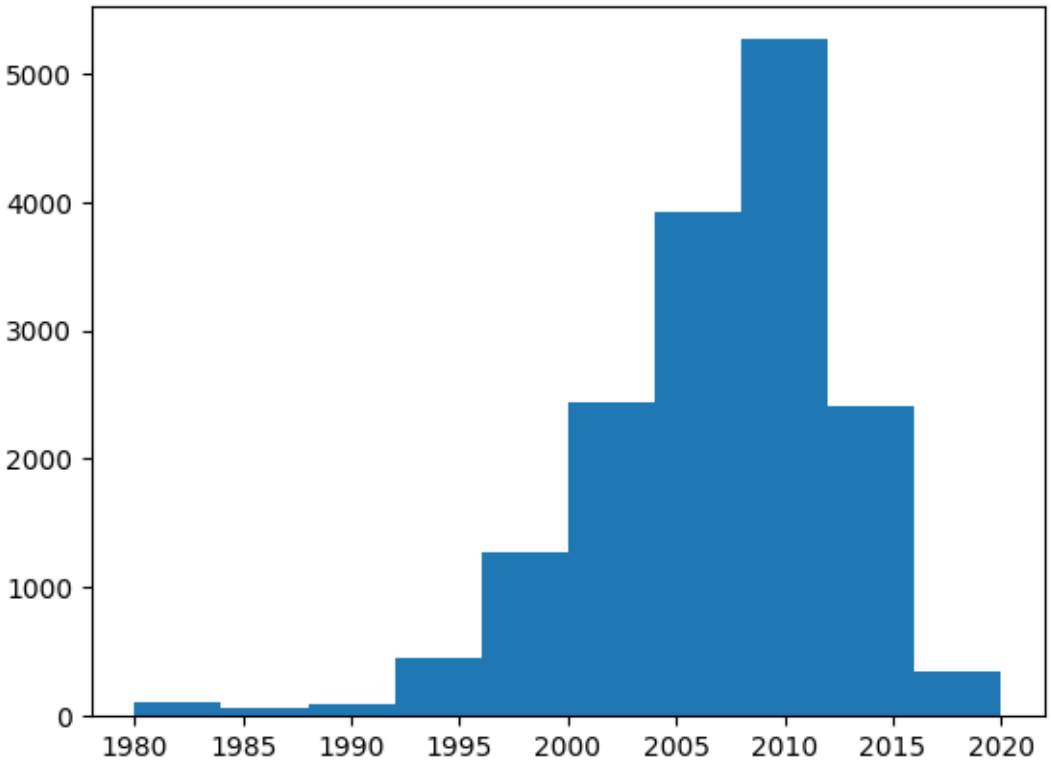
[17]:
```
plt.
 ↪pie(total_sales,labels=['NA_Sales','EU_Sales','JP_Sales','Other_Sales'],autopct='%1.
 ↪1f%%')
plt.savefig("Plots/Contributiontosales.jpg")
plt.show()
```



10 Univariate Analysis - Numerical

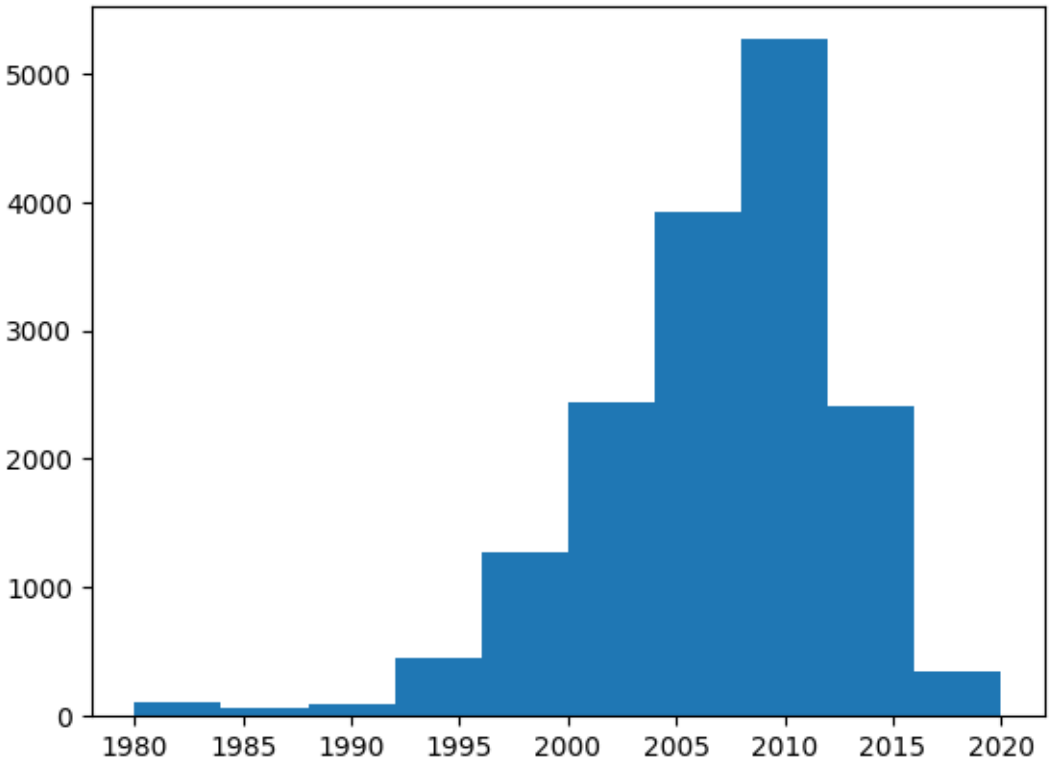
10.1 How to identify the popularity of a video game year by year

[18]:
```
plt.hist(data['Year'])
plt.show()
```



10.2 We can reduce or increase the bars by using bins

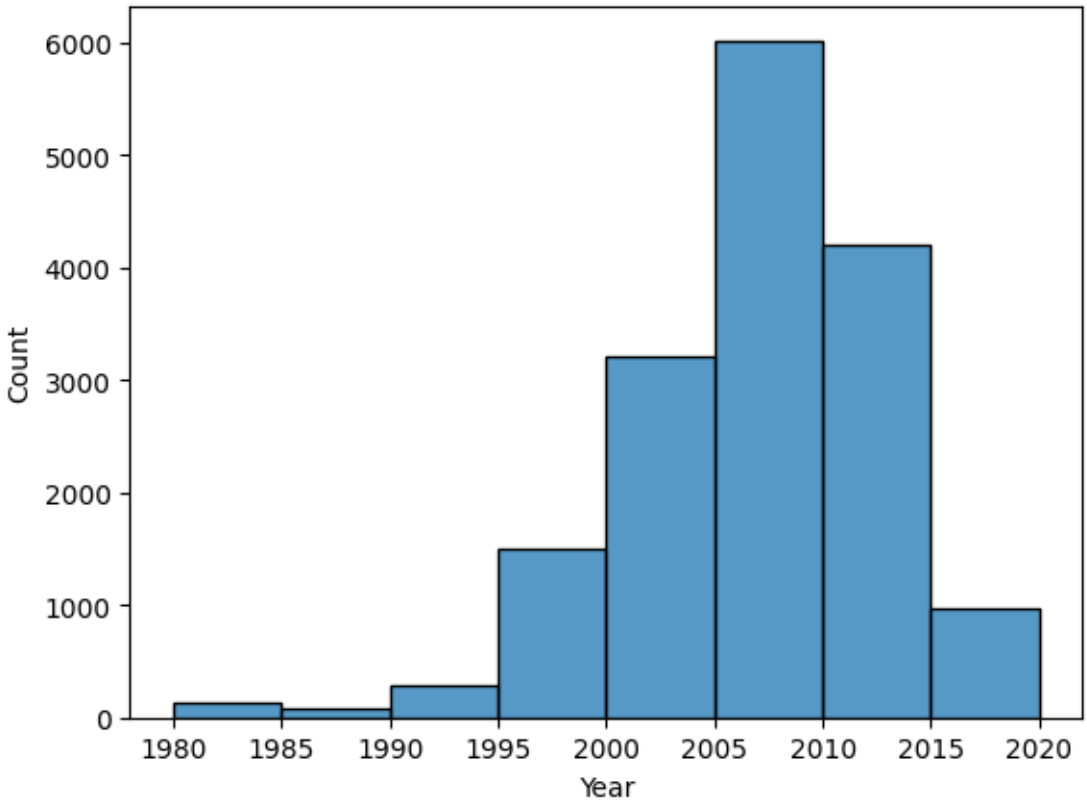
[19]:
```
plt.hist(data['Year'],bins=10)
plt.show()
```



11 Using seaborn

[20]:
```
sns.histplot(data['Year'],bins=8)
```

[20]:
```
<Axes: xlabel='Year', ylabel='Count'>
```



12 Bi variate Analysis

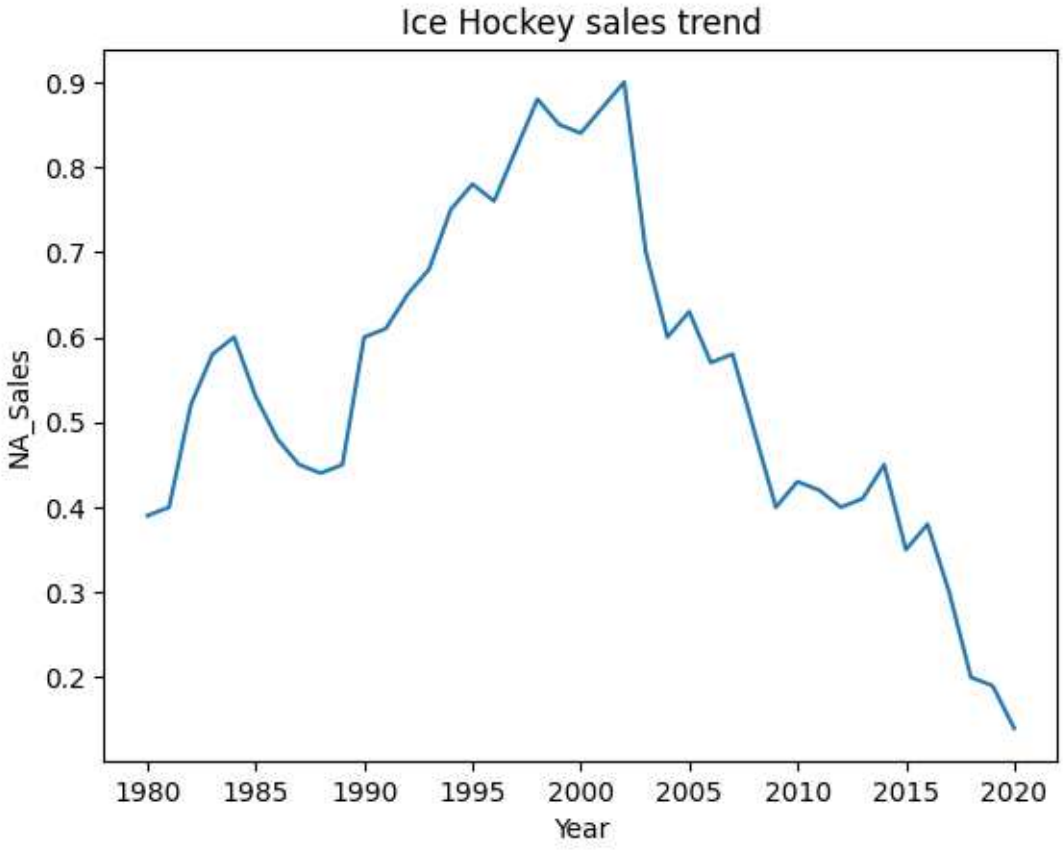
- It has 2 types of data points

[21]:
```
data.loc[data['Name']=='Ice Hockey'].head()
```

[21]:
```
      Unnamed: 0  Rank         Name Platform    Year   Genre   Publisher  \
6073        6073   639  Ice Hockey      NES  1988.0  Sports    Nintendo
6074        6074  4027  Ice Hockey     2600  1980.0  Sports  Activision
6075        6075  4149  Ice Hockey     2600  1991.0  Sports  Activision
6076        6076  4149  Ice Hockey     2600  1992.0  Sports  Activision
6077        6077  4149  Ice Hockey     SNES  1993.0  Sports  Activision

      NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales
6073      0.44  3.860566  4.751539     2.004268     15.855389
6074      0.39  1.493442  2.741701     0.394830      4.956249
6075      0.61  0.020000  0.000000     0.010000      0.470000
6076      0.65  0.020000  0.000000     0.010000      0.470000
6077      0.68  0.020000  0.000000     0.010000      0.470000
```

13 Getting the Sales trend of Ice Hockey

[22]:
```
plt.title("Ice Hockey sales trend")
ih = data.loc[data['Name']=='Ice Hockey']
sns.lineplot(x='Year',y='NA_Sales',data=ih)
plt.savefig("Plots/Ice Hockey sales trend.jpg")
plt.show()
```



14 Including two or more trends in one plot

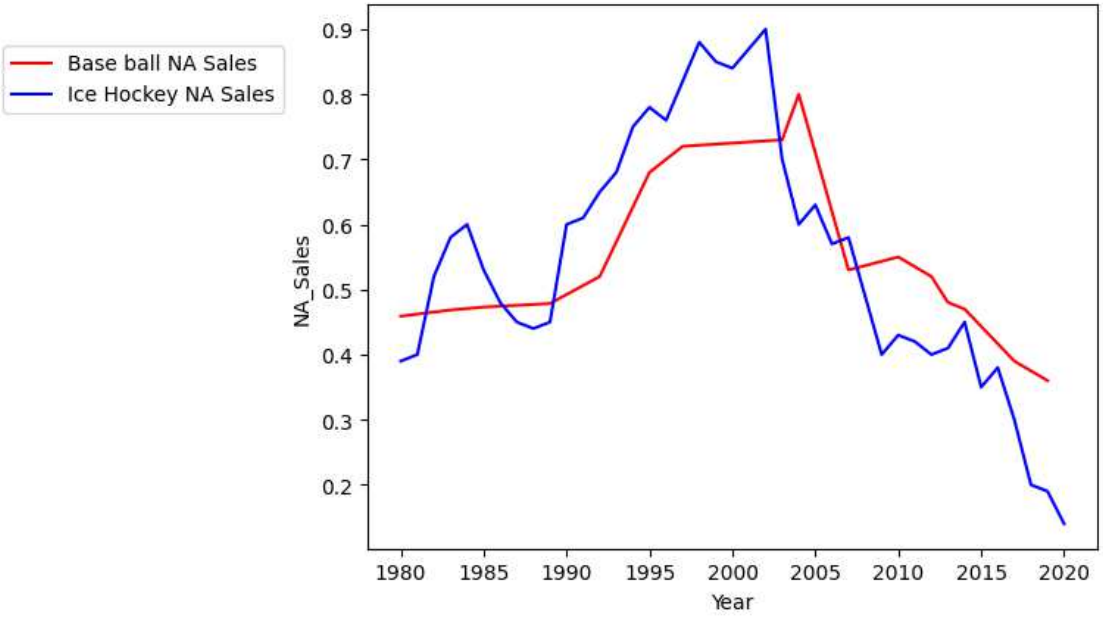
[23]:
```
baseball = data.loc[data['Name']=='Baseball']
baseball.head()
```

[23]:

	Unnamed: 0	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	\
941	941	324	Baseball	NES	1980.0	Sports	Nintendo	0.459000	
942	942	422	Baseball	NES	1983.0	Sports	Nintendo	0.468529	
943	943	231	Baseball	GB	1985.0	Sports	Nintendo	0.473000	
944	944	1144	Baseball	GB	1989.0	Sports	Nintendo	0.478448	
945	945	134	Baseball	GB	1992.0	Sports	Nintendo	0.520000	

	EU_Sales	JP_Sales	Other_Sales	Global_Sales
941	2.320000	5.230000	1.230000	9.239000
942	2.697415	5.854415	1.087977	10.108336
943	3.074830	6.478831	0.945954	10.972614
944	3.452245	7.103246	0.803931	11.837870
945	3.829660	7.727661	0.661908	12.739229

[24]:
```
sns.lineplot(x='Year',y='NA_Sales',data=baseball,color='r',label='Base ball NA␣
 ↪Sales')
sns.lineplot(x='Year',y='NA_Sales',data=ih,color='b',label='Ice Hockey NA␣
 ↪Sales')
plt.legend(loc=(-0.5,0.8)) # Changing the location of legend
plt.show()
```



Day_41_071223

January 23, 2024

1 Correlation

1.1 If two values are correlated

1.1.1 Positively Correlated - If one value increase other value also increase

1.1.2 Negative Correlated - If one increase other descrease vice versa

For example - If you take more calories the weight will also increase So Calories and Weight are positively Correlated

[37]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("final_vg.csv")
```

[38]:
```
data.head()
```

[38]:

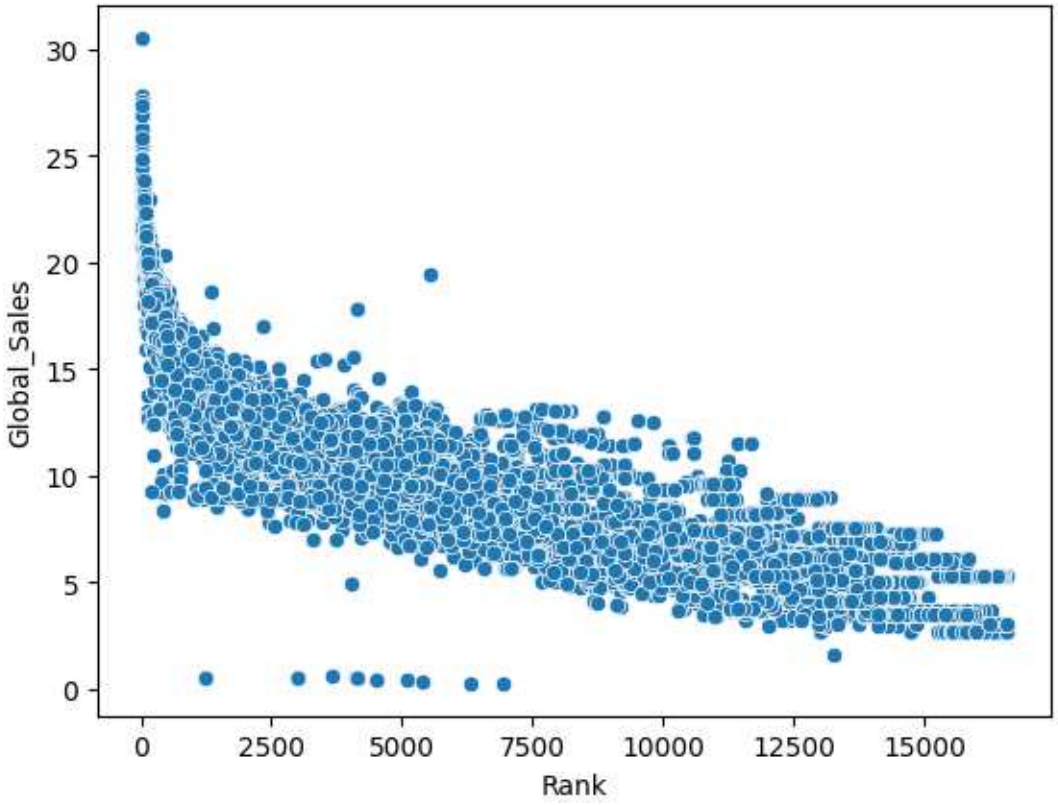
	Unnamed: 0	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	0	2061	1942	NES	1985.0	Shooter	Capcom	4.569217	3.033887	3.439352	1.991671	12.802935
1	1	9137	¡Shin Chan Flipa en colores!	DS	2007.0	Platform	505 Games	2.076955	1.493442	3.033887	0.394830	7.034163
2	2	14279	.hack: Sekai no Mukou ni + Versus	PS3	2012.0	Action	Namco Bandai Games	1.145709	1.762339	1.493442	0.408693	4.982552
3	3	8359	.hack//G.U. Vol.1//Rebirth	PS2	2006.0	Role-Playing	Namco Bandai Games	2.031986	1.389856	3.228043	0.394830	7.226880
4	4	7109	.hack//G.U. Vol.2//Reminisce	PS2	2006.0	Role-Playing	Namco Bandai Games	2.792725	2.592054	1.440483	1.493442	8.363113

2 Suppose we have find relation between two data points

3 Scatter Plot using Seaborn

[39]:
```
sns.scatterplot(data,x='Rank',y='Global_Sales')
```

[39]:
```
<Axes: xlabel='Rank', ylabel='Global_Sales'>
```



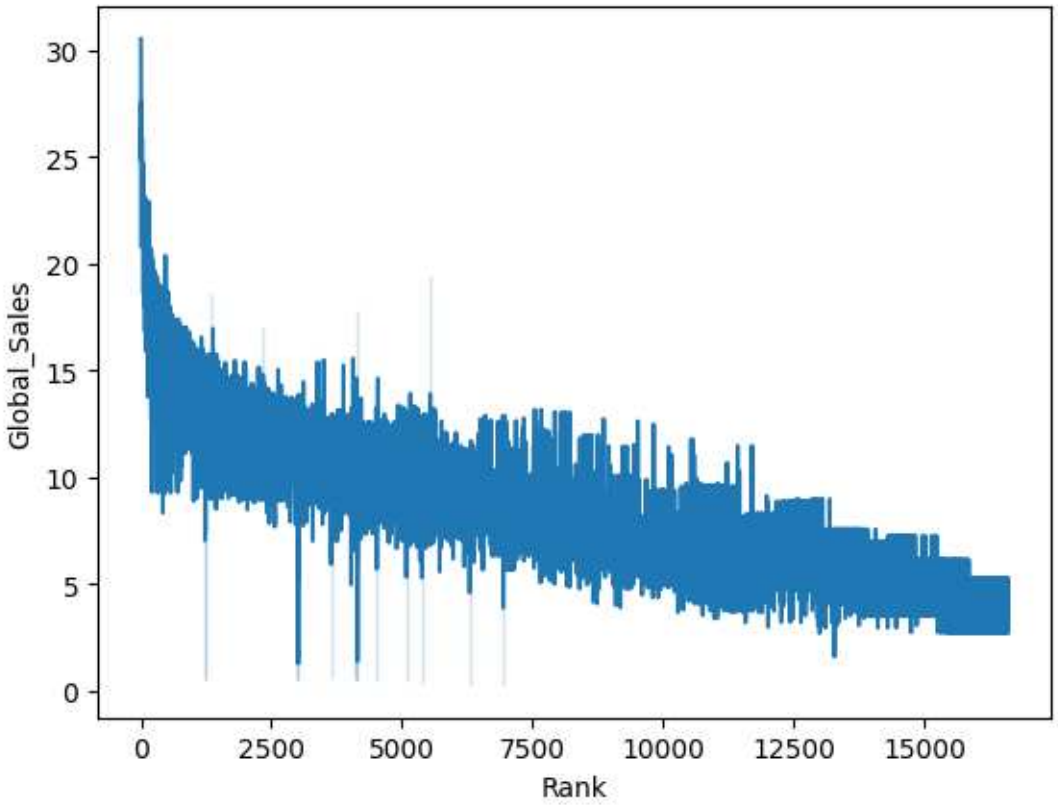
- In this scatter plot we can observe that when there is higher sales the rank is less and vice-versa
- So it is negatively correlated

3.0.1 Same can be visualized using line plot

4 Line plot using Seabornm

[40]:
```
sns.lineplot(data,x='Rank',y='Global_Sales')
```

[40]:
```
<Axes: xlabel='Rank', ylabel='Global_Sales'>
```



- But it is not that effective So we consider scatter plot

Upto now we looked into continous - continous data

5 Categorical - Categorical data

[41]:
```
data.head(3)
```

[41]:
```
   Unnamed: 0   Rank                                 Name Platform    Year  \
0           0   2061                                 1942      NES  1985.0
1           1   9137       ¡Shin Chan Flipa en colores!       DS  2007.0
2           2  14279  .hack: Sekai no Mukou ni + Versus      PS3  2012.0

      Genre           Publisher  NA_Sales  EU_Sales  JP_Sales  Other_Sales  \
0   Shooter             Capcom  4.569217  3.033887  3.439352     1.991671
1  Platform          505 Games  2.076955  1.493442  3.033887     0.394830
2    Action  Namco Bandai Games  1.145709  1.762339  1.493442     0.408693

   Global_Sales
0     12.802935
```

```
1      7.034163
2      4.982552
```

[42]:
```
top3_pub = data['Publisher'].value_counts().index[:3]
top3_gen = data['Genre'].value_counts().index[:3]
top3_plat = data['Platform'].value_counts().index[:3]
```

[43]:
```
top3_pub
```

[43]:
```
Index(['Electronic Arts', 'Activision', 'Namco Bandai Games'], dtype='object',
name='Publisher')
```

[44]:
```
top3_gen
```

[44]:
```
Index(['Action', 'Sports', 'Misc'], dtype='object', name='Genre')
```

[45]:
```
top3_plat
```

[45]:
```
Index(['DS', 'PS2', 'PS3'], dtype='object', name='Platform')
```

[46]:
```
top3_data = data.loc[(data['Publisher'].isin(top3_pub)) & (data['Genre'].
↪isin(top3_gen)) & (data['Platform'].isin(top3_plat))]
```

[47]:
```
top3_data
```

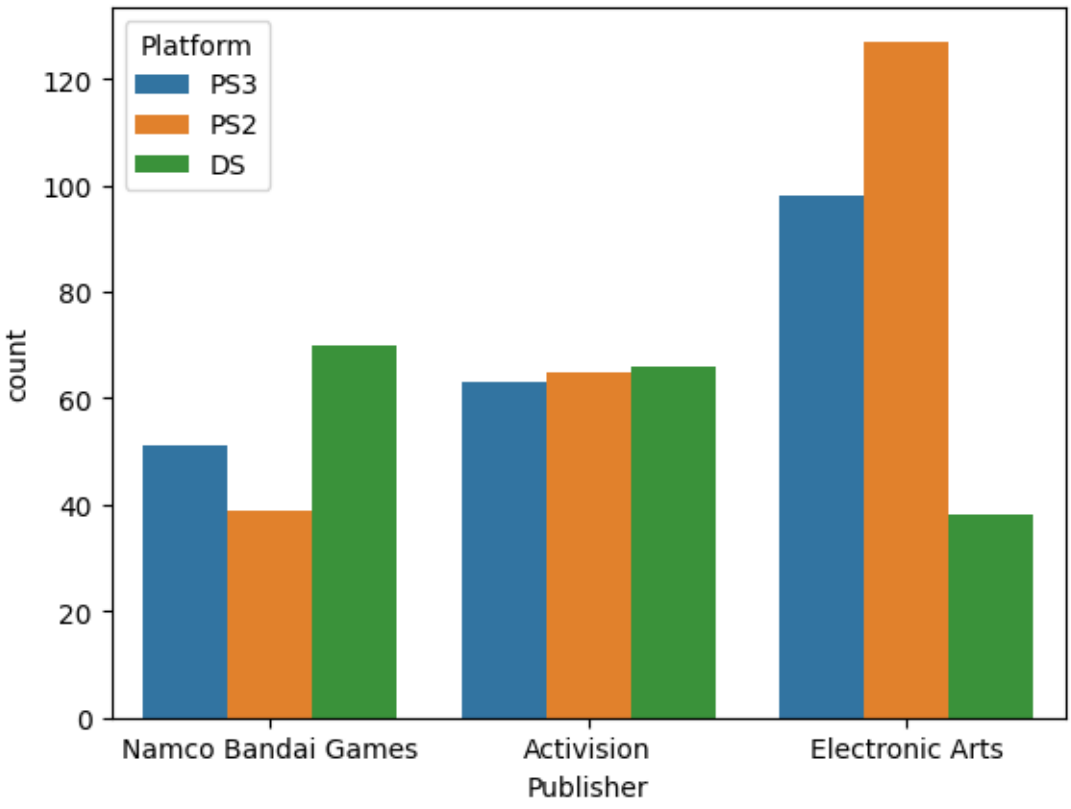
[47]:
```
       Unnamed: 0   Rank                                                Name  \
2               2  14279                 .hack: Sekai no Mukou ni + Versus
13             13   2742                                       [Prototype 2]
16             16   1604                                         [Prototype]
19             19   1741                           007: Quantum of Solace
21             21   4501                           007: Quantum of Solace
…             …    …                                                   …
16438       16438  14938  Yes! Precure 5 Go Go Zenin Shu Go! Dream Festival
16479       16479  10979                          Young Justice: Legacy
16601       16601  11802                      ZhuZhu Pets: Quest for Zhu
16636       16636   9196                        Zoobles! Spring to Life!
16640       16640   9816                                            Zubo

      Platform    Year   Genre           Publisher  NA_Sales  EU_Sales  \
2          PS3  2012.0  Action  Namco Bandai Games  1.145709  1.762339
13         PS3  2012.0  Action         Activision  3.978349  3.727034
16         PS3  2009.0  Action         Activision  4.569217  4.108402
19         PS3  2008.0  Action         Activision  4.156030  4.346074
21         PS2  2008.0  Action         Activision  3.228043  2.738800
…        …      …     …                      …       …       …
16438       DS  2008.0  Action  Namco Bandai Games  1.087977  0.592445
16479      PS3  2013.0  Action  Namco Bandai Games  2.186589  1.087977
16601       DS  2011.0    Misc         Activision  2.340740  1.525543
```

```
16636       DS  2011.0    Misc          Activision  2.697415  1.087977
16640       DS  2008.0    Misc     Electronic Arts  2.592054  1.493442

       JP_Sales  Other_Sales  Global_Sales
2      1.493442     0.408693      4.982552
13     0.848807     2.792725     11.447989
16     1.187272     3.339269     13.181205
19     1.087977     3.390562     12.980643
21     2.585598     3.652926     11.780257
…           …            …            …
16438  1.087977     0.394830      3.509168
16479  3.409089     0.394830      7.359902
16601  3.103825     0.394830      7.372592
16636  2.760718     0.394830      6.915540
16640  1.493442     0.394830      5.969572

[617 rows x 12 columns]
```

6 Dodged Bar chart

[48]:
```
sns.countplot(x='Publisher',data=top3_data,hue='Platform')
```

[48]:
```
<Axes: xlabel='Publisher', ylabel='count'>
```



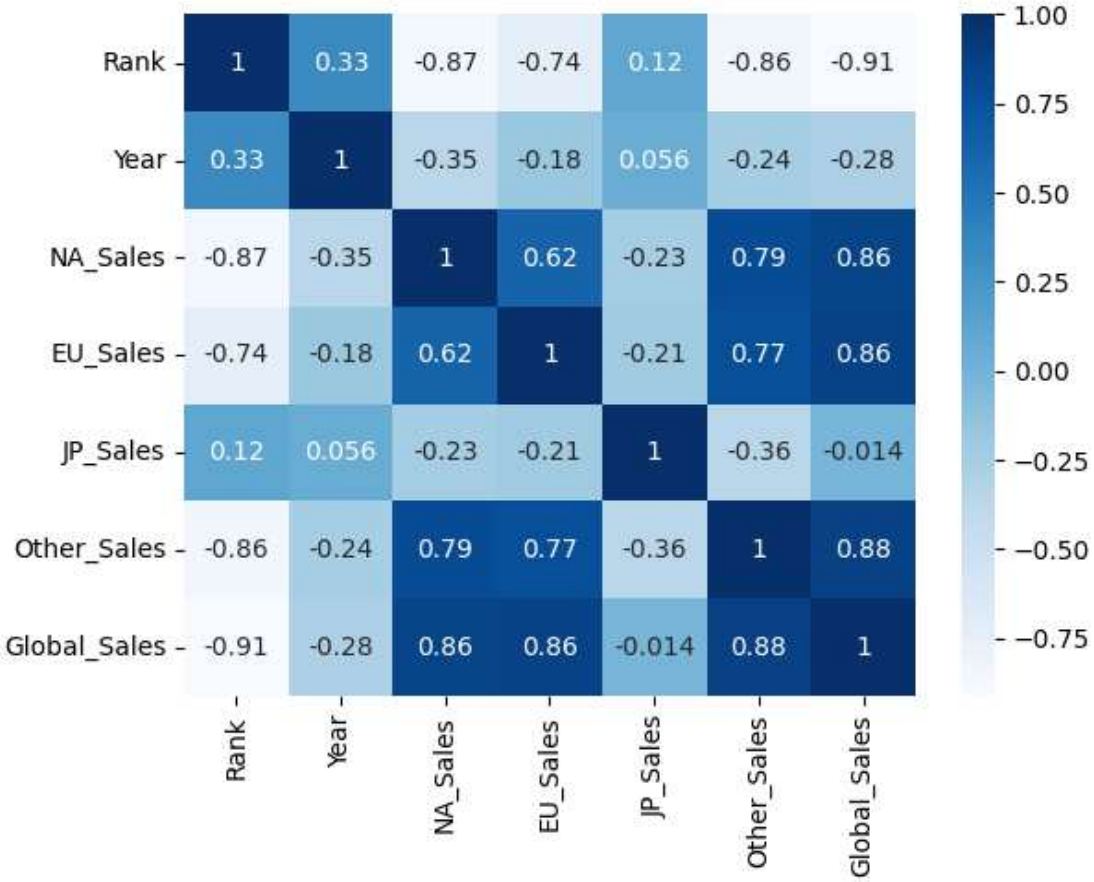
7 Multivariate data

8 Heatmap - It shows the correlation between numerical columns

[49]:
```
top3_multi = top3_data.drop(['Name','Unnamed:␣
↪0','Platform','Genre','Publisher'],axis=1)
```

[52]:
```
sns.heatmap(top3_multi.corr(),cmap='Blues',annot=True)
```

[52]: <Axes: >



Day_36_011223

January 23, 2024

[121]:
```
import pandas as pd
import numpy as np
```

[122]:
```
movies = pd.read_csv("movies.csv") # to choose index col throw an argument␣
↪index_col = 0
```

[123]:
```
directors = pd.read_csv("directors.csv")
```

[124]:
```
movies.shape
```

[124]:
```
(1465, 12)
```

[125]:
```
directors.shape
```

[125]:
```
(2349, 4)
```

[126]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1465 entries, 0 to 1464
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Unnamed: 0    1465 non-null   int64  
 1   id            1465 non-null   int64  
 2   budget        1465 non-null   int64  
 3   popularity    1465 non-null   int64  
 4   revenue       1465 non-null   int64  
 5   title         1465 non-null   object 
 6   vote_average  1465 non-null   float64
 7   vote_count    1465 non-null   int64  
 8   director_id   1465 non-null   int64  
 9   year          1465 non-null   int64  
 10  month         1465 non-null   object 
 11  day           1465 non-null   object 
dtypes: float64(1), int64(8), object(3)
memory usage: 137.5+ KB
```

[127]:
```
directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2349 entries, 0 to 2348
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     2349 non-null   int64
 1   director_name  2349 non-null   object
 2   id             2349 non-null   int64
 3   gender         1724 non-null   object
dtypes: int64(2), object(2)
memory usage: 73.5+ KB
```

[128]:
```
movies.drop('Unnamed: 0',axis=1,inplace=True)
```

[129]:
```
directors.drop('Unnamed: 0',axis=1,inplace=True)
```

[130]:
```
movies.sort_values('vote_count',ascending=False)
```

[130]:

	id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day
59	43693	160000000	167	825532764	Inception	8.1	13752	4765	2010	Jul	Wednesday
45	43662	185000000	187	1004558444	The Dark Knight	8.2	12002	4765	2008	Jul	Wednesday
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday
58	43692	165000000	724	675120017	Interstellar	8.1	10867	4765			
178	43884	100000000	82	425368238	Django Unchained	7.8	10099	4927			
…	…	…	…	…	…	…	…	…	…	…	…
1431	47962	0	0	0	Walking and Talking	6.6	7	6204			
879	45373	0	0	0	The Magic Flute	6.9	6	4847			
1438	48145	500000	0	0	Everything Put Together	5.0	2	4773			
1440	48155	0	0	0	Alleluia! The Devil's Carnival	6.0	2	6056			
1378	47387	0	0	0	An Everlasting Piece	6.0	1	5037			

```
58    2014   Nov  Wednesday
178   2012   Dec    Tuesday
…     …    …          …
1431  1996   Jul  Wednesday
879   2006   Sep   Thursday
1438  2001   Nov     Friday
1440  2016   Mar    Tuesday
1378  2000   Dec     Friday

[1465 rows x 11 columns]
```

[131]:
```
movies.head()
```

[131]:
```
      id     budget  popularity     revenue  \
0  43597  237000000         150  2787965087
1  43598  300000000         139   961000000
2  43599  245000000         107   880674609
3  43600  250000000         112  1084939099
4  43602  258000000         115   890871626

                                    title  vote_average  vote_count  \
0                                  Avatar           7.2       11800
1  Pirates of the Caribbean: At World's End           6.9        4500
2                                 Spectre           6.3        4466
3                    The Dark Knight Rises           7.6        9106
4                             Spider-Man 3           5.9        3576

   director_id  year month       day
0         4762  2009   Dec  Thursday
1         4763  2007   May  Saturday
2         4764  2015   Oct    Monday
3         4765  2012   Jul    Monday
4         4767  2007   May   Tuesday
```

[132]:
```
directors.tail()
```

[132]:
```
         director_name    id gender
2344     Shane Carruth  7106   Male
2345  Neill Dela Llana  7107    NaN
2346       Scott Smith  7108    NaN
2347       Daniel Hsia  7109   Male
2348  Brian Herzlinger  7110   Male
```

1 Unique Count

[133]:
```
movies.title.nunique()  #Return the no of unique titles
```

[133]: 1465

[134]:
```
directors.id.nunique()
```

[134]: 2349

[135]:
```
movies.director_id.nunique()
```

[135]: 199

2 Whether all the directors in directors data is present in movies data

[136]:
```
np.all(movies.director_id.isin(directors.id))
```

[136]: True

3 Join both Movies and Directors table

[137]:
```
data = movies.merge(directors,left_on='director_id',right_on='id',how='left')
```

[138]:
```
data.head()
```

[138]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day	director_name	id_y	gender
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday	James Cameron	4762	Male
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	4763	2007	May	Saturday	Gore Verbinski	4763	Male
2	43599	245000000	107	880674609	Spectre	6.3	4466	4764	2015	Oct	Monday	Sam Mendes	4764	Male
3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106	4765	2012	Jul	Monday	Christopher Nolan	4765	Male
4	43602	258000000	115	890871626	Spider-Man 3	5.9	3576							

```
4         4767  2007    May    Tuesday          Sam Raimi  4767    Male
```

[139]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1465 entries, 0 to 1464
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id_x           1465 non-null   int64
 1   budget         1465 non-null   int64
 2   popularity     1465 non-null   int64
 3   revenue        1465 non-null   int64
 4   title          1465 non-null   object
 5   vote_average   1465 non-null   float64
 6   vote_count     1465 non-null   int64
 7   director_id    1465 non-null   int64
 8   year           1465 non-null   int64
 9   month          1465 non-null   object
 10  day            1465 non-null   object
 11  director_name  1465 non-null   object
 12  id_y           1465 non-null   int64
 13  gender         1341 non-null   object
dtypes: float64(1), int64(8), object(5)
memory usage: 160.4+ KB
```

[140]:
```
data.drop(['id_y'],axis=1,inplace=True)
```

[141]:
```
data
```

[141]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count
0	43597	237000000	150	2787965087	Avatar	7.2	11800
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500
2	43599	245000000	107	880674609	Spectre	6.3	4466
3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106
4	43602	258000000	115	890871626			
…	…	…	…	…			
1460	48363	0	3	321952			
1461	48370	27000	19	3151130			
1462	48375	0	7	0			
1463	48376	0	3	0			
1464	48395	220000	14	2040920			

```
4                                  Spider-Man 3           5.9        3576
…                                           …           …          …
1460                          The Last Waltz           7.9          64
1461                                  Clerks           7.4         755
1462                                Rampage            6.0         131
1463                                 Slacker           6.4          77
1464                             El Mariachi           6.6         238

      director_id  year month       day      director_name gender
0            4762  2009   Dec  Thursday      James Cameron   Male
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …    …     …       …                 …    …
1460         4809  1978   May    Monday    Martin Scorsese   Male
1461         5369  1994   Sep   Tuesday        Kevin Smith   Male
1462         5148  2009   Aug    Friday           Uwe Boll   Male
1463         5535  1990   Jul    Friday  Richard Linklater   Male
1464         5097  1992   Sep    Friday   Robert Rodriguez    NaN

[1465 rows x 13 columns]
```

[142]:
```
data.rename({'id_x':'movies_id'},axis=1,inplace=True)
```

[143]:
```
data
```

[143]:
```
      movies_id     budget  popularity     revenue  \
0         43597  237000000         150  2787965087
1         43598  300000000         139   961000000
2         43599  245000000         107   880674609
3         43600  250000000         112  1084939099
4         43602  258000000         115   890871626
…           …        …          …          …
1460      48363          0           3      321952
1461      48370      27000          19     3151130
1462      48375          0           7           0
1463      48376          0           3           0
1464      48395     220000          14     2040920

                                      title  vote_average  vote_count  \
0                                    Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                       The Dark Knight Rises           7.6        9106
4                                  Spider-Man 3           5.9        3576
…                                           …           …          …
```

```
1460                          The Last Waltz          7.9          64
1461                                  Clerks          7.4         755
1462                                 Rampage          6.0         131
1463                                  Slacker          6.4          77
1464                             El Mariachi          6.6         238

      director_id  year month       day      director_name gender
0            4762  2009   Dec  Thursday      James Cameron   Male
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …    …     …        …                …     …
1460         4809  1978   May    Monday    Martin Scorsese   Male
1461         5369  1994   Sep   Tuesday        Kevin Smith   Male
1462         5148  2009   Aug    Friday           Uwe Boll   Male
1463         5535  1990   Jul    Friday  Richard Linklater   Male
1464         5097  1992   Sep    Friday   Robert Rodriguez    NaN

[1465 rows x 13 columns]
```

4 Describe function will give the description of Numeric data

[144]:
```
data.describe()
```

[144]:
```
          movies_id        budget   popularity       revenue  vote_average  \
count   1465.000000  1.465000e+03  1465.000000  1.465000e+03   1465.000000
mean   45225.191126  4.802295e+07    30.855973  1.432539e+08      6.368191
std     1189.096396  4.935541e+07    34.845214  2.064918e+08      0.818033
min    43597.000000  0.000000e+00     0.000000  0.000000e+00      3.000000
25%    44236.000000  1.400000e+07    11.000000  1.738013e+07      5.900000
50%    45022.000000  3.300000e+07    23.000000  7.578164e+07      6.400000
75%    45990.000000  6.600000e+07    41.000000  1.792469e+08      6.900000
max    48395.000000  3.800000e+08   724.000000  2.787965e+09      8.300000

         vote_count  director_id         year
count   1465.000000  1465.000000  1465.000000
mean    1146.396587  5040.192491  2002.615017
std     1578.077438   258.059631     8.680141
min        1.000000  4762.000000  1976.000000
25%      216.000000  4845.000000  1998.000000
50%      571.000000  4964.000000  2004.000000
75%     1387.000000  5179.000000  2009.000000
max    13752.000000  6204.000000  2016.000000
```

5 Describe for non-numeric data

[145]:
```
data.describe(include=object)
```

[145]:
```
          title month     day     director_name gender
count      1465  1465    1465              1465   1341
unique     1465    12       7               199      2
top      Avatar   Dec  Friday  Steven Spielberg   Male
freq          1   193     654                26   1309
```

6 Changing the number into Millions, Lakhs, Thousands (Short form)

Currency Meter	
10 Lakhs	1 Million
1 Crore	10 Million
100 Crore	1 Billion

[146]:
```
data['budget']=data['budget']/1000000
```

[147]:
```
data
```

[147]:
```
      movies_id   budget  popularity     revenue  \
0         43597  237.000         150  2787965087
1         43598  300.000         139   961000000
2         43599  245.000         107   880674609
3         43600  250.000         112  1084939099
4         43602  258.000         115   890871626
…           …      …           …          …
1460      48363    0.000           3      321952
1461      48370    0.027          19     3151130
1462      48375    0.000           7           0
1463      48376    0.000           3           0
1464      48395    0.220          14     2040920

                                      title  vote_average  vote_count  \
0                                    Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                              Spider-Man 3           5.9        3576
…                                       …             …           …
1460                    The Last Waltz           7.9          64
1461                            Clerks           7.4         755
1462                            Rampage           6.0         131
```

```
1463                                Slacker           6.4          77
1464                           El Mariachi           6.6         238

      director_id  year month       day      director_name gender
0            4762  2009   Dec  Thursday      James Cameron   Male
1            4763  2007   May  Saturday     Gore Verbinski   Male
2            4764  2015   Oct    Monday         Sam Mendes   Male
3            4765  2012   Jul    Monday  Christopher Nolan   Male
4            4767  2007   May   Tuesday          Sam Raimi   Male
…             …    …     …        …                …    …
1460         4809  1978   May    Monday    Martin Scorsese   Male
1461         5369  1994   Sep   Tuesday        Kevin Smith   Male
1462         5148  2009   Aug    Friday           Uwe Boll   Male
1463         5535  1990   Jul    Friday  Richard Linklater   Male
1464         5097  1992   Sep    Friday   Robert Rodriguez    NaN

[1465 rows x 13 columns]
```

7 Find out highly rated movies and there director details

[148]:
```
a = data.loc[data.vote_average > 7]
a[['title','vote_average','vote_count','year']]
```

[148]:
```
                                        title  vote_average  vote_count  \
0                                      Avatar           7.2       11800
3                       The Dark Knight Rises           7.6        9106
14   The Hobbit: The Battle of the Five Armies           7.1        4760
16         The Hobbit: The Desolation of Smaug           7.6        4524
19                                     Titanic           7.5        7562
…                                          …             …           …
1456                             Eraserhead           7.5         485
1457                             The Mighty           7.1          51
1458                                     Pi           7.1         586
1460                        The Last Waltz            7.9          64
1461                                 Clerks           7.4         755

      year
0     2009
3     2012
14    2014
16    2013
19    1997
…     …
1456  1977
1457  1998
1458  1998
```

```
1460  1978
1461  1994

[301 rows x 4 columns]
```

8 Highly rated movies release after 2014

[149]:
```
b = data.loc[(data.vote_average > 7) & (data.year > 2014)]
```

[150]:
```
b.reset_index()
```

[150]:

	index	movies_id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day	director_name	gender
0	30	43641	190.0	102	1506249360	Furious 7	7.3	4176	4794	2015	Apr	Wednesday	James Wan	Male
1	78	43724	150.0	434	378858340	Mad Max: Fury Road	7.2	9427	4845	2015	May	Wednesday	George Miller	Male
2	106	43773	135.0	100	532950503	The Revenant	7.3	6396	4874	2015	Dec	Friday	Alejandro González Iñárritu	Male
3	162	43867	108.0	167	630161890	The Martian	7.6	7268	4779	2015	Sep	Wednesday	Ridley Scott	Male
4	312	44128	75.0	48	108145109	The Man from U.N.C.L.E.	7.1	2265	4888	2015	Aug	Thursday	Guy Ritchie	Male
5	394	44281	44.0	68	155760117	The Hateful Eight	7.6	4274	4927	2015	Dec	Friday	Quentin Tarantino	Male
6	625	44770	35.0	53	194564672	The Intern	7.1	1881	4978	2015	Sep	Thursday		
7	635	44784	40.0	48	165478348	Bridge of Spies	7.2	2583	4799	2015	Oct	Thursday		
8	808	45194	30.0	65	91709827	Southpaw	7.3	2067	5034	2015	Jun	Monday		
9	833	45293	28.0	61	201634991	Straight Outta Compton	7.7	1355	5033	2015	Aug	Thursday		
10	839	45301	28.0	57	133346506	The Big Short	7.3	2607	4925	2015	Dec	Friday		
11	1344	47181	5.0	22	24804129	Race	7.1	478	5008	2016	Feb	Friday		

```
6                 Nancy Meyers  Female
7              Steven Spielberg    Male
8                 Antoine Fuqua    Male
9                  F. Gary Gray    Male
10                   Adam McKay    Male
11               Stephen Hopkins    Male
```

9 Find the movies release on either friday's or Sunday's

[151]:
```
c = data.loc[(data.day == 'Friday') | (data.day == 'Sunday')]
```

[152]:
```
c
```

[152]:
```
      movies_id  budget  popularity    revenue  \
22        43627  200.00          35  783766341
25        43632  150.00          21  836297228
53        43672  175.00          44  264218220
61        43696   38.00           6  207283925
65        43701  160.00          21  181674817
…           …       …           …          …
1458      48335    0.06          27    3221152
1459      48359    0.00           2          0
1462      48375    0.00           7          0
1463      48376    0.00           3          0
1464      48395    0.22          14    2040920

                                  title  vote_average  vote_count  \
22                          Spider-Man 2           6.7        4321
25    Transformers: Revenge of the Fallen           6.0        3138
53                             Waterworld           5.9         992
61                The Fast and the Furious           6.6        3428
65                               Poseidon           5.5         583
…                                     …             …           …
1458                                Pi           7.1         586
1459                    George Washington           6.4          36
1462                              Rampage           6.0         131
1463                              Slacker           6.4          77
1464                           El Mariachi           6.6         238

      director_id  year month     day       director_name gender
22           4767  2004   Jun  Friday           Sam Raimi   Male
25           4788  2009   Jun  Friday         Michael Bay   Male
53           4814  1995   Jul  Friday      Kevin Reynolds    NaN
61           4810  2001   Jun  Friday           Rob Cohen   Male
65           4833  2006   May  Friday   Wolfgang Petersen   Male
…             …     …     …       …                   …      …
```

```
1458         4881  1998   Jul  Friday    Darren Aronofsky   Male
1459         5231  2000   Oct  Sunday  David Gordon Green   Male
1462         5148  2009   Aug  Friday           Uwe Boll   Male
1463         5535  1990   Jul  Friday   Richard Linklater   Male
1464         5097  1992   Sep  Friday    Robert Rodriguez    NaN

[700 rows x 13 columns]
```

10 Display top 5 Popular movies

[153]:
```
data.sort_values('popularity',ascending=False).head()
```

[153]:
```
     movies_id  budget  popularity     revenue  \
58       43692   165.0         724   675120017
78       43724   150.0         434   378858340
119      43796   140.0         271   655011224
120      43797   125.0         206   752100229
45       43662   185.0         187  1004558444

                                                title  vote_average  \
58                                       Interstellar           8.1
78                                   Mad Max: Fury Road           7.2
119  Pirates of the Caribbean: The Curse of the Bla…           7.5
120              The Hunger Games: Mockingjay - Part 1           6.6
45                                     The Dark Knight           8.2

     vote_count  director_id  year month        day      director_name gender
58        10867         4765  2014   Nov  Wednesday  Christopher Nolan   Male
78         9427         4845  2015   May  Wednesday      George Miller   Male
119        6985         4763  2003   Jul  Wednesday     Gore Verbinski   Male
120        5584         4831  2014   Nov    Tuesday   Francis Lawrence   Male
45        12002         4765  2008   Jul  Wednesday  Christopher Nolan   Male
```

11 Convert all Males directors into 0 and Female Directors to 1 in your dataframe

[154]:
```
def change_to_num(gender):
        if gender == 'Male':
            return 0
        else:
            return 1

data['gender'] = data['gender'].apply(change_to_num)
```

[155]:
```
data.head()
```

[155]:

	movies_id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day	director_name	gender
0	43597	237.0	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday	James Cameron	0
1	43598	300.0	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	4763	2007	May	Saturday	Gore Verbinski	0
2	43599	245.0	107	880674609	Spectre	6.3	4466	4764	2015	Oct	Monday	Sam Mendes	0
3	43600	250.0	112	1084939099	The Dark Knight Rises	7.6	9106	4765	2012	Jul	Monday	Christopher Nolan	0
4	43602	258.0	115	890871626	Spider-Man 3	5.9	3576	4767	2007	May	Tuesday	Sam Raimi	0

12 Find the sum of Revenue and Budget

[166]:
```
data['revenue'].sum()/1000000
```

[166]: 209866.997305

[ ]:
```

```

Day_31_261123

January 23, 2024

[17]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[18]:
```
!gdown 17tYTDPBU5hpby9t0kGd7w_-zBsbY7sEd
```

```
Downloading…
From: https://drive.google.com/uc?id=17tYTDPBU5hpby9t0kGd7w_-zBsbY7sEd
To: C:\Data\Data_science\Data Science RIA\3 Python\Codes\fruits.png

  0%|          | 0.00/4.71M [00:00<?, ?B/s]
 11%|#1        | 524k/4.71M [00:00<00:01, 2.62MB/s]
 33%|###3      | 1.57M/4.71M [00:00<00:00, 5.61MB/s]
 56%|#####5    | 2.62M/4.71M [00:00<00:00, 6.80MB/s]
 78%|#######7  | 3.67M/4.71M [00:00<00:00, 7.80MB/s]
100%|##########| 4.71M/4.71M [00:00<00:00, 7.39MB/s]
```

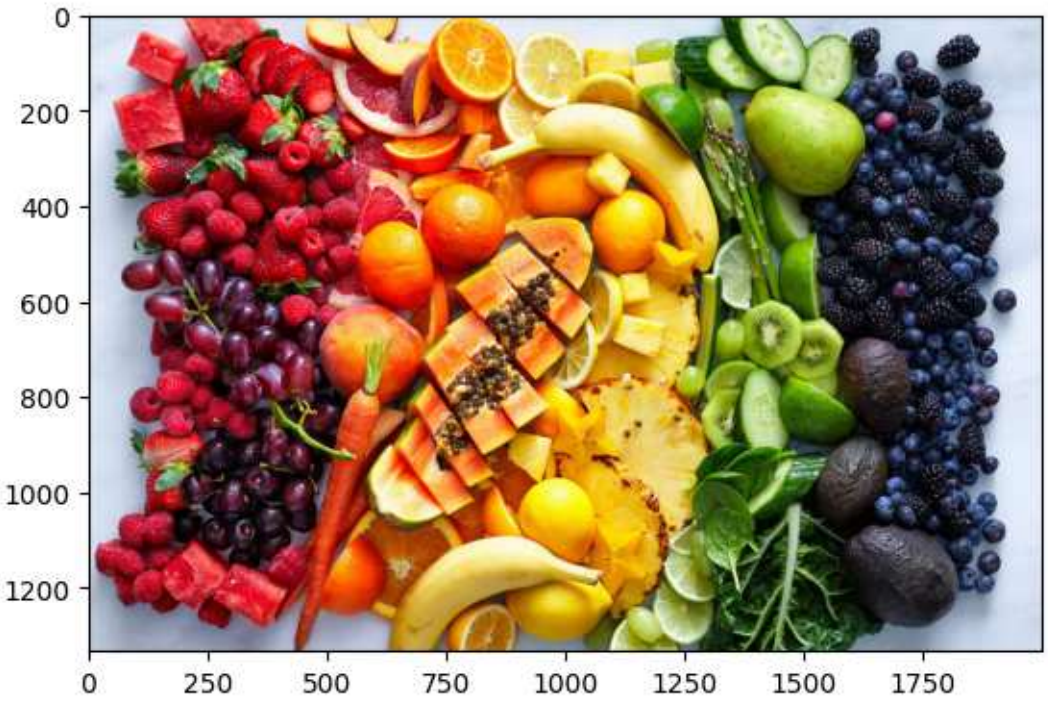
1 Standard Colors - R G B

- In these RGB they ranges from 0 to 255
- (0, 0, 0) is Black
- (255, 255, 255) is White

2 Reading Image

[19]:
```
img = plt.imread("fruits.png")
plt.imshow(img)
```

[19]:
```
<matplotlib.image.AxesImage at 0x2d70078f770>
```



3 3D Array of Fruits Image

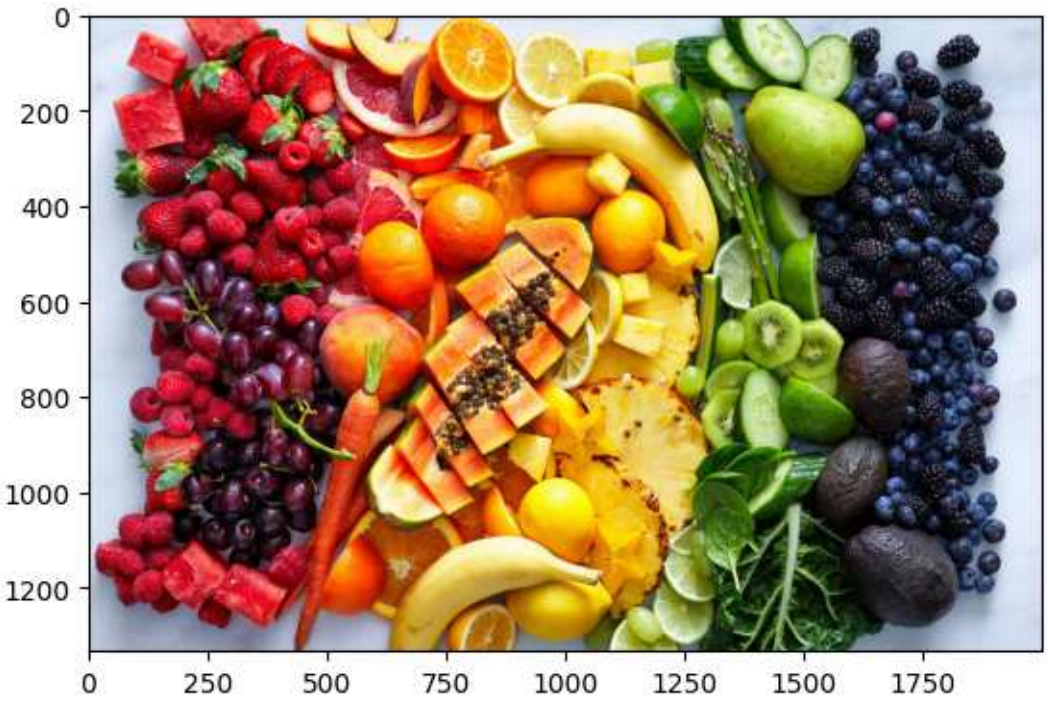
[20]:
```
img
```

[20]:
```
array([[[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        …,
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ]],

       [[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        …,
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ],
        [0.8       , 0.85490197, 0.9098039 ]],

       [[0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
        [0.8784314 , 0.9137255 , 0.972549  ],
```

```
       …,
       [0.8039216 , 0.85882354, 0.9137255 ],
       [0.8039216 , 0.85882354, 0.9137255 ],
       [0.8039216 , 0.85882354, 0.9137255 ]],

      …,

      [[0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       …,
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ]],

      [[0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       …,
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ]],

      [[0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       [0.74509805, 0.79607844, 0.87058824],
       …,
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ],
       [0.83137256, 0.8627451 , 0.9411765 ]]], dtype=float32)
```

[21]:
```
img_r = img.copy()
plt.imshow(img_r)
```

[21]:
```
<matplotlib.image.AxesImage at 0x2d700635d00>
```



[22]:
```
img_r.shape
```

[22]: (1333, 2000, 3)

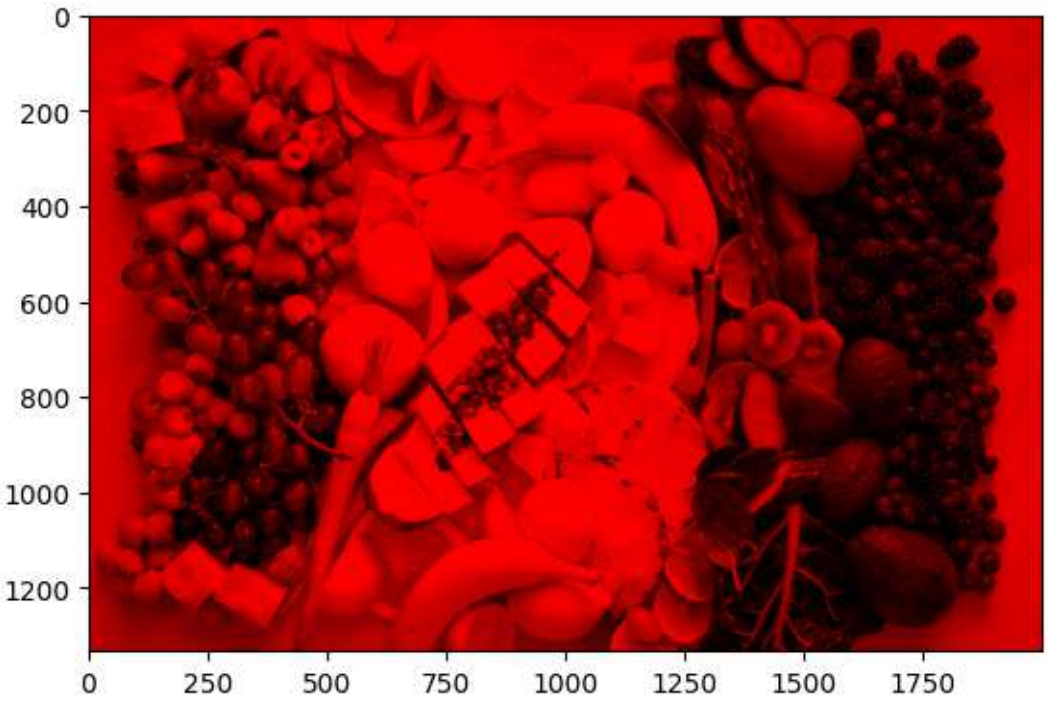
[23]:
```
img_r.ndim
```

[23]: 3

4 Changing Image colors

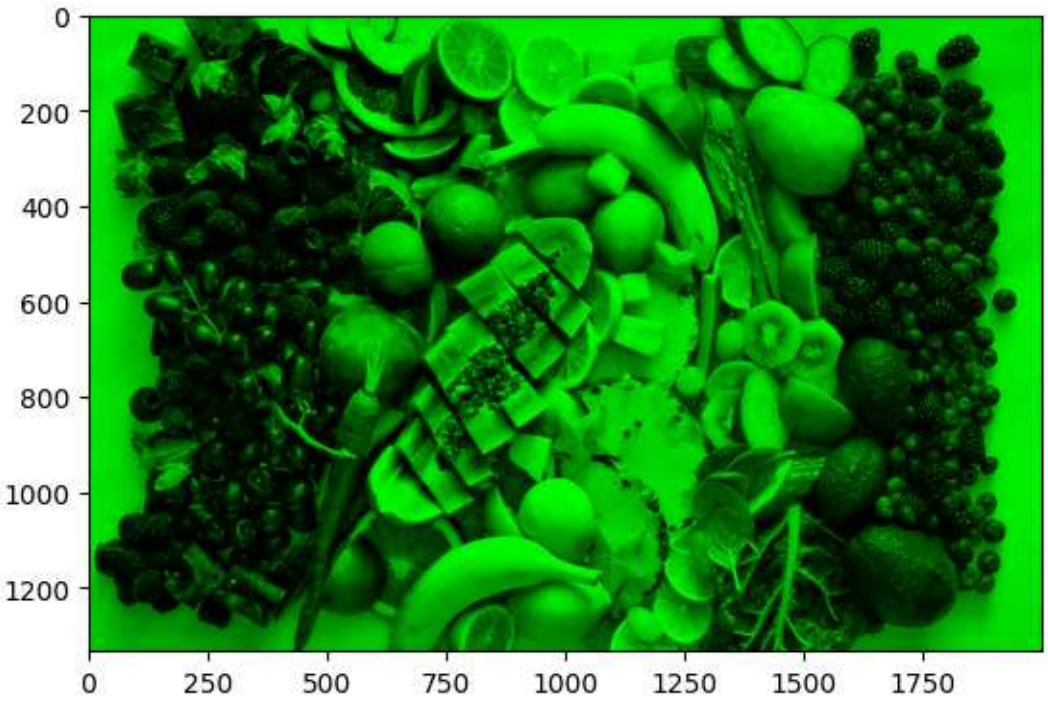
[24]:
```
img_r = img.copy()
img_r[:,:,(1,2)] = 0
plt.imshow(img_r)
```

[24]: <matplotlib.image.AxesImage at 0x2d7009c4530>



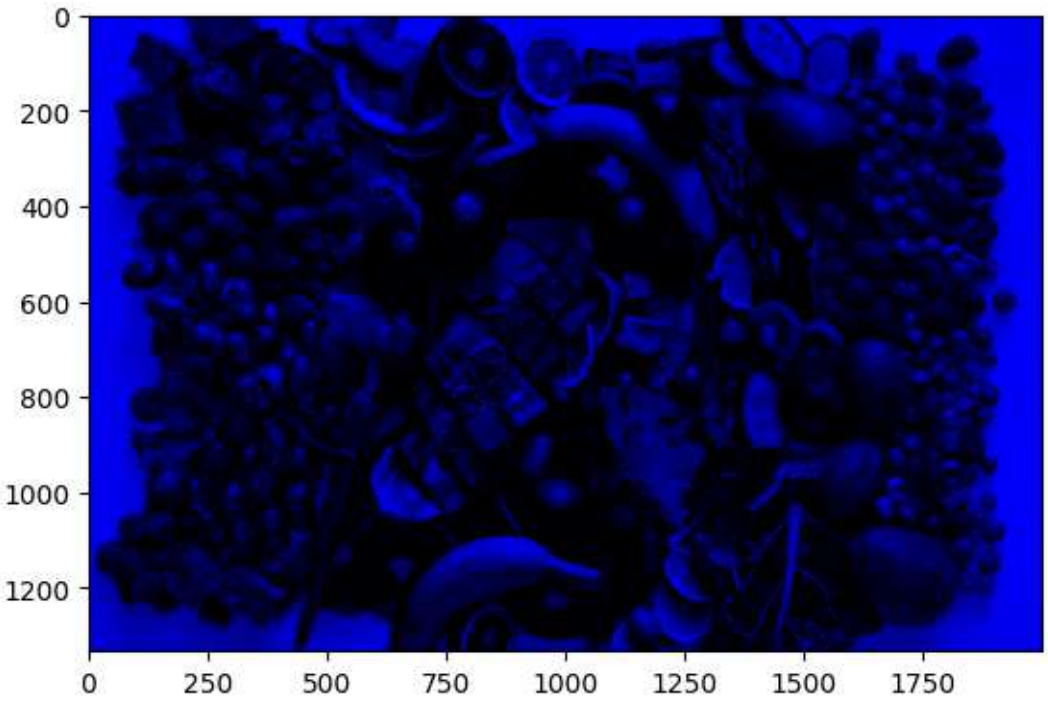
[25]:
```
img_g = img.copy()
img_g[:,:,(0,2)] = 0
plt.imshow(img_g)
```

[25]: <matplotlib.image.AxesImage at 0x2d700689610>



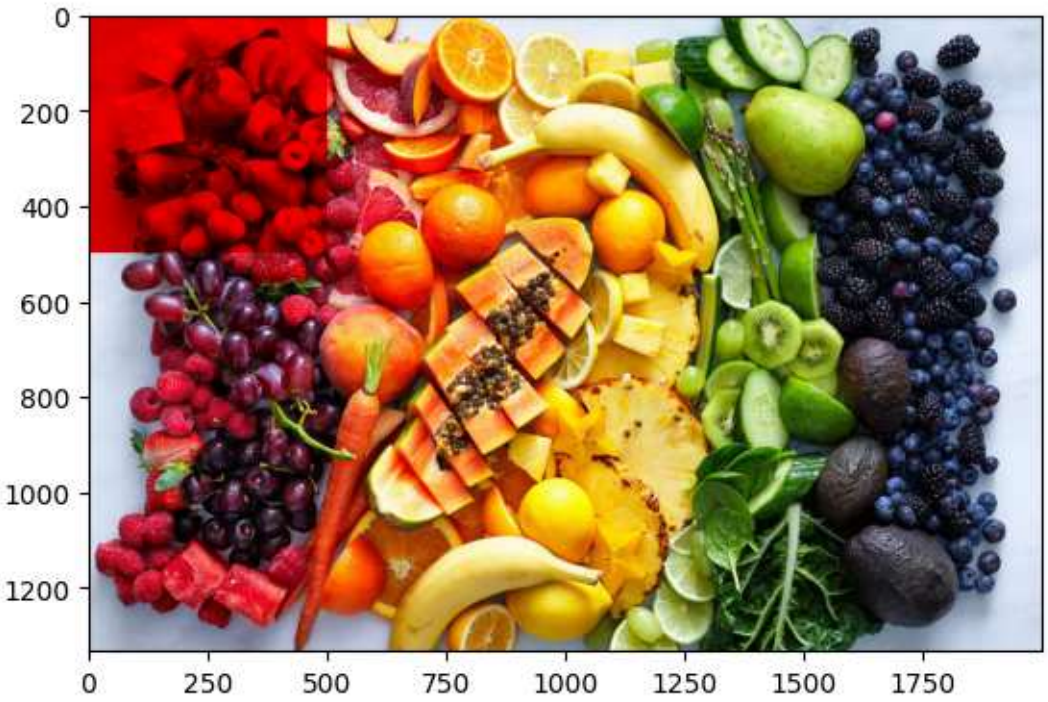
[26]:
```
img_b = img.copy()
img_b[:,:,(0,1)] = 0
plt.imshow(img_b)
```

[26]: <matplotlib.image.AxesImage at 0x2d700a7f530>



[27]:
```
img_m = img.copy()
img_m[:500,:500,(1,2)] = 0
plt.imshow(img_m)
```

[27]:
```
<matplotlib.image.AxesImage at 0x2d700ae5280>
```

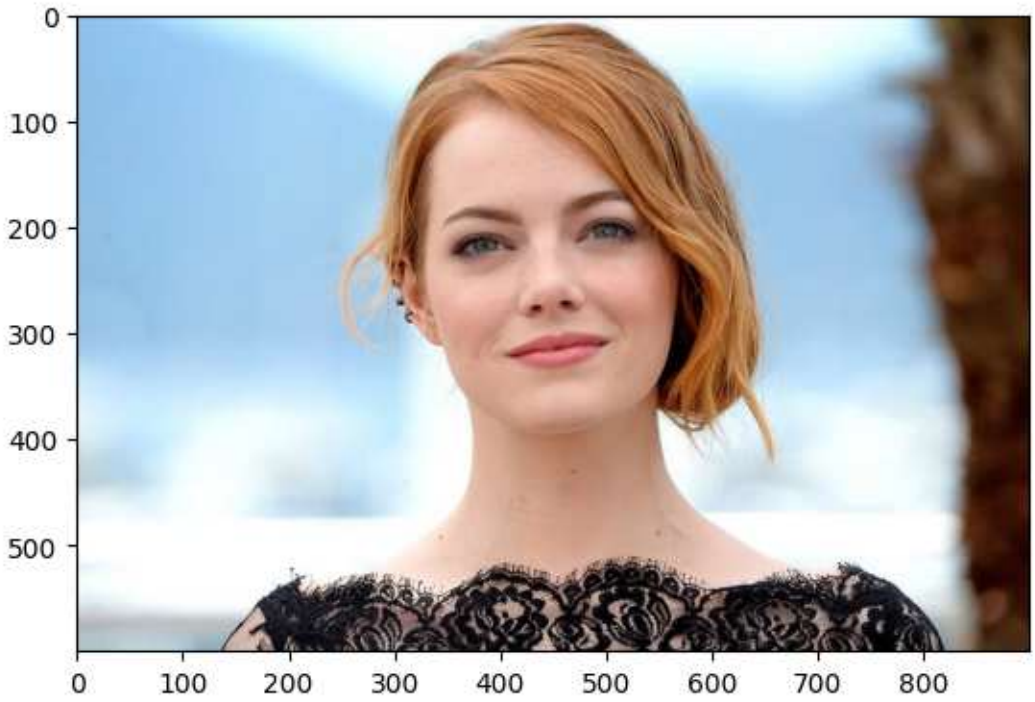


[28]:
```
!gdown 1o-8yqdTM7cfz_mAaNCi2nH0urFu7pcqI
```

```
Downloading…
From: https://drive.google.com/uc?id=1o-8yqdTM7cfz_mAaNCi2nH0urFu7pcqI
To: C:\Data\Data_science\Data Science RIA\3 Python\Codes\emma_stone.jpeg

  0%|          | 0.00/80.3k [00:00<?, ?B/s]
100%|##########| 80.3k/80.3k [00:00<00:00, 1.25MB/s]
```

[29]:
```
img_emma = plt.imread("emma_stone.jpeg")
plt.imshow(img_emma)
```

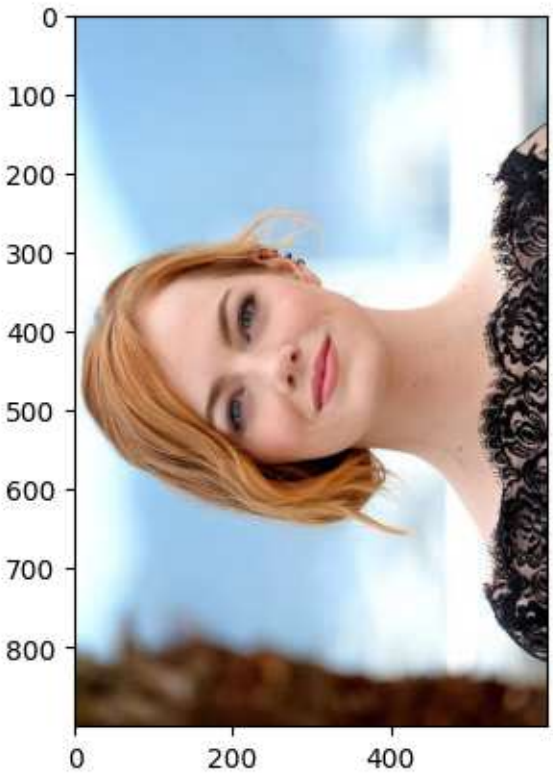
[29]:
```
<matplotlib.image.AxesImage at 0x2d700b51f40>
```



5 Rotating Image

[30]:
```
img_rotated = np.transpose(img_emma,(1,0,2))
plt.imshow(img_rotated)
```

[30]:
```
<matplotlib.image.AxesImage at 0x2d700f823f0>
```



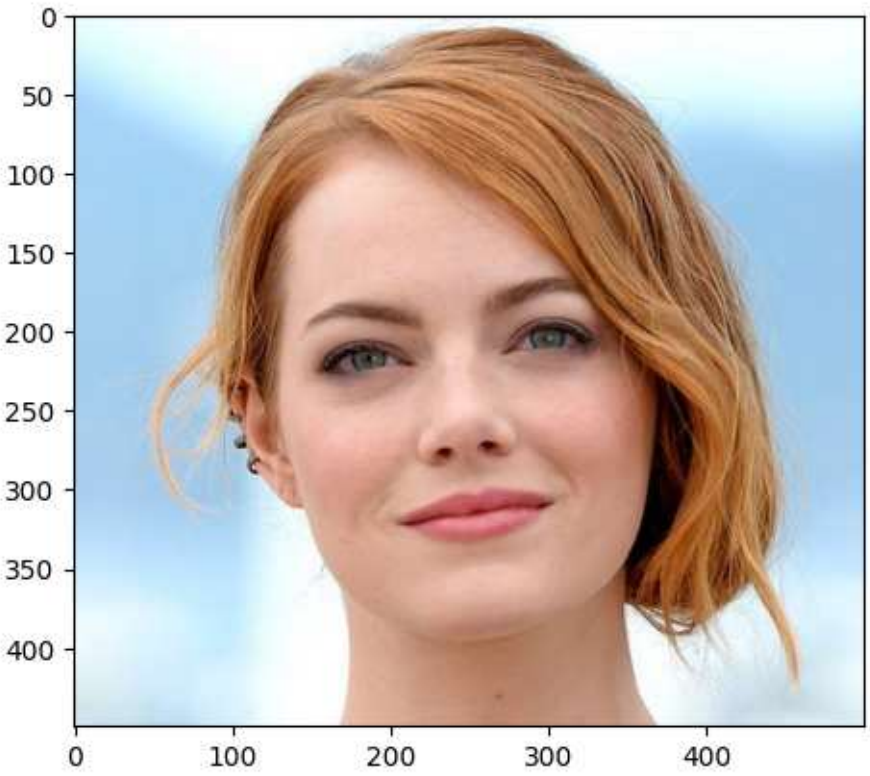
6 Flexible Rotation

[ ]:

7 Cropping Image

[31]:
```
img_cropped = img_emma[0:450,200:700]
plt.imshow(img_cropped)
```

[31]: <matplotlib.image.AxesImage at 0x2d700706450>



8 Saving the Image

[32]:
```
plt.imsave("emma_crop.jpg",img_cropped)
```