```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install mlxtend
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.8/dist-packages (0.
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.8/dist-packag
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/loca
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (f
```

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

import pickle
from os import path

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder


from sklearn import preprocessing
import xgboost as xgb
from sklearn import svm
from sklearn.metrics import classification_report,accuracy_score,roc_auc_score,average_pre
import sklearn
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from yellowbrick.classifier import ClassificationReport
from sklearn.ensemble import AdaBoostClassifier,RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
import six
import sys
sys.modules['sklearn.externals.six'] = six
```

```
from mlxtend.classifier import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import joblib

sys.modules['sklearn.externals.joblib'] = joblib
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.externals import joblib
import warnings
warnings.filterwarnings("ignore")

data = pd.read_csv('/content/drive/My Drive/NIDS_ML/UNSW_NB15_training-set.csv')
data.head()
```

sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_
1	2	0	0	0	1	
1	2	0	0	0	1	
1	3	0	0	0	1	
1	3	0	0	0	2	
1	3	0	0	0	2	

```
data.shape
```

```
(82332, 45)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82332 entries, 0 to 82331
Data columns (total 45 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 82332 non-null  int64
 1   dur                82332 non-null  float64
 2   proto              82332 non-null  object
 3   service            82332 non-null  object
 4   state              82332 non-null  object
 5   spkts              82332 non-null  int64
 6   dpkts              82332 non-null  int64
 7   sbytes             82332 non-null  int64
 8   dbytes             82332 non-null  int64
 9   rate               82332 non-null  float64
 10  sttl               82332 non-null  int64
 11  dttl               82332 non-null  int64
```

```
 12  sload              82332 non-null  float64
 13  dload              82332 non-null  float64
 14  sloss              82332 non-null  int64
 15  dloss              82332 non-null  int64
 16  sinpkt             82332 non-null  float64
 17  dinpkt             82332 non-null  float64
 18  sjit               82332 non-null  float64
 19  djit               82332 non-null  float64
 20  swin               82332 non-null  int64
 21  stcpb              82332 non-null  int64
 22  dtcpb              82332 non-null  int64
 23  dwin               82332 non-null  int64
 24  tcprtt             82332 non-null  float64
 25  synack             82332 non-null  float64
 26  ackdat             82332 non-null  float64
 27  smean              82332 non-null  int64
 28  dmean              82332 non-null  int64
 29  trans_depth        82332 non-null  int64
 30  response_body_len  82332 non-null  int64
 31  ct_srv_src         82332 non-null  int64
 32  ct_state_ttl       82332 non-null  int64
 33  ct_dst_ltm         82332 non-null  int64
 34  ct_src_dport_ltm   82332 non-null  int64
 35  ct_dst_sport_ltm   82332 non-null  int64
 36  ct_dst_src_ltm     82332 non-null  int64
 37  is_ftp_login       82332 non-null  int64
 38  ct_ftp_cmd         82332 non-null  int64
 39  ct_flw_http_mthd   82332 non-null  int64
 40  ct_src_ltm         82332 non-null  int64
 41  ct_srv_dst         82332 non-null  int64
 42  is_sm_ips_ports    82332 non-null  int64
 43  attack_cat         82332 non-null  object
 44  label              82332 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 28.3+ MB
```

```
data.drop('service',axis='columns',inplace=True)
```

```
data.isnull().sum()
```

```
id                   0
dur                  0
proto                0
state                0
spkts                0
dpkts                0
sbytes               0
dbytes               0
rate                 0
sttl                 0
dttl                 0
sload                0
dload                0
sloss                0
dloss                0
sinpkt               0
dinpkt               0
sjit                 0
```

```
djit                0
swin                0
stcpb               0
dtcpb               0
dwin                0
tcprtt              0
synack              0
ackdat              0
smean               0
dmean               0
trans_depth         0
response_body_len   0
ct_srv_src          0
ct_state_ttl        0
ct_dst_ltm          0
ct_src_dport_ltm    0
ct_dst_sport_ltm    0
ct_dst_src_ltm      0
is_ftp_login        0
ct_ftp_cmd          0
ct_flw_http_mthd    0
ct_src_ltm          0
ct_srv_dst          0
is_sm_ips_ports     0
attack_cat          0
label               0
dtype: int64
```

```
data['attack_cat'].value_counts()
```

```
Normal            37000
Generic           18871
Exploits          11132
Fuzzers            6062
DoS                4089
Reconnaissance     3496
Analysis            677
Backdoor            583
Shellcode           378
Worms                44
Name: attack_cat, dtype: int64
```

```
data['state'].value_counts()
```

```
FIN    39339
INT    34163
CON     6982
REQ     1842
ACC        4
RST        1
CLO        1
Name: state, dtype: int64
```

```
data
```

	id	dur	proto	state	spkts	dpkts	sbytes	dbytes	rate	st
0	1	0.000011	udp	INT	2	0	496	0	90909.090200	2
1	2	0.000008	udp	INT	2	0	1762	0	125000.000300	2
2	3	0.000005	udp	INT	2	0	1068	0	200000.005100	2
3	4	0.000006	udp	INT	2	0	900	0	166666.660800	2
4	5	0.000010	udp	INT	2	0	2126	0	100000.002500	2
...	...	...	...	...	...	...	...	...	...	
82327	82328	0.000005	udp	INT	2	0	104	0	200000.005100	2
82328	82329	1.106101	tcp	FIN	20	8	18062	354	24.410067	2
82329	82330	0.000000	arp	INT	1	0	46	0	0.000000	
82330	82331	0.000000	arp	INT	1	0	46	0	0.000000	
82331	82332	0.000009	udp	INT	2	0	104	0	111111.107200	2

82332 rows × 44 columns

```
features = pd.read_csv('/content/drive/MyDrive/NIDS_ML/NUSW-NB15_features.csv',encoding='c
```

```
features.head()
```

	No.	Name	Type	Description
0	1	srcip	nominal	Source IP address
1	2	sport	integer	Source port number
2	3	dstip	nominal	Destination IP address
3	4	dsport	integer	Destination port number
4	5	proto	nominal	Transaction protocol

```
features['Type '] = features['Type '].str.lower()
```

```
# selecting column names of all data types
nominal_names = features['Name'][features['Type ']=='nominal']
integer_names = features['Name'][features['Type ']=='integer']
binary_names = features['Name'][features['Type ']=='binary']
float_names = features['Name'][features['Type ']=='float']
```

```
cols = data.columns
nominal_names = cols.intersection(nominal_names)
integer_names = cols.intersection(integer_names)
binary_names = cols.intersection(binary_names)
float_names = cols.intersection(float_names)
```

```
for c in integer_names:
  pd.to_numeric(data[c])


for c in binary_names:
  pd.to_numeric(data[c])


for c in float_names:
  pd.to_numeric(data[c])


data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82332 entries, 0 to 82331
Data columns (total 44 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 82332 non-null  int64
 1   dur                82332 non-null  float64
 2   proto              82332 non-null  object
 3   state              82332 non-null  object
 4   spkts              82332 non-null  int64
 5   dpkts              82332 non-null  int64
 6   sbytes             82332 non-null  int64
 7   dbytes             82332 non-null  int64
 8   rate               82332 non-null  float64
 9   sttl               82332 non-null  int64
 10  dttl               82332 non-null  int64
 11  sload              82332 non-null  float64
 12  dload              82332 non-null  float64
 13  sloss              82332 non-null  int64
 14  dloss              82332 non-null  int64
 15  sinpkt             82332 non-null  float64
 16  dinpkt             82332 non-null  float64
 17  sjit               82332 non-null  float64
 18  djit               82332 non-null  float64
 19  swin               82332 non-null  int64
 20  stcpb              82332 non-null  int64
 21  dtcpb              82332 non-null  int64
 22  dwin               82332 non-null  int64
 23  tcprtt             82332 non-null  float64
 24  synack             82332 non-null  float64
 25  ackdat             82332 non-null  float64
 26  smean              82332 non-null  int64
 27  dmean              82332 non-null  int64
 28  trans_depth        82332 non-null  int64
 29  response_body_len  82332 non-null  int64
 30  ct_srv_src         82332 non-null  int64
 31  ct_state_ttl       82332 non-null  int64
 32  ct_dst_ltm         82332 non-null  int64
 33  ct_src_dport_ltm   82332 non-null  int64
 34  ct_dst_sport_ltm   82332 non-null  int64
 35  ct_dst_src_ltm     82332 non-null  int64
 36  is_ftp_login       82332 non-null  int64
 37  ct_ftp_cmd         82332 non-null  int64
```

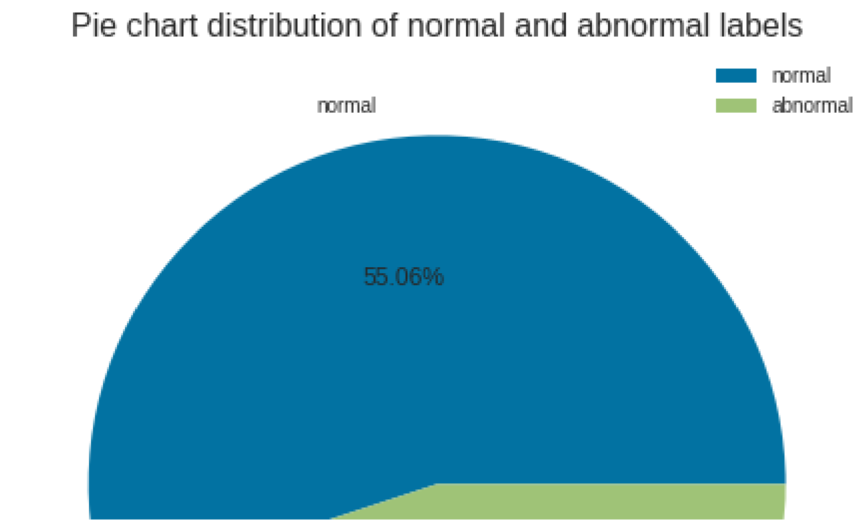
```
 38  ct_flw_http_mthd   82332 non-null  int64
 39  ct_src_ltm         82332 non-null  int64
 40  ct_srv_dst         82332 non-null  int64
 41  is_sm_ips_ports    82332 non-null  int64
 42  attack_cat         82332 non-null  object
 43  label              82332 non-null  int64
dtypes: float64(11), int64(30), object(3)
memory usage: 27.6+ MB
```

```
data
```

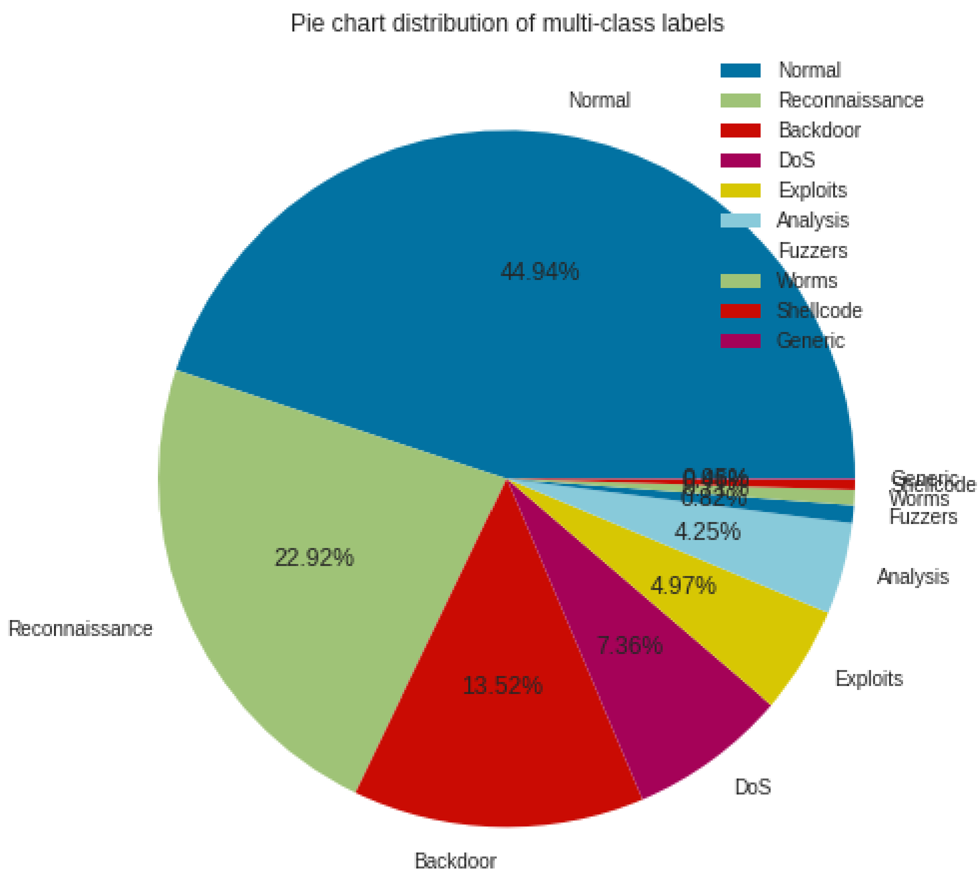
	id	dur	proto	state	spkts	dpkts	sbytes	dbytes	rate	st
0	1	0.000011	udp	INT	2	0	496	0	90909.090200	2
1	2	0.000008	udp	INT	2	0	1762	0	125000.000300	2
2	3	0.000005	udp	INT	2	0	1068	0	200000.005100	2
3	4	0.000006	udp	INT	2	0	900	0	166666.660800	2
4	5	0.000010	udp	INT	2	0	2126	0	100000.002500	2
...	...	...	...	...	...	...	...	...	...	
82327	82328	0.000005	udp	INT	2	0	104	0	200000.005100	2
82328	82329	1.106101	tcp	FIN	20	8	18062	354	24.410067	2
82329	82330	0.000000	arp	INT	1	0	46	0	0.000000	
82330	82331	0.000000	arp	INT	1	0	46	0	0.000000	
82331	82332	0.000009	udp	INT	2	0	104	0	111111.107200	2

82332 rows × 44 columns

```
plt.figure(figsize=(8,8))
plt.pie(data.label.value_counts(),labels=['normal','abnormal'],autopct='%0.2f%%')
plt.title("Pie chart distribution of normal and abnormal labels",fontsize=16)
plt.legend()
plt.show()
```

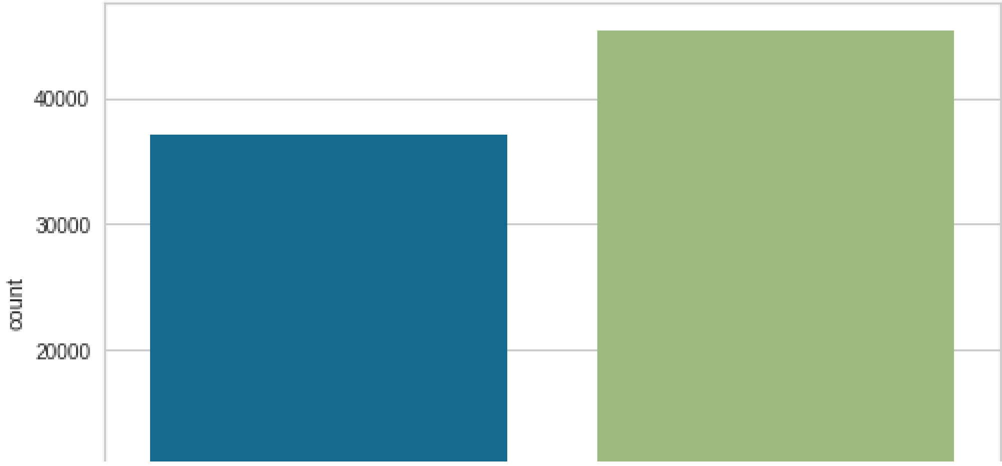



```
plt.figure(figsize=(8,8))
plt.pie(data.attack_cat.value_counts(),labels=data.attack_cat.unique(),autopct='%0.2f%%')
plt.title('Pie chart distribution of multi-class labels')
plt.legend(loc='best')
plt.show()
```



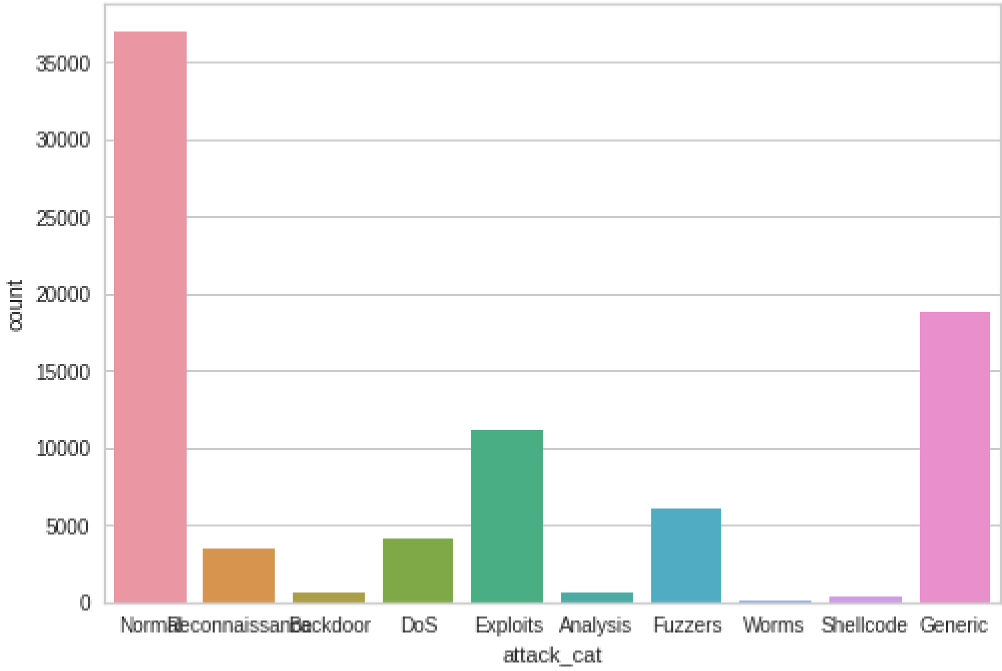

```
sns.countplot(data['label'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b2e6e99a0>
```




```
sns.countplot(data['attack_cat'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b2e7494f0>
```




```
num_col = data.select_dtypes(include='number').columns

# selecting categorical data attributes
cat_col = data.columns.difference(num_col)
cat_col = cat_col[1:]
cat_col
```

```
Index(['proto', 'state'], dtype='object')
```

```
data_cat = data[cat_col].copy()
data_cat.head()
```

	proto	state
0	udp	INT
1	udp	INT

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
data_cat['proto'] = le.fit_transform(data_cat['proto'])
data_cat['state'] = le.fit_transform(data_cat['state'])
```

```
data_cat
```

	proto	state
0	117	4
1	117	4
2	117	4
3	117	4
4	117	4
...	...	...
82327	117	4
82328	111	3
82329	6	4
82330	6	4
82331	117	4

82332 rows × 2 columns

```
data.drop(columns=cat_col,inplace=True)
```

```
data = pd.concat([data, data_cat],axis=1)
```

```
# selecting numeric attributes columns from data
num_col = list(data.select_dtypes(include='number').columns)
num_col.remove('id')
num_col.remove('label')
print(num_col)
```

```
, 'dbytes', 'rate', 'sttl', 'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'di
```

```
# using minmax scaler for normalizing data
minmax_scale = MinMaxScaler(feature_range=(0, 1))
def normalization(df,col):
```

```
    for i in col:
      arr = df[i]
      arr = np.array(arr)
      df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))
    return df
```

```
data.head()
```

	id	dur	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload
0	1	0.000011	2	0	496	0	90909.0902	254	0	180363632.0
1	2	0.000008	2	0	1762	0	125000.0003	254	0	881000000.0
2	3	0.000005	2	0	1068	0	200000.0051	254	0	854400000.0
3	4	0.000006	2	0	900	0	166666.6608	254	0	600000000.0
4	5	0.000010	2	0	2126	0	100000.0025	254	0	850400000.0

5 rows × 44 columns

```
data = normalization(data.copy(),num_col)
```

```
data.head()
```

load	...	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst	is_sm_i
4238	...	0.0	0.0	0.0	0.000000	0.016393	
7236	...	0.0	0.0	0.0	0.000000	0.016393	
2187	...	0.0	0.0	0.0	0.000000	0.032787	
3895	...	0.0	0.0	0.0	0.016949	0.032787	
1427	...	0.0	0.0	0.0	0.016949	0.032787	

Binary Labels

```
bin_label = pd.DataFrame(data.label.map(lambda x:'normal' if x==0 else 'abnormal'))

# creating a dataframe with binary labels (normal,abnormal)
bin_data = data.copy()
bin_data['label'] = bin_label
```

```
# label encoding (0,1) binary labels
le1 = preprocessing.LabelEncoder()
enc_label = bin_label.apply(le1.fit_transform)
bin_data['label'] = enc_label
```

```
le1.classes_
```

```
array(['abnormal', 'normal'], dtype=object)
```

```
np.save("le1_classes.npy",le1.classes_,allow_pickle=True)
```

Multi-class Labels

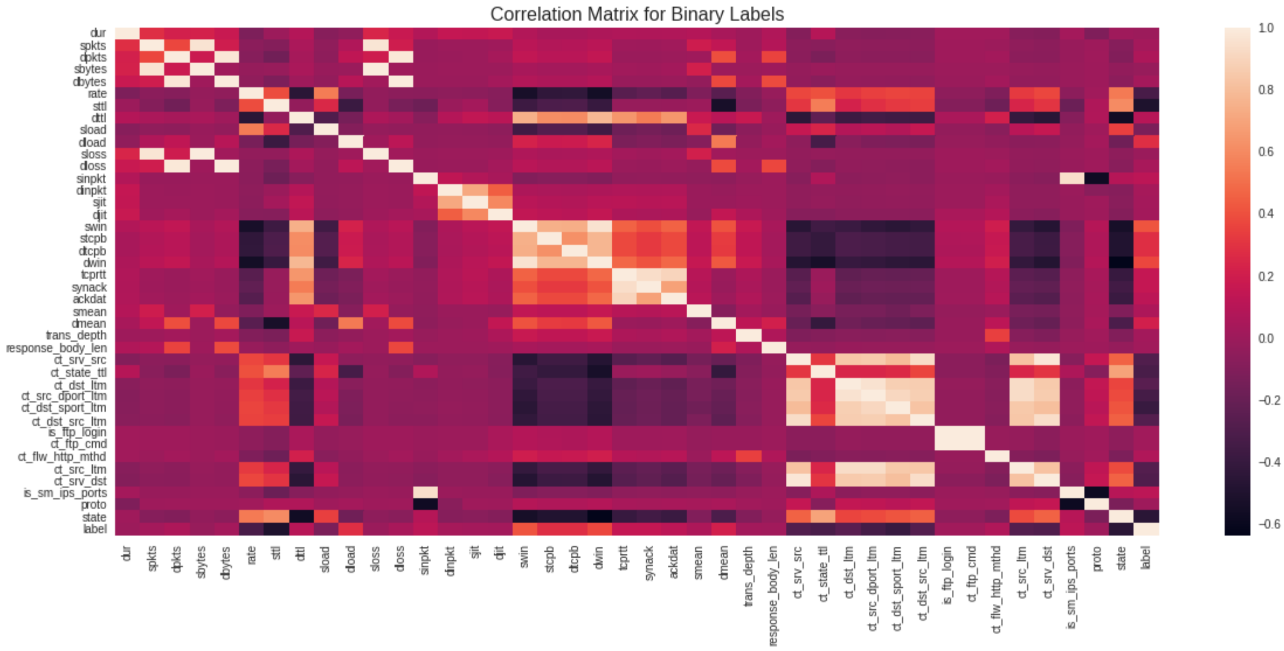
```
multi_data = data.copy()
multi_label = pd.DataFrame(multi_data.attack_cat)
multi_data = pd.get_dummies(multi_data,columns=['attack_cat'])
le2 = preprocessing.LabelEncoder()
enc_label = multi_label.apply(le2.fit_transform)
multi_data['label'] = enc_label
le2.classes_
```

```
array(['Analysis', 'Backdoor', 'DoS', 'Exploits', 'Fuzzers', 'Generic',
       'Normal', 'Reconnaissance', 'Shellcode', 'Worms'], dtype=object)
```

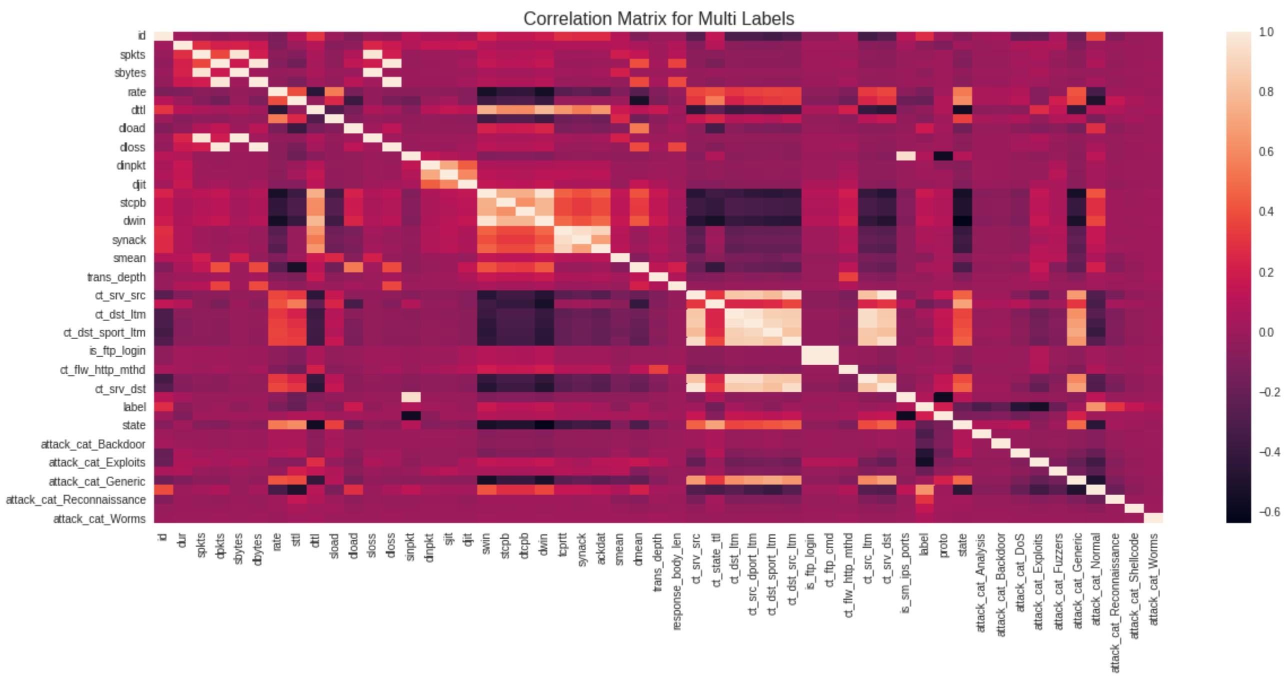
```
np.save("le2_classes.npy",le2.classes_,allow_pickle=True)
```

```
num_col.append('label')
```

```
# Correlation Matrix for Binary Labels
plt.figure(figsize=(20,8))
corr_bin = bin_data[num_col].corr()
sns.heatmap(corr_bin,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Binary Labels',fontsize=16)
plt.show()
```




```
num_col = list(multi_data.select_dtypes(include='number').columns)
# Correlation Matrix for Multi-class Labels
plt.figure(figsize=(20,8))
corr_multi = multi_data[num_col].corr()
sns.heatmap(corr_multi,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Multi Labels',fontsize=16)
plt.show()
```



Feature Selection

Binary Labels

```
corr_ybin = abs(corr_bin['label'])
highest_corr_bin = corr_ybin[corr_ybin >0.3]
highest_corr_bin.sort_values(ascending=True)
```

```
ct_state_ttl        0.318517
rate                0.328629
ct_src_dport_ltm    0.341513
dwin                0.369257
ct_dst_sport_ltm    0.393668
swin                0.414504
state               0.459040
sttl                0.504159
label               1.000000
Name: label, dtype: float64
```

```
bin_cols = highest_corr_bin.index
bin_cols
```

```
Index(['rate', 'sttl', 'swin', 'dwin', 'ct_state_ttl', 'ct_src_dport_ltm',
       'ct_dst_sport_ltm', 'state', 'label'],
      dtype='object')
```

```
bin_data = bin_data[bin_cols].copy()
bin_data
```

	rate	sttl	swin	dwin	ct_state_ttl	ct_src_dport_ltm	ct_dst_sport_l
0	0.090909	0.996078	0.0	0.0	0.333333	0.000000	
1	0.125000	0.996078	0.0	0.0	0.333333	0.000000	
2	0.200000	0.996078	0.0	0.0	0.333333	0.000000	
3	0.166667	0.996078	0.0	0.0	0.333333	0.017241	
4	0.100000	0.996078	0.0	0.0	0.333333	0.017241	
...	...	...	...	...	...	...	
82327	0.200000	0.996078	0.0	0.0	0.333333	0.000000	
82328	0.000024	0.996078	1.0	1.0	0.166667	0.000000	
82329	0.000000	0.000000	0.0	0.0	0.333333	0.000000	
82330	0.000000	0.000000	0.0	0.0	0.333333	0.000000	
82331	0.111111	0.996078	0.0	0.0	0.333333	0.000000	

82332 rows × 9 columns

```
bin_data.to_csv('./bin_data.csv')
```

Multi-class Labels

```
# finding the attributes which have more than 0.3 correlation with encoded attack label at
corr_ymulti = abs(corr_multi['label'])
highest_corr_multi = corr_ymulti[corr_ymulti >0.2]
highest_corr_multi.sort_values(ascending=True)
```

```
state                         0.214254
attack_cat_Backdoor           0.235245
id                            0.274428
attack_cat_Reconnaissance     0.296008
attack_cat_Analysis           0.317254
attack_cat_DoS                0.477123
attack_cat_Exploits           0.549046
attack_cat_Normal             0.638825
label                         1.000000
Name: label, dtype: float64
```

```
# selecting attributes found by using pearson correlation coefficient
multi_cols = highest_corr_multi.index
multi_cols
```

```
Index(['id', 'label', 'state', 'attack_cat_Analysis', 'attack_cat_Backdoor',
       'attack_cat_DoS', 'attack_cat_Exploits', 'attack_cat_Normal',
       'attack_cat_Reconnaissance'],
      dtype='object')
```

```
multi_data = multi_data[multi_cols].copy()
```

```
multi_data.to_csv('./multi_data.csv')
```

BINARY CLASSIFICATION

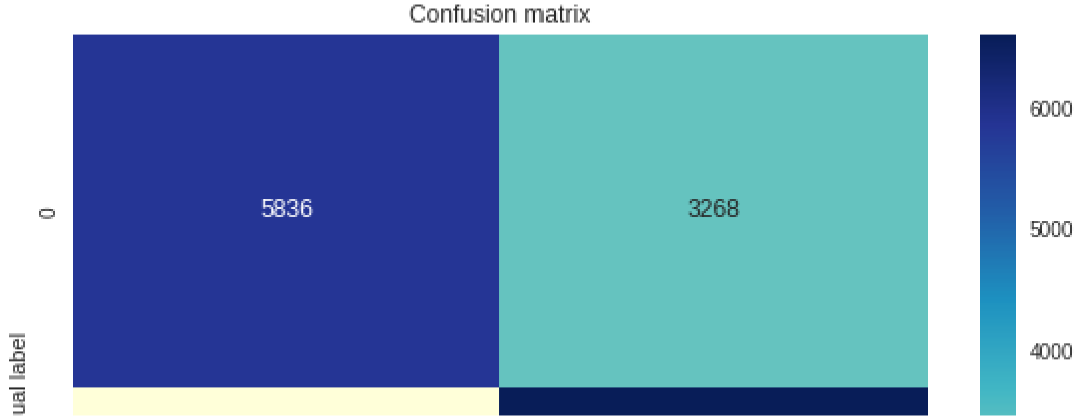
```
X = bin_data.drop(columns=['label'],axis=1)
Y = bin_data['label']
```

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.20, random_state=50)
```

```
#GaussianNB
def GNB(X_train,y_train,X_test,y_test):
    gnb_clf = GaussianNB()
    pred = gnb_clf.fit(X_train, y_train).predict(X_test)
    pred= gnb_clf.predict(X_test)
    print ("GaussianNB:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
```

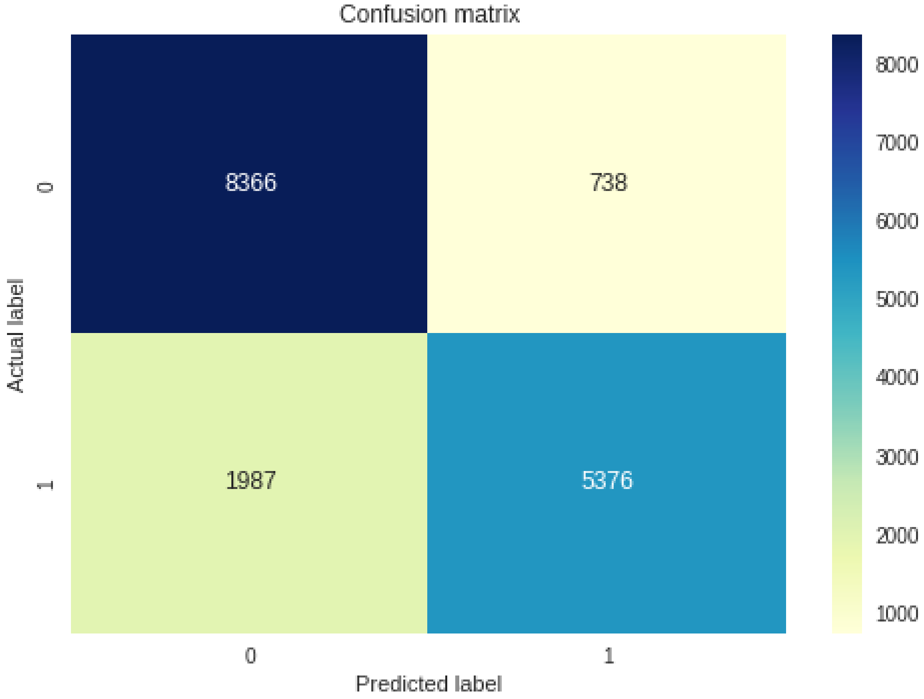
```
        plt.tight_layout()
        plt.title('Confusion matrix', y=1.1)
        plt.ylabel('Actual label')
        plt.xlabel('Predicted label')
        plt.show()

        #Classification Report
        prediction=gnb_clf.predict(X_test)
        print(classification_report(y_test, prediction))
        visualizer = ClassificationReport(gnb_clf, support=True)
        visualizer.fit(X_train, y_train)
        visualizer.score(X_test, y_test)
        g = visualizer.poof()
    GNB(X_train,y_train,X_test,y_test)
```

```
GaussianNB:Accuracy :  75.55110220440882
```




```
#KNeighbours
def KNN1(X_train,y_train,X_test,y_test):
    knn = KNeighborsClassifier(n_neighbors=2)
    knn.fit(X_train, y_train)
    pred = knn.predict(X_test)
    print ("KNN:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=knn.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(knn, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
KNN1(X_train,y_train,X_test,y_test)
```

```
KNN:Accuracy :  83.45175198882615
```

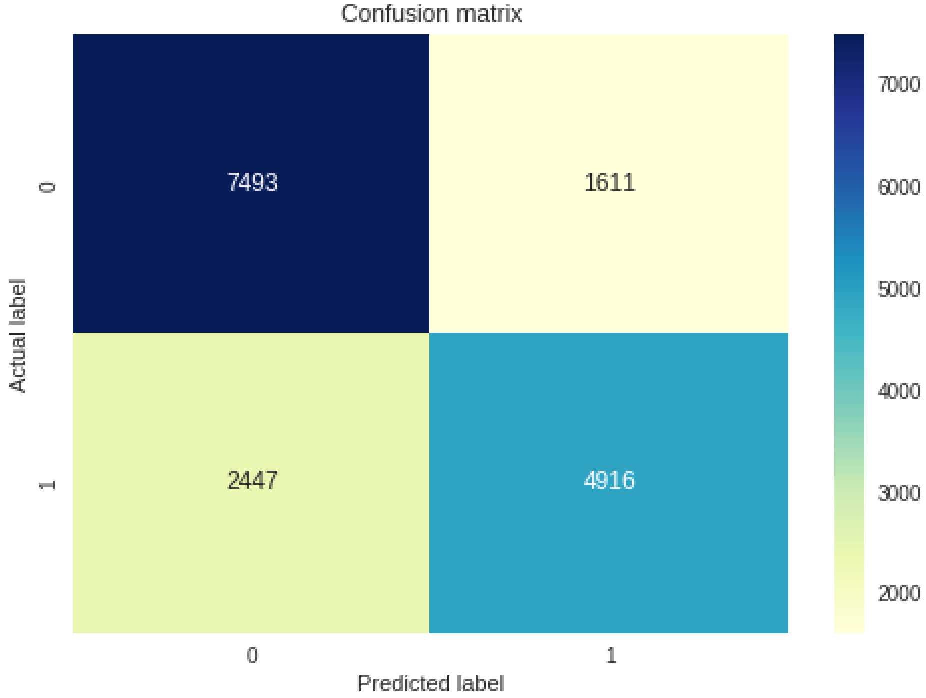



```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86      9104
           1       0.88      0.73      0.80      7363

    accuracy                           0.83     16467
   macro avg       0.84      0.82      0.83     16467
weighted avg       0.84      0.83      0.83     16467
```

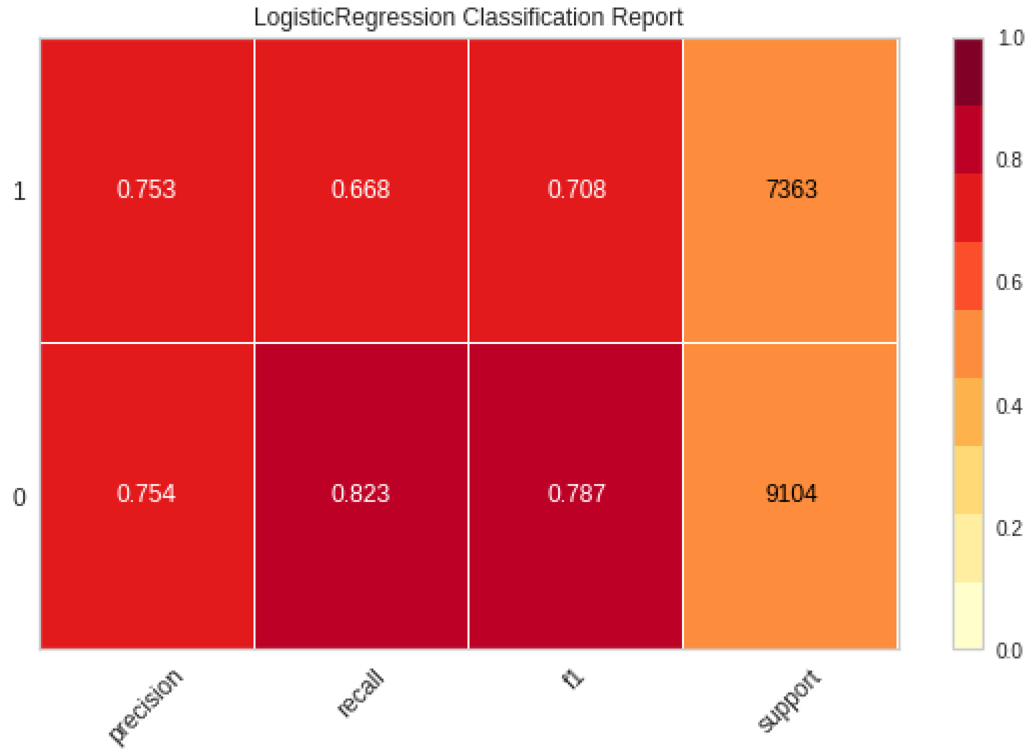

```
#LogisticRegression
from sklearn.linear_model import LogisticRegression
def logisticreg(X_train,y_train,X_test,y_test):
    lr = LogisticRegression()
    lr.fit(X_train,y_train)
    pred = lr.predict(X_test)
    print ("LogisticRegression:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=lr.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(lr, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
```

```
    g = visualizer.poof()
logisticreg(X_train,y_train,X_test,y_test)
```

```
LogisticRegression:Accuracy :  75.35677415436935
```



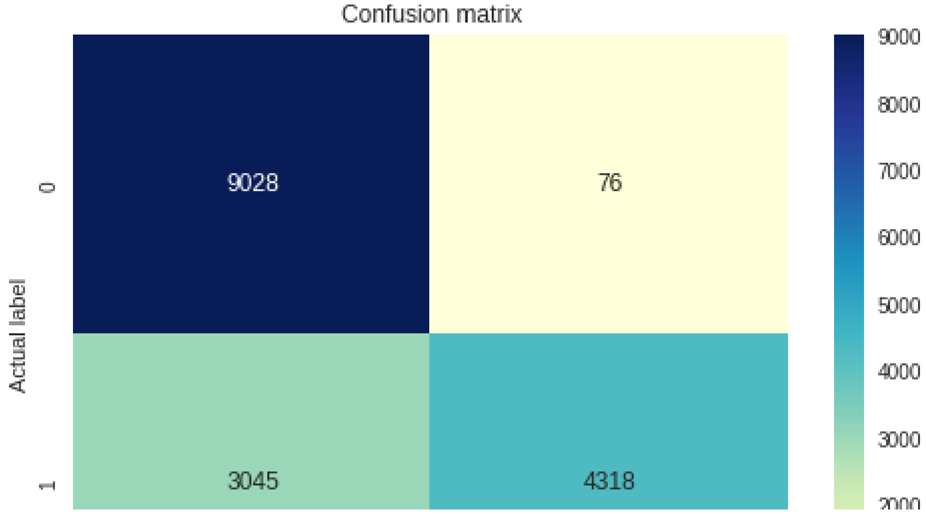

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.79      9104
           1       0.75      0.67      0.71      7363

    accuracy                           0.75     16467
   macro avg       0.75      0.75      0.75     16467
weighted avg       0.75      0.75      0.75     16467
```




```
#Randon Forest
def random_forest(X_train,y_train,X_test,y_test):
    rf = RandomForestClassifier(max_depth=2, min_samples_split=2)
```

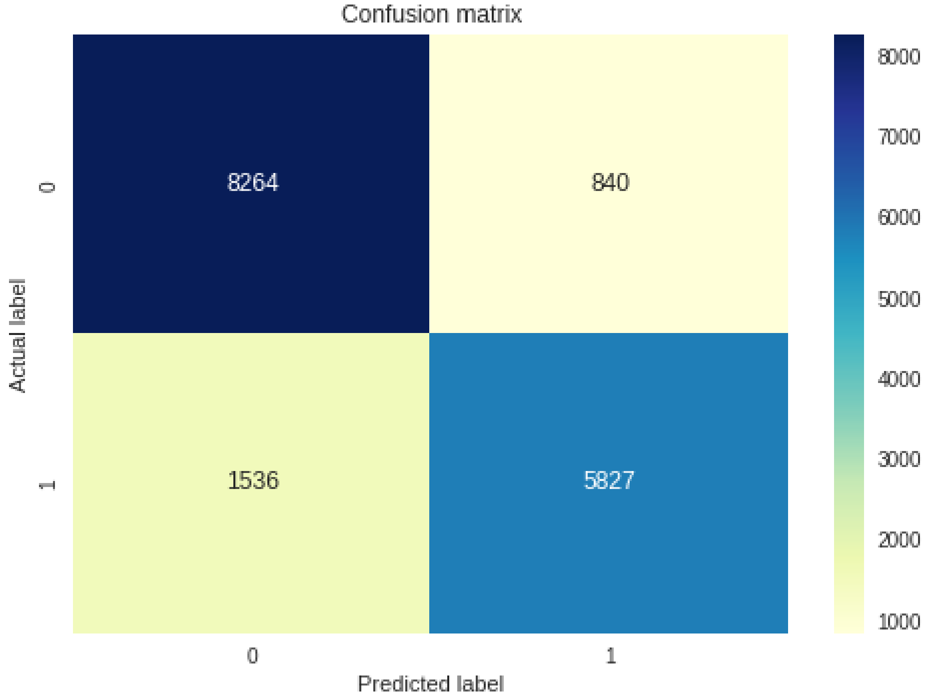
```
    rf.fit(X_train,y_train)
    pred = rf.predict(X_test)
    print ("Random Forest:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=rf.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(rf, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
random_forest(X_train,y_train,X_test,y_test)
```

```
Random Forest:Accuracy :  81.04694236958765
```

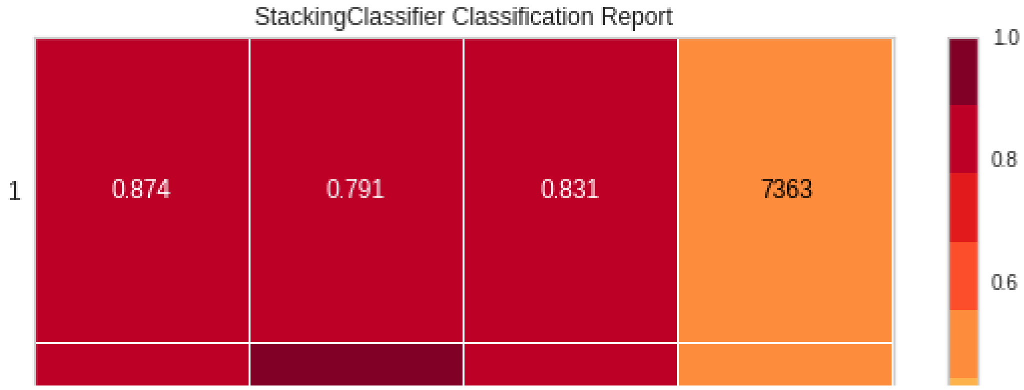



```
#Stacking Classifier
xg = xgb.XGBClassifier(max_depth=5, learning_rate=0.01, n_estimators=100, gamma=0,min_chil
rf = RandomForestClassifier(bootstrap=True,max_depth= 70,max_features= 'auto',min_samples_
knn=KNeighborsClassifier()

def stacking(X_train,y_train,X_test,y_test):
    classifiers=[rf,knn]
    sc = StackingClassifier(classifiers,meta_classifier=xg)
    sc.fit(X_train,y_train)
    pred = sc.predict(X_test)
    print ("Stacking Classifier:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=sc.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(sc, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
stacking(X_train,y_train,X_test,y_test)
```

```
Stacking Classifier:Accuracy :  85.57114228456913
```




```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      9104
           1       0.87      0.79      0.83      7363

    accuracy                           0.86     16467
   macro avg       0.86      0.85      0.85     16467
weighted avg       0.86      0.86      0.85     16467
```



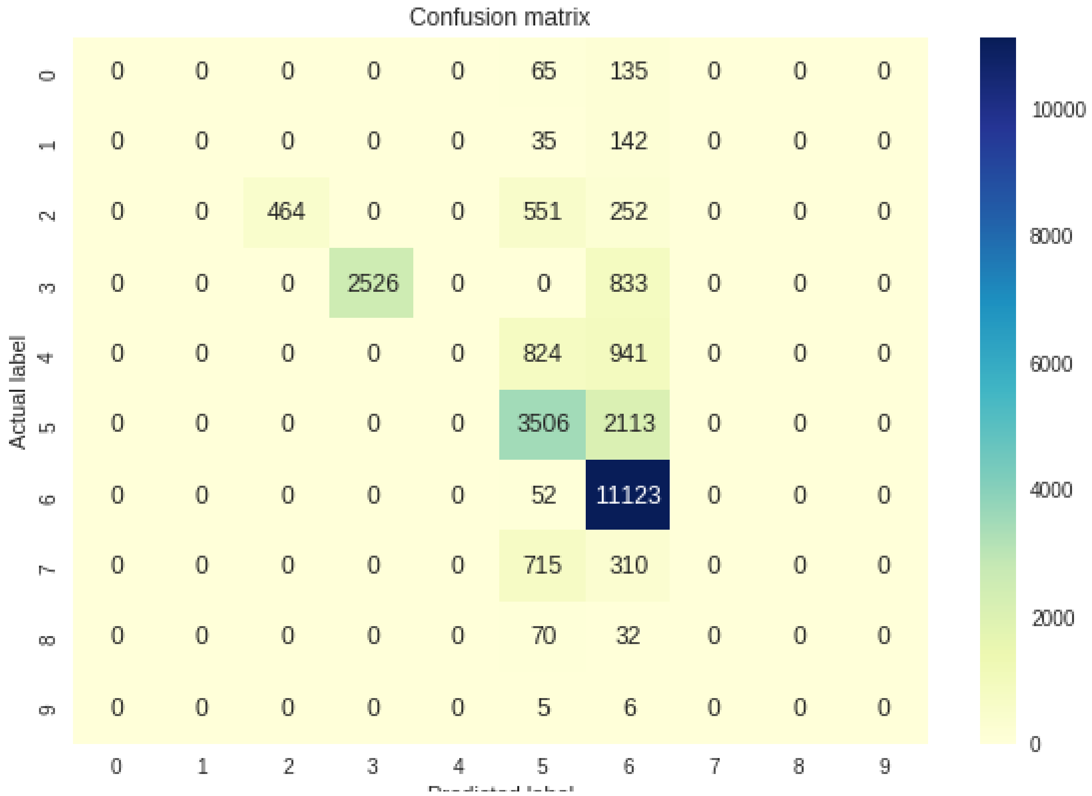
MULTI-CLASS CLASSIFICATION


```
X = multi_data.drop(columns=['label'],axis=1)
Y = multi_data['label']
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.30, random_state=100)
```

```
#GaussianNB
def GNB(X_train,y_train,X_test,y_test):
    gnb_clf = GaussianNB()
    pred = gnb_clf.fit(X_train, y_train).predict(X_test)
    pred= gnb_clf.predict(X_test)

    print ("GaussianNB:Accuracy : ", accuracy_score(y_test,pred)*100)
```

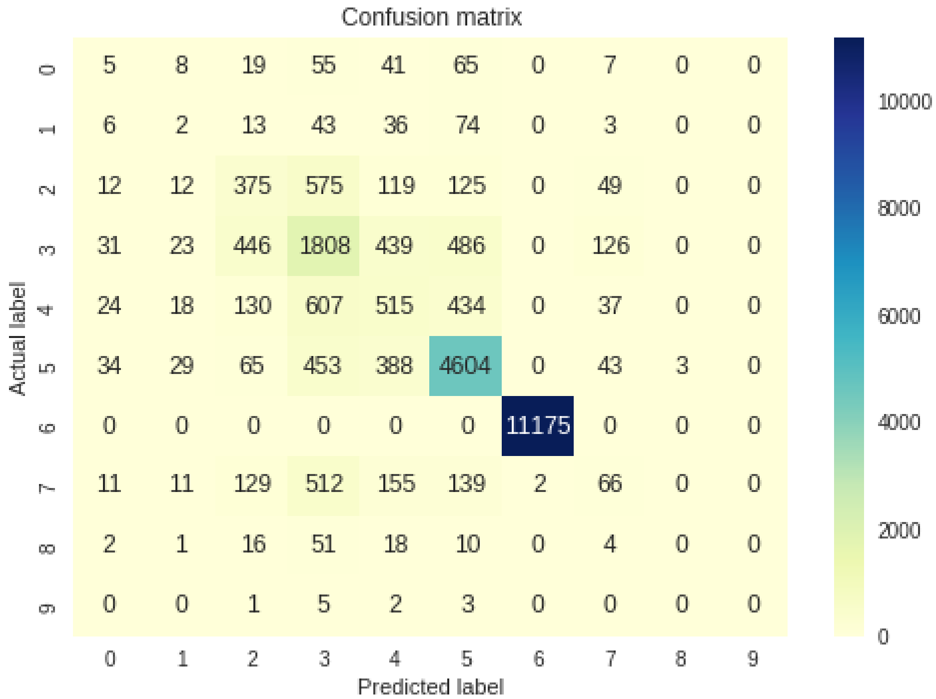
```
    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=gnb_clf.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(gnb_clf, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
GNB(X_train,y_train,X_test,y_test)
```

```
GaussianNB:Accuracy :  71.33198380566802
```

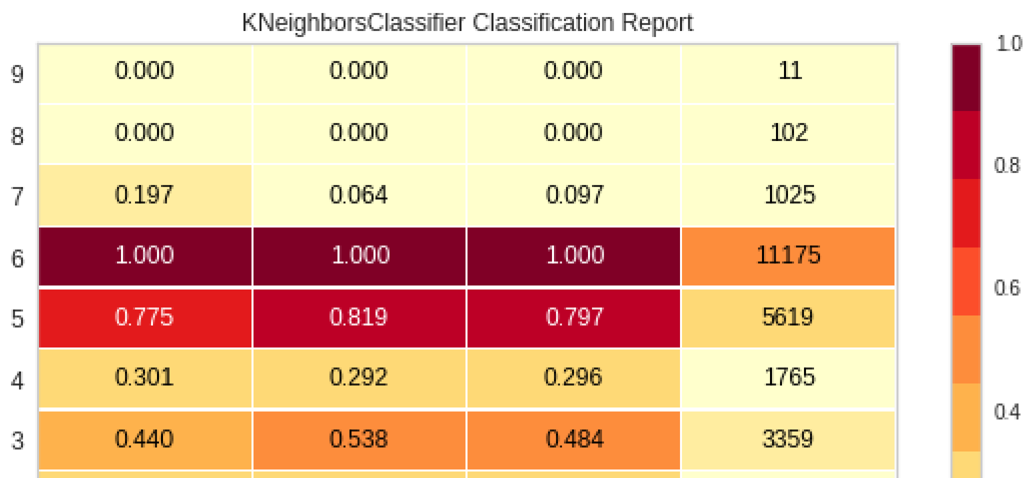



```
#KNeighbours
def KNN1(X_train,y_train,X_test,y_test):
    knn = KNeighborsClassifier()
    knn.fit(X_train, y_train)
    pred = knn.predict(X_test)
    print ("KNN:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=knn.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(knn, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
KNN1(X_train,y_train,X_test,y_test)
```

```
KNN:Accuracy :  75.10121457489879
```



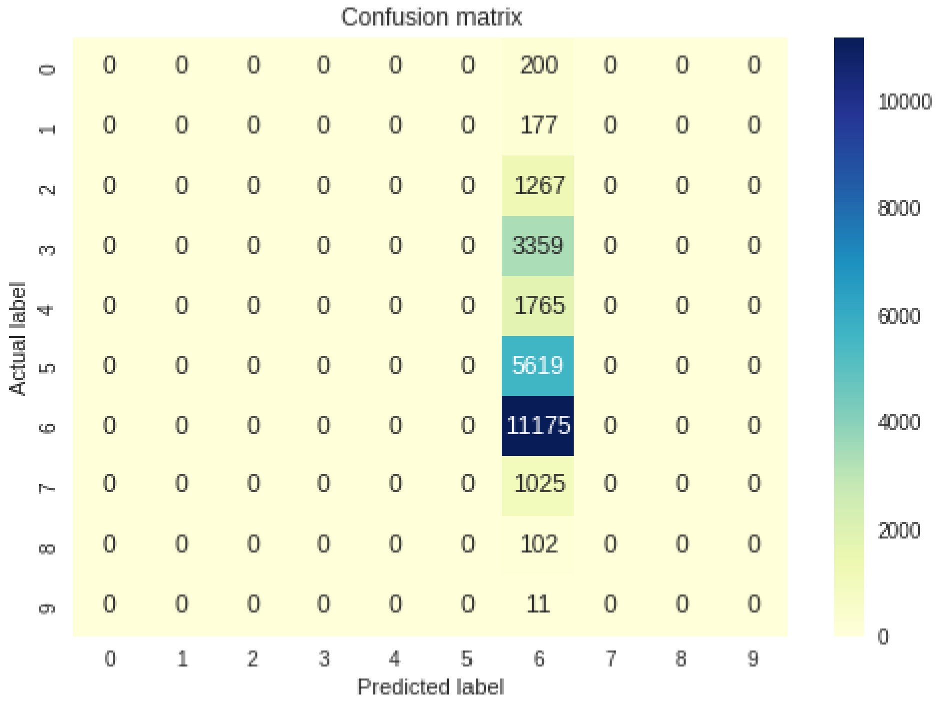

```
              precision    recall  f1-score   support

           0       0.04      0.03      0.03       200
           1       0.02      0.01      0.01       177
           2       0.31      0.30      0.30      1267
           3       0.44      0.54      0.48      3359
           4       0.30      0.29      0.30      1765
           5       0.78      0.82      0.80      5619
           6       1.00      1.00      1.00     11175
           7       0.20      0.06      0.10      1025
           8       0.00      0.00      0.00       102
           9       0.00      0.00      0.00        11

    accuracy                           0.75     24700
   macro avg       0.31      0.30      0.30     24700
weighted avg       0.73      0.75      0.74     24700
```




```
#LogisticRegression
from sklearn.linear_model import LogisticRegression
def logisticreg(X_train,y_train,X_test,y_test):
    lr = LogisticRegression()
    lr.fit(X_train,y_train)
    pred = lr.predict(X_test)
```

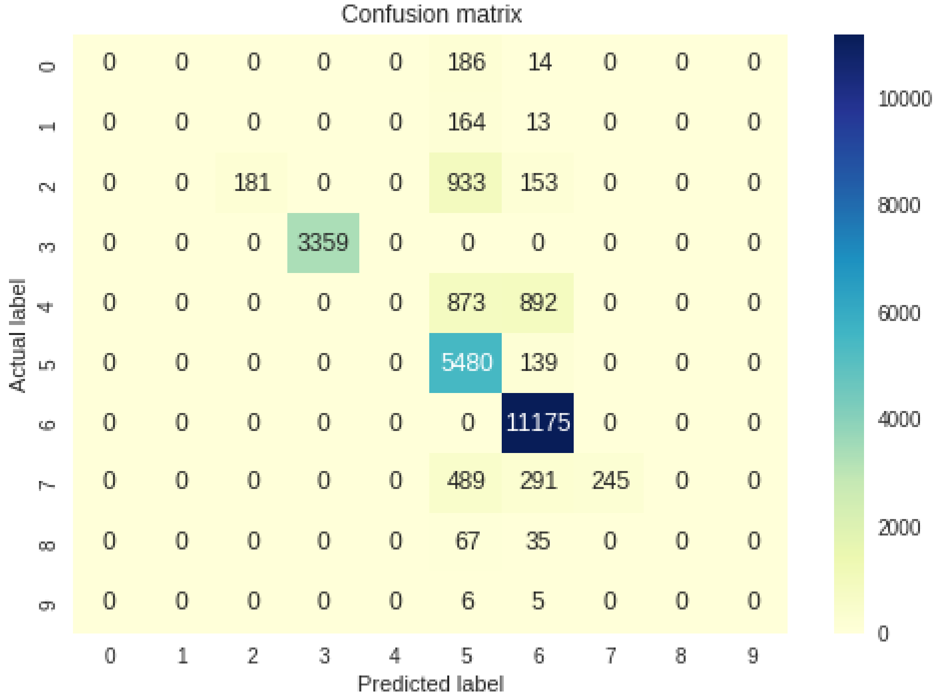
```
    print ("LogisticRegression:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=lr.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(lr, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
logisticreg(X_train,y_train,X_test,y_test)
```

```
LogisticRegression:Accuracy :  45.24291497975709
```

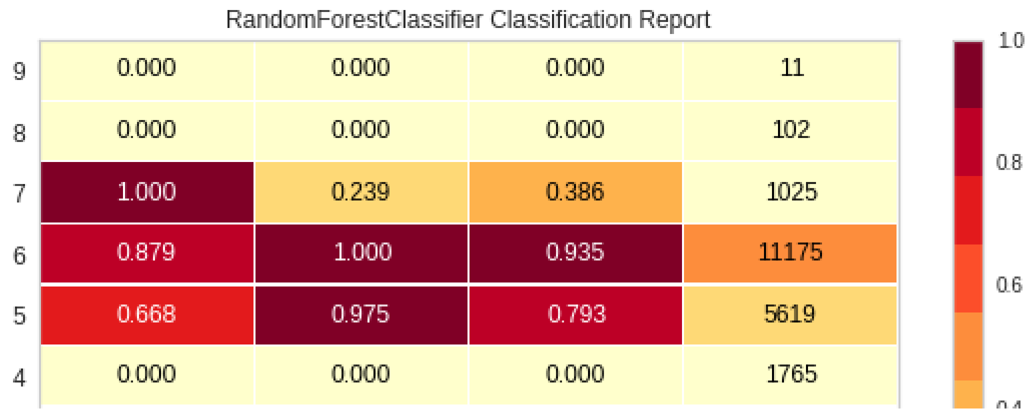



```
#Random Forest
def random_forest(X_train,y_train,X_test,y_test):
    rf = RandomForestClassifier(max_depth=2, min_samples_split=2,)
    rf.fit(X_train,y_train)
    pred = rf.predict(X_test)
    print ("Random Forest:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=rf.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(rf, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
random_forest(X_train,y_train,X_test,y_test)
```

```
Random Forest:Accuracy :  82.75303643724696
```



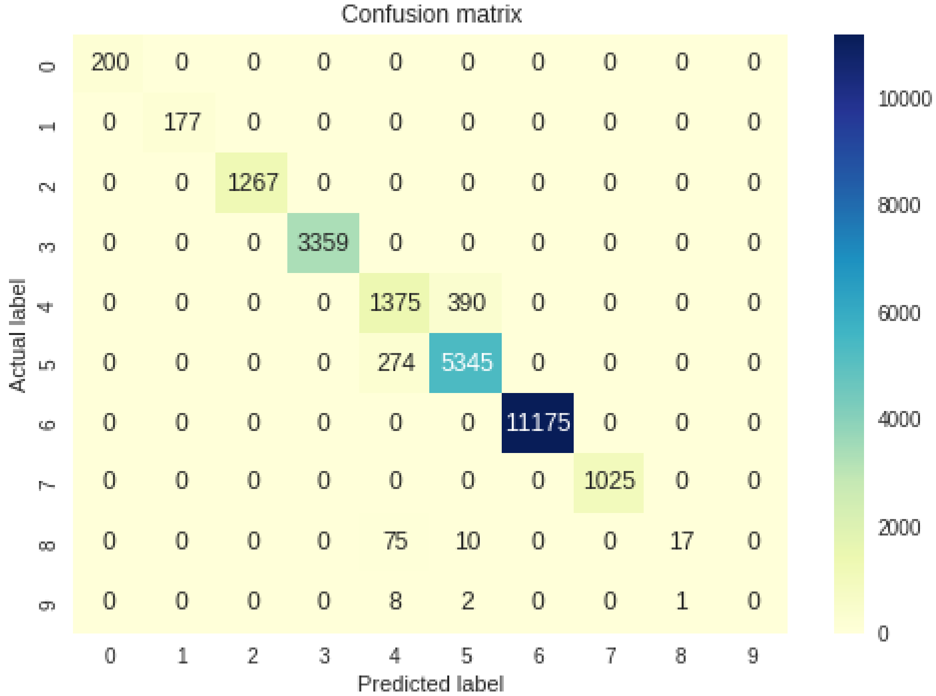

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       200
           1       0.00      0.00      0.00       177
           2       1.00      0.14      0.25      1267
           3       1.00      1.00      1.00      3359
           4       0.00      0.00      0.00      1765
           5       0.67      0.98      0.79      5619
           6       0.88      1.00      0.94     11175
           7       1.00      0.24      0.39      1025
           8       0.00      0.00      0.00       102
           9       0.00      0.00      0.00        11

    accuracy                           0.83     24700
   macro avg       0.45      0.34      0.34     24700
weighted avg       0.78      0.83      0.77     24700
```



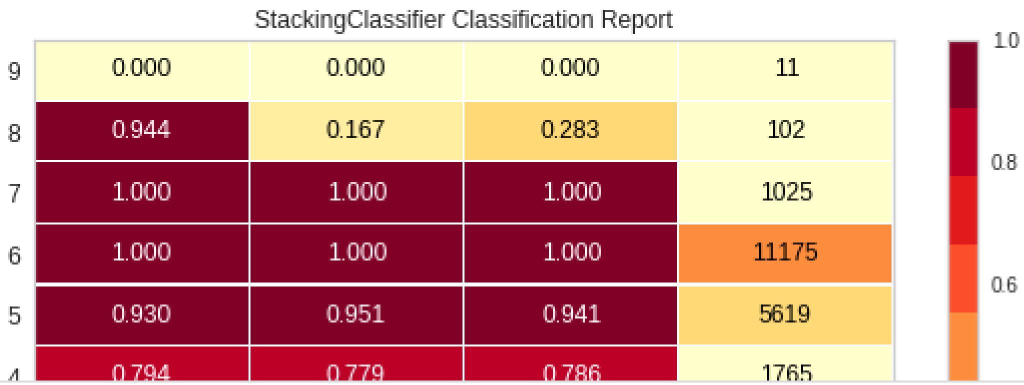

```
#Stacking Classifier
xg = xgb.XGBClassifier(max_depth=5, learning_rate=0.01, n_estimators=100, gamma=0,min_chi]
rf = RandomForestClassifier(bootstrap=True,max_depth= 70,max_features= 'auto',min_samples_
knn=KNeighborsClassifier()

def stacking(X_train,y_train,X_test,y_test):
    classifiers=[rf,knn]
    sc = StackingClassifier(classifiers,meta_classifier=xg)
```

```
    sc.fit(X_train,y_train)
    pred = sc.predict(X_test)
    print ("Stacking Classifier:Accuracy : ", accuracy_score(y_test,pred)*100)

    #confusion Matrix
    matrix =confusion_matrix(y_test, pred)
    sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu" ,fmt='g')
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()

    #Classification Report
    prediction=sc.predict(X_test)
    print(classification_report(y_test, prediction))
    visualizer = ClassificationReport(sc, support=True)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    g = visualizer.poof()
stacking(X_train,y_train,X_test,y_test)
```

```
Stacking Classifier:Accuracy :  96.92307692307692
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       200
           1       1.00      1.00      1.00       177
           2       1.00      1.00      1.00      1267
           3       1.00      1.00      1.00      3359
           4       0.79      0.78      0.79      1765
           5       0.93      0.95      0.94      5619
           6       1.00      1.00      1.00     11175
           7       1.00      1.00      1.00      1025
           8       0.94      0.17      0.28       102
           9       0.00      0.00      0.00        11

    accuracy                           0.97     24700
   macro avg       0.87      0.79      0.80     24700
weighted avg       0.97      0.97      0.97     24700
```



✓ 35s completed at 12:45 PM