In [2]:
```
import pandas as pd
```

In [3]:
```
ratings=pd.read_csv(r"C:\Users\admin\Downloads\archive\rating.csv")
```

In [4]:
```
ratings.shape
```

Out[4]:
```
(20000263, 4)
```

In [5]:
```
ratings
```

Out[5]:

	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
...	...	...	...	...
20000258	138493	68954	4.5	2009-11-13 15:42:00
20000259	138493	69526	4.5	2009-12-03 18:31:48
20000260	138493	69644	3.0	2009-12-07 18:10:57
20000261	138493	70286	5.0	2009-11-13 15:42:24
20000262	138493	71619	2.5	2009-10-17 20:25:36

20000263 rows × 4 columns

In [6]:
```
movies=pd.read_csv(r"C:\Users\admin\Downloads\archive\movie.csv",sep=',')
```

In [7]:
```
movies.head(1)
```

Out[7]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

In [8]:
```
movies.shape
```

Out[8]:
```
(27278, 3)
```

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

In [11]:
```
tags.shape
```

Out[11]:
```
(465564, 4)
```

In [13]:
```
print(movies.shape)
print(ratings.shape)
print(tags.shape)
```

```
(27278, 3)
(20000263, 4)
(465564, 4)
```

In [14]:
```
print(movies.columns)
print(ratings.columns)
print(tags.columns)
```

```
Index(['movieId', 'title', 'genres'], dtype='object')
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
Index(['userId', 'movieId', 'tag', 'timestamp'], dtype='object')
```

In [15]:
```
del ratings['timestamp']
del tags['timestamp']
```

In [17]:
```
print(movies.columns)
print(ratings.columns)
print(tags.columns)
```

```
Index(['movieId', 'title', 'genres'], dtype='object')
Index(['userId', 'movieId', 'rating'], dtype='object')
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [18]:
```
tags.head(2)
```

Out[18]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero

In [19]:
```
tags.iloc[0]
#iloc use in ml,gives index location
```

Out[19]:
```
userId              18
movieId           4141
tag        Mark Waters
Name: 0, dtype: object
```

In [20]:
```
tags.head()
```

Out[20]:

	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 [ ]:
```
row_0=print
```

####### iloc[ ]: select and access data in DataFrames or Series using integer-based indexing iloc is used for purely integer-location-based indexing, meaning it selects rows based on their position in the DataFrame, not based on any label.

In [21]:
```
tags.iloc[0]  # iloc --> index location | Row indent starting from zero
```

Out[21]:
```
userId              18
movieId           4141
tag        Mark Waters
Name: 0, dtype: object
```

In [22]:
```
tags.iloc[1]
```

Out[22]:
```
userId            65
movieId          208
tag        dark hero
Name: 1, dtype: object
```

In [23]:
```
tags.iloc[2]
```

Out[23]:
```
userId            65
movieId          353
tag        dark hero
Name: 2, dtype: object
```

In [24]:
```
row_0 = tags.iloc[0]
type(row_0)
```

Out[24]:
```
pandas.core.series.Series
```

In [25]:
```
print(row_0)
##### .index : returns the index information of the DataFrame.
```

```
userId              18
movieId           4141
tag        Mark Waters
Name: 0, dtype: object
```

In [26]:
```
row_0.index
```

```
Out[26]: Index(['userId', 'movieId', 'tag'], dtype='object')
```

```
In [27]: row_0['userId']
```

```
Out[27]: 18
```

```
In [28]: row_0['movieId']
```

```
Out[28]: 4141
```

```
In [29]: row_0['tag']
```

```
Out[29]: 'Mark Waters'
```

```
In [30]: row_0.name
```

```
Out[30]: 0
```

```
In [31]: row_0 = row_0.rename('firstRow') # here we named 0th row as firstRow
         row_0
```

```
Out[31]: userId              18
         movieId           4141
         tag        Mark Waters
         Name: firstRow, dtype: object
```

```
In [32]: tags.head(2)
```

Out[32]:

	userId	movieId	tag
0	18	4141	Mark Waters
1	65	208	dark hero

```
In [33]: tags.index
```

```
Out[33]: RangeIndex(start=0, stop=465564, step=1)
```

```
In [34]: tags.columns
```

```
Out[34]: Index(['userId', 'movieId', 'tag'], dtype='object')
```

```
In [35]: #### if you want to see specific values of indexs (rows)
```

```
In [36]: tags.iloc[ [0,18,500] ]
```

Out[36]:

	userId	movieId	tag
0	18	4141	Mark Waters
18	65	3052	jesus
500	342	55908	entirely dialogue

In [37]:
```
#descriptive stastics
ratings
```

Out[37]:

	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
...	...	...	...
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

20000263 rows × 3 columns

In [38]:
```
ratings['rating'].describe()
```

Out[38]:
```
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 [39]:
```
ratings.describe()
```

Out[39]:

	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 [40]:
```
ratings['rating'].mean()
```
Out[40]:
```
3.5255285642993797
```
In [41]:
```
ratings['rating'].min()
```
Out[41]:
```
0.5
```
In [42]:
```
ratings['rating'].max()
```
Out[42]:
```
5.0
```
In [43]:
```
ratings['rating'].std()
#.mode : The mode of a set of values is the value that appears most often. It can b
>axis
>
>{0 or ‘index’, 1 or ‘columns’}, default 0
>
>The axis to iterate over while searching for the mode:
>
>0 or ‘index’ : get mode of each column
>
>1 or ‘columns’ : get mode of each row.
```
Out[43]:
```
1.051988919275684
```
In [44]:
```
ratings['rating'].mode()  #mode: occured most times
```
Out[44]:
```
0    4.0
Name: rating, dtype: float64
```

.corr() function is used to calculate the correlation between columns in a DataFrame.

Correlation is a measure of how closely two variables move together.

> The correlation coefficient is a number between -1 and 1:
>
> 1 means a perfect positive correlation.
>
> -1 means a perfect negative correlation.
>
> 0 means no correlation.

If you have a DataFrame with several columns of numerical data, .corr() will return a correlation matrix showing the correlation coefficients for each pair of columns.

In [46]:
```
ratings.corr()
```

Out[46]:

	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 [47]:
```
#### When you call .any() on a Series, it checks if any of the elements are True. I
```

In [48]:
```
filter1 = ratings['rating'] > 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[48]:
```
False
```

In [49]:
```
#### .all() function is used to check if all elements in a Series or DataFrame meet
```

In [50]:
```
filter2 = ratings['rating'] >0
print(filter2)
filter2.any()
```

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

```
Out[50]: True
```

```
In [51]: #data cleaning handles missing data
```

```
In [52]: movies.shape
```

```
Out[52]: (27278, 3)
```

```
In [53]: movies.isnull().any().any() # FALSE means No NULL values
```

```
Out[53]: False
```

```
In [54]: ratings.shape
```

```
Out[54]: (20000263, 3)
```

```
In [55]: ratings.isnull().any().any()  #FALSE means No NULL values !
```

```
Out[55]: False
```

```
In [56]: tags.isnull().any().any()  # TRUE means have some NULL values !
```

```
Out[56]: True
```

```
In [57]: #### to drop null values: .dropna( ) --> Remove missing values.
```

```
In [58]: tags = tags.dropna()
```

```
In [59]: tags.isnull().any().any()
```

```
Out[59]: False
```

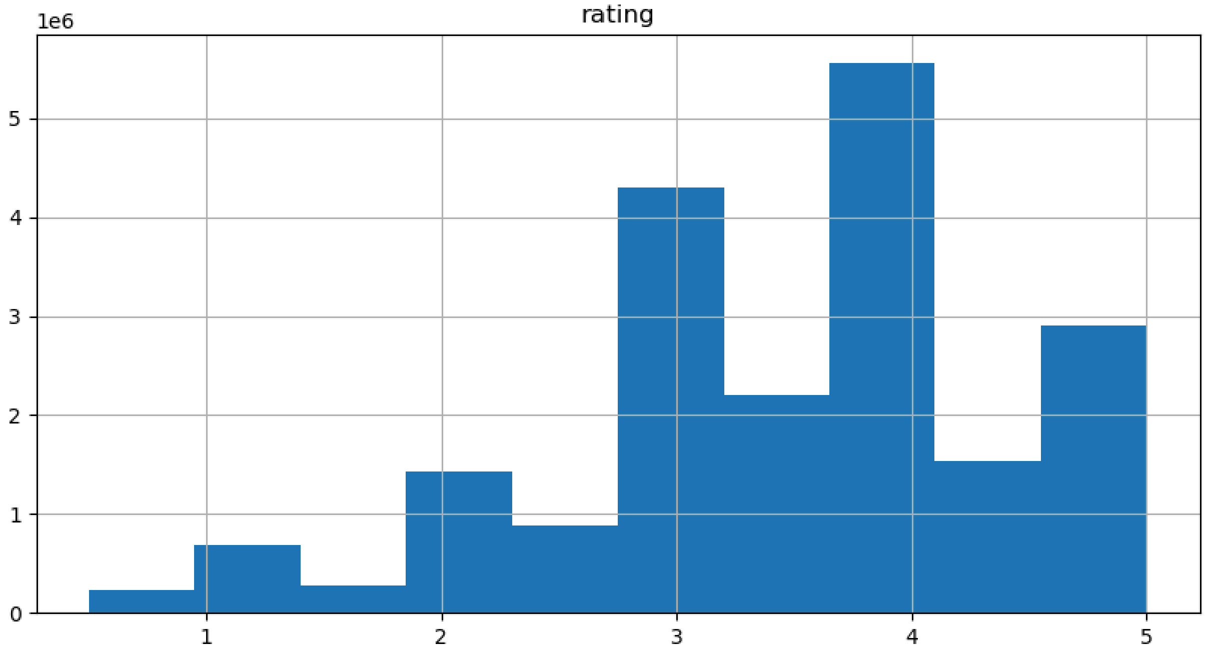
```
In [60]: tags.shape
```

```
Out[60]: (465548, 3)
```

```
In [61]: #Data visualization
```

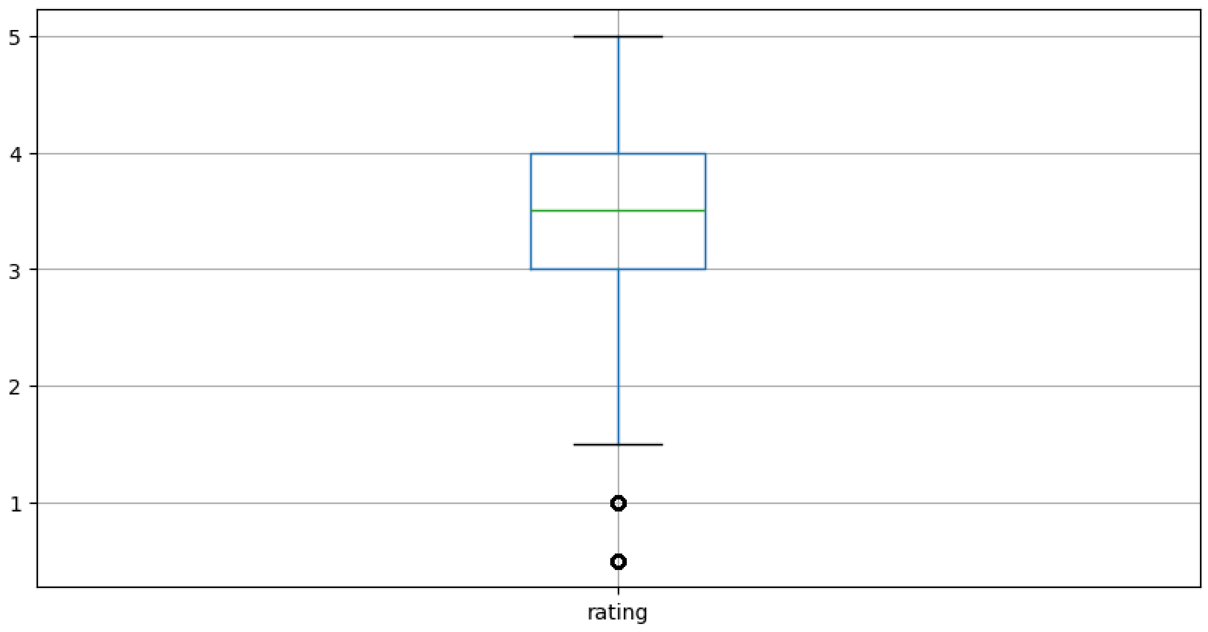
```
In [70]: %matplotlib inline
In [69]: ratings.hist(column='rating', figsize=(10,5))
```

```
Out[69]: array([[<Axes: title={'center': 'rating'}>]], dtype=object)
```



```
In [68]: ratings.boxplot(column='rating', figsize=(10,5))
```

```
Out[68]: <Axes: >
```



```
In [71]: ##Slicing out columns
```

```
In [72]: tags['tag'].head()
```

Out[72]:
```
0      Mark Waters
1        dark hero
2        dark hero
3    noir thriller
4        dark hero
Name: tag, dtype: object
```

In [73]:
```
movies[['title','genres']].head()
```

Out[73]:

	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 [74]:
```
ratings[-10:]  # last 10 indexes/rows
```

Out[74]:

	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 [75]:
```
### value_counts() function returns object containing counts of unique values.
```

In [76]:
```
tag_count = tags['tag'].value_counts()
tag_count
```

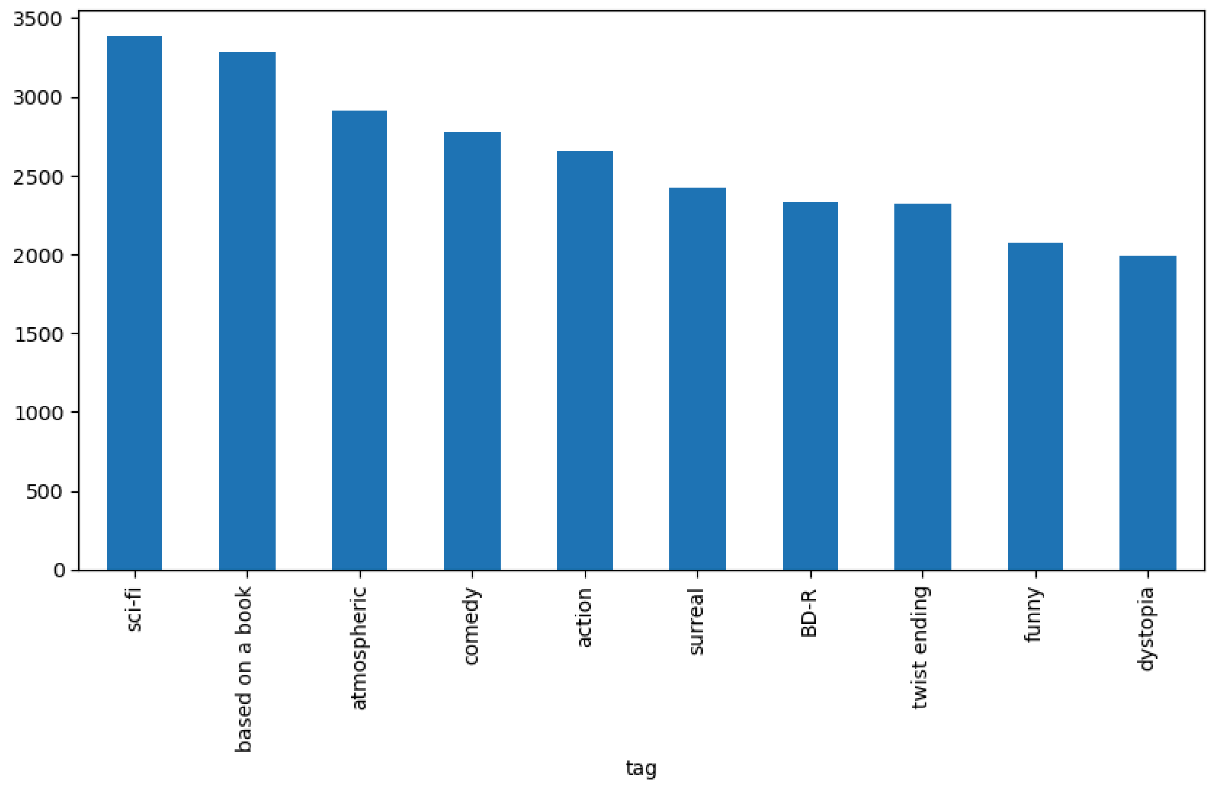
```
Out[76]:  tag
          sci-fi                             3384
          based on a book                    3281
          atmospheric                        2917
          comedy                             2779
          action                             2657
                                             ...
          Paul Adelstein                        1
          the wig                               1
          killer fish                           1
          genetically modified monsters         1
          topless scene                         1
          Name: count, Length: 38643, dtype: int64
```

```
In [77]:  tag_count[-10:]
```

```
Out[77]:  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 [78]:  tag_count[:10].plot(kind='bar', figsize=(10,5))
          #tag_count.head(10) selects the top 10 values from the tag_count series.
          #plot(kind='bar') creates a bar plot.
          #figsize=(10, 5) sets the size of the plot.
```

```
Out[78]:  <Axes: xlabel='tag'>
```


3500
3000
2500
2000
1500
1000
500
0
sci-fi
based on a book
atmospheric
comedy
action
surreal
BD-R
twist ending
funny
dystopia
tag

```
In [ ]:
```