In [4]:
```
#https://www.kaggle.com/code/harunshimanto/pandas-with-data-science-ai

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as snb
```

In [6]:
```
movie_data=pd.read_csv("A:\LLM\movie.csv")
movie_data
```

Out[6]:

	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 [9]:
```
rating_data=pd.read_csv(r"A:\LLM\rating.csv")
rating_data
```

Out[9]:

	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 [10]:
```
tag_data=pd.read_csv(r"A:\LLM\tag.csv")
tag_data
```

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 [17]:
```
#to check number of rows and columns
movie_data.shape
```

Out[17]:
```
(27278, 3)
```

In [12]:
```
#to check number of rows and columns
rating_data.shape
```

Out[12]:
```
(20000263, 4)
```

In [13]:
```
#to check number of rows and columns
tag_data.shape
```

Out[13]:
```
(465564, 4)
```

In [14]:
```
#to check columns names
movie_data.columns
```

Out[14]:
```
Index(['movieId', 'title', 'genres'], dtype='object')
```

In [15]:
```
#to check columns names
rating_data.columns
```

Out[15]:
```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

In [16]:
```
#to check columns names
movie_data.columns
```

Out[16]:
```
Index(['movieId', 'title', 'genres'], dtype='object')
```

In [18]:
```
#to check top 5 rows
movie_data.head()
```

Out[18]:

	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 [19]:
```
#to check top 5 rows
rating_data.head()
```

Out[19]:

	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

In [20]:
```
#to check top 5 rows
tag_data.head()
```

Out[20]:

	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 [21]:
```
#to check bottom 5 rows
tag_data.tail()
```

Out[21]:

	userId	movieId	tag	timestamp
465559	138446	55999	dragged	2013-01-23 23:29:32
465560	138446	55999	Jason Bateman	2013-01-23 23:29:38

	userId	movieId	tag	timestamp
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

In [22]:
```
#to check bottom 5 rows
movie_data.tail()
```

Out[22]:

	movieId	title	genres
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

In [23]:
```
#to check bottom 5 rows
rating_data.tail()
```

Out[23]:

	userId	movieId	rating	timestamp
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

In [32]:
```
#how to delete particular column
#To delete particular column make sure you are using del keyowrd and square bracket

rating_data.tail()

del rating_data['timestamp']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, metho
d, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngin
e.get_loc()

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngin
e.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.g
et_item()
```

```
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'timestamp'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
C:\Users\GOURAV~1\AppData\Local\Temp/ipykernel_7948/787852989.py in <module>
      3 rating_data.tail()
      4
----> 5 del rating_data['timestamp']
      6
      7 rating_data.tail()

~\anaconda3\lib\site-packages\pandas\core\generic.py in __delitem__(self, key)
   3961             # there was no match, this call should raise the appropriate
   3962             # exception:
-> 3963             loc = self.axes[-1].get_loc(key)
   3964             self._mgr = self._mgr.idelete(loc)
   3965

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'timestamp'
```

In [33]:

```
#After removing columns verify data
rating_data.tail()
```

Out[33]:

	userId	movieId	rating
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 [34]:

```
tag_data.head()
```

Out[34]:

	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 [35]:
```
del tag_data['timestamp']
```

In [36]:
```
tag_data.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

In [37]:
```
rating_data.head()
```

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

In [38]:
```
#if you want to check mean value, median value and count, standard deviation and 25%
rating_data.describe()
```

Out[38]:

	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 [41]:
```
#how to apply particular mean, max and min function

rating_data['rating'].mean()
```

Out[41]:
```
3.5255285642993797
```

In [42]:
```
#how to check min value
```

```
rating_data['rating'].min()
```

Out[42]:
```
0.5
```

In [43]:
```
#how to check maxiumn value

rating_data['rating'].max()
```

Out[43]:
```
5.0
```

In [44]:
```
#how to calculate standard deviation

rating_data['rating'].std()
```

Out[44]:
```
1.051988919275684
```

In [45]:
```
#how to clauclate mode for particular columns

rating_data['rating'].mode()
```

Out[45]:
```
0    4.0
dtype: float64
```

In [46]:
```
rating_data.head()
```

Out[46]:

	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 [47]:
```
#for example I want all movies details where rating is more than 3

filter_data=rating_data['rating']> 3
filter_data
```

Out[47]:
```
0            True
1            True
2            True
3            True
4            True
            ...
20000258     True
20000259     True
20000260    False
20000261     True
20000262    False
Name: rating, Length: 20000263, dtype: bool
```

In [48]:
```
#for example I want all movies details where rating is more than 3
#The .loc[] function in pandas is used for label-based indexing. It allows you to fi
```

```
filter_data=rating_data['rating']> 3
filter_data

rating_data.loc[filter_data]
```

Out[48]:

	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
...	...	...	...
20000256	138493	66762	4.5
20000257	138493	68319	4.5
20000258	138493	68954	4.5
20000259	138493	69526	4.5
20000261	138493	70286	5.0

12195566 rows × 3 columns

In [49]:

```
#how to validate data is null or not in particular columns
#we can use isnull function

rating_data.isnull()
```

Out[49]:

	userId	movieId	rating
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
...	...	...	...
20000258	False	False	False
20000259	False	False	False
20000260	False	False	False
20000261	False	False	False
20000262	False	False	False

20000263 rows × 3 columns

In [51]:

```
rating_data.isnull().any()

#Checks each column and returns True if at least one missing value exists in that co
```

Out[51]:
```
userId     False
movieId    False
rating     False
dtype: bool
```

In [54]:
```
rating_data.max()
```

Out[54]:
```
userId     138493.0
movieId    131262.0
rating          5.0
dtype: float64
```

In [56]:
```
#here in tag data we found the duplicate specialy in tag column
tag_data.isnull().any()
```

Out[56]:
```
userId     False
movieId    False
tag         True
dtype: bool
```

In [64]:
```
#now we want to remove the null values form the tag columns
#remove only missing values

new_tag_data=tag_data.dropna()
tag_data.shape
new_tag_data.shape
```

Out[64]:
```
(465548, 3)
```

In [58]:
```
#remove only missing values
tag_data.isnull().any()
```

Out[58]:
```
userId     False
movieId    False
tag         True
dtype: bool
```

In [65]:
```
new_tag_data.columns
```

Out[65]:
```
Index(['userId', 'movieId', 'tag'], dtype='object')
```

In [67]:
```
#to create a new column

new_tag_data['id_detail']=new_tag_data['userId']
```

```
C:\Users\GOURAV~1\AppData\Local\Temp/ipykernel_7948/2585515831.py:1: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  new_tag_data['id_detail']=new_tag_data['userId']
```

In [68]:
```
new_tag_data.columns
```

Out[68]:
```
Index(['userId', 'movieId', 'tag', 'id_detail'], dtype='object')
```

In [70]:
```
new_tag_data.head()
#now if want to remove the old column

del new_tag_data['userId']
```

In [71]:
```
new_tag_data.head()
```

Out[71]:

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

In [72]:
```
#how to select particular row in dataframes

new_tag_data[1:5]
```

Out[72]:

	movieId	tag	id_detail
1	208	dark hero	65
2	353	dark hero	65
3	521	noir thriller	65
4	592	dark hero	65

In [73]:
```
new_tag_data.head()
```

Out[73]:

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

In [76]:
```
#we can use slicing for particlur rows and we can use the iloc function for slecting
row_number_0=new_tag_data.iloc[0]
row_number_0
```

Out[76]:
```
movieId            4141
tag          Mark Waters
id_detail             18
Name: 0, dtype: object
```

In [77]:
```
row_number_5=new_tag_data.iloc[4]
row_number_5
```

Out[77]:
```
movieId            592
tag          dark hero
id_detail           65
Name: 4, dtype: object
```

In [78]:
```
#if we want to print particular column of vaule

row_number_5['movieId']
```

Out[78]:
```
592
```

In [80]:
```
#if you want to return low labels
new_tag_data.index
```

Out[80]:
```
Int64Index([     0,      1,      2,      3,      4,      5,      6,      7,
                 8,      9,
            ...
            465554, 465555, 465556, 465557, 465558, 465559, 465560, 465561,
            465562, 465563],
           dtype='int64', length=465548)
```

In [85]:
```
#if we need more than one rows and randoms rows
#but make sure to pass as a list

new_tag_data.iloc[[0,11,500]]
```

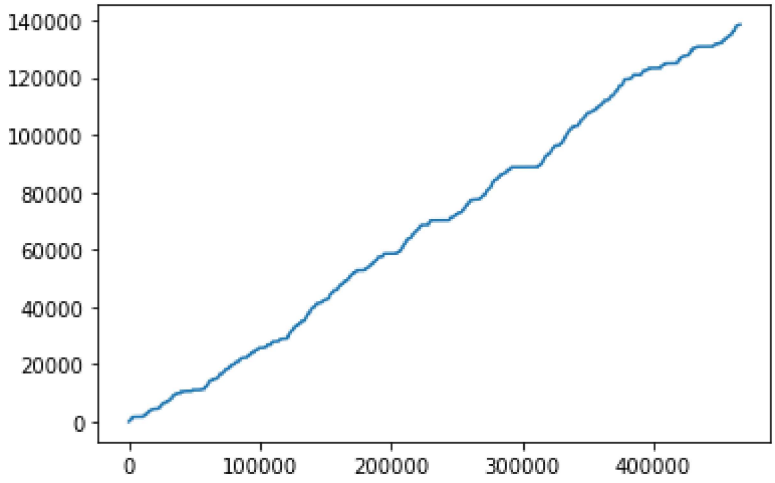
Out[85]:

	movieId	tag	id_detail
0	4141	Mark Waters	18
11	1783	noir thriller	65
500	55908	entirely dialogue	342

In [87]:
```
#doing visulaization

plt.plot(new_tag_data['id_detail'])
```

Out[87]:
```
[<matplotlib.lines.Line2D at 0x198004ca910>]
```

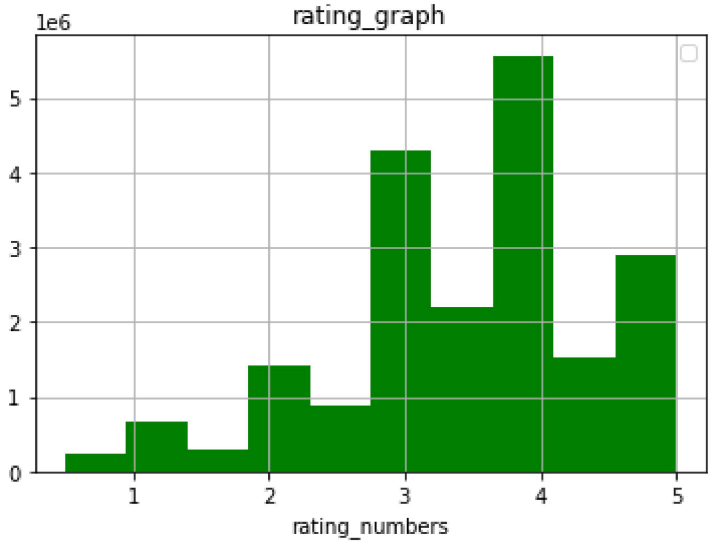


In [88]:
```
rating_data.columns
```

Out[88]:
```
Index(['userId', 'movieId', 'rating'], dtype='object')
```

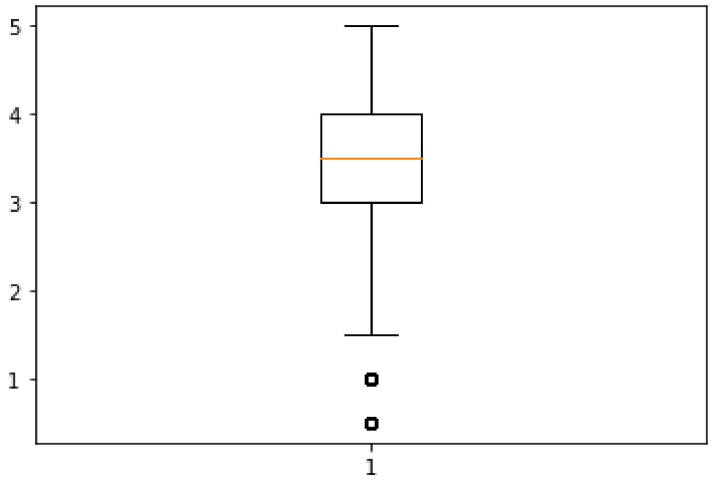
In [94]:
```
plt.hist(rating_data['rating'],color='green')
plt.title('rating_graph')
plt.grid()
plt.legend()
plt.xlabel('rating_numbers')
```

Out[94]:
```
No handles with labels found to put in legend.
Text(0.5, 0, 'rating_numbers')
```



In [95]:
```
plt.boxplot(rating_data['rating'])
```

Out[95]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x19802862580>,
  <matplotlib.lines.Line2D at 0x19802862850>],
 'caps': [<matplotlib.lines.Line2D at 0x19802862be0>,
  <matplotlib.lines.Line2D at 0x19802862f70>],
 'boxes': [<matplotlib.lines.Line2D at 0x19802831880>],
 'medians': [<matplotlib.lines.Line2D at 0x19802871340>],
 'fliers': [<matplotlib.lines.Line2D at 0x198028716d0>],
 'means': []}
```



In [96]:
```
#now select particular columns
#for example slect movieid and rating columns only
rating_data.head()
```

Out[96]:

	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 [101…

```
#columns names are case senstive

rating_data[['rating','movieId']].head()
```

Out[101…

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

In [103…

```
#if we need last 10 rows
#use slicing

rating_data[-10:]
```

Out[103…

	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 [105…

```
#total count

rating_data['rating'].count()
```

Out[105…

```
20000263
```

In [107…
```
#count the number of occurance
#Value_counts will work as group by with count function

rating_data['rating'].value_counts()
```

Out[107…
```
4.0    5561926
3.0    4291193
5.0    2898660
3.5    2200156
4.5    1534824
2.0    1430997
2.5     883398
1.0     680732
1.5     279252
0.5     239125
Name: rating, dtype: int64
```

In [108…
```
new_tag_data.head()
```

Out[108…

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

In [110…
```
group_by_data=new_tag_data['tag'].value_counts()
group_by_data
```

Out[110…
```
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: tag, Length: 38643, dtype: int64
```

In [ ]:
```
#bar graph required two values
#x axis
#y axis

plt.bar(group_by_data[:10],[1,2,3,4,5])
plt.show()
```

In [ ]:
```
plt.bar(new_tag_data['tag'], new_tag_data['id_detail'])
plt.show()
```

In [ ]:
```
rating_data.head()
```

In [ ]:
```
rating_data.head()
```

In [ ]:
```

```