APPENDIX A: PYTHON CODING FOR TOP PERFORMANCE MODELS

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import csv

cd C:/Users/Azrul/Desktop/KIG4002 FYP/data/disregard earthenware

# importing dataset
dataset = pd.read_csv("data to feed.csv")
dataset.name = 'dataset'
dataset.head()

# separating features and result vectors
y = dataset["Type"]
X = dataset.drop(['Type'], axis = 1)

# splitting the dataset into the training set and test set

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.40, random_state = 0)

# K Nearest Neighbors (KNN)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
k_range = range(1,26)
scores = {}
scores_list = []
for k in k_range:
    knn_model = KNeighborsClassifier(n_neighbors = k)
    knn_model.fit(X_train, y_train)
    y_pred = knn_model.predict(X_test)
    scores[k] = metrics.accuracy_score(y_test, y_pred)
    scores_list.append(metrics.accuracy_score(y_test, y_pred))
scores

# Finding the best k-value
# plotting the relationship between K and testing accuracy

plt.plot(k_range, scores_list)
plt.xlabel('value of K for KNN')
plt.ylabel('Testing Accuracy')

# Choose 8 as the value of K for KNN
knn_model = KNeighborsClassifier(n_neighbors = 8)
```

```
knn_model.fit(X_train, y_train)

# KNN prediction
knn_pred = knn_model.predict(X_test)
knn_pred

# KNN Model Evaluation
from sklearn.metrics import confusion_matrix

cm_knn = confusion_matrix(y_test, knn_pred)
cm1_knn = metrics.accuracy_score(y_test, knn_pred)

print('COnfusion Matrix:\n', cm_knn)
print('Accuracy:', "% 0.4f" % cm1_knn)

acc_knn = knn_model.score(X_test, y_test)*100
print('K Nearest Neighbor Accuracy', round(acc_knn, 2), '%')

print(metrics.classification_report(y_test, knn_pred, digits = 2))

tp_knn = confusion_matrix(y_test, knn_pred)[1,1]
fp_knn = confusion_matrix(y_test, knn_pred)[0,1]
tn_knn = confusion_matrix(y_test, knn_pred)[0,0]
fn_knn = confusion_matrix(y_test, knn_pred)[1,0]
accuracy_knn = cm1_knn

test_input = pd.read_csv("data to test.csv")
test_input.name = 'test_input'
test_input.head()

knn_model.predict(test_input)

# Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
NB_model = GaussianNB()
NB_model = GaussianNB().fit(X_train, y_train)

#NB prediction
NB_pred = NB_model.predict(X_test)
NB_pred

# Model Evaluation
from sklearn.metrics import confusion_matrix

cm_NB = confusion_matrix(y_test, NB_pred)
cm1_NB = metrics.accuracy_score(y_test, NB_pred)

print('Confusion Matrix:\n', cm_NB)
print('Accuracy:', "% 0.4f" % cm1_NB)
```

```
acc_NB = NB_model.score(X_test, y_test)*100
print('Naive Bayes Accuracy', round(acc_NB, 2), '%')

print(metrics.classification_report(y_test, NB_pred, digits = 2))

tp_NB = confusion_matrix(y_test, NB_pred)[1,1]
fp_NB = confusion_matrix(y_test, NB_pred)[0,1]
tn_NB = confusion_matrix(y_test, NB_pred)[0,0]
fn_NB = confusion_matrix(y_test, NB_pred)[1,0]
accuracy_NB = cm1_NB

NB_model.predict(test_input)

# Support Vector Machine (SVM)
from sklearn.svm import SVC
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
SVM_model = SVC(kernel = 'rbf', probability = True)
SVM_model = SVM_model.fit(X_train, y_train)

# SVM prediction
SVM_pred = SVM_model.predict(X_test)
SVM_pred

# Model evaluation (SVM)
from sklearn.metrics import confusion_matrix

cm_SVM = confusion_matrix(y_test, SVM_pred)
cm1_SVM = metrics.accuracy_score(y_test, SVM_pred)

print('Confusion Matrix:\n', cm_SVM)
print('Accuracy:', "% 0.4f" % cm1_SVM)

acc_SVM = SVM_model.score(X_test, y_test)*100
print('Support Vector Machine Accuracy', round(acc_SVM, 2), '%')

tp_SVM = confusion_matrix(y_test, SVM_pred)[1,1]
fp_SVM = confusion_matrix(y_test, SVM_pred)[0,1]
tn_SVM = confusion_matrix(y_test, SVM_pred)[0,0]
fn_SVM = confusion_matrix(y_test, SVM_pred)[1,0]
accuracy_SVM = cm1_SVM

SVM_model.predict(test_input)

print(metrics.classification_report(y_test, SVM_pred, digits = 2))

# Decision Tree
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn import metrics
DT_model = DecisionTreeClassifier()
DT_model = DT_model.fit(X_train, y_train)

# DT prediction
DT_pred = DT_model.predict(X_test)
DT_pred

# Model evaluation (DT)
from sklearn.metrics import confusion_matrix

cm_DT = confusion_matrix(y_test, DT_pred)
cm1_DT = metrics.accuracy_score(y_test, DT_pred)

print('Confusion Matrix:\n', cm_DT)
print('Accuracy:', "%0.4f" % cm1_DT)

acc_DT = DT_model.score(X_test, y_test)*100
print('Decision Tree Accuracy', round(acc_DT, 2), '%')

print(metrics.classification_report(y_test, DT_pred, digits = 2))

tp_DT = confusion_matrix(y_test, DT_pred)[1,1]
fp_DT = confusion_matrix(y_test, DT_pred)[0,1]
tn_DT = confusion_matrix(y_test, DT_pred)[0,0]
fn_DT = confusion_matrix(y_test, DT_pred)[1,0]
accuracy_DT = cm1_DT

DT_model.predict(test_input)

# Logistic Regression (LR)
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
LR_model = LogisticRegression()
LR_model = LR_model.fit(X_train, y_train)

# LR prediction
LR_pred = LR_model.predict(X_test)
LR_pred

from sklearn.metrics import confusion_matrix

cm_LR = confusion_matrix(y_test, LR_pred)
cm1_LR = metrics.accuracy_score(y_test, LR_pred)

print('Confusion Marix:\n', cm_LR)
print('Logistic Regression Accuracy', "% 0.4f" % cm1_LR)
```

```
print(metrics.classification_report(y_test, LR_pred, digits = 2))

tp_LR = confusion_matrix(y_test, LR_pred)[1,1]
fp_LR = confusion_matrix(y_test, LR_pred)[0,1]
tn_LR = confusion_matrix(y_test, LR_pred)[0,0]
fn_LR = confusion_matrix(y_test, LR_pred)[1,0]
accuracy_LR = cm1_LR

LR_model.predict(test_input)

# random forest (RF)
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
RF_model = RandomForestClassifier()
RF_model = RF_model.fit(X_train, y_train)

# RF prediction
RF_pred = RF_model.predict(X_test)
RF_pred

# Model evaluation (RF)
from sklearn.metrics import confusion_matrix

cm_RF = confusion_matrix(y_test, RF_pred)
cm1_RF = metrics.accuracy_score(y_test, RF_pred)

print('Confusion Matrix:\n', cm_RF)
print('Accuracy:' "%0.4f" % cm1_RF)

acc_RF = RF_model.score(X_test, y_test)*100
print('Random Forest Accuracy', round(acc_RF, 2), '%')

print(metrics.classification_report(y_test, RF_pred, digits = 2))

tp_RF = confusion_matrix(y_test, RF_pred)[1,1]
fp_RF = confusion_matrix(y_test, RF_pred)[0,1]
tn_RF = confusion_matrix(y_test, RF_pred)[0,0]
fn_RF = confusion_matrix(y_test, RF_pred)[1,0]
accuracy_RF = cm1_RF

RF_model.predict(test_input)

#Measuring the error
models = [('KNN', tp_knn, fp_knn, tn_knn, fn_knn, accuracy_knn), ('NB',
tp_NB, fp_NB, tn_NB, fn_NB, accuracy_NB), ('SVM', tp_SVM, fp_SVM,
tn_SVM, fn_SVM, accuracy_SVM), ('DT', tp_DT, fp_DT, tn_DT, fn_DT,
```

```
accuracy_DT), ('LR', tp_LR, fp_LR, tn_LR, fn_LR, accuracy_LR), ('RF',
tp_RF, fp_RF, tn_RF, fn_RF, accuracy_RF)]

predict = pd.DataFrame(data=models, columns=['Model', 'True Postive',
'False Positive', 'True Negative', 'False Negative', 'Accuracy'])

recall_knn = tp_knn/(tp_knn+fn_knn)
recall_NB = tp_NB/(tp_NB+fn_NB)
recall_SVM = tp_SVM/(tp_SVM+fn_SVM)
recall_DT = tp_DT/(tp_DT+fn_DT)
recall_LR = tp_LR/(tp_LR+fn_LR)
recall_RF = tp_RF/(tp_RF+fn_RF)

precision_knn = tp_knn/(tp_knn+fp_knn)
precision_NB = tp_NB/(tp_NB+fp_NB)
precision_SVM = tp_SVM/(tp_SVM+fp_SVM)
precision_DT = tp_DT/(tp_DT+fp_DT)
precision_LR = tp_LR/(tp_LR+fp_LR)
precision_RF = tp_RF/(tp_RF+fp_RF)

f1score_knn = precision_knn * recall_knn / (precision_knn + recall_knn)
f1score_NB = precision_NB * recall_NB / (precision_NB + recall_NB)
f1score_SVM = precision_SVM * recall_SVM / (precision_SVM + recall_SVM)
f1score_DT = precision_DT * recall_DT / (precision_DT + recall_DT)
f1score_LR = precision_LR * recall_LR / (precision_LR + recall_LR)
f1score_RF = precision_RF * recall_RF / (precision_RF + recall_RF)

recall_knn = tp_knn/(tp_knn+fn_knn)
precision_knn = tp_knn/(tp_knn+fp_knn)
f1score_knn = precision_knn * recall_knn / (precision_knn + recall_knn)
accuracy_knn = (tp_knn+tn_knn)/(tp_knn+tn_knn+fp_knn+fn_knn)

#Measuring the error
benchmarks = [('KNN', recall_knn, precision_knn, f1score_knn,
accuracy_knn), ('NB', recall_NB, precision_NB, f1score_NB,
accuracy_NB), ('SVM', recall_SVM, precision_SVM, f1score_SVM,
accuracy_SVM), ('DT', recall_DT, precision_DT, f1score_DT,
accuracy_DT), ('LR', recall_LR, precision_LR, f1score_LR, accuracy_LR),
('RF', recall_RF, precision_RF, f1score_RF, accuracy_RF)]

predictb = pd.DataFrame(data=benchmarks, columns=['Model', 'Recall',
'Precision', 'F1 Score', 'Accuracy'])

#getting AUCROC curve
from sklearn.metrics import (precision_recall_curve, auc, roc_curve,
recall_score, f1_score, average_precision_score,
precision_recall_fscore_support )
from sklearn.preprocessing import LabelEncoder

# create label encoder
```

```
le = LabelEncoder()

# fit and transform y_true
y_test = le.fit_transform(y_test)

#KNN
y_pred_prob_knn = knn_model.predict_proba(X_test)[:,1]
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_prob_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)
precision_knn, recall_knn, th_knn = precision_recall_curve(y_test,
y_pred_prob_knn)
pr_auc_knn = auc(recall_knn, precision_knn)

#NB
y_pred_prob_NB = NB_model.predict_proba(X_test)[:,1]
fpr_NB, tpr_NB, thresholds_NB = roc_curve(y_test, y_pred_prob_NB)
roc_auc_NB = auc(fpr_NB, tpr_NB)
precision_NB, recall_NB, th_NB = precision_recall_curve(y_test,
y_pred_prob_NB)
pr_auc_NB = auc(recall_NB, precision_NB)

#SVM
y_pred_prob_SVM = SVM_model.predict_proba(X_test)[:,1]
fpr_SVM, tpr_SVM, thresholds_SVM = roc_curve(y_test, y_pred_prob_SVM)
roc_auc_SVM = auc(fpr_SVM, tpr_SVM)
precision_SVM, recall_SVM, th_SVM = precision_recall_curve(y_test,
y_pred_prob_SVM)
pr_auc_SVM = auc(recall_SVM, precision_SVM)

#DT
y_pred_prob_DT = DT_model.predict_proba(X_test)[:,1]
fpr_DT, tpr_DT, thresholds_DT = roc_curve(y_test, y_pred_prob_DT)
roc_auc_DT = auc(fpr_DT, tpr_DT)
precision_DT, recall_DT, th_DT = precision_recall_curve(y_test,
y_pred_prob_DT)
pr_auc_DT = auc(recall_DT, precision_DT)

#LR
y_pred_prob_LR = LR_model.predict_proba(X_test)[:,1]
fpr_LR, tpr_LR, thresholds_LR = roc_curve(y_test, y_pred_prob_LR)
roc_auc_LR = auc(fpr_LR, tpr_LR)
precision_LR, recall_LR, th_LR = precision_recall_curve(y_test,
y_pred_prob_LR)
pr_auc_LR = auc(recall_LR, precision_LR)

#RF
y_pred_prob_RF = NB_model.predict_proba(X_test)[:,1]
fpr_RF, tpr_RF, thresholds_RF = roc_curve(y_test, y_pred_prob_RF)
roc_auc_RF = auc(fpr_RF, tpr_RF)
```

```
precision_RF, recall_RF, th_RF = precision_recall_curve(y_test,
y_pred_prob_RF)
pr_auc_RF = auc(recall_RF, precision_RF)

#PLot ROC curve
plt.figure(figsize=(10,8))
plt.plot([0,1],[0,1], 'k--')
plt.plot(fpr_knn, tpr_knn, label='KNN (area = %0.3f)' % roc_auc_knn)
plt.plot(fpr_NB, tpr_NB, label='NB (area = %0.3f)' % roc_auc_NB)
plt.plot(fpr_SVM, tpr_SVM, label='SVM (area = %0.3f)' % roc_auc_SVM)
plt.plot(fpr_DT, tpr_DT, label='DT (area = %0.3f)' % roc_auc_DT)
plt.plot(fpr_LR, tpr_LR, label='LR (area = %0.3f)' % roc_auc_LR)
plt.plot(fpr_RF, tpr_RF, label='RF (area = %0.3f)' % roc_auc_RF)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves")
plt.legend(loc='best')
plt.show()
```

APPENDIX B: PYTHON CODING FOR ENSEMBLE LEARNING

```
cd C:/Users/Azrul/Desktop/KIG4002 FYP/data/disregard earthenware

# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import csv
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix

# importing dataset
dataset = pd.read_csv("data to feed.csv")
dataset.name = 'dataset'
dataset.head()

# separating features and result vectors
y = dataset["Type"]
X = dataset.drop(['Type'], axis = 1)

# splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.40, random_state = 0)

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score


# Define the parameter grid to search over
param_grid = {'n_estimators': [10, 20, 50],
              'max_depth': [3, 4, 5],
              'min_samples_split': [5, 10, 20]}

# Create a Random Forest classifier
rf_model = RandomForestClassifier(random_state=0)

# Perform a grid search over the parameter grid
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
cv=5)
grid_search.fit(X_train, y_train)
```

```
# Retrieve the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)

# Train the Random Forest model with the best hyperparameters
rf_model = RandomForestClassifier(random_state=0, **best_params)
rf_model.fit(X_train, y_train)

# Evaluate the performance of the model on the test set
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Naive Bayes
NB_model = GaussianNB()
NB_model = NB_model.fit(X_train, y_train)

# NB prediction
NB_pred = NB_model.predict(X_test)

# Logistic Regression (LR)
LR_model = LogisticRegression()
LR_model = LR_model.fit(X_train, y_train)

# LR prediction
LR_pred = LR_model.predict(X_test)

# random forest (RF)
RF_model = RandomForestClassifier(n_estimators=50, max_depth=3,
min_samples_split=20)
RF_model = RF_model.fit(X_train, y_train)

# RF prediction
RF_pred = RF_model.predict(X_test)

# create the voting classifier
ensemble_model = VotingClassifier(estimators=[('rf', RF_model), ('nb',
NB_model), ('lr', LR_model)], voting='hard')

# fit the model on the training set
ensemble_model.fit(X_train, y_train)

# predict using the ensemble model
ensemble_pred = ensemble_model.predict(X_test)

# Model evaluation (ensemble)
cm_ensemble = confusion_matrix(y_test, ensemble_pred)
cm1_ensemble = metrics.accuracy_score(y_test, ensemble_pred)

print('Confusion Matrix:\n', cm_ensemble)
```

```
print('Accuracy:', "% 0.4f" % cm1_ensemble)

print(metrics.classification_report(y_test, ensemble_pred, digits = 2))

tp_ensemble = confusion_matrix(y_test, ensemble_pred)[1,1]
fp_ensemble = confusion_matrix(y_test, ensemble_pred)[0,1]
tn_ensemble = confusion_matrix(y_test, ensemble_pred)[0,0]
fn_ensemble = confusion_matrix(y_test, ensemble_pred)[1,0]
accuracy_ensemble = cm1_ensemble

test_input = pd.read_csv("data to test.csv")
ensemble_model.predict(test_input)

#Measuring the error
models = [ ('Ensemble Learning', tp_ensemble, fp_ensemble, tn_ensemble,
fn_ensemble, accuracy_ensemble)]

predict = pd.DataFrame(data=models, columns=['Model', 'True Postive',
'False Positive', 'True Negative', 'False Negative', 'Accuracy'])

recall_ensemble = tp_ensemble/(tp_ensemble+fn_ensemble)

precision_ensemble = tp_ensemble/(tp_ensemble+fp_ensemble)

f1score_ensemble = precision_ensemble * recall_ensemble / (preci-
sion_ensemble + recall_ensemble)


#Measuring the error
benchmarks = [('Ensemble Learning', recall_ensemble, precision_ensem-
ble, f1score_ensemble, accuracy_ensemble)]

predictb = pd.DataFrame(data=benchmarks, columns=['Model', 'Recall',
'Precision', 'F1 Score', 'Accuracy'])
print(predictb)

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Predict class probabilities using the individual models
y_pred_prob_nb = NB_model.predict_proba(X_test)[:, 1]
y_pred_prob_lr = LR_model.predict_proba(X_test)[:, 1]
y_pred_prob_rf = RF_model.predict_proba(X_test)[:, 1]

# Calculate ensemble class probabilities using hard voting
y_pred_prob_ensemble = (y_pred_prob_nb + y_pred_prob_lr +
y_pred_prob_rf) / 3.0

# Calculate false positive rate (FPR), true positive rate (TPR), and
thresholds for ensemble
```

```
fpr_ensemble, tpr_ensemble, thresholds_ensemble = roc_curve(y_test,
y_pred_prob_ensemble)

# Calculate AUCROC for ensemble
roc_auc_ensemble = auc(fpr_ensemble, tpr_ensemble)

# Plot the ROC curve for ensemble
plt.figure(figsize=(8, 6))
plt.plot(fpr_ensemble, tpr_ensemble, label='Ensemble (AUCROC
= %0.4f)' % roc_auc_ensemble)
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```