In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,accuracy_score
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
train_set = pd.read_csv(r"C:\Users\kunal perane\Downloads\Training.csv.zip")
test_set = pd.read_csv(r"C:\Users\kunal perane\Downloads\Testing.csv")
train_set = train_set.iloc[:,:-1]
train_set.head()
```

Out[2]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	ac
0	1	1	1	0	0	0	0	0	
1	0	1	1	0	0	0	0	0	
2	1	0	1	0	0	0	0	0	
3	1	1	0	0	0	0	0	0	
4	1	1	1	0	0	0	0	0	

5 rows × 133 columns

In [3]:
```python
train_set.shape
```

Out[3]:
```
(4920, 133)
```

In [4]:
```python
test_set.shape
```

Out[4]:
```
(42, 133)
```

In [5]:
```python
train_set.describe()
```

Out[5]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.00
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.13
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.34
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

8 rows × 132 columns

In [6]:
```
test_set.describe()
```

Out[6]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stom
count	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	4
mean	0.166667	0.190476	0.023810	0.047619	0.023810	0.166667	0.142857	
std	0.377195	0.397437	0.154303	0.215540	0.154303	0.377195	0.354169	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 132 columns

In [7]:
```
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 133 entries, itching to prognosis
dtypes: int64(132), object(1)
memory usage: 5.0+ MB
```

In [8]:
```
test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Columns: 133 entries, itching to prognosis
dtypes: int64(132), object(1)
memory usage: 43.8+ KB
```

In [9]:
```
train_set.tail(3)
```

Out[9]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain
4917	0	0	0	0	0	0	0	0
4918	0	1	0	0	0	0	1	0
4919	0	1	0	0	0	0	0	0

3 rows × 133 columns

In [10]:
```
test_set.tail(4)
```

Out[10]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain
38	0	0	0	0	0	0	0	0
39	0	1	0	0	0	0	1	0
40	0	1	0	0	0	0	0	0
41	1	1	0	0	0	0	0	0

4 rows × 133 columns

In [11]:
```
train_set.isnull().sum()
```

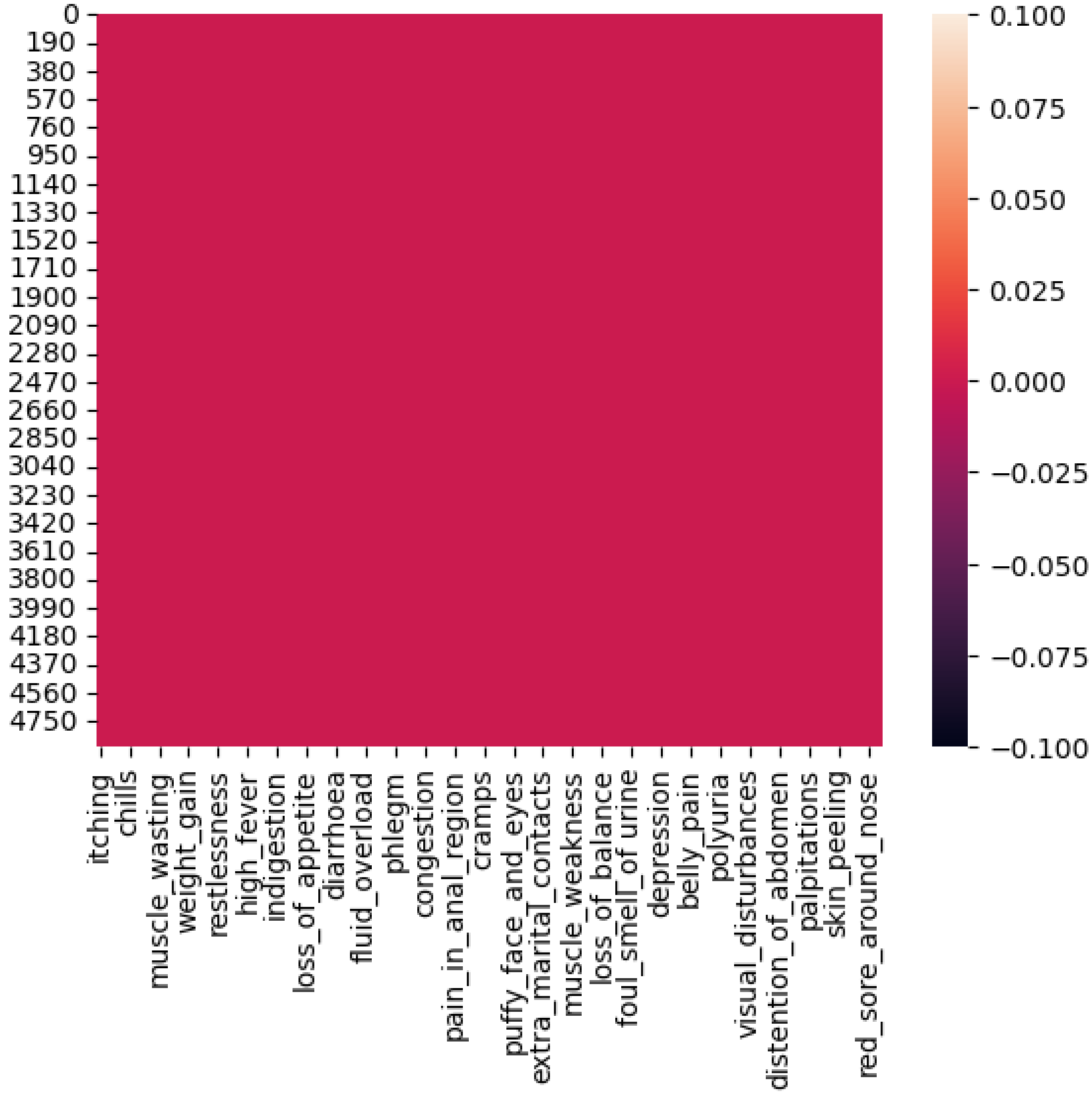
Out[11]:
```
itching                 0
skin_rash               0
nodal_skin_eruptions    0
continuous_sneezing     0
shivering               0
                       ..
inflammatory_nails      0
blister                 0
red_sore_around_nose    0
yellow_crust_ooze       0
prognosis               0
Length: 133, dtype: int64
```

In [12]:
```
test_set.isnull().sum()
```

Out[12]:
```
itching                 0
skin_rash               0
nodal_skin_eruptions    0
continuous_sneezing     0
shivering               0
                       ..
inflammatory_nails      0
blister                 0
red_sore_around_nose    0
yellow_crust_ooze       0
prognosis               0
Length: 133, dtype: int64
```

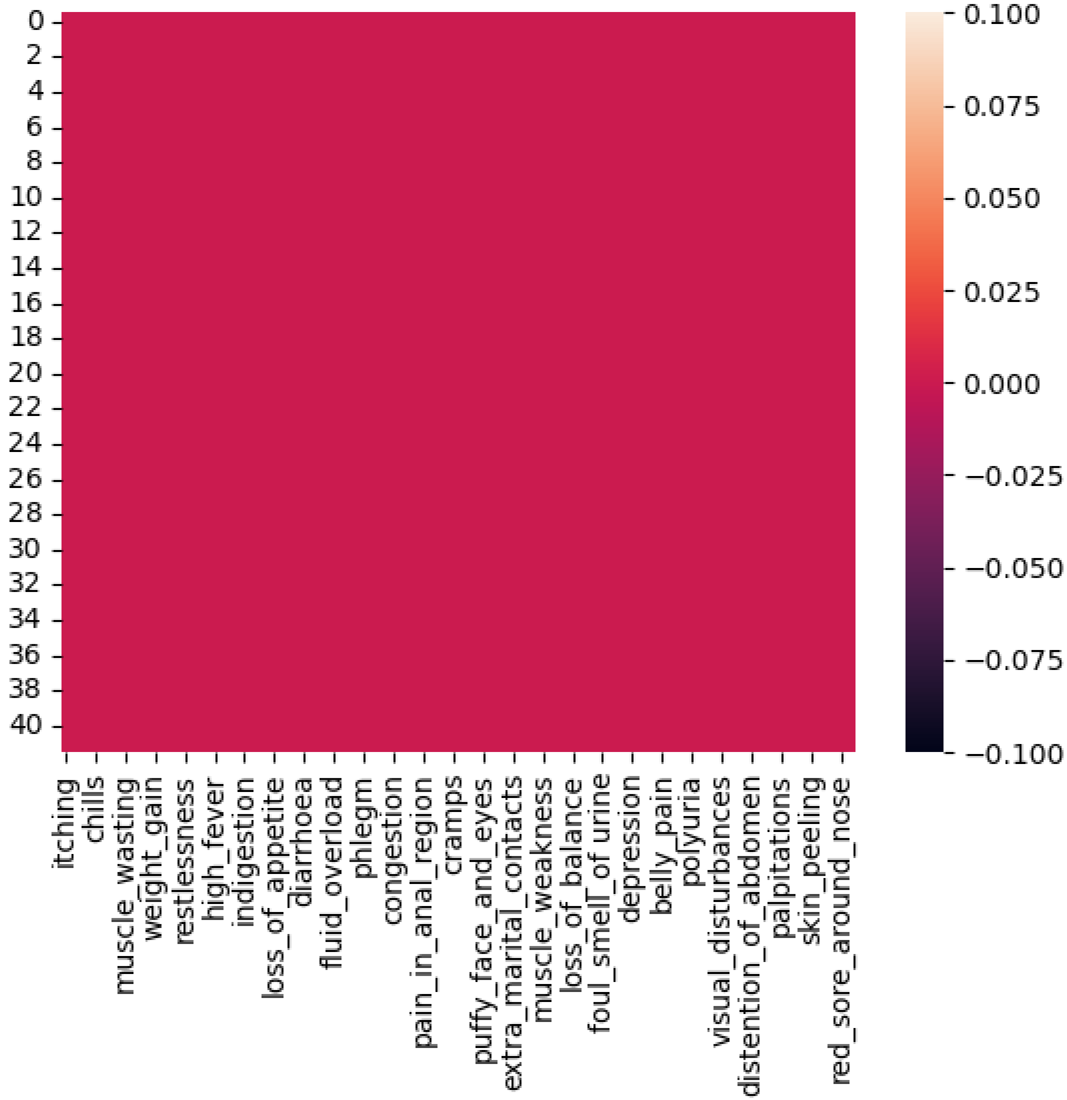
In [13]:
```
sns.heatmap(train_set.isnull())
```

Out[13]:
```
<Axes: >
```



In [14]:
```
sns.heatmap(test_set.isnull())
```

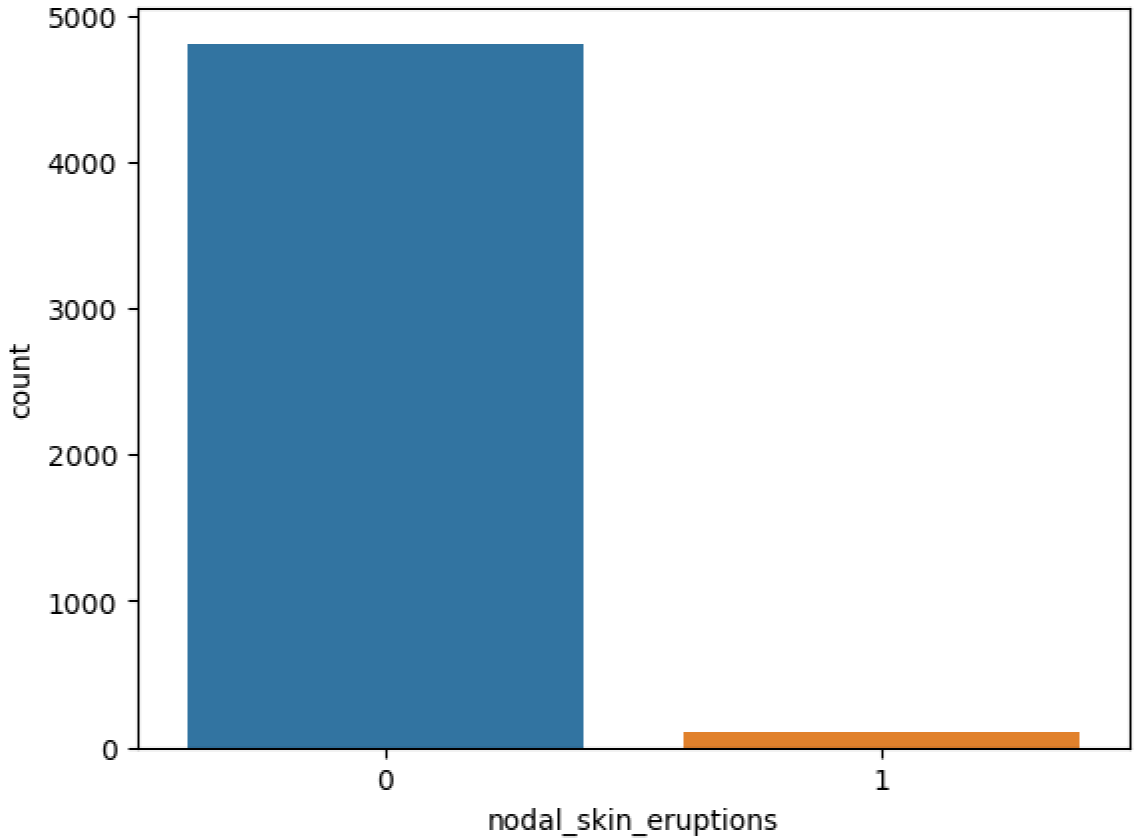
Out[14]: <Axes: >



In [15]:
```
sns.countplot(x=train_set['nodal_skin_eruptions'])
```

Out[15]: <Axes: xlabel='nodal_skin_eruptions', ylabel='count'>

Loading [MathJax]/extensions/Safe.js



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

Out[16]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	ac
0	1	1	1	0	0	0	0	0	
1	0	0	0	1	1	1	0	0	
2	0	0	0	0	0	0	0	1	
3	1	0	0	0	0	0	0	0	
4	1	1	0	0	0	0	0	1	

5 rows × 133 columns

In [17]:
```
train_set.dtypes
```

Out[17]:
```
itching                  int64
skin_rash                int64
nodal_skin_eruptions     int64
continuous_sneezing      int64
shivering                int64
                         ...
inflammatory_nails       int64
blister                  int64
red_sore_around_nose     int64
yellow_crust_ooze        int64
prognosis               object
Length: 133, dtype: object
```

In [18]:
```
test_set.dtypes
```

Out[18]:
```
itching                    int64
skin_rash                  int64
nodal_skin_eruptions       int64
continuous_sneezing        int64
shivering                  int64
                            ...
inflammatory_nails         int64
blister                    int64
red_sore_around_nose       int64
yellow_crust_ooze          int64
prognosis                 object
Length: 133, dtype: object
```

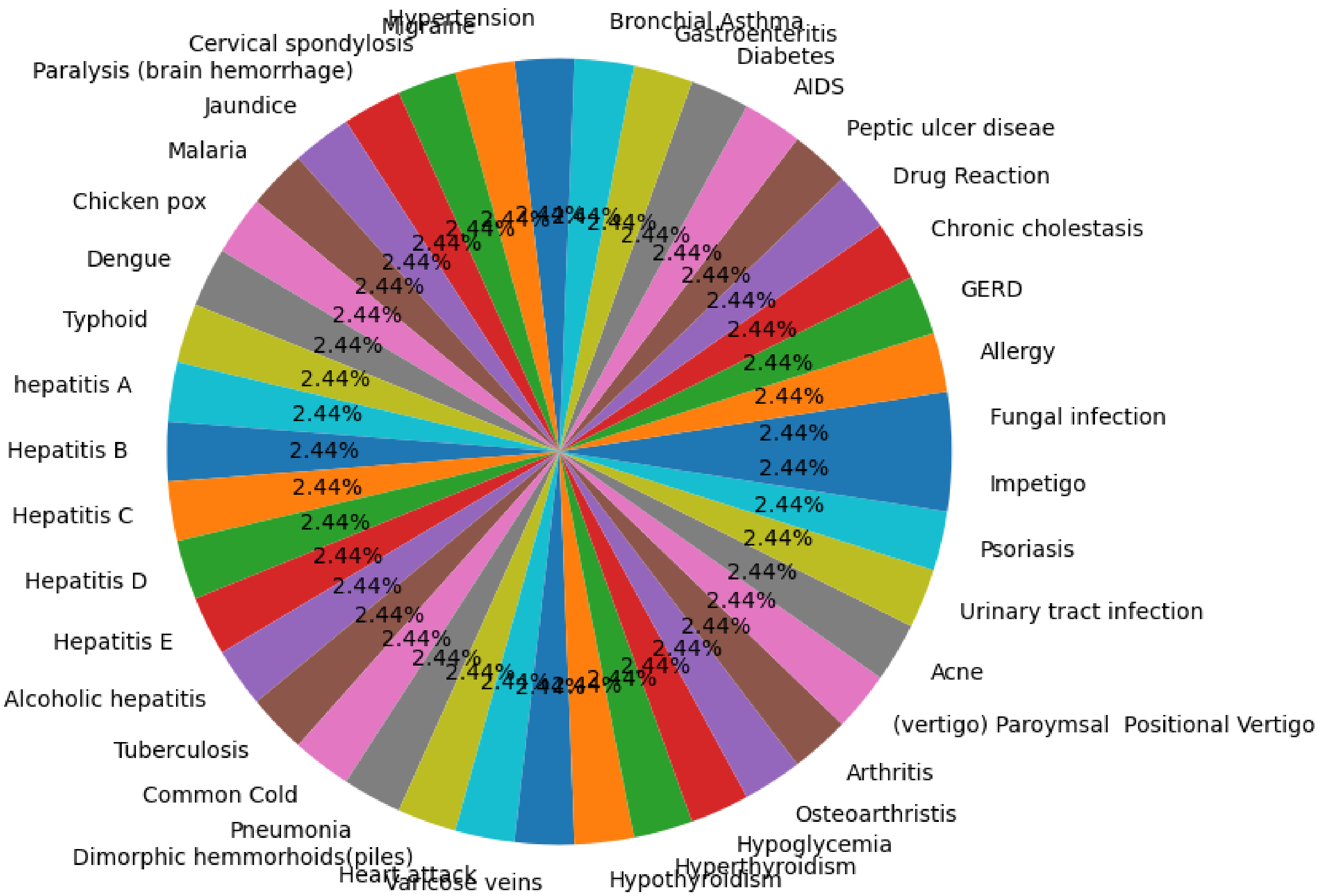
In [19]:
```python
train_set.nunique()
```

Out[19]:
```
itching                    2
skin_rash                  2
nodal_skin_eruptions       2
continuous_sneezing        2
shivering                  2
                          ..
inflammatory_nails         2
blister                    2
red_sore_around_nose       2
yellow_crust_ooze          2
prognosis                 41
Length: 133, dtype: int64
```

In [20]:
```python
test_set.nunique()
```

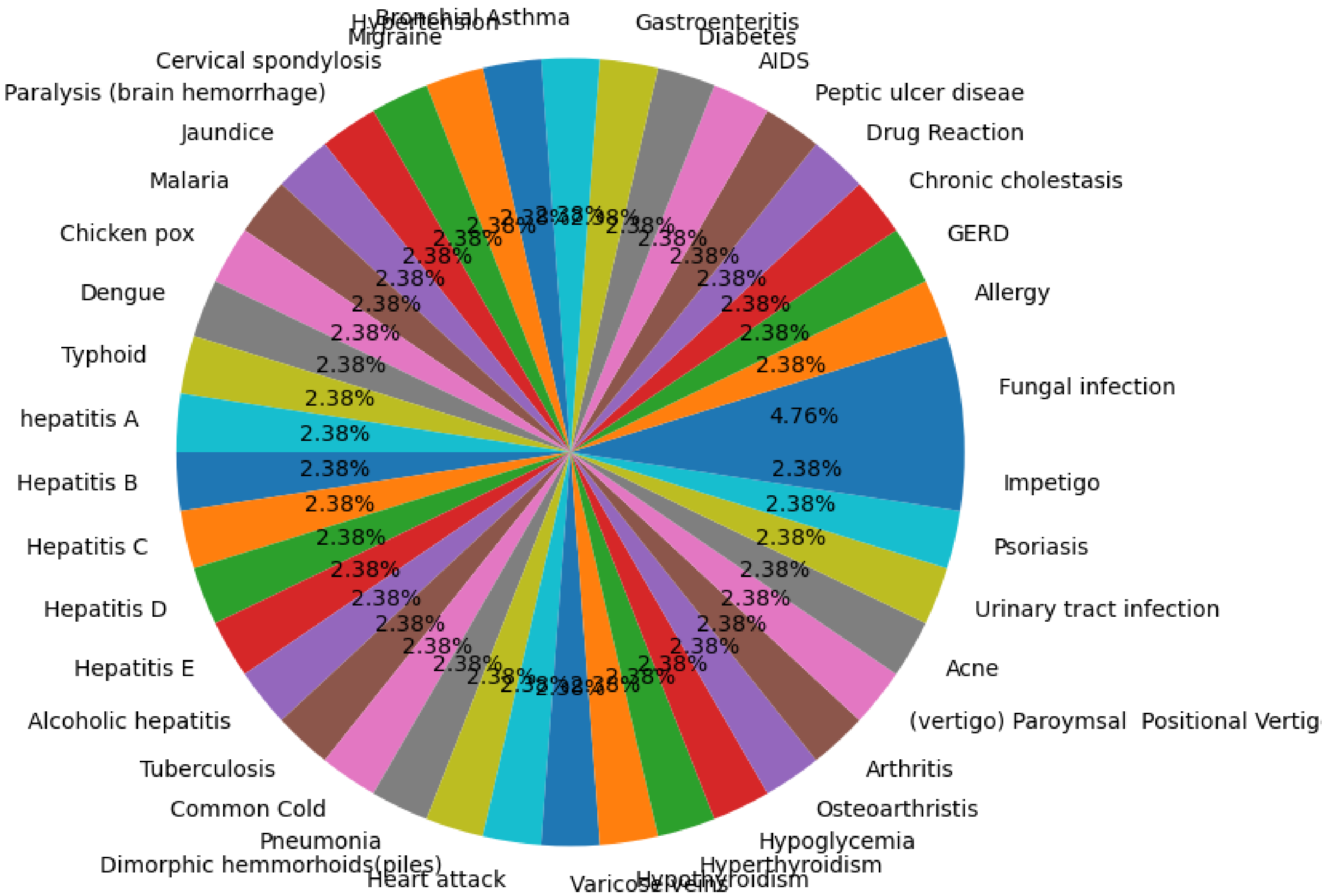
Out[20]:
```
itching                    2
skin_rash                  2
nodal_skin_eruptions       2
continuous_sneezing        2
shivering                  2
                          ..
inflammatory_nails         2
blister                    2
red_sore_around_nose       2
yellow_crust_ooze          2
prognosis                 41
Length: 133, dtype: int64
```

In [21]:
```python
plt.figure(figsize=(8, 8))
plt.pie(train_set['prognosis'].value_counts(), labels=train_set['prognosis'].unique(), a
plt.show()
```



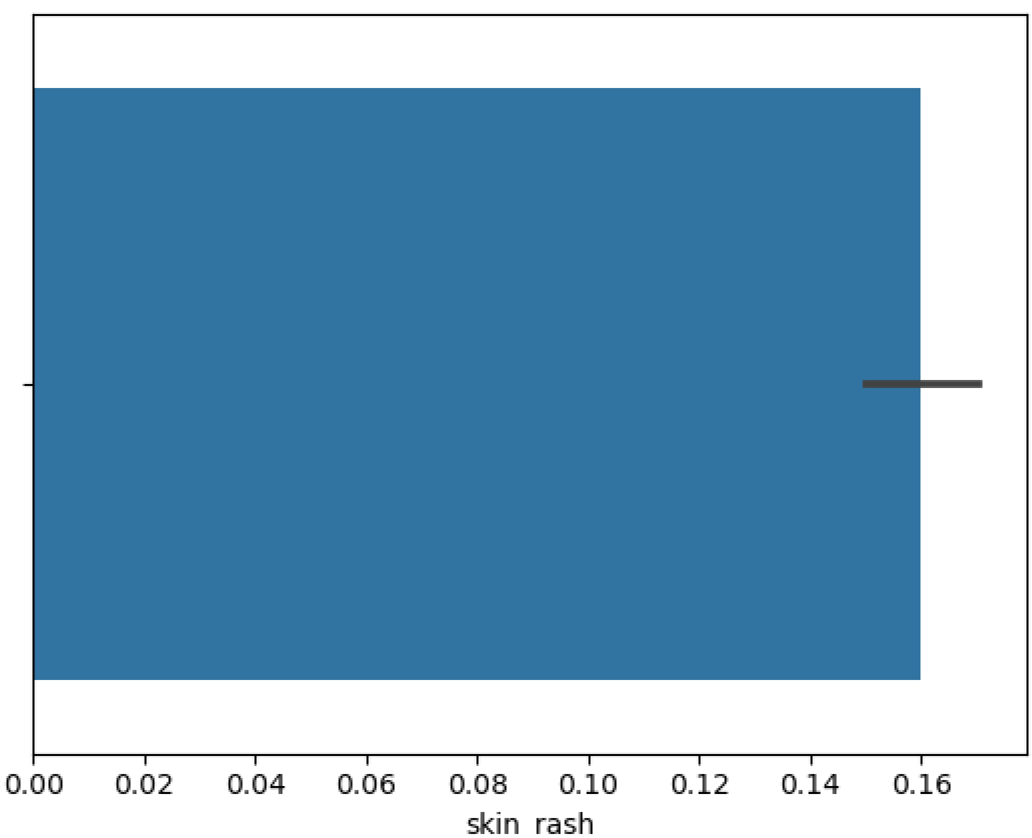
In [22]:
```
plt.figure(figsize=(8, 8))
plt.pie(test_set['prognosis'].value_counts(), labels=test_set['prognosis'].unique(), aut
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [23]:
```
sns.barplot(x=train_set['skin_rash'])
```

Out[23]: `<Axes: xlabel='skin_rash'>`



In [24]:
```python
train_set['prognosis'].value_counts()
```

Out[24]:
```
Fungal infection                           120
Hepatitis C                                120
Hepatitis E                                120
Alcoholic hepatitis                        120
Tuberculosis                               120
Common Cold                                120
Pneumonia                                  120
Dimorphic hemmorhoids(piles)               120
Heart attack                               120
Varicose veins                             120
Hypothyroidism                             120
Hyperthyroidism                            120
Hypoglycemia                               120
Osteoarthristis                            120
Arthritis                                  120
(vertigo) Paroymsal  Positional Vertigo    120
Acne                                       120
Urinary tract infection                    120
Psoriasis                                  120
Hepatitis D                                120
Hepatitis B                                120
Allergy                                    120
hepatitis A                                120
GERD                                       120
Chronic cholestasis                        120
Drug Reaction                              120
Peptic ulcer diseae                        120
AIDS                                       120
Diabetes                                   120
Gastroenteritis                            120
Bronchial Asthma                           120
Hypertension                               120
Migraine                                   120
Cervical spondylosis                       120
Paralysis (brain hemorrhage)               120
Jaundice                                   120
Malaria                                    120
Chicken pox                                120
Dengue                                     120
Typhoid                                    120
Impetigo                                   120
Name: prognosis, dtype: int64
```

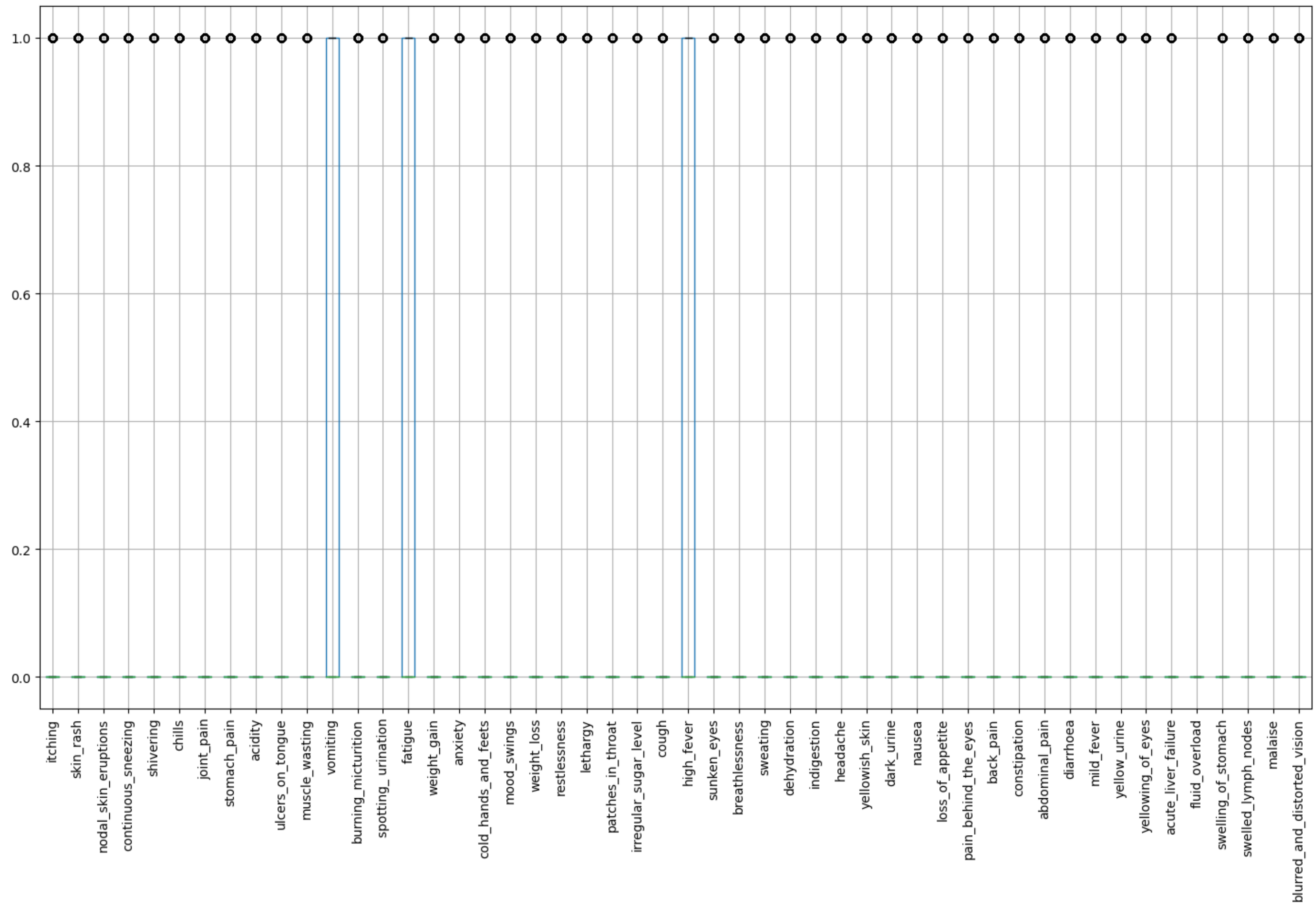
In [25]:
```python
print(f'**Summary**:\n There are 41 diseases in the dataset and each containing 120 rows
```

```
**Summary**:
 There are 41 diseases in the dataset and each containing 120 rows. So, the dataset is e
qually balanced.
```

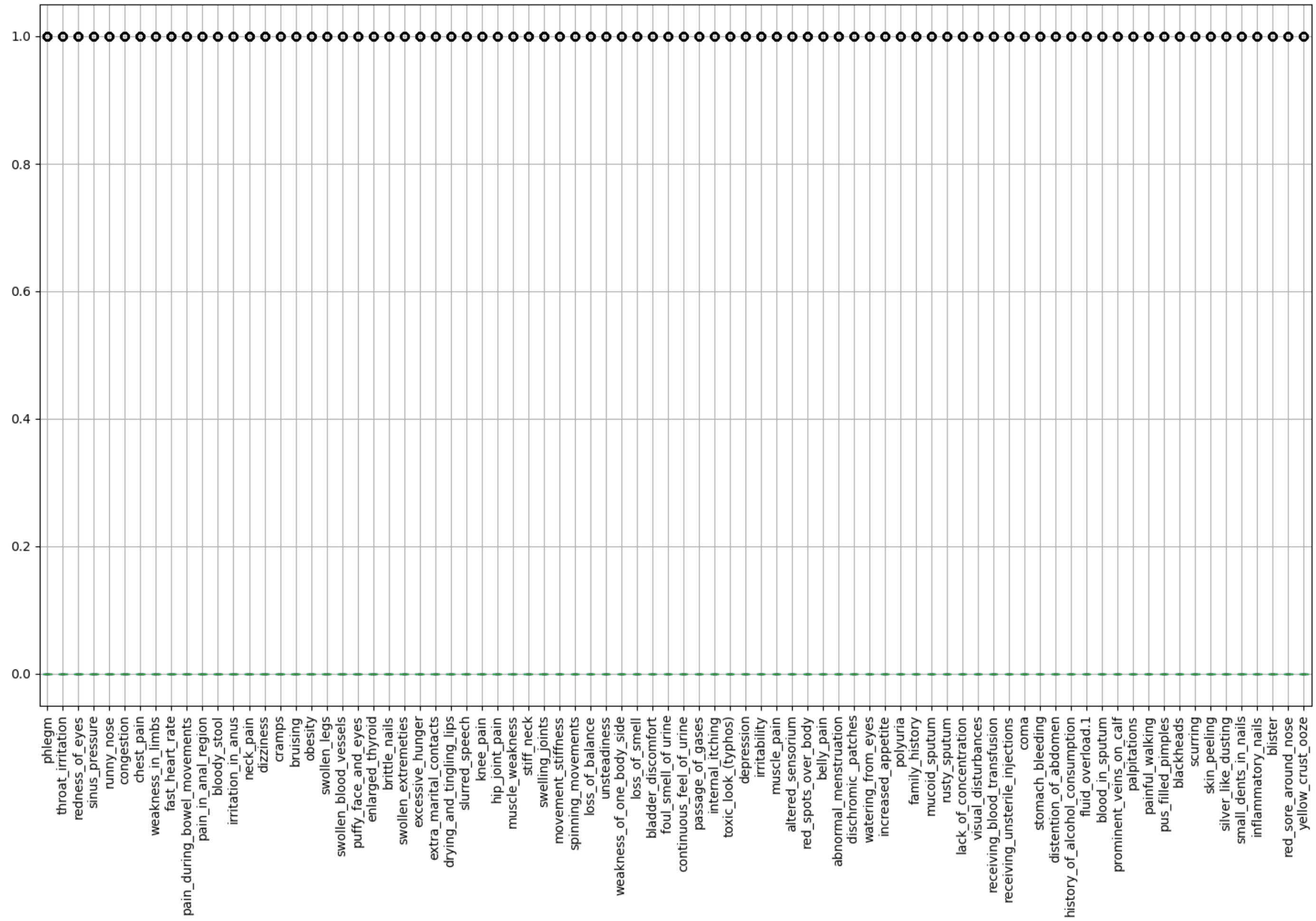
In [26]:
```python
total = train_set.isnull().sum().sort_values(ascending=False)
percent = (train_set.isnull().sum()/train_set.isnull().count()).sort_values(ascending=Fa
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

total = train_set.isna().sum().sort_values(ascending=False)
percent = (train_set.isna().sum()/train_set.isna().count()).sort_values(ascending=False)
na_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

if((na_data.all()).all()>0 or (na_data.all()).all()>0):
     print('Found Missing Data or NA values')

else:
    print('There is no missing data or null values in the collected data. Additionally,
```

There is no missing data or null values in the collected data. Additionally, the length of each column is same.

In [27]:
```
temp_df=train_set.iloc[:,:-1]
#Detect outliers
plt.subplots(figsize=(18,10))
temp_df.iloc[:,:50].boxplot()
plt.xticks(rotation=90)
plt.show()

plt.subplots(figsize=(18,10))
temp_df.iloc[:,50:].boxplot()
plt.xticks(rotation=90)
plt.show()

print(f'**Summary**:\n No outliers')
```



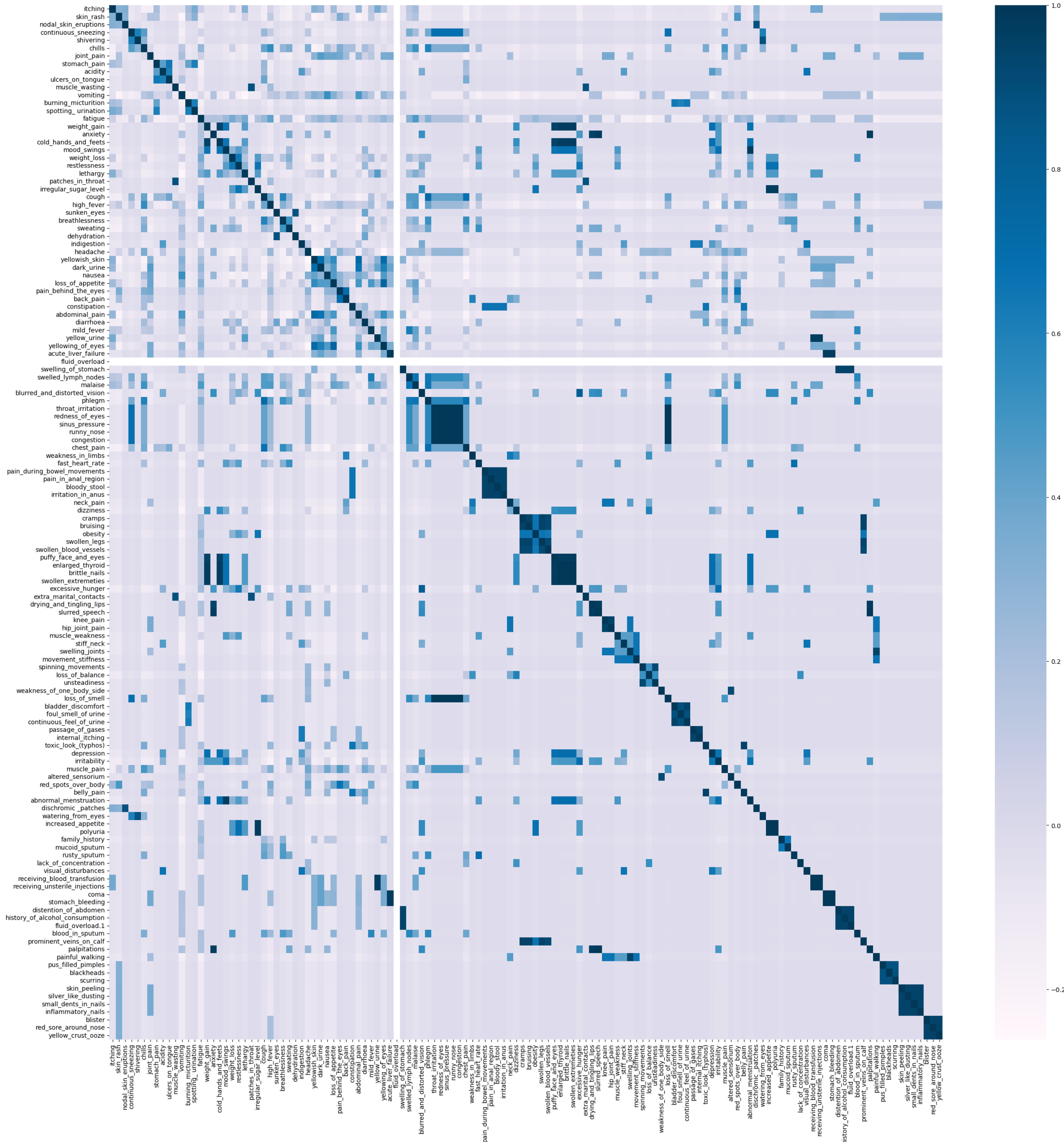
Loading [MathJax]/extensions/Safe.js




```
**Summary**:
 No outliers
```

In [28]:

```python
plt.figure(figsize = (30, 30))
sns.heatmap(train_set.corr(), cmap = 'PuBu', annot = False)
plt.show()
```



In [29]:
```
import numpy.testing as testing
```

In [30]:
```
corr_matrix=train_set.corr()
upper =upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
upper
```

Out[30]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	join
itching	NaN	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.1
skin_rash	NaN	NaN	0.298143	-0.094786	-0.065324	-0.029324	0.1
nodal_skin_eruptions	NaN	NaN	NaN	-0.032566	-0.022444	-0.065917	-0.0
continuous_sneezing	NaN	NaN	NaN	NaN	0.608981	0.446238	-0.0
shivering	NaN	NaN	NaN	NaN	NaN	0.295332	-0.0
...	...	...	...	...	...	...	
small_dents_in_nails	NaN	NaN	NaN	NaN	NaN	NaN	
inflammatory_nails	NaN	NaN	NaN	NaN	NaN	NaN	
blister	NaN	NaN	NaN	NaN	NaN	NaN	
red_sore_around_nose	NaN	NaN	NaN	NaN	NaN	NaN	
yellow_crust_ooze	NaN	NaN	NaN	NaN	NaN	NaN	

132 rows × 132 columns

In [31]:

```
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
print(to_drop,len(to_drop))

train_set=train_set.drop(to_drop, axis=1)
test_set=test_set.drop(to_drop, axis=1)
```

```
['cold_hands_and_feets', 'redness_of_eyes', 'sinus_pressure', 'runny_nose', 'congestio
n', 'pain_in_anal_region', 'bloody_stool', 'irritation_in_anus', 'bruising', 'swollen_le
gs', 'swollen_blood_vessels', 'puffy_face_and_eyes', 'enlarged_thyroid', 'brittle_nail
s', 'swollen_extremeties', 'drying_and_tingling_lips', 'slurred_speech', 'hip_joint_pai
n', 'unsteadiness', 'loss_of_smell', 'continuous_feel_of_urine', 'internal_itching', 'al
tered_sensorium', 'belly_pain', 'abnormal_menstruation', 'increased_appetite', 'polyuri
a', 'receiving_blood_transfusion', 'receiving_unsterile_injections', 'coma', 'stomach_bl
eeding', 'distention_of_abdomen', 'history_of_alcohol_consumption', 'fluid_overload.1',
'prominent_veins_on_calf', 'palpitations', 'painful_walking', 'silver_like_dusting', 'sm
all_dents_in_nails', 'inflammatory_nails', 'red_sore_around_nose', 'yellow_crust_ooze']
42
```

In [32]:

```
temp_train=train_set.iloc[:,:-1]
```

In [33]:

```
from sklearn.feature_selection import VarianceThreshold
```

In [34]:

```
sel = VarianceThreshold(threshold=0.03)
sel.fit(temp_train)
```

Out[34]:

```
▼ VarianceThreshold
VarianceThreshold(threshold=0.03)
```

In [35]:

```
print(
    len([
        x for x in temp_train.columns
        if x not in temp_train.columns[sel.get_support()]
    ]))

to_drop=[x for x in temp_train.columns if x not in temp_train.columns[sel.get_support()]]
train_set=train_set.drop(to_drop, axis=1)
test_set=test_set.drop(to_drop, axis=1)
```

In [36]:
```python
encoder = LabelEncoder()
train_set["prognosis"] = encoder.fit_transform(train_set["prognosis"])
test_set["prognosis"] = encoder.transform(test_set["prognosis"])
```

In [37]:
```python
X_train, X_valid, y_train, y_valid = train_test_split(train_set.drop('prognosis', 1), tr
```

In [38]:
```python
X_train.shape
```

Out[38]:
```
(2952, 49)
```

In [39]:
```python
test_set = pd.concat([test_set,pd.concat([X_valid,y_valid],axis=1)],axis=0)
test_set.shape
```

Out[39]:
```
(2010, 50)
```

In [40]:
```python
svm = SVC()
svm.fit(X_train, y_train)
y_pred=svm.predict(X_valid)
print("SVM Train score with ",format(svm.score(X_train, y_train)))
```

```
SVM Train score with  0.9854336043360433
```

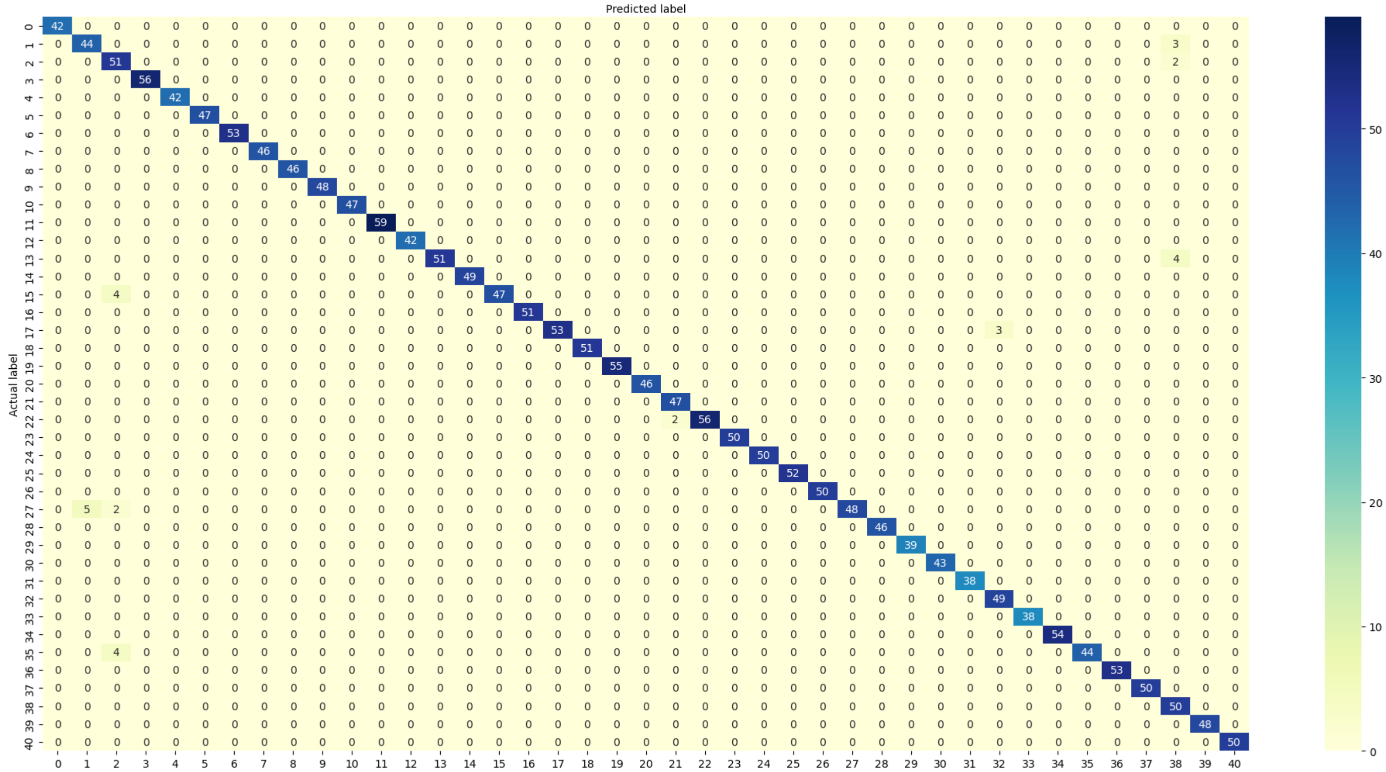
In [41]:
```python
print("SVM Test score with ",format(svm.score(test_set.iloc[:,:-1], test_set['prognosis'
```

```
SVM Test score with  0.9855721393034826
```

In [42]:
```python
y_pred = svm.predict(test_set.iloc[:,:-1])
class_names=encoder.classes_
fig, ax = plt.subplots(figsize = (20,10))
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
cm = confusion_matrix(test_set['prognosis'], y_pred)
sns.heatmap(cm, annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[42]:
```
Text(0.5, 885.5555555555555, 'Predicted label')
```

Confusion matrix



In [43]:

```
print(classification_report( test_set['prognosis'], y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        42
           1       0.90      0.94      0.92        47
           2       0.84      0.96      0.89        53
           3       1.00      1.00      1.00        56
           4       1.00      1.00      1.00        42
           5       1.00      1.00      1.00        47
           6       1.00      1.00      1.00        53
           7       1.00      1.00      1.00        46
           8       1.00      1.00      1.00        46
           9       1.00      1.00      1.00        48
          10       1.00      1.00      1.00        47
          11       1.00      1.00      1.00        59
          12       1.00      1.00      1.00        42
          13       1.00      0.93      0.96        55
          14       1.00      1.00      1.00        49
          15       1.00      0.92      0.96        51
          16       1.00      1.00      1.00        51
          17       1.00      0.95      0.97        56
          18       1.00      1.00      1.00        51
          19       1.00      1.00      1.00        55
          20       1.00      1.00      1.00        46
          21       0.96      1.00      0.98        47
          22       1.00      0.97      0.98        58
          23       1.00      1.00      1.00        50
          24       1.00      1.00      1.00        50
          25       1.00      1.00      1.00        52
          26       1.00      1.00      1.00        50
          27       1.00      0.87      0.93        55
          28       1.00      1.00      1.00        46
          29       1.00      1.00      1.00        39
          30       1.00      1.00      1.00        43
          31       1.00      1.00      1.00        38
          32       0.94      1.00      0.97        49
          33       1.00      1.00      1.00        38
          34       1.00      1.00      1.00        54
          35       1.00      0.92      0.96        48
          36       1.00      1.00      1.00        53
          37       1.00      1.00      1.00        50
          38       0.85      1.00      0.92        50
          39       1.00      1.00      1.00        48
          40       1.00      1.00      1.00        50

    accuracy                           0.99      2010
   macro avg       0.99      0.99      0.99      2010
weighted avg       0.99      0.99      0.99      2010
```

In [44]:

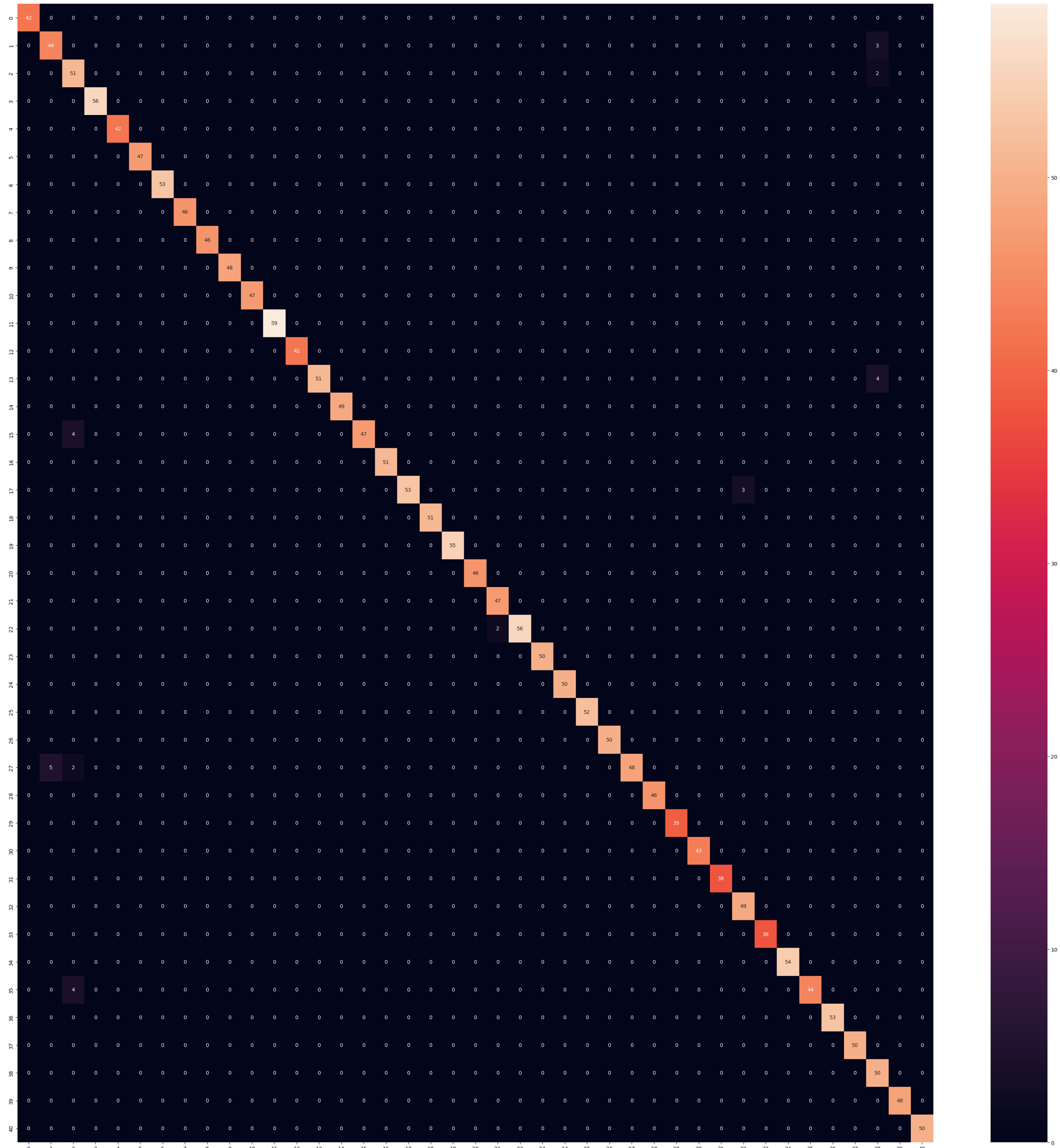
```
cm = confusion_matrix(test_set['prognosis'], y_pred)
print('Confusion Matrix:')
print(cm)
```

```
Confusion Matrix:
[[42  0  0 ...  0  0  0]
 [ 0 44  0 ...  3  0  0]
 [ 0  0 51 ...  2  0  0]
 ...
 [ 0  0  0 ... 50  0  0]
 [ 0  0  0 ...  0 48  0]
 [ 0  0  0 ...  0  0 50]]
```

In [45]:

```
plt.figure(figsize = (40, 40))
sns.heatmap(cm,annot=True)
```

Out[45]: <Axes: >



In [46]:
```
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
```

In [47]:
```
yb=label_binarize(test_set['prognosis'],classes=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,1
```

In [48]:
```
nc=yb.shape[1]
```

In [49]:
```
classifier=OneVsRestClassifier(SVC(kernel='linear',probability=True,random_state=42))
```

In [50]:
```
svm_classifier = SVC(probability=True)
svm_classifier.fit(X_train, y_train)
```

Loading [MathJax]/extensions/Safe.js

Out[50]:

```
▼          SVC
SVC(probability=True)
```

In [51]:

```
y_score = svm_classifier.decision_function(X_valid)
```

In [52]:

```
print("y_score for the first 5 samples:")
print(y_score[:5])
```

```
y_score for the first 5 samples:
[[21.86158193 35.31605563 40.32520354 28.28737217 34.31223023 26.2773816
  18.72010354 23.80368093  7.68847269 15.69353494  2.68514062  0.6854393
  14.69406504 30.31388998 29.30826924 38.32432379 19.71490952 32.31165209
  21.83163656  8.68730457 17.70207148  9.68787455  3.68495509 23.82396322
   4.68609704 10.68903078 16.69828844 39.32478612 11.68946481 12.6895649
  13.68998022 27.27823604 33.31492039 22.86124676  5.68613547 36.32254339
  -0.31694068  6.68688946 37.31919178 31.31259668  1.68470662]
 [21.86158193 35.31605563 40.32520354 28.28737217 34.31223023 26.2773816
  18.72010354 23.80368093  7.68847269 15.69353494  2.68514062  0.6854393
  14.69406504 30.31388998 29.30826924 38.32432379 19.71490952 32.31165209
  21.83163656  8.68730457 17.70207148  9.68787455  3.68495509 23.82396322
   4.68609704 10.68903078 16.69828844 39.32478612 11.68946481 12.6895649
  13.68998022 27.27823604 33.31492039 22.86124676  5.68613547 36.32254339
  -0.31694068  6.68688946 37.31919178 31.31259668  1.68470662]
 [23.83456635  1.70923505  4.71614246 22.8625736  16.87172602 36.23374086
  33.09592164 20.8246673   7.7584857  13.7715227   6.75441234  3.74867728
  39.28542832 -0.29237315 19.91719316 12.78483245 23.81280433 29.21106443
  35.20265061  9.76684094 25.87805492 13.80556376  9.75123956 20.84572736
  40.32644599 38.27201957 37.25166492 11.73145074 29.9378065  27.89120683
  29.91544485 23.91914853 13.82898978 22.86731409 30.92335255  9.74197989
   4.75287557 18.80637311  4.7149726  32.23466916  6.75372492]
 [21.93938204 40.32520406 37.31913306 29.29248036 36.31445246 27.28540627
  26.28979018 23.84769252  4.68848404 13.69317858  3.68823014  0.68532037
  12.69405027 31.31563432 22.84842482 35.31353416 18.7168079  33.31390391
  20.88276728  5.68711185 17.70200242  7.68713147  6.68975368 22.88090956
   2.68599468  9.68891004 14.69845259 38.32212059 16.70363063 15.70380476
  10.68994449 28.28702305 34.31661238 21.94566488  8.69079789 30.30969676
  -0.3161259  11.69561008 39.32077274 32.3145932   1.68456016]
 [31.27873909 14.70645571 17.71245641 35.31438707 21.75714572 19.73571931
   9.71554899 11.72145241  5.71095854 40.32682363 -0.29851255  7.71178904
   6.7080031  11.70405244 27.94541676 28.24615308 24.7531733  29.2265053
  25.83054034 34.29142775 37.30706834 39.31558273 38.30834713 12.72335083
   0.70204532  9.72688171  4.70611475 14.71304487 32.26477348 13.73244217
   3.70595448 19.73826005 29.25762006 36.30965732  2.70251232 17.71817129
   1.70557798 22.74422509 19.71472817 21.74582191 33.28657843]]
```

In [53]:

```
y_valid_binary = label_binarize(y_valid, classes=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
```

In [54]:

```
n_classes = y_valid_binary.shape[1]
```

In [55]:

```
fpr = dict()
tpr = dict()
roc_auc = dict()
```

In [56]:

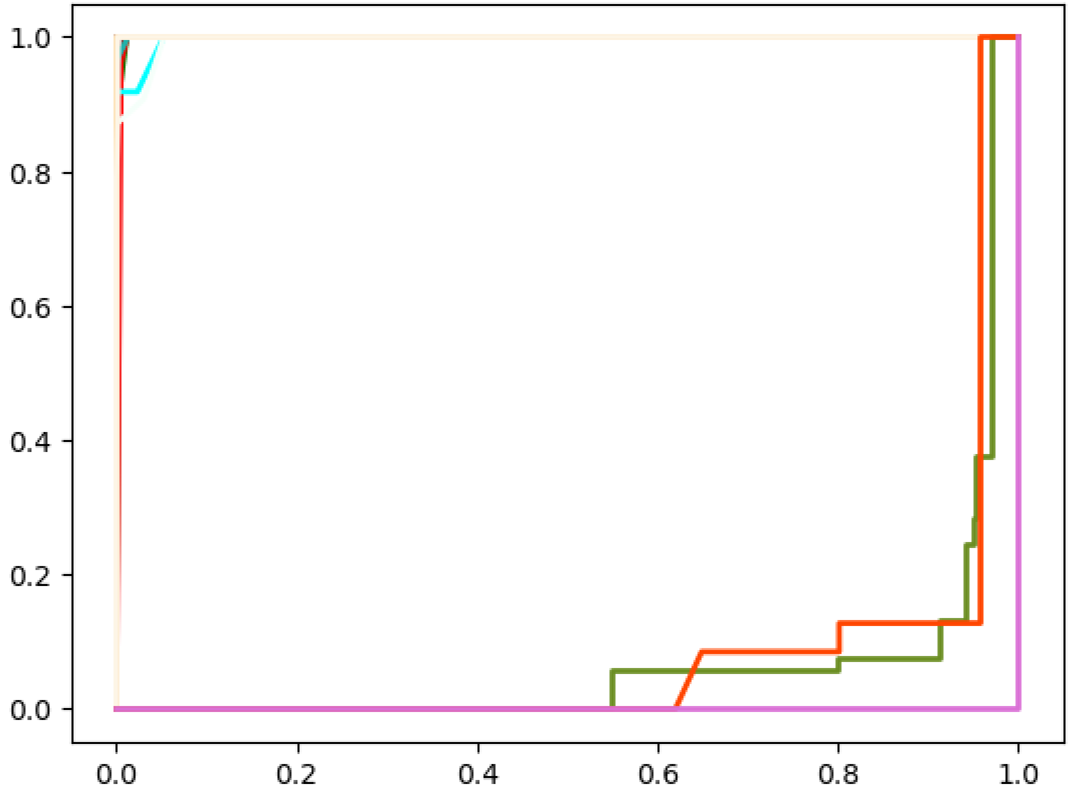
```
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_valid_binary[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [57]:

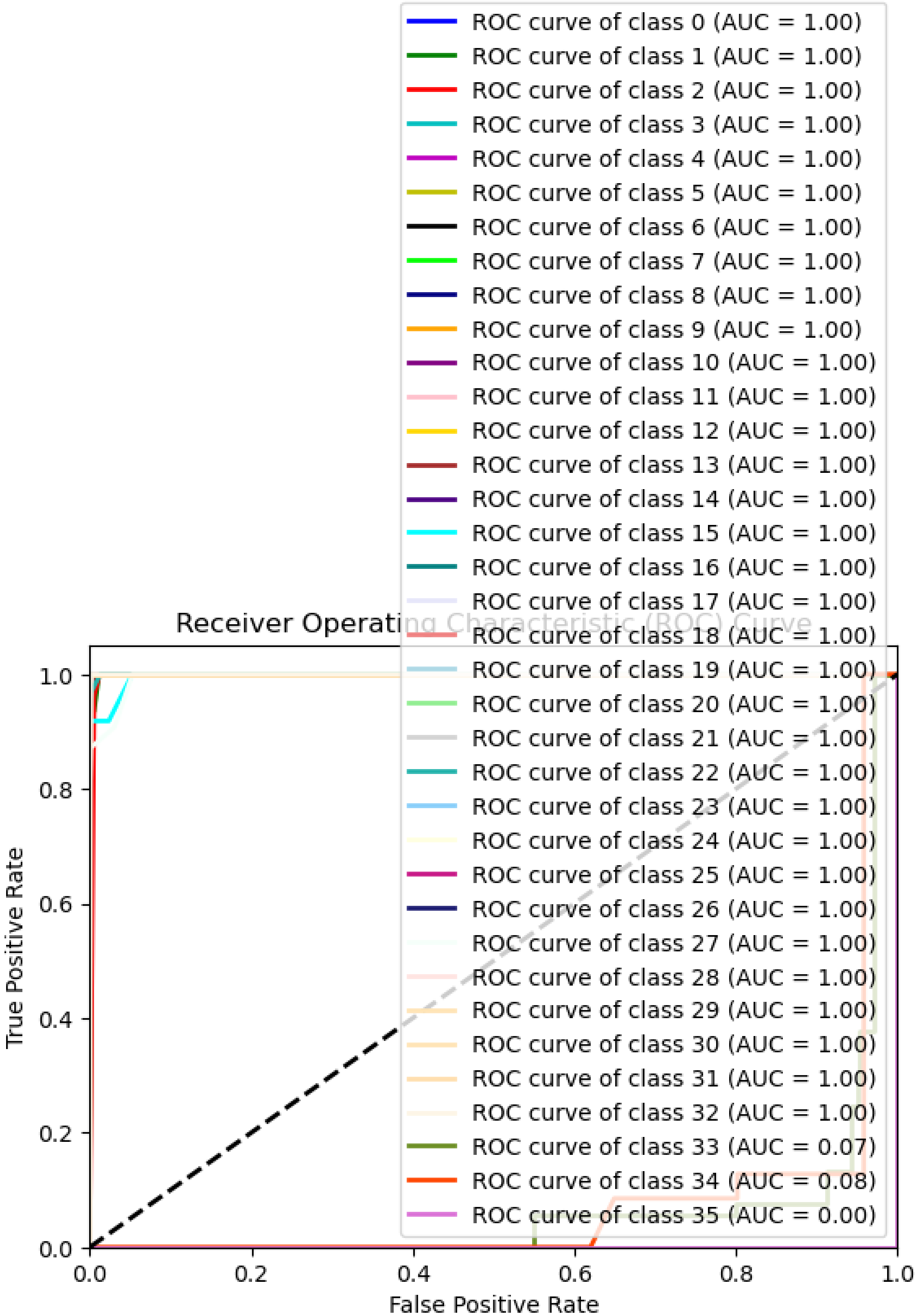
```
plt.figure(figsize=(10, 8))
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'lime', 'navy', 'orange', 'purple', 'pink',
```

Loading [MathJax]/extensions/Safe.js 1000x800 with 0 Axes>

In [58]:
```
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label='ROC curve of class {0} (AUC = {1:
             ''.format(i, roc_auc[i]))
```



In [59]:
```
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label='ROC curve of class {0} (AUC = {1:
             ''.format(i, roc_auc[i]))
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right", bbox_to_anchor=(1, 0))
plt.show()
```



In [60]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

In [61]:
```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear'],
}
```

In [62]:
```
svm = SVC()
```

In [63]:
```
grid_search = GridSearchCV(svm, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Out[63]:

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

In [64]:
```
print("Best hyperparameters found: ", grid_search.best_params_)
print("Best accuracy on the validation set: {:.2f}".format(grid_search.best_score_))
```

```
Best hyperparameters found:  {'C': 10, 'kernel': 'linear'}
Best accuracy on the validation set: 0.98
```

In [65]:
```
best_svm = grid_search.best_estimator_
best_svm.fit(X_train, y_train)
```

Out[65]:

▾ SVC

```
SVC(C=10, kernel='linear')
```

In [66]:
```
test_features = test_set.drop('prognosis', axis=1)
test_accuracy = best_svm.score(test_features, test_set['prognosis'])
print("Test accuracy: {:.2f}".format(test_accuracy))
```

```
Test accuracy: 0.99
```

In [67]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

In [68]:
```
svm=SVC(kernel='linear')
param_dist={
    'C':uniform(loc=0,scale=10),
    'gamma':['scale','auto']+list(uniform(loc=0,scale=1).rvs(10)),
}
```

In [69]:
```
random_search=RandomizedSearchCV(svm,param_distributions=param_dist,n_iter=10,cv=10,n_jo
random_search.fit(X_train,y_train)
```

Out[69]:

▸ **RandomizedSearchCV**

▸ **estimator: SVC**

▸ SVC

In [70]:
```
best_model = random_search.best_estimator_
best_params = random_search.best_params_
```

In [71]:
```
y_pred = best_model.predict(test_set.iloc[:,:-1])
accuracy = accuracy_score( test_set['prognosis'], y_pred)
print('Accuracy:',accuracy)
print('Classification Report:')
print(classification_report( test_set['prognosis'],y_pred))
```

```
Accuracy: 0.9855721393034826
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        42
           1       0.90      0.94      0.92        47
           2       0.84      0.96      0.89        53
           3       1.00      1.00      1.00        56
           4       1.00      1.00      1.00        42
           5       1.00      1.00      1.00        47
           6       1.00      1.00      1.00        53
           7       1.00      1.00      1.00        46
           8       1.00      1.00      1.00        46
           9       1.00      1.00      1.00        48
          10       1.00      1.00      1.00        47
          11       1.00      1.00      1.00        59
          12       1.00      1.00      1.00        42
          13       1.00      0.93      0.96        55
          14       1.00      1.00      1.00        49
          15       1.00      0.92      0.96        51
          16       1.00      1.00      1.00        51
          17       1.00      0.95      0.97        56
          18       1.00      1.00      1.00        51
          19       1.00      1.00      1.00        55
          20       1.00      1.00      1.00        46
          21       0.96      1.00      0.98        47
          22       1.00      0.97      0.98        58
          23       1.00      1.00      1.00        50
          24       1.00      1.00      1.00        50
          25       1.00      1.00      1.00        52
          26       1.00      1.00      1.00        50
          27       1.00      0.87      0.93        55
          28       1.00      1.00      1.00        46
          29       1.00      1.00      1.00        39
          30       1.00      1.00      1.00        43
          31       1.00      1.00      1.00        38
          32       0.94      1.00      0.97        49
          33       1.00      1.00      1.00        38
          34       1.00      1.00      1.00        54
          35       1.00      0.92      0.96        48
          36       1.00      1.00      1.00        53
          37       1.00      1.00      1.00        50
          38       0.85      1.00      0.92        50
          39       1.00      1.00      1.00        48
          40       1.00      1.00      1.00        50

    accuracy                           0.99      2010
   macro avg       0.99      0.99      0.99      2010
weighted avg       0.99      0.99      0.99      2010
```

In [72]:
```python
from sklearn.naive_bayes import MultinomialNB
```

In [73]:
```python
naive_bayes_classifier = MultinomialNB()
```

In [74]:
```python
naive_bayes_classifier.fit(X_train, y_train)
```

Out[74]:
```
▼ MultinomialNB
MultinomialNB()
```

In [75]:
```python
y_pred = naive_bayes_classifier.predict(X_valid)
```

In [76]:
```
accuracy = accuracy_score(y_valid, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9822154471544715
```

In [ ]: