MIT | Academy of Engineering

EDS ACTIVITY NO. 1

NAME : ABHIJIT POPAT SAWANT

DIV : CS6

PRN : 202401080043

ROLL NO. CS6-73

BATCH : C64

- For pandas : -
  - Dataset:

```
[1]: import pandas as pd

data = {
    'title': ['The Shawshank Redemption', 'The Godfather', 'The Dark Knight', 'Pulp Fiction',
              'Schindler\'s List', 'The Lord of the Rings', 'Fight Club', 'Forrest Gump',
              'Inception', 'The Matrix'],
    'year': [1994, 1972, 2008, 1994, 1993, 2003, 1999, 1994, 2010, 1999],
    'genre': ['Drama', 'Crime', 'Action', 'Crime', 'Biography', 'Adventure', 'Drama',
              'Drama', 'Sci-Fi', 'Sci-Fi'],
    'rating': [9.3, 9.2, 9.0, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.7],
    'votes': [2343110, 1620367, 2456711, 1892623, 1389640, 1738653, 2042952, 2052217, 2352073, 1867775]
}

df = pd.DataFrame(data)
print(df)

                     title  year      genre  rating    votes
```

Grains : -

1.**head()** shows the first few rows of a DataFrame — useful for a quick look at the data.

2. **Row filtering with conditions** (e.g., df[df['rating'] > 9]) lets you extract data that meets criteria.

3.sort_values() allows you to sort your data by any column, ascending or descending.

4.groupby() with aggregation (like mean or sum) lets you analyze data by categories (e.g., genre).

5.apply() with a lambda function enables row-by-row or column-by-column transformation.

6.isnull().sum() checks for missing data in each column — key for cleaning.

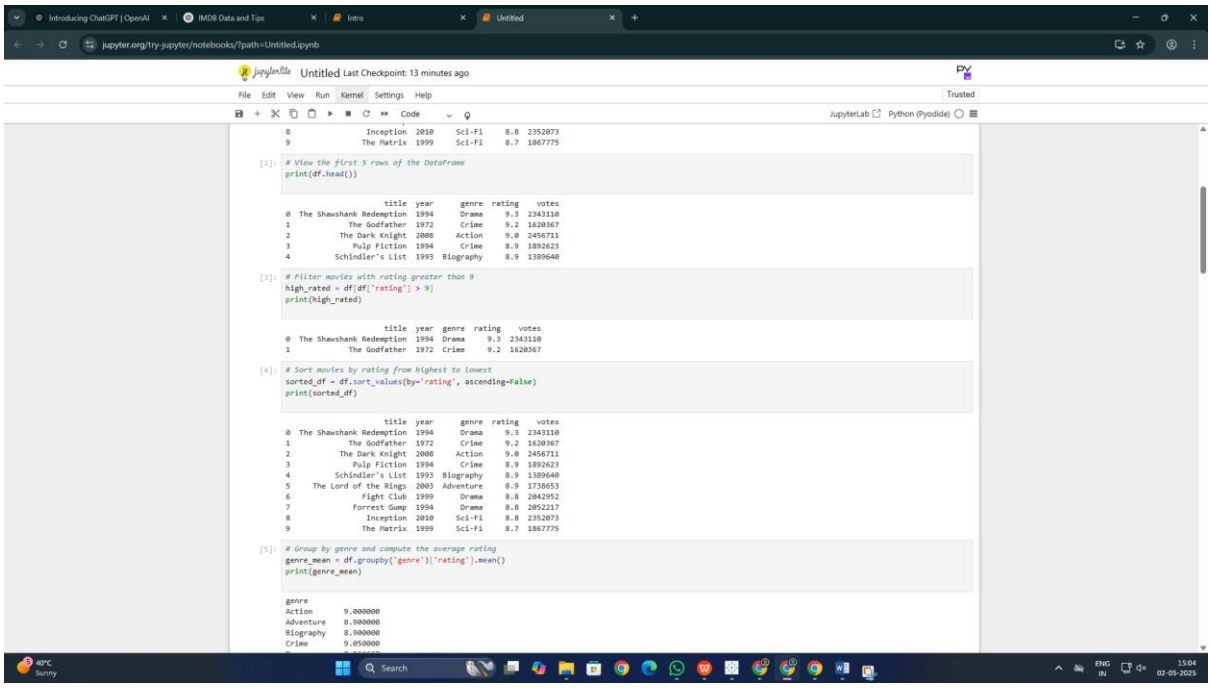
7. **Column selection** (df[['title', 'rating']]) helps you work with just the data you care about.

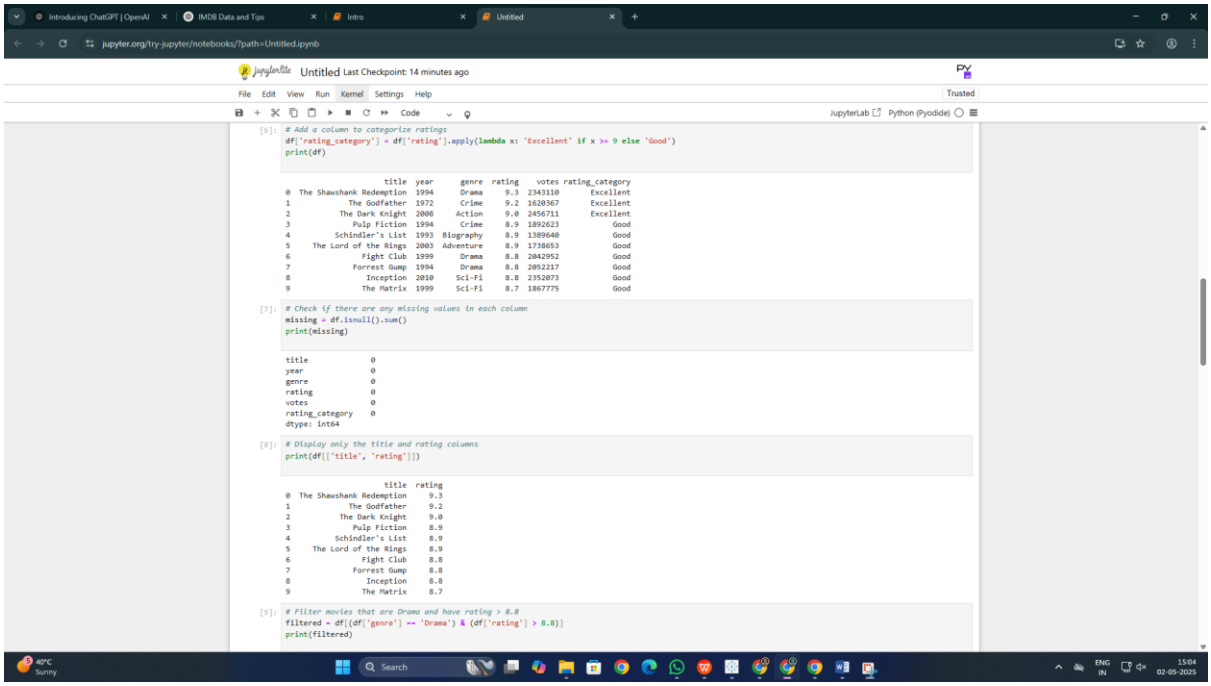
8. **Multiple condition filtering** lets you combine logical conditions with & or | (AND/OR).

9 **.reset_index()** resets the row index after filtering or sorting, which is good for clean output.

10 **.rename()** changes column names to something clearer or more consistent.

Outputs in jupyter notebook for all grains: -





Introducing ChatGPT | OpenAI | IMDB Data and Tips | Intro | Untitled

jupyter.org/try-jupyter/notebooks/?path=Untitled.ipynb

Untitled Last Checkpoint: 14 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python (Pyodide)

```
7             Forrest Gump     8.8
8               Inception     8.8
9              The Matrix     8.7
```

[9]:
```python
# Filter movies that are Drama and have rating > 8.8
filtered = df[(df['genre'] == 'Drama') & (df['rating'] > 8.8)]
print(filtered)
```

```
                      title  year genre  rating    votes rating_category
0  The Shawshank Redemption  1994 Drama     9.3  2343110       Excellent
```

[10]:
```python
# Reset the index after filtering or sorting
df_reset = df.reset_index(drop=True)
print(df_reset)
```

```
                      title  year      genre  rating    votes rating_category
0  The Shawshank Redemption  1994      Drama     9.3  2343110       Excellent
1             The Godfather  1972      Crime     9.2  1620367       Excellent
2           The Dark Knight  2008     Action     9.0  2456711       Excellent
3              Pulp Fiction  1994      Crime     8.9  1892623            Good
4            Schindler's List  1993  Biography     8.9  1389640            Good
5     The Lord of the Rings  2003  Adventure     8.9  1738653            Good
6                 Fight Club  1999      Drama     8.8  2042952            Good
7             Forrest Gump  1994      Drama     8.8  2052217            Good
8               Inception  2010     Sci-Fi     8.8  2352073            Good
9              The Matrix  1999     Sci-Fi     8.7  1867775            Good
```

[11]:
```python
# Rename columns to more descriptive names
df_renamed = df.rename(columns={'title': 'movie_title', 'year': 'release_year'})
print(df_renamed.head())
```

```
                movie_title  release_year      genre  rating    votes  \
0  The Shawshank Redemption          1994      Drama     9.3  2343110
1             The Godfather          1972      Crime     9.2  1620367
2           The Dark Knight          2008     Action     9.0  2456711
3              Pulp Fiction          1994      Crime     8.9  1892623
4            Schindler's List          1993  Biography     8.9  1389640

  rating_category
0       Excellent
1       Excellent
2       Excellent
3            Good
4            Good
```

[12]:
```python
import numpy as np
```

40°C Sunny

Search

ENG IN

15:04 02-05-2025

- For numpy : -
  - Dataset : -

```
[12]: import numpy as np

# Movie titles (object dtype for strings)
titles = np.array([
    'The Shawshank Redemption', 'The Godfather', 'The Dark Knight', 'Pulp Fiction',
    'Schindler\'s List', 'The Lord of the Rings', 'Fight Club', 'Forrest Gump',
    'Inception', 'The Matrix'
])

# Release years
years = np.array([1994, 1972, 2008, 1994, 1993, 2003, 1999, 1994, 2010, 1999])

# Genres
genres = np.array([
    'Drama', 'Crime', 'Action', 'Crime', 'Biography', 'Adventure',
    'Drama', 'Drama', 'Sci-Fi', 'Sci-Fi'
])

# IMDb ratings
ratings = np.array([9.3, 9.2, 9.0, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.7])

# Vote counts
votes = np.array([2343110, 1620367, 2456711, 1892623, 1389640, 1738653, 2042952, 2052217, 2352073, 1867775])
```

Grains : -

1.**Use np.mean() and np.sum()** to quickly compute the average or total of an array.

2.**Element-wise operations** (like ratings * 10) work directly on NumPy arrays without loops.

3.**Logical indexing** lets you filter values with conditions (e.g., ratings > 9).

4.**np.min() and np.max()** help you find the smallest and largest values in an array.

5.**np.std()** calculates the standard deviation — useful for measuring variability.

6.**np.column_stack()** combines multiple 1D arrays into a single 2D array.

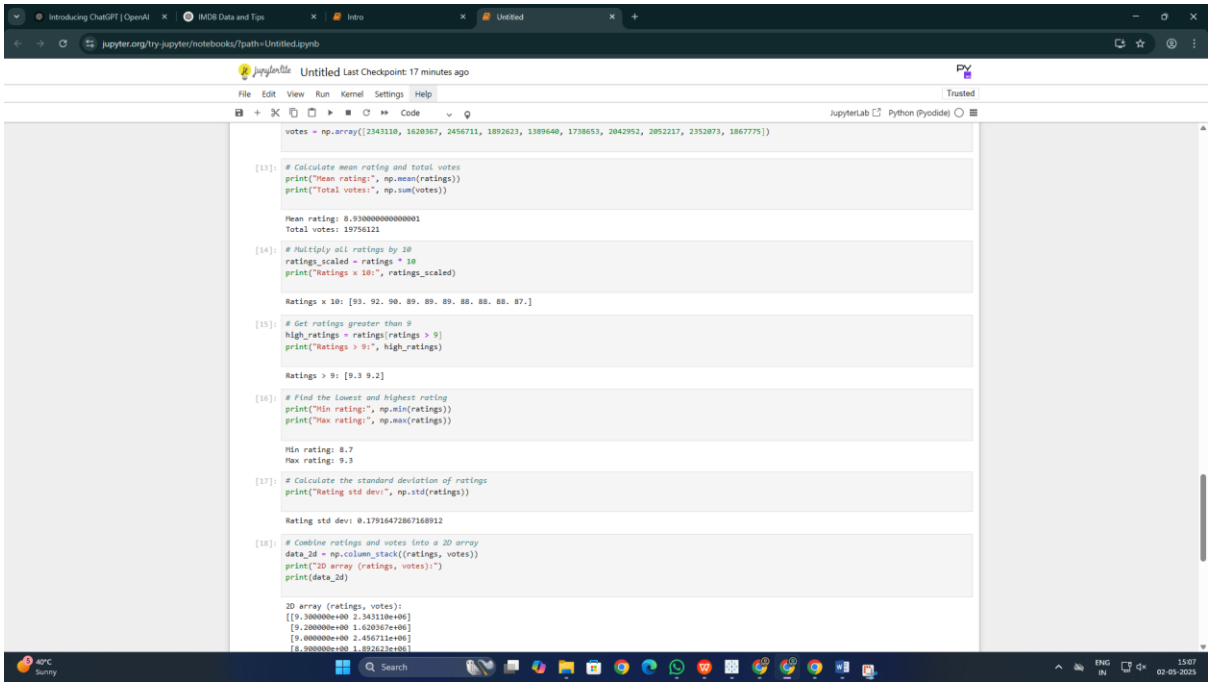
7.**shape** gives the dimensions (rows, columns) of any NumPy array.

8.**Min-max normalization** rescales values to a range of 0–1:
(x - min) / (max - min).

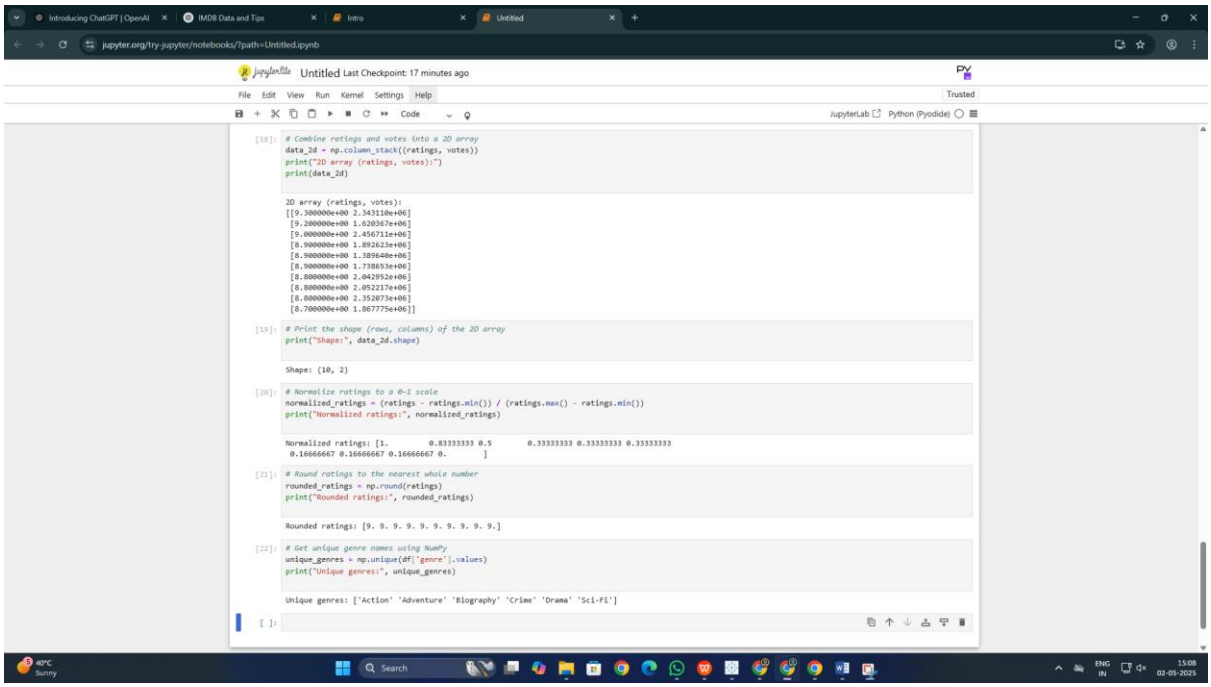
9.**np.round()** lets you round values to the nearest whole number (or set decimal places).

10.**np.unique()** finds all the unique elements in an array (e.g., unique genres).

Output in jupyter notebook for all grains : -



```
votes = np.array([2343110, 1620367, 2456711, 1892623, 1389640, 1738653, 2042952, 2052217, 2352073, 1867775])

[13]: # Calculate mean rating and total votes
print("Mean rating:", np.mean(ratings))
print("Total votes:", np.sum(votes))

Mean rating: 8.930000000000001
Total votes: 19756121

[14]: # Multiply all ratings by 10
ratings_scaled = ratings * 10
print("Ratings x 10:", ratings_scaled)

Ratings x 10: [93. 92. 90. 89. 89. 89. 88. 88. 88. 87.]

[15]: # Get ratings greater than 9
high_ratings = ratings[ratings > 9]
print("Ratings > 9:", high_ratings)

Ratings > 9: [9.3 9.2]

[16]: # Find the lowest and highest rating
print("Min rating:", np.min(ratings))
print("Max rating:", np.max(ratings))

Min rating: 8.7
Max rating: 9.3

[17]: # Calculate the standard deviation of ratings
print("Rating std dev:", np.std(ratings))

Rating std dev: 0.17916472867168912

[18]: # Combine ratings and votes into a 2D array
data_2d = np.column_stack((ratings, votes))
print("2D array (ratings, votes):")
print(data_2d)

2D array (ratings, votes):
[[9.300000e+00 2.343110e+06]
 [9.200000e+00 1.620367e+06]
 [9.000000e+00 2.456711e+06]
 [8.900000e+00 1.892623e+06]
```



```
[18]: # Combine ratings and votes into a 2D array
data_2d = np.column_stack((ratings, votes))
print("2D array (ratings, votes):")
print(data_2d)

2D array (ratings, votes):
[[9.300000e+00 2.343110e+06]
 [9.200000e+00 1.620367e+06]
 [9.000000e+00 2.456711e+06]
 [8.900000e+00 1.892623e+06]
 [8.900000e+00 1.389640e+06]
 [8.900000e+00 1.738653e+06]
 [8.800000e+00 2.042952e+06]
 [8.800000e+00 2.052217e+06]
 [8.800000e+00 2.352073e+06]
 [8.700000e+00 1.867775e+06]]

[19]: # Print the shape (rows, columns) of the 2D array
print("Shape:", data_2d.shape)

Shape: (10, 2)

[20]: # Normalize ratings to a 0-1 scale
normalized_ratings = (ratings - ratings.min()) / (ratings.max() - ratings.min())
print("Normalized ratings:", normalized_ratings)

Normalized ratings: [1.         0.83333333 0.5        0.33333333 0.33333333 0.33333333
 0.16666667 0.16666667 0.16666667 0.        ]

[21]: # Round ratings to the nearest whole number
rounded_ratings = np.round(ratings)
print("Rounded ratings:", rounded_ratings)

Rounded ratings: [9. 9. 9. 9. 9. 9. 9. 9. 9. 9.]

[22]: # Get unique genre names using NumPy
unique_genres = np.unique(df['genre'].values)
print("Unique genres:", unique_genres)

Unique genres: ['Action' 'Adventure' 'Biography' 'Crime' 'Drama' 'Sci-Fi']

[ ]:
```