Multinomial Classification (normal or DOS or PROBE or R2L or U2R)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
pip install -U ydata-profiling
```

```
Requirement already satisfied: ydata-profiling in /usr/local/lib/python3.10/dist-packages (4.12.0)
Requirement already satisfied: scipy<1.14,>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.13.1)
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.2.2)
Requirement already satisfied: matplotlib<3.10,>=3.5 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.8.0)
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.10.3)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (6.0.2)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.1.4)
Requirement already satisfied: visions<0.7.7,>=0.7.5 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]<0.7.7
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.26.4)
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.12.4)
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.32.3)
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.66.6)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.13.2)
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.12)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.14.4)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.4.1)
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.3.1)
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.9.4)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.8.1)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.60.0)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (1.8.0
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (11.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profil
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profiling
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profi
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profi
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profil
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-profil
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-pr
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydata-p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profilin
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling
Requirement already satisfied: pydantic-core==2.27.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profil
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profil
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profil
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions<0.7.7,>=0.7.5->visions[type_in
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packages (from visions<0.7.7,>=0.7.5->visions[type_in
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib<3.10,>=3.5
```

```
import itertools
import seaborn as sns
import ydata_profiling # Change pandas_profiling to ydata_profiling
import statsmodels.formula.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from patsy import dmatrices
```

```
from sklearn import datasets
from sklearn.feature_selection import RFE
import sklearn.metrics as metrics
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_classif
```

```
train=pd.read_csv('Train.txt',sep=',')
test=pd.read_csv('Test.txt',sep=',')
```

```
train.head()
```

	0	tcp	ftp_data	SF	491	0.1	0.2	0.3	0.4	0.5	...	0.17	0.03	0.17.1	0.00.6	0.00.7	0.00.8	0.05	0.00.9	normal	2
0	0	udp	other	SF	146.0	0.0	0.0	0.0	0.0	0.0	...	0.00	0.60	0.88	0.00	0.00	0.00	0.0	0.00	normal	15
1	0	tcp	private	S0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.10	0.05	0.00	0.00	1.00	1.00	0.0	0.00	neptune	19
2	0	tcp	http	SF	232.0	8153.0	0.0	0.0	0.0	0.0	...	1.00	0.00	0.03	0.04	0.03	0.01	0.0	0.01	normal	21
3	0	tcp	http	SF	199.0	420.0	0.0	0.0	0.0	0.0	...	1.00	0.00	0.00	0.00	0.00	0.00	0.0	0.00	normal	21
4	0	tcp	private	REJ	0.0	0.0	0.0	0.0	0.0	0.0	...	0.07	0.07	0.00	0.00	0.00	0.00	1.0	1.00	neptune	21

5 rows × 43 columns

```
columns=["duration","protocol_type","service","flag","src_bytes","dst_bytes","land","wrong_fragment","urgent","hot",
        "num_failed_logins","logged_in","num_compromised","root_shell","su_attempted","num_root","num_file_creations",
        "num_shells","num_access_files","num_outbound_cmds","is_host_login","is_guest_login","count","srv_count","serror_rate",
        "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate","diff_srv_rate","srv_diff_host_rate",
        "dst_host_count","dst_host_srv_count","dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
        "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate",
        "dst_host_srv_rerror_rate","attack","last_flag"]
```

```
len(columns)
```

```
43
```

```
train.columns=columns
test.columns=columns
```

```
train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_
0	0	udp	other	SF	146.0	0.0	0.0	0.0	0.0	0.0	...	0.00	
1	0	tcp	private	S0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.10	
2	0	tcp	http	SF	232.0	8153.0	0.0	0.0	0.0	0.0	...	1.00	
3	0	tcp	http	SF	199.0	420.0	0.0	0.0	0.0	0.0	...	1.00	
4	0	tcp	private	REJ	0.0	0.0	0.0	0.0	0.0	0.0	...	0.07	

5 rows × 43 columns

```
test.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst
0	0	tcp	private	REJ	0	0	0	0	0	0	...	0.00	
1	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	0.61	
2	0	icmp	eco_i	SF	20	0	0	0	0	0	...	1.00	
3	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	0.31	
4	0	tcp	http	SF	267	14515	0	0	0	0	...	1.00	

5 rows × 43 columns

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13822 entries, 0 to 13821
Data columns (total 43 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     13822 non-null  int64
 1   protocol_type                13822 non-null  object
 2   service                      13821 non-null  object
 3   flag                         13821 non-null  object
 4   src_bytes                    13821 non-null  float64
 5   dst_bytes                    13821 non-null  float64
 6   land                         13821 non-null  float64
 7   wrong_fragment               13821 non-null  float64
 8   urgent                       13821 non-null  float64
 9   hot                          13821 non-null  float64
 10  num_failed_logins            13821 non-null  float64
 11  logged_in                    13821 non-null  float64
 12  num_compromised              13821 non-null  float64
```

```
 13  root_shell                  13821 non-null  float64
 14  su_attempted                13821 non-null  float64
 15  num_root                    13821 non-null  float64
 16  num_file_creations          13821 non-null  float64
 17  num_shells                  13821 non-null  float64
 18  num_access_files            13821 non-null  float64
 19  num_outbound_cmds           13821 non-null  float64
 20  is_host_login               13821 non-null  float64
 21  is_guest_login              13821 non-null  float64
 22  count                       13821 non-null  float64
 23  srv_count                   13821 non-null  float64
 24  serror_rate                 13821 non-null  float64
 25  srv_serror_rate             13821 non-null  float64
 26  rerror_rate                 13821 non-null  float64
 27  srv_rerror_rate             13821 non-null  float64
 28  same_srv_rate               13821 non-null  float64
 29  diff_srv_rate               13821 non-null  float64
 30  srv_diff_host_rate          13821 non-null  float64
 31  dst_host_count              13821 non-null  float64
 32  dst_host_srv_count          13821 non-null  float64
 33  dst_host_same_srv_rate      13821 non-null  float64
 34  dst_host_diff_srv_rate      13821 non-null  float64
 35  dst_host_same_src_port_rate 13821 non-null  float64
 36  dst_host_srv_diff_host_rate 13821 non-null  float64
 37  dst_host_serror_rate        13821 non-null  float64
 38  dst_host_srv_serror_rate    13821 non-null  float64
 39  dst_host_rerror_rate        13821 non-null  float64
 40  dst_host_srv_rerror_rate    13821 non-null  float64
 41  attack                      13821 non-null  object
 42  last_flag                   13821 non-null  float64
dtypes: float64(38), int64(1), object(4)
memory usage: 4.5+ MB
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13734 entries, 0 to 13733
Data columns (total 43 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                    13734 non-null  int64
 1   protocol_type               13734 non-null  object
 2   service                     13734 non-null  object
 3   flag                        13734 non-null  object
 4   src_bytes                   13734 non-null  int64
 5   dst_bytes                   13734 non-null  int64
 6   land                        13734 non-null  int64
 7   wrong_fragment              13734 non-null  int64
 8   urgent                      13734 non-null  int64
 9   hot                         13734 non-null  int64
 10  num_failed_logins           13734 non-null  int64
 11  logged_in                   13734 non-null  int64
 12  num_compromised             13734 non-null  int64
 13  root_shell                  13734 non-null  int64
 14  su_attempted                13734 non-null  int64
 15  num_root                    13734 non-null  int64
 16  num_file_creations          13734 non-null  int64
 17  num_shells                  13734 non-null  int64
 18  num_access_files            13734 non-null  int64
 19  num_outbound_cmds           13734 non-null  int64
 20  is_host_login               13734 non-null  int64
 21  is_guest_login              13734 non-null  int64
 22  count                       13734 non-null  int64
 23  srv_count                   13734 non-null  int64
 24  serror_rate                 13734 non-null  float64
 25  srv_serror_rate             13734 non-null  float64
 26  rerror_rate                 13734 non-null  float64
 27  srv_rerror_rate             13734 non-null  float64
 28  same_srv_rate               13734 non-null  float64
 29  diff_srv_rate               13734 non-null  float64
 30  srv_diff_host_rate          13734 non-null  float64
 31  dst_host_count              13733 non-null  float64
 32  dst_host_srv_count          13733 non-null  float64
 33  dst_host_same_srv_rate      13733 non-null  float64
 34  dst_host_diff_srv_rate      13733 non-null  float64
 35  dst_host_same_src_port_rate 13733 non-null  float64
 36  dst_host_srv_diff_host_rate 13733 non-null  float64
 37  dst_host_serror_rate        13733 non-null  float64
 38  dst_host_srv_serror_rate    13733 non-null  float64
 39  dst_host_rerror_rate        13733 non-null  float64
 40  dst_host_srv_rerror_rate    13733 non-null  float64
 41  attack                      13733 non-null  object
 42  last_flag                   13733 non-null  float64
dtypes: float64(18), int64(21), object(4)
memory usage: 4.5+ MB
```

```
train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
duration	13822.0	307.416799	2.715400e+03	0.0	0.00	0.00	0.00	42260.0
src_bytes	13821.0	37061.403155	3.251583e+06	0.0	0.00	44.00	280.00	381709090.0
dst_bytes	13821.0	3552.783445	9.149915e+04	0.0	0.00	0.00	575.00	5150772.0
land	13821.0	0.000072	8.506096e-03	0.0	0.00	0.00	0.00	1.0
wrong_fragment	13821.0	0.023587	2.594537e-01	0.0	0.00	0.00	0.00	3.0
urgent	13821.0	0.000072	8.506096e-03	0.0	0.00	0.00	0.00	1.0
hot	13821.0	0.208451	2.228620e+00	0.0	0.00	0.00	0.00	77.0
num_failed_logins	13821.0	0.001158	4.335886e-02	0.0	0.00	0.00	0.00	3.0
logged_in	13821.0	0.395557	4.889878e-01	0.0	0.00	0.00	1.00	1.0
num_compromised	13821.0	0.136604	6.585177e+00	0.0	0.00	0.00	0.00	558.0
root_shell	13821.0	0.001592	3.986680e-02	0.0	0.00	0.00	0.00	1.0
su_attempted	13821.0	0.001085	4.251817e-02	0.0	0.00	0.00	0.00	2.0
num_root	13821.0	0.149049	7.333570e+00	0.0	0.00	0.00	0.00	629.0
num_file_creations	13821.0	0.012951	4.451606e-01	0.0	0.00	0.00	0.00	29.0
num_shells	13821.0	0.000434	2.083182e-02	0.0	0.00	0.00	0.00	1.0
num_access_files	13821.0	0.003907	7.878832e-02	0.0	0.00	0.00	0.00	5.0
num_outbound_cmds	13821.0	0.000000	0.000000e+00	0.0	0.00	0.00	0.00	0.0
is_host_login	13821.0	0.000000	0.000000e+00	0.0	0.00	0.00	0.00	0.0
is_guest_login	13821.0	0.009768	9.835159e-02	0.0	0.00	0.00	0.00	1.0
count	13821.0	85.519210	1.146702e+02	1.0	2.00	15.00	145.00	511.0
srv_count	13821.0	28.016641	7.332972e+01	1.0	2.00	8.00	18.00	511.0
serror_rate	13821.0	0.289716	4.491936e-01	0.0	0.00	0.00	1.00	1.0
srv_serror_rate	13821.0	0.287055	4.493109e-01	0.0	0.00	0.00	1.00	1.0
rerror_rate	13821.0	0.117925	3.181468e-01	0.0	0.00	0.00	0.00	1.0
srv_rerror_rate	13821.0	0.119803	3.220427e-01	0.0	0.00	0.00	0.00	1.0
same_srv_rate	13821.0	0.657631	4.406419e-01	0.0	0.09	1.00	1.00	1.0
diff_srv_rate	13821.0	0.061564	1.759797e-01	0.0	0.00	0.00	0.06	1.0
srv_diff_host_rate	13821.0	0.094152	2.528349e-01	0.0	0.00	0.00	0.00	1.0
dst_host_count	13821.0	182.723464	9.916096e+01	0.0	85.00	255.00	255.00	255.0
dst_host_srv_count	13821.0	114.613848	1.107906e+02	0.0	10.00	60.00	255.00	255.0
dst_host_same_srv_rate	13821.0	0.516795	4.490932e-01	0.0	0.05	0.50	1.00	1.0
dst_host_same_src_port_rate	13821.0	0.147501	3.084197e-01	0.0	0.00	0.00	0.06	1.0

In attack_class normal means 0, DOS means 1, PROBE means 2, R2L means 3 and U2R means 4.

```
train.loc[train.attack=='normal','attack_class']=0

train.loc[(train.attack=='back') | (train.attack=='land') | (train.attack=='pod') | (train.attack=='neptune') |
         (train.attack=='smurf') | (train.attack=='teardrop') | (train.attack=='apache2') | (train.attack=='udpstorm') |
         (train.attack=='processtable') | (train.attack=='worm') | (train.attack=='mailbomb'),'attack_class']=1

train.loc[(train.attack=='satan') | (train.attack=='ipsweep') | (train.attack=='nmap') | (train.attack=='portsweep') |
          (train.attack=='mscan') | (train.attack=='saint'),'attack_class']=2

train.loc[(train.attack=='guess_passwd') | (train.attack=='ftp_write') | (train.attack=='imap') | (train.attack=='phf') |
          (train.attack=='multihop') | (train.attack=='warezmaster') | (train.attack=='warezclient') | (train.attack=='spy') |
          (train.attack=='xlock') | (train.attack=='xsnoop') | (train.attack=='snmpguess') | (train.attack=='snmpgetattack') |
          (train.attack=='httptunnel') | (train.attack=='sendmail') | (train.attack=='named'),'attack_class']=3

train.loc[(train.attack=='buffer_overflow') | (train.attack=='loadmodule') | (train.attack=='rootkit') | (train.attack=='perl') |
          (train.attack=='sqlattack') | (train.attack=='xterm') | (train.attack=='ps'),'attack_class']=4


test.loc[test.attack=='normal','attack_class']=0

test.loc[(test.attack=='back') | (test.attack=='land') | (test.attack=='pod') | (test.attack=='neptune') |
         (test.attack=='smurf') | (test.attack=='teardrop') | (test.attack=='apache2') | (test.attack=='udpstorm') |
         (test.attack=='processtable') | (test.attack=='worm') | (test.attack=='mailbomb'),'attack_class']=1

test.loc[(test.attack=='satan') | (test.attack=='ipsweep') | (test.attack=='nmap') | (test.attack=='portsweep') |
          (test.attack=='mscan') | (test.attack=='saint'),'attack_class']=2
```

```
test.loc[(test.attack=='guess_passwd') | (test.attack=='ftp_write') | (test.attack=='imap') | (test.attack=='phf') |
          (test.attack=='multihop') | (test.attack=='warezmaster') | (test.attack=='warezclient') | (test.attack=='spy') |
          (test.attack=='xlock') | (test.attack=='xsnoop') | (test.attack=='snmpguess') | (test.attack=='snmpgetattack') |
          (test.attack=='httptunnel') | (test.attack=='sendmail') | (test.attack=='named'),'attack_class']=3

test.loc[(test.attack=='buffer_overflow') | (test.attack=='loadmodule') | (test.attack=='rootkit') | (test.attack=='perl') |
          (test.attack=='sqlattack') | (test.attack=='xterm') | (test.attack=='ps'),'attack_class']=4
```

```
train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_diff_srv_rate	dst_
0	0	udp	other	SF	146.0	0.0	0.0	0.0	0.0	0.0	...	0.60	
1	0	tcp	private	S0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.05	
2	0	tcp	http	SF	232.0	8153.0	0.0	0.0	0.0	0.0	...	0.00	
3	0	tcp	http	SF	199.0	420.0	0.0	0.0	0.0	0.0	...	0.00	
4	0	tcp	private	REJ	0.0	0.0	0.0	0.0	0.0	0.0	...	0.07	

5 rows × 44 columns

```
train.shape
```

```
(13822, 44)
```

```
# Import the ProfileReport class from ydata_profiling
from ydata_profiling import ProfileReport

output = ProfileReport(train) # Change pandas_profiling to ydata_profiling
output
```

```
Summarize dataset: 100%                783/783 [03:44<00:00,  1.41it/s, Completed]
Generate report structure: 100%        1/1 [00:31<00:00, 31.52s/it]
Render HTML: 100%                      1/1 [00:27<00:00, 27.78s/it]
```

Pandas Profiling Report   Overview Variables Interactions Correlations Missing values Sample

Overview

Brought to you by YData

Overview | Alerts 74 | Reproduction

Dataset statistics

Number of variables	44
Number of observations	13822
Missing cells	42
Missing cells (%)	< 0.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	4.6 MiB
Average record size in memory	352.0 B

Variable types

Numeric	27
Categorical	16
Text	1

Variables

Select Columns

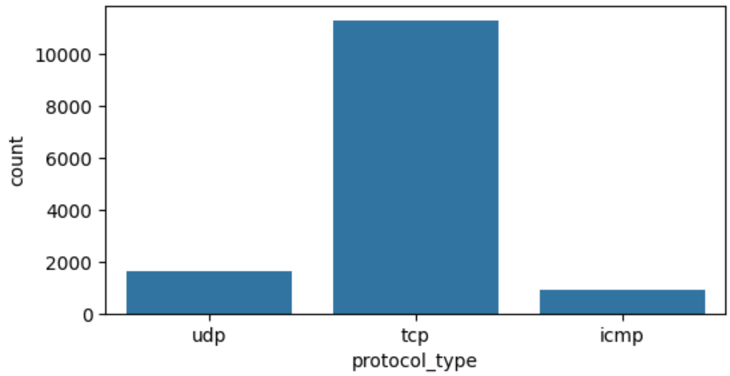
Exporting pandas profiling output to html file

```
output.to_file('pandas_profiling.html')
```

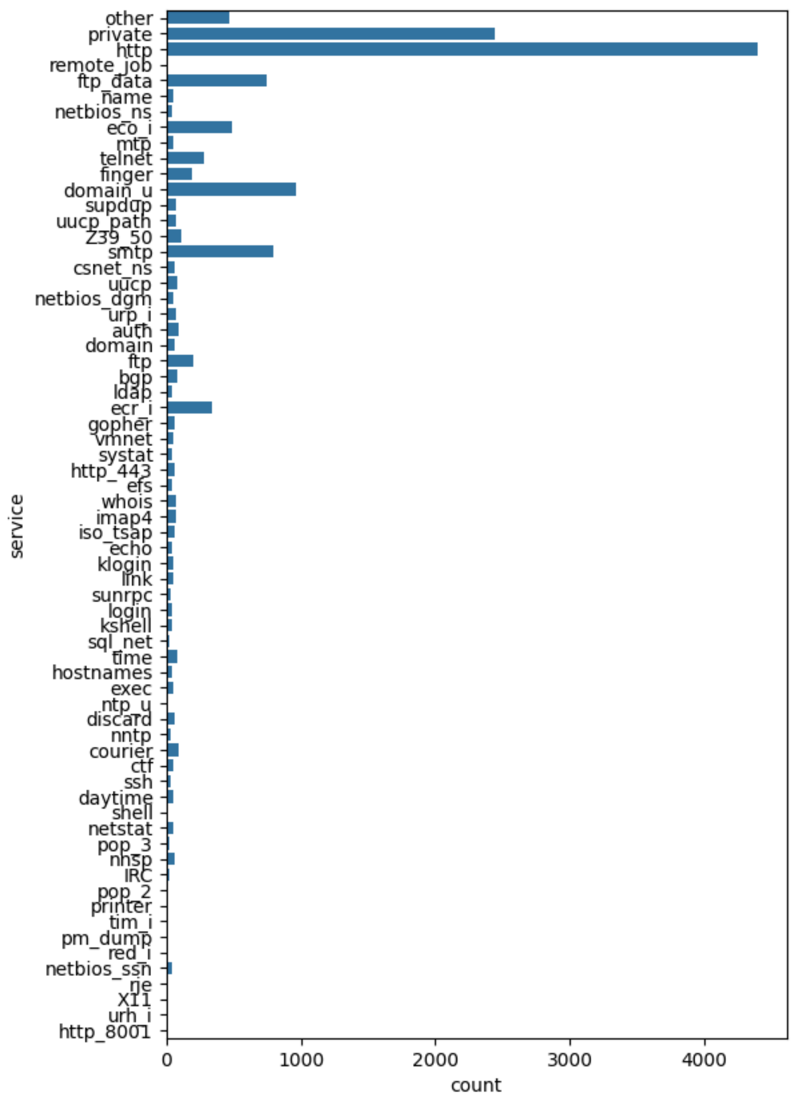
```
Export report to file: 100%        1/1 [00:00<00:00,  1.19it/s]
```

Basic Exploratory Analysis

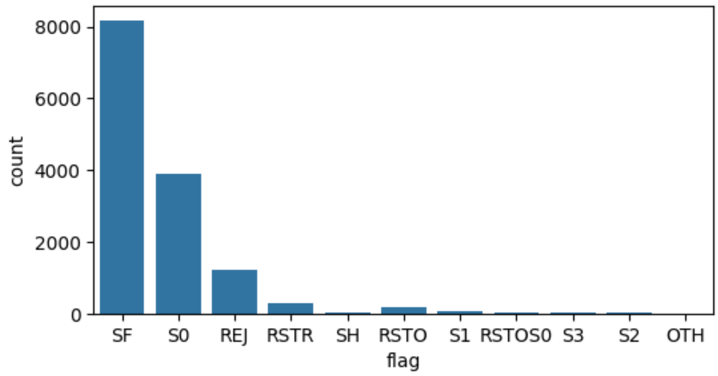
```
# Protocol type distribution
plt.figure(figsize=(6,3))
sns.countplot(x="protocol_type", data=train)
plt.show()
```



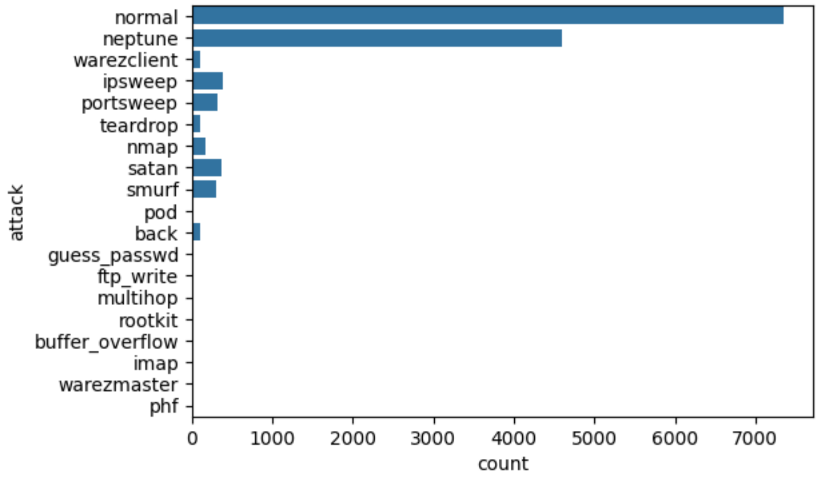

```
# service distribution
plt.figure(figsize=(6,10))
sns.countplot(y="service", data=train)
plt.show()
```



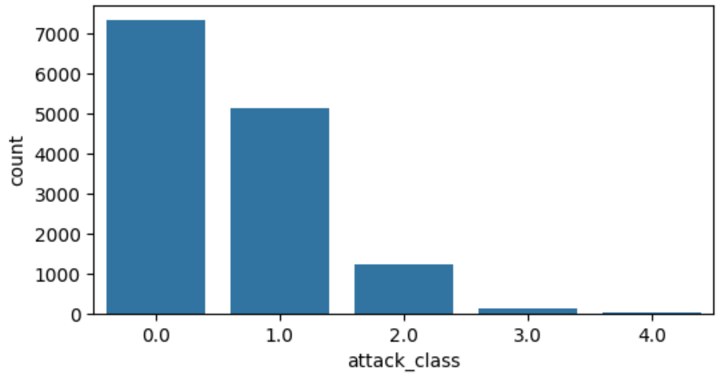

```
# flag distribution
plt.figure(figsize=(6,3))
sns.countplot(x="flag", data=train)
plt.show()
```




```
# attack distribution
plt.figure(figsize=(6,4))
sns.countplot(y="attack", data=train)
plt.show()
```




```
# attack class distribution
plt.figure(figsize=(6,3))
sns.countplot(x="attack_class", data=train)
plt.show()
```



identifying relationships (between Y & numerical independent variables by comparing means)

```
# Calculate the mean only for numeric columns.
train.groupby('attack_class').agg({col: 'mean' for col in train.select_dtypes(include=np.number).columns}).T
```

attack_class	0.0	1.0	2.0	3.0	4.0
duration	175.826051	0.000000	2345.770665	536.130435	197.000
src_bytes	12108.573976	1082.232031	309338.049433	313007.582609	781.400
dst_bytes	4447.782905	153.186719	17.149109	135416.226087	9363.000
land	0.000136	0.000000	0.000000	0.000000	0.000
wrong_fragment	0.000000	0.063672	0.000000	0.000000	0.000
urgent	0.000000	0.000000	0.000000	0.000000	0.200
hot	0.243637	0.035742	0.001621	7.843478	0.800
num_failed_logins	0.001497	0.000000	0.000000	0.043478	0.000
logged_in	0.714850	0.018945	0.006483	0.913043	1.000
num_compromised	0.237784	0.017188	0.000000	0.330435	3.000
root_shell	0.002314	0.000000	0.000000	0.026087	0.400
su_attempted	0.002042	0.000000	0.000000	0.000000	0.000
num_root	0.270178	0.000000	0.000000	0.469565	4.200
num_file_creations	0.020416	0.000000	0.000000	0.234783	0.400
num_shells	0.000681	0.000000	0.000000	0.008696	0.000
num_access_files	0.006669	0.000000	0.000000	0.043478	0.000
num_outbound_cmds	0.000000	0.000000	0.000000	0.000000	0.000
is_host_login	0.000000	0.000000	0.000000	0.000000	0.000
is_guest_login	0.013611	0.000000	0.000000	0.304348	0.000
count	22.873418	180.729492	71.652350	1.304348	1.200
srv_count	27.877501	33.035156	10.418152	3.478261	1.200
serror_rate	0.014117	0.751146	0.042464	0.018870	0.000
srv_serror_rate	0.012566	0.748811	0.031207	0.023043	0.000
rerror_rate	0.042813	0.150621	0.436880	0.043478	0.000
srv_rerror_rate	0.043772	0.151395	0.448533	0.048522	0.000
same_srv_rate	0.969147	0.189047	0.714652	0.991304	1.000
diff_srv_rate	0.029012	0.065393	0.243857	0.017391	0.000
srv_diff_host_rate	0.123140	0.004406	0.299028	0.043565	0.000
dst_host_count	147.949231	244.512305	142.846840	84.756522	103.000
dst_host_srv_count	189.394719	25.899219	44.235008	46.200000	18.400
dst_host_same_srv_rate	0.809649	0.120018	0.396021	0.764609	0.606

Observations:

- The length of time duration of connection for attack is higher than normal.
- Wrong fragments in the connection is only present in attack.
- Number of outbound commands in an ftp session are 0 in both normal and attack.

```
numeric_var_names=[key for key in dict(train.dtypes) if dict(train.dtypes)[key] in ['float64', 'int64', 'float32', 'int32']]
cat_var_names=[key for key in dict(train.dtypes) if dict(train.dtypes)[key] in ['object', 'O']]
```

```
numeric_var_names
```

```
['duration',
 'src_bytes',
 'dst_bytes',
 'land',
 'wrong_fragment',
 'urgent',
 'hot',
 'num_failed_logins',
 'logged_in',
 'num_compromised',
 'root_shell',
 'su_attempted',
 'num_root',
 'num_file_creations',
 'num_shells',
 'num_access_files',
```

```
        'num_outbound_cmds',
        'is_host_login',
        'is_guest_login',
        'count',
        'srv_count',
        'serror_rate',
        'srv_serror_rate',
        'rerror_rate',
        'srv_rerror_rate',
        'same_srv_rate',
        'diff_srv_rate',
        'srv_diff_host_rate',
        'dst_host_count',
        'dst_host_srv_count',
        'dst_host_same_srv_rate',
        'dst_host_diff_srv_rate',
        'dst_host_same_src_port_rate',
        'dst_host_srv_diff_host_rate',
        'dst_host_serror_rate',
        'dst_host_srv_serror_rate',
        'dst_host_rerror_rate',
        'dst_host_srv_rerror_rate',
        'last_flag',
        'attack_class']
```

```
cat_var_names
```

```
['protocol_type', 'service', 'flag', 'attack']
```

```
train_num=train[numeric_var_names]
test_num=test[numeric_var_names]
train_num.head(5)
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	dst_host_sa
0	0	146.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
2	0	232.0	8153.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	
3	0	199.0	420.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	
4	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 40 columns

```
train_cat=train[cat_var_names]
test_cat=test[cat_var_names]
train_cat.head(5)
```

	protocol_type	service	flag	attack
0	udp	other	SF	normal
1	tcp	private	S0	neptune
2	tcp	http	SF	normal
3	tcp	http	SF	normal
4	tcp	private	REJ	neptune

Data Audit Report

```
# Creating Data audit Report
def var_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.sum(), x.mean(), x.median(),  x.std(), x.var(), x.min(), x.dropna().quantile(0.01),
                  index=['N', 'NMISS', 'SUM', 'MEAN','MEDIAN', 'STD', 'VAR', 'MIN', 'P1' , 'P5' ,'P10' ,'P25' ,'P50' ,'P75' ,'P90' ,'P9

num_summary=train_num.apply(lambda x: var_summary(x)).T
```

```
num_summary
```

	N	NMISS	SUM	MEAN	MEDIAN	STD	VAR	MIN	P1	P5	P10
duration	13822.0	0.0	4.249115e+06	307.416799	0.00	2.715400e+03	7.373399e+06	0.0	0.0	0.00	0.00
src_bytes	13821.0	1.0	5.122257e+08	37061.403155	44.00	3.251583e+06	1.057279e+13	0.0	0.0	0.00	0.00
dst_bytes	13821.0	1.0	4.910302e+07	3552.783445	0.00	9.149915e+04	8.372094e+09	0.0	0.0	0.00	0.00
land	13821.0	1.0	1.000000e+00	0.000072	0.00	8.506096e-03	7.235366e-05	0.0	0.0	0.00	0.00
wrong_fragment	13821.0	1.0	3.260000e+02	0.023587	0.00	2.594537e-01	6.731625e-02	0.0	0.0	0.00	0.00
urgent	13821.0	1.0	1.000000e+00	0.000072	0.00	8.506096e-03	7.235366e-05	0.0	0.0	0.00	0.00
hot	13821.0	1.0	2.881000e+03	0.208451	0.00	2.228620e+00	4.966748e+00	0.0	0.0	0.00	0.00
num_failed_logins	13821.0	1.0	1.600000e+01	0.001158	0.00	4.335886e-02	1.879991e-03	0.0	0.0	0.00	0.00
logged_in	13821.0	1.0	5.467000e+03	0.395557	0.00	4.889878e-01	2.391091e-01	0.0	0.0	0.00	0.00
num_compromised	13821.0	1.0	1.888000e+03	0.136604	0.00	6.585177e+00	4.336455e+01	0.0	0.0	0.00	0.00
root_shell	13821.0	1.0	2.200000e+01	0.001592	0.00	3.986680e-02	1.589362e-03	0.0	0.0	0.00	0.00
su_attempted	13821.0	1.0	1.500000e+01	0.001085	0.00	4.251817e-02	1.807795e-03	0.0	0.0	0.00	0.00
num_root	13821.0	1.0	2.060000e+03	0.149049	0.00	7.333570e+00	5.378126e+01	0.0	0.0	0.00	0.00
num_file_creations	13821.0	1.0	1.790000e+02	0.012951	0.00	4.451606e-01	1.981680e-01	0.0	0.0	0.00	0.00
num_shells	13821.0	1.0	6.000000e+00	0.000434	0.00	2.083182e-02	4.339649e-04	0.0	0.0	0.00	0.00
num_access_files	13821.0	1.0	5.400000e+01	0.003907	0.00	7.878832e-02	6.207599e-03	0.0	0.0	0.00	0.00
num_outbound_cmds	13821.0	1.0	0.000000e+00	0.000000	0.00	0.000000e+00	0.000000e+00	0.0	0.0	0.00	0.00
is_host_login	13821.0	1.0	0.000000e+00	0.000000	0.00	0.000000e+00	0.000000e+00	0.0	0.0	0.00	0.00
is_guest_login	13821.0	1.0	1.350000e+02	0.009768	0.00	9.835159e-02	9.673036e-03	0.0	0.0	0.00	0.00
count	13821.0	1.0	1.181961e+06	85.519210	15.00	1.146702e+02	1.314925e+04	1.0	1.0	1.00	1.00
srv_count	13821.0	1.0	3.872180e+05	28.016641	8.00	7.332972e+01	5.377248e+03	1.0	1.0	1.00	1.00
serror_rate	13821.0	1.0	4.004160e+03	0.289716	0.00	4.491936e-01	2.017749e-01	0.0	0.0	0.00	0.00
srv_serror_rate	13821.0	1.0	3.967390e+03	0.287055	0.00	4.493109e-01	2.018803e-01	0.0	0.0	0.00	0.00
rerror_rate	13821.0	1.0	1.629840e+03	0.117925	0.00	3.181468e-01	1.012174e-01	0.0	0.0	0.00	0.00
srv_rerror_rate	13821.0	1.0	1.655800e+03	0.119803	0.00	3.220427e-01	1.037115e-01	0.0	0.0	0.00	0.00
same_srv_rate	13821.0	1.0	9.089120e+03	0.657631	1.00	4.406419e-01	1.941653e-01	0.0	0.0	0.01	0.03
diff_srv_rate	13821.0	1.0	8.508800e+02	0.061564	0.00	1.759797e-01	3.096886e-02	0.0	0.0	0.00	0.00
srv_diff_host_rate	13821.0	1.0	1.301280e+03	0.094152	0.00	2.528349e-01	6.392548e-02	0.0	0.0	0.00	0.00
dst_host_count	13821.0	1.0	2.525421e+06	182.723464	255.00	9.916096e+01	9.832897e+03	0.0	1.0	3.00	11.00
dst_host_srv_count	13821.0	1.0	1.584078e+06	114.613848	60.00	1.107906e+02	1.227456e+04	0.0	1.0	1.00	2.00
dst_host_same_srv_rate	13821.0	1.0	7.142630e+03	0.516795	0.50	4.490932e-01	2.016847e-01	0.0	0.0	0.00	0.01
dst_host_same_src_port_rate	13821.0	1.0	2.038610e+03	0.147501	0.00	3.084197e-01	9.512274e-02	0.0	0.0	0.00	0.00
dst_host_serror_rate	13821.0	1.0	3.997680e+03	0.289247	0.00	4.471870e-01	1.999762e-01	0.0	0.0	0.00	0.00

```
num_summary.to_csv('num_summary.csv')
```

Handling Outlier

```
#Handling Outliers
def outlier_capping(x):
    x = x.clip(upper=x.quantile(0.99))
    x = x.clip(lower=x.quantile(0.01))
    return x

train_num=train_num.apply(outlier_capping)
```

No missing in train dataset . So , Missing treatment not required .

```
def cat_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.value_counts()],
                  index=['N', 'NMISS', 'ColumnsNames'])

cat_summary=train_cat.apply(cat_summary)
```

```
cat_summary
```

	protocol_type	service	flag	attack
N	13822	13821	13821	13821
NMISS	0	1	1	1
ColumnsNames	protocol_type tcp 11294 udp 1629 icmp...	service http 4390 private 2443 do...	flag SF 8181 S0 3887 REJ 1...	attack normal 7347 neptune ...

Dummy Variable Creation

```
# An utility function to create dummy variable
def create_dummies( df, colname ):
    col_dummies = pd.get_dummies(df[colname], prefix=colname, drop_first=True)
    df = pd.concat([df, col_dummies], axis=1)
    df.drop( colname, axis = 1, inplace = True )
    return(df)
```

```
#for c_feature in categorical_features
for c_feature in ['protocol_type', 'service', 'flag', 'attack']:
    train_cat = create_dummies(train_cat,c_feature)
    test_cat = create_dummies(test_cat,c_feature)
train_cat.head()
```

	protocol_type_tcp	protocol_type_udp	service_X11	service_Z39_50	service_auth	service_bgp	service_courier	service_csnet_ns
0	False	True	False	False	False	False	False	False
1	True	False	False	False	False	False	False	False
2	True	False	False	False	False	False	False	False
3	True	False	False	False	False	False	False	False
4	True	False	False	False	False	False	False	False

5 rows × 95 columns

Final file for analysis

```
train_new = pd.concat([train_num, train_cat], axis=1)
test_new = pd.concat([test_num, test_cat], axis=1)
train_new.head()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	attack_norm
0	0.0	146.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	T
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	Fa
2	0.0	232.0	8153.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	T
3	0.0	199.0	420.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	T
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	Fa

5 rows × 135 columns

```
# correlation matrix (ranges from 1 to -1)
corrm=train_new.corr()
corrm
```

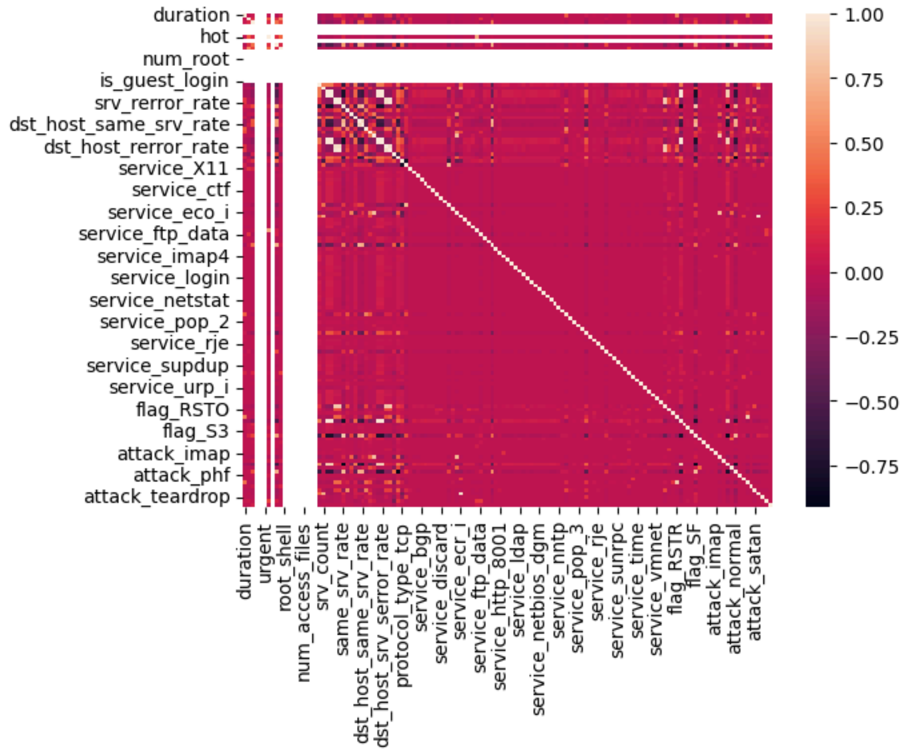
	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_comprom
duration	1.000000	0.018243	0.041559	NaN	NaN	NaN	0.010566	NaN	-0.059639	0.07
src_bytes	0.018243	1.000000	0.133922	NaN	NaN	NaN	0.354761	NaN	0.159882	0.54
dst_bytes	0.041559	0.133922	1.000000	NaN	NaN	NaN	0.131102	NaN	0.425328	0.26
land	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
wrong_fragment	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	
attack_satan	-0.023579	-0.027021	-0.056571	NaN	NaN	NaN	-0.021732	NaN	-0.129007	-0.01
attack_smurf	-0.021243	-0.002502	-0.053120	NaN	NaN	NaN	-0.020720	NaN	-0.120294	-0.01
attack_teardrop	-0.012379	-0.013924	-0.030954	NaN	NaN	NaN	-0.012074	NaN	-0.070097	-0.00
attack_warezclient	0.028960	0.048900	-0.013064	NaN	NaN	NaN	0.252143	NaN	0.104998	-0.00
attack_warezmaster	-0.002113	-0.002521	0.071466	NaN	NaN	NaN	0.029867	NaN	0.000173	-0.00

135 rows × 135 columns

```
corrm.to_csv('corrm.csv')
```

```
# visualize correlation matrix in Seaborn using a heatmap
sns.heatmap(corrm)
```

<Axes: >



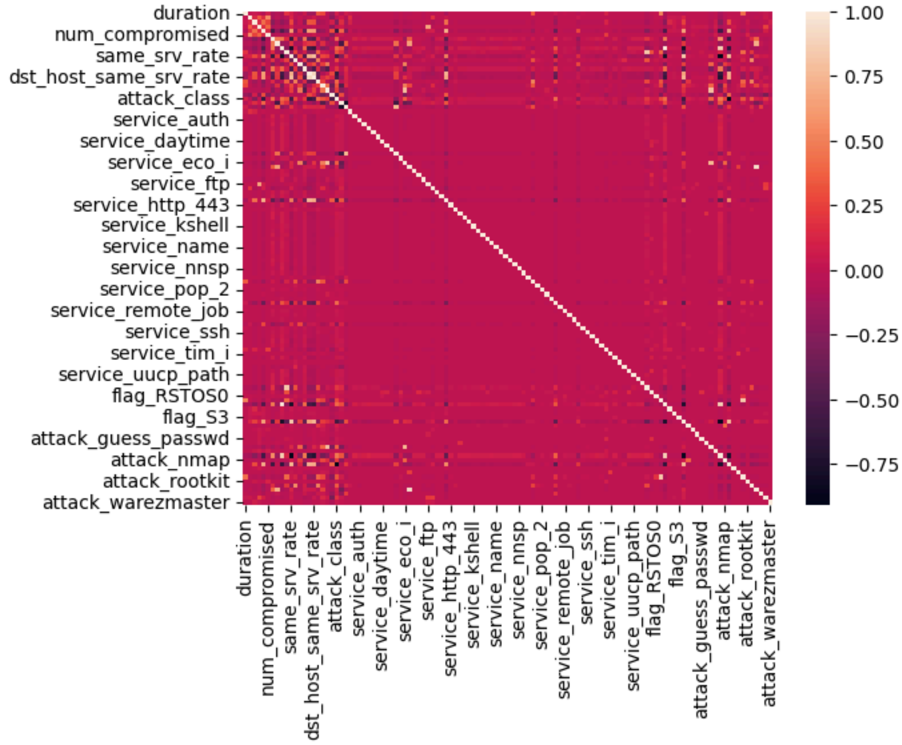
Dropping columns based on data audit report

```
- Based on low variance (near zero variance)
- High missings (>25% missings)
- High correlations between two numerical variables
```

```
train_new.drop(columns=['land','wrong_fragment','urgent','num_failed_logins',"root_shell","su_attempted","num_root",
                        "num_file_creations","num_shells","num_access_files","num_outbound_cmds","is_host_login","is_guest_login",
                        'dst_host_rerror_rate','dst_host_serror_rate','dst_host_srv_rerror_rate','dst_host_srv_serror_rate',
                        'num_root','num_outbound_cmds','srv_rerror_rate','srv_serror_rate'], axis=1, inplace=True)
```

```
sns.heatmap(train_new.corr())
```

<Axes: >



Variable reduction using Select K-Best technique

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer

# ... (your existing code) ...

X = train_new[train_new.columns.difference(['attack_class'])]

# Impute missing values using the mean strategy
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'
X_imputed = imputer.fit_transform(X)

# Before applying SelectKBest, ensure 'attack_class' has no missing values:
# If it's a classification problem and 'attack_class' is categorical,
# you might want to use the most frequent value


import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer

# ... (your existing code) ...

X = train_new[train_new.columns.difference(['attack_class'])]

# Impute missing values using the mean strategy
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'
X_imputed = imputer.fit_transform(X)

# Before applying SelectKBest, ensure 'attack_class' has no missing values:
# If it's a classification


import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer

# ... (your existing code) ...

X = train_new[train_new.columns.difference(['attack_class'])]
y = train_new['attack_class'] # Define the target variable 'y'

# Impute missing values using the mean strategy
imputer
```

```
▾ SimpleImputer
SimpleImputer()
```

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer

# ... (your existing code) ...

X = train_new[train_new.columns.difference(['attack_class'])]
y = train_new['attack_class'] # Define the target variable 'y'

# Impute missing values using the mean strategy for X
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'
X_imputed = imputer.fit_transform(X)

# Handle missing values in 'y' (attack_class)
# Here, we fill NaN values with the most frequent value
# Assuming 'attack_class' is a categorical variable
most_frequent_class = y.mode()[0]  # Get the most frequent class
y = y.fillna(most_frequent_class) # Fill NaN with most frequent class


# Apply SelectKBest with f_classif scoring function
# Selecting the top 15 features
X_new = SelectKBest(f_classif, k=15).fit(X_imputed, y)

# Now you can access the scores:
X_new.scores_
```

```
array([1.38400746e+01, 4.60653876e+00, 2.31009894e+01, 9.21989192e+00,
       2.42274173e+03, 4.60653876e+00, 3.85637880e+04, 8.88875386e+02,
       2.37770772e+07, 9.21989192e+00, 1.70590417e+01, 1.92098007e+03,
       9.21989192e+00, 2.31750904e+03, 2.69791876e+02, 8.93075006e+01,
       4.91659002e+02, 2.31009894e+01, 4.94335135e+03, 7.80980961e+02,
       6.60544901e+02, 2.05637313e+03, 2.32602436e+03, 3.17895984e+03,
       7.45668355e+03, 7.46047579e+03, 2.07550416e+03, 3.01658220e+02,
       3.07993355e+02, 6.24219343e+01, 5.10492303e+01, 1.19812383e+03,
       1.22656708e+04, 1.98745300e+01, 1.54424368e+00, 3.96875280e+00,
       1.23846795e+04, 1.12468817e+02, 1.70059180e+01, 3.48035127e+03,
       6.51663283e+03, 1.69270206e+01, 5.67712919e+02, 4.15071997e+02,
       8.67729879e+02, 1.52344585e+04, 1.13326752e+04, 4.27419702e+00,
       8.92813753e+01, 2.63609315e+01, 6.40914481e+01, 7.78339975e+01,
       4.56603911e+01, 4.89735673e+01, 4.54753148e+01, 4.72072911e+01,
       3.56652493e+01, 4.91024437e+02, 2.83421150e+01, 2.55241331e+03,
       2.43721069e+02, 3.59802014e+01, 4.05093523e+01, 3.63674852e+01,
       1.57259336e+01, 5.77594077e+01, 4.63694836e+01, 4.11677190e+01,
       3.29218360e+03, 4.85878557e+01, 4.60653876e+00, 5.41803058e+01,
       5.50211165e+01, 4.28804807e+01, 3.25414006e+01, 3.68411317e+01,
       4.00775926e+01, 3.33895237e+01, 4.28804807e+01, 3.94267293e+01,
       4.02895060e+01, 3.25414006e+01, 3.08251134e+01, 4.28804807e+01,
       5.68072165e+01, 2.56902495e+01, 7.50471371e+00, 2.96921932e+02,
       4.60653876e+00, 7.24674739e+00, 9.75000021e-01, 5.10370235e+00,
       1.78599452e+03, 8.81104662e-01, 8.53855066e+00, 5.12346063e+00,
       3.57910229e+00, 3.29169186e+02, 1.39487593e+01, 1.67821924e+01,
       2.31318911e+01, 5.94246365e+01, 2.85971690e+01, 1.64715973e+01,
       8.49786215e-01, 4.20146869e+01, 4.40492368e-01, 2.84132035e+01,
       7.34368876e+01, 5.42920032e+01, 4.89735673e+01, 5.80380925e+01,
       7.85873303e+00, 5.35112950e+01, 8.06074391e+02])
```

Final list of variable selected for the model building using Select KBest

attack_neptune, attack_normal, attack_satan, count, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_same_srv_rate, dst_host_srv_count, flag_S0, flag_SF, last_flag, logged_in, same_srv_rate, serror_rate, service_http

```
train=train_new
test=test_new
```

Model Building

```
top_features=['attack_neptune','attack_normal','attack_satan','count','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_s
X_train = train[top_features]
y_train = train['attack_class']
X_test = test[top_features]
y_test = test['attack_class']
```

Building logistic Regression

1) LogisticRegression

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Handle missing values in 'y_train' (attack_class) before fitting the model
# Here,
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
from sklearn.impute import SimpleImputer
imputer = SimpleImputer()
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming X_train, X_test, y_train, y_test are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
```

2) RidgeClassifier

```
from sklearn.linear_model import RidgeClassifier
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import RidgeClassifier

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Handle missing values in y_train (if any) before fitting the model
# Replace NaN values with the most frequent value in y_train
# Assuming 'attack_class' is a categorical variable
most_frequent_class = y_train.mode()[0]  # Get the most frequent class
y_train = y_train.fillna(most_frequent_class) # Fill NaN with most frequent class


# Create and train the RidgeClassifier using the imputed data
rc_clf = RidgeClassifier().fit(X_train_imputed, y_train)

# Predict using the imputed test data
y_pred = rc_clf.predict(X_test_imputed)
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import RidgeClassifier

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Handle missing values in y_train (if any) before fitting the model
# Replace NaN values with the most frequent value in y_train
# Assuming 'attack_class' is a categorical variable
most_frequent_class = y_train.mode()[0]  # Get the most frequent class
y_train = y_train.fillna(most_frequent_class) # Fill NaN with most frequent class


# Create and train the RidgeClassifier using the imputed data
rc_clf = RidgeClassifier().fit(X_train_imputed, y_train)

# Predict using the imputed test data
y_pred = rc_clf.predict(X_test_imputed) # Use X_test_imputed here, not X_test
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import accuracy_score

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Handle missing values in y_train (if any) before fitting the model
# Replace NaN values with the most frequent value in y_train
# Assuming 'attack_class' is a categorical variable
most_frequent_class = y_train.mode()[0]  # Get the most frequent class
y_train = y_train.fillna(most_frequent_class) # Fill NaN with most frequent class

# Create and train the RidgeClassifier using the imputed data
rc_clf = RidgeClassifier().fit(X_train_imputed, y_train)

# Predict using the imputed test data
y_pred = rc_clf.predict(X_test_imputed) # Use X_test_imputed here, not X_test

# Handle missing values in y_test before calculating accuracy
# Replace NaN values with the most frequent value in y_test
# (or any other strategy appropriate for your data)
# Assuming 'attack_class' is a categorical variable
most_frequent_class_test = y_test.mode()[0]
y_test = y_test.fillna(most_frequent_class_test) # Fill NaN in y_test
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.7569535459443716
```

K-Nearest Neighbors

1) KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier

# ... (your existing code) ...
```

```
# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Now, use the imputed data for training the KNeighborsClassifier:
k_neigh = KNeighborsClassifier(n_neighbors=3)
k_neigh.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# For prediction, also use the imputed test data:
y_pred = k_neigh.predict(X_test_imputed)  # Use X_test_imputed here
```

```
# Use the imputed X_test data for prediction:
y_pred = k_neigh.predict(X_test_imputed)
y_pred
```

```
array([1., 0., 2., ..., 1., 0., 0.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.716251638269987
```

3) NearestCentroid

```
from sklearn.neighbors import NearestCentroid # Correct import statement
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.neighbors import NearestCentroid

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test) # Changed 'imp' to 'imputer'
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.neighbors import NearestCentroid

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Create and fit the NearestCentroid classifier using the imputed training data
nc = NearestCentroid()  # You likely missed this step in your original code
nc.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# Predict using the imputed test data
y_pred = nc.predict(X_test_imputed)  # Use X_test_imputed here, not X_test
y_pred
```

```
array([1., 2., 2., ..., 0., 0., 0.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.6051405271588758
```

Discriminant Analysis

1) LinearDiscriminantAnalysis

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test) # Apply the transform method to X_test

# Create and fit the LinearDiscriminantAnalysis model
lda = LinearDiscriminantAnalysis() # Make sure to create the 'lda' object
lda.fit(X_train_imputed, y_train)   # Use the imputed training data

# Now predict using the imputed test data
y_pred = lda.predict(X_test_imputed) # Use X_test_imputed here, not X_test
y_pred
```

```
array([1., 0., 2., ..., 2., 2., 2.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.753822629969419
```

2) QuadraticDiscriminantAnalysis

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Now, fit the QDA model using the imputed data
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# For prediction, also use the imputed test data:
y_pred = qda.predict(X_test_imputed)  # Use X_test_imputed here
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:947: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: divide by zero encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: invalid value encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:975: RuntimeWarning: divide by zero encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

```
# In cell ipython-input-129-1b48cfcd2990
# Replace:
# y_pred=qda.predict(X_test)
# With:
y_pred = qda.predict(X_test_imputed)  # Use the imputed test data for prediction
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: divide by zero encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: invalid value encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:975: RuntimeWarning: divide by zero encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.43017329255861364
```

Decision Trees

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
import sklearn.tree as dt
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz # Removed 'export'
# import export_text if you need to export the tree as text
from sklearn.model_selection import GridSearchCV
```

```
clf_tree = DecisionTreeClassifier( max_depth = 5)
clf_tree=clf_tree.fit( X_train, y_train )
```

```
# In cell ipython-input-135-1b48cfcd2990
# Replace:
# y_pred=qda.predict(X_test)
# With:
y_pred = qda.predict(X_test_imputed)  # Use the imputed test data for prediction
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: divide by zero encountered in power
  X2 = np.dot(Xm, R * (S ** (-0.5)))
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:972: RuntimeWarning: invalid value encountered in multiply
  X2 = np.dot(Xm, R * (S ** (-0.5)))
/usr/local/lib/python3.10/dist-packages/sklearn/discriminant_analysis.py:975: RuntimeWarning: divide by zero encountered in log
  u = np.asarray([np.sum(np.log(s)) for s in self.scalings_])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.43017329255861364
```

Fine Tuning the parameters

```
param_grid = {'max_depth': np.arange(3, 9),
             'max_features': np.arange(3,9)}
```

```
tree = GridSearchCV(DecisionTreeClassifier(), param_grid, cv = 5)
tree.fit( X_train, y_train )
```

```
GridSearchCV
best_estimator_: DecisionTreeClassifier
DecisionTreeClassifier
```

```
tree.best_score_
```

```
0.9973229823904868
```

```
tree.best_estimator_
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=8, max_features=8)
```

```
tree.best_params_
```

```
{'max_depth': 8, 'max_features': 8}
```

Building Final Decision Tree Model

```
clf_tree = DecisionTreeClassifier( max_depth = 8, max_features=8 )
clf_tree.fit( X_train, y_train )
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=8, max_features=8)
```

Feature Relative Importance

```
clf_tree.feature_importances_
```

```
array([1.03884452e-03, 7.30091426e-01, 1.43908666e-02, 1.25069325e-01,
       1.10756908e-01, 5.52147191e-03, 8.55001244e-03, 1.62838862e-03,
       0.00000000e+00, 0.00000000e+00, 2.56159959e-04, 1.51379177e-03,
       2.09875752e-04, 2.50507955e-04, 7.22421562e-04])
```

```
# summarize the selection of the attributes
import itertools
feature_map = [(i, v) for i, v in itertools.zip_longest(X_train.columns, clf_tree.feature_importances_)]
```

```
feature_map
```

```
[('attack_neptune', 0.0010388445177278512),
 ('attack_normal', 0.7300914262850192),
 ('attack_satan', 0.014390866557389389),
 ('count', 0.12506932501707269),
 ('dst_host_diff_srv_rate', 0.11075690764883102),
 ('dst_host_same_src_port_rate', 0.005521471914380816),
 ('dst_host_same_srv_rate', 0.008550012441896423),
 ('dst_host_srv_count', 0.0016283886246263005),
 ('flag_S0', 0.0),
 ('flag_SF', 0.0),
 ('last_flag', 0.00025615995934337264),
 ('logged_in', 0.001513791765490236),
 ('same_srv_rate', 0.00020987575182477013),
 ('serror_rate', 0.00025050795483314204),
 ('service_http', 0.000722421561564624)]
```

```
Feature_importance = pd.DataFrame(feature_map, columns=['Feature', 'importance'])
Feature_importance.sort_values('importance', inplace=True, ascending=False)
Feature_importance
```

	Feature	importance
1	attack_normal	0.730091
3	count	0.125069
4	dst_host_diff_srv_rate	0.110757
2	attack_satan	0.014391
6	dst_host_same_srv_rate	0.008550
5	dst_host_same_src_port_rate	0.005521
7	dst_host_srv_count	0.001628
11	logged_in	0.001514
0	attack_neptune	0.001039
14	service_http	0.000722
10	last_flag	0.000256
13	serror_rate	0.000251
12	same_srv_rate	0.000210
8	flag_S0	0.000000
9	flag_SF	0.000000

```
tree_test_pred = pd.DataFrame( { 'actual':  y_test,
                                'predicted': clf_tree.predict( X_test ) } )
```

```
tree_test_pred.sample( n = 10 )
```

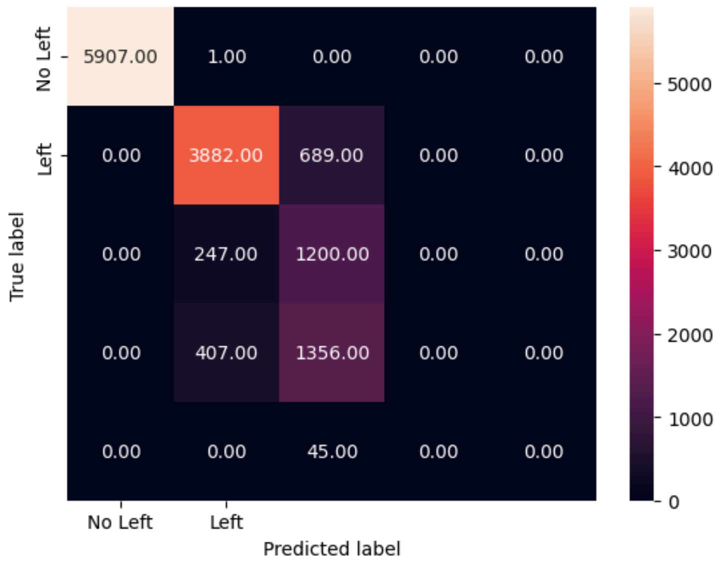
	actual	predicted
6971	1.0	1.0
6970	3.0	2.0
12573	0.0	0.0
7465	0.0	0.0
9923	0.0	0.0
753	2.0	2.0
3313	1.0	1.0
4658	1.0	2.0
12053	0.0	0.0
4543	0.0	0.0

```
accuracy_score( tree_test_pred.actual, tree_test_pred.predicted )
```

```
0.8001310615989515
```

```
tree_cm = metrics.confusion_matrix(tree_test_pred.actual, tree_test_pred.predicted)
# Removed [1, 0] argument as it's not required.

sns.heatmap(tree_cm, annot=True,
         fmt='.2f',
         xticklabels = ["No Left", "Left"] , # Assuming 0 is "No Left" and 1 is "Left"
         yticklabels = ["No Left", "Left"]) # Assuming 0 is "No Left" and 1 is "Left"

plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```



Naive Bayes Model

1) BernoulliNB

```
from sklearn.naive_bayes import BernoulliNB
```

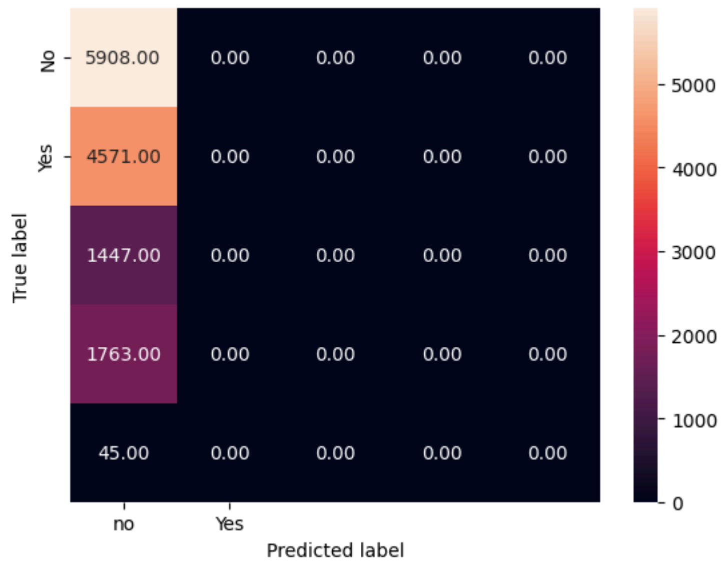
```
bnb_clf = BernoulliNB()
bnb_clf.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-153-0d38021cd815> in <cell line: 2>()
      1 bnb_clf = BernoulliNB()
----> 2 bnb_clf.fit(X_train, y_train)

                              8 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype,
estimator_name, input_name)
    170                 "#estimators-that-handle-nan-values"
    171             )
--> 172         raise ValueError(msg_err)
    173
    174

ValueError: Input X contains NaN.
BernoulliNB does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider
sklearn.ensemble.HistGradientBoostingClassifier and Regressor which accept missing values encoded as NaNs natively. Alternatively,
it is possible to preprocess the data, for instance by using an imputer transformer in a pipeline or drop samples with missing
values. See https://scikit-learn.org/stable/modules/impute.html You can find a list of all estimators that handle NaN values at the
following page: https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import BernoulliNB

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'
```

```
nb_cm = metrics.confusion_matrix( y_test,y_pred )
sns.heatmap(nb_cm, annot=True,  fmt='.2f', xticklabels = ["no", "Yes"] , yticklabels = ["No", "Yes"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```




```
accuracy_score( y_test, y_pred )
```

```
0.43017329255861364
```

2) GaussianNB

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb_clf = GaussianNB()
gnb_clf.fit(X_train, y_train)
```

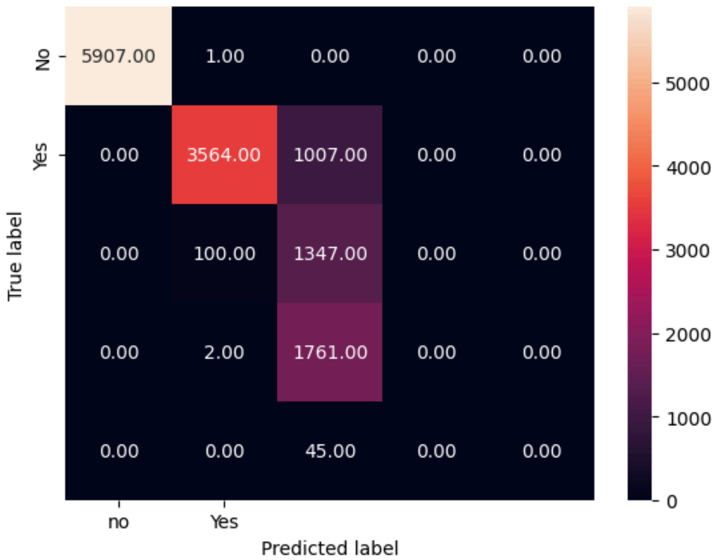
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-158-e874259f9bbe> in <cell line: 2>()
      1 gnb_clf = GaussianNB()
----> 2 gnb_clf.fit(X_train, y_train)

7 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in _assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name, input_name)
    170                 "#estimators-that-handle-nan-values"
    171             )
--> 172         raise ValueError(msg_err)
    173 
    174 

ValueError: Input X contains NaN.
GaussianNB does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider sklearn.ensemble.HistGradientBoostingClassifier and Regressor which accept missing values encoded as NaNs natively. Alternatively, it is possible to preprocess the data, for instance by using an imputer transformer in a pipeline or drop samples with missing values. See https://scikit-learn.org/stable/modules/impute.html You can find a list of all estimators that handle NaN values at the following page: https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values
```

```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB

# Assuming X_train, X_test, y_train, y_test are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Now, fit the GaussianNB model using the imputed data
gnb_clf = GaussianNB()
gnb_clf.fit(X_train_imputed, y_train)

# For prediction, also use the imputed test data:
y_pred = gnb_clf.predict(X_test_imputed)
y_pred
```

```
array([1., 0., 2., ..., 1., 2., 1.])
```

```python
nb_cm = metrics.confusion_matrix( y_test, y_pred )
sns.heatmap(nb_cm, annot=True,  fmt='.2f', xticklabels = ["no", "Yes"] , yticklabels = ["No", "Yes"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```




```python
accuracy_score( y_test, y_pred )
```

```
0.7876802096985583
```

Support Vector Machine (SVM)

1) LinearSVC

```
from sklearn.svm import LinearSVC
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.svm import LinearSVC

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Now, fit the LinearSVC model using the imputed data
svm_clf = LinearSVC(random_state=0, tol=1e-5)
svm_clf.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# ... (rest of your code) ...
```

```
LinearSVC
LinearSVC(random_state=0, tol=1e-05)
```


```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.svm import LinearSVC

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Now, fit the LinearSVC model using the imputed data
svm_clf = LinearSVC(random_state=0, tol=1e-5)
svm_clf.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# ... (rest of your code) ...
```

```
LinearSVC
LinearSVC(random_state=0, tol=1e-05)
```


```
accuracy_score( y_test, y_pred )
```

```
0.7876802096985583
```

2) SVC

```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline

model = SVC(kernel='rbf', class_weight='balanced',gamma='scale')
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
```

```
X_train_imputed = imputer.fit_transform(X_train)

# Transform the test data using the trained imputer
X_test_imputed = imputer.transform(X_test)

# Now, fit the SVC model using the imputed data
model = SVC(kernel='rbf', class_weight='balanced', gamma='scale')
model.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# For prediction, also use the imputed test data:
y_pred = model.predict(X_test_imputed)  # Use X_test_imputed here
```

```
y_pred = model.predict(X_test_imputed)  # Use the imputed data for prediction
```

```
accuracy_score( y_test, y_pred )
```

```
0.6791175185670598
```

Stochastic Gradient Descent (SGD)

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Now, fit the SGDClassifier model using the imputed data
model = SGDClassifier(loss="hinge", penalty="l2")
model.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# For prediction, also use the imputed test data after fitting the imputer on X_test
# ... imputer.fit(X_test) ... if required for X_test
# X_test_imputed = imputer.transform(X_test)
# y_pred = model.predict(X_test_imputed)
```

```
SGDClassifier
SGDClassifier()
```

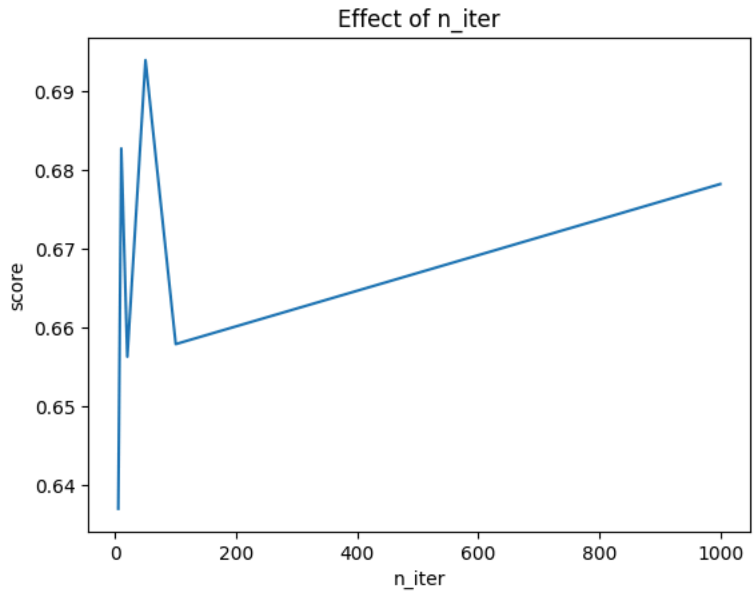
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.linear_model import SGDClassifier

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Transform the test data using the trained imputer
X_test_imputed = imputer.transform(X_test)

# Now, fit the SGDClassifier model using the imputed data
model = SGDClassifier(loss="hinge", penalty="l2")
model.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# For prediction, use the imputed test data
y_pred = model.predict(X_test_imputed) # Use X_test_imputed here
```

```
accuracy_score( y_test, y_pred )
```

```
0.6680500946555993
```

```
import matplotlib.pyplot as plt # Importing matplotlib for plotting

n_iters = [5, 10, 20, 50, 100, 1000]
scores = []
for n_iter in n_iters:
    model = SGDClassifier(loss="hinge", penalty="l2", max_iter=n_iter)

    # Create a SimpleImputer to replace NaN values with the mean of each column
    imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

    # Fit the imputer on the training data and transform it
    X_train_imputed = imputer.fit_transform(X_train)
    X_test_imputed = imputer.transform(X_test)  # Transform test data as well

    model.fit(X_train_imputed, y_train)  # Use imputed data for training
    scores.append(model.score(X_test_imputed, y_test))  # Use imputed data for scoring

plt.title("Effect of n_iter")
plt.xlabel("n_iter")
plt.ylabel("score")
plt.plot(n_iters, scores)
plt.show() # Display the plot
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of iter
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of iter
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of iter
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of iter
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of iter
  warnings.warn(
```




```
# losses
losses = ["hinge", "log_loss", "modified_huber", "perceptron", "squared_hinge"] # Changed 'log' to 'log_loss'
scores = []
for loss in losses:
    model = SGDClassifier(loss=loss, penalty="l2", max_iter=1000)

    # Create a SimpleImputer to replace NaN values with the mean of each column
    imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

    # Fit the imputer on the training data and transform it
    X_train_imputed = imputer.fit_transform(X_train)
    X_test_imputed = imputer.transform(X_test)  # Transform test data as well

    model.fit(X_train_imputed, y_train) # Use the imputed data for training
    scores.append(model.score(X_test_imputed, y_test)) # Use the imputed data for scoring

plt.xlabel("loss")
plt.ylabel("score")
plt.title("Effect of loss")
x = np.arange(len(losses))
plt.xticks(x, losses)
plt.plot(x, scores)
```

```
[<matplotlib.lines.Line2D at 0x78fb75858d60>]
```


```
from sklearn.model_selection import GridSearchCV

params = {
    "loss" : ["hinge", "log", "squared_hinge", "modified_huber"],
    "alpha" : [0.0001, 0.001, 0.01, 0.1],
    "penalty" : ["l2", "l1", "none"],
}

model = SGDClassifier(max_iter=100)
clf = GridSearchCV(model, param_grid=params)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier


params = {
    "sgd__loss" : ["hinge", "log_loss", "squared_hinge", "modified_huber"], # Updated 'log' to 'log_loss'
    "sgd__alpha" : [0.0001, 0.001, 0.01, 0.1],
    "sgd__penalty" : ["l2", "l1", "none"],
}

# Create a pipeline with imputation and SGDClassifier
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values
    ('sgd', SGDClassifier(max_iter=100))  # Your SGDClassifier
])

# Use the pipeline in GridSearchCV
clf = GridSearchCV(pipeline, param_grid=params)

clf.fit(X_train, y_train)
print(clf.best_score_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning: Maximum number of i
```

```
y_pred=clf.predict(X_test)
y_pred
```

```
array([1., 0., 2., ..., 1., 1., 1.])
```

```
accuracy_score( y_test, y_pred )
```

```
0.8048638415610893
```

Neural Network Model

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(X_train)
```

```
▾ StandardScaler
StandardScaler()
```

```
# Now apply the transformations to the data:
train_X = scaler.transform(X_train)
test_X = scaler.transform(X_test)
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

# Assuming X_train, X_test, y_train are already defined...

# 1. Create a SimpleImputer to handle missing values
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# 2. Create a StandardScaler for feature scaling
scaler = StandardScaler()

# 3. Fit the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# 4. Fit the scaler on the imputed training data and transform both training and testing data
train_X = scaler.fit_transform(X_train_imputed)
```

```
test_X = scaler.transform(X_test_imputed)

# 5. Now, fit the MLPClassifier using the imputed and scaled data
mlp = MLPClassifier(hidden_layer_sizes=(30,30,30))
mlp.fit(train_X,y_train)

# ... (rest of your code) ...
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(30, 30, 30))
```

```
y_pred=mlp.predict(test_X)
y_pred
```

```
array([1., 0., 2., ..., 1., 1., 1.])
```

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

```
[[5836    1   71    0    0]
 [  28 4196  347    0    0]
 [   2  170 1275    0    0]
 [   0  194 1569    0    0]
 [   0    0   45    0    0]]
```

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

```
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99      5908
         1.0       0.92      0.92      0.92      4571
         2.0       0.39      0.88      0.54      1447
         3.0       0.00      0.00      0.00      1763
         4.0       0.00      0.00      0.00        45

    accuracy                           0.82     13734
   macro avg       0.46      0.56      0.49     13734
weighted avg       0.77      0.82      0.79     13734

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
mlp.coefs_
```

```
[array([[-0.16381647, -0.13886762, -0.42705598, -0.11983467, -0.12828418,
         -0.068113  , -0.45247142,  0.27226362, -0.34375123,  0.44536006,
          0.26585685, -0.21292827,  0.20069089, -0.45266859,  0.1301322 ,
         -0.29335015,  0.25772786, -0.16088247,  0.20378545,  0.10460678,
         -0.00772882, -0.04721868, -0.07463312, -0.3811215 , -0.00948838,
          0.40312568, -0.10944239,  0.15784948,  0.02013705, -0.18850366],
        [-0.13745087, -0.31400943,  0.32557044, -0.09326639,  0.10851818,
         -0.33335371,  0.04204226,  0.12491171, -0.32007162, -0.46614801,
          0.08393662,  0.35602809, -0.37454684, -0.19767313, -0.08023737,
          0.1091493 ,  0.1168984 , -0.35463926, -0.07590011,  0.399434  ,
          0.41451984,  0.58258095,  0.05959135, -0.46114715, -0.08765147,
         -0.48270997, -0.48176136, -0.4135145 , -0.07613329,  0.24144511],
        [-0.02900025,  0.26782692, -0.15381072, -0.03874818, -0.06009253,
         -0.30779476, -0.41554704, -0.15148227,  0.15423687, -0.15857007,
          0.14579337,  0.38027141,  0.13787317,  0.16519661, -0.16317745,
          0.21973051, -0.03461887,  0.01554681, -0.06018077, -0.38116682,
          0.10266403, -0.50437255,  0.1029926 ,  0.48356823, -0.17064275,
          0.0050792 ,  0.19351038, -0.22190426,  0.12694695,  0.21835333],
        [-0.03623359,  0.27202212, -0.27293167,  0.08550123, -0.16747281,
         -0.10816762, -0.28372442,  0.18837056, -0.34875198,  0.36722723,
          0.31968847,  0.07011801,  0.19657569,  0.13603313, -0.61495636,
         -0.14684475, -0.06576889,  0.30546058,  0.55615809, -0.3152281 ,
         -0.36142523,  0.17955186, -0.57268175, -0.26426678, -0.24639946,
         -0.06252453,  0.0285455 ,  0.30317299,  0.37782117,  0.32898267],
        [ 0.37763542, -0.27664256, -0.35498207,  0.33671142,  0.14407578,
         -0.19158696, -0.27953037,  0.20244677,  0.20413902,  0.19914399,
         -0.07769039,  0.11908977,  0.27976912,  0.42464719, -0.42820243,
         -0.31311612, -0.21654518,  0.23940578, -0.50374315,  0.36248117,
         -0.26582813, -0.12707359,  0.17078448,  0.18943372,  0.13763424,
         -0.07671088,  0.0784714 ,  0.08636306, -0.18532381, -0.00168784],
        [ 0.01463417, -0.01595736, -0.0648834 ,  0.18121716, -0.22260824,
          0.10125081, -0.19262256,  0.19193269,  0.22174033, -0.12165301,
         -0.36703021, -0.25028795, -0.30359202,  0.13001923,  0.00411871,
          0.39727941,  0.23023959, -0.21693795,  0.04626115,  0.10831735,
```

```
       -0.20964172, -0.0717448 ,  0.38470593,  0.2777985 , -0.29655654,
       -0.21981664, -0.06714021,  0.03120111, -0.25366644,  0.41744564],
      [-0.11562287, -0.0064908 ,  0.13668061, -0.08354727,  0.10564777,
        0.23892476, -0.01581051, -0.32925181, -0.09567781, -0.15514981,
        0.14923213,  0.06703209, -0.02254422, -0.08036625,  0.14554192,
        0.08320596,  0.29516748,  0.23784174,  0.12644663,  0.3227909 ,
        0.39904543, -0.03242125,  0.01421783,  0.19076823,  0.32888299,
       -0.15009442, -0.34315811,  0.12240914,  0.45769156, -0.35449893],
      [-0.30309617,  0.30664128,  0.20128167, -0.16065271, -0.10690856,
       -0.20160023, -0.01130471,  0.17087794,  0.09731222,  0.2251395 ,
       -0.19097565, -0.06098674, -0.26514486, -0.37822862, -0.07349166,
       -0.24729698,  0.439292  ,  0.0984922 ,  0.32569211, -0.03305327,
       -0.30754262, -0.14371729, -0.28440105, -0.19403465, -0.25567332,
        0.17920834,  0.0782346 , -0.0388143 , -0.00608198, -0.22251162],
      [ 0.08873796, -0.17024812,  0.07567146,  0.19010715,  0.16008412,
       -0.19485302,  0.13663009, -0.32636003, -0.10541986,  0.33875046,
        0.12079149,  0.28307021, -0.02788808,  0.13909932, -0.04869957,
       -0.10545785, -0.25316429, -0.27220661,  0.24599567, -0.2486589 ,
       -0.21269697,  0.12871918, -0.13983505,  0.07586136,  0.21095537,
        0.04116207, -0.27930651, -0.10076516, -0.31878606, -0.44666628],
      [ 0.51997505,  0.37284496, -0.15254913, -0.46853094, -0.28994355,
       -0.29100967,  0.14187794,  0.3150806 , -0.12914963,  0.1890737 ,
       -0.2345409 , -0.19172794, -0.26878816, -0.52433866,  0.28036026,
```

```
accuracy_score( y_test, y_pred )
```

```
0.8232852774137178
```

Combine Model Predictions Into Ensemble Predictions

The three most popular methods for combining the predictions from different models are:

Bagging-> Building multiple models (typically of the same type) from different subsamples of the training dataset.

Boosting-> Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.

Voting-> Building multiple models (typically of differing types) and simple statistics (like calculating the mean) are used to combine predictions.

Bagging Algorithms

Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset (with replacement) and training a model for each sample.

The final output prediction is averaged across the predictions of all of the sub-models.

The three bagging models covered in this section are as follows:

1) Bagged Decision Trees

2) Random Forest

3) Extra Trees

1. Bagged Decision Trees

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning.

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier


seed = 7
# Set shuffle=True to enable shuffling and ensure random_state is effective
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
cart = DecisionTreeClassifier()
num_trees = 100
# Use 'estimator' instead of 'base_estimator'
model = BaggingClassifier(estimator=cart, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean())
```

```
0.9984809339791744
```

```
model.fit(X_train, y_train)
```

```
BaggingClassifier
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

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

```
array([1., 0., 2., ..., 2., 1., 1.])
```

```
accuracy_score( y_test, y_pred )
```

```
0.8189893694480851
```

2. Random Forest

Random forest is an extension of bagged decision trees.

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

```
seed = 7
num_trees = 100
max_features = 3
# Set shuffle=True to enable shuffling and ensure random_state is effective
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean())
```

```
0.9989872369992037
```

```
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_features=3)
```

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

```
array([1., 0., 2., ..., 1., 1., 1.])
```

```
accuracy_score( y_test, y_pred )
```

```
0.8210281054317752
```

3. Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset.

```
from sklearn.ensemble import ExtraTreesClassifier
```

```
seed = 7
num_trees = 100
max_features = 7
# Set shuffle=True to enable shuffling and ensure random_state is effective
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean())
```

```
nan
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1000: UserWarning: Scoring failed. The score on this
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_scorer.py", line 143, in __call__
    score = scorer(estimator, *args, **routed_params.get(name).score)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_scorer.py", line 455, in __call__
    return estimator.score(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 764, in score
    return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 904, in predict
    proba = self.predict_proba(X)
```

```
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 946, in predict_proba
    X = self._validate_X_predict(X)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 641, in _validate_X_predict
    X = self._validate_data(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 633, in _validate_data
    out = check_array(X, input_name="X", **check_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 1064, in check_array
    _assert_all_finite(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 123, in _assert_all_finite
    _assert_all_finite_element_wise(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 172, in _assert_all_finite_element_wise
    raise ValueError(msg_err)
ValueError: Input X contains NaN.
ExtraTreesClassifier does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider skl

  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:540: FitFailedWarning:
9 fits failed out of a total of 10.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
9 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 377, in fit
    estimator._compute_missing_values_in_feature_mask(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py", line 214, in _compute_missing_values_in_feature_mask
    assert_all_finite(X, **common_kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 213, in assert_all_finite
    _assert_all_finite(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 123, in _assert_all_finite
    _assert_all_finite_element_wise(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 172, in _assert_all_finite_element_wise
    raise ValueError(msg_err)
ValueError: Input X contains NaN.
ExtraTreesClassifier does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider skl

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```

```
!pip install scikit-learn
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score

# Assuming X_train, y_train, X_test, y_test are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Transform the test data using the trained imputer
X_test_imputed = imputer.transform(X_test)

# Now, fit the ExtraTreesClassifier model using the imputed data
seed = 7
num_trees = 100
max_features = 7
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
!pip install scikit-learn
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score

# Assuming X_train, y_train, X_test, y_test are already defined...
```

```
# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Transform the test data using the trained imputer
X_test_imputed = imputer.transform(X_test)

# Now, fit the ExtraTreesClassifier model using the imputed data
seed = 7
num_trees = 100
max_features = 7
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)

# Fit the model to the entire training data
model.fit(X_train_imputed, y_train) # This line is added to fit the model
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
    ExtraTreesClassifier
ExtraTreesClassifier(max_features=7)
```

```
accuracy_score( y_test, y_pred )
```

```
0.8210281054317752
```

Boosting Algorithms

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence.

Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.

The two most common boosting ensemble machine learning algorithms are:

1) AdaBoost

2) Stochastic Gradient Boosting

1. AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or or less attention to them in the construction of subsequent models.

```
from sklearn.ensemble import AdaBoostClassifier
```

```
seed = 7
num_trees = 30
# Set shuffle=True to enable shuffling and ensure random_state is effective
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold)
print(results.mean())
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the defau
  warnings.warn(
nan
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1000: UserWarning: Scoring failed. The score on th
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_scorer.py", line 143, in __call__
    score = scorer(estimator, *args, **routed_params.get(name).score)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_scorer.py", line 455, in __call__
    return estimator.score(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 764, in score
    return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py", line 727, in predict
    pred = self.decision_function(X)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py", line 788, in decision_function
    X = self._check_X(X)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py", line 98, in _check_X
    return self._validate_data(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 633, in _validate_data
```

```
    out = check_array(X, input_name="X", **check_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 1064, in check_array
    _assert_all_finite(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 123, in _assert_all_finite
    _assert_all_finite_element_wise(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 172, in _assert_all_finite_element_wise
    raise ValueError(msg_err)
ValueError: Input X contains NaN.
AdaBoostClassifier does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider sk

  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:540: FitFailedWarning:
9 fits failed out of a total of 10.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
9 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py", line 133, in fit
    X, y = self._validate_data(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 650, in _validate_data
    X, y = check_X_y(X, y, **check_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 1301, in check_X_y
    X = check_array(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 1064, in check_array
    _assert_all_finite(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 123, in _assert_all_finite
    _assert_all_finite_element_wise(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py", line 172, in _assert_all_finite_element_wise
    raise ValueError(msg_err)
ValueError: Input X contains NaN.
AdaBoostClassifier does not accept missing values encoded as NaN natively. For supervised learning, you might want to consider sk
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.ensemble import AdaBoostClassifier
from sklearn import model_selection

# Assuming X_train, y_train are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Now, fit the AdaBoostClassifier using the imputed data
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X_train_imputed, y_train, cv=kfold)  # Use X_train_imputed here
print(results.mean())

model.fit(X_train_imputed, y_train)  # Use X_train_imputed here
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
0.9802471713059029
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
```

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=30, random_state=7)
```

```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.ensemble import AdaBoostClassifier
from sklearn import model_selection

# Assuming X_train, y_train, X_test are already defined...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)

# Transform the test data using the trained imputer
X_test_imputed = imputer.transform(X_test)

# Now, fit the AdaBoostClassifier using the imputed data
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X_train_imputed, y_train, cv=kfold)  # Use X_train_imputed here
print(results.mean())

model.fit(X_train_imputed, y_train)  # Use X_train_imputed here

# Now, use the imputed X_test data for prediction
y_pred = model.predict(X_test_imputed)  # Use X_test_imputed here
y_pred
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
0.9802471713059029
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default
  warnings.warn(
array([1., 0., 2., ..., 1., 1., 1.])
```

```
accuracy_score( y_test, y_pred )
```

```
0.8430173292558614
```

2. Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps of the the best techniques available for improving performance via ensembles.

```
from sklearn.ensemble import GradientBoostingClassifier


# Original code:
# kfold = model_selection.KFold(n_splits=10, random_state=seed)

# Option 1: Enable shuffling
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)

# Option 2: Remove random_state
# kfold = model_selection.KFold(n_splits=10)


!pip install scikit-learn
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.ensemble import GradientBoostingClassifier # Or AdaBoostClassifier

# ... (your existing code) ...

# Create a SimpleImputer to replace NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')  # You can use other strategies like 'median' or 'most_frequent'

# Fit the imputer on the training data and transform it
X_train_imputed = imputer.fit_transform(X_train)
```