Mortalty Metric Extraction

Benjamin Frost 2022

In [ ]:
```
import pandas as pd
import numpy as np
import torch.multiprocessing as mp
from sklearn.preprocessing import KBinsDiscretizer, OneHotEncoder, MinMaxScaler
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import PolynomialFeatures
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor, as_completed
from scipy.interpolate import interp1d
from Categorization import Categorizer
import torch
import copy
from torch.nn.functional import one_hot
import imblearn
from collections import Counter
from tslearn.clustering import TimeSeriesKMeans
from tslearn.utils import to_time_series_dataset
from tslearn.preprocessing import TimeSeriesScalerMeanVariance
from sklearn.metrics import silhouette_score
from tsfresh import extract_features, select_features, extract_relevant_features
from tsfresh.utilities.dataframe_functions import impute
from dask.dataframe import from_pandas
from tsfresh.utilities.distribution import MultiprocessingDistributor
import hashlib
from sklearn.metrics import precision_recall_fscore_support
from importlib import reload
from temporalHelper import TemporalHelper as TH
```

Loading in the time-series dataset

In [ ]:
```
th = TH()

mimicDF = th.get_mimic()

mimicDF
```

Out[ ]:

	PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial PaCO2	...	SVI	SVR	SVRI	SaO2	Sodi
0	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
1	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
2	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
3	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
4	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
47083	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
47084	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
47085	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
47086	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N
47087	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	N

47088 rows × 44 columns

In [ ]:
```
mimicDF.describe()
```

Out[ ]:

	PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	A

	PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	A
count	47088.000000	47088.000000	484.000000	481.000000	705.000000	245.000000	31415.000000	31503.000000	31504.000000	8601.(
mean	151079.910805	0.039755	630.123967	954.301455	67.382553	2.721633	77.058157	57.226709	117.544502	40.5
std	29378.613191	0.195386	1245.805613	2384.326867	5.448469	0.584693	14.324177	11.197415	21.311674	7.(
min	100059.000000	0.000000	3.000000	3.000000	48.000000	1.400000	0.000000	0.000000	0.000000	15.(
25%	126241.000000	0.000000	25.000000	49.000000	64.000000	2.400000	68.000000	50.000000	103.000000	36.(
50%	151857.000000	0.000000	97.000000	137.000000	68.000000	2.700000	75.000000	56.000000	115.000000	40.(
75%	176484.000000	0.000000	554.750000	797.000000	70.000000	3.000000	84.000000	63.000000	129.000000	44.(
max	199998.000000	1.000000	8100.000000	23060.000000	160.000000	4.400000	287.000000	191.000000	255.000000	91.(

8 rows × 44 columns

In [ ]:
```
print(f"There are {mimicDF['PatientID'].nunique()} unique patients in the dataset")
```

```
There are 1126 unique patients in the dataset
```

Defining the hashing function

In [ ]:
```
def find_cached(df=None, my_hash=None):

    if my_hash is None:
        my_hash = hashlib.sha256(bytes(str(df), 'utf-8')).hexdigest()

    display(my_hash)


    try:
        cachedDF = pd.read_csv("./processingCache/" + my_hash + ".csv").set_index("Unnamed: 0")

        print("Using cached df")

        return cachedDF, my_hash

    except:

        return False, my_hash
```

In [ ]:
```
# Fixing 'arterial pH', 'ionized calcium' since they contain erroneous values.

mimicDF['Arterial pH'][mimicDF['Arterial pH'] > 8] = mimicDF['Arterial pH'].mean()
mimicDF['Ionized Calcium'][mimicDF['Ionized Calcium'] > 8] = mimicDF['Ionized Calcium'].mean()
```

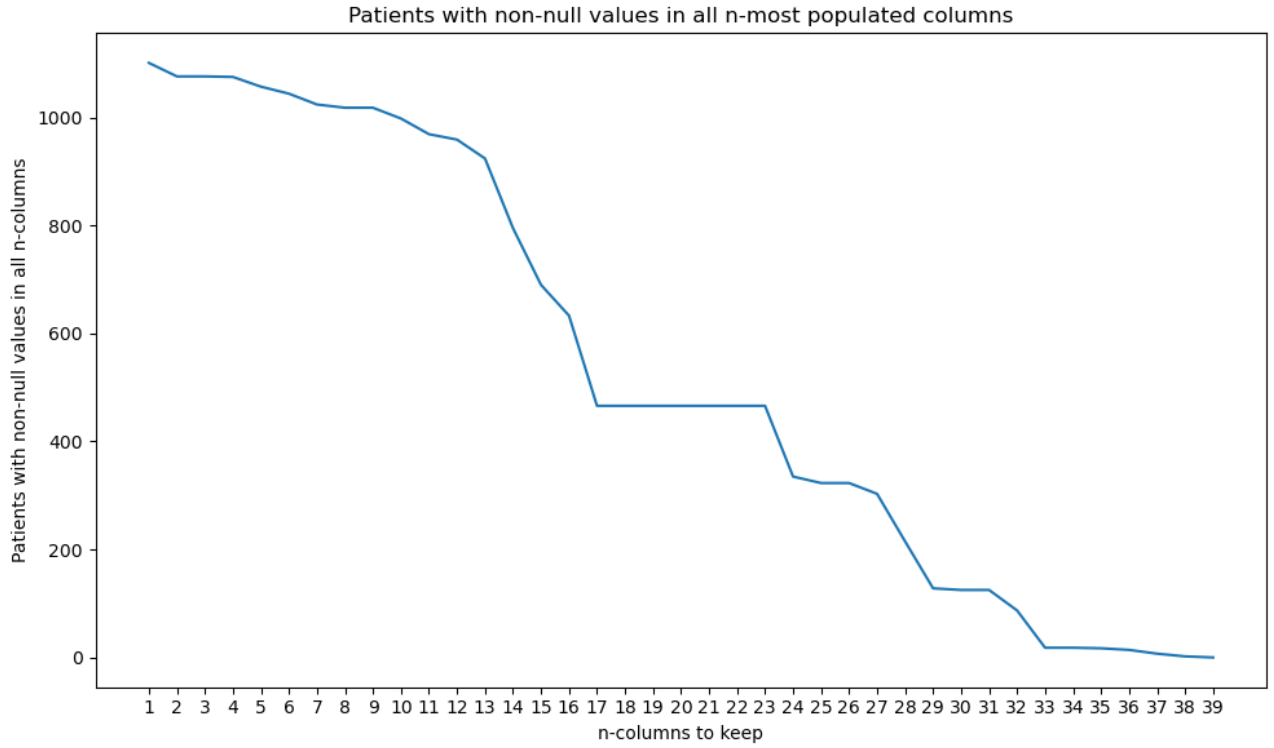
```
C:\Users\benma\AppData\Local\Temp/ipykernel_14644/4144374585.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  mimicDF['Arterial pH'][mimicDF['Arterial pH'] > 8] = mimicDF['Arterial pH'].mean()
C:\Users\benma\AppData\Local\Temp/ipykernel_14644/4144374585.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  mimicDF['Ionized Calcium'][mimicDF['Ionized Calcium'] > 8] = mimicDF['Ionized Calcium'].mean()
```

Calculating the 'missingness' of columns

In [ ]:
```
patients = th.get_patients(mimicDF)

patientsKept, columnsExplored = th.count_null(patients)

patients = th.get_top_columns(patients, 12)
```

```
100%|██████████| 1126/1126 [00:01<00:00, 869.42it/s]
[1/4] Counting null values...
100%|██████████| 1126/1126 [00:00<00:00, 1744.45it/s]
[2/3] Copying dataset...
[3/4] Dropping null columns...
```

```
100%|██████████| 39/39 [00:03<00:00, 12.05it/s]
[4/4] Graphing...
```




```
100%|██████████| 1126/1126 [00:00<00:00, 2428.63it/s]
```

In [ ]:
```
patients[0].topColumns
```

Out[ ]:

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	NaN	7.33	NaN	53.0	437.0	NaN	1.19	NaN
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	NaN	NaN	NaN	NaN	NaN	7.45	NaN	39.0	288.0	NaN	NaN	NaN
19	NaN	NaN	NaN	NaN	NaN	7.39	NaN	40.0	431.0	NaN	NaN	NaN
20	129.0	NaN	NaN	NaN	NaN	7.34	NaN	46.0	283.0	NaN	1.12	NaN
21	107.0	43.0	108.0	61.0	12.0	7.36	NaN	42.0	265.0	890.755981	1.27	1819.739990
22	NaN	47.0	136.0	72.0	11.0	7.34	NaN	41.0	108.0	1060.040039	NaN	2160.340088
23	NaN	51.0	123.0	69.0	10.0	NaN	NaN	NaN	NaN	1209.760010	NaN	2467.659912
24	NaN	51.0	114.0	67.0	8.0	7.29	NaN	42.0	114.0	1151.219971	0.94	2348.260010

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
25	NaN	61.0	132.0	77.0	10.0	7.34	NaN	45.0	150.0	1001.929993	1.15	2078.739990
26	150.0	54.0	125.0	102.0	10.0	7.32	NaN	50.0	83.0	NaN	1.29	NaN
27	NaN	53.0	120.0	73.0	9.0	7.34	NaN	41.0	95.0	NaN	1.05	NaN
28	NaN	54.0	133.0	77.0	15.0	7.32	NaN	45.0	97.0	997.987976	NaN	2032.790039
29	NaN	53.0	128.0	72.0	12.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	149.0	47.0	118.0	66.0	9.0	7.32	11.4	46.0	94.0	NaN	1.15	NaN
31	NaN	50.0	128.0	72.0	10.0	NaN	NaN	NaN	NaN	934.086975	NaN	1907.689941
32	NaN	61.0	150.0	86.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
33	NaN	49.0	132.0	75.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
34	NaN	58.0	147.0	83.0	18.0	7.33	NaN	43.0	120.0	NaN	1.16	NaN
35	NaN	44.0	110.0	60.0	15.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
36	NaN	46.0	107.0	62.0	15.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
37	NaN	52.0	121.0	71.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
38	NaN	45.0	113.0	64.0	16.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
39	162.0	44.0	109.0	60.0	16.0	7.31	11.7	45.0	98.0	739.495972	1.22	1510.729980
40	NaN	47.0	113.0	64.0	NaN	7.29	NaN	NaN	NaN	NaN	1.17	NaN
41	NaN	44.0	106.0	62.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42	NaN	61.0	133.0	80.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
43	NaN	62.0	133.0	81.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	NaN	64.0	135.0	84.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	NaN	60.0	139.0	81.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	NaN	66.0	145.0	92.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	NaN	52.0	132.0	75.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [ ]:

```
# tsfresh can't handle missing values. So we'll interpolate them.

for patient in tqdm(patients):
    patient.cleanedDF = copy.copy(patient.topColumns)
    for col in patient.cleanedDF:
        nonNullCount = patient.cleanedDF[col].count()

        if (nonNullCount >= 3):
            # Polynomial if there are >=3 non-null values
            patient.cleanedDF[col] = patient.cleanedDF[col].interpolate(method='polynomial', order=2)
        elif nonNullCount == 1:
            # Pad with three values either side if only 1 data point exists.
            patient.cleanedDF[col] = patient.cleanedDF[col].interpolate(method='linear', limit_direction='both', limit
        else:
            # Otherwise, we can linearly interpolate.
            patient.cleanedDF[col] = patient.cleanedDF[col].interpolate(method='linear', limit_direction='both', limit

        # Fill in the start and end with the closest data point.
        patient.cleanedDF[col] = patient.cleanedDF[col].interpolate(method='linear', limit_direction='both', limit_are

    patient.cleanedDF["PatientID"] = patient.patientID


formattedMimicDF = pd.concat([patient.cleanedDF for patient in patients])


display(formattedMimicDF)
```

```
100%|██████████| 959/959 [00:13<00:00, 68.92it/s]
```

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI	PatientID
0	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
1	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
2	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
3	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI	PatientID
4	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
...	...	...	...	...	...	...	...	...	...	...	...	...	...
43	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
44	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
45	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
46	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
47	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998

41376 rows × 13 columns

Display interpolation and fillna results here

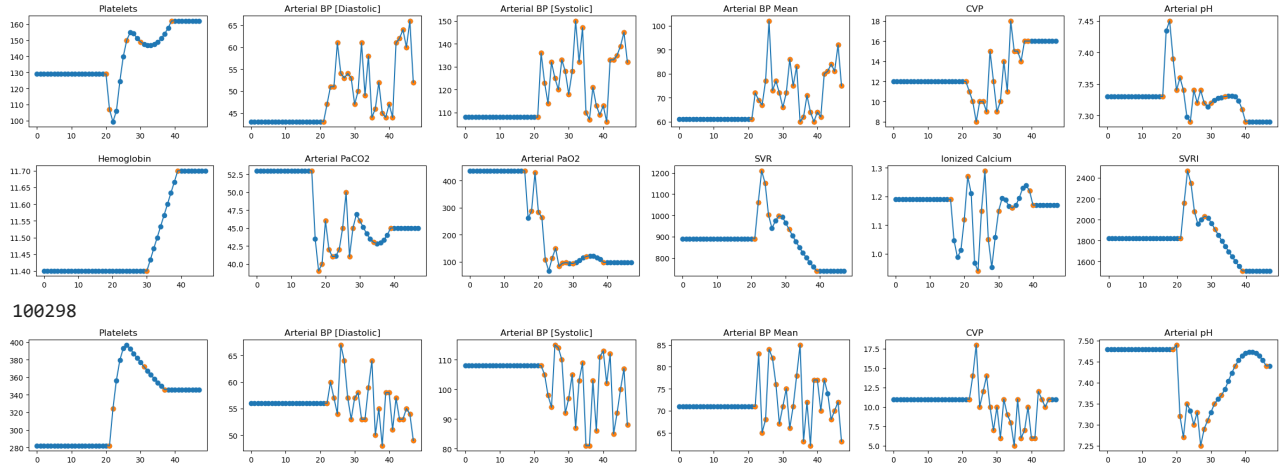
In [ ]:
```
formattedMimicDF.columns
```

Out[ ]:
```
Index(['Platelets', 'Arterial BP [Diastolic]', 'Arterial BP [Systolic]',
       'Arterial BP Mean', 'CVP', 'Arterial pH', 'Hemoglobin',
       'Arterial PaCO2', 'Arterial PaO2', 'SVR', 'Ionized Calcium', 'SVRI',
       'PatientID'],
      dtype='object')
```

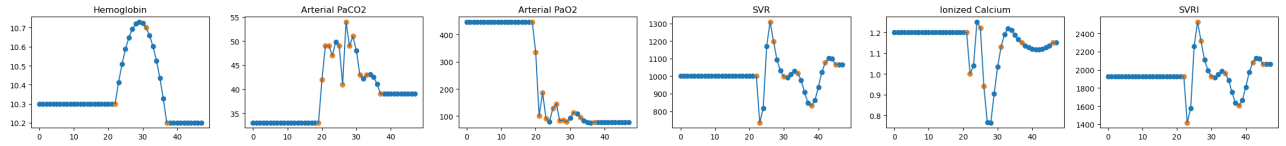
Visualise the interpolation

In [ ]:
```
for patient in patients[:2]:


    print(patient.patientID)


    step = 6

    columns = list(patient.cleanedDF.columns)[:-1]

    for idx in range(0, len(columns), step):


        fig = plt.figure(figsize = (30, 6), dpi=100)

        cols = columns[idx:idx+step]


        for k, col in enumerate(cols):

            plt.subplot(2, step, k+1)

            plt.plot(patient.cleanedDF.index, patient.cleanedDF[col])
            plt.scatter(patient.cleanedDF.index, patient.cleanedDF[col])
            plt.scatter(patient.cleanedDF.index, patient.data[col])
            plt.title(f"{col}")

    plt.show()
```

100059



100298



In [ ]:
```
targetSeries = pd.Series(data=[patient.label for patient in patients], index=[patient.patientID for patient in patient

targetSeries
```

Out[ ]:
```
100059    0
100298    0
100321    0
100336    0
100392    0
         ..
199876    0
199877    0
199963    0
199993    0
199998    0
Length: 959, dtype: int64
```

In [ ]:
```
formattedMimicDF
```

Out[ ]:

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI	PatientID
0	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
1	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
2	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
3	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
4	129.0	43.0	108.0	61.0	12.0	7.33	11.4	53.0	437.0	890.755981	1.19	1819.739990	100059
...	...	...	...	...	...	...	...	...	...	...	...	...	...
43	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
44	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
45	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
46	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998
47	144.0	45.0	124.0	68.0	13.0	7.39	9.3	38.0	109.0	695.081970	1.18	1376.619995	199998

41376 rows × 13 columns

Caching to get he extracted features since processing can take a while

In [ ]:
```
## Checking if this DF already has cached time series features. Will save approx 15 mins if so.

# my_hash = hashlib.sha256(pd.util.hash_pandas_object(formattedMimicDF, index=True).values).hexdigest()

my_hash = "26128f35758669a6e2001768a98df93bff7b0b8ac8fdf241d94641c49a83b901"

filtered_features, my_hash = find_cached(formattedMimicDF, my_hash = my_hash)


if filtered_features is False:

    filtered_features = extract_relevant_features(formattedMimicDF, y=targetSeries, column_id="PatientID", n_jobs=4)

    filtered_features.to_csv("./processingCache/" + my_hash + ".csv")
```

```
'26128f35758669a6e2001768a98df93bff7b0b8ac8fdf241d94641c49a83b901'
Using cached df
```

Adding back the patient ID column, visualising the data

In [ ]:
```
filtered_features = filtered_features.reindex(formattedMimicDF['PatientID'].unique())

filtered_features['PatientID'] = formattedMimicDF['PatientID'].unique()

filtered_features = filtered_features.set_index('PatientID')
```

```
display(filtered_features)
```

PatientID	CVP__quantile__q_0.3	CVP__quantile__q_0.4	CVP__quantile__q_0.2	CVP__minimum	CVP__quantile__q_0.1	CVP__median	CVP__mean
100059	12.0	12.0	12.0	8.0	10.0	12.000000	12.812500
100298	11.0	11.0	10.0	5.0	6.7	11.000000	10.395833
100321	8.0	8.0	8.0	2.0	8.0	9.000000	10.291667
100336	18.0	18.0	18.0	9.0	18.0	21.226301	19.642885
100392	8.0	8.0	8.0	8.0	8.0	10.000000	11.404526
...	...	...	...	...	...	...	...
199876	8.0	8.0	8.0	8.0	8.0	8.000000	9.833333
199877	12.0	12.0	10.0	7.0	9.7	13.000000	12.000000
199963	12.1	14.0	10.4	8.0	9.0	14.000000	13.611312
199993	13.0	13.0	13.0	12.0	13.0	13.000000	14.400796
199998	11.0	11.0	11.0	6.0	9.0	11.000000	11.437500

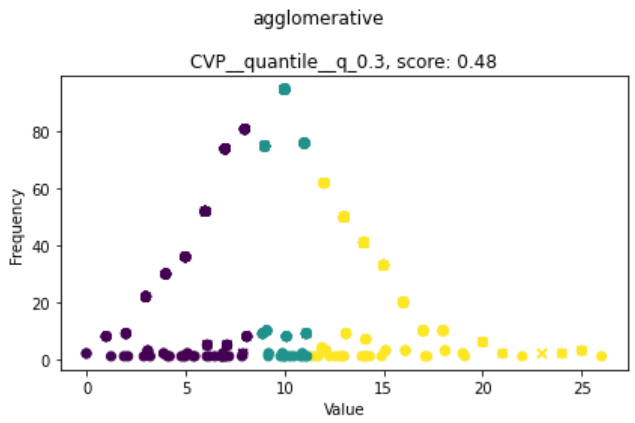
959 rows × 14 columns

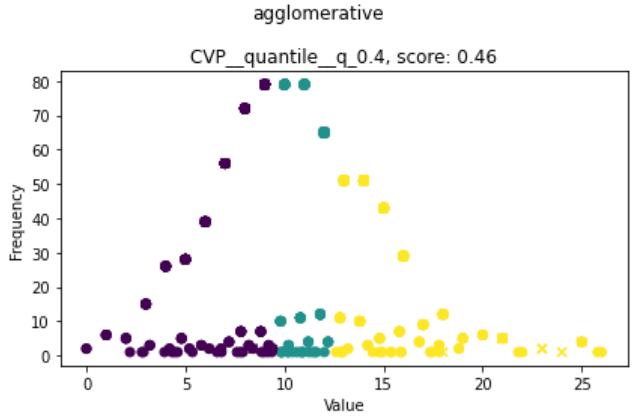
In [ ]:
```
# These are the columns deemed most impactful on the prediction.

for col in filtered_features.columns:
    print(col)
```

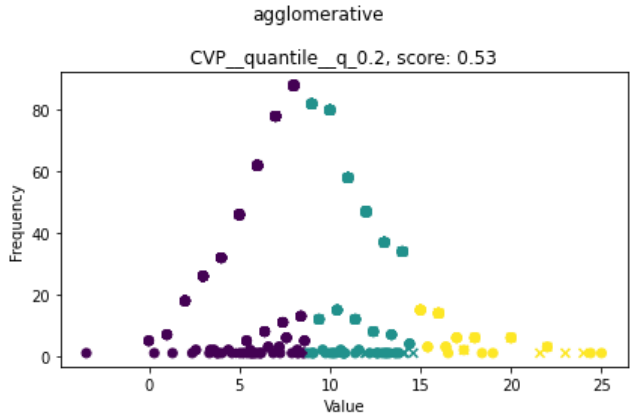
```
CVP__quantile__q_0.3
CVP__quantile__q_0.4
CVP__quantile__q_0.2
CVP__minimum
CVP__quantile__q_0.1
CVP__median
CVP__mean
CVP__root_mean_square
CVP__quantile__q_0.6
CVP__c3__lag_3
CVP__c3__lag_2
CVP__c3__lag_1
CVP__variation_coefficient
CVP__quantile__q_0.7
```

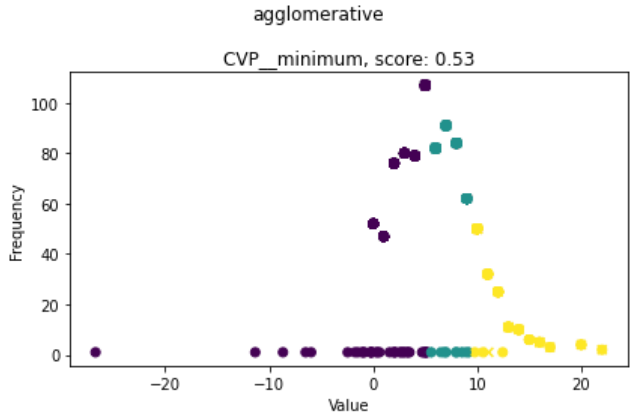
Agglomerative clustering and labelling as seen in static dataset

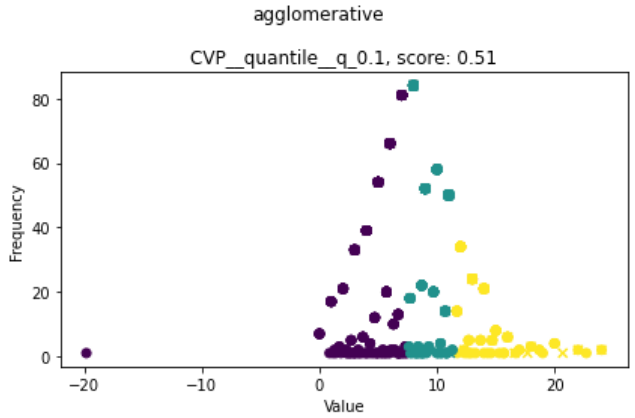
In [ ]:
```
cat = Categorizer(filtered_features)

cat.agglomerative(n_clusters=3)

cat.display(target=targetSeries)

categories = {0: 'very_low', 1: 'low', 2: 'medium', 3: 'high', 4: 'very_high'}

mapped = cat.map_types(mapping=categories)['agglomerative']
```



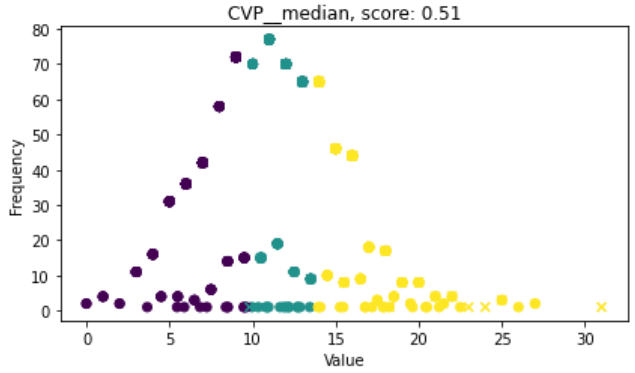

agglomerative
CVP__quantile__q_0.4, score: 0.46
Frequency
Value


agglomerative
CVP__quantile__q_0.2, score: 0.53
Frequency
Value

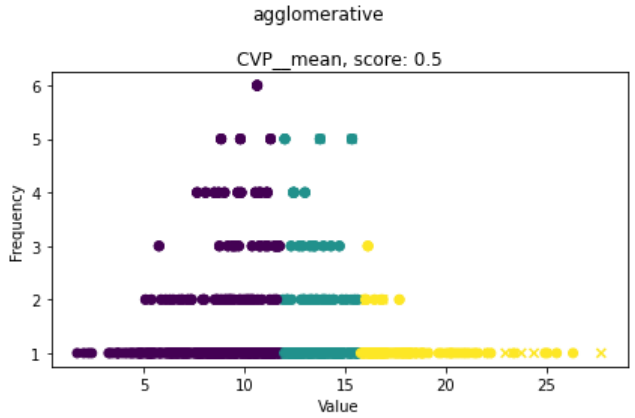

agglomerative
CVP__minimum, score: 0.53
Frequency
Value


agglomerative
CVP__quantile__q_0.1, score: 0.51
Frequency
Value

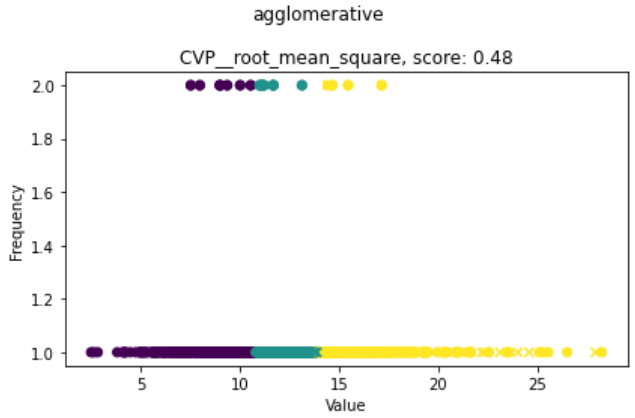
agglomerative



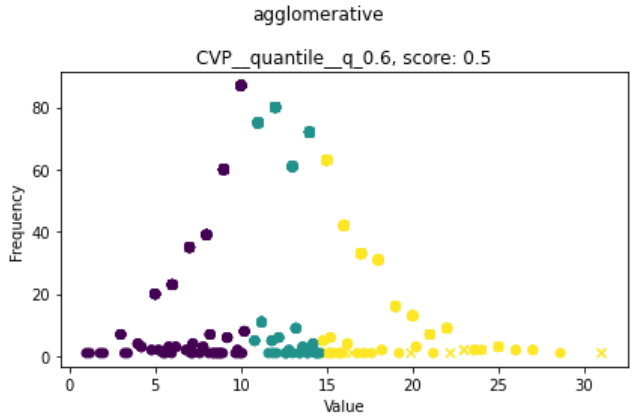
agglomerative



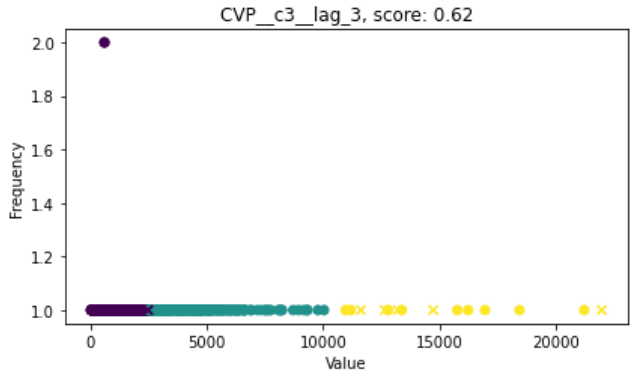
agglomerative



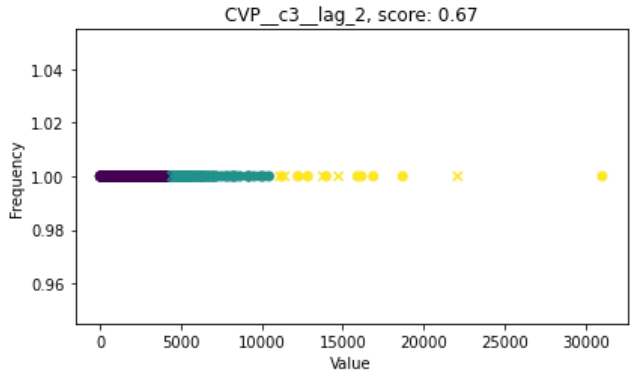
agglomerative



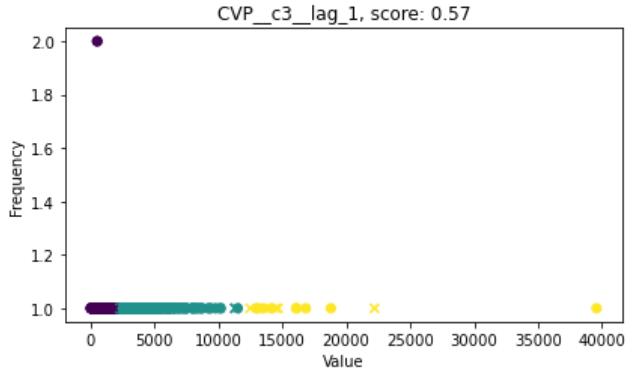
agglomerative



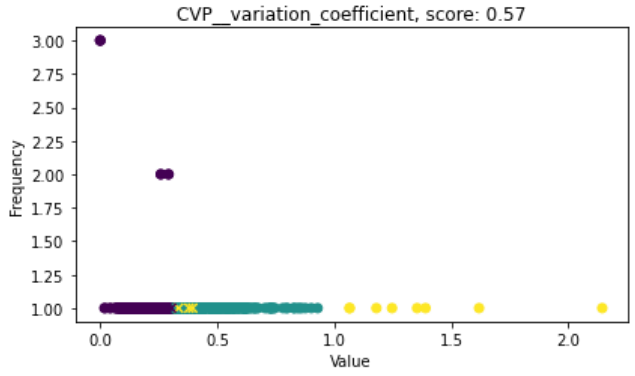
agglomerative

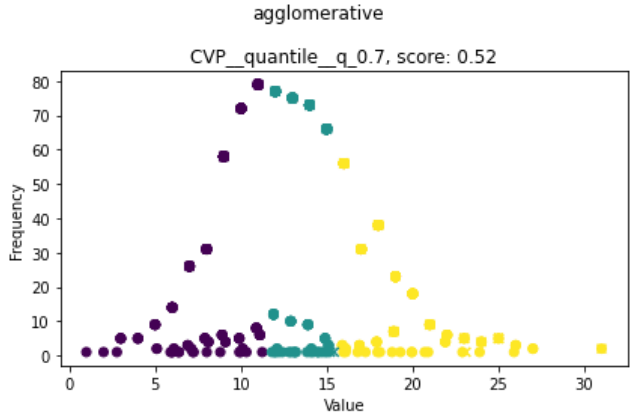


agglomerative



agglomerative





In [ ]:

```
mapped['Mortality14Days'] = targetSeries

mapped.to_csv("./categorisedData/metricExtractedData.csv")
```