In [ ]:

```
import pandas as pd
from joblib import load
from sklearn.metrics import f1_score
```

In [ ]:

```
def final_fun_1(X):
    """
    Function to make final predictions
    takes raw test data as input and prepocesses
    returns predicted class label
    """

    # loading the minimax scaler
    scaler = load('minimax_scaler.joblib')
    # loading the trained model
    model = load('random_forest.joblib')

    # final sensors
    final_sensors = ['sensor_00', 'sensor_04', 'sensor_06', 'sensor_07',
                 'sensor_08', 'sensor_09', 'sensor_10', 'sensor_11',
                 'sensor_12']

    data = {}

    for sensor in final_sensors:
        # filling missing values with -1
        X[sensor].fillna(-1, inplace=True)
        data[sensor] = X[sensor]

    # creating dataframe
    data_df = pd.DataFrame(data)

    # normalizing the data
    data_df = scaler.transform(data_df)

    # prediction
    y = model.predict(data_df)

    return y
```

In [ ]:

```
def final_fun_2(X, Y):
    """
    Function to make predictions
    takes raw test data as input and prepocesses
    returns predicted macro f1-score
    """

    # loading the minimax scaler
    scaler = load('minimax_scaler.joblib')
    # loading the trained model
    model = load('random_forest.joblib')

    # convert series to dataframe
    Y = Y.to_frame()
    # converting recovery state to broken state
    Y['machine_status'] = Y['machine_status'].map(lambda
                          label:  'BROKEN' if label != 'NORMAL' else 'NORMAL'

    # encoding machine status
    # 0: Normal state
    # 1: Broken state
    Y['label'] = Y['machine_status'].map(lambda label: 0
                                            if label == 'NORMAL' else 1)

    # final sensors
    final_sensors = ['sensor_00', 'sensor_04', 'sensor_06', 'sensor_07',
                 'sensor_08', 'sensor_09', 'sensor_10', 'sensor_11',
                 'sensor_12']

    data = {}

    for sensor in final_sensors:
        # filling missing values with -1
        X[sensor].fillna(-1, inplace=True)
        data[sensor] = X[sensor]

    labels = [None] * (X.shape[0])

    for i in range(0, X.shape[0]-10):
        labels[i] = Y['label'][i+10]

    data['label'] = labels

    # creating dataframe
    data_df = pd.DataFrame(data)

    # dropping last rows with null value
    data_df.drop(data_df.tail(10).index, inplace=True)

    # y data
    data_y = data_df['label']
    # x data
    data_x = data_df.drop(columns='label')


    # normalizing the data
    X_test = scaler.transform(data_x)
    # prediction
    y_pred = model.predict(X_test)

    y_true = data_y

    # macro f1 score
    f1_macro = f1_score(y_true, y_pred, average='macro')

    return f1_macro
```

## Testing final_fun_1:

In [ ]:

```
X = pd.read_csv("raw_X_test.csv", nrows=1)
X.head()
```

Out[ ]:

| | Unnamed: 0 | timestamp | sensor_00 | sensor_01 | sensor_02 | sensor_03 | sensor_04 | sensor_05 | se |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 131000 | 2018-06-30 23:20:00 | NaN | 36.501736 | 39.0625 | 35.763889 | 3.451967 | 99.999878 | |

1 rows × 55 columns

In [ ]:

```
print(final_fun_1(X))
```

```
[1.]
```

## Testing final_fun_2:

In [ ]:

```
X = pd.read_csv("raw_X_test.csv")
X.head()
```

Out[ ]:

| | Unnamed: 0 | timestamp | sensor_00 | sensor_01 | sensor_02 | sensor_03 | sensor_04 | sensor_05 | se |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 131000 | 2018-06-30 23:20:00 | NaN | 36.501736 | 39.0625 | 35.763889 | 3.451967 | 99.999878 | |
| **1** | 131001 | 2018-06-30 23:21:00 | NaN | 36.501740 | 39.0625 | 35.763889 | 3.336227 | 99.999878 | |
| **2** | 131002 | 2018-06-30 23:22:00 | NaN | 36.458330 | 39.0625 | 35.763889 | 3.336227 | 99.999878 | |
| **3** | 131003 | 2018-06-30 23:23:00 | NaN | 36.458332 | 39.0625 | 35.763889 | 3.104745 | 99.999878 | |
| **4** | 131004 | 2018-06-30 23:24:00 | NaN | 36.458330 | 39.0625 | 35.763890 | 2.798032 | 99.999878 | |

5 rows × 55 columns

In [ ]:

```
Y = X['machine_status']
Y.head()
```

Out[ ]:

```
0    RECOVERING
1    RECOVERING
2    RECOVERING
3    RECOVERING
4    RECOVERING
Name: machine_status, dtype: object
```

In [ ]:

```
print(final_fun_2(X, Y))
```

```
0.9963262396049439
```