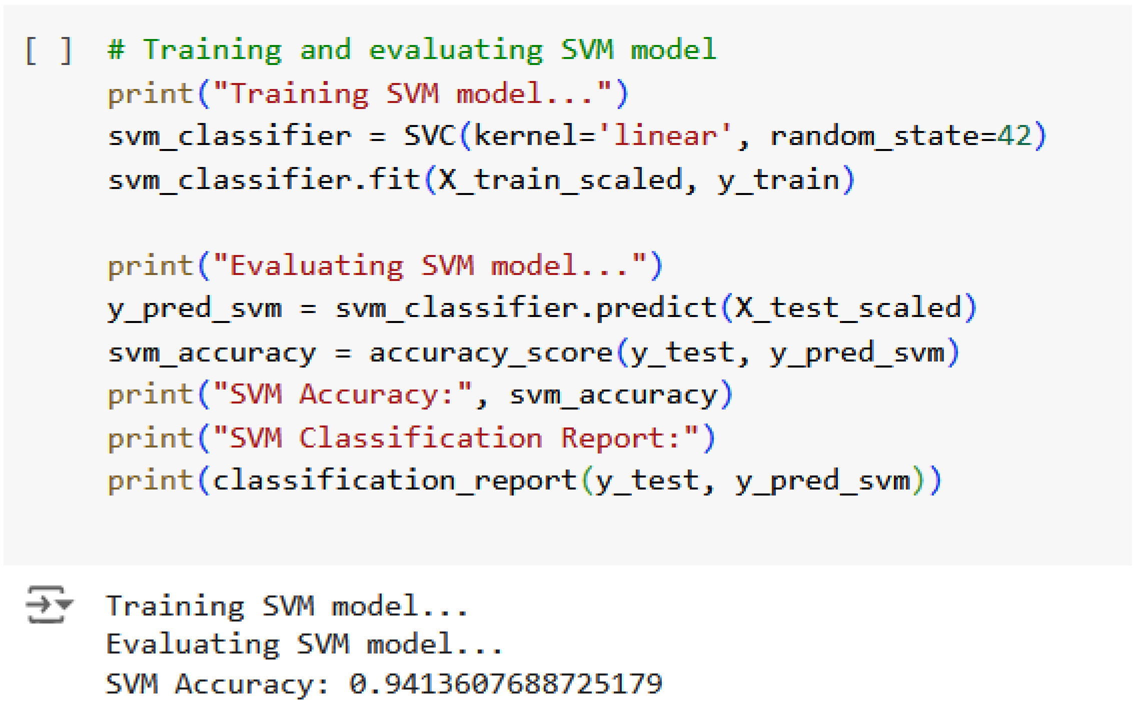
Fruits Recognition Testing Results

<u>We Have SVM, KNN, Random Forest Models:</u>

1- Accuracy of the Models before Performing Random Search

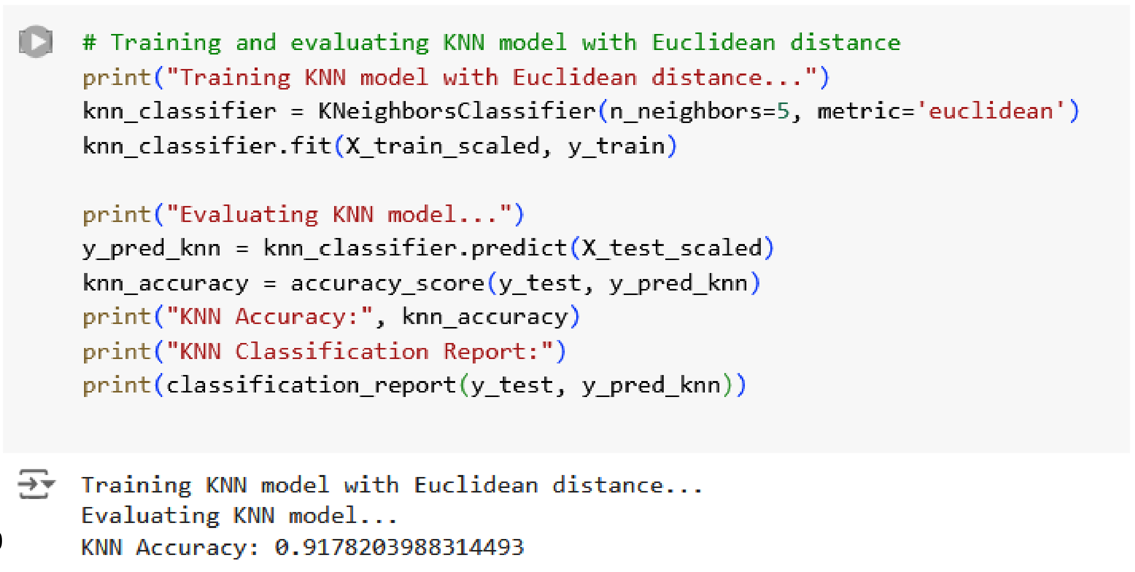
- SVM:



```
[ ] # Training and evaluating SVM model
    print("Training SVM model...")
    svm_classifier = SVC(kernel='linear', random_state=42)
    svm_classifier.fit(X_train_scaled, y_train)

    print("Evaluating SVM model...")
    y_pred_svm = svm_classifier.predict(X_test_scaled)
    svm_accuracy = accuracy_score(y_test, y_pred_svm)
    print("SVM Accuracy:", svm_accuracy)
    print("SVM Classification Report:")
    print(classification_report(y_test, y_pred_svm))
```

```
Training SVM model...
Evaluating SVM model...
SVM Accuracy: 0.9413607688725179
```

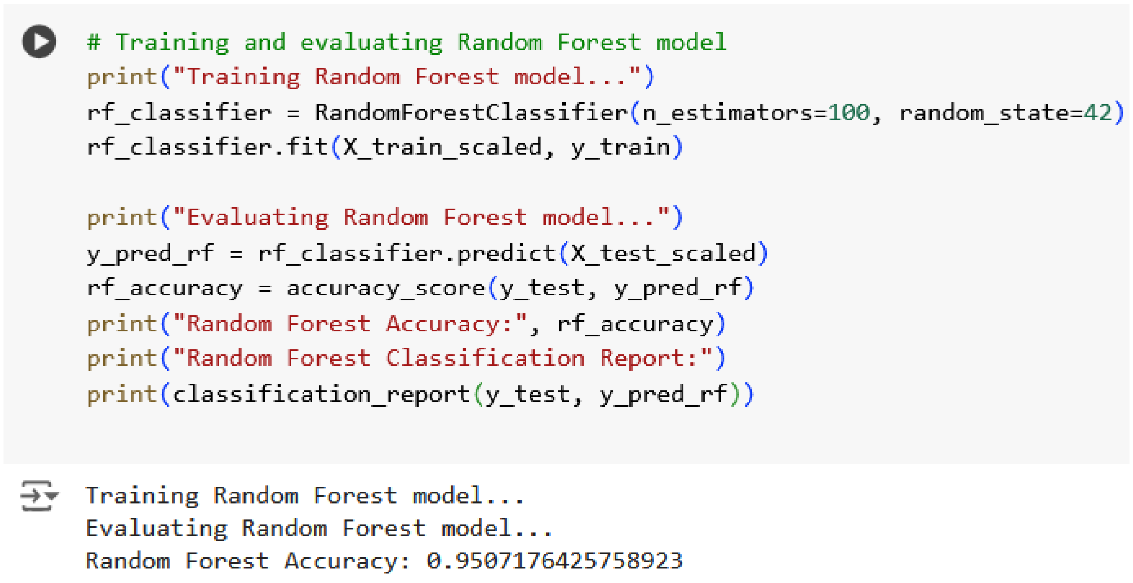
- KNN



```
# Training and evaluating KNN model with Euclidean distance
print("Training KNN model with Euclidean distance...")
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_classifier.fit(X_train_scaled, y_train)

print("Evaluating KNN model...")
y_pred_knn = knn_classifier.predict(X_test_scaled)
knn_accuracy = accuracy_score(y_test, y_pred_knn)
print("KNN Accuracy:", knn_accuracy)
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

```
Training KNN model with Euclidean distance...
Evaluating KNN model...
KNN Accuracy: 0.9178203988314493
```

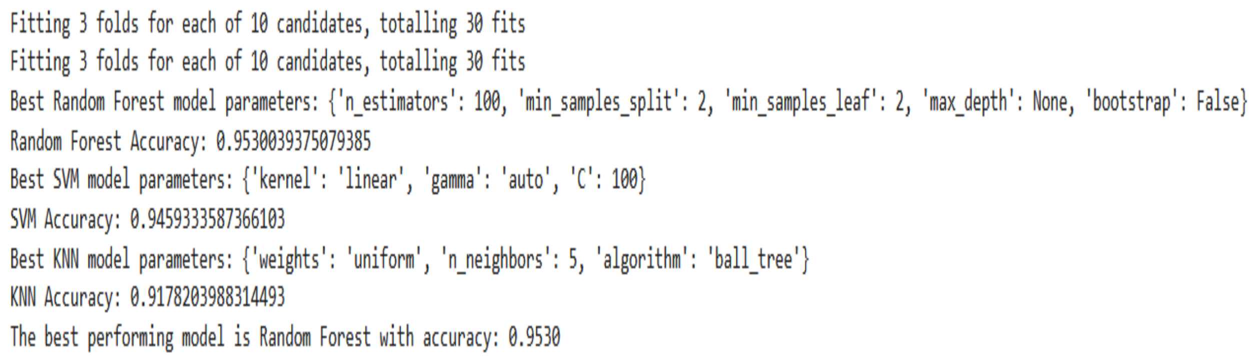
- Random Forest



```
# Training and evaluating Random Forest model
print("Training Random Forest model...")
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_scaled, y_train)

print("Evaluating Random Forest model...")
y_pred_rf = rf_classifier.predict(X_test_scaled)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

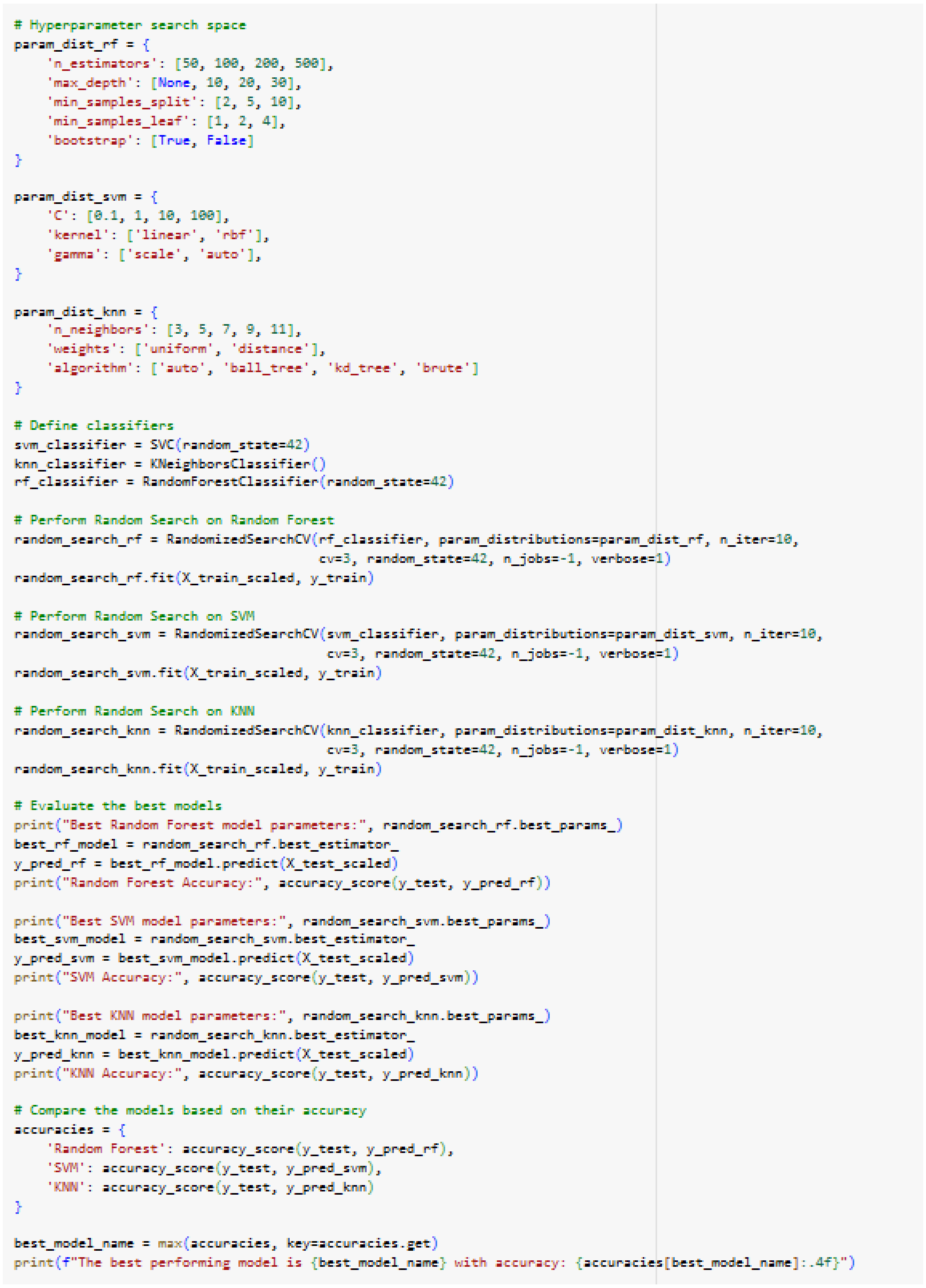
```
Training Random Forest model...
Evaluating Random Forest model...
Random Forest Accuracy: 0.9507176425758923
```

2- Models Accuracy after Random Search implementation



```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Random Forest model parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': None, 'bootstrap': False}
Random Forest Accuracy: 0.9530039375079385
Best SVM model parameters: {'kernel': 'linear', 'gamma': 'auto', 'C': 100}
SVM Accuracy: 0.9459333587366103
Best KNN model parameters: {'weights': 'uniform', 'n_neighbors': 5, 'algorithm': 'ball_tree'}
KNN Accuracy: 0.9178203988314493
The best performing model is Random Forest with accuracy: 0.9530
```

- The highest accuracy reached was **95.3%** using **random search** with the **model Random Forest**
- Implementation:

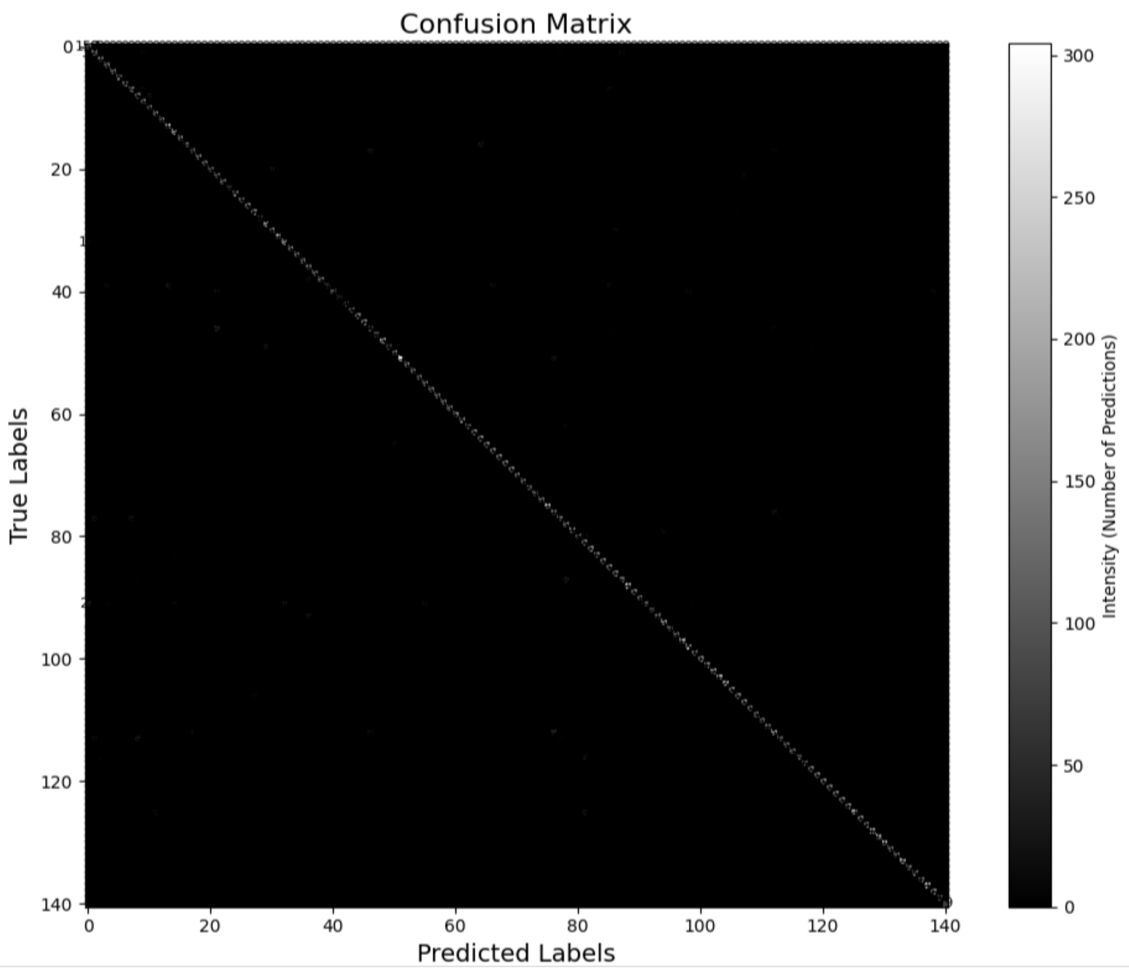


```python
# Hyperparameter search space
param_dist_rf = {
    'n_estimators': [50, 100, 200, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

param_dist_svm = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto'],
}

param_dist_knn = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Define classifiers
svm_classifier = SVC(random_state=42)
knn_classifier = KNeighborsClassifier()
rf_classifier = RandomForestClassifier(random_state=42)

# Perform Random Search on Random Forest
random_search_rf = RandomizedSearchCV(rf_classifier, param_distributions=param_dist_rf, n_iter=10,
                                      cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_rf.fit(X_train_scaled, y_train)

# Perform Random Search on SVM
random_search_svm = RandomizedSearchCV(svm_classifier, param_distributions=param_dist_svm, n_iter=10,
                                       cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_svm.fit(X_train_scaled, y_train)

# Perform Random Search on KNN
random_search_knn = RandomizedSearchCV(knn_classifier, param_distributions=param_dist_knn, n_iter=10,
                                       cv=3, random_state=42, n_jobs=-1, verbose=1)
random_search_knn.fit(X_train_scaled, y_train)

# Evaluate the best models
print("Best Random Forest model parameters:", random_search_rf.best_params_)
best_rf_model = random_search_rf.best_estimator_
y_pred_rf = best_rf_model.predict(X_test_scaled)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

print("Best SVM model parameters:", random_search_svm.best_params_)
best_svm_model = random_search_svm.best_estimator_
y_pred_svm = best_svm_model.predict(X_test_scaled)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))

print("Best KNN model parameters:", random_search_knn.best_params_)
best_knn_model = random_search_knn.best_estimator_
y_pred_knn = best_knn_model.predict(X_test_scaled)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))

# Compare the models based on their accuracy
accuracies = {
    'Random Forest': accuracy_score(y_test, y_pred_rf),
    'SVM': accuracy_score(y_test, y_pred_svm),
    'KNN': accuracy_score(y_test, y_pred_knn)
}

best_model_name = max(accuracies, key=accuracies.get)
print(f"The best performing model is {best_model_name} with accuracy: {accuracies[best_model_name]:.4f}")
```

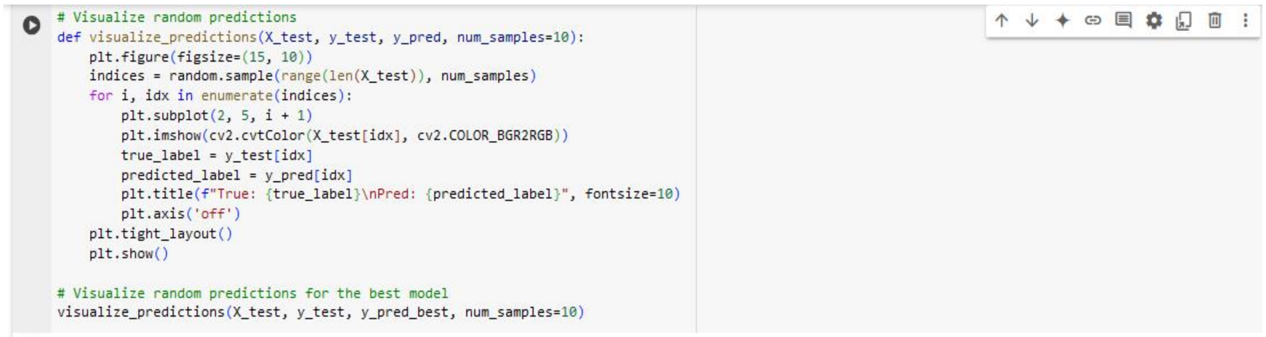
3- Confusion Matrix

```
[ ] # Confusion matrix
    def plot_confusion_matrix(y_true, y_pred, class_names):
        cm = confusion_matrix(y_true, y_pred, labels=class_names)
        plt.figure(figsize=(10, 8))
        plt.imshow(cm, cmap='gray', interpolation='nearest')
        plt.title('Confusion Matrix', fontsize=16)
        plt.xlabel('Predicted Labels', fontsize=14)
        plt.ylabel('True Labels', fontsize=14)
        for i in range(len(class_names)):
            for j in range(len(class_names)):
                plt.text(j, i, cm[i, j], ha='center', va='center', color='black', fontsize=8)
        plt.tight_layout()
        plt.colorbar(label="Intensity (Number of Predictions)")
        plt.show()
```

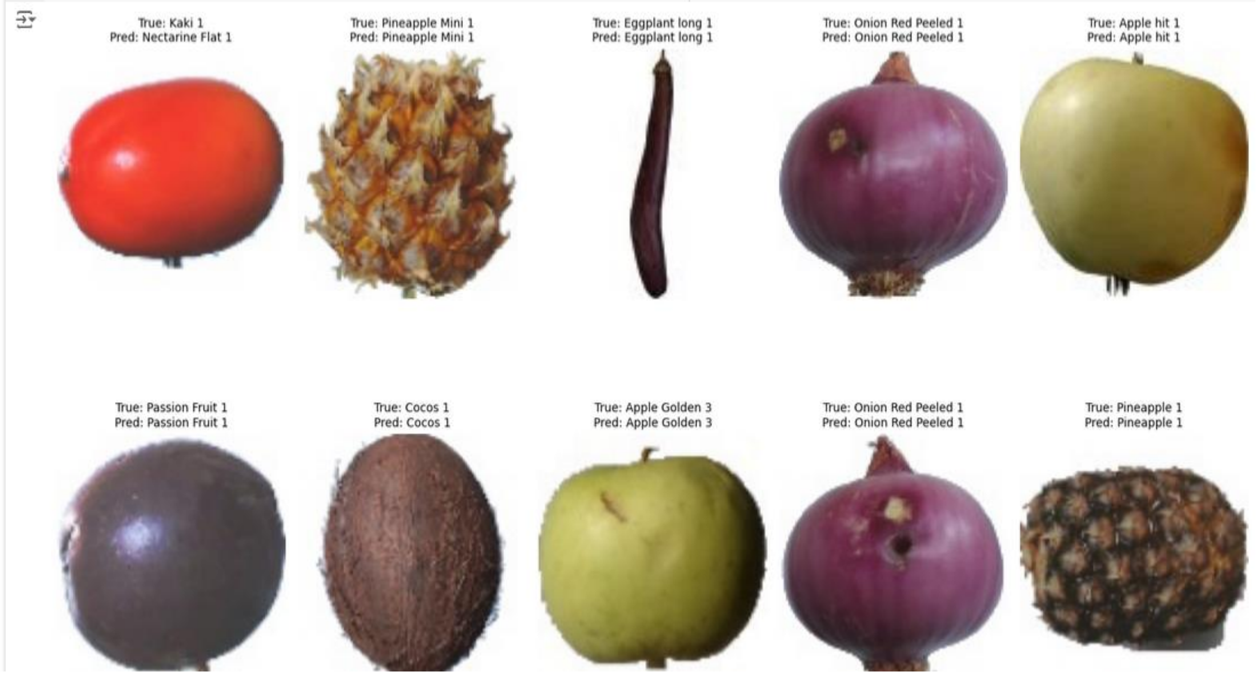
```
[ ] # Generate confusion matrix for the best model
    class_names = np.unique(y_test)
    plot_confusion_matrix(y_test, y_pred_best, class_names)
```



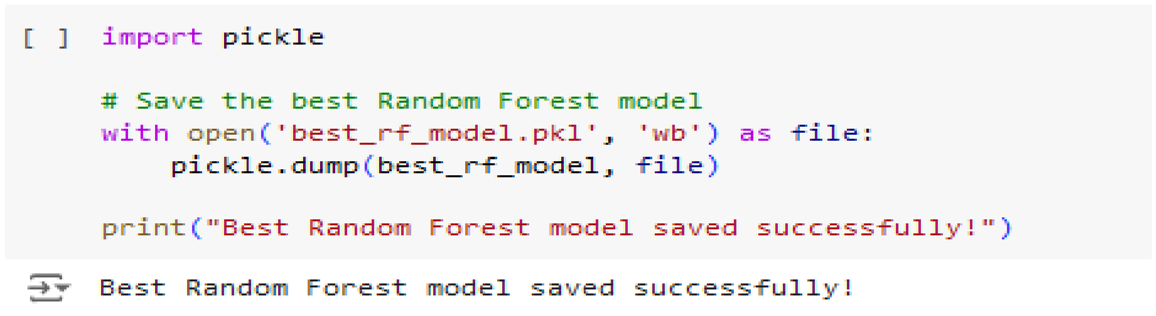
4- Testing the prediction using the testing images in the dataset



```
# Visualize random predictions
def visualize_predictions(X_test, y_test, y_pred, num_samples=10):
    plt.figure(figsize=(15, 10))
    indices = random.sample(range(len(X_test)), num_samples)
    for i, idx in enumerate(indices):
        plt.subplot(2, 5, i + 1)
        plt.imshow(cv2.cvtColor(X_test[idx], cv2.COLOR_BGR2RGB))
        true_label = y_test[idx]
        predicted_label = y_pred[idx]
        plt.title(f"True: {true_label}\nPred: {predicted_label}", fontsize=10)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Visualize random predictions for the best model
visualize_predictions(X_test, y_test, y_pred_best, num_samples=10)
```



5- Saving the best model to use it in the website



```
[ ] import pickle

    # Save the best Random Forest model
    with open('best_rf_model.pkl', 'wb') as file:
        pickle.dump(best_rf_model, file)

    print("Best Random Forest model saved successfully!")
```

```
Best Random Forest model saved successfully!
```