INTRODUCTION TO DATA SCIENCE

Group 14 -- Project phase 2

Additional Models

Name	Student ID
Pham, Quoc Huy	2299356
Hussain, Zakiuddin	2338350
Lee, Daeul Haven	2308018
Preetham	2288949
Jayanth	2288552
Srikavya	2311351

1. Import the Library and read the data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.model_selection import train_test_split
```

```
selected_features = ["TP2", "H1", "DV_pressure", "Reservoirs", "Oil_temperature", "Motor_current", "Oil_leve
```

```
data = pd.read_csv("Group_14_Clean_Data.csv")
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
0	562564	2020-04-18 00:00:01	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
1	562565	2020-04-18 00:00:13	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
2	562566	2020-04-18 00:00:24	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
3	562567	2020-04-18 00:00:36	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	562568	2020-04-18 00:00:49	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1

```
# Choose only features selected from Question 2
#Shuffle the data
data = data.sample(frac = 1)
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
32984	1484156	2020-08-27 12:38:48	-0.010	8.212	8.198	-0.016	8.214	64.100	0.0400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0
56604	416777	2020-03-27 15:17:44	10.376	9.996	-0.006	-0.014	9.992	71.875	6.0925	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0
52785	1512857	2020-08-31 10:27:03	-0.014	9.382	9.368	-0.018	9.382	66.950	0.0450	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0
54798	588344	2020-04-21 03:57:12	-0.014	8.844	8.832	-0.024	8.846	55.500	0.0425	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0
30977	231046	2020-03-03 06:38:59	9.760	9.378	-0.012	-0.026	9.372	73.450	5.9900	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0

2. XGBOOST

- XGBoost, a Gradient Boosted decision tree implementation, excels in Kaggle Competitions.
- It sequentially creates decision trees, assigning weights to variables that are adjusted based on prediction errors.
- This ensemble method, with optimizations like the Approximate Greedy Algorithm and Cash-Aware Access, proves effective for regression, classification, ranking, and user-defined prediction problems.

reference:
https://xgboost.readthedocs.io/en/stable/python/python_intro.html
https://www.geeksforgeeks.org/xgboost/
https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/

```
#!pip install xgboost
import xgboost as xgb
```

```
X = data[selected_features]
y = data['status']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

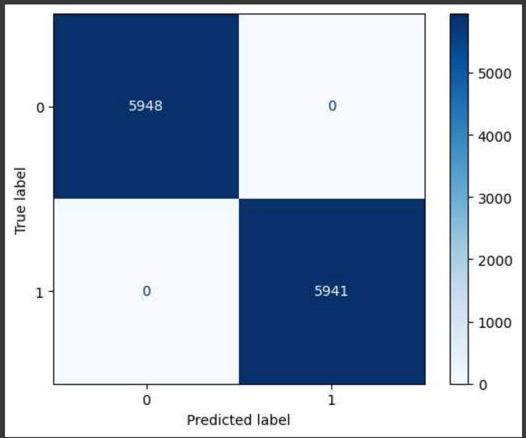
```
class XGBoost():
  def __init__(self):
    self._model = xgb.XGBClassifier()
    self._encoder = OneHotEncoder(categories='auto')

  def fit(self, x, y ):
    self._model.fit(x,y)

  def predict(self, x):
    y_pred = self._model.predict(x)
    return self._encoder.fit_transform(y_pred[:, np.newaxis]).toarray()
```

```
# Fitting XGBoost to the training data
model = XGBoost()
model.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = model.predict(X_test)
```

```
y_pred = np.argmax(y_pred, axis = 1)
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

print(classification_report(y_test, y_pred))

disp = ConfusionMatrixDisplay(cm, display_labels=['0', '1'])
disp.plot(cmap='Blues', values_format='d')
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5948
           1       1.00      1.00      1.00      5941

    accuracy                           1.00     11889
   macro avg       1.00      1.00      1.00     11889
weighted avg       1.00      1.00      1.00     11889
```



3. Extreme Learning Machine

- Extreme Learning Machine is a variation of Feed forward neural network with just 1 hidden layer.
- Extreme Machine Learning does not use iterative method such as gradient descent to tuning the weights, instead it use linear algebra to solve for the optimal solution.

- Extreme Machine Learning is believed to be able to approximate any abitrary function given sufficient number of hidden units and data to learn.

```
import pandas as pd
import numpy as np
from timeit import timeit as time
import cupy as cp
from  cupy.linalg import pinv as pinv2

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

```
selected_features = ["TP2", "H1", "DV_pressure", "Reservoirs", "Oil_temperature", "Motor_current", "Oil_level", 'status']
```

```
data = pd.read_csv("Group_14_Clean_Data.csv")
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
0	562564	2020-04-18 00:00:01	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
1	562565	2020-04-18 00:00:13	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
2	562566	2020-04-18 00:00:24	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1
3	562567	2020-04-18 00:00:36	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	562568	2020-04-18 00:00:49	-0.018	8.248	8.238	-0.024	8.248	49.45	0.04	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1

```
# Choose only features selected from Question 2
#Shuffle the data
data = data.sample(frac = 1)
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
5197	567761	2020-04-18 14:27:31	9.040	8.858	-0.006	2.024	8.858	75.500	5.7275	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
14297	894090	2020-06-05 19:01:58	8.296	8.108	-0.004	2.200	8.108	75.925	5.5075	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
4083	566647	2020-04-18 11:23:28	8.992	8.798	-0.006	2.014	8.800	75.525	5.7075	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
10756	842839	2020-05-30 05:16:58	7.274	8.362	-0.010	1.660	8.364	76.275	5.2675	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
27418	907333	2020-06-07 12:09:00	7.994	7.928	-0.002	1.892	7.930	75.725	5.4150	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1

```
X = data[selected_features]
y = data['status']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#this Extreme Machine Learning class is implemented specificly to work on GPU
#this will not run on a CPU machine

class ExtremeLearningMachine():
  def __init__(self, hidden_size = 10, threshold = 0.5):
    self._hidden_size = hidden_size
    self._input_size  = None
    self._w = None
    self._b = None
    self._beta = None
    self._threshold = 0.5
    self._encoder = OneHotEncoder(categories='auto')
    self._scaler  = StandardScaler()

  def encode(self, y):
    return self._encoder.fit_transform(y[:, np.newaxis]).toarray()

  def scale(self,x):
    return self._scaler.transform(x)

  def _h(self,x):
    return self._tanh(cp.dot(x, self._w) + self._b )

  @property
  def hidden_size(self):
    return self._hidden_size

  @staticmethod
  def _tanh(x):
    return cp.tanh(x)

  def fit(self, x, y):
    x = self._scaler.fit_transform(x)
    y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()

    x_gpu = cp.asarray(x)
    y_gpu = cp.asarray(y)

    self._input_size = x.shape[1]
    self._w = cp.random.normal( size = [self._input_size, self._hidden_size])
    self._b = cp.random.normal(size = [self._hidden_size])

    H = self._h(x_gpu)
    self._beta = cp.dot(pinv2(H), y_gpu)

    del x_gpu
    del y_gpu
    cp._default_memory_pool.free_all_blocks()


  def predict(self, x):
    x = self.scale(x)
    x_gpu = cp.asarray(x)
    out_gpu = cp.dot(self._h(x_gpu), self._beta)
    out = cp.asnumpy(out_gpu)
    del out_gpu
    cp._default_memory_pool.free_all_blocks()
    return out
```

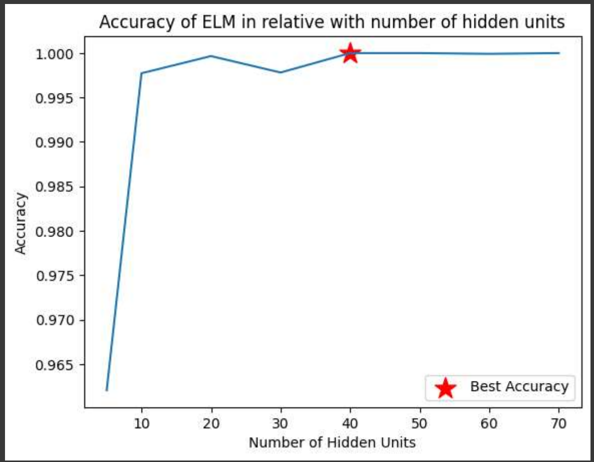
```
num_hidden = [5,10,20,30,40,50,60,70]
scores = []

for h in num_hidden:
  elm = ExtremeLearningMachine(h)
  elm.fit(X_train, y_train)
  y_pred = elm.predict(X_test)
  y_pred = np.argmax(y_pred, axis = 1)
  scores.append(accuracy_score(y_test, y_pred))
  #Clear GPU memmory
  del elm
  cp._default_memory_pool.free_all_blocks()
```

```
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
```

```
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
```

```
best_hidden_size_indx = np.argmax(scores)
best_hidden_size = num_hidden[best_hidden_size_indx]
best_score       = scores[best_hidden_size_indx]
```

```
plt.plot(num_hidden, scores)
plt.scatter(best_hidden_size, best_score, s = 250, marker = "*", color = 'r' , label = 'Best Accuracy')
plt.xlabel('Number of Hidden Units')
plt.ylabel('Accuracy')
plt.title('Accuracy of ELM in relative with number of hidden units' )
plt.legend()
plt.show()
```




```
elm = ExtremeLearningMachine(best_hidden_size)
elm.fit(X_train, y_train)
```

```
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
```

```
y_pred = elm.predict(X_test)
```

```
y_pred = np.argmax(y_pred, axis = 1)

print(f'Best result achieved with {elm.hidden_size} hidden units')
print(classification_report(y_test, y_pred))
c_mat  = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(c_mat, display_labels=['0', '1'])
disp.plot(cmap='Blues', values_format='d')
plt.show()
```

```
Best result achieved with 40 hidden units
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5967
           1       1.00      1.00      1.00      5922

    accuracy                           1.00     11889
   macro avg       1.00      1.00      1.00     11889
weighted avg       1.00      1.00      1.00     11889
```

4. Neural Network with 2 hidden layers

About the Model : The neural network model created for this metro dataset consists of two hidden layers, each with 64 neurons, designed to capture complex patterns and relationships within the data. By utilizing a deep learning approach, the model aims to accurately predict the 'status' variable, effectively handling the non-linear and intricate dependencies likely present in the diverse range of features from the metro operational metrics.

Why TensorFlow and Keras?

Handling Large and Complex Datasets: since metro dataset is large or complex, TensorFlow and Keras can efficiently handle such datasets. Deep learning models, particularly those built with these frameworks, are known for their ability to process and extract patterns from large volumes of data.

Feature Learning: Deep learning models have the capability to automatically learn and extract features from raw data. This can be particularly useful since dataset contains complex patterns or relationships that are not easily captured with traditional machine learning models.

Non-linear Relationships: Neural networks, which can be easily built using TensorFlow and Keras, are adept at capturing non-linear relationships in the data. The metro dataset has intricate dependencies between variables, a neural network might model these relationships more effectively than a simpler linear model.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix, roc_curve, auc

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

```python
# Read the dataset
data = pd.read_csv('Group_14_Clean_Data.csv')

# Choose only features selected from Question 2
#Shuffle the data
data = data.sample(frac = 1)
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
23262	903144	2020-06-06 23:44:43	8.194	7.974	-0.010	2.128	7.976	75.300	5.6025	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
41633	736676	2020-05-14 12:09:29	-0.012	9.574	9.560	-0.022	9.574	58.400	3.4900	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0
14025	893818	2020-06-05 18:17:01	5.582	8.156	-0.006	1.280	8.158	75.675	5.0925	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1
21627	901420	2020-06-06 15:12:59	6.064	8.090	-0.006	1.444	8.090	75.825	5.1375	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1
35681	1514037	2020-08-31 13:41:59	9.026	8.578	-0.018	-0.016	8.576	62.475	5.9575	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0

```python
# Preprocessing the data
selected_features = ["TP2", "H1", "DV_pressure", "Reservoirs", "Oil_temperature", "Motor_current", "Oil_level", 'status']
X = data[selected_features]
y = data['status']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Create the neural network model

class NeuralNet(tf.keras.Model):
  def __init__(self, *args, **kwargs):
    super().__init__(**kwargs)
    self._model = Sequential(name = "Model")
    self._model.add(Dense(32, input_dim=X_train.shape[1], activation='tanh'))
    self._model.add(Dense(32, activation='tanh'))  # First hidden layer
    self._model.add(Dense(32, activation='tanh'))  # Second hidden layer
    self._model.add(Dense(2, activation='softmax'))  # Output layer

    early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
    lr_scheduler = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=0, verbose=1, min_lr=1e-6)

    self._callbacks = [early_stopping, lr_scheduler]
    # Compile the model
    self._model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])

    self._scaler = StandardScaler()
    self._encoder= OneHotEncoder(categories='auto')

  def encode(self, y):
    return self._encoder.transform(y[:, np.newaxis]).toarray()

  def fit(self, x, y, **kwargs):

    x = self._scaler.fit_transform(x)
    y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()

    x = tf.convert_to_tensor(x)
    y = tf.convert_to_tensor(y)
    self._model.fit(x, y , callbacks = self._callbacks,  **kwargs)

  def predict(self, x):
    x = self._scaler.transform(x)
    x = tf.convert_to_tensor(x)
    y_pred = self._model.predict(x)
    return y_pred
```

```python
model = NeuralNet()
```

```python
# Train the model
model.fit(X_train, y_train, epochs=5, validation_split=0.2, verbose = 1)
```

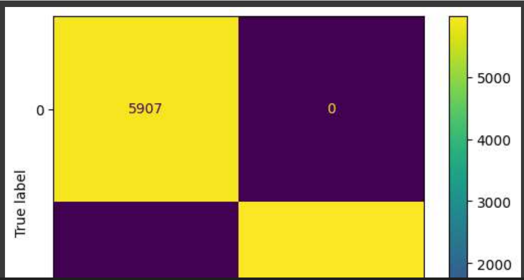
```
Epoch 1/5
<ipython-input-139-2c0b25ae3745>:28: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
1189/1189 [==============================] - 8s 5ms/step - loss: 0.0122 - accuracy: 0.9982 - val_loss: 2.5327e-04 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 2/5
1186/1189 [============================>.] - ETA: 0s - loss: 1.3997e-04 - accuracy: 1.0000
Epoch 2: ReduceLROnPlateau reducing learning rate to 0.0006000000284984708.
1189/1189 [==============================] - 5s 4ms/step - loss: 1.3981e-04 - accuracy: 1.0000 - val_loss: 7.3704e-05 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 3/5
1180/1189 [============================>.] - ETA: 0s - loss: 5.6537e-05 - accuracy: 1.0000
Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0003600000170990825.
1189/1189 [==============================] - 7s 6ms/step - loss: 5.6459e-05 - accuracy: 1.0000 - val_loss: 4.0886e-05 - val_accuracy: 1.0000 - lr: 6.0000e-04
Epoch 4/5
1186/1189 [============================>.] - ETA: 0s - loss: 3.4330e-05 - accuracy: 1.0000
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00021600000327453016.
1189/1189 [==============================] - 5s 4ms/step - loss: 3.4323e-05 - accuracy: 1.0000 - val_loss: 2.7443e-05 - val_accuracy: 1.0000 - lr: 3.6000e-04
```

```python
# Evaluate the model
y_pred = model.predict(X_test)
```

```
372/372 [==============================] - 1s 3ms/step
```

```python
y_pred = tf.math.argmax(y_pred, axis = 1)
```

```python
# Display the confusion matrix
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Accuracy score
print(accuracy_score(y_test, y_pred))

# Classification report
print(classification_report(y_test, y_pred))
```

```
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5907
           1       1.00      1.00      1.00      5982

    accuracy                           1.00     11889
   macro avg       1.00      1.00      1.00     11889
weighted avg       1.00      1.00      1.00     11889
```

5. Ensemble Method

From the performance of all algorithms, we decided to choose 3 algorithm that have highest scores and relatively short training time:

- XGBoost
- Extreme Learning Machine
- Neural network (2 hidden layers)

Since each model will output a tensor of shape [None, 2] represent the probability of the sample belong to class 0 and class 1 respectively, our ensemble method simply calculate the average of 3 predicted probability.

```
# Read the dataset
data = pd.read_csv('Group_14_Clean_Data.csv')

# Choose only features selected from Question 2
# Shuffle the data
data = data.sample(frac = 1)
data.head()
```

	Unnamed: 0	timestamp	TP2	TP3	H1	DV_pressure	Reservoirs	Oil_temperature	Motor_current	COMP	DV_eletric	Towers	MPG	LPS	Pressure_switch	Oil_level	Caudal_impulses	status
57257	517540	2020-04-11 09:23:09	-0.022	10.060	10.056	-0.024	10.058	65.950	3.8475	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	0
43132	297575	2020-03-12 08:42:28	9.758	9.554	-0.008	0.786	9.558	75.925	5.8825	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0
7988	570552	2020-04-18 22:08:34	9.110	8.956	-0.010	2.026	8.954	73.725	5.8200	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1
32841	289916	2020-03-11 02:00:23	-0.012	9.222	9.212	-0.022	9.218	61.975	0.0400	1.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0	0
24629	904544	2020-06-07 04:28:11	8.134	7.912	-0.006	1.928	7.912	75.200	5.4975	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1

```
# Preprocessing the data
selected_features = ["TP2", "H1", "DV_pressure", "Reservoirs", "Oil_temperature", "Motor_current", "Oil_level", 'status']
X = data[selected_features]
y = data['status']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
class EnsembleModel():

  def __init__(self, clfs = [], *args, **kwargs):
    self._clfs = clfs

  def fit(self, x, y):
    for c in self._clfs:
      c.fit(x,y)

clf1 = NeuralNet()
clf2 = ExtremeLearningMachine(40)
clf3 = XGBoost()

ensembled = EnsembleModel(clfs = [clf1, clf2, clf3])
```

```
ensembled.fit(X_train, y_train)
```

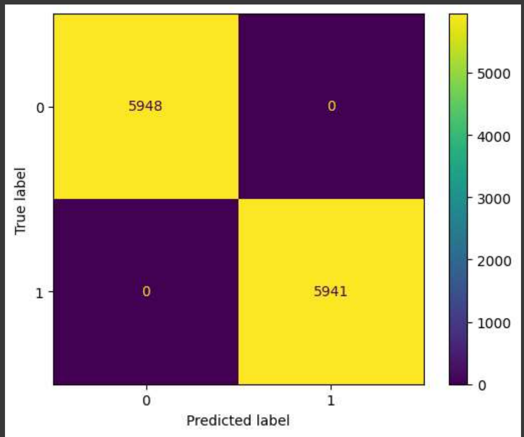
```
<ipython-input-139-2c0b25ae3745>:28: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
1479/1487 [============================>.] - ETA: 0s - loss: 0.0128 - accuracy: 0.9978WARNING:tensorflow:Early stopping conditioned on metric `val_accuracy` which is not available. Available metrics are: loss,accuracy
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_accuracy` which is not available. Available metrics are: loss,accuracy,lr
1487/1487 [==============================] - 8s 5ms/step - loss: 0.0128 - accuracy: 0.9978 - lr: 0.0010
<ipython-input-61-5b2890666025>:34: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = self._encoder.fit_transform(y[:, np.newaxis]).toarray()
```

```
y_pred = ensembled.predict(X_test)
```

```
372/372 [==============================] - 1s 2ms/step
```

```
y_pred = np.argmax(y_pred, axis = 1)
```

```
# Display the confusion matrix
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```




```
# Accuracy score
print(f"Accuracy of Ensemble method {accuracy_score(y_test, y_pred)}")

# Classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy of Ensemble method 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5948
           1       1.00      1.00      1.00      5941
```