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

```
df=pd.read_csv("/content/vg.csv")
df
```

	index	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP
0	0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	
1	1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
2	2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	
3	3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	
4	4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
...	...	...	...	...	...	...	...	...	
3015	3015	Red Steel	Wii	2006.0	Shooter	Ubisoft	0.54	0.03	
3016	3016	Summer Sports: Paradise Island (Others sales)	Wii	2008.0	Sports	Ubisoft	0.00	0.66	
3017	3017	Need for Speed: Shift 2 Unleashed	PS3	2011.0	Racing	Electronic Arts	0.20	0.35	
3018	3018	The Fairly Odd Parents: Breakin Da Rules	PS2	2003.0	Platform	THQ	0.33	0.25	
3019	3019	Virtua Tennis: World Tour (US & Others sales)	PSP	2005.0	Sports	Sega	0.16	0.36	

3020 rows × 17 columns

Display basic information about the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3020 entries, 0 to 3019
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   index            3020 non-null   int64
 1   Name             3019 non-null   object
 2   Platform         3020 non-null   object
 3   Year_of_Release  2981 non-null   float64
 4   Genre            3019 non-null   object
 5   Publisher        3016 non-null   object
 6   NA_Sales         3020 non-null   float64
 7   EU_Sales         3020 non-null   float64
 8   JP_Sales         3020 non-null   float64
 9   Other_Sales      3020 non-null   float64
 10  Global_Sales     3020 non-null   float64
 11  Critic_Score     1988 non-null   float64
 12  Critic_Count     1988 non-null   float64
 13  User_Score       2112 non-null   object
 14  User_Count       2006 non-null   float64
 15  Developer        2121 non-null   object
 16  Rating           2114 non-null   object
dtypes: float64(9), int64(1), object(7)
memory usage: 401.2+ KB
```

```
df.head()
```

	index	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
0	0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	
1	1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
2	2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	
3	3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	
4	4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	

```
df.tail()
```

	index	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
3015	3015	Red Steel	Wii	2006.0	Shooter	Ubisoft	0.54	0.03	
3016	3016	Summer Sports: Paradise Island (Others sales)	Wii	2008.0	Sports	Ubisoft	0.00	0.66	
3017	3017	Need for Speed: Shift 2 Unleashed	PS3	2011.0	Racing	Electronic Arts	0.20	0.35	
3018	3018	The Fairly Odd Parents: Breakin Da Rules	PS2	2003.0	Platform	THQ	0.33	0.25	
3019	3019	Virtua Tennis: World Tour (US & Others sales)	PSP	2005.0	Sports	Sega	0.16	0.36	

Display the shape of the dataset

```
row,columns=df.shape
print("The number of rows are",row)
print("The number of columns are",columns)
```

```
The number of rows are 3020
The number of columns are 17
```

```
df.size
```

```
51340
```

```
df.columns
```

```
Index(['index', 'Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher',
       'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales',
       'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count', 'Developer',
       'Rating'],
      dtype='object')
```

```
df.dtypes
```

```
index              int64
Name              object
Platform          object
Year_of_Release  float64
Genre             object
Publisher         object
NA_Sales         float64
EU_Sales         float64
JP_Sales         float64
Other_Sales      float64
Global_Sales     float64
```

```
Critic_Score       float64
Critic_Count       float64
User_Score          object
User_Count         float64
Developer           object
Rating              object
dtype: object
```

```
df.isna().sum()
```

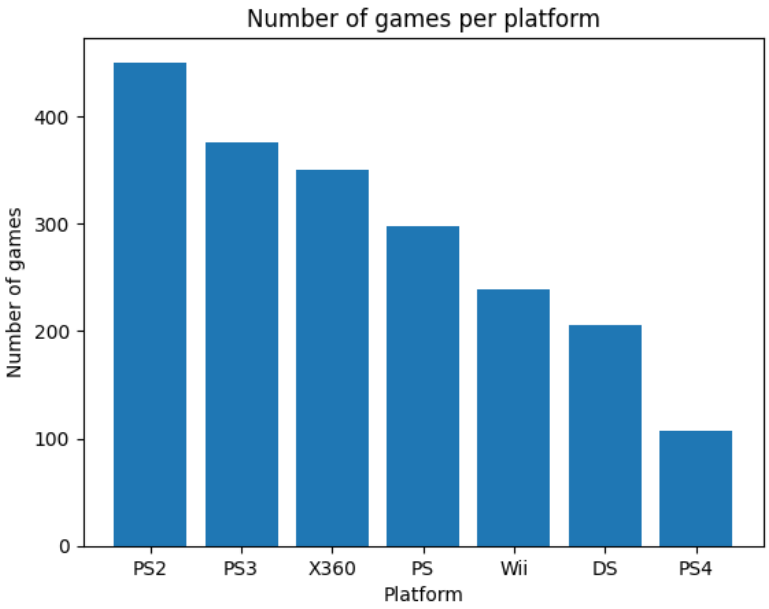
```
index                0
Name                 1
Platform             0
Year_of_Release     39
Genre                1
Publisher            4
NA_Sales             0
EU_Sales             0
JP_Sales             0
Other_Sales          0
Global_Sales         0
Critic_Score      1032
Critic_Count      1032
User_Score         908
User_Count        1014
Developer          899
Rating             906
dtype: int64
```

DATA VISUALIZATION

```
unique_values_in_platform=df['Platform'].value_counts()
print(unique_values_in_platform)
```

```
PS2     450
PS3     376
X360    350
PS      298
Wii     239
DS      206
PS4     107
PSP     106
GBA     103
XB      101
PC       95
NES      86
GC       73
3DS      71
N64      69
XOne     65
SNES     63
GB       61
2600     41
WiiU     29
GEN      12
PSV       9
DC        6
SAT       3
SCD       1
Name: Platform, dtype: int64
```

```
platform_counts = df["Platform"].value_counts().head(7)
plt.bar(platform_counts.index,platform_counts.values)
plt.xlabel("Platform")
plt.ylabel("Number of games")
plt.title("Number of games per platform")
plt.show()
```

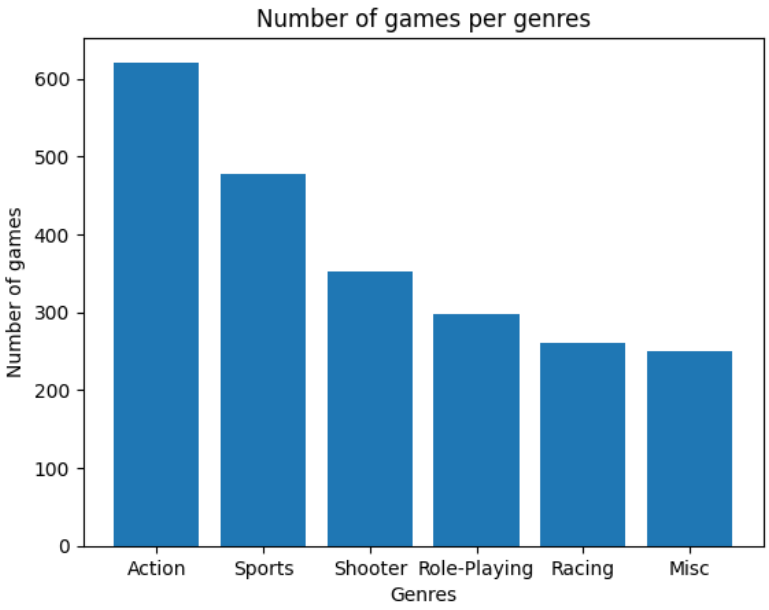


genres present in the dataset and their counts

```
genre_counts = df['Genre'].value_counts()
genre_counts
```

```
Action          621
Sports          477
Shooter         352
Role-Playing    297
Racing          261
Misc            250
Platform        244
Fighting        184
Simulation      135
Puzzle           77
Adventure        69
Strategy         52
Name: Genre, dtype: int64
```

```
genres_counts = df["Genre"].value_counts().head(6)
plt.bar(genres_counts.index,genres_counts.values)
plt.xlabel("Genres")
plt.ylabel("Number of games")
plt.title("Number of games per genres")
plt.show()
```

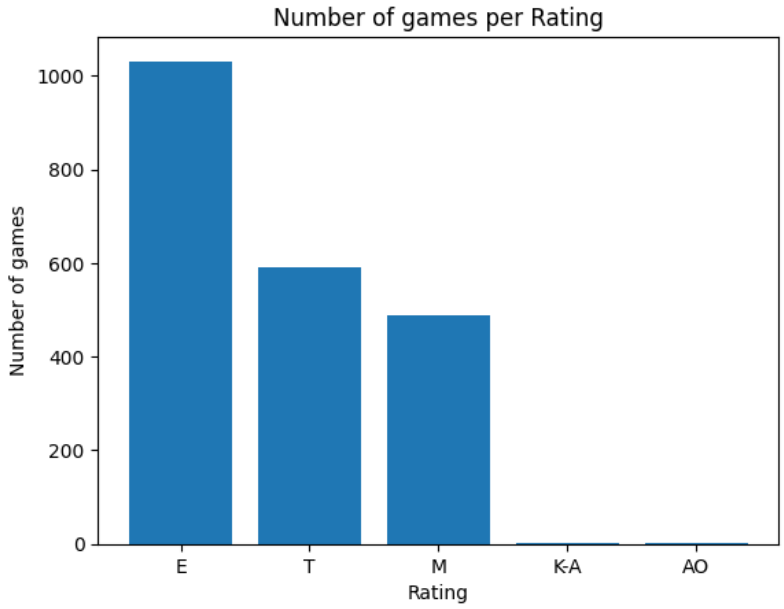


distribution of games across different ratings

```
rating_countsrating_counts=df['Rating'].value_counts()
rating_countsrating_counts
```

```
E      1031
T       591
M       489
K-A       2
AO        1
Name: Rating, dtype: int64
```

```
rating_counts = df["Rating"].value_counts().head(6)
plt.bar(rating_counts.index,rating_counts.values)
plt.xlabel("Rating")
plt.ylabel("Number of games")
plt.title("Number of games per Rating")
plt.show()
```



games were released each year

```
games_per_year=df['Year_of_Release'].value_counts().sort_values(ascending=True)
games_per_year
```

```
1980.0      5
1983.0      9
1985.0      9
1987.0     10
1982.0     11
1988.0     12
1984.0     13
1989.0     14
1990.0     14
1991.0     14
1981.0     18
1986.0     18
1993.0     21
1992.0     22
1995.0     28
1994.0     30
2016.0     44
1996.0     59
1997.0     77
2000.0     81
1999.0     84
2015.0     94
1998.0    103
2014.0    112
2001.0    113
2013.0    122
2006.0    133
2002.0    138
2012.0    138
2003.0    140
2005.0    142
2004.0    150
2011.0    164
2010.0    181
2009.0    199
2007.0    210
2008.0    249
Name: Year_of_Release, dtype: int64
```

total global sales for each genre

```
total_sales_per_genre = df.groupby('Genre')['Global_Sales'].sum().sort_values(ascending=False)
total_sales_per_genre
```

```
Genre
Action          1225.31
Sports           937.52
Shooter          863.66
Role-Playing     718.84
Platform         693.98
Racing           549.55
Misc             540.12
Fighting         320.58
Simulation       261.01
Puzzle           174.11
Adventure        113.87
Strategy          82.64
Name: Global_Sales, dtype: float64
```

top 10 games with the highest global sales

```
top_10_global_sales = df[['Name', 'Global_Sales']].head(10)
top_10_global_sales
```

	Name	Global_Sales
0	Wii Sports	82.53
1	Super Mario Bros.	40.24
2	Mario Kart Wii	35.52
3	Wii Sports Resort	32.77
4	Pokemon Red/Pokemon Blue	31.37
5	Tetris	30.26
6	New Super Mario Bros.	29.80
7	Wii Play	28.92
8	New Super Mario Bros. Wii	28.32
9	Duck Hunt	28.31

How many games fall under each ESRB rating category

```
esrb_counts = df['Rating'].value_counts()
esrb_counts
```

```
E      1031
T       591
M       489
K-A       2
AO        1
Name: Rating, dtype: int64
```

How many games were developed and published by each company

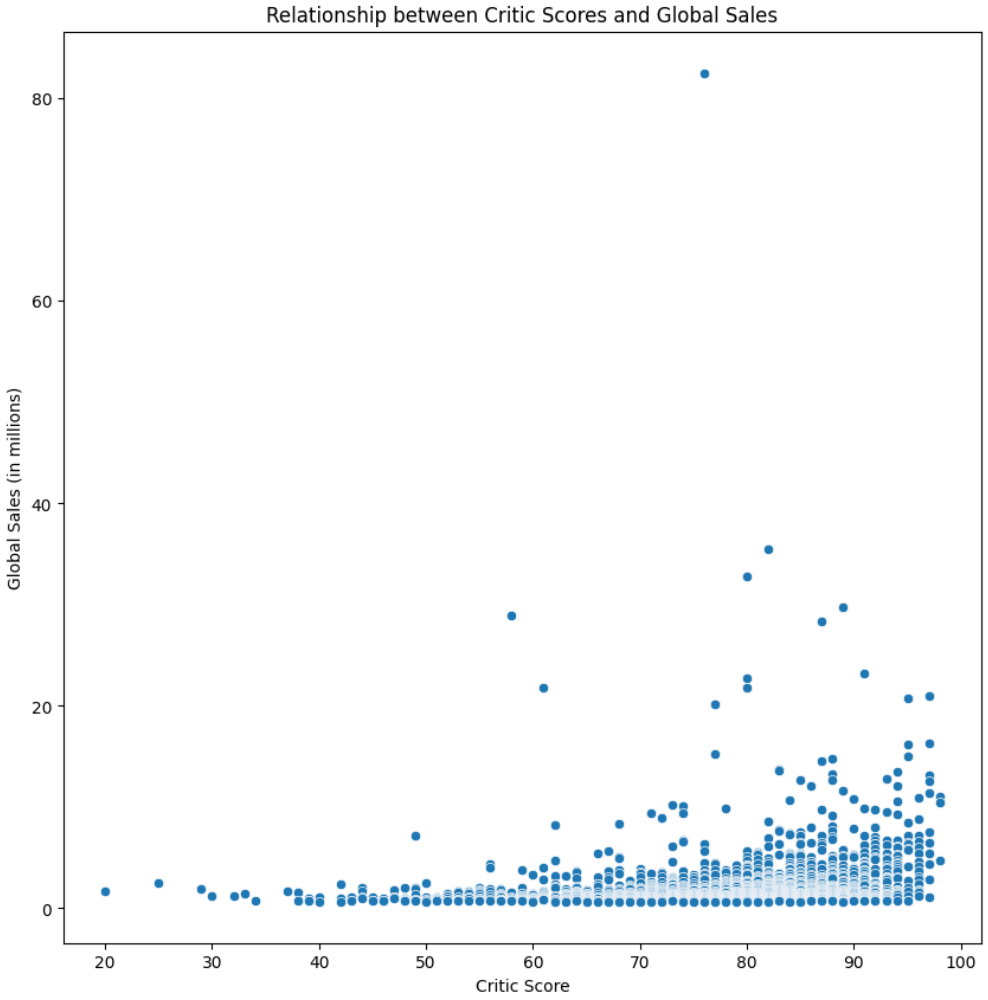
```
developer_counts = df['Developer'].value_counts()
publisher_counts = df['Publisher'].value_counts()
developer_counts, publisher_counts
```

```
(EA Sports                                82
 Nintendo                                 62
 EA Canada                                55
 EA Tiburon                               49
 Capcom                                   47
                                          ..
 Spike Chunsoft Co. Ltd., Spike Chunsoft   1
 Chris Sawyer                              1
 49Games                                   1
 neo Software                              1
 Engine Software                           1
 Name: Developer, Length: 531, dtype: int64,
 Electronic Arts               504
 Nintendo                      395
 Activision                    232
 Sony Computer Entertainment   222
 Ubisoft                       175
```

```
                                ...
Zoo Digital Publishing           1
Ocean                            1
Black Label Games                1
Alchemist                        1
Metro 3D                         1
Name: Publisher, Length: 139, dtype: int64)
```

relationship between critic scores and global sales

```
plt.figure(figsize=(10, 10))
sns.scatterplot(x='Critic_Score', y='Global_Sales', data=df)
plt.title('Relationship between Critic Scores and Global Sales')
plt.xlabel('Critic Score')
plt.ylabel('Global Sales (in millions)')
plt.show()
```



variation of sales across different regions (NA, EU, JP, Other)

```
sales_summary = df[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']].describe()
sales_summary
```

	NA_Sales	EU_Sales	JP_Sales	Other_Sales
count	3020.000000	3020.000000	3020.000000	3020.000000
mean	1.054593	0.620705	0.271487	0.200272
std	1.686798	1.052360	0.671041	0.401989
min	0.000000	0.000000	0.000000	0.000000
25%	0.410000	0.180000	0.000000	0.060000
50%	0.670000	0.370000	0.010000	0.110000
75%	1.150000	0.670000	0.210000	0.210000
max	41.360000	28.960000	10.220000	10.570000

How many games have both critic and user scores available

```
games_with_scores = df[['Critic_Score', 'User_Score']].dropna()
games_with_scores_count = games_with_scores.shape[0]
games_with_scores_count
```

```
1983
```

proportion of games released for each platform

```
platform_proportions = df['Platform'].value_counts(normalize=True)
platform_proportions
```

```
PS2     0.149007
PS3     0.124503
X360    0.115894
PS      0.098675
Wii     0.079139
DS      0.068212
PS4     0.035430
PSP     0.035099
GBA     0.034106
XB      0.033444
PC      0.031457
NES     0.028477
GC      0.024172
3DS     0.023510
N64     0.022848
XOne    0.021523
SNES    0.020861
GB      0.020199
2600    0.013576
WiiU    0.009603
GEN     0.003974
PSV     0.002980
DC      0.001987
SAT     0.000993
SCD     0.000331
Name: Platform, dtype: float64
```

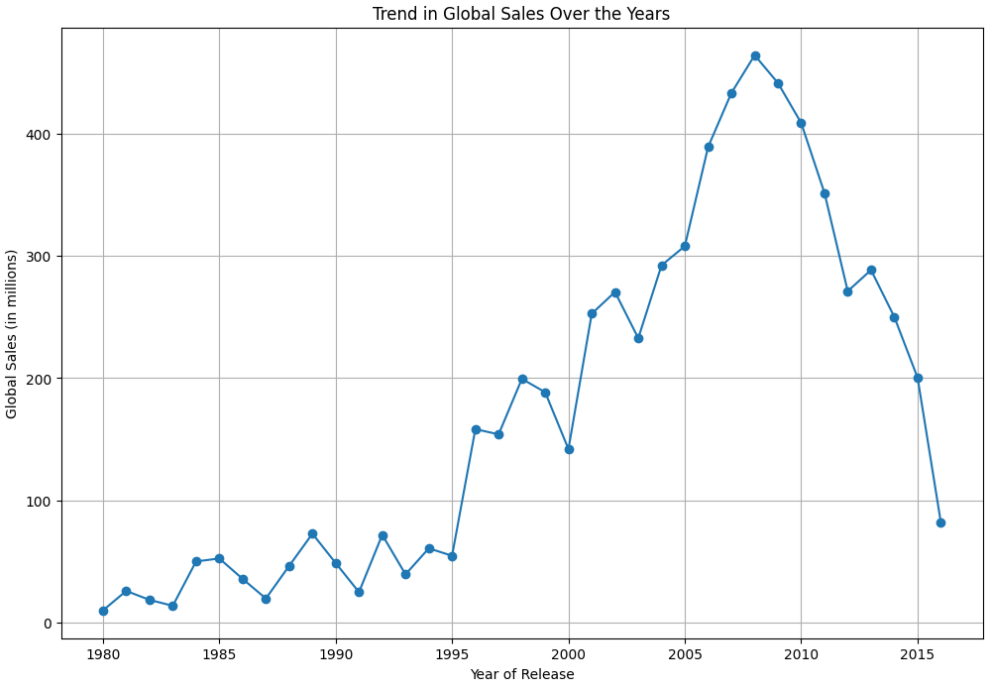
average number of critic and user reviews for games

```
average_reviews = df[['Critic_Count', 'User_Count']].mean()
average_reviews
```

```
Critic_Count     39.478370
User_Count      375.826022
dtype: float64
```

the trend in global sales over the years

```
plt.figure(figsize=(12, 8))
df.groupby('Year_of_Release')['Global_Sales'].sum().plot(marker='o')
plt.title('Trend in Global Sales Over the Years')
plt.xlabel('Year of Release')
plt.ylabel('Global Sales (in millions)')
plt.grid(True)
plt.show()
```



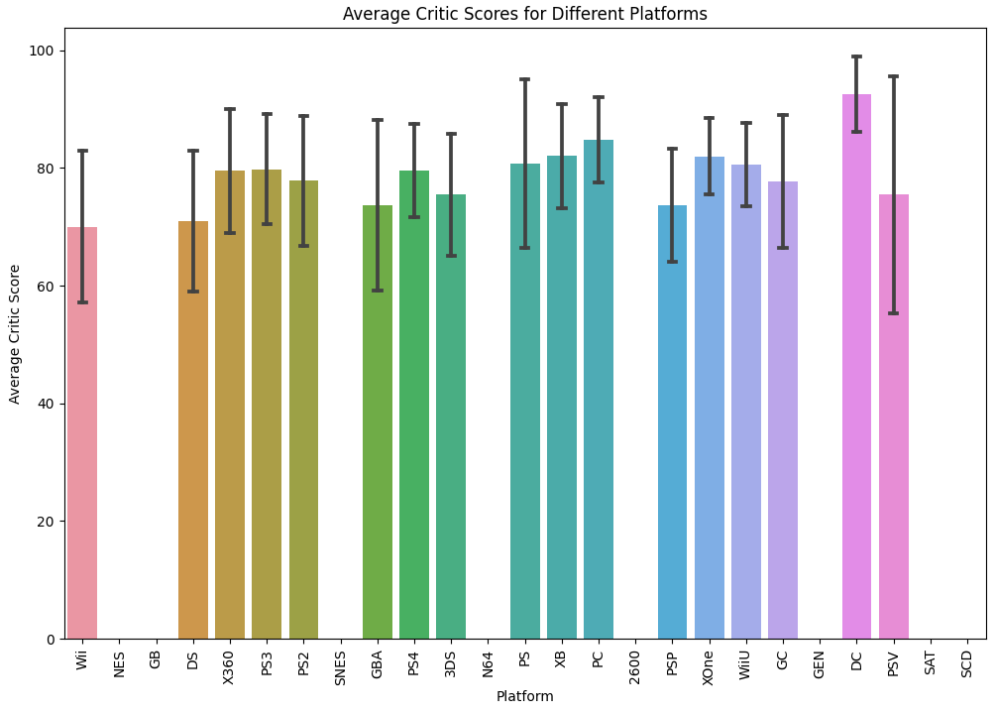
How do the average critic scores vary for different platforms

```
plt.figure(figsize=(12, 8))
sns.barplot(x='Platform', y='Critic_Score', data=df, ci='sd', capsize=0.2)
plt.title('Average Critic Scores for Different Platforms')
plt.xlabel('Platform')
plt.ylabel('Average Critic Score')
plt.xticks(rotation=90)
plt.show()
```

```
<ipython-input-93-0bc8586f1817>:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  sns.barplot(x='Platform', y='Critic_Score', data=df, ci='sd', capsize=0.2)
```



DEALING WITH MISSING VALUES

```
df.isna().sum()
```

```
index                0
Name                 1
Platform             0
Year_of_Release     39
Genre                1
Publisher            4
NA_Sales             0
EU_Sales             0
JP_Sales             0
Other_Sales          0
Global_Sales         0
Critic_Score      1032
Critic_Count      1032
User_Score         908
User_Count        1014
Developer          899
Rating             906
dtype: int64
```

```
df["Name"]=df["Name"].fillna(df["Name"].mode().iloc[0])
df['Year_of_Release']=df['Year_of_Release'].fillna(df['Year_of_Release'].mean())
df["Genre"]=df["Genre"].fillna(df["Genre"].mode().iloc[0])
df["Publisher"]=df["Publisher"].fillna(df["Publisher"].mode().iloc[0])
df["Critic_Score"]=df["Critic_Score"].fillna(df["Critic_Score"].mean())
df["Critic_Count"]=df["Critic_Count"].fillna(df["Critic_Count"].mean())
df["User_Count"]=df["User_Count"].fillna(df["User_Count"].mean())
df["Developer"]=df["Developer"].fillna(df["Developer"].mode().iloc[0])
df["Rating"]=df["Rating"].fillna(df["Rating"].mode().iloc[0])
```

Assuming df is your DataFrame and 'column_name' is the column with mixed types,'coerce' will replace non-numeric values with NaN

```
df['User_Score'] = pd.to_numeric(df['User_Score'], errors='coerce')
df["User_Score"]=df["User_Score"].fillna(df["User_Score"].mean())
```

```
df.isna().sum()
```

```
index              0
Name               0
Platform           0
Year_of_Release    0
Genre              0
Publisher          0
NA_Sales           0
EU_Sales           0
JP_Sales           0
Other_Sales        0
Global_Sales       0
Critic_Score       0
Critic_Count       0
User_Score         0
User_Count         0
Developer          0
Rating             0
dtype: int64
```

ENCODING

```
df.dtypes
```

```
index              int64
Name              object
Platform          object
Year_of_Release  float64
Genre             object
Publisher         object
NA_Sales         float64
EU_Sales         float64
JP_Sales         float64
Other_Sales      float64
Global_Sales     float64
Critic_Score     float64
Critic_Count     float64
User_Score       float64
User_Count       float64
Developer         object
Rating            object
dtype: object
```

```
dummy=pd.get_dummies(df[['Name','Platform','Genre','Publisher','Developer']],drop_first=True)
dummy
```

	Name_ Tales of Xillia 2	Name_.hack//Infection Part 1	Name_.hack//Mutation Part 2	Name_007: Quantum of Solace	Name_007: The World is not Enough	Name_007: Tomorrow Never Dies	Na Snc
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	0	
...	...	...	...	...	...	...	
3015	0	0	0	0	0	0	
3016	0	0	0	0	0	0	
3017	0	0	0	0	0	0	
3018	0	0	0	0	0	0	
3019	0	0	0	0	0	0	

3020 rows × 2956 columns

```
df1=pd.concat([df,dummy],axis=1)
df1
```

	index	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP
0	0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	
1	1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	
2	2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	
3	3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	
4	4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	
...	...	...	...	...	...	...	...	...	
3015	3015	Red Steel	Wii	2006.0	Shooter	Ubisoft	0.54	0.03	
3016	3016	Summer Sports: Paradise Island (Others sales)	Wii	2008.0	Sports	Ubisoft	0.00	0.66	
3017	3017	Need for Speed: Shift 2 Unleashed	PS3	2011.0	Racing	Electronic Arts	0.20	0.35	
3018	3018	The Fairly Odd Parents: Breakin Da Rules	PS2	2003.0	Platform	THQ	0.33	0.25	
3019	3019	Virtua Tennis: World Tour (US & Others sales)	PSP	2005.0	Sports	Sega	0.16	0.36	

3020 rows × 2973 columns

```
df1=df1.drop(['Name','Platform','Genre','Publisher','Developer'],axis=1)
df1
```

	index	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_S
0	0	2006.0	41.36	28.96	3.77	8.45	82.53	76.00
1	1	1985.0	29.08	3.58	6.81	0.77	40.24	77.65
2	2	2008.0	15.68	12.76	3.79	3.29	35.52	82.00
3	3	2009.0	15.61	10.93	3.28	2.95	32.77	80.00
4	4	1996.0	11.27	8.89	10.22	1.00	31.37	77.65
...	...	...	...	...	...	...	...	
3015	3015	2006.0	0.54	0.03	0.04	0.05	0.67	63.00
3016	3016	2008.0	0.00	0.66	0.00	0.01	0.67	77.65
3017	3017	2011.0	0.20	0.35	0.00	0.12	0.67	77.65
3018	3018	2003.0	0.33	0.25	0.00	0.09	0.67	77.65
3019	3019	2005.0	0.16	0.36	0.00	0.14	0.67	77.65

3020 rows × 2968 columns

```
df1.dtypes
```

```
index                                int64
Year_of_Release                    float64
NA_Sales                           float64
EU_Sales                           float64
JP_Sales                           float64
                                    ...
Developer_id Software                uint8
```

```
Developer_id Software, Nerve Software      uint8
Developer_n-Space                          uint8
Developer_neo Software                     uint8
Developer_syn Sophia                       uint8
Length: 2968, dtype: object
```

```
x=df1.drop(['Rating'],axis=1)
x
```

	index	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_S
0	0	2006.0	41.36	28.96	3.77	8.45	82.53	76.00
1	1	1985.0	29.08	3.58	6.81	0.77	40.24	77.65
2	2	2008.0	15.68	12.76	3.79	3.29	35.52	82.00
3	3	2009.0	15.61	10.93	3.28	2.95	32.77	80.00
4	4	1996.0	11.27	8.89	10.22	1.00	31.37	77.65
...	...	...	...	...	...	...	...	
3015	3015	2006.0	0.54	0.03	0.04	0.05	0.67	63.00
3016	3016	2008.0	0.00	0.66	0.00	0.01	0.67	77.65
3017	3017	2011.0	0.20	0.35	0.00	0.12	0.67	77.65
3018	3018	2003.0	0.33	0.25	0.00	0.09	0.67	77.65
3019	3019	2005.0	0.16	0.36	0.00	0.14	0.67	77.65

3020 rows × 2967 columns

```
y=df1['Rating']
y
```

```
0       E
1       E
2       E
3       E
4       E
       ..
3015    T
3016    E
3017    E
3018    E
3019    E
Name: Rating, Length: 3020, dtype: object
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42)
x_train
```

	index	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_S
1657	1657	2002.0	1.03	0.14	0.00	0.04	1.21	86.00
2335	2335	2014.0	0.48	0.33	0.00	0.08	0.89	77.65
157	157	1988.0	2.97	0.69	1.81	0.11	5.58	77.65
1953	1953	2001.0	1.03	0.02	0.00	0.00	1.06	87.00
2223	2223	2009.0	0.40	0.43	0.00	0.10	0.93	85.00
...	...	...	...	...	...	...	...	
1638	1638	1997.0	0.13	0.07	1.00	0.02	1.22	77.65
1095	1095	2013.0	0.83	0.66	0.00	0.18	1.67	86.00
1130	1130	2014.0	1.35	0.11	0.00	0.17	1.63	77.65
1294	1294	2012.0	0.71	0.43	0.07	0.26	1.47	33.00
860	860	2010.0	0.79	0.88	0.00	0.32	1.98	80.00

2265 rows × 2967 columns

```
x_test
```

	index	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_S
1412	1412	2010.0	0.80	0.45	0.00	0.12	1.37	80.00
1564	1564	2011.0	0.74	0.40	0.00	0.12	1.27	71.00
203	203	1996.0	2.05	1.16	1.11	0.73	5.05	91.00
1407	1407	2009.0	1.26	0.02	0.00	0.10	1.38	60.00
52	52	1997.0	4.02	3.87	2.54	0.52	10.95	96.00
...	...	...	...	...	...	...	...	
2925	2925	2005.0	0.52	0.00	0.12	0.05	0.70	66.00
1710	1710	2013.0	0.48	0.50	0.01	0.20	1.18	78.00
857	857	1989.0	0.75	0.30	0.90	0.04	1.99	77.65
2417	2417	2009.0	0.46	0.21	0.12	0.06	0.86	76.00
461	461	2008.0	1.91	0.84	0.00	0.29	3.04	75.00

755 rows × 2967 columns

```
y_train
```

```
1657    E
2335    E
157     E
1953    M
2223    T
       ..
1638    E
1095    T
1130    E
1294    M
860     T
Name: Rating, Length: 2265, dtype: object
```

```
y_test
```

```
1412    M
1564    T
203     M
1407    E
52      E
       ..
2925    E
1710    T
857     E
2417    E
461     E
Name: Rating, Length: 755, dtype: object
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
```

```
▾ StandardScaler
StandardScaler()
```

```
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=8)
knn.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8)
```

```
y_pred=knn.predict(x_test)
y_pred
```

```
array(['E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'M', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
```

```
mat=confusion_matrix(y_pred,y_test)
mat
```

```
array([[487, 117, 138],
       [  0,   7,   0],
       [  0,   0,   6]])
```

```
score=accuracy_score(y_pred,y_test)
score
```

```
0.6622516556291391
```

SVM

```
from sklearn.svm import SVC
model=SVC()
model.fit(x_train,y_train)
```

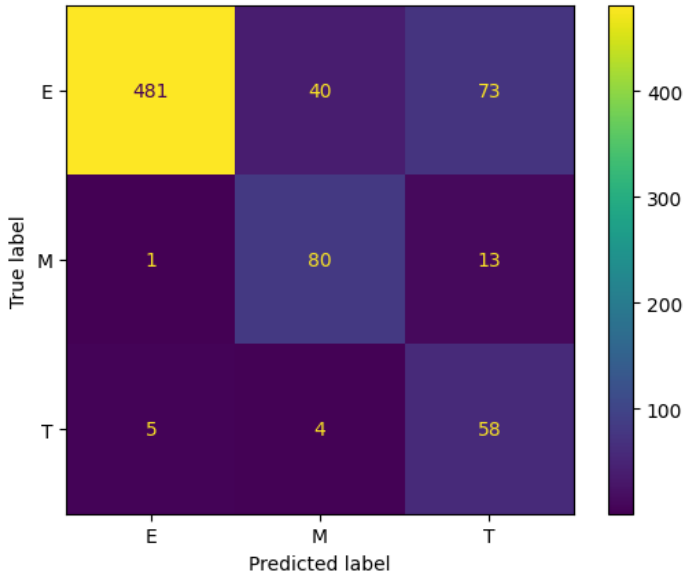
```
▾ SVC
SVC()
```

```
y_pred=model.predict(x_test)
y_pred
```

```
array(['E', 'E', 'T', 'E', 'T', 'M', 'E', 'E', 'M', 'M', 'E', 'M', 'E',
       'M', 'E', 'T', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
       'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'T', 'M', 'E', 'E',
       'E', 'E', 'E', 'E', 'M', 'M', 'E', 'E', 'E', 'E', 'T', 'E', 'E',
       'E', 'E', 'E', 'E', 'M', 'M', 'M', 'E', 'M', 'M', 'E', 'M', 'E',
       'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'M', 'M', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'M', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'T', 'E', 'E',
       'T', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'T',
       'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
       'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
       'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'T', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'T', 'M',
       'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'M', 'M', 'E', 'E', 'T', 'T', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E',
       'E', 'M', 'E', 'E', 'E', 'E', 'M', 'E', 'M', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
       'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'M', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'T', 'T', 'E', 'E', 'E',
       'E', 'E', 'T', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'M', 'T', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
       'T', 'E', 'E', 'E', 'M', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'M',
       'E', 'M', 'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'T',
       'T', 'E', 'E', 'E', 'E', 'E', 'M', 'T', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'M', 'E', 'E', 'E',
       'M', 'E', 'E', 'E', 'E', 'M', 'M', 'M', 'E', 'E', 'E', 'E', 'T',
       'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'M', 'T', 'M', 'E', 'E', 'T', 'E', 'E', 'M', 'E', 'E',
       'M', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E',
       'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'M', 'M',
       'M', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'M', 'E', 'E', 'E', 'M',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'T', 'E',
       'M', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
       'T', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'M', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'T', 'E', 'M', 'E', 'E', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'M',
       'E', 'E', 'E', 'T', 'E', 'E', 'E', 'T', 'E', 'E', 'T', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
       'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'M', 'E', 'E',
       'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
       'E', 'T', 'M', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'T', 'E',
       'M', 'E', 'T', 'E', 'E', 'M', 'E', 'E', 'E', 'M', 'E', 'E', 'E',
       'E', 'E', 'E', 'E', 'E', 'E', 'M', 'T', 'M', 'M', 'E', 'E', 'T', 'E',
       'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
       'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E',
       'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
       'E'], dtype=object)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,ConfusionMatrixDisplay
mat=confusion_matrix(y_pred,y_test)
label=['E','M','T']
cmd=ConfusionMatrixDisplay(mat,display_labels=label)
cmd.plot()
print(mat)
```

```
[[481  40  73]
 [  1  80  13]
 [  5   4  58]]
```




```
score=accuracy_score(y_pred,y_test)
score
```

```
0.8198675496688742
```

```
report=classification_report(y_pred,y_test)
report
```

```
'              precision    recall  f1-score   support\n\n           E       0.99      0.81
0.89       594\n           M       0.65      0.85      0.73        94\n           T       0.40
0.87      0.55        67\n\n    accuracy                           0.82       755\n   macro av
g       0.68      0.84      0.72       755\nweighted avg       0.89      0.82      0.84
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(max_iter=1000)
model.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred=model.predict(x_test)
y_pred
```

```
'E', 'E', 'T', 'M', 'M', 'M', 'E', 'E', 'M', 'M', 'E', 'M', 'E',
'M', 'E', 'T', 'M', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E',
'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'T', 'T', 'M', 'E', 'E',
'E', 'E', 'E', 'E', 'M', 'M', 'E', 'E', 'E', 'T', 'T', 'E', 'E',
'E', 'E', 'E', 'E', 'M', 'M', 'M', 'E', 'M', 'M', 'E', 'M', 'E',
'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E',
'E', 'M', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E',
'E', 'M', 'M', 'M', 'M', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E',
'E', 'M', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'T', 'E', 'E',
'T', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
'M', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'T',
'E', 'E', 'T', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'T', 'M',
'M', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E',
'E', 'E', 'E', 'M', 'M', 'M', 'E', 'T', 'T', 'E', 'E', 'E', 'T',
'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E',
'E', 'M', 'T', 'T', 'E', 'E', 'M', 'E', 'M', 'E', 'T', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E',
'E', 'M', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
'M', 'E', 'T', 'E', 'E', 'E', 'E', 'T', 'T', 'E', 'E', 'E', 'E',
'E', 'E', 'T', 'T', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'M', 'E',
'E', 'M', 'T', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
'M', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
'T', 'E', 'T', 'E', 'M', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'M',
```

```
 'E', 'E', 'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'T',
'T', 'E', 'E', 'E', 'E', 'E', 'M', 'T', 'E', 'E', 'E', 'E', 'E',
'M', 'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'M', 'E', 'E', 'E',
'M', 'E', 'E', 'E', 'E', 'M', 'M', 'M', 'M', 'E', 'E', 'E', 'T',
'E', 'T', 'T', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'M', 'T', 'M', 'E', 'E', 'T', 'E', 'E', 'M', 'T', 'E',
'M', 'T', 'M', 'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E',
'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'M', 'E', 'M', 'M',
'M', 'E', 'E', 'E', 'M', 'E', 'T', 'E', 'E', 'M', 'E', 'E', 'E',
'E', 'E', 'M', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'T', 'E', 'M', 'M', 'E', 'E', 'E', 'T',
'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'T', 'T', 'E',
'M', 'M', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E',
'T', 'E', 'E', 'E', 'T', 'E', 'T', 'E', 'E', 'E', 'M', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'T', 'E', 'M', 'E', 'M', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'M', 'M', 'T',
'E', 'E', 'T', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E',
'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
'M', 'E', 'M', 'E', 'E', 'E', 'E', 'E', 'M', 'M', 'E', 'E', 'E',
'E', 'E', 'M', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
'E', 'T', 'M', 'E', 'E', 'E', 'E', 'E', 'T', 'T', 'E', 'T', 'E',
'M', 'E', 'T', 'E', 'E', 'M', 'E', 'E', 'E', 'M', 'E', 'E', 'E',
'E', 'E', 'E', 'E', 'E', 'M', 'T', 'M', 'M', 'E', 'E', 'T', 'E',
'T', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M',
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'T', 'E', 'E',
'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
'E'], dtype=object)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
```

```
mat=confusion_matrix(y_pred,y_test)
mat
```

```
array([[477,  17,  54],
       [  2,  95,  15],
       [  8,  12,  75]])
```

```
score=accuracy_score(y_pred,y_test)
score
```

```
0.856953642384106
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_classifier = RandomForestClassifier(n_estimators=100,random_state=42)
rf_classifier.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred = rf_classifier.predict(x_test)
y_pred
```

```
'T', 'E', 'T', 'M', 'M', 'M', 'E', 'E', 'T', 'M', 'E', 'M', 'E',
'M', 'T', 'T', 'M', 'E', 'E', 'E', 'T', 'E', 'E', 'E', 'E', 'M',
'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'M', 'E', 'E',
```