- Visualization

```python
train_data = pd.read_csv(f'./{category}/{category}_TRAIN.tsv', sep='\t', header=None).to_numpy()
test_data = pd.read_csv(f'./{category}/{category}_TEST.tsv', sep='\t', header=None).to_numpy()
plot_data, plot_label = resample_plot(test_data)
```

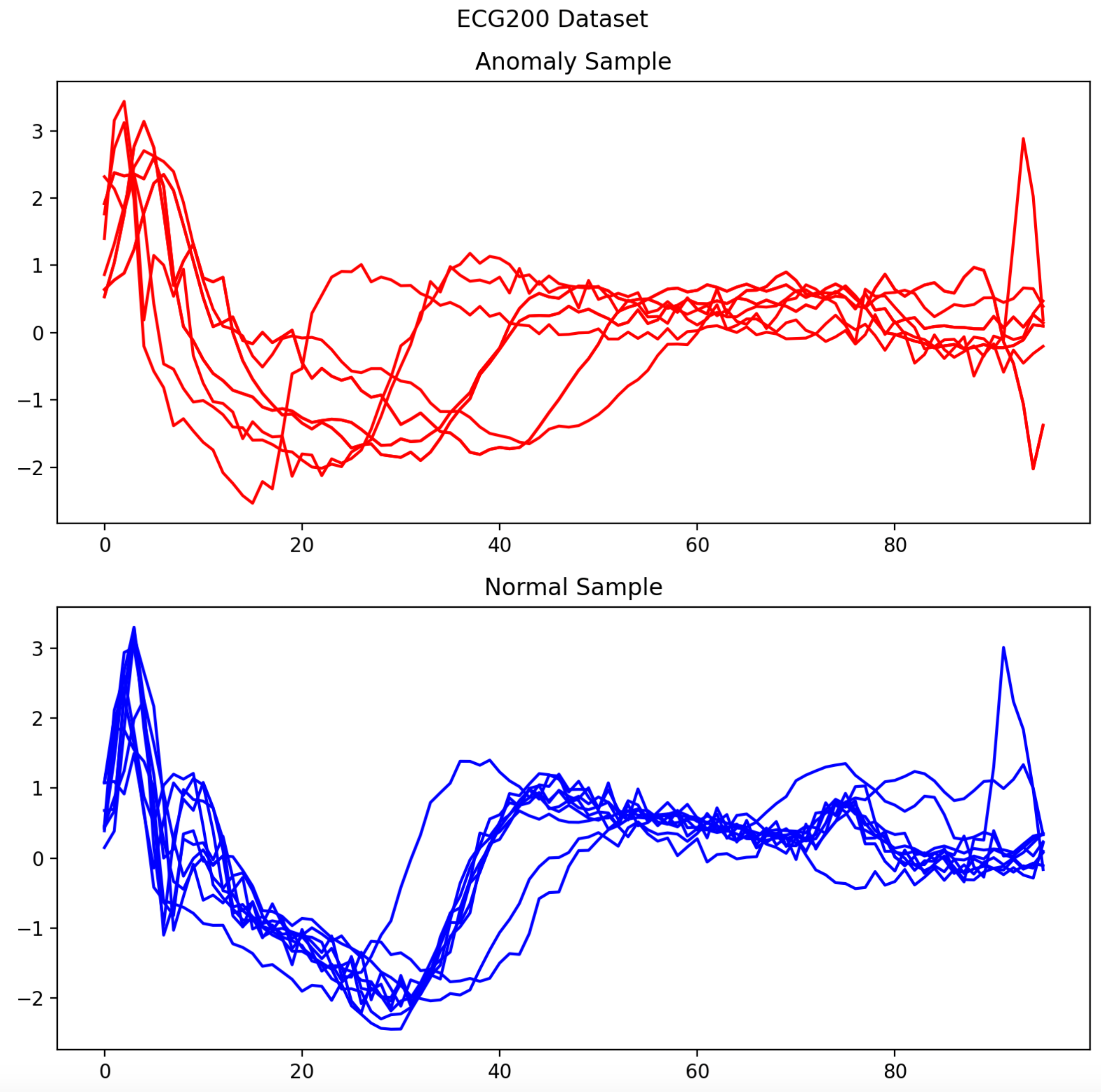
```python
def resample_plot(plot_data):
    # Resample data to plot
    plot_label = plot_data[:, 0].flatten()
    plot_data = plot_data[:, 1:]
    outlier_ratio = 0.1 if category == "Wafer" else 0.2
    plot_data, plot_label = resample(plot_data, plot_label, outlier_ratio=outlier_ratio, target_label=1)

    return plot_data, plot_label
```

因為原始的 test_data 的 abnormal data 不足 10 個，所以我重做一次 resample，讓 abnormal data 可以有 10 個以上，即為 plot_data 和 plot_label，專門用於 visualization。

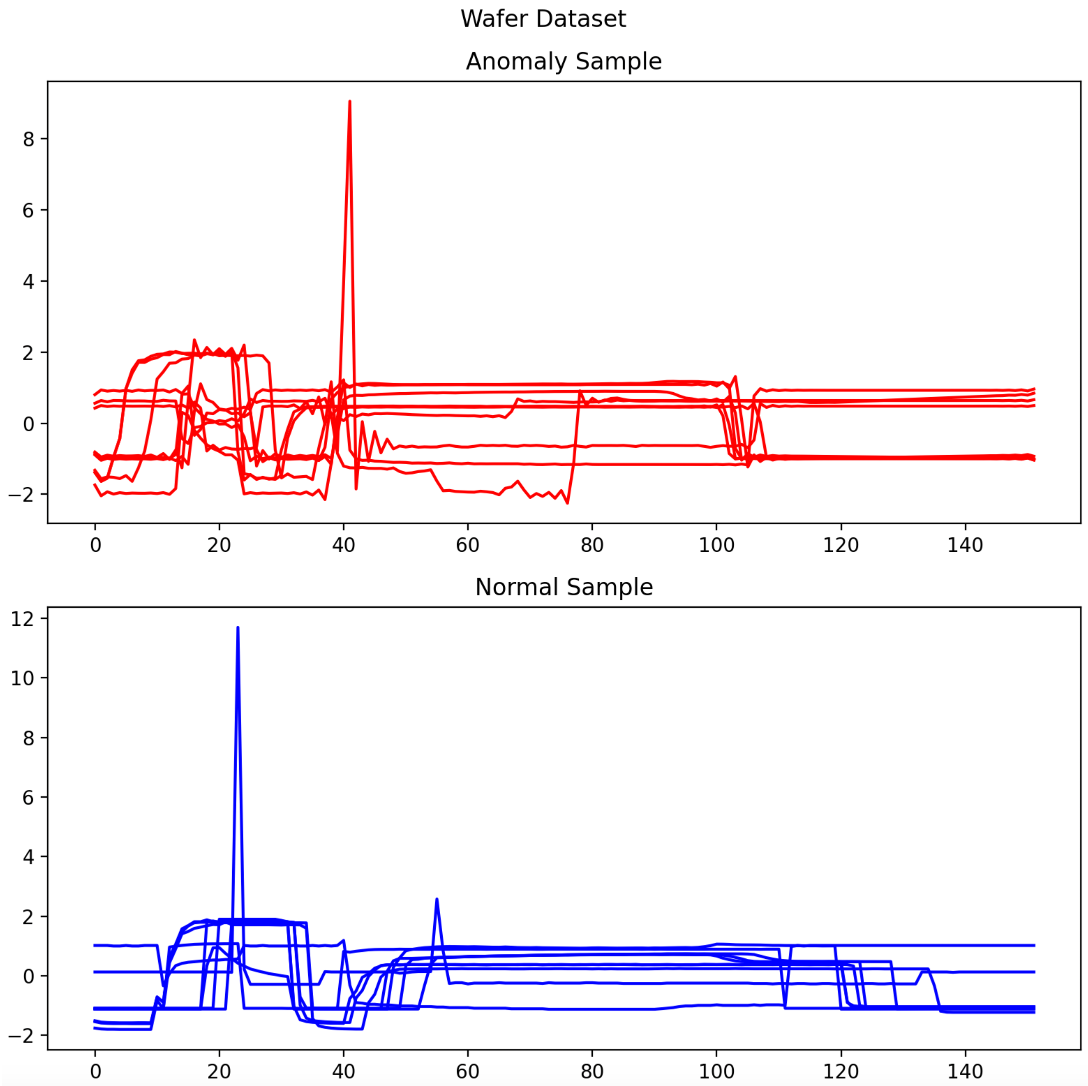
```python
def Visualization(data, label, subtitle, n_samples=10):
    # Get normal and abnormal samples
    normal_data = data[label == 0]
    abnormal_data = data[label == 1]

    # Randomly select samples
    normal_indices = np.random.choice(normal_data.shape[0], n_samples, replace=False)
    abnormal_indices = np.random.choice(abnormal_data.shape[0], n_samples, replace=False)

    # Plotting
    fig, axes = plt.subplots(2, 1, figsize=(8, 8))
    fig.suptitle(subtitle)

    # Plot abnormal samples
    for idx in abnormal_indices:
        axes[0].plot(abnormal_data[idx], 'r-')
    axes[0].set_title('Anomaly Sample')

    # Plot normal samples
    for idx in normal_indices:
        axes[1].plot(normal_data[idx], 'b-')
    axes[1].set_title('Normal Sample')

    plt.tight_layout()
    plt.show()
```

```python
Visualization(plot_data, plot_label, category + " Dataset", n_samples=10)
```

Visualization 裡先是把 normal 和 abnormal 區分開，隨後隨機挑選完各 10 個之後就將其 plot 出來。



ECG200 Dataset
Anomaly Sample
3
2
1
0
−1
−2
0
20
40
60
80
Normal Sample
3
2
1
0
−1
−2
0
20
40
60
80



- KNN on Raw Data

```
def KNN(train_data, test_data, k):
    # Calculate distance
    distances_matrix = pairwise_distances(test_data, train_data, n_jobs=-1)
    k_nearest = np.sort(distances_matrix)[:, :k]
    anomaly_score = np.mean(k_nearest, axis=1)

    return anomaly_score
```

此處的 KNN 使用的是 Hw1 所寫的，讓每個 test_data 的 time series 和 train_data 計算完距離，取前 K 個距離做平均即為 anomaly score。

```
KNN with k=1 on ECG200 score: 0.8671875
KNN with k=2 on ECG200 score: 0.8619791666666667
KNN with k=3 on ECG200 score: 0.8671875
KNN with k=4 on ECG200 score: 0.8619791666666667
KNN with k=5 on ECG200 score: 0.8645833333333333
KNN with k=6 on ECG200 score: 0.859375
KNN with k=7 on ECG200 score: 0.8567708333333334
KNN with k=1 has max score on ECG200: 0.8671875
```

KNN 在 EGG200 上 k=5 時的分數是 0.8645，而表現最好的是 k=1，分數為 0.8671，僅僅是略高一點點，差異並沒有太多。

```
KNN with k=1 on Wafer score: 0.9911923048767601
KNN with k=2 on Wafer score: 0.9894963515472759
KNN with k=3 on Wafer score: 0.989150204822801
KNN with k=4 on Wafer score: 0.9888196264199055
KNN with k=5 on Wafer score: 0.9883118341437142
KNN with k=6 on Wafer score: 0.9875291119332509
KNN with k=7 on Wafer score: 0.9861534685392377
KNN with k=1 has max score on Wafer: 0.9911923048767601
```

KNN 在 Wafer 上 k=5 的分數是 0.9888，而同樣是 k=1 時表現最好，可以來到 0.9911，但同樣差異並沒有太多。

此處我認為是因為 KNN 的方法對於 time series 的 data 來說比較簡單，所以在參數的調整上並不會造成太多的差異。

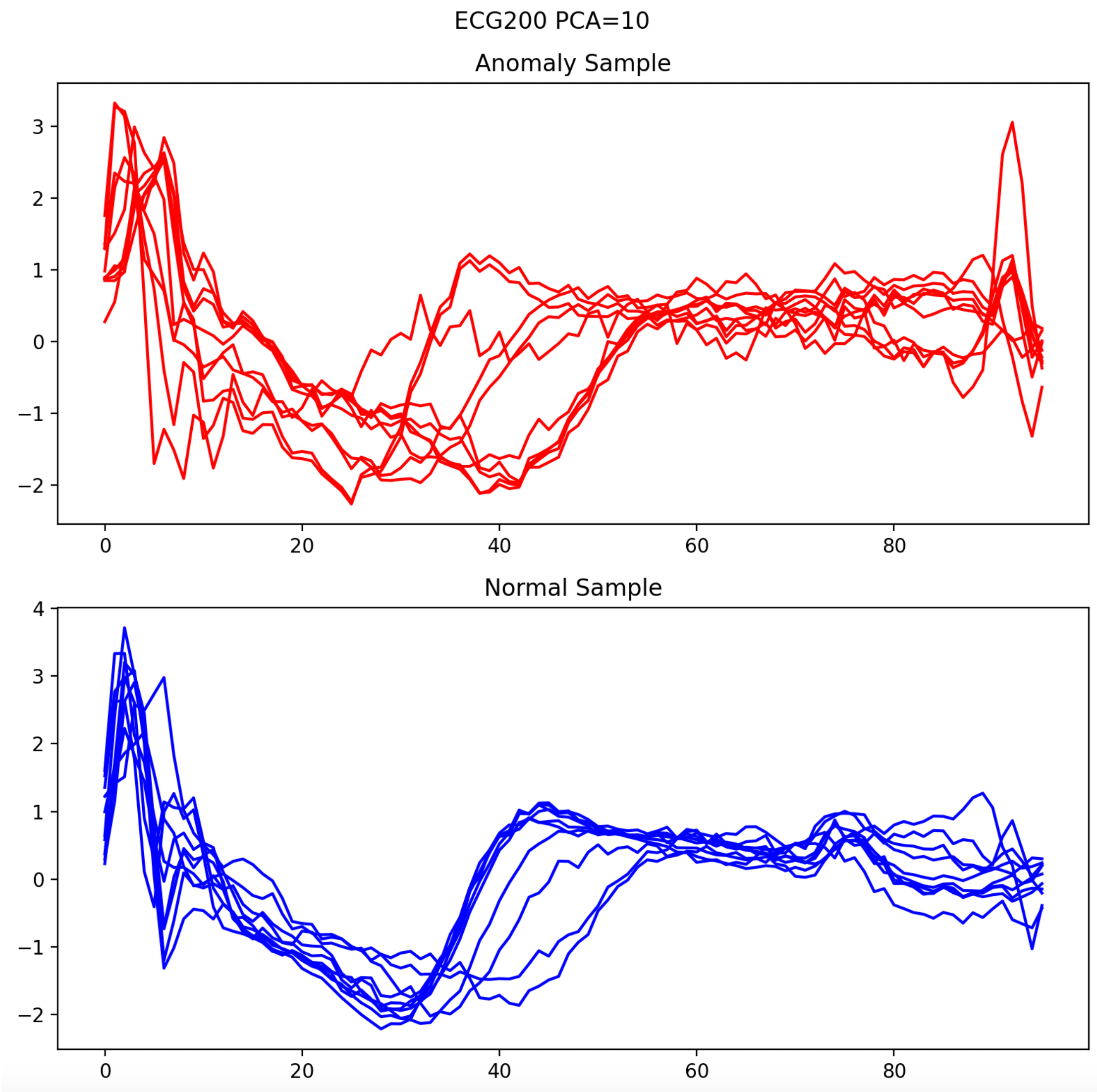
- PCA

```python
def PCA_Reconstruction(train_data, test_data, n):
    # PCA
    pca = PCA(n)
    pca.fit(train_data)
    transform_data = pca.transform(test_data)
    reconstruct_test = pca.inverse_transform(transform_data) # for visualization purpose

    # Calculate anomaly score
    anomaly_score = []
    for i in range(test_data.shape[0]):
        anomaly_score.append(np.linalg.norm(test_data[i] - reconstruct_test[i]))

    return np.array(anomaly_score), reconstruct_test
```

在 PCA 裡，首先以 train_data 去做 fit，獲得的 n 個 primary component 再去對 test_data 做 transform，隨後再將 transform 完的 data 做重建，最後就是去計算原始和重建的距離。

```
PCA with n=5 on ECG200 score: 0.7057291666666667
PCA with n=10 on ECG200 score: 0.7578125
PCA with n=15 on ECG200 score: 0.7135416666666667
PCA with n=20 on ECG200 score: 0.7213541666666667
PCA with n=25 on ECG200 score: 0.734375
PCA with n=30 on ECG200 score: 0.7317708333333333
PCA with n=35 on ECG200 score: 0.7421875
PCA with n=40 on ECG200 score: 0.7395833333333334
PCA with n=45 on ECG200 score: 0.75
PCA with n=50 on ECG200 score: 0.7552083333333334
PCA with n=55 on ECG200 score: 0.7395833333333333
PCA with n=60 on ECG200 score: 0.7213541666666666
PCA with n=65 on ECG200 score: 0.7473958333333334
PCA with n=10 has max score on ECG200: 0.7578125
```

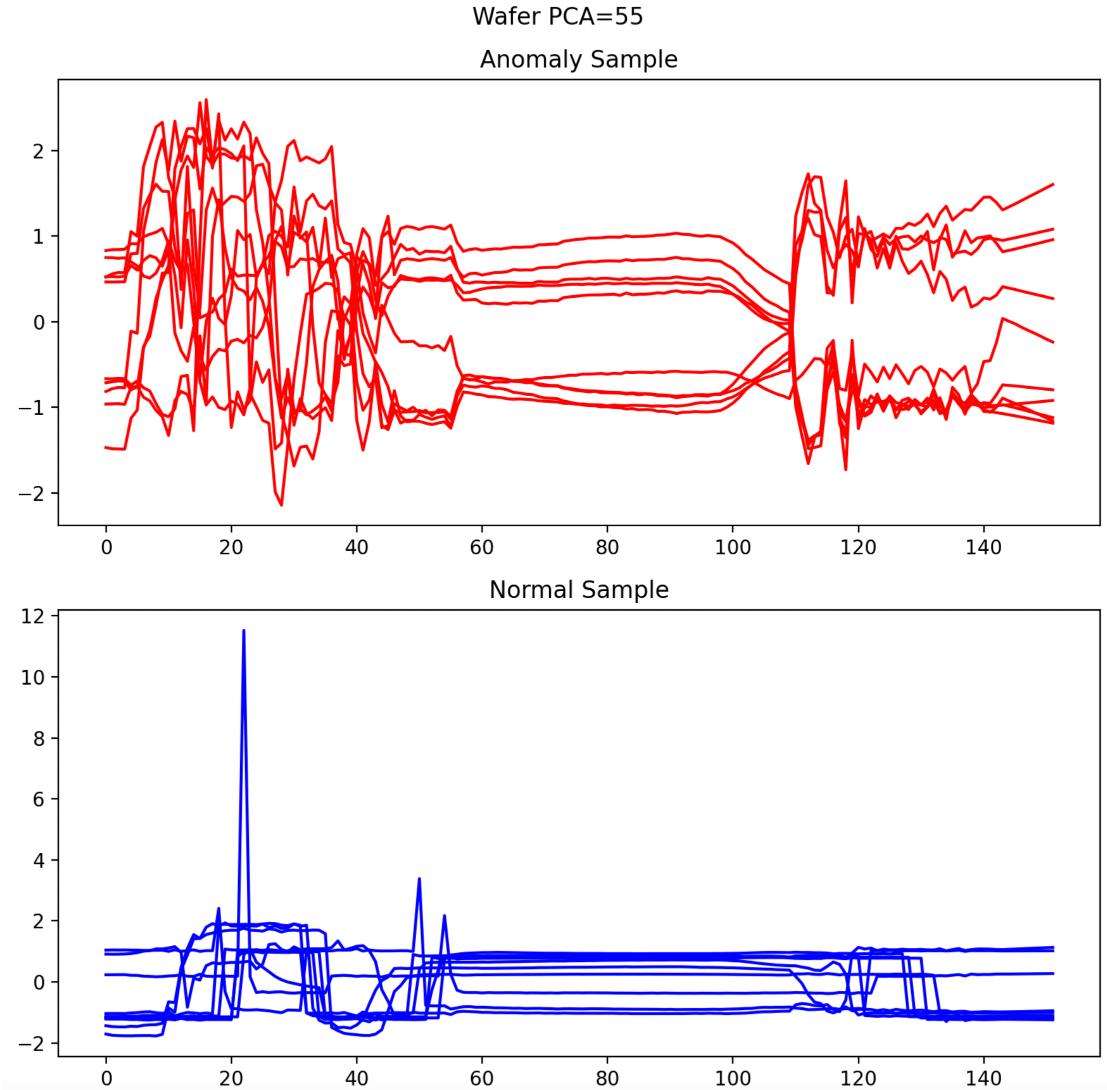
PCA 在 ECG200 的重建上，從圖中可以看到，當 n=10 時是所有 n 中最好的，但是其表現只有 0.7578，遠低於 KNN 的 0.8645，猜測是因為 PCA 在 ECG200 的降維後再重建，其結構不能較好的區分 normal 和 abnormal。以下是繪圖。




```
PCA with n=5 on Wafer score: 0.9552930802785472
PCA with n=10 on Wafer score: 0.9741552612149055
PCA with n=15 on Wafer score: 0.9828539780870905
PCA with n=20 on Wafer score: 0.9907722914350051
PCA with n=25 on Wafer score: 0.9913065829819695
PCA with n=30 on Wafer score: 0.9934493802648668
PCA with n=35 on Wafer score: 0.994565330805303
PCA with n=40 on Wafer score: 0.9949015403032377
PCA with n=45 on Wafer score: 0.9956925435358175
PCA with n=50 on Wafer score: 0.9960635333266422
PCA with n=55 on Wafer score: 0.9961638993146957
PCA with n=60 on Wafer score: 0.9958054966774883
PCA with n=65 on Wafer score: 0.9954504064491275
PCA with n=70 on Wafer score: 0.9954507376900121
PCA with n=55 has max score on Wafer: 0.9961638993146957
```

PCA 在 Wafer 的重建上，從圖中可以看到當 n=55 時表現最好，分數來到

0.9961，比 KNN 的效果略好一點，但整體差異並沒有很大，以下是繪圖。



- DFT

```python
def DFT(data, m):
    # DFT
    fft_data = fft(data)

    # apply FFT frequency on feature dimension
    frequency = fftfreq(data.shape[1])

    # get lowest m coefficient
    lowest_index = np.argsort(np.abs(frequency))[:m]
    lowest_data = fft_data[:, lowest_index]
    magnitude = np.real(lowest_data)

    # for visualization purpose
    reconstruct_data = np.zeros_like(fft_data)
    reconstruct_data[:, lowest_index] = lowest_data

    return magnitude, reconstruct_data

def DFT_Function(train_data, test_data, m, k):
    # get lowest m DFT coefficient
    magnitude_train, _ = DFT(train_data, m)
    magnitude_test, reconstruct_data = DFT(test_data, m)

    # apply KNN anomaly detection
    anomaly_score = KNN(magnitude_train, magnitude_test, k)

    return anomaly_score, reconstruct_data
```

DFT_Function 為核心 function，主要就是獲得 train_data 和 test_data 的 DFT coefficient 後做 KNN。

DFT 內首先是先對 data 做轉換，隨後以轉換的 frequency 去做 sort，取得最小的 m 個 coefficient 作為 KNN 的比較，而此處需要注意的是，要做 reconstruct 時，m 個最小的 coefficient 要放回對應的 column，其他地方補 0，這樣重建出來的圖才會是正確的。

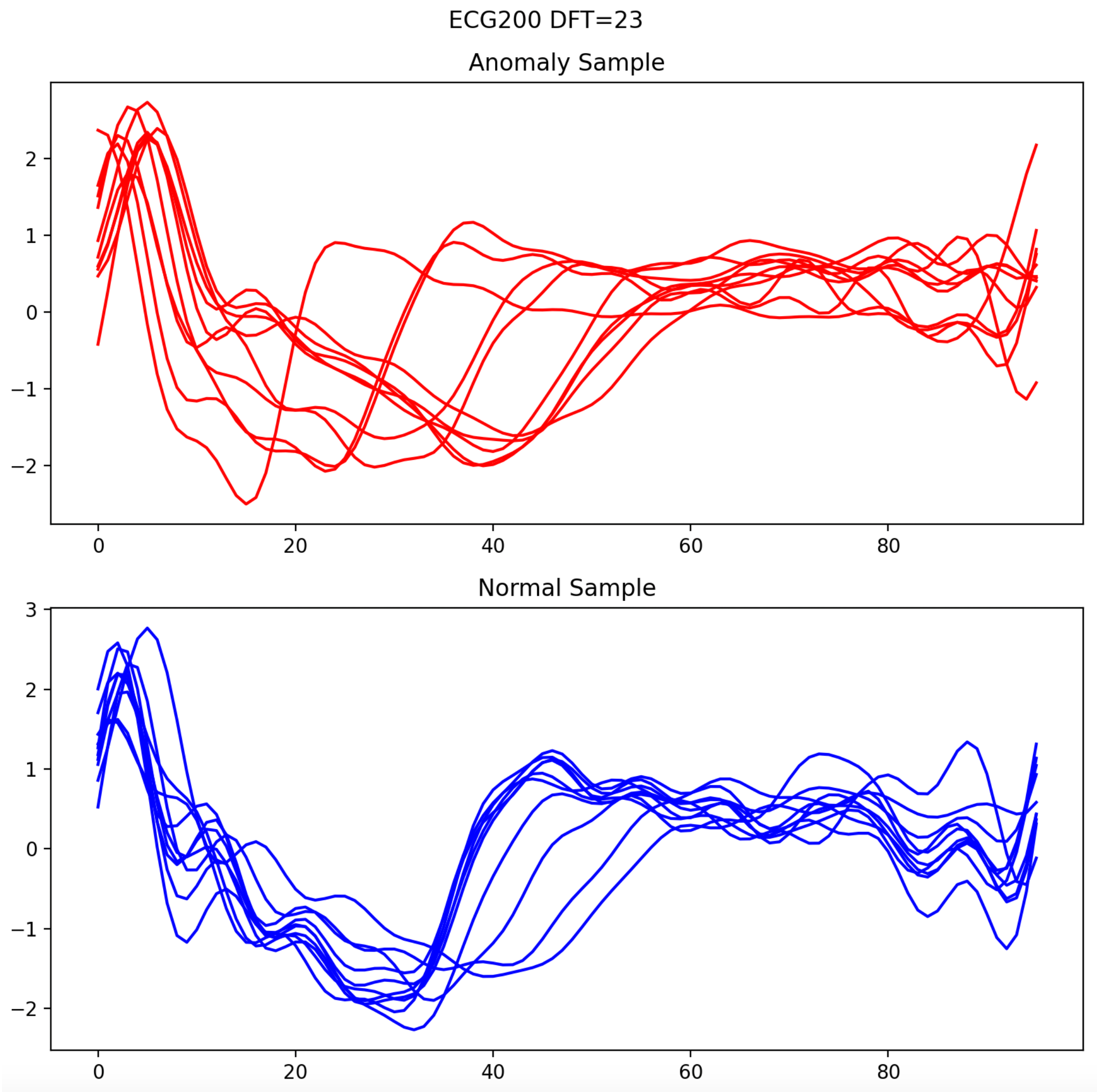
```python
_, reconstruct_data = DFT_Function(train_data, plot_data, max_m, 5)
reconstruct_data = ifft(reconstruct_data).real
Visualization(reconstruct_data, plot_label, category + f" DFT={max_m}", n_samples=10)
```

繪圖的部分，因為無法將虛數給呈現出來，所以這邊做完 inverse transform 後，取實數的部分進行繪圖。

```
DFT with m=23 and k=1 has max score on ECG200: 0.9427083333333334
```

```
DFT with m=23 and k=2 has max score on ECG200: 0.9348958333333334
DFT with m=19 and k=3 has max score on ECG200: 0.9322916666666666
DFT with m=22 and k=4 has max score on ECG200: 0.9270833333333334
DFT with m=23 and k=5 has max score on ECG200: 0.9166666666666666
```

以上是 DFT 的 m=0～40，k=1～5 在 ECG200 上最好的參數搭配，其中最好的是 m=23，k=1 時，分數可以來到 0.9427，比單純 KNN 的 k=1 為 0.8617 高出很多，以下是以最佳參數進行繪圖的結果。



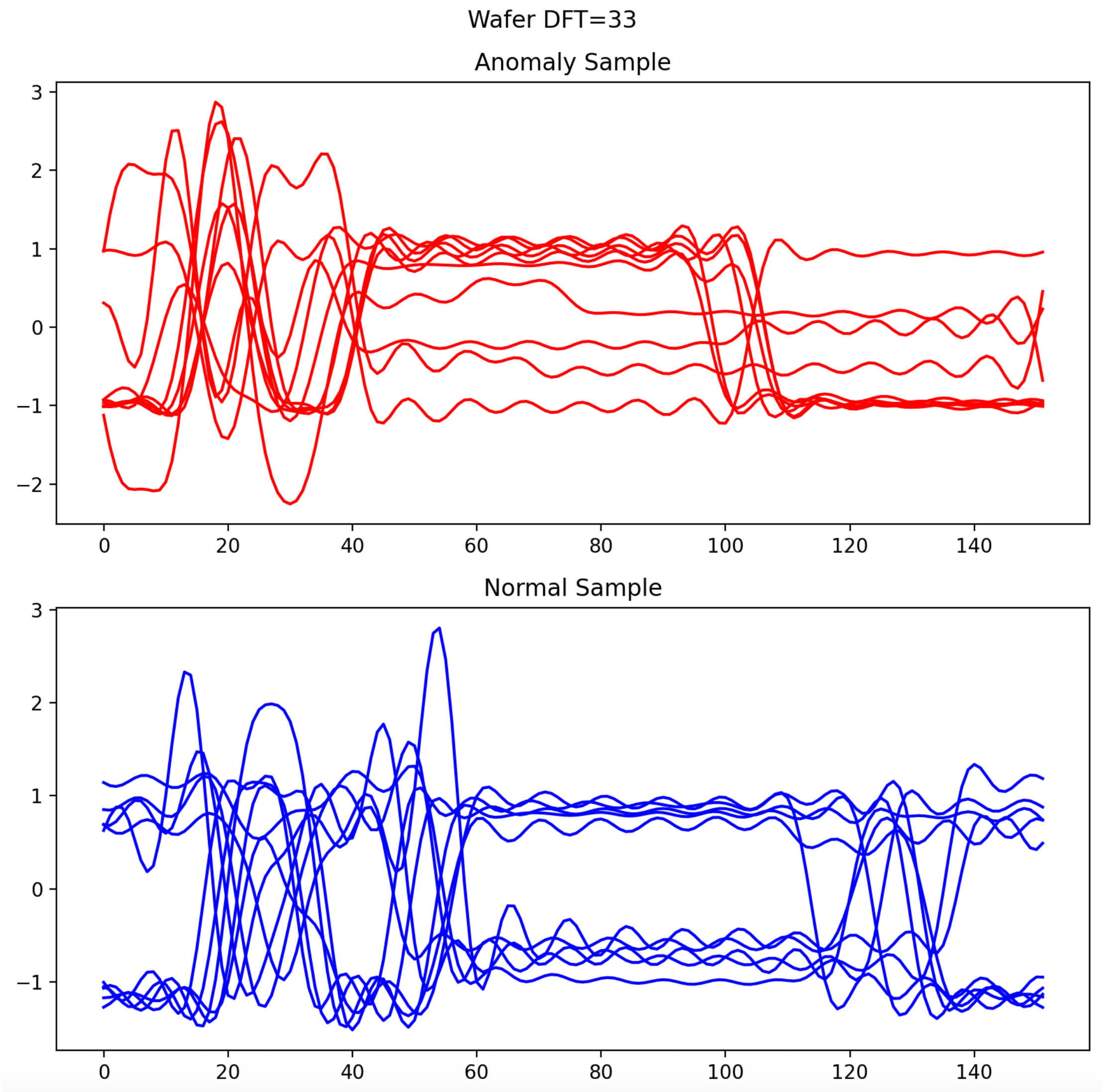

```
DFT with m=33 and k=1 has max score on Wafer: 0.9854585251632106
```

```
DFT with m=33 and k=2 has max score on Wafer: 0.9892475896428925
```

```
DFT with m=33 and k=3 has max score on Wafer: 0.9883403208597953
DFT with m=13 and k=4 has max score on Wafer: 0.987516193538749
DFT with m=13 and k=5 has max score on Wafer: 0.986707634539282
```

在 DFT 中參數搭配，可以看到 m=33 及 k=2 時的表現最好，分數為

0.9892，比 KNN 略低一些，猜測可能是因為 Wafter 本身在區分 anomaly 上的效果就已經不錯，使得比較複雜的方法可能與簡單一點的方法差異不大，以下是以最佳參數進行繪圖的結果。



- DWT

```python
def DWT_Function(train_data, test_data, s, k):
    # compute the next power of 2
    feature_dim = train_data.shape[1]
    max_power = math.ceil(math.log2(feature_dim))
    max_length = int(math.pow(2, max_power))

    # pad train and test data
    padding_width = max_length - feature_dim
    train_data_padded = np.pad(train_data, ((0, 0), (0, padding_width)), 'constant')
    test_data_padded = np.pad(test_data, ((0, 0), (0, padding_width)), 'constant')

    # get dwt_coefficient
    train_dwt = DWT(train_data_padded, s, max_power)
    test_dwt = DWT(test_data_padded, s, max_power)

    # apply KNN anomaly detection
    anomaly_score = KNN(train_dwt, test_dwt, k)

    return anomaly_score
```

在 DWT 的主要 function 內，首先計算 feature dimension 的下一個 2 的次方為何，並將 train_data 和 test_data 都 padding 到該長度，隨後便以 padded 後的 data 取獲得個別的 DWT coefficients，最後透過 KNN 獲得 anomaly score。

```python
def DWT(data, s, max_pow):
    dwt_coefs = []
    # build DWT table
    for i in range(data.shape[0]):
        approx = data[i, :].tolist() # previous level approximate
        coef = []
        detail_level = []
        for j in range(max_pow): # max_pow indicate max level to do
            new_approx = []
            new_detail = []
            for k in range(0, len(approx), 2):
                new_approx.append((approx[k] + approx[k+1]) / 2)
                new_detail.append((approx[k+1] - approx[k]) / 2)
            approx = new_approx
            detail_level.append(new_detail)
        # get S coefficients according to level
        coef.append(approx[0]) # first coef always be the last level aprox
        level_s = int(math.log2(s))
        details_index = max_pow - 1
        for j in range(level_s):
            for detail in detail_level[details_index]:
                coef.append(detail)
            details_index -= 1
        dwt_coefs.append(coef)

    return np.array(dwt_coefs)
```

DWT 內首先是建表，具體就是下一 level 的 approximate 和 detail 是透過上一 level 的 approximate 計算得到，而 approximate 一開始初始化為每一個 time series 的 features。
在建表完成後，便是去取得 S 個 coefficient，第一個 coef 一定是最後一個 level 的 approximate，隨後就是取每一個 level 的 details，直到取到 S 個

```
DWT with s=1 and k=5 on ECG200 score: 0.3216145833333333
DWT with s=2 and k=5 on ECG200 score: 0.6770833333333334
DWT with s=4 and k=5 on ECG200 score: 0.9375
DWT with s=8 and k=5 on ECG200 score: 0.8854166666666666
DWT with s=16 and k=5 on ECG200 score: 0.8984375
DWT with s=32 and k=5 on ECG200 score: 0.859375
DWT with s=64 and k=5 on ECG200 score: 0.8489583333333333
DWT with s=128 and k=5 on ECG200 score: 0.7265625
DWT with s=4 and k=5 has max score on ECG200: 0.9375
```

```
DWT with s=4 and k=1 has max score on ECG200: 0.9401041666666666
```

```
DWT with s=4 and k=2 has max score on ECG200: 0.9557291666666666
```

```
DWT with s=4 and k=3 has max score on ECG200: 0.9479166666666666
```

```
DWT with s=4 and k=4 has max score on ECG200: 0.9479166666666666
```

以上是 DWT 在 ECG200 上的參數搜尋，s 以 2 的次方增加直到最大長度，k 則是 1～5，從圖中可以看出，當 s=4，k=2 時表現最好，分數來到了 0.9557，比 DFT 最好的參數 0.9427 又高上了一些，猜測是因為該方法比較能考慮到 time series 內每個 time step 的關係，這樣在比較上更精準。

```
DWT with s=1 and k=5 on Wafer score: 0.22562174742153812
DWT with s=2 and k=5 on Wafer score: 0.6255888220776025
DWT with s=4 and k=5 on Wafer score: 0.8496192882892104
DWT with s=8 and k=5 on Wafer score: 0.9984951726609673
DWT with s=16 and k=5 on Wafer score: 0.9974663384731982
DWT with s=32 and k=5 on Wafer score: 0.9977979105987478
DWT with s=64 and k=5 on Wafer score: 0.9973000555490963
DWT with s=128 and k=5 on Wafer score: 0.9920064949712666
DWT with s=256 and k=5 on Wafer score: 0.9604862748683234
DWT with s=8 and k=5 has max score on Wafer: 0.9984951726609673
```

```
DWT with s=32 and k=1 has max score on Wafer: 0.9974037339459965
```

```
DWT with s=16 and k=2 has max score on Wafer: 0.9977647865102812
```

```
DWT with s=8 and k=3 has max score on Wafer: 0.9982530355742772
```

```
DWT with s=8 and k=4 has max score on Wafer: 0.9984222996663411
```

以上是 DWT 在 Wafer 上的參數搜尋，可以從圖表看出，當 s=8，k=5 時表現最好，分數可以來到 0.9984，比起單純 KNN 和 DFT 都來的好，並且已經很接近滿分，