In [1]:
```
import pandas as pd
from sklearn import preprocessing
from IPython.display import display, HTML
```

In [2]:
```
# getting the dataset
df = pd.read_csv('C:/Users/nayak/Downloads/malware_MultiClass.csv')
```

In [3]:
```
# selecting columns specified in readme file
df = df.loc[:, ['classification', 'os', 'usage_counter', 'prio', 'static_prio', 'normal_prio', 'vm_pgoff', 'vm_
df.head()
```

Out[3]:

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater
0	malware	CentOS	0	3069378560	14274	0	0	13173	0	6850	
1	malware	Windows	0	3069378560	14274	0	0	13173	0	6850	
2	malware	Mac	0	3069378560	14274	0	0	13173	0	6850	
3	malware	Ubuntu	0	3069378560	14274	0	0	13173	0	6850	
4	malware	Mac	0	3069378560	14274	0	0	13173	0	6850	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   classification     100000 non-null  object
 1   os                 100000 non-null  object
 2   usage_counter      100000 non-null  int64
 3   prio               100000 non-null  int64
 4   static_prio        100000 non-null  int64
 5   normal_prio        100000 non-null  int64
 6   vm_pgoff           100000 non-null  int64
 7   vm_truncate_count  100000 non-null  int64
 8   task_size          100000 non-null  int64
 9   map_count          100000 non-null  int64
 10  hiwater_rss        100000 non-null  int64
 11  total_vm           100000 non-null  int64
 12  shared_vm          100000 non-null  int64
 13  exec_vm            100000 non-null  int64
 14  reserved_vm        100000 non-null  int64
 15  nr_ptes            100000 non-null  int64
 16  nvcsw              100000 non-null  int64
 17  nivcsw             100000 non-null  int64
 18  signal_nvcsw       100000 non-null  int64
dtypes: int64(17), object(2)
memory usage: 14.5+ MB
```

In [5]:
```
# checking min and max values
df.describe()
```

Out[5]:

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	t
count	100000.0	1.000000e+05	100000.000000	100000.0	100000.0	100000.000000	100000.0	100000.00000	100000.0	100000
mean	0.0	3.069706e+09	18183.900070	0.0	0.0	15312.739510	0.0	8771.13948	0.0	266
std	0.0	2.963061e+05	4609.792765	0.0	0.0	3256.475008	0.0	3785.30516	0.0	311
min	0.0	3.069190e+09	13988.000000	0.0	0.0	9695.000000	0.0	2588.00000	0.0	4
25%	0.0	3.069446e+09	14352.000000	0.0	0.0	12648.000000	0.0	6428.00000	0.0	99
50%	0.0	3.069698e+09	16159.000000	0.0	0.0	15245.000000	0.0	7865.00000	0.0	177
75%	0.0	3.069957e+09	22182.000000	0.0	0.0	17663.000000	0.0	10684.00000	0.0	327
max	0.0	3.070222e+09	31855.000000	0.0	0.0	27157.000000	0.0	28184.00000	0.0	2810

In [6]:
```
df.isna().sum()
```

Out[6]:
```
classification       0
os                   0
usage_counter        0
prio                 0
static_prio          0
normal_prio          0
vm_pgoff             0
vm_truncate_count    0
task_size            0
map_count            0
hiwater_rss          0
total_vm             0
shared_vm            0
exec_vm              0
reserved_vm          0
nr_ptes              0
nvcsw                0
nivcsw               0
signal_nvcsw         0
dtype: int64
```

In [7]:
```
y = df['classification'] # define label as nominal values
le = preprocessing.LabelEncoder()
le.fit(y)
y_encoded = le.transform(y) # encode nominal labels to integers
df['classification'] = y_encoded
```

In [8]:
```
# displaying after encoding
df.sample(10)
```

Out[8]:

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiv
27912	0	Windows	0	3069227008	24345	0	0	21891	0	14179	
50239	0	Ubuntu	0	3069390848	17048	0	0	15627	0	7321	
78082	1	Windows	0	3069419520	14263	0	0	12816	0	6503	
52892	0	CentOS	0	3069820928	25900	0	0	19291	0	26822	
82289	1	CentOS	0	3069202432	13989	0	0	10422	0	3720	
11562	0	CentOS	0	3069689856	30622	0	0	18219	0	13797	
82142	1	CentOS	0	3069202432	13989	0	0	10422	0	3720	
98546	1	Mac	0	3069968384	14273	0	0	11199	0	4247	
90615	1	CentOS	0	3069812736	14020	0	0	9743	0	2664	
39298	0	Debian	0	3069460480	14118	0	0	15023	0	9453	

In [9]:
```
print('Column Datatypes:\n',df.dtypes)
```

```
Column Datatypes:
 classification        int32
os                    object
usage_counter          int64
prio                   int64
static_prio            int64
normal_prio            int64
vm_pgoff               int64
vm_truncate_count      int64
task_size              int64
map_count              int64
hiwater_rss            int64
total_vm               int64
shared_vm              int64
exec_vm                int64
reserved_vm            int64
nr_ptes                int64
nvcsw                  int64
nivcsw                 int64
signal_nvcsw           int64
dtype: object
```

In [10]:
```
df_num=df.copy(deep=True)
# create new binary columns
df_dummies=pd.get_dummies(df_num['os'], drop_first=True)
df_dummies = df_dummies.astype(int)
# add them to dataframe
df_num=df_num.join(df_dummies)
# drop original columns
df_num.drop('os', axis=1, inplace=True)
```

In [11]:
```
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 22 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   classification     100000 non-null  int32
 1   usage_counter      100000 non-null  int64
 2   prio               100000 non-null  int64
 3   static_prio        100000 non-null  int64
 4   normal_prio        100000 non-null  int64
 5   vm_pgoff           100000 non-null  int64
 6   vm_truncate_count  100000 non-null  int64
 7   task_size          100000 non-null  int64
 8   map_count          100000 non-null  int64
 9   hiwater_rss        100000 non-null  int64
 10  total_vm           100000 non-null  int64
 11  shared_vm          100000 non-null  int64
 12  exec_vm            100000 non-null  int64
 13  reserved_vm        100000 non-null  int64
 14  nr_ptes            100000 non-null  int64
 15  nvcsw              100000 non-null  int64
 16  nivcsw             100000 non-null  int64
 17  signal_nvcsw       100000 non-null  int64
 18  Debian             100000 non-null  int32
 19  Mac                100000 non-null  int32
 20  Ubuntu             100000 non-null  int32
 21  Windows            100000 non-null  int32
dtypes: int32(5), int64(17)
memory usage: 14.9 MB
```

In [12]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_num_std = df_num.copy(deep=True)
x_features = df_num_std.loc[:, df_num_std.columns != 'classification']
cols = x_features.columns
df_num_std = pd.DataFrame(scaler.fit_transform(x_features),
                          columns = cols)
df_num_std['classification'] = y_encoded
display('df_num_std:',HTML(df_num_std.head(10).to_html()))
```

'df_num_std:'

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	total_vm	shared_vm
0	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
1	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
2	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
3	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
4	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
5	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
6	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
7	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
8	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507528	0.0	-0.373375	0.66725
9	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	0.0	-0.507000	0.0	-0.373375	0.66725

In [13]:
```
df_num_std.describe()
```

Out[13]:

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task_size	map_count	hiwater_rss	to
count	100000.0	1.000000e+05	1.000000e+05	100000.0	100000.0	1.000000e+05	100000.0	1.000000e+05	100000.0	1.00000
mean	0.0	7.724202e-13	8.185452e-17	0.0	0.0	1.909939e-16	0.0	4.547474e-17	0.0	-8.86
std	0.0	1.000005e+00	1.000005e+00	0.0	0.0	1.000005e+00	0.0	1.000005e+00	0.0	1.00000
min	0.0	-1.740945e+00	-9.102190e-01	0.0	0.0	-1.725107e+00	0.0	-1.633467e+00	0.0	-8.41
25%	0.0	-8.769698e-01	-8.312563e-01	0.0	0.0	-8.182937e-01	0.0	-6.190125e-01	0.0	-5.36
50%	0.0	-2.681780e-02	-4.392627e-01	0.0	0.0	-2.080159e-02	0.0	-2.393847e-01	0.0	-2.86
75%	0.0	8.475255e-01	8.673101e-01	0.0	0.0	7.217228e-01	0.0	5.053411e-01	0.0	1.9394
max	0.0	1.742604e+00	2.965680e+00	0.0	0.0	3.637160e+00	0.0	5.128505e+00	0.0	8.1523

8 rows × 22 columns

In [14]:
```
# removed as they are zero
df_num_std.drop(['usage_counter', 'normal_prio', 'vm_pgoff', 'task_size', 'hiwater_rss', 'nr_ptes', 'signal_nvc
```

In [15]:
```
df_num.drop(['usage_counter', 'normal_prio', 'vm_pgoff', 'task_size', 'hiwater_rss', 'nr_ptes', 'signal_nvcsw']
```

In [16]:
```
df_num_std.head()
```

Out[16]:

	prio	static_prio	vm_truncate_count	map_count	total_vm	shared_vm	exec_vm	reserved_vm	nvcsw	nivcsw	Debian	
0	-1.105059	-0.848177	-0.657076	-0.507528	-0.373375	0.667258	-0.165103	0.041477	-0.695251	-0.625663	-0.498656	-0.4
1	-1.105059	-0.848177	-0.657076	-0.507528	-0.373375	0.667258	-0.165103	0.041477	-0.695251	-0.625663	-0.498656	-0.4
2	-1.105059	-0.848177	-0.657076	-0.507528	-0.373375	0.667258	-0.165103	0.041477	-0.695251	-0.625663	-0.498656	2.0
3	-1.105059	-0.848177	-0.657076	-0.507528	-0.373375	0.667258	-0.165103	0.041477	-0.695251	-0.625663	-0.498656	-0.4
4	-1.105059	-0.848177	-0.657076	-0.507528	-0.373375	0.667258	-0.165103	0.041477	-0.695251	-0.625663	-0.498656	2.0

In [17]:
```
df_binary = df_num.copy(deep=True)
numCols = [1,2,3,4,5,6,7,8,9,10]
df_numerical = df_binary.iloc[:,numCols]
df_dummy = df_binary.drop(df_binary.columns[numCols], axis=1)
display('df_numerical:',HTML(df_numerical.head(10).to_html()))
display('df_dummy:',HTML(df_dummy.head(10).to_html()))
```

'df_numerical:'

	prio	static_prio	vm_truncate_count	map_count	total_vm	shared_vm	exec_vm	reserved_vm	nvcsw	nivcsw
0	3069378560	14274	13173	6850	150	120	124	210	341974	0
1	3069378560	14274	13173	6850	150	120	124	210	341974	0
2	3069378560	14274	13173	6850	150	120	124	210	341974	0
3	3069378560	14274	13173	6850	150	120	124	210	341974	0
4	3069378560	14274	13173	6850	150	120	124	210	341974	0
5	3069378560	14274	13173	6850	150	120	124	210	341974	0
6	3069378560	14274	13173	6850	150	120	124	210	341974	0
7	3069378560	14274	13173	6850	150	120	124	210	341974	0
8	3069378560	14274	13173	6850	150	120	124	210	341974	0
9	3069378560	14274	13173	6852	150	120	124	211	341974	0

'df_dummy:'

	classification	Debian	Mac	Ubuntu	Windows
0	1	0	0	0	0
1	1	0	0	0	1
2	1	0	1	0	0
3	1	0	0	1	0
4	1	0	1	0	0
5	1	0	0	0	1
6	1	0	0	1	0
7	1	0	1	0	0
8	1	0	0	0	0
9	1	0	1	0	0

In [18]:
```
group_names = ['L','M','H']
for col in df_numerical.columns:
    df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
display('df_numerical:',HTML(df_numerical.head(10).to_html()))
```

```
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
C:\Users\nayak\AppData\Local\Temp\ipykernel_23300\1765263982.py:3: 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#r
eturning-a-view-versus-a-copy
  df_numerical[col] = pd.cut(df_numerical[col], 3, labels=group_names)
'df_numerical:'
```

	prio	static_prio	vm_truncate_count	map_count	total_vm	shared_vm	exec_vm	reserved_vm	nvcsw	nivcsw
0	L	L	L	L	L	H	L	L	L	L
1	L	L	L	L	L	H	L	L	L	L
2	L	L	L	L	L	H	L	L	L	L
3	L	L	L	L	L	H	L	L	L	L
4	L	L	L	L	L	H	L	L	L	L
5	L	L	L	L	L	H	L	L	L	L
6	L	L	L	L	L	H	L	L	L	L
7	L	L	L	L	L	H	L	L	L	L
8	L	L	L	L	L	H	L	L	L	L
9	L	L	L	L	L	H	L	L	L	L

In [19]:
```
display('df_numerical:',HTML(df_numerical.sample(10).to_html()))
```

'df_numerical:'

	prio	static_prio	vm_truncate_count	map_count	total_vm	shared_vm	exec_vm	reserved_vm	nvcsw	nivcsw
66802	L	L	L	L	L	H	L	L	L	L
59577	H	L	L	L	L	H	L	L	L	L
6106	L	L	L	L	L	H	L	L	L	L
30717	M	H	M	M	L	L	H	L	H	L
24819	L	L	M	L	L	H	M	M	L	M
81478	M	L	M	L	L	L	M	M	L	L
68396	H	L	L	L	L	H	L	L	L	L
87253	H	M	L	L	L	L	M	L	L	L
24854	L	L	M	L	L	H	M	M	L	M
30742	M	H	M	M	L	L	H	L	H	L

In [20]:
```
df_dummies=pd.get_dummies(df_numerical).astype(int)
display('df_dummies:',HTML(df_dummies.head(10).to_html()))
```

'df_dummies:'

	prio_L	prio_M	prio_H	static_prio_L	static_prio_M	static_prio_H	vm_truncate_count_L	vm_truncate_count_M	vm_truncate_count_H	map
0	1	0	0	1	0	0	1	0	0	
1	1	0	0	1	0	0	1	0	0	
2	1	0	0	1	0	0	1	0	0	
3	1	0	0	1	0	0	1	0	0	
4	1	0	0	1	0	0	1	0	0	
5	1	0	0	1	0	0	1	0	0	
6	1	0	0	1	0	0	1	0	0	
7	1	0	0	1	0	0	1	0	0	
8	1	0	0	1	0	0	1	0	0	
9	1	0	0	1	0	0	1	0	0	

In [21]:
```
display('df_dummies:',HTML(df_dummies.sample(10).to_html()))
```

'df_dummies:'

	prio_L	prio_M	prio_H	static_prio_L	static_prio_M	static_prio_H	vm_truncate_count_L	vm_truncate_count_M	vm_truncate_count_H
45478	1	0	0	1	0	0	1	0	0
53360	0	1	0	1	0	0	1	0	0
93309	0	0	1	1	0	0	1	0	0
57040	0	1	0	1	0	0	1	0	0
7281	0	1	0	1	0	0	1	0	0
61802	0	0	1	1	0	0	1	0	0
92932	0	0	1	1	0	0	1	0	0
25985	0	1	0	0	0	1	0	1	0
41597	1	0	0	0	1	0	0	1	0
38655	1	0	0	0	0	1	0	0	1

In [22]:
```
cols_removed = ['prio_L', 'static_prio_L', 'vm_truncate_count_L', 'map_count_L', 'total_vm_L', 'shared_vm_L'\
```

```
                , 'exec_vm_L', 'reserved_vm_L', 'nvcsw_L', 'nivcsw_L']
df_dummies = df_dummies.drop(cols_removed, axis=1)
```

In [23]:
```
df_binary = pd.concat([df_dummies, df_dummy], axis=1)
```

In [24]:
```
display('df_binary:',HTML(df_binary.sample(10).to_html()))
```

'df_binary:'

	prio_M	prio_H	static_prio_M	static_prio_H	vm_truncate_count_M	vm_truncate_count_H	map_count_M	map_count_H	total_vm_M	t
47985	1	0	0	0	0	1	0	0	0	
36736	1	0	0	0	0	0	0	0	0	
15423	0	1	1	0	1	0	0	0	0	
22815	0	1	0	0	1	0	0	0	0	
71703	0	1	0	0	0	0	0	0	0	
47648	1	0	0	0	1	0	0	0	0	
17152	1	0	0	0	0	0	0	0	0	
19328	1	0	0	0	0	0	0	0	0	
25359	1	0	0	0	1	0	0	0	0	
22902	0	1	0	0	1	0	0	0	0	

In [25]:
```
from sklearn.naive_bayes import CategoricalNB

display('df_binary:',HTML(df_binary.head(10).to_html()))
```

'df_binary:'

	prio_M	prio_H	static_prio_M	static_prio_H	vm_truncate_count_M	vm_truncate_count_H	map_count_M	map_count_H	total_vm_M	total_
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	

In [33]:
```
# splitting the data into train and test
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score, f1_score, roc_auc_score

x_features = df_binary.drop('classification', axis=1)
y_encoded = df_binary['classification']

x_train, x_test, y_train, y_test = train_test_split(x_features, y_encoded,
                                                    test_size=0.2)
```

In [34]:
```
# 1. Bernoulli Naive Bayes using Hold Out Evaluation 80% Training Data.

from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix

clf = BernoulliNB(alpha=1)

# training the classifier
clf.fit(x_train, y_train)

# predict the labels for x_test
y_pred = clf.predict(x_test)
# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
micro_precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
micro_recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
micro_f1 = f1_score(y_test, y_pred, average='micro')


# Print the evaluation results
```

```
print('Results of Bernoulli Naive Bayes using df_binary.\n')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",micro_precision, \
    "\nMicro Recall = ",micro_recall, "\nMicro F1 = ", micro_f1)
```

```
Results of Bernoulli Naive Bayes using df_binary.

Hold-out Evaluation: Accuracy = 0.75245
Micro Precision = 0.5152312661838794
Micro Recall =  0.5020511374980915
Micro F1 =  0.4988803368922123
```

```
f:\Test_Environment\venv\Lib\site-packages\sklearn\metrics\_classification.py:1497: UndefinedMetricWarning: Pre
cision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [31]:
```
# 2. Gaussian Naive Bayes using Hold Out Evaluation 80% Training Data.
from sklearn.naive_bayes import GaussianNB

x_features_gnb = df_num.drop('classification', axis=1)
y_encoded_gnb = df_num['classification']

x_train, x_test, y_train, y_test = train_test_split(x_features_gnb, y_encoded_gnb,
                                                    test_size=0.2)
clf = GaussianNB()
clf.fit(x_train, y_train)
# Predict the labels
y_pred = clf.predict(x_test)
# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
micro_precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
micro_recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
micro_f1 = f1_score(y_test, y_pred, average='micro')


# Print the evaluation results
print('Results of Gaussian Naive Bayes using df_num.\n')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",micro_precision, \
    "\nMicro Recall = ",micro_recall, "\nMicro F1 = ", micro_f1)
#-----------------------------------------------------------------------------------------------
# changing the dataset to df_num_std
x_features_gnb = df_num_std.drop('classification', axis=1)
y_encoded_gnb = df_num_std['classification']

x_train, x_test, y_train, y_test = train_test_split(x_features_gnb, y_encoded_gnb,
                                                    test_size=0.2)

clf1 = GaussianNB()
clf1.fit(x_train, y_train)
# Predict the labels
y_pred = clf1.predict(x_test)
# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
micro_precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
micro_recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
micro_f1 = f1_score(y_test, y_pred, average='micro')


# Print the evaluation results
print('\n\n')
print('Results of Gaussian Naive Bayes using df_num_std.\n')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",micro_precision, \
    "\nMicro Recall = ",micro_recall, "\nMicro F1 = ", micro_f1)
```

```
Results of Gaussian Naive Bayes using df_num.

Hold-out Evaluation: Accuracy = 0.62035
Micro Precision = 0.41496285020557977
Micro Recall =  0.4139662069923887
Micro F1 =  0.41229899112558305



Results of Gaussian Naive Bayes using df_num_std.

Hold-out Evaluation: Accuracy = 0.4965
Micro Precision = 0.48699805879014524
Micro Recall =  0.5007695544857476
Micro F1 =  0.48800514738413575
```

```
f:\Test_Environment\venv\Lib\site-packages\sklearn\metrics\_classification.py:1497: UndefinedMetricWarning: Pre
cision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [30]:
```
# 3. Multinomial Naive Bayes using Hold Out Evaluation 80% Training Data.
from sklearn.naive_bayes import MultinomialNB

x_features_mnb = df_num.drop('classification', axis=1)
y_encoded_mnb = df_num['classification']
x_train, x_test, y_train, y_test = train_test_split(x_features_mnb, y_encoded_mnb,
                                                    test_size=0.2)
clf = MultinomialNB()
clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)
# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
f1 = f1_score(y_test, y_pred, average='micro')


# Printing the results
print('Results of Multinomial Naive Bayes using df_num.\n')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",precision, \
    "\nMicro Recall = ",recall, "\nMicro F1 = ", f1)
```

```
Results of Multinomial Naive Bayes using df_num.

Hold-out Evaluation: Accuracy = 0.4276
Micro Precision = 0.40461753405242207
Micro Recall =  0.4331720639831598
Micro F1 =  0.33394587046104807
```

In [35]:
```
# 4. Complement Naive Bayes with Hold Out Evaluation.(80% Training Data)
from sklearn.naive_bayes import ComplementNB

x_features_mnb = df_num.drop('classification', axis=1)
y_encoded_mnb = df_num['classification']
x_train, x_test, y_train, y_test = train_test_split(x_features_mnb, y_encoded_mnb,
                                                    test_size=0.2)
clf = ComplementNB()
clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)

# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
f1 = f1_score(y_test, y_pred, average='micro')

# Printing the results
print('Results of Complement Naive Bayes using df_num.')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",precision, \
    "\nMicro Recall = ",recall, "\nMicro F1 = ", f1)
```

```
Results of Complement Naive Bayes using df_num.
Hold-out Evaluation: Accuracy = 0.6269
Micro Precision = 0.4218787728786581
Micro Recall =  0.41950221121791903
Micro F1 =  0.4162742928577072
f:\Test_Environment\venv\Lib\site-packages\sklearn\metrics\_classification.py:1497: UndefinedMetricWarning: Precision is 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, f"{metric.capitalize()} is", len(result))
```

In [37]:
```
# 5. Categorical Naive Bayes with Hold Out Evaluation - 80% training data
from sklearn.model_selection import train_test_split

x_features = df_binary.drop('classification', axis=1)
y_encoded = df_binary['classification']

x_train, x_test, y_train, y_test = train_test_split(x_features, y_encoded,
                                                    test_size=0.2)
clf = CategoricalNB(alpha=1)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
# Calculate accuracy
accuracy = clf.score(x_test, y_test)
# Calculate micro-precision
precision = precision_score(y_test, y_pred, average='micro')
# Calculate micro-recall
recall = recall_score(y_test, y_pred, average='micro')
# Calculate micro-F1
f1 = f1_score(y_test, y_pred, average='micro')
```

```
# Printing the results
print('Results of Categorical Naive Bayes using df_binary.')
print("Hold-out Evaluation: Accuracy =", accuracy, "\nMicro Precision =",precision, \
    "\nMicro Recall = ",recall, "\nMicro F1 = ", f1)
```

```
Results of Categorical Naive Bayes using df_binary.
Hold-out Evaluation: Accuracy = 0.7537
Micro Precision = 0.5167249811875679
Micro Recall =  0.5050058382591812
Micro F1 =  0.500570411976774
```

```
f:\Test_Environment\venv\Lib\site-packages\sklearn\metrics\_classification.py:1497: UndefinedMetricWarning: Pre
cision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [ ]: