Name : Ayush Panchal

Roll No : P24DS013

In [1]:
```
import pandas as pd
import numpy as np
import re
import sys
```

1. Perform encoding techniques studied in the class on the datasets for the quantitative data and observe the range of data

Importing dataset

In [2]:
```
df = pd.read_csv("..\\dataset\\imdb_top_1000.csv")
df.head()
```

Out[2]:

	Poster_Link	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	A	142 min	Drama	9.3	Two imprisoned men bond over a number of years...	80.0
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t...	100.0
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo...	84.0
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	A	202 min	Crime, Drama	9.0	The early life and career of Vito Corleone in ...	90.0
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	U	96 min	Crime, Drama	9.0	A jury holdout attempts to prevent a miscarria...	96.0

In [3]:

```
df.isnull().sum()
```

Out[3]:
```
Poster_Link         0
Series_Title        0
Released_Year       0
Certificate       101
Runtime             0
Genre               0
IMDB_Rating         0
Overview            0
Meta_score        157
Director            0
Star1               0
Star2               0
Star3               0
Star4               0
No_of_Votes         0
Gross             169
dtype: int64
```

In [4]:
```
# df.dropna(inplace=True)
```

Out[4]:
```
Poster_Link         0
Series_Title        0
Released_Year       0
Certificate       101
Runtime             0
Genre               0
IMDB_Rating         0
Overview            0
Meta_score        157
Director            0
Star1               0
Star2               0
Star3               0
Star4               0
No_of_Votes         0
Gross             169
dtype: int64
```

In [5]:
```
df.shape
```

Out[5]: (1000, 16)

In [6]:
```
df.describe()
```

Out[6]:

	IMDB_Rating	Meta_score	No_of_Votes
count	1000.000000	843.000000	1.000000e+03
mean	7.949300	77.971530	2.736929e+05
std	0.275491	12.376099	3.273727e+05
min	7.600000	28.000000	2.508800e+04
25%	7.700000	70.000000	5.552625e+04
50%	7.900000	79.000000	1.385485e+05
75%	8.100000	87.000000	3.741612e+05
max	9.300000	100.000000	2.343110e+06

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Poster_Link    1000 non-null   object 
 1   Series_Title   1000 non-null   object 
 2   Released_Year  1000 non-null   object 
 3   Certificate    899 non-null    object 
 4   Runtime        1000 non-null   object 
 5   Genre          1000 non-null   object 
 6   IMDB_Rating    1000 non-null   float64
 7   Overview       1000 non-null   object 
 8   Meta_score     843 non-null    float64
 9   Director       1000 non-null   object 
 10  Star1          1000 non-null   object 
 11  Star2          1000 non-null   object 
 12  Star3          1000 non-null   object 
 13  Star4          1000 non-null   object 
 14  No_of_Votes    1000 non-null   int64  
 15  Gross          831 non-null    object 
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

In [8]:
```python
# Clean the 'Gross' column: Remove commas and convert to numeric
df['Gross'] = pd.to_numeric(df['Gross'].str.replace(',', ''), errors='coerce')

# Quantitative columns for scaling
quantitative_cols = ['IMDB_Rating', 'Meta_score', 'No_of_Votes', 'Gross']

def min_max_scaling(column):
    min_val = column.min()
    max_val = column.max()
    return (column - min_val) / (max_val - min_val)

def z_score_scaling(column):
    mean_val = column.mean()
    std_val = column.std()
    return (column - mean_val) / std_val

# Apply Min-Max Scaling and Z-score Scaling
```

```python
df_min_max_scaled = df[quantitative_cols].apply(min_max_scaling)
df_z_score_scaled = df[quantitative_cols].apply(z_score_scaling)

# Observe the range of Min-Max scaled data
df_min_max_scaled.describe()
```

Out[8]:

	IMDB_Rating	Meta_score	No_of_Votes	Gross
count	1000.000000	843.000000	1000.000000	831.000000
mean	0.205471	0.694049	0.107249	0.072634
std	0.162054	0.171890	0.141229	0.117172
min	0.000000	0.000000	0.000000	0.000000
25%	0.058824	0.583333	0.013131	0.003472
50%	0.176471	0.708333	0.048947	0.025121
75%	0.294118	0.819444	0.150591	0.086210
max	1.000000	1.000000	1.000000	1.000000

In [9]:

```python
# Observe the range of Z-score scaled data
df_z_score_scaled.describe()
```

Out[9]:

	IMDB_Rating	Meta_score	No_of_Votes	Gross
count	1.000000e+03	8.430000e+02	1.000000e+03	8.310000e+02
mean	3.012701e-15	3.202921e-16	-5.684342e-17	-3.420182e-17
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.267917e+00	-4.037745e+00	-7.593941e-01	-6.198945e-01
25%	-9.049291e-01	-6.441068e-01	-6.664168e-01	-5.902612e-01
50%	-1.789531e-01	8.310128e-02	-4.128151e-01	-4.055020e-01
75%	5.470229e-01	7.295085e-01	3.068928e-01	1.158646e-01
max	4.902879e+00	1.779920e+00	6.321288e+00	7.914598e+00

In [10]:

```
df.columns
```

Out[10]:

```
Index(['Poster_Link', 'Series_Title', 'Released_Year', 'Certificate',
       'Runtime', 'Genre', 'IMDB_Rating', 'Overview', 'Meta_score', 'Director',
       'Star1', 'Star2', 'Star3', 'Star4', 'No_of_Votes', 'Gross'],
      dtype='object')
```

Perform encoding techniques studied in the class on the datasets for the qualitative data and observe the size of data in Bytes

In [11]:

```
# Select qualitative columns for encoding

# Split the Genre column by commas to handle multiple genres
df['Genre'] = df['Genre'].apply(lambda x: x.split(', '))

# Perform Multi-Label One-Hot Encoding on the 'Genre' column
# Create a separate column for each unique genre
df_genre_encoded = df['Genre'].str.join('|').str.get_dummies()

# Combine the encoded genre columns back to the main DataFrame
```

```python
df_encoded = pd.concat([df, df_genre_encoded], axis=1).drop(columns=['Genre'])

qualitative_cols = ['Certificate', "Director"]


# Calculate initial size of the dataset in Bytes
initial_size = sys.getsizeof(df)
print(f"Initial dataset size: {initial_size} Bytes")

# Label Encoding (manual implementation)
def label_encode(column):
    unique_vals = column.unique()
    val_map = {val: idx for idx, val in enumerate(unique_vals)}
    return column.map(val_map)

# One-Hot Encoding (using pandas)
def one_hot_encode(df, columns):
    return pd.get_dummies(df, columns=columns)

# Apply Label Encoding to columns where it might make sense (e.g., Certificate)
df_label_encoded = df_encoded.copy()
df_label_encoded['Certificate'] = label_encode(df_encoded['Certificate'])

# Apply One-Hot Encoding to other categorical columns
df_one_hot_encoded = one_hot_encode(df_label_encoded, qualitative_cols)

# Calculate size of the dataset after encoding
encoded_size = sys.getsizeof(df_one_hot_encoded)
print(f"Dataset size after encoding: {encoded_size} Bytes")

# Display a summary of the encoded dataset
df_one_hot_encoded.head()
```

```
Initial dataset size: 1097990 Bytes
Dataset size after encoding: 1572292 Bytes
```

Out[11]:

	Poster_Link	Series_Title	Released_Year	Runtime	IMDB_Rating	Overview	Meta_score	Star1	Star2
0	https://m.media-amazon.com/images/M/MV5BMDFkYT...	The Shawshank Redemption	1994	142 min	9.3	Two imprisoned men bond over a number of years...	80.0	Tim Robbins	Morgan Freeman
1	https://m.media-amazon.com/images/M/MV5BM2MyNj...	The Godfather	1972	175 min	9.2	An organized crime dynasty's aging patriarch t...	100.0	Marlon Brando	Al Pacino
2	https://m.media-amazon.com/images/M/MV5BMTMxNT...	The Dark Knight	2008	152 min	9.0	When the menace known as the Joker wreaks havo...	84.0	Christian Bale	Heath Ledger
3	https://m.media-amazon.com/images/M/MV5BMWMwMG...	The Godfather: Part II	1974	202 min	9.0	The early life and career of Vito Corleone in ...	90.0	Al Pacino	Robert De Niro
4	https://m.media-amazon.com/images/M/MV5BMWU4N2...	12 Angry Men	1957	96 min	9.0	A jury holdout attempts to prevent a miscarria...	96.0	Henry Fonda	Lee J. Cobb

5 rows × 599 columns

In [12]:

```
# Drop rows with missing values (blank, NULL, NA, etc.)
df_dropped_rows = df.dropna()

# Print the shape of the new DataFrame
print("Shape after dropping rows with missing values:", df_dropped_rows.shape)
```

```
Shape after dropping rows with missing values: (714, 16)
```

In [13]:

```
# Check if any column has all missing values
if df.isnull().all().any():
    # Find the column with all missing values
    column_with_all_missing = df.isnull().all()[df.isnull().all() == True].index[0]
    print("Column with all missing values:", column_with_all_missing)

    # Drop the column
    df = df.drop(column_with_all_missing, axis=1)
```

In [14]:

```
# Set the threshold for non-missing values
threshold = 5  # Adjust this value as needed

# Drop rows with fewer than X non-missing values
df_filtered = df.dropna(thresh=threshold)

# Print the shape of the new DataFrame
print("Shape after dropping rows with fewer than", threshold, "non-missing values:", df_filtered.shape)
```

```
Shape after dropping rows with fewer than 5 non-missing values: (1000, 16)
```

In [15]:

```
import pandas as pd
import numpy as np

# Load the dataset
data_path = r"..\dataset\imdb_top_1000.csv"
df = pd.read_csv(data_path)

# 1. Display columns and count of missing values
missing_values_count = df.isnull().sum()
print("Missing values in each column:\n", missing_values_count[missing_values_count > 0])
```

```python
# a. Drop rows having missing values in various formats like blank, NULL, NA, etc.
df_dropped_rows = df.dropna()

# b. Drop columns if all the values are missing
df_dropped_columns = df.dropna(axis=1, how='all')

# c. Drop rows that contain less than user-given X non-missing values
def drop_rows_less_than_x_non_missing(df, x):
    return df.dropna(thresh=x)

# User-specified threshold for minimum non-missing values in a row
x = 10  # for example, you can change it
df_dropped_by_threshold = drop_rows_less_than_x_non_missing(df, x)

# d. Replace the missing value cells with various strategies
def replace_missing_values(df, method):
    df_copy = df.copy()
    if method == 'zeros':
        return df_copy.fillna(0)
    elif method == 'min':
        # Replace missing values with the minimum value column-wise
        for col in df_copy.select_dtypes(include=[np.number]):
            df_copy[col] = df_copy[col].fillna(df_copy[col].min())
        return df_copy
    elif method == 'max':
        # Replace missing values with the maximum value column-wise
        for col in df_copy.select_dtypes(include=[np.number]):
            df_copy[col] = df_copy[col].fillna(df_copy[col].max())
        return df_copy
    elif method == 'mean':
        # Replace missing values with the mean value column-wise
        for col in df_copy.select_dtypes(include=[np.number]):
            df_copy[col] = df_copy[col].fillna(df_copy[col].mean())
        return df_copy
    elif method == 'variance':
        # Replace missing values with the variance column-wise
        for col in df_copy.select_dtypes(include=[np.number]):
            df_copy[col] = df_copy[col].fillna(df_copy[col].var())
        return df_copy
    elif method == 'std_dev':
        # Replace missing values with the standard deviation column-wise
```

```
        for col in df_copy.select_dtypes(include=[np.number]):
            df_copy[col] = df_copy[col].fillna(df_copy[col].std())
        return df_copy
    else:
        return df_copy

# List of replacement methods
methods = ['zeros', 'min', 'max', 'mean', 'variance', 'std_dev']

# Initialize a dictionary to store the description of each strategy
description_stats = {}

for method in methods:
    df_filled = replace_missing_values(df, method)

    # Calculate statistics for the filled DataFrame
    mean_val = df_filled.mean(numeric_only=True)
    median_val = df_filled.median(numeric_only=True)
    mode_val = df_filled.mode(numeric_only=True).iloc[0]
    variance_val = df_filled.var(numeric_only=True)
    std_val = df_filled.std(numeric_only=True)

    # Print the statistics
    print(f"\nStatistics after replacing missing values with '{method}':")
    print(f"Mean:\n{mean_val}")
    print()
    print(f"Median:\n{median_val}")
    print()
    print(f"Mode:\n{mode_val}")
    print()
    print(f"Variance:\n{variance_val}")
    print()
    print(f"Standard Deviation:\n{std_val}")

    # Store the description for analysis
    description_stats[method] = {
        'mean': mean_val,
        'median': median_val,
        'mode': mode_val,
        'variance': variance_val,
        'std_dev': std_val
```

```python
    }

# Summary of best method choice
print("\nDescription of which method is best:")
for method, stats in description_stats.items():
    print(f"\nMethod: {method}")
    print()
    print("Mean:\n", stats['mean'])
    print()
    print("Median:\n", stats['median'])
    print()
    print("Mode:\n", stats['mode'])
    print()
    print("Variance:\n", stats['variance'])
    print()
    print("Standard Deviation:\n", stats['std_dev'])
```

```
Missing values in each column:
 Certificate    101
Meta_score     157
Gross          169
dtype: int64

Statistics after replacing missing values with 'zeros':
Mean:
IMDB_Rating         7.9493
Meta_score         65.7300
No_of_Votes    273692.9110
dtype: float64

Median:
IMDB_Rating         7.9
Meta_score         76.0
No_of_Votes    138548.5
dtype: float64

Mode:
IMDB_Rating        7.7
Meta_score         0.0
No_of_Votes    65341.0
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     9.345376e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
IMDB_Rating         0.275491
Meta_score         30.570208
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'min':
Mean:
IMDB_Rating         7.9493
Meta_score         70.1260
```

```
No_of_Votes    273692.9110
dtype: float64

Median:
IMDB_Rating         7.9
Meta_score         76.0
No_of_Votes    138548.5
dtype: float64

Mode:
IMDB_Rating        7.7
Meta_score        28.0
No_of_Votes    65341.0
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     4.599281e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
IMDB_Rating         0.275491
Meta_score         21.445933
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'max':
Mean:
IMDB_Rating         7.9493
Meta_score         81.4300
No_of_Votes    273692.9110
dtype: float64

Median:
IMDB_Rating         7.9
Meta_score         82.0
No_of_Votes    138548.5
dtype: float64

Mode:
```

```
IMDB_Rating        7.7
Meta_score       100.0
No_of_Votes    65341.0
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     1.933845e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
IMDB_Rating         0.275491
Meta_score         13.906275
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'mean':
Mean:
IMDB_Rating         7.94930
Meta_score         77.97153
No_of_Votes    273692.91100
dtype: float64

Median:
IMDB_Rating         7.90000
Meta_score         77.97153
No_of_Votes    138548.50000
dtype: float64

Mode:
IMDB_Rating        7.70000
Meta_score        77.97153
No_of_Votes    65341.00000
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     1.290964e+02
No_of_Votes    1.071729e+11
dtype: float64
```

```
Standard Deviation:
IMDB_Rating         0.275491
Meta_score         11.362060
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'variance':
Mean:
IMDB_Rating         7.94930
Meta_score         89.77735
No_of_Votes    273692.91100
dtype: float64

Median:
IMDB_Rating         7.9
Meta_score         82.0
No_of_Votes    138548.5
dtype: float64

Mode:
IMDB_Rating        7.700000
Meta_score       153.167835
No_of_Votes    65341.000000
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     8.782222e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
IMDB_Rating         0.275491
Meta_score         29.634814
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'std_dev':
Mean:
IMDB_Rating         7.949300
```

```
Meta_score         67.673048
No_of_Votes    273692.911000
dtype: float64

Median:
IMDB_Rating         7.9
Meta_score         76.0
No_of_Votes    138548.5
dtype: float64

Mode:
IMDB_Rating        7.700000
Meta_score        12.376099
No_of_Votes    65341.000000
Name: 0, dtype: float64

Variance:
IMDB_Rating    7.589541e-02
Meta_score     6.991411e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
IMDB_Rating         0.275491
Meta_score         26.441277
No_of_Votes    327372.703934
dtype: float64

Description of which method is best:

Method: zeros

Mean:
 IMDB_Rating         7.9493
Meta_score         65.7300
No_of_Votes    273692.9110
dtype: float64

Median:
 IMDB_Rating         7.9
Meta_score         76.0
```

```
No_of_Votes    138548.5
dtype: float64

Mode:
 IMDB_Rating        7.7
Meta_score         0.0
No_of_Votes    65341.0
Name: 0, dtype: float64

Variance:
 IMDB_Rating    7.589541e-02
Meta_score     9.345376e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         30.570208
No_of_Votes    327372.703934
dtype: float64

Method: min

Mean:
 IMDB_Rating         7.9493
Meta_score         70.1260
No_of_Votes    273692.9110
dtype: float64

Median:
 IMDB_Rating         7.9
Meta_score         76.0
No_of_Votes    138548.5
dtype: float64

Mode:
 IMDB_Rating        7.7
Meta_score        28.0
No_of_Votes    65341.0
Name: 0, dtype: float64
```

```
Variance:
 IMDB_Rating    7.589541e-02
Meta_score     4.599281e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         21.445933
No_of_Votes    327372.703934
dtype: float64

Method: max

Mean:
 IMDB_Rating         7.9493
Meta_score         81.4300
No_of_Votes    273692.9110
dtype: float64

Median:
 IMDB_Rating         7.9
Meta_score         82.0
No_of_Votes    138548.5
dtype: float64

Mode:
 IMDB_Rating        7.7
Meta_score       100.0
No_of_Votes    65341.0
Name: 0, dtype: float64

Variance:
 IMDB_Rating    7.589541e-02
Meta_score     1.933845e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         13.906275
```

```
No_of_Votes    327372.703934
dtype: float64

Method: mean

Mean:
 IMDB_Rating         7.94930
Meta_score         77.97153
No_of_Votes    273692.91100
dtype: float64

Median:
 IMDB_Rating         7.90000
Meta_score         77.97153
No_of_Votes    138548.50000
dtype: float64

Mode:
 IMDB_Rating        7.70000
Meta_score        77.97153
No_of_Votes    65341.00000
Name: 0, dtype: float64

Variance:
 IMDB_Rating    7.589541e-02
Meta_score     1.290964e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         11.362060
No_of_Votes    327372.703934
dtype: float64

Method: variance

Mean:
 IMDB_Rating         7.94930
Meta_score         89.77735
No_of_Votes    273692.91100
```

```
dtype: float64

Median:
 IMDB_Rating         7.9
Meta_score         82.0
No_of_Votes    138548.5
dtype: float64

Mode:
 IMDB_Rating        7.700000
Meta_score       153.167835
No_of_Votes    65341.000000
Name: 0, dtype: float64

Variance:
 IMDB_Rating    7.589541e-02
Meta_score     8.782222e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         29.634814
No_of_Votes    327372.703934
dtype: float64

Method: std_dev

Mean:
 IMDB_Rating         7.949300
Meta_score         67.673048
No_of_Votes    273692.911000
dtype: float64

Median:
 IMDB_Rating         7.9
Meta_score         76.0
No_of_Votes    138548.5
dtype: float64

Mode:
```

```
 IMDB_Rating        7.700000
Meta_score        12.376099
No_of_Votes    65341.000000
Name: 0, dtype: float64

Variance:
 IMDB_Rating    7.589541e-02
Meta_score     6.991411e+02
No_of_Votes    1.071729e+11
dtype: float64

Standard Deviation:
 IMDB_Rating         0.275491
Meta_score         26.441277
No_of_Votes    327372.703934
dtype: float64
```

Zeros: This method can drastically affect the statistics, especially for large datasets where zeros aren't meaningful. It can reduce the mean and introduce bias if zeros do not represent valid data points.

Minimum Value: Replacing with the minimum value is useful when missing values represent the lowest possible outcome in the context. However, it can skew the dataset toward lower values, which might distort the overall data distribution.

Maximum Value: Similar to minimum, but it skews the dataset toward higher values. It might make sense in cases where missing values are expected to represent outliers or extreme positive cases.

Mean of the Column: This is a common imputation technique because it maintains the overall average of the dataset. It can be the best option when missing data is randomly distributed. However, it reduces variance and can mask the natural spread of the data.

Variance: Replacing with variance is less common but could make sense when trying to maintain variability in the dataset. However, since variance is a measure of spread rather than a central tendency, it’s not often the most appropriate for replacing missing values.

Standard Deviation: Similar to variance, using standard deviation maintains the data’s spread but may not be as intuitive or useful for imputation.

In [16]:

```
# Select only numeric columns (int64, float64)
df_numeric = df.select_dtypes(include=[np.number])
```

```python
# Function to replace missing values with the mean of 2 forward neighbors
def replace_with_forward_neighbors_mean(df):
    df_copy = df.copy()

    # Only apply this to numeric columns
    for col in df_copy.select_dtypes(include=[np.number]):
        for i in range(len(df_copy[col])):
            if pd.isnull(df_copy[col].iloc[i]):
                # Use loc[] to avoid chained assignment and calculate the mean of the next two forward neighbors
                forward_mean = df_copy[col].iloc[i+1:i+3].mean()
                df_copy.loc[i, col] = forward_mean  # Use .loc[] for safe assignment

    return df_copy

# Function to replace missing values with the mean of 2 backward neighbors
def replace_with_backward_neighbors_mean(df):
    df_copy = df.copy()

    # Only apply this to numeric columns
    for col in df_copy.select_dtypes(include=[np.number]):
        for i in range(len(df_copy[col])):
            if pd.isnull(df_copy[col].iloc[i]):
                # Calculate the mean of the previous two backward neighbors
                backward_mean = df_copy[col].iloc[i-2:i].mean()
                df_copy.loc[i, col] = backward_mean

    return df_copy

# Function to replace missing values with the mean of 2 forward and 2 backward neighbors
def replace_with_neighbors_mean(df):
    df_copy = df.copy()

    # Only apply this to numeric columns
    for col in df_copy.select_dtypes(include=[np.number]):
        for i in range(len(df_copy[col])):
            if pd.isnull(df_copy[col].iloc[i]):
                # Calculate the mean of both backward and forward neighbors
                backward_mean = df_copy[col].iloc[max(i-2, 0):i].mean()  # Ensure the index is non-negative
                forward_mean = df_copy[col].iloc[i+1:i+3].mean()
                overall_mean = np.nanmean([backward_mean, forward_mean])  # Use np.nanmean to handle NaN values
                df_copy.loc[i, col] = overall_mean
```

```python
    return df_copy

# List of replacement methods
methods = {
    'backward_neighbors': replace_with_forward_neighbors_mean,
    'forward_neighbors': replace_with_backward_neighbors_mean,
    'both_neighbors': replace_with_neighbors_mean
}

# Initialize a dictionary to store the description of each strategy
description_stats = {}

for method_name, method_func in methods.items():
    df_filled = method_func(df_numeric)

    # Calculate statistics for the filled DataFrame
    mean_val = df_filled.mean(numeric_only=True)
    median_val = df_filled.median(numeric_only=True)
    mode_val = df_filled.mode(numeric_only=True).iloc[0]
    variance_val = df_filled.var(numeric_only=True)
    std_val = df_filled.std(numeric_only=True)

    # Print the statistics
    print(f"\nStatistics after replacing missing values with '{method_name}':")
    print(f"Mean:\n{mean_val}")
    print(f"Median:\n{median_val}")
    print(f"Mode:\n{mode_val}")
    print(f"Variance:\n{variance_val}")
    print(f"Standard Deviation:\n{std_val}")

    # Store the description for analysis
    description_stats[method_name] = {
        'mean': mean_val,
        'median': median_val,
        'mode': mode_val,
        'variance': variance_val,
        'std_dev': std_val
    }
```

```
Statistics after replacing missing values with 'backward_neighbors':
Mean:
IMDB_Rating         7.949300
Meta_score         78.526849
No_of_Votes    273692.911000
dtype: float64
Median:
IMDB_Rating         7.9
Meta_score         79.0
No_of_Votes    138548.5
dtype: float64
Mode:
IMDB_Rating        7.7
Meta_score        76.0
No_of_Votes    65341.0
Name: 0, dtype: float64
Variance:
IMDB_Rating    7.589541e-02
Meta_score     1.465158e+02
No_of_Votes    1.071729e+11
dtype: float64
Standard Deviation:
IMDB_Rating         0.275491
Meta_score         12.104369
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'forward_neighbors':
Mean:
IMDB_Rating         7.949300
Meta_score         78.732988
No_of_Votes    273692.911000
dtype: float64
Median:
IMDB_Rating         7.9
Meta_score         80.0
No_of_Votes    138548.5
dtype: float64
Mode:
IMDB_Rating        7.7
Meta_score        76.0
```

```
No_of_Votes    65341.0
Name: 0, dtype: float64
Variance:
IMDB_Rating    7.589541e-02
Meta_score     1.461735e+02
No_of_Votes    1.071729e+11
dtype: float64
Standard Deviation:
IMDB_Rating         0.275491
Meta_score         12.090225
No_of_Votes    327372.703934
dtype: float64

Statistics after replacing missing values with 'both_neighbors':
Mean:
IMDB_Rating         7.949300
Meta_score         78.669079
No_of_Votes    273692.911000
dtype: float64
Median:
IMDB_Rating         7.9
Meta_score         80.0
No_of_Votes    138548.5
dtype: float64
Mode:
IMDB_Rating        7.7
Meta_score        76.0
No_of_Votes    65341.0
Name: 0, dtype: float64
Variance:
IMDB_Rating    7.589541e-02
Meta_score     1.413915e+02
No_of_Votes    1.071729e+11
dtype: float64
Standard Deviation:
IMDB_Rating         0.275491
Meta_score         11.890815
No_of_Votes    327372.703934
dtype: float64
```

Part : B

1. Write a code to input the following values and validate them:

a. Email

b. Indian Name

c. Mobile number with and without country code

d. Your Admission No.

In [17]:
```
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    return False

def validate_indian_name(name):
    # Indian names typically contain letters and may have spaces
    pattern = r'^[A-Za-z ]+$'
    if re.match(pattern, name):
        return True
    return False

def validate_mobile(mobile):
    # Validate with and without country code
    pattern_with_code = r'^\+91[6-9][0-9]{9}$'
    pattern_without_code = r'^[6-9][0-9]{9}$'
    if re.match(pattern_with_code, mobile) or re.match(pattern_without_code, mobile):
        return True
    return False
```

```python
def validate_admission_no(admission_no):
    # Admission number format (p24ds013 or similar, allowing mixed case)
    pattern = r'^[A-Za-z][0-9]{2}[A-Za-z]{2}[0-9]{3}$'
    if re.match(pattern, admission_no):
        return True
    return False

email = input("Enter your Email: ")
if validate_email(email):
    print("Valid Email!")
else:
    print("Invalid Email!")

# Input Indian Name
name = input("Enter your Name: ")
if validate_indian_name(name):
    print("Valid Indian Name!")
else:
    print("Invalid Indian Name!")

# Input Mobile Number
mobile = input("Enter your Mobile Number: ")
if validate_mobile(mobile):
    print("Valid Mobile Number!")
else:
    print("Invalid Mobile Number!")

# Input Admission Number
admission_no = input("Enter your Admission Number: ")
if validate_admission_no(admission_no):
    print("Valid Admission Number!")
else:
    print("Invalid Admission Number!")
```

```
Valid Email!
Valid Indian Name!
Valid Mobile Number!
Valid Admission Number!
```

2. Write a code to take the full address with country, state, city, and pincode from the user and validate them all.

In [18]:

```
def validate_country(country):
    # Country names generally contain letters and spaces, allowing both uppercase and lowercase
    pattern = r'^[A-Za-z ]+$'
    if re.match(pattern, country):
        return True
    return False

def validate_state(state):
    # States can also contain letters and spaces, allowing both uppercase and lowercase
    pattern = r'^[A-Za-z ]+$'
    if re.match(pattern, state):
        return True
    return False

def validate_city(city):
    # Cities typically contain letters and spaces
    pattern = r'^[A-Za-z ]+$'
    if re.match(pattern, city):
        return True
    return False

def validate_pincode(pincode):
    # Indian pincodes are 6-digit numbers; adjust for different countries if needed
    pattern = r'^[1-9][0-9]{5}$'
    if re.match(pattern, pincode):
        return True
    return False

# Input for Country
country = input("Enter your Country: ")
if validate_country(country):
    print("Valid Country!")
else:
    print("Invalid Country!")
```

```python
# Input for State
state = input("Enter your State: ")
if validate_state(state):
    print("Valid State!")
else:
    print("Invalid State!")

# Input for City
city = input("Enter your City: ")
if validate_city(city):
    print("Valid City!")
else:
    print("Invalid City!")

# Input for Pincode
pincode = input("Enter your Pincode: ")
if validate_pincode(pincode):
    print("Valid Pincode!")
else:
    print("Invalid Pincode!")
```

```
Valid Country!
Valid State!
Valid City!
Valid Pincode!
```

3. Write a code to convert the following fields:

a. DD/MM/YY format to MM/DD/YY b. Fee amount from Rs. to $

In [19]:

```python
# Function to convert DD/MM/YY to MM/DD/YY
def convert_date(date):
    try:
        # Split the input date based on "/"
        day, month, year = date.split('/')

        # Rearrange it to MM/DD/YY
        return f"{month}/{day}/{year}"
```

```python
    except ValueError:
        return "Invalid Date Format! Please use DD/MM/YY."

# Function to convert fee from Rs. to $
def convert_fee_to_usd(fee_in_rs, exchange_rate):
    try:
        # Convert the fee from INR to USD
        fee_in_usd = fee_in_rs / exchange_rate
        return round(fee_in_usd, 2)  # Rounded to 2 decimal places
    except Exception as e:
        return f"Error in conversion: {str(e)}"

# Convert Date
date_ddmmyy = input("Enter date in DD/MM/YY format: ")
converted_date = convert_date(date_ddmmyy)
print(f"Converted Date (MM/DD/YY): {converted_date}")

# Convert Fee
try:
    fee_in_rs = float(input("Enter fee amount in Rs.: "))
    exchange_rate = float(input("Enter the exchange rate from Rs. to $: "))  # Example: 1 USD = 83 INR
    converted_fee = convert_fee_to_usd(fee_in_rs, exchange_rate)
    print(f"Fee Amount in $: ${converted_fee}")
except ValueError:
    print("Please enter valid numeric values for fee and exchange rate.")
```

```
Converted Date (MM/DD/YY): 11/18/02
Fee Amount in $: $161.29
```

4. Without using a built-in function to write a code to calculate the Pearson's correlation of your selected dataset.

In [20]:

```python
def pearson_correlation(x, y):
    # Ensure the two lists have the same length
    if len(x) != len(y):
        raise ValueError("Both datasets must have the same number of elements")

    # Number of data points
```

```python
    n = len(x)

    # Initialize variables for sums
    sum_x = sum_y = sum_xy = sum_x2 = sum_y2 = 0

    # Calculate the required sums
    for i in range(n):
        sum_x += x[i]
        sum_y += y[i]
        sum_xy += x[i] * y[i]
        sum_x2 += x[i] * x[i]
        sum_y2 += y[i] * y[i]

    # Calculate Pearson's correlation coefficient
    numerator = (n * sum_xy) - (sum_x * sum_y)
    denominator = ((n * sum_x2 - sum_x**2) * (n * sum_y2 - sum_y**2)) ** 0.5

    if denominator == 0:
        return 0  # Return 0 correlation if denominator is zero (e.g., when there's no variation)

    return numerator / denominator

# Example dataset
x = df["IMDB_Rating"]
y = df["No_of_Votes"]

# Calculate and print the Pearson correlation
correlation = pearson_correlation(x, y)
print(f"Pearson's Correlation Coefficient: {correlation}")
```

```
Pearson's Correlation Coefficient: 0.49497883586203517
```

In [21]:

```python
# Function to rank the elements of a list
def rank_data(data):
    # Create a sorted version of the dataset with its indices
    sorted_data = sorted((val, idx) for idx, val in enumerate(data))

    # Initialize the rank list
    ranks = [0] * len(data)

    # Assign ranks to each data point
```

```python
    for rank, (val, idx) in enumerate(sorted_data):
        ranks[idx] = rank + 1  # Rank starts at 1

    return ranks

# Function to calculate Spearman's correlation
def spearman_correlation(x, y):
    # Ensure both datasets have the same number of elements
    if len(x) != len(y):
        raise ValueError("Both datasets must have the same number of elements")

    n = len(x)

    # Get ranks for both datasets
    rank_x = rank_data(x)
    rank_y = rank_data(y)

    # Calculate the sum of squared differences of ranks
    d_squared_sum = 0
    for i in range(n):
        d_squared_sum += (rank_x[i] - rank_y[i]) ** 2

    # Apply Spearman's rank correlation formula
    spearman_corr = 1 - (6 * d_squared_sum) / (n * (n**2 - 1))

    return spearman_corr

# Example dataset
x = df["IMDB_Rating"]
y = df["No_of_Votes"]

# Calculate and print the Spearman's correlation
correlation = spearman_correlation(x, y)
print(f"Spearman's Correlation Coefficient: {correlation}")
```

```
Spearman's Correlation Coefficient: 0.17748504948504495
```

In [ ]: