

Preprocessing

In [1]:

```
#!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 |
|---|--------|------|-------|----|-------|-------|-------|-------|-----|-----|-----|-------|-------|------|-----|-------|
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |

5 rows × 26 columns

Because of the large difference in scales of the data, it may be necessary to standardize it all. This is so that the different scales can be compared.

In [4]:

```
from sklearn.preprocessing import Normalizer

#THIS CODE HAS BEEN COMMENTED OUT BECAUSE IT PRODUCES ERRORS

#normalizer = Normalizer(norm='l2')

#because the normalizer cannot adjust categorical columns, we only adjust numeric columns

#health_data_normalised = normalizer.fit_transform(health_data[numerical_col_labels])

#we cannot normalise because of the missing data. we should do a little bit of preprocessing b
```

As you can see above, an error has occurred - there are still missing values in our dataset.

(From slideshow): Outliers are either

1. Data object that, in some sense, have characteristics different from most of the other objects in the data set (multi-dimensional outliers), or
2. Values of an attribute that are unusual with respect to the typical value for that attribute.

The danger with dealing with missing values is making our dataset biased if the missing values are not missing at random. Because of the lack of feature names, it is difficult to infer whether an attribute is MCAR (missing completely at random), MAR (missing at random), or MNAR (missing not at random). We should be careful when imputing the missing values in case they are not missing at random, as this may bias any models we make.

To become familiar with how the missing values are spread, we will look at how they are distributed to the different target classes.

In [36]:

```
#finding target class distributions of records with missing attribute, by attribute.

#attributes with missing values: x5, x8 and x10

#find the indexes of records with missing values by attribute
health_data.isnull()
x5_missing_indexes = health_data.index[health_data.isnull()['x5'] == True].tolist()
x8_missing_indexes = health_data.index[health_data.isnull()['x8'] == True].tolist()
x10_missing_indexes = health_data.index[health_data.isnull()['x10'] == True].tolist()

#Code I used for checking that the fields did in fact have missin values (I replaced x10 with x10)
#for index, row in health_data.iloc[x10_missing_indexes].iterrows():
    #print(row['x10'])

hd_just_missing_x5 = health_data.iloc[x5_missing_indexes]
hd_just_missing_x8 = health_data.iloc[x8_missing_indexes]
hd_just_missing_x10 = health_data.iloc[x10_missing_indexes]

hd_missing_column_variations = [hd_just_missing_x5, hd_just_missing_x8, hd_just_missing_x10]

#find the target class distribtuions of the dataset, reducing the data set to only rows with missing values

missing_val_col_names = ['x5', 'x8', 'x10']
count = 0;
for dataset in hd_missing_column_variations:

    ds_name = missing_val_col_names[count]
    count = count + 1

    #get distribution of target class labels
    target_field_value_counts_missing = dataset["target"].value_counts()
    target_field_value_percentages_missing = target_field_value_counts_missing / dataset.shape[0]

    print("\n")
    print(ds_name + ":")
    print("Number of records missing " + ds_name + " values: ", dataset.shape[0])
    print("Target class distribution percentages:")
    #todo check id distribution should be percentage or fraction
    print(target_field_value_percentages_missing)

print("\n")
print("Number of records in entire dataset: ", health_data.shape[0])
print("Target class distribution percentages, for the entire dataset:")
print(target_field_value_percentages)
```

```
x5:  
Number of records missing x5 values: 44  
Target class distribution percentages:  
Low risk      65.909091  
High risk     18.181818  
Moderate risk 15.909091  
Name: target, dtype: float64
```

```
x8:  
Number of records missing x8 values: 17  
Target class distribution percentages:  
Low risk      64.705882  
Moderate risk 23.529412  
High risk     11.764706  
Name: target, dtype: float64
```

```
x10:  
Number of records missing x10 values: 27  
Target class distribution percentages:  
Low risk      92.592593  
High risk     3.703704  
Moderate risk 3.703704  
Name: target, dtype: float64
```

```
Number of records in entire dataset: 1584  
Target class distribution percentages, for the entire dataset:  
Low risk      77.967172  
Moderate risk 13.825758  
High risk     8.207071  
Name: target, dtype: float64
```

The distributions seem roughly similar - only slightly off for all, and the number of records we are using is quite small so doesn't matter if it doesn't exactly reflect the distribution of the whole, given the fact that it becomes less likely that a distribution will be exactly the same as a larger one the smaller the sample size.

Because the missing records seem to be roughly distributed across the classes, we will assume that they are roughly distributed at random.

getting the dictionary of missing data stuff parameters - needs to be in format of 'imputer':[Array of imputers set up in different ways]

In [7]:

```

from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import BayesianRidge, Ridge
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import ParameterGrid
from sklearn.svm import SVC

imputer_params = [
    (SimpleImputer, {"strategy" : ["mean", "median", "most_frequent",]}),
    (IterativeImputer, {"estimator" : [BayesianRidge(), KNeighborsRegressor(n_neighbors = 15),
                                       "max_iter" : [5, 10, 20],
                                       "random_state" : [1234]}),
    (KNNImputer, {"n_neighbors" : [5,10,20,30]})}
]

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

imputers = [ctor(**para) for ctor, paras in imputer_params for para in ParameterGrid(paras)]
print(imputers)

params = dict(encoder = [OrdinalEncoder()],imputer = imputers, clf = [SVC()])
params

```

[SimpleImputer(), SimpleImputer(strategy='median'), SimpleImputer(strategy='most_frequent'), IterativeImputer(estimator=BayesianRidge(), max_iter=5, random_state=1234), IterativeImputer(estimator=BayesianRidge(), random_state=1234), IterativeImputer(estimator=BayesianRidge(), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=7, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=25, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=20, random_state=1234), KNNImputer(), KNNImputer(n_neighbors=10), KNNImputer(n_neighbors=20), KNNImputer(n_neighbors=30)]

Out[7]:

```
{'encoder': [OrdinalEncoder()],
'imputer': [SimpleImputer(),
SimpleImputer(strategy='median'),
SimpleImputer(strategy='most_frequent'),
IterativeImputer(estimator=BayesianRidge(), max_iter=5, random_state=1234),
IterativeImputer(estimator=BayesianRidge(), random_state=1234),
IterativeImputer(estimator=BayesianRidge(), max_iter=20, random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=5,
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15),
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=20,
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=5,
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7),
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=20,
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=5,
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25),
random_state=1234),
IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=20,
random_state=1234),
KNNImputer(),
KNNImputer(n_neighbors=10),
KNNImputer(n_neighbors=20),
KNNImputer(n_neighbors=30)],
'clf': [SVC()]}]
```

In [9]:

```
#get the data out, Leaving behind the target column (last feature).
X = health_data.iloc[:, :-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 | F | 0.005 | 0.000 | 0.002 | 0.000 | 0.0 | 0.000 | 46.0 | ... | |
| 1 | 258 | 127.0 | M | 0.012 | 0.000 | 0.008 | 0.004 | 0.0 | 0.000 | 13.0 | ... | |
| 2 | 479 | 145.0 | F | 0.000 | 0.000 | 0.000 | 0.002 | 0.0 | 0.000 | 57.0 | ... | |
| 3 | 906 | 146.0 | F | 0.004 | 0.000 | 0.005 | 0.003 | 0.0 | 0.000 | 29.0 | ... | |
| 4 | 1921 | 140.0 | F | 0.002 | 0.003 | 0.006 | 0.006 | 0.0 | 0.000 | 62.0 | ... | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1579 | 2077 | 130.0 | M | 0.005 | 0.001 | 0.001 | 0.000 | 0.0 | 0.000 | 72.0 | ... | |
| 1580 | 664 | 138.0 | F | 0.000 | 0.003 | 0.003 | 0.000 | 0.0 | 0.002 | 60.0 | ... | |
| 1581 | 1431 | 144.0 | F | 0.000 | 0.000 | 0.006 | 0.000 | 0.0 | 0.000 | 45.0 | ... | |
| 1582 | 630 | 134.0 | F | 0.017 | 0.002 | 0.004 | 0.000 | 0.0 | 0.000 | 48.0 | ... | |
| 1583 | 436 | 151.0 | F | 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 | Low risk |
| 1 | Low risk |
| 2 | Low risk |
| 3 | Low risk |
| 4 | Low risk |
| ... | |
| 1579 | Low risk |
| 1580 | Moderate risk |
| 1581 | Moderate risk |
| 1582 | Low risk |
| 1583 | Moderate risk |

Name: target, Length: 1584, dtype: object

In [10]:

```
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=
```

I need to impute missing values, but i've left the categorical data in there. luckily, none of the categorical data has missing values. I should probably encode the categorical data or omit it completely. probably best to convert it to numbers so that the multivariate feature imputation can use it in its maths.

In [11]:

```
##Commented out because this code produces errors

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

process = [("encoder", OrdinalEncoder()), ("imputer", SimpleImputer()), ("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_)"""


```

Out[11]:

```
'from sklearn.model_selection import GridSearchCV\nfrom sklearn.pipeline import\nPipeline\n\nprocess = [("encoder", OrdinalEncoder()), ("imputer", SimpleImputer()), ("clf", SVC())]\n\nclf_cv = GridSearchCV(Pipeline(process), params)\n\nclf_cv.fit(X_train, y_train)\nscore = clf_cv.score(X_train, y_train)\nprint(score)\n\nprint(clf_cv.best_params_)'
```

I am using ordinalencoder, but i dont know if this will mess up the maths for non ordinal stuff like blood type. - it doesnt seem that there are any other options though.

- seems like the ordinal encoder is trying to code numeric values - including NaN. I need to encode JUST the categorical columns where there is no missing data prior to doing all this.

In [12]:

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

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

#encode these 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_)

#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_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[12]:

```

In [13]:  id x1 x2 x3 x4 x5 x6 x7 x8 x9 ... x16 x17 x18 x19
          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 1
          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 1
          health_data["x3"] = x3_encoded
          health_data["x14"] = x14_encoded
          health_data["target"] = target_encoded
          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 1
          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 1
          health_data.head()
          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 1
Out[13]: ...
          ...
          1579 PA2580 x1 2077 x2 130.0 x3 M x4 0.005 x5 0.001 x6 0.001 x7 0.000 x8 0.0 x9 0.000 ... x16 127.0 x17 158.0 x18 20.0 x19 0.0 x20 1
          0580 PA1001 258 1406 6666145.0 88.0 0 0.005 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.002 104.0 697.0 0.0187400 10.0 155.0 1
          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
          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 1
          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
          1582 PA1004 906 630 146.0 134.0 0 5.000 0.017 0.000 0.005 0.004 0.003 0.000 0.000 0.000 0.000 0.000 107.0 500.0 0.170200 5.0 0.0 1
          1583 PA1005 2584 1921 1436 151.0 0 0.002 0.000 0.003 0.000 0.006 0.000 0.000 0.000 0.000 0.000 75.0 522.0 0.200900 10.0 142.0 1
          5 rows x 26 columns
          1584 rows x 26 columns

```

In [14]:

```
#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 [15]:

```
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=
```

CLEANED UP TO HERE

In [16]:

```
#paste in code from before but remove references to encoder
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import BayesianRidge, Ridge
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.svm import OneClassSVM

imputer_params = [
    (SimpleImputer, {"strategy" : ["mean", "median", "most_frequent",]}),
    (IterativeImputer, {"estimator" : [BayesianRidge(), KNeighborsRegressor(n_neighbors = 15),
                                       "max_iter" : [5, 10, 20],
                                       "random_state" : [1234]}),
    (KNNImputer, {"n_neighbors" : [5,10,20,30]})}
]
#choosing to omit "constant" strategy, as it is just filling in all missing values with a constant
#need to check how my missing values are represented - I assume np.nan

imputers = [ctor(**para) for ctor, paras in imputer_params for para in ParameterGrid(paras)]
print(imputers)

params = dict(imputer = imputers, clf = [SVC()])
params
```

[SimpleImputer(), SimpleImputer(strategy='median'), SimpleImputer(strategy='most_frequent'), IterativeImputer(estimator=BayesianRidge(), max_iter=5, random_state=1234), IterativeImputer(estimator=BayesianRidge(), random_state=1234), IterativeImputer(estimator=BayesianRidge(), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=7, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=20, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=5, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=25, random_state=1234), IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=20, random_state=1234), KNNImputer(), KNNImputer(n_neighbors=10), KNNImputer(n_neighbors=20), KNNImputer(n_neighbors=30)]

Out[16]:

```
{'imputer': [SimpleImputer(),
    SimpleImputer(strategy='median'),
    SimpleImputer(strategy='most_frequent'),
    IterativeImputer(estimator=BayesianRidge(), max_iter=5, random_state=1234),
    IterativeImputer(estimator=BayesianRidge(), random_state=1234),
    IterativeImputer(estimator=BayesianRidge(), max_iter=20, random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=5,
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15),
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=15), max_iter=20,
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=5,
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7),
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=7), max_iter=20,
                    random_state=1234),
    IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25), max_iter=5,
                    random_state=1234),
In [17]: IterativeImputer(estimator=KNeighborsRegressor(n_neighbors=25),
from sklearn.model_selection import GridSearchCV
from IterativeImputer.estimator=KNeighborsRegressor(n_neighbors=25), max_iter=20,
                    random_state=1234),
KNNImputer(),
process = Imputer(strategy='median'), SimpleImputer(), ("clf", SVC()))
KNNImputer(n_neighbors=20),
clf_cv = GridSearchCV(process, param_grid=process, params)
'clf': [SVC()])
clf_cv.fit(X_train, y_train)
score = clf_cv.score(X_train, y_train)
print(score)

print(clf_cv.best_params_)
```

0.7842960288808665

```
{'clf': SVC(), 'imputer': SimpleImputer()}
```

gridsearchcv uses the scoring system for the clf provided, which in this case is SVC. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#:~:text=provided%2C%20and%20th Parameters%3A (this means that this is scored using mean accuracy. [https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#:~:text=X%2C%C2%A0y%5B%2C%C2%A0sample_weight_params\(**params\)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#:~:text=X%2C%C2%A0y%5B%2C%C2%A0sample_weight_params(**params))) ([https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#:~:text=X%2C%C2%A0y%5B%2C%C2%A0sample_weight_params\(**params\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#:~:text=X%2C%C2%A0y%5B%2C%C2%A0sample_weight_params(**params)))

Outlier removal

Outlier removal cannot be done in a pipeline - there is no transform method. Therefore, I am wrapping isolationforest in a transformer to try to make it work in a pipeline

In [27]:

```
from sklearn.base import TransformerMixin
from sklearn.ensemble import IsolationForest

class IsolationForestPipePart(TransformerMixin):

    def __init__(self, **kwargs):
        """Impute missing values.

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

        Columns of other types are imputed with mean of column.

        """
        #we are getting the keyword arguments used when we set up the pipe part, so the isolat
        self.kwargs = kwargs
        self.iso = IsolationForest(**self.kwargs)

    def fit(self, X, y=None):

        #when fit is ran, fit on the isolation forest
        self.iso.fit(X)

        return self

    def transform(self, X, y):

        #predict the outliers
        preds = self.iso.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
        print("X Before: ", X.shape)
        print("X After: ", X[mask].shape)

        if y is not None:
            print("y Before: ", y.shape)
            #print("y After: ", self.y[mask].shape)
            return (X[mask], y[mask])
        else:
            return X[mask]

X_train_imputed = SimpleImputer().fit_transform(X_train, y_train)
myIsoPipe = IsolationForestPipePart(max_samples=31258, random_state = 1, contamination= 0.01)
myIsoPipe.fit(X_train_imputed, y_train)
X_train_transformed, y_train_transformed = myIsoPipe.transform(X_train_imputed, y_train)
print(X_train.shape)
print(X_train_transformed.shape)
print(y_train.shape)
print(y_train_transformed.shape)
#print(y_validate_transformed)
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: User
Warning: max_samples (31258) is greater than the total number of samples (1108).
max_samples will be set to n_samples for estimation.
warn(
X Before: (1108, 24)
X After: (1096, 24)
y Before: (1108,)
(1108, 24)
(1096, 24)
(1108,)
(1096,)
```

```
Seems to be working
```


In [31]:

```
#the same pipe part, but any outlier detector can be passed in
from sklearn.neighbors import LocalOutlierFactor
from sklearn.base import TransformerMixin
class OutlierDetectionPipePart(TransformerMixin):

    def __init__(self, **kwargs):
        """Impute missing values.

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

        Columns of other types are imputed with mean of column.

        """
        #we are getting the keyword arguments used when we set up the pipe part, so the outlier
        self.outlierStrat = kwargs.pop("strategy", IsolationForest())
        self.kwargs = kwargs
        self.outlierStrat.set_params(**self.kwargs)

    def fit(self, X, y=None):

        self.outlierStrat.fit(X)

        return self
#maybe change y so it doesn't = none
    def transform(self, X, y=None):

        preds = self.outlierStrat.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

        if y is not None:
            #print("y Before: ", y.shape)
            #print("y After: ", self.y[mask].shape)
            return (X[mask], y[mask])
        else:
            return X[mask]

X_train_imputed = SimpleImputer().fit_transform(X_train, y_train)
myIsoPipe = OutlierDetectionPipePart(strategy = IsolationForest(), max_samples=31258, random_state=42)
myIsoPipe.fit(X_train_imputed, y_train)
X_train_transformed, y_train_transformed = myIsoPipe.transform(X_train_imputed, y_train)
print(X_train.shape)
print(X_train_transformed.shape)
print(y_train.shape)
print(y_train_transformed.shape)

#setting novelty to true, LOF should only be used to transform things which aren't the training
myIsoPipe = OutlierDetectionPipePart(strategy = LocalOutlierFactor(n_neighbors=20, contamination=0.01))
myIsoPipe.fit(X_train_imputed, y_train)
X_train_transformed, y_train_transformed = myIsoPipe.transform(X_train_imputed, y_train)
print(X_train.shape)
print(X_train_transformed.shape)
print(y_train.shape)
print(y_train_transformed.shape)
```

```
print(y_train_transformed.shape)
# print(y_validate_transformed)
C:\Users\Willi\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: User
Warning: max_samples (31258) is greater than the total number of samples (1108).
max_samples will be set to n_samples for estimation.
warn(
(1108, 24)
(1096, 24)
(1108,)
(1096,)
(1108, 24)
(1098, 24)
(1108,)
(1098,)
```

In [32]:

```
outlier_params = [
    (IsolationForestPipePart, {
        "n_estimators" : [10, 50, 100, 200],
        "max_samples" : ["auto", 128, 256, 516, 1024],
        "contamination" : ["auto", 0.01, 0.05, 0.1, 0.25, 0.5]
    })
]

#Loop through outlier_params, and make a List of every combination of each parameter
outliers = [ctor(**para) for ctor, paras in outlier_params for para in ParameterGrid(paras)]
print(outliers)

params2 = dict(outlier = outliers, clf = [SVC()])
params2
```

```
t object at 0x00000265AFD353A0>, <__main__.IsolationForestPipePart object at
0x00000265AFD8BC10>, <__main__.IsolationForestPipePart object at 0x00000265AF
D8B820>, <__main__.IsolationForestPipePart object at 0x00000265AFD8BF10>, <
__main__.IsolationForestPipePart object at 0x00000265B5415610>, <__main__.Isola
tionForestPipePart object at 0x00000265B5415850>, <__main__.IsolationForestPi
pePart object at 0x00000265B5415F10>, <__main__.IsolationForestPipePart objec
t at 0x00000265B5415CA0>, <__main__.IsolationForestPipePart object at 0x00000
265B5415820>, <__main__.IsolationForestPipePart object at 0x00000265B5410CA0
>, <__main__.IsolationForestPipePart object at 0x00000265B54105E0>, <__main_
_.IsolationForestPipePart object at 0x00000265B5410B20>, <__main__.IsolationF
orestPipePart object at 0x00000265B5410A90>, <__main__.IsolationForestPipePar
t object at 0x00000265B5410820>, <__main__.IsolationForestPipePart object at
0x00000265B54109A0>, <__main__.IsolationForestPipePart object at 0x00000265B5
406730>, <__main__.IsolationForestPipePart object at 0x00000265B5406E20>, <
__main__.IsolationForestPipePart object at 0x00000265B5406EB0>, <__main__.Isola
tionForestPipePart object at 0x00000265B5406940>, <__main__.IsolationForestPi
pePart object at 0x00000265B5406130>, <__main__.IsolationForestPipePart objec
t at 0x00000265B54089D0>, <__main__.IsolationForestPipePart object at 0x00000
265B5408220>, <__main__.IsolationForestPipePart object at 0x00000265B54085E0
` \ main` IsolationForestPipePart object at 0x00000265B54085E0` \ main`
```

In [33]:

```
process = [("outlier", IsolationForestPipePart()), ("clf", SVC())]

clf_cv = GridSearchCV(Pipeline(process), params2)

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

print(clf_cv.best_params_)
```

```

-----
ValueError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18120\2999149146.py in <module>
      3 clf_cv = GridSearchCV(Pipeline(process), params2)
      4
----> 5 clf_cv.fit(X_train, y_train)
      6 score = clf_cv.score(X_validate, y_validate)
      7 print(score)

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X,
y, groups, **fit_params)
    872         return results
    873
--> 874     self._run_search(evaluate_candidates)
    875
    876     # multimetric is determined here because in the case of a ca
lable

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in _run_search
(self, evaluate_candidates)
    1386     def _run_search(self, evaluate_candidates):
    1387         """Search all candidates in param_grid"""
--> 1388         evaluate_candidates(ParameterGrid(self.param_grid))
    1389
    1390

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in evaluate_can
didates(candidate_params, cv, more_results)
    849
    850
--> 851     _warn_or_raise_about_fit_failures(out, self.error_score)
    852
    853     # For callable self.scoring, the return type is only kno
w after

~\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in _warn_or
_raise_about_fit_failures(results, error_score)
    365         f"Below are more details about the failures:\n{fit_error
s_summary}"
    366
--> 367         raise ValueError(all_fits_failed_message)
    368
    369     else:

ValueError:
All the 600 fits failed.
It is very likely that your model is misconfigured.
You can try to debug the error by setting error_score='raise'.

Below are more details about the failures:
-----
600 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\model_selection\_valid
ation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\pipeline.py", line 40
1, in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\pipeline.py", line 35
9, in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "C:\Users\Will\anaconda3\lib\site-packages\joblib\memory.py", line 349, i

```

```
n __call__
    return self.func(*args, **kwargs)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\pipeline.py", line 89
3, in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\utils\_set_output.py",
line 140, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\base.py", line 881, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
File "C:\Users\Will\AppData\Local\Temp\ipykernel_18120/1983944836.py", line 2
1, in fit
    self.iso.fit(X)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py",
line 290, in fit
    X = self._validate_data(X, accept_sparse=["csc"], dtype=tree_dtype)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\base.py", line 565, in
_validate_data
    X = check_array(X, input_name="X", **check_params)
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\utils\validation.py",
line 921, in check_array
    _assert_all_finite(
File "C:\Users\Will\anaconda3\lib\site-packages\sklearn\utils\validation.py",
line 161, in _assert_all_finite
    raise ValueError(msg_err)
ValueError: Input X contains NaN.
```

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

I don't think I'm ever going to be able to have outlier removal in a pipeline. Even if I create a custom transformer which implements "fit" and "transform", pipelines do not always pass in the y values by default - meaning that the y values cannot be transformed, meaning that when outlier rows are deleted from x, there are too many rows in y, as none have been deleted. I could try doing the steps then cross validating tho?

In [46]:

```
from sklearn.ensemble import IsolationForest
#finding all combos of isolation forest params

outlier_params = [
    (IsolationForest, {
        "n_estimators" : [10, 50, 100, 200],
        "max_samples" : ["auto", 128, 256, 516, 1024],
        "contamination" : ["auto", 0.01, 0.05, 0.1, 0.25, 0.5],
        "random_state" : [1]
    })
]

iso_outliers = [ctor(**para) for ctor, paras in outlier_params for para in ParameterGrid(paras)]
print(iso_outliers)
```

```
[IsolationForest(n_estimators=10, random_state=1), IsolationForest(n_estimators=50, random_state=1), IsolationForest(random_state=1), IsolationForest(n_estimators=200, random_state=1), IsolationForest(max_samples=128, n_estimators=10, random_state=1), IsolationForest(max_samples=128, n_estimators=50, random_state=1), IsolationForest(max_samples=128, random_state=1), IsolationForest(max_samples=128, n_estimators=200, random_state=1), IsolationForest(max_samples=256, n_estimators=10, random_state=1), IsolationForest(max_samples=256, n_estimators=50, random_state=1), IsolationForest(max_samples=256, n_estimators=200, random_state=1), IsolationForest(max_samples=512, n_estimators=10, random_state=1), IsolationForest(max_samples=512, n_estimators=50, random_state=1), IsolationForest(max_samples=512, n_estimators=200, random_state=1), IsolationForest(max_samples=1024, n_estimators=10, random_state=1), IsolationForest(max_samples=1024, n_estimators=50, random_state=1), IsolationForest(max_samples=1024, n_estimators=200, random_state=1), IsolationForest(contamination=0.01, n_estimators=10, random_state=1), IsolationForest(contamination=0.01, n_estimators=50, random_state=1), IsolationForest(contamination=0.01, n_estimators=200, random_state=1), IsolationForest(contamination=0.01, max_samples=128, n_estimators=10, random_state=1), IsolationForest(contamination=0.01, max_samples=128, n_estimators=50, random_state=1), IsolationForest(contamination=0.01, max_samples=128, random_state=1), IsolationForest(contamination=0.01, max_samples=128, n_estimators=200, random_state=1), IsolationForest(contamination=0.01, max_samples=256, n_estimators=10, random_state=1), IsolationForest(contamination=0.01, max_samples=256, n_estimators=50, random_state=1), IsolationForest(contamination=0.01, max_samples=256, random_state=1), IsolationForest(contamination=0.01, max_samples=256, n_estimators=200, random_state=1), IsolationForest(contamination=0.01, max_samples=516, n_estimators=10, random_state=1), IsolationForest(contamination=0.01, max_samples=516, n_estimators=50, random_state=1), IsolationForest(contamination=0.01, max_samples=516, n_estimators=200, random_state=1), IsolationForest(contamination=0.01, max_samples=1024, n_estimators=10, random_state=1), IsolationForest(contamination=0.01, max_samples=1024, n_estimators=50, random_state=1), IsolationForest(contamination=0.01, max_samples=1024, n_estimators=200, random_state=1), IsolationForest(contamination=0.05, n_estimators=10, random_state=1), IsolationForest(contamination=0.05, n_estimators=50, random_state=1), IsolationForest(contamination=0.05, n_estimators=200, random_state=1), IsolationForest(contamination=0.05, max_samples=128, n_estimators=10, random_state=1), IsolationForest(contamination=0.05, max_samples=128, n_estimators=50, random_state=1), IsolationForest(contamination=0.05, max_samples=128, random_state=1), IsolationForest(contamination=0.05, max_samples=128, n_estimators=200, random_state=1), IsolationForest(contamination=0.05, max_samples=256, n_estimators=10, random_state=1), IsolationForest(contamination=0.05, max_samples=256, n_estimators=50, random_state=1), IsolationForest(contamination=0.05, max_samples=256, random_state=1), IsolationForest(contamination=0.05, max_samples=256, n_estimators=200, random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=10, random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=50, random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=200, random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=10, random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=50, random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=200, random_state=1)]
```

```
random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.05, max_samples=516, random_state=1), IsolationForest(contamination=0.05, max_samples=516, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.05, max_samples=1024, random_state=1), IsolationForest(contamination=0.05, max_samples=1024, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.1, n_estimators=10, random_state=1), IsolationForest(contamination=0.1, n_estimators=50, random_state=1), IsolationForest(contamination=0.1, random_state=1), IsolationForest(contamination=0.1, n_estimators=200, random_state=1), IsolationForest(contamination=0.1, max_samples=128, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=128, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=128, random_state=1), IsolationForest(contamination=0.1, max_samples=128, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=256, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=256, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=256, random_state=1), IsolationForest(contamination=0.1, max_samples=256, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=516, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=516, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=516, random_state=1), IsolationForest(contamination=0.1, max_samples=516, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=1024, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=1024, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.1, max_samples=1024, random_state=1), IsolationForest(contamination=0.1, max_samples=1024, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.25, n_estimators=10, random_state=1), IsolationForest(contamination=0.25, n_estimators=50, random_state=1), IsolationForest(contamination=0.25, random_state=1), IsolationForest(contamination=0.25, n_estimators=200, random_state=1), IsolationForest(contamination=0.25, max_samples=128, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=128, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=128, random_state=1), IsolationForest(contamination=0.25, max_samples=128, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=256, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=256, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=256, random_state=1), IsolationForest(contamination=0.25, max_samples=256, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=512, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=512, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=512, random_state=1), IsolationForest(contamination=0.25, max_samples=512, n_estimators=200,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=1024, n_estimators=10,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=1024, n_estimators=50,
                                 random_state=1), IsolationForest(contamination=0.25, max_samples=1024, random_state=1), IsolationForest(contamination=0.25, max_samples=1024, n_estimators=200,
```

```
=516, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.25, max_samples
=516, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.25, max_samples
=516, random_state=1), IsolationForest(contamination=0.25, max_samples=516, n_es
timators=200,
           random_state=1), IsolationForest(contamination=0.25, max_samples
=1024, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.25, max_samples
=1024, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.25, max_samples
=1024, random_state=1), IsolationForest(contamination=0.25, max_samples=1024, n_
estimators=200,
           random_state=1), IsolationForest(contamination=0.5, n_estimators
=10, random_state=1), IsolationForest(contamination=0.5, n_estimators=50, random
_state=1), IsolationForest(contamination=0.5, random_state=1), IsolationForest(c
ontamination=0.5, n_estimators=200, random_state=1), IsolationForest(contaminati
on=0.5, max_samples=128, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
128, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
128, random_state=1), IsolationForest(contamination=0.5, max_samples=128, n_esi
mators=200,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
256, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
256, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
256, random_state=1), IsolationForest(contamination=0.5, max_samples=256, n_esi
mators=200,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
516, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
516, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
516, random_state=1), IsolationForest(contamination=0.5, max_samples=516, n_esi
mators=200,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
1024, n_estimators=10,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
1024, n_estimators=50,
           random_state=1), IsolationForest(contamination=0.5, max_samples=
1024, random_state=1), IsolationForest(contamination=0.5, max_samples=1024, n_es
timators=200,
           random_state=1)]
```

In [47]:

```
params3 = dict(outlier = iso_outliers, clf = [SVC()])
params3
IsolationForest(contamination=0.01, max_samples=256, n_estimators=50,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=256, random_state=1),
IsolationForest(contamination=0.01, max_samples=256, n_estimators=200,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=10,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=50,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=516, random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=200,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=10,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=50,
                 random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=200,
                 random_state=1),
IsolationForest(contamination=0.05, n_estimators=10, random_state=1),
```


In [48]:

```
diff_iso_forests = [IsolationForest(n_estimators=10, random_state=1),
IsolationForest(n_estimators=50, random_state=1),
IsolationForest(random_state=1),
IsolationForest(n_estimators=200, random_state=1),
IsolationForest(max_samples=128, n_estimators=10, random_state=1),
IsolationForest(max_samples=128, n_estimators=50, random_state=1),
IsolationForest(max_samples=128, random_state=1),
IsolationForest(max_samples=128, n_estimators=200, random_state=1),
IsolationForest(max_samples=256, n_estimators=10, random_state=1),
IsolationForest(max_samples=256, n_estimators=50, random_state=1),
IsolationForest(max_samples=256, random_state=1),
IsolationForest(max_samples=256, n_estimators=200, random_state=1),
IsolationForest(max_samples=516, n_estimators=10, random_state=1),
IsolationForest(max_samples=516, n_estimators=50, random_state=1),
IsolationForest(max_samples=516, random_state=1),
IsolationForest(max_samples=516, n_estimators=200, random_state=1),
IsolationForest(max_samples=1024, n_estimators=10, random_state=1),
IsolationForest(max_samples=1024, n_estimators=50, random_state=1),
IsolationForest(max_samples=1024, random_state=1),
IsolationForest(max_samples=1024, n_estimators=200, random_state=1),
IsolationForest(contamination=0.01, n_estimators=10, random_state=1),
IsolationForest(contamination=0.01, n_estimators=50, random_state=1),
IsolationForest(contamination=0.01, random_state=1),
IsolationForest(contamination=0.01, n_estimators=200, random_state=1),
IsolationForest(contamination=0.01, max_samples=128, n_estimators=10,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=128, n_estimators=50,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=128, random_state=1),
IsolationForest(contamination=0.01, max_samples=128, n_estimators=200,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=256, n_estimators=10,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=256, n_estimators=50,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=256, random_state=1),
IsolationForest(contamination=0.01, max_samples=256, n_estimators=200,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=10,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=50,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=516, random_state=1),
IsolationForest(contamination=0.01, max_samples=516, n_estimators=200,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=10,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=50,
    random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, random_state=1),
IsolationForest(contamination=0.01, max_samples=1024, n_estimators=200,
    random_state=1),
IsolationForest(contamination=0.05, n_estimators=10, random_state=1),
IsolationForest(contamination=0.05, n_estimators=50, random_state=1),
IsolationForest(contamination=0.05, random_state=1),
IsolationForest(contamination=0.05, n_estimators=200, random_state=1),
IsolationForest(contamination=0.05, max_samples=128, n_estimators=10,
    random_state=1),
IsolationForest(contamination=0.05, max_samples=128, n_estimators=50,
    random_state=1),
IsolationForest(contamination=0.05, max_samples=128, random_state=1),
```

```
IsolationForest(contamination=0.05, max_samples=128, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=256, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=256, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=256, random_state=1),  
IsolationForest(contamination=0.05, max_samples=256, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=516, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=516, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=516, random_state=1),  
IsolationForest(contamination=0.05, max_samples=516, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=1024, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=1024, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.05, max_samples=1024, random_state=1),  
IsolationForest(contamination=0.05, max_samples=1024, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.1, n_estimators=10, random_state=1),  
IsolationForest(contamination=0.1, n_estimators=50, random_state=1),  
IsolationForest(contamination=0.1, random_state=1),  
IsolationForest(contamination=0.1, n_estimators=200, random_state=1),  
IsolationForest(contamination=0.1, max_samples=128, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=128, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=128, random_state=1),  
IsolationForest(contamination=0.1, max_samples=128, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=256, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=256, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=256, random_state=1),  
IsolationForest(contamination=0.1, max_samples=256, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=516, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=516, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=516, random_state=1),  
IsolationForest(contamination=0.1, max_samples=516, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=1024, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=1024, n_estimators=50,  
                 random_state=1),  
IsolationForest(contamination=0.1, max_samples=1024, random_state=1),  
IsolationForest(contamination=0.1, max_samples=1024, n_estimators=200,  
                 random_state=1),  
IsolationForest(contamination=0.25, n_estimators=10, random_state=1),  
IsolationForest(contamination=0.25, n_estimators=50, random_state=1),  
IsolationForest(contamination=0.25, random_state=1),  
IsolationForest(contamination=0.25, n_estimators=200, random_state=1),  
IsolationForest(contamination=0.25, max_samples=128, n_estimators=10,  
                 random_state=1),  
IsolationForest(contamination=0.25, max_samples=128, n_estimators=50,  
                 random_state=1),
```

```
IsolationForest(contamination=0.25, max_samples=128, random_state=1),
IsolationForest(contamination=0.25, max_samples=128, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=256, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=256, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=256, random_state=1),
IsolationForest(contamination=0.25, max_samples=256, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=516, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=516, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=516, random_state=1),
IsolationForest(contamination=0.25, max_samples=516, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=1024, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=1024, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.25, max_samples=1024, random_state=1),
IsolationForest(contamination=0.25, max_samples=1024, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.5, n_estimators=10, random_state=1),
IsolationForest(contamination=0.5, n_estimators=50, random_state=1),
IsolationForest(contamination=0.5, random_state=1),
IsolationForest(contamination=0.5, n_estimators=200, random_state=1),
IsolationForest(contamination=0.5, max_samples=128, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=128, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=128, random_state=1),
IsolationForest(contamination=0.5, max_samples=128, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=256, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=256, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=256, random_state=1),
IsolationForest(contamination=0.5, max_samples=256, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=516, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=516, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=516, random_state=1),
IsolationForest(contamination=0.5, max_samples=516, n_estimators=200,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=1024, n_estimators=10,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=1024, n_estimators=50,
                random_state=1),
IsolationForest(contamination=0.5, max_samples=1024, random_state=1),
IsolationForest(contamination=0.5, max_samples=1024, n_estimators=200,
                random_state=1)]
```

In [52]:

```
#finding all combos of LOF params
#no random_state because LOF doesnt have a random state
from sklearn.neighbors import LocalOutlierFactor

lof_outlier_params = [
    (LocalOutlierFactor, {
        "n_neighbors" : [5, 10, 20, 40],
        "algorithm" : ["auto", "ball_tree", "kd_tree", "brute"],
        "contamination" : ["auto", 0.01, 0.05, 0.1, 0.25, 0.5],
        "novelty" : [True]
    })
]

lof_outliers = [ctor(**para) for ctor, para in lof_outlier_params for para in ParameterGrid(p
params4 = dict(outlier = lof_outliers)
params4
```

Out[52]:

```
{'outlier': [LocalOutlierFactor(n_neighbors=5, novelty=True),
LocalOutlierFactor(n_neighbors=10, novelty=True),
LocalOutlierFactor(novelty=True),
LocalOutlierFactor(n_neighbors=40, novelty=True),
LocalOutlierFactor(contamination=0.01, n_neighbors=5, novelty=True),
LocalOutlierFactor(contamination=0.01, n_neighbors=10, novelty=True),
LocalOutlierFactor(contamination=0.01, novelty=True),
LocalOutlierFactor(contamination=0.01, n_neighbors=40, novelty=True),
LocalOutlierFactor(contamination=0.05, n_neighbors=5, novelty=True),
LocalOutlierFactor(contamination=0.05, n_neighbors=10, novelty=True),
LocalOutlierFactor(contamination=0.05, novelty=True),
LocalOutlierFactor(contamination=0.05, n_neighbors=40, novelty=True),
LocalOutlierFactor(contamination=0.1, n_neighbors=5, novelty=True),
LocalOutlierFactor(contamination=0.1, n_neighbors=10, novelty=True),
LocalOutlierFactor(contamination=0.1, novelty=True),
LocalOutlierFactor(contamination=0.1, n_neighbors=40, novelty=True),
LocalOutlierFactor(contamination=0.25, n_neighbors=5, novelty=True),
LocalOutlierFactor(contamination=0.25, n_neighbors=10, novelty=True).
```


In [53]:

```
diff_lofs = [LocalOutlierFactor(n_neighbors=5, novelty=True),
 LocalOutlierFactor(n_neighbors=10, novelty=True),
 LocalOutlierFactor(novelty=True),
 LocalOutlierFactor(n_neighbors=40, novelty=True),
 LocalOutlierFactor(contamination=0.01, n_neighbors=5, novelty=True),
 LocalOutlierFactor(contamination=0.01, n_neighbors=10, novelty=True),
 LocalOutlierFactor(contamination=0.01, novelty=True),
 LocalOutlierFactor(contamination=0.01, n_neighbors=40, novelty=True),
 LocalOutlierFactor(contamination=0.05, n_neighbors=5, novelty=True),
 LocalOutlierFactor(contamination=0.05, n_neighbors=10, novelty=True),
 LocalOutlierFactor(contamination=0.05, novelty=True),
 LocalOutlierFactor(contamination=0.05, n_neighbors=40, novelty=True),
 LocalOutlierFactor(contamination=0.1, n_neighbors=5, novelty=True),
 LocalOutlierFactor(contamination=0.1, n_neighbors=10, novelty=True),
 LocalOutlierFactor(contamination=0.1, novelty=True),
 LocalOutlierFactor(contamination=0.1, n_neighbors=40, novelty=True),
 LocalOutlierFactor(contamination=0.25, n_neighbors=5, novelty=True),
 LocalOutlierFactor(contamination=0.25, n_neighbors=10, novelty=True),
 LocalOutlierFactor(contamination=0.25, novelty=True),
 LocalOutlierFactor(contamination=0.25, n_neighbors=40, novelty=True),
 LocalOutlierFactor(contamination=0.5, n_neighbors=5, novelty=True),
 LocalOutlierFactor(contamination=0.5, n_neighbors=10, novelty=True),
 LocalOutlierFactor(contamination=0.5, novelty=True),
 LocalOutlierFactor(contamination=0.5, n_neighbors=40, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', n_neighbors=5, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', n_neighbors=10, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', n_neighbors=40, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.01, n_neighbors=5,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.01, n_neighbors=10,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.01, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.01, n_neighbors=40,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.05, n_neighbors=5,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.05, n_neighbors=10,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.05, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.05, n_neighbors=40,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.1, n_neighbors=5,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.1, n_neighbors=10,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.1, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.1, n_neighbors=40,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.25, n_neighbors=5,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.25, n_neighbors=10,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.25, novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.25, n_neighbors=40,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.5, n_neighbors=5,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.5, n_neighbors=10,
                    novelty=True),
 LocalOutlierFactor(algorithm='ball_tree', contamination=0.5, novelty=True),
```

```
LocalOutlierFactor(algorithm='ball_tree', contamination=0.5, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', n_neighbors=5, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', n_neighbors=10, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', novelty=True),
LocalOutlierFactor(algorithm='kd_tree', n_neighbors=40, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.01, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.01, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.01, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.01, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.05, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.05, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.05, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.05, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.1, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.1, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.1, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.25, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.25, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.25, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.25, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.5, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.5, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.5, novelty=True),
LocalOutlierFactor(algorithm='kd_tree', contamination=0.5, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', n_neighbors=5, novelty=True),
LocalOutlierFactor(algorithm='brute', n_neighbors=10, novelty=True),
LocalOutlierFactor(algorithm='brute', novelty=True),
LocalOutlierFactor(algorithm='brute', n_neighbors=40, novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.01, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.01, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.01, novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.01, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.05, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.05, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.05, novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.05, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.1, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.1, n_neighbors=10,
                   novelty=True),
```

```
LocalOutlierFactor(algorithm='brute', contamination=0.1, novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.1, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.25, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.25, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.25, novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.25, n_neighbors=40,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.5, n_neighbors=5,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.5, n_neighbors=10,
                   novelty=True),
LocalOutlierFactor(algorithm='brute', contamination=0.5, novelty=True),
In []: LocalOutlierFactor(algorithm='brute', contamination=0.5, n_neighbors=40,
                           novelty=True)]
```

```
#finding all combos of DBSCAN params
#no random_state because DBSCANM doesnt have a random state
#accding to sklearn, eps is the mos timportant dbSCAN paramatrer to choose, so it is given to
from sklearn.cluster import DBSCAN
```

```
dbSCAN_outlier_params = [
    (DBSCAN, {
        "eps" : [0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 2.0, 5.0],
        "min_samples" : [2,3,5,10],
        "algorithm" : ["auto", "ball_tree", "kd_tree", "brute"]
    })
]
```

```
dbSCAN_outliers = [ctor(**para) for ctor, paras in dbSCAN_outlier_params for para in Parameter
params4 = dict(outlier = dbSCAN_outliers)
params4
```


In []:

```
diff_dbscans = [DBSCAN(eps=0.1, min_samples=2),
 DBSCAN(eps=0.1, min_samples=3),
 DBSCAN(eps=0.1),
 DBSCAN(eps=0.1, min_samples=10),
 DBSCAN(eps=0.2, min_samples=2),
 DBSCAN(eps=0.2, min_samples=3),
 DBSCAN(eps=0.2),
 DBSCAN(eps=0.2, min_samples=10),
 DBSCAN(eps=0.3, min_samples=2),
 DBSCAN(eps=0.3, min_samples=3),
 DBSCAN(eps=0.3),
 DBSCAN(eps=0.3, min_samples=10),
 DBSCAN(eps=0.4, min_samples=2),
 DBSCAN(eps=0.4, min_samples=3),
 DBSCAN(eps=0.4),
 DBSCAN(eps=0.4, min_samples=10),
 DBSCAN(min_samples=2),
 DBSCAN(min_samples=3),
 DBSCAN(),
 DBSCAN(min_samples=10),
 DBSCAN(eps=1.0, min_samples=2),
 DBSCAN(eps=1.0, min_samples=3),
 DBSCAN(eps=1.0),
 DBSCAN(eps=1.0, min_samples=10),
 DBSCAN(eps=2.0, min_samples=2),
 DBSCAN(eps=2.0, min_samples=3),
 DBSCAN(eps=2.0),
 DBSCAN(eps=2.0, min_samples=10),
 DBSCAN(eps=5.0, min_samples=2),
 DBSCAN(eps=5.0, min_samples=3),
 DBSCAN(eps=5.0),
 DBSCAN(eps=5.0, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=0.1, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=0.1, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=0.1),
 DBSCAN(algorithm='ball_tree', eps=0.1, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=0.2, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=0.2, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=0.2),
 DBSCAN(algorithm='ball_tree', eps=0.2, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=0.3, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=0.3, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=0.3),
 DBSCAN(algorithm='ball_tree', eps=0.3, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=0.4, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=0.4, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=0.4),
 DBSCAN(algorithm='ball_tree', eps=0.4, min_samples=10),
 DBSCAN(algorithm='ball_tree', min_samples=2),
 DBSCAN(algorithm='ball_tree', min_samples=3),
 DBSCAN(algorithm='ball_tree'),
 DBSCAN(algorithm='ball_tree', min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=1.0, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=1.0, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=1.0),
 DBSCAN(algorithm='ball_tree', eps=1.0, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=2.0, min_samples=2),
 DBSCAN(algorithm='ball_tree', eps=2.0, min_samples=3),
 DBSCAN(algorithm='ball_tree', eps=2.0),
 DBSCAN(algorithm='ball_tree', eps=2.0, min_samples=10),
 DBSCAN(algorithm='ball_tree', eps=5.0, min_samples=2),
```

```
DBSCAN(algorithm='ball_tree', eps=5.0, min_samples=3),  
DBSCAN(algorithm='ball_tree', eps=5.0),  
DBSCAN(algorithm='ball_tree', eps=5.0, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=0.1, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=0.1, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=0.1),  
DBSCAN(algorithm='kd_tree', eps=0.1, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=0.2, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=0.2, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=0.2),  
DBSCAN(algorithm='kd_tree', eps=0.2, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=0.3, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=0.3, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=0.3),  
DBSCAN(algorithm='kd_tree', eps=0.3, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=0.4, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=0.4, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=0.4),  
DBSCAN(algorithm='kd_tree', eps=0.4, min_samples=10),  
DBSCAN(algorithm='kd_tree', min_samples=2),  
DBSCAN(algorithm='kd_tree', min_samples=3),  
DBSCAN(algorithm='kd_tree'),  
DBSCAN(algorithm='kd_tree', min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=1.0, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=1.0, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=1.0),  
DBSCAN(algorithm='kd_tree', eps=1.0, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=2.0, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=2.0, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=2.0),  
DBSCAN(algorithm='kd_tree', eps=2.0, min_samples=10),  
DBSCAN(algorithm='kd_tree', eps=5.0, min_samples=2),  
DBSCAN(algorithm='kd_tree', eps=5.0, min_samples=3),  
DBSCAN(algorithm='kd_tree', eps=5.0),  
DBSCAN(algorithm='kd_tree', eps=5.0, min_samples=10),  
DBSCAN(algorithm='brute', eps=0.1, min_samples=2),  
DBSCAN(algorithm='brute', eps=0.1, min_samples=3),  
DBSCAN(algorithm='brute', eps=0.1),  
DBSCAN(algorithm='brute', eps=0.1, min_samples=10),  
DBSCAN(algorithm='brute', eps=0.2, min_samples=2),  
DBSCAN(algorithm='brute', eps=0.2, min_samples=3),  
DBSCAN(algorithm='brute', eps=0.2),  
DBSCAN(algorithm='brute', eps=0.2, min_samples=10),  
DBSCAN(algorithm='brute', eps=0.3, min_samples=2),  
DBSCAN(algorithm='brute', eps=0.3, min_samples=3),  
DBSCAN(algorithm='brute', eps=0.3),  
DBSCAN(algorithm='brute', eps=0.3, min_samples=10),  
DBSCAN(algorithm='brute', eps=0.4, min_samples=2),  
DBSCAN(algorithm='brute', eps=0.4, min_samples=3),  
DBSCAN(algorithm='brute', eps=0.4),  
DBSCAN(algorithm='brute', eps=0.4, min_samples=10),  
DBSCAN(algorithm='brute', min_samples=2),  
DBSCAN(algorithm='brute', min_samples=3),  
DBSCAN(algorithm='brute'),  
DBSCAN(algorithm='brute', min_samples=10),  
DBSCAN(algorithm='brute', eps=1.0, min_samples=2),  
DBSCAN(algorithm='brute', eps=1.0, min_samples=3),  
DBSCAN(algorithm='brute', eps=1.0),  
DBSCAN(algorithm='brute', eps=1.0, min_samples=10),  
DBSCAN(algorithm='brute', eps=2.0, min_samples=2),  
DBSCAN(algorithm='brute', eps=2.0, min_samples=3),  
DBSCAN(algorithm='brute', eps=2.0),  
DBSCAN(algorithm='brute', eps=2.0, min_samples=10),  
DBSCAN(algorithm='brute', eps=2.0, min_samples=10)
```

```
DBSCAN(algorithm='brute', eps=5.0, min_samples=2),
DBSCAN(algorithm='brute', eps=5.0, min_samples=3),
DBSCAN(algorithm='brute', eps=5.0),
According to https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623 (https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623), DBSCAN is bad for high dimensionality data. therefore we will not be using it
```

In [54]:

```
#finding all combos of OneClassSVM params
#no random_state because OneClassSVM doesnt have a random state

#not using precomputed kernel as it requires a square matrix, and the dataset is not square.

from sklearn.svm import OneClassSVM

OneClassSVM_outlier_params = [
    (OneClassSVM, {
        "kernel": ["linear", "poly", "rbf", "sigmoid"],
        "gamma": ["scale", "auto"]
    })
]

OneClassSVM_outliers = [ctor(**para) for ctor, paras in OneClassSVM_outlier_params for para in
params4 = dict(outlier = OneClassSVM_outliers)
params4
```

Out[54]:

```
{'outlier': [OneClassSVM(kernel='linear'),
  OneClassSVM(kernel='poly'),
  OneClassSVM(),
  OneClassSVM(kernel='sigmoid'),
  OneClassSVM(gamma='auto', kernel='linear'),
  OneClassSVM(gamma='auto', kernel='poly'),
  OneClassSVM(gamma='auto'),
  OneClassSVM(gamma='auto', kernel='sigmoid')]}  
}
```

In [55]:

```
#diff_onesclass = [OneClassSVM(kernel='linear'),
#  OneClassSVM(kernel='poly'),
#  OneClassSVM(),
#  #OneClassSVM(kernel='sigmoid'),
#  #OneClassSVM(gamma='auto', kernel='linear'),
#  #OneClassSVM(gamma='auto', kernel='poly'),
#  #OneClassSVM(gamma='auto'),
#  #OneClassSVM(gamma='auto', kernel='sigmoid')]

#Removed the combination of gamma = "auto" and kernel = "sigmoid", because it was causing error

diff_onesclass = [OneClassSVM(kernel='linear'),
  OneClassSVM(kernel='poly'),
  OneClassSVM(),
  OneClassSVM(kernel='sigmoid'),
  OneClassSVM(gamma='auto', kernel='linear'),
  OneClassSVM(gamma='auto', kernel='poly'),
  OneClassSVM(gamma='auto')]
```

In [56]:

```
#finding all combos of SGDOneClassSVM params

from sklearn.linear_model import SGDOneClassSVM

SGDOneClassSVM_outlier_params = [
    (SGDOneClassSVM, {
        "nu": [0.1, 0.25, 0.5, 0.75],
        "fit_intercept": [True, False],
        "max_iter": [250, 500, 1000, 2000],
        "random_state": [1],
        "learning_rate": ["constant", "optimal", "invscaling", "adaptive"],
        "eta0": [0.5, 1.0, 2.0]
    })
]

SGDOneClassSVM_outliers = [ctor(**para) for ctor, paras in SGDOneClassSVM_outlier_params for para in paras]
params5 = dict(outlier = SGDOneClassSVM_outliers)
params5

SGDOneClassSVM(eta0=0.5, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, random_state=1),
SGDOneClassSVM(eta0=0.5, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=0.5, max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250, nu=0.1,
                random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250, nu=0.25,
                random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250,
                random_state=1),
```


In [57]:

```
diff_sgdoneclass = [SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=250, nu=0.1,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=250, nu=0.25,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=250, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=250, nu=0.75,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=500, nu=0.1,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=500, nu=0.25,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=500, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=500, nu=0.75,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', nu=0.1, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', nu=0.25, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', nu=0.75, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=2000, nu=0.1,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=2000, nu=0.25,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=2000,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='constant', max_iter=2000, nu=0.75,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=250, nu=0.1, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=250, nu=0.25, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=250, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=250, nu=0.75, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.1, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.25, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=500, random_state=1),
                     SGDOneClassSVM(eta0=0.5, max_iter=500, nu=0.75, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250, nu=0.1,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250, nu=0.25,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=250, nu=0.75,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=500, nu=0.1,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=500, nu=0.25,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=500,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=500, nu=0.75,
                                     random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', nu=0.1, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', nu=0.25, random_state=1),
                     SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', random_state=1),
```

```
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=2000, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=2000, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=2000,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='invscaling', max_iter=2000, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=250, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=250, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=250, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=500, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=500, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=500, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=2000, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=2000, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=2000,
               random_state=1),
SGDOneClassSVM(eta0=0.5, learning_rate='adaptive', max_iter=2000, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant', nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant', nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant', nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
               max_iter=2000, nu=0.25, random_state=1),
```

```
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=250, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=250, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=250, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=500, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=500, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=500, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=2000, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=2000, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=2000, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.1, random_state=1),
```

```
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive', nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive', nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive', nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=0.5, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=250, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=250, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=250, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=250, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=500, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=500, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=500, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=2000, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=2000, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=2000,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, learning_rate='constant', max_iter=2000, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, max_iter=500, random_state=1),
```

```
SGDOneClassSVM(eta0=1.0, max_iter=500, nu=0.75, random_state=1),  
SGDOneClassSVM(eta0=1.0, nu=0.1, random_state=1),  
SGDOneClassSVM(eta0=1.0, nu=0.25, random_state=1),  
SGDOneClassSVM(eta0=1.0, random_state=1),  
SGDOneClassSVM(eta0=1.0, nu=0.75, random_state=1),  
SGDOneClassSVM(eta0=1.0, max_iter=2000, nu=0.1, random_state=1),  
SGDOneClassSVM(eta0=1.0, max_iter=2000, nu=0.25, random_state=1),  
SGDOneClassSVM(eta0=1.0, max_iter=2000, random_state=1),  
SGDOneClassSVM(eta0=1.0, max_iter=2000, nu=0.75, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=250, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=250, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=250,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=250, nu=0.75,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=500, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=500, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=500,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=500, nu=0.75,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', nu=0.1, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', nu=0.25, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', nu=0.75, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=2000, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=2000, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=2000,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='invscaling', max_iter=2000, nu=0.75,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=250, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=250, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=250, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=250, nu=0.75,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=500, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=500, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=500, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=500, nu=0.75,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', nu=0.1, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', nu=0.25, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', nu=0.75, random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=2000, nu=0.1,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=2000, nu=0.25,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=2000,  
    random_state=1),  
SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=2000, nu=0.75,  
    random_state=1),
```

```
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=250, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant', nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant', nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant', nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='constant',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=250, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=250, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=250, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=500, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=500, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=500, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=2000, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=2000, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, max_iter=2000, nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, random_state=1),
```

```
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='invscaling',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive', nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive', nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive', nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=1.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=250, nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=250, nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=250, random_state=1),
```

```
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=250, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=500, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=500, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=500, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=2000, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=2000, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=2000,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=2000, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, random_state=1),
SGDOneClassSVM(eta0=2.0, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=2.0, max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=250, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=250, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=250,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=250, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=500, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=500, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=500,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=500, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=2000, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=2000, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=2000,
    random_state=1),
```

```
SGDOneClassSVM(eta0=2.0, learning_rate='invscaling', max_iter=2000, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=250, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=250, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=250, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=500, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=500, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=500, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=2000, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=2000, nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=2000,
    random_state=1),
SGDOneClassSVM(eta0=2.0, learning_rate='adaptive', max_iter=2000, nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant', nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant', nu=0.25,
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant', nu=0.75,
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='constant',
    max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=250, nu=0.1,
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=250, nu=0.25,
```

```
    random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=250, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=500, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=500, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=500, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=2000, nu=0.1,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=2000, nu=0.25,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, max_iter=2000, nu=0.75,
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling',
               max_iter=2000, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
               max_iter=250, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
               max_iter=250, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
               max_iter=250, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
               max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
               max_iter=250, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
```

```
        max_iter=500, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=500, nu=0.75, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive', nu=0.1,
                 random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive', nu=0.25,
                 random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive', nu=0.75,
                 random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.1, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.25, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, random_state=1),
SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='adaptive',
                 max_iter=2000, nu=0.75, random_state=1)]
```

In [58]:

```
#finding all combos of EllipticEnvelope params
#dont think our dataset has a gaussian distribution, will still be interesting to have a look

from sklearn.covariance import EllipticEnvelope

EllipticEnvelope_outlier_params = [
    (EllipticEnvelope, {
        "assume_centered" : [True, False],
        "support_fraction" : [0.1, 0.25, 0.5, 0.75],
        "contamination" : [0.01, 0.05, 0.1, 0.25, 0.5],
        "random_state" : [1]
    })
]

EllipticEnvelope_outliers = [ctor(**para) for ctor, paras in EllipticEnvelope_outlier_params if
params6 = dict(outlier = EllipticEnvelope_outliers)
params6
```

Out[58]:

```
{'outlier': [EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                               support_fraction=0.1),
              EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                               support_fraction=0.25),
              EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                               support_fraction=0.5),
              EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                               support_fraction=0.75),
              EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                               support_fraction=0.1),
              EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                               support_fraction=0.25),
              EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                               support_fraction=0.5),
              EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                               support_fraction=0.75),
              EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.1),
              EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.25),
              EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.5),
              EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.75),
              EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                               support_fraction=0.1),
              EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                               support_fraction=0.25),
              EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                               support_fraction=0.5),
              EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                               support_fraction=0.75),
              EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                               support_fraction=0.1),
              EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                               support_fraction=0.25),
              EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                               support_fraction=0.5),
              EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                               support_fraction=0.75),
              EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.1),
              EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.25),
              EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.5),
              EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75),
              EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.1),
              EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.25),
              EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.5),
              EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.75),
              EllipticEnvelope(random_state=1, support_fraction=0.1),
              EllipticEnvelope(random_state=1, support_fraction=0.25),
              EllipticEnvelope(random_state=1, support_fraction=0.5),
              EllipticEnvelope(random_state=1, support_fraction=0.75),
              EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.1),
              EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.25),
              EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.5),
              EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.75),
              EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.1),
              EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.25),
              EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.5),
              EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.75)]}
```

In [59]:

```
diff_envelopes = [EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                                    support_fraction=0.1),
                  EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                                    support_fraction=0.25),
                  EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                                    support_fraction=0.5),
                  EllipticEnvelope(assume_centered=True, contamination=0.01, random_state=1,
                                    support_fraction=0.75),
                  EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                                    support_fraction=0.1),
                  EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                                    support_fraction=0.25),
                  EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                                    support_fraction=0.5),
                  EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1,
                                    support_fraction=0.75),
                  EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.1),
                  EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.25),
                  EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.5),
                  EllipticEnvelope(assume_centered=True, random_state=1, support_fraction=0.75),
                  EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                                    support_fraction=0.1),
                  EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                                    support_fraction=0.25),
                  EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                                    support_fraction=0.5),
                  EllipticEnvelope(assume_centered=True, contamination=0.25, random_state=1,
                                    support_fraction=0.75),
                  EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                                    support_fraction=0.1),
                  EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                                    support_fraction=0.25),
                  EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                                    support_fraction=0.5),
                  EllipticEnvelope(assume_centered=True, contamination=0.5, random_state=1,
                                    support_fraction=0.75),
                  EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.1),
                  EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.25),
                  EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.5),
                  EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75),
                  EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.1),
                  EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.25),
                  EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.5),
                  EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.75),
                  EllipticEnvelope(random_state=1, support_fraction=0.1),
                  EllipticEnvelope(random_state=1, support_fraction=0.25),
                  EllipticEnvelope(random_state=1, support_fraction=0.5),
                  EllipticEnvelope(random_state=1, support_fraction=0.75),
                  EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.1),
                  EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.25),
                  EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.5),
                  EllipticEnvelope(contamination=0.25, random_state=1, support_fraction=0.75),
                  EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.1),
                  EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.25),
                  EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.5),
                  EllipticEnvelope(contamination=0.5, random_state=1, support_fraction=0.75)]
```

In [154]:

```
#A Lot of code has been cleaned prior to and after this point.
#I originally misunderstood how cross_val_score worked, and was fitting and transforming the whole training set with the imputer,
#only cross validating the SVC's results, meaning that the training sets had been fitted with
#before and after this point I did tonnes of iterating to try to find the best outlier removal
#with or without the standard scaler or robust scaler, etc. but it has been omitted. I have left
#so you can see the error i was making, and how it has been solved.

from sklearn.model_selection import cross_val_score

def best_outlier_params(diff_params, threshold):
    scores = []
    maxScore = 0;
    maxScoreIndex = 0;
    maxScoreOutliersRemoved = 0;

    index = 0

    imputer = SimpleImputer()
    imputer.fit(X_train)
    X_train_imp = imputer.transform(X_train)

    for permutation in diff_params:

        #here i was fitting each step on the whole pipeline, then only cross validating the SVC
        pipe = OutlierDetectionPipePart(strategy = permutation)
        pipe.fit(X_train_imp, y_train)
        X_train_imp_trans, y_train_trans = pipe.transform(X_train_imp, y_train)
        clf = SVC();
        scores = cross_val_score(clf, X_train_imp_trans, y_train_trans, cv=5)
        mean = scores.mean()

        #print("mean accuracy: %0.2f accuracy, standard deviation: %0.2f" % (mean, scores.std())
        #scores.append(mean)
        scores = np.append(scores, mean)

        outliersRemoved = X_train_imp.shape[0] - X_train_imp_trans.shape[0];
        proportion = outliersRemoved / X_train_imp.shape[0]
        if proportion < threshold:

            if (mean > maxScore):
                maxScore = mean
                maxScoreIndex = index
                maxScoreOutliersRemoved = X_train_imp.shape[0] - X_train_imp_trans.shape[0]
            index = index + 1

    print("\n Outlier removal with best cross val accuracy: ", diff_params[maxScoreIndex])
    print("Max accuracy score: ", maxScore)
    print("Index of max accuracy score: ", maxScoreIndex)
    print("Number of outliers removed: ", maxScoreOutliersRemoved)
    print("Size before: ", X_train_imp.shape[0])
    print("Size After: ", X_train_imp.shape[0] - maxScoreOutliersRemoved)
```

It was here I realised i was doing cross validation wrong. I had been editing the whole training set with the imputer, outlier removal, etc. and then only cross validating the svc - this meant that the validation sets within the cross validation had been used or the calculations, meaning the above work was invalid.

After a fair amount of research, i found that: Using imblearns pipelines and imblearns functionsampler, i can put the outlier detection into an imblearnpipeline, and then cross validate the whole pipeline.

I am also going to add scaling/normalizing into the pipeline, so that the distance based algorithms have more of a

In []:

```
#pip install -U imbalanced-Learn
```

In [37]:

```
from imblearn.pipeline import make_pipeline
from imblearn import FunctionSampler

def OutlierRemoverFuncSampler(X, y, strategy):
    #fit the outlier detector on X
    strategy.fit(X)
    #predict which rows are outliers
    preds = strategy.predict(X)
    #create a mask to delete the outliers
    mask = preds != -1
    #return X and Y with the outliers removed
    return (X[mask], y[mask])
```


In [38]:

```

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from imblearn.pipeline import make_pipeline
from imblearn import FunctionSampler

def best_outlier_params(diff_params, scaler):
    scores = []
    maxScore = 0;
    maxScoreIndex = 0;
    maxScoreOutliersRemoved = 0;
    ballAccOfMaxAcc = 0

    maxBallAcc = 0
    maxBallAccIndex = 0
    maxAccOfMaxBallAcc = 0

    index = 0

    #Loop through every permutation passed in
    for permutation in diff_params:

        #make an IMBLearn pipeline, using a functionsampler with my custom function passed in.
        clf = make_pipeline(SimpleImputer(),
                            scaler,
                            FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy": "SVC(random_state = 1)"}))

        #cross validate THE WHOLE pipeline
        cv = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state

        #get the accuracy and balanced accuracy scores
        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()

        scores = np.append(scores, [meanAcc, meanBallAcc])

        #if either score is the best, replace the best score with this one and save the permutation
        if (meanAcc > maxScore):
            maxScore = meanAcc
            maxScoreIndex = index
            ballAccOfMaxAcc = meanBallAcc

        if (meanBallAcc > maxBallAcc):
            maxBallAcc = meanBallAcc
            maxBallAccIndex = index
            accOfMaxBallAcc = meanAcc

    index = index + 1

    print("\n Outlier removal with best cross val accuracy: ", diff_params[maxScoreIndex])
    print("Max accuracy score: ", maxScore)
    print("Index of max accuracy score: ", maxScoreIndex)
    print("Balanced accuracy of system with most accuracy: ", ballAccOfMaxAcc)

    print("\n Outlier removal with best cross val balanced accuracy: ", diff_params[maxBallAccIndex])

```

```
print("Max balanced accuracy score: ", maxBallAcc)
print("index of max ball ac score: ", maxBallAccIndex)
print("Accuracy of system with most balanced accuracy: ", acc0fMaxBallAcc)
```

originally i was worried that because outlier removal was included in the pipeline, that the number of y values would be reduced. this is not that case. resampling is not applied to validation sets in imblearn pipelines, and the isolation forest has been built as a resampler /undersampler

In [252]:

```
best_outlier_params(diff_iso_forests, StandardScaler())
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (887). max_samples will be set to n_samples for estimation.
```

```
    warn(
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
```

```
    warn(
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
```

```
    warn(
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
```

```
    warn(
```

```
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (887). max_samples will be set to n_samples for estimation.
```

```
    warn(
```

```
)
```

```
Outlier removal with best cross val accuracy: IsolationForest(contamination=0.01, n_estimators=10, random_state=1)
```

```
Max accuracy score: 0.902515184868126
```

```
Index of max accuracy score: 20
```

```
Balanced accuracy of system with most accuracy: 0.7502980522788176
```

```
Outlier removal with best cross val balanced accuracy: IsolationForest(contamination=0.01, max_samples=128, n_estimators=10, random_state=1)
```

```
Max balanced accuracy score: 0.7544224862739414
```

```
index of max ball ac score: 24
```

```
Accuracy of system with most balanced accuracy: 0.9025111083934613
```

In [253]:

```
best_outlier_params(diff_lofs, StandardScaler())
```

```
Outlier removal with best cross val accuracy: LocalOutlierFactor(contamination=0.01, n_neighbors=5, novelty=True)
Max accuracy score: 0.9034120092943623
Index of max accuracy score: 4
Balanced accuracy of system with most accuracy: 0.7545300649508843
```

```
Outlier removal with best cross val balanced accuracy: LocalOutlierFactor(contamination=0.01, n_neighbors=5, novelty=True)
Max balanced accuracy score: 0.7545300649508843
index of max ball ac score: 4
Accuracy of system with most balanced accuracy: 0.9034120092943623
```

In [254]:

```
best_outlier_params(diff_onesclass, StandardScaler())
```

```
Outlier removal with best cross val accuracy: OneClassSVM(kernel='poly')
Max accuracy score: 0.8961803432391668
Index of max accuracy score: 1
Balanced accuracy of system with most accuracy: 0.763023507951752
```

```
Outlier removal with best cross val balanced accuracy: OneClassSVM(kernel='poly')
Max balanced accuracy score: 0.763023507951752
index of max ball ac score: 1
Accuracy of system with most balanced accuracy: 0.8961803432391668
```

In [255]:

```
best_outlier_params(diff_sgdoneclass, StandardScaler())
```

```
Outlier removal with best cross val accuracy: SGDOneClassSVM(eta0=1.0, learning_rate='adaptive', max_iter=250, nu=0.25,
random_state=1)
Max accuracy score: 0.907040071745954
Index of max accuracy score: 177
Balanced accuracy of system with most accuracy: 0.7590846626520336
```

```
Outlier removal with best cross val balanced accuracy: SGDOneClassSVM(eta0=2.0, learning_rate='constant', max_iter=250, nu=0.1,
random_state=1)
Max balanced accuracy score: 0.7714745093953287
index of max ball ac score: 256
Accuracy of system with most balanced accuracy: 0.9070156128979658
```

In [256]:

```
best_outlier_params(diff_envelopes, StandardScaler())
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: RuntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_det) (-144.286952378308371 > -151.865478378199498). You may want to try with a higher value of support_fraction (current value: 0.749).
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: RuntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_det) (-144.362957611511575 > -150.110198083479986). You may want to try with a higher value of support_fraction (current value: 0.749).
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: RuntimeWarning: Determinant has increased; this should not happen: log(det) > log(previous_det) (-87.455885385998329 > -216.551983661265268). You may want to try with a higher value of support_fraction (current value: 0.749).
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:184: UserWarning: The covariance matrix associated to your dataset is n
Outlier removal with best cross val accuracy: EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75)
Max accuracy score: 0.9061147119970648
Index of max accuracy score: 23
Balanced accuracy of system with most accuracy: 0.7751504556226003

Outlier removal with best cross val balanced accuracy: EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75)
Max balanced accuracy score: 0.7751504556226003
index of max ball ac score: 23
Accuracy of system with most balanced accuracy: 0.9061147119970648
```

(originally I was worried, because the custom pipeline was deleting records from the WHOLE train set (meaning that the validation set would have records deleted too) and I was getting very good accuracies for outliers removers with contamination set to 0.5 - they were deleting half the dataset)

Now I understand how cross validation works properly, I don't need to worry about contamination being high! the fact that the validation set is separated from the four other folds completely means that it won't have had anything done to it.

with the highest balanced accuracy of the set, and an accuracy equivalent to the highest accuracies of other outlier removers, EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75) seems to be the best outlier removal strategy.

But, before we select it completely, let's test it out without outlier removal.

In [219]:

```

clf = make_pipeline(SimpleImputer(),
                     StandardScaler(),
                     SVC(random_state = 1))

#could add a threshold to the funcsample - so that if the answer has more outliers removed than
cv = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state is none
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()

print("Mean balanced accuracy: ", meanBallAcc)
print("Mean accuracy: ", meanAcc)

```

Mean balanced accuracy: 0.7714745093953287
 Mean accuracy: 0.9070156128979658

The balanced accuracy and mean balanced accuracy of having no outlier removal vs EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75) is similar, but I am going to stick with the envelope because its balanced accuracy is slightly better.

current pipeline: SimpleImputer(), StandardScaler(), FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy":EllipticEnvelope(contamination=0.01, random_state=1, support_fraction=0.75)})

And finally, lets check with the robustScaler:

In [257]:

```

best_outlier_params(diff_iso_forests, RobustScaler())
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
  warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
  warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (886). max_samples will be set to n_samples for estimation.
  warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (887). max_samples will be set to n_samples for estimation.
  warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:307: UserWarning: max_samples (1024) is greater than the total number of samples (887). max_samples will be set to n_samples for estimation.
  warn(

```

```
Outlier removal with best cross val accuracy: IsolationForest(contamination=0.1, ma
x_samples=1024, n_estimators=200,
random_state=1)
Max accuracy score: 0.8619379560556031
Index of max accuracy score: 79
Balanced accuracy of system with most accuracy: 0.5633445779299876

Outlier removal with best cross val balanced accuracy: IsolationForest(contaminatio
n=0.1, max_samples=1024, n_estimators=200,
random_state=1)
Max balanced accuracy score: 0.5633445779299876
index of max ball ac score: 79
Accuracy of system with most balanced accuracy: 0.8619379560556031
```

In [258]:

```
best_outlier_params(diff_lofs, RobustScaler())
```

```
Outlier removal with best cross val accuracy: LocalOutlierFactor(novelty=True)
Max accuracy score: 0.8925971220088866
Index of max accuracy score: 2
Balanced accuracy of system with most accuracy: 0.7134172500367437

Outlier removal with best cross val balanced accuracy: LocalOutlierFactor(nove
lty=True)
Max balanced accuracy score: 0.7134172500367437
index of max ball ac score: 2
Accuracy of system with most balanced accuracy: 0.8925971220088866
```

In [259]:

```
best_outlier_params(diff_onesclass, RobustScaler())
```

```
Outlier removal with best cross val accuracy: OneClassSVM(gamma='auto')
Max accuracy score: 0.8266968325791856
Index of max accuracy score: 6
Balanced accuracy of system with most accuracy: 0.4527732351419315

Outlier removal with best cross val balanced accuracy: OneClassSVM(gamma='aut
o', kernel='poly')
Max balanced accuracy score: 0.48689549219522305
index of max ball ac score: 5
Accuracy of system with most balanced accuracy: 0.8195100077453018
```

In [260]:

```
best_outlier_params(diff_sgdoneclass, RobustScaler())
```

```
Outlier removal with best cross val accuracy: SGDOneClassSVM(eta0=2.0, fit_intercept=False, learning_rate='invscaling', max_iter=250, random_state=1)
Max accuracy score: 0.8447841506665036
Index of max accuracy score: 354
Balanced accuracy of system with most accuracy: 0.5339654794909429

Outlier removal with best cross val balanced accuracy: SGDOneClassSVM(eta0=0.5, fit_intercept=False, max_iter=250, nu=0.1, random_state=1)
Max balanced accuracy score: 0.5340286108040742
index of max ball ac score: 80
Accuracy of system with most balanced accuracy: 0.843879173290938
```

In [261]:

```
best_outlier_params(diff_envelopes, RobustScaler())
```

```
warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:745: UserWarning: The covariance matrix associated to your dataset is not full rank
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:745: UserWarning: The covariance matrix associated to your dataset is not full rank
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:745: UserWarning: The covariance matrix associated to your dataset is not full rank
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:745: UserWarning: The covariance matrix associated to your dataset is not full rank
    warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\covariance\_robust_covariance.py:745: UserWarning: The covariance matrix associated to your dataset is not full rank
```

```
Outlier removal with best cross val accuracy: EllipticEnvelope(contamination=0.05, random_state=1, support_fraction=0.75)
Max accuracy score: 0.8880885410297175
Index of max accuracy score: 27
Balanced accuracy of system with most accuracy: 0.6925839167034105
```

```
Outlier removal with best cross val balanced accuracy: EllipticEnvelope(assume_centered=True, contamination=0.05, random_state=1, support_fraction=0.1)
Max balanced accuracy score: 0.695984299845173
index of max ball ac score: 4
Accuracy of system with most balanced accuracy: 0.8871876401288166
```

With much worse balanced accuracy and slightly worse accuracy across the board, RobustScaler seems like a bad

Balancing & feature selection

Now its time to do balancing and feature selection

In [39]:

```
#now we need to find the best feature selection strategy.  
from sklearn.feature_selection import SelectKBest  
from sklearn.feature_selection import SelectPercentile  
from sklearn.feature_selection import GenericUnivariateSelect  
from sklearn.feature_selection import SelectFpr  
from sklearn.feature_selection import SelectFwe  
from sklearn.feature_selection import SelectFdr  
from sklearn.model_selection import ParameterGrid  
from sklearn.covariance import EllipticEnvelope  
from imblearn.over_sampling import SMOTE, ADASYN  
from sklearn.preprocessing import StandardScaler  
  
featSelection_params = [  
    (SelectKBest, {"k" : [5,10,15,20,22]}),  
    (SelectPercentile, {"percentile" : [5,10,15,20,50,75]}),  
    (GenericUnivariateSelect, {"mode" : ["percentile", "k_best", "fpr", "fdr", "fwe"]}),  
    (SelectFpr, {}),  
    (SelectFwe, {}),  
    (SelectFdr, {})  
]  
  
featSelectors = [ctor(**para) for ctor, paras in featSelection_params for para in ParameterGrid(paras)]  
print(featSelectors)  
  
params = dict(impute = [SimpleImputer()], scale = [StandardScaler()], outlier = [FunctionSampler(  
    outlier_function, outlier_kwarg)])
```

```
[SelectKBest(k=5), SelectKBest(), SelectKBest(k=15), SelectKBest(k=20), SelectKBest(k=22), SelectPercentile(percentile=5), SelectPercentile(), SelectPercentile(percentile=15), SelectPercentile(percentile=20), SelectPercentile(percentile=50), SelectPercentile(percentile=75), GenericUnivariateSelect(), GenericUnivariateSelect(mode='k_best'), GenericUnivariateSelect(mode='fpr'), GenericUnivariateSelect(mode='fdr'), GenericUnivariateSelect(mode='fwe'), SelectFpr(), SelectFwe(), SelectFdr()]
```

Out[39]:

```
{'impute': [SimpleImputer()],
'scale': [StandardScaler()],
'outlier': [FunctionSampler(func=<function OutlierRemoverFuncSampler at 0x000000
265B5EF1D30>,
kw_args={'strategy': EllipticEnvelope(contamination=0.01,
random_state=1,
support_fraction=0.7
In [40]:
```

#need to do feature selection, then resampling, and find the best combo.

```
"feat_selector": [SelectKBest(k=5),
SelectKBest(),
SelectKBest(k=15),
SelectKBest(k=20),
SelectKBest(k=22),
SelectPercentile(percentile=5),
SelectPercentile(),
SelectPercentile(percentile=15),
SelectPercentile(percentile=20),
SelectPercentile(percentile=50),
SelectPercentile(percentile=75),
GenericUnivariateSelect(),
GenericUnivariateSelect(mode='k_best'),
GenericUnivariateSelect(mode='fpr'),
GenericUnivariateSelect(mode='fdr'),
GenericUnivariateSelect(mode='fwe'),
SelectFpr(),
SelectFwe(),
SelectFdr()],""
'deleted[SMC(as)]it was causing errors
diff_feat_selectors = [SelectKBest(k=5),
SelectKBest(),
SelectKBest(k=15),
SelectKBest(k=20),
SelectKBest(k=22),
SelectPercentile(percentile=5),
SelectPercentile(),
SelectPercentile(percentile=15),
SelectPercentile(percentile=20),
SelectPercentile(percentile=50),
SelectPercentile(percentile=75),
GenericUnivariateSelect(),
GenericUnivariateSelect(mode='fpr'),
GenericUnivariateSelect(mode='fdr'),
GenericUnivariateSelect(mode='fwe'),
SelectFpr(),
SelectFwe(),
SelectFdr()]
```


In [41]:

```
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import RobustScaler
#from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline
from imblearn import FunctionSampler

def best_feat_SMOTE_params(diff_params, balancing_strategy, balance_first):
    scores = []
    maxScore = 0;
    maxScoreIndex = 0;
    maxScoreOutliersRemoved = 0;
    ballAccOfMaxAcc = 0

    maxBallAcc = 0
    maxBallAccIndex = 0
    maxAccOfMaxBallAcc = 0

    index = 0
    best_clf = None;

    index = 0

    for permutation in diff_params:

        #adjust the order of the pipeline depending on whether there is a balancing strategy,
        #the balancing startegy should be before feature selection
        if(balancing_strategy != None):
            if(balance_first == True):
                clf = make_pipeline(
                    SimpleImputer(),
                    StandardScaler(),
                    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
balancing_strategy,
permutation,
SVC()
)
            else:
                clf = make_pipeline(
                    SimpleImputer(),
                    StandardScaler(),
                    FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
permutation,
balancing_strategy,
SVC()
)
        else:
            clf = make_pipeline(
                SimpleImputer(),
                StandardScaler(),
                FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
permutation,
SVC()
)

        #cross validate the pipeline and get different scores
        cv = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state
        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()
#print("mean accuracy: %0.2f accuracy, standard deviation: %0.2f" % (mean, scores.std())
#scores.append(mean)
scores = np.append(scores, [meanAcc, meanBallAcc])

#save scores if they are the best
if (meanAcc > maxScore):
    maxScore = meanAcc
    maxScoreIndex = index
    ballAccOfMaxAcc = meanBallAcc

if (meanBallAcc > maxBallAcc):
    maxBallAcc = meanBallAcc
    maxBallAccIndex = index
    accOfMaxBallAcc = meanAcc

index = index + 1

print("\n Feature selection then balancing combo with best cross val accuracy: ", diff_params)
print("Max accuracy score: ", maxScore)
print("Index of max accuracy score: ", maxScoreIndex)
print("Balanced accuracy of system with most accuracy: ", ballAccOfMaxAcc)

print("\n Feature selection then balancing combo with best cross val balanced accuracy: ", diff_params)
print("Max balanced accuracy score: ", maxBallAcc)
print("index of max ball ac score: ", maxBallAccIndex)
print("Accuracy of system with most balanced accuracy: ", accOfMaxBallAcc)

```

In [42]:

```
best_feat_SMOTE_params(diff_feat_selectors, SMOTE(), True)
```

```

Feature selection then balancing combo with best cross val accuracy: SelectKBest(k=15)
Max accuracy score: 0.8916473034120094
Index of max accuracy score: 2
Balanced accuracy of system with most accuracy: 0.8477364939005362

```

```

Feature selection then balancing combo with best cross val balanced accuracy:
SelectKBest(k=15)
Max balanced accuracy score: 0.8477364939005362
index of max ball ac score: 2
Accuracy of system with most balanced accuracy: 0.8916473034120094

```

In [43]:

```
best_feat_SMOTE_params(diff_feat_selectors, ADASYN(), True)
```

Feature selection then balancing combo with best cross val accuracy: SelectKBest(k=15)
Max accuracy score: 0.8907260201377849
Index of max accuracy score: 2
Balanced accuracy of system with most accuracy: 0.8594279777040889

Feature selection then balancing combo with best cross val balanced accuracy:
SelectKBest(k=15)
Max balanced accuracy score: 0.8594279777040889
index of max ball ac score: 2
Accuracy of system with most balanced accuracy: 0.8907260201377849

In [44]:

```
best_feat_SMOTE_params(diff_feat_selectors, SMOTE(), False)
```

Feature selection then balancing combo with best cross val accuracy: GenericUnivariateSelect(mode='fwe')
Max accuracy score: 0.8898414251355428
Index of max accuracy score: 14
Balanced accuracy of system with most accuracy: 0.8428457763805083

Feature selection then balancing combo with best cross val balanced accuracy:
SelectKBest()
Max balanced accuracy score: 0.8682833039383266
index of max ball ac score: 1
Accuracy of system with most balanced accuracy: 0.8789939260527497

In [45]:

```
best_feat_SMOTE_params(diff_feat_selectors, ADASYN(), False)
```

Feature selection then balancing combo with best cross val accuracy: SelectFpr()
Max accuracy score: 0.8853369206310383
Index of max accuracy score: 15
Balanced accuracy of system with most accuracy: 0.8346479243300052

Feature selection then balancing combo with best cross val balanced accuracy:
SelectKBest()
Max balanced accuracy score: 0.8456761228993142
index of max ball ac score: 1
Accuracy of system with most balanced accuracy: 0.8591455709102768

Although accuracy has gone slightly down, the balancing has clearly made the estimator more effective for predicting the minority target classes, as the balanced accuracy has gone up a lot.

SelectKBest() followed by SMOTE() provides the best balanced accuracy, with a balanced accuracy of 0.866 (3d.p), and an accuracy of 0.883 (3.d.p) (diff 0.008)

SMOTE() followed by GenericUnivariateSelect(mode='fpr') provides the best accuracy, with an accuracy of 0.891 (3.dp), and a balanced accuracy of 0.815 (3 d.p)

SelectKBest followed by SMOTE() will be the next steps in the pipeline. Its balanced accuracy is the highest of the selection, and its accuracy is only 0.008 below the combination with the highest accuracy. Although the accuracy score is 0.023 below the highest accuracy score at the previous step, its balanced accuracy is 0.091 above the previous step's balanced accuracy - the feature selection and balancing has made the pipeline better at correctly classifying minority classes.

In [46]:

```

def best_params_balance_only(balancing_strategies):
    scores = []
    maxScore = 0;
    maxScoreIndex = 0;
    maxScoreOutliersRemoved = 0;
    ballAccOfMaxAcc = 0

    maxBallAcc = 0
    maxBallAccIndex = 0
    meanAccOfMaxBallAcc = 0

    index = 0
    best_clf = None;
    #maxScoreOutliersRemoved = 0;

    index = 0

    for balancing_strategy in balancing_strategies:

        clf = make_pipeline(
            SimpleImputer(),
            StandardScaler(),
            FunctionSampler(func=OutlierRemoverFuncSampler, kw_args={"strategy" : LocalOutlierF
            balancing_strategy,
            SVC()
        )

        cv = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state
        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()

        scores = np.append(scores, [meanAcc, meanBallAcc])

        if (meanAcc > maxScore):
            maxScore = meanAcc
            maxScoreIndex = index
            ballAccOfMaxAcc = meanBallAcc

        if (meanBallAcc > maxBallAcc):
            maxBallAcc = meanBallAcc
            maxBallAccIndex = index
            accOfMaxBallAcc = meanAcc

        index = index + 1

    print("\n Balancing strategy with best cross val accuracy: ", balancing_strategies[maxScoreIndex])
    print("Max accuracy score: ", maxScore)
    print("Index of max accuracy score: ", maxScoreIndex)
    print("Balanced accuracy of strategy with most accuracy: ", ballAccOfMaxAcc)

    print("\n Balancing strategy with best cross val balanced accuracy: ", balancing_strategies[maxBallAccIndex])
    print("Max balanced accuracy score: ", maxBallAcc)
    print("index of max ball ac score: ", maxBallAccIndex)

```

```
In [47]: print("Accuracy of strategy with most balanced accuracy: ", accOfMaxBallAcc)
```

```
best_params_balance_only([SMOTE(), ADASYN()])
```

Balancing strategy with best cross val accuracy: ADASYN()
 Max accuracy score: 0.8826382944030003
 Index of max accuracy score: 1
 Balanced accuracy of strategy with most accuracy: 0.8177479272493724

Balancing strategy with best cross val balanced accuracy: ADASYN()
 Max balanced accuracy score: 0.8177479272493724
 index of max ball ac score: 1
 Accuracy of strategy with most balanced accuracy: 0.8826382944030003

In [48]:

```
best_feat_SMOTE_params(diff_feat_selectors, None, False)
```

Feature selection then balancing combo with best cross val accuracy: SelectKBest(k=15)
 Max accuracy score: 0.9052138110961641
 Index of max accuracy score: 2
 Balanced accuracy of system with most accuracy: 0.7480423051523809

Feature selection then balancing combo with best cross val balanced accuracy:
 SelectPercentile(percentile=50)
 Max balanced accuracy score: 0.758775896799367
 index of max ball ac score: 9
 Accuracy of system with most balanced accuracy: 0.9043210631445925

Balancing without feature selection leaves accuracy the same as the previous iteration, but with worse balanced accuracy.

Feature selection without balancing provides better accuracy, but worse balanced accuracy.

Although the accuracy score of feature selection without balancing is much better, the balanced accuracy is much worse - the train dataset did not have an even proportion of target classes, so balanced accuracy is important to ensure that the less likely cases will be classified correctly more often.

Because we are working out whether patients are high risk, medium risk, or low risk, it is important not to misclassify patients who are truly high risk incorrectly, as this could have dangerous impacts for their health. Sensitivity / recall is "the proportion of people that tested positive and are positive of all the people who are actually positive". We want to get the sensitivity / recall of the high risk patients - where positive is them being classified as high risk, and negative is them being classified as low or medium risk. Higher sensitivity for high risk patients means that fewer truly high risk patients will remain undetected.

In [51]:

```
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import RobustScaler
# from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline
from imblearn import FunctionSampler
from sklearn.metrics import recall_score
from sklearn.metrics import make_scorer

def best_feat_SMOTE_params(diff_params, balancing_strategy, balance_first):
    scores = []
    maxScore = 0;
    maxScoreIndex = 0;
    maxScoreOutliersRemoved = 0;
    ballAccOfMaxAcc = 0
    recallOfMaxAcc = None;

    maxBallAcc = 0
    maxBallAccIndex = 0
    maxAccOfMaxBallAcc = 0
    recallOfMaxBallAcc = None;

    best_hr_recall = 0
    best_hr_recall_index = 0
    acc_best_hr_recall = 0
    ball_acc_best_hr_recall = 0
    all_recalls_best_hr_recall = None;

    index = 0
    best_clf = None;
    #maxScoreOutliersRemoved = 0;

    index = 0

#Loop through permutations
for permutation in diff_params:

    #adjust the order of the pipeline depending on whether there is a balancing strategy,
    #the balancing startegy should be before feature seelction
    if(balancing_strategy != None):
        if(balance_first == True):
            clf = make_pipeline(
                SimpleImputer(),
                StandardScaler(),
                FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
balancing_strategy,
permutation,
SVC()
)
        else:
            clf = make_pipeline(
                SimpleImputer(),
                StandardScaler(),
                FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
permutation,
balancing_strategy,
SVC()
)
    else:
        clf = make_pipeline(
            SimpleImputer(),
```

```

        StandardScaler(),
        FunctionSampler(func=OutlierRemoverFuncSampler,kw_args={"strategy" : Local
permutation,
SVC()
)

#cross validate and get scores
cv = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)#random state
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_accu
meanBallAcc = scoresBallAcc.mean()

recall_low_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scoring
recall_medium_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scoring
recall_high_risk = cross_val_score(clf, X_train, y_train, cv=cv, scoring = make_scoring

recall = {
    "Low risk" : recall_low_risk,
    "Medium risk" : recall_medium_risk,
    "High risk" : recall_high_risk
}
#print("mean accuracy: %0.2f accuracy, standard deviation: %0.2f" % (mean, scores.std(
#scores.append(mean)
scores = np.append(scores, [meanAcc, meanBallAcc])

#save scores if one is the best
if (meanAcc > maxScore):
    maxScore = meanAcc
    maxScoreIndex = index
    ballAccOfMaxAcc = meanBallAcc
    recallOfMaxAcc = recall
    #maxScoreOutliersRemoved = X_train_imp.shape[0] - X_train_imp_trans.shape[0]

if (meanBallAcc > maxBallAcc):
    maxBallAcc = meanBallAcc
    maxBallAccIndex = index
    accOfMaxBallAcc = meanAcc
    recallOfMaxBallAcc = recall

if (recall_high_risk > best_hr_recall):
    best_hr_recall = recall_high_risk
    best_hr_recall_index = index
    acc_best_hr_recall = meanAcc
    ball_acc_best_hr_recall = meanBallAcc
    all_recalls_best_hr_recall = recall

index = index + 1

print("\n Feature selection then balancing combo with best cross val accuracy: ", diff_par
print("Max accuracy score: ", maxScore)
print("Index of max accuracy score: ", maxScoreIndex)
print("Balanced accuracy of system with most accuracy: ", ballAccOfMaxAcc)
print("Recall: ", recallOfMaxAcc)

print("\n Feature selection then balancing combo with best cross val balanced accuracy: ", print("Max balanced accuracy score: ", maxBallAcc)
print("index of max ball ac score: ", maxBallAccIndex)
print("Accuracy of system with most balanced accuracy: ", accOfMaxBallAcc)

```

```

print("Recall: ", recallOfMaxBallAcc)

print("\nMethod with best high risk recall: ", diff_params[best_hr_recall_index])
print("High risk recall: ", best_hr_recall)
print("index of system with best high risk recall: ", best_hr_recall_index)
print("Accuracy: ", acc_best_hr_recall)
print("balanced accuracy: ", ball_acc_best_hr_recall)
In [92]: print("All recalls: ", all_recalls_best_hr_recall)

```

```
best_feat_SMOTE_params(diff_feat_selectors, SMOTE(), True)
```

1396: UserWarning: Note that pos_label (set to 0) is ignored when average != 'binary' (got None). You may use labels=[pos_label] to specify a single positive class.

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

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

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

```
warnings.warn(
C:\Users\Will\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
```

Feature selection then balancing combo with best cross val accuracy: SelectPercentile(percentile=75)

Max accuracy score: 0.8907464025111084

Index of max accuracy score: 10

Balanced accuracy of system with most accuracy: 0.8394920552948257

Recall: {'Low risk': 0.903322038402764, 'Medium risk': 0.8136363636363637, 'High risk': 0.777}

Feature selection then balancing combo with best cross val balanced accuracy: SelectPercentile(percentile=50)

Max balanced accuracy score: 0.862956402776016

index of max ball ac score: 9

Accuracy of system with most balanced accuracy: 0.884440096204802

Recall: {'Low risk': 0.8941266361039133, 'Medium risk': 0.8321969696969698, 'High risk': 0.7566666666666666}

Method with best high risk recall: SelectPercentile(percentile=15)

High risk recall: 0.8358333333333334

index of system with best high risk recall: 7

Accuracy: 0.8050262932615875

balanced accuracy: 0.7950214321810893

All recalls: {'Low risk': 0.8193741279649194, 'Medium risk': 0.7509469696969697, 'High risk': 0.8358333333333334}

In [53]:

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

```
Feature selection then balancing combo with best cross val accuracy: SelectKBest(k=15)
Max accuracy score: 0.8880192409604174
Index of max accuracy score: 2
Balanced accuracy of system with most accuracy: 0.8611890858958329
Recall: {'Low risk': 0.9067503820344163, 'Medium risk': 0.8077651515151516, 'High risk': 0.8600000000000001}
```

```
Feature selection then balancing combo with best cross val balanced accuracy: SelectKBest(k=15)
Max balanced accuracy score: 0.8611890858958329
index of max balanced accuracy score: 2
Accuracy of system with most balanced accuracy: 0.8880192409604174
Recall: {'Low risk': 0.9067503820344163, 'Medium risk': 0.8077651515151516, 'High risk': 0.8600000000000001}
```

```
Method with best high risk recall: SelectKBest(k=5)
High risk recall: 0.8841666666666667
index of system with best high risk recall: 0
Accuracy: 0.7941543353308059
balanced accuracy: 0.8074929431850899
All recalls: {'Low risk': 0.7870905587668593, 'Medium risk': 0.7392045454545455, 'High risk': 0.8841666666666667}
```

In [54]:

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

Feature selection then balancing combo with best cross val accuracy: SelectFpr()

Max accuracy score: 0.8916554563613387

Index of max accuracy score: 15

Balanced accuracy of system with most accuracy: 0.8294215978803383

Recall: {'Low risk': 0.9171284300046508, 'Medium risk': 0.8011363636363636, 'High risk': 0.77}

Feature selection then balancing combo with best cross val balanced accuracy: SelectKBest()

Max balanced accuracy score: 0.8608185326455707

index of max balanced score: 1

Accuracy of system with most balanced accuracy: 0.8862215156332803

Recall: {'Low risk': 0.8883595774367151, 'Medium risk': 0.8507575757575758, 'High risk': 0.8708333333333333}

Method with best high risk recall: SelectKBest(k=5)

High risk recall: 0.8841666666666667

index of system with best high risk recall: 0

Accuracy: 0.8410827116709468

Balanced accuracy: 0.8351033606948496

All recalls: {'Low risk': 0.852687529067836, 'Medium risk': 0.7827651515151516, 'High risk': 0.8841666666666667}

In [55]:

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

Feature selection then balancing combo with best cross val accuracy: SelectFpr()

Max accuracy score: 0.8862459744812685

Index of max accuracy score: 15

Balanced accuracy of system with most accuracy: 0.8398094156635117

Recall: {'Low risk': 0.9102119460500964, 'Medium risk': 0.7517045454545455, 'High risk': 0.8091666666666667}

Feature selection then balancing combo with best cross val balanced accuracy: SelectKBest()

Max balanced accuracy score: 0.8556020691265858

index of max balanced score: 1

Accuracy of system with most balanced accuracy: 0.86184012066365

Recall: {'Low risk': 0.8733971164706663, 'Medium risk': 0.7763257575757576, 'High risk': 0.8983333333333334}

Method with best high risk recall: SelectPercentile(percentile=20)

High risk recall: 0.9233333333333335

index of system with best high risk recall: 8

Accuracy: 0.7950470832823774

Balanced accuracy: 0.8349118854480626

All recalls: {'Low risk': 0.7813500764068833, 'Medium risk': 0.7952651515151515, 'High risk': 0.9233333333333335}

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 out whether 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.