2021 Summer ELG 5142 Ubiquitous Sensing and Smart City

Group assignment 3

Team members-Group 8:

- Abdelrhman Gaber Youssef Saad Rezkallah
- Eman Metwally Mohammed Abood
- Basma Reda Shaban Abd-Elsalam Abd-Elwahab

Introduction:

Anomalies data can affects on the result of the machine learning models and every system, so we need to define the anomalies data, because anomalies data are outliers are different from the right data.

So, we can use four algorithms for anomaly detection:

1- KNN model.
2- SVM model.
3- PCA.
4- DBSCAN

Implementation steps:

PyCaret anomaly detection module provides several pre-processing features that can be configured when initializing the setup through `setup` function. It has over 12 algorithms and a few plots to analyze the results of anomaly detection. PyCaret's anomaly detection module also implements a unique function `tune_model` that allows you to tune the hyperparameters of the anomaly detection model to optimize the supervised learning objective such as `AUC` for classification or `R2` for regression.

1- Read the dataset without label:

```
[47] #read the dataset
     df = pd.read_csv('/content/Dataset_to_be_used_in_anomaly_detection.csv')
     df.head()
```

	Unnamed: 0	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader
0	9	-1.042570	-0.241098	-1.267957	0.414568
1	10	-1.056986	-0.245590	-1.165454	0.411869
2	11	-1.071858	-0.256787	-1.028780	0.407472
3	12	-1.084518	-0.257502	-0.850609	0.367564
4	13	-0.974811	-0.105985	-0.625045	0.236174

Get some information about the dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Data columns (total 4 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Follower_measure_x_follower  98 non-null     float64
 1   Follower_measure_y_follower  98 non-null     float64
 2   Leader_measure_x_leader      98 non-null     float64
 3   Leader_measure_y_leader      98 non-null     float64
dtypes: float64(4)
memory usage: 3.2 KB
```

visualize the data using TSNE:

```
#to visualize the data using tsne
tsne = TSNE(n_components= 2 ,random_state=0)
z = tsne.fit_transform(data)
```

Show the features of the data:

```
from pycaret.anomaly import *
exp_ano101 = setup(data, session_id = 123)
```

	Description	Value
0	session_id	123
1	Original Data	(98, 4)
2	Missing Values	False
3	Numeric Features	4
4	Categorical Features	0
5	Ordinal Features	False
6	High Cardinality Features	False
7	High Cardinality Method	None
8	Transformed Data	(98, 4)
9	CPU Jobs	-1
10	Use GPU	False
11	Log Experiment	False
12	Experiment Name	anomaly-default-name
13	USI	772e
14	Imputation Type	simple
15	Iterative Imputation Iteration	None
16	Numeric Imputer	mean
17	Iterative Imputation Numeric Model	None

View the models:

ID	Name	Reference
abod	Angle-base Outlier Detection	pyod.models.abod.ABOD
cluster	Clustering-Based Local Outlier	pyod.models.cblof.CBLOF
cof	Connectivity-Based Local Outlier	pyod.models.cof.COF
iforest	Isolation Forest	pyod.models.iforest.IForest
histogram	Histogram-based Outlier Detection	pyod.models.hbos.HBOS
knn	K-Nearest Neighbors Detector	pyod.models.knn.KNN
lof	Local Outlier Factor	pyod.models.lof.LOF
svm	One-class SVM detector	pyod.models.ocsvm.OCSVM
pca	Principal Component Analysis	pyod.models.pca.PCA
mcd	Minimum Covariance Determinant	pyod.models.mcd.MCD
sod	Subspace Outlier Detection	pyod.models.sod.SOD
sos	Stochastic Outlier Selection	pyod.models.sos.SOS

Apply SVM model:

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples.

```
[19] svm = create_model('svm')
     print(svm)
```

```
OCSVM(cache_size=200, coef0=0.0, contamination=0.05, degree=3, gamma='auto',
   kernel='rbf', max_iter=-1, nu=0.5, shrinking=True, tol=0.001,
   verbose=False)
```

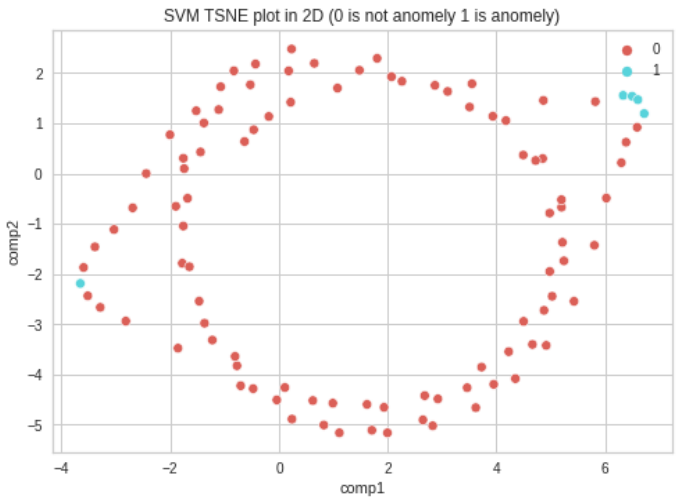
Assign model function assigns anomaly labels to the dataset for a given model and show Two features (Anomaly, Anomaly_score).

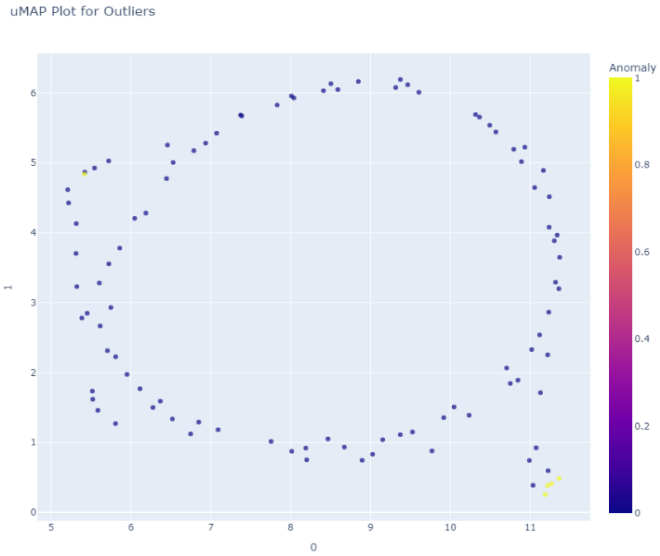
```
svm_model = assign_model(svm)
svm_model.head()
```

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
0	-1.042570	-0.241098	-1.267957	0.414568	0	-0.846592
1	-1.056986	-0.245590	-1.165454	0.411869	0	-0.859350
2	-1.071858	-0.256787	-1.028780	0.407472	0	-0.760461
3	-1.084518	-0.257502	-0.850609	0.367564	0	-0.665501
4	-0.974811	-0.105985	-0.625045	0.236174	0	-0.400389

TSNE plot in 2D (0 is not anomaly, 1 is anomaly) and umap:

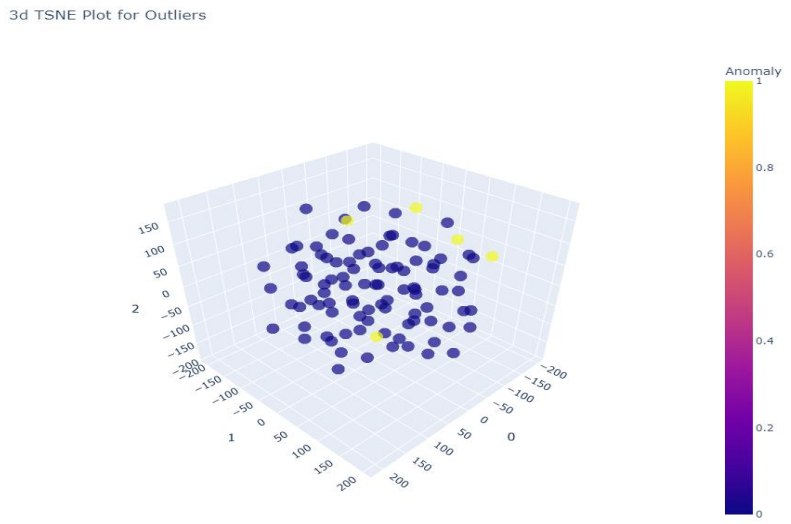
```
df_3 = pd.DataFrame()
df_3['comp1'] = z[:,0]
df_3['comp2'] = z[:,1]
df_3['target'] = svm_model['Anomaly']
sns.scatterplot(x="comp1", y="comp2", hue=df_3.target.tolist(),
                palette=sns.color_palette("hls", 2),
                data=df_3).set(title="SVM TSNE plot in 2D (0 is not anomely 1 is anomely)")
```





3D TSNE Plot for outliers:

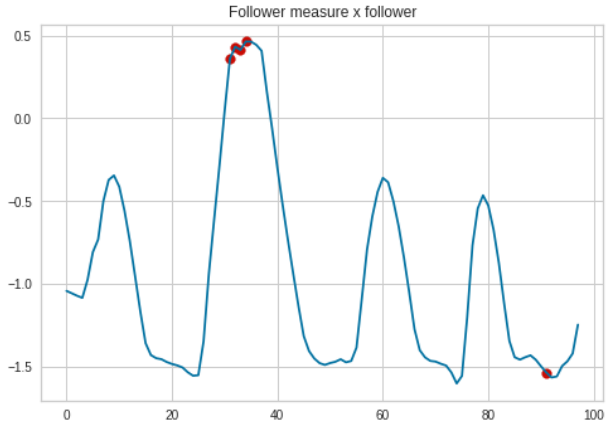
The yellow points are the anomaly, the blue points are the data

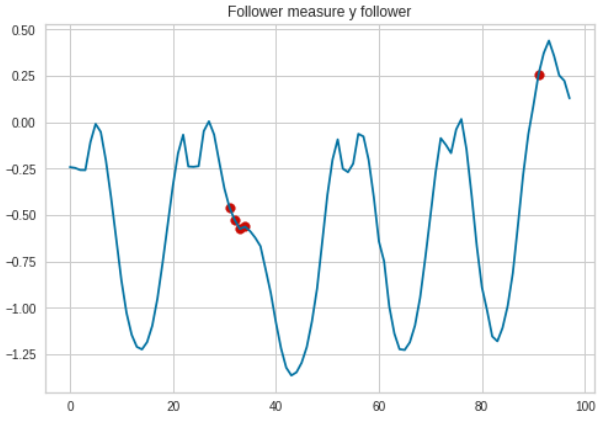


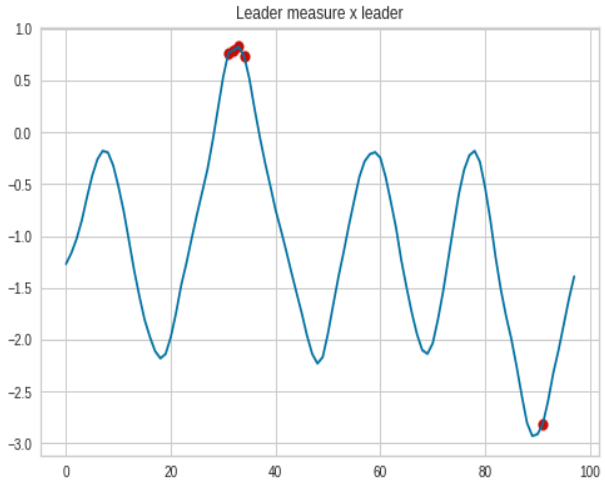
show only the anomaly data:

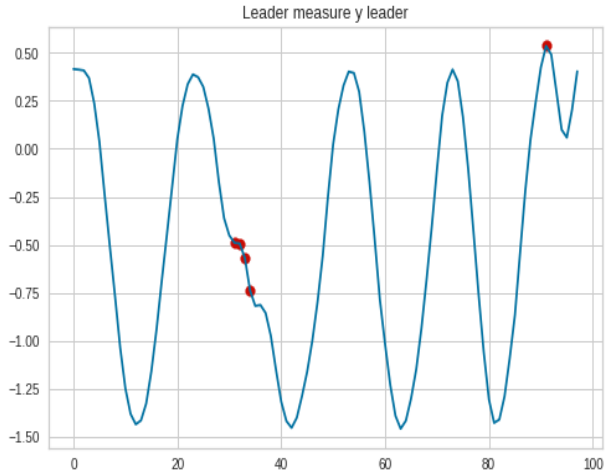
	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
31	0.360847	-0.459538	0.767382	-0.489506	1	8.172418
32	0.432356	-0.526272	0.795491	-0.493113	1	8.637892
33	0.412900	-0.574391	0.827058	-0.568237	1	8.695015
34	0.463492	-0.562305	0.734001	-0.737032	1	8.338360
91	-1.535425	0.260158	-2.816451	0.535399	1	7.785698

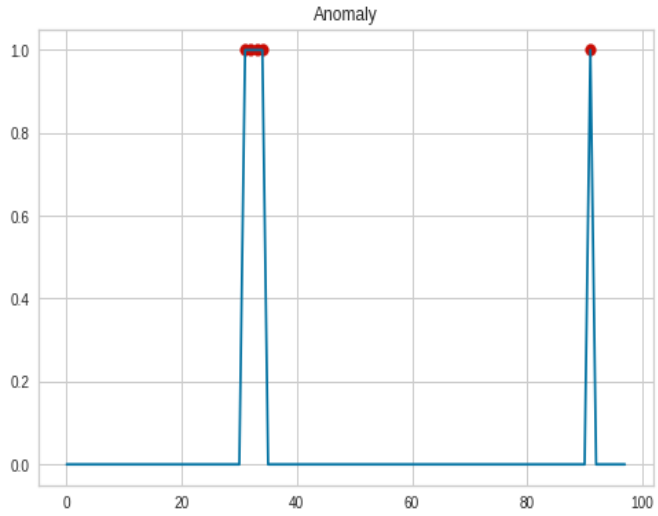
Plot SVM model results along the data:











Model evaluation for SVM:

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
0	-1.042570	-0.241098	-1.267957	0.414568	0	-0.846592
1	-1.056986	-0.245590	-1.165454	0.411869	0	-0.859350
2	-1.071858	-0.256787	-1.028780	0.407472	0	-0.760461
3	-1.084518	-0.257502	-0.850609	0.367564	0	-0.665501
4	-0.974811	-0.105985	-0.625045	0.236174	0	-0.400389

Classification report for SVM model:

	precision	recall	f1-score	support
0	1.00	0.92	0.96	93
1	0.42	1.00	0.59	5
accuracy			0.93	98
macro avg	0.71	0.96	0.77	98
weighted avg	0.97	0.93	0.94	98

So, the accuracy of SVM model is 93%

Apply KNN model:

kNN Is a Supervised Learner for Both Classification and Regression. Supervised machine learning algorithms can be split into two groups based on the type of target variable that they can predict: Classification is a prediction task with a categorical target variable.

```
#create KNN model
knn = create_model('knn')
print(knn)
```

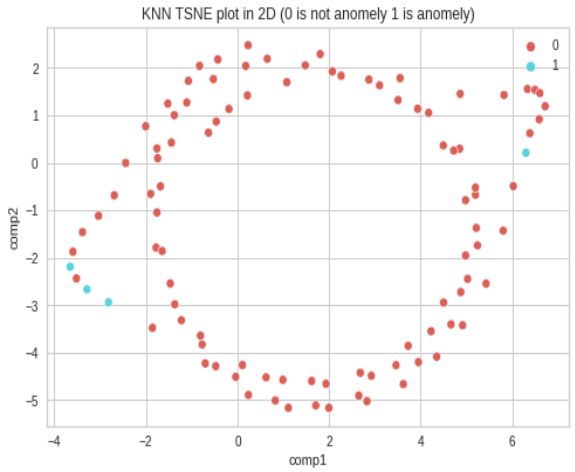
```
KNN(algorithm='auto', contamination=0.05, leaf_size=30, method='largest',
  metric='minkowski', metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
  radius=1.0)
```

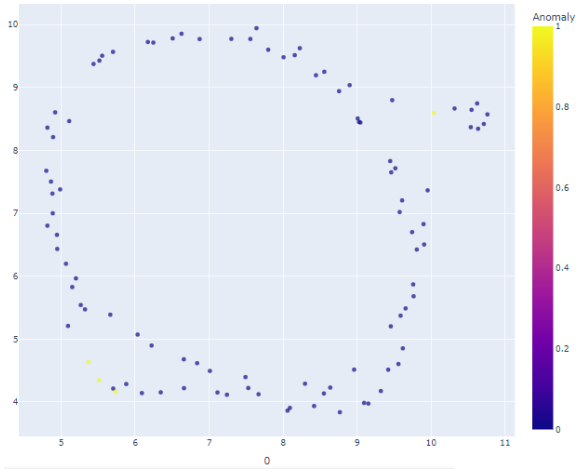
Assign model function assigns anomaly labels to the dataset for a given model and show Two features (Anomaly, Anomaly score)

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
0	-1.042570	-0.241098	-1.267957	0.414568	0	0.446403
1	-1.056986	-0.245590	-1.165454	0.411869	0	0.476462
2	-1.071858	-0.256787	-1.028780	0.407472	0	0.420715
3	-1.084518	-0.257502	-0.850609	0.367564	0	0.385327
4	-0.974811	-0.105985	-0.625045	0.236174	0	0.380718

TSNE plot in 2D (0 is not anomaly, 1 is anomaly) and umap:

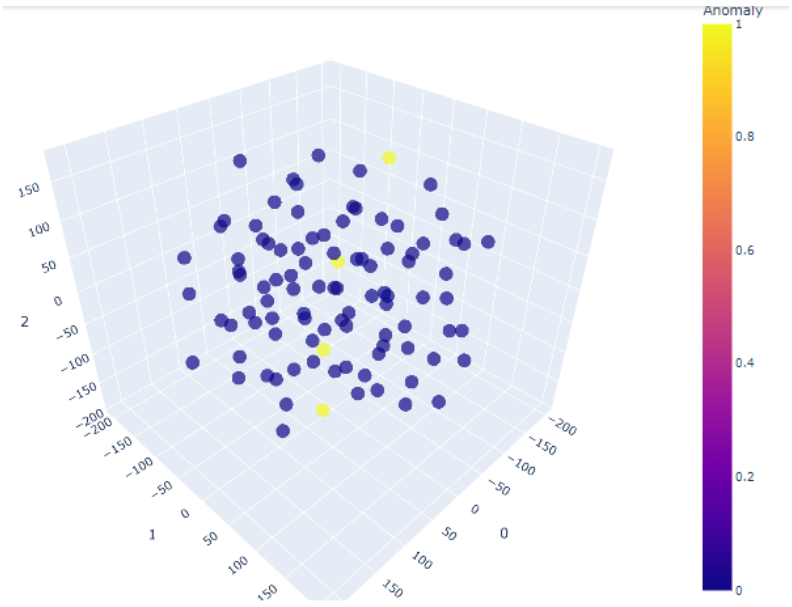
```
df_2 = pd.DataFrame()
df_2['comp1'] = z[:,0]
df_2['comp2'] = z[:,1]
df_2['target'] = knn_model['Anomaly']
sns.scatterplot(x="comp1", y="comp2", hue=df_2.target.tolist(),
                palette=sns.color_palette("hls", 2),
                data=df_2).set(title="KNN TSNE plot in 2D (0 is not anomely 1 is anomely)")
```





3D TSNE Plot for outliers:

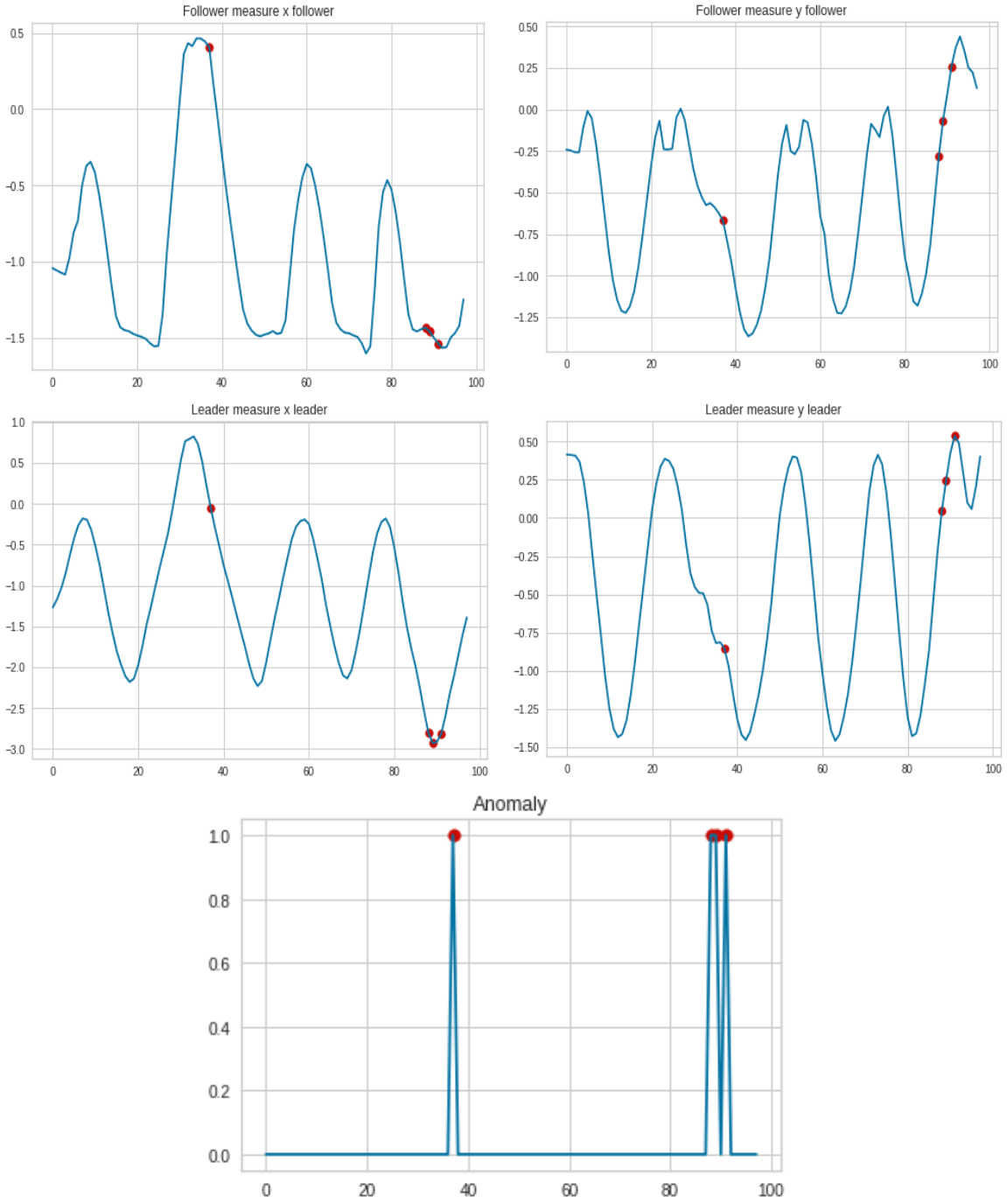
The yellow points are the anomaly, the blue points are the data



Anomaly data:

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
37	0.407215	-0.667258	-0.046350	-0.854807	1	0.798124
88	-1.431881	-0.282536	-2.802852	0.047603	1	0.758008
89	-1.458214	-0.064960	-2.928637	0.242762	1	0.775092
91	-1.535425	0.260158	-2.816451	0.535399	1	0.737135

Plot KNN model results along the data:



Model evaluation for KNN

```
[12] knn_model.tail()
```

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
93	-1.559131	0.440215	-2.325538	0.295837	0	0.697068
94	-1.496434	0.357878	-2.105013	0.098846	0	0.651218
95	-1.467606	0.253125	-1.857816	0.058397	0	0.540269
96	-1.420551	0.223617	-1.606946	0.202749	0	0.402136
97	-1.246517	0.129141	-1.390368	0.402667	0	0.421741

Evaluate the dataset that has labels:

```
df_evaluate = pd.read_csv('/content/Dataset_to_be_used_in_performance_comparison.csv')
df_evaluate
```

	Unnamed: 0	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	labels
0	9	-1.042570	-0.241098	-1.267957	0.414568	0.0
1	10	-1.056986	-0.245590	-1.165454	0.411869	0.0
2	11	-1.071858	-0.256787	-1.028780	0.407472	0.0
3	12	-1.084518	-0.257502	-0.850609	0.367564	0.0
4	13	-0.974811	-0.105985	-0.625045	0.236174	0.0
...	...	...	...	...	...	...
93	102	-1.559131	0.440215	-2.325538	0.295837	0.0
94	103	-1.496434	0.357878	-2.105013	0.098846	0.0
95	104	-1.467606	0.253125	-1.857816	0.058397	0.0
96	105	-1.420551	0.223617	-1.606946	0.202749	0.0
97	106	-1.246517	0.129141	-1.390368	0.402667	0.0

98 rows × 6 columns

show only the data that has label=1

```
df_evaluate[df_evaluate['labels']==1]
```

	Unnamed: 0	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	labels
30	39	0.046769	-0.356131	0.536808	-0.450804	1.0
31	40	0.360847	-0.459538	0.767382	-0.489506	1.0
32	41	0.432356	-0.526272	0.795491	-0.493113	1.0
33	42	0.412900	-0.574391	0.827058	-0.568237	1.0
34	43	0.463492	-0.562305	0.734001	-0.737032	1.0
35	44	0.462689	-0.586497	0.517651	-0.817752	1.0
87	96	-1.443234	-0.551990	-2.540817	-0.218693	1.0
88	97	-1.431881	-0.282536	-2.802852	0.047603	1.0
89	98	-1.458214	-0.064960	-2.928637	0.242762	1.0
90	99	-1.498133	0.092471	-2.912866	0.423525	1.0
91	100	-1.535425	0.260158	-2.816451	0.535399	1.0
92	101	-1.565423	0.372923	-2.596673	0.489932	1.0

Classification report of KNN model:

	precision	recall	f1-score	support
0	0.99	0.90	0.94	94
1	0.25	0.75	0.38	4
accuracy			0.90	98
macro avg	0.62	0.83	0.66	98
weighted avg	0.96	0.90	0.92	98

So, the accuracy of KNN model is 90%

Apply PCA model:

```
[26] pca = create_model('pca')
     print(pca)
```

```
PCA(contamination=0.05, copy=True, iterated_power='auto', n_components=None,
  n_selected_components=None, random_state=123, standardization=True,
  svd_solver='auto', tol=0.0, weighted=True, whiten=False)
```

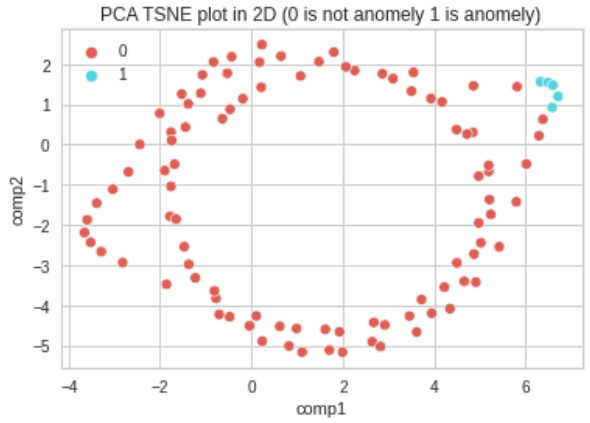
Assign model function assigns anomaly labels to the dataset for a given model and show Two features (Anomaly, Anomaly_score).

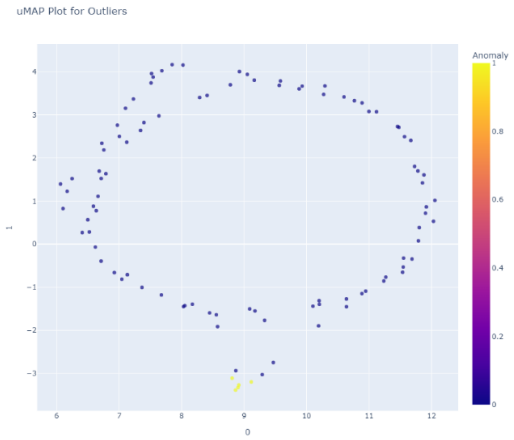
```
pca_model = assign_model(pca)
pca_model.head()
```

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
0	-1.042570	-0.241098	-1.267957	0.414568	0	171.243668
1	-1.056986	-0.245590	-1.165454	0.411869	0	171.205205
2	-1.071858	-0.256787	-1.028780	0.407472	0	171.879672
3	-1.084518	-0.257502	-0.850609	0.367564	0	170.120084
4	-0.974811	-0.105985	-0.625045	0.236174	0	163.779848

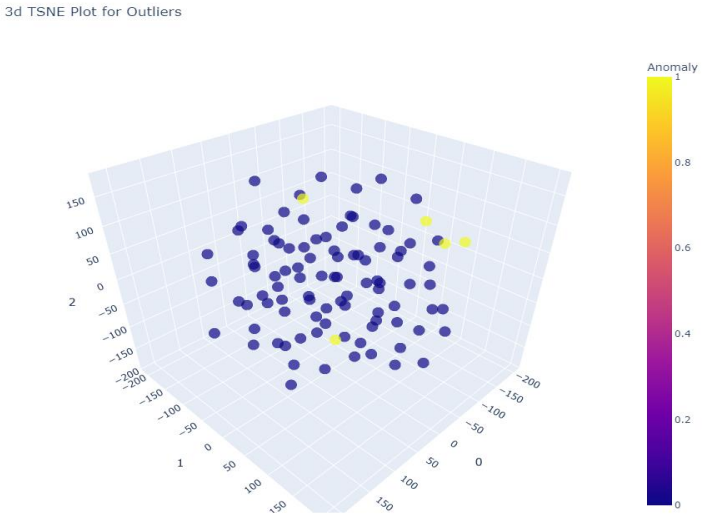
TSNE plot in 2D (0 is not anomaly, 1 is anomaly) and umap:

```
[74] df_4 = pd.DataFrame()
     df_4['comp1'] = z[:,0]
     df_4['comp2'] = z[:,1]
     df_4['target'] = pca_model['Anomaly']
     sns.scatterplot(x="comp1", y="comp2", hue=df_4.target.tolist(),
                     palette=sns.color_palette("hls", 2),
                     data=df_4).set(title="PCA TSNE plot in 2D (0 is not anomely 1 is anomely)")
```





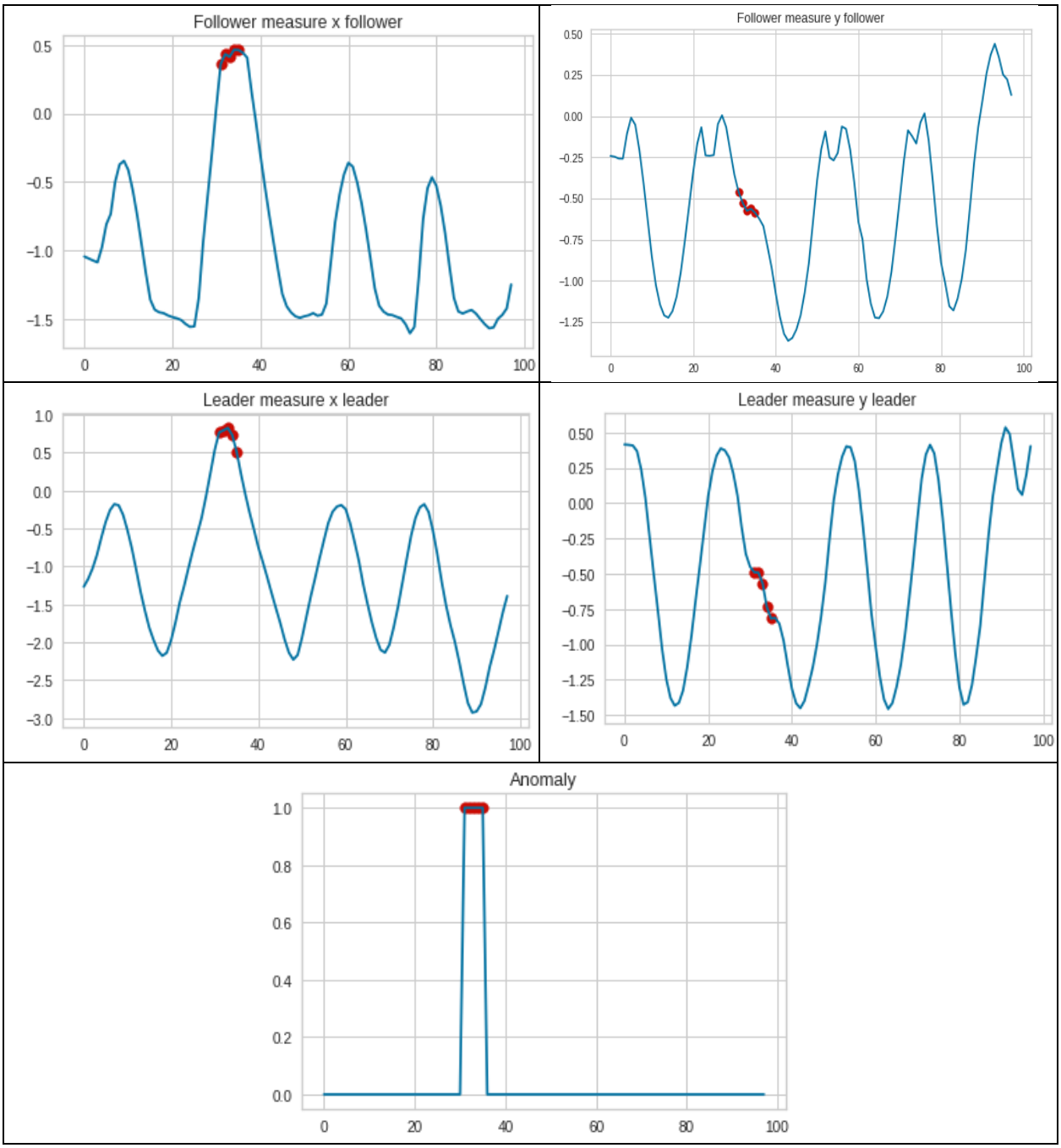
3D TSNE Plot for outliers:



Anomaly data:

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
31	0.360847	-0.459538	0.767382	-0.489506	1	293.785369
32	0.432356	-0.526272	0.795491	-0.493113	1	304.704596
33	0.412900	-0.574391	0.827058	-0.568237	1	305.129763
34	0.463492	-0.562305	0.734001	-0.737032	1	303.068371
35	0.462689	-0.586497	0.517651	-0.817752	1	290.703717

Plot PCA model results along the data:



Model evaluation for PCA:

```
[31] pca_model.head()
```

	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly	Anomaly_Score
0	-1.042570	-0.241098	-1.267957	0.414568	0	171.243668
1	-1.056986	-0.245590	-1.165454	0.411869	0	171.205205
2	-1.071858	-0.256787	-1.028780	0.407472	0	171.879672
3	-1.084518	-0.257502	-0.850609	0.367564	0	170.120084
4	-0.974811	-0.105985	-0.625045	0.236174	0	163.779848

Classification report for PCA:

	precision	recall	f1-score	support
0	1.00	0.92	0.96	93
1	0.42	1.00	0.59	5
accuracy			0.93	98
macro avg	0.71	0.96	0.77	98
weighted avg	0.97	0.93	0.94	98

So, the accuracy of PCA model is 93%

Apply DBSCAN Model:

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

Apply DBSCAN model:

```
[35] from sklearn.cluster import DBSCAN
     dbscan=DBSCAN(eps=0.5,min_samples=7)
     dbscan.fit(data)
     y_pred = dbscan.labels_
```

```
## here 0 is repersent for non anomely and -1 represent for tne anomoly points as nois
y_pred
```

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

To delete the noise mapping -1 to 1 as anomely

```
[37] # here we will map -1 to 1 as anomely
     y_pred = y_pred*-1
```

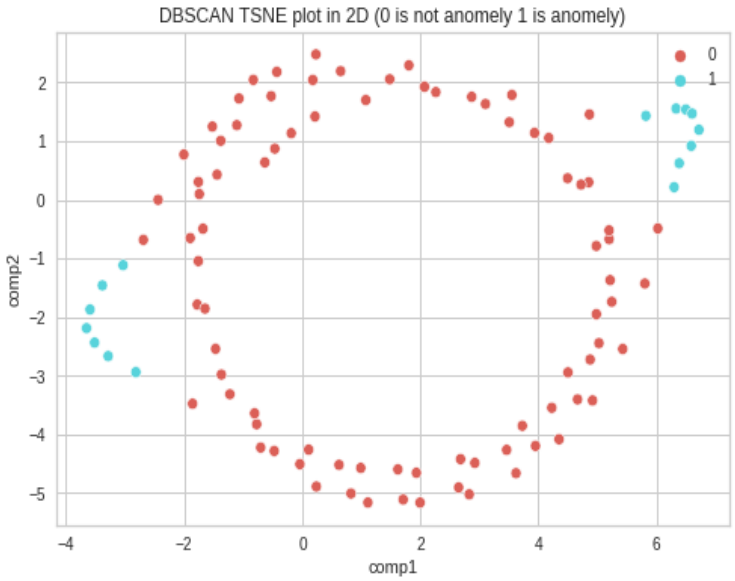
```
[38] y_pred
```

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

TSNE plot in 2D (0 is not anomaly, 1 is anomaly):

```
[39] df_1 = pd.DataFrame()
     df_1['comp1'] = z[:,0]
     df_1['comp2'] = z[:,1]
     df_1['target'] = y_pred
```

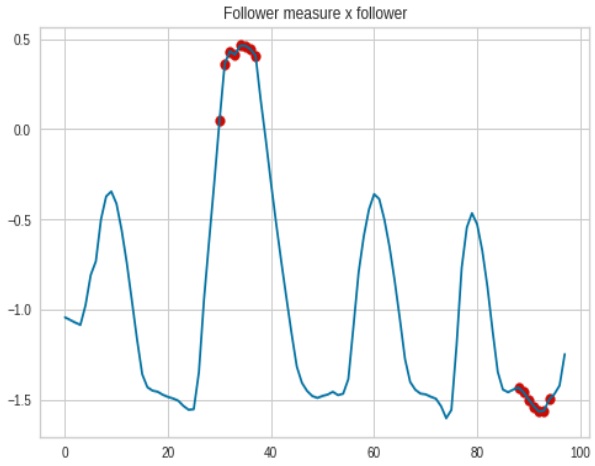
```
[40] sns.scatterplot(x="comp1", y="comp2", hue=df_1.target.tolist(),
                     palette=sns.color_palette("hls", 2),
                     data=df_1).set(title="DBSCAN TSNE plot in 2D (0 is not anomely 1 is anomely)")
```

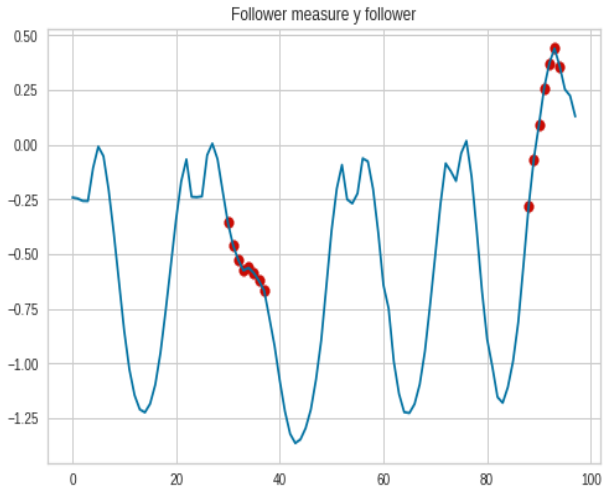


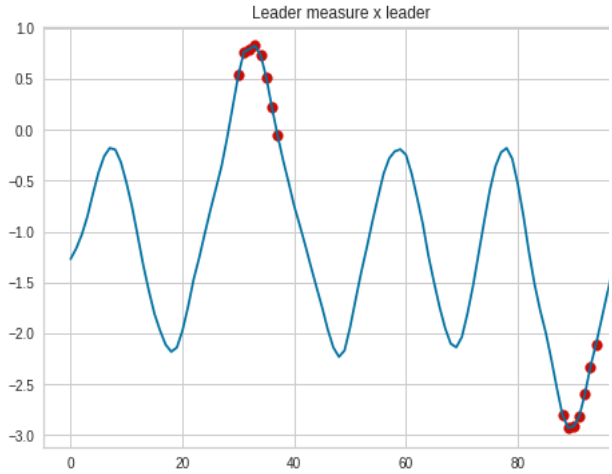
Anomaly points is data:

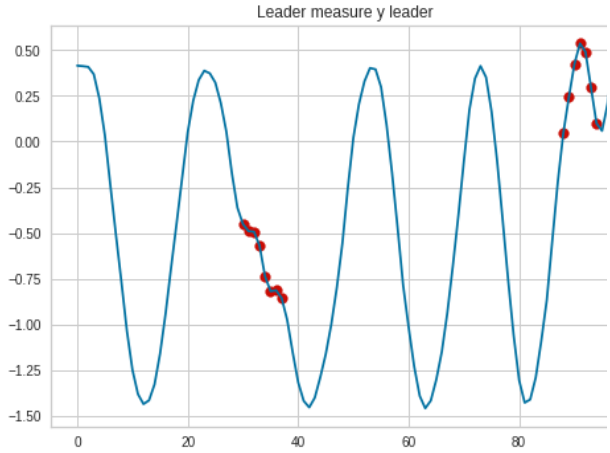
	Follower_measure_x_follower	Follower_measure_y_follower	Leader_measure_x_leader	Leader_measure_y_leader	Anomaly
30	0.046769	-0.356131	0.536808	-0.450804	1
31	0.360847	-0.459538	0.767382	-0.489506	1
32	0.432356	-0.526272	0.795491	-0.493113	1
33	0.412900	-0.574391	0.827058	-0.568237	1
34	0.463492	-0.562305	0.734001	-0.737032	1
35	0.462689	-0.586497	0.517651	-0.817752	1
36	0.443363	-0.622341	0.224326	-0.813084	1
37	0.407215	-0.667258	-0.046350	-0.854807	1
88	-1.431881	-0.282536	-2.802852	0.047603	1
89	-1.458214	-0.064960	-2.928637	0.242762	1
90	-1.498133	0.092471	-2.912866	0.423525	1
91	-1.535425	0.260158	-2.816451	0.535399	1
92	-1.565423	0.372923	-2.596673	0.489932	1
93	-1.559131	0.440215	-2.325538	0.295837	1
94	-1.496434	0.357878	-2.105013	0.098846	1

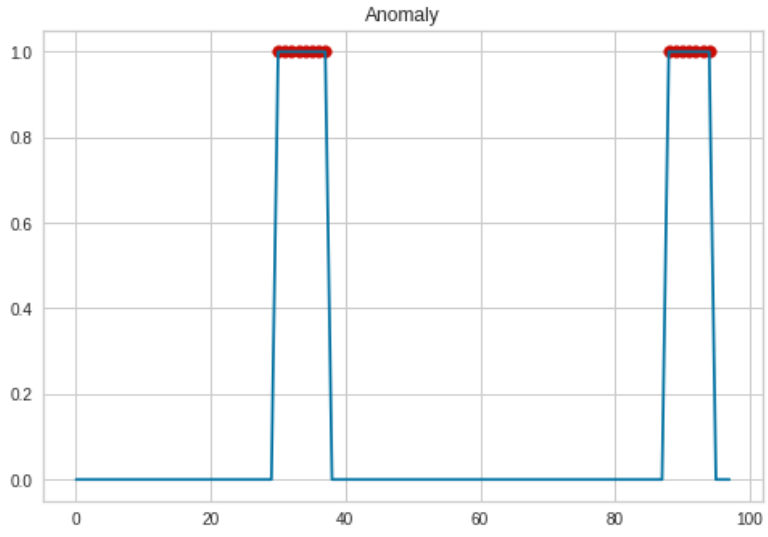
Plot DBSCAN model results along the data:











Classification report for DBSCAN model:

	precision	recall	f1-score	support
0	0.95	0.99	0.97	83
1	0.92	0.73	0.81	15
accuracy			0.95	98
macro avg	0.94	0.86	0.89	98
weighted avg	0.95	0.95	0.95	98

So, the accuracy for DBSCAN model is 95%

Conclusion:

After applying the four models and evaluating them, we notice that the accuracy differs between the 4 models, so the accuracies are shown in the following tables:

Model	Percision 0 label	Recall 0 label	F1 score 0 label	Percision 1 label	Recall 1 label	F1 score 1 label
SVM	1.00	0.92	0.96	0.42	1.00	0.59
KNN	0.99	0.90	0.94	0.25	0.78	0.38
PCA	1.00	0.92	0,96	0.42	1.00	0.59
DBSCAN	0.95	0.99	0.97	0.92	0.73	0.81

Model	Accuracy
SVM	93%
KNN	90%
PCA	93%
DBSCAN	95%

So, the best model to be used in anomaly detection according to the accuracies is the **DBSCAN** model, because it depends on denisty, but it needs to tune epsilon and minpoints.

And the worst model according to the accuracies is **KNN** model.

Referances:

https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html

https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

https://www.freecodecamp.org/news/svm-machine-learning-tutorial-what-is-the-support-vector-machine-algorithm-explained-with-code-examples/