```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from scipy.stats import skew, kurtosis

# Load the CSV files containing concentration profiles
C_high_diff = pd.read_csv('/content/C_high_diff.csv', header=None)
C_low_diff = pd.read_csv('/content/C_low_diff.csv', header=None)
C_normal = pd.read_csv('/content/C_normal.csv', header=None)
C_low_init = pd.read_csv('/content/C_low_init.csv', header=None)
C_anisotropic = pd.read_csv('/content/C_anisotropic.csv', header=None)
C_multiple_high = pd.read_csv('/content/C_multiple_high.csv', header=None)

# Inspect the data shapes
print("C_high_diff shape:", C_high_diff.shape)
print("C_low_diff shape:", C_low_diff.shape)
print("C_normal shape:", C_normal.shape)

# Define a function to extract features from the concentration profiles
def extract_features(df_row):
    """
    Safely extract features from a single row of concentration profile data.
    """
    if df_row.isnull().any():  # Check for missing values
        return [np.nan] * 7  # Return placeholder values

    last_col = df_row
    mean = last_col.mean()
    max_val = last_col.max()
    min_val = last_col.min()
    std_dev = last_col.std()
    skewness = skew(last_col)
    kurt = kurtosis(last_col)

    # Check for very large or infinite values and replace with 0
    if np.isinf(mean) or np.isnan(mean):
        mean = 0
    if np.isinf(max_val) or np.isnan(max_val):
        max_val = 0
    if np.isinf(min_val) or np.isnan(min_val):
        min_val = 0
    if np.isinf(std_dev) or np.isnan(std_dev):
        std_dev = 0
    if np.isinf(skewness) or np.isnan(skewness):
        skewness = 0
    if np.isinf(kurt) or np.isnan(kurt):
        kurt = 0

    return [mean, max_val, min_val, std_dev, skewness, kurt]

# Extract features from all datasets
features = []
labels = []

# High Diffusion
labels.extend([0] * len(C_high_diff))
features.extend(C_high_diff.apply(lambda row: extract_features(row), axis=1))

# Low Diffusion
labels.extend([1] * len(C_low_diff))
features.extend(C_low_diff.apply(lambda row: extract_features(row), axis=1))

# Normal Diffusion
labels.extend([2] * len(C_normal))
features.extend(C_normal.apply(lambda row: extract_features(row), axis=1))

# Low Initial Concentration
labels.extend([3] * len(C_low_init))
features.extend(C_low_init.apply(lambda row: extract_features(row), axis=1))

# Anisotropic Diffusion
labels.extend([4] * len(C_anisotropic))
features.extend(C_anisotropic.apply(lambda row: extract_features(row), axis=1))

# Multiple Source High Diffusion
labels.extend([5] * len(C_multiple_high))
features.extend(C_multiple_high.apply(lambda row: extract_features(row), axis=1))

# Convert to pandas DataFrame
features_df = pd.DataFrame(features, columns=['Mean', 'Max', 'Min', 'Std', 'Skewness', 'Kurtosis'])
labels_df = pd.Series(labels)

# Check for inf or NaN values in the features
if np.any(np.isinf(features_df)) or np.any(np.isnan(features_df)):
    print("Features contain NaN or Inf values. Replacing with column mean.")
    features_df = features_df.replace([np.inf, -np.inf], np.nan).fillna(features_df.mean())

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_df, labels_df, test_size=0.3, random_state=42)

# Check for inf or NaN values in X_train and X_test
if np.any(np.isinf(X_train)) or np.any(np.isnan(X_train)):
    print("X_train contains NaN or Inf values. Replacing with column mean.")
    X_train = X_train.replace([np.inf, -np.inf], np.nan).fillna(X_train.mean())

if np.any(np.isinf(X_test)) or np.any(np.isnan(X_test)):
    print("X_test contains NaN or Inf values. Replacing with column mean.")
    X_test = X_test.replace([np.inf, -np.inf], np.nan).fillna(X_test.mean())

# Initialize the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
print("Training the model...")
clf.fit(X_train, y_train)
print("Model trained.")

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model's performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

# Misclassification Analysis:
misclassified_samples = X_test[y_test != y_pred]
misclassified_labels = y_test[y_test != y_pred]
misclassified_preds = y_pred[y_test != y_pred]
```

```
print(f"Misclassified samples:\n{misclassified_samples}")
print(f"True Labels:\n{misclassified_labels}")
print(f"Predicted Labels:\n{misclassified_preds}")

# Confusion Matrix with colorful visualization
cm = confusion_matrix(y_test, y_pred)

# Create a more vibrant and informative heatmap using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='YlGnBu', cbar=True,
            xticklabels=np.unique(labels), yticklabels=np.unique(labels),
            linewidths=0.5, linecolor='black', annot_kws={'size': 12, 'weight': 'bold'})

# Titles and labels
plt.title('Confusion Matrix', fontsize=16, weight='bold')
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.xticks(rotation=45)
plt.yticks(rotation=45)

# Display the plot
plt.show()

# Misclassification Rate per Class
misclassification_rate = 1 - np.diagonal(cm) / np.sum(cm, axis=1)
print(f"Misclassification rate per class:\n{misclassification_rate}")

# Visualize the feature importance
feature_importances = clf.feature_importances_
plt.barh(features_df.columns, feature_importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importance from Random Forest Model')
plt.show()

# Predict on the full dataset (for visualization purposes)
y_full_pred = clf.predict(features_df)

# Visualize true vs predicted labels
plt.figure(figsize=(12, 8))
plt.scatter(range(len(labels)), labels, color='blue', label='True Labels', alpha=0.6)
plt.scatter(range(len(y_full_pred)), y_full_pred, color='red', label='Predicted Labels', alpha=0.6)
plt.legend()
plt.title('True vs Predicted Labels (Full Dataset)')
plt.xlabel('Index')
plt.ylabel('Diffusion Scenario')
plt.grid()
plt.show()
```

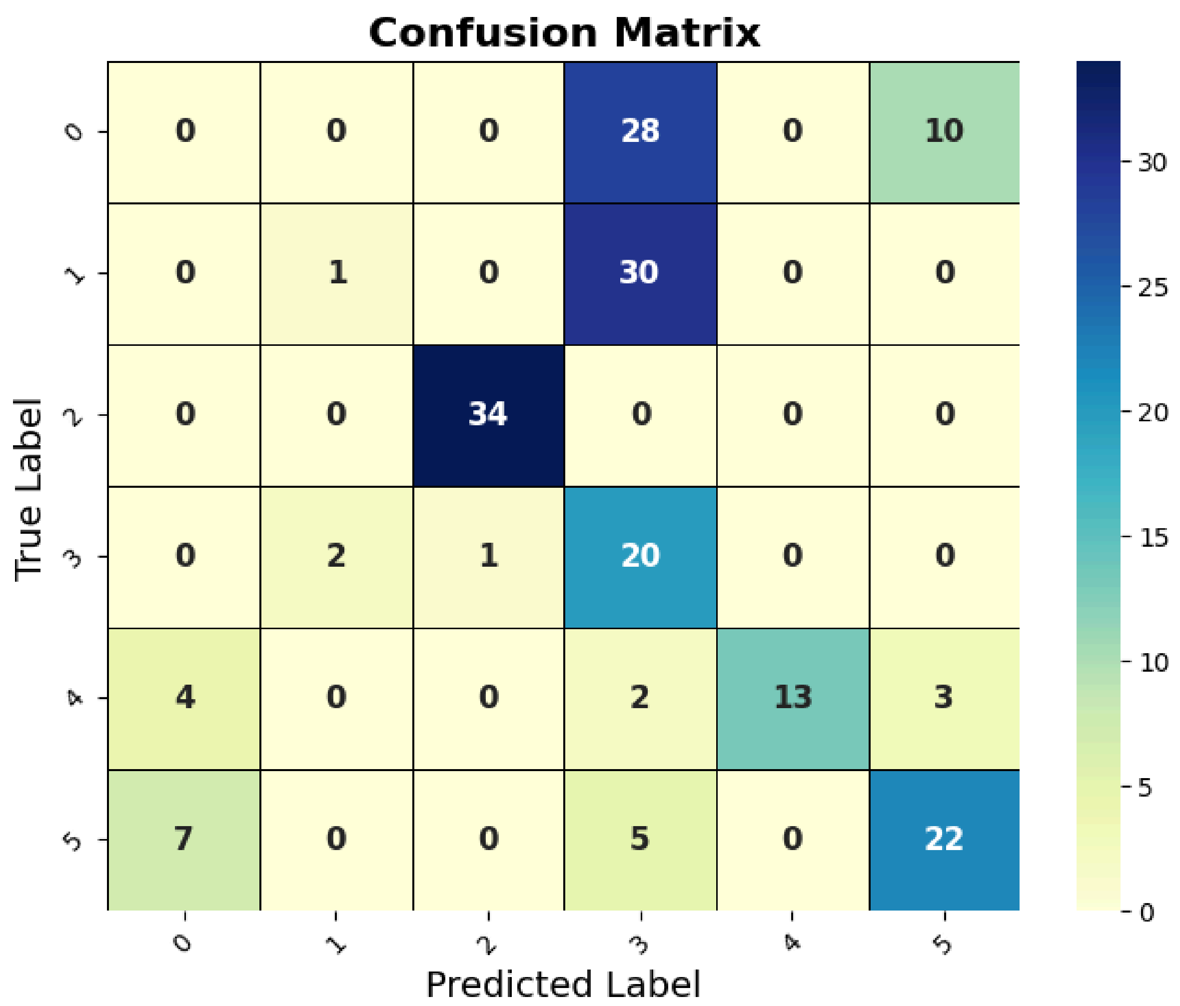
```
C_high_diff shape: (101, 501)
C_low_diff shape: (101, 501)
C_normal shape: (101, 501)
Training the model...
Model trained.
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        38
           1       0.33      0.03      0.06        31
           2       0.97      1.00      0.99        34
           3       0.24      0.87      0.37        23
           4       1.00      0.59      0.74        22
           5       0.63      0.65      0.64        34

    accuracy                           0.49       182
   macro avg       0.53      0.52      0.47       182
weighted avg       0.51      0.49      0.45       182

Accuracy: 0.4945054945054945
Misclassified samples:
              Mean            Max  Min           Std  Skewness  Kurtosis
76   7.062867e-141  1.906968e-139  0.0  2.560962e-140  0.000000  0.000000
78   1.950135e-152  5.655372e-151  0.0  7.350500e-152  0.000000  0.000000
182  2.054248e-232  6.573594e-231  0.0   0.000000e+00  0.000000  0.000000
10   5.473337e-223  2.244063e-221  0.0   0.000000e+00  0.000000  0.000000
131  3.810280e-147  8.001588e-146  0.0  1.204629e-146  0.000000  0.000000
..             ...            ...  ...            ...       ...       ...
54    5.145637e-20   2.572776e-19  0.0   6.889118e-20  1.389982  0.803161
434   1.306360e-27   9.144211e-27  0.0   2.184952e-27  1.906617  2.702452
46    5.145637e-20   2.572776e-19  0.0   6.889118e-20  1.389982  0.803161
93   6.660670e-241  2.930689e-239  0.0   0.000000e+00  0.000000  0.000000
108   0.000000e+00   0.000000e+00  0.0   0.000000e+00  0.000000  0.000000

[92 rows x 6 columns]
True Labels:
76     0
78     0
182    1
10     0
131    1
      ..
54     0
434    4
46     0
93     0
108    1
Length: 92, dtype: int64
Predicted Labels:
[3 3 3 3 3 0 3 5 5 5 3 0 3 3 3 3 3 3 1 3 3 3 3 0 0 3 3 3 3 3 5 3 3 3 3 3 3
 3 3 3 3 3 3 5 0 3 5 0 0 3 3 3 3 3 3 5 3 3 3 3 3 3 3 3 0 3 3 5 3 3 3 3 0 5
 5 3 0 1 3 3 3 2 3 0 3 3 3 5 5 5 3 3]
```




```
Misclassification rate per class:
[1.         0.96774194 0.         0.13043478 0.40909091 0.35294118]
```