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

Read the Dataset¶ In this notebook, we will be using three CSV files: ratings.csv : userId,movieId,rating, timestamp tags.csv : userId,movieId, tag, timestamp movies.csv : movieId, title, genres

In [2]:
```
movies = pd.read_csv(r"C:\Users\arifa\Downloads\archive\movie\movie.csv")
```

In [3]:
```
movies
```

Out[3]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
27273	131254	Kein Bund für's Leben (2007)	Comedy
27274	131256	Feuer, Eis & Dosenbier (2002)	Comedy
27275	131258	The Pirates (2014)	Adventure
27276	131260	Rentun Ruusu (2001)	(no genres listed)
27277	131262	Innocence (2014)	Adventure|Fantasy|Horror

27278 rows × 3 columns

In [4]:
```
rating = pd.read_csv(r"C:\Users\arifa\Downloads\archive\movie\rating.csv")
```

In [24]:
```
rating.columns
```

Out[24]:
```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

In [9]:
```
tag = pd.read_csv(r"C:\Users\arifa\Downloads\archive\movie\tag.csv")
```

In [10]:
```
tag
```

Out[10]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	2009-04-24 18:19:40
1	65	208	dark hero	2013-05-10 01:41:18
2	65	353	dark hero	2013-05-10 01:41:19
3	65	521	noir thriller	2013-05-10 01:39:43
4	65	592	dark hero	2013-05-10 01:41:18
...	...	...	...	...
465559	138446	55999	dragged	2013-01-23 23:29:32
465560	138446	55999	Jason Bateman	2013-01-23 23:29:38
465561	138446	55999	quirky	2013-01-23 23:29:38
465562	138446	55999	sad	2013-01-23 23:29:32
465563	138472	923	rise to power	2007-11-02 21:12:47

465564 rows × 4 columns

In [11]:
```
print(type(movies))
movies.head(20)
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[11]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller
10	11	American President, The (1995)	Comedy|Drama|Romance
11	12	Dracula: Dead and Loving It (1995)	Comedy|Horror
12	13	Balto (1995)	Adventure|Animation|Children
13	14	Nixon (1995)	Drama
14	15	Cutthroat Island (1995)	Action|Adventure|Romance
15	16	Casino (1995)	Crime|Drama
16	17	Sense and Sensibility (1995)	Drama|Romance
17	18	Four Rooms (1995)	Comedy
18	19	Ace Ventura: When Nature Calls (1995)	Comedy
19	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller

In [12]:
```
tag.head()
```

Out[12]:

	userId	movieId	tag	timestamp
0	18	4141	Mark Waters	2009-04-24 18:19:40
1	65	208	dark hero	2013-05-10 01:41:18
2	65	353	dark hero	2013-05-10 01:41:19
3	65	521	noir thriller	2013-05-10 01:39:43
4	65	592	dark hero	2013-05-10 01:41:18

In [13]:
```
rating.head()
```

Out[13]:

	userId	movieId	rating	timestamp
0	1	2	3.5	2005-04-02 23:53:47
1	1	29	3.5	2005-04-02 23:31:16
2	1	32	3.5	2005-04-02 23:33:39
3	1	47	3.5	2005-04-02 23:32:07
4	1	50	3.5	2005-04-02 23:29:40

for current analysis, we will remove timestamp

In [28]:
```
rating.head()
```

Out[28]:

	userId	movieId	rating
0	1	2	3.5
1	1	29	3.5
2	1	32	3.5
3	1	47	3.5
4	1	50	3.5

In [32]:
```
tag.columns
```

Out[32]:
```
Index(['userId', 'movieId', 'tag', 'timestamp'], dtype='object')
```

In [34]:
```
del tag['timestamp']
```

In [36]:
```
tag.head()
```

Out[36]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero
2	65	353	dark hero
3	65	521	noir thriller
4	65	592	dark hero

Series

In [41]:
```
row_0 = tag.iloc[0] #Return the o index value
row_0
```

Out[41]:
```
userId              18
movieId           4141
tag        Mark Waters
Name: 0, dtype: object
```

In [43]:
```
row_0.index #Return the o index attribute
```

Out[43]:
```
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [45]:
```
row_0['userId'] #Return the userId of 0 index data
```

Out[45]:
```
18
```

In [47]:
```
'rating' in row_0
```

Out[47]:
```
False
```

In [49]:
```
row_0.name
```

Out[49]:
```
0
```

In [51]:
```
row_0 = row_0.rename('firstRow')
row_0.name
```

Out[51]:
```
'firstRow'
```

Data Frames

In [54]:
```
tag.head() #Return the top 5 value
```

Out[54]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero
2	65	353	dark hero
3	65	521	noir thriller
4	65	592	dark hero

In [58]:
```
tag.index #Return the total index no in form of slicing
```

Out[58]:
```
RangeIndex(start=0, stop=465564, step=1)
```

In [62]:
```
tag.columns
```

Out[62]:
```
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [86]:
```
tag.iloc[[0,11,500]] #return the particular index
```

Out[86]:

	userId	movieId
0	18	4141
11	65	1783
500	342	55908

Descriptive Statistics

Let's look how the ratings are distributed!

In [70]:
```
rating['rating'].describe()
```

Out[70]:
```
count    2.000026e+07
mean     3.525529e+00
std      1.051989e+00
min      5.000000e-01
25%      3.000000e+00
50%      3.500000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

In [72]:
```
rating.describe()
```

Out[72]:

	userId	movieId	rating
count	2.000026e+07	2.000026e+07	2.000026e+07
mean	6.904587e+04	9.041567e+03	3.525529e+00
std	4.003863e+04	1.978948e+04	1.051989e+00
min	1.000000e+00	1.000000e+00	5.000000e-01
25%	3.439500e+04	9.020000e+02	3.000000e+00
50%	6.914100e+04	2.167000e+03	3.500000e+00
75%	1.036370e+05	4.770000e+03	4.000000e+00
max	1.384930e+05	1.312620e+05	5.000000e+00

In [74]:
```
rating['rating'].mean() #Return the mean value
```

Out[74]:
```
3.5255285642993797
```

In [76]:
```
rating.mean()
```

Out[76]:
```
userId     69045.872583
movieId     9041.567330
rating         3.525529
dtype: float64
```

In [88]:
```
rating['rating'].min()
```

Out[88]:
```
0.5
```

In [90]:
```
rating['rating'].max()
```

Out[90]:
```
5.0
```

In [92]:
```
rating['rating'].std()
```

Out[92]:
```
1.051988919275684
```

std(): This is a pandas Series method that computes the standard deviation of the data in the Series. The standard deviation is a measure of how spread out the numbers in the column are around the mean (average). A low standard deviation means the data points are close to the mean, while a high standard deviation indicates that the data points are spread out over a wider range of values.

In [98]:
```
rating['rating'].mode() #If there are multiple values with the same highest frequen
```

Out[98]:
```
0    4.0
Name: rating, dtype: float64
```

In [96]:
```
rating.corr()
```

Out[96]:

	userId	movieId	rating
userId	1.000000	-0.000850	0.001175
movieId	-0.000850	1.000000	0.002606
rating	0.001175	0.002606	1.000000

In [100…
```
rating.corr()
```

Out[100…

	userId	movieId	rating
userId	1.000000	-0.000850	0.001175
movieId	-0.000850	1.000000	0.002606
rating	0.001175	0.002606	1.000000

Correlation Coefficient: A statistical measure that describes the extent to which two variables change together. It ranges from -1 to 1: 1: Perfect positive correlation (as one variable increases, the other also increases). -1: Perfect negative correlation (as one variable increases, the other decreases). 0: No correlation (the variables are independent of each other).

In [102…
```
filter1 = rating['rating'] > 10 #No one get the rating greater than 10
print(filter1)
filter1.any()
```

```
0           False
1           False
2           False
3           False
4           False
            ...
20000258    False
20000259    False
20000260    False
20000261    False
20000262    False
Name: rating, Length: 20000263, dtype: bool
```

Out[102… False

In [104…
```
filter2 = rating['rating'] > 0 #Return the True bcoz all rating are greater than 0
filter2.all()
```

Out[104… True

Data Cleaning: Handling Missing Data

In [107…
```
movies.shape
```

Out[107… (27278, 3)

In [113…
```
movies.isnull().any().any() #It means no NULL values!
```

Out[113… False

In [115…
```
rating.shape
```

Out[115… (20000263, 3)

In [117…
```
rating.isnull().any().any()  #It means no NULL values in rating data set!
```

Out[117… False

In [121…
```
tag.shape
```

Out[121… (465564, 3)

In [123…
```
tag.isnull().any().any() #We have some tags which are NULL
```

Out[123… True

In [127…
```
tag= tag.dropna() #used to remove any rows with missing values
```

In [129…
```
tag.isnull().any().any() #Now there is no NULL value
```

Out[129… False

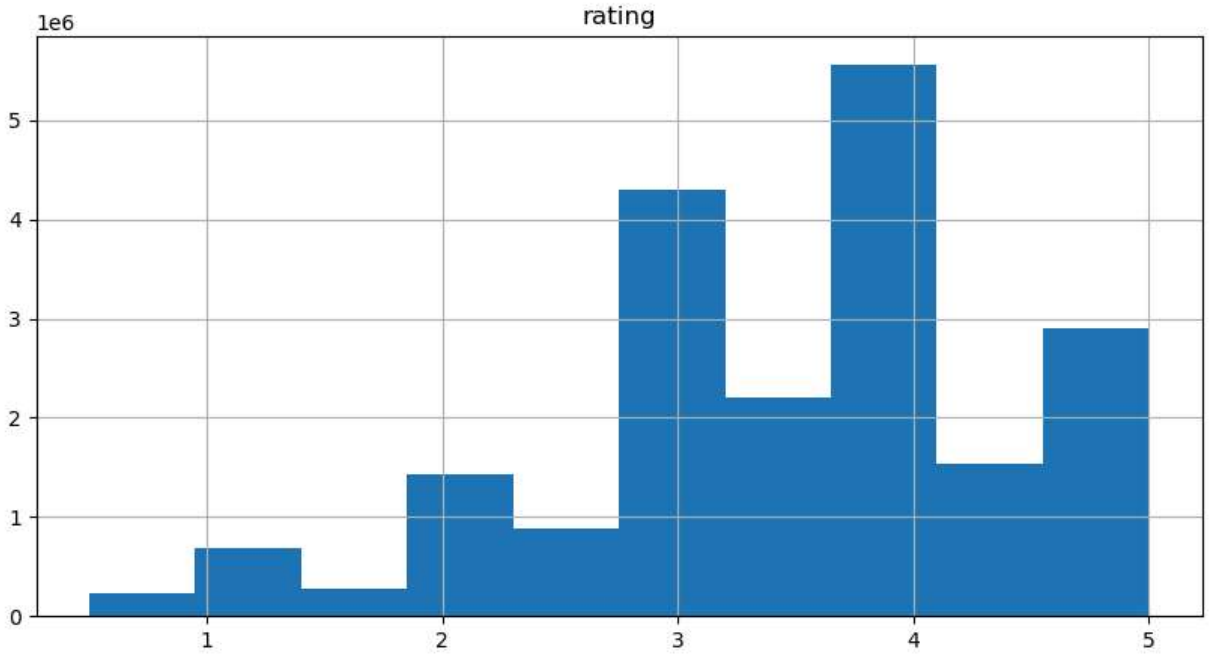
In [131…
```
tag.shape #after the cleaning the tag dataset, number of lines has been reduced
```

Out[131…
```
(465548, 3)
```

Data Visualization

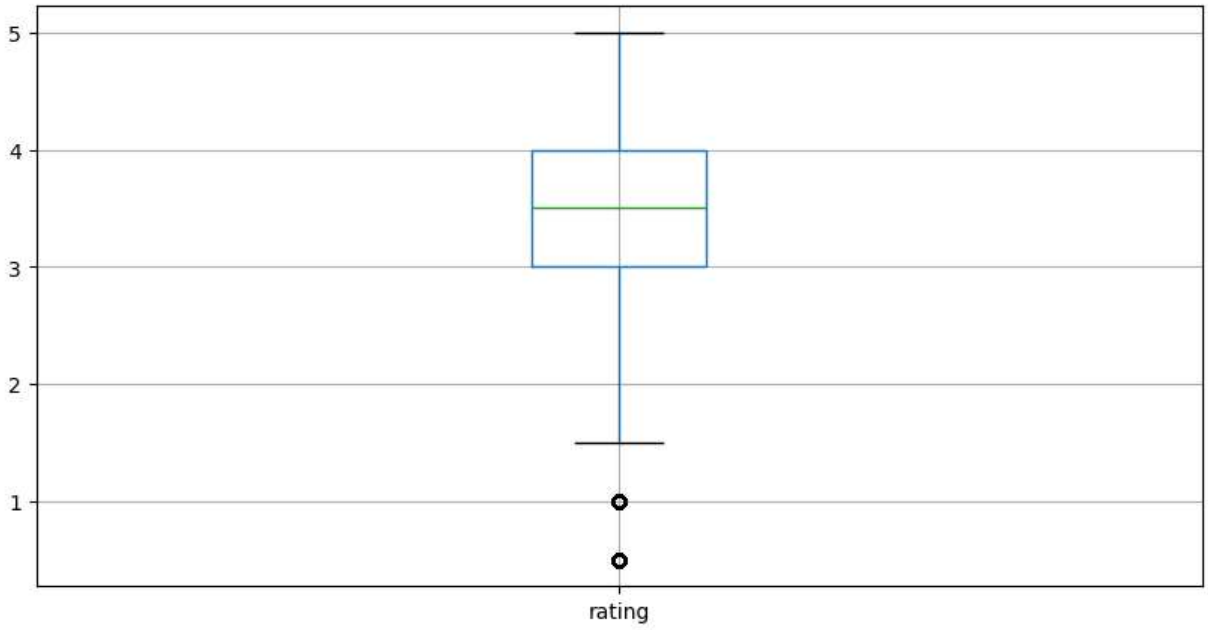
In [138…
```
%matplotlib inline

rating.hist(column='rating',figsize=(10,5))
```

Out[138…
```
array([[<Axes: title={'center': 'rating'}>]], dtype=object)
```



In [140…
```
rating.boxplot(column = 'rating',figsize=(10,5))
```

Out[140…
```
<Axes: >
```



Slicing Out Columns

In [143…
```
tag['tag'].head()
```

Out[143…
```
0      Mark Waters
1        dark hero
2        dark hero
3    noir thriller
4        dark hero
Name: tag, dtype: object
```

In [147…
```
movies[['title','genres']].head()
```

Out[147…

	title	genres
0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	Jumanji (1995)	Adventure|Children|Fantasy
2	Grumpier Old Men (1995)	Comedy|Romance
3	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	Father of the Bride Part II (1995)	Comedy

In [149…
```
rating[-10:]
```

Out[149…

	userId	movieId	rating
20000253	138493	60816	4.5
20000254	138493	61160	4.0
20000255	138493	65682	4.5
20000256	138493	66762	4.5
20000257	138493	68319	4.5
20000258	138493	68954	4.5
20000259	138493	69526	4.5
20000260	138493	69644	3.0
20000261	138493	70286	5.0
20000262	138493	71619	2.5

In [153…
```
rating[-10:-19]
```

Out[153…

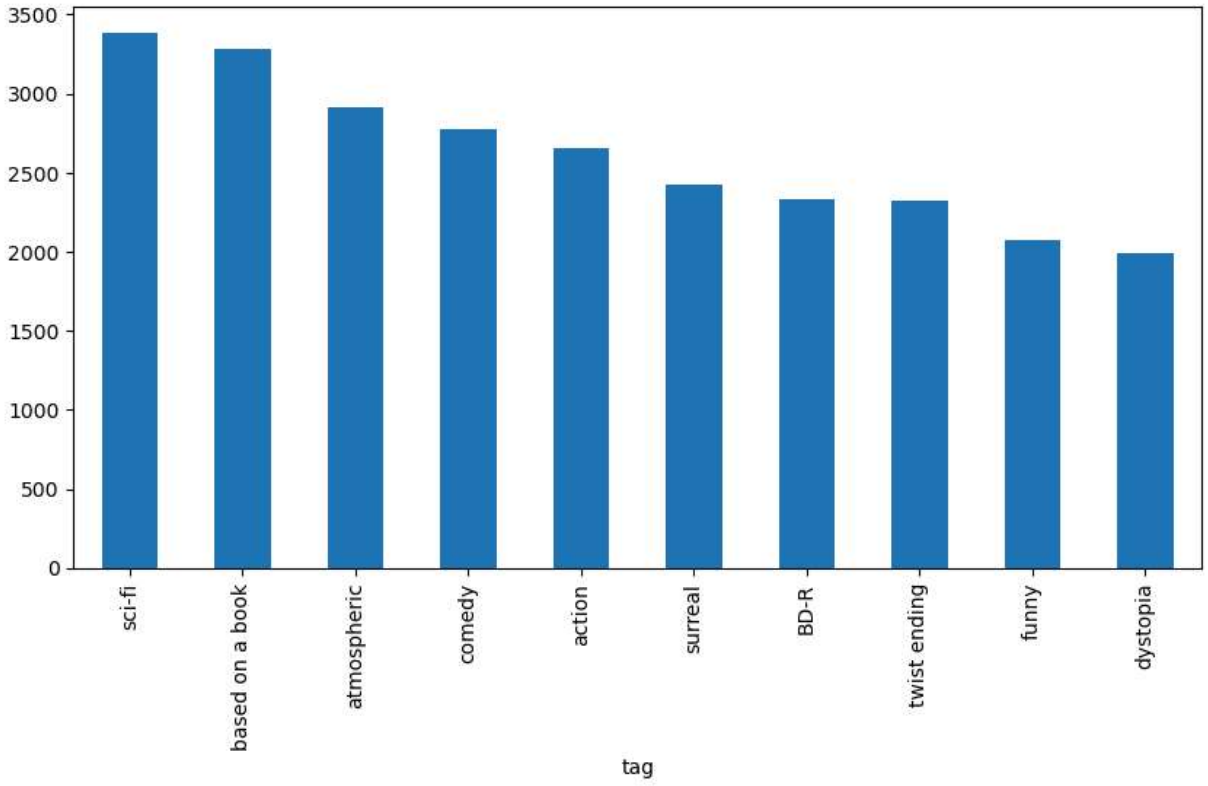
	userId	movieId	rating

In [161…
```
tag_counts = tag['tag'].value_counts()
tag_counts[-10:]
```

Out[161…
```
tag
missing child                  1
Ron Moore                      1
Citizen Kane                   1
mullet                         1
biker gang                     1
Paul Adelstein                 1
the wig                        1
killer fish                    1
genetically modified monsters  1
topless scene                  1
Name: count, dtype: int64
```

In [163…
```
tag_counts[:10].plot(kind='bar',figsize=(10,5))
```

Out[163…
```
<Axes: xlabel='tag'>
```



Filter for Selecting Rows

In [172…

```
is_highly_rated = rating['rating']>=5.0 #Return the element which rating is greater
rating[is_highly_rated][30:50]
```

Out[172…

	userId	movieId	rating
239	3	50	5.0
242	3	175	5.0
244	3	223	5.0
245	3	260	5.0
246	3	316	5.0
247	3	318	5.0
248	3	329	5.0
252	3	457	5.0
253	3	480	5.0
254	3	490	5.0
256	3	541	5.0
258	3	593	5.0
263	3	858	5.0
264	3	904	5.0
267	3	924	5.0
268	3	953	5.0
271	3	1060	5.0
272	3	1073	5.0
275	3	1084	5.0
276	3	1089	5.0

In [168…

```
is_action = movies['genres'].str.contains('Action')
movies[is_action][5:15] #this method is applied to each element in the 'genres' col
```

Out[168…

	movieId	title	genres
22	23	Assassins (1995)	Action|Crime|Thriller
41	42	Dead Presidents (1995)	Action|Crime|Drama
43	44	Mortal Kombat (1995)	Action|Adventure|Fantasy
50	51	Guardian Angel (1994)	Action|Drama|Thriller
65	66	Lawnmower Man 2: Beyond Cyberspace (1996)	Action|Sci-Fi|Thriller
69	70	From Dusk Till Dawn (1996)	Action|Comedy|Horror|Thriller
70	71	Fair Game (1995)	Action
75	76	Screamers (1995)	Action|Sci-Fi|Thriller
77	78	Crossing Guard, The (1995)	Action|Crime|Drama|Thriller
85	86	White Squall (1996)	Action|Adventure|Drama

In [170…
```
movies[is_action].head(15)
```

Out[170…

	movieId	title	genres
5	6	Heat (1995)	Action|Crime|Thriller
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller
14	15	Cutthroat Island (1995)	Action|Adventure|Romance
19	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
22	23	Assassins (1995)	Action|Crime|Thriller
41	42	Dead Presidents (1995)	Action|Crime|Drama
43	44	Mortal Kombat (1995)	Action|Adventure|Fantasy
50	51	Guardian Angel (1994)	Action|Drama|Thriller
65	66	Lawnmower Man 2: Beyond Cyberspace (1996)	Action|Sci-Fi|Thriller
69	70	From Dusk Till Dawn (1996)	Action|Comedy|Horror|Thriller
70	71	Fair Game (1995)	Action
75	76	Screamers (1995)	Action|Sci-Fi|Thriller
77	78	Crossing Guard, The (1995)	Action|Crime|Drama|Thriller
85	86	White Squall (1996)	Action|Adventure|Drama

In [174…
```
movies[is_action].head(15) #Return the top 15 action movie
```

Out[174...

	movieId	title	genres
5	6	Heat (1995)	Action|Crime|Thriller
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller
14	15	Cutthroat Island (1995)	Action|Adventure|Romance
19	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
22	23	Assassins (1995)	Action|Crime|Thriller
41	42	Dead Presidents (1995)	Action|Crime|Drama
43	44	Mortal Kombat (1995)	Action|Adventure|Fantasy
50	51	Guardian Angel (1994)	Action|Drama|Thriller
65	66	Lawnmower Man 2: Beyond Cyberspace (1996)	Action|Sci-Fi|Thriller
69	70	From Dusk Till Dawn (1996)	Action|Comedy|Horror|Thriller
70	71	Fair Game (1995)	Action
75	76	Screamers (1995)	Action|Sci-Fi|Thriller
77	78	Crossing Guard, The (1995)	Action|Crime|Drama|Thriller
85	86	White Squall (1996)	Action|Adventure|Drama

Group BY and Aggregate

The groupby() function in pandas is used to split the DataFrame into groups based on one or more columns.

Key Concept:

groupby() allows you to specify one or more columns to group the data by. It creates a "groupby object" which can then be used to apply aggregation functions (like sum, mean, count, etc.) to each group independently

Aggregate in Pandas aggregate() or agg() is used to perform specific aggregation operations on each group formed by the groupby() function. You can use standard aggregation functions (like sum, mean, min, max, etc.) or define custom aggregation functions.

Key Concept:

agg() applies one or more aggregation functions to the groups, summarizing the data in some way. You can pass a single function, a list of functions, or a dictionary specifying different functions for different columns.

In [180…
```
rating_count = rating[['movieId','rating']].groupby('rating').count()
rating_count #Groupby rating
```

Out[180…

rating	movieId
0.5	239125
1.0	680732
1.5	279252
2.0	1430997
2.5	883398
3.0	4291193
3.5	2200156
4.0	5561926
4.5	1534824
5.0	2898660

In [184…
```
average_rating = rating[['movieId','rating']].groupby('movieId').mean()
average_rating.head() #Group by movieId and also calculating same movieid rating me
                      #return the top 5 value
```

Out[184…

movieId	rating
1	3.921240
2	3.211977
3	3.151040
4	2.861393
5	3.064592

In [188…
```
movie_count = rating[['movieId','rating']].groupby('movieId').mean()
movie_count.tail()#same but return the bottom 5
```

Out[188…

movieId	rating
131254	4.0
131256	4.0
131258	2.5
131260	3.0
131262	4.0

Merge Datagrames

In [191…
```
tag.head()
```

Out[191…

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero
2	65	353	dark hero
3	65	521	noir thriller
4	65	592	dark hero

In [194…
```
movies.head()
```

Out[194…

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [200…
```
t = movies.merge(tag, on = 'movieId', how = 'inner')
t.head()
```

Out[200...

	movieId	title	genres	userId	tag
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1644	Watched
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	computer animation
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	Disney animated feature
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	Pixar animation
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1741	TÃ©a Leoni does not star in this movie

Combine aggregation , merging,and filters to get useful analytics

In [202...

```
avg_rating= rating.groupby('movieId', as_index=False).mean()
del avg_rating['userId']
avg_rating.head()
```

Out[202...

	movieId	rating
0	1	3.921240
1	2	3.211977
2	3	3.151040
3	4	2.861393
4	5	3.064592

In [207...

```
box_office = movies.merge(avg_rating, on='movieId', how='inner')
box_office.tail()
```

Out[207...

	movieId	title	genres	rating
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.0
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.0
26741	131258	The Pirates (2014)	Adventure	2.5
26742	131260	Rentun Ruusu (2001)	(no genres listed)	3.0
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.0

In [209…
```
is_highly_rated = box_office['rating'] >= 4.0
box_office[is_highly_rated][-5:]
```

Out[209…

	movieId	title	genres	rating
26737	131250	No More School (2000)	Comedy	4.0
26738	131252	Forklift Driver Klaus: The First Day on the Jo…	Comedy|Horror	4.0
26739	131254	Kein Bund für's Leben (2007)	Comedy	4.0
26740	131256	Feuer, Eis & Dosenbier (2002)	Comedy	4.0
26743	131262	Innocence (2014)	Adventure|Fantasy|Horror	4.0

In [ ]: