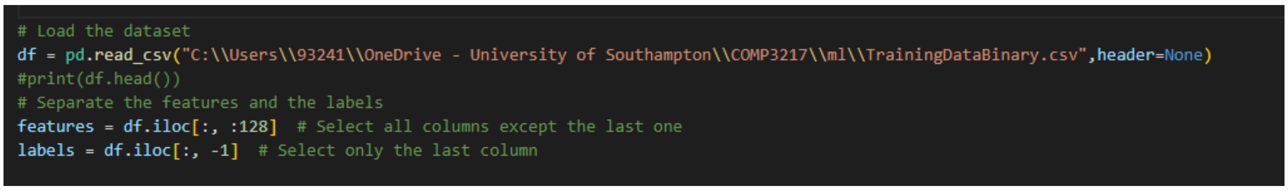
Task1

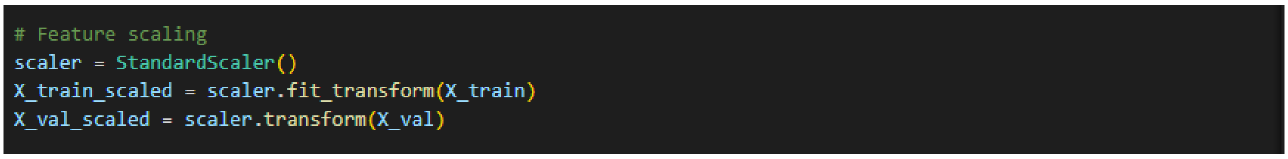
1. Data Preprocessing: Load the training data (TrainingDataBinary.csv) and testing data (TestingDataBinary.csv) in my project. Separate the features (128 numbers) and labels (0 or 1) in the training data.



```
# Load the dataset
df = pd.read_csv("C:\\Users\\93241\\OneDrive - University of Southampton\\COMP3217\\ml\\TrainingDataBinary.csv",header=None)
#print(df.head())
# Separate the features and the labels
features = df.iloc[:, :128]  # Select all columns except the last one
labels = df.iloc[:, -1]  # Select only the last column
```

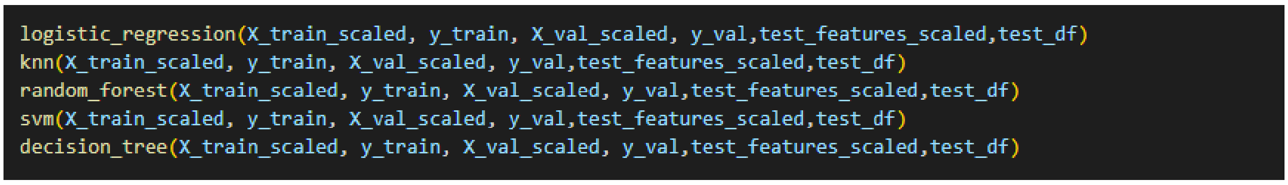
2. Divide 80% of the training data into a training set(X_train, X_val) and the rest into a validation set(y_train, y_val).

3. Feature scaling. Use StandardScaler() function in sklearn to standlize the data.



```
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

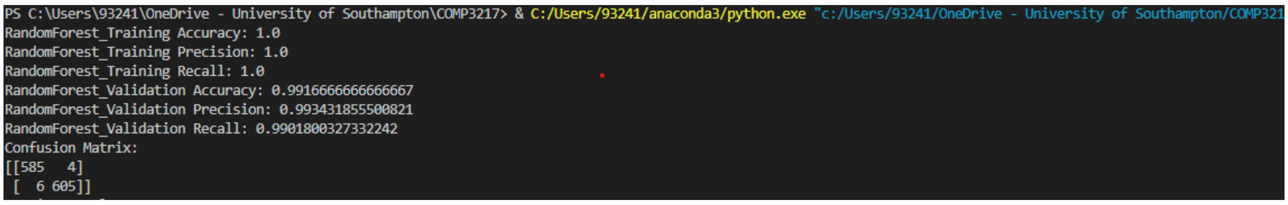
4. Model Selection: Choose an appropriate machine learning algorithm for binary classification. Some popular choices include Logistic Regression, Random Forest, Support Vector Machines, decision tree and Knn. Bring the data into the above models and try one by one.



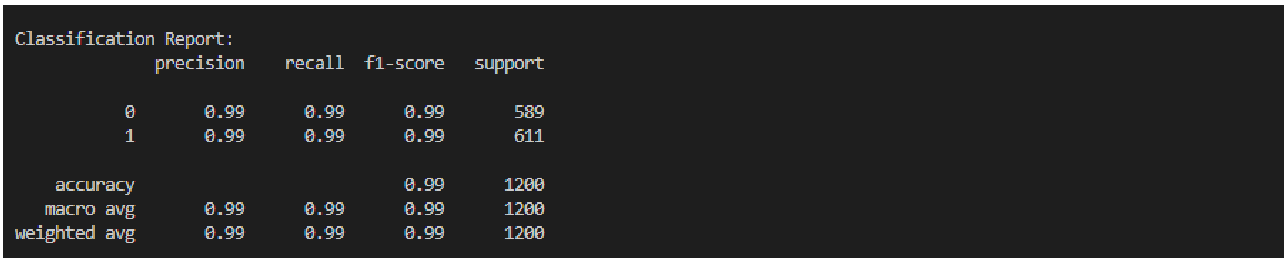
```
logistic_regression(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
knn(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
random_forest(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
svm(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
decision_tree(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
```

5. Model Training: Train the selected model using the labeled training data. Split the training data into training and validation subsets to evaluate the model's performance during training. Use appropriate evaluation metrics such as accuracy, precision, recall to assess the model's performance.

6. Model Evaluation: Evaluate the trained model on the validation set and analyze the training error and accuracy. Adjust the model hyperparameters, if necessary, to optimize its performance. Tried all the models, Random Forest gave the best result and Random Forest was chosen to be the final model.



```
PS C:\Users\93241\OneDrive - University of Southampton\COMP3217> & C:/Users/93241/anaconda3/python.exe "c:/Users/93241/OneDrive - University of Southampton/COMP321
RandomForest_Training Accuracy: 1.0
RandomForest_Training Precision: 1.0
RandomForest_Training Recall: 1.0
RandomForest_Validation Accuracy: 0.9916666666666667
RandomForest_Validation Precision: 0.993431855500821
RandomForest_Validation Recall: 0.9901800327332242
Confusion Matrix:
[[585   4]
 [  6 605]]
```



```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       589
           1       0.99      0.99      0.99       611

    accuracy                           0.99      1200
   macro avg       0.99      0.99      0.99      1200
weighted avg       0.99      0.99      0.99      1200
```

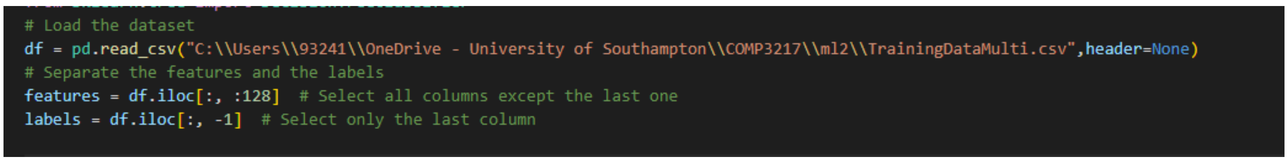
7. Prediction on Testing Data: Once the model is trained and tuned, use it to predict the labels for the testing data. Output the predicted labels in the same order as given.

```
 Testing Results:
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

8. Generate Testing Results File: Create a file (RandomForestTestingResultsBinary.csv) with the same format as the training data, including the computed labels for each system trace in the testing data.

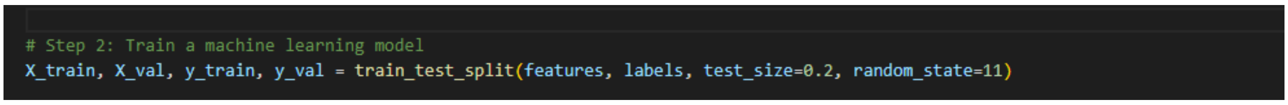
Task2(Most of the steps are consistent with Task1)

1. Data Preprocessing: Load the training data (TrainingDataMulti.csv) and testing data (TestingDataMulti.csv) in my project. Separate the features (128 numbers) and labels (0 1 2) in the training data.



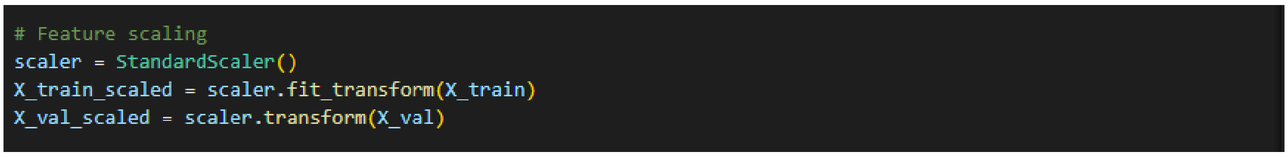
```python
# Load the dataset
df = pd.read_csv("C:\\Users\\93241\\OneDrive - University of Southampton\\COMP3217\\ml2\\TrainingDataMulti.csv",header=None)
# Separate the features and the labels
features = df.iloc[:, :128]  # Select all columns except the last one
labels = df.iloc[:, -1]  # Select only the last column
```

2. Divide 80% of the training data into a training set(X_train, X_val) and the rest into a validation set(y_train, y_val).



```python
# Step 2: Train a machine learning model
X_train, X_val, y_train, y_val = train_test_split(features, labels, test_size=0.2, random_state=11)
```

3. Feature scaling. Use StandardScaler() function in sklearn to standlize the data.



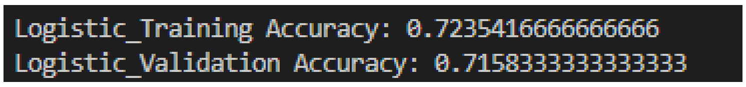
```python
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

4. Model Selection: Choose an appropriate machine learning algorithm for binary classification. Some popular choices include Logistic Regression, Random Forest, Support Vector Machines, decision tree and Knn. Bring the data into the above models and try one by one.

```
logistic_regression(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
knn(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
random_forest(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
svm(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
decision_tree(X_train_scaled, y_train, X_val_scaled, y_val,test_features_scaled,test_df)
```

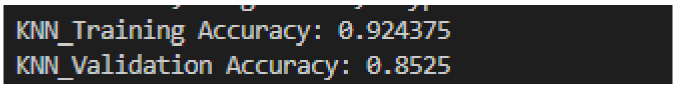
5. Model Training: Train the selected model using the labeled training data. Split the training data into training and validation subsets to evaluate the model's performance during training. Use appropriate evaluation metrics such as accuracy, precision, recall to assess the model's performance.

   logistic_regression:

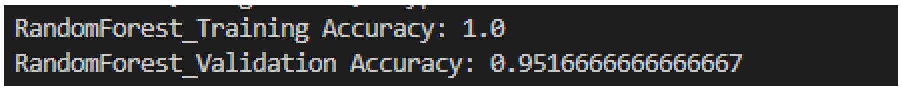
   ```
   Logistic_Training Accuracy: 0.7235416666666666
   Logistic_Validation Accuracy: 0.7158333333333333
   ```

   knn:

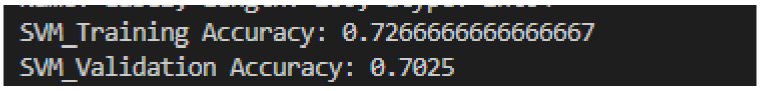
   ```
   KNN_Training Accuracy: 0.924375
   KNN_Validation Accuracy: 0.8525
   ```

   random forest:

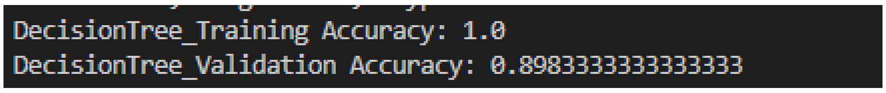
   ```
   RandomForest_Training Accuracy: 1.0
   RandomForest_Validation Accuracy: 0.9516666666666667
   ```

   svm:

   ```
   SVM_Training Accuracy: 0.7266666666666667
   SVM_Validation Accuracy: 0.7025
   ```

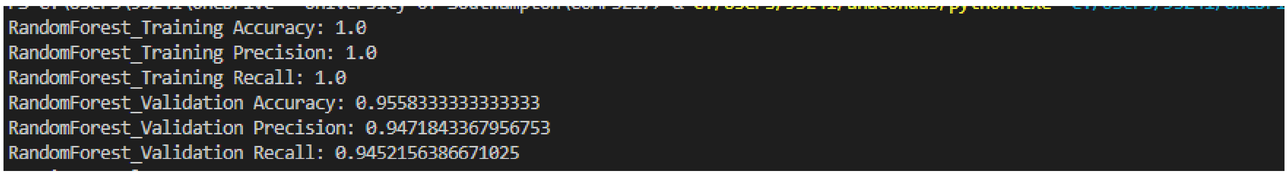
   decision tree:

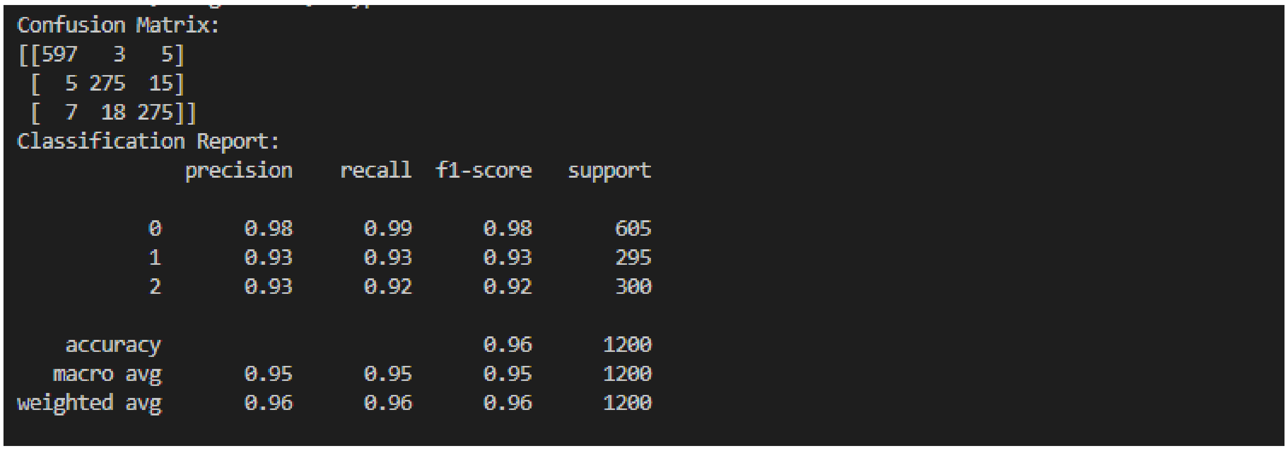
   ```
   DecisionTree_Training Accuracy: 1.0
   DecisionTree_Validation Accuracy: 0.8983333333333333
   ```

   To sum up, we choose random forest as the final model.

6. Model Evaluation: Evaluate the trained model on the validation set and analyze the training error and accuracy. Adjust the model hyperparameters, if necessary, to optimize its performance. Tried all the models, Random Forest gave the best result and Random Forest was chosen to be the final model.

   ```
   RandomForest_Training Accuracy: 1.0
   RandomForest_Training Precision: 1.0
   RandomForest_Training Recall: 1.0
   RandomForest_Validation Accuracy: 0.9558333333333333
   RandomForest_Validation Precision: 0.9471843367956753
   RandomForest_Validation Recall: 0.9452156386671025
   ```



```
Confusion Matrix:
[[597   3   5]
 [  5 275  15]
 [  7  18 275]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       605
           1       0.93      0.93      0.93       295
           2       0.93      0.92      0.92       300

    accuracy                           0.96      1200
   macro avg       0.95      0.95      0.95      1200
weighted avg       0.96      0.96      0.96      1200
```

7. Prediction on Testing Data: Once the model is trained and tuned, use it to predict the labels for the testing data. Output the predicted labels in the same order as given.

```
 Testing Results:
 2  2  2  2  2  2  1  1  2  2  2  1  1  1  1  1  2  1  1  1  1  2  2  2  2
2  2  2  2  2  0  2  0  1  1  1  1  1  2  1  1  1  1  2  2  2  2  2  2  1  2
2  2  1  2  2  2  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
```

8. Generate Testing Results File: Create a file (RandomForestTestingResultsMulti.csv) with the same format as the training data, including the computed labels for each system trace in the testing data.