In [1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [2]:
```
df_0 = pd.read_csv('Training_Data.csv')
```

In [3]:
```
X_train=df_0[['area', 'major_axis_length', 'perimeter','equiv_diameter']]
y_train=df_0[['category']]
```

In [4]:
```
import sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [5]:
```
model=RandomForestClassifier()
```

In [6]:
```
model.fit(X_train, y_train)
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_9872\180087699.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model.fit(X_train, y_train)
```

Out[6]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [7]:
```
model.score(X_train, y_train) #R-squared
```

Out[7]:
```
0.983
```

In [ ]:

Sample1

In [113…
```
df_1 = pd.read_csv('Sample1_Data.csv')
```

In [114…
```
X_test1=df_1[['area', 'major_axis_length',  'perimeter','equiv_diameter']]
y_test1=df_1[['category']]
```

In [115…
```
import time

start_time = time.time()

y_pred1=model.predict(X_test1)

end_time = time.time()

Computation_time = end_time - start_time
```

In [116…
```
test1=pd.concat([X_test1, y_test1], axis='columns')
```

In [117…
```
dc1=pd.concat([test1.reset_index(), pd.Series(y_pred1, name='predicted')], axis='columns')
```

In [118…
```
dc1
```

Out[118]:

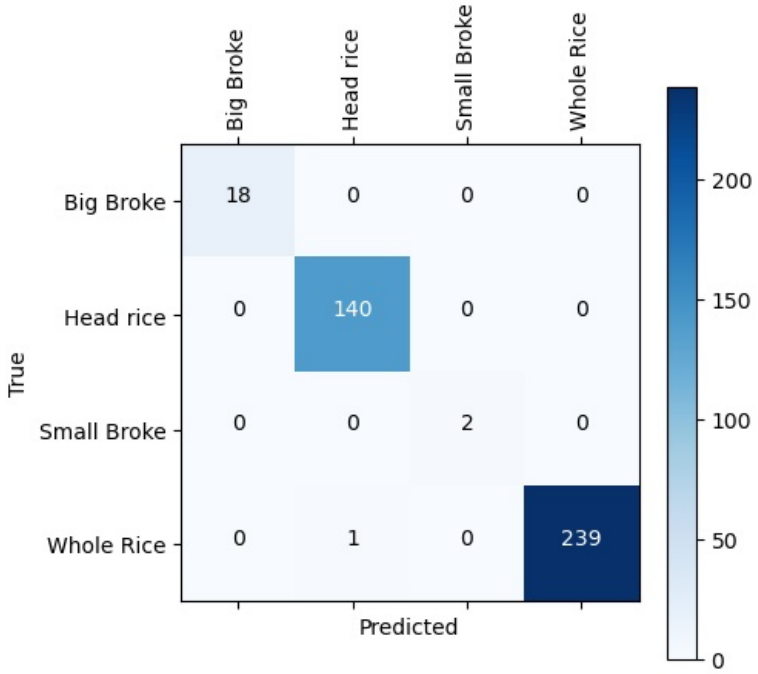
	index	area	major_axis_length	perimeter	equiv_diameter	category	predicted
0	0	1096	73.743430	153.470	37.355997	Whole Rice	Whole Rice
1	1	1138	71.509387	151.187	38.065031	Whole Rice	Whole Rice
2	2	1183	73.072335	152.104	38.810339	Whole Rice	Whole Rice
3	3	1129	67.134645	146.268	37.914212	Whole Rice	Whole Rice
4	4	1132	68.838994	146.680	37.964551	Whole Rice	Whole Rice
...	...	...	...	...	...	...	...
395	395	715	43.090911	102.285	30.172277	Big Broke	Big Broke
396	396	632	43.107959	98.665	28.367012	Big Broke	Big Broke
397	397	636	42.307958	99.102	28.456640	Big Broke	Big Broke
398	398	544	33.561338	84.977	26.318099	Small Broke	Small Broke
399	399	554	33.481004	86.772	26.558891	Small Broke	Small Broke

400 rows × 7 columns

In [119…
```
Testing_accuracy = accuracy_score(y_test1, y_pred1)
print('Accuracy on testing data: {:.2f}%'.format(Testing_accuracy * 100))
```

```
Accuracy on testing data: 99.75%
```

In [120…
```
category_labels = sorted(y_test1['category'].unique())

cm = confusion_matrix(y_test1, y_pred1)
plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar()

plt.xticks(np.arange(len(category_labels)), category_labels, rotation='vertical')
plt.yticks(np.arange(len(category_labels)), category_labels)

plt.xlabel('Predicted')
plt.ylabel('True')

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), horizontalalignment="center", color="white" if cm[i, j] > cm.max() / 2. e

plt.show()
```



In [121…
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(f"Total Computation Time: {Computation_time:.4f} seconds")
print("Accuracy Score:", accuracy_score(y_test1, y_pred1) * 100)
print("Testing Report:\n", classification_report(y_test1, y_pred1, digits=4))
```

```
Total Computation Time: 0.0209 seconds
Accuracy Score: 99.75
Testing Report:
               precision    recall  f1-score   support

   Big Broke     1.0000    1.0000    1.0000        18
   Head rice     0.9929    1.0000    0.9964       140
 Small Broke     1.0000    1.0000    1.0000         2
  Whole Rice     1.0000    0.9958    0.9979       240

    accuracy                         0.9975       400
   macro avg     0.9982    0.9990    0.9986       400
weighted avg     0.9975    0.9975    0.9975       400
```

In [122…
```
# Calculate metrics
accuracy = accuracy_score(y_test1, y_pred1)
classification_rep = classification_report(y_test1, y_pred1, digits=4, output_dict=True)
macro_avg_precision = classification_rep['macro avg']['precision']
macro_avg_recall = classification_rep['macro avg']['recall']
macro_avg_f1_score = classification_rep['macro avg']['f1-score']

# Create the result1 dictionary with the calculated values
result1 = {
    'Total Computation Time': Computation_time,
    'Accuracy': accuracy * 100,
    'Macro Avg Precision': macro_avg_precision * 100,
    'Macro Avg Recall': macro_avg_recall * 100,
    'Macro Avg F1-Score': macro_avg_f1_score * 100
}
```

```
# Print the formatted values
for key, value in result1.items():
    if isinstance(value, float):
        # Format floats to display exactly four decimal places
        formatted_value = '{:.4f}'.format(value)
    else:
        formatted_value = value  # Leave non-float values as they are

    print(f'{key}: {formatted_value}')
```

```
Total Computation Time: 0.0209
Accuracy: 99.7500
Macro Avg Precision: 99.8227
Macro Avg Recall: 99.8958
Macro Avg F1-Score: 99.8588
```

In [ ]:

Sample2

In [123…
```
df_2 = pd.read_csv('Sample2_Data.csv')
```

In [124…
```
X_test2=df_2[['area', 'major_axis_length',  'perimeter','equiv_diameter']]
y_test2=df_2[['category']]
```

In [125…
```
start_time = time.time()

y_pred2=model.predict(X_test2)

end_time = time.time()

Computation_time = end_time - start_time
```

In [126…
```
test2=pd.concat([X_test2, y_test2], axis='columns')
```

In [127…
```
dc2=pd.concat([test2.reset_index(), pd.Series(y_pred2, name='predicted')], axis='columns')
```

In [128…
```
dc2
```

Out[128]:

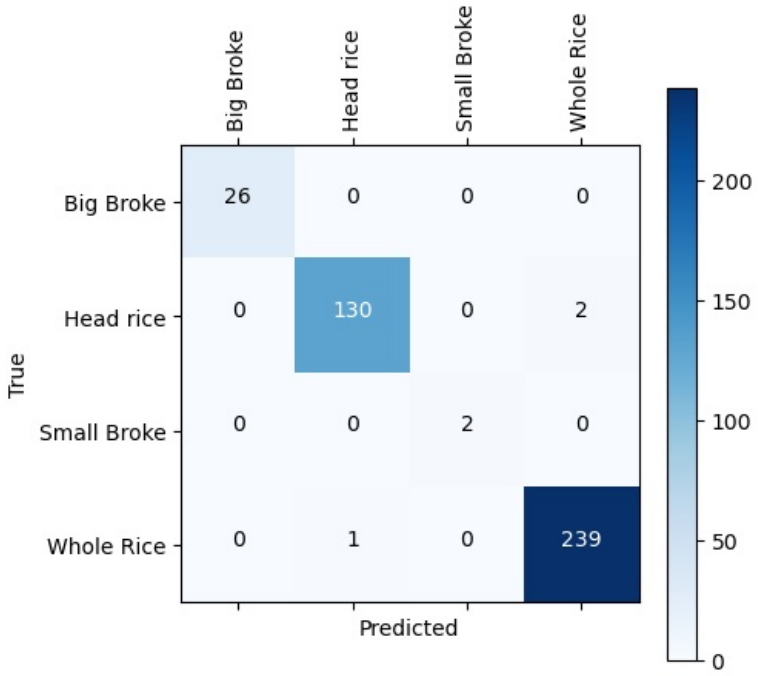
	index	area	major_axis_length	perimeter	equiv_diameter	category	predicted
0	0	1159	66.744855	144.132	38.414641	Whole Rice	Whole Rice
1	1	1087	65.328214	140.643	37.202303	Whole Rice	Whole Rice
2	2	1085	62.647573	137.119	37.168063	Whole Rice	Whole Rice
3	3	1154	64.571988	141.660	38.331690	Whole Rice	Whole Rice
4	4	1098	64.987673	142.644	37.390066	Whole Rice	Whole Rice
...	...	...	...	...	...	...	...
395	395	901	61.521619	131.561	33.870176	Big Broke	Big Broke
396	396	827	61.572143	130.254	32.449485	Big Broke	Big Broke
397	397	869	64.193722	134.498	33.263270	Big Broke	Big Broke
398	398	551	33.988487	84.274	26.486883	Small Broke	Small Broke
399	399	497	32.980543	82.083	25.155517	Small Broke	Small Broke

400 rows × 7 columns

In [129…
```
Testing_accuracy = accuracy_score(y_test2, y_pred2)
print('Accuracy on testing data: {:.2f}%'.format(Testing_accuracy * 100))
```

```
Accuracy on testing data: 99.25%
```

In [130…
```
category_labels = sorted(y_test2['category'].unique())

cm = confusion_matrix(y_test2, y_pred2)
plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar()

plt.xticks(np.arange(len(category_labels)), category_labels, rotation='vertical')
plt.yticks(np.arange(len(category_labels)), category_labels)

plt.xlabel('Predicted')
plt.ylabel('True')

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), horizontalalignment="center", color="white" if cm[i, j] > cm.max() / 2. e

plt.show()
```



In [131…
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(f"Total Computation Time: {Computation_time:.4f} seconds")
print("Accuracy Score:", accuracy_score(y_test2, y_pred2) * 100)
print("Testing Report:\n", classification_report(y_test2, y_pred2, digits=4))
```

```
Total Computation Time: 0.0160 seconds
Accuracy Score: 99.25
Testing Report:
               precision    recall  f1-score   support

   Big Broke     1.0000    1.0000    1.0000        26
   Head rice     0.9924    0.9848    0.9886       132
 Small Broke     1.0000    1.0000    1.0000         2
  Whole Rice     0.9917    0.9958    0.9938       240

    accuracy                         0.9925       400
   macro avg     0.9960    0.9952    0.9956       400
weighted avg     0.9925    0.9925    0.9925       400
```

In [132…
```
# Calculate metrics
accuracy = accuracy_score(y_test2, y_pred2)
classification_rep = classification_report(y_test2, y_pred2, digits=4, output_dict=True)
macro_avg_precision = classification_rep['macro avg']['precision']
macro_avg_recall = classification_rep['macro avg']['recall']
macro_avg_f1_score = classification_rep['macro avg']['f1-score']

# Create the result1 dictionary with the calculated values
result2 = {
    'Total Computation Time': Computation_time,
    'Accuracy': accuracy * 100,
    'Macro Avg Precision': macro_avg_precision * 100,
    'Macro Avg Recall': macro_avg_recall * 100,
    'Macro Avg F1-Score': macro_avg_f1_score * 100
}

# Print the formatted values
for key, value in result2.items():
    if isinstance(value, float):
        # Format floats to display exactly four decimal places
        formatted_value = '{:.4f}'.format(value)
    else:
        formatted_value = value  # Leave non-float values as they are

    print(f'{key}: {formatted_value}')
```

```
Total Computation Time: 0.0160
Accuracy: 99.2500
Macro Avg Precision: 99.6017
Macro Avg Recall: 99.5170
Macro Avg F1-Score: 99.5589
```

In [ ]:

Sample3

```
df_2 = pd.read_csv('Sample2_Data.csv')
```

In [133…
```
df_3 = pd.read_csv('Sample3_Data.csv')
```

In [134…
```
X_test3=df_3[['area', 'major_axis_length',  'perimeter','equiv_diameter']]
y_test3=df_3[['category']]
```

In [135…
```
start_time = time.time()

y_pred3=model.predict(X_test3)

end_time = time.time()

Computation_time = end_time - start_time
```

In [136…
```
test3=pd.concat([X_test3, y_test3], axis='columns')
```

In [137…
```
dc3=pd.concat([test3.reset_index(), pd.Series(y_pred3, name='predicted')], axis='columns')
```

In [138…
```
dc3
```

Out[138]:

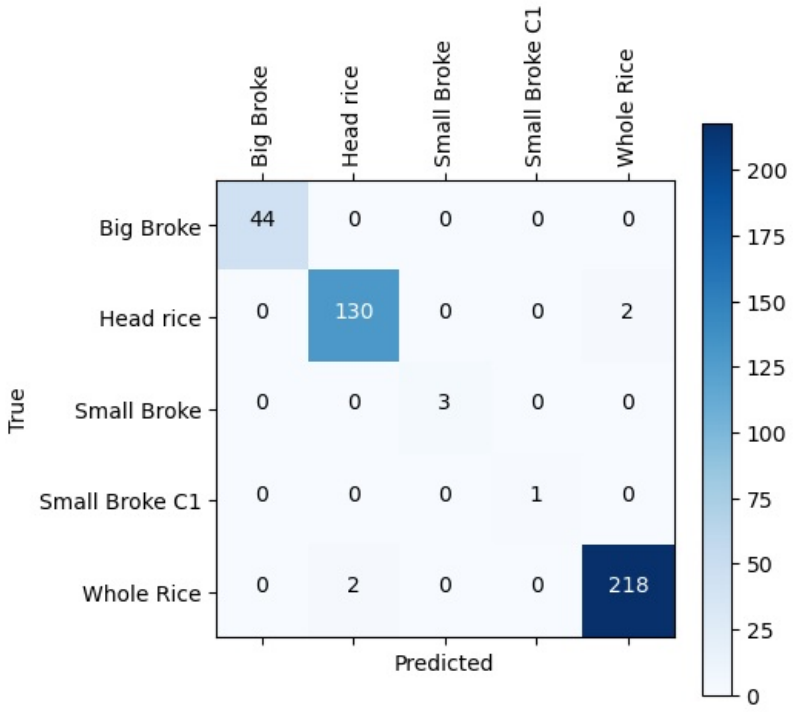
	index	area	major_axis_length	perimeter	equiv_diameter	category	predicted
0	0	1101	64.433811	139.707	37.441110	Whole Rice	Whole Rice
1	1	1105	65.867849	143.454	37.509062	Whole Rice	Whole Rice
2	2	1105	66.476728	144.347	37.509062	Whole Rice	Whole Rice
3	3	1080	65.570792	141.610	37.082323	Whole Rice	Whole Rice
4	4	1083	66.847603	142.999	37.133791	Whole Rice	Whole Rice
...	...	...	...	...	...	...	...
395	395	849	56.702281	124.807	32.878266	Big Broke	Big Broke
396	396	515	32.980736	82.710	25.606998	Small Broke	Small Broke
397	397	505	35.550758	84.001	25.357168	Small Broke	Small Broke
398	398	455	34.490506	80.808	24.069150	Small Broke	Small Broke
399	399	328	21.786792	67.559	20.435816	Small Broke C1	Small Broke C1

400 rows × 7 columns

In [139…
```
Testing_accuracy = accuracy_score(y_test3, y_pred3)
print('Accuracy on testing data: {:.2f}%'.format(Testing_accuracy * 100))
```

```
Accuracy on testing data: 99.00%
```

In [140…
```
category_labels = sorted(y_test3['category'].unique())

cm = confusion_matrix(y_test3, y_pred3)
plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar()

plt.xticks(np.arange(len(category_labels)), category_labels, rotation='vertical')
plt.yticks(np.arange(len(category_labels)), category_labels)

plt.xlabel('Predicted')
plt.ylabel('True')

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), horizontalalignment="center", color="white" if cm[i, j] > cm.max() / 2. e

plt.show()
```



In [141…
```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(f"Total Computation Time: {Computation_time:.4f} seconds")
print("Accuracy Score:", accuracy_score(y_test3, y_pred3) * 100)
print("Testing Report:\n", classification_report(y_test3, y_pred3, digits=4))
```

```
Total Computation Time: 0.0160 seconds
Accuracy Score: 99.0
Testing Report:
                precision    recall  f1-score   support

     Big Broke     1.0000    1.0000    1.0000        44
     Head rice     0.9848    0.9848    0.9848       132
   Small Broke     1.0000    1.0000    1.0000         3
Small Broke C1     1.0000    1.0000    1.0000         1
    Whole Rice     0.9909    0.9909    0.9909       220

      accuracy                         0.9900       400
     macro avg     0.9952    0.9952    0.9952       400
  weighted avg     0.9900    0.9900    0.9900       400
```

In [142…
```python
# Calculate metrics
accuracy = accuracy_score(y_test3, y_pred3)
classification_rep = classification_report(y_test3, y_pred3, digits=4, output_dict=True)
macro_avg_precision = classification_rep['macro avg']['precision']
macro_avg_recall = classification_rep['macro avg']['recall']
macro_avg_f1_score = classification_rep['macro avg']['f1-score']

# Create the result1 dictionary with the calculated values
result3 = {
    'Total Computation Time': Computation_time,
    'Accuracy': accuracy * 100,
    'Macro Avg Precision': macro_avg_precision * 100,
    'Macro Avg Recall': macro_avg_recall * 100,
    'Macro Avg F1-Score': macro_avg_f1_score * 100
}

# Print the formatted values
for key, value in result3.items():
    if isinstance(value, float):
        # Format floats to display exactly four decimal places
        formatted_value = '{:.4f}'.format(value)
    else:
        formatted_value = value  # Leave non-float values as they are

    print(f'{key}: {formatted_value}')
```

```
Total Computation Time: 0.0160
Accuracy: 99.0000
Macro Avg Precision: 99.5152
Macro Avg Recall: 99.5152
Macro Avg F1-Score: 99.5152
```

In [ ]:

Sample4

In [143…
```
df_4 = pd.read_csv('Sample4_Data.csv')
```

In [144…
```
X_test4=df_4[['area', 'major_axis_length',  'perimeter','equiv_diameter']]
y_test4=df_4[['category']]
```

In [145…
```
start_time = time.time()

y_pred4=model.predict(X_test4)

end_time = time.time()

Computation_time = end_time - start_time
```

In [146…
```
test4=pd.concat([X_test4, y_test4], axis='columns')
```

In [147…
```
dc4=pd.concat([test4.reset_index(), pd.Series(y_pred4, name='predicted')], axis='columns')
```

In [148…
```
dc4
```

Out[148]:

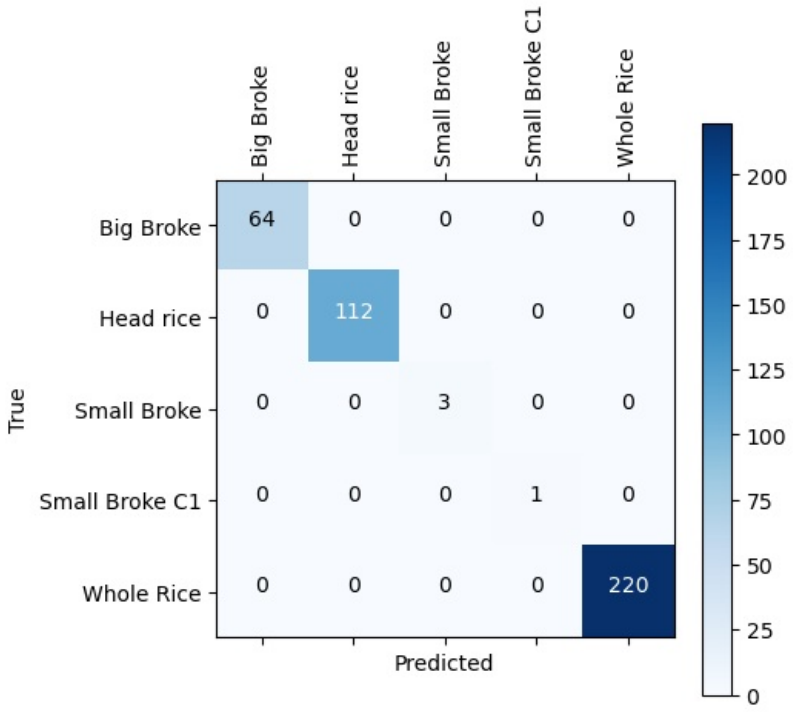
	index	area	major_axis_length	perimeter	equiv_diameter	category	predicted
0	0	1140	70.631431	149.690	38.098466	Whole Rice	Whole Rice
1	1	1164	70.585455	150.840	38.497413	Whole Rice	Whole Rice
2	2	1084	70.207957	148.689	37.150931	Whole Rice	Whole Rice
3	3	1310	72.602047	157.009	40.840468	Whole Rice	Whole Rice
4	4	1114	68.128409	145.498	37.661504	Whole Rice	Whole Rice
...	...	...	...	...	...	...	...
395	395	887	51.414365	117.668	33.606004	Big Broke	Big Broke
396	396	456	29.597383	78.153	24.095585	Small Broke	Small Broke
397	397	457	29.612799	77.865	24.121991	Small Broke	Small Broke
398	398	462	29.267010	77.666	24.253591	Small Broke	Small Broke
399	399	294	21.948536	58.818	19.347672	Small Broke C1	Small Broke C1

400 rows × 7 columns

In [149…
```
Testing_accuracy = accuracy_score(y_test4, y_pred4)
print('Accuracy on testing data: {:.2f}%'.format(Testing_accuracy * 100))
```

```
Accuracy on testing data: 100.00%
```

In [150…
```
category_labels = sorted(y_test4['category'].unique())

cm = confusion_matrix(y_test4, y_pred4)
plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar()

plt.xticks(np.arange(len(category_labels)), category_labels, rotation='vertical')
plt.yticks(np.arange(len(category_labels)), category_labels)

plt.xlabel('Predicted')
plt.ylabel('True')

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, str(cm[i, j]), horizontalalignment="center", color="white" if cm[i, j] > cm.max() / 2. e

plt.show()
```



In [151…
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print(f"Total Computation Time: {Computation_time:.4f} seconds")
print("Accuracy Score:", accuracy_score(y_test4, y_pred4) * 100)
print("Testing Report:\n", classification_report(y_test4, y_pred4, digits=4))
```

```
Total Computation Time: 0.0192 seconds
Accuracy Score: 100.0
Testing Report:
                precision    recall  f1-score   support

     Big Broke     1.0000    1.0000    1.0000        64
     Head rice     1.0000    1.0000    1.0000       112
   Small Broke     1.0000    1.0000    1.0000         3
Small Broke C1     1.0000    1.0000    1.0000         1
    Whole Rice     1.0000    1.0000    1.0000       220

      accuracy                         1.0000       400
     macro avg     1.0000    1.0000    1.0000       400
  weighted avg     1.0000    1.0000    1.0000       400
```

In [152…
```
# Calculate metrics
accuracy = accuracy_score(y_test4, y_pred4)
classification_rep = classification_report(y_test4, y_pred4, digits=4, output_dict=True)
macro_avg_precision = classification_rep['macro avg']['precision']
macro_avg_recall = classification_rep['macro avg']['recall']
macro_avg_f1_score = classification_rep['macro avg']['f1-score']

# Create the result1 dictionary with the calculated values
result4 = {
    'Total Computation Time': Computation_time,
    'Accuracy': accuracy * 100,
    'Macro Avg Precision': macro_avg_precision * 100,
    'Macro Avg Recall': macro_avg_recall * 100,
    'Macro Avg F1-Score': macro_avg_f1_score * 100
}

# Print the formatted values
for key, value in result4.items():
    if isinstance(value, float):
        # Format floats to display exactly four decimal places
        formatted_value = '{:.4f}'.format(value)
    else:
        formatted_value = value  # Leave non-float values as they are

    print(f'{key}: {formatted_value}')
```

```
Total Computation Time: 0.0192
Accuracy: 100.0000
Macro Avg Precision: 100.0000
Macro Avg Recall: 100.0000
Macro Avg F1-Score: 100.0000
```

In [ ]: