In [1]:
```
#importing the necessary libraries

import pandas as pd
import numpy as np
#pd.set_option('max_rows', 20)
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
dis_path = ("disease_train.csv")
```

In [3]:
```
disease = pd.read_csv(dis_path)
```

In [4]:
```
disease.head()
```

Out[4]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medicatic
0	PA1001	59	male	no	no	7.8	NaN	89.0	0.85	105.0	...	no	no	no	
1	PA1002	48	female	no	no	1.5	2.5	101.0	0.97	104.0	...	no	no	yes	
2	PA1003	77	male	no	no	7.3	1.2	57.0	1.28	44.0	...	no	no	no	
3	PA1004	42	female	no	no	1.2	2.5	106.0	0.98	108.0	...	no	no	no	
4	PA1005	38	female	no	no	0.6	1.9	95.0	NaN	NaN	...	no	no	no	

5 rows × 24 columns

In [5]:
```
disease.tail()
```

Out[5]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
4245	PA5246	29	male	no	no	0.5	2.3	105.0	0.86	122.0	...	no	no	no	
4246	PA5247	46	female	no	no	35.0	1.2	16.0	0.86	19.0	...	no	no	no	
4247	PA5248	72	female	no	no	3.4	2.1	88.0	0.96	92.0	...	no	no	no	
4248	PA5249	94	female	no	no	3.9	NaN	157.0	1.02	154.0	...	no	no	no	
4249	PA5250	41	male	no	no	1.8	2.8	60.0	0.95	63.0	...	no	no	no	

5 rows × 24 columns

In [6]:
```
disease.shape
```

Out[6]:
```
(4250, 24)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4250 entries, 0 to 4249
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               4250 non-null   object 
 1   age              4250 non-null   int64  
 2   gender           4109 non-null   object 
 3   sick             4250 non-null   object 
 4   pregnant         4250 non-null   object 
 5   test_X1          3839 non-null   float64
 6   test_X2          3007 non-null   float64
 7   test_X3          4034 non-null   float64
 8   test_X4          3858 non-null   float64
 9   test_X5          3863 non-null   float64
 10  test_X6          154 non-null    float64
 11  concern_type1    4250 non-null   object 
 12  concern_type2    4250 non-null   object 
 13  enlargement      4250 non-null   object 
 14  tumor            4250 non-null   object 
 15  disorder         4250 non-null   object 
 16  medication_A     4250 non-null   object 
 17  medication_B     4250 non-null   object 
 18  mental_health    4250 non-null   object 
 19  mood_stabiliser  4250 non-null   object 
 20  surgery          4250 non-null   object 
 21  treatment_type1  4250 non-null   object 
 22  suspect          4250 non-null   object 
 23  target           4250 non-null   object 
dtypes: float64(6), int64(1), object(17)
memory usage: 797.0+ KB
```

In [8]:
```
disease.isnull().sum()
```

Out[8]:
```
id                   0
age                  0
gender             141
sick                 0
pregnant             0
test_X1            411
test_X2           1243
test_X3            216
test_X4            392
test_X5            387
test_X6           4096
concern_type1        0
concern_type2        0
enlargement          0
tumor                0
disorder             0
medication_A         0
medication_B         0
mental_health        0
mood_stabiliser      0
surgery              0
treatment_type1      0
suspect              0
target               0
dtype: int64
```

In [9]:
```
#getting the missing values
missing_values_count = disease.isnull().sum()
```

In [10]:
```
# how many total missing values do we have?
total_cells = np.product(disease.shape)
total_missing = missing_values_count.sum()

# percent of data that is missing
percent_missing = (total_missing/total_cells) * 100
print(percent_missing)
```

```
6.750980392156864
```

In [11]:
```
disease.columns
```

Out[11]:
```
Index(['id', 'age', 'gender', 'sick', 'pregnant', 'test_X1', 'test_X2',
       'test_X3', 'test_X4', 'test_X5', 'test_X6', 'concern_type1',
       'concern_type2', 'enlargement', 'tumor', 'disorder', 'medication_A',
       'medication_B', 'mental_health', 'mood_stabiliser', 'surgery',
       'treatment_type1', 'suspect', 'target'],
      dtype='object')
```

In [12]:
```
CAT_COLUMNS = [ 'gender', 'sick', 'pregnant',  'concern_type1',
       'concern_type2', 'enlargement', 'tumor', 'disorder', 'medication_A',
```

```
       'medication_B', 'mental_health', 'mood_stabiliser', 'surgery',
       'treatment_type1', 'suspect']
NUM_COLUMNS = [ 'age',  'test_X1', 'test_X2',
       'test_X3', 'test_X4', 'test_X5', 'test_X6' ]
```

In [13]:

```
#Let us run loop of value_counts of each column to find out unique values.
for col in disease[CAT_COLUMNS]:
    print ("---- %s ---" % col)
    print (disease[col].value_counts())
```

```
---- gender ---
gender
female    2787
male      1322
Name: count, dtype: int64
---- sick ---
sick
no     4095
yes     155
Name: count, dtype: int64
---- pregnant ---
pregnant
no     4235
yes      15
Name: count, dtype: int64
---- concern_type1 ---
concern_type1
no     4183
yes      67
Name: count, dtype: int64
---- concern_type2 ---
concern_type2
no     3905
yes     345
Name: count, dtype: int64
---- enlargement ---
enlargement
no     4215
yes      35
Name: count, dtype: int64
---- tumor ---
tumor
no     4139
yes     111
Name: count, dtype: int64
---- disorder ---
disorder
no    4250
Name: count, dtype: int64
---- medication_A ---
medication_A
no     3760
yes     490
Name: count, dtype: int64
---- medication_B ---
medication_B
no     4196
yes      54
Name: count, dtype: int64
---- mental_health ---
mental_health
no     4056
yes     194
Name: count, dtype: int64
---- mood_stabiliser ---
mood_stabiliser
no     4205
yes      45
Name: count, dtype: int64
---- surgery ---
surgery
no     4188
yes      62
Name: count, dtype: int64
---- treatment_type1 ---
treatment_type1
no     4169
yes      81
Name: count, dtype: int64
---- suspect ---
suspect
no     3951
yes     299
Name: count, dtype: int64
```

In [14]:
```
disease.describe()
```

Out[14]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	3839.000000	3007.000000	4034.000000	3858.000000	3863.000000	154.000000
mean	67.374824	7.342463	2.035580	104.919623	0.970846	110.090834	23.325974
std	1004.518821	32.657963	0.920404	35.496255	0.162474	39.837621	5.317032
min	1.000000	0.005000	0.050000	2.000000	0.250000	1.400000	8.400000
25%	37.000000	0.600000	1.600000	87.000000	0.870000	92.000000	20.000000
50%	55.000000	1.500000	1.900000	102.000000	0.960000	107.000000	24.000000
75%	67.000000	3.000000	2.300000	121.000000	1.060000	125.000000	27.000000
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

In [15]:
```
disease[CAT_COLUMNS].describe()
```

Out[15]:

	gender	sick	pregnant	concern_type1	concern_type2	enlargement	tumor	disorder	medication_A	medication_B	menta
count	4109	4250	4250	4250	4250	4250	4250	4250	4250	4250	
unique	2	2	2	2	2	2	2	1	2	2	
top	female	no	no	no	no	no	no	no	no	no	
freq	2787	4095	4235	4183	3905	4215	4139	4250	3760	4196	

In [16]:
```
disease.loc[ (disease.target == 'high_risk') & (disease.pregnant == 'yes')]
```

Out[16]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
1605	PA2606	41	female	no	yes	0.2	NaN	430.0	1.09	395.0	...	no	no	no	

1 rows × 24 columns

In [17]:
```
disease.loc[ (disease.target == 'high_risk') & (disease.gender == 'female')]
```

Out[17]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
92	PA1093	71	female	no	no	0.050	NaN	178.0	1.13	158.0	...	no	no	no	
161	PA1162	39	female	no	no	0.020	4.0	209.0	1.06	197.0	...	no	no	no	
273	PA1274	65	female	no	no	0.150	3.5	217.0	0.98	221.0	...	no	no	no	
336	PA1337	71	female	no	no	0.200	4.5	125.0	0.82	152.0	...	yes	no	no	
377	PA1378	58	female	no	no	0.150	3.0	148.0	0.91	163.0	...	no	no	no	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4123	PA5124	28	female	no	no	0.100	3.0	235.0	1.38	171.0	...	no	no	no	
4130	PA5131	50	female	no	no	0.030	3.4	131.0	1.02	129.0	...	no	no	no	
4164	PA5165	72	female	no	no	0.015	2.9	198.0	0.91	217.0	...	no	no	no	
4204	PA5205	57	female	no	no	0.005	3.6	144.0	0.84	171.0	...	no	no	no	
4216	PA5217	47	female	no	no	0.150	4.3	189.0	1.07	176.0	...	no	no	no	

104 rows × 24 columns

In [18]:
```
disease.loc[ (disease.target == 'high_risk') & (disease.gender == 'male')]
```

Out[18]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	me
114	PA1115	37	male	no	no	NaN	4.300000	187.0	0.96	194.0	...	no	no	no	
123	PA1124	43	male	no	no	0.020	2.900000	130.0	0.80	163.0	...	no	no	no	
257	PA1258	36	male	no	no	0.070	4.100000	153.0	0.86	178.0	...	no	no	no	
316	PA1317	42	male	no	no	0.045	3.100000	119.0	0.80	149.0	...	no	no	no	
343	PA1344	39	male	no	no	0.105	4.100000	131.0	0.95	138.0	...	no	no	yes	
369	PA1370	76	male	no	no	0.065	3.900000	186.0	0.94	198.0	...	no	no	no	
720	PA1721	36	male	no	no	0.030	4.000000	178.0	1.14	156.0	...	no	no	no	
818	PA1819	41	male	no	no	0.100	6.200000	207.0	0.89	232.0	...	no	no	no	
827	PA1828	60	male	no	no	0.005	7.100000	184.0	1.03	177.0	...	no	no	no	
1009	PA2010	20	male	no	no	0.015	4.800000	22.0	NaN	NaN	...	no	no	no	
1135	PA2136	60	male	no	no	0.035	7.600000	236.0	0.91	259.0	...	no	no	no	
1211	PA2212	57	male	no	no	0.050	3.700000	244.0	0.95	256.0	...	no	no	no	
1279	PA2280	79	male	no	no	0.005	3.400000	151.0	0.74	203.0	...	no	no	no	
1280	PA2281	15	male	no	no	NaN	NaN	188.0	0.73	258.0	...	no	no	no	
1394	PA2395	34	male	no	no	0.050	7.600000	261.0	1.01	258.0	...	no	no	no	
1422	PA2423	24	male	no	no	0.300	3.800000	54.0	NaN	NaN	...	no	no	no	
1562	PA2563	34	male	no	no	0.300	4.700000	242.0	1.02	237.0	...	no	no	no	
1985	PA2986	80	male	no	no	0.050	3.700000	152.0	0.93	164.0	...	no	no	no	
2093	PA3094	25	male	no	no	0.100	3.300000	164.0	0.95	172.0	...	no	no	no	
2310	PA3311	63	male	no	no	NaN	NaN	135.0	0.64	211.0	...	no	no	no	
2623	PA3624	38	male	no	no	NaN	NaN	308.0	0.87	354.0	...	no	no	no	
2773	PA3774	28	male	no	no	0.020	5.900000	183.0	0.82	223.0	...	no	no	no	
2831	PA3832	75	male	yes	no	0.020	NaN	150.0	0.88	171.0	...	no	no	no	
3015	PA4016	34	male	no	no	0.015	4.000000	239.0	0.90	266.0	...	no	no	no	
3146	PA4147	35	male	no	no	0.250	4.600000	275.0	1.06	259.0	...	no	no	no	
3282	PA4283	79	male	no	no	0.030	4.100000	160.0	0.78	204.0	...	no	no	no	
3328	PA4329	29	male	no	no	0.020	5.700000	172.0	0.86	200.0	...	no	no	no	
3375	PA4376	43	male	no	no	0.005	3.000000	130.0	0.84	155.0	...	no	no	no	
3437	PA4438	77	male	no	no	0.015	10.599999	226.0	0.65	349.0	...	no	no	no	
3451	PA4452	59	male	no	no	NaN	3.100000	198.0	0.59	334.0	...	no	no	no	
3494	PA4495	60	male	no	no	0.200	4.000000	68.0	1.00	67.0	...	no	no	no	
4111	PA5112	57	male	no	no	0.250	4.200000	236.0	0.70	337.0	...	no	no	no	
4121	PA5122	79	male	no	no	NaN	3.100000	162.0	0.78	209.0	...	no	no	no	

33 rows × 24 columns

In [19]:
```
disease.loc[ (disease.target == 'low_risk') & (disease.gender == 'female')]
```

Out[19]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
1	PA1002	48	female	no	no	1.5	2.5	101.0	0.97	104.0	...	no	no	yes	
3	PA1004	42	female	no	no	1.2	2.5	106.0	0.98	108.0	...	no	no	no	
4	PA1005	38	female	no	no	0.6	1.9	95.0	NaN	NaN	...	no	no	no	
6	PA1007	90	female	no	no	1.5	1.8	98.0	0.94	104.0	...	no	no	no	
8	PA1009	33	female	no	no	0.1	1.7	104.0	0.80	130.0	...	no	no	no	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4240	PA5241	32	female	no	no	1.6	1.7	125.0	0.96	130.0	...	no	no	no	
4243	PA5244	58	female	no	no	1.7	2.4	133.0	1.15	116.0	...	no	no	no	
4244	PA5245	37	female	no	no	1.0	2.3	89.0	0.94	95.0	...	no	no	no	
4247	PA5248	72	female	no	no	3.4	2.1	88.0	0.96	92.0	...	no	no	no	
4248	PA5249	94	female	no	no	3.9	NaN	157.0	1.02	154.0	...	no	no	no	

2324 rows × 24 columns

In [20]:
```
disease.loc[ (disease.target == 'moderate_risk') & (disease.gender == 'female')]
```

Out[20]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
7	PA1008	69	female	no	no	6.9	NaN	109.0	1.03	106.0	...	no	no	no	
13	PA1014	71	female	no	no	6.7	1.9	91.0	1.03	89.0	...	no	no	no	
15	PA1016	77	female	no	no	6.2	1.2	89.0	0.74	121.0	...	no	no	no	
18	PA1019	37	female	no	no	12.0	2.4	84.0	1.24	68.0	...	no	no	no	
94	PA1095	37	female	no	no	18.0	1.5	98.0	0.88	112.0	...	no	no	no	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4199	PA5200	48	female	no	no	6.6	1.5	111.0	0.86	129.0	...	no	no	no	
4206	PA5207	68	female	no	no	43.0	0.3	14.0	NaN	NaN	...	no	no	no	
4215	PA5216	61	female	no	no	109.0	1.3	44.0	0.98	45.0	...	no	no	no	
4223	PA5224	71	female	no	no	18.0	1.9	97.0	0.93	104.0	...	no	no	no	
4246	PA5247	46	female	no	no	35.0	1.2	16.0	0.86	19.0	...	no	no	no	

359 rows × 24 columns

In [21]:
```
disease.loc[ (disease.test_X1 < 1) & (disease.gender == 'female')]
```

Out[21]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
4	PA1005	38	female	no	no	0.60	1.9	95.0	NaN	NaN	...	no	no	no	
8	PA1009	33	female	no	no	0.10	1.7	104.0	0.80	130.0	...	no	no	no	
26	PA1027	74	female	yes	no	0.99	1.9	121.0	1.09	111.0	...	no	no	no	
41	PA1042	88	female	no	no	0.83	1.5	89.0	0.94	95.0	...	no	no	no	
47	PA1048	54	female	no	no	0.70	2.0	95.0	0.99	96.0	...	yes	no	no	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4220	PA5221	36	female	no	no	0.10	2.6	98.0	NaN	NaN	...	no	no	no	
4222	PA5223	59	female	no	no	0.82	1.4	91.0	0.97	94.0	...	no	no	no	
4231	PA5232	59	female	no	no	0.40	1.6	113.0	0.99	114.0	...	no	no	no	
4237	PA5238	58	female	no	no	0.10	2.0	129.0	0.95	136.0	...	no	no	no	
4238	PA5239	50	female	yes	no	0.32	1.5	138.0	1.05	131.0	...	no	no	yes	

891 rows × 24 columns

In [22]:
```
low = disease.loc[ (disease.test_X1 < 1) & (disease.gender == 'female')]
```

In [23]:
```
low.sample(50)
```

Out[23]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medic
907	PA1908	66	female	no	no	0.040	4.9	249.0	0.86	290.0	...	no	no	no	

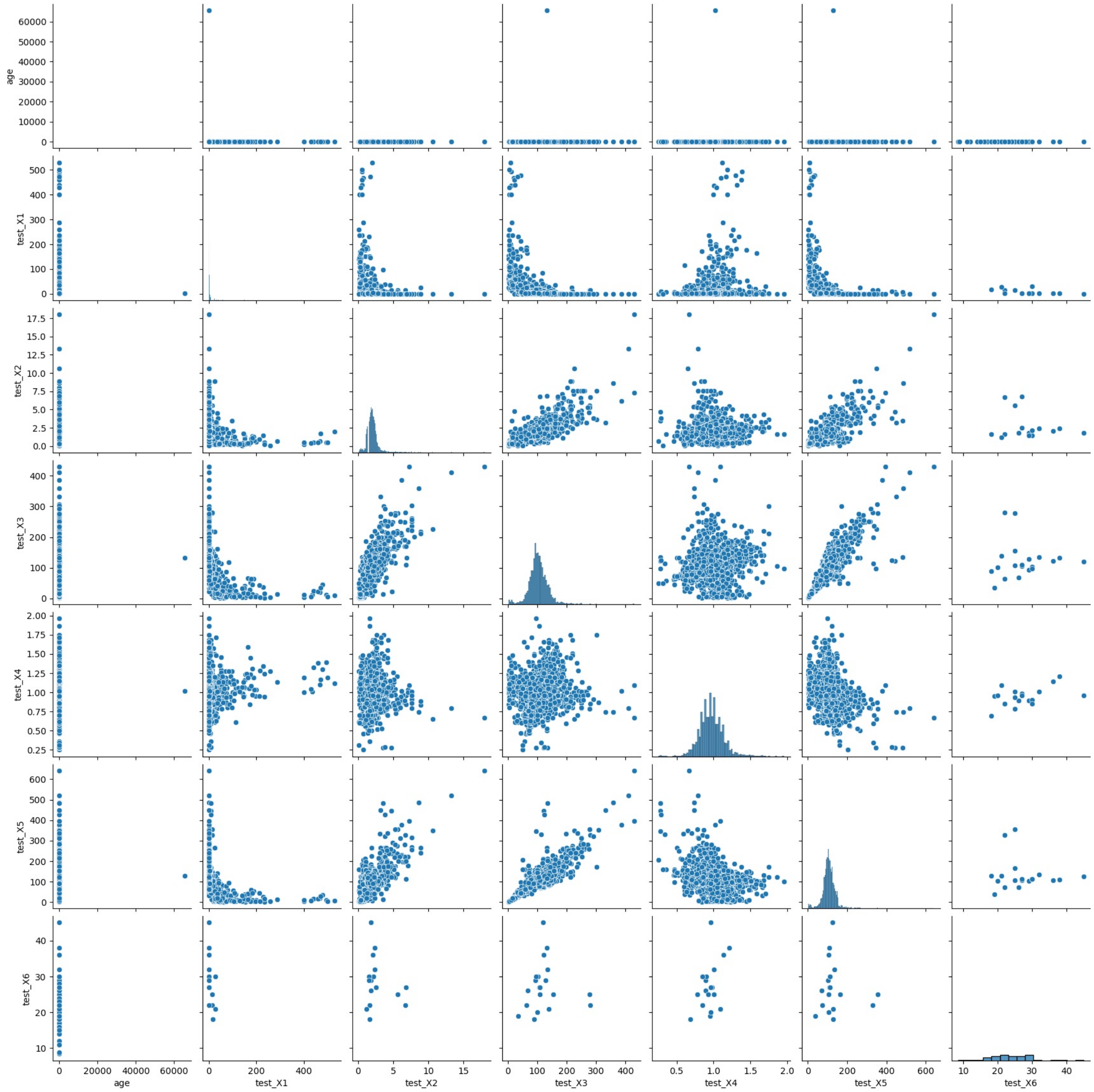
3440	PA4441	65	female	no	no	0.400	1.6	108.0	0.93	117.0	...	no	no	no
2264	PA3265	69	female	no	no	0.020	NaN	102.0	0.77	133.0	...	no	no	yes
1476	PA2477	48	female	no	no	0.450	NaN	113.0	1.05	107.0	...	no	no	no
1180	PA2181	23	female	no	no	0.250	2.6	139.0	0.90	153.0	...	no	no	no
1902	PA2903	56	female	no	no	0.300	1.6	131.0	0.95	138.0	...	no	no	no
4164	PA5165	72	female	no	no	0.015	2.9	198.0	0.91	217.0	...	no	no	no
2947	PA3948	51	female	yes	no	0.730	NaN	110.0	0.85	129.0	...	no	no	no
3206	PA4207	67	female	no	no	0.500	1.9	109.0	1.01	108.0	...	no	no	no
1653	PA2654	29	female	no	no	0.620	NaN	119.0	0.98	121.0	...	no	no	no
1362	PA2363	77	female	no	no	0.700	2.2	72.0	1.18	62.0	...	no	no	no
2663	PA3664	78	female	no	no	0.580	2.2	92.0	0.96	96.0	...	no	no	no
2593	PA3594	71	female	no	no	0.800	2.3	133.0	1.10	121.0	...	no	no	no
1516	PA2517	62	female	no	no	0.740	NaN	97.0	1.01	96.0	...	no	no	no
503	PA1504	50	female	no	no	0.200	2.0	121.0	1.04	116.0	...	no	no	no
3880	PA4881	62	female	no	no	0.530	NaN	118.0	0.90	131.0	...	no	no	no
2841	PA3842	25	female	no	no	0.100	2.8	NaN	NaN	NaN	...	no	no	no
2242	PA3243	63	female	no	no	0.200	1.1	187.0	0.95	197.0	...	no	no	no
201	PA1202	39	female	no	no	0.500	2.2	121.0	1.01	121.0	...	no	no	no
1050	PA2051	32	female	no	no	0.900	2.3	129.0	1.29	100.0	...	no	no	no
2321	PA3322	56	female	no	no	0.050	2.5	152.0	1.16	131.0	...	no	no	no
3114	PA4115	49	female	no	no	0.050	NaN	193.0	1.05	184.0	...	no	no	no
4238	PA5239	50	female	yes	no	0.320	1.5	138.0	1.05	131.0	...	no	no	yes
984	PA1985	24	female	no	no	0.110	3.8	96.0	0.81	119.0	...	no	no	no
4167	PA5168	61	female	no	no	0.250	1.8	102.0	0.76	135.0	...	no	no	no
2301	PA3302	55	female	no	no	0.900	2.3	117.0	0.95	123.0	...	yes	no	no
3829	PA4830	47	female	no	no	0.010	4.2	214.0	1.45	148.0	...	no	no	yes
4203	PA5204	80	female	no	no	0.700	1.7	119.0	0.87	137.0	...	no	no	no
630	PA1631	31	female	no	yes	0.440	1.5	75.0	0.92	81.0	...	no	no	yes
1379	PA2380	64	female	no	no	0.800	1.2	129.0	1.03	125.0	...	no	no	yes
3093	PA4094	36	female	no	no	0.590	NaN	115.0	1.20	96.0	...	no	no	no
1262	PA2263	47	female	no	no	0.300	2.1	114.0	0.90	126.0	...	no	no	no
1128	PA2129	54	female	no	no	0.700	2.1	98.0	NaN	NaN	...	no	no	no
3542	PA4543	54	female	no	no	0.150	NaN	127.0	0.89	143.0	...	no	no	yes
3335	PA4336	72	female	no	no	0.400	1.3	91.0	1.02	89.0	...	no	no	no
310	PA1311	38	female	no	no	0.590	2.3	99.0	0.96	103.0	...	no	no	no
706	PA1707	56	female	no	no	0.030	6.4	191.0	0.97	197.0	...	no	no	no
1375	PA2376	60	female	no	no	0.100	NaN	150.0	0.99	151.0	...	no	no	yes
2721	PA3722	66	female	no	no	0.065	2.0	153.0	0.99	154.0	...	no	no	no
2210	PA3211	55	female	no	no	0.035	2.0	109.0	0.93	117.0	...	no	no	no
2158	PA3159	76	female	no	no	0.450	NaN	184.0	1.21	152.0	...	no	no	no
3207	PA4208	41	female	no	no	0.400	NaN	NaN	NaN	NaN	...	no	no	no
972	PA1973	57	female	no	no	0.100	6.6	190.0	0.86	222.0	...	no	no	no
2309	PA3310	65	female	no	no	0.100	NaN	NaN	NaN	NaN	...	no	no	no
3592	PA4593	56	female	no	no	0.400	1.8	139.0	0.97	143.0	...	no	no	yes
1244	PA2245	57	female	no	no	0.055	1.7	75.0	0.81	93.0	...	no	no	no
2098	PA3099	29	female	no	no	0.680	2.0	142.0	1.05	135.0	...	no	no	no
3937	PA4938	49	female	no	no	0.100	2.0	99.0	0.89	111.0	...	no	no	no
208	PA1209	38	female	no	no	0.560	2.9	158.0	0.75	211.0	...	no	no	yes
892	PA1893	49	female	no	no	0.100	3.1	139.0	1.14	122.0	...	no	no	no

50 rows × 24 columns

In [24]:
```python
sns.pairplot(disease,kind="scatter")
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x17dc0ad7d90>



In [25]:
```python
CAT_COLUMNS = [ 'gender', 'sick', 'pregnant',  'concern_type1',
       'concern_type2', 'enlargement', 'tumor', 'disorder', 'medication_A',
       'medication_B', 'mental_health', 'mood_stabiliser', 'surgery',
       'treatment_type1', 'suspect']
```

```
NUM_COLUMNS = [ 'age',  'test_X1', 'test_X2',
       'test_X3', 'test_X4', 'test_X5', 'test_X6' ]
```

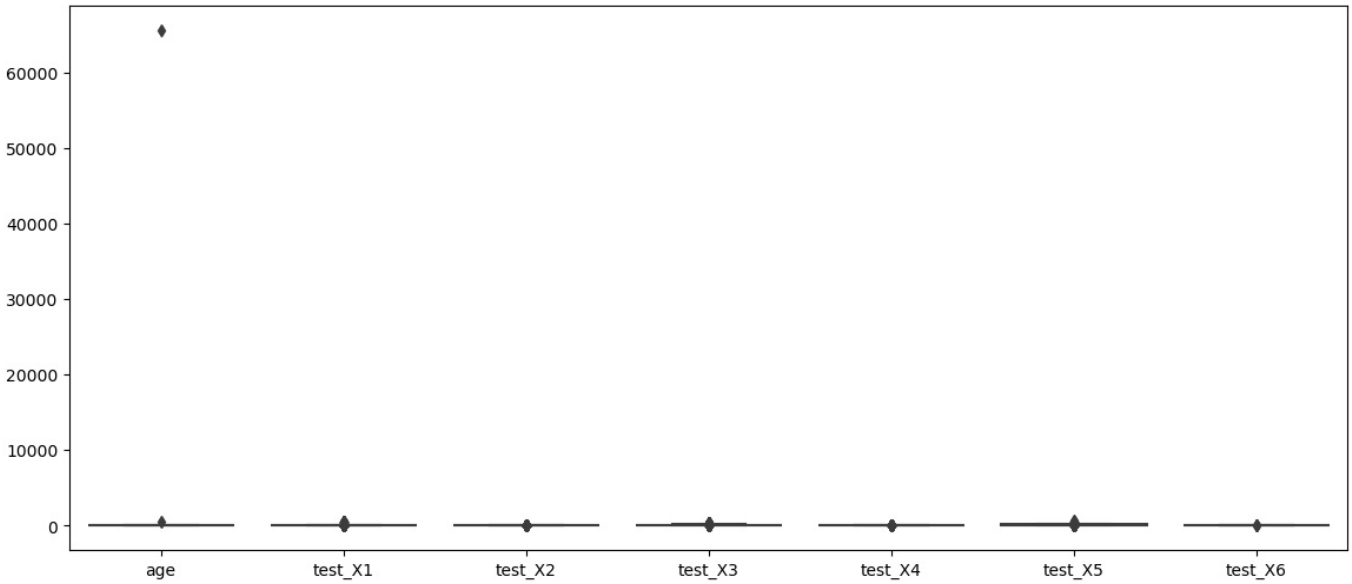
In [26]:
```
#Let us run loop of value_counts of each column to find out unique values.
for col in disease[CAT_COLUMNS]:
    print ("---- %s ---" % col)
    print (disease[col].value_counts())
```

```
---- gender ---
gender
female    2787
male      1322
Name: count, dtype: int64
---- sick ---
sick
no     4095
yes     155
Name: count, dtype: int64
---- pregnant ---
pregnant
no     4235
yes      15
Name: count, dtype: int64
---- concern_type1 ---
concern_type1
no     4183
yes      67
Name: count, dtype: int64
---- concern_type2 ---
concern_type2
no     3905
yes     345
Name: count, dtype: int64
---- enlargement ---
enlargement
no     4215
yes      35
Name: count, dtype: int64
---- tumor ---
tumor
no     4139
yes     111
Name: count, dtype: int64
---- disorder ---
disorder
no    4250
Name: count, dtype: int64
---- medication_A ---
medication_A
no     3760
yes     490
Name: count, dtype: int64
---- medication_B ---
medication_B
no     4196
yes      54
Name: count, dtype: int64
---- mental_health ---
mental_health
no     4056
yes     194
Name: count, dtype: int64
---- mood_stabiliser ---
mood_stabiliser
no     4205
yes      45
Name: count, dtype: int64
---- surgery ---
surgery
no     4188
yes      62
Name: count, dtype: int64
---- treatment_type1 ---
treatment_type1
no     4169
yes      81
Name: count, dtype: int64
---- suspect ---
suspect
no     3951
yes     299
Name: count, dtype: int64
```

In [27]:
```
plt.figure(figsize = (14,6))
```

```
plt.title = ("Average")
plt.show
sns.boxplot(data = disease)
```

Out[27]: <Axes: >



In [28]:
```
# replace all NA's with 0
Filled0_disease=disease.fillna(0)
Filled0_disease.describe()
```

Out[28]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000
mean	67.374824	6.632404	1.440233	99.587238	0.881300	100.066092	0.845224
std	1004.518821	31.114110	1.207047	41.558330	0.320777	49.455649	4.474787
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	37.000000	0.282500	0.000000	83.000000	0.840000	86.000000	0.000000
50%	55.000000	1.300000	1.700000	101.000000	0.950000	104.000000	0.000000
75%	67.000000	2.700000	2.100000	120.000000	1.050000	123.000000	0.000000
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

In [29]:
```
# replace all NA's with the value that comes directly after it in the same column,
# then replace all the remaining na's with 0
FilledNext_disease= disease.fillna(method='bfill', axis=0).fillna(0)
FilledNext_disease.describe()
```

```
C:\Users\student\AppData\Local\Temp\ipykernel_18164\2101240759.py:3: FutureWarning: DataFrame.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  FilledNext_disease= disease.fillna(method='bfill', axis=0).fillna(0)
```

Out[29]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000
mean	67.374824	7.188836	2.045280	104.848108	0.970624	110.086704	23.043576
std	1004.518821	31.602570	0.927492	35.556326	0.162934	40.433803	5.406017
min	1.000000	0.005000	0.050000	2.000000	0.250000	1.400000	0.000000
25%	37.000000	0.600000	1.600000	87.000000	0.870000	92.000000	20.000000
50%	55.000000	1.500000	1.900000	102.000000	0.960000	107.000000	22.000000
75%	67.000000	3.000000	2.300000	121.000000	1.060000	124.000000	27.000000
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

In [30]:
```
#for categorical
from sklearn.base import TransformerMixin

class DataFrameImputer(TransformerMixin):

    def __init__(self):
        """Impute missing values.

        Columns of dtype object are imputed with the most frequent value
```

```
        in column.

        Columns of other types are imputed with mean of column.

        """
    def fit(self, x, y=None):
        # Find most common value with value_counts() which returns
        # counts in descending order so that the first element is the most frequently-occurring element.
        self.fill = pd.Series([x[c].value_counts().index[0]
            #Use that if type is object otherwise use mean
            if x[c].dtype == np.dtype('O') else x[c].mean() for c in x],
            index=x.columns)

        return self

    def transform(self, x, y=None):
        return x.fillna(self.fill)


#Define the data to be filled as X, we can pass the whole data frame and apply our new class
x = disease
xt = DataFrameImputer().fit_transform(x)

print('before...')
#Let us see missing value for the train data before
missing_val_count_by_column = (x.isnull().sum())
print('Missing columns for the Train data:\n',missing_val_count_by_column[missing_val_count_by_column>0])

#and after
print('after...')
missing_val_count_by_column = (xt.isnull().sum())
print('Missing columns for the Train data:\n',missing_val_count_by_column[missing_val_count_by_column>0])
```

```
before...
Missing columns for the Train data:
 gender      141
test_X1     411
test_X2    1243
test_X3     216
test_X4     392
test_X5     387
test_X6    4096
dtype: int64
after...
Missing columns for the Train data:
 Series([], dtype: int64)
```

In [31]:
```
xt.describe()
```

Out[31]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000	4250.000000
mean	67.374824	7.342463	2.035580	104.919623	0.970846	110.090834	23.325974
std	1004.518821	31.038321	0.774158	34.582253	0.154798	37.980106	1.008955
min	1.000000	0.005000	0.050000	2.000000	0.250000	1.400000	8.400000
25%	37.000000	0.700000	1.700000	88.000000	0.880000	94.000000	23.325974
50%	55.000000	1.700000	2.035580	104.000000	0.970846	110.000000	23.325974
75%	67.000000	4.500000	2.100000	120.000000	1.050000	123.000000	23.325974
max	65526.000000	530.000000	18.000000	430.000000	1.960000	642.000000	45.000000

In [32]:
```
dise = xt
```

In [33]:
```
dise_NUM_COLUMNS = [ 'age',  'test_X1', 'test_X2',
       'test_X3', 'test_X4', 'test_X5', 'test_X6' ]
```

In [34]:
```
outlier = dise[dise_NUM_COLUMNS]
```

In [35]:
```
#import the implementation of this algorihm from sklearn
from sklearn.cluster import DBSCAN

#Use the algorithm for outlier detection, the retun in clusters will show the membership of each point
#Any point labelled as -1 is an outlier

outlier_detection = DBSCAN(min_samples = 3, eps = 10)
clusters = outlier_detection.fit_predict(outlier)

#Count total number of outliers as count of those labelled as -1
TotalOutliers=list(clusters).count(-1)
```

```
#print (clusters)
print("Total number of outliers identified is: ",TotalOutliers)
```

```
Total number of outliers identified is:  397
```

In [36]:
```
#import the implementation of this algorihm from sklearn
from sklearn.cluster import DBSCAN

#Use the algorithm for outlier detection, the retun in clusters will show the membership of each point
#Any point labelled as -1 is an outlier

outlier_detection = DBSCAN(min_samples = 5, eps = 10)
clusters = outlier_detection.fit_predict(outlier)

#Count total number of outliers as count of those labelled as -1
TotalOutliers=list(clusters).count(-1)
#print (clusters)
print("Total number of outliers identified is: ",TotalOutliers)
```

```
Total number of outliers identified is:  483
```

In [37]:
```
#import the implementation of this algorihm from sklearn
from sklearn.cluster import DBSCAN

#Use the algorithm for outlier detection, the retun in clusters will show the membership of each point
#Any point labelled as -1 is an outlier

outlier_detection = DBSCAN(min_samples = 2, eps = 10)
clusters = outlier_detection.fit_predict(outlier)

#Count total number of outliers as count of those labelled as -1
TotalOutliers=list(clusters).count(-1)
#print (clusters)
print("Total number of outliers identified is: ",TotalOutliers)
```

```
Total number of outliers identified is:  301
```

In [38]:
```
#using isolation forest
#import the implementation of this algorihm from sklearn
from sklearn.ensemble import IsolationForest

#Use the algorithm for outlier detection, then use it to predict each point
#Any point labelled as -1 is an outlier
clf = IsolationForest(max_samples=4250, random_state = 1, contamination= 0.01)
preds = clf.fit_predict(outlier)
#print(preds)
totalOutliers=0
for pred in preds:
    if pred == -1:
        totalOutliers=totalOutliers+1
print("Total number of outliers identified is: ",totalOutliers)
```

```
Total number of outliers identified is:  43
```

In [39]:
```
median_age = dise['age'].median()
```

In [40]:
```
median_age
```

Out[40]:
```
55.0
```

In [41]:
```
dise.loc[disease.age > 100]
```

Out[41]:

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	mec
2459	PA3460	65526	female	no	no	1.5	2.03558	132.0	1.02	129.0	...	no	no	no	
3477	PA4478	455	female	no	no	1.1	2.00000	118.0	1.13	104.0	...	no	no	no	

2 rows × 24 columns

In [42]:
```
disea = dise.replace(to_replace=[455, 65526], value=55.0)
```

In [43]:
```
disea.loc[disea.age > 100]
```

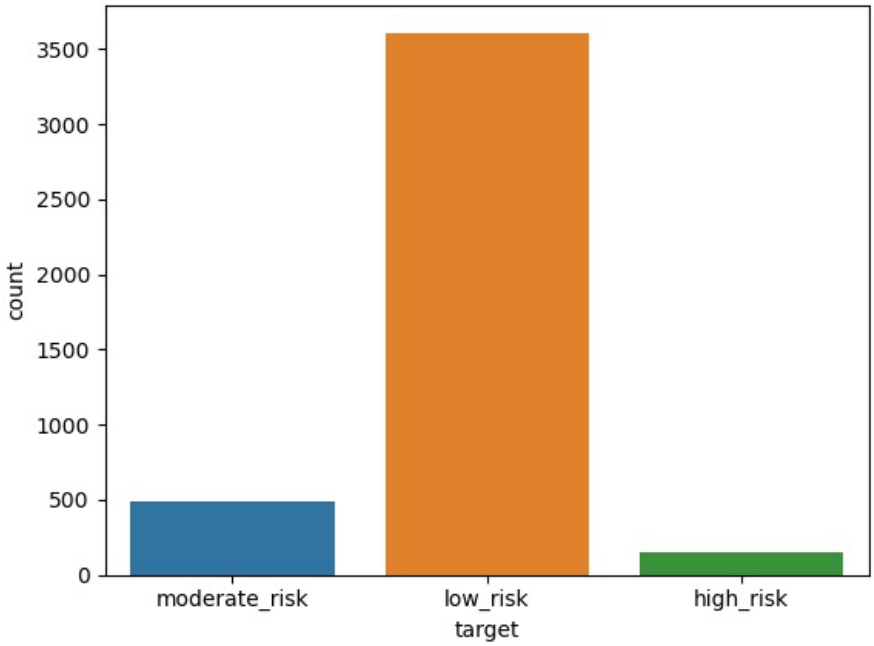
Out[43]:

id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medication_B	r

0 rows × 24 columns

In [44]:
```
#Show distribution of the class on whole dataset
sns.countplot(x= 'target', data=disea)
```

Out[44]: <Axes: xlabel='target', ylabel='count'>



In [45]:
```
disea.target.describe()
```

Out[45]:
```
count         4250
unique           3
top       low_risk
freq          3612
Name: target, dtype: object
```

In [46]:
```
disea.target.value_counts()
```

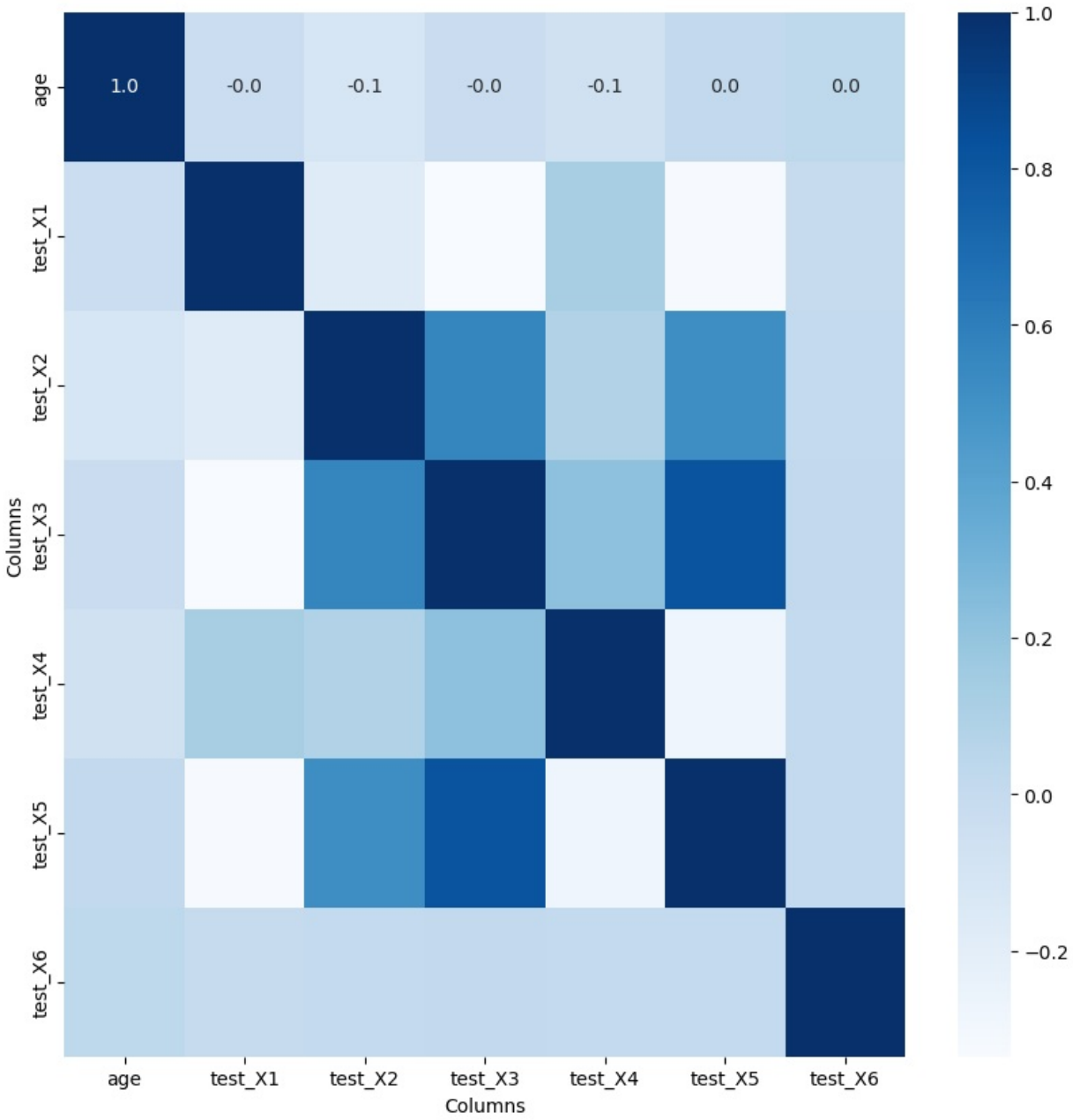
Out[46]:
```
target
low_risk         3612
moderate_risk     489
high_risk         149
Name: count, dtype: int64
```

In [47]:
```
correlation = disea[NUM_COLUMNS].corr()
correlation
```

Out[47]:

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
age	1.000000	-0.033273	-0.117303	-0.024380	-0.069996	0.009717	0.031025
test_X1	-0.033273	1.000000	-0.176346	-0.335067	0.127406	-0.325578	-0.003336
test_X2	-0.117303	-0.176346	1.000000	0.563615	0.084432	0.519876	0.006198
test_X3	-0.024380	-0.335067	0.563615	1.000000	0.218210	0.820681	0.011062
test_X4	-0.069996	0.127406	0.084432	0.218210	1.000000	-0.277262	0.007746
test_X5	0.009717	-0.325578	0.519876	0.820681	-0.277262	1.000000	0.005095
test_X6	0.031025	-0.003336	0.006198	0.011062	0.007746	0.005095	1.000000

In [48]:
```
# Create a heatmap of the correlation matrix
#we can plot it
plt.figure(figsize=(10, 10))
sns.heatmap(correlation, annot=True, fmt=".1f", cmap="Blues", xticklabels=disea[NUM_COLUMNS].columns, yticklabe
plt.xlabel("Columns")
plt.ylabel("Columns")
plt.show()
```



In [49]:
```
Disease_low_risk = disea.loc[(disea.target == 'low_risk')]
Disease_moderate_risk = disea.loc[(disea.target == 'moderate_risk')]
Disease_high_risk = disea.loc[(disea.target == 'high_risk')]
```

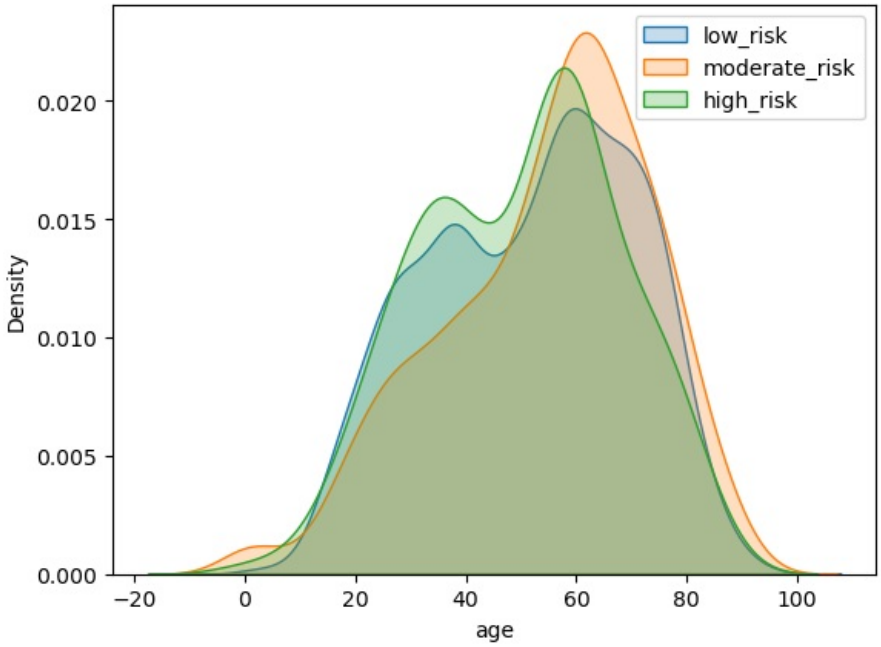
In [50]:
```
def KDEPlots(df1,df2,df3,label1,label2,label3,column):
    """ Make KDE plots by superimposing KDE plots for df1, df2 and df3"""
    #make sure that plot merges the two plots below by using the following command
    plt.figure()
    # produce a plot for each label so they interlink
    sns.kdeplot(data=df1,label=label1, shade=True)
    sns.kdeplot(data=df2,label=label2,shade=True)
    sns.kdeplot(data=df3,label=label3,shade=True)
     #include a legend
    plt.legend()
    #include a dynamic title

    return

for column in NUM_COLUMNS:
    KDEPlots(Disease_low_risk[column], Disease_moderate_risk[column], Disease_high_risk[column],'low_risk', 'mo
```

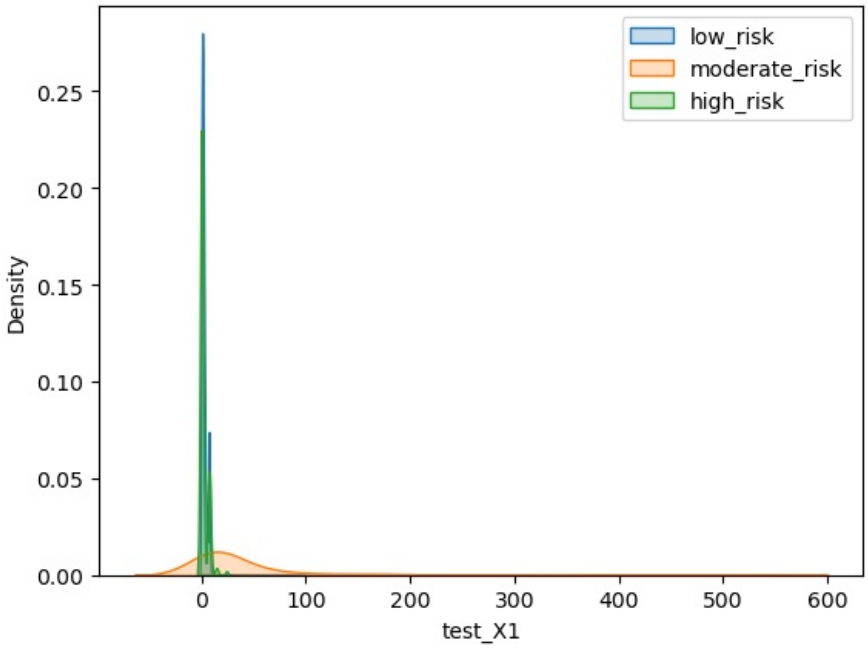
```
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:
```

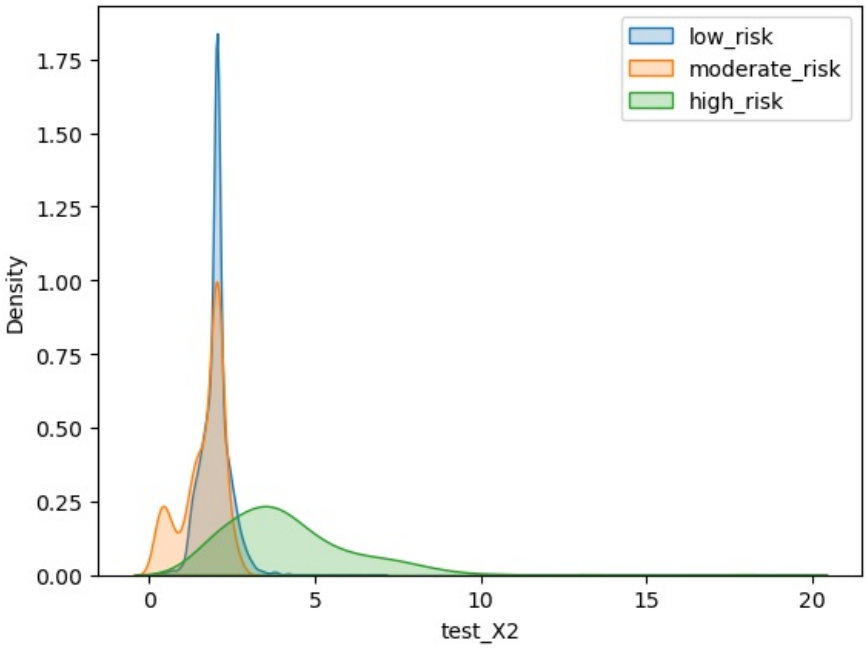
```
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

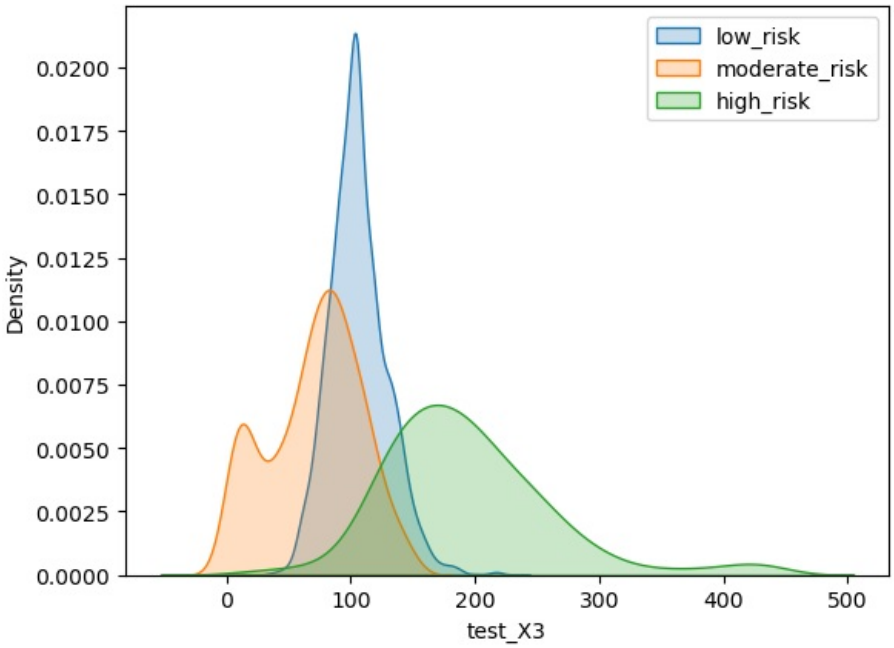
```
  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df1,label=label1, shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df2,label=label2,shade=True)
```

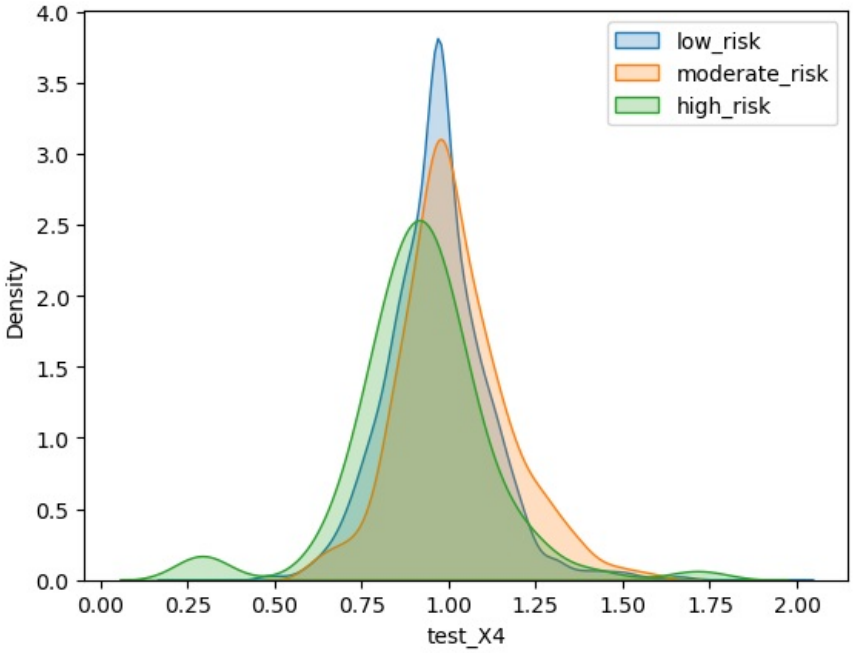
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\3459052310.py:8: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=df3,label=label3,shade=True)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

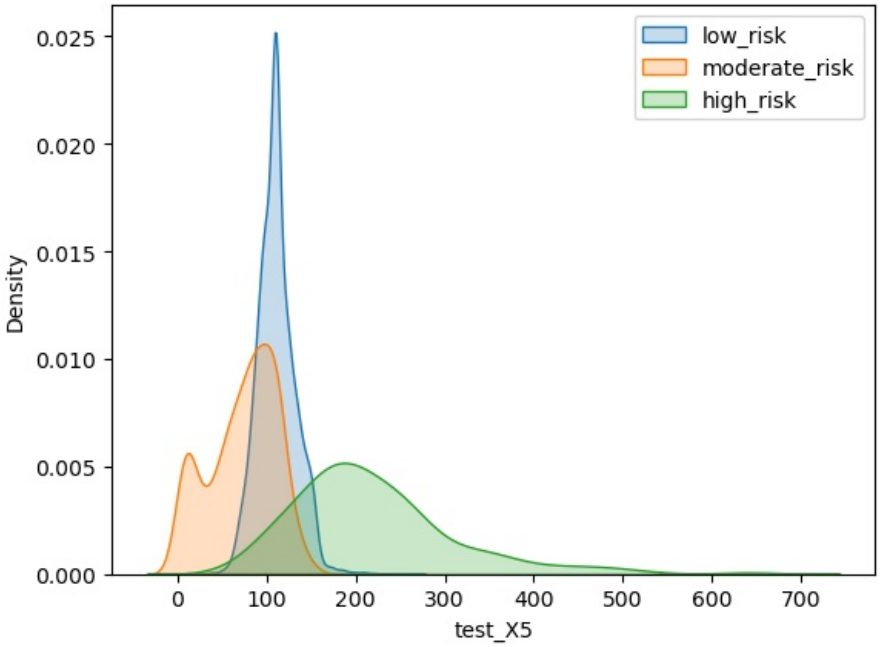


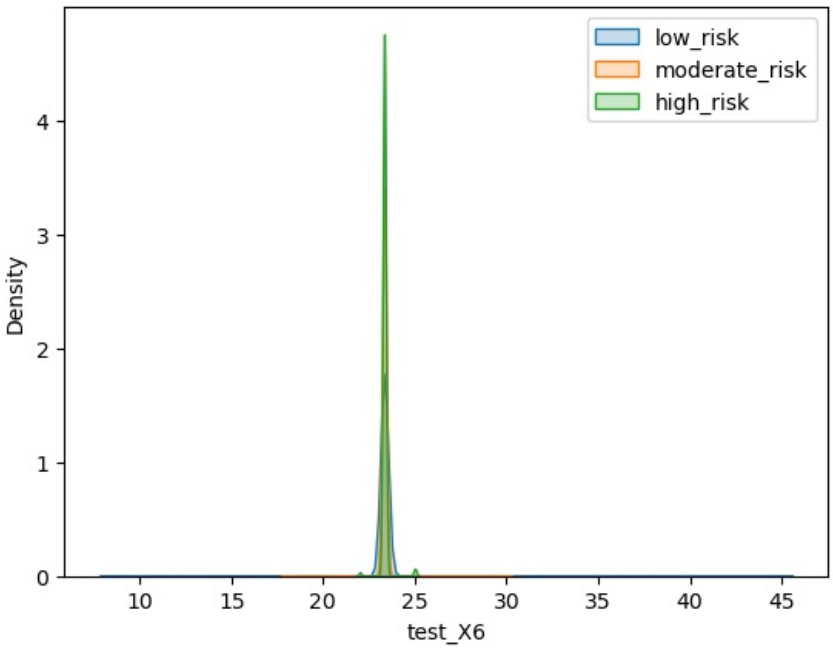



low_risk
moderate_risk
high_risk
Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
test_X2


low_risk
moderate_risk
high_risk
Density
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
100
200
300
400
500
test_X3


low_risk
moderate_risk
high_risk
Density
test_X4
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


low_risk
moderate_risk
high_risk
Density
test_X5
0
100
200
300
400
500
600
700
0.000
0.005
0.010
0.015
0.020
0.025



In [51]:
```
#dropping test x6
diseases = disea.drop(['test_X6','test_X5' ,'id'], axis=1)
```

In [52]:
```
diseases.head()
```

Out[52]:

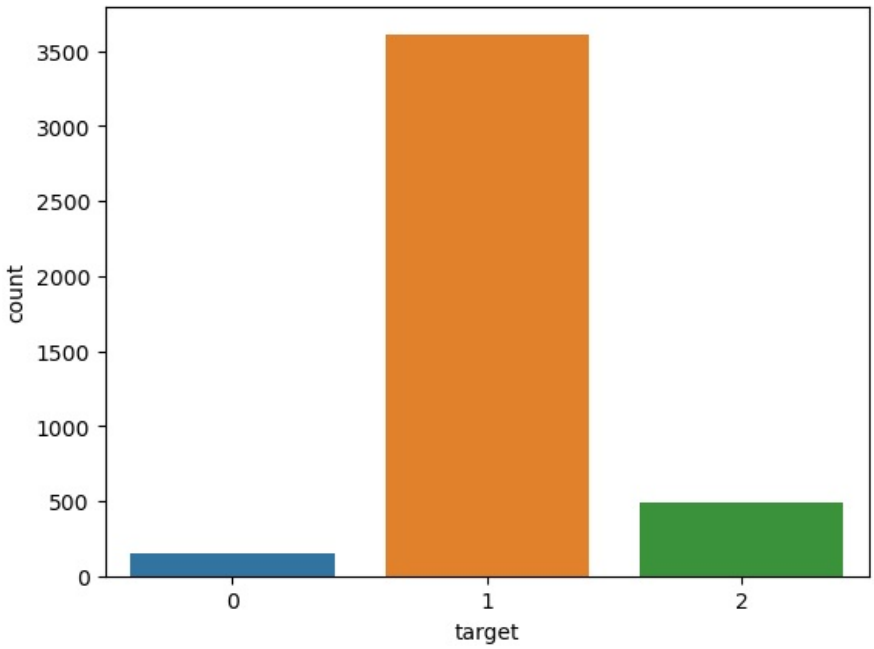
	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	concern_type1	concern_type2	...	tumor	disorder	medicatic
0	59	male	no	no	7.8	2.03558	89.0	0.850000	no	yes	...	no	no	
1	48	female	no	no	1.5	2.50000	101.0	0.970000	no	no	...	no	no	
2	77	male	no	no	7.3	1.20000	57.0	1.280000	no	no	...	no	no	
3	42	female	no	no	1.2	2.50000	106.0	0.980000	no	no	...	no	no	
4	38	female	no	no	0.6	1.90000	95.0	0.970846	no	no	...	no	no	

5 rows × 21 columns

In [53]:
```
from sklearn.preprocessing import LabelEncoder
#label_encoder object knows how to understand word labels.
label_encoder = LabelEncoder()

# Encode labels in column 'species'.
diseases['target']= label_encoder.fit_transform(diseases['target'])
target_mapping =dict(zip(label_encoder.classes_,label_encoder.transform(label_encoder.classes_)))
print('target',target_mapping)
sns.countplot(x='target',data=diseases)
```

```
target {'high_risk': 0, 'low_risk': 1, 'moderate_risk': 2}
```

Out[53]:
```
<Axes: xlabel='target', ylabel='count'>
```



In [54]:
```
# Assuming 'disease_new' is a DataFrame and 'target' is the name of the column to be excluded
X = diseases.drop(columns ='target')
```

In [55]:
```
X.columns
```

Out[55]:
```
Index(['age', 'gender', 'sick', 'pregnant', 'test_X1', 'test_X2', 'test_X3',
       'test_X4', 'concern_type1', 'concern_type2', 'enlargement', 'tumor',
       'disorder', 'medication_A', 'medication_B', 'mental_health',
       'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect'],
      dtype='object')
```

In [56]:
```
columns_to_encode = ['gender', 'sick', 'pregnant', 'concern_type1',
                     'concern_type2', 'enlargement', 'tumor', 'disorder',
                     'medication_A', 'medication_B', 'mental_health',
                     'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect']
```

In [57]:
```
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
#enc = OneHotEncoder(handle_unknown='ignore')
# Apply one-hot encoding to the categorical columns
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(X[columns_to_encode])
#Create a DataFrame with the one-hot encoded columns
#We use get_feature_names_out() to get the column names for the encoded data
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(columns_to_encode))
# Concatenate the one-hot encoded dataframe with the original dataframe
X_encoded = pd.concat([X, one_hot_df], axis=1)

# Drop the original categorical columns
df_encoded = X_encoded.drop(columns_to_encode, axis=1)

# Display the resulting dataframe
print(f"Encoded Disease data : \n{df_encoded}")
```

```
Encoded Disease data :
      age  test_X1  test_X2  test_X3   test_X4  gender_female  gender_male  \
0      59      7.8  2.03558     89.0  0.850000            0.0          1.0
1      48      1.5  2.50000    101.0  0.970000            1.0          0.0
2      77      7.3  1.20000     57.0  1.280000            0.0          1.0
3      42      1.2  2.50000    106.0  0.980000            1.0          0.0
4      38      0.6  1.90000     95.0  0.970846            1.0          0.0
...   ...      ...      ...      ...       ...            ...          ...
4245   29      0.5  2.30000    105.0  0.860000            0.0          1.0
4246   46     35.0  1.20000     16.0  0.860000            1.0          0.0
4247   72      3.4  2.10000     88.0  0.960000            1.0          0.0
4248   94      3.9  2.03558    157.0  1.020000            1.0          0.0
4249   41      1.8  2.80000     60.0  0.950000            0.0          1.0

      sick_no  sick_yes  pregnant_no  ...  mental_health_no  \
0         1.0       0.0          1.0  ...               1.0
1         1.0       0.0          1.0  ...               1.0
2         1.0       0.0          1.0  ...               1.0
3         1.0       0.0          1.0  ...               1.0
4         1.0       0.0          1.0  ...               1.0
...       ...       ...          ...  ...               ...
4245      1.0       0.0          1.0  ...               1.0
4246      1.0       0.0          1.0  ...               1.0
4247      1.0       0.0          1.0  ...               1.0
4248      1.0       0.0          1.0  ...               1.0
4249      1.0       0.0          1.0  ...               1.0

      mental_health_yes  mood_stabiliser_no  mood_stabiliser_yes  surgery_no  \
0                   0.0                 1.0                  0.0         1.0
1                   0.0                 0.0                  1.0         1.0
2                   0.0                 1.0                  0.0         1.0
3                   0.0                 1.0                  0.0         1.0
4                   0.0                 1.0                  0.0         1.0
...                 ...                 ...                  ...         ...
4245                0.0                 1.0                  0.0         1.0
4246                0.0                 1.0                  0.0         1.0
4247                0.0                 1.0                  0.0         1.0
4248                0.0                 1.0                  0.0         1.0
4249                0.0                 1.0                  0.0         1.0

      surgery_yes  treatment_type1_no  treatment_type1_yes  suspect_no  \
0             0.0                 1.0                  0.0         1.0
1             0.0                 1.0                  0.0         1.0
2             0.0                 1.0                  0.0         1.0
3             0.0                 1.0                  0.0         1.0
4             0.0                 1.0                  0.0         1.0
...           ...                 ...                  ...         ...
4245          0.0                 1.0                  0.0         1.0
4246          0.0                 1.0                  0.0         1.0
4247          0.0                 1.0                  0.0         1.0
4248          0.0                 1.0                  0.0         1.0
4249          0.0                 1.0                  0.0         1.0

      suspect_yes
0             0.0
1             0.0
2             0.0
3             0.0
4             0.0
...           ...
4245          0.0
4246          0.0
4247          0.0
4248          0.0
4249          0.0

[4250 rows x 34 columns]
```

In [58]:
```
obj_df = df_encoded.select_dtypes(exclude=['object']).copy()
FS_Cols=obj_df.columns
FS_Cols
```

Out[58]:
```
Index(['age', 'test_X1', 'test_X2', 'test_X3', 'test_X4', 'gender_female',
       'gender_male', 'sick_no', 'sick_yes', 'pregnant_no', 'pregnant_yes',
       'concern_type1_no', 'concern_type1_yes', 'concern_type2_no',
       'concern_type2_yes', 'enlargement_no', 'enlargement_yes', 'tumor_no',
       'tumor_yes', 'disorder_no', 'medication_A_no', 'medication_A_yes',
       'medication_B_no', 'medication_B_yes', 'mental_health_no',
       'mental_health_yes', 'mood_stabiliser_no', 'mood_stabiliser_yes',
       'surgery_no', 'surgery_yes', 'treatment_type1_no',
       'treatment_type1_yes', 'suspect_no', 'suspect_yes'],
      dtype='object')
```

In [59]:
```
X=df_encoded[FS_Cols]
```

```
y=diseases.target
```

In [60]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
# configure to select all features
selector = SelectKBest(score_func=mutual_info_classif, k='all')

# transform train input data
Xfs = selector.fit_transform(X,y)
```

In [61]:
```
#Retrieve the column names for the selected columns
names = X.columns.values[selector.get_support()]
#and their scores
scores = selector.scores_[selector.get_support()]
#pair the values together with the zip function
names_scores = list(zip(names, scores))
#Store the information in a data frame
ns_df = pd.DataFrame(data = names_scores, columns=['Feat_names', 'Mutual_info'])
#Sort the dataframe for better visualization
ns_df_sorted = ns_df.sort_values(['Mutual_info', 'Feat_names'], ascending = [False, True])
print(ns_df_sorted)
```

```
            Feat_names  Mutual_info
1              test_X1     0.376091
3              test_X3     0.168649
2              test_X2     0.118399
11    concern_type1_no     0.019547
13    concern_type2_no     0.016296
4              test_X4     0.015683
21    medication_A_yes     0.015088
14   concern_type2_yes     0.012415
24    mental_health_no     0.011995
20     medication_A_no     0.011592
15      enlargement_no     0.009633
0                  age     0.007251
16     enlargement_yes     0.006447
33         suspect_yes     0.004588
17            tumor_no     0.004113
9           pregnant_no     0.003377
25   mental_health_yes     0.001176
5        gender_female     0.000557
29         surgery_yes     0.000060
12   concern_type1_yes     0.000000
19         disorder_no     0.000000
6          gender_male     0.000000
22     medication_B_no     0.000000
23    medication_B_yes     0.000000
26  mood_stabiliser_no     0.000000
27 mood_stabiliser_yes     0.000000
10        pregnant_yes     0.000000
7              sick_no     0.000000
8             sick_yes     0.000000
28          surgery_no     0.000000
32          suspect_no     0.000000
30  treatment_type1_no     0.000000
31 treatment_type1_yes     0.000000
18           tumor_yes     0.000000
```

In [62]:
```
from sklearn.feature_selection import SelectKBest, f_classif

#Specify the features to consider and the Y values.
feature_cols = X
Y=diseases['target']

#Note that currently our X values which are in ImputedX after imputation are an numpy ndarray
print(type(X))

# Keep all 12 features but rank them by best f_classif score
selector = SelectKBest(f_classif, k='all')

#Fit and apply feature ranking method
X_new = selector.fit_transform(X, Y)

#Retrieve the column names for the selected columns
names = X.columns.values[selector.get_support()]
#and their scores
scores = selector.scores_[selector.get_support()]
#pair the values together with the zip function
names_scores = list(zip(names, scores))
#Store the information in a data frame
ns_df = pd.DataFrame(data = names_scores, columns=['Feat_names', 'F_Scores'])
#Sort the dataframe for better visualization
ns_df_sorted = ns_df.sort_values(['F_Scores', 'Feat_names'], ascending = [False, True])
print(ns_df_sorted)
```

```
#Look at shape of new data
X_new.shape
```

```
<class 'pandas.core.frame.DataFrame'>
             Feat_names     F_Scores
3               test_X3  1117.123617
2               test_X2  1035.252842
1               test_X1   544.254844
14    concern_type2_yes   128.211490
13     concern_type2_no   128.211490
4               test_X4    30.017901
21     medication_A_yes    22.534140
20      medication_A_no    22.534140
33          suspect_yes    19.881031
32           suspect_no    19.881031
6           gender_male    12.836200
5         gender_female    12.836200
24     mental_health_no    10.674245
25    mental_health_yes    10.674245
0                   age     5.894048
17            tumor_no     5.285783
18           tumor_yes     5.285783
16      enlargement_yes     3.119127
15       enlargement_no     3.119127
28           surgery_no     2.702888
29          surgery_yes     2.702888
23     medication_B_yes     2.684701
22      medication_B_no     2.684701
7               sick_no     1.824980
8              sick_yes     1.824980
10         pregnant_yes     1.141960
9           pregnant_no     1.141960
27  mood_stabiliser_yes     0.870550
26   mood_stabiliser_no     0.870550
12    concern_type1_yes     0.593090
11     concern_type1_no     0.593090
31  treatment_type1_yes     0.457011
30   treatment_type1_no     0.457011
19          disorder_no          NaN
```

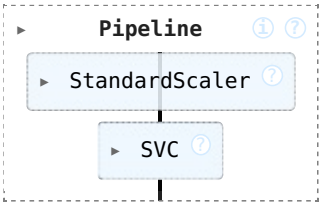
```
C:\Users\student\AppData\Roaming\Python\Python311\site-packages\sklearn\feature_selection\_univariate_selection.
py:112: UserWarning: Features [19] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
C:\Users\student\AppData\Roaming\Python\Python311\site-packages\sklearn\feature_selection\_univariate_selection.
py:113: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
```

Out[62]: (4250, 34)

In [63]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=41, stratify=y)
```

In [64]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

process = [("transformer", StandardScaler()), ("clf", SVC())]
pipe = Pipeline(process)

pipe.fit(X_train, y=y_train)
```

Out[64]:



In [65]:
```
y_pred = pipe.predict(X_test)
```

In [66]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

import warnings
```

In [67]:
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average = 'weighted')
recall = recall_score(y_test, y_pred, average = 'weighted')
f1 = f1_score(y_test, y_pred, average = 'weighted')
```

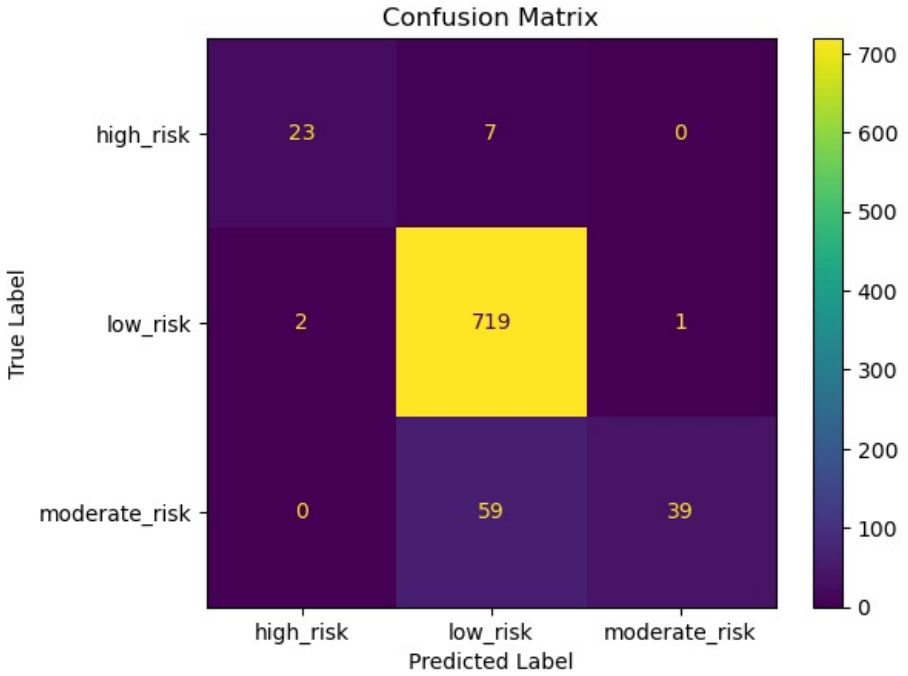
```
# Display the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.9188
Precision: 0.9229
Recall: 0.9188
F1 Score: 0.9052
```

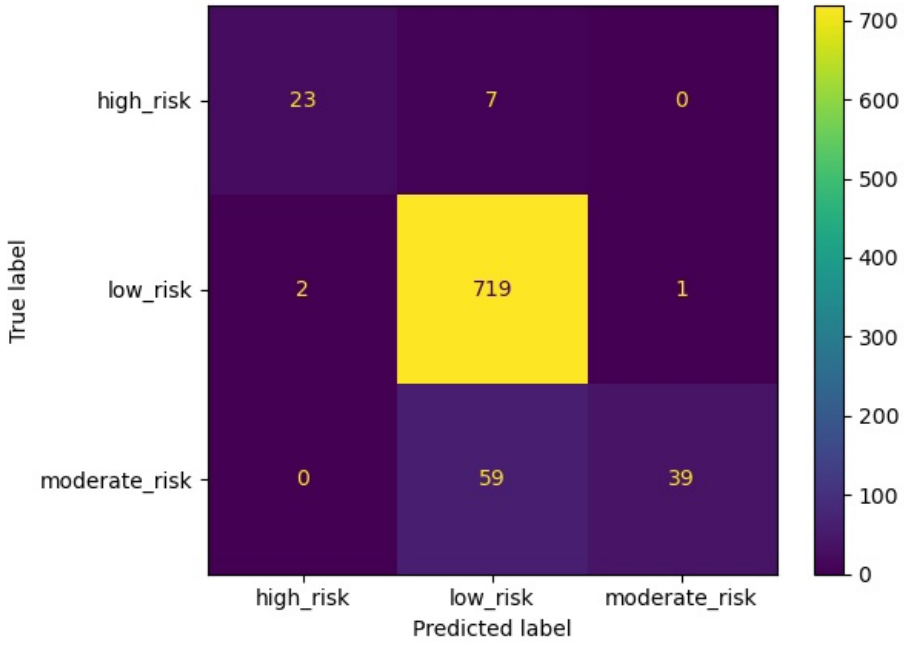
In [68]:
```
 #Generate the classification report
report = classification_report(y_test, y_pred)

# Display the classification report
print("Classification Report:\n", report)
```

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

           0       0.92      0.77      0.84        30
           1       0.92      1.00      0.95       722
           2       0.97      0.40      0.57        98

    accuracy                           0.92       850
   macro avg       0.94      0.72      0.79       850
weighted avg       0.92      0.92      0.91       850
```

In [69]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [70]:
```
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score

y_pred = pipe.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print('accuracy', acc)

bacc = balanced_accuracy_score(y_test, y_pred)
print('balanced accuracy', bacc)

f1 = f1_score(y_test, y_pred,average = 'weighted')
print('f1-score', f1)
```

```
accuracy 0.9188235294117647
balanced accuracy 0.7201569085621321
f1-score 0.9052052739390093
```

In [71]:
```
pipe.score(X_train, y_train)
acc = round(pipe.score(X_train, y_train) * 100, 2)
print("Pipleine accuracy (train set):", acc, '%')

pipe.score(X_test, y_test)
acc = round(pipe.score(X_test, y_test) * 100, 2)
print("Pipeline accuracy (test set):", acc, '%')
```

```
Pipleine accuracy (train set): 92.15 %
Pipeline accuracy (test set): 91.88 %
```

In [72]:
```
# trying another model
from sklearn.tree import DecisionTreeClassifier, plot_tree
clf_tree = DecisionTreeClassifier(criterion='entropy')
```

In [73]:
```
#normalizing the dataset
from sklearn.preprocessing import Normalizer

normalizer = Normalizer(norm='l2')
X_train = normalizer.fit_transform(X_train)
X_test = normalizer.transform(X_test)
```

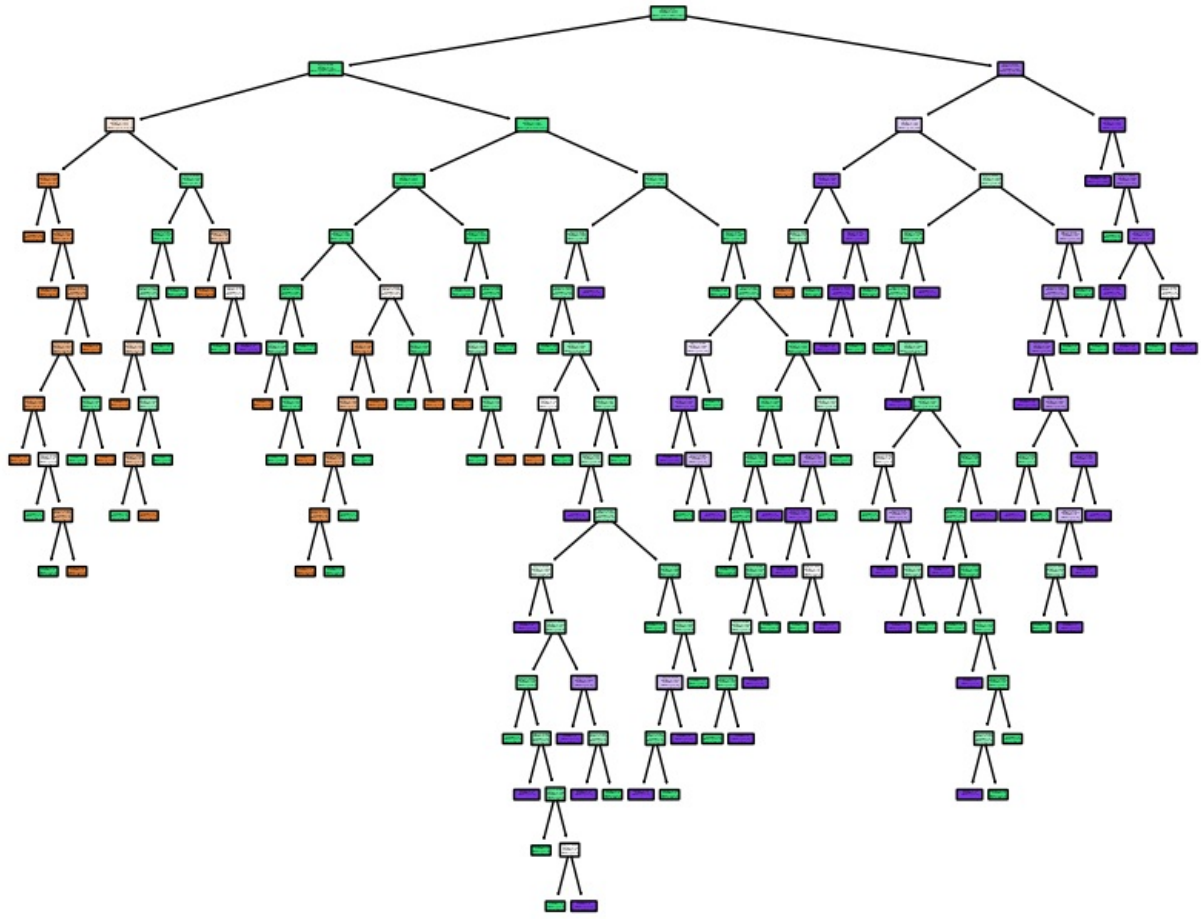
In [74]:
```
# Fit the model
clf_tree.fit(X_train, y_train)
```

Out[74]:

DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy')

In [75]:
```
from sklearn import tree

plt.figure(figsize=(10,8))
tree_plot = tree.plot_tree(clf_tree, class_names=label_encoder.classes_, filled = True, rounded=True);
```



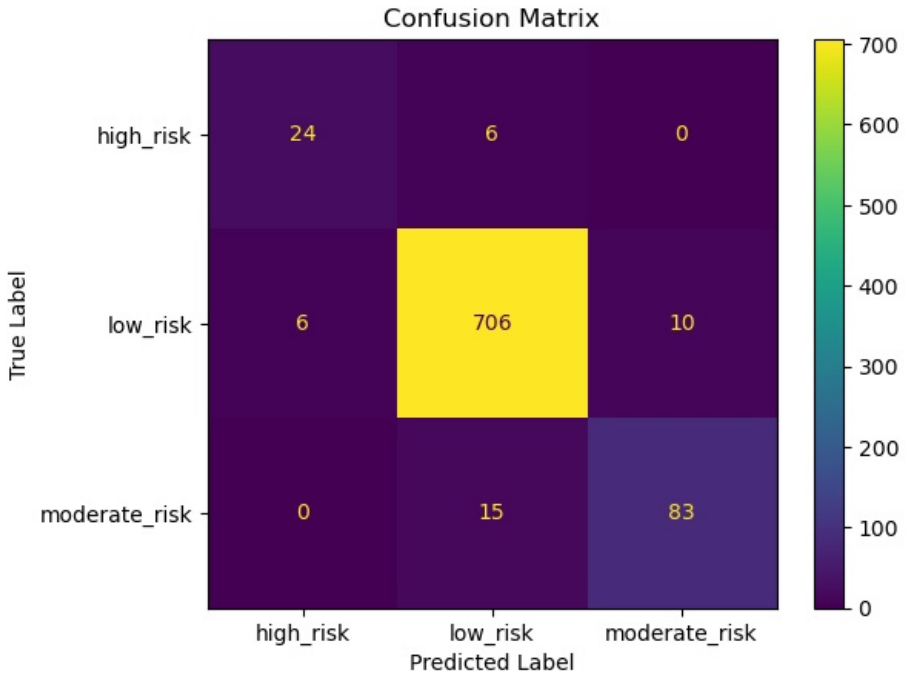
In [76]:
```
y_hat = clf_tree.predict(X_test)
y_hat
```

Out[76]:
```
array([1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1,
       2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1,
       1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1,
       2, 1, 2, 0, 1, 1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1,
       2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2, 2, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 1, 0, 1, 1, 2, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 2, 2,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 2, 0, 1, 2, 1, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 2, 1,
       1, 1, 0, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 2])
```

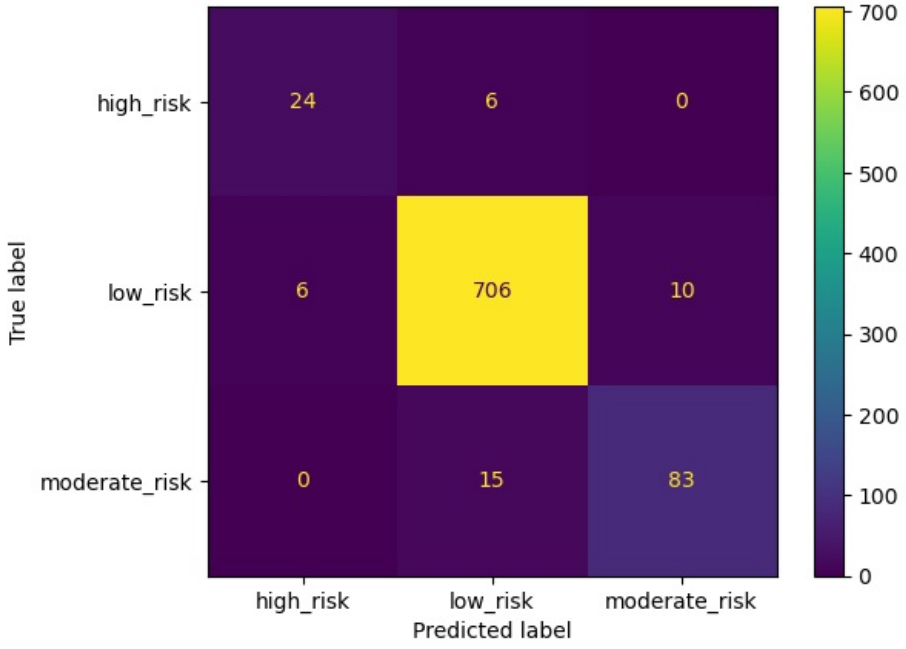
In [77]:
```
y_hat_proba = clf_tree.predict_proba(X_test)
y_hat_proba
```

Out[77]:
```
array([[0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       ...,
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [78]:
```
#we can use the built in accuracy metrics in sklearn.

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,accuracy_score
cm = confusion_matrix(y_test, y_hat)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [79]:
```
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score

#y_pred = pipe.predict(X1_test)

acc = accuracy_score(y_test, y_hat)
print('accuracy', acc)

bacc = balanced_accuracy_score(y_test, y_hat)
print('balanced accuracy', bacc)

f1 = f1_score(y_test, y_pred,average = 'weighted')
print('f1-score', f1)
```

```
accuracy 0.9564705882352941
balanced accuracy 0.8749260368967532
f1-score 0.9052052739390093
```

In [80]:
```
#Generate the classification report
report = classification_report(y_test, y_pred)

# Display the classification report
print("Classification Report:\n", report)
```

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

           0       0.92      0.77      0.84        30
           1       0.92      1.00      0.95       722
           2       0.97      0.40      0.57        98

    accuracy                           0.92       850
   macro avg       0.94      0.72      0.79       850
weighted avg       0.92      0.92      0.91       850
```

In [81]:
```
clf_tree.score(X_train, y_train)
acc = round(clf_tree.score(X_train, y_train) * 100, 2)
print("Decision tree accuracy (train set):", acc, '%')
```

```
Decision tree accuracy (train set): 100.0 %
```

In [82]:
```
#he accuracy for the training set is much better than the accuracy for the test set.

#This means we have higher variance than bias. The model cannot predict as well as the training set.
clf_tree.score(X_test, y_test)
acc = round(clf_tree.score(X_test, y_test) * 100, 2)
print("Decision tree accuracy (test set):", acc, '%')
```

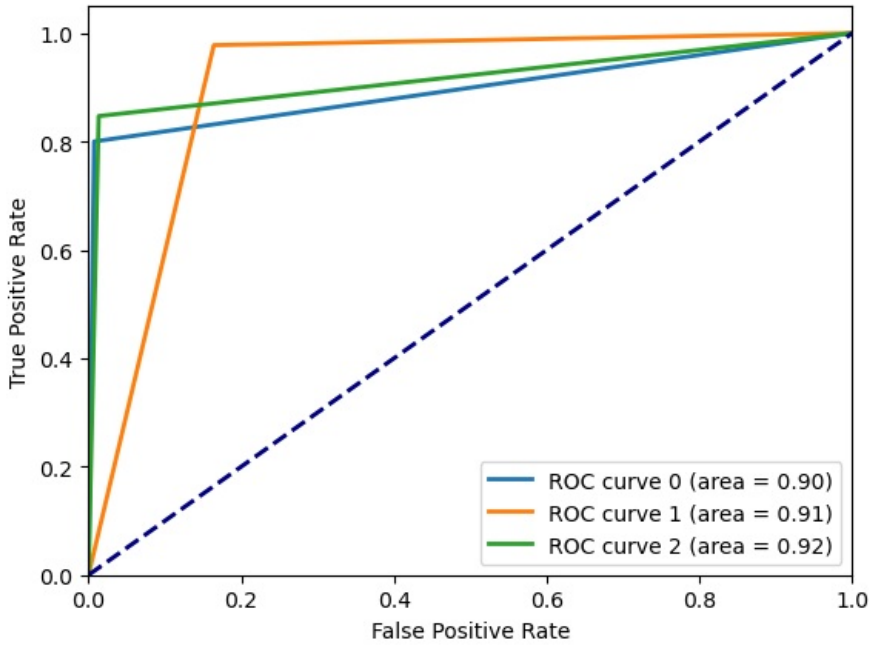
```
Decision tree accuracy (test set): 95.65 %
```

In [83]:
```
from sklearn import metrics

clf_tree.score(X_train, y_train)
y_hat = clf_tree.predict(X_test)
print(metrics.classification_report(y_test, y_hat, target_names=['high_risk', 'low_risk', 'moderate_risk']))
```

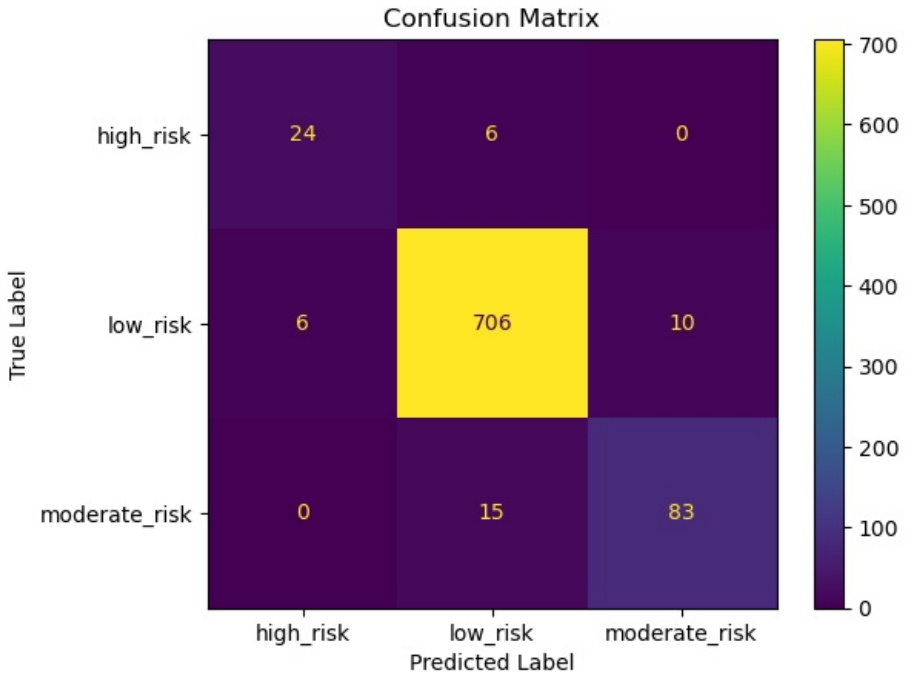
```
              precision    recall  f1-score   support

   high_risk       0.80      0.80      0.80        30
    low_risk       0.97      0.98      0.97       722
moderate_risk       0.89      0.85      0.87        98

    accuracy                           0.96       850
   macro avg       0.89      0.87      0.88       850
weighted avg       0.96      0.96      0.96       850
```

In [84]:
```
# Learn to predict each class against the other

from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_auc_score, roc_curve, auc

label_binarizer = LabelBinarizer().fit(y_train)

y_onehot_test = label_binarizer.transform(y_test)

y_onehot_test.shape  # (n_samples, n_classes)


clf_tree.score(X_train, y_train)

y_score = clf_tree.predict_proba(X_test)

y_pred = clf_tree.predict(X_test)


# Compute ROC curve and ROC area for each class

classes = np.unique(y)
n_classes = len(np.unique(y))


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

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_onehot_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

plt.figure()
lw = 2


#plot each roc curve in the onevsrest classifier.
```

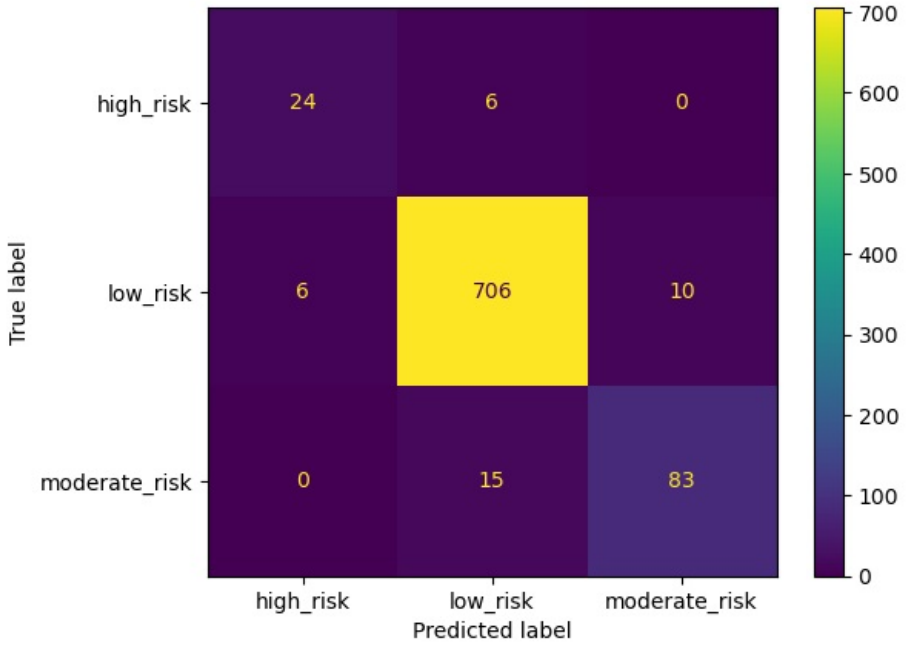
```
#class setosa vs rest, class versicolor vs rest and class virginica vs rest
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             lw=lw, label='ROC curve %s (area = %0.2f)' % (classes[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")
plt.show()
```



In [85]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_hat)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```



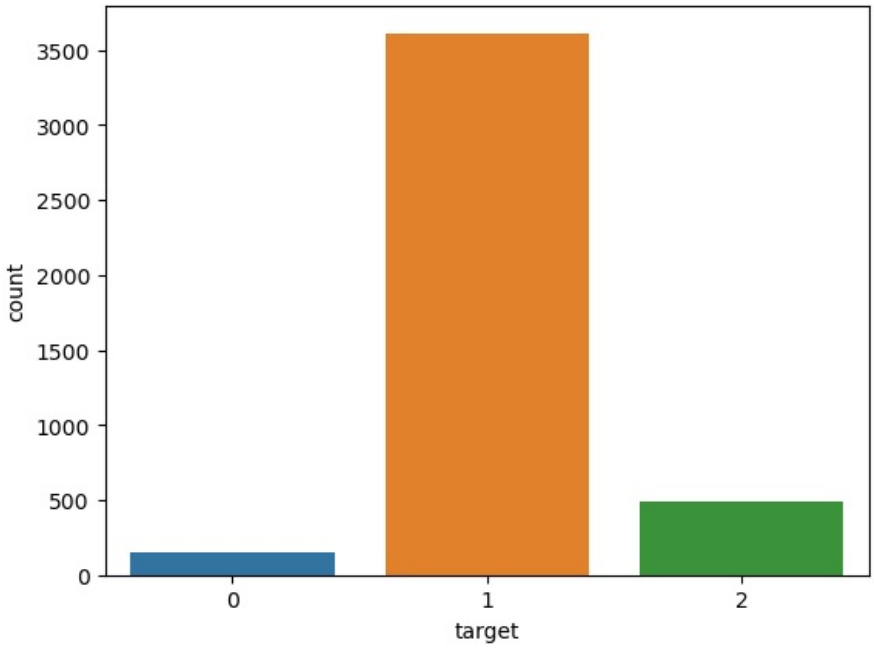

```
<Figure size 1000x1000 with 0 Axes>
```



Balancing of the target variable

In [86]:
```
#Show distribution of the class on whole dataset
sns.countplot(x= 'target', data=diseases)
```

Out[86]: <Axes: xlabel='target', ylabel='count'>



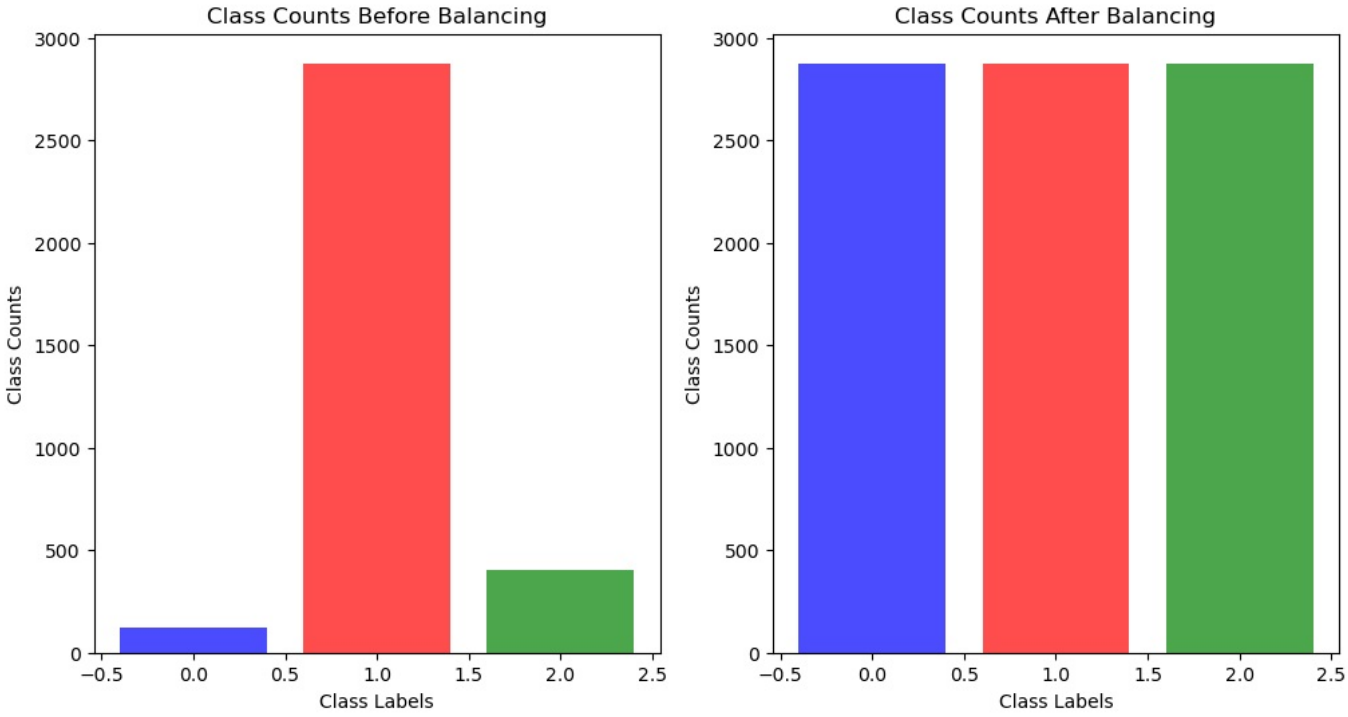
In [87]:
```
data = diseases.target
```

In [88]:
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Assuming 'X' is your feature matrix and 'y' is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, data, test_size=0.2, random_state=42)

# Count the occurrences of each class in the training set
#class_counts = y_train.value_counts()
class_counts = np.unique(y_train, return_counts=True)

# Display the class counts before balancing
print("Class Counts Before Balancing:")
print(class_counts)

# Apply SMOTE to the training set
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Count the occurrences of each class after balancing
```

```
balanced_class_counts = np.unique(y_train_balanced, return_counts=True)

# Display the class counts after balancing
print("\nClass Counts After Balancing:")
print(balanced_class_counts)
```

```
Class Counts Before Balancing:
(array([0, 1, 2]), array([ 122, 2877,  401], dtype=int64))

Class Counts After Balancing:
(array([0, 1, 2]), array([2877, 2877, 2877], dtype=int64))
```

In [89]:
```
import matplotlib.pyplot as plt

class_labels_before, counts_before = class_counts[0], class_counts[1]
class_labels_after, counts_after = balanced_class_counts[0], balanced_class_counts[1]

# Specify colors for each class
colors_before = ['blue', 'red', 'green']  # Replace with your desired colors
colors_after = ['blue', 'red', 'green']  # Replace with your desired colors

# Plotting class counts before and after balancing
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.bar(class_labels_before, counts_before, color=colors_before, alpha=0.7)
ax1.set_title('Class Counts Before Balancing')
ax1.set_xlabel('Class Labels')
ax1.set_ylabel('Class Counts')

ax2.bar(class_labels_after, counts_after, color=colors_after, alpha=0.7)
ax2.set_title('Class Counts After Balancing')
ax2.set_xlabel('Class Labels')
ax2.set_ylabel('Class Counts')

plt.show()
```



In [90]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier on the training data
knn_classifier.fit(X_train_balanced, y_train_balanced)
```

Out[90]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [91]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Ensure the arrays are contiguous in memory
X_train_balanced = np.ascontiguousarray(X_train_balanced)
X_test = np.ascontiguousarray(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=1)
```

```
knn_classifier.fit(X_train_balanced, y_train_balanced)
y_pred = knn_classifier.predict(X_test)

# Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Accuracy: 0.90
Precision: 0.91
Recall: 0.90
```

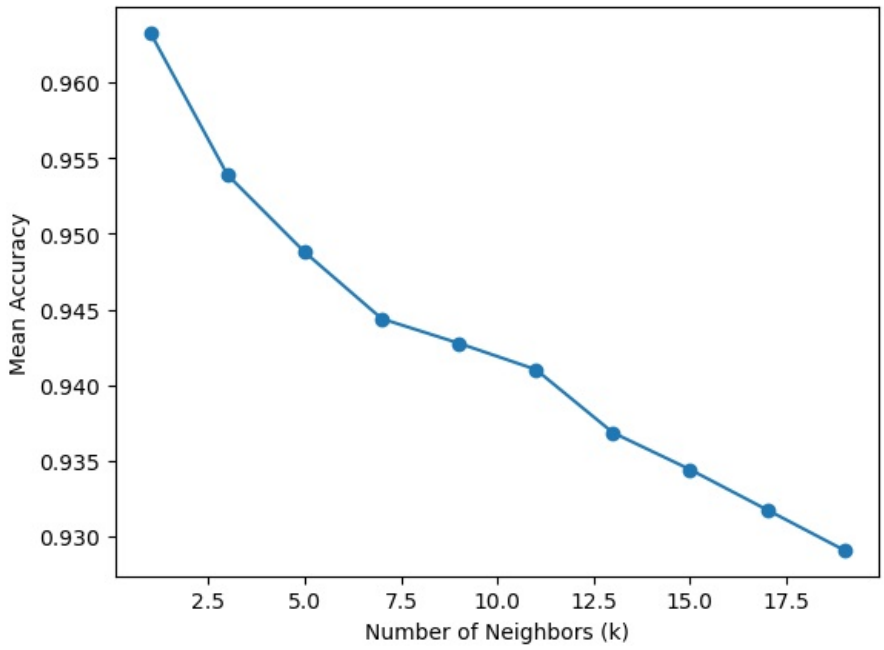
In [92]:
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.30      0.48      0.37        27
           1       0.96      0.93      0.94       735
           2       0.72      0.77      0.74        88

    accuracy                           0.90       850
   macro avg       0.66      0.73      0.68       850
weighted avg       0.91      0.90      0.90       850
```

In [93]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(knn_classifier.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(knn_classifier.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.8953
```

In [94]:
```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import numpy as np

#your training data and labels loaded in X_train_balanced and y_train_balanced

# Create a range of odd k values to try
k_values = np.arange(1, 21, 2)

# List to store mean accuracy for each k
mean_accuracies = []

# Iterate over each k value
for k in k_values:
    # Create a k-NN classifier with the current k value
    knn_classifier_cv = KNeighborsClassifier(n_neighbors=k)

    # Perform 5-fold cross-validation and get mean accuracy
    accuracies = cross_val_score(knn_classifier_cv, X_train_balanced, y_train_balanced, cv=5, scoring='accuracy
    mean_accuracy = np.mean(accuracies)

    # Append mean accuracy to the list
    mean_accuracies.append(mean_accuracy)

# Plot the mean accuracy for each k
plt.plot(k_values, mean_accuracies, marker='o')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Accuracy')

plt.show()

# Find the best k value
best_k = k_values[np.argmax(mean_accuracies)]
print(f'The best value of k is: {best_k}')
```

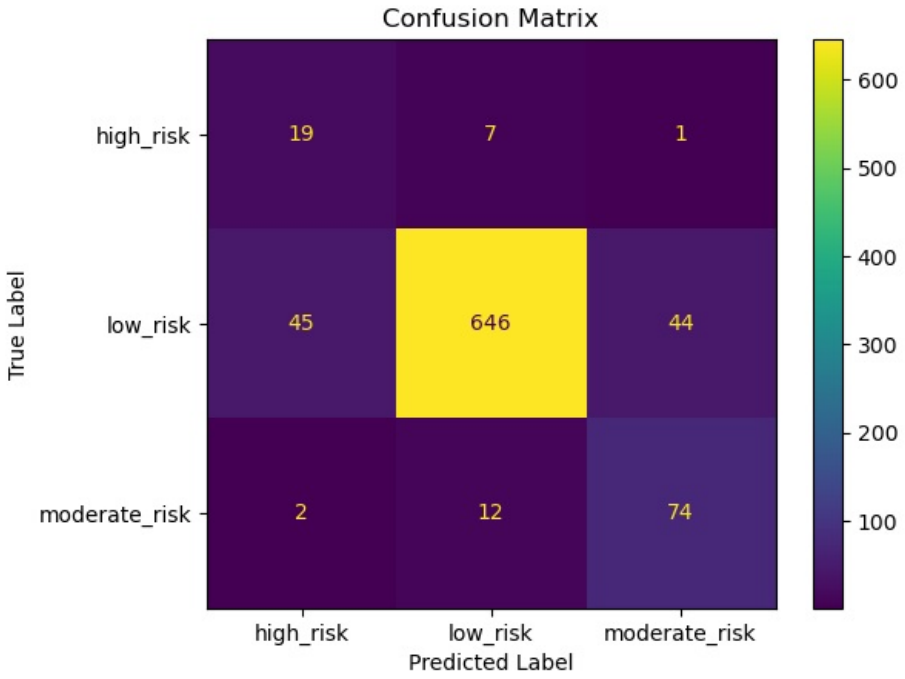



```
The best value of k is: 1
```

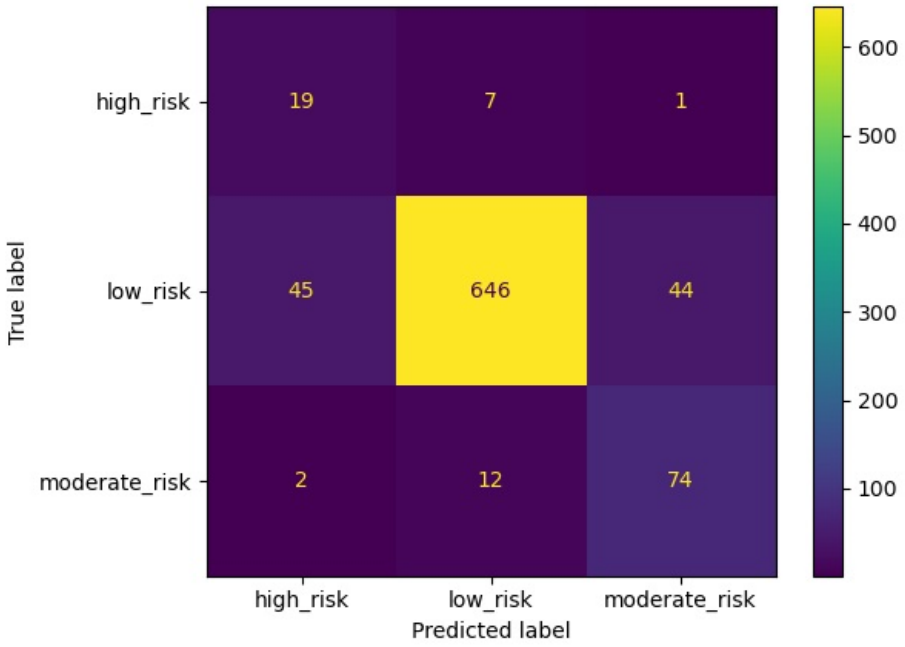
In [95]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Ensure the arrays are contiguous in memory
X_train_balanced = np.ascontiguousarray(X_train_balanced)
X_test = np.ascontiguousarray(X_test)

knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_balanced, y_train_balanced)
y_pred = knn_classifier.predict(X_test)

# Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Accuracy: 0.87
Precision: 0.91
Recall: 0.87
```

In [96]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```



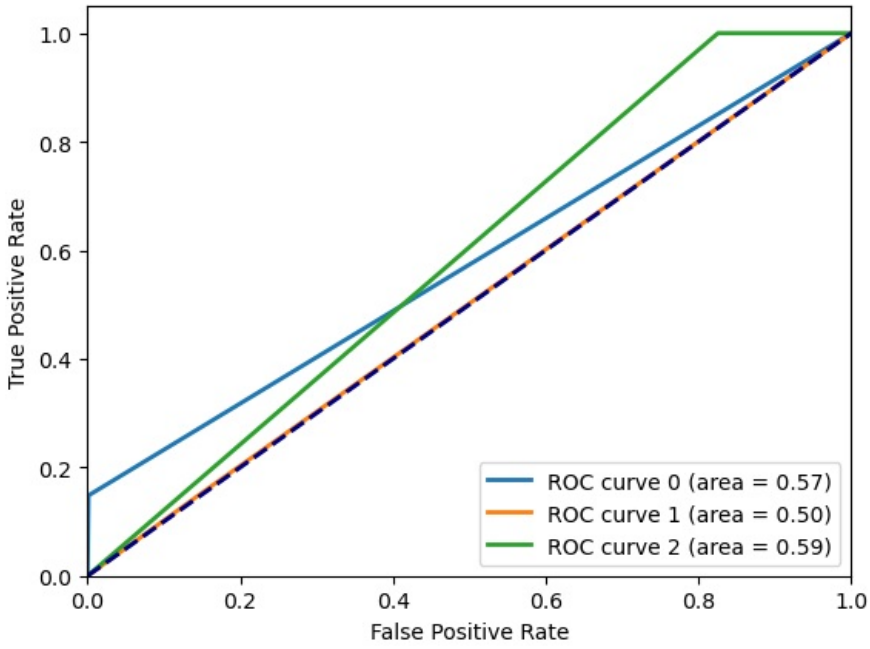

```
<Figure size 1000x1000 with 0 Axes>
```



In [97]:

```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_auc_score, roc_curve, auc

label_binarizer = LabelBinarizer().fit(y_train_balanced)

y_onehot_test = label_binarizer.transform(y_test)

y_onehot_test.shape  # (n_samples, n_classes)


clf_tree.score(X_train_balanced, y_train_balanced)

y_score = clf_tree.predict_proba(X_test)

y_pred = clf_tree.predict(X_test)


# Compute ROC curve and ROC area for each class

classes = np.unique(y)
n_classes = len(np.unique(y))


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

```
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_onehot_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

plt.figure()
lw = 2


#plot each roc curve in the onevsrest classifier.
#class setosa vs rest, class versicolor vs rest and class virginica vs rest
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             lw=lw, label='ROC curve %s (area = %0.2f)' % (classes[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")
plt.show()
```



SVC MODEL

In [98]:
```
from sklearn.svm import SVC

svc = SVC()

# Train the classifier on the training data
svc.fit(X_train_balanced, y_train_balanced)
```

Out[98]:
```
SVC()
```

In [99]:
```
# Make predictions on the test set
y_pred = svc.predict(X_test)
```

In [100…
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.24      0.81      0.38        27
           1       0.99      0.81      0.89       735
           2       0.52      0.95      0.67        88

    accuracy                           0.82       850
   macro avg       0.58      0.86      0.65       850
weighted avg       0.92      0.82      0.85       850
```

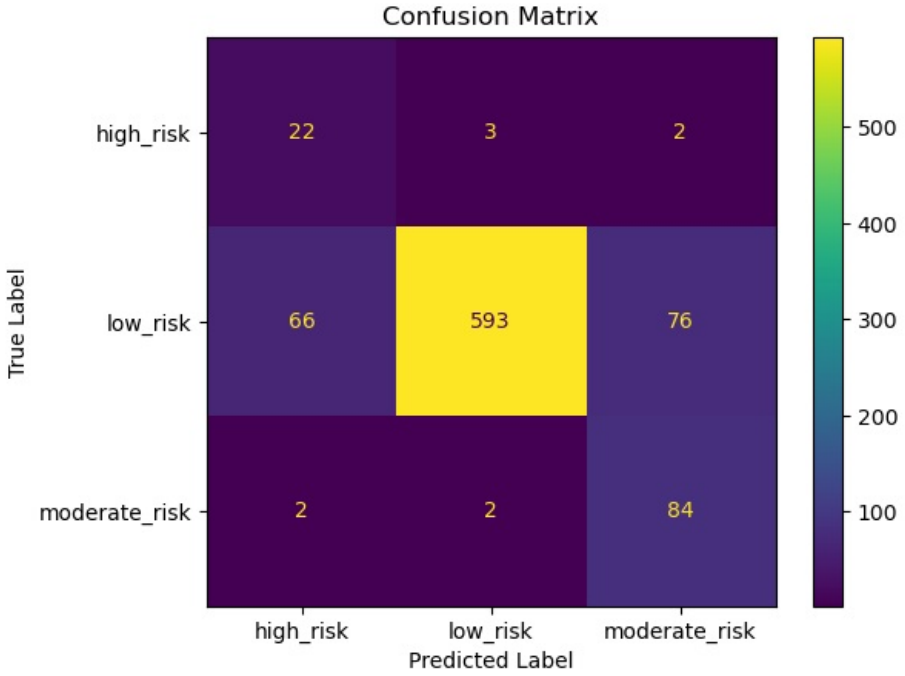
In [101…
```
# print the scores on training and test set
```

```
print('Training set score: {:.4f}'.format(svc.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

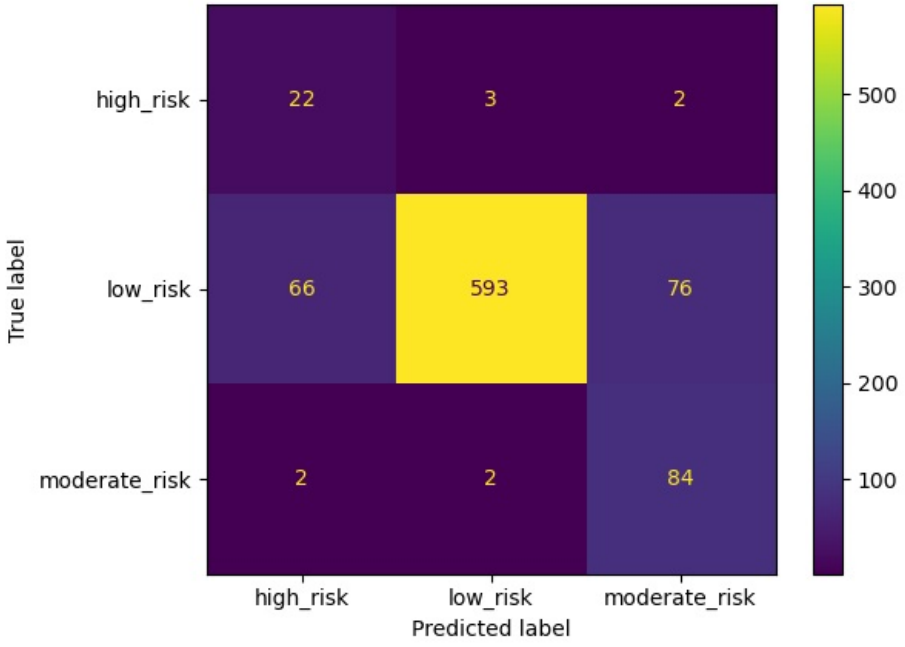
```
Training set score: 0.8666
Test set score: 0.8224
```

In [102…

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [103…

```
 #Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
#accuracy = svc.score(y_test, y_pred)
#precision = svc.score(y_test, y_pred, average='weighted')
```

```
#recall = svc.score(y_test, y_pred, average='weighted')
# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Accuracy: 0.82
Precision: 0.92
Recall: 0.82
```

In [104…
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
```

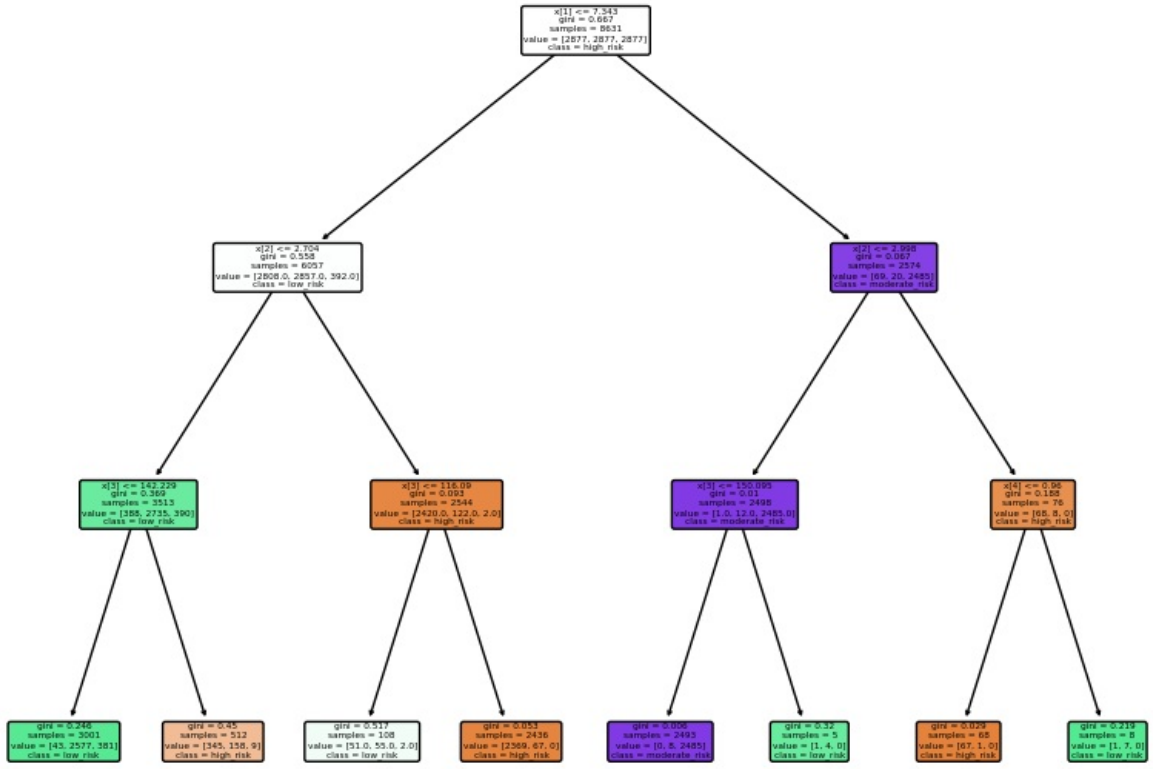
In [105…
```
clf = DecisionTreeClassifier(max_depth = 3,
                             random_state = 0)
# Step 3: Train the model on the data
clf.fit(X_train_balanced, y_train_balanced)
```

Out[105…

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [106…
```
from sklearn import tree

plt.figure(figsize=(10,8))
tree_plot = tree.plot_tree(clf, class_names=label_encoder.classes_, filled = True, rounded=True);
```



In [107…
```
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

In [108…
```
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.31      0.89      0.46        27
           1       0.97      0.92      0.95       735
           2       0.96      0.77      0.86        88

    accuracy                           0.91       850
   macro avg       0.74      0.86      0.75       850
weighted avg       0.95      0.91      0.92       850
```

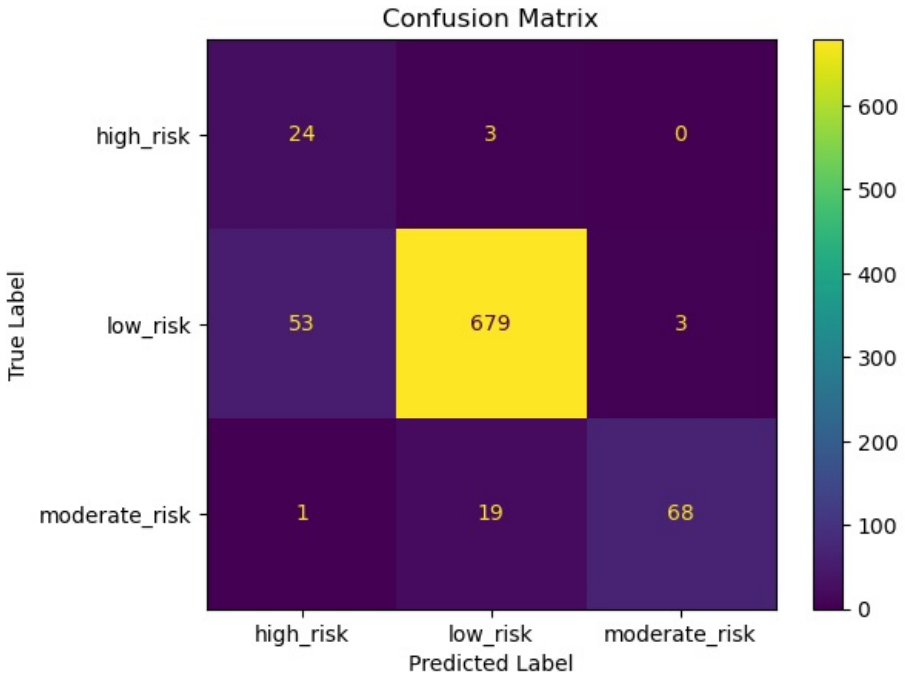
In [109…
```python
#Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Accuracy: 0.91
Precision: 0.95
Recall: 0.91
```

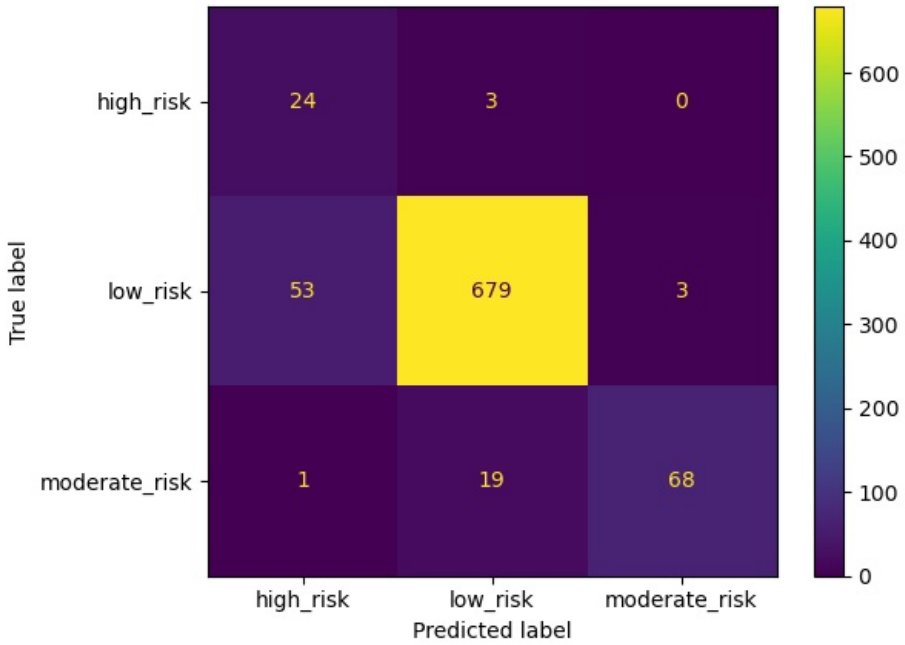
In [110…
```python
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.9163
Test set score: 0.9071
```

In [111…
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [112…
```
# Random Forests in `scikit-learn` (with N = 100)
rf = RandomForestClassifier(n_estimators=10,
                            random_state=0)
rf.fit(X_train_balanced, y_train_balanced)
```

Out[112…
```
RandomForestClassifier(n_estimators=10, random_state=0)
```

In [113…
```
rf.estimators_
```

Out[113…
```
[DecisionTreeClassifier(max_features='sqrt', random_state=209652396),
 DecisionTreeClassifier(max_features='sqrt', random_state=398764591),
 DecisionTreeClassifier(max_features='sqrt', random_state=924231285),
 DecisionTreeClassifier(max_features='sqrt', random_state=1478610112),
 DecisionTreeClassifier(max_features='sqrt', random_state=441365315),
 DecisionTreeClassifier(max_features='sqrt', random_state=1537364731),
 DecisionTreeClassifier(max_features='sqrt', random_state=192771779),
 DecisionTreeClassifier(max_features='sqrt', random_state=1491434855),
 DecisionTreeClassifier(max_features='sqrt', random_state=1819583497),
 DecisionTreeClassifier(max_features='sqrt', random_state=530702035)]
```

In [114…
```
rf.estimators_[0]
```

Out[114…
```
DecisionTreeClassifier(max_features='sqrt', random_state=209652396)
```

In [115…
```
# Make predictions
y_pred = rf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9752941176470589
```

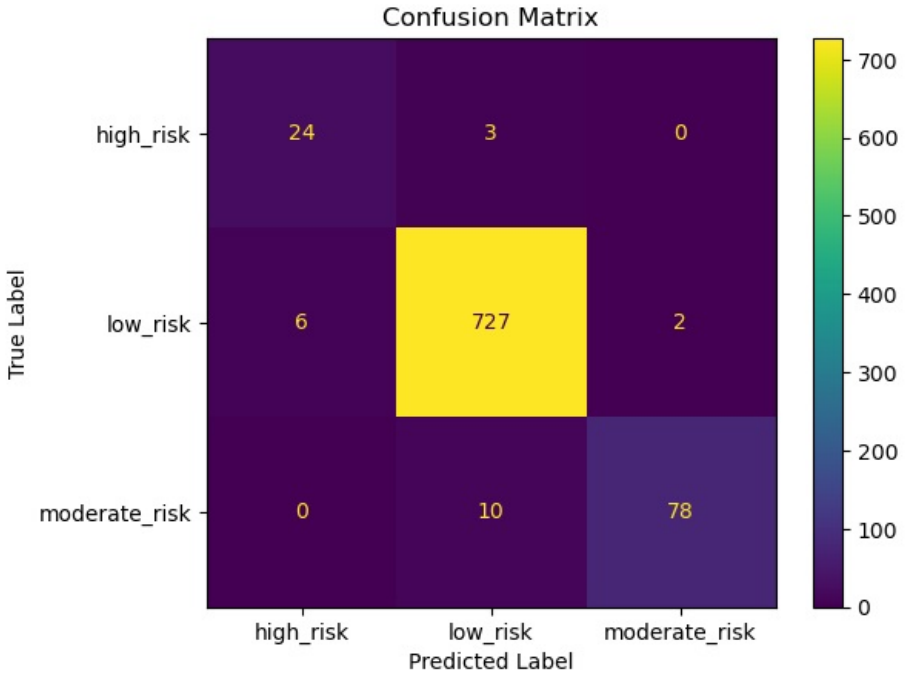
In [116…
```
# Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# print the scores on training and test set


print('Training set score: {:.4f}'.format(rf.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(rf.score(X_test, y_test)))
```

```
Training set score: 0.9994
Test set score: 0.9753
```

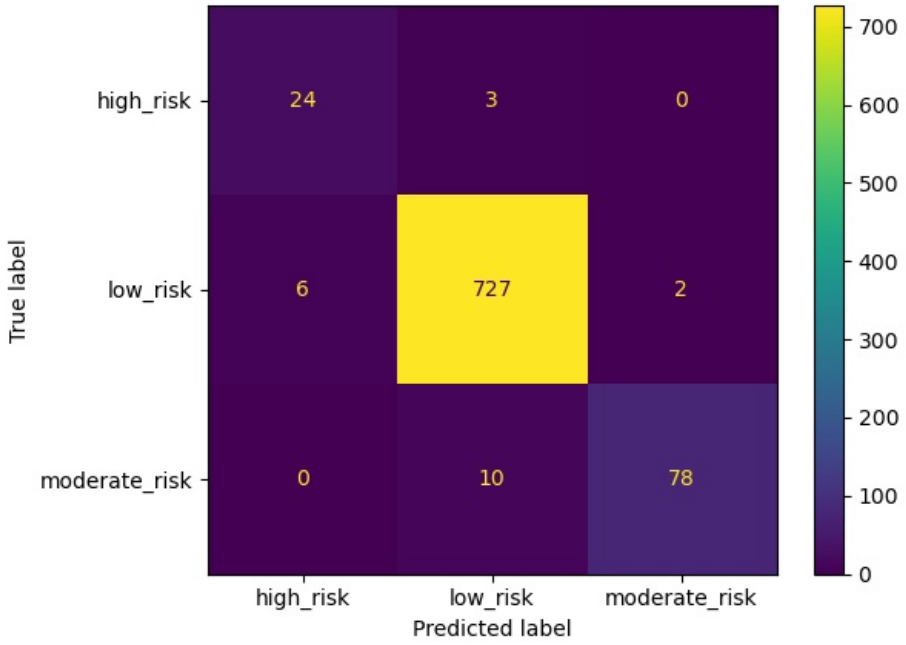
In [117…
```
# Calculate and print the scores on the training and test set
training_score = rf.score(X_train_balanced, y_train_balanced)
test_score = rf.score(X_test, y_test)

print(f'Training set score: {training_score:.4f}')
print(f'Test set score: {test_score:.4f}')
```

```
Training set score: 0.9994
Test set score: 0.9753
```

In [118…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [119…
```
# Random Forests in `scikit-learn` (with N = 100)
rf2 = RandomForestClassifier(n_estimators=100,
                             random_state=42)
rf2.fit(X_train_balanced, y_train_balanced)
```

Out[119…
```
RandomForestClassifier(random_state=42)
```

In [120…
```
rf2.estimators_
```

Out[120…
```
[DecisionTreeClassifier(max_features='sqrt', random_state=1608637542),
 DecisionTreeClassifier(max_features='sqrt', random_state=1273642419),
 DecisionTreeClassifier(max_features='sqrt', random_state=1935803228),
 DecisionTreeClassifier(max_features='sqrt', random_state=787846414),
 DecisionTreeClassifier(max_features='sqrt', random_state=996406378),
 DecisionTreeClassifier(max_features='sqrt', random_state=1201263687),
 DecisionTreeClassifier(max_features='sqrt', random_state=423734972),
 DecisionTreeClassifier(max_features='sqrt', random_state=415968276),
 DecisionTreeClassifier(max_features='sqrt', random_state=670094950),
 DecisionTreeClassifier(max_features='sqrt', random_state=1914837113),
 DecisionTreeClassifier(max_features='sqrt', random_state=669991378),
 DecisionTreeClassifier(max_features='sqrt', random_state=429389014),
 DecisionTreeClassifier(max_features='sqrt', random_state=249467210),
 DecisionTreeClassifier(max_features='sqrt', random_state=1972458954),
 DecisionTreeClassifier(max_features='sqrt', random_state=1572714583),
 DecisionTreeClassifier(max_features='sqrt', random_state=1433267572),
 DecisionTreeClassifier(max_features='sqrt', random_state=434285667),
 DecisionTreeClassifier(max_features='sqrt', random_state=613608295),
 DecisionTreeClassifier(max_features='sqrt', random_state=893664919),
 DecisionTreeClassifier(max_features='sqrt', random_state=648061058),
 DecisionTreeClassifier(max_features='sqrt', random_state=88409749),
 DecisionTreeClassifier(max_features='sqrt', random_state=242285876),
 DecisionTreeClassifier(max_features='sqrt', random_state=2018247425),
 DecisionTreeClassifier(max_features='sqrt', random_state=953477463),
 DecisionTreeClassifier(max_features='sqrt', random_state=1427830251),
 DecisionTreeClassifier(max_features='sqrt', random_state=1883569565),
 DecisionTreeClassifier(max_features='sqrt', random_state=911989541),
 DecisionTreeClassifier(max_features='sqrt', random_state=3344769),
 DecisionTreeClassifier(max_features='sqrt', random_state=780932287),
```

```
 DecisionTreeClassifier(max_features='sqrt', random_state=2114032571),
 DecisionTreeClassifier(max_features='sqrt', random_state=787716372),
 DecisionTreeClassifier(max_features='sqrt', random_state=504579232),
 DecisionTreeClassifier(max_features='sqrt', random_state=1306710475),
 DecisionTreeClassifier(max_features='sqrt', random_state=479546681),
 DecisionTreeClassifier(max_features='sqrt', random_state=106328085),
 DecisionTreeClassifier(max_features='sqrt', random_state=30349564),
 DecisionTreeClassifier(max_features='sqrt', random_state=1855189739),
 DecisionTreeClassifier(max_features='sqrt', random_state=99052376),
 DecisionTreeClassifier(max_features='sqrt', random_state=1250819632),
 DecisionTreeClassifier(max_features='sqrt', random_state=106406362),
 DecisionTreeClassifier(max_features='sqrt', random_state=480404538),
 DecisionTreeClassifier(max_features='sqrt', random_state=1717389822),
 DecisionTreeClassifier(max_features='sqrt', random_state=599121577),
 DecisionTreeClassifier(max_features='sqrt', random_state=200427519),
 DecisionTreeClassifier(max_features='sqrt', random_state=1254751707),
 DecisionTreeClassifier(max_features='sqrt', random_state=2034764475),
 DecisionTreeClassifier(max_features='sqrt', random_state=1573512143),
 DecisionTreeClassifier(max_features='sqrt', random_state=999745294),
 DecisionTreeClassifier(max_features='sqrt', random_state=1958805693),
 DecisionTreeClassifier(max_features='sqrt', random_state=389151677),
 DecisionTreeClassifier(max_features='sqrt', random_state=1224821422),
 DecisionTreeClassifier(max_features='sqrt', random_state=508464061),
 DecisionTreeClassifier(max_features='sqrt', random_state=857592370),
 DecisionTreeClassifier(max_features='sqrt', random_state=1642661739),
 DecisionTreeClassifier(max_features='sqrt', random_state=61136438),
 DecisionTreeClassifier(max_features='sqrt', random_state=2075460851),
 DecisionTreeClassifier(max_features='sqrt', random_state=396917567),
 DecisionTreeClassifier(max_features='sqrt', random_state=2004731384),
 DecisionTreeClassifier(max_features='sqrt', random_state=199502978),
 DecisionTreeClassifier(max_features='sqrt', random_state=1545932260),
 DecisionTreeClassifier(max_features='sqrt', random_state=461901618),
 DecisionTreeClassifier(max_features='sqrt', random_state=774414982),
 DecisionTreeClassifier(max_features='sqrt', random_state=732395540),
 DecisionTreeClassifier(max_features='sqrt', random_state=1934879560),
 DecisionTreeClassifier(max_features='sqrt', random_state=279394470),
 DecisionTreeClassifier(max_features='sqrt', random_state=56972561),
 DecisionTreeClassifier(max_features='sqrt', random_state=1927948675),
 DecisionTreeClassifier(max_features='sqrt', random_state=1899242072),
 DecisionTreeClassifier(max_features='sqrt', random_state=1999874363),
 DecisionTreeClassifier(max_features='sqrt', random_state=271820813),
 DecisionTreeClassifier(max_features='sqrt', random_state=1324556529),
 DecisionTreeClassifier(max_features='sqrt', random_state=1655351289),
 DecisionTreeClassifier(max_features='sqrt', random_state=1308306184),
 DecisionTreeClassifier(max_features='sqrt', random_state=68574553),
 DecisionTreeClassifier(max_features='sqrt', random_state=419498548),
 DecisionTreeClassifier(max_features='sqrt', random_state=991681409),
 DecisionTreeClassifier(max_features='sqrt', random_state=791274835),
 DecisionTreeClassifier(max_features='sqrt', random_state=1035196507),
 DecisionTreeClassifier(max_features='sqrt', random_state=1890440558),
 DecisionTreeClassifier(max_features='sqrt', random_state=787110843),
 DecisionTreeClassifier(max_features='sqrt', random_state=524150214),
 DecisionTreeClassifier(max_features='sqrt', random_state=472432043),
 DecisionTreeClassifier(max_features='sqrt', random_state=2126768636),
 DecisionTreeClassifier(max_features='sqrt', random_state=1431061255),
 DecisionTreeClassifier(max_features='sqrt', random_state=147697582),
 DecisionTreeClassifier(max_features='sqrt', random_state=744595490),
 DecisionTreeClassifier(max_features='sqrt', random_state=1758017741),
 DecisionTreeClassifier(max_features='sqrt', random_state=1679592528),
 DecisionTreeClassifier(max_features='sqrt', random_state=1111451555),
 DecisionTreeClassifier(max_features='sqrt', random_state=782698033),
 DecisionTreeClassifier(max_features='sqrt', random_state=698027879),
 DecisionTreeClassifier(max_features='sqrt', random_state=1096768899),
 DecisionTreeClassifier(max_features='sqrt', random_state=1338788865),
 DecisionTreeClassifier(max_features='sqrt', random_state=1826030589),
 DecisionTreeClassifier(max_features='sqrt', random_state=86191493),
 DecisionTreeClassifier(max_features='sqrt', random_state=893102645),
 DecisionTreeClassifier(max_features='sqrt', random_state=200619113),
 DecisionTreeClassifier(max_features='sqrt', random_state=290770691),
 DecisionTreeClassifier(max_features='sqrt', random_state=793943861),
 DecisionTreeClassifier(max_features='sqrt', random_state=134489564)]
```

In [121…

```
rf2.estimators_[0]
```

Out[121…

```
DecisionTreeClassifier(max_features='sqrt', random_state=1608637542)
```

In [122…

```
# Make predictions
y_pred = rf2.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9847058823529412
```

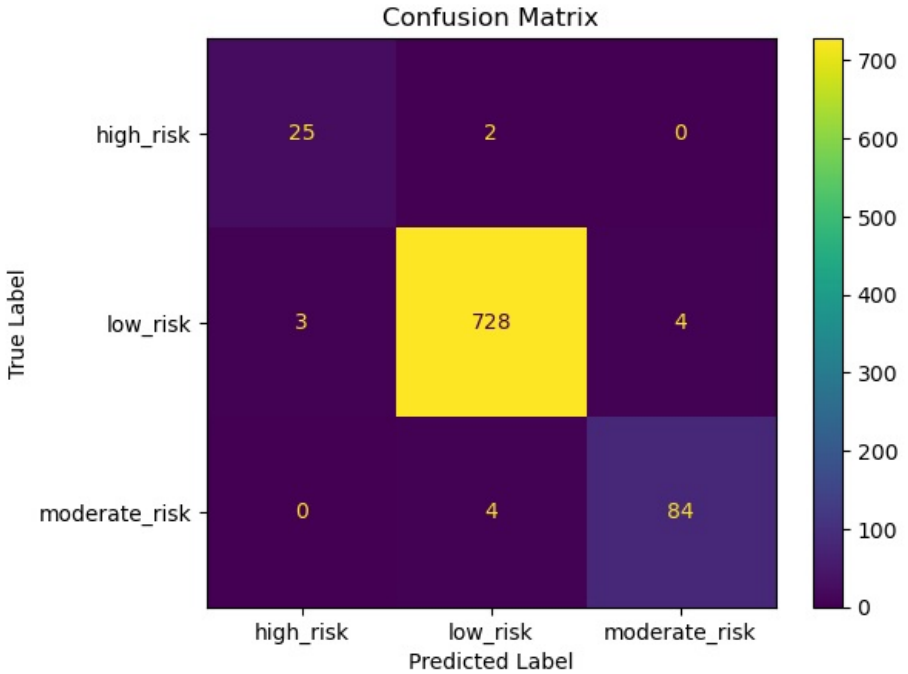
In [123…
```
# Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
# print the scores on training and test set
```

```
Accuracy: 0.98
Precision: 0.98
Recall: 0.98
```

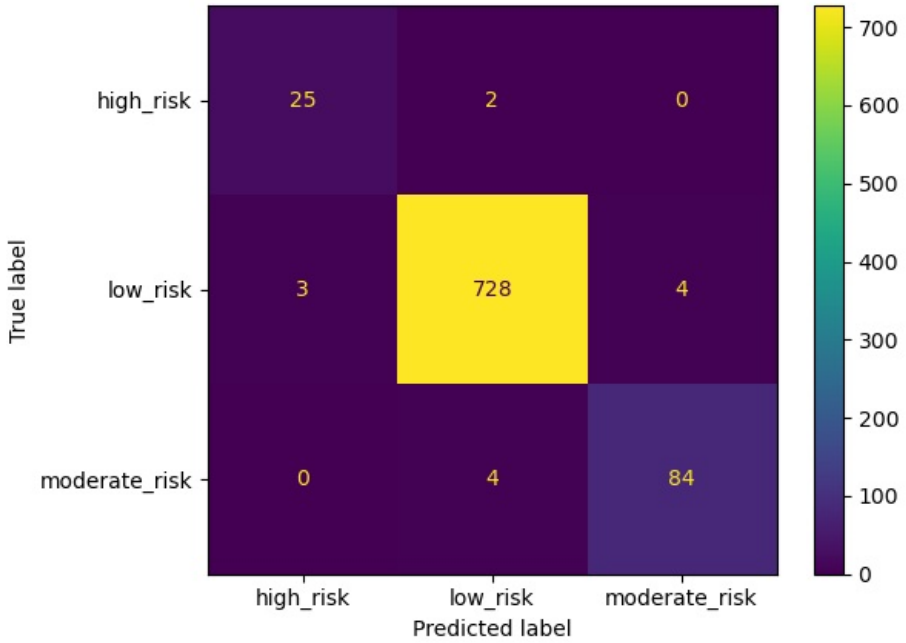
In [124…
```
print('Training set score: {:.4f}'.format(rf2.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(rf2.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9847
```

In [125…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [126…
```
from sklearn.ensemble import GradientBoostingClassifier # GradientBoostingClassifier(n_estimators=100, random_
```

In [127…
```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=42)
```

In [128…
```
gb.fit(X_train_balanced, y_train_balanced)
```

Out[128…

GradientBoostingClassifier

```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
```

In [129…
```
# Make predictions
y_pred = gb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

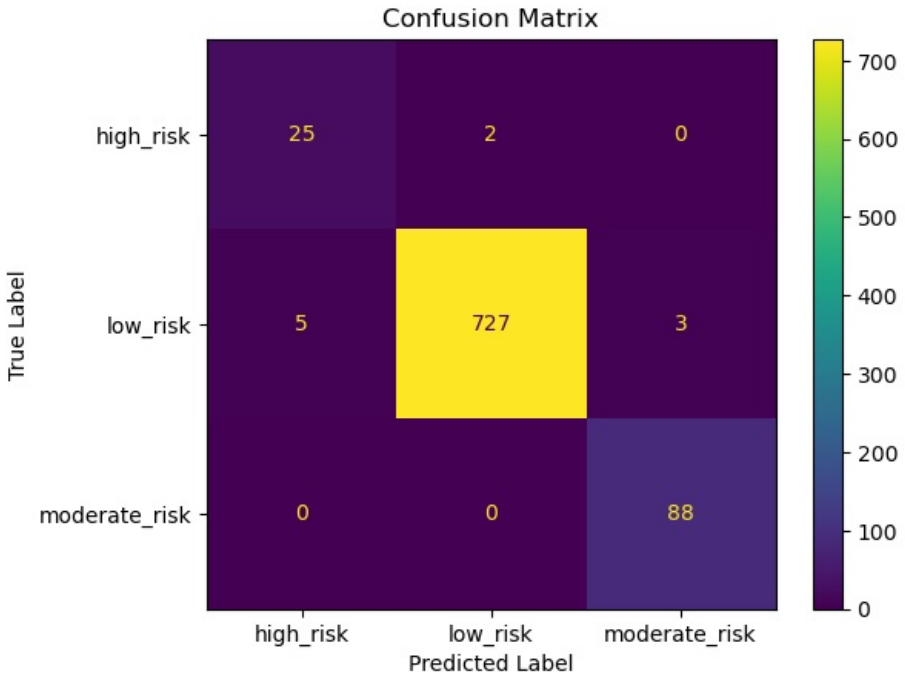
```
Accuracy: 0.9882352941176471
```

In [130…
```
# Evaluate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# Print results
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
# print the scores on training and test set

print('Training set score: {:.4f}'.format(gb.score(X_train_balanced, y_train_balanced)))

print('Test set score: {:.4f}'.format(gb.score(X_test, y_test)))
```

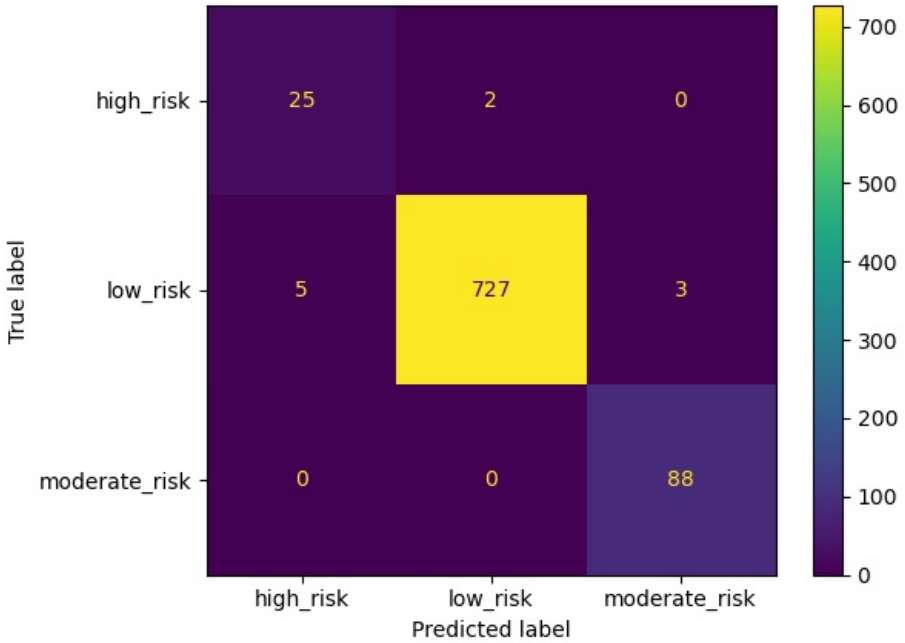
```
Accuracy: 0.99
Precision: 0.99
Recall: 0.99
Training set score: 0.9958
Test set score: 0.9882
```

In [131…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
cm = confusion_matrix(y_test, y_pred)
class_names = ['high_risk', 'low_risk', 'moderate_risk']
# Adjust according to your specific classes
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

cmd.plot()

# Customize the plot with titles and labels
cmd.ax_.set(title='Confusion Matrix', xlabel='Predicted Label', ylabel='True Label')

# Show the plot
plt.figure(figsize=(10, 10))  # Adjust size as needed
cmd.plot()
plt.show()
```




```
<Figure size 1000x1000 with 0 Axes>
```



In [132…

```
from sklearn.metrics import accuracy_score

classifiers = [pipe,knn_classifier, svc,clf_tree,clf, rf, rf2,gb]
classifier_names = ['Pipeline','kNeighbour', 'SVC','DecisionTreeClassifier','DecisionTreeClassifier','RandomFor

# for each classifier get the test accuracy score
scores = [accuracy_score(clf.predict(X_test), y_test) for clf in classifiers]

# Print classifier names and accuracy scores
for name, score in zip(classifier_names, scores):
    print(f"{name} Test Accuracy: {score}")

index = np.argmax(scores)
```

```
print("Test Accuracy Scores: ", scores)

print("Best Classifier: ", classifiers[index])
print("Best Accuracy Score: ", scores[index])
```

```
C:\Users\student\AppData\Roaming\Python\Python311\site-packages\sklearn\base.py:493: UserWarning: X does not hav
e valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
Pipeline Test Accuracy: 0.9317647058823529
kNeighbour Test Accuracy: 0.8694117647058823
SVC Test Accuracy: 0.8223529411764706
DecisionTreeClassifier Test Accuracy: 0.2388235294117647
DecisionTreeClassifier Test Accuracy: 0.9070588235294118
RandomForestClassifier Test Accuracy: 0.9752941176470589
RandomForestClassifierGradientBoostingClassifier Test Accuracy: 0.9847058823529412
Test Accuracy Scores:  [0.9317647058823529, 0.8694117647058823, 0.8223529411764706, 0.2388235294117647, 0.907058
8235294118, 0.9752941176470589, 0.9847058823529412, 0.9882352941176471]
Best Classifier:  GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
Best Accuracy Score:  0.9882352941176471
```

Best model Gradient Boosting Classifier

In [133…
```
dis_test_path = ("disease_test.csv")
```

In [134…
```
disease_teat = pd.read_csv(dis_test_path)
```

In [135…
```
disease_teat.head()
```

Out[135…

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	enlargement	tumor	disorder	medication
0	PA6001	22	female	no	no	0.500	1.7	83.0	0.86	97.0	...	no	no	no	
1	PA6002	42	male	no	no	0.060	2.0	79.0	0.81	98.0	...	no	no	no	
2	PA6003	42	male	no	no	0.045	2.1	111.0	0.89	125.0	...	no	no	no	
3	PA6004	42	female	no	no	1.900	2.0	114.0	1.33	86.0	...	no	no	no	
4	PA6005	55	male	no	no	0.570	1.4	75.0	0.72	104.0	...	no	no	no	

5 rows × 23 columns

In [136…
```
disease_teat.isnull().sum()
```

Out[136…
```
id                   0
age                  0
gender              26
sick                 0
pregnant             0
test_X1             76
test_X2            208
test_X3             33
test_X4             72
test_X5             71
test_X6            725
concern_type1        0
concern_type2        0
enlargement          0
tumor                0
disorder             0
medication_A         0
medication_B         0
mental_health        0
mood_stabiliser      0
surgery              0
treatment_type1      0
suspect              0
dtype: int64
```

In [137…
```
#for categorical
from sklearn.base import TransformerMixin

class DataFrameImputer(TransformerMixin):

    def __init__(self):
        """Impute missing values.

        Columns of dtype object are imputed with the most frequent value
        in column.

        Columns of other types are imputed with mean of column.
```

```
        """
    def fit(self, x, y=None):
        # Find most common value with value_counts() which returns
        # counts in descending order so that the first element is the most frequently-occurring element.
        self.fill = pd.Series([x[c].value_counts().index[0]
            #Use that if type is object otherwise use mean
            if x[c].dtype == np.dtype('O') else x[c].mean() for c in x],
            index=x.columns)

        return self

    def transform(self, x, y=None):
        return x.fillna(self.fill)


#Define the data to be filled as X, we can pass the whole data frame and apply our new class
x = disease_teat
train = DataFrameImputer().fit_transform(x)

print('before...')
#Let us see missing value for the train data before
missing_val_count_by_column = (x.isnull().sum())
print('Missing columns for the Train data:\n',missing_val_count_by_column[missing_val_count_by_column>0])

#and after
print('after...')
missing_val_count_by_column = (xt.isnull().sum())
print('Missing columns for the Train data:\n',missing_val_count_by_column[missing_val_count_by_column>0])
```

```
before...
Missing columns for the Train data:
 gender      26
test_X1     76
test_X2    208
test_X3     33
test_X4     72
test_X5     71
test_X6    725
dtype: int64
after...
Missing columns for the Train data:
 Series([], dtype: int64)
```

In [138…
```
train.isnull().sum()
```

Out[138…
```
id                0
age               0
gender            0
sick              0
pregnant          0
test_X1           0
test_X2           0
test_X3           0
test_X4           0
test_X5           0
test_X6           0
concern_type1     0
concern_type2     0
enlargement       0
tumor             0
disorder          0
medication_A      0
medication_B      0
mental_health     0
mood_stabiliser   0
surgery           0
treatment_type1   0
suspect           0
dtype: int64
```

In [139…
```
train.describe()
```

Out[139…

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6
count	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000
mean	52.956000	6.674659	2.001181	104.772664	0.963451	111.738586	20.578000
std	18.639921	24.551751	0.653176	34.636241	0.162832	46.095014	1.634402
min	1.000000	0.005000	0.200000	2.900000	0.190000	2.400000	0.150000
25%	38.000000	0.600000	1.700000	87.000000	0.880000	95.000000	20.578000
50%	56.000000	1.600000	2.001181	104.000000	0.963451	111.738586	20.578000
75%	68.000000	4.900000	2.100000	120.000000	1.040000	123.000000	20.578000
max	92.000000	393.000000	8.500000	273.000000	2.120000	839.000000	30.000000

In [140…
```
disease_test = train.drop(['test_X6', 'test_X5'], axis=1)
```

In [141…
```
disease_test.head()
```

Out[141…

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	concern_type1	...	enlargement	tumor	disorder	me
0	PA6001	22	female	no	no	0.500	1.7	83.0	0.86	no	...	no	no	no	
1	PA6002	42	male	no	no	0.060	2.0	79.0	0.81	no	...	no	no	no	
2	PA6003	42	male	no	no	0.045	2.1	111.0	0.89	no	...	no	no	no	
3	PA6004	42	female	no	no	1.900	2.0	114.0	1.33	no	...	no	no	no	
4	PA6005	55	male	no	no	0.570	1.4	75.0	0.72	no	...	no	no	no	

5 rows × 21 columns

In [142…
```
columns_to_encode = ['gender', 'sick', 'pregnant', 'concern_type1',
                     'concern_type2', 'enlargement', 'tumor', 'disorder',
                     'medication_A', 'medication_B', 'mental_health',
                     'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect']
```

In [143…
```
X_1 = disease_test
```

In [144…
```
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
#enc = OneHotEncoder(handle_unknown='ignore')
# Apply one-hot encoding to the categorical columns
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(X_1[columns_to_encode])
#Create a DataFrame with the one-hot encoded columns
#We use get_feature_names_out() to get the column names for the encoded data
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(columns_to_encode))
# Concatenate the one-hot encoded dataframe with the original dataframe
Xencoded = pd.concat([X_1, one_hot_df], axis=1)

# Drop the original categorical columns
d_encoded = Xencoded.drop(columns_to_encode, axis=1)

# Display the resulting dataframe
print(f"Encoded Disease data : \n{d_encoded}")
```

```
Encoded Disease data :
         id  age  test_X1   test_X2  test_X3  test_X4  gender_female  \
0    PA6001   22     0.500  1.700000     83.0     0.86            1.0
1    PA6002   42     0.060  2.000000     79.0     0.81            0.0
2    PA6003   42     0.045  2.100000    111.0     0.89            0.0
3    PA6004   42     1.900  2.000000    114.0     1.33            1.0
4    PA6005   55     0.570  1.400000     75.0     0.72            0.0
..      ...  ...       ...       ...      ...      ...            ...
745  PA6746   55     1.300  2.001181     72.0     0.75            0.0
746  PA6747   62     1.400  2.001181    104.0     1.12            1.0
747  PA6748   54     1.900  2.000000     88.0     0.95            1.0
748  PA6749   69     0.010  2.400000    127.0     0.84            1.0
749  PA6750   51     4.000  1.700000     70.0     0.90            1.0

     gender_male  sick_no  sick_yes  ...  mental_health_no  mental_health_yes  \
0            0.0      1.0       0.0  ...               1.0                0.0
1            1.0      1.0       0.0  ...               1.0                0.0
2            1.0      1.0       0.0  ...               1.0                0.0
3            0.0      1.0       0.0  ...               1.0                0.0
4            1.0      1.0       0.0  ...               0.0                1.0
..           ...      ...       ...  ...               ...                ...
745          1.0      1.0       0.0  ...               1.0                0.0
746          0.0      1.0       0.0  ...               1.0                0.0
747          0.0      1.0       0.0  ...               1.0                0.0
748          0.0      1.0       0.0  ...               1.0                0.0
749          0.0      1.0       0.0  ...               1.0                0.0

     mood_stabiliser_no  mood_stabiliser_yes  surgery_no  surgery_yes  \
0                   1.0                  0.0         1.0          0.0
1                   1.0                  0.0         1.0          0.0
2                   1.0                  0.0         1.0          0.0
3                   1.0                  0.0         1.0          0.0
4                   1.0                  0.0         1.0          0.0
..                  ...                  ...         ...          ...
745                 1.0                  0.0         1.0          0.0
746                 1.0                  0.0         1.0          0.0
747                 1.0                  0.0         1.0          0.0
748                 1.0                  0.0         1.0          0.0
749                 1.0                  0.0         1.0          0.0

     treatment_type1_no  treatment_type1_yes  suspect_no  suspect_yes
0                   1.0                  0.0         1.0          0.0
1                   1.0                  0.0         1.0          0.0
2                   1.0                  0.0         1.0          0.0
3                   1.0                  0.0         1.0          0.0
4                   1.0                  0.0         1.0          0.0
..                  ...                  ...         ...          ...
745                 1.0                  0.0         1.0          0.0
746                 1.0                  0.0         1.0          0.0
747                 1.0                  0.0         1.0          0.0
748                 1.0                  0.0         1.0          0.0
749                 1.0                  0.0         1.0          0.0

[750 rows x 35 columns]
```

In [145…
```
new_data = d_encoded
```

In [146…
```
new_data.head()
```

Out[146…

	id	age	test_X1	test_X2	test_X3	test_X4	gender_female	gender_male	sick_no	sick_yes	...	mental_health_no	mental_
0	PA6001	22	0.500	1.7	83.0	0.86	1.0	0.0	1.0	0.0	...	1.0	
1	PA6002	42	0.060	2.0	79.0	0.81	0.0	1.0	1.0	0.0	...	1.0	
2	PA6003	42	0.045	2.1	111.0	0.89	0.0	1.0	1.0	0.0	...	1.0	
3	PA6004	42	1.900	2.0	114.0	1.33	1.0	0.0	1.0	0.0	...	1.0	
4	PA6005	55	0.570	1.4	75.0	0.72	0.0	1.0	1.0	0.0	...	0.0	

5 rows × 35 columns

In [147…
```
new_data = new_data.drop(['id'], axis=1)
```

In [148…
```
new_data.head()
```

Out[148…

	age	test_X1	test_X2	test_X3	test_X4	gender_female	gender_male	sick_no	sick_yes	pregnant_no	...	mental_health_no	m
0	22	0.500	1.7	83.0	0.86	1.0	0.0	1.0	0.0	1.0	...	1.0	
1	42	0.060	2.0	79.0	0.81	0.0	1.0	1.0	0.0	1.0	...	1.0	
2	42	0.045	2.1	111.0	0.89	0.0	1.0	1.0	0.0	1.0	...	1.0	
3	42	1.900	2.0	114.0	1.33	1.0	0.0	1.0	0.0	1.0	...	1.0	
4	55	0.570	1.4	75.0	0.72	0.0	1.0	1.0	0.0	1.0	...	0.0	

5 rows × 34 columns

In [149…
```
predicted_target = gb.predict(new_data)
```

```
C:\Users\student\AppData\Roaming\Python\Python311\site-packages\sklearn\base.py:486: UserWarning: X has feature
names, but GradientBoostingClassifier was fitted without feature names
  warnings.warn(
```

In [150…
```
#decoding the predicted target
risk = {0:'high_risk', 1: 'low_risk', 2: 'moderate_risk'}
target_new = [risk[code] for code in predicted_target]
# convert to data frame
new_data['Disease predicted'] = target_new
print(new_data[['Disease predicted']])
new_data.to_csv('Disease_predicted_2.csv',index = True)
print('New file available ')
```

```
    Disease predicted
0            low_risk
1            low_risk
2            low_risk
3            low_risk
4            low_risk
..                ...
745          low_risk
746          low_risk
747          low_risk
748          low_risk
749          low_risk

[750 rows x 1 columns]
New file available
```

unsupervised learning

In [151…
```
disease.head()
```

Out[151…

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	medicatic
0	PA1001	59	male	no	no	7.8	NaN	89.0	0.85	105.0	...	no	no	no	
1	PA1002	48	female	no	no	1.5	2.5	101.0	0.97	104.0	...	no	no	yes	
2	PA1003	77	male	no	no	7.3	1.2	57.0	1.28	44.0	...	no	no	no	
3	PA1004	42	female	no	no	1.2	2.5	106.0	0.98	108.0	...	no	no	no	
4	PA1005	38	female	no	no	0.6	1.9	95.0	NaN	NaN	...	no	no	no	

5 rows × 24 columns

In [152…
```
dise.head()
```

Out[152…

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	med
0	PA1001	59	male	no	no	7.8	2.03558	89.0	0.850000	105.000000	...	no	no	no	
1	PA1002	48	female	no	no	1.5	2.50000	101.0	0.970000	104.000000	...	no	no	yes	
2	PA1003	77	male	no	no	7.3	1.20000	57.0	1.280000	44.000000	...	no	no	no	
3	PA1004	42	female	no	no	1.2	2.50000	106.0	0.980000	108.000000	...	no	no	no	
4	PA1005	38	female	no	no	0.6	1.90000	95.0	0.970846	110.090834	...	no	no	no	

5 rows × 24 columns

In [153…
```
diseases.head()
```

Out[153…

	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	concern_type1	concern_type2	...	tumor	disorder	medicatic
0	59	male	no	no	7.8	2.03558	89.0	0.850000	no	yes	...	no	no	
1	48	female	no	no	1.5	2.50000	101.0	0.970000	no	no	...	no	no	
2	77	male	no	no	7.3	1.20000	57.0	1.280000	no	no	...	no	no	
3	42	female	no	no	1.2	2.50000	106.0	0.980000	no	no	...	no	no	
4	38	female	no	no	0.6	1.90000	95.0	0.970846	no	no	...	no	no	

5 rows × 21 columns

In [154…

```
columns_to_encode = ['gender', 'sick', 'pregnant', 'concern_type1',
                     'concern_type2', 'enlargement', 'tumor', 'disorder',
                     'medication_A', 'medication_B', 'mental_health',
                     'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect']
```

In [155…

```
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
#enc = OneHotEncoder(handle_unknown='ignore')
# Apply one-hot encoding to the categorical columns
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(diseases[columns_to_encode])
#Create a DataFrame with the one-hot encoded columns
#We use get_feature_names_out() to get the column names for the encoded data
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(columns_to_encode))
# Concatenate the one-hot encoded dataframe with the original dataframe
X1_encoded = pd.concat([diseases, one_hot_df], axis=1)

# Drop the original categorical columns
df1_encoded = X1_encoded.drop(columns_to_encode, axis=1)

# Display the resulting dataframe
print(f"Encoded Disease data : \n{df1_encoded}")
```

Encoded Disease data :

	age	test_X1	test_X2	test_X3	test_X4	target	gender_female	\
0	59	7.8	2.03558	89.0	0.850000	2	0.0	
1	48	1.5	2.50000	101.0	0.970000	1	1.0	
2	77	7.3	1.20000	57.0	1.280000	2	0.0	
3	42	1.2	2.50000	106.0	0.980000	1	1.0	
4	38	0.6	1.90000	95.0	0.970846	1	1.0	
...	...	...	...	...	...	...	...	
4245	29	0.5	2.30000	105.0	0.860000	1	0.0	
4246	46	35.0	1.20000	16.0	0.860000	2	1.0	
4247	72	3.4	2.10000	88.0	0.960000	1	1.0	
4248	94	3.9	2.03558	157.0	1.020000	1	1.0	
4249	41	1.8	2.80000	60.0	0.950000	1	0.0	

	gender_male	sick_no	sick_yes	...	mental_health_no	\
0	1.0	1.0	0.0	...	1.0	
1	0.0	1.0	0.0	...	1.0	
2	1.0	1.0	0.0	...	1.0	
3	0.0	1.0	0.0	...	1.0	
4	0.0	1.0	0.0	...	1.0	
...	...	...	...	...	...	
4245	1.0	1.0	0.0	...	1.0	
4246	0.0	1.0	0.0	...	1.0	
4247	0.0	1.0	0.0	...	1.0	
4248	0.0	1.0	0.0	...	1.0	
4249	1.0	1.0	0.0	...	1.0	

	mental_health_yes	mood_stabiliser_no	mood_stabiliser_yes	surgery_no	\
0	0.0	1.0	0.0	1.0	
1	0.0	0.0	1.0	1.0	
2	0.0	1.0	0.0	1.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	
...	...	...	...	...	
4245	0.0	1.0	0.0	1.0	
4246	0.0	1.0	0.0	1.0	
4247	0.0	1.0	0.0	1.0	
4248	0.0	1.0	0.0	1.0	
4249	0.0	1.0	0.0	1.0	

	surgery_yes	treatment_type1_no	treatment_type1_yes	suspect_no	\
0	0.0	1.0	0.0	1.0	
1	0.0	1.0	0.0	1.0	
2	0.0	1.0	0.0	1.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	
...	...	...	...	...	
4245	0.0	1.0	0.0	1.0	
4246	0.0	1.0	0.0	1.0	
4247	0.0	1.0	0.0	1.0	
4248	0.0	1.0	0.0	1.0	
4249	0.0	1.0	0.0	1.0	

	suspect_yes
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
4245	0.0
4246	0.0
4247	0.0
4248	0.0
4249	0.0

[4250 rows x 35 columns]

In [156…
```
df1_encoded
```

Out[156…

	age	test_X1	test_X2	test_X3	test_X4	target	gender_female	gender_male	sick_no	sick_yes	...	mental_health_no	mer
0	59	7.8	2.03558	89.0	0.850000	2	0.0	1.0	1.0	0.0	...	1.0	
1	48	1.5	2.50000	101.0	0.970000	1	1.0	0.0	1.0	0.0	...	1.0	
2	77	7.3	1.20000	57.0	1.280000	2	0.0	1.0	1.0	0.0	...	1.0	
3	42	1.2	2.50000	106.0	0.980000	1	1.0	0.0	1.0	0.0	...	1.0	
4	38	0.6	1.90000	95.0	0.970846	1	1.0	0.0	1.0	0.0	...	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4245	29	0.5	2.30000	105.0	0.860000	1	0.0	1.0	1.0	0.0	...	1.0	
4246	46	35.0	1.20000	16.0	0.860000	2	1.0	0.0	1.0	0.0	...	1.0	
4247	72	3.4	2.10000	88.0	0.960000	1	1.0	0.0	1.0	0.0	...	1.0	
4248	94	3.9	2.03558	157.0	1.020000	1	1.0	0.0	1.0	0.0	...	1.0	
4249	41	1.8	2.80000	60.0	0.950000	1	0.0	1.0	1.0	0.0	...	1.0	

4250 rows × 35 columns

In [157…
```
disea.head()
```

Out[157…

	id	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	...	tumor	disorder	medication_A	med
0	PA1001	59	male	no	no	7.8	2.03558	89.0	0.850000	105.000000	...	no	no	no	
1	PA1002	48	female	no	no	1.5	2.50000	101.0	0.970000	104.000000	...	no	no	yes	
2	PA1003	77	male	no	no	7.3	1.20000	57.0	1.280000	44.000000	...	no	no	no	
3	PA1004	42	female	no	no	1.2	2.50000	106.0	0.980000	108.000000	...	no	no	no	
4	PA1005	38	female	no	no	0.6	1.90000	95.0	0.970846	110.090834	...	no	no	no	

5 rows × 24 columns

In [158…
```
new_data= disea.drop(['id'], axis=1)
```

In [159…
```
new_data
```

Out[159…

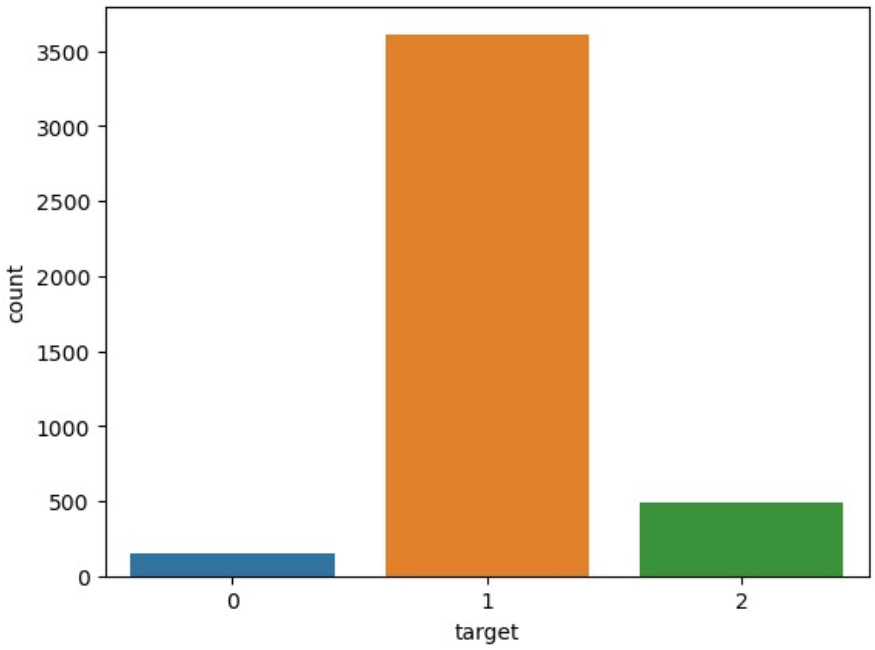
	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	...	tumor	disorder	medication_A
0	59	male	no	no	7.8	2.03558	89.0	0.850000	105.000000	23.325974	...	no	no	no
1	48	female	no	no	1.5	2.50000	101.0	0.970000	104.000000	23.325974	...	no	no	yes
2	77	male	no	no	7.3	1.20000	57.0	1.280000	44.000000	23.325974	...	no	no	no
3	42	female	no	no	1.2	2.50000	106.0	0.980000	108.000000	27.000000	...	no	no	no
4	38	female	no	no	0.6	1.90000	95.0	0.970846	110.090834	23.325974	...	no	no	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4245	29	male	no	no	0.5	2.30000	105.0	0.860000	122.000000	23.325974	...	no	no	no
4246	46	female	no	no	35.0	1.20000	16.0	0.860000	19.000000	23.325974	...	no	no	no
4247	72	female	no	no	3.4	2.10000	88.0	0.960000	92.000000	23.325974	...	no	no	no
4248	94	female	no	no	3.9	2.03558	157.0	1.020000	154.000000	23.325974	...	no	no	no
4249	41	male	no	no	1.8	2.80000	60.0	0.950000	63.000000	23.325974	...	no	no	no

4250 rows × 23 columns

In [160…
```
from sklearn.preprocessing import LabelEncoder
#label_encoder object knows how to understand word labels.
label_encoder = LabelEncoder()

# Encode labels in column 'species'.
new_data['target']= label_encoder.fit_transform(new_data['target'])
target_mapping =dict(zip(label_encoder.classes_,label_encoder.transform(label_encoder.classes_)))
print('target',target_mapping)
sns.countplot(x='target',data=new_data)
```

```
target {'high_risk': 0, 'low_risk': 1, 'moderate_risk': 2}
```

Out[160… <Axes: xlabel='target', ylabel='count'>



In [161…
```
new_data
```

Out[161…

	age	gender	sick	pregnant	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	...	tumor	disorder	medication_A
0	59	male	no	no	7.8	2.03558	89.0	0.850000	105.000000	23.325974	...	no	no	no
1	48	female	no	no	1.5	2.50000	101.0	0.970000	104.000000	23.325974	...	no	no	yes
2	77	male	no	no	7.3	1.20000	57.0	1.280000	44.000000	23.325974	...	no	no	no
3	42	female	no	no	1.2	2.50000	106.0	0.980000	108.000000	27.000000	...	no	no	no
4	38	female	no	no	0.6	1.90000	95.0	0.970846	110.090834	23.325974	...	no	no	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4245	29	male	no	no	0.5	2.30000	105.0	0.860000	122.000000	23.325974	...	no	no	no
4246	46	female	no	no	35.0	1.20000	16.0	0.860000	19.000000	23.325974	...	no	no	no
4247	72	female	no	no	3.4	2.10000	88.0	0.960000	92.000000	23.325974	...	no	no	no
4248	94	female	no	no	3.9	2.03558	157.0	1.020000	154.000000	23.325974	...	no	no	no
4249	41	male	no	no	1.8	2.80000	60.0	0.950000	63.000000	23.325974	...	no	no	no

4250 rows × 23 columns

In [162…
```
columns_to_encode = ['gender', 'sick', 'pregnant', 'concern_type1',
                     'concern_type2', 'enlargement', 'tumor', 'disorder',
                     'medication_A', 'medication_B', 'mental_health',
                     'mood_stabiliser', 'surgery', 'treatment_type1', 'suspect']
```

In [163…
```
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
#enc = OneHotEncoder(handle_unknown='ignore')
# Apply one-hot encoding to the categorical columns
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(new_data[columns_to_encode])
#Create a DataFrame with the one-hot encoded columns
#We use get_feature_names_out() to get the column names for the encoded data
one_hot_dff = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(columns_to_encode))
# Concatenate the one-hot encoded dataframe with the original dataframe
X_encoded = pd.concat([new_data, one_hot_dff], axis=1)

# Drop the original categorical columns
dff_encoded = X_encoded.drop(columns_to_encode, axis=1)

# Display the resulting dataframe
print(f"Encoded Disease data : \n{dff_encoded}")
```

```
Encoded Disease data :
      age  test_X1  test_X2  test_X3   test_X4     test_X5    test_X6  target  \
0      59      7.8  2.03558     89.0  0.850000  105.000000  23.325974       2
1      48      1.5  2.50000    101.0  0.970000  104.000000  23.325974       1
2      77      7.3  1.20000     57.0  1.280000   44.000000  23.325974       2
3      42      1.2  2.50000    106.0  0.980000  108.000000  27.000000       1
4      38      0.6  1.90000     95.0  0.970846  110.090834  23.325974       1
...   ...      ...      ...      ...       ...         ...        ...     ...
4245   29      0.5  2.30000    105.0  0.860000  122.000000  23.325974       1
4246   46     35.0  1.20000     16.0  0.860000   19.000000  23.325974       2
4247   72      3.4  2.10000     88.0  0.960000   92.000000  23.325974       1
4248   94      3.9  2.03558    157.0  1.020000  154.000000  23.325974       1
4249   41      1.8  2.80000     60.0  0.950000   63.000000  23.325974       1

      gender_female  gender_male  ...  mental_health_no  mental_health_yes  \
0               0.0          1.0  ...               1.0                0.0
1               1.0          0.0  ...               1.0                0.0
2               0.0          1.0  ...               1.0                0.0
3               1.0          0.0  ...               1.0                0.0
4               1.0          0.0  ...               1.0                0.0
...             ...          ...  ...               ...                ...
4245            0.0          1.0  ...               1.0                0.0
4246            1.0          0.0  ...               1.0                0.0
4247            1.0          0.0  ...               1.0                0.0
4248            1.0          0.0  ...               1.0                0.0
4249            0.0          1.0  ...               1.0                0.0

      mood_stabiliser_no  mood_stabiliser_yes  surgery_no  surgery_yes  \
0                    1.0                  0.0         1.0          0.0
1                    0.0                  1.0         1.0          0.0
2                    1.0                  0.0         1.0          0.0
3                    1.0                  0.0         1.0          0.0
4                    1.0                  0.0         1.0          0.0
...                  ...                  ...         ...          ...
4245                 1.0                  0.0         1.0          0.0
4246                 1.0                  0.0         1.0          0.0
4247                 1.0                  0.0         1.0          0.0
4248                 1.0                  0.0         1.0          0.0
4249                 1.0                  0.0         1.0          0.0

      treatment_type1_no  treatment_type1_yes  suspect_no  suspect_yes
0                    1.0                  0.0         1.0          0.0
1                    1.0                  0.0         1.0          0.0
2                    1.0                  0.0         1.0          0.0
3                    1.0                  0.0         1.0          0.0
4                    1.0                  0.0         1.0          0.0
...                  ...                  ...         ...          ...
4245                 1.0                  0.0         1.0          0.0
4246                 1.0                  0.0         1.0          0.0
4247                 1.0                  0.0         1.0          0.0
4248                 1.0                  0.0         1.0          0.0
4249                 1.0                  0.0         1.0          0.0

[4250 rows x 37 columns]
```

In [164…
```
new_diseases = dff_encoded
```

In [165…
```
new_diseases.head()
```

Out[165…

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	target	gender_female	gender_male	...	mental_health_no	me
0	59	7.8	2.03558	89.0	0.850000	105.000000	23.325974	2	0.0	1.0	...	1.0	
1	48	1.5	2.50000	101.0	0.970000	104.000000	23.325974	1	1.0	0.0	...	1.0	
2	77	7.3	1.20000	57.0	1.280000	44.000000	23.325974	2	0.0	1.0	...	1.0	
3	42	1.2	2.50000	106.0	0.980000	108.000000	27.000000	1	1.0	0.0	...	1.0	
4	38	0.6	1.90000	95.0	0.970846	110.090834	23.325974	1	1.0	0.0	...	1.0	

5 rows × 37 columns

In [166…
```
new_diseases.head()
```

Out[166…

	age	test_X1	test_X2	test_X3	test_X4	test_X5	test_X6	target	gender_female	gender_male	...	mental_health_no	me
0	59	7.8	2.03558	89.0	0.850000	105.000000	23.325974	2	0.0	1.0	...	1.0	
1	48	1.5	2.50000	101.0	0.970000	104.000000	23.325974	1	1.0	0.0	...	1.0	
2	77	7.3	1.20000	57.0	1.280000	44.000000	23.325974	2	0.0	1.0	...	1.0	
3	42	1.2	2.50000	106.0	0.980000	108.000000	27.000000	1	1.0	0.0	...	1.0	
4	38	0.6	1.90000	95.0	0.970846	110.090834	23.325974	1	1.0	0.0	...	1.0	

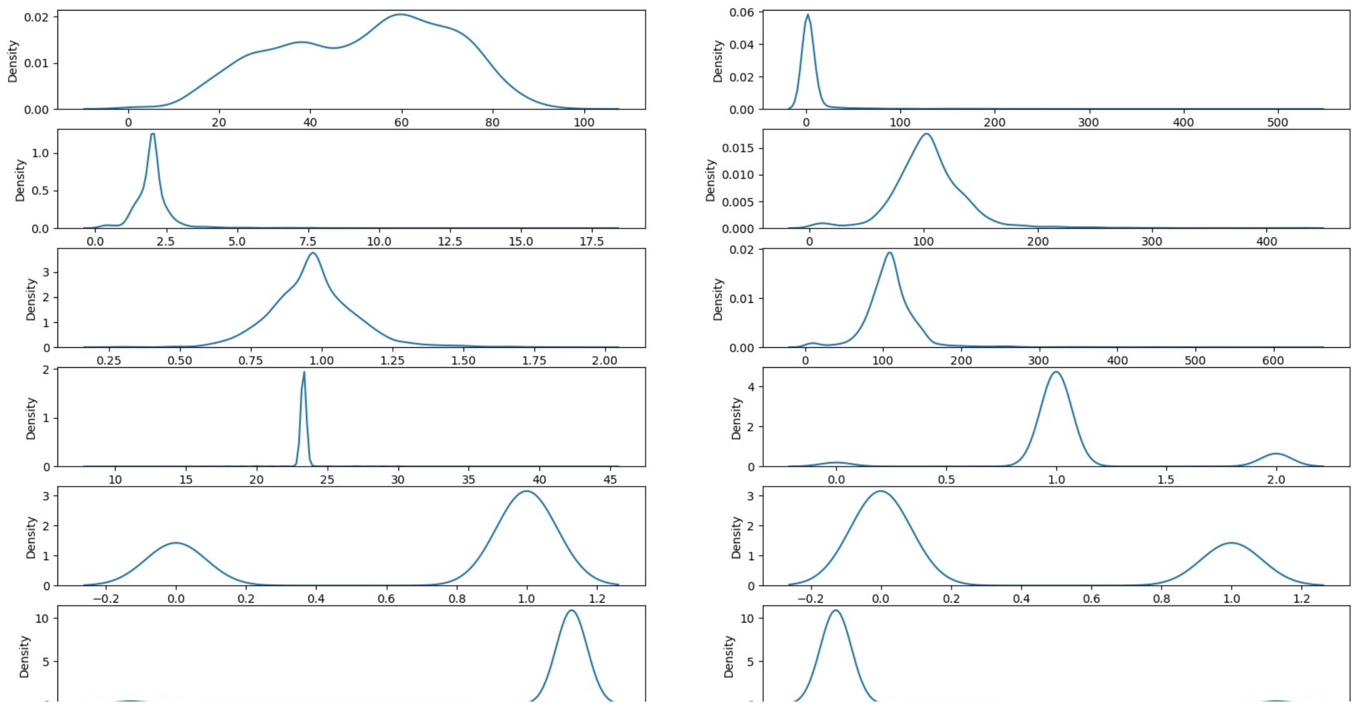
5 rows × 37 columns

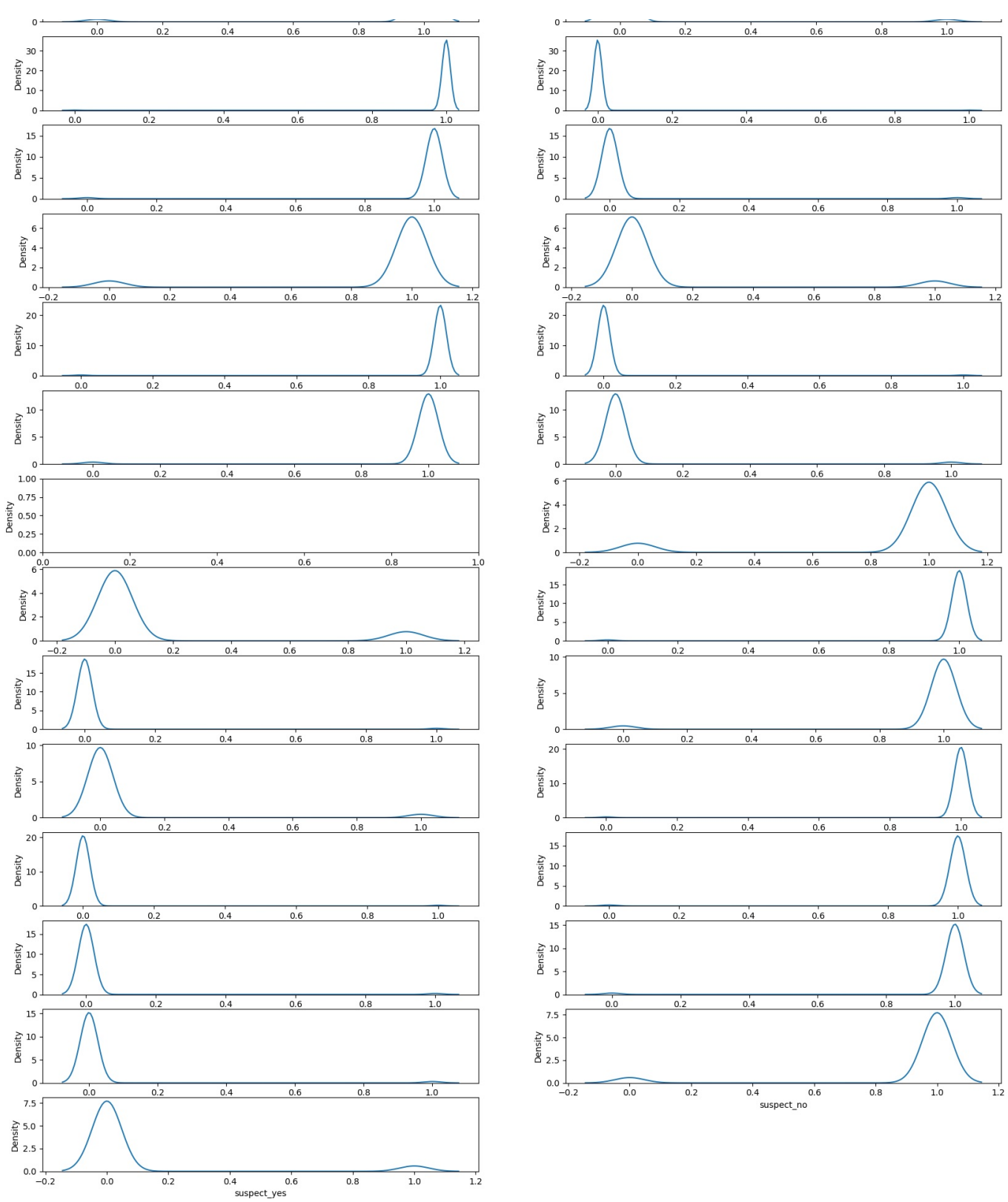
In [167…

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,35))
for i, col in enumerate(new_diseases.columns):
    if new_diseases[col].dtype != 'object':
        ax = plt.subplot(19, 2, i+1)
        sns.kdeplot(new_diseases[col], ax=ax)
        plt.xlabel(col)

plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\student\AppData\Local\Temp\ipykernel_18164\4191389466.py:8: UserWarning: Dataset has 0 variance; skippi
ng density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(new_diseases[col], ax=ax)
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

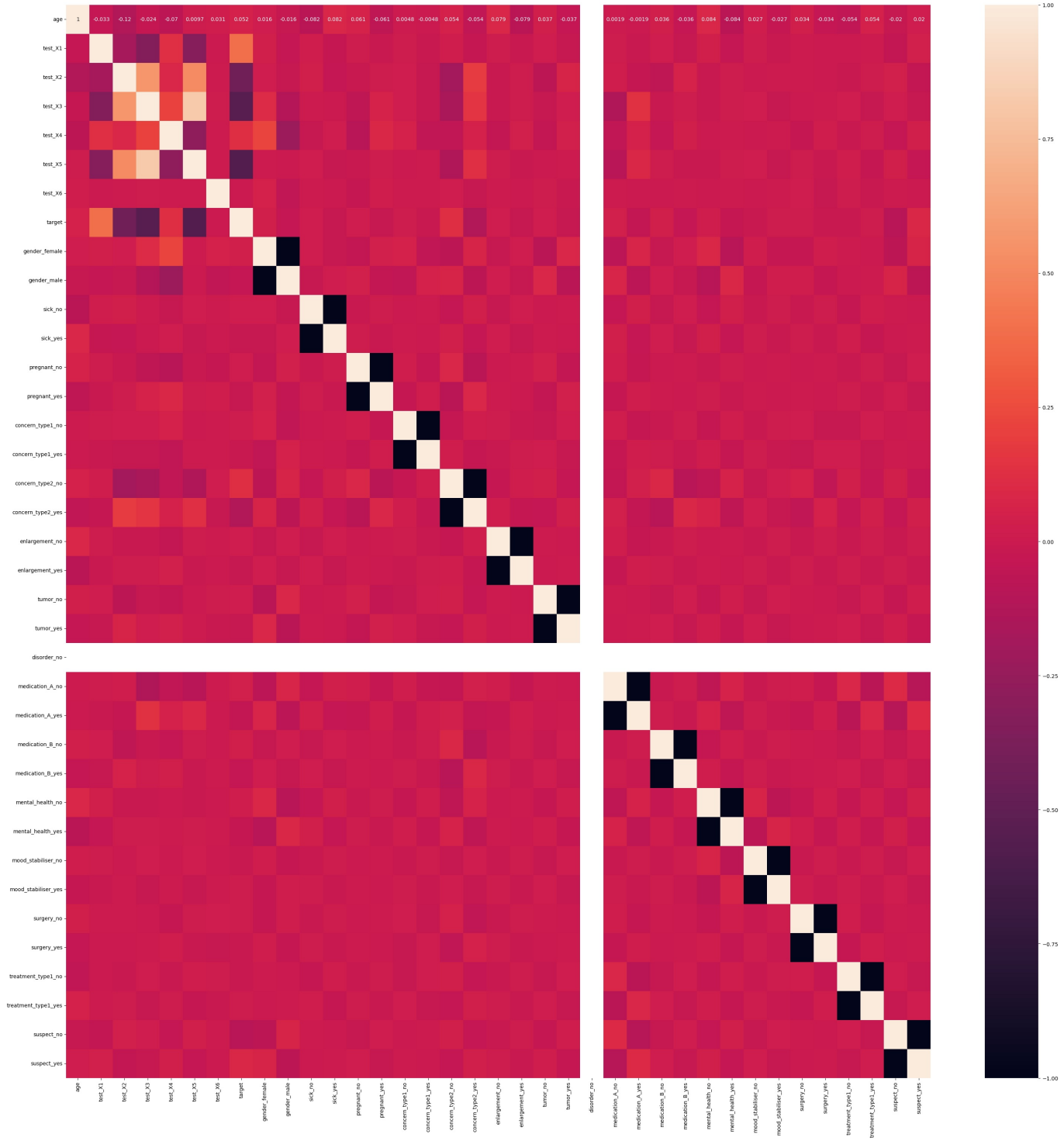




In [168…

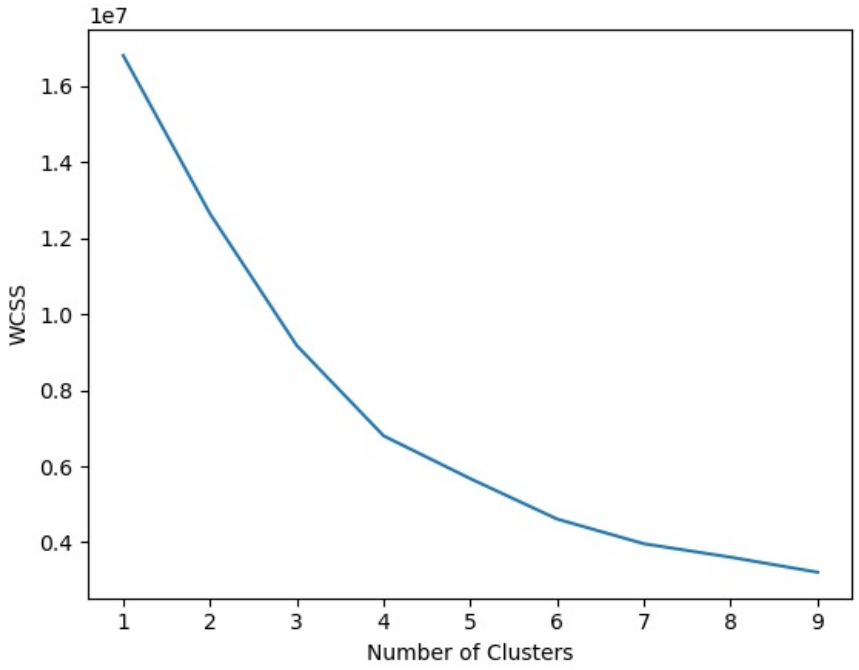
```
plt.figure(figsize=(37,37))
sns.heatmap(new_diseases.corr(), annot=True)
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\matrix.py:260: FutureWarning: Format strings passed to MaskedConstant are ignored, but in future may error or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```

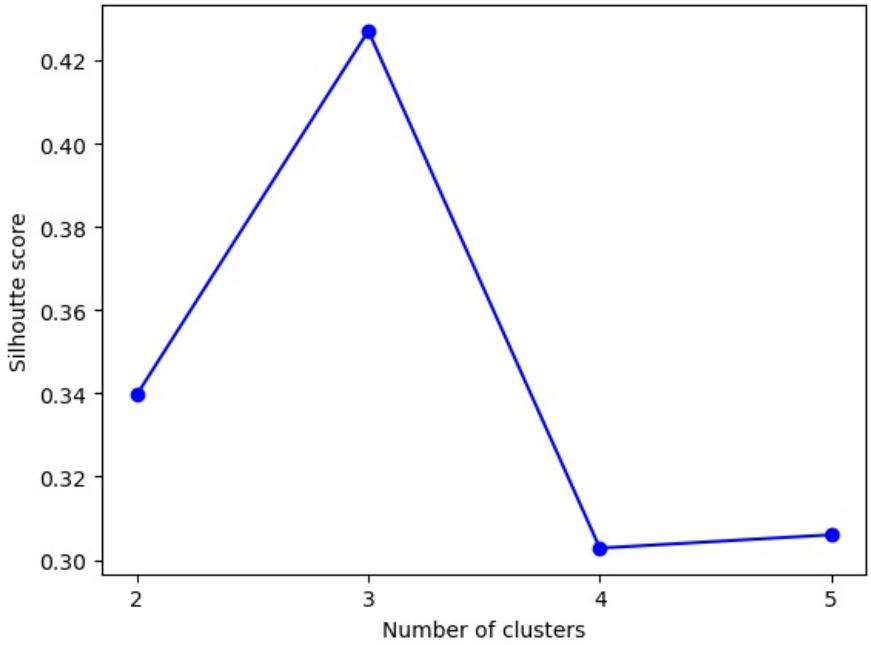


In [169…

```
from sklearn.cluster import KMeans

kmeans_models = [KMeans(n_clusters=k).fit(new_diseases) for k in range (1, 10)]
innertia = [model.inertia_ for model in kmeans_models]

plt.plot(range(1, 10), innertia)

plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



In [171…]
```
from sklearn.metrics import silhouette_score

silhoutte_scores = [silhouette_score(new_diseases, model.labels_) for model in kmeans_models[1:5]]
plt.plot(range(2,6), silhoutte_scores, "bo-")
plt.xticks([2, 3, 4, 5])

plt.xlabel('Number of clusters')
plt.ylabel('Silhoutte score')
plt.show()
```



In [172…]
```
df = new_diseases[['age','target']]
```

In [173…]
```
df1 = new_diseases[['test_X1','target']]
```

In [174…]
```
df2 = new_diseases[['test_X2','target']]
```

In [175…]
```
df3 = new_diseases[['test_X3','target']]
```

In [176…]
```
df4 = new_diseases[['test_X4','target']]
```

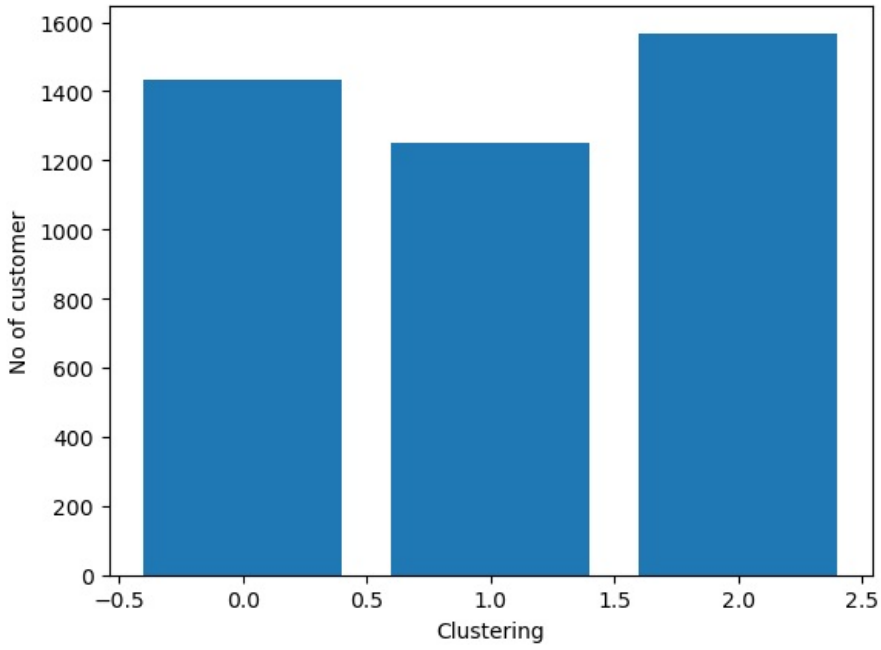
In [177…]
```
df5 = new_diseases[['test_X5','target']]
```

In [178…]
```
df6 = new_diseases[['test_X6','target']]
```

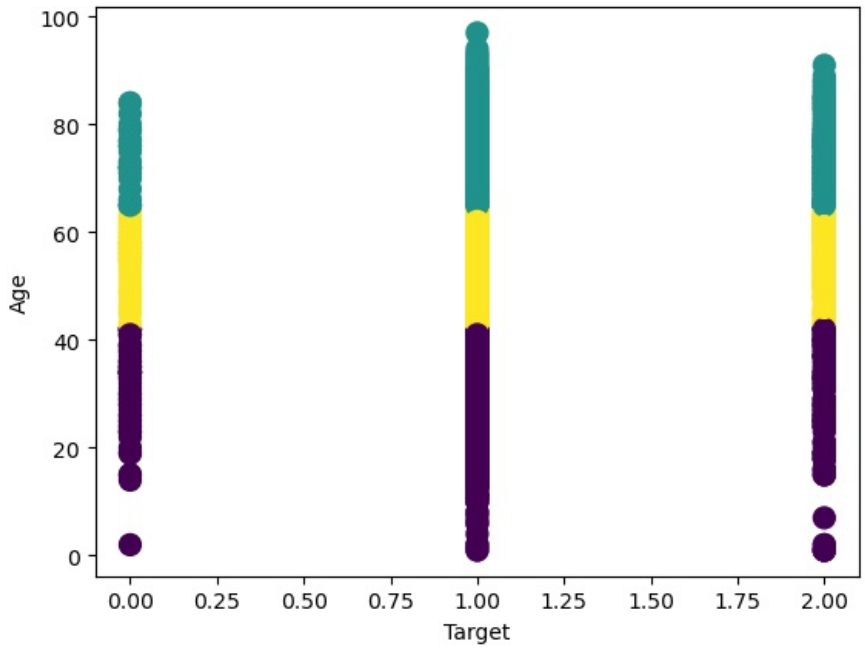
In [179…]
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df)
y_km = km.fit_predict(df)
```

```
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[179… Text(0.5, 0, 'Clustering')

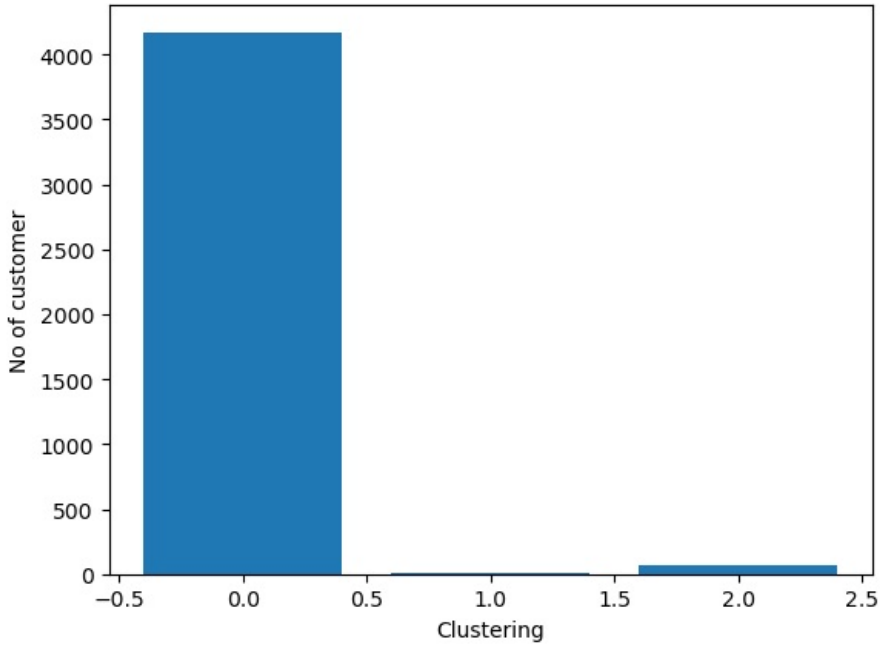


In [180…
```
plt.scatter(df['target'],
            df['age'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('Age')


plt.show()
```

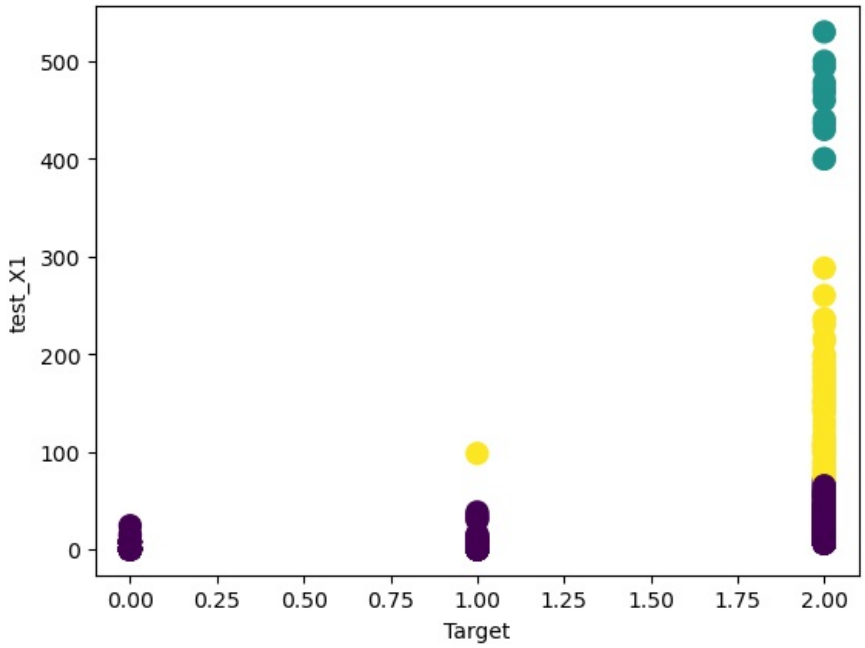


In [181…
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df1)
y_km = km.fit_predict(df1)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[181… Text(0.5, 0, 'Clustering')

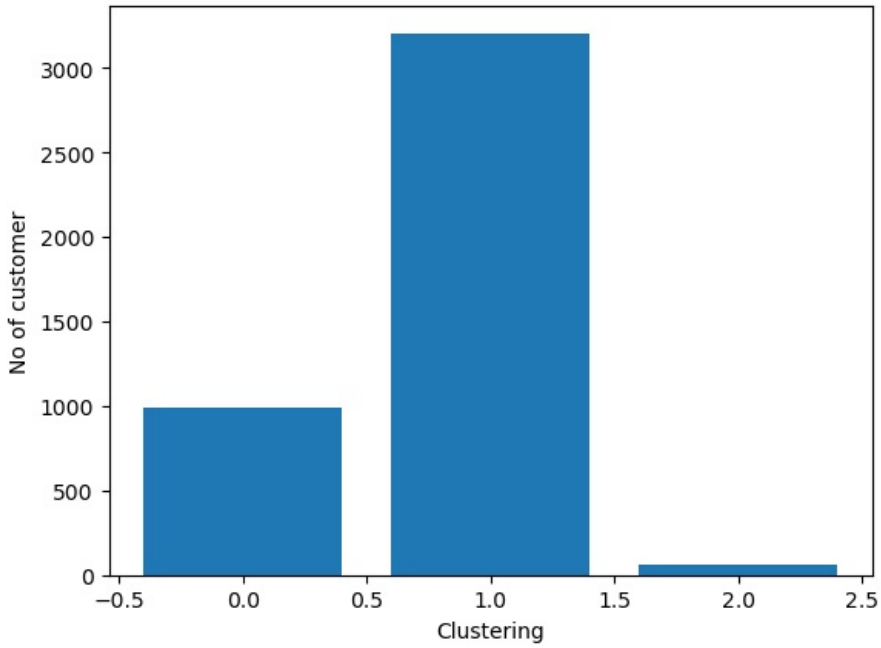


In [182…
```
plt.scatter(df1['target'],
            df1['test_X1'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X1')


plt.show()
```



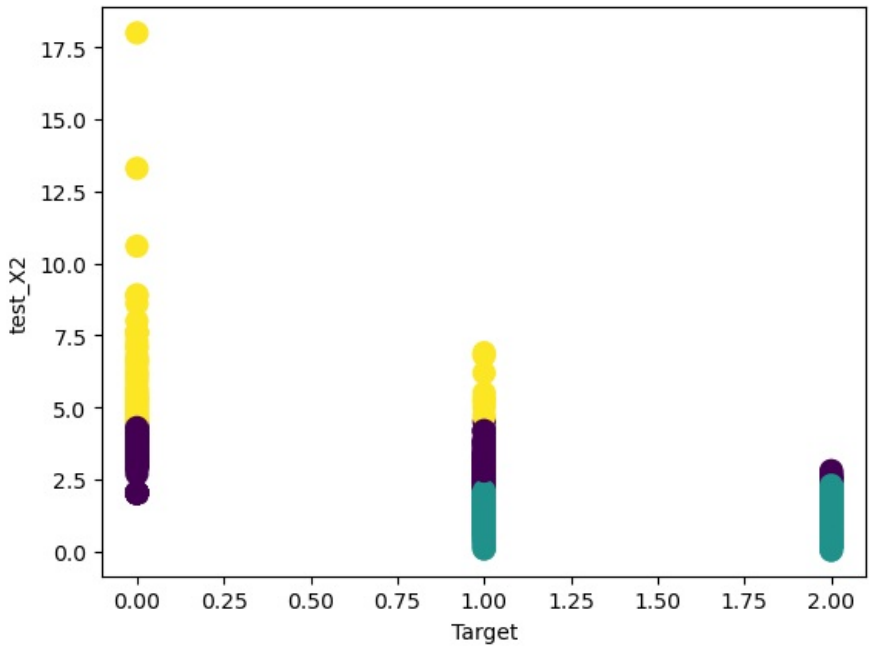
In [183…
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df2)
y_km = km.fit_predict(df2)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[183… Text(0.5, 0, 'Clustering')



In [184…

```
plt.scatter(df2['target'],
            df2['test_X2'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X2')


plt.show()
```

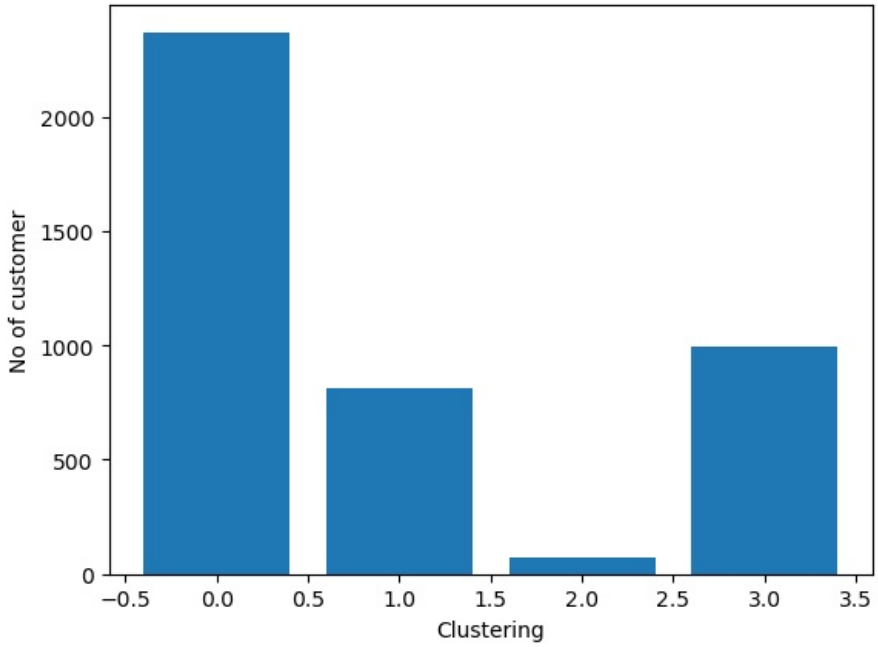


In [185…

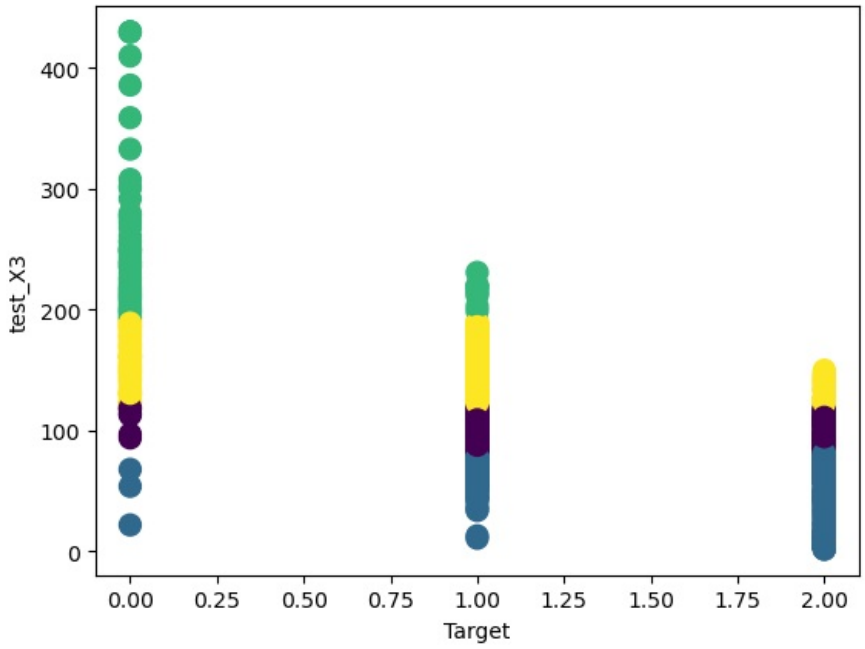
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=4).fit(df3)
y_km = km.fit_predict(df3)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[185…

```
Text(0.5, 0, 'Clustering')
```

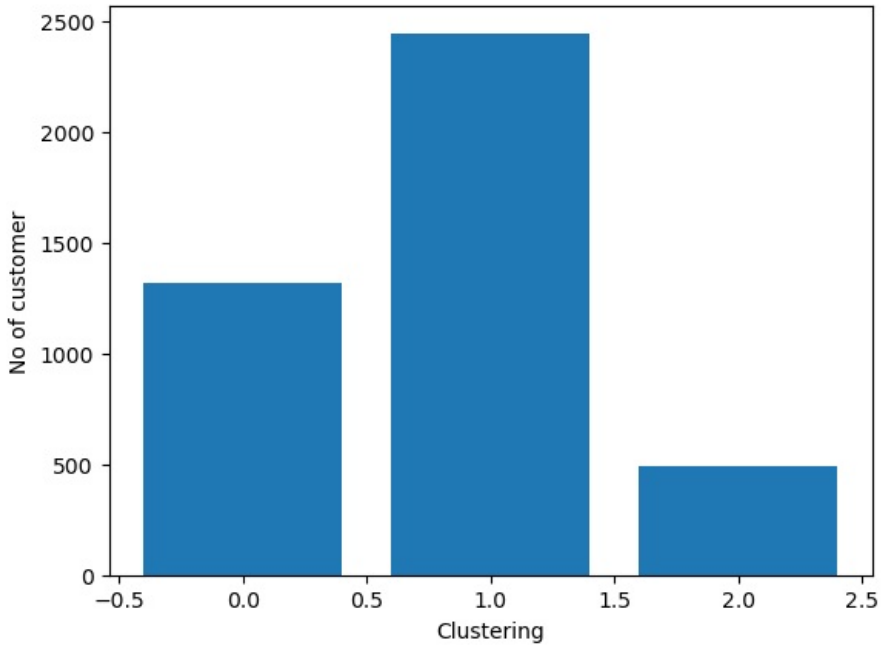


In [186…
```
plt.scatter(df3['target'],
            df3['test_X3'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X3')


plt.show()
```



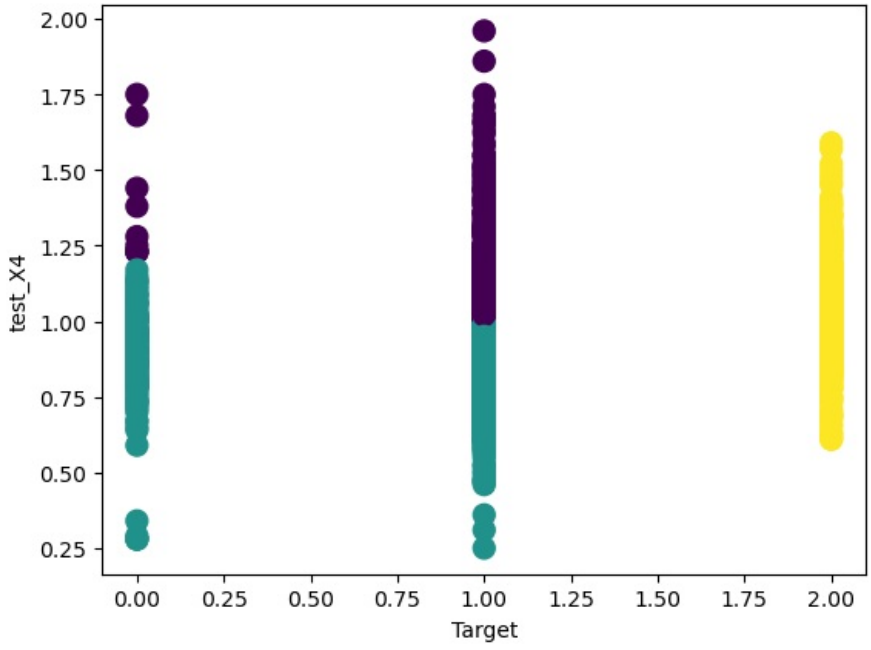
In [187…
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df4)
y_km = km.fit_predict(df4)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[187… Text(0.5, 0, 'Clustering')



In [188…

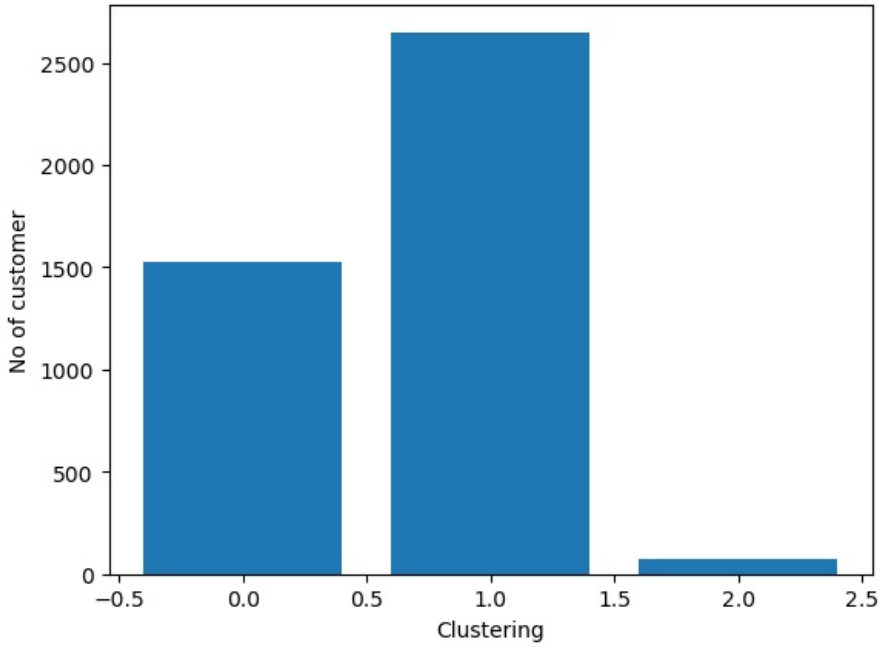
```
plt.scatter(df4['target'],
            df4['test_X4'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X4')


plt.show()
```



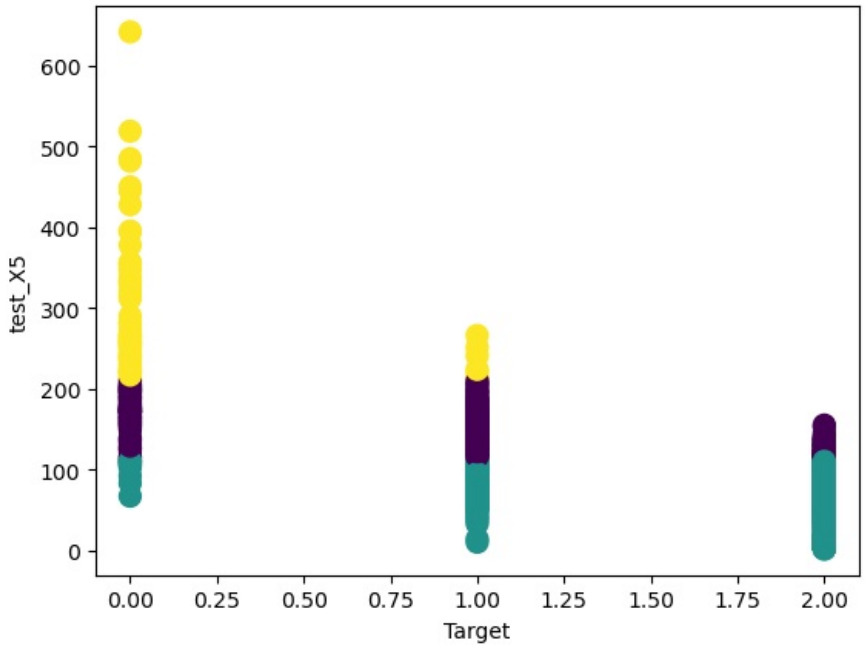
In [189…

```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df5)
y_km = km.fit_predict(df5)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[189… `Text(0.5, 0, 'Clustering')`

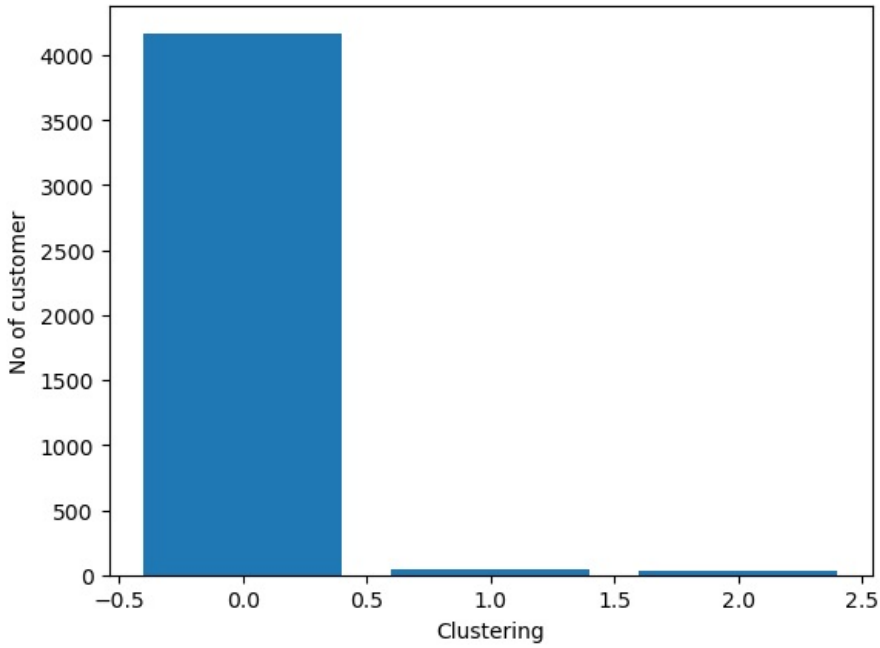


In [190…
```
plt.scatter(df5['target'],
            df5['test_X5'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X5')


plt.show()
```



In [191…
```
from sklearn.cluster import KMeans
# No of customers in each group respectively
km = KMeans(n_clusters=3).fit(df6)
y_km = km.fit_predict(df6)
n_cluster, km_count = np.unique(y_km, return_counts=True)

plt.bar(n_cluster, km_count)
plt.ylabel('No of customer')
plt.xlabel('Clustering')
```

Out[191… Text(0.5, 0, 'Clustering')



In [192…

```
plt.scatter(df6['target'],
            df6['test_X6'],
            c=y_km, s=100)

plt.xlabel('Target')
plt.ylabel('test_X6')


plt.show()
```