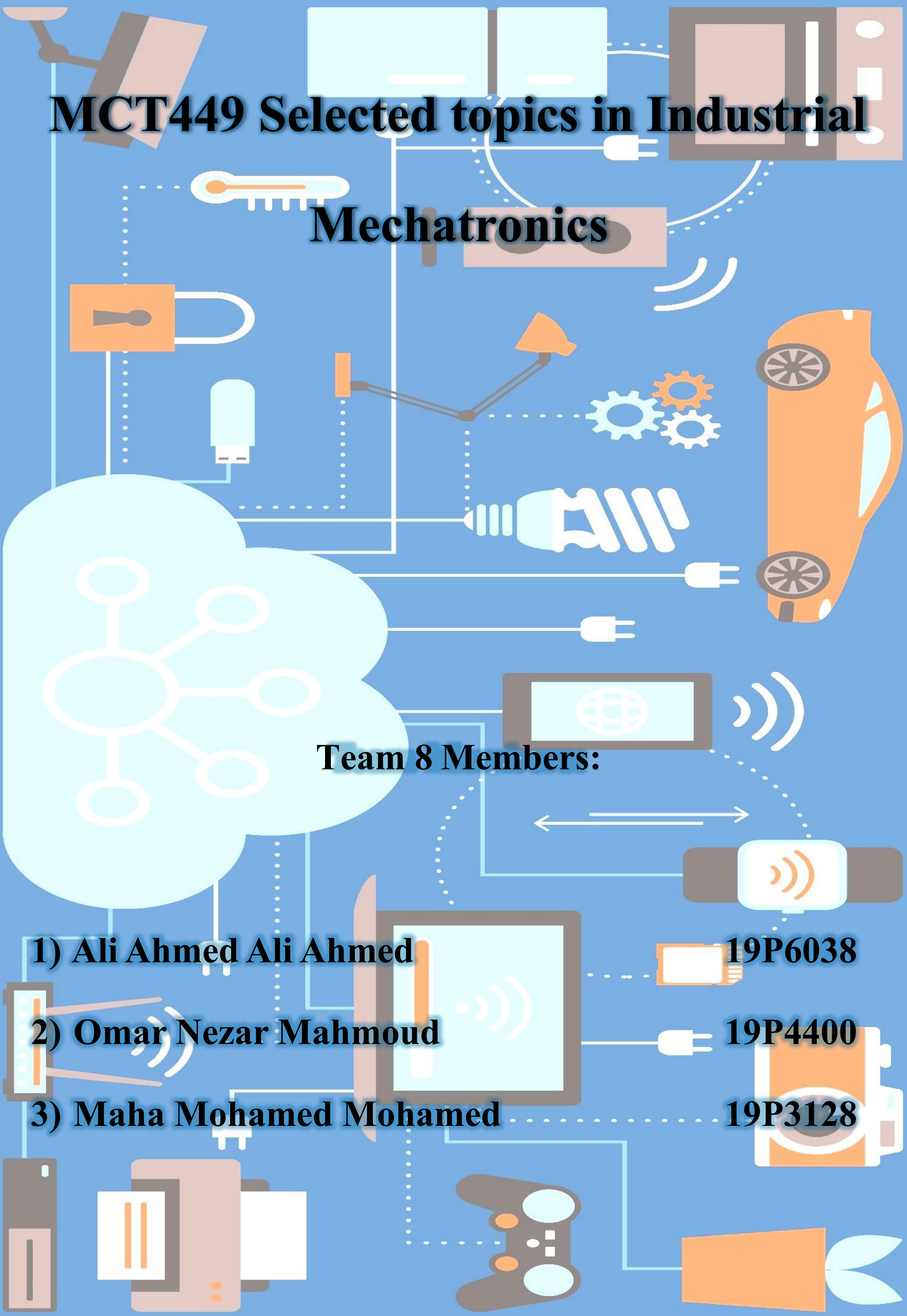


MCT449 Selected topics in Industrial Mechatronics

Team 8 Members:

1) Ali Ahmed Ali Ahmed	19P6038
2) Omar Nezar Mahmoud	19P4400
3) Maha Mohamed Mohamed	19P3128

Table of Contents

List of Figures and Tables

INTRODUCTION

Smart city is an urban area that utilizes advanced technologies to make life easier for its citizens. Smart cities focus on improving the quality of services provided to individuals through the management of public resources, convenience, maintenance, and sustainability. They can overcome issues related to the fields of health, education, environment, governance, economic, and transportation. By 2025, it is expected that there will be 88 smart cities around the world. Based on the global smart cities index, the top ten smart cities in terms of smart infrastructure, economy, and governance are London, New York, Paris, Berlin, Tokyo, Los Angeles, Singapore, Seoul, Chicago, and Hong Kong

One of the steps to build a smart city is to make the vehicles in it smart too. And this can be done by building a reliable V2X communication, which means that vehicles can communicate with everything around them allowing them to send and receive data that helps vehicles to make a decision.

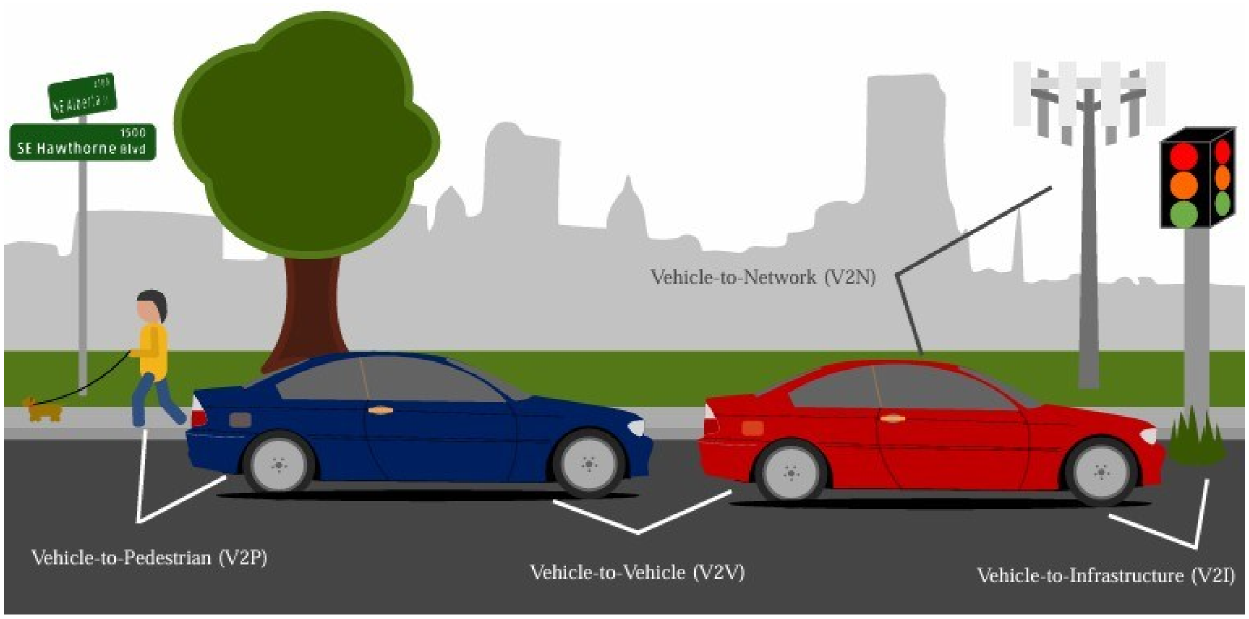


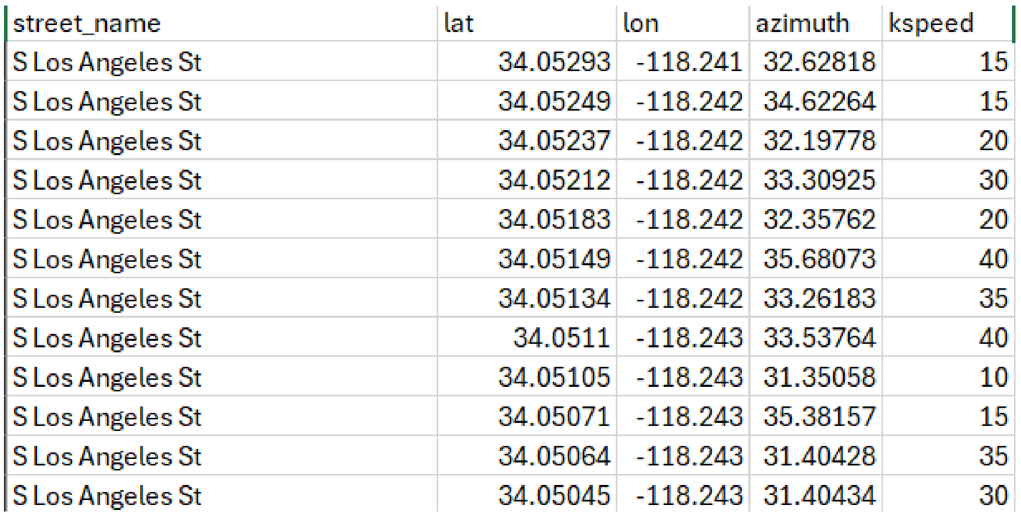
Figure 1: Examples of Vehicle to things communication.

PROBLEM

The main problem or in other words; the main purpose is to allow vehicles to send and receive data from things around it, for example, sending and receiving data from and to base stations in order to manage and organize the vehicles movement in streets (autonomous vehicles).
One of the very useful applications of autonomous driving is the intersection management applications. Which helps in accelerating reaching destinations without stopping.
Benefits of Intersection Management Systems:

- Reduced congestion and travel times: Optimized traffic flow and fewer red lights can lead to faster commutes.
- Improved safety: Collision avoidance systems and better traffic management can decrease accidents.
- Lower emissions: Reduced idling and smoother traffic flow can contribute to lower fuel consumption and emissions.
- Increased efficiency: Dynamic systems can respond to changing traffic conditions, improving overall network performance.

DATASET



street_name	lat	lon	azimuth	kspeed
S Los Angeles St	34.05293	-118.241	32.62818	15
S Los Angeles St	34.05249	-118.242	34.62264	15
S Los Angeles St	34.05237	-118.242	32.19778	20
S Los Angeles St	34.05212	-118.242	33.30925	30
S Los Angeles St	34.05183	-118.242	32.35762	20
S Los Angeles St	34.05149	-118.242	35.68073	40
S Los Angeles St	34.05134	-118.242	33.26183	35
S Los Angeles St	34.0511	-118.243	33.53764	40
S Los Angeles St	34.05105	-118.243	31.35058	10
S Los Angeles St	34.05071	-118.243	35.38157	15
S Los Angeles St	34.05064	-118.243	31.40428	35
S Los Angeles St	34.05045	-118.243	31.40434	30

Figure 2: Dataset.

Dataset Elements:

1. STREET_NAME: The name of the Los Angeles Street where the vehicle sample is located.
2. LAT: The latitude coordinate of the vehicle, which indicates the north-south position of the vehicle on the Earth's surface.
3. LON: The longitude coordinate of the vehicle, which indicates the east-west position of the vehicle.
4. AZIMUTH: The azimuth of the vehicle, which refers to the angle between the vehicle direction and the north, in degrees.
5. KSPEED: The speed of the vehicle in kilometers per hour (km/h).

DataSet Link: https://github.com/Ibtihal-Alablani/Vehicle-Dataset-in-Los-Angeles

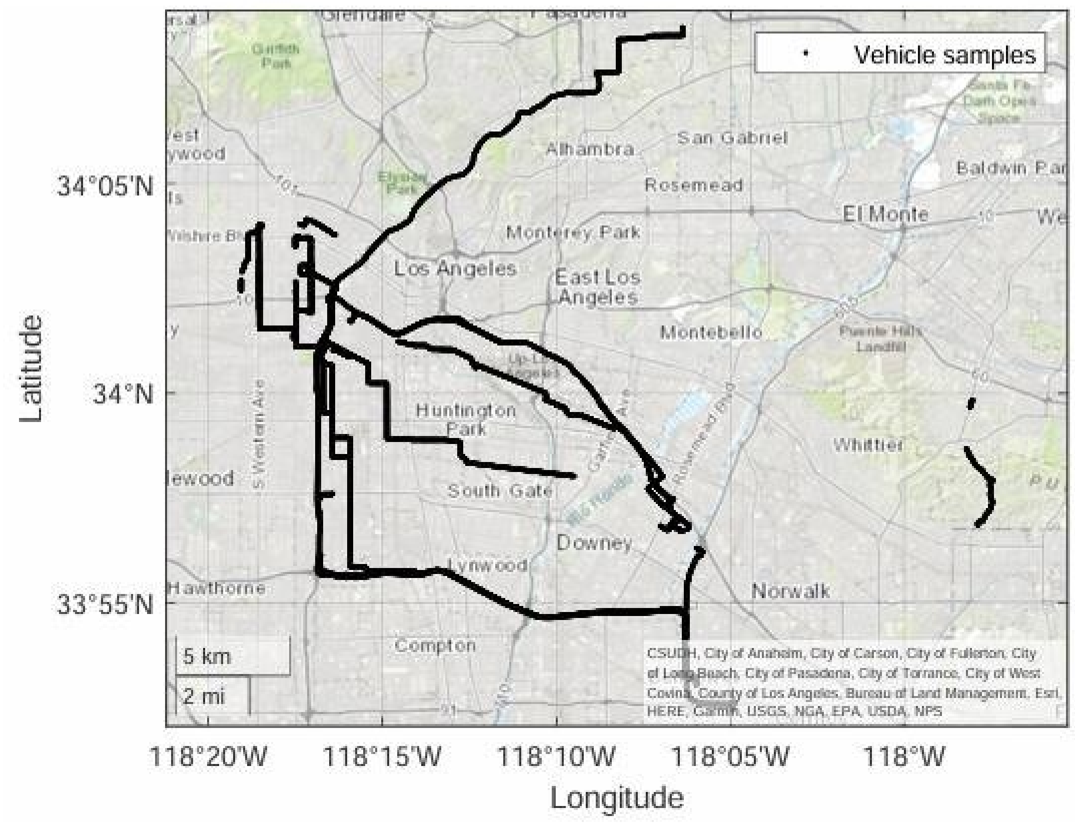


Figure 3: Illustration on the LA map of vehicle samples.

Dataset Generation Method:

A real vehicle dataset in the city of Los Angeles is proposed. The VehDS LA was generated by utilizing Google Maps and the MATLAB R2021b simulator. The database production process is divided into two main phases:

Phase1: Creating Driving Routes: This phase was implemented through Google Maps. It includes three steps: –

Step 1: Creating a new map of the city of Los Angeles.

Step 2: Adding driving routes for all the selected streets (15 streets in this study).

Step 3: Exporting a Keyhole Markup Language (KMZ) file for each driving route. An example of the contents of a KMZ file.

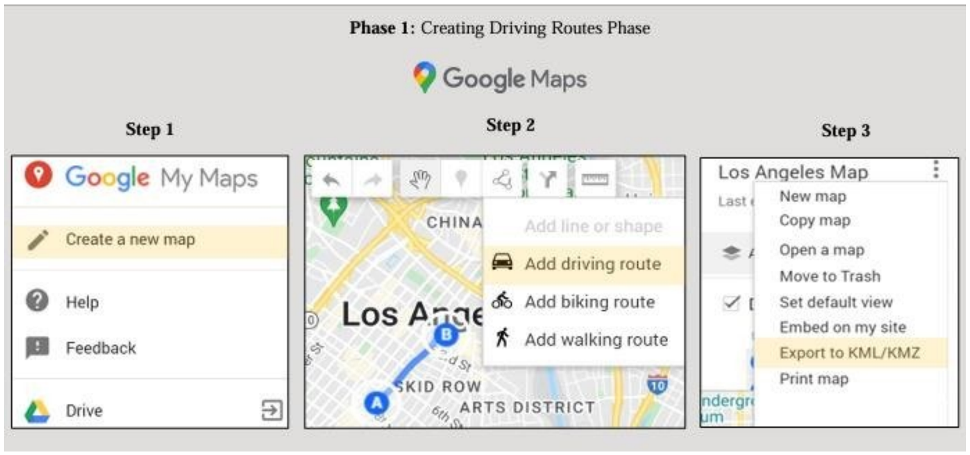


Figure 4: Creating Driving Routes

Phase2: Generating the Vehicle Dataset: This phase was performed using the MATLAB simulator. This phase has three steps: –

Step 1: Reading the KMZ files and converting them into structure objects.

Step 2: Generating extra vehicle samples so that the distance between two samples is small (0.25 m in this study). For each vehicle sample, four features were assigned: (1) latitude coordinate, (2) longitude coordinate, (3) vehicle speed, and (4) vehicle azimuth. The speeds were generated randomly in the range from 10 to 40 km per hour (km/h).

Step 3: Exporting the proposed VehDS-LA as a comma-separated values (CSV)file.

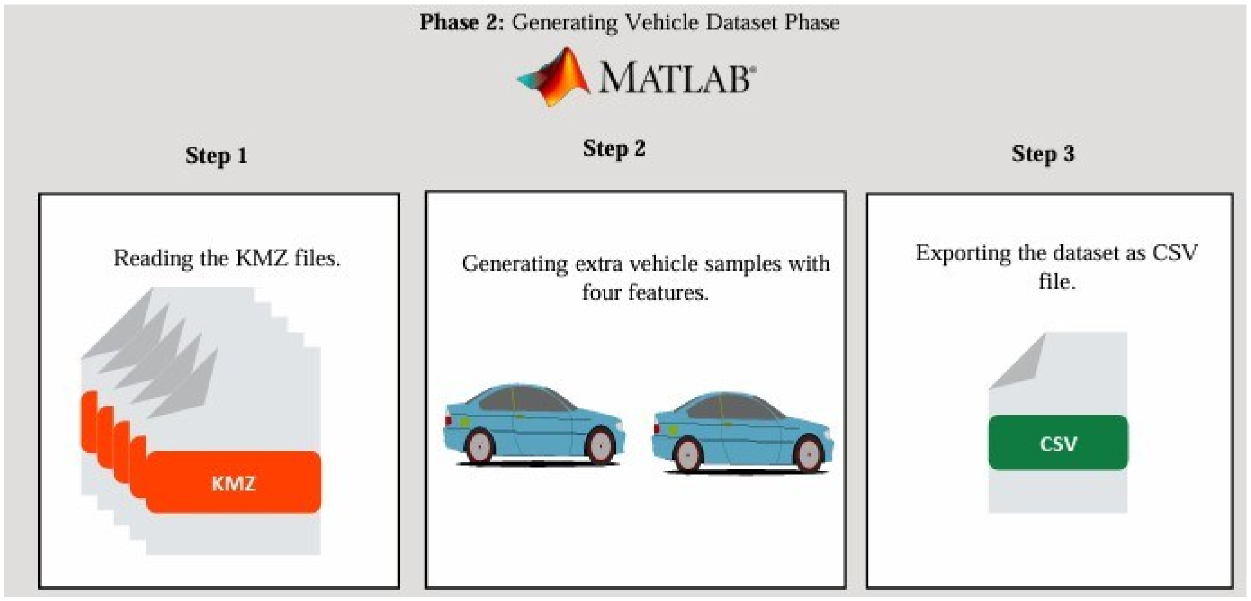

Phase 2: Generating Vehicle Dataset Phase
MATLAB
Step 1
Step 2
Step 3
Reading the KMZ files.
KMZ
Generating extra vehicle samples with four features.
Exporting the dataset as CSV file.
CSV

Figure 5: Generating the dataset.

The Advantages of the Proposed Dataset

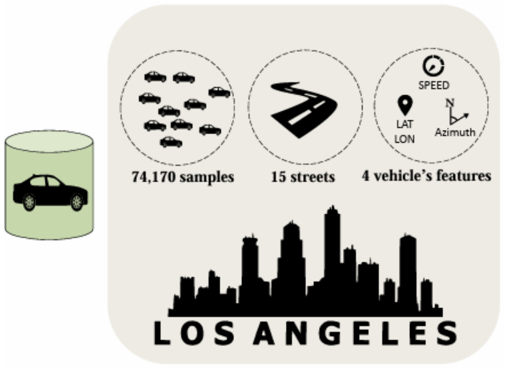


Figure 6: Los Angeles area statistics

- Generating the database does not require a long time, as in the related works, where it took days and months.
- The accuracy of the positions of vehicle samples which were produced based on Google Maps and the MATLAB simulator. It was verified that the samples are located on the LA streets without any deviation.

• Thereisnoneedtoinstall special equipment and devices in the vehicle, such as a GPS receiver, small computer, or smartphone.

• The number of dataset samples is large, and each sample has four features, which are the most important features of a vehicle for traffic simulation purposes.

• The method of generating the proposed VehDS-LA dataset introduces a general mechanism that can be followed in generating new databases in any region of the world on the basis of Google Maps.

ML-based vehicle mobility studies: ML techniques provide remarkable opportunities in several fields, including transportation. A good machine learning model needs a large number of samples to train the ML model. Recent works that focus on vehicle movement issues relied on solving research problems using machine learning algorithms, such as artificial neural networks (ANN) and support vector machine (SVM), Naive Bayes (NB), and Tree-based techniques. Figure 16 represents the process of building a machine learning model that is based on supervised learning to solve a vehicle mobility issue. The building process passes through many phases: data cleaning, data labeling, data dividing, ML model training, and ML model testing.

- Data cleaning: In this phase, data that will not be used to solve the research problem are removed.
- Data Labeling: This refers to the process of tagging vehicle samples so that the ML model can learn from it.
- Data Dividing: This refers to splitting the dataset into two parts: training and testing sets. The dataset is usually divided into 80:20 or 70:30 ratios.
- ML Model Training: The training set is used to train the ML model.
- ML Model Testing: The test set is used to evaluate the performance of the trained ML model.

Research that is based on solving vehicle mobility problems using ML algorithms can utilize the proposed database. It provides a sufficient number of vehicle samples, i.e., 74,170 samples, that can be used for ML model training and testing. Moreover, the accuracy of the locations of vehicle samples was verified without any deviation.

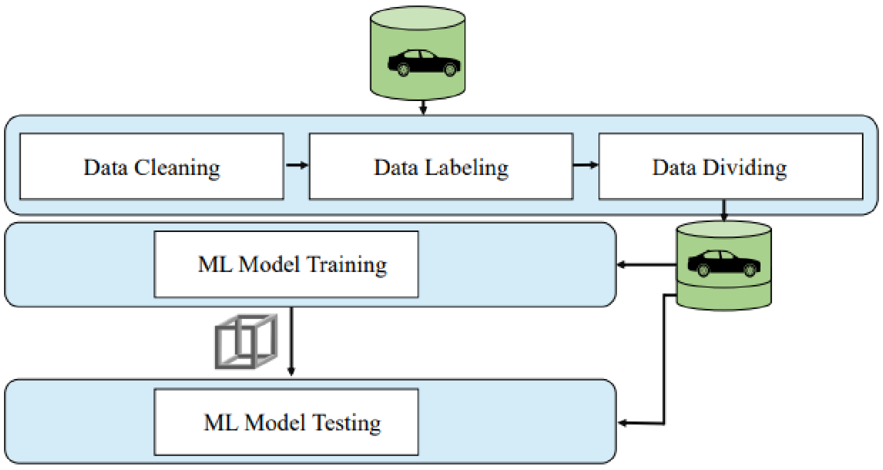


Figure 7: Building Machine Learning Model

Conclusion

After Multiple Research we found that although this dataset would be very suitable for Machine learning Applications, most people focused on using this dataset for studying the V2X communications and obtain results of the Handover that Occurs between base stations and Average Sojourn Time under various speeds and study the relation between then which is out of our scope. So we decided to track another Dataset that can be used for ML applications and some people had already obtained some results from it.

ACCELEROMETER DATA SET

Introduction

This dataset was generated for use on 'Prediction of Motor Failure Time Using An Artificial Neural Network' project (DOI: 10.3390/s19194342). A cooler fan with weights on its blades was used to generate vibrations. To this fan cooler was attached an accelerometer to collect the vibration data. With this data, motor failure time predictions were made, using an artificial neural networks. To generate three distinct vibration scenarios, the weights were distributed in three different ways: 1) 'red' - normal configuration: two weight pieces positioned on neighboring blades; 2) 'blue' - perpendicular configuration: two weight pieces positioned on blades forming a 90Â° angle; 3) 'green' - opposite configuration: two weight pieces positioned on opposite blades.

Data Collection and Data Preprocessing

The device model used in this work, to collect data, was comprised of a computer cooling fan with small magnets fixed to its blades. Adding a second magnet to certain blades created a weight difference between those with only one magnet and those with two magnets. This weight difference generated vibrations during the rotation of the fan's motor, allowing the vibration to be controlled in order to generate different vibration scenarios. A microcontroller, the Arduino UNO, was responsible for setting the motor speeds and performing the readings of the vibration values from the accelerometer. An Akasa AK-FN059 12cm Viper cooling fan was used in the construction of the device model and an MMA8452Q accelerometer was used to measure vibration, attached to the cooling fan. This accelerometer has 12 bits of resolution and communicates with the microcontroller through the I 2C (Inter-Integrated Circuit) protocol. It was developed a software using Processing program language to collect the data from the serial port and store it in a text file. Figure 7 shows the device model developed to simulate motor vibrations.

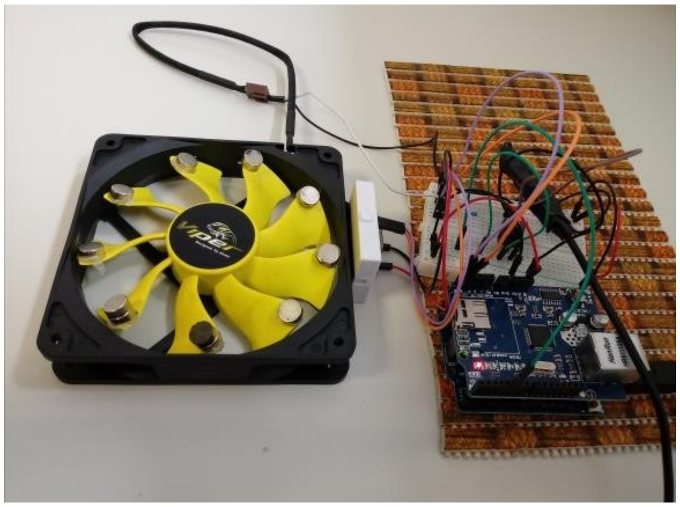


Figure 8: Device Model

In order for the training dataset to cover different levels of vibration, three weight distribution configurations were done in the cooler blades. Figure 8 presents these configurations, where the color pairs represent the position where the weights were doubled to generate different vibration behaviors. For each of these configurations, 17 rotation speeds were set up, ranging from 20% to 100% of the cooler maximum speed at 5% intervals. The vibration measurements of each of these speeds were collected by the accelerometer at a frequency of 20 ms for 1 min, generating 3000 records per speed. Thus, in total, 153,000 vibration records were collected from the simulation model.

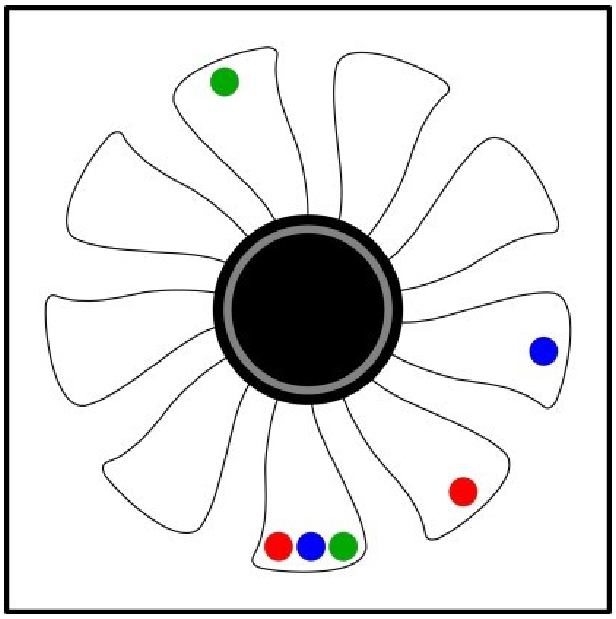


Figure 9: Weights Distribution Illustration..

The vibration data was pre-processed to standardize the inputs and outputs used in the training dataset. When analyzing the behavior of the collected signals, it was observed that the vibrations had no harmonic behavior, i.e., the signal did not show constant amplitude and frequency. For the same window of measurement of the same rotation speed, different and non-standard data were collected. Therefore, it was necessary to define a single frequency and amplitude value for each of the axes in a measurement window. Once 1 min of data was collected for each motor speed of the cooler, represented by 3000 observations, a set of 50 observations of the dataset were defined as the measuring window, which represents 1 s of signal. Figure 9 shows an example of the difference in measurements taken at the same speed rotation of the fan at two different measuring windows.

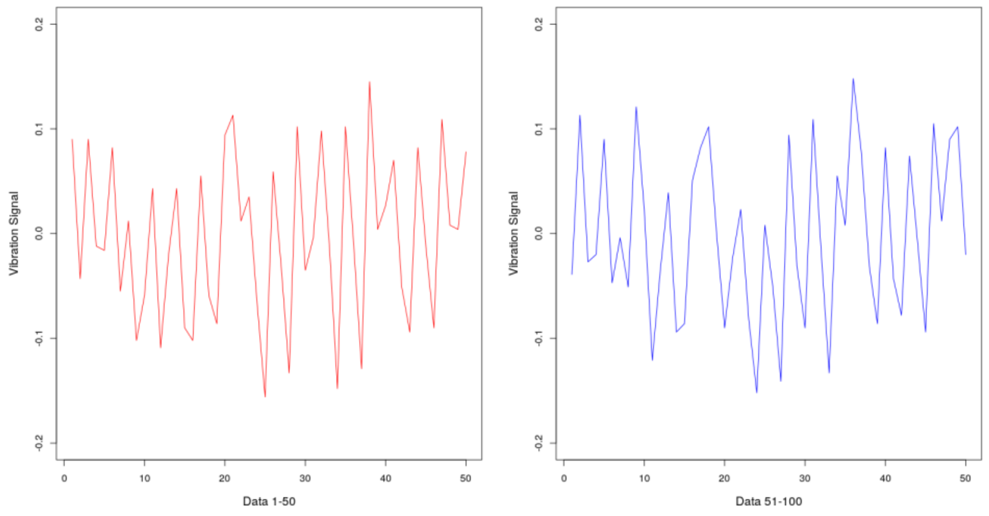


Figure 10: Vibration signals from two different and sequential measuring windows.

Dataset Feature Engineering

The Fourier series and Fourier transform were used to describe the signals in the frequency domain, mapping the various frequencies and amplitudes of the signal. From this transformation it was possible to define a single amplitude and frequency value per axis for each measurement window. For this purpose, the calculation of the Fourier transform using the Fast Fourier Transform function was implemented in R, with the aid of the 'spectral' library, generating all pairs of amplitude and frequency of a measurement window. To generate a unique value, for each window, the Root Mean Square (RMS) value of the signal, or effective value, for the amplitude and frequency sets, was calculated by the following equation:

$$x_{rms} = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}$$

where x_{rms} represents the effective value of the amplitude or frequency; x_i represents each amplitude or frequency that compose the signal; and n the total number of amplitudes or frequencys that compose the signal. A new dataset containing 3060 records of pairs of amplitude and frequency data for each vibration axis was generated. Figure 10 presents an example of the transformation performed on the vibration measurements graphically.

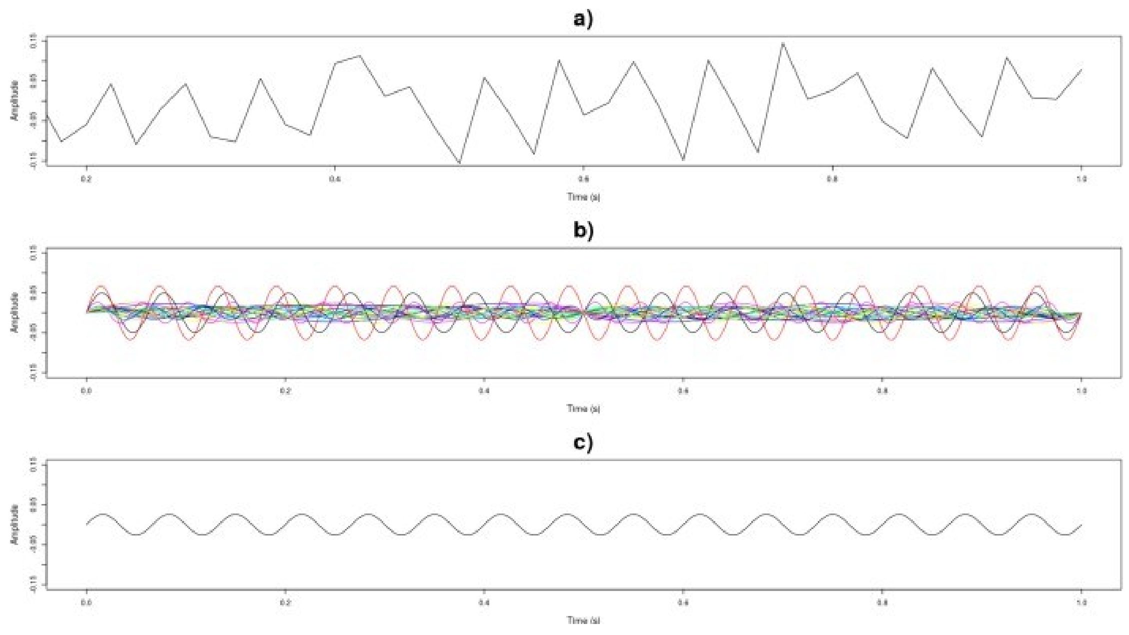


Figure 11: Example of Transformation

Figure 10. Process of simplification of measured vibration signal. (a) Vibration signal collected from a measuring window; (b) Application of the Fourier Transform in the collected signal, generating all pairs of amplitude and frequency present in the signal; (c) Calculation of the RMS value of the amplitudes and frequencies, generating only one pair for each measurement window.

With the unique values calculated for each measurement window it was possible to establish the threshold values of amplitude and frequency needed to calculate the estimated failure time. The limit values, i.e., the maximum possible amplitude or frequency values before equipment failure, were defined by means of the analysis of the measurements made with the cooler rotating at the maximum speed. Thus, the highest values of each amplitude and frequency pair were found for each axis. The maximum amplitudes on the accelerometer measurement scale ([−8g, 8g]) were 0.25 for the x axis and 0.7 for the y and z axes with a frequency of 18 Hz for all the 3 axes. The data were analyzed to remove attributes that would not be representative. The first attributes removed were the amplitude and frequency of the x axis since the way the accelerometer was installed did not generate vibrations on that axis because it is the axis of height. The frequency attributes of the y and z axes were then removed. Although they generated values, the interval between the measured frequencies and the threshold was small and almost unchanged. Thus, only the amplitudes of the y and z axes were used for the training dataset. Figure 11 graphically presents the process of generating the amplitudes used in training dataset, where an amplitude value of the training dataset corresponds to the RMS value of a collected signal measurement window. In this figure, the amplitudes of y axis were represented.

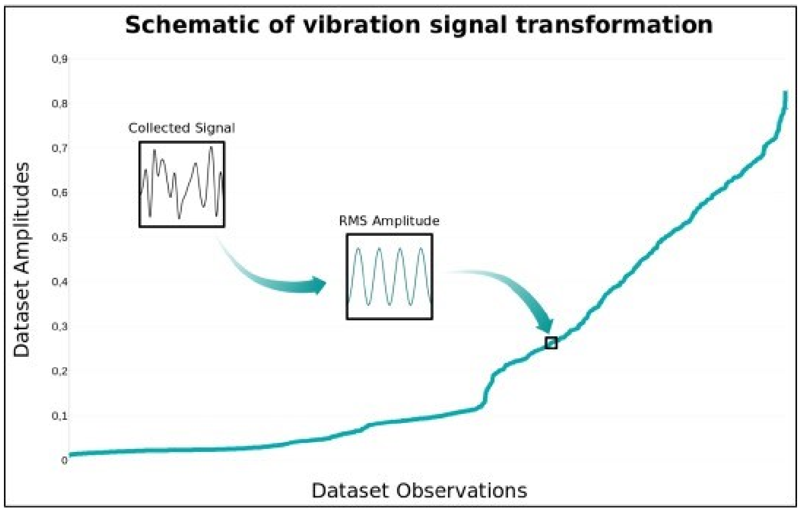


Figure 12: Schematic of vibration signal transformation to generate amplitude dataset (AY).

To calculate the estimated failure time, three steps were performed in the dataset. The first was to establish vibration signal growth rates for the observations, doubling the dataset for the amount of desired growth rates. Three growth rates gr of the signal were set: 0.01, 0.02 and 0.05. In a real-time system this rate would be found by averaging the amplitude and frequency difference of two windows of sequential measurements, indicating how the vibration signal is growing. The second step was to calculate the time, in windows of measurement, so that each amplitude reached the threshold value according to the growth rate established for each amplitude of the axes. This calculation is given by:

$$FT_x = \frac{x_{lim} - x}{x \cdot gr}$$

where FT_x represents the failure time considering one variable x (amplitude); x_{lim} represents the threshold value of x and gr represents the growth rate of the signal. Finally, the third step was to calculate the average between the failure times, generating the expected equipment failure time. Thus, given the input $x = \{x_1, x_2, \dots, x_n\}$ and the signal growth rate gr, the expected output representing the estimated failure time FT_e is given by:

$$FT_e = \frac{\sum_{i=1}^{n} \frac{x_{ilim} - x_i}{x_i \cdot gr}}{n}$$

This was a simple and efficient way to characterize the vibration dataset and can be performed at any time interval for the measurement windows. In the case of this work, the measurement window used was 1 s; however, it could have a duration of 1 min, 1 h or 1 day, with the result of calculating the equipment failure time given in the same units as the measuring window. The equation presented considers a linear growth rate, however exponential or logarithmic growth behaviors could be used depending on the desired or expected growth type. In a real dataset, this growth behavior would be discovered by the ANN.

Related Studies with Results

The models, using the machine learning techniques described in the previous section, were trained, and tested. During ANN model training, the learning algorithm tends to reduce the output error interactively. Figure 12 shows the evolution of the weighted sum of squared error, for the k=1 folder. It can be seen from this figure that the error value tends asymptotically to 0 quickly and the error no longer varies after few training epochs. The other folders presented similar behavior in respect of error evolution during the training. The training phase involved k-fold cross-validation, to verify in a standardized way the performance of the trained model. The k-fold cross-validation enables the simulation of k data scenarios, where the test dataset is not used to train the model. After each folder training, the prediction model was tested with the test dataset and the values predicted and estimated are compared with the performance index. By means of the RMSE calculation, using the estimated and predicted values, it was possible to measure if the generated model had precision and reflected the reality of the system being studied. In this work, the k value was set as 5. Also, the RMSE values can be used to compare the model's performance. Figure 13 presents the graphs which demonstrate this comparison for the models generated in the k = 1 folder, where the x axis represents the set of amplitudes of the signal and the signal growth rate (AY-AZ-GR) and y axis represents the failure time in seconds. For small amplitudes and small vibration signal growth rate the time until the equipment fails is longer; as the vibrations increase and vibration increases, the time until the equipment fails is shorter. The dataset used was generated considering a window of measurement of 1 s which generated a failure time also in seconds, but this window could consider a measurement of 1 day, which would generate a failure time in days, meaning that the measuring window is flexible. Table 1 shows the RMSE values for each model, validation folder and the average of this index.

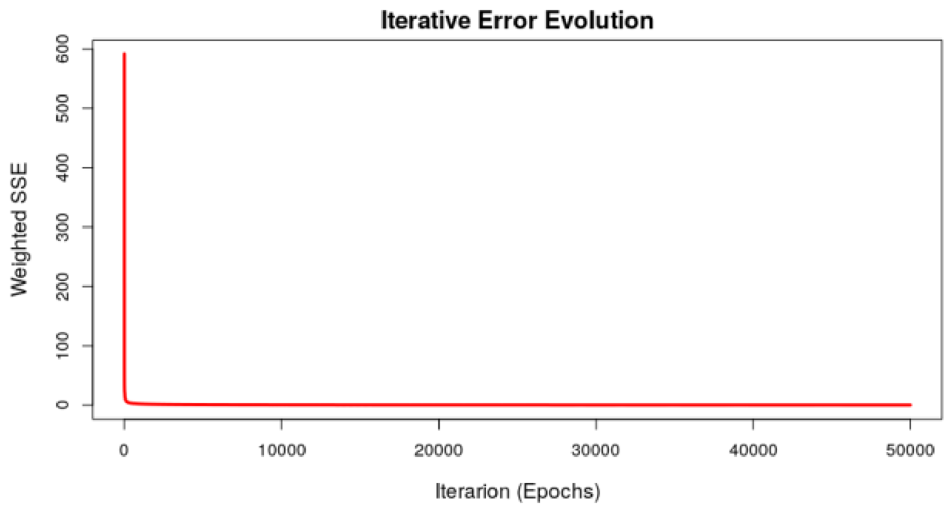


Figure 13: Iterative error evolution of the ANN training process (k = 1)

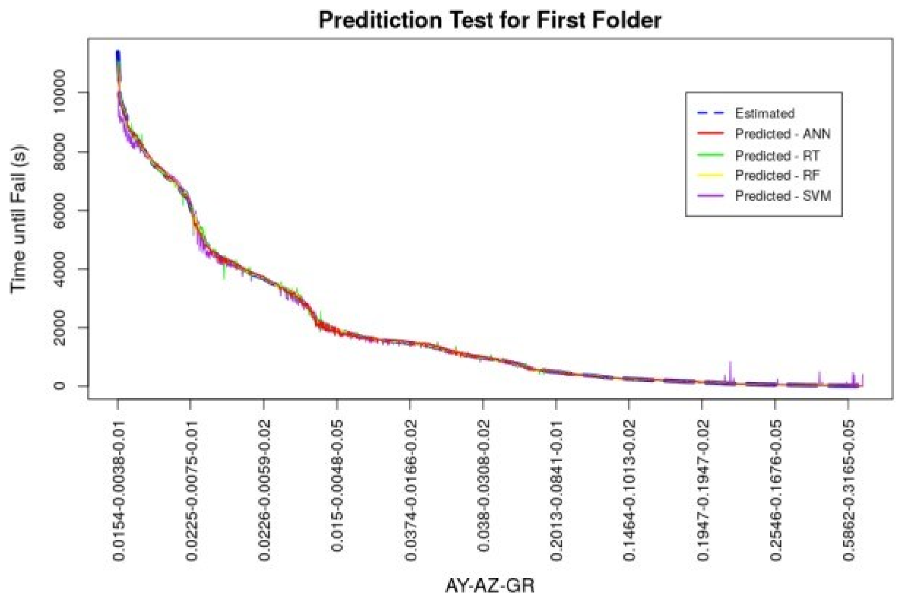


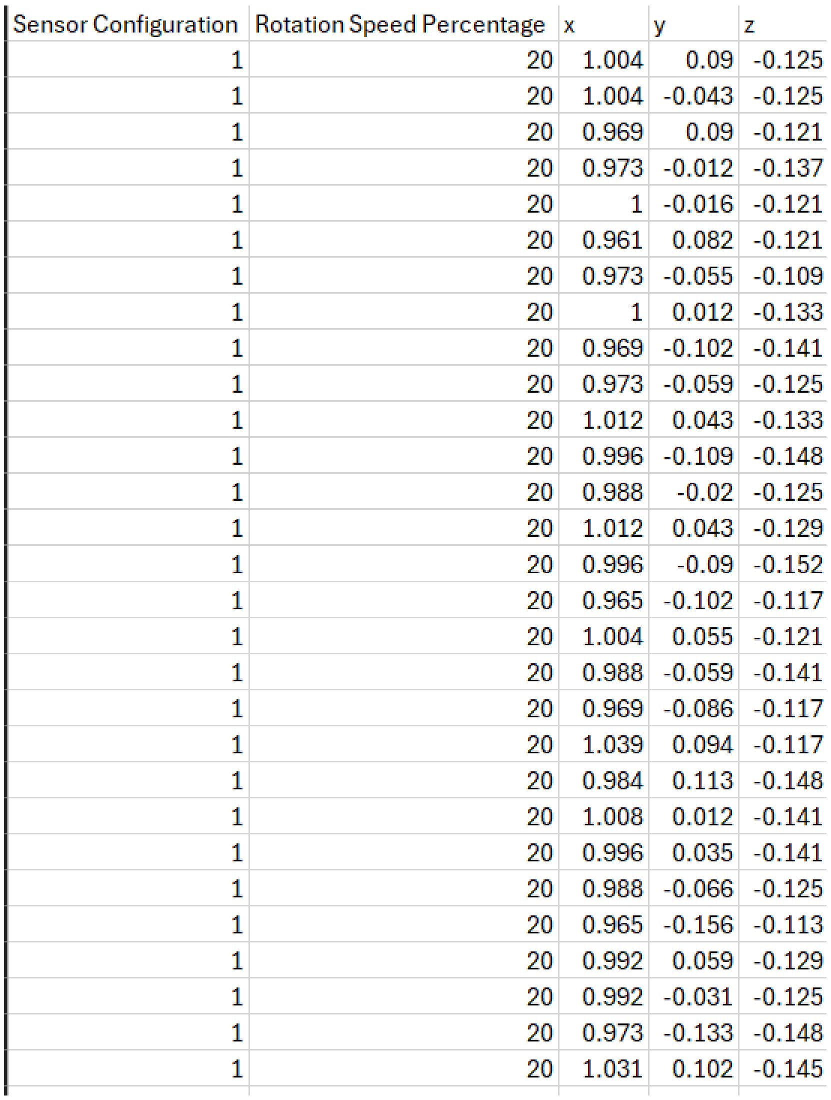
Figure 14: Results of the tests of the first folder (k = 1) carried out with the machine learning techniques.

Folder	ANN	RT	RF	SVM
1	0.0039	0.0047	0.0025	0.0106
2	0.0035	0.0054	0.0035	0.0129
3	0.0028	0.0051	0.0022	0.0105
4	0.0041	0.0052	0.0024	0.0120
5	0.0049	0.0052	0.0026	0.0123
Average	0.0038	0.0051	0.0026	0.0117

Table 1: . RMSE values of the folders used in the test of the machine learning techniques.

IMPLEMENTATION

Data shape:



Sensor Configuration	Rotation Speed Percentage	x	y	z
1	20	1.004	0.09	-0.125
1	20	1.004	-0.043	-0.125
1	20	0.969	0.09	-0.121
1	20	0.973	-0.012	-0.137
1	20	1	-0.016	-0.121
1	20	0.961	0.082	-0.121
1	20	0.973	-0.055	-0.109
1	20	1	0.012	-0.133
1	20	0.969	-0.102	-0.141
1	20	0.973	-0.059	-0.125
1	20	1.012	0.043	-0.133
1	20	0.996	-0.109	-0.148
1	20	0.988	-0.02	-0.125
1	20	1.012	0.043	-0.129
1	20	0.996	-0.09	-0.152
1	20	0.965	-0.102	-0.117
1	20	1.004	0.055	-0.121
1	20	0.988	-0.059	-0.141
1	20	0.969	-0.086	-0.117
1	20	1.039	0.094	-0.117
1	20	0.984	0.113	-0.148
1	20	1.008	0.012	-0.141
1	20	0.996	0.035	-0.141
1	20	0.988	-0.066	-0.125
1	20	0.965	-0.156	-0.113
1	20	0.992	0.059	-0.129
1	20	0.992	-0.031	-0.125
1	20	0.973	-0.133	-0.148
1	20	1.031	0.102	-0.145

Figure 15: Dataset excel file

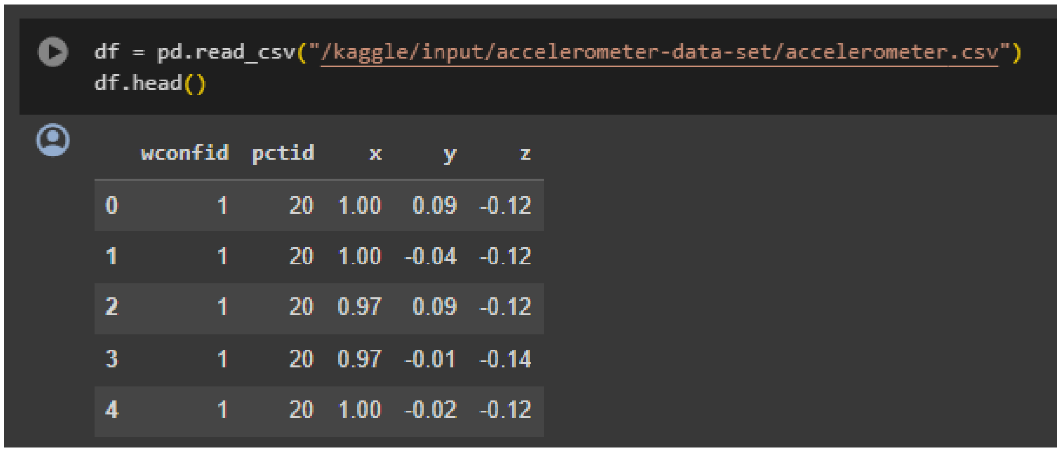
As previously demonstrated, we have 3 Sensor configurations. At each configuration we test the fan with a rotational speed equal to a percentage from the maximum rotational speed of the fan.

The accelerometer returns a reading every 20ms for the x, y, z of the sensor at this moment.

The total Number of data collected is: 153,000

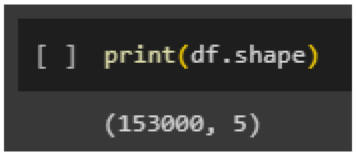
Data Processing

Importing and displaying data:



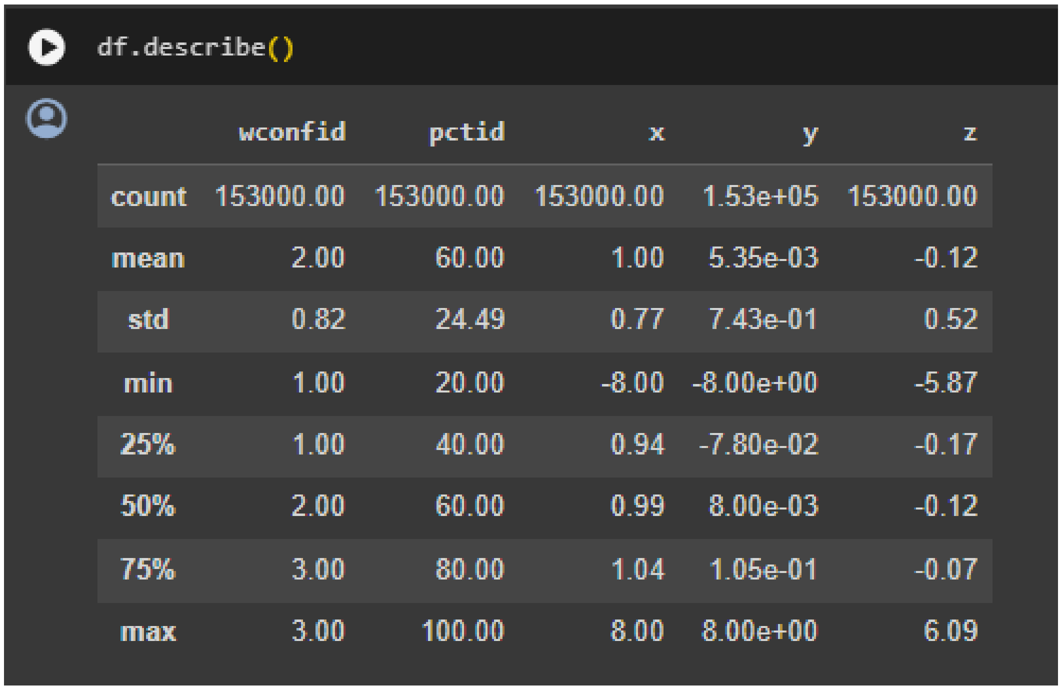
```
df = pd.read_csv("/kaggle/input/accelerometer-data-set/accelerometer.csv")
df.head()
```

	wconfid	pctid	x	y	z
0	1	20	1.00	0.09	-0.12
1	1	20	1.00	-0.04	-0.12
2	1	20	0.97	0.09	-0.12
3	1	20	0.97	-0.01	-0.14
4	1	20	1.00	-0.02	-0.12



```
[ ] print(df.shape)

(153000, 5)
```

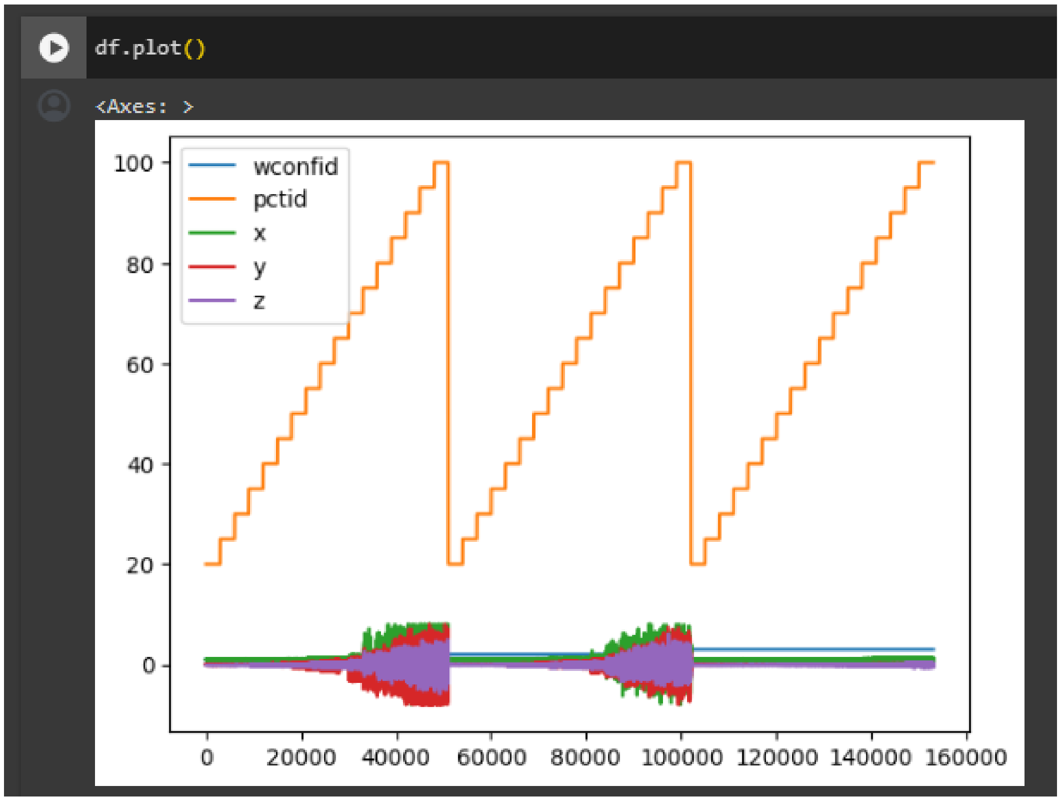
Calculating some parameters for the data distribution:



```
df.describe()
```

	wconfid	pctid	x	y	z
count	153000.00	153000.00	153000.00	1.53e+05	153000.00
mean	2.00	60.00	1.00	5.35e-03	-0.12
std	0.82	24.49	0.77	7.43e-01	0.52
min	1.00	20.00	-8.00	-8.00e+00	-5.87
25%	1.00	40.00	0.94	-7.80e-02	-0.17
50%	2.00	60.00	0.99	8.00e-03	-0.12
75%	3.00	80.00	1.04	1.05e-01	-0.07
max	3.00	100.00	8.00	8.00e+00	6.09

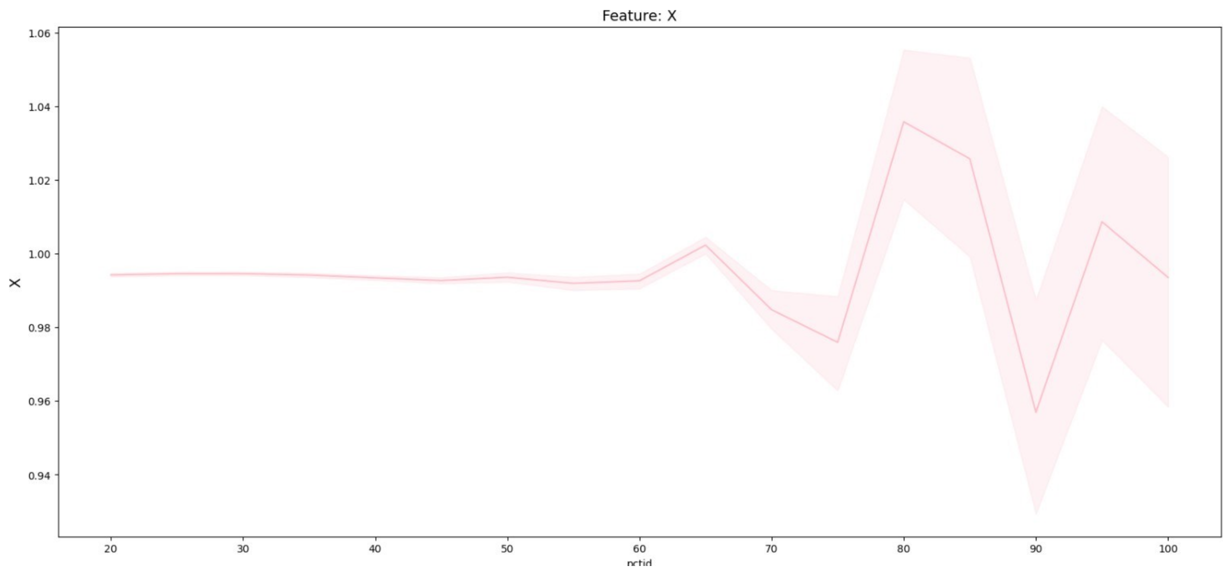
Plotting data:

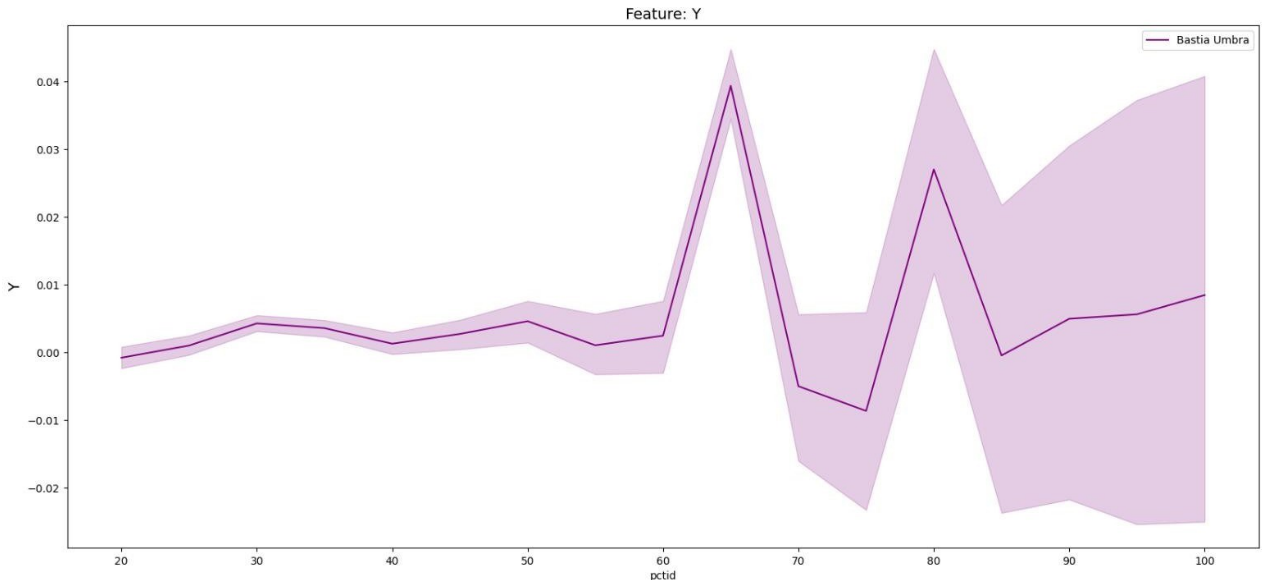


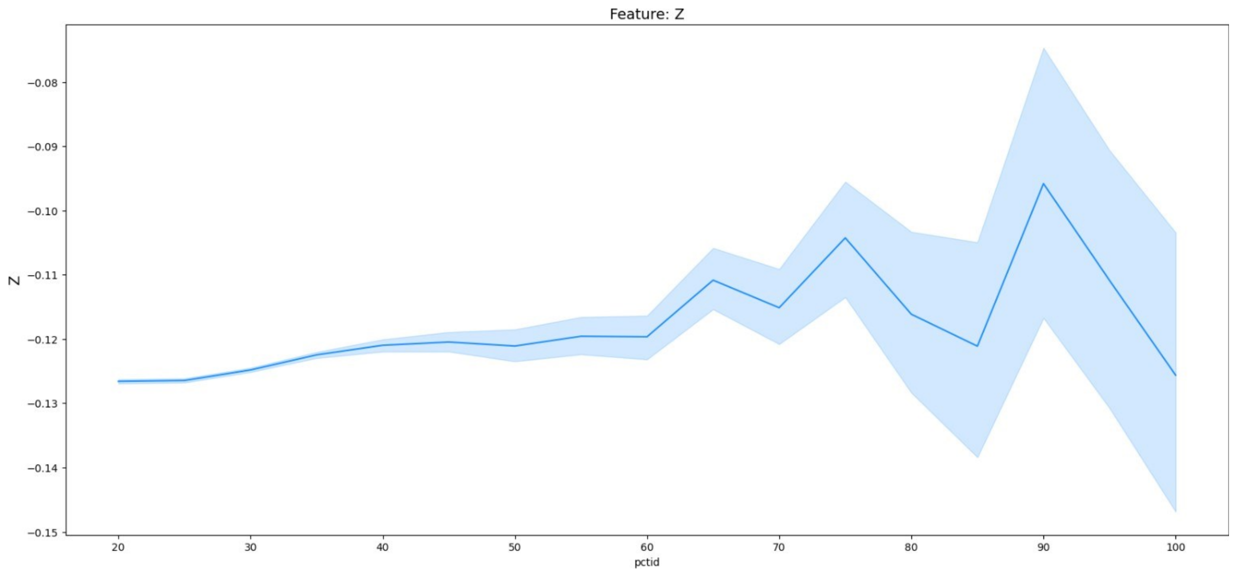
```
df.plot()
```

```
<Axes: >
```

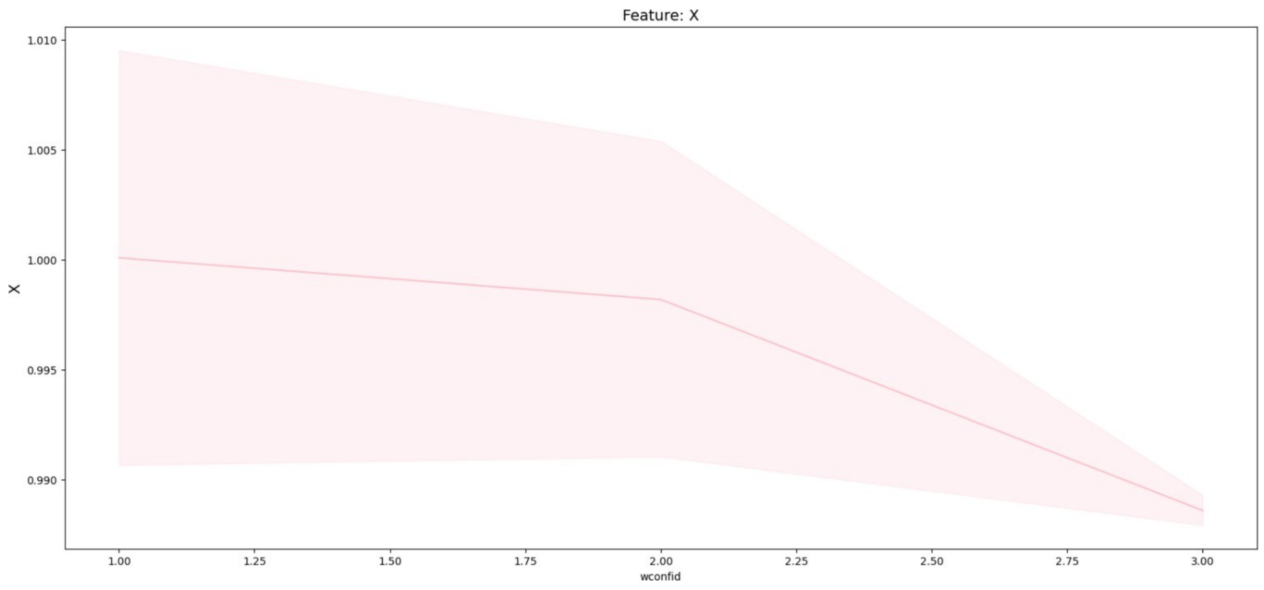
X, Y, Z Features changes upon different Speed percentages:

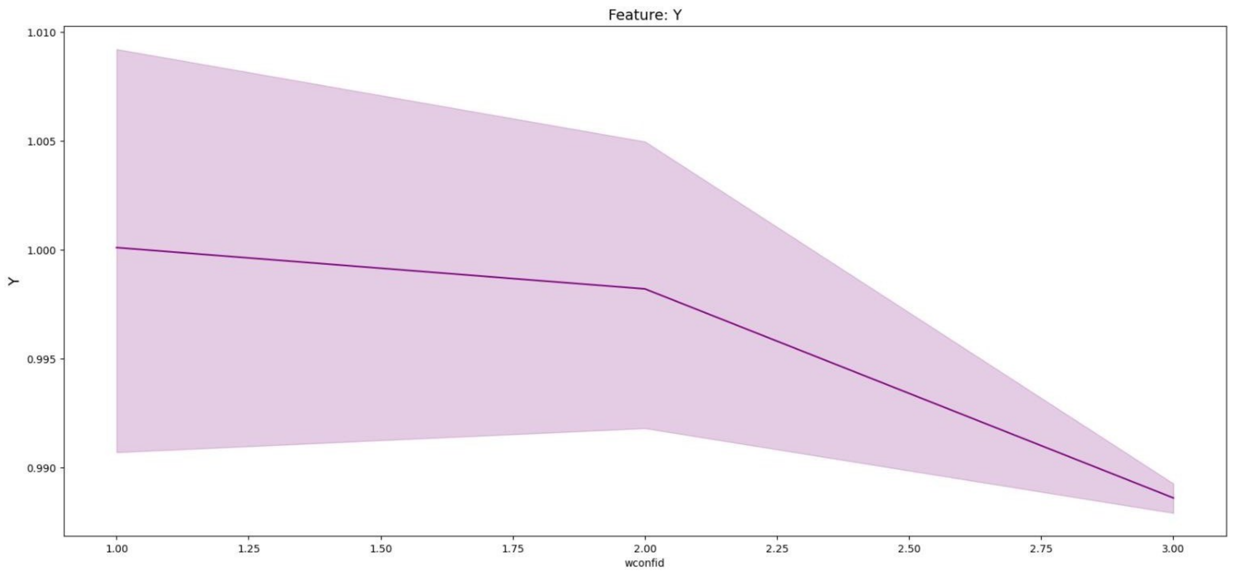


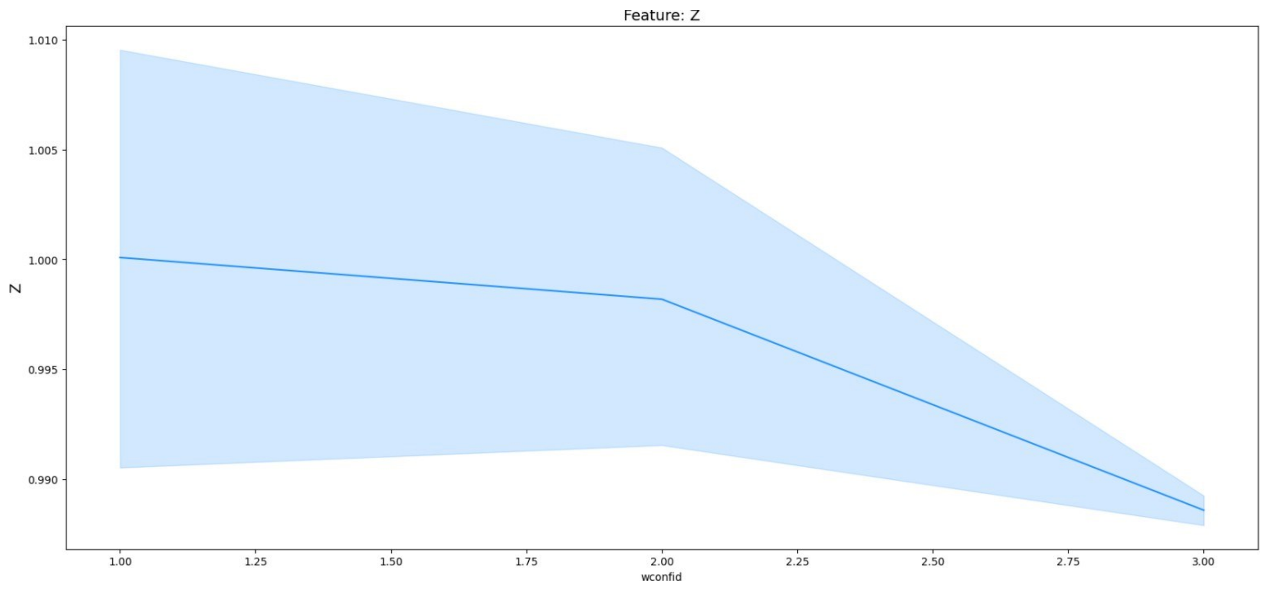

Feature: Y
Bastia Umbra
Y
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
20
30
40
50
60
70
80
90
100
pctid


Feature: Z
Z
−0.08
−0.09
−0.10
−0.11
−0.12
−0.13
−0.14
−0.15
20
30
40
50
60
70
80
90
100
pctid

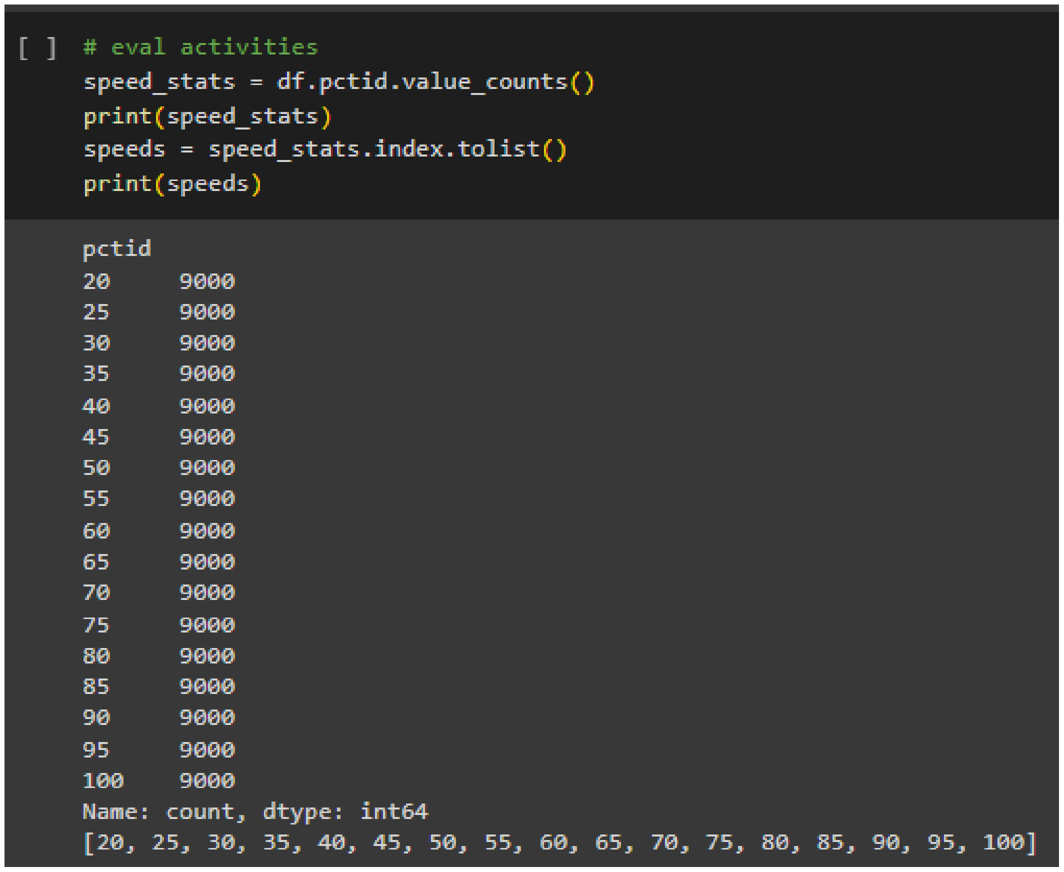
X, Y, Z Features changes upon different Sensor Configurations:





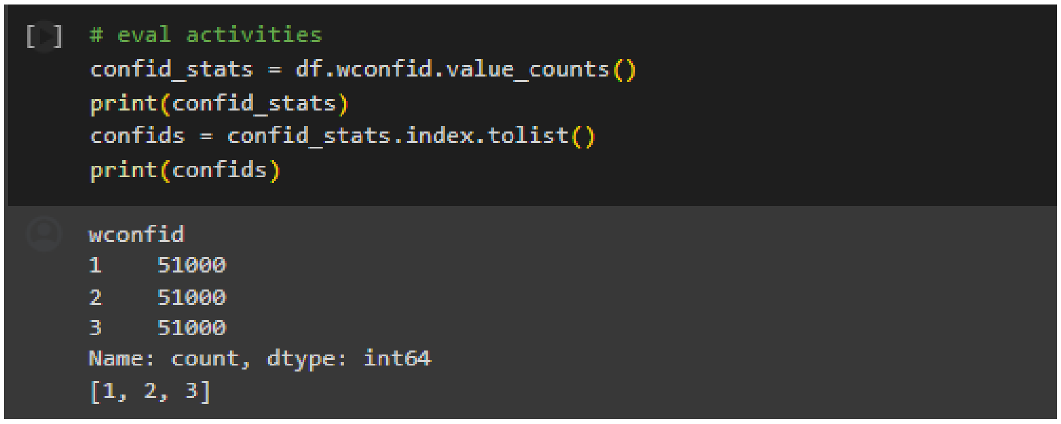


Data Counting:



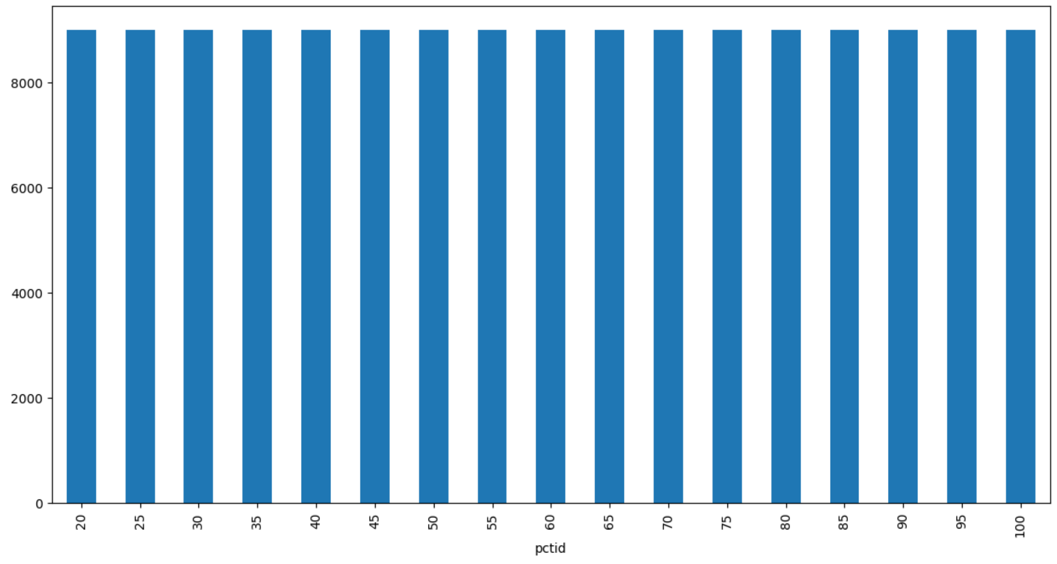
```
[ ]  # eval activities
     speed_stats = df.pctid.value_counts()
     print(speed_stats)
     speeds = speed_stats.index.tolist()
     print(speeds)
```

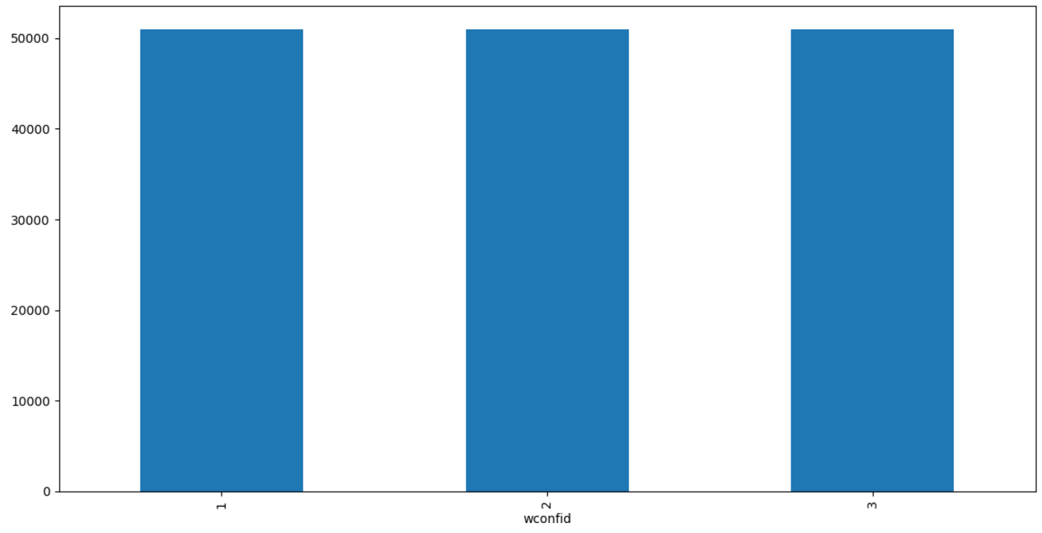
```
pctid
20     9000
25     9000
30     9000
35     9000
40     9000
45     9000
50     9000
55     9000
60     9000
65     9000
70     9000
75     9000
80     9000
85     9000
90     9000
95     9000
100    9000
Name: count, dtype: int64
[20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
```



```
[ ] # eval activities
    confid_stats = df.wconfid.value_counts()
    print(confid_stats)
    confids = confid_stats.index.tolist()
    print(confids)
```

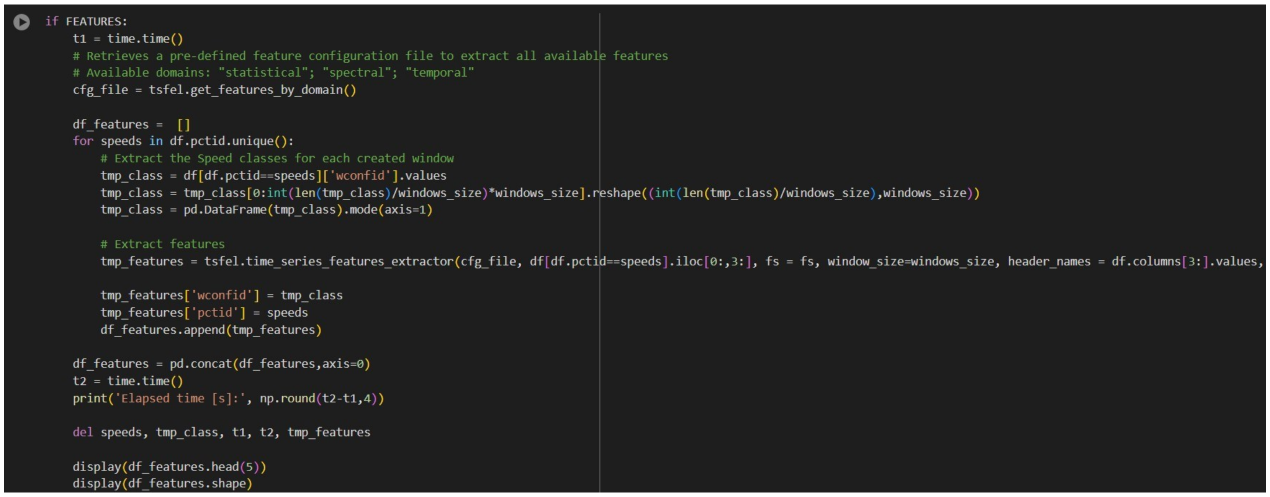
```
wconfid
1    51000
2    51000
3    51000
Name: count, dtype: int64
[1, 2, 3]
```



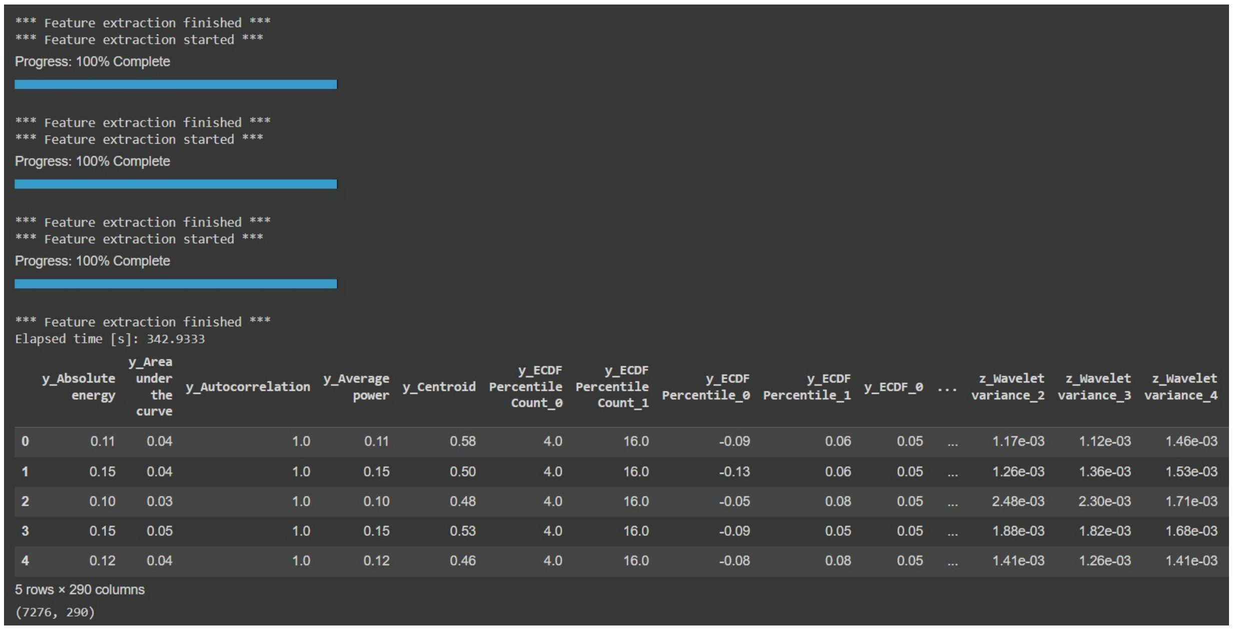


Application

Feature extraction:



```python
if FEATURES:
    t1 = time.time()
    # Retrieves a pre-defined feature configuration file to extract all available features
    # Available domains: "statistical"; "spectral"; "temporal"
    cfg_file = tsfel.get_features_by_domain()

    df_features =  []
    for speeds in df.pctid.unique():
        # Extract the Speed classes for each created window
        tmp_class = df[df.pctid==speeds]['wconfid'].values
        tmp_class = tmp_class[0:int(len(tmp_class)/windows_size)*windows_size].reshape((int(len(tmp_class)/windows_size),windows_size))
        tmp_class = pd.DataFrame(tmp_class).mode(axis=1)

        # Extract features
        tmp_features = tsfel.time_series_features_extractor(cfg_file, df[df.pctid==speeds].iloc[0:,3:], fs = fs, window_size=windows_size, header_names = df.columns[3:].values,

        tmp_features['wconfid'] = tmp_class
        tmp_features['pctid'] = speeds
        df_features.append(tmp_features)

    df_features = pd.concat(df_features,axis=0)
    t2 = time.time()
    print('Elapsed time [s]:', np.round(t2-t1,4))

    del speeds, tmp_class, t1, t2, tmp_features

    display(df_features.head(5))
    display(df_features.shape)
```

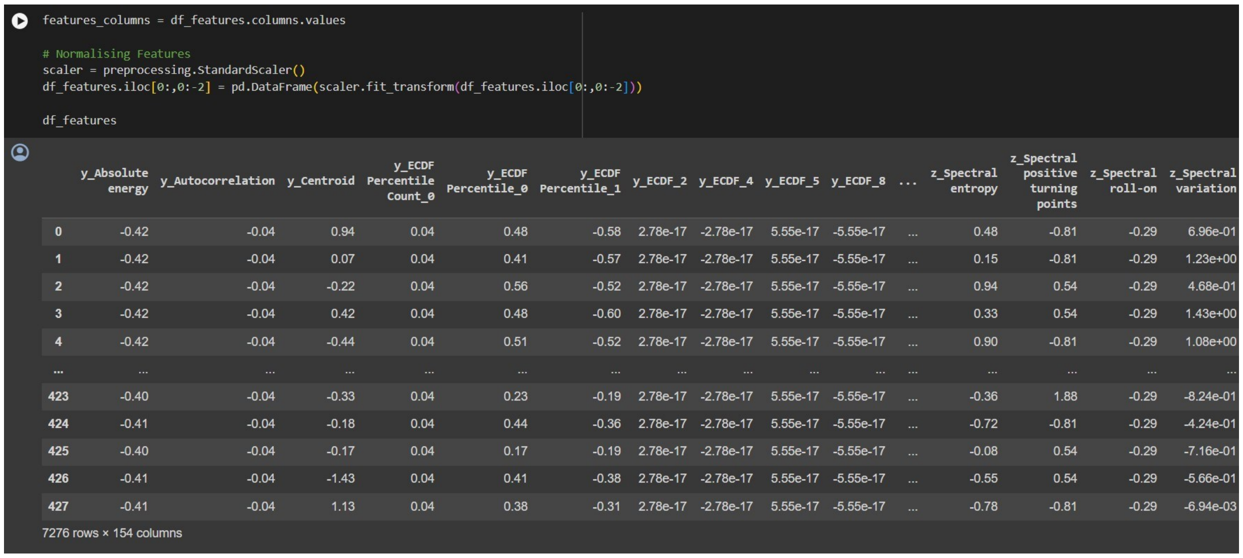


```
*** Feature extraction finished ***
*** Feature extraction started ***
Progress: 100% Complete

*** Feature extraction finished ***
*** Feature extraction started ***
Progress: 100% Complete

*** Feature extraction finished ***
*** Feature extraction started ***
Progress: 100% Complete

*** Feature extraction finished ***
Elapsed time [s]: 342.9333
```

	y_Absolute energy	y_Area under the curve	y_Autocorrelation	y_Average power	y_Centroid	y_ECDF Percentile Count_0	y_ECDF Percentile Count_1	y_ECDF Percentile_0	y_ECDF Percentile_1	y_ECDF_0	...	z_Wavelet variance_2	z_Wavelet variance_3	z_Wavelet variance_4
0	0.11	0.04	1.0	0.11	0.58	4.0	16.0	-0.09	0.06	0.05	...	1.17e-03	1.12e-03	1.46e-03
1	0.15	0.04	1.0	0.15	0.50	4.0	16.0	-0.13	0.06	0.05	...	1.26e-03	1.36e-03	1.53e-03
2	0.10	0.03	1.0	0.10	0.48	4.0	16.0	-0.05	0.08	0.05	...	2.48e-03	2.30e-03	1.71e-03
3	0.15	0.05	1.0	0.15	0.53	4.0	16.0	-0.09	0.05	0.05	...	1.88e-03	1.82e-03	1.68e-03
4	0.12	0.04	1.0	0.12	0.46	4.0	16.0	-0.08	0.08	0.05	...	1.41e-03	1.26e-03	1.41e-03

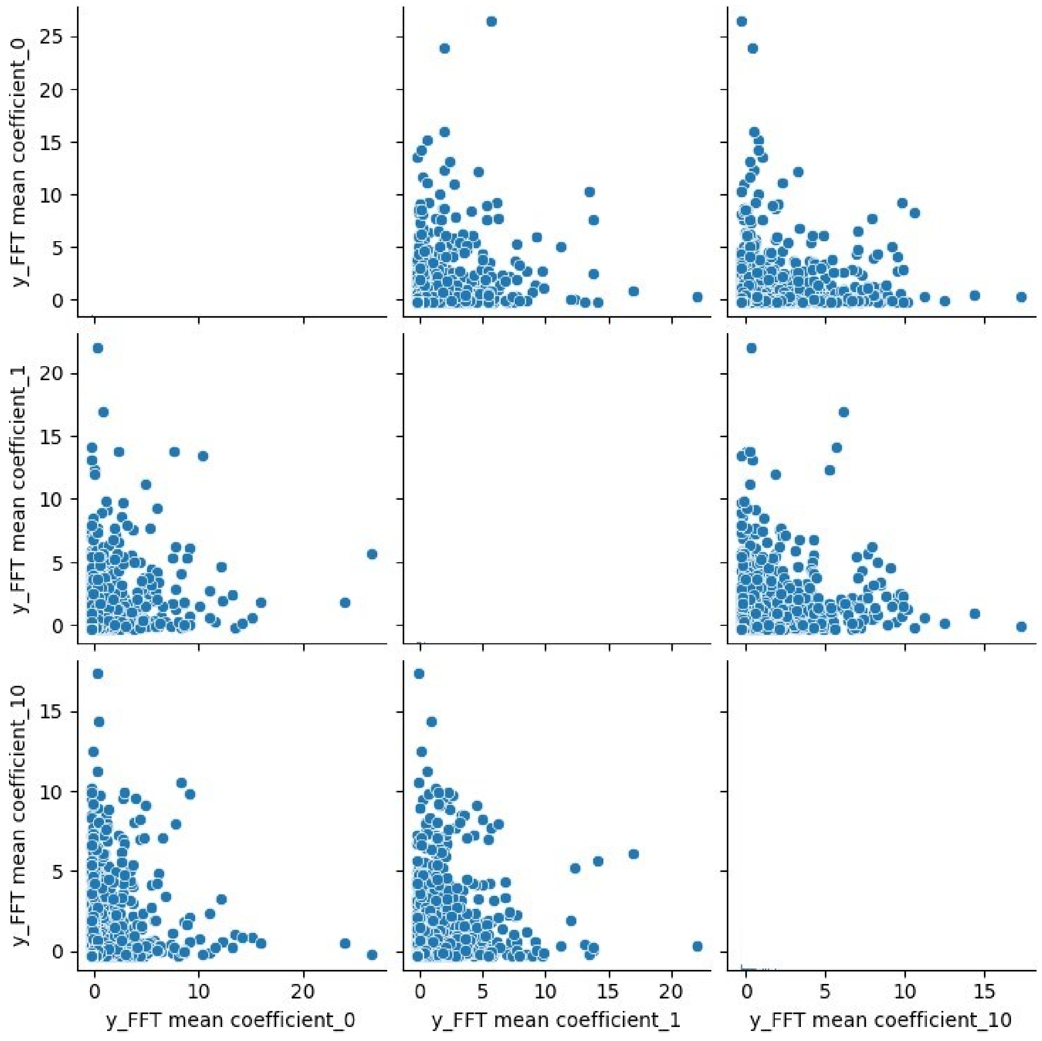
```
5 rows × 290 columns
(7276, 290)
```

Features Normalizing:



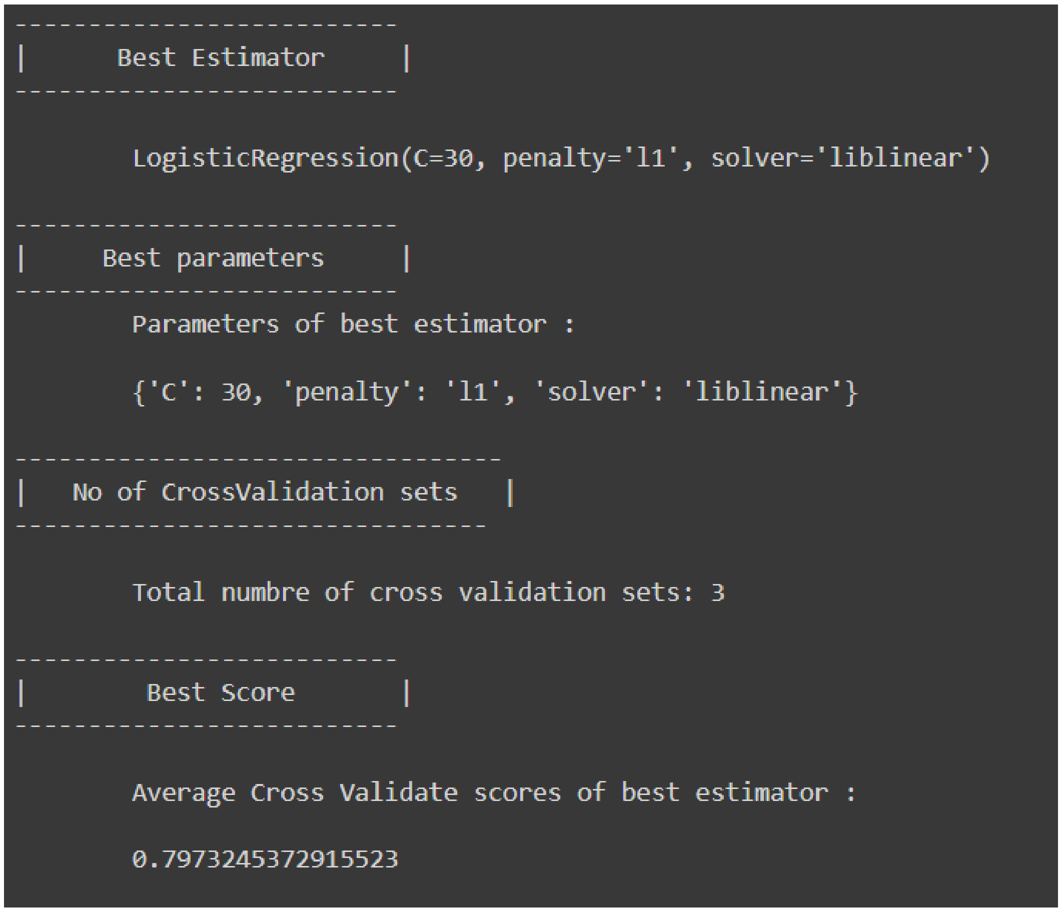
```
features_columns = df_features.columns.values

# Normalising Features
scaler = preprocessing.StandardScaler()
df_features.iloc[0:,0:-2] = pd.DataFrame(scaler.fit_transform(df_features.iloc[0:,0:-2]))

df_features
```

```
                                                y_ECDF                                                                                                  z_Spectral
     y_Absolute                            Percentile      y_ECDF       y_ECDF                                                      z_Spectral  positive  z_Spectral  z_Spectral
         energy  y_Autocorrelation  y_Centroid  Count_0  Percentile_0  Percentile_1  y_ECDF_2  y_ECDF_4  y_ECDF_5  y_ECDF_8  ...   entropy   turning     roll-on   variation
                                                                                                                                           points
0         -0.42              -0.04        0.94     0.04          0.48         -0.58  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...      0.48     -0.81       -0.29   6.96e-01
1         -0.42              -0.04        0.07     0.04          0.41         -0.57  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...      0.15     -0.81       -0.29   1.23e+00
2         -0.42              -0.04       -0.22     0.04          0.56         -0.52  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...      0.94      0.54       -0.29   4.68e-01
3         -0.42              -0.04        0.42     0.04          0.48         -0.60  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...      0.33      0.54       -0.29   1.43e+00
4         -0.42              -0.04       -0.44     0.04          0.51         -0.52  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...      0.90     -0.81       -0.29   1.08e+00
...         ...                ...         ...      ...           ...           ...       ...       ...       ...       ...  ...       ...       ...         ...        ...
423       -0.40              -0.04       -0.33     0.04          0.23         -0.19  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...     -0.36      1.88       -0.29  -8.24e-01
424       -0.41              -0.04       -0.18     0.04          0.44         -0.36  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...     -0.72     -0.81       -0.29  -4.24e-01
425       -0.40              -0.04       -0.17     0.04          0.17         -0.19  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...     -0.08      0.54       -0.29  -7.16e-01
426       -0.41              -0.04       -1.43     0.04          0.41         -0.38  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...     -0.55      0.54       -0.29  -5.66e-01
427       -0.41              -0.04        1.13     0.04          0.38         -0.31  2.78e-17 -2.78e-17  5.55e-17 -5.55e-17  ...     -0.78     -0.81       -0.29  -6.94e-03
7276 rows × 154 columns
```

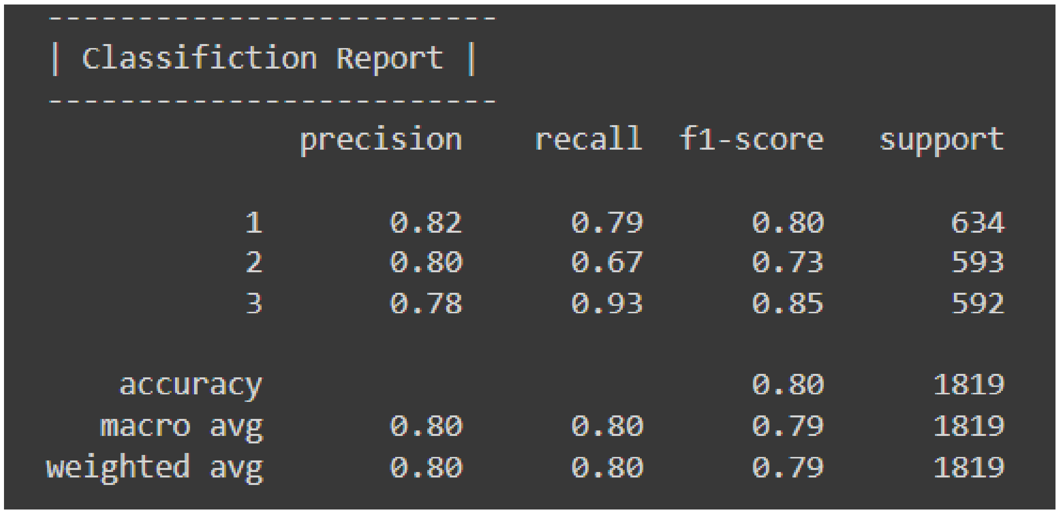


Applying Different Classifier Models and Obtaining Results

Logistic Regression

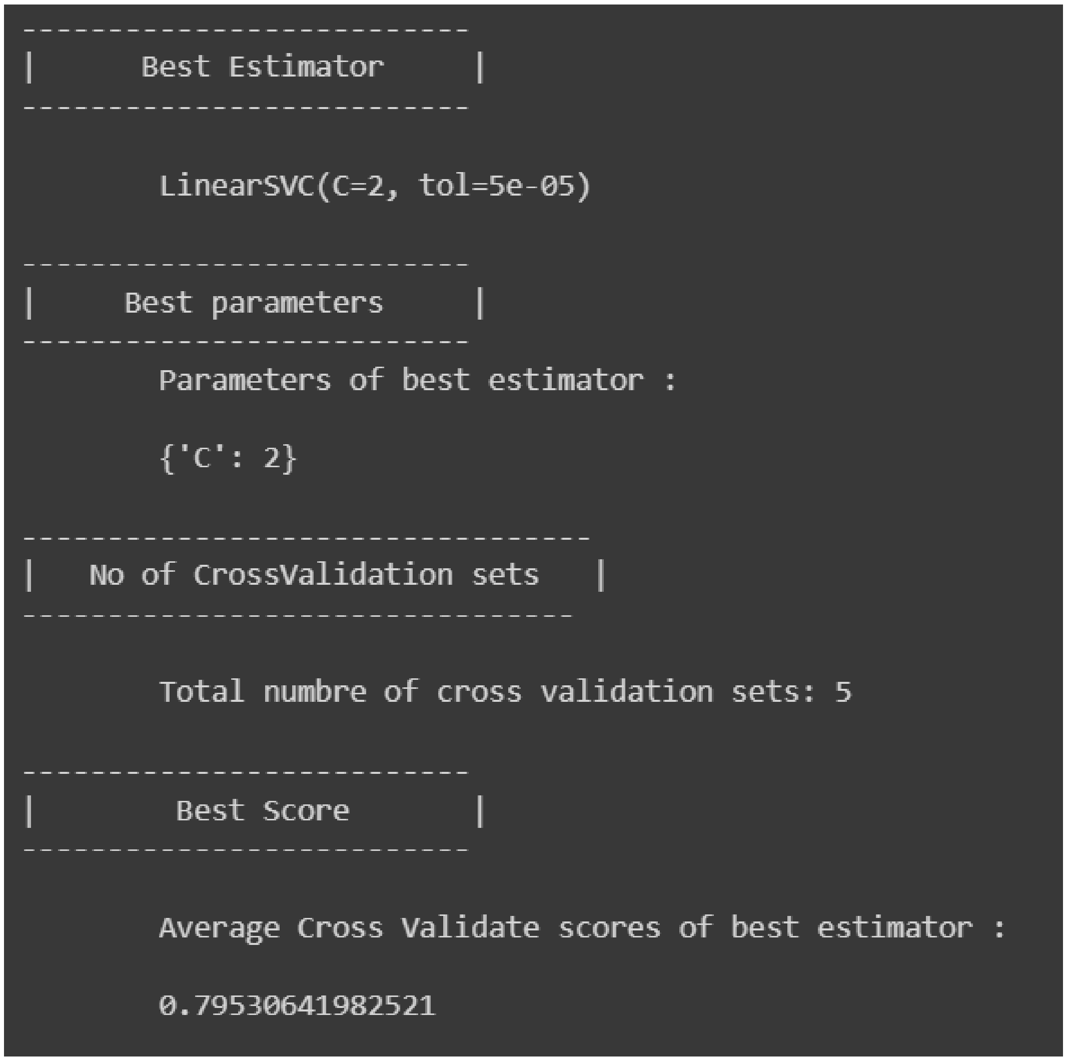


```
-------------------------
|      Best Estimator     |
-------------------------


        LogisticRegression(C=30, penalty='l1', solver='liblinear')

-------------------------
|     Best parameters     |
-------------------------
        Parameters of best estimator :

        {'C': 30, 'penalty': 'l1', 'solver': 'liblinear'}

--------------------------------
|   No of CrossValidation sets   |
-------------------------------


        Total numbre of cross validation sets: 3

-------------------------
|        Best Score       |
-------------------------


        Average Cross Validate scores of best estimator :

        0.7973245372915523
```

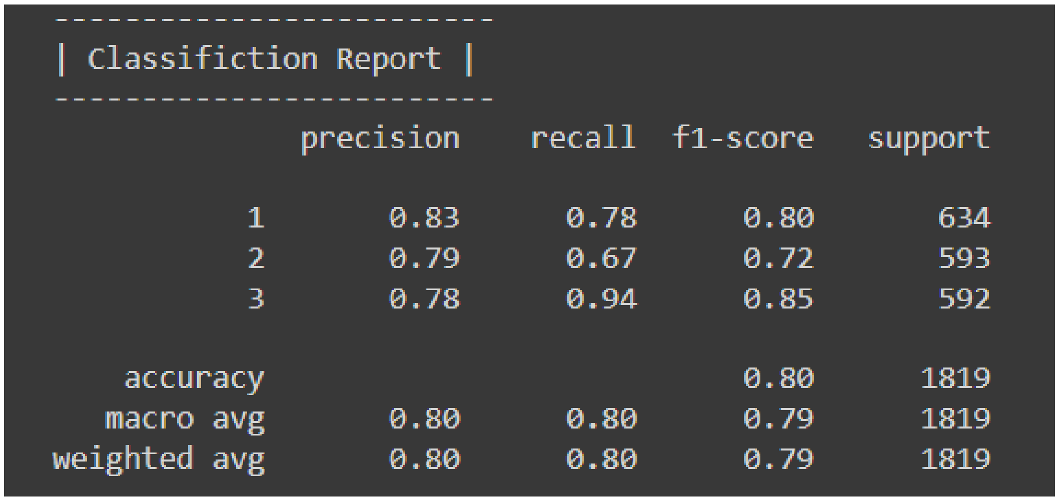


```
 -------------------------
 | Classifiction Report |
 -------------------------
              precision    recall  f1-score   support

           1       0.82      0.79      0.80       634
           2       0.80      0.67      0.73       593
           3       0.78      0.93      0.85       592

    accuracy                           0.80      1819
   macro avg       0.80      0.80      0.79      1819
weighted avg       0.80      0.80      0.79      1819
```

Linear SVC

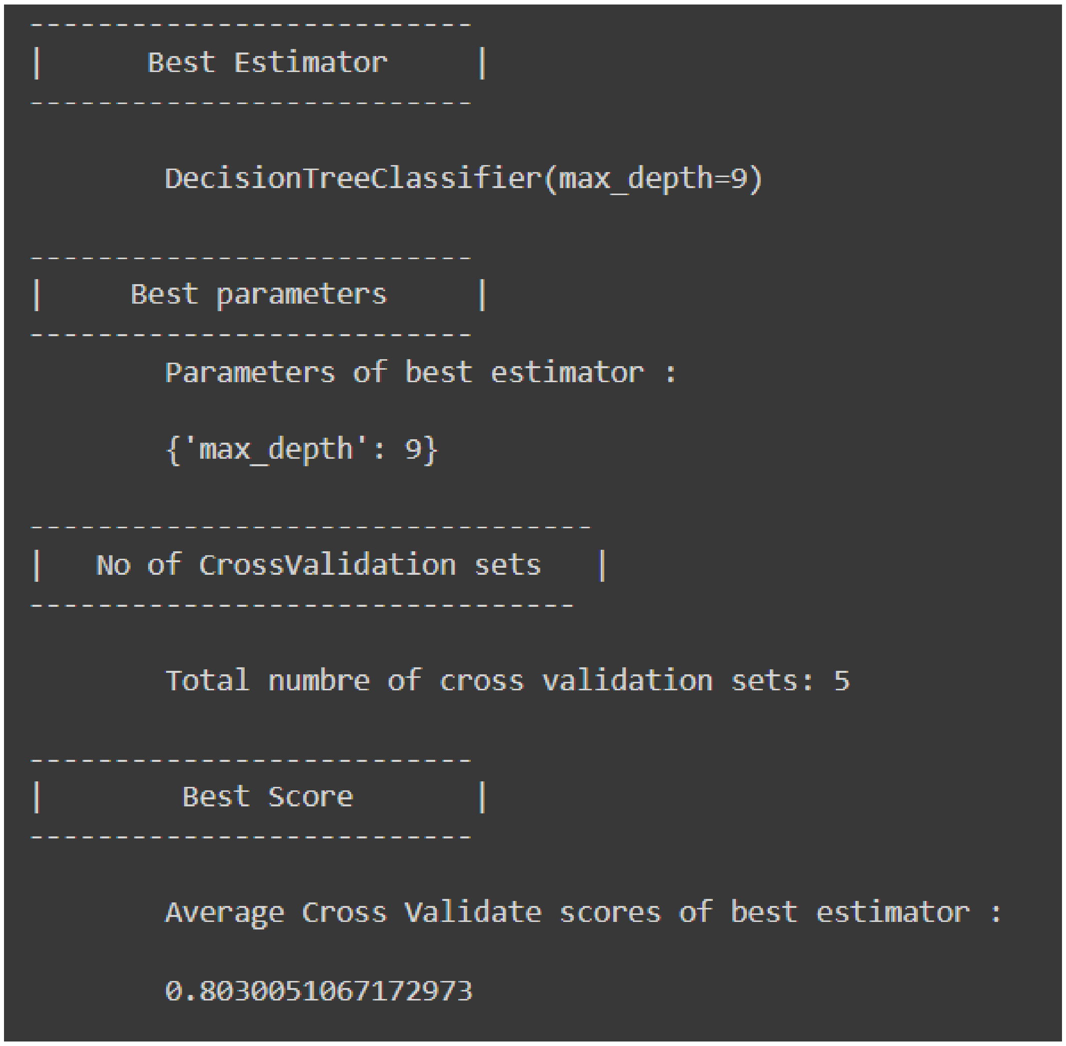


```
--------------------------
|      Best Estimator     |
--------------------------

        LinearSVC(C=2, tol=5e-05)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'C': 2}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.79530641982521
```

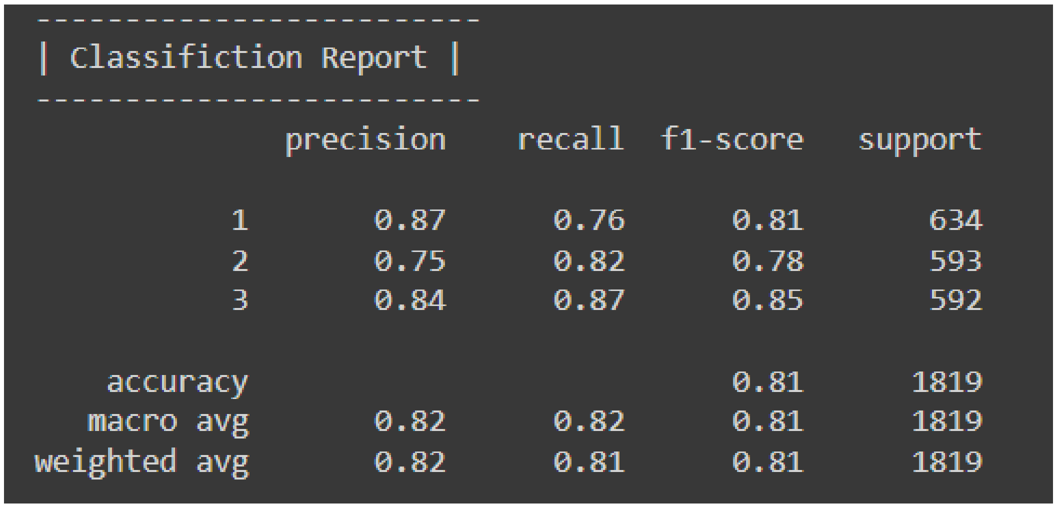


```
  --------------------------
  | Classifiction Report |
  --------------------------
              precision    recall  f1-score   support

           1       0.83      0.78      0.80       634
           2       0.79      0.67      0.72       593
           3       0.78      0.94      0.85       592

    accuracy                           0.80      1819
   macro avg       0.80      0.80      0.79      1819
weighted avg       0.80      0.80      0.79      1819
```

Decision Tree:

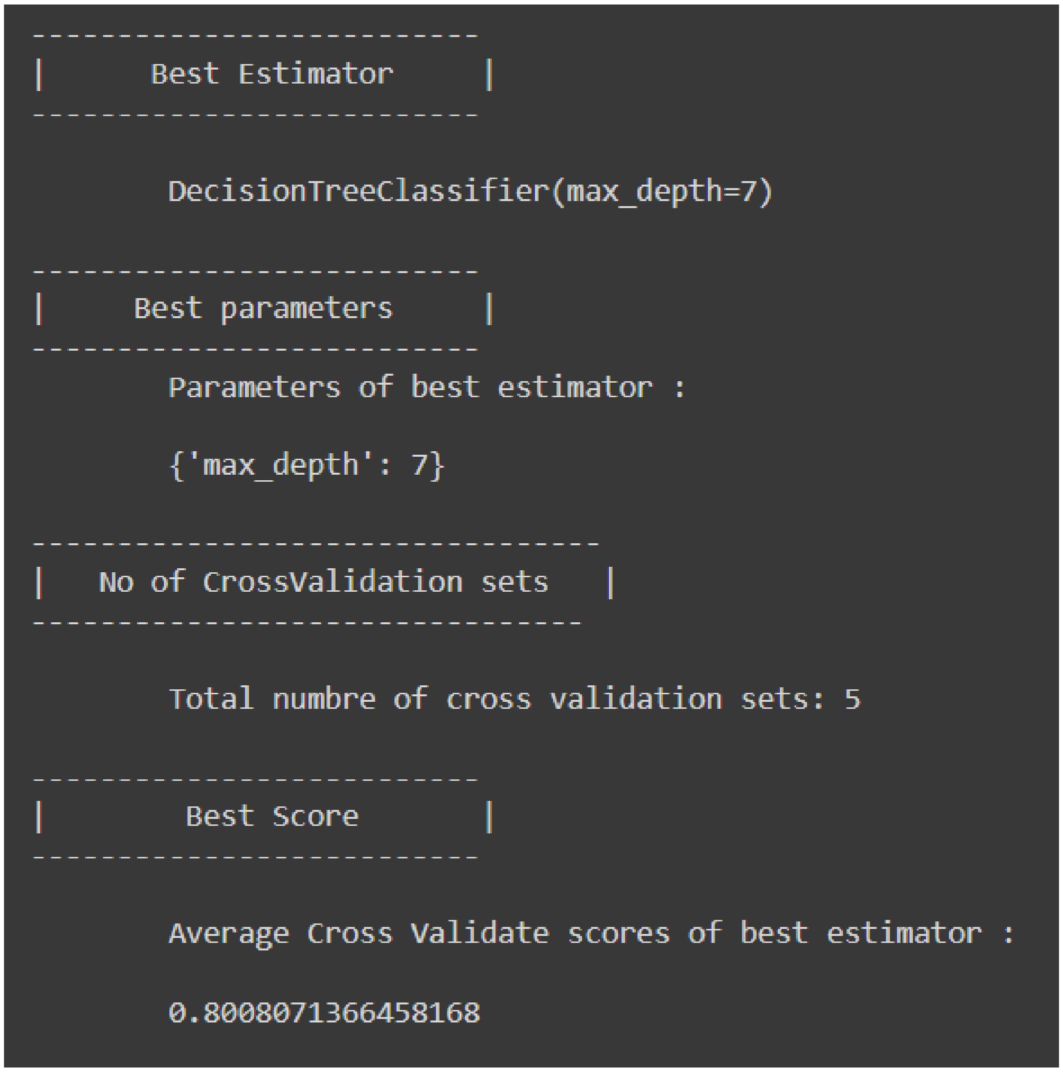


```
-------------------------
|      Best Estimator     |
-------------------------


        DecisionTreeClassifier(max_depth=9)

-------------------------
|     Best parameters     |
-------------------------
        Parameters of best estimator :

        {'max_depth': 9}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 5

-------------------------
|        Best Score       |
-------------------------

        Average Cross Validate scores of best estimator :

        0.8030051067172973
```

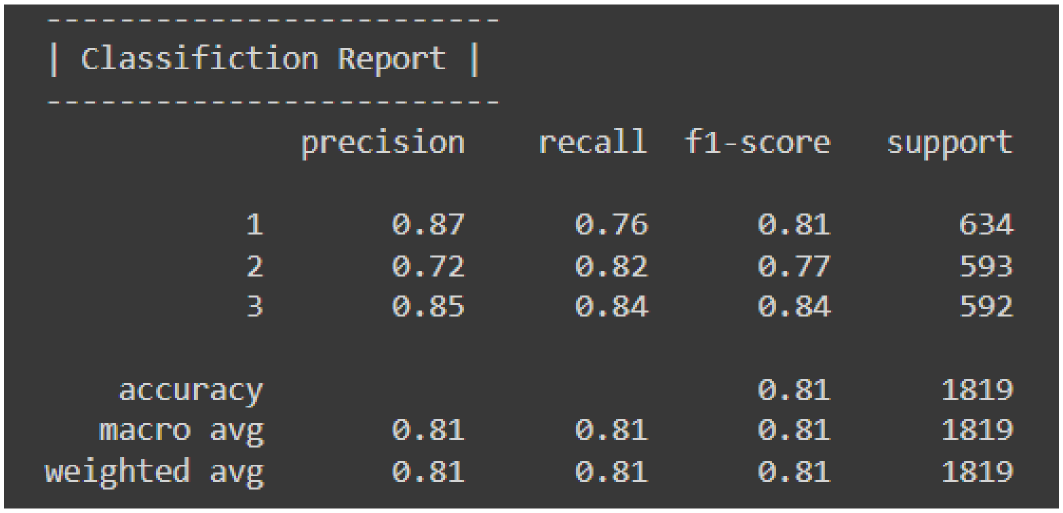


```
-------------------------
| Classifiction Report |
-------------------------
              precision    recall  f1-score   support

           1       0.87      0.76      0.81       634
           2       0.75      0.82      0.78       593
           3       0.84      0.87      0.85       592

    accuracy                           0.81      1819
   macro avg       0.82      0.82      0.81      1819
weighted avg       0.82      0.81      0.81      1819
```

Random Forest:

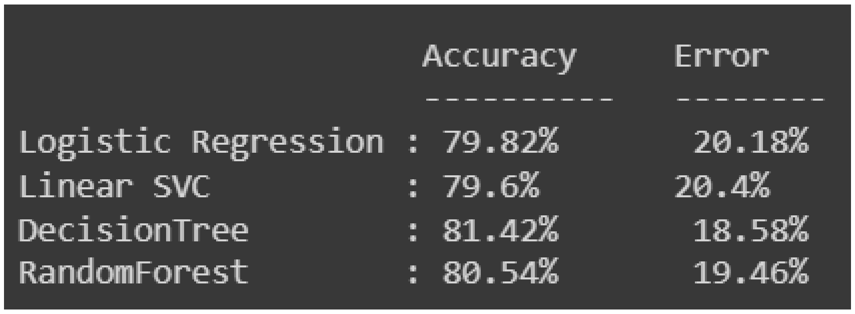


```
--------------------------
|      Best Estimator     |
--------------------------

        DecisionTreeClassifier(max_depth=7)

--------------------------
|     Best parameters     |
--------------------------
        Parameters of best estimator :

        {'max_depth': 7}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

        Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

        Average Cross Validate scores of best estimator :

        0.8008071366458168
```



```
 --------------------------
 | Classifiction Report |
 --------------------------
              precision    recall  f1-score   support

           1       0.87      0.76      0.81       634
           2       0.72      0.82      0.77       593
           3       0.85      0.84      0.84       592

    accuracy                           0.81      1819
   macro avg       0.81      0.81      0.81      1819
weighted avg       0.81      0.81      0.81      1819
```

Comparing Between the Four Classification Techniques:



```
                       Accuracy     Error
                       ----------   --------
Logistic Regression : 79.82%        20.18%
Linear SVC          : 79.6%        20.4%
DecisionTree        : 81.42%        18.58%
RandomForest        : 80.54%        19.46%
```

Colab Link:

https://colab.research.google.com/drive/1uvckn8DSo2iTnLhTKEr9VdatX56qqtvv?usp=sharing

References:

1. https://www.mdpi.com/2076-3417/12/8/3751
2. https://www.kaggle.com/datasets/dhinaharp/accelerometer-data-set/data
3. https://www.mdpi.com/1424-8220/19/19/4342