Final preprocessing / getting the final estimator

In [176]:

```
#!pip install --upgrade scikit-learn
```

In [2]:

```
#Loading the data and np numpy and pandas
import pandas as pd
import numpy as np
print("Setup complete.")
```

```
Setup complete.
```

In [3]:

```
#designate the path of the health train data
health_data_path = "health_train.csv"

#Load the data using pandas read_csv function.

health_data = pd.read_csv(health_data_path)

health_data.head()
```

Out[3]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x16	x17	x18	x19	x20	x21	x22	x
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.0	...	104.0	171.0	4.0	0.0	155.0	153.0	154.0	
1	PA1002	258	127.0	M	0.012	0.000	0.008	0.004	0.0	0.0	...	53.0	191.0	12.0	1.0	133.0	126.0	131.0	4
2	PA1003	479	145.0	F	0.000	0.000	0.000	0.002	0.0	0.0	...	111.0	157.0	1.0	1.0	150.0	146.0	149.0	
3	PA1004	906	146.0	F	0.004	0.000	0.005	0.003	0.0	0.0	...	107.0	169.0	2.0	2.0	150.0	147.0	149.0	
4	PA1005	1921	140.0	F	0.002	0.003	0.006	0.006	0.0	0.0	...	75.0	228.0	9.0	0.0	142.0	118.0	142.0	2

5 rows × 26 columns

In [4]:

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder

#find which columns are categorical
categorical_labels = ['target', 'x14', 'id', 'x3']

#encode thjese using the ordinal encoder

encoder = OrdinalEncoder()
x3_encoded = encoder.fit_transform(health_data[["x3"]])
print("x3 mapping: " , encoder.categories_)

x14_encoded = encoder.fit_transform(health_data[["x14"]])
print("x14 mapping: " , encoder.categories_)

label_enoder = LabelEncoder()
target_encoded = label_enoder.fit_transform(health_data["target"])
target_name_mapping = dict(zip(label_enoder.classes_, label_enoder.transform(label_enoder.classes_)))
print("\n Label mapping dictionary:")
print(target_name_mapping)
#target_encoded = label_enoder.transform(health_data[["target"]])
#print(target_name_mapping)

health_data
```

```
x3 mapping:  [array(['F', 'M'], dtype=object)]
x14 mapping:  [array(['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-'], dtype=object)]

 Label mapping dictionary:
{'High risk': 0, 'Low risk': 1, 'Moderate risk': 2}
```

Out[4]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x16	x17	x18	x19	x20	x21	x
0	PA1001	1406	145.0	F	0.005	0.000	0.002	0.000	0.0	0.000	...	104.0	171.0	4.0	0.0	155.0	153.0	154
1	PA1002	258	127.0	M	0.012	0.000	0.008	0.004	0.0	0.000	...	53.0	191.0	12.0	1.0	133.0	126.0	13
2	PA1003	479	145.0	F	0.000	0.000	0.000	0.002	0.0	0.000	...	111.0	157.0	1.0	1.0	150.0	146.0	149
3	PA1004	906	146.0	F	0.004	0.000	0.005	0.003	0.0	0.000	...	107.0	169.0	2.0	2.0	150.0	147.0	149
4	PA1005	1921	140.0	F	0.002	0.003	0.006	0.006	0.0	0.000	...	75.0	228.0	9.0	0.0	142.0	118.0	142
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1579	PA2580	2077	130.0	M	0.005	0.001	0.001	0.000	0.0	0.000	...	127.0	158.0	2.0	0.0	139.0	139.0	140
1580	PA2581	664	138.0	F	0.000	0.003	0.003	0.000	0.0	0.002	...	69.0	187.0	10.0	1.0	142.0	130.0	140
1581	PA2582	1431	144.0	F	0.000	0.000	0.006	0.000	0.0	0.000	...	139.0	169.0	2.0	0.0	157.0	155.0	157
1582	PA2583	630	134.0	F	0.017	0.002	0.004	0.000	0.0	0.000	...	50.0	170.0	5.0	0.0	160.0	150.0	155
1583	PA2584	436	151.0	F	0.000	0.000	0.006	0.006	0.0	0.000	...	50.0	200.0	11.0	2.0	156.0	150.0	156

1584 rows × 26 columns

In [5]:

```
health_data["x3"] = x3_encoded
health_data["x14"] = x14_encoded
health_data["target"] = target_encoded

health_data.head()
```

Out[5]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x16	x17	x18	x19	x20	x21	x22	
0	PA1001	1406	145.0	0.0	0.005	0.000	0.002	0.000	0.0	0.0	...	104.0	171.0	4.0	0.0	155.0	153.0	154.0	
1	PA1002	258	127.0	1.0	0.012	0.000	0.008	0.004	0.0	0.0	...	53.0	191.0	12.0	1.0	133.0	126.0	131.0	4
2	PA1003	479	145.0	0.0	0.000	0.000	0.000	0.002	0.0	0.0	...	111.0	157.0	1.0	1.0	150.0	146.0	149.0	
3	PA1004	906	146.0	0.0	0.004	0.000	0.005	0.003	0.0	0.0	...	107.0	169.0	2.0	2.0	150.0	147.0	149.0	
4	PA1005	1921	140.0	0.0	0.002	0.003	0.006	0.006	0.0	0.0	...	75.0	228.0	9.0	0.0	142.0	118.0	142.0	2

5 rows × 26 columns

In [6]:

```
#create new x train and y train etc.

#get the data out, leaving behind the target column (last feature).
X = health_data.iloc[:, 1:-1]
#extract the target column.
y = health_data["target"]

print(X)
print(y)
```

```
        x1     x2   x3     x4     x5     x6     x7   x8     x9   x10  ...  \
0     1406  145.0  0.0  0.005  0.000  0.002  0.000  0.0  0.000  46.0  ...
1      258  127.0  1.0  0.012  0.000  0.008  0.004  0.0  0.000  13.0  ...
2      479  145.0  0.0  0.000  0.000  0.000  0.002  0.0  0.000  57.0  ...
3      906  146.0  0.0  0.004  0.000  0.005  0.003  0.0  0.000  29.0  ...
4     1921  140.0  0.0  0.002  0.003  0.006  0.006  0.0  0.000  62.0  ...
...    ...    ...  ...    ...    ...    ...    ...  ...    ...   ...  ...
1579  2077  130.0  1.0  0.005  0.001  0.001  0.000  0.0  0.000  72.0  ...
1580   664  138.0  0.0  0.000  0.003  0.003  0.000  0.0  0.002  60.0  ...
1581  1431  144.0  0.0  0.000  0.000  0.006  0.000  0.0  0.000  45.0  ...
1582   630  134.0  0.0  0.017  0.002  0.004  0.000  0.0  0.000  48.0  ...
1583   436  151.0  0.0  0.000  0.000  0.006  0.006  0.0  0.000  64.0  ...

        x15    x16    x17   x18  x19    x20    x21    x22   x23  x24
0      67.0  104.0  171.0   4.0  0.0  155.0  153.0  154.0   4.0  1.0
1     138.0   53.0  191.0  12.0  1.0  133.0  126.0  131.0  41.0  0.0
2      46.0  111.0  157.0   1.0  1.0  150.0  146.0  149.0   6.0  1.0
3      62.0  107.0  169.0   2.0  2.0  150.0  147.0  149.0   7.0  0.0
4     153.0   75.0  228.0   9.0  0.0  142.0  118.0  142.0  20.0  0.0
...     ...    ...    ...   ...  ...    ...    ...    ...   ...  ...
1579   31.0  127.0  158.0   2.0  0.0  139.0  139.0  140.0   3.0  0.0
1580  118.0   69.0  187.0  10.0  1.0  142.0  130.0  140.0  61.0  0.0
1581   30.0  139.0  169.0   2.0  0.0  157.0  155.0  157.0   2.0  0.0
1582  120.0   50.0  170.0   5.0  0.0  160.0  150.0  155.0  28.0  1.0
1583  150.0   50.0  200.0  11.0  2.0  156.0  150.0  156.0  38.0  1.0

[1584 rows x 24 columns]
0       1
1       1
2       1
3       1
4       1
       ..
1579    1
1580    2
1581    2
1582    1
1583    2
Name: target, Length: 1584, dtype: int32
```

In [7]:

```
from sklearn.model_selection import train_test_split


X_train, X_validate, y_train, y_validate = train_test_split(X, y, test_size=0.3, random_state=3)
```

In [14]:

```
def OutlierRemoverFuncSampler(X, y, strategy):
    strategy.fit(X)
    preds = strategy.predict(X)

    totalOutliers=0
    for pred in preds:
        if pred == -1:
            totalOutliers=totalOutliers+1
    #print("Total number of outliers identified is: ",totalOutliers)

    # select all rows that are not outliers and create a boolean mask
    mask = preds != -1

    # Apply mask to X and y and check shape
    return (X[mask], y[mask])
```

The previous best setup was SelectKBest() followed by SMOTE(): best balanced accuracy. (also best high risk recall of feature selection then smote) balanced accuracy: 0.866 (3d.p), accuracy: 0.883 (3.d.p), HR recall : 0.872 (3.d.p) NEW with best recall - SelectKBest(k=5) followed by ADASYN(): balanced accuracy : 0.836 (3d.p), accuracy : 0.794 (3.d.p), HR recall : 0.911 (3.d.p)

The other best in class combinations do not compare to these two.

With the new statistics about the recall of the models, and access to the model with the best recall for high risk patients, I am choosing to stick with SelectKBest() followed by SMOTE(). Its recall for high risk patients is still very good, and not too different from the best high risk recall score, whereas its balanced accuracy is better.

However, when deciding the estimator, it may be interesting to swap around the pipeline so it uses a SelectKBest(k=5) followed by ADASYN() pipeline, and see the differences.

Next, we are working out what we want out of feature discretisation, feature agglomeration, and some of sklearns transformers. originally I was going to work outwhether it would be better to have feature discretisation or feature construction first - but feature discretisation helps linear classifiers classify, and we have been clasifying using an SVC. therefore I am going to wait until we are trying lots of different classifiers in a gridsearchCV at the end, and include feature discretisation as an optional step when looking through all the combinations.

Feature construction also adjusts the number of clusters, which may effect different classifiers differently in the end because of they different ways they work. For this reason, we will add this to the combinations of things which maybe used.

Lets get the possible combinations of feature discretisation:

In [196]:

```
#getting KBinsDiscretizer Configurations
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import ParameterGrid

discretizer_params = [
    (KBinsDiscretizer, {
        "n_bins" : [2,3,5,10,20],
        "encode" : ["onehot", "onehot-dense", "ordinal"],
        "strategy" : ["uniform", "quantile", "kmeans"],
        "random_state" : [1]
    })
]

discretizers = ["passthrough"] + [ctor(**para) for ctor, paras in discretizer_params for para in ParameterG
len(discretizers)
```

Out[196]:

46

In [195]:

```
#Getting FeatureAgglomeration configurations
from sklearn.cluster import FeatureAgglomeration

agglomeration_params = [
    (FeatureAgglomeration, {
        "n_clusters" : [2,3,5,10,20,None],
        "compute_full_tree" : [True, False, "Auto"],
        "linkage" : ["ward", "complete", "average", "single"],
        "compute_distances" : [True, False]
    })
]

agglomeration_params_with_threshold = [
    (FeatureAgglomeration, {
        "n_clusters" : [None],
        "distance_threshold" : [0.1, 0.25 ,0.5 ,1 ,2 ,5 , 10, 20],
        "compute_full_tree" : [True],
    })
]

agglomeration_no_threshold = [ctor(**para) for ctor, paras in agglomeration_params for para in ParameterGr
agglomeration_with_threshold = [ctor(**para) for ctor, paras in agglomeration_params_with_threshold for pa

agglomerators = ["passthrough"] + agglomeration_no_threshold + agglomeration_with_threshold
len(agglomerators)
```

Out[195]:

153

In [194]:

```
#getting transformer Configurations
from sklearn.preprocessing import StandardScaler, MaxAbsScaler, MinMaxScaler, QuantileTransformer, PowerTra
#not using box-cox transform method for powertransformer, as some scalers may make values negative.

transformer_params = [
    (QuantileTransformer, {
        "n_quantiles" : [500,1000,1500,2000],
        "output_distribution" : ["uniform", "normal"],
        "random_state" : [1]
    }),
    (Normalizer , {
       "norm" : ["l1", "l2", "max"]
    })
]


transformers_with_params = [ctor(**para) for ctor, paras in transformer_params for para in ParameterGrid(pa
print(transformers_with_params)

transformers = ["passthrough", StandardScaler(), MaxAbsScaler(), MinMaxScaler(), PowerTransformer()] + tra
len(transformers)
```

```
[QuantileTransformer(n_quantiles=500, random_state=1), QuantileTransformer(n_quantiles=500,
output_distribution='normal',
                    random_state=1), QuantileTransformer(random_state=1), QuantileTransforme
r(output_distribution='normal', random_state=1), QuantileTransformer(n_quantiles=1500, rando
m_state=1), QuantileTransformer(n_quantiles=1500, output_distribution='normal',
                    random_state=1), QuantileTransformer(n_quantiles=2000, random_state=1),
QuantileTransformer(n_quantiles=2000, output_distribution='normal',
                    random_state=1), Normalizer(norm='l1'), Normalizer(), Normalizer(norm='m
ax')]
```

Out[194]:

16

In [193]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, HistGradientBoostingClassifier

clf_params = [
                (SVC, {
                    "kernel": ["linear", "poly", "rbf", "sigmoid", "precomputed"],
                    "C": [0.01, 0.1, 1.0, 2.0],
                    "shrinking" : [True, False],
                    "random_state" : [1]
                }),
                (LogisticRegression,
                 {"penalty" : ["l1", "l2", "elasticnet", "none"],
                  "C": [0.01, 0.1, 1.0, 2.0],
                  "random_state" : [1]
                 }),
                (DecisionTreeClassifier,
                 {"criterion" : ["gini", "entropy", "log_loss"],
                  "splitter" : ["best", "random"],
                  "random_state" : [1]
                 }),
                (KNeighborsClassifier,
                 {"n_neighbors" : [3, 5, 10, 20, 40],
                  "weights" : ["uniform", "distance"],
                  "algorithm" : ["auto", "ball_tree", "kd_tree", "brute"]
                 }),
                (MLPClassifier,
                 {"hidden_layer_sizes" : [(100,), (150,), (200,)],
                  "activation" : ["identity", "logistic", "tanh", "relu"],
                  "solver" : ["adam"], #we are using the adam solver becuase it works well on training sets
                  "learning_rate" : ["constant", "invscaling", "adaptive"],
                  "random_state" : [1]
                 }),
                (RandomForestClassifier,
                 {"n_estimators" : [50,100,150,200],
                  "criterion" : ["gini", "entropy", "log_loss"],
                  "min_samples_leaf" : [1,2,4,8],
                  "max_features" : ["sqrt", "log2", None],
                  "random_state" : [1]
                 }),
                (AdaBoostClassifier,
                 {"estimator" : [DecisionTreeClassifier(), LogisticRegression(), SVC(), KNeighborsClassifie
                  "n_estimators" : [25,50,75,100],
                  "learning_rate" : [0.0, 0.5, 1.0, 2.0, 5.0, 10.0],
                  "random_state" : [1]
                 }),
                (HistGradientBoostingClassifier,
                 {"loss" : ["log_loss", "auto", "binary_crossentropy", "categorical_crossentropy"],
                  "learning_rate" : [0.1,0.5,1,2],
                  "min_samples_leaf": [10,20,40],
                  "categorical_features" : [["x3", "x14"]],
                  "random_state" : [1]
                 })
            ]

svc_poly_params = [
    (SVC, {"kernel": ["poly"], "C": [0.01, 0.1, 1.0, 2.0], "degree" : [1,2,3,5,7], "shrinking" : [True, Fa
]

clfs_without_svc_stuff = [ctor(**para) for ctor, paras in clf_params for para in ParameterGrid(paras)]
svcs = [ctor(**para) for ctor, paras in svc_poly_params for para in ParameterGrid(paras)]

clfs = clfs_without_svc_stuff + svcs
len(clfs)
```

```
#clfs
```

Out[193]:

```
490
```

In [30]:

```
params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest()],
              balancer = [SMOTE()],
              transformer= transformers,
              agglomerator = agglomerators,
              discretizer = discretizers,
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest()),
           ("balancer", SMOTE()),
           ("transformer", StandardScaler()),
           ("agglomerator", FeatureAgglomeration()),
           ("discretizer", KBinsDiscretizer()),
           ("clf", SVC)]

#clf_cv = GridSearchCV(Pipeline(process), params)

#clf_cv.fit(X_train, y_train)
#score = clf_cv.score(X_train, y_train)
#print(score)

#print(clf_cv.best_params_)
```

Doing it like this gives 55,177,920 (490 * 16 * 153 * 46) combinations which may be too many. In aniother tab, i am going to time how long it takes to do just the clfs - 490 combinations, and compare them.

In [ ]:

```
params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest()],
              balancer = [SMOTE()],
              discretizer = discretizers,
              agglomerator = agglomerators,
              transformer= transformers,
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest()),
           ("balancer", SMOTE()),
           ("discretizer", KBinsDiscretizer()),
           ("agglomerator", FeatureAgglomeration()),
           ("transformer", StandardScaler()),
           ("clf", SVC)]

clf_cv = GridSearchCV(Pipeline(process), params)

clf_cv.fit(X_train, y_train)
score = clf_cv.score(X_train, y_train)
print(score)

print(clf_cv.best_params_)
```

just tested out how long it would take to get the best CLF using just the estimators - so 490 combinations (compared to the 55 million of the code above.) Its taken over 13 minutes already. 55,177,920 / 490 = 112608 (112608 * 13) / 60 = 24398.4 hours. hmm. this isnt going to be finished processing before the deadline!!!

Now, I am going to reduce the number of combinations of each step., while leaving the original 490 running in another tab in the background. I think the fact that some of the ensembles have 50 - 200 estimators within them will be slowing things down A LOT.

In [72]:

```
#Getting FeatureAgglomeration configurations
from sklearn.cluster import FeatureAgglomeration

agglomeration_params = [
    (FeatureAgglomeration, {
        "n_clusters" : [2,5,None],
        "linkage" : ["ward", "complete", "average", "single"],
    })
]

agglomeration_params_with_threshold = [
    (FeatureAgglomeration, {
        "n_clusters" : [None],
        "distance_threshold" : [0.25 ,1 , 10,],
        "compute_full_tree" : [True],
    })
]

agglomeration_no_threshold = [ctor(**para) for ctor, paras in agglomeration_params for para in ParameterGr
agglomeration_with_threshold = [ctor(**para) for ctor, paras in agglomeration_params_with_threshold for pa

agglomerators = ["passthrough"] + agglomeration_no_threshold + agglomeration_with_threshold
print(agglomerators)
print(len(agglomerators))
#params = dict(impute = [SimpleImputer()], scale = [StandardScaler()], outlier = [FunctionSampler(func=Outl
#params
```

```
['passthrough', FeatureAgglomeration(), FeatureAgglomeration(n_clusters=5), FeatureAgglomera
tion(n_clusters=None), FeatureAgglomeration(linkage='complete'), FeatureAgglomeration(linkag
e='complete', n_clusters=5), FeatureAgglomeration(linkage='complete', n_clusters=None), Feat
ureAgglomeration(linkage='average'), FeatureAgglomeration(linkage='average', n_clusters=5),
FeatureAgglomeration(linkage='average', n_clusters=None), FeatureAgglomeration(linkage='sing
le'), FeatureAgglomeration(linkage='single', n_clusters=5), FeatureAgglomeration(linkage='si
ngle', n_clusters=None), FeatureAgglomeration(compute_full_tree=True, distance_threshold=0.2
5,
                      n_clusters=None), FeatureAgglomeration(compute_full_tree=True, distance
_threshold=1,
                      n_clusters=None), FeatureAgglomeration(compute_full_tree=True, distance
_threshold=10,
                      n_clusters=None)]
16
```

In [73]:

```
#getting KBinsDiscretizer Configurations
from sklearn.preprocessing import KBinsDiscretizer

discretizer_params = [
    (KBinsDiscretizer, {
        "n_bins" : [2,5,10],
        "strategy" : ["uniform", "quantile", "kmeans"],
        "random_state" : [1]
    })
]
#choosing to omit "constant" strategy, as it is just filling in all missing values with a constant, and doe
#need to check how my missing values are represented - I assume np.nan

discretizers = ["passthrough"] + [ctor(**para) for ctor, paras in discretizer_params for para in ParameterG
print(discretizers)
print(len(discretizers))
```

```
['passthrough', KBinsDiscretizer(n_bins=2, random_state=1, strategy='uniform'), KBinsDiscret
izer(n_bins=2, random_state=1), KBinsDiscretizer(n_bins=2, random_state=1, strategy='kmean
s'), KBinsDiscretizer(random_state=1, strategy='uniform'), KBinsDiscretizer(random_state=1),
KBinsDiscretizer(random_state=1, strategy='kmeans'), KBinsDiscretizer(n_bins=10, random_stat
e=1, strategy='uniform'), KBinsDiscretizer(n_bins=10, random_state=1), KBinsDiscretizer(n_bi
ns=10, random_state=1, strategy='kmeans')]
10
```

In [80]:

```
#getting transformer Configurations
from sklearn.preprocessing import StandardScaler, MaxAbsScaler, MinMaxScaler, QuantileTransformer, PowerTra
#not using box-cox transform method for powertransformer, as some scalers may make values negative.

transformer_params = [
    (QuantileTransformer, {
        "random_state" : [1]
    }),
    (Normalizer , {
       "norm" : ["l2"]
    })
]
#choosing to omit "constant" strategy, as it is just filling in all missing values with a constant, and doe
#need to check how my missing values are represented - I assume np.nan

transformers_with_params = [ctor(**para) for ctor, paras in transformer_params for para in ParameterGrid(pa
print(transformers_with_params)

transformers = ["passthrough", StandardScaler(), MaxAbsScaler(), MinMaxScaler(), PowerTransformer()] + tran
transformers
```

```
[QuantileTransformer(random_state=1), Normalizer()]
```

Out[80]:

```
['passthrough',
 StandardScaler(),
 MaxAbsScaler(),
 MinMaxScaler(),
 PowerTransformer(),
 QuantileTransformer(random_state=1),
 Normalizer()]
```

In [69]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, HistGradientBoostingClassifier

clf_params = [
                (SVC, {
                    "kernel": ["linear", "poly", "rbf", "sigmoid", "precomputed"],
                    "C": [0.01, 0.1, 1.0],
                    "random_state" : [1]
                 }),
                (LogisticRegression,
                 {"penalty" : ["l2"],
                  "C": [0.01, 0.1, 1.0],
                  "random_state" : [1]
                 }),
                (DecisionTreeClassifier,
                 {"criterion" : ["gini", "entropy"],
                  "splitter" : ["best"],
                  "random_state" : [1]
                 }),
                (KNeighborsClassifier,
                 {"n_neighbors" : [3, 5, 15],
                  "weights" : ["uniform"],
                  "algorithm" : ["auto"]
                 }),
                (MLPClassifier,
                 {"activation" : ["identity", "logistic", "tanh", "relu"],
                  "solver" : ["adam"], #we are using the adam solver becuase it works well on training sets
                  "random_state" : [1]
                 }),
                (RandomForestClassifier,
                 {"n_estimators" : [50,100],
                  "min_samples_leaf" : [1,2,4,8],
                  "random_state" : [1]
                 })
             ]



clfs= [ctor(**para) for ctor, paras in clf_params for para in ParameterGrid(paras)]

clfs
```

Out[69]:

```
[SVC(C=0.01, kernel='linear', random_state=1),
 SVC(C=0.01, kernel='poly', random_state=1),
 SVC(C=0.01, random_state=1),
 SVC(C=0.01, kernel='sigmoid', random_state=1),
 SVC(C=0.01, kernel='precomputed', random_state=1),
 SVC(C=0.1, kernel='linear', random_state=1),
 SVC(C=0.1, kernel='poly', random_state=1),
 SVC(C=0.1, random_state=1),
 SVC(C=0.1, kernel='sigmoid', random_state=1),
 SVC(C=0.1, kernel='precomputed', random_state=1),
 SVC(kernel='linear', random_state=1),
 SVC(kernel='poly', random_state=1),
 SVC(random_state=1),
 SVC(kernel='sigmoid', random_state=1),
 SVC(kernel='precomputed', random_state=1),
 LogisticRegression(C=0.01, random_state=1),
 LogisticRegression(C=0.1, random_state=1),
 LogisticRegression(random_state=1),
 DecisionTreeClassifier(random_state=1),
 DecisionTreeClassifier(criterion='entropy', random_state=1),
 KNeighborsClassifier(n_neighbors=3),
 KNeighborsClassifier(),
 KNeighborsClassifier(n_neighbors=15),
 MLPClassifier(activation='identity', random_state=1),
 MLPClassifier(activation='logistic', random_state=1),
 MLPClassifier(activation='tanh', random_state=1),
 MLPClassifier(random_state=1),
 RandomForestClassifier(n_estimators=50, random_state=1),
 RandomForestClassifier(random_state=1),
 RandomForestClassifier(min_samples_leaf=2, n_estimators=50, random_state=1),
 RandomForestClassifier(min_samples_leaf=2, random_state=1),
 RandomForestClassifier(min_samples_leaf=4, n_estimators=50, random_state=1),
 RandomForestClassifier(min_samples_leaf=4, random_state=1),
 RandomForestClassifier(min_samples_leaf=8, n_estimators=50, random_state=1),
 RandomForestClassifier(min_samples_leaf=8, random_state=1)]
```

This has massively reduced the number of clfs. The 490 clfs that are running in another tab (with no discretization, agglomeration or transformation) have still not finished running after 50 minutes. I am halting this process, as it clearly will take far too long - especially as all the ensembles could have 50, 100, 150 or 200 estimators within them. At that rate it would take at least 10.7123 years for the grid search to finish… ((55,177,920 / 490 * 50) / 60 / 24 / 365)

First I am going to try running all the clfs on there own with no prior steps to get an idea of time.

In [29]:

```
params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest()],
              balancer = [SMOTE()],
              transformer= ["passthrough"],
              agglomerator = ["passthrough"],
              discretizer = ["passthrough"],
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest()),
           ("balancer", SMOTE()),
           ("transformer", StandardScaler()),
           ("agglomerator", FeatureAgglomeration()),
           ("discretizer", KBinsDiscretizer()),
           ("clf", SVC)]

#clf_cv = GridSearchCV(Pipeline(process), params)

#clf_cv.fit(X_train, y_train)
#score = clf_cv.score(X_train, y_train)
#print(score)

#print(clf_cv.best_params_)
```

That only took a minute and a half. now lets work out the rough time it would take to check with all the other options: number of transformers: 7 number of agglomerators: 16 number of discretizers : 10

7 * 10 * 16 * 1.5 minutees = 1680 minutes, or 28 hours. I am going to reduce the number of discretizers and agglomerators again.

In [17]:

```
#getting KBinsDiscretizer Configurations
from sklearn.preprocessing import KBinsDiscretizer

discretizer_params = [
    (KBinsDiscretizer, {
        "n_bins" : [2,5],
        "random_state" : [1]
    })
]
#choosing to omit "constant" strategy, as it is just filling in all missing values with a constant, and doe
#need to check how my missing values are represented - I assume np.nan

discretizers = ["passthrough"] + [ctor(**para) for ctor, paras in discretizer_params for para in ParameterG
print(discretizers)
print(len(discretizers))
```

```
['passthrough', KBinsDiscretizer(n_bins=2, random_state=1), KBinsDiscretizer(random_state=
1)]
3
```

In [18]:

```
#Getting FeatureAgglomeration configurations
from sklearn.cluster import FeatureAgglomeration

agglomeration_params = [
    (FeatureAgglomeration, {
        "n_clusters" : [2,5]
    })
]

#choosing to omit "constant" strategy, as it is just filling in all missing values with a constant, and doe
#need to check how my missing values are represented - I assume np.nan

agglomeration = [ctor(**para) for ctor, paras in agglomeration_params for para in ParameterGrid(paras)]

agglomerators = agglomeration
print(agglomerators)
print(len(agglomerators))
```

```
[FeatureAgglomeration(), FeatureAgglomeration(n_clusters=5)]
2
```

now lets work out the rough time it would take to check all options: number of transformers: 7 number of agglomerators: 4 number of discretizers : 4

7 * 3 * 3 * 1.5 = 94 minutes, or 1.5 hours. I am going to reduce the number of scalers.

In [19]:

```
#getting transformer Configurations
from sklearn.preprocessing import StandardScaler, MaxAbsScaler, MinMaxScaler, QuantileTransformer, PowerTra
#not using box-cox transform method for powertransformer, as some scalers may make values negative.


transformers = ["passthrough", StandardScaler(), MaxAbsScaler(), PowerTransformer(), Normalizer()]
transformers
```

Out[19]:

```
['passthrough',
 StandardScaler(),
 MaxAbsScaler(),
 PowerTransformer(),
 Normalizer()]
```

5 * 3 * 3 * 1.5 = 67 = 1 hour. I am going to reduce the number of estimator combinations and re time it to get a new idea of the time.

In [20]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, HistGradientBoostingClassifier

clf_params = [
                (SVC, {
                    "kernel": ["linear", "poly"],
                    "C": [0.01, 0.1, 1.0],
                    "random_state" : [1]
                }),
                (LogisticRegression,
                 {"penalty" : ["l2"],
                  "C": [0.01, 0.1, 1.0],
                  "random_state" : [1]
                 }),
                (DecisionTreeClassifier,
                 {"splitter" : ["best"],
                  "random_state" : [1]
                 }),
                (KNeighborsClassifier,
                 {"n_neighbors" : [3, 5, 15],
                  "weights" : ["uniform"],
                  "algorithm" : ["auto"]
                 }),
                (MLPClassifier,
                 {"activation" : ["identity", "logistic", "tanh", "relu"],
                  "solver" : ["adam"], #we are using the adam solver becuase it works well on training sets
                  "random_state" : [1]
                 }),
                (RandomForestClassifier,
                 {"n_estimators" : [35,70],
                  "min_samples_leaf" : [1,2,],
                  "random_state" : [1]
                 })
            ]



clfs= [ctor(**para) for ctor, paras in clf_params for para in ParameterGrid(paras)]

clfs
```

Out[20]:

```
[SVC(C=0.01, kernel='linear', random_state=1),
 SVC(C=0.01, kernel='poly', random_state=1),
 SVC(C=0.1, kernel='linear', random_state=1),
 SVC(C=0.1, kernel='poly', random_state=1),
 SVC(kernel='linear', random_state=1),
 SVC(kernel='poly', random_state=1),
 LogisticRegression(C=0.01, random_state=1),
 LogisticRegression(C=0.1, random_state=1),
 LogisticRegression(random_state=1),
 DecisionTreeClassifier(random_state=1),
 KNeighborsClassifier(n_neighbors=3),
 KNeighborsClassifier(),
 KNeighborsClassifier(n_neighbors=15),
 MLPClassifier(activation='identity', random_state=1),
 MLPClassifier(activation='logistic', random_state=1),
 MLPClassifier(activation='tanh', random_state=1),
 MLPClassifier(random_state=1),
 RandomForestClassifier(n_estimators=35, random_state=1),
 RandomForestClassifier(n_estimators=70, random_state=1),
 RandomForestClassifier(min_samples_leaf=2, n_estimators=35, random_state=1),
 RandomForestClassifier(min_samples_leaf=2, n_estimators=70, random_state=1)]
```

In [28]:

```
params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest()],
              balancer = [SMOTE()],
              transformer= ["passthrough"],
              agglomerator = ["passthrough"],
              discretizer = ["passthrough"],
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest()),
           ("balancer", SMOTE()),
           ("transformer", StandardScaler()),
           ("agglomerator", FeatureAgglomeration()),
           ("discretizer", KBinsDiscretizer()),
           ("clf", SVC)]

clf_cv = GridSearchCV(Pipeline(process), params)

#clf_cv.fit(X_train, y_train)
#score = clf_cv.score(X_train, y_train)
#print(score)

#print(clf_cv.best_params_)
```

this only took 50 seconds.

5 * 3 * 3 * 0.83 = 37 minutes

to reduce times more, we are going to put agglomerisation and discretisation as the same step. not all classifiers will be aided by agglomeration and not all by discretisation - they do opposite things, so we should put them in the same step so its either one or the other. hopefully we can see which combination provides the best out of these. it should only take 5 * 5 * 0.83 = 20 minutes.

In [92]:

```
params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest()],
              balancer = [SMOTE()],
              transformer= transformers,
              feature_edit = (agglomerators + discretizers),
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest()),
           ("balancer", SMOTE()),
           ("transformer", StandardScaler()),
           ("feature_edit", FeatureAgglomeration()),
           ("clf", SVC)]

clf_cv = GridSearchCV(Pipeline(process), params)

clf_cv.fit(X_train, y_train)
score = clf_cv.score(X_train, y_train)
print(score)

print(clf_cv.best_params_)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.064321595011961 > -146.100691775009864). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(

0.9972924187725631
{'balancer': SMOTE(), 'clf': RandomForestClassifier(n_estimators=70, random_state=1), 'fea
t_selector': SelectKBest(), 'feature_edit': 'passthrough', 'imputer': SimpleImputer(), 'ou
tlier_remover': FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x00000237F09A
E0D0>,
                kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                      random_state=1,
                                                      support_fraction=0.75)}), 'scaler':
StandardScaler(), 'transformer': MaxAbsScaler()}


 0.9972924187725631
{'balancer': SMOTE(), 'clf': RandomForestClassifier(n_estimators=70, random_state=1), 'feat_selector':
SelectKBest(), 'feature_edit': 'passthrough', 'imputer': SimpleImputer(), 'outlier_remover':
FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x00000237F09AE0D0>,
                kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                      random_state=1,
                                                      support_fraction=0.75)}), 'scaler':
StandardScaler(), 'transformer': MaxAbsScaler()}
```

Ok. From this we got the best parameters being no discretization or agglomeration, for transformer we got MaxAbsScaler() and for our estimator RandomForestClassifier(n_estimators=70, random_state=1).

This was with mean accuracy - random forests scoring metric is mean accuracy. Now lets try this again with the balanced accuracy.

In [93]:

```
clf_cv2 = GridSearchCV(Pipeline(process), params, scoring = "balanced_accuracy")

clf_cv2.fit(X_train, y_train)
score2 = clf_cv2.score(X_train, y_train)
print(score2)

print(clf_cv2.best_params_)
```

```
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(

0.9547467758805128
{'balancer': SMOTE(), 'clf': MLPClassifier(random_state=1), 'feat_selector': SelectKBest
(), 'feature_edit': 'passthrough', 'imputer': SimpleImputer(), 'outlier_remover': Function
Sampler(func=<function OutlierRemoverFuncSampler at 0x00000237F09AE0D0>,
                kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                      random_state=1,
                                                      support_fraction=0.75)}), 'scaler':
StandardScaler(), 'transformer': StandardScaler()}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  warnings.warn(
```

```
0.9547467758805128
{'balancer': SMOTE(), 'clf': MLPClassifier(random_state=1), 'feat_selector': SelectKBest(), 'featu
re_edit': 'passthrough', 'imputer': SimpleImputer(), 'outlier_remover': FunctionSampler(func=<func
tion OutlierRemoverFuncSampler at 0x00000237F09AE0D0>,
                kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                      random_state=1,
                                                      support_fraction=0.75)}), 'scaler': Standard
Scaler(), 'transformer': StandardScaler()}
```

For balnced accuracy, mlp classifier(random_state=1), with no discritization or agglomeration, and a standard scaler, scores the best.

now lets look at the best recall on high risk patients.

In [99]:

```
clf_cv3 = GridSearchCV(Pipeline(process), params, scoring = make_scorer(score_func=recall_score, pos_label=

clf_cv3.fit(X_train, y_train)
score3 = clf_cv3.score(X_train, y_train)
print(score3)

print(clf_cv3.best_params_)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-84.444205807837690 > -242.913761138954726). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.064321595011961 > -146.100691775009864). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWar
ning: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You m
ay use labels=[pos_label] to specify a single positive class.
  warnings.warn(


  [1.]
  {'balancer': SMOTE(), 'clf': KNeighborsClassifier(n_neighbors=3), 'feat_selector': SelectKBest(),
  'feature_edit': 'passthrough', 'imputer': SimpleImputer(), 'outlier_remover': FunctionSampler(func
  =<function OutlierRemoverFuncSampler at 0x00000237F09AE0D0>,
                  kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                        random_state=1,
                                                        support_fraction=0.75)}), 'scaler': Standard
  Scaler(), 'transformer': Normalizer()}
```

In [102]:

```
score3[0]
```

Out[102]:

```
1.0
```

According to this - a recall of 1 on high risk patients is possible with the combination of normalizer, KNeighboursClassifier(n_neighbours = 3) and no discretization or agglomeration. This seems dubious though, and will be investigated further.

Now we will try running the grid search, finding the best of these three metrics again, but with SelectKBest(k=5) followed by ADASYN() rather than SelectKBest() followed by SMOTE(). Once this is completed, all the metrics of all 6 best pipelines will be calculated.

In [25]:

```
from sklearn.metrics import precision_score
from sklearn.metrics import make_scorer
from imblearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from imblearn import FunctionSampler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.feature_selection import SelectKBest
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.covariance import EllipticEnvelope

params = dict(imputer = [SimpleImputer()],
              scaler = [StandardScaler()],
              outlier_remover = [FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Ellip
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})],
              feat_selector = [SelectKBest(k=5)],
              balancer = [ADASYN()],
              transformer= transformers,
              feature_edit = (agglomerators + discretizers),
              clf = clfs)

from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

process = [("imputer", SimpleImputer()),
           ("scaler" , StandardScaler()),
           ("outlier_remover", FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" :Elliptic
                  contamination=0.01,
                  random_state=1,
                  support_fraction=0.75)})),
           ("feat_selector", SelectKBest(k=5)),
           ("balancer", ADASYN()),
           ("transformer", StandardScaler()),
           ("feature_edit", FeatureAgglomeration()),
           ("clf", SVC)]
```

In [105]:

```
clf_cv4 = GridSearchCV(Pipeline(process), params)

clf_cv4.fit(X_train, y_train)
score4 = clf_cv4.score(X_train, y_train)
print(score4)

print(clf_cv4.best_params_)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-86.878554642322555 > -243.742705489545983). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-84.444205807837690 > -242.913761138954726). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.064321595011961 > -146.100691775009864). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(


0.9963898916967509
{'balancer': ADASYN(), 'clf': RandomForestClassifier(n_estimators=35, random_state=1), 'feat_selec
tor': SelectKBest(k=5), 'feature_edit': FeatureAgglomeration(n_clusters=5), 'imputer': SimpleImput
er(), 'outlier_remover': FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x00000237F09
AE0D0>,
                 kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                       random_state=1,
                                                       support_fraction=0.75)}), 'scaler': Standard
Scaler(), 'transformer': StandardScaler()}
```

In [106]:

```
clf_cv5 = GridSearchCV(Pipeline(process), params, scoring = "balanced_accuracy")

clf_cv5.fit(X_train, y_train)
score5 = clf_cv5.score(X_train, y_train)
print(score5)

print(clf_cv25.best_params_)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.064321595011961 > -146.100691775009864). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3732/218121173.py in <module>
      5 print(score5)
      6
----> 7 print(clf_cv25best_params_)

NameError: name 'clf_cv25best_params_' is not defined
```

In [107]:

```
print(score5)
print(clf_cv5.best_params_)
```

```
0.9922544892275217
{'balancer': ADASYN(), 'clf': RandomForestClassifier(n_estimators=35, random_state=1), 'feat
_selector': SelectKBest(k=5), 'feature_edit': 'passthrough', 'imputer': SimpleImputer(), 'ou
tlier_remover': FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x00000237F09AE0
D0>,
                kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                      random_state=1,
                                                      support_fraction=0.75)}), 'scaler': St
andardScaler(), 'transformer': 'passthrough'}
```

In [27]:

```
from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline
from sklearn.metrics import recall_score


clf_cv6 = GridSearchCV(Pipeline(process), params, scoring = make_scorer(score_func=recall_score, pos_label=

clf_cv6.fit(X_train, y_train)
score6 = clf_cv6.score(X_train, y_train)
print(score6)

print(clf_cv6.best_params_)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-84.444205807837690 > -242.913761138954726). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.064321595011961 > -146.100691775009864). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: Ru
ntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_d
et) (-141.837903416998813 > -147.034527781739911). You may want to try with a higher value
of support_fraction (current value: 0.750).
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWar
ning: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You m
ay use labels=[pos_label] to specify a single positive class.
  warnings.warn(


  [0.98717949]
  {'balancer': ADASYN(), 'clf': SVC(C=0.01, kernel='linear', random_state=1), 'feat_selector': Selec
  tKBest(k=5), 'feature_edit': KBinsDiscretizer(n_bins=2, random_state=1), 'imputer': SimpleImputer
  (), 'outlier_remover': FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x0000011C2C8C7
  E50>,
                  kw_args={'strategy': EllipticEnvelope(contamination=0.01,
                                                        random_state=1,
                                                        support_fraction=0.75)}), 'scaler': Standard
  Scaler(), 'transformer': Normalizer()}
```

SelectKBest() then smote: best accuracy: 0.9972924187725631 clf': RandomForestClassifier(n_estimators=70, random_state=1) transformer': MaxAbsScaler() feature_edit': 'passthrough'

best balanced accuracy: 0.9547467758805128 clf': MLPClassifier(random_state=1) transformer': StandardScaler() feature_edit': 'passthrough'

best high risk patient recall clf': KNeighborsClassifier(n_neighbors=3) transformer': Normalizer() feature_edit': 'passthrough

SelectKBest(k=5) then ADASYN(): best accuracy: 0.9963898916967509 clf': RandomForestClassifier(n_estimators=35, random_state=1) feature_edit': FeatureAgglomeration(n_clusters=5) transformer': StandardScaler()

best balanced accuracy: 0.9922544892275217 clf': RandomForestClassifier(n_estimators=35, random_state=1) transformer': 'passthrough' feature_edit': 'passthrough'

best high risk patient recall : 0.98717949 clf': SVC(C=0.01, kernel='linear', random_state=1) transformer': Normalizer() feature_edit': KBinsDiscretizer(n_bins=2, random_state=1)

In [32]:

```
def pipelineTester(clf):

        cv =  StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state is none bec
        scoresAcc = cross_val_score(clf, X_train, y_train, cv=cv)
        meanAcc = scoresAcc.mean()

        scoresBallAcc = cross_val_score(clf, X_train, y_train, cv=cv, scoring = "balanced_accuracy")
        meanBallAcc = scoresBallAcc.mean()

        recall_low_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_func=r
        recall_medium_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_fun
        recall_high_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_func=

        recall = {
            "Low risk" : recall_low_risk,
            "Medium risk" : recall_medium_risk,
            "High risk" : recall_high_risk
        }

        print("Accuracy: ", meanAcc)
        print("Balanced accuracy: ", meanBallAcc)
        print("Recall: ", recall)
```

In [33]:

```
#had good accuracy
pipe1 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=
    SelectKBest(),
    SMOTE(),
    MaxAbsScaler(),
    RandomForestClassifier(n_estimators=70, random_state=1))

pipelineTester(pipe1)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(

Accuracy:  0.9259793730381964
Balanced accuracy:  0.8647280156558608
Recall:  {'Low risk': 0.949378778818683, 'Medium risk': 0.7573863636363637, 'High risk': 0.8
724999999999999}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

In [34]:

```
#had good balanced accuracy
pipe2 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=
    SelectKBest(),
    SMOTE(),
    StandardScaler(),
    MLPClassifier(random_state=1))

pipelineTester(pipe2)
```

```
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWar
ning: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You m
ay use labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWar
ning: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You m
ay use labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWar
```

```
Accuracy:  0.9051648934001875
Balanced accuracy:  0.8740471638852145
Recall:  {'Low risk': 0.9194405687329746, 'Medium risk': 0.8571969696969697, 'High risk': 0.872499
9999999999}
```

In [38]:

```
#had good high risk recall
pipe3 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(),
    SMOTE(),
    Normalizer(),
    KNeighborsClassifier(n_neighbors=3))

pipelineTester(pipe3)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(

Accuracy:  0.8862215156332803
Balanced accuracy:  0.8512352370626738
Recall:  {'Low risk': 0.8998538303102783, 'Medium risk': 0.8388257575757576, 'High risk': 0.
8308333333333333}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

In [35]:

```
#had good accuracy
#seems pretty good, better than pipe 3
pipe4 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(k=5),
    ADASYN(),
    StandardScaler(),
    FeatureAgglomeration(n_clusters=5),
    RandomForestClassifier(n_estimators=35, random_state=1))

pipelineTester(pipe4)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(

Accuracy:  0.8988830459418693
Balanced accuracy:  0.8582841168310571
Recall:  {'Low risk': 0.9171350740814563, 'Medium risk': 0.7825757575757576, 'High risk': 0.
8708333333333333}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

In [36]:

```
#had good balanced accuracy
pipe5 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(k=5),
    ADASYN(),
    RandomForestClassifier(n_estimators=35, random_state=1))

pipelineTester(pipe5)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(

Accuracy:  0.9006848477436712
Balanced accuracy:  0.8542108699109894
Recall:  {'Low risk': 0.9240449139592053, 'Medium risk': 0.7513257575757576, 'High risk': 0.
8708333333333333}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

In [37]:

```
#had good high risk recall
pipe6 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(k=5),
    ADASYN(),
    Normalizer(),
    KBinsDiscretizer(n_bins=2, random_state=1),
    SVC(C=0.01, kernel='linear', random_state=1))

pipelineTester(pipe6)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 2) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(

Accuracy:  0.6326240267416738
Balanced accuracy:  0.6753445507496734
Recall:  {'Low risk': 0.6340575377051358, 'Medium risk': 0.4166666666666667, 'High risk': 0.
9741666666666667}

C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarni
ng: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may u
se labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

```
RESULTS:

Pipe1:
Accuracy:  0.9259793730381964
Balanced accuracy:  0.8647280156558608
Recall:  {'Low risk': 0.949378778818683, 'Medium risk': 0.7573863636363637, 'High risk': 0.8724999
999999999}

Pipe2:
Accuracy:  0.9051648934001875
Balanced accuracy:  0.8740471638852145
Recall:  {'Low risk': 0.9194405687329746, 'Medium risk': 0.8571969696969697, 'High risk': 0.872499
9999999999}

Pipe3:
Accuracy:  0.8862215156332803
Balanced accuracy:  0.8512352370626738
Recall:  {'Low risk': 0.8998538303102783, 'Medium risk': 0.8388257575757576, 'High risk': 0.830833
3333333333}

Pipe4:
Accuracy:  0.8988830459418693
Balanced accuracy:  0.8582841168310571
Recall:  {'Low risk': 0.9171350740814563, 'Medium risk': 0.7825757575757576, 'High risk': 0.870833
3333333333}

Pipe5:
Accuracy:  0.9006848477436712
Balanced accuracy:  0.8542108699109894
Recall:  {'Low risk': 0.9240449139592053, 'Medium risk': 0.7513257575757576, 'High risk': 0.870833
3333333333}

Pipe6:
Accuracy:  0.6326240267416738
Balanced accuracy:  0.6753445507496734
Recall:  {'Low risk': 0.6340575377051358, 'Medium risk': 0.4166666666666667, 'High risk': 0.974166
6666666667}
```

The pipe picked out of these is pipe 2. With great accuracy, balanced accuracy, and recall.

This is the specifics of it:

```
pipe2 = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contam
ination=0.05, n_neighbors=40, novelty=True)}),
    SelectKBest(),
    SMOTE(),
    StandardScaler(),
    MLPClassifier(random_state=1))
```

Now lets do one more run, with the above pipeline, but lets tune the MLP Classifier.

In [24]:

```
from sklearn.model_selection import ParameterGrid

mlp_params = [
                (MLPClassifier,
                 {"solver" : ["adam"], #we are using the adam solver becuase it works well on training sets
                  "learning_rate" : ["constant", "invscaling", "adaptive"],
                  "max_iter" : [100,200,400],
                  "random_state" : [1]
                 }),
             ]



mlps= [ctor(**para) for ctor, paras in mlp_params for para in ParameterGrid(paras)]

mlps
```

Out[24]:

```
[MLPClassifier(max_iter=100, random_state=1),
 MLPClassifier(random_state=1),
 MLPClassifier(max_iter=400, random_state=1),
 MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1),
 MLPClassifier(learning_rate='invscaling', random_state=1),
 MLPClassifier(learning_rate='invscaling', max_iter=400, random_state=1),
 MLPClassifier(learning_rate='adaptive', max_iter=100, random_state=1),
 MLPClassifier(learning_rate='adaptive', random_state=1),
 MLPClassifier(learning_rate='adaptive', max_iter=400, random_state=1)]
```

In [31]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer, recall_score
def pipelineTester(clf):

        cv =  StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state is none bec
        scoresAcc = cross_val_score(clf, X_train, y_train, cv=cv)
        meanAcc = scoresAcc.mean()

        scoresBallAcc = cross_val_score(clf, X_train, y_train, cv=cv, scoring = "balanced_accuracy")
        meanBallAcc = scoresBallAcc.mean()

        recall_low_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_func=r
        recall_medium_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_fun
        recall_high_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scorer(score_func=

        recall = {
            "Low risk" : recall_low_risk,
            "Medium risk" : recall_medium_risk,
            "High risk" : recall_high_risk
        }

        print(type(meanAcc))
        print(type(meanBallAcc))
        print(type(recall))

        result = "Accuracy: " + str(meanAcc) + "\n Balanced accuracy: " + str(meanBallAcc) + "\nRecall: " +
        return result
```

In [32]:

```
results = []

for mlp in mlps:
    #print("\n", mlp)

    pipe = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=
    SelectKBest(),
    SMOTE(),
    StandardScaler(),
    mlp)

    result = pipelineTester(pipe)
    results.append(result)

for i in range(len(results)):
    print("\n", mlps[i])
    print(results[i])
```

```
<class 'numpy.float64'>
<class 'dict'>

 MLPClassifier(max_iter=100, random_state=1)
Accuracy: 0.8961436549671845
 Balanced accuracy: 0.8707665125442183
Recall:
         Low risk0.9044714636901203
         Medium risk0.8630681818181818
         High risk0.885

 MLPClassifier(random_state=1)
Accuracy: 0.9042599160246219
 Balanced accuracy: 0.8736640221227624
Recall:
         Low risk0.9194339246561691
         Medium risk0.8571969696969697
         High risk0.86
```

The results are: (original output from cell above deleted because there are too many warnings and it makes the file too big)

```
 MLPClassifier(max_iter=100, random_state=1)
Accuracy: 0.8961436549671845
 Balanced accuracy: 0.8707665125442183
Recall:
     Low risk0.9044714636901203
     Medium risk0.8630681818181818
     High risk0.885

 MLPClassifier(random_state=1)
Accuracy: 0.9042599160246219
 Balanced accuracy: 0.8736640221227624
Recall:
     Low risk0.9194339246561691
     Medium risk0.8571969696969697
     High risk0.86

 MLPClassifier(max_iter=400, random_state=1)
Accuracy: 0.9114956585544821
 Balanced accuracy: 0.8712542985163575
Recall:
     Low risk0.9332469603348613
     Medium risk0.8196969696969697
     High risk0.8466666666666667

 MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1)
Accuracy: 0.8934409522644817
 Balanced accuracy: 0.874372683381553
Recall:
     Low risk0.8998604743870839
     Medium risk0.8445075757575757
     High risk0.885

 MLPClassifier(learning_rate='invscaling', random_state=1)
Accuracy: 0.9069666952019894
 Balanced accuracy: 0.8703681321889414
Recall:
     Low risk0.9194405687329746
     Medium risk0.8571969696969697
     High risk0.8466666666666667

 MLPClassifier(learning_rate='invscaling', max_iter=400, random_state=1)
Accuracy: 0.9114997350291467
 Balanced accuracy: 0.8683980375813144
Recall:
     Low risk0.9332469603348613
     Medium risk0.8446969696969697
     High risk0.8341666666666667

 MLPClassifier(learning_rate='adaptive', max_iter=100, random_state=1)
Accuracy: 0.8898332721862134
 Balanced accuracy: 0.8665932018007462
Recall:
     Low risk0.896412198525014
```

The final estimator is:

```
    MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1)
    Accuracy: 0.8952427540662835
     Balanced accuracy: 0.8812302011544586
    Recall:
         Low risk0.9021659690386021
```

This means that the final pipeline is:

```
    SimpleImputer(),
        StandardScaler(),
        FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contam
    ination=0.05, n_neighbors=40, novelty=True)}),
        SelectKBest(),
        SMOTE(),
        StandardScaler(),
        MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1)
```

Now we've decided our final pipeline, lets get some statistics to evaluate it using our validation set.

In [67]:

```
#!pip install scikit-plot
```

```
Collecting scikit-plot
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: matplotlib>=1.4.0 in c:\users\will\anaconda3\lib\site-package
s (from scikit-plot) (3.4.3)
Requirement already satisfied: joblib>=0.10 in c:\users\will\anaconda3\lib\site-packages (fr
om scikit-plot) (1.2.0)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\will\anaconda3\lib\site-packag
es (from scikit-plot) (1.2.2)
Requirement already satisfied: scipy>=0.9 in c:\users\will\anaconda3\lib\site-packages (from
scikit-plot) (1.7.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\will\anaconda3\lib\site-packages
(from matplotlib>=1.4.0->scikit-plot) (3.0.4)
Requirement already satisfied: pillow>=6.2.0 in c:\users\will\anaconda3\lib\site-packages (f
rom matplotlib>=1.4.0->scikit-plot) (8.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\will\anaconda3\lib\site-package
s (from matplotlib>=1.4.0->scikit-plot) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\will\anaconda3\lib\site-packages (fr
om matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: numpy>=1.16 in c:\users\will\anaconda3\lib\site-packages (fro
m matplotlib>=1.4.0->scikit-plot) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\will\anaconda3\lib\site-pack
ages (from matplotlib>=1.4.0->scikit-plot) (2.8.2)
Requirement already satisfied: six in c:\users\will\anaconda3\lib\site-packages (from cycler
>=0.10->matplotlib>=1.4.0->scikit-plot) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\will\anaconda3\lib\site-pack
ages (from scikit-learn>=0.18->scikit-plot) (2.2.0)
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
```

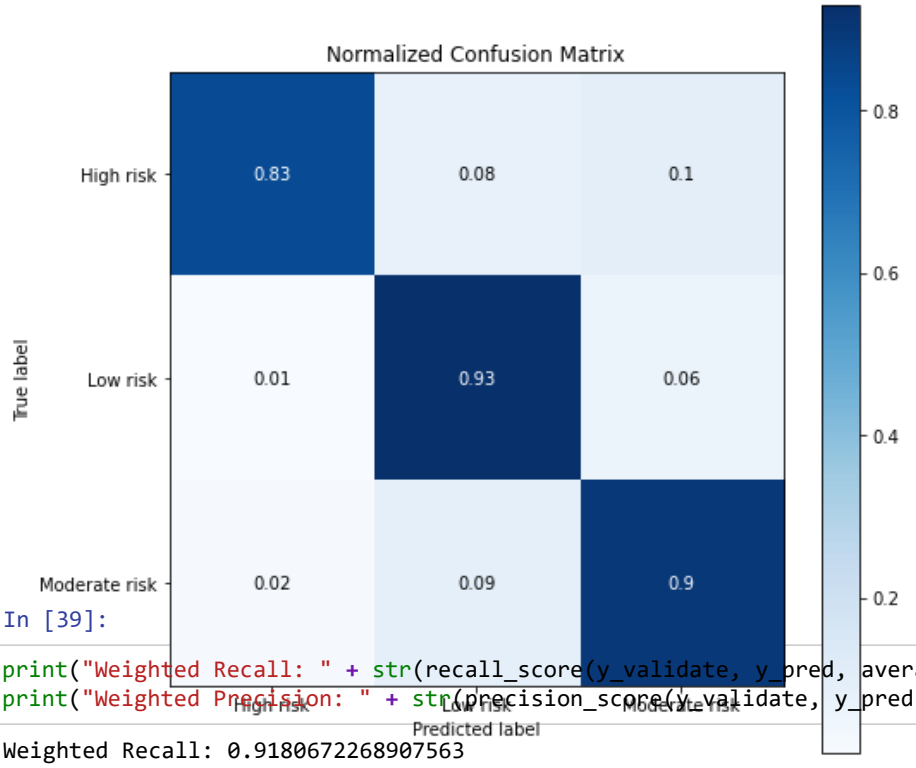
In [38]:

```
from imblearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from imblearn import FunctionSampler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.feature_selection import SelectKBest
from imblearn.over_sampling import SMOTE
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import f1_score

pipe = make_pipeline(SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(),
    SMOTE(),
    StandardScaler(),
    MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1))

pipe.fit(X_train, y_train)
y_hat = pipe.predict(X_validate)

from sklearn.metrics import RocCurveDisplay
import matplotlib.pyplot as plt
import scikitplot as skplt

y_pred = pipe.predict(X_validate)
y_probas = pipe.predict_proba(X_validate)

from sklearn.metrics import mean_squared_error, accuracy_score, balanced_accuracy_score, precision_score, r
print("Accuracy: " + str(accuracy_score(y_validate, y_pred)))
print("Balanced Accuracy: " + str(balanced_accuracy_score(y_validate, y_pred)))
print("Macro Precision: " + str(precision_score(y_validate, y_pred, average = "macro")))
print("Micro Precision: " + str(precision_score(y_validate, y_pred, average = "micro")))
print("Macro Recall: " + str(recall_score(y_validate, y_pred, average = "macro")))
print("Micro Recall: " + str(recall_score(y_validate, y_pred, average = "micro")))
print("Mean squared error: " + str(mean_squared_error(y_validate, y_pred)))
print("Micro F1 score:" + str(f1_score(y_validate, y_pred, average="micro")))
print("Macro F1 score:" + str(f1_score(y_validate, y_pred, average="macro")))


skplt.metrics.plot_confusion_matrix(label_enoder.inverse_transform(y_validate), label_enoder.inverse_trans
#skplt.metrics.plot_confusion_matrix(y_validate, y_pred, normalize=True, figsize=(8,8))
```

```
Accuracy: 0.9180672268907563
Balanced Accuracy: 0.8859670101897349
Macro Precision: 0.8519730860156391
Micro Precision: 0.9180672268907563
Macro Recall: 0.8859670101897349
Micro Recall: 0.9180672268907563
Mean squared error: 0.11974789915966387
Micro F1 score:0.9180672268907561
Macro F1 score:0.8624558821457664

C:\Users\Will\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and the optim
ization hasn't converged yet.
  warnings.warn(
```

Out[38]:

```
<AxesSubplot:title={'center':'Normalized Confusion Matrix'}, xlabel='Predicted label', ylabe
l='True label'>
```



In [39]:

```
print("Weighted Recall: " + str(recall_score(y_validate, y_pred, average = "weighted")))
print("Weighted Precision: " + str(precision_score(y_validate, y_pred, average = "weighted")))
```

```
Weighted Recall: 0.9180672268907563
Weighted Precision: 0.9303708390066338
```

In [41]:

```
#get distribution of target class labels
target_field_value_counts = pd.DataFrame(label_enoder.inverse_transform(y_validate), columns = ['target'])
target_field_value_percentages = target_field_value_counts / y_validate.shape[0] * 100
print("Target class distribution percentages: \n")
#todo check id distribution should be percentage or fraction
print(target_field_value_percentages)
```

```
Target class distribution percentages:

target
Low risk         76.890756
Moderate risk    12.184874
High risk        10.924370
dtype: float64
```

These are pretty pleasing results.

Good accuracy, with slightly worse balanced accuracy. Would have been nice to have the balanced accuracy slightly higher.

It's a shame my efforts to get better recall for high risk patients weren't as successful as I hoped - ~16% of high risk patients being predicted as low or medium risk isn't ideal (recall of 0.84158415841) - we actually ended up getting better precision for the high risk patients instead (0.95505617977). This model should not be used for the complete analysis of a patients health, but it's interesting to note that if a patient were to be predicted to be high risk, it is very liekly that they are truely high risk - only 4% of people predicted as being high risk were incorrectly labelled.

The micro precision score makes sense - the overall precision is pretty good, but it is not balanced well for classes with less labels. The macro precision's score being higher also reflects this, as macro precision does not take label imbalance into account. (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html)). The same can be said for macro and micro recall - overall pretty good, but not as good for the smaller classes in the uneven distribution of class labels.

Mean squared error is promising, with a nice low 0.1.

In [16]:

```
print("\n Label mapping dictionary:")
print(target_name_mapping)
```

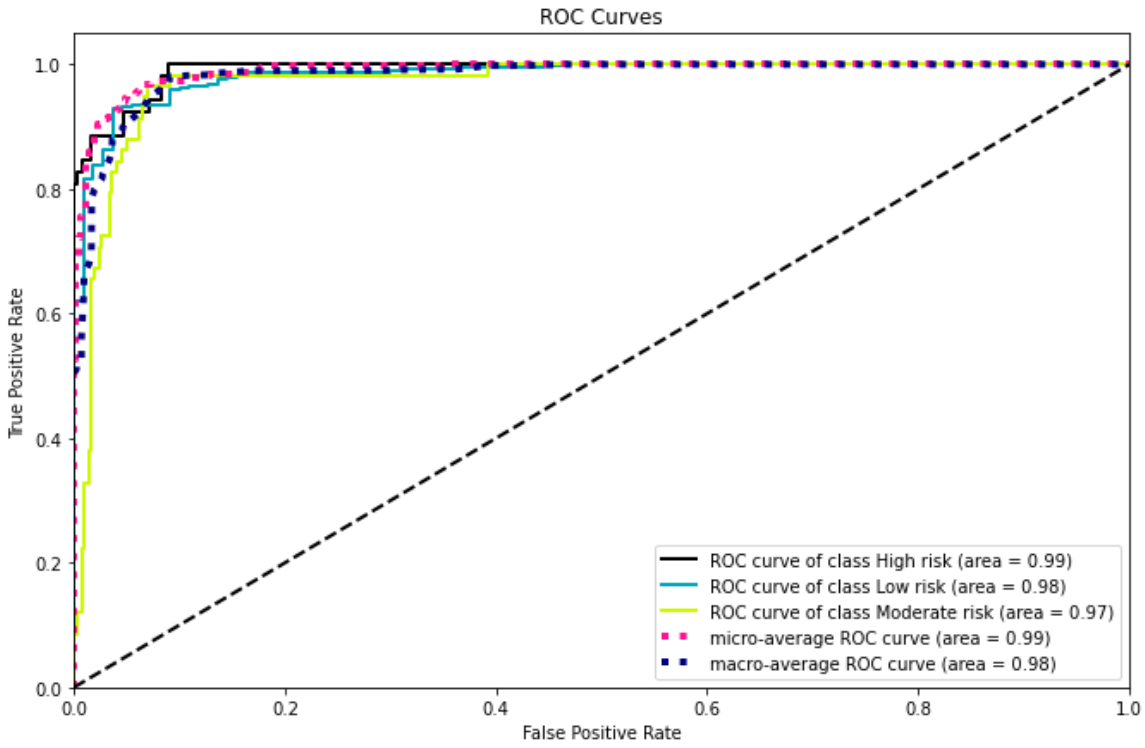
```
 Label mapping dictionary:
{'High risk': 0, 'Low risk': 1, 'Moderate risk': 2}
```

In [42]:

```
skplt.metrics.plot_roc(label_enoder.inverse_transform(y_validate), y_probas, figsize = (11,7))
```

Out[42]:

```
<AxesSubplot:title={'center':'ROC Curves'}, xlabel='False Positive Rate', ylabel='True Posit
ive Rate'>
```



(Reminder for myself - point on the dotted line means that correctly classified is the same as incorrectly classified)

Good balance between true positive rate and false positive rates for all classes. The number of correctly classified samples is a lot higher than incorrectly classified samples. However, AUC isn't the most useful here, as it would be preferred to have more false positive (for high risk especically) than false negatives. (https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=AUC%20stands%20for%20%22Area%20under,across%20all%20possible%20classification%20thresholds (https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=AUC%20stands%20for%20%22Area%20under,across%20all%20possible%20classification%20thresholds).)

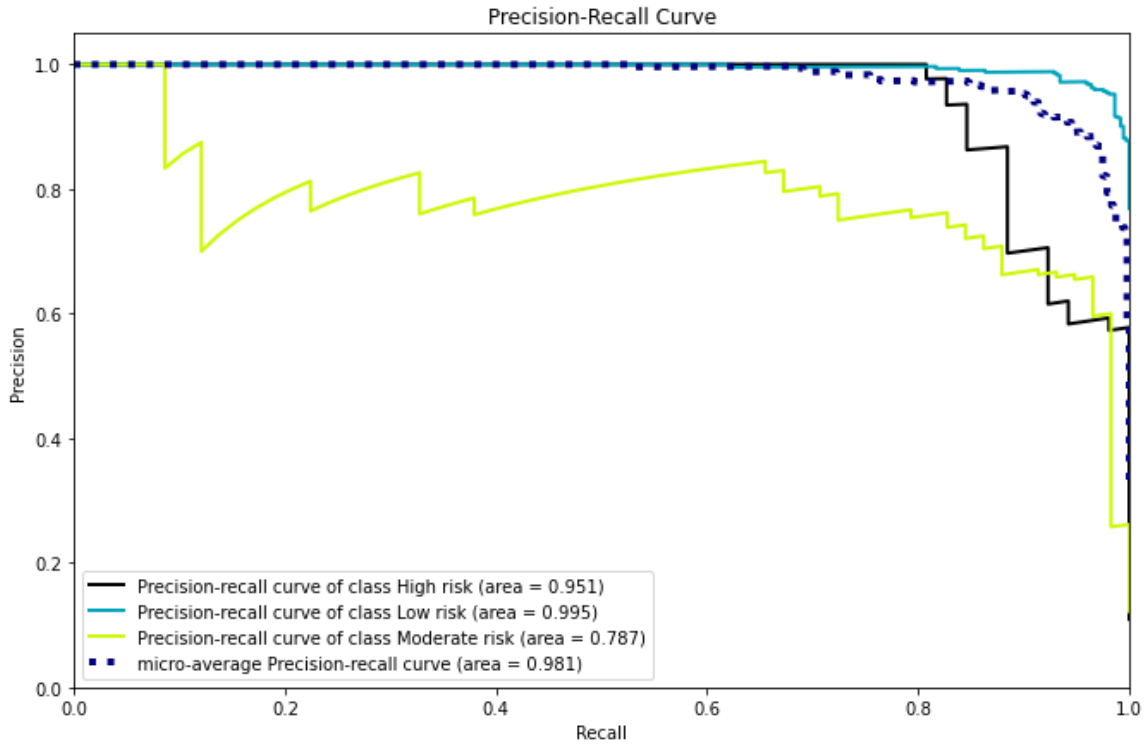
Performance is best for high risk, and worst for moderate risk - though all three curves are very similar.

In [43]:

```
skplt.metrics.plot_precision_recall(label_enoder.inverse_transform(y_validate), y_probas, figsize = (11,7))
```

Out[43]:

```
<AxesSubplot:title={'center':'Precision-Recall Curve'}, xlabel='Recall', ylabel='Precision'>
```



As can be seen from this curve, precision for high risk labels is better than recall. Our precision of 0.95 meets the recall of 0.84, but if we were to achieve high recall scores then the prcision would be very low - any recalls above 0.95 achieved would result in ~0.6 recall.

Overall these results pretty pleasing. The classifier, overall, is pretty good at correctly labelling records. However, in the context of health data, my main issue is that I am dissapointed about the potential missclassifcation of truely high risk patients.

Predicting the test set

In [50]:

```
#designate the path of the health test data
health_test_data_path = "health_test.csv"
health_data_path = "health_train.csv"
#Load the data using pandas read_csv function.

health_test_data = pd.read_csv(health_test_data_path)
health_data = pd.read_csv(health_data_path)

health_test_data.head()
```

Out[50]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
0	PA3001	767	135	F	0.000	0.0	0.000	0.0	0.0	0.0	...	76	67	143	2	0	137	136	138	0	1
1	PA3002	1592	134	F	0.000	0.0	0.010	0.0	0.0	0.0	...	74	89	163	7	1	138	134	138	13	0
2	PA3003	1115	122	M	0.000	0.0	0.000	0.0	0.0	0.0	...	39	103	142	1	0	120	120	122	3	0
3	PA3004	299	148	F	0.000	NaN	0.000	0.0	0.0	0.0	...	14	139	153	1	0	150	148	150	0	1
4	PA3005	1194	133	M	0.003	0.0	0.005	0.0	0.0	0.0	...	58	113	171	5	1	150	147	149	5	0

5 rows × 25 columns

In [57]:

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder

#find which columns are categorical
categorical_labels = ['target', 'x14', 'id', 'x3']

#encode thjese using the ordinal encoder

encoder = OrdinalEncoder()
encoder.fit(health_data[["x3"]])
x3_train_encoded = encoder.transform(health_data[["x3"]])
x3_test_encoded = encoder.transform(health_test_data[["x3"]])
print("x3 mapping: " , encoder.categories_)

encoder.fit(health_data[["x14"]])
x14_train_encoded = encoder.transform(health_data[["x14"]])
x14_test_encoded = encoder.transform(health_test_data[["x14"]])
print("x14 mapping: " , encoder.categories_)

#target_encoded = encoder.fit_transform(health_data[["target"]])
#print("target mapping: " , encoder.categories_)
#print("target feature names: ", encoder.feature_names_in_)


label_enoder = LabelEncoder()
target_train_encoded = label_enoder.fit_transform(health_data["target"])

target_name_mapping = dict(zip(label_enoder.classes_, label_enoder.transform(label_enoder.classes_)))
print("\n Label mapping dictionary:")
print(target_name_mapping)
#target_encoded = label_enoder.transform(health_data[["target"]])
#print(target_name_mapping)
```

```
x3 mapping:  [array(['F', 'M'], dtype=object)]
x14 mapping:  [array(['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-'], dtype=object)]

 Label mapping dictionary:
{'High risk': 0, 'Low risk': 1, 'Moderate risk': 2}
```

In [62]:

```
#apply the encoded values to the DFs
health_data["x3"] = x3_train_encoded
health_test_data["x3"] = x3_test_encoded

health_data["x14"] = x14_train_encoded
health_test_data["x14"] = x14_test_encoded

health_data["target"] = target_train_encoded

health_test_data.head()
```

Out[62]:

	id	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
0	PA3001	767	135	0.0	0.000	0.0	0.000	0.0	0.0	0.0	...	76	67	143	2	0	137	136	138	0	1
1	PA3002	1592	134	0.0	0.000	0.0	0.010	0.0	0.0	0.0	...	74	89	163	7	1	138	134	138	13	0
2	PA3003	1115	122	1.0	0.000	0.0	0.000	0.0	0.0	0.0	...	39	103	142	1	0	120	120	122	3	0
3	PA3004	299	148	0.0	0.000	NaN	0.000	0.0	0.0	0.0	...	14	139	153	1	0	150	148	150	0	1
4	PA3005	1194	133	1.0	0.003	0.0	0.005	0.0	0.0	0.0	...	58	113	171	5	1	150	147	149	5	0

5 rows × 25 columns

In [68]:

```
#get the data out, leaving behind the target column (last feature).
X_train = health_data.iloc[:, 1:-1]
#extract the target column.
y_train = health_data["target"]

#remove the id column from the train data
X_test = health_test_data.iloc[:, 1:]

X_test
```

Out[68]:

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	x15	x16	x17	x18	x19	x20	x21	x22	x23	x
0	767	135	0.0	0.000	0.000	0.000	0.000	0.0	0.0	67.0	...	76	67	143	2	0	137	136	138	0	
1	1592	134	0.0	0.000	0.000	0.010	0.000	0.0	0.0	27.0	...	74	89	163	7	1	138	134	138	13	
2	1115	122	1.0	0.000	0.000	0.000	0.000	0.0	0.0	19.0	...	39	103	142	1	0	120	120	122	3	
3	299	148	0.0	0.000	NaN	0.000	0.000	0.0	0.0	72.0	...	14	139	153	1	0	150	148	150	0	
4	1194	133	1.0	0.003	0.000	0.005	0.000	0.0	0.0	38.0	...	58	113	171	5	1	150	147	149	5	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
523	1071	133	1.0	0.001	0.000	0.007	0.004	0.0	0.0	27.0	...	95	82	177	4	0	147	133	138	43	
524	364	135	0.0	0.000	0.016	0.002	0.000	0.0	0.0	70.0	...	9	132	141	1	0	136	136	137	0	
525	531	142	0.0	0.016	0.060	0.004	0.000	0.0	0.0	38.0	...	130	68	198	5	0	180	173	177	14	
526	878	136	0.0	0.002	0.000	0.006	0.000	0.0	0.0	39.0	...	47	107	154	1	0	138	139	139	2	
527	1733	134	0.0	0.008	NaN	0.009	0.004	0.0	0.0	60.0	...	109	80	189	4	1	156	147	151	40	

528 rows × 24 columns

In [69]:

```
#make the predictions
from imblearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from imblearn import FunctionSampler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.feature_selection import SelectKBest
from imblearn.over_sampling import SMOTE
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import f1_score

final_pipe = make_pipeline(SimpleImputer(),
    StandardScaler(),
    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : LocalOutlierFactor(contamination=(
    SelectKBest(),
    SMOTE(),
    StandardScaler(),
    MLPClassifier(learning_rate='invscaling', max_iter=100, random_state=1))

pipe.fit(X_train, y_train)
y_test_pred = pipe.predict(X_test)

y_test_pred
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:6
86: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and the optim
ization hasn't converged yet.
  warnings.warn(
```

Out[69]:

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

In [70]:

```
y_test_pred_unencoded = label_enoder.inverse_transform(y_test_pred)
y_test_pred_unencoded
```

```
       'Moderate risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'High risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'High risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'High risk',
       'Low risk', 'Low risk', 'High risk', 'Moderate risk', 'Low risk',
       'High risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'Moderate risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Moderate risk', 'Low risk', 'Low risk', 'Low risk', 'High risk',
       'Moderate risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'Low risk', 'High risk', 'Moderate risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'Low risk',
       'Low risk', 'Low risk', 'Low risk', 'Low risk', 'High risk',
       'Low risk', 'Moderate risk', 'Low risk', 'High risk', 'Low risk',
       'Low risk', 'Moderate risk', 'Low risk', 'Low risk',
       'Moderate risk', 'Moderate risk', 'High risk', 'Low risk',
       'Moderate risk', 'Low risk', 'Low risk', 'High risk', 'Low risk',
       'Low risk', 'Low risk', 'Moderate risk', 'Low risk', 'Low risk',
```

In [71]:

```
#checking to see if the distribution of the test labels is roughly correct
#get distribution of target class labels
target_field_value_counts = pd.DataFrame(y_test_pred_unencoded, columns = ['target']).value_counts()
target_field_value_percentages = target_field_value_counts / y_validate.shape[0] * 100
print("Target class distribution percentages: \n")
#todo check id distribution should be percentage or fraction
print(target_field_value_percentages)
```

```
Target class distribution percentages:

target
Low risk         80.462185
Moderate risk    18.907563
High risk        11.554622
dtype: float64
```

looks good

In [72]:

```
y_test_pred_unencoded.shape
```

Out[72]:

```
(528,)
```

In [79]:

```
#designate the path of the predicted target
predictedTarget_path = "predictedTarget_empty.csv"

#Load the data using pandas read_csv function.
predictedTarget = pd.read_csv(predictedTarget_path)


predictedTarget.head()
```

Out[79]:

	id	predicted_target
0	PA3001	NaN
1	PA3002	NaN
2	PA3003	NaN
3	PA3004	NaN
4	PA3005	NaN

In [80]:

```
predictedTarget["predicted_target"] = y_test_pred_unencoded
```

In [81]:

```
predictedTarget
```

Out[81]:

	id	predicted_target
0	PA3001	High risk
1	PA3002	Low risk
2	PA3003	Low risk
3	PA3004	Moderate risk
4	PA3005	Low risk
...	...	...
523	PA3524	Low risk
524	PA3525	High risk
525	PA3526	Low risk
526	PA3527	Low risk
527	PA3528	Low risk

528 rows × 2 columns

In [82]:

```
predictedTarget.to_csv('predictedTarget.csv', index = False)
```

In [83]:

```
#designate the path of the predicted target - this is for checking the csv got filled correctly
predictedTarget_filled_path = "predictedTarget.csv"

#Load the data using pandas read_csv function.
predictedTarget_filled = pd.read_csv(predictedTarget_filled_path)


predictedTarget_filled.head()
```

Out[83]:

	id	predicted_target
0	PA3001	High risk
1	PA3002	Low risk
2	PA3003	Low risk
3	PA3004	Moderate risk
4	PA3005	Low risk

All done :)

In [84]:

```
!pip install nbconvert
!pip install pyppeteer
```

```
Requirement already satisfied: nbconvert in c:\users\will\anaconda3\lib\site-packages (6.1.
0)
Requirement already satisfied: jinja2>=2.4 in c:\users\will\anaconda3\lib\site-packages (fro
m nbconvert) (2.11.3)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\will\anaconda3\lib\site-packag
es (from nbconvert) (0.3)
Requirement already satisfied: bleach in c:\users\will\anaconda3\lib\site-packages (from nbc
onvert) (4.0.0)
Requirement already satisfied: defusedxml in c:\users\will\anaconda3\lib\site-packages (from
nbconvert) (0.7.1)
Requirement already satisfied: jupyter-core in c:\users\will\anaconda3\lib\site-packages (fr
om nbconvert) (4.8.1)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\will\anaconda3\lib\site-package
s (from nbconvert) (0.8.4)
Requirement already satisfied: jupyterlab-pygments in c:\users\will\anaconda3\lib\site-packa
ges (from nbconvert) (0.1.2)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\will\anaconda3\lib\site-pack
ages (from nbconvert) (1.4.3)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in c:\users\will\anaconda3\lib\site-pa
ckages (from nbconvert) (0.5.3)
Requirement already satisfied: testpath in c:\users\will\anaconda3\lib\site-packages (from n
bconvert) (0.5.0)
Requirement already satisfied: traitlets>=5.0 in c:\users\will\anaconda3\lib\site-packages
(from nbconvert) (5.1.0)
Requirement already satisfied: nbformat>=4.4 in c:\users\will\anaconda3\lib\site-packages (f
rom nbconvert) (5.1.3)
Requirement already satisfied: pygments>=2.4.1 in c:\users\will\anaconda3\lib\site-packages
(from nbconvert) (2.10.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\will\anaconda3\lib\site-packages
(from jinja2>=2.4->nbconvert) (1.1.1)
Requirement already satisfied: nest-asyncio in c:\users\will\anaconda3\lib\site-packages (fr
om nbclient<0.6.0,>=0.5.0->nbconvert) (1.5.1)
Requirement already satisfied: jupyter-client>=6.1.5 in c:\users\will\anaconda3\lib\site-pac
kages (from nbclient<0.6.0,>=0.5.0->nbconvert) (6.1.12)
Requirement already satisfied: async-generator in c:\users\will\anaconda3\lib\site-packages
(from nbclient<0.6.0,>=0.5.0->nbconvert) (1.10)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\will\anaconda3\lib\site-pack
ages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: pyzmq>=13 in c:\users\will\anaconda3\lib\site-packages (from
jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert) (22.2.1)
Requirement already satisfied: tornado>=4.1 in c:\users\will\anaconda3\lib\site-packages (fr
om jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert) (6.1)
Requirement already satisfied: pywin32>=1.0 in c:\users\will\anaconda3\lib\site-packages (fr
om jupyter-core->nbconvert) (228)
Requirement already satisfied: ipython-genutils in c:\users\will\anaconda3\lib\site-packages
(from nbformat>=4.4->nbconvert) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in c:\users\will\anaconda3\lib\site-p
ackages (from nbformat>=4.4->nbconvert) (3.2.0)
Requirement already satisfied: setuptools in c:\users\will\anaconda3\lib\site-packages (from
jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (58.0.4)
Requirement already satisfied: six>=1.11.0 in c:\users\will\anaconda3\lib\site-packages (fro
m jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (1.16.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\will\anaconda3\lib\site-packages (f
rom jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (21.2.0)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\will\anaconda3\lib\site-packag
es (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert) (0.18.0)
Requirement already satisfied: webencodings in c:\users\will\anaconda3\lib\site-packages (fr
om bleach->nbconvert) (0.5.1)
Requirement already satisfied: packaging in c:\users\will\anaconda3\lib\site-packages (from
bleach->nbconvert) (21.0)
Requirement already satisfied: pyparsing>=2.0.2 in c:\users\will\anaconda3\lib\site-packages
(from packaging->bleach->nbconvert) (3.0.4)
Collecting pyppeteer
  Downloading pyppeteer-1.0.2-py3-none-any.whl (83 kB)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in c:\users\will\anaconda3\lib\site-pa
ckages (from pyppeteer) (1.26.7)
Requirement already satisfied: certifi>=2021 in c:\users\will\anaconda3\lib\site-packages (f
rom pyppeteer) (2021.10.8)
Collecting websockets<11.0,>=10.0
  Downloading websockets-10.4-cp39-cp39-win_amd64.whl (101 kB)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in c:\users\will\anaconda3\lib\site-pac
```

```
kages (from pyppeteer) (1.4.4)
Requirement already satisfied: importlib-metadata>=1.4 in c:\users\will\anaconda3\lib\site-p
ackages (from pyppeteer) (4.8.1)
Collecting pyee<9.0.0,>=8.1.0
  Downloading pyee-8.2.2-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in c:\users\will\anaconda3\lib\site-packa
ges (from pyppeteer) (4.62.3)
Requirement already satisfied: zipp>=0.5 in c:\users\will\anaconda3\lib\site-packages (from
importlib-metadata>=1.4->pyppeteer) (3.6.0)
Requirement already satisfied: colorama in c:\users\will\anaconda3\lib\site-packages (from t
qdm<5.0.0,>=4.42.1->pyppeteer) (0.4.4)
Installing collected packages: websockets, pyee, pyppeteer
Successfully installed pyee-8.2.2 pyppeteer-1.0.2 websockets-10.4
```

In [ ]: