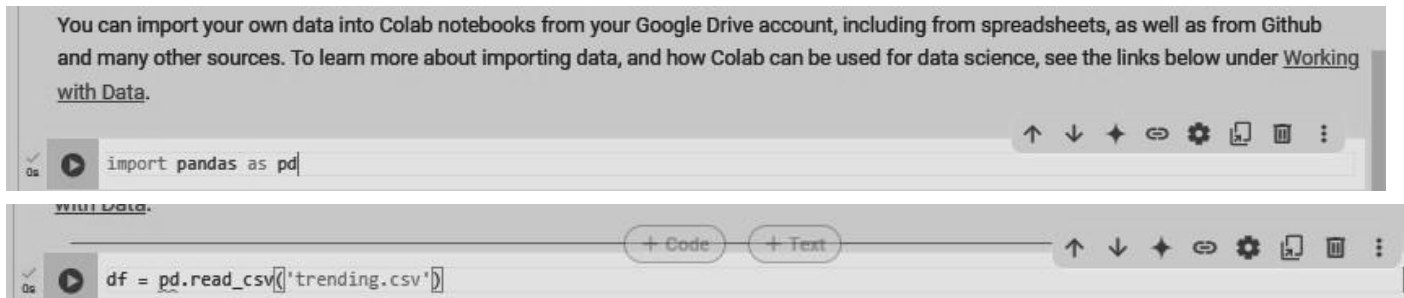


DS-1 Lab Exp 1

AIM: Introduction to Data science and Data preparation using Pandas steps.

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

Step 1: Firstly import Pandas Library as pd and then Load data in Pandas using `pd.read_csv`.



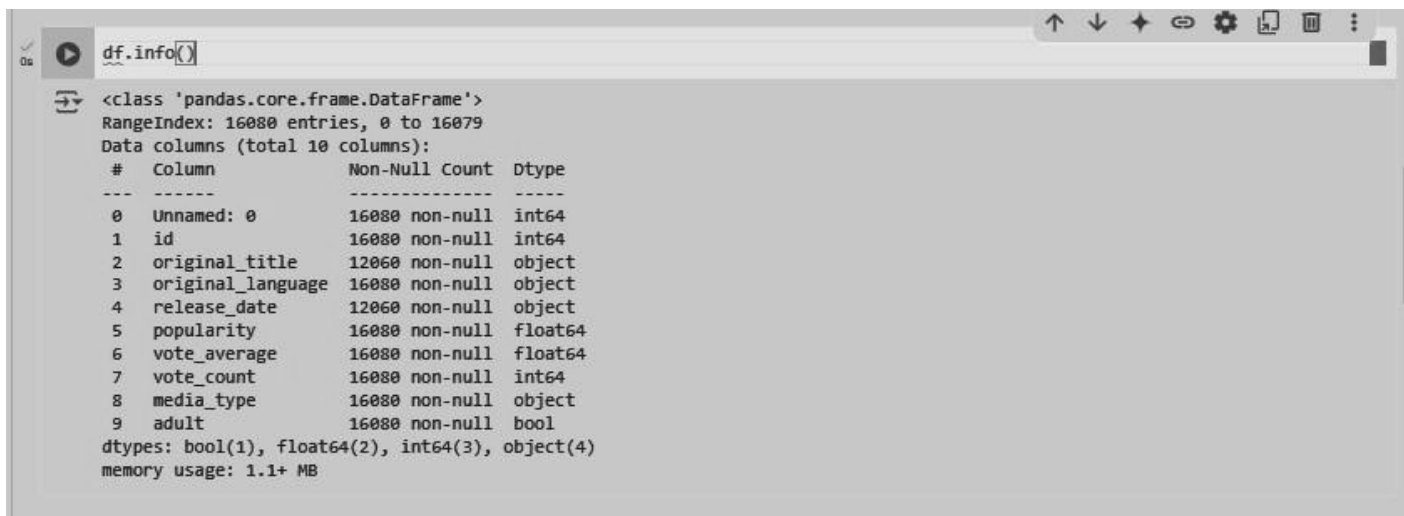
```
import pandas as pd

df = pd.read_csv('trending.csv')
```

Step 2: Get Description of the Dataset by using following 2 commands

`df.info()` -> Get basic information about the dataset

`df.describe()` -> Summary statistics of the dataset



```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16080 entries, 0 to 16079
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  --
 0   Unnamed: 0           16080 non-null  int64  
 1   id                   16080 non-null  int64  
 2   original_title       12060 non-null  object  
 3   original_language    16080 non-null  object  
 4   release_date         12060 non-null  object  
 5   popularity           16080 non-null  float64 
 6   vote_average         16080 non-null  float64 
 7   vote_count           16080 non-null  int64  
 8   media_type           16080 non-null  object  
 9   adult                16080 non-null  bool    
dtypes: bool(1), float64(2), int64(3), object(4)
memory usage: 1.1+ MB
```

df.describe()

	Unnamed: 0	id	popularity	vote_average	vote_count
count	16080.0000	1.608000e+04	16080.000000	16080.000000	16080.00000
mean	8039.5000	5.758387e+05	934.262000	7.536350	1039.75000
std	4642.0405	3.271352e+05	2229.935599	1.057306	2326.96193
min	0.0000	7.660000e+04	30.374000	4.800000	3.00000
25%	4019.7500	1.963872e+05	61.750250	6.875000	9.75000
50%	8039.5000	6.580765e+05	94.859000	7.721000	53.50000
75%	12059.2500	8.569955e+05	804.053750	8.040250	363.00000
max	16079.0000	1.049638e+06	10224.280000	10.000000	8697.00000

Step 3: Drop Columns that aren't useful. From Our Dataset we are dropping the "adult" column .

```
cols = ['adult']
df = df.drop(cols,axis=1)
```

We can see that it returned total 9 columns as it dropped the adult column

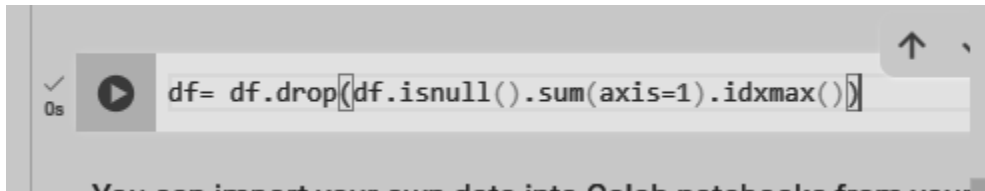
```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 16079 entries, 0 to 16079
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Unnamed: 0            16079 non-null  int64
1   id                    16079 non-null  int64
2   original_title        12060 non-null  object
3   original_language     16079 non-null  object
4   release_date          12060 non-null  object
5   popularity            16079 non-null  float64
6   vote_average          16079 non-null  float64
7   vote_count            16079 non-null  int64
8   media_type            16079 non-null  object
dtypes: float64(2), int64(3), object(4)
memory usage: 1.2+ MB
```

Step 4: Drop row with maximum missing values.

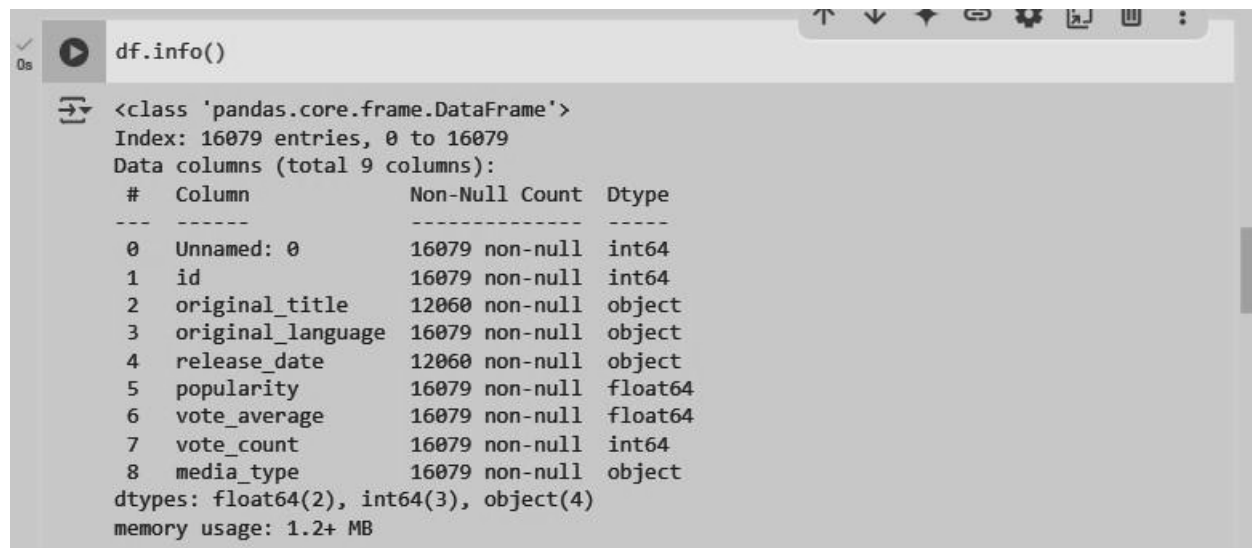
`df.isnull().sum(axis=1)` -> Computes the number of missing values (NaN) for each row.

`.idxmax()` -> Returns the index of row with max. no. of missing value



```
df= df.drop(df.isnull().sum(axis=1).idxmax())
```

We can see below that `df.info()` returns total 16079 entries, initially there were 16080 entries

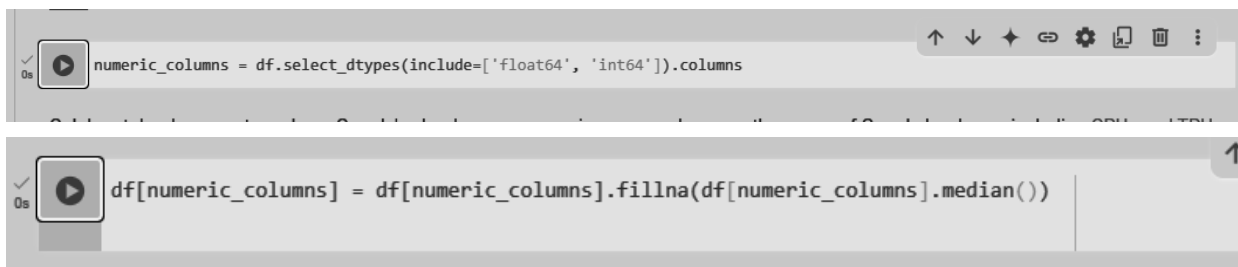


```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 16079 entries, 0 to 16079
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  --
 0   Unnamed: 0             16079 non-null  int64  
 1   id                     16079 non-null  int64  
 2   original_title         12060 non-null  object  
 3   original_language      16079 non-null  object  
 4   release_date           12060 non-null  object  
 5   popularity             16079 non-null  float64 
 6   vote_average           16079 non-null  float64 
 7   vote_count             16079 non-null  int64  
 8   media_type             16079 non-null  object  
dtypes: float64(2), int64(3), object(4)
memory usage: 1.2+ MB
```

Step 5: Taking care of missing data.

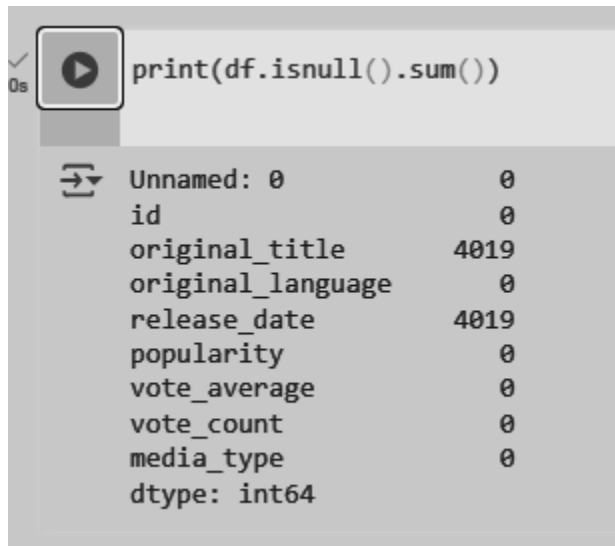
We can fill the empty numeric values with mode or median or mean. Below we had filled it with median. Firstly we had fetched the numeric values and then using **`.fillna().median`** we had filled it.



```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].median())
```

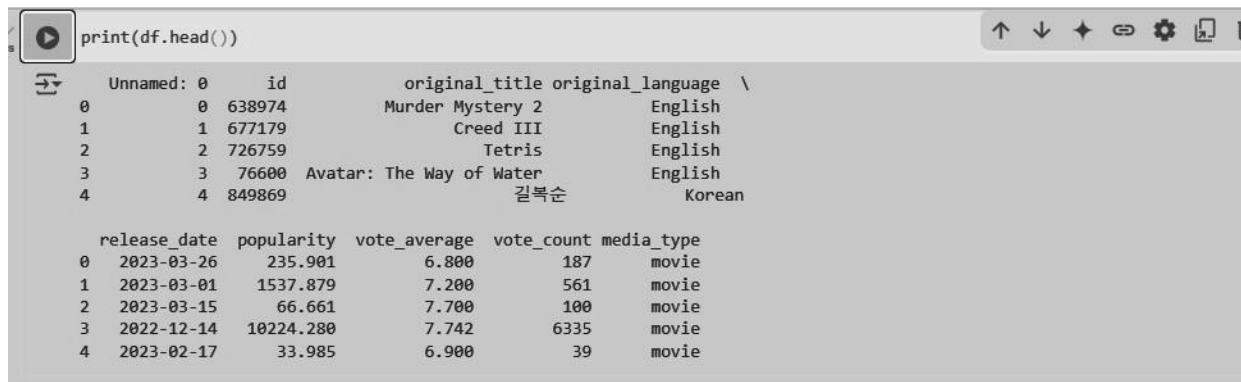
We can see that all the columns which had empty are filled. As they returned the sum 0



A screenshot of a Jupyter Notebook cell. The code `print(df.isnull().sum())` is entered. Below the code, the output is displayed as a table with two columns: the column name and its sum of null values. The 'Unnamed: 0' column has a sum of 0. The 'id' column has a sum of 0. The 'original_title' column has a sum of 4019. The 'original_language' column has a sum of 0. The 'release_date' column has a sum of 4019. The 'popularity' column has a sum of 0. The 'vote_average' column has a sum of 0. The 'vote_count' column has a sum of 0. The 'media_type' column has a sum of 0. The dtype is int64.

Unnamed: 0	0
id	0
original_title	4019
original_language	0
release_date	4019
popularity	0
vote_average	0
vote_count	0
media_type	0
dtype:	int64

`df.head()` returns starting 5 values



A screenshot of a Jupyter Notebook cell. The code `print(df.head())` is entered. Below the code, the output is displayed as a table with 6 columns: 'Unnamed: 0', 'id', 'original_title', 'original_language', 'release_date', 'popularity', 'vote_average', 'vote_count', and 'media_type'. The first 5 rows are shown, corresponding to the first 5 values of the dataset.

	Unnamed: 0	id	original_title	original_language	release_date	popularity	vote_average	vote_count	media_type
0	0	638974	Murder Mystery 2	English	2023-03-26	235.901	6.800	187	movie
1	1	677179	Creed III	English	2023-03-01	1537.879	7.200	561	movie
2	2	726759	Tetris	English	2023-03-15	66.661	7.700	100	movie
3	3	76600	Avatar: The Way of Water	English	2022-12-14	10224.280	7.742	6335	movie
4	4	849869	길복순	Korean	2023-02-17	33.985	6.900	39	movie

```
print(df.head(20))
```

6	6	493529	Dungeons & Dragons: Honor Among Thieves
7	7	932430	Prom Pact
9	9	816904	Momias
10	10	514999	Murder Mystery
11	11	1049638	Rye Lane
12	12	739405	Operation Fortune: Ruse de Guerre
13	13	158876	NaN
14	14	921355	Assassin
15	15	117465	NaN
16	16	933419	Champions
17	17	208891	NaN
18	18	878375	On a Wing and a Prayer
19	19	82856	NaN
20	20	638974	Murder Mystery 2

	original_language	release_date	popularity	vote_average	vote_count
0	English	2023-03-26	235.901	6.800	187
1	English	2023-03-01	1537.879	7.200	561

0s completed at 12:09 PM

Step 6: Create dummy variables. By using the below commands separate columns are created for each unique value in a column

```
df = pd.get_dummies(df)
```

+ Code + Text

```
print(df.head(20))
```

7	False	False	False
9	False	False	False
10	False	False	False
11	False	True	False
12	False	False	False
13	False	False	False
14	False	False	False
15	False	False	False
16	False	False	False
17	False	False	False
18	False	False	False
19	False	False	False
20	False	False	False

	release_date_2023-03-23	release_date_2023-03-26	release_date_2023-03-30
0	False	True	False

We can understand the working here,

As we can see that we now it have returned 42 columns. But previously our data had 9 columns .

So this change is because of the dummy variables , it have created separate column for each unique value in a column

Below it shows original_title_Assassin, original_language_English.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16079 entries, 0 to 16079
Data columns (total 42 columns):
 #   Column                                          Non-Null Count  Dtype
---  -
 0   Unnamed: 0                                    16079 non-null  int64
 1   id                                             16079 non-null  int64
 2   popularity                                    16079 non-null  float64
 3   vote_average                                16079 non-null  float64
 4   vote_count                                   16079 non-null  int64
 5   original_title_Assassin                     16079 non-null  bool
 6   original_title_Avatar: The Way of Water     16079 non-null  bool
 7   original_title_Champions                    16079 non-null  bool
 8   original_title_Creed III                    16079 non-null  bool
 9   original_title_Dungeons & Dragons: Honor Among Thieves 16079 non-null  bool
10  original_title_John Wick: Chapter 4          16079 non-null  bool
11  original_title_Momias                        16079 non-null  bool
12  original_title_Murder Mystery                16079 non-null  bool
13  original_title_Murder Mystery 2              16079 non-null  bool
14  original_title_On a Wing and a Prayer        16079 non-null  bool
15  original_title_Operation Fortune: Ruse de Guerre 16079 non-null  bool
16  original_title_Prom Pact                     16079 non-null  bool
17  original_title_Rye Lane                      16079 non-null  bool
18  original_title_Tetris                        16079 non-null  bool
19  original_title_김복순                        16079 non-null  bool
20  original_language_Chinese                    16079 non-null  bool
21  original_language_English                    16079 non-null  bool
```

✓ 0s completed at 12:21 PM

Step 7: Create Outliers

They identify and handle unusual values in a dataset.

We are using Z-score to handle the data

```
{x} ✓ 0s ▶ from scipy import stats

# Select only numerical columns
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Remove constant or problematic columns
numerical_df = numerical_df.loc[:, numerical_df.nunique() > 1]
numerical_df = numerical_df.dropna(axis=1)

# Calculate Z-scores
z_scores = stats.zscore(numerical_df)

# Handle cases with NaN Z-scores
z_scores = pd.DataFrame(z_scores, columns=numerical_df.columns).fillna(0)

# Identify rows with Z-scores > 3 or < -3
outliers = (abs(z_scores) > 3).any(axis=1)

# Filter the outliers
outlier_rows = df[outliers]
print(outlier_rows)
```

	Unnamed: 0	id	popularity	vote_average	vote_count	\
3	3	76600	10224.280	7.742	6335	
19	19	82856	1108.646	8.488	8697	
23	23	76600	10224.280	7.742	6335	
39	39	82856	1108.646	8.488	8697	
43	43	76600	10224.280	7.742	6335	
...
16039	16039	82856	1108.646	8.488	8697	
16043	16043	76600	10224.280	7.742	6335	
16059	16059	82856	1108.646	8.488	8697	
...

Step 8: Standardization and Normalization

Import StandardScaler and MinMaxScaler

```
✓ 0s [23] from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

Standardization (z-score scaling) transforms the data by subtracting the mean and dividing by the standard deviation for each feature.

```
# Select numerical columns
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Initialize the StandardScaler
scaler = StandardScaler()

# Standardize the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the results
print(df.head())
```

Unnamed: 0	id	popularity	vote_average	vote_count	\
0	-1.732158	0.192916	-0.313201	-0.696417	-0.366495
1	-1.731943	0.309711	0.270665	-0.318094	-0.205769
2	-1.731727	0.461279	-0.389096	0.154808	-0.403883
3	-1.731512	-1.526286	4.166043	0.194532	2.275593
4	-1.731296	0.837632	-0.403749	-0.601836	-0.430097

original_title_Assassin	original_title_Avatar: The Way of Water	\
0	False	False
1	False	False
2	False	False
3	False	True
4	False	False

Normalization scales numerical data to a fixed range, usually [0, 1]. Use MinMaxScaler for this process.

```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the results
print(df.head())
```

Unnamed: 0	id	popularity	vote_average	vote_count	\
0	0.000000	0.577957	0.020162	0.384615	0.021164
1	0.000062	0.617220	0.147883	0.461538	0.064182
2	0.000124	0.668174	0.003560	0.557692	0.011157
3	0.000187	0.000000	1.000000	0.565769	0.728318
4	0.000249	0.794696	0.000354	0.403846	0.004141

original_title_Assassin	original_title_Avatar: The Way of Water	\
0	False	False
1	False	False
2	False	False
3	False	True
4	False	False

Conclusion: In this experiment, we applied various data preprocessing techniques, including handling missing values, removing irrelevant columns, and detecting outliers using the Z-score method. We then scaled the numerical data using standardization (Z-score method) and normalization (Min-Max scaling) to bring all features onto a uniform scale.

Some Challenges we faced :

1. Handling Missing Data: Identifying the appropriate method to handle missing values and replacing them with mean, median, or mode.
2. Scaling and Normalization: Deciding between standardization and normalization for different features can be tricky. Using incorrect scaling methods may distort the data and affect model accuracy.
3. Selection of Columns: Determining which columns are relevant for the model and dropping them is challenging.