DATA MINING PROJECT REPORT

Shankara Narayanan Bangalore Ramalingam

Table of Contents

Problem Statement:

The main aim of the project is finding interesting patterns in the dataset and try to solve these interesting problems by using algorithms and AI techniques. In this project I have solved 2 supervised learning problems and 1 unsupervised learning techniques.

Keywords: Supervised learning, classification, regression, unsupervised learning, Anomaly detection, Time Series forecasting

Introduction:

As mentioned in the above problem statement, we will be solving 2 supervised learning tasks, and in this case, I have worked on a classification and a regression problem. The dataset consists of almost 900 thousand records and 76 features. Some of the features included in the csv file are Vehicle_ID, EMMISION_LEVEL, Battery_Replacement_Date etc.

To perform the classification task, I have taken the EMMISSION_LEVEL column which has 2 classes in it, and they are namely '1' and '2'. This EMMISSION_LEVEL classes represent the European Emission standards which are followed in many countries. To perform this task, I employed the use of Pandas, Scikit-learn, seaborn and matplotlib. One of the main reasons to choose this binary classification problem was because of the class imbalance which was present in the column.

For the second supervised learning task, I had taken a regression problem wherein I was trying to predict the number of batteries which will be replaced on a monthly basis. The features which was used for this task was the Battery Replacement Date and the Mounted Battery Generation. One of the main reasons which I found this interesting was because it is a very practical problem and a good business case to solve. This also resulted in creating new features using existing ones.

In the first task I have implemented the following classification algorithms such as KN Classifier, Logistic Regression and MLP Classifier. In all these tasks we use accuracy as the metric for judging the model performance. In the regression task I had implemented the Long-term short Memory (LSTM) method which is quite popular for these cases. To measure the model's performance in the regression task we are using root mean square error (RMSE) as the metric.

For the unsupervised learning task, I am performing anomaly detection of the battery state health at different times of the day. I have done feature engineering and created features such as Dayoftheweek, weekday etc. For the anomaly detection I have used algorithms such as LocalOutlierFactors, OneClassSVM etc. To evaluate the performance of these algorithms I have used RMSE as the metric.

Discussion:

In this part the whole project methodology will be explained that means the flow of the project including loading dataset, explanatory analysis, data cleaning, handling class imbalance, downsampling, feature extraction, standard scaler, MinMax scaler, machine learning model. For all the models we are using accuracy for classification, and root mean square error for regression and anomaly detection.

Dataset:

In this dataset each vehicle has a number which is provided in the “Vehicle_ID” column. All the vehicle in the dataset will operating either in electric, diesel or hybrid modes. A number of features are measurements that are calculated at the corresponding time. The features such as “State of Battery Health”, “Operation_TIME” etc. are measured with corresponding time. There are also other features in the dataset, such as country, geographical area in which vehicle is used, date of the battery replacement date etc. The dataset contains about 861916 samples and is corresponding to 4031 vehicles. All the information in the dataset is anonymized.

Python Libraries:

This section briefly explains all about the libraries which are used in the project. In this project I have used pandas, Matplotlib, seaborn, keras, miceforest, numpy, scikit-learn.

Pandas:

Pandas is a software library which is written in python and it offers data manipulation and analysis features. It also offers data structures and operations for manipulating numerical tables and time series. Some of the most salient features of pandas include data merging, slicing, reshaping etc. In the time series functionality side include date range generation, frequency conversion etc.

Matplotlib and Seaborn:

Matplotlib and seaborn are both plotting libraries which are provided in python and is used for visualization of data present in the dataset. Pandas and matplotlib have a built-in integration, i.e. if we use pandas for visualization in uses matplotlib internally.

Seaborn is another library which was built on top of matplotlib and has a close integration with python. The seaborn library offers a variety of functionality such as automatic estimation and plotting linear regression plots. It also supports univariate and bivariate distribution.

Keras:

Keras is another open source library that provides a python interface for ANN. This library also has multiple backends supported such as TensorFlow, Microsoft Cognitive Toolkit etc. This library is very popular for implementing deep learning algorithms etc. Some of them are convolution and recurrent neural networks etc.

Miceforest:

Miceforest library provides fast, memory efficient Multiple Imputation by Chained Equations (MICE) using random forests. It can impute both categorical and numeric data without much setup and has an array of diagnostic plots available. This package is available in both Python as well with R.

Scikit-Learn:

This is an open source machine learning written in Python. It is used to implement various machine learning algorithms which ranges from classification, regression, and clustering algorithms. Some of the most popular machine learning algorithms which are present in the Logistic Regression, Decision Tree, Support Vector Machines, DBSCAN etc.

The same library was also used in the unsupervised learning task as well. I have used it for the algorithms such as LocalOutlierFactor, OneClassSVM etc.

Methodology:

The dataset was loaded using the pandas library. The below screenshot shoes the loading of the dataset using pandas library.

```
In [7]: data=pd.read_csv('Project_Dataset.csv')
        data
```

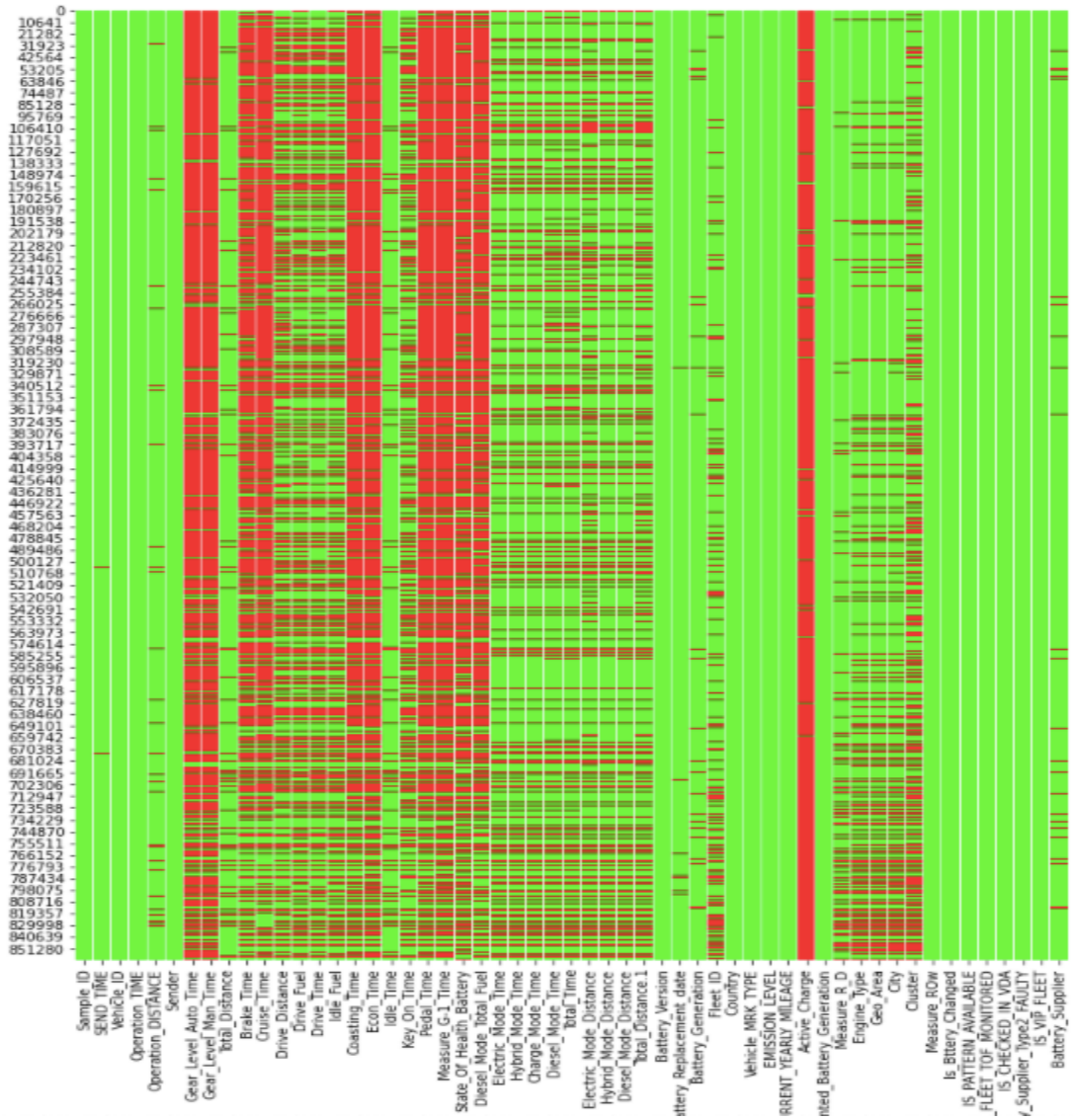
Out[7]:

	Sample_ID	Charge_mode_DISTANCE	SEND_TIME	Vehicle_ID	Operation_TIME	Operation_DISTANCE	Sender	Gear_Level_Auto_Time	Gear_Level_Man_
0	262526303	NaN	2019-03-11 17:35:14	1	9157.192	247897.59	1	NaN	
1	115517796	NaN	2017-11-10 18:09:35	1	3146.041	161516.35	1	NaN	
2	251857645	NaN	2019-02-21 07:24:22	1	8900.673	134506.02	2	NaN	
3	47893312	NaN	2015-12-20 16:26:27	1	1386.207	27599.18	1	NaN	
4	96601019	NaN	2017-06-12 13:01:16	1	1146.683	129568.38	1	NaN	
...	...	...	...	...	...	...	...	...	
861911	9339753	0.0	2012-06-26 06:24:59	4027	4.520	11.36	3	0.567	
861912	375757668	NaN	2019-09-11 00:23:34	4028	708.737	12986.23	1	NaN	
861913	156703893	0.0	2018-05-30 17:14:19	4029	15.085	17.89	1	1.275	
861914	32584182	NaN	2015-03-17 11:05:49	4030	0.000	0.00	1	0.696	
861915	348660913	NaN	2019-08-08 11:51:28	4031	5.763	14.70	1	1.314	

861916 rows × 76 columns

Exploratory Analysis:

After loading the dataset to the pandas library, I had started to work with exploratory analysis where I had performed finding the percentage of missing values in the data frame column wise, and I have also visualized it using the heatmap, the image of the heatmap is shown below

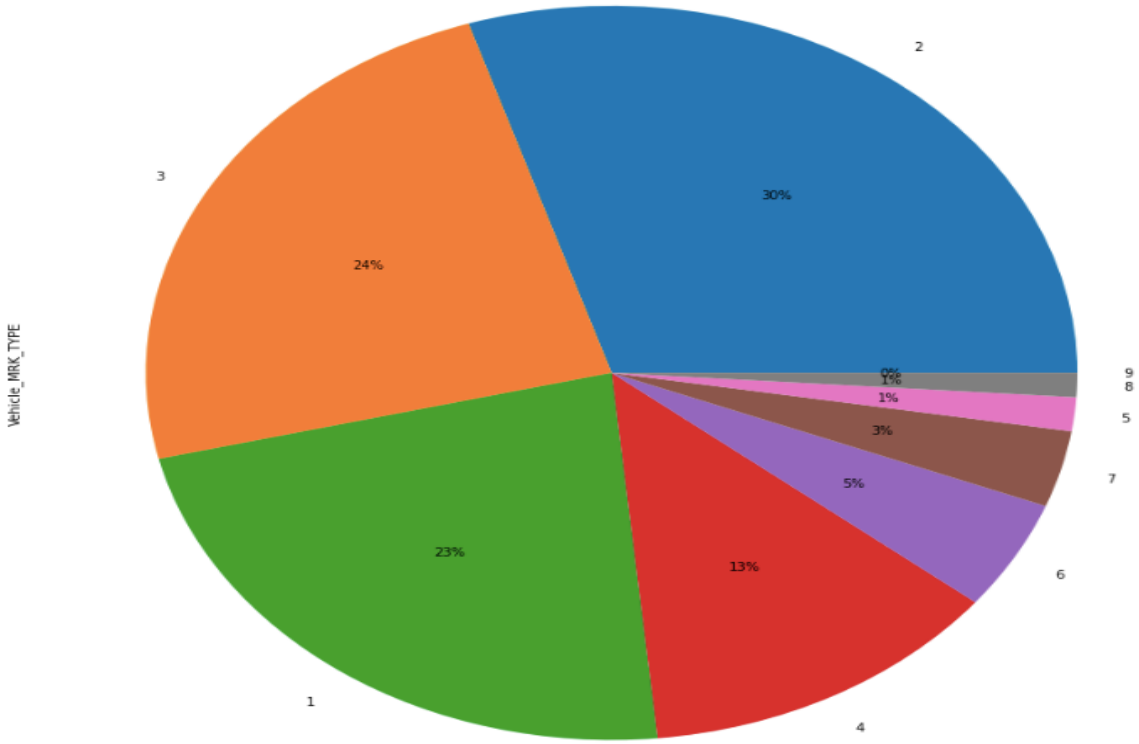


The red in the heatmap represents the empty values and green is the non-empty ones. From the above heatmap I had dropped the features which had empty values of more than 70%, as it would be computationally more expensive to impute these values. Hence, I had dropped all the following features from the data frame.

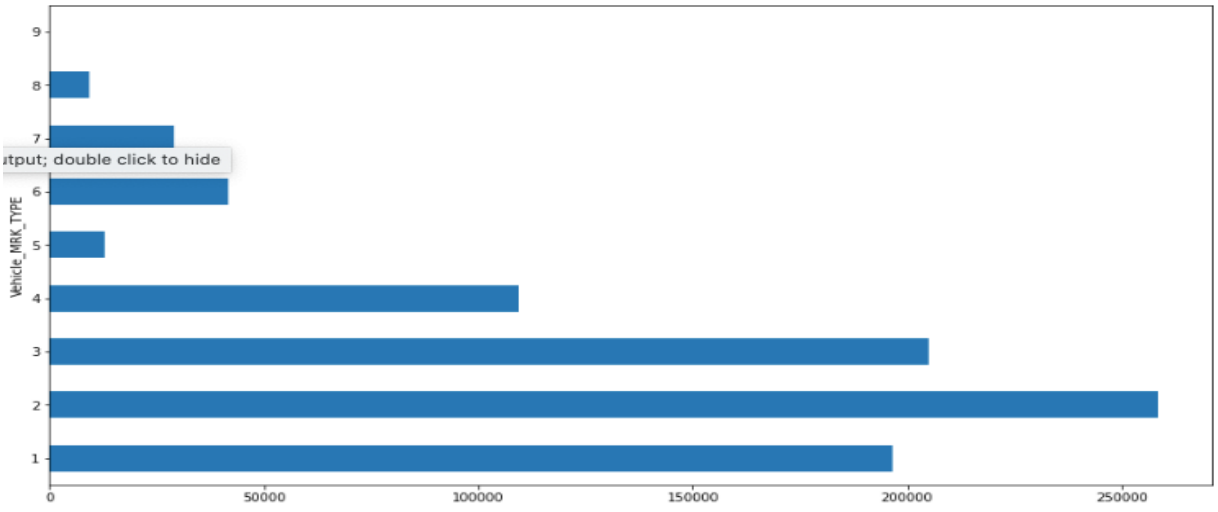
```
del data['Active_Charge']
del data['Gear_Level_Auto_Time']
del data['Gear_Level_Man_Time']
del data['Measure_G-1_Time']
del data['Econ_Time']
del data['Cruise_Time']
del data['Coasting_Time']
del data['Pedal_Time']
del data['Diesel_Mode_Total_Fuel']
```

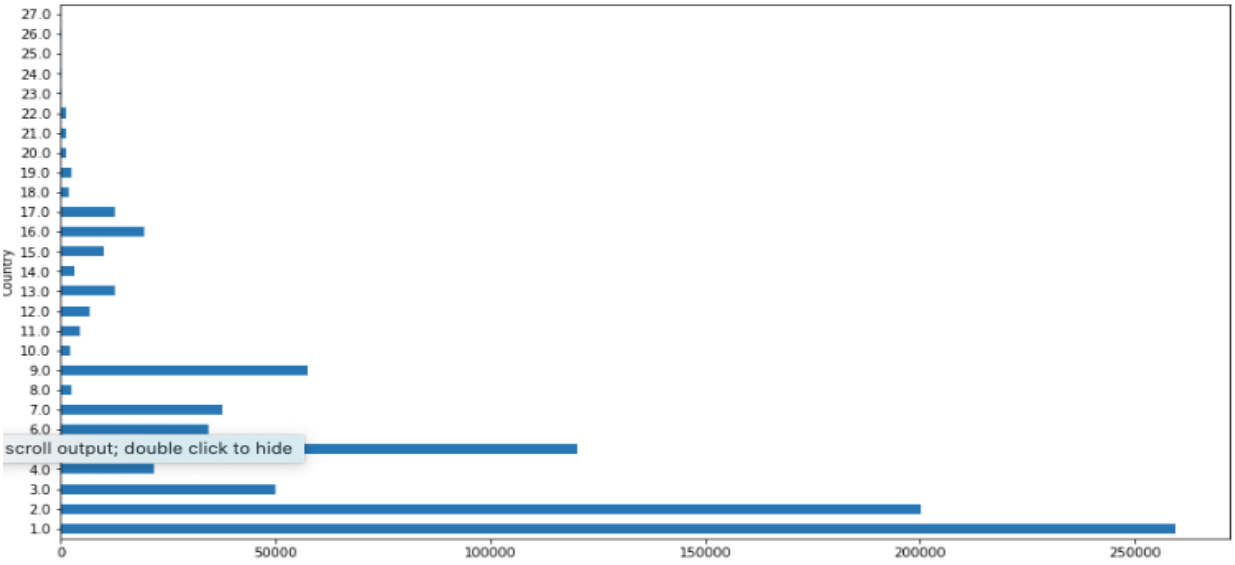
After dropping these columns, I had started to work on the data analysis part of the task. After this I had performed analysis of different features present in the column, I had initially worked

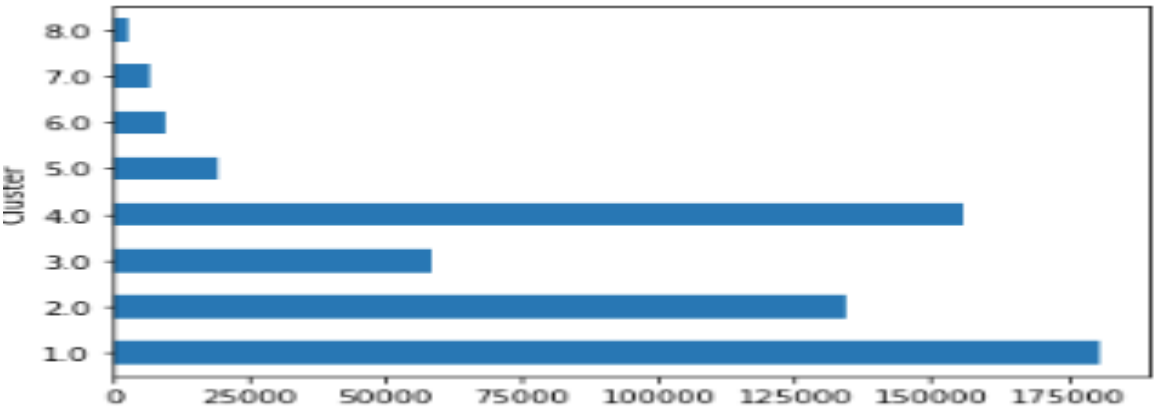
on “Vehicle_MRK_TYPE”, and the “EMMISION_LEVEL”. On performing the analysis in the, I had taken the “Vehicle_MRK_TYPE”, feature. I wanted to check the number of values present in each category, I have attached the screenshot below



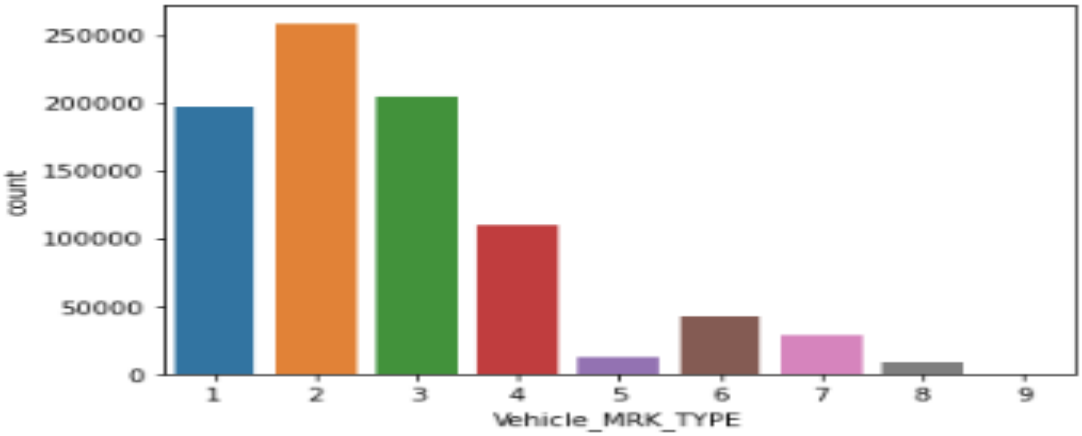
I have also checked for more analysis on the above-mentioned feature with other features such as City, Cluster, Current Yearly Mileage. We are checking the count category wise and we are visualizing it.





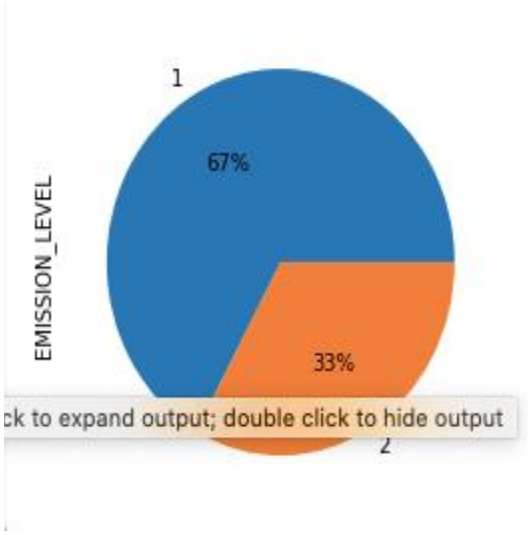


In the below graph I also wanted to check which vehicle MRK Type had the highest number of battery replacements. So, I have done a plot to check the count of each category. I have attached the plot below

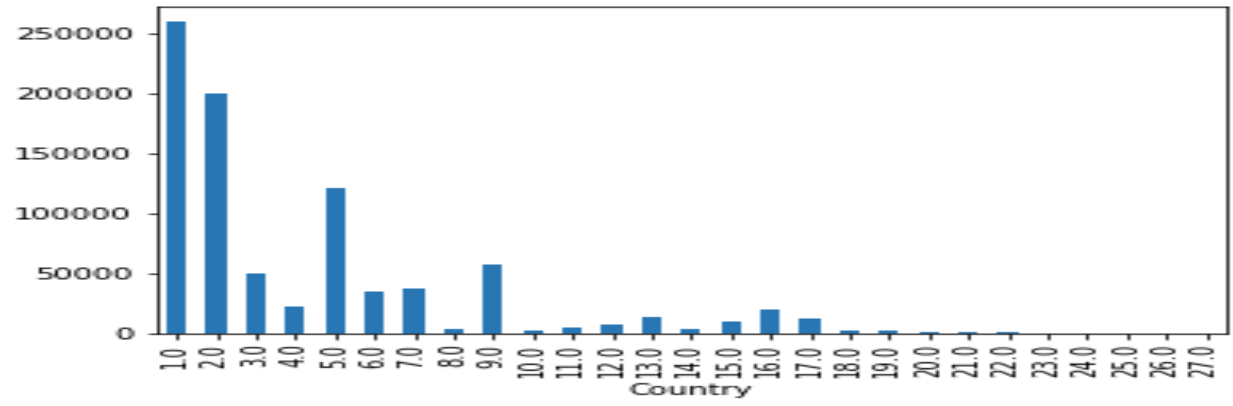


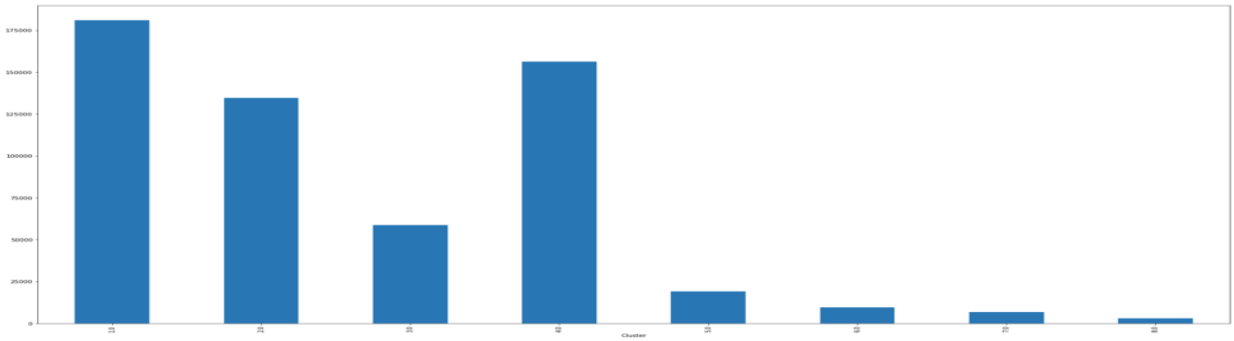
EMMISSION_LEVEL Feature Analysis:

Now after analyzing the Vehicle_MRK_TYPE I had wanted to analyze the EMMISSION_LEVEL feature. This feature has just 2 values in them it was either 1 or 2. Similar to what was done in the Vehicle_MRK_type, I have checked the number values of 1 and 2. The below pie chart shows the number of 1's and 2's in the feature.

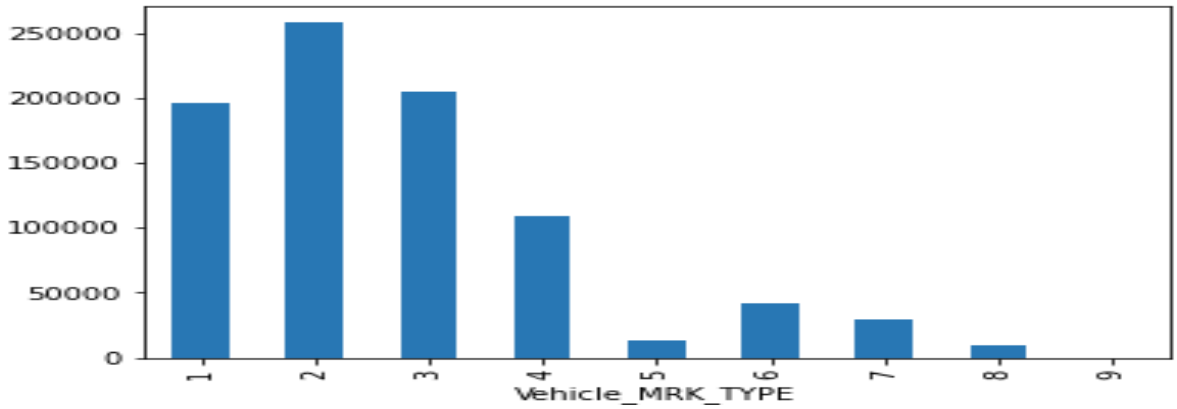


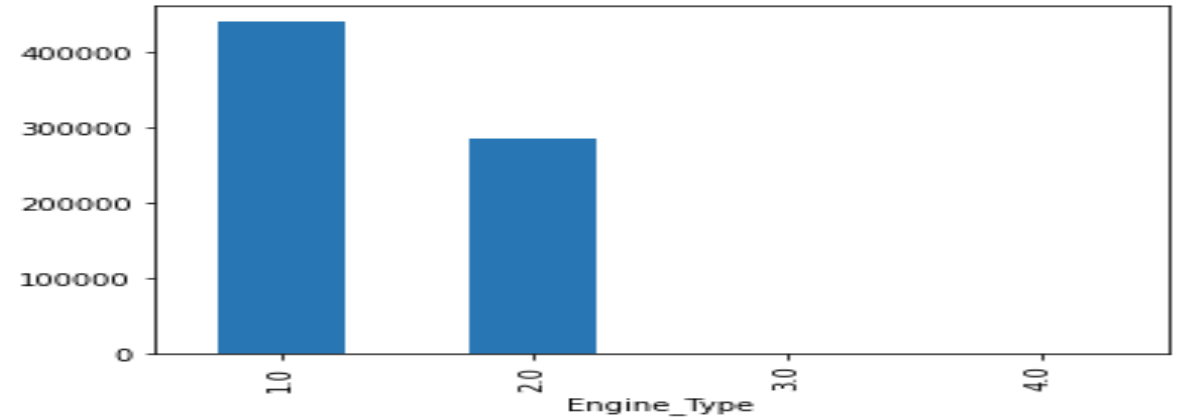
Straightaway we can see that there are a lot more 1's than 2's. This is called class imbalance. Hence, I thought this will be an interesting problem, as it has this problem of class imbalance. I chose this problem for solving my classification process. I had done some more analysis to get a better idea on the feature.





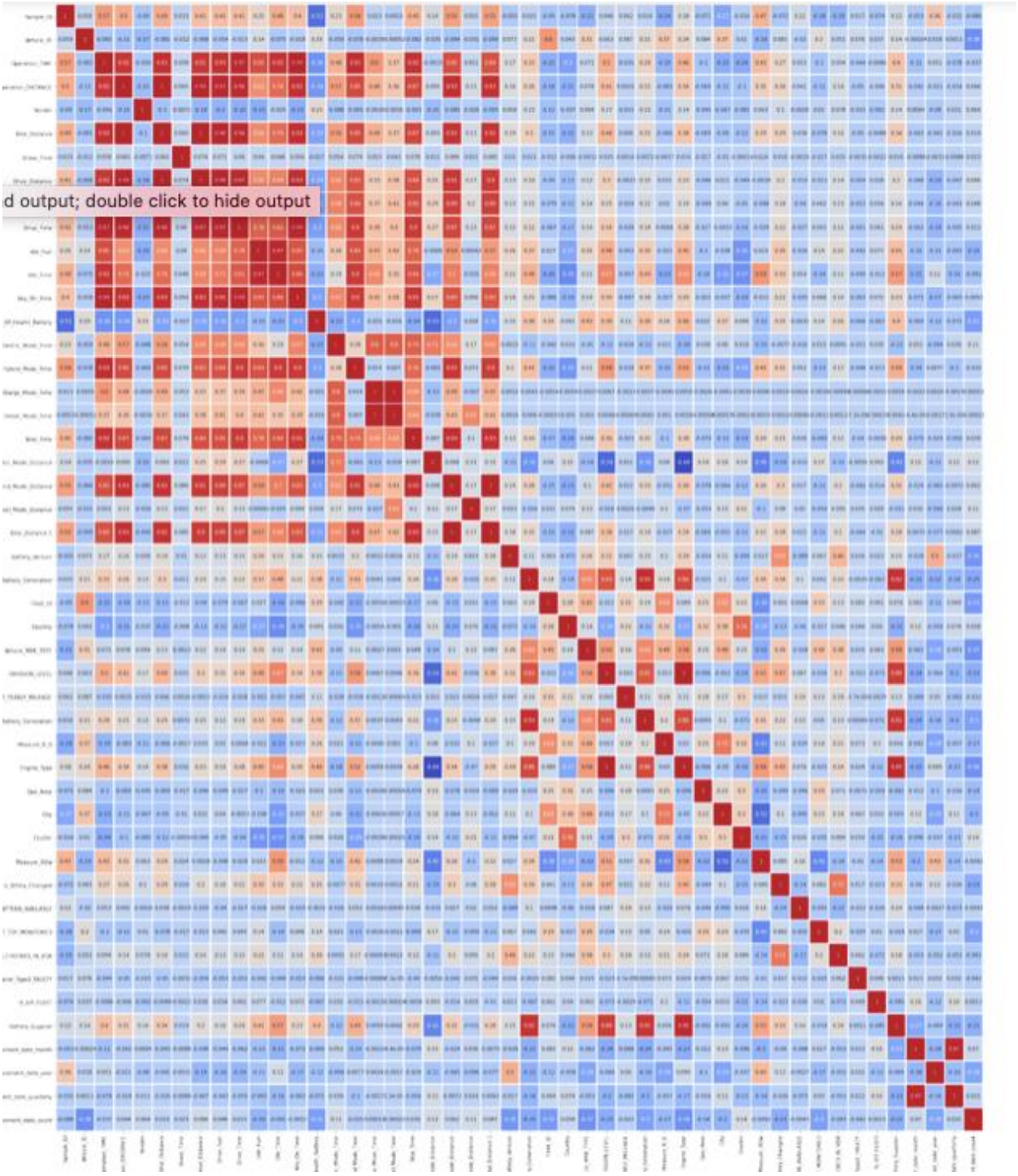
The above the figure shows the cluster-wise emission levels. The below figure shows the EMMISSION_LEVEL, per ‘Vehicle_MRK_TYPE’. The below figure is shown below





I have also done a similar analysis for the emission level separately. In the next step I have taken the features which are positive correlation with the EMMISSION_LEVEL feature and created a smaller dataframe.

This dataframe consisted of 16 features. I have selected it based on the below heatmap.



Class Imbalance:

Class imbalance is a scenario in which there the number of occurrences of one type of class is more than the other class. It is very important to handle this as it could lead to bias while implementing our machine learning algorithms.

To handle this problem, I have performed under sampling of the majority class, and from the pie chart above we can see that class 1 has greater number of 1's than 2's. The below figure shows the differences before and after down sampling.

```
emission_level_target=data['EMISSION_LEVEL'].value_counts()
print('Class label 1:',emission_level_target[1])
print('Class label 2:',emission_level_target[2])
print('Proportion',round(emission_level_target[1]/emission_level_target[2],2),':1')
```

```
Class label 1: 577046
Class label 2: 284870
Proportion 2.03 :1
```

```
print('Class label 1:',emission_level_target[1])
print('Class label 2:',emission_level_target[2])
```

```
Class label 1: 284870
Class label 2: 284870
```

Now we can see that the dataset is balanced as it has equal number of entries.

Data Imputation:

The data did have some more missing values even after down sampling; hence I have used the MICE algorithm to impute the missing values. Here I have used the "miceforest" library to impute the missing values. The below screenshot shoes the percentage of missing values of the features present in it.

```
Vehicle_ID - 0.0%
Sample_ID - 0.0%
Operation_TIME - 0.0%
Total_Distance - 0.0%
Idle_Fuel - 0.0%
Idle_Time - 0.0%
Hybrid_Mode_Time - 0.0%
Total_Time - 0.0%
Hybrid_Mode_Distance - 0.0%
Country - 0.0%
Vehicle_MRK_TYPE - 0.0%
Engine_Type - 0.0%
Geo_Area - 0.0%
City - 0.0%
Cluster - 0.0%
EMISSION_LEVEL - 0.0%
```

As we can see the missing values are not present anymore. This cleaned dataframe is stored in another csv file which will be used hereon.

```
data.to_csv(r'exported-dataframe.csv', index = False, header=True)
```

```
exp_data=pd.read_csv('exported-dataframe.csv')
```

Classification Algorithm:

After this I had started to work on the implementation of machine learning algorithms. Before implementing the machine learning algorithm, I have applied the StandardScaler function in the dataset, this will cause the mean to be 0 and standard deviation of 1.

Following which I have split the data into train and test set, so that we can implement the ML algorithms.

click to expand output; double click to hide output

```
TrainX,TestX,TrainY,TestY=train_test_split(exp_data,em,test_size=0.30,shuffle=True,random_state=2)
```

After this I had have started to implement the ML algorithms, and in this case. I have implemented 3 algorithms, namely KNN, Logistic Regression, MLP Classifier. The below table shows the performance of the each of the models.

Performance Metric	MLP Classifier	Logistic Regression	KNN Classifier
Accuracy	0.54014111	0.86857	0.91095679

From the above table we can see that the KNN classifier works better than all the algorithms.

Time Series Forecasting:

This task is a regression task, where in I have chosen to find the number of battery replacements which would happen monthly. One of the main reasons why this problem was chosen was the amount of feature engineering which goes into developing this. Another reason is because it is a very interesting business case, and would be useful to know this information as it would be useful for manufacturing, supplier chain management of the company etc. It also presents a new challenge to me personally of working on time series forecasting problem.

This is a regression problem where we can use the battery replacement, Battery Mounted Generation, to create new features such as Battery_Replacement_count, Battery_Replacement_monthwise etc.

The below screenshot the number of new features which were used in the process, for this process.

	Battery_Replacement_date_month	Battery_Replacement_date
0	8.0	2015-08-01
1	8.0	2015-08-01
2	8.0	2015-08-01
3	8.0	2015-08-01
4	8.0	2015-08-01
...	...	...
861911	NaN	NaN
861912	5.0	2019-05-01
861913	NaN	NaN
861914	NaN	NaN
861915	NaN	NaN

	Battery_Replacement_date_year	Battery_Replacement_date
0	2015.0	2015-08-01
1	2015.0	2015-08-01
2	2015.0	2015-08-01
3	2015.0	2015-08-01
4	2015.0	2015-08-01
...	...	...
861911	NaN	NaN
861912	2019.0	2019-05-01
861913	NaN	NaN
861914	NaN	NaN
861915	NaN	NaN

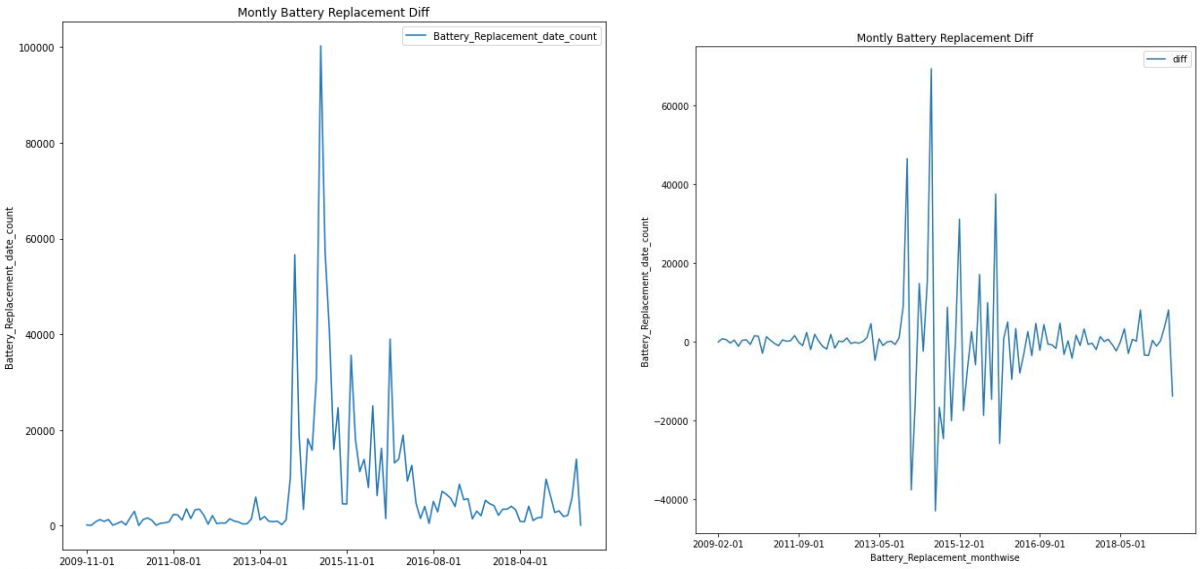
	Battery_Replacement_date_quarterly	Battery_Replacement_date
0	3.0	2015-08-01
1	3.0	2015-08-01
2	3.0	2015-08-01
3	3.0	2015-08-01
4	3.0	2015-08-01
...	...	...
861911	NaN	NaN
861912	2.0	2019-05-01
861913	NaN	NaN
861914	NaN	NaN
861915	NaN	NaN

	Battery_Replacement_date_day_name	Battery_Replacement_date
0	Saturday	2015-08-01
1	Saturday	2015-08-01
2	Saturday	2015-08-01
3	Saturday	2015-08-01
4	Saturday	2015-08-01
...	...	...
861911	NaN	NaN
861912	Wednesday	2019-05-01
861913	NaN	NaN
861914	NaN	NaN
861915	NaN	NaN

	Battery_Replacement_date_count	Battery_Replacement_date
0	16064.0	2015-08-01
1	16064.0	2015-08-01
2	16064.0	2015-08-01
3	16064.0	2015-08-01
4	16064.0	2015-08-01
...	...	...
861911	NaN	NaN
861912	30.0	2019-05-01
861913	NaN	NaN
861914	NaN	NaN
861915	NaN	NaN

The last above screenshot shows the use Battery_Replacement_Date and Mounted_battery_Generation to find the count of the battery replacement on the daily basis. From the screenshot we can see that there are many duplicate dates present in the dataframe.

To overcome this, I have dropped the duplicates, after removing all the duplicates, I have sorted all the dates in the ascending order. Following this I have plotted the monthly battery sales current and taken the difference in battery replacement numbers.



After calculating the difference, I have checked the forecasting of battery replacements. In order to achieve this I have introduced a look back period which will be 1 to 12, as we are calculating month wise. So, what we need to do is to create columns from lag_1 to lag_12 and assign values by using **shift()** method.

Regression Algorithm:

In this model, we will basically fit the linear regression model and calculate the adjusted Rsquared. Following which I have used the MinMax scaler to and then split the train and test set. The below screenshot shows the train-test split.

```
from sklearn.preprocessing import MinMaxScaler
data_model = data_supervised.drop(['Battery_Replacement_date_count','Battery_Replacement_monthwise'],axis=1)
#split train and test set
train_set, test_set = data_model[0:-20].values, data_model[-20:].values
```

```
#apply Min Max Scaler
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = scaler.fit(train_set)
# reshape training set
train_set = train_set.reshape(train_set.shape[0], train_set.shape[1])
train_set_scaled = scaler.transform(train_set)
# reshape test set
test_set = test_set.reshape(test_set.shape[0], test_set.shape[1])
test_set_scaled = scaler.transform(test_set)
```

```
X_train, y_train = train_set_scaled[:, 1:], train_set_scaled[:, 0:1]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test, y_test = test_set_scaled[:, 1:], test_set_scaled[:, 0:1]
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

Following which I have also implemented the machine learning algorithm, in this case I have used the keras library. The Long term short memory (LSTM) algorithm is used in this case, it is a very common deep learning algorithm for these cases. From the below code, we can see that the error decreases after every epoch. The below screenshot shows the same.

```
Epoch 1/100
41/41 [==============================] - 0s 5ms/step - loss: 0.4427
Epoch 2/100
41/41 [==============================] - 0s 4ms/step - loss: 0.3627
Epoch 3/100
41/41 [==============================] - 0s 5ms/step - loss: 0.3767
Epoch 4/100
41/41 [==============================] - 0s 4ms/step - loss: 0.3739
Epoch 5/100
41/41 [==============================] - 0s 4ms/step - loss: 0.3692
Epoch 6/100
41/41 [==============================] - 0s 8ms/step - loss: 0.3649
Epoch 7/100
41/41 [==============================] - 0s 6ms/step - loss: 0.3603
Epoch 8/100
41/41 [==============================] - 0s 4ms/step - loss: 0.3561
Epoch 9/100
41/41 [==============================] - 1s 16ms/step - loss: 0.3503
Epoch 10/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3463
Epoch 11/100
41/41 [==============================] - 0s 8ms/step - loss: 0.3380
Epoch 12/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3313
Epoch 13/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3196
Epoch 14/100
41/41 [==============================] - 0s 10ms/step - loss: 0.3019
Epoch 15/100
41/41 [==============================] - 0s 7ms/step - loss: 0.2911
Epoch 16/100
41/41 [==============================] - 1s 13ms/step - loss: 0.2856
Epoch 17/100
41/41 [==============================] - 0s 7ms/step - loss: 0.2888
Epoch 18/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2759
Epoch 19/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2729
Epoch 20/100
41/41 [==============================] - 0s 10ms/step - loss: 0.2616
Epoch 21/100
41/41 [==============================] - 0s 11ms/step - loss: 0.2552
Epoch 22/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2427
Epoch 23/100
41/41 [==============================] - 0s 12ms/step - loss: 0.2345
Epoch 24/100
41/41 [==============================] - 0s 5ms/step - loss: 0.2228
Epoch 25/100
41/41 [==============================] - 0s 6ms/step - loss: 0.2147
Epoch 26/100
41/41 [==============================] - 0s 7ms/step - loss: 0.2049
Epoch 27/100
41/41 [==============================] - 0s 4ms/step - loss: 0.1981
```

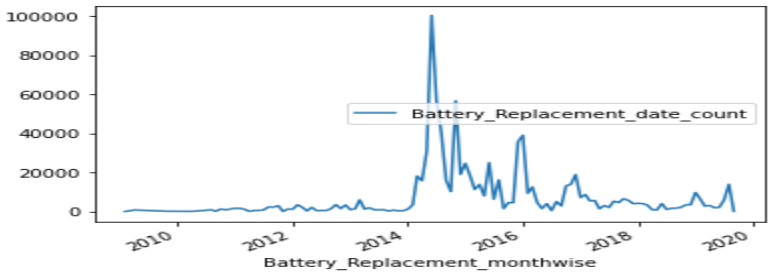
```
Epoch 1/100
30/30 [==============================] - 0s 11ms/step - loss: 0.4490
Epoch 2/100
30/30 [==============================] - 0s 11ms/step - loss: 0.3585
Epoch 3/100
30/30 [==============================] - 0s 5ms/step - loss: 0.3581
Epoch 4/100
30/30 [==============================] - 0s 6ms/step - loss: 0.3568
Epoch 5/100
30/30 [==============================] - 0s 7ms/step - loss: 0.3533
Epoch 6/100
30/30 [==============================] - 0s 4ms/step - loss: 0.3504
Epoch 7/100
30/30 [==============================] - 0s 4ms/step - loss: 0.3480
Epoch 8/100
30/30 [==============================] - 0s 7ms/step - loss: 0.3458
Epoch 9/100
30/30 [==============================] - 0s 6ms/step - loss: 0.3440
Epoch 10/100
30/30 [==============================] - ETA: 0s - loss: 0.230 - 0s 6ms/step - loss: 0.3423
Epoch 11/100
30/30 [==============================] - 0s 5ms/step - loss: 0.3408
Epoch 12/100
30/30 [==============================] - 0s 4ms/step - loss: 0.3394
Epoch 13/100
30/30 [==============================] - 0s 6ms/step - loss: 0.3382
Epoch 14/100
30/30 [==============================] - 0s 6ms/step - loss: 0.3370
Epoch 15/100
30/30 [==============================] - 0s 4ms/step - loss: 0.3360
Epoch 16/100
30/30 [==============================] - 0s 8ms/step - loss: 0.3350
Epoch 17/100
30/30 [==============================] - 0s 6ms/step - loss: 0.3340
Epoch 18/100
30/30 [==============================] - 0s 5ms/step - loss: 0.3331
```

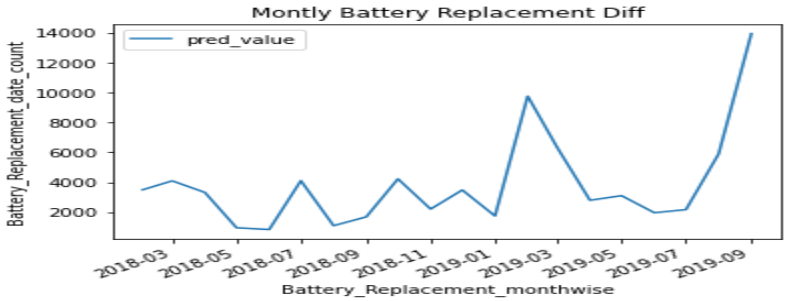
Following which I have implemented the model in the training and test sets. The screenshot is shown below for both of them.

```
y_pred_Train = model.predict(X_train,batch_size=2)
print(y_pred_Train)
```

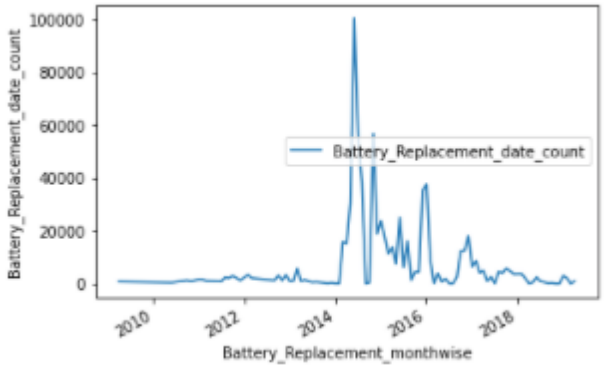
```
y_pred = model.predict(X_test,batch_size=2)
```

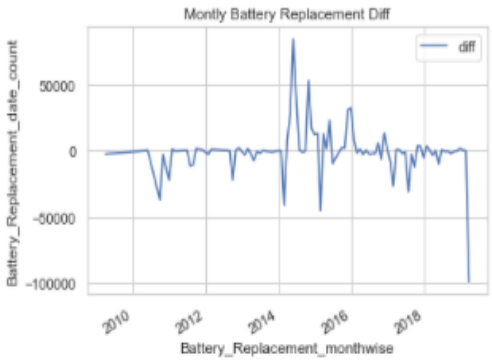
Following this we will be predicting the monthwise sales, the y_pred and monthwise predictions are given below. The plot is shown below.





Following which I had also done the same prediction for the top 2 countries where the battery replacements were high.





	City_wise	clusterwise_sales	Geo_area_sales	Vehicle_MRK_Type_sales	pred_value	Battery_Replacement_monthwise
13	50.0	1.0	1.0	6	94	2010-06-01
6	81.0	1.0	1.0	6	1282	2011-03-01
14	1.0	1.0	1.0	6	461	2011-07-01
15	1.0	1.0	1.0	6	841	2012-03-01
5	1.0	1.0	1.0	6	692	2012-11-01
7	1.0	1.0	1.0	6	1166	2013-06-01
16	79.0	1.0	1.0	6	2135	2013-07-01
4	79.0	1.0	1.0	6	511	2013-08-01
8	1.0	1.0	1.0	6	876	2013-11-01
17	1.0	1.0	1.0	8	589	2013-12-01
11	58.0	1.0	1.0	6	140	2014-01-01
10	1.0	1.0	1.0	6	954	2014-09-01
0	55.0	1.0	1.0	6	1291	2018-03-01
3	4.0	1.0	1.0	1	204	2018-05-01
9	4.0	1.0	1.0	1	173	2018-07-01
2	50.0	1.0	1.0	6	304	2018-09-01
1	50.0	1.0	1.0	6	2109	2018-10-01
18	119.0	1.0	1.0	6	370	2018-11-01
12	4.0	1.0	1.0	1	151	2018-12-01
19	54.0	1.0	1.0	6	79	2019-03-01

Algorithm Performance:

The algorithm works well with Country 1.0 as the RMSE for the train set around 1.53, and for the test set was 1.2989, but it did not work well for Country 2.0 as the train set RMSE is 0.05464,but the test set RMSE was 3.81, which is a case of overfitting.

Unsupervised Learning:

Unsupervised learning refers to the use of artificial intelligence (AI) algorithms to identify patterns in dataset containing data points that are neither classified nor labelled.

The algorithms are thus allowed to classify, label and/or group the data points contained withing the datasets without having any external guidance in performing that task. In other words, unsupervised learning allows the system to identify patterns within datasets on its own.

In supervised learning, an AI system will group unsorted information according to similarities and difference even though there are no categories provided.

Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. Additionally, subjecting a system to unsupervised learning is one way of testing AI.

In this unsupervised learning task, I am finding anomalies in the State_of_Battery_health feature using different features which were created such dayoftheweek, weekday, daylight etc.

Feature Engineering:

To get new features out of existing ones, I have done feature engineering to create features such as Electric Mode Speed, Hours, daylight, Dayoftheweek, time epoch etc. I have provided the screenshot for some of the feature engineering steps I have done

```
clustering['Electric_Mode_SPEED']=(clustering['Electric_Mode_Distance'])/(clustering['Electric_Mode_Time'])
```

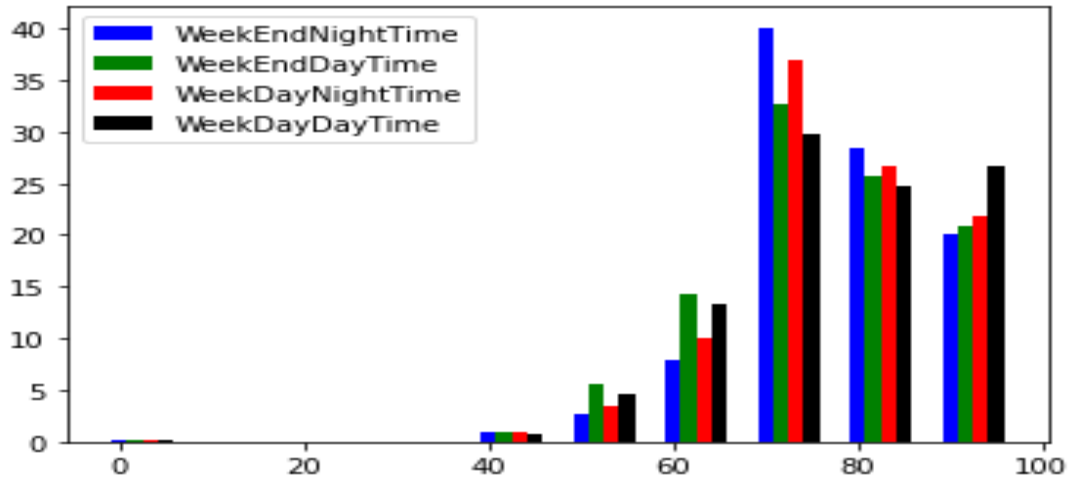
```
clustering['Hours']=pd.to_datetime(clustering['SEND_TIME']).dt.hour
```

```
clustering['daylight']=((clustering['Hours']>=7) & (clustering['Hours']<=22)).astype(int)
```

```
clustering['Dayoftheweek']=pd.to_datetime(clustering['S_Date']).dt.dayofweek
```

```
clustering['WeekDay'] = (clustering['Dayoftheweek'] < 5).astype(int)
```

Following which I have checked the “State_of_Battery_Health” feature at different times of the day. The graph below shows the Battery State of health the different times.



Following which there are several missing values in the dataframe, namely there was 1681 empty values present in the dataset. The below screenshot is shown below

	Vehicle_ID	SEND_TIME	Electric_Mode_Time	Electric_Mode_Distance	State_Of_Health_Battery	Country	City	Geo_Area	Vehicle_MRK_TYPE	Battery_V
46264	52	NaT	1.720000	0.520	100.00	1.0	1.0	1.0	6	
46265	52	NaT	1.720000	0.520	100.00	1.0	1.0	1.0	6	
46280	52	NaT	2.620000	0.520	100.00	1.0	1.0	1.0	6	
46281	52	NaT	2.620000	0.520	98.50	1.0	1.0	1.0	6	
46284	52	NaT	3427.082222	5704.842	100.00	1.0	1.0	1.0	6	
...	...	...	...	...	...	...	...	...	...	
861323	3945	NaT	70.010000	94.730	100.00	21.0	41.0	1.0	7	
861325	3945	NaT	150.170000	241.240	100.00	21.0	46.0	1.0	7	
861329	3945	NaT	67.490000	94.640	89.50	21.0	36.0	1.0	7	
861330	3945	NaT	177.580000	288.510	100.00	21.0	40.0	1.0	7	
861338	3945	NaT	150.790000	242.780	86.75	21.0	30.0	1.0	7	

1681 rows × 16 columns

The missing values where them dropped from the dataframe using the dropna() function.

Anomaly detection algorithms:

In this project I have used the following anomaly detection algorithms

1. Elliptic Envelope
2. Isolation Forest
3. Local Outlier Factor
4. OneclassSVM

Elliptic Envelope:

The Elliptic Envelope method fits a multivariant gaussian distribution to the dataset. Use the contamination hyperparameter to specify the percentage of observations the algorithm will assign the outliers.

The sklearn library is one which is used for using this algorithm.

```
: from sklearn.covariance import EllipticEnvelope
```

Isolation Forest:

Isolation forest is an unsupervised learning algorithm for anomaly detection that works on the principle of isolating anomalies, instead of the common techniques of profiling normal points.

Isolation forest is a method which in principle is like the well-known and popular Random Forest. This is a tree-based algorithm wherein partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.

The sklearn library is used for this algorithm as well.

```
from sklearn.ensemble import IsolationForest
```

Local Outlier Factor:

Local Outlier Factor (LOF) is an unsupervised machine learning algorithm that uses the density of datapoints in the distribution as a key factor to detect outliers.

LOF compares the density of any given data point to the density of its neighbors. Since outliers come from low-density areas, the ratio will be higher for anomalous data points.

The sklearn library is used for this algorithm as well.

```
from sklearn.neighbors import LocalOutlierFactor
```

OneClassSVM:

OneclassSVM (one-class Support Vector Machine) is an unsupervised learning algorithm that is trained only on the 'normal' data. It is used to detect the outliers and anomalies in the dataset. Based on Support Vector Machines (SVM) evaluation, the One-class SVM applies a one-class classification method for novelty detection.

The sklearn library is used for this algorithm.

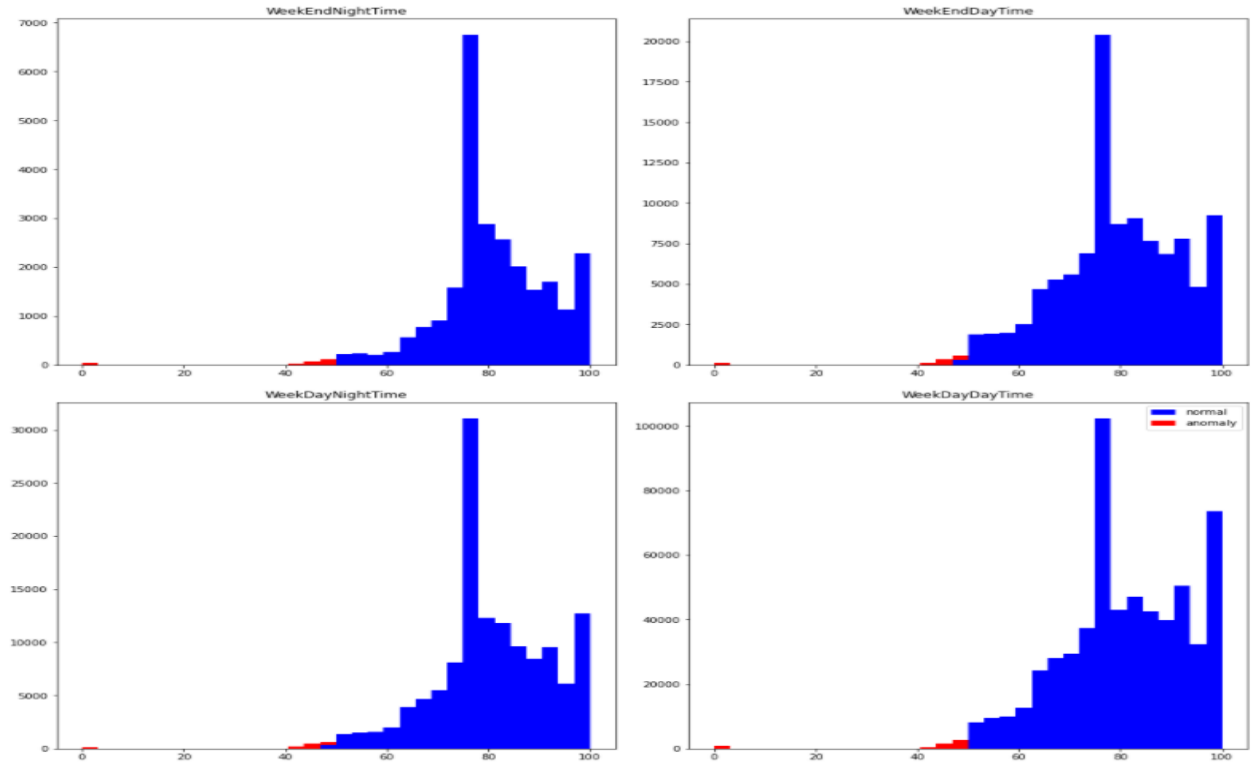
```
from sklearn.svm import OneClassSVM
```

Algorithms Implementation:

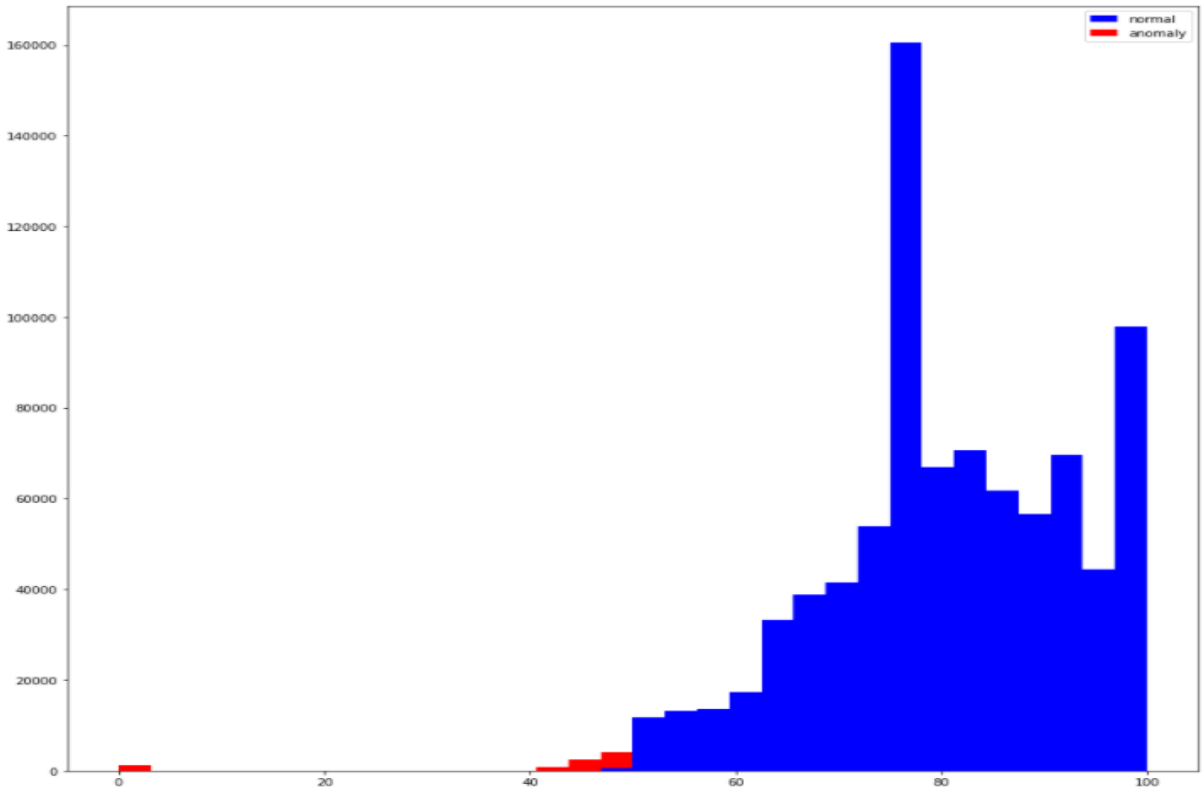
The first algorithm which I have used for anomaly detection is the Elliptic Envelope algorithm. To implement the algorithm, I have split the dataset based on the category, in this case we have split in into 4 categories. The below screenshot shows the split into different datasets.

```
# creation of 4 differents data set based on categories defined before
cluster_class0 = clustering.loc[clustering['categories'] == 0, 'State_Of_Health_Battery']
cluster_class1 = clustering.loc[clustering['categories'] == 1, 'State_Of_Health_Battery']
cluster_class2 = clustering.loc[clustering['categories'] == 2, 'State_Of_Health_Battery']
cluster_class3 = clustering.loc[clustering['categories'] == 3, 'State_Of_Health_Battery']
```

The algorithm is implemented after this step, and the anomaly performance is shown in the below figure for



After which we have concatenated all the different data frames based on different categories. I below figure shows the anomalies after concatenating together.



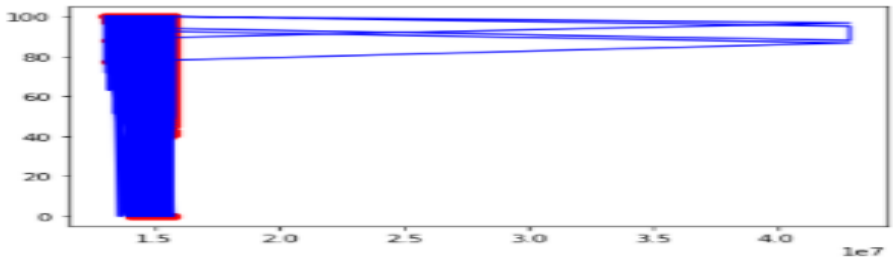
The next algorithm which I have used in the project was the Isolation forest algorithm. I have implemented StandardScaler before applying the algorithm. After, that I have implemented the algorithm in the dataset.

The below figure the number of anomaly points detected by the algorithm

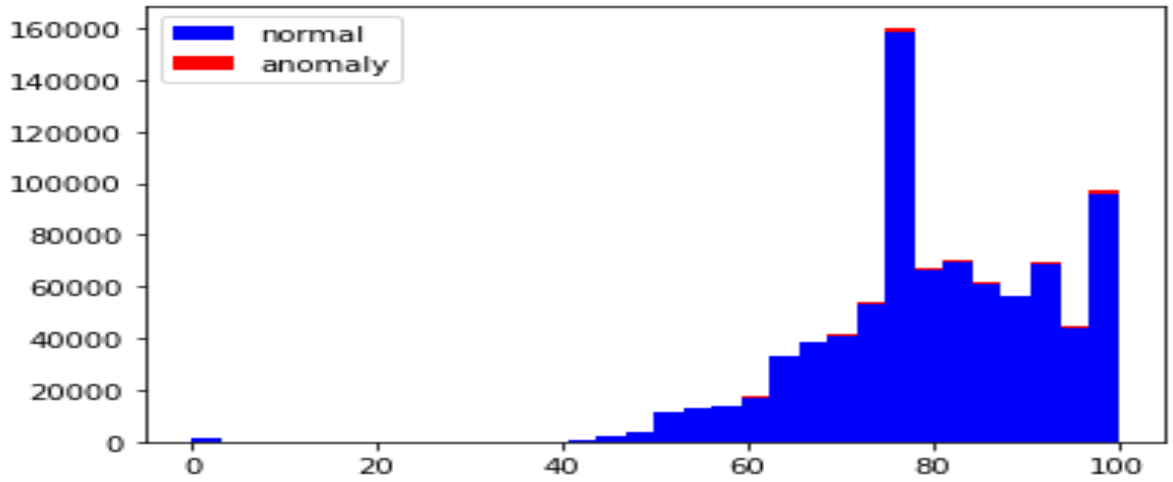
```
y_pred=model.predict(cluster_isolationForest)
clustering['anomaly25'] = pd.Series(y_pred)
clustering['anomaly25'] = clustering['anomaly25'].map( {1: 0, -1: 1} )
print(clustering['anomaly25'].value_counts())
```

```
0.0    849648
1.0      8575
Name: anomaly25, dtype: int64
```

The below screenshot shows the anomaly with respect to time.



The below figure shows the anomaly points which are detected in the dataset,

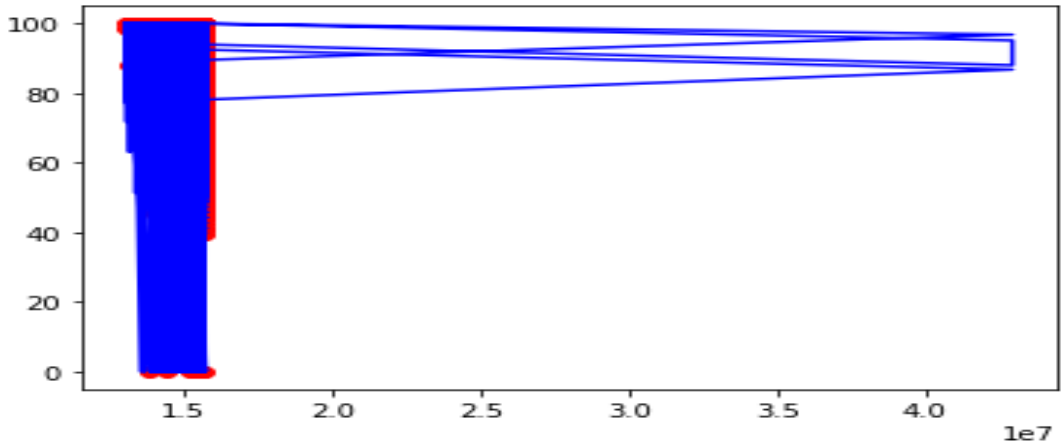


In the next step, I have implemented the Local Outlier factor algorithm, and like before I have implemented the StandardScaler function before the algorithm. The below screenshot shows the number of anomaly points found by the algorithm

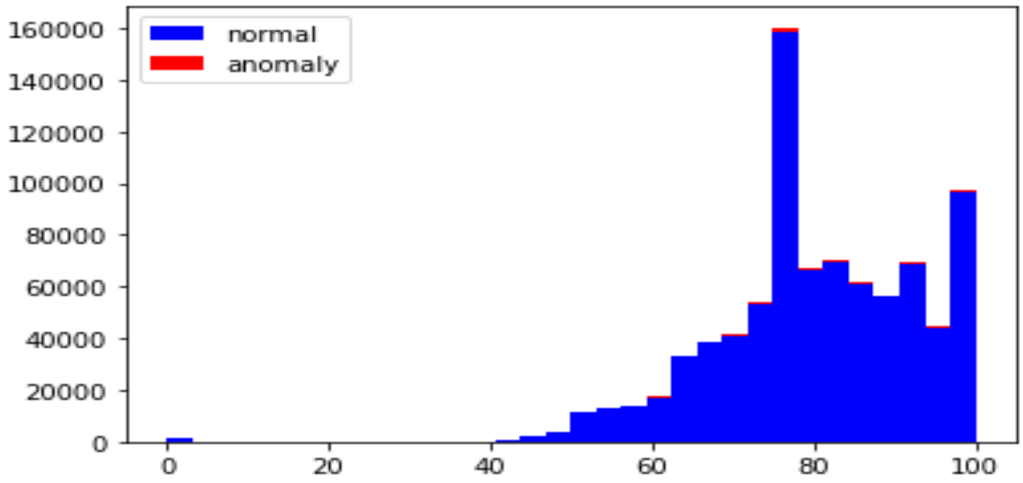
```
y_pred=model.predict(cluster_localoutFactor)
clustering['anomaly26'] = pd.Series(y_pred)
clustering['anomaly26'] = clustering['anomaly26'].map( {1: 0, -1: 1} )
print(clustering['anomaly26'].value_counts())
```

```
0.0    849983
1.0      8240
Name: anomaly26, dtype: int64
```

The next screenshot shoes the anomaly points with respect to time.



The next figure shows the anomaly detected points with respect the State of Battery Health.

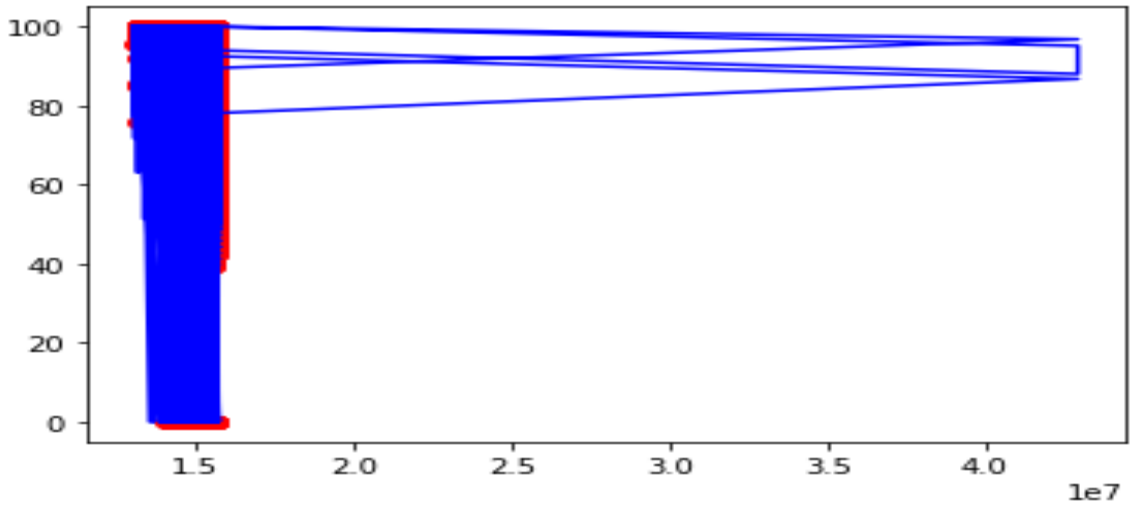


The last algorithm which I have implementing is the OneclassSVM algorithm. The below screenshot shows the number of anomaly points detected by the algorithm.

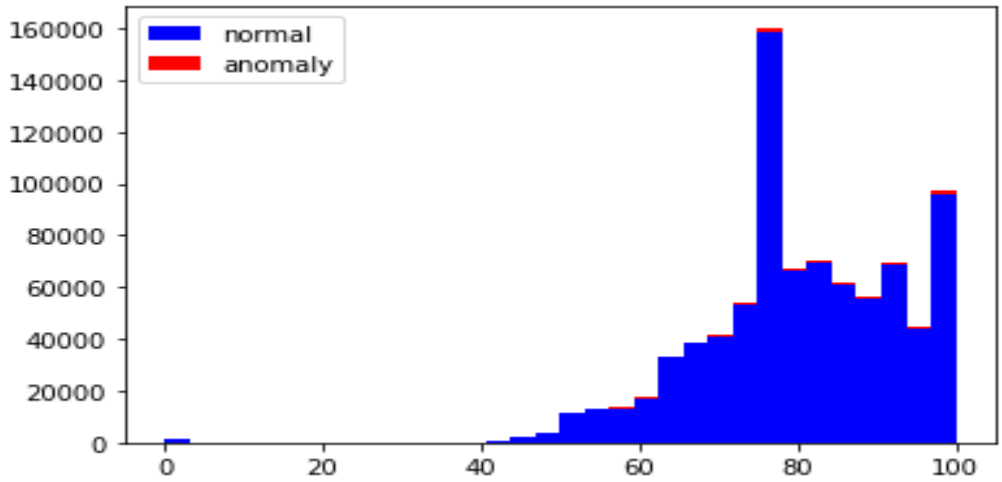
```
model =  OneClassSVM(nu=0.95 * outliers_fraction)
cluster_ClassSVM = pd.DataFrame(np_scaled)
model.fit(cluster_ClassSVM)
# add the data to the main
y_pred=model.predict(cluster_ClassSVM)
clustering['anomaly27'] = pd.Series(y_pred)
clustering['anomaly27'] = clustering['anomaly27'].map( {1: 0, -1: 1} )
print(clustering['anomaly27'].value_counts())
```

```
0.0    848515
1.0      9708
Name: anomaly27, dtype: int64
```

The below screenshot shoes the anomalies in the battery health based on the time.



In the next screenshot I have plotted the anomaly detection of the battery health state.



The below table shows the MSE of all the models implemented in the project.

Performance metric	OneClassSVM	LocalOutlierFactor	Isolation Forest
Mean Square Error	6449.8817999	6449.501804489	6449.802499

Conclusion:

Classification: From the above tasks we can see that the accuracy of KNN is better than all the other models.

Regression: From the above task we can see that the prediction of the LSTM model performs better for country 1.0 but does not perform well for country 2.0.

Unsupervised Learning: Based only on the MSE we can see that LocalOutlierFactor algorithm as it has the least score.