Import library

In [1]:

```
import numpy as np
import pandas as pd
```

Read dataset

In [2]:

```
df_movie = pd.read_csv('D:\dataset\Amazon - Movies and TV Ratings.csv')
```

describe

In [3]:

```
df_movie.describe()
```

Out[3]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198	Movie199	M
count	1.0	1.0	1.0	2.0	29.000000	1.0	1.0	1.0	1.0	1.0	...	5.000000	2.0	1.0	8
mean	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.800000	5.0	5.0	4
std	NaN	NaN	NaN	0.0	1.496301	NaN	NaN	NaN	NaN	NaN	...	1.643168	0.0	NaN	0
min	5.0	5.0	2.0	5.0	1.000000	4.0	5.0	5.0	5.0	5.0	...	1.000000	5.0	5.0	4
25%	5.0	5.0	2.0	5.0	4.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	4
50%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	5
75%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5
max	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5

8 rows × 206 columns

Info

In [4]:

```
df_movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4848 entries, 0 to 4847
Columns: 207 entries, user_id to Movie206
dtypes: float64(206), object(1)
memory usage: 7.7+ MB
```

Count

In [5]:

```
df_movie.count()
```

Out[5]:

```
user_id      4848
Movie1          1
```

```
Movie1          1
Movie2          1
Movie3          1
Movie4          2
              ...
Movie202        6
Movie203        1
Movie204        8
Movie205       35
Movie206       13
Length: 207, dtype: int64
```

In [6]:

```
#add column count#
#df_movie['count']=df_movie.count()
```

In [7]:

```
#print the columns
df_movie.columns
```

Out[7]:

```
Index(['user_id', 'Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6',
       'Movie7', 'Movie8', 'Movie9',
       ...
       'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
       'Movie203', 'Movie204', 'Movie205', 'Movie206'],
      dtype='object', length=207)
```

In [8]:

```
#print user id
df_movie['user_id']
```

Out[8]:

```
0       A3R5OBKS7OM2IR
1        AH3QC2PC1VTGP
2       A3LKP6WPMP9UKX
3        AVIY68KEPQ5ZD
4       A1CV1WROP5KTTW
              ...
4843    A1IMQ9WMFYKWH5
4844    A1KLIKPUF5E88I
4845     A5HG6WFZLO10D
4846    A3UU690TWXCG1X
4847     AI4J762YI6S06
Name: user_id, Length: 4848, dtype: object
```

High Viewership having lowest count of no rating

In [9]:

```
df_movie_no_rating = df_movie.isnull().sum().sort_values(ascending=True)
df_movie_no_rating = df_movie_no_rating[df_movie_no_rating>0]
df_movie_no_rating.head(5)
```

Out[9]:

```
Movie127    2535
Movie140    4270
Movie16     4528
Movie103    4576
Movie29     4605
dtype: int64
```

Lowest Viewership having highest count of no rating

Lowest viewership having highest count of no rating

In [10]:

```
df_movie_no_rating.tail(5)
```

Out[10]:

```
Movie2      4847
Movie66     4847
Movie67     4847
Movie153    4847
Movie146    4847
dtype: int64
```

In [11]:

```
#replace null values by 0
df_movie.fillna(0)
#df1.fillna(df1.mean()) replace with mean
```

Out[11]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	...	Movie197	Movie198	Movi
0	A3R5OBKS7OM2IR	5.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	AH3QC2PC1VTGP	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	A3LKP6WPMP9UKX	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	AVIY68KEPQ5ZD	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	A1CV1WROP5KTTW	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4843	A1IMQ9WMFYKWH5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4844	A1KLIKPUF5E88I	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4845	A5HG6WFZLO10D	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4846	A3UU690TWXCG1X	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4847	AI4J762YI6S06	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

4848 rows × 207 columns

In [12]:

```
#describe
df_movie.describe()
```

Out[12]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198	Movie199	M
count	1.0	1.0	1.0	2.0	29.000000	1.0	1.0	1.0	1.0	1.0	...	5.000000	2.0	1.0	8
mean	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.800000	5.0	5.0	4
std	NaN	NaN	NaN	0.0	1.496301	NaN	NaN	NaN	NaN	NaN	...	1.643168	0.0	NaN	0
min	5.0	5.0	2.0	5.0	1.000000	4.0	5.0	5.0	5.0	5.0	...	1.000000	5.0	5.0	4
25%	5.0	5.0	2.0	5.0	4.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	4
50%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	5
75%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5
max	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5

8 rows × 206 columns

Convert movie column to row

In [13]:

```
final=pd.melt(df_movie, id_vars =['user_id'], value_vars =df_movie.columns[1:].values)
```

print first five records

In [14]:

```
final.head(5)
```

Out[14]:

	user_id	variable	value
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	NaN
2	A3LKP6WPMP9UKX	Movie1	NaN
3	AVIY68KEPQ5ZD	Movie1	NaN
4	A1CV1WROP5KTTW	Movie1	NaN

Info

In [15]:

```
final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 998688 entries, 0 to 998687
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   user_id   998688 non-null  object 
 1   variable  998688 non-null  object 
 2   value     5000 non-null    float64
dtypes: float64(1), object(2)
memory usage: 22.9+ MB
```

Print last five records

In [16]:

```
final.tail(5)
```

Out[16]:

	user_id	variable	value
998683	A1IMQ9WMFYKWH5	Movie206	5.0
998684	A1KLIKPUF5E88I	Movie206	5.0
998685	A5HG6WFZLO10D	Movie206	5.0
998686	A3UU690TWXCG1X	Movie206	5.0
998687	AI4J762YI6S06	Movie206	5.0

movie rating

In [17]:

```
df_group = final.groupby(['value','variable'])['value'].mean().sort_values(ascending=False)
```

Movie with high rating

In [18]:

```
df_group.head()
```

Out[18]:

```
value  variable
5.0    Movie99     5.0
       Movie187    5.0
       Movie162    5.0
       Movie161    5.0
       Movie160    5.0
Name: value, dtype: float64
```

Movie with low rating

In [19]:

```
df_group.tail()
```

Out[19]:

```
value  variable
1.0    Movie89     1.0
       Movie90     1.0
       Movie91     1.0
       Movie95     1.0
       Movie103    1.0
Name: value, dtype: float64
```

Replace NAN with zero

In [20]:

```
df_final = final.fillna(0)
```

Rating

In [21]:

```
df_rating = df_final.groupby('value').size()
df_rating.head(10)
```

Out[21]:

```
value
0.0    993688
1.0       363
2.0       185
3.0       272
4.0       521
5.0      3659
dtype: int64
```

In [22]:

```
df_rating.agg({'value':'mean'})
```

Out[22]:

```
value    166448.0
dtype: float64
```

dtype: float64

In [23]:

```
#replace NAN with mean  does not run  looks like infinite loop
#df_mean = final.fillna(final.mean())

#remove null values
df_clean=final[final['value'].notnull()]
df_clean.shape
```

Out[23]:

```
(5000, 3)
```

In [24]:

```
#print first five record
df_final.head(5)
```

Out[24]:

	user_id	variable	value
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	0.0
2	A3LKP6WPMP9UKX	Movie1	0.0
3	AVIY68KEPQ5ZD	Movie1	0.0
4	A1CV1WROP5KTTW	Movie1	0.0

In [25]:

```
#print last five records
df_final.tail(5)
```

Out[25]:

	user_id	variable	value
998683	A1IMQ9WMFYKWH5	Movie206	5.0
998684	A1KLIKPUF5E88I	Movie206	5.0
998685	A5HG6WFZLO10D	Movie206	5.0
998686	A3UU690TWXCG1X	Movie206	5.0
998687	AI4J762YI6S06	Movie206	5.0

In [26]:

```
#Select features from the dataset to create the model
feature_select_cols = ['user_id','variable']
#Create the feature object
X_feature = df_final[feature_select_cols] # with null replaced with zeros
X_clean = df_clean[feature_select_cols]   # null values removed
```

print feature data

In [27]:

```
X_feature
```

Out[27]:

	user_id	variable
0	A3R5OBKS7OM2IR	Movie1

	user_id	variable
1	AH3QC2PC1VTGP	Movie1
2	A3LKP6WPMP9UKX	Movie1
3	AVIY68KEPQ5ZD	Movie1
4	A1CV1WROP5KTTW	Movie1
...	...	...
998683	A1IMQ9WMFYKWH5	Movie206
998684	A1KLIKPUF5E88I	Movie206
998685	A5HG6WFZLO10D	Movie206
998686	A3UU690TWXCG1X	Movie206
998687	AI4J762YI6S06	Movie206

998688 rows × 2 columns

In [28]:

```
#print feature data with no null values
X_clean
```

Out[28]:

	user_id	variable
0	A3R5OBKS7OM2IR	Movie1
4848	A3R5OBKS7OM2IR	Movie2
9697	AH3QC2PC1VTGP	Movie3
14546	A3LKP6WPMP9UKX	Movie4
14547	AVIY68KEPQ5ZD	Movie4
...	...	...
998683	A1IMQ9WMFYKWH5	Movie206
998684	A1KLIKPUF5E88I	Movie206
998685	A5HG6WFZLO10D	Movie206
998686	A3UU690TWXCG1X	Movie206
998687	AI4J762YI6S06	Movie206

5000 rows × 2 columns

Create the target object

In [29]:

```
Y_target = df_final['value'] # with null replaced with zeros
Y_clean = df_clean['value']  # null values removed
```

In [30]:

```
#import KNN from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

In [31]:

```
#instantiate KNN estimator
knn = KNeighborsClassifier(n_neighbors=1)
```

In [32]:

```
#print the knn
print(knn)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [33]:

```
#Split the dataset to test and train the model with user id and movie name as feature
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X_feature,Y_target,random_state=1)
x_train
```

Out[33]:

	user_id	variable
336451	AMSAQB9DNQRUS	Movie70
501095	A1V4QPN5W2WS8T	Movie104
976524	A1UO5RU0DV7ZIG	Movie202
528927	A2UKWIDERJ8IHA	Movie110
468437	A2UL0A8GJB4318	Movie97
...	...	...
491263	A3IY2JLE4X829W	Movie102
791624	A1H143CZ4AM6VO	Movie164
470924	A35RXHCGGWEXQH	Movie98
491755	A1T8CV2VBAFXZO	Movie102
128037	A3AGZG0PDAGCK3	Movie27

749016 rows × 2 columns

In [34]:

```
#Split the dataset to test and train clean data no null values
from sklearn.model_selection import train_test_split
x_train_c,x_test_c,y_train_c,y_test_c = train_test_split(X_clean,Y_clean,random_state=1)
x_train_c
```

Out[34]:

	user_id	variable
613933	AF18LCM3NQQG6	Movie127
136375	A2JSR55IWC7HWO	Movie29
136320	AYCT65TQ1PLL4	Movie29
613153	A15G80TVI5M8IR	Movie127
780248	A2G5SNWDL61VJ7	Movie161
...	...	...
613642	A21A17T3TG912R	Movie127
613510	A3HVAHHHY9D7AC	Movie127
437207	A1E8OGV9SO0E0P	Movie91
672889	ANYDEWJQN2GBX	Movie139
72954	A2F3GFAA3XQDIB	Movie16

3750 rows × 2 columns

In [35]:

```
#import LabelEncoder from sklearn
from sklearn.preprocessing import LabelEncoder
#instantiate LabelEncoder
number = LabelEncoder()
#Transform non-numerical label to numerical label and create train , test data
```

```
x_train.user_id = number.fit_transform(x_train["user_id"].astype("str"))
x_test.user_id = number.fit_transform(x_test["user_id"].astype("str"))
x_train.variable = number.fit_transform(x_train["variable"].astype("str"))
x_test.variable = number.fit_transform(x_test["variable"].astype("str"))
y_train = number.fit_transform(y_train.astype("int"))
y_test = number.fit_transform(y_test.astype("int"))
```

```
C:\Users\suppy\anaconda3\lib\site-packages\pandas\core\generic.py:5303: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[name] = value
```

In [36]:

```
#Transform non-numerical label to numerical label and create train , test data for clean data with
no null values
x_train_c.user_id = number.fit_transform(x_train_c["user_id"].astype("str"))
x_test_c.user_id = number.fit_transform(x_test_c["user_id"].astype("str"))
x_train_c.variable = number.fit_transform(x_train_c["variable"].astype("str"))
x_test_c.variable = number.fit_transform(x_test_c["variable"].astype("str"))
y_train_c = number.fit_transform(y_train_c.astype("int"))
y_test_c = number.fit_transform(y_test_c.astype("int"))
```

In [37]:

```
#print shape of training data
x_train.shape
x_train
```

Out[37]:

	user_id	variable
336451	4390	174
501095	1095	6
976524	1078	115
528927	2409	13
468437	2410	203
...	...	...
491263	3249	4
791624	607	72
470924	2788	204
491755	1026	4
128037	2963	126

749016 rows × 2 columns

In [38]:

```
#print shape of training data
x_train_c.shape
```

Out[38]:

```
(3750, 2)
```

In [39]:

```
#print first five records of training data
x_train.head(5)
```

Out[39]:

	user_id	variable
336451	4390	174
501095	1095	6
976524	1078	115
528927	2409	13
468437	2410	203

In [40]:

```
#fit data in KNN estimator with movie and userid as feature
knn.fit(x_train,y_train)
```

Out[40]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [41]:

```
#fit data in KNN estimator with clean data no null values
knn.fit(x_train_c,y_train_c)
```

Out[41]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

Testing

In [42]:

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

Out[42]:

```
array([4, 4, 4, ..., 1, 1, 1], dtype=int64)
```

In [43]:

```
#testing with  clean data with no null values
predicted_clean = knn.predict(x_test_c)
predicted_clean
```

Out[43]:

```
array([1, 4, 4, ..., 0, 4, 2], dtype=int64)
```

In [44]:

```
#Evaluate the accuracy of model with null replaced with zero
from sklearn import metrics
x= metrics.accuracy_score(y_test,predicted)
print(x)
```

```
0.04123009388317472
```

In [45]:

```
#Evaluate the accuracy of model with null removed
from sklearn import metrics
```

```
x_c= metrics.accuracy_score(y_test_c,predicted_clean)
print(x_c)
```

```
0.5528
```

In [46]:

```
#Confusion Matrix with null replaced with zero
from sklearn import metrics
print(metrics.confusion_matrix(y_test, predicted))
```

```
[[ 10192  38511  28578  19019 152110      0]
 [     7     28      4      6     35      0]
 [     4     11      2      8     31      0]
 [     5     13      4      5     33      0]
 [    14     23      7      5     67      0]
 [    50    218     59     78    545      0]]
```

In [47]:

```
#Confusion Matrix with clean data no null values
from sklearn import metrics
print(metrics.confusion_matrix(y_test_c, predicted_clean))
```

```
[[  7   7   5   8  58]
 [  4   2   3   1  35]
 [  7   1   6   2  56]
 [  8   4   7  18 108]
 [ 62  29  50 104 658]]
```

In [48]:

```
#Classification Report with null replaced with zero
print(metrics.classification_report(y_test, predicted))
```

```
C:\Users\suppy\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1272:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
              precision    recall  f1-score   support

           0       0.99      0.04      0.08    248410
           1       0.00      0.35      0.00        80
           2       0.00      0.04      0.00        56
           3       0.00      0.08      0.00        60
           4       0.00      0.58      0.00       116
           5       0.00      0.00      0.00       950

    accuracy                           0.04    249672
   macro avg       0.17      0.18      0.01    249672
weighted avg       0.99      0.04      0.08    249672
```

In [49]:

```
#Classification Report with clean data
print(metrics.classification_report(y_test_c, predicted_clean))
```

```
              precision    recall  f1-score   support

           0       0.08      0.08      0.08        85
           1       0.05      0.04      0.05        45
           2       0.08      0.08      0.08        72
           3       0.14      0.12      0.13       145
           4       0.72      0.73      0.72       903

    accuracy                           0.55      1250
   macro avg       0.21      0.21      0.21      1250
```

```
weighted avg       0.55      0.55      0.55      1250
```

Create a logistic regression model using the training set

In [50]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [51]:

```
#Training with null replaced with zero
logreg.fit(x_train,y_train)
```

```
C:\Users\suppy\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[51]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [52]:

```
#Training with clean data no null values
logreg.fit(x_train_c,y_train_c)
```

```
C:\Users\suppy\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[52]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [53]:

```
#Make predictions using the testing set with null replaced with zero
y_pred = logreg.predict(x_test)
```

In [54]:

```
#Make predictions using the testing set clean data no null values
y_pred_c = logreg.predict(x_test_c)
```

In [55]:

```
#Evaluate the accuracy of model with null replaced with zero
from sklearn import metrics
x= metrics.accuracy_score(y_test,y_pred)
print(x)
```

```
0.0004646095677528918
```

Evaluate the accuracy of model of clean data with no null values

In [56]:

```
from sklearn import metrics
x_c= metrics.accuracy_score(y_test_c,y_pred_c)
print(x_c)
```

```
0.7224
```

Confusion Matrix with null replaced with zero

In [57]:

```
from sklearn import metrics
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[     0      0      0      0 248410      0]
 [     0      0      0      0     80      0]
 [     0      0      0      0     56      0]
 [     0      0      0      0     60      0]
 [     0      0      0      0    116      0]
 [     0      0      0      0    950      0]]
```

Confusion Matrix of clean data with no null values

In [58]:

```
from sklearn import metrics
print(metrics.confusion_matrix(y_test_c, y_pred_c))
```

```
[[  0   0   0   0  85]
 [  0   0   0   0  45]
 [  0   0   0   0  72]
 [  0   0   0   0 145]
 [  0   0   0   0 903]]
```

Classification Report with null replaced with zero

In [59]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
C:\Users\suppy\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1272:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00    248410
           1       0.00      0.00      0.00        80
           2       0.00      0.00      0.00        56
```

```
           3       0.00      0.00      0.00        60
           4       0.00      1.00      0.00       116
           5       0.00      0.00      0.00       950

    accuracy                           0.00    249672
   macro avg       0.00      0.17      0.00    249672
weighted avg       0.00      0.00      0.00    249672
```

Classification Report clean data with no null values

In [60]:

```
print(metrics.classification_report(y_test_c, y_pred_c))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        85
           1       0.00      0.00      0.00        45
           2       0.00      0.00      0.00        72
           3       0.00      0.00      0.00       145
           4       0.72      1.00      0.84       903

    accuracy                           0.72      1250
   macro avg       0.14      0.20      0.17      1250
weighted avg       0.52      0.72      0.61      1250
```

In [ ]:

```

```