Report Assignment 3

ELG 5142 Ubiquitous Sensing and Smart City

Group Number: G_26

Prepared by:

Sawsan Awad (300327224)

Sondos Ali (300327219)

Toka Mostafa (300327284)

1. Overview

The main objective of this assignment is to detect 2 different anomalies are injected into the trajectory of the QBot, and thus it supposedly reflects onto the trajectory of the QDrone as an anomaly:

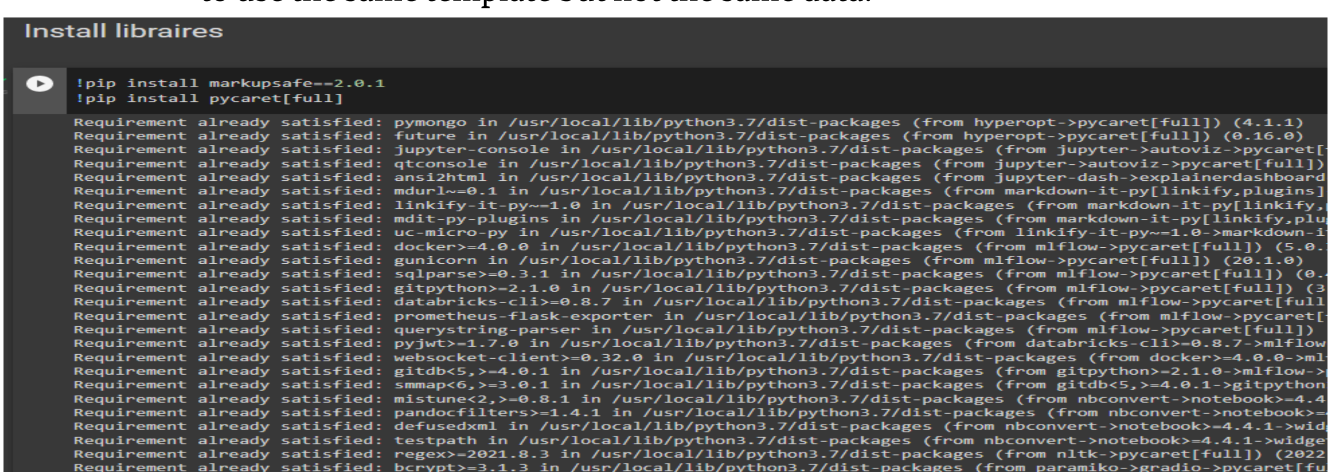
the drone will follow the bot by moving out of the trajectory temporarily, and the bot enters a specific area that is a restricted region for the QDrone. we have a dataset from a 2-minute experiment. We will extract 4 attributes from the dataset: “follower x data”, “follower y data”, “leader x data” and “leader y data” (‘x’ and ‘y’ refers to coordinate). We will implement and compare the performance of different machine learning algorithms which are SVM, PCA, KNN, and DBSCAN. Then We will plot the model results alongside with data and compare unsupervised models with respect to accuracy, precision (for both anomaly and normal instances), recall (for both anomaly and normal instances) and F1 scores (for both anomaly and normal instances).

2. Methodology

We followed some defined steps to obtain the aimed results:

2.1. Install important packages:

- Markupsafe package: MarkupSafe escapes characters so text is safe to use in HTML and XML. Characters that have special meanings are replaced so that they display as the actual characters. This mitigates injection attacks, meaning untrusted user input can safely be displayed on a page.
- Pycaret package: is an open-source, low-code machine learning library in Python that automates machine learning workflows.
- Jinja2: The main idea of Jinja is to separate data and template. This allows you to use the same template but not the same data.



```
Install libraires

!pip install markupsafe==2.0.1
!pip install pycaret[full]
Requirement already satisfied: pymongo in /usr/local/lib/python3.7/dist-packages (from hyperopt->pycaret[full]) (4.1.1)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from hyperopt->pycaret[full]) (0.16.0)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.7/dist-packages (from jupyter->autoviz->pycaret[
Requirement already satisfied: qtconsole in /usr/local/lib/python3.7/dist-packages (from jupyter->autoviz->pycaret[full])
Requirement already satisfied: ansi2html in /usr/local/lib/python3.7/dist-packages (from jupyter-dash->explainerdashboard
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.7/dist-packages (from markdown-it-py[linkify,plugins]
Requirement already satisfied: linkify-it-py~=1.0 in /usr/local/lib/python3.7/dist-packages (from markdown-it-py[linkify,
Requirement already satisfied: mdit-py-plugins in /usr/local/lib/python3.7/dist-packages (from markdown-it-py[linkify,plu
Requirement already satisfied: uc-micro-py in /usr/local/lib/python3.7/dist-packages (from linkify-it-py~=1.0->markdown-i
Requirement already satisfied: docker>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full]) (5.0.
Requirement already satisfied: gunicorn in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full]) (20.1.0)
Requirement already satisfied: sqlparse>=0.3.1 in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full]) (0.
Requirement already satisfied: gitpython>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full]) (3
Requirement already satisfied: databricks-cli>=0.8.7 in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full
Requirement already satisfied: prometheus-flask-exporter in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[
Requirement already satisfied: querystring-parser in /usr/local/lib/python3.7/dist-packages (from mlflow->pycaret[full])
Requirement already satisfied: pyjwt>=1.7.0 in /usr/local/lib/python3.7/dist-packages (from databricks-cli>=0.8.7->mlflow
Requirement already satisfied: websocket-client>=0.32.0 in /usr/local/lib/python3.7/dist-packages (from docker>=4.0.0->ml
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.7/dist-packages (from gitpython>=2.1.0->mlflow->
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.7/dist-packages (from gitdb<5,>=4.0.1->gitpython
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert->notebook>=4.4
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7/dist-packages (from nbconvert->notebook>=
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-packages (from nbconvert->notebook>=4.4.1->wid
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-packages (from nbconvert->notebook>=4.4.1->widge
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-packages (from nltk->pycaret[full]) (2022
Requirement already satisfied: bcrypt>=3.1.3 in /usr/local/lib/python3.7/dist-packages (from paramiko->gradio->pycaret[fu
```

```
[2] !pip uninstall Jinja2 --yes
    !pip install Jinja2

    Found existing installation: Jinja2 2.11.3
    Uninstalling Jinja2-2.11.3:
      Successfully uninstalled Jinja2-2.11.3
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting Jinja2
      Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
    Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.7/dist-packages (from Jinja2) (2.1.1)
    Installing collected packages: Jinja2
```

2.2.Importing important libraries:

- NumPy library: it provides a lot of supporting functions that make working with ndarray very easy.
- Pandas library: it helps us to analyze and understand data better.
- Matplotlib.pyplot library: used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc.
- Seaborn library: is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.
- TSNE: is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.
- IPython.core.pylabtools import figsize: a tuple of ints giving the figure numbers of the figures to return.
- pycaret.utils: is a python open-source machine learning library with the aim of using low code and a low number of hypotheses for insights within a cycle of machine learning experimentation and development. Using this library we can perform end-to-end machine learning experiments efficiently without consuming so much time.
- DBSCAN: is a density-based clustering algorithm that works on the assumption that clusters are dense regions in space separated by regions of lower density. It groups 'densely grouped' data points into a single cluster.
- Tqdm: uses smart algorithms to predict the remaining time and to skip unnecessary iteration displays, which allows for a negligible overhead in most cases.
- from sklearn.metrics import classification_report, accuracy_score:

- Classification_report: is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of your trained classification model, and it will return accuracy.
- The accuracy_score: is function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.

- Other libraries will be shown their importance in the code.

Importing the libraries

```
[3] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.manifold import TSNE
    from IPython.core.pylabtools import figsize
    from pycaret.utils import enable_colab
    enable_colab()
    from pycaret.anomaly import *
    from sklearn.cluster import DBSCAN
    from tqdm import tqdm
    from sklearn.metrics import classification_report
    %matplotlib inline
```

2.3. Importing dataset:

- First, we use read the first dataset which we will use it to predict anomalies.
- Second, we use .head() function to display the first five rows of the data frame by default.

Importng the *dataset*

```
[4] anomaly_detection = pd.read_csv('Dataset_to_be_used_in_anomaly_detection.csv')
    anomaly_detection.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

- Third, we use.drop() function to drop the first column which called Unnamed because it likes id, we won't benefit from it.

```
[5] anomaly_detection.columns

    Index(['Unnamed: 0', 'Follower_measure_x_follower',
           'Follower_measure_y_follower', 'Leader_measure_x_leader',
           'Leader_measure_y_leader'],
          dtype='object')

[6] anomaly_detection.drop('Unnamed: 0' ,axis =1, inplace=True)
    anomaly_detection.head()
```

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

2.4. Modeling:

- First: Setup initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function. It takes one mandatory parameter: data. All the other parameters are optional.
- Second: we make model function (to generalize), we can create and plot any model by it.
    - SVM: is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.
    - KNeighborsClassifier: is used to implement classification based on voting by nearest k-neighbors of target point, t, while RadiusNeighborsClassifier implements classification based on all neighborhood points within a fixed radius, r, of target point, t.
    - PCA Principal Component Analysis (PCA): is a technique that comes from the field of linear algebra and can be used as a data preparation technique to create a projection of a dataset prior to fitting a model.
    - DBSCAN: (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning technique used to identify clusters of varying shape in a data set.

- We initialize DBSCAN with our values for epsilon and minpoints:
    - ε (epsilon or “eps”): the maximum distance two points can be from one another while still belonging to the same cluster.
    - Minimum samples (“MinPoints”): indicates the minimum number of samples that should be within the epsilon range.
- We tried values in eps and minpoints which brought 2 clusters:
    - As the original data contains labels 1 (anomalies), 0 (the correct path) and the output is:
        - ✓ 0: is the correct path & -1 is the anomalies. So, we converted -1 to 1 to refer to anomalies.

Modeling

```
[7]  s = setup(anomaly_detection, 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	f9ee
14	Imputation Type	simple
15	Iterative Imputation Iteration	None
16	Numeric Imputer	mean
17	Iterative Imputation Numeric Model	None
18	Categorical Imputer	mode
19	Iterative Imputation Categorical Model	None
20	Unknown Categoricals Handling	least_frequent

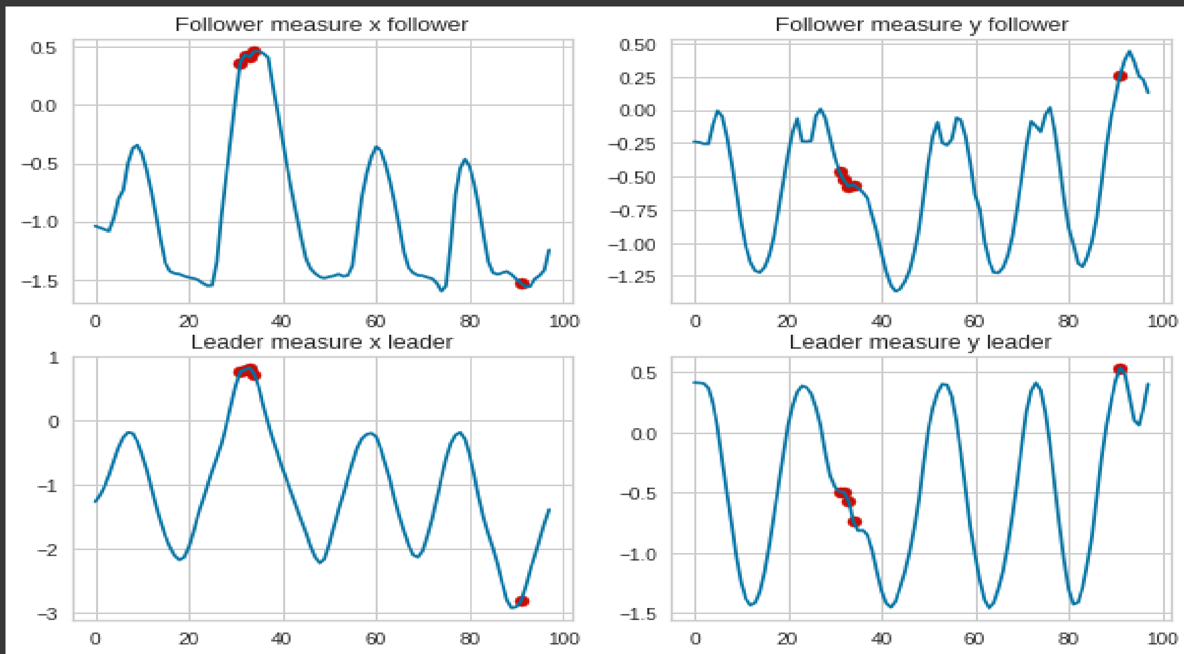
```
[8] models()
```

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

```
[9] def models(model):
      ml_model = create_model(model,fraction=0.05)
      ml_model_results = assign_model(ml_model)
      anomalies = ml_model_results[ml_model_results['Anomaly'] == 1]
      results = ml_model_results.iloc[:,:-2]
      c = 1
      figsize(10,7)
      for column in results.columns:
        plt.subplot(2,2,c)
        plt.plot(ml_model_results[column])
        plt.scatter(anomalies.index,anomalies[column],c = 'r', marker ='o', s = 50)
        plt.title(" ".join(column.split('_')))
        c = c+1
      return anomalies, ml_model_results
```

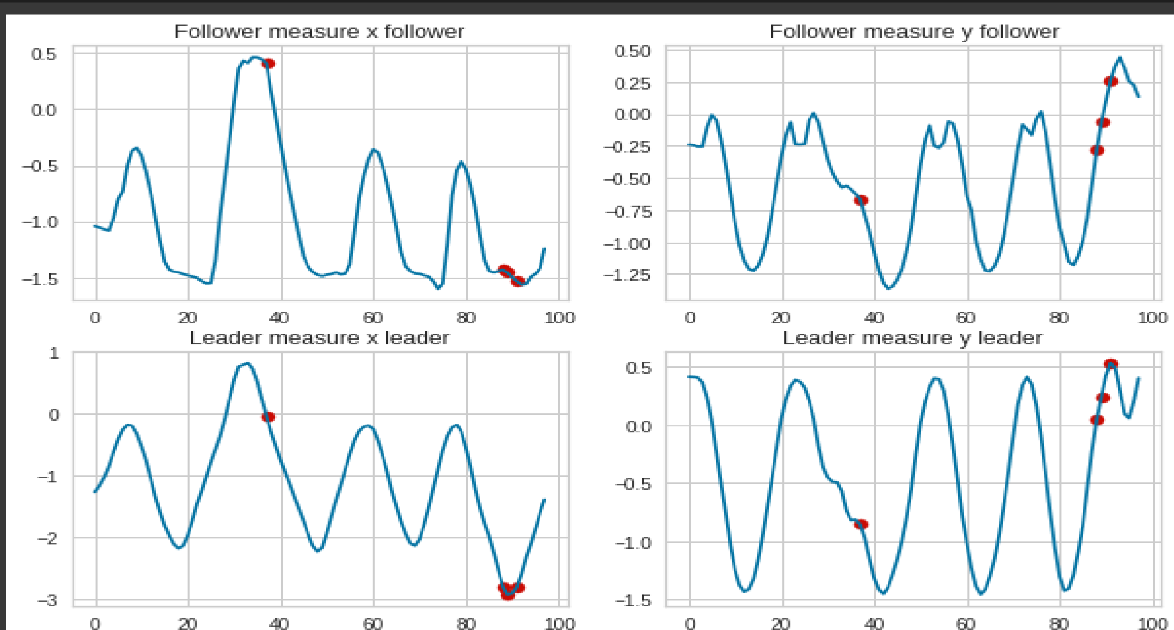
1. SVM

```
[10] anomalies_svm, svm_model_results = models('svm')
```



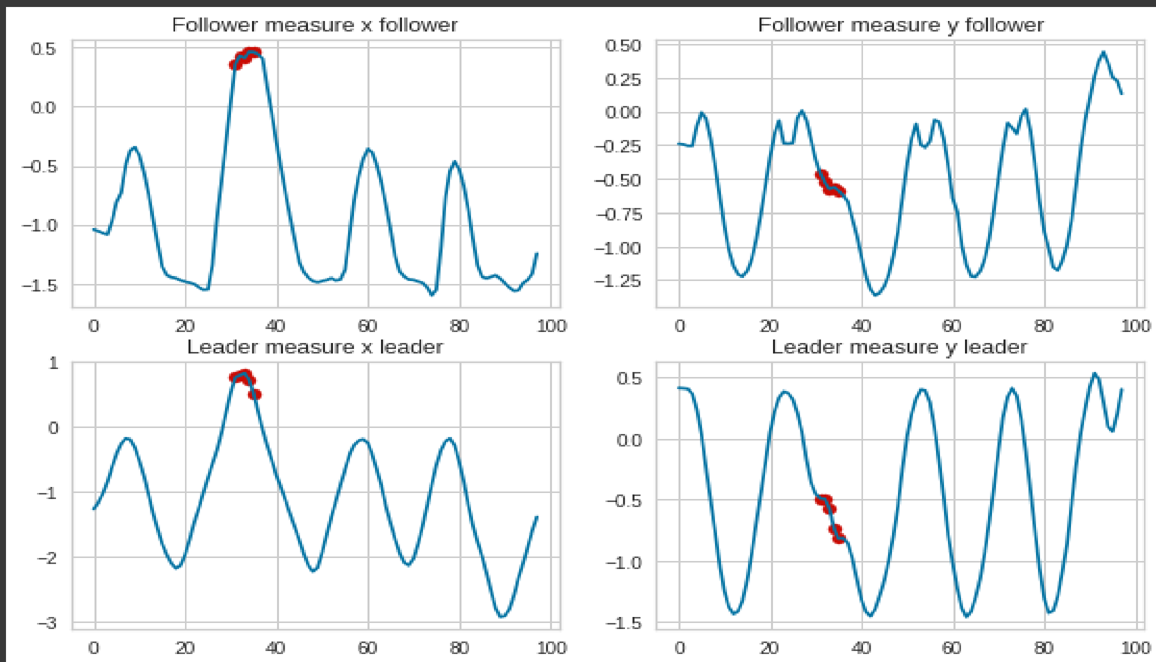
2. KNN

```
[11] anomalies_KNN, knn_model_results = models('knn')
```



3. PCA

```
[12] anomalies_PCA, pca_model_results = models('pca')
```



4. DBSCAN

```
[13] model = DBSCAN(eps=0.5, min_samples=7)
     predLabels = model.fit_predict(anomaly_detection)
     predLabels
```

```
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])
```

```
[14] # replace -1 to 1
     Dbscan=np.where(predLabels==-1,1,predLabels)
     Dbscan
```

```
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])
```

2.5. TSNE Plot:

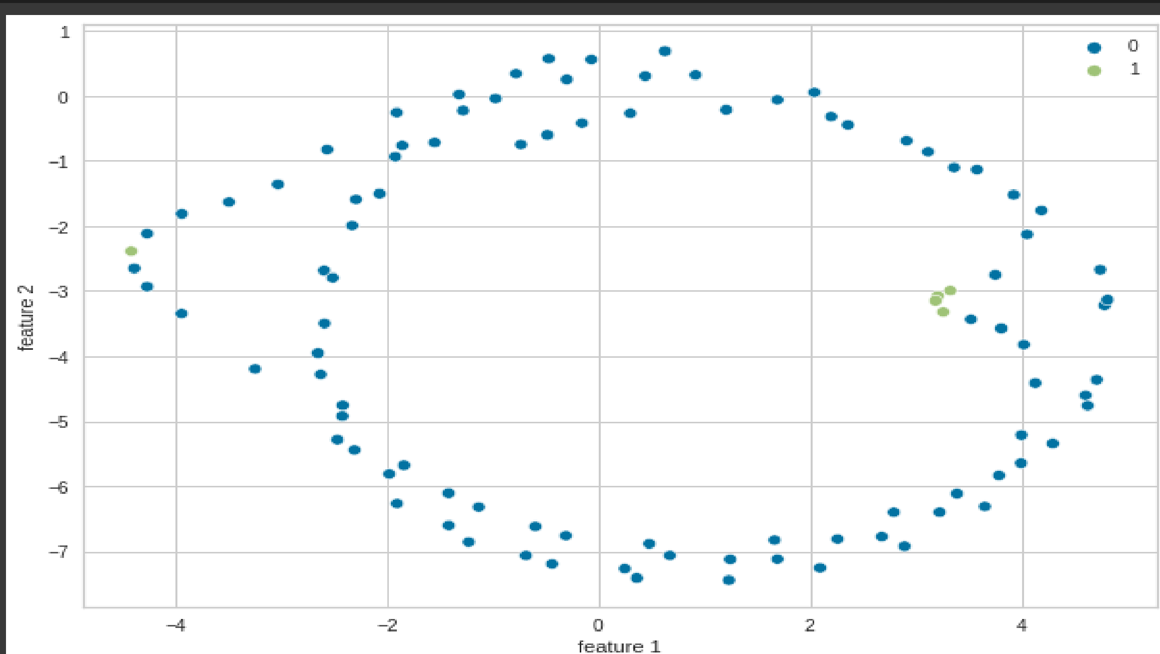
- T-distributed Stochastic Neighbor Embedding (TSNE): is a tool for visualizing high-dimensional data. T-SNE, based on stochastic neighbor embedding, is a nonlinear dimensionality reduction technique to visualize data in a two- or three-dimensional space.
- Second: we make tsne function (to generalize), we can apply it in many models which we will use and plot them.
- Third: we apply tsne in all model which we used and plot them:

TSNE plot

```
[15] def tsne(x,y):
       Tsne = TSNE(n_components = 2)
       tsne_results = Tsne.fit_transform(x)
       df = pd.DataFrame()
       df['feature 1'] = tsne_results[:,0]
       df['feature 2'] = tsne_results[:,1]
       df['y'] = y
       sns.scatterplot(x = 'feature 1', y = 'feature 2', hue = df.y.tolist(), data = df)
```

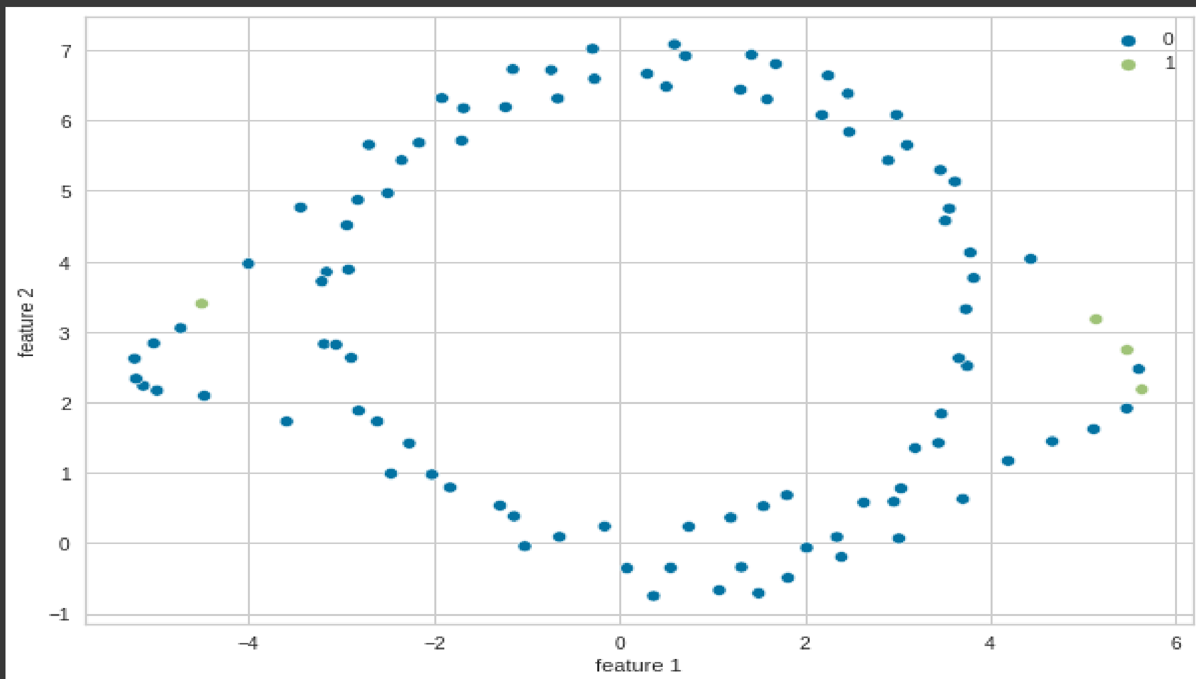
  - First model SVM:

```
[16] tsne(svm_model_results.iloc[:,:-2],svm_model_results.iloc[:,-2])
```



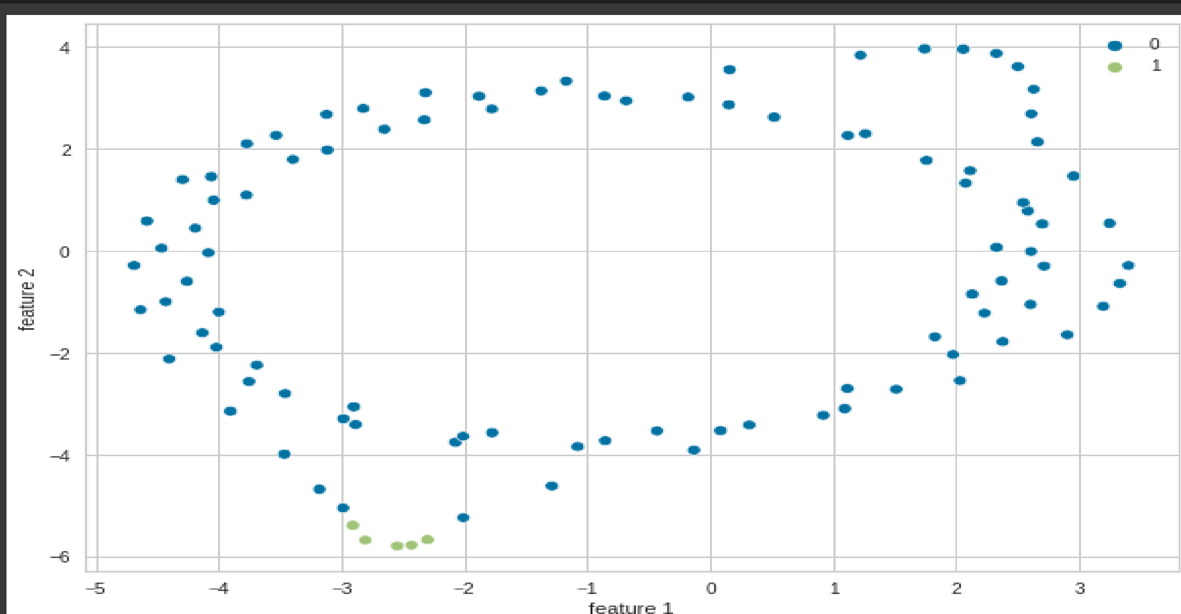
- Second model KNN:

```
[17] tsne(knn_model_results.iloc[:,:-2], knn_model_results.iloc[:,-2])
```



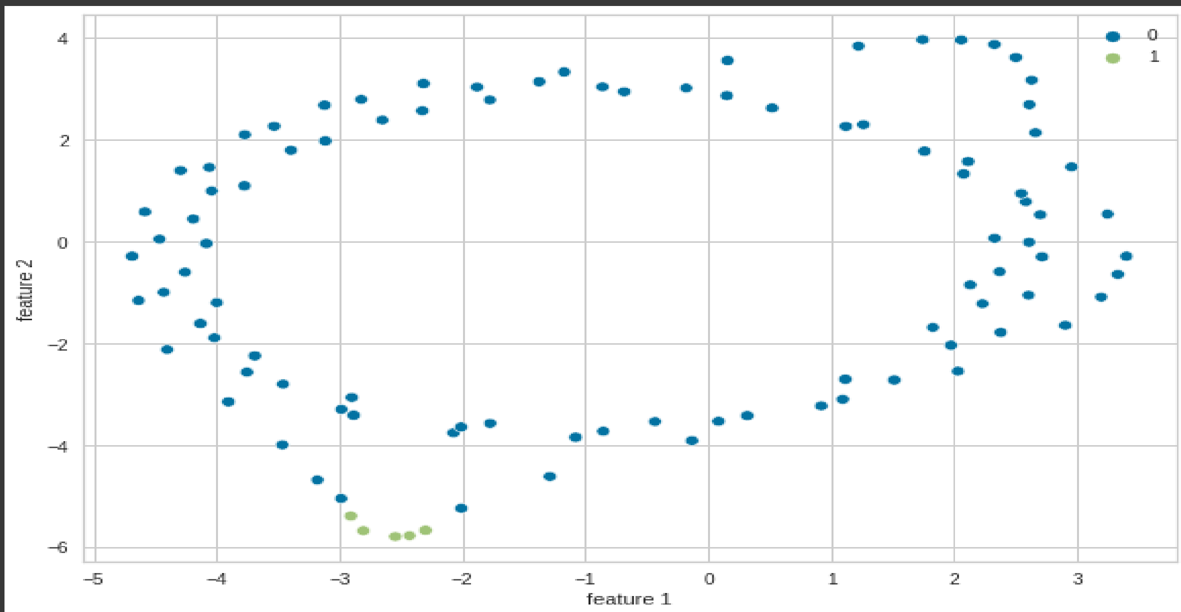
- Third model PCA:

```
[18] tsne(pca_model_results.iloc[:,:-2],pca_model_results.iloc[:,-2])
```

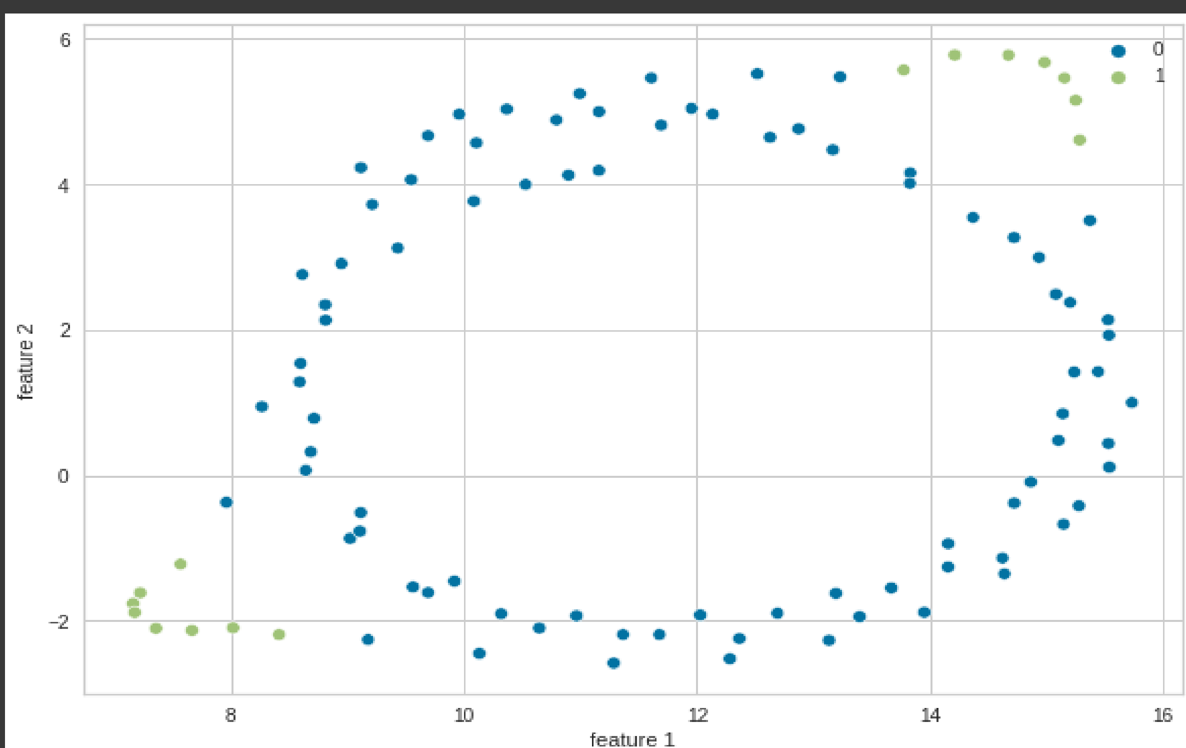


- Third model PCA & DBSCAN:

```
[18] tsne(pca_model_results.iloc[:,:-2],pca_model_results.iloc[:,-2])
```




```
[19] tsne(pca_model_results.iloc[:,:-2],Dbscan)
```



2.6. Evaluation:

- First, we use read the second dataset to evaluate the output of the first dataset.
- Second, we use .head() function to display the first five rows of the data frame by default.
- Third, We use the actual output from the labeled data to compare the predicted output with the actual output
- Forth, We use classification report to evaluate the data.

Evaluation

```
[20] preformance = pd.read_csv('/content/Dataset_to_be_used_in_performance_comparison.csv')
     preformance.head()
```

	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

```
[21] true_label = preformance.iloc[:,-1]
     svm_label = svm_model_results.iloc[:,-2]
     knn_label = knn_model_results.iloc[:,-2]
     pca_label = pca_model_results.iloc[:,-2]
```

```
[22] cr_svm = classification_report(true_label,svm_label)
     print(cr_svm)

              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96        86
         1.0       1.00      0.42      0.59        12

    accuracy                           0.93        98
   macro avg       0.96      0.71      0.77        98
weighted avg       0.93      0.93      0.92        98
```

```
[23] cr_knn = classification_report(true_label,knn_label)
     print(cr_knn)

                  precision    recall  f1-score   support

             0.0       0.90      0.99      0.94        86
             1.0       0.75      0.25      0.38        12

        accuracy                           0.90        98
       macro avg       0.83      0.62      0.66        98
    weighted avg       0.89      0.90      0.87        98
```

```
[24] cr_pca = classification_report(true_label,pca_label)
     print(cr_pca)

                  precision    recall  f1-score   support

             0.0       0.92      1.00      0.96        86
             1.0       1.00      0.42      0.59        12

        accuracy                           0.93        98
       macro avg       0.96      0.71      0.77        98
    weighted avg       0.93      0.93      0.92        98
```

```
[25] cr_DB= classification_report(true_label,Dbscan)
     print(cr_DB)

                  precision    recall  f1-score   support

             0.0       0.99      0.95      0.97        86
             1.0       0.73      0.92      0.81        12

        accuracy                           0.95        98
       macro avg       0.86      0.94      0.89        98
    weighted avg       0.96      0.95      0.95        98
```

Conclusion:

In this project, we need to detect the anomalies injected into the trajectory of the bot, and thus, it supposedly reflects onto the trajectory of the drone. Firstly, we implement four models which are SVM, PCA, KNN, and DBSCAN to compare between them. After that, we plotted the TSNE plot to visualize the result of Qbot and Qdrone in each model. Then we compute the evaluation of each model using the true label found in Dataset_to_be_used_in_performance_comparison.csv. the results showed that the DBSCAN model produce a better accuracy score with 95%. PCA & SVM give an accuracy score of 93%. Finally, KNN produce the worst accuracy score with 90%.