week1-assignment-da-1

March 18, 2024

1 Week 1 Lab: Data Analytics - Answering Questions

we have three files read the file and check header for each

Importing libraries

[31]:
```
#### Importing the pandas library as pd
import pandas as pd
```

Check the header section of data frame along with added labels

[2]:
```
#### read the file and project header
m_cols = ['movie_id', 'title', 'genres']
movies_df = pd.read_csv('Z:/assign_wk1/movies.dat', sep=';', names=m_cols,␣
 ↪encoding='latin1')
movies_df.head(10)
```

[2]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
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's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

We can see data is loaded and labels were added for the movies.dat file

Check the header section of data frame along with added labels

[3]:
```
#### read the file and project header
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings_df = pd.read_csv('Z:/assign_wk1/ratings.dat', sep=';', names=r_cols)
ratings_df.head(10)
```

[3]:

	user_id	movie_id	rating	unix_timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
5	1	1197	3	978302268
6	1	1287	5	978302039
7	1	2804	5	978300719
8	1	594	4	978302268
9	1	919	4	978301368

We can see data is loaded and labels were added for the ratings.dat file

Check the header section of data frame along with added labels

[29]:
```
#### read the file and project header
u_cols = ['user_id','sex', 'age', 'occupation', 'zip_code']
users_df = pd.read_csv('Z:/assign_wk1/users.dat', sep=';', names=u_cols)
users_df.head(10)
```

[29]:

	user_id	sex	age	occupation	zip_code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455
5	6	F	50	9	55117
6	7	M	35	1	06810
7	8	M	25	12	11413
8	9	M	25	17	61614
9	10	F	35	1	95370

We can see data is loaded and labels were added for the users.dat file

Now we can merge the individal dataframes into a single dataframe.

[30]:
```
#### merge both dataframes with merge function
movie_ratings_df = pd.merge(movies_df, ratings_df)

lens_df = pd.merge(movie_ratings_df, users_df)
lens_df.head(20)
```

[30]:

	movie_id	title	\
0	1	Toy Story (1995)	
1	48	Pocahontas (1995)	
2	150	Apollo 13 (1995)	
3	260	Star Wars: Episode IV - A New Hope (1977)	
4	527	Schindler's List (1993)	

```
5        531                          Secret Garden, The (1993)
6        588                                       Aladdin (1992)
7        594              Snow White and the Seven Dwarfs (1937)
8        595                         Beauty and the Beast (1991)
9        608                                          Fargo (1996)
10       661                      James and the Giant Peach (1996)
11       720  Wallace & Gromit: The Best of Aardman Animatio…
12       745                                   Close Shave, A (1995)
13       783                  Hunchback of Notre Dame, The (1996)
14       914                                   My Fair Lady (1964)
15       919                               Wizard of Oz, The (1939)
16       938                                          Gigi (1958)
17      1022                                    Cinderella (1950)
18      1028                                 Mary Poppins (1964)
19      1029                                         Dumbo (1941)

                                  genres  user_id  rating  unix_timestamp sex  \
0            Animation|Children's|Comedy        1       5       978824268   F
1   Animation|Children's|Musical|Romance        1       5       978824351   F
2                                 Drama        1       5       978301777   F
3        Action|Adventure|Fantasy|Sci-Fi        1       4       978300760   F
4                              Drama|War        1       5       978824195   F
5                     Children's|Drama        1       4       978302149   F
6    Animation|Children's|Comedy|Musical        1       4       978824268   F
7           Animation|Children's|Musical        1       4       978302268   F
8           Animation|Children's|Musical        1       5       978824268   F
9                   Crime|Drama|Thriller        1       4       978301398   F
10          Animation|Children's|Musical        1       3       978302109   F
11                            Animation        1       3       978300760   F
12              Animation|Comedy|Thriller        1       3       978824268   F
13          Animation|Children's|Musical        1       4       978824291   F
14                      Musical|Romance        1       3       978301968   F
15   Adventure|Children's|Drama|Musical        1       4       978301368   F
16                             Musical        1       4       978301752   F
17          Animation|Children's|Musical        1       5       978300055   F
18             Children's|Comedy|Musical        1       5       978301777   F
19          Animation|Children's|Musical        1       5       978302205   F

    age  occupation zip_code
0     1          10    48067
1     1          10    48067
2     1          10    48067
3     1          10    48067
4     1          10    48067
5     1          10    48067
6     1          10    48067
7     1          10    48067
```

```
8     1          10    48067
9     1          10    48067
10    1          10    48067
11    1          10    48067
12    1          10    48067
13    1          10    48067
14    1          10    48067
15    1          10    48067
16    1          10    48067
17    1          10    48067
18    1          10    48067
19    1          10    48067
```

We can see data frame Movie ratings and users is merged

1.1 1. `info()` and `shape` functions with our lens_df data structure.

1.1.1 info()

The info() function is used to display the summary of the DataFrame

1.1.2 What information is returned from these functions?

Information returned by info() are The number of entries (rows) in each column. The data type of each column (e.g., integer, float, object). The number of non-null values in each column. The memory usage of the DataFrame.

This information is helpful for understanding the completeness and consistency of the dataset, identifying missing values, and determining the memory footprint of the DataFrame. It allows analysts to make informed decisions about data cleaning, preprocessing, and memory optimization.

[32]:
```
# Displaying information about the lens_df DataFrame
lens_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 10 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   movie_id        1000209 non-null  int64
 1   title           1000209 non-null  object
 2   genres          1000209 non-null  object
 3   user_id         1000209 non-null  int64
 4   rating          1000209 non-null  int64
 5   unix_timestamp  1000209 non-null  int64
 6   sex             1000209 non-null  object
 7   age             1000209 non-null  int64
 8   occupation      1000209 non-null  int64
 9   zip_code        1000209 non-null  object
```

```
dtypes: int64(6), object(4)
memory usage: 76.3+ MB
```

As per the output we can see there is no null value ie 1000209 entries in every coloumn, we have integer and object datatypes in our dataframe. we have 10 columns with labelled and entries is captured in first coloumn with total memory usage of 76.3+ MB.

1.1.3 Shape function

The shape attribute is used to determine the dimensions of the DataFrame, i.e., the number of rows and columns in a list format.

1.1.4 What information is returned from shape function?

It returns The number of rows in the DataFrame. The number of columns in the DataFrame.

This information is helpful for understanding the overall size of the dataset and its structure. It provides a quick overview of the dataset's extent and allows analysts to assess its complexity and potential computational requirements.

[33]:
```
# Displaying the dimensions
lens_df.shape
```

[33]:
```
(1000209, 10)
```

After running we can see there are 1000209 rows and 10 coloumns iin our dataframe lens_df

1.1.5 Why is this information helpful?

info() provides detailed information about the data types and completeness of each column in the DataFrame, while shape gives a concise overview of its dimensions. Together, these tools help us get a quick understanding of what's in our dataset and how big it is, which is important for working with and analyzing data effectively.

1.2 2. Answer the following questions:

1.2.1 Which movie(s) has the highest average rating?

For getting highest average movie rating, we have to group it by movie id and do mean for each group , sort it out in a descending order and print first record

1. use groupby function with movie_id, name data set and see the output

[34]:
```
# Grouping the ratings_df DataFrame by 'movie_id' column
grouped_ratings = ratings_df.groupby('movie_id')
grouped_ratings.head()
```

[34]:

	user_id	movie_id	rating	unix_timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968

```
3              1      3408       4       978300275
4              1      2355       5       978824291
…            …        …          …             …
993739      6001       138       1       956807238
996467      6016      3245       5       994453507
996637      6016      3336       3       995663888
997451      6024      3443       4       956749779
998383      6035       981       1       956712771

[17678 rows x 4 columns]
```

2. we use describe to get the statistics of each group and include mean to get average rating

[9]:
```
# Calculating the mean rating for each movie by grouping ratings_df DataFrame␣
↪by 'movie_id' and extracting the mean from the describe() function output
average_ratings = grouped_ratings['rating'].describe()['mean']
average_ratings.head(20)
```

[9]:
```
movie_id
1     4.146846
2     3.201141
3     3.016736
4     2.729412
5     3.006757
6     3.878723
7     3.410480
8     3.014706
9     2.656863
10    3.540541
11    3.793804
12    2.362500
13    3.262626
14    3.542484
15    2.458904
16    3.793255
17    4.027545
18    3.337580
19    2.480720
20    2.537500
Name: mean, dtype: float64
```

Sort in descending order

display all top rated movied in a list

[35]:
```
# Sorting the average_ratings Series in descending order to get highest rated␣
↪movies
average_ratings_sorted = average_ratings.sort_values(ascending=False)
```

```
average_ratings_sorted.head(12)
```

[35]:
```
movie_id
989     5.00
3881    5.00
1830    5.00
3382    5.00
787     5.00
3280    5.00
3607    5.00
3233    5.00
3172    5.00
3656    5.00
3245    4.80
53      4.75
Name: mean, dtype: float64
```

display low top rated movied in a list

[11]:
```
#to get tail of dat frame
average_ratings_sorted.tail(15)
```

[11]:
```
movie_id
1311    1.0
2039    1.0
1115    1.0
3312    1.0
3376    1.0
1386    1.0
1430    1.0
1142    1.0
1165    1.0
3237    1.0
826     1.0
3228    1.0
2845    1.0
3209    1.0
142     1.0
Name: mean, dtype: float64
```

using index and iloc for first record

[36]:
```
# Finding the movie with the highest average rating
highest_average_rating_movie_id = average_ratings_sorted.index[0]  # Extracting␣
 ↪the movie ID with the highest average rating
highest_average_rating = average_ratings_sorted.iloc[0]  # Extracting the␣
 ↪highest average rating
```

```
print("Movie with the highest average rating:")
print("Movie ID:", highest_average_rating_movie_id)
print("Average Rating:", highest_average_rating)
```

```
Movie with the highest average rating:
Movie ID: 989
Average Rating: 5.0
```

we have a data set contains filered average ratings , to display highest average rating movie id's - you have to sort with max() function and make it into list, then print the movie id's

[13]:
```
# Finding the highest average rating and corresponding movie(s) with the␣
↪highest average rating
highest_average_rating = average_ratings_sorted.max()  # Finding the highest␣
↪average rating
highest_rated_movies = average_ratings_sorted[average_ratings_sorted ==␣
↪highest_average_rating]  # Filtering movies with the highest average rating

# Extracting movie ID(s) of the highest rated movie(s)
highest_rated_movie_ids = highest_rated_movies.index.tolist()

# Printing the movie(s) with the highest average rating
print("Movie(s) with the highest average rating:")
print(highest_rated_movie_ids)
```

```
Movie(s) with the highest average rating:
[989, 3881, 1830, 3382, 787, 3280, 3607, 3233, 3172, 3656]
```

The above printed movie Id's has the highest average ratings

1.3 What about the movie(s) with the lowest rating?

Sort the grouped data set with min function and print the data frame to get the lowest rated movies

[37]:
```
# Finding the lowest rated movie(s) by getting the minimum rating for each movie
lowest_rated_movies = grouped_ratings.min()
print(lowest_rated_movies)
```

```
          user_id  rating  unix_timestamp
movie_id
1               1       1       956712849
2              10       1       956719238
3              26       1       956713372
4               8       1       956712120
5              26       1       956713165
…             …      …              …
3948            9       1       970946482
3949           30       1       973450177
3950          151       1       974245048
```

```
3951          173        1       974692384
3952           23        1       971348504

[3706 rows x 3 columns]
```

There are lot of low rated movies i.e 3706 with rating 1

To display lowest average rating movie id's - you have to sort with min() function and make it into list, then print the movie id's

[38]:
```
# Finding the lowest average rating and corresponding movie(s) with the lowest␣
 ↪average rating
lowest_average_rating = average_ratings_sorted.min()  # Finding the lowest␣
 ↪average rating
lowest_rated_movies = average_ratings_sorted[average_ratings_sorted ==␣
 ↪lowest_average_rating]  # Filtering movies with the lowest average rating

# Extracting movie ID(s) of the lowest rated movie(s)
lowest_rated_movie_ids = lowest_rated_movies.index.tolist()

# Printing the movie(s) with the lowest average rating
print("Movie(s) with the lowest average rating:")
print(lowest_rated_movie_ids)
```

```
Movie(s) with the lowest average rating:
[655, 3904, 2217, 843, 3493, 730, 133, 3123, 684, 895, 3651, 3202, 2213, 3460,
641, 127, 1311, 2039, 1115, 3312, 3376, 1386, 1430, 1142, 1165, 3237, 826, 3228,
2845, 3209, 142]
```

The output states the overall movie_id's which were having lowest average rating , i.e '1'

1.4 Which movie(s) has the most ratings in our dataset?

Use value_counts() function to know the count for each movie id ,

For that take ratings_count as a new data frame and for each movie_id in ratings_df, check counts and then sort in a order

[39]:
```
# Counting the number of ratings for each movie
ratings_count = ratings_df['movie_id'].value_counts()

# Sorting movies by the number of ratings in descending order
most_rating_movies = ratings_count.sort_values(ascending=False)

# Printing movies with the most ratings
print("Movies with the most ratings:")
print(most_rating_movies)
```

```
Movies with the most ratings:
movie_id
```

```
2858    3428
260     2991
1196    2990
1210    2883
480     2672
        …
3656       1
868        1
1316       1
1843       1
2909       1
Name: count, Length: 3706, dtype: int64
```

we can see that movie_id = '2858' has highest number of ratings i.e 3428.

1.5 List the 10 users who have rated the most movies?

for the data frame we will count ratings for each user and stor in new data frame with descending order gives the user_id who rated most in a order

[40]:
```
# Counting the number of ratings given by each user
ratings_count_per_user = ratings_df['user_id'].value_counts()

# Selecting the top 10 users with the most ratings
top_10_users = ratings_count_per_user.head(10)

# Printing the top 10 users who have rated the most movies
print("Top 10 users who have rated the most movies:")
print(top_10_users)
```

```
Top 10 users who have rated the most movies:
user_id
4169    2314
1680    1850
4277    1743
1941    1595
1181    1521
889     1518
3618    1344
2063    1323
1150    1302
1015    1286
Name: count, dtype: int64
```

We can see top Ten users who rated for most movies among those user 4169 is on top and he provided ratings for 2314 movies

1.6 Define 5 questions you would like to investigate within this dataset?

1.7 1. What is the distribution of movie ratings?

Covering : Histogram and Describe()

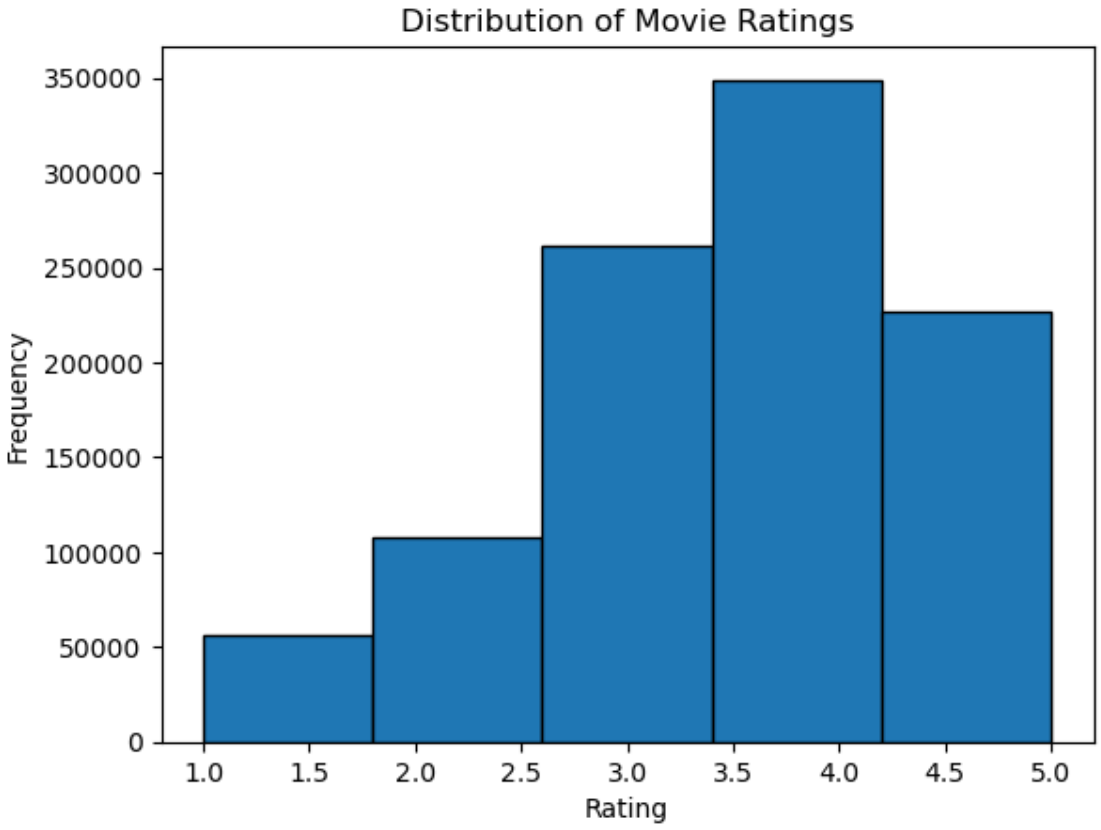
plotting the histogram to visually see the distribution of movie ratings , for this import matplot library and lebel x,y axis.

[41]:
```
# Importing matplotlib library for plotting
import matplotlib.pyplot as plt

# Creating a histogram of movie ratings with 5 bins
plt.hist(ratings_df['rating'], bins=5, edgecolor='black')

# Adding labels and title to the plot
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.title('Distribution of Movie Ratings')

# Displaying the plot
plt.show()
```



From Histogram, we can conclude there are lot of users who are rating in between 3 and 4

1.7.1 2. Apply Discriptive analysis for random 10 users who rated low

covering :info(), describe(), shape

To get random numbers use sample function
1. print new data set having random users 2. info(): display the summary of the DataFrame 3. Describe(): provides summary statistics of numerical data in a DataFrame, including count, mean, standard deviation, minimum, maximum, and quartile values. 4. Shape : determine the dimensions of the DataFrame

[42]:
```
# Taking a random sample of 10 lowest rated movies
random_sample = lowest_rated_movies.sample(n=10)

# Printing the random sample
print(random_sample)

# Printing information about the random sample
print("\nInfo:")
print(random_sample.info())

# Printing description of the random sample
print("\nDescription:")
print(random_sample.describe())

# Printing shape of the random sample
print("\nShape:")
print(random_sample.shape)
```

```
movie_id
3209    1.0
1386    1.0
133     1.0
2039    1.0
3651    1.0
730     1.0
826     1.0
3460    1.0
3493    1.0
641     1.0
Name: mean, dtype: float64

Info:
<class 'pandas.core.series.Series'>
Index: 10 entries, 3209 to 641
Series name: mean
Non-Null Count  Dtype
--------------  -----
```

```
10 non-null     float64
dtypes: float64(1)
memory usage: 160.0 bytes
None

Description:
count    10.0
mean      1.0
std       0.0
min       1.0
25%       1.0
50%       1.0
75%       1.0
max       1.0
Name: mean, dtype: float64

Shape:
(10,)
```

For the random_sample data frame , we can see the datatypes,memory, row count , entries, rating as 1, mean, std and movie id's

1.8 3.Indentify attribute which is less dependent on rating?

covering - Dataframe reshaping: removal of rows or columns

By observing data we can tell zip_code is not much dependable as users from same zipcode providing diffirent rating we can just drop it

[20]:
```
# Dropping the 'zip_code' column from the lens_df DataFrame
lens_df.drop(columns=['zip_code'], inplace=True)
```

[21]:
```
# Printing the lens_df DataFrame and displaying its information
print(lens_df)
lens_df.info()
```

```
         movie_id                                      title  \
0               1                          Toy Story (1995)
1              48                         Pocahontas (1995)
2             150                          Apollo 13 (1995)
3             260  Star Wars: Episode IV - A New Hope (1977)
4             527                    Schindler's List (1993)
…             …                                          …
1000204      3513                 Rules of Engagement (2000)
1000205      3535                     American Psycho (2000)
1000206      3536                  Keeping the Faith (2000)
1000207      3555                                U-571 (2000)
1000208      3578                          Gladiator (2000)
```

```
                                      genres  user_id  rating  \
0                  Animation|Children's|Comedy        1       5
1         Animation|Children's|Musical|Romance        1       5
2                                      Drama        1       5
3              Action|Adventure|Fantasy|Sci-Fi        1       4
4                                  Drama|War        1       5
…                                          …      …      …
1000204                      Drama|Thriller     5727       4
1000205                 Comedy|Horror|Thriller     5727       2
1000206                       Comedy|Romance     5727       5
1000207                      Action|Thriller     5727       3
1000208                         Action|Drama     5727       5

         unix_timestamp sex  age  occupation
0             978824268   F    1          10
1             978824351   F    1          10
2             978301777   F    1          10
3             978300760   F    1          10
4             978824195   F    1          10
…                 …  ..  …          …
1000204       958489970   M   25           4
1000205       958489970   M   25           4
1000206       958489902   M   25           4
1000207       958490699   M   25           4
1000208       958490171   M   25           4

[1000209 rows x 9 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 9 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   movie_id        1000209 non-null  int64
 1   title           1000209 non-null  object
 2   genres          1000209 non-null  object
 3   user_id         1000209 non-null  int64
 4   rating          1000209 non-null  int64
 5   unix_timestamp  1000209 non-null  int64
 6   sex             1000209 non-null  object
 7   age             1000209 non-null  int64
 8   occupation      1000209 non-null  int64
dtypes: int64(6), object(3)
memory usage: 68.7+ MB
```

we can see now 9 rows istead of 10 , Zip_code coloumn is droped as part of data cleaning

1.9 4. Add a coloumn which shows the length of the movie Title ?

covering - apply, lambda, aggregation and group by

First apply lambda function to our data set for coloumn title

[43]:
```
# Adding a new column 'title_length' to lens_df DataFrame, containing the␣
↪length of each movie title
lens_df['title_length'] = lens_df['title'].apply(lambda x: len(x))
```

This function adds a new colomn title_lenght to our data frame which shows the data of lenght of the tittle

[23]:
```
#print head
lens_df.head()
```

[23]:
```
   movie_id                                        title  \
0         1                            Toy Story (1995)
1        48                           Pocahontas (1995)
2       150                            Apollo 13 (1995)
3       260  Star Wars: Episode IV - A New Hope (1977)
4       527                    Schindler's List (1993)

                                genres  user_id  rating  unix_timestamp sex  \
0           Animation|Children's|Comedy        1       5       978824268   F
1  Animation|Children's|Musical|Romance        1       5       978824351   F
2                                 Drama        1       5       978301777   F
3       Action|Adventure|Fantasy|Sci-Fi        1       4       978300760   F
4                             Drama|War        1       5       978824195   F

   age  occupation  title_length
0    1          10            16
1    1          10            17
2    1          10            16
3    1          10            41
4    1          10            23
```

i can see tittle_length coloumn added to the left end of data frame

1.10 5. what is the average rating of each Genre?

covering - groupby, agg, boxplot

Lets use aggregate average method

lets do mean for ratings using agg function and group it by genre now, lets print the average rating of each genre

[44]:
```
# Calculating the average rating for each genre by grouping lens_df DataFrame␣
↪by 'genres' column and aggregating mean ratings
```

```
avg_rating_by_genre = lens_df.groupby('genres')['rating'].agg('mean')

# Printing the average rating by genre
print("\nAverage Rating by Genre:")
print(avg_rating_by_genre)
```

```
Average Rating by Genre:
genres
Action                                          3.354886
Action|Adventure                                3.676814
Action|Adventure|Animation                      4.147826
Action|Adventure|Animation|Children's|Fantasy    2.703704
Action|Adventure|Animation|Horror|Sci-Fi        3.546926
                                                   …
Sci-Fi|Thriller|War                             3.439286
Sci-Fi|War                                      4.449890
Thriller                                        3.555879
War                                             3.889001
Western                                         3.853226
Name: rating, Length: 301, dtype: float64
```

We can see most users are watching Action|Adventure|Animation and Sci-Fi|War genre which having average rating greater than 4

1.11 6.Create boxplot for ratings data

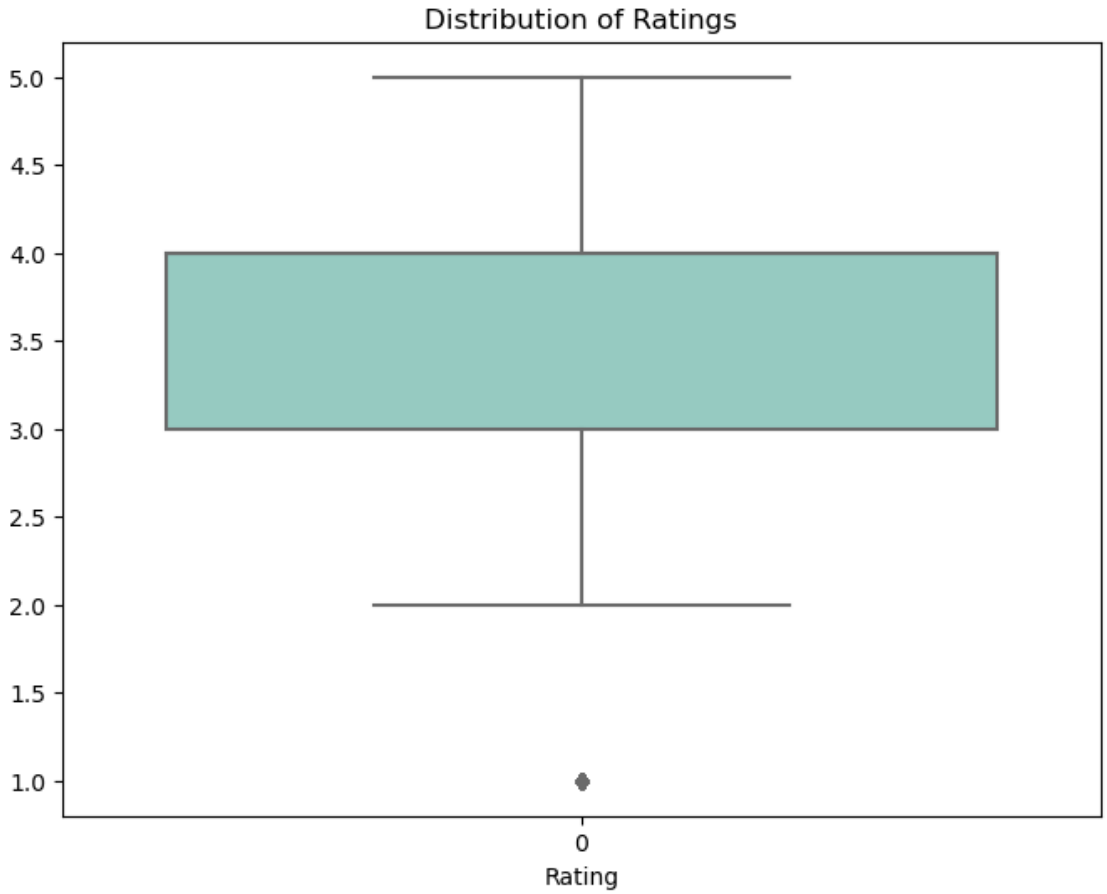
Take the ratings_df for boxplot analysis

[45]:
```
# Importing seaborn library for visualization and setting the figure size
import seaborn as sns
plt.figure(figsize=(8, 6))

# Creating a boxplot of ratings using seaborn, with a specified color palette
sns.boxplot(data=ratings_df['rating'], palette='Set3')

# Adding labels and title to the plot
plt.xlabel('Rating')
plt.title('Distribution of Ratings')

# Displaying the plot
plt.show()
```



We can see most of the ratings are between 3 and 4 which is uniformly distributed, as per outliners we can say less low rated movies compared to high rated movies

1.12 7. Conditional formating

use data frame by groupimg with tittle and agg min,max,mean,median for ratings applying conditional formatting by highlighting min and max values for the attributes agg applied

[46]:
```
# Importing pandas library
import pandas as pd

# Grouping lens_df DataFrame by 'title' and calculating mean, median, min, and␣
 ↪max ratings for each movie
rating_stats = lens_df.groupby(['title']).agg({'rating': ['mean', 'median',␣
 ↪'min', 'max']})

# Creating styled_rating_stats DataFrame with highlighting max and min values
styled_rating_stats = (rating_stats
                       .style
```

```
                        .highlight_max(color='green')
                        .highlight_min(color='red'))

# Displaying styled_rating_stats DataFrame with highlighted max and min values
styled_rating_stats
```

[46]: <pandas.io.formats.style.Styler at 0x229e0dea650>

We can see min, max values are highlighted in our data set

2 summary:

1. **Exploratory Data Analysis (EDA)**:
   - We began by examining the structure of the DataFrame, which consists of ten columns: 'movie_id', 'title', 'genres', 'user_id', 'rating', 'unix_timestamp', 'sex', 'age', 'occupation', and 'zip_code'. This initial step provided us with an overview of the dataset's attributes.
2. **Questions for Investigation**:
   - To delve into the dataset, we formulated five questions for analysis, leveraging functions such as histograms, boxplots, info(), describe(), shape, removal of rows or columns, apply(), lambda(), aggregation (.agg), groupby(), and conditional formatting in a DataFrame. These questions aimed to uncover insights into various aspects of the dataset, ranging from data distribution to demographic patterns.
3. **Tasks Undertaken**:
   - We utilized descriptive functions (info(), describe(), shape) to gain insights into the dataset's structure and characteristics. These functions allowed us to understand the data types, summary statistics, and dimensions of the DataFrame.
   - Employed data visualization techniques (histograms, boxplots) to visualize distributions and patterns within the data. These visualizations provided insights into the spread and central tendencies of numerical variables such as ratings and ages.
   - Explored DataFrame reshaping through the removal of rows or columns to focus on relevant information. This involved filtering or dropping rows/columns based on specific criteria to tailor the dataset for analysis.
   - Utilized functions such as apply() and lambda() for custom data transformations. These functions enabled us to perform element-wise operations or apply custom functions to DataFrame columns.
   - Performed aggregation using .agg() to summarize data based on specific criteria. This allowed us to calculate summary statistics (e.g., mean, median) for numerical variables grouped by categorical variables (e.g., genres, occupation).
   - Utilized groupby() to analyze data based on categorical variables. This involved grouping the DataFrame by one or more categorical variables and performing operations (e.g., aggregation, transformation) within each group.
   - Implemented conditional formatting in the DataFrame to highlight specific data points. This allowed us to visually identify important insights or outliers within the dataset.
4. **Conclusion**:
   - By performing these tasks, we aimed to gain a comprehensive understanding of the dataset, uncover insights, and facilitate further analysis and decision-making processes.

Through exploratory data analysis we were able to extract valuable insights and patterns from the dataset.

[ ]: