```python
In [1]:   1  import pandas as pd
          2  import numpy as np
          3  from sklearn.model_selection import train_test_split
          4  from sklearn.preprocessing import StandardScaler
          5  from sklearn.neighbors import KNeighborsClassifier
          6  from sklearn.metrics import classification_report
          7  from sklearn.metrics import recall_score, f1_score, confusion_matrix
          8  from sklearn.metrics import accuracy_score, precision_score
          9  import matplotlib.pyplot as plt
```

```python
In [2]:   1  # Load the dataset
          2  dataset_path = "D:/MTP/Mid Term/MId Term 256 models.csv"
          3  data = pd.read_csv(dataset_path)
          4  data.head()
```

Out[2]:

	SI No	x1	y1	x2	y2	area	Model horizontal Class	Model Vertical class	Model Box Class	natural frequency of Mode 1	natural frequency of Mode 2	natural frequency of Mode 3
0	1	0.00	0.0	0.06	0.06	0.0036	1	1	1	6.8818	25.361	27.925
1	2	0.06	0.0	0.12	0.06	0.0036	1	1	1	6.8854	25.535	27.926
2	3	0.12	0.0	0.18	0.06	0.0036	1	1	1	6.9176	25.713	27.932
3	4	0.18	0.0	0.24	0.06	0.0036	2	1	1	6.9214	25.759	27.926
4	5	0.24	0.0	0.30	0.06	0.0036	2	1	1	6.9401	25.767	27.916

```python
In [3]:   1  # Extract features and target variables
          2  X = data.iloc[:, 9:]   # Features: natural frequencies
          3  y_horizontal = data['Model horizontal Class']
          4  y_vertical = data['Model Vertical class']
          5  y_box = data['Model Box  Class']
```

```python
In [4]:   1  def classify_knn(X, y, label):
          2      # Split the data into training and testing sets
          3      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
          4
          5      # Standardize features
          6      scaler = StandardScaler()
          7      X_train_scaled = scaler.fit_transform(X_train)
          8      X_test_scaled = scaler.transform(X_test)
          9
         10      # Train the KNN classifier
         11      clf = KNeighborsClassifier()
         12      clf.fit(X_train_scaled, y_train)
         13
         14      # Make predictions
         15      y_train_pred = clf.predict(X_train_scaled)
         16      y_test_pred = clf.predict(X_test_scaled)
         17
         18      return y_train, y_train_pred, y_test, y_test_pred
```

```
In [5]:    1  # Perform classification for each target variable
           2  train_actual_horizontal, train_predicted_horizontal, test_actual_horizonta
           3  train_actual_vertical, train_predicted_vertical, test_actual_vertical, tes
           4  train_actual_box, train_predicted_box, test_actual_box, test_predicted_box
```

...

```
In [6]:    1  train_results_df = pd.DataFrame({
           2      'Actual Horizontal Class': train_actual_horizontal,
           3      'Predicted Horizontal Class': train_predicted_horizontal,
           4      'Actual Vertical Class': train_actual_vertical,
           5      'Predicted Vertical Class': train_predicted_vertical,
           6      'Actual Box Class': train_actual_box,
           7      'Predicted Box Class': train_predicted_box
           8  })
           9
          10  test_results_df = pd.DataFrame({
          11      'Actual Horizontal Class': test_actual_horizontal,
          12      'Predicted Horizontal Class': test_predicted_horizontal,
          13      'Actual Vertical Class': test_actual_vertical,
          14      'Predicted Vertical Class': test_predicted_vertical,
          15      'Actual Box Class': test_actual_box,
          16      'Predicted Box Class': test_predicted_box
          17  })
```

```
In [7]:    1  train_results_df['Horizontal Class Match'] = np.where(train_results_df['Ac
           2  train_results_df['Vertical Class Match'] = np.where(train_results_df['Actu
           3  train_results_df['Box Class Match'] = np.where(train_results_df['Actual Bc
           4
           5  test_results_df['Horizontal Class Match'] = np.where(test_results_df['Actu
           6  test_results_df['Vertical Class Match'] = np.where(test_results_df['Actual
           7  test_results_df['Box Class Match'] = np.where(test_results_df['Actual Box
           8
           9  train_results_df['Cell Number'] = range(1, len(train_results_df) + 1)
          10  test_results_df['Cell Number'] = range(len(train_results_df) + 1, len(trai
```

```
In [8]:  1  # Counting 'Yes' and 'No' in training results
         2  train_yes_no_counts = {
         3      'Horizontal Class Match': train_results_df['Horizontal Class Match'].v
         4      'Vertical Class Match': train_results_df['Vertical Class Match'].value
         5      'Box Class Match': train_results_df['Box Class Match'].value_counts()
         6  }
         7
         8  print("Training Results 'Yes' and 'No' Counts:")
         9  for key, value in train_yes_no_counts.items():
        10      print(f"{key}:")
        11      print(f"Yes: {value['Yes']}, No: {value['No']}")
        12      print()
```

```
Training Results 'Yes' and 'No' Counts:
Horizontal Class Match:
Yes: 163, No: 41

Vertical Class Match:
Yes: 166, No: 38

Box Class Match:
Yes: 162, No: 42
```

```
In [9]:  1  # Counting 'Yes' and 'No' in testing results
         2  test_yes_no_counts = {
         3      'Horizontal Class Match': test_results_df['Horizontal Class Match'].va
         4      'Vertical Class Match': test_results_df['Vertical Class Match'].value_
         5      'Box Class Match': test_results_df['Box Class Match'].value_counts()
         6  }
         7
         8  print("\nTesting Results 'Yes' and 'No' Counts:")
         9  for key, value in test_yes_no_counts.items():
        10      print(f"{key}:")
        11      print(f"Yes: {value['Yes']}, No: {value['No']}")
        12      print()
```

```
Testing Results 'Yes' and 'No' Counts:
Horizontal Class Match:
Yes: 38, No: 14

Vertical Class Match:
Yes: 36, No: 16

Box Class Match:
Yes: 41, No: 11
```

```
In [10]:  1  combined_results_df = pd.concat([train_results_df, test_results_df], ignor
```

```python
def calculate_metrics_and_counts_df(actual, predicted):
    report = classification_report(actual, predicted, output_dict=True)
    counts = {'Yes': 0, 'No': 0}
    for key, value in report.items():
        if key not in ['accuracy', 'macro avg', 'weighted avg']:
            counts['Yes'] += value['precision'] * value['support']
            counts['No'] += (1 - value['precision']) * value['support']
    metrics_df = pd.DataFrame(report).transpose()
    metrics_df = metrics_df[metrics_df.index.isin(['0', '1'])]  # Consider
    counts_df = pd.DataFrame([counts], index=['Counts'])
    metrics_counts_df = pd.concat([metrics_df, counts_df])
    return metrics_counts_df
```

```python
# Calculate metrics for entire dataset
accuracy_total_horizontal = accuracy_score(train_results_df['Actual Horizo
precision_total_horizontal = precision_score(train_results_df['Actual Hori
recall_total_horizontal = recall_score(train_results_df['Actual Horizontal
f1_total_horizontal = f1_score(train_results_df['Actual Horizontal Class']
confusion_matrix_total_horizontal = confusion_matrix(train_results_df['Act

accuracy_total_vertical = accuracy_score(train_results_df['Actual Vertical
precision_total_vertical = precision_score(train_results_df['Actual Vertic
recall_total_vertical = recall_score(train_results_df['Actual Vertical Cla
f1_total_vertical = f1_score(train_results_df['Actual Vertical Class'], tr
confusion_matrix_total_vertical = confusion_matrix(train_results_df['Actua

accuracy_total_box = accuracy_score(train_results_df['Actual Box Class'],
precision_total_box = precision_score(train_results_df['Actual Box Class']
recall_total_box = recall_score(train_results_df['Actual Box Class'], trai
f1_total_box = f1_score(train_results_df['Actual Box Class'], train_result
confusion_matrix_total_box = confusion_matrix(train_results_df['Actual Box
```

```
In [13]:  1  # Print results for entire dataset - Horizontal Class
          2  print("Metrics for Entire Dataset - Horizontal Class:")
          3  print("Accuracy:", accuracy_total_horizontal)
          4  print("Precision:", precision_total_horizontal)
          5  print("Recall:", recall_total_horizontal)
          6  print("F1 Score:", f1_total_horizontal)
          7  print("Confusion Matrix:\n", confusion_matrix_total_horizontal)
          8
          9  # Print results for entire dataset - Vertical Class
         10  print("\nMetrics for Entire Dataset - Vertical Class:")
         11  print("Accuracy:", accuracy_total_vertical)
         12  print("Precision:", precision_total_vertical)
         13  print("Recall:", recall_total_vertical)
         14  print("F1 Score:", f1_total_vertical)
         15  print("Confusion Matrix:\n", confusion_matrix_total_vertical)
         16
         17  # Print results for entire dataset - Box Class
         18  print("\nMetrics for Entire Dataset - Box Class:")
         19  print("Accuracy:", accuracy_total_box)
         20  print("Precision:", precision_total_box)
         21  print("Recall:", recall_total_box)
         22  print("F1 Score:", f1_total_box)
         23  print("Confusion Matrix:\n", confusion_matrix_total_box)
```

```
Metrics for Entire Dataset - Horizontal Class:
Accuracy: 0.7990196078431373
Precision: 0.8051776812259525
Recall: 0.7990196078431373
F1 Score: 0.7985833020517908
Confusion Matrix:
 [[31  3  3  0  1]
 [ 3 31  4  0  0]
 [ 3  2 41  0  0]
 [ 2  0  6 33  3]
 [ 2  0  1  8 27]]

Metrics for Entire Dataset - Vertical Class:
Accuracy: 0.8137254901960784
Precision: 0.813093745250038
Recall: 0.8137254901960784
F1 Score: 0.8124064497730993
Confusion Matrix:
 [[26  5  0  0  4]
 [ 5 36  0  0  1]
 [ 0  0 50  4  1]
 [ 0  1  5 26  3]
 [ 1  1  5  2 28]]

Metrics for Entire Dataset - Box Class:
Accuracy: 0.7941176470588235
Precision: 0.8084463446011889
Recall: 0.7941176470588235
F1 Score: 0.7942801079498097
Confusion Matrix:
 [[12  3  0  2  0  0  0  0  1]
 [ 2 18  0  0  0  0  0  0  0]
```

```
[ 0  0 23  0  0  0  1  0  0]
[ 4  0  0 12  3  0  2  1  0]
[ 0  3  0  0 15  0  0  1  0]
[ 0  1  2  0  1 25  0  0  0]
[ 0  0  0  0  1  0 20  1  0]
[ 0  0  0  2  3  0  3 13  0]
[ 2  1  0  0  1  0  0  1 24]]
```

In [14]:
```python
 1  # Calculate metrics for training data
 2  accuracy_train_horizontal = accuracy_score(train_results_df['Actual Horizc
 3  precision_train_horizontal = precision_score(train_results_df['Actual Hori
 4  recall_train_horizontal = recall_score(train_results_df['Actual Horizontal
 5  f1_train_horizontal = f1_score(train_results_df['Actual Horizontal Class']
 6  confusion_matrix_train_horizontal = confusion_matrix(train_results_df['Act
 7
 8  accuracy_train_vertical = accuracy_score(train_results_df['Actual Vertical
 9  precision_train_vertical = precision_score(train_results_df['Actual Vertic
10  recall_train_vertical = recall_score(train_results_df['Actual Vertical Cla
11  f1_train_vertical = f1_score(train_results_df['Actual Vertical Class'], tr
12  confusion_matrix_train_vertical = confusion_matrix(train_results_df['Actua
13
14  accuracy_train_box = accuracy_score(train_results_df['Actual Box Class'],
15  precision_train_box = precision_score(train_results_df['Actual Box Class']
16  recall_train_box = recall_score(train_results_df['Actual Box Class'], trai
17  f1_train_box = f1_score(train_results_df['Actual Box Class'], train_result
18  confusion_matrix_train_box = confusion_matrix(train_results_df['Actual Box
```

```
In [15]:  1  # Print results for training data - Horizontal Class
          2  print("\nMetrics for Training Data - Horizontal Class:")
          3  print("Accuracy:", accuracy_train_horizontal)
          4  print("Precision:", precision_train_horizontal)
          5  print("Recall:", recall_train_horizontal)
          6  print("F1 Score:", f1_train_horizontal)
          7  print("Confusion Matrix:\n", confusion_matrix_train_horizontal)
          8
          9  # Print results for training data - Vertical Class
         10  print("\nMetrics for Training Data - Vertical Class:")
         11  print("Accuracy:", accuracy_train_vertical)
         12  print("Precision:", precision_train_vertical)
         13  print("Recall:", recall_train_vertical)
         14  print("F1 Score:", f1_train_vertical)
         15  print("Confusion Matrix:\n", confusion_matrix_train_vertical)
         16
         17  # Print results for training data - Box Class
         18  print("\nMetrics for Training Data - Box Class:")
         19  print("Accuracy:", accuracy_train_box)
         20  print("Precision:", precision_train_box)
         21  print("Recall:", recall_train_box)
         22  print("F1 Score:", f1_train_box)
         23  print("Confusion Matrix:\n", confusion_matrix_train_box)
```

```
Metrics for Training Data - Horizontal Class:
Accuracy: 0.7990196078431373
Precision: 0.8051776812259525
Recall: 0.7990196078431373
F1 Score: 0.7985833020517908
Confusion Matrix:
 [[31  3  3  0  1]
 [ 3 31  4  0  0]
 [ 3  2 41  0  0]
 [ 2  0  6 33  3]
 [ 2  0  1  8 27]]

Metrics for Training Data - Vertical Class:
Accuracy: 0.8137254901960784
Precision: 0.813093745250038
Recall: 0.8137254901960784
F1 Score: 0.8124064497730993
Confusion Matrix:
 [[26  5  0  0  4]
 [ 5 36  0  0  1]
 [ 0  0 50  4  1]
 [ 0  1  5 26  3]
 [ 1  1  5  2 28]]

Metrics for Training Data - Box Class:
Accuracy: 0.7941176470588235
Precision: 0.8084463446011889
Recall: 0.7941176470588235
F1 Score: 0.7942801079498097
Confusion Matrix:
 [[12  3  0  2  0  0  0  0  1]
```

```
[ 2 18  0  0  0  0  0  0  0]
[ 0  0 23  0  0  0  1  0  0]
[ 4  0  0 12  3  0  2  1  0]
[ 0  3  0  0 15  0  0  1  0]
[ 0  1  2  0  1 25  0  0  0]
[ 0  0  0  0  1  0 20  1  0]
[ 0  0  0  2  3  0  3 13  0]
[ 2  1  0  0  1  0  0  1 24]]
```

In [16]:

```python
# Calculate metrics for testing data
accuracy_test_horizontal = accuracy_score(test_results_df['Actual Horizont
precision_test_horizontal = precision_score(test_results_df['Actual Horizo
recall_test_horizontal = recall_score(test_results_df['Actual Horizontal C
f1_test_horizontal = f1_score(test_results_df['Actual Horizontal Class'],
confusion_matrix_test_horizontal = confusion_matrix(test_results_df['Actua

accuracy_test_vertical = accuracy_score(test_results_df['Actual Vertical C
precision_test_vertical = precision_score(test_results_df['Actual Vertical
recall_test_vertical = recall_score(test_results_df['Actual Vertical Class
f1_test_vertical = f1_score(test_results_df['Actual Vertical Class'], test
confusion_matrix_test_vertical = confusion_matrix(test_results_df['Actual

accuracy_test_box = accuracy_score(test_results_df['Actual Box Class'], te
precision_test_box = precision_score(test_results_df['Actual Box Class'],
recall_test_box = recall_score(test_results_df['Actual Box Class'], test_r
f1_test_box = f1_score(test_results_df['Actual Box Class'], test_results_d
confusion_matrix_test_box = confusion_matrix(test_results_df['Actual Box C
```

```
In [17]:  1  # Print results for testing data - Horizontal Class
          2  print("\nMetrics for Testing Data - Horizontal Class:")
          3  print("Accuracy:", accuracy_test_horizontal)
          4  print("Precision:", precision_test_horizontal)
          5  print("Recall:", recall_test_horizontal)
          6  print("F1 Score:", f1_test_horizontal)
          7  print("Confusion Matrix:\n", confusion_matrix_test_horizontal)
          8
          9  # Print results for testing data - Vertical Class
         10  print("\nMetrics for Testing Data - Vertical Class:")
         11  print("Accuracy:", accuracy_test_vertical)
         12  print("Precision:", precision_test_vertical)
         13  print("Recall:", recall_test_vertical)
         14  print("F1 Score:", f1_test_vertical)
         15  print("Confusion Matrix:\n", confusion_matrix_test_vertical)
         16
         17  # Print results for testing data - Box Class
         18  print("\nMetrics for Testing Data - Box Class:")
         19  print("Accuracy:", accuracy_test_box)
         20  print("Precision:", precision_test_box)
         21  print("Recall:", recall_test_box)
         22  print("F1 Score:", f1_test_box)
         23  print("Confusion Matrix:\n", confusion_matrix_test_box)
```

```
Metrics for Testing Data - Horizontal Class:
Accuracy: 0.7307692307692307
Precision: 0.8072899322899324
Recall: 0.7307692307692307
F1 Score: 0.7361651169343477
Confusion Matrix:
 [[ 9  0  1  0  0]
 [ 2  8  0  0  0]
 [ 3  3 11  1  0]
 [ 0  0  0  4  0]
 [ 0  0  0  4  6]]

Metrics for Testing Data - Vertical Class:
Accuracy: 0.6923076923076923
Precision: 0.7589285714285714
Recall: 0.6923076923076923
F1 Score: 0.7025677778186139
Confusion Matrix:
 [[8 4 0 0 1]
 [0 2 2 1 1]
 [0 1 8 0 0]
 [0 0 3 9 1]
 [0 0 2 0 9]]

Metrics for Testing Data - Box Class:
Accuracy: 0.7884615384615384
Precision: 0.8602335164835165
Recall: 0.7884615384615384
F1 Score: 0.7936036186036185
Confusion Matrix:
 [[3 0 0 3 0 0 0 0 1]
```

```
[1 4 0 0 0 0 0 0 0]
[0 0 6 0 0 0 0 0 0]
[0 0 0 3 0 0 0 0 0]
[0 0 0 0 6 0 0 0 0]
[0 0 0 0 0 1 0 0 0]
[0 0 0 1 1 0 6 0 0]
[0 0 0 2 1 0 1 5 0]
[0 0 0 0 0 0 0 0 7]]
```

In [18]:
```python
1  excel_file_path = "D:/MTP/Second part ML project/combined_results_KNN.xlsx
2  combined_results_df.to_excel(excel_file_path, index=False)
3
4  box_class_df = combined_results_df[['Actual Box Class','Predicted Box Clas
5
6  random_20_rows = box_class_df.sample(n=20, random_state=42)
7
8  print("Random 20 rows from the box_class_df DataFrame:")
9  random_20_rows
```

Random 20 rows from the box_class_df DataFrame:

Out[18]:

|	Actual Box Class	Predicted Box Class
228	1	1
6	6	6
79	1	2
206	3	3
117	7	8
185	5	5
242	3	3
167	9	9
9	9	2
30	6	6
180	7	7
222	4	4
230	9	9
217	8	4
136	6	6
68	6	6
199	6	6
15	7	7
96	4	5
24	4	8

```python
In [19]:   1  # Define the Damage_Plate grid size
           2  plate_size = 3   # Adjusted for 20 alphabets
           3
           4  # Create a 3x3 matrix to represent the Damage_Plate grid
           5  Damage_Plate_grid = np.zeros((plate_size, plate_size), dtype=int)
           6
           7  # Assign box numbers to each box in the grid
           8  box_number = 1
           9  for i in range(plate_size):
          10      for j in range(plate_size):
          11          Damage_Plate_grid[i, j] = box_number
          12          box_number += 1
          13
          14  # Function to mark alphabets and write box numbers in the center
          15  def mark_alphabets(box_number_input, alphabet, color):
          16      # Find the row and column indices of the box
          17      for i in range(plate_size):
          18          for j in range(plate_size):
          19              if Damage_Plate_grid[i, j] == box_number_input:
          20                  row_index = i
          21                  col_index = j
          22
          23      # Plot the alphabet randomly in the box
          24      rand_row = row_index + np.random.rand()
          25      rand_col = col_index + np.random.rand()
          26      ax.text(rand_col, rand_row, alphabet, fontsize=14, ha='center', va='ce
          27
          28      # Write the box number in the center
          29      ax.text(col_index + 0.5, row_index + 0.5, str(box_number_input), fonts
          30
          31  # Create the Damage_Plate grid plot
          32  fig, ax = plt.subplots(figsize=(6, 6))
          33
          34  # Plot the Damage_Plate grid lines
          35  for i in range(plate_size + 1):
          36      ax.axhline(y=i, color='black', linewidth=2)
          37      ax.axvline(x=i, color='black', linewidth=2)
          38
          39  # Get box numbers and corresponding values from your DataFrame (replace wi
          40  box1_numbers = box_class_df['Actual Box Class'].sample(n=20, random_state=
          41  box2_numbers = box_class_df['Predicted Box Class'].sample(n=20, random_sta
          42  actual_values = [np.random.randint(1, 10) for _ in range(20)]   # Replace w
          43  predicted_values = [np.random.randint(1, 10) for _ in range(20)]   # Replac
          44
          45  alphabets = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', '
          46  color1 = 'red'
          47  color2 = 'blue'
          48
          49  # Mark alphabets and write box numbers
          50  for i in range(20):
          51      mark_alphabets(box1_numbers[i], alphabets[i], color1)
          52      mark_alphabets(box2_numbers[i], alphabets[i], color2)
          53
          54  # Set plot limits and labels
          55  ax.set_xlim(0, plate_size)
```

```
56  ax.set_ylim(0, plate_size)
57  ax.set_xticks(np.arange(0.5, plate_size, 1))
58  ax.set_yticks(np.arange(0.5, plate_size, 1))
59  ax.set_xticklabels([])
60  ax.set_yticklabels([])
61  ax.grid(False)
62
63  # Create text labels outside the plot
64  offset_x = 3.08   # Adjust x-offset for label placement
65  offset_y = 2.85   # Adjust y-offset for label placement
66
67  label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
68  label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l
69
70  # Remove box around the labels (optional)
71  label_text1.set_bbox(dict(facecolor='none', edgecolor='none', pad=0))
72  label_text2.set_bbox(dict(facecolor='none', edgecolor='none', pad=0))
73
74  # Show the plot
75  plt.title('Damage Plate classified as 9 Boxes')
76  plt.show()
```


Damage Plate classified as 9 Boxes
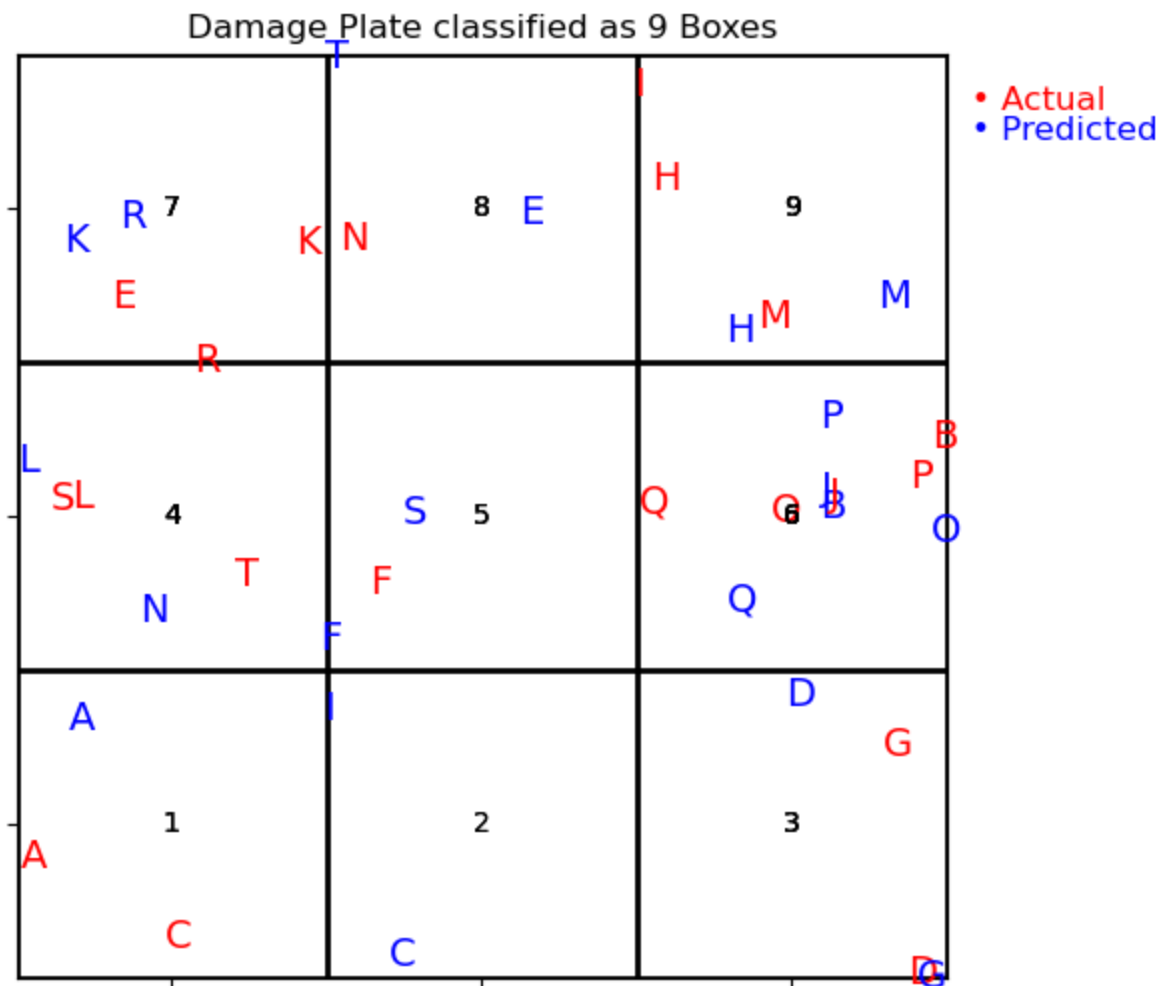
```
1  # Selecting actual and predicted classes for the horizontal class
2  horizontal_class_df = combined_results_df[['Actual Horizontal Class', 'Pre
3
4  random_20_rows_horizontal = horizontal_class_df.sample(n=20)
5
6  print("Random 20 rows from the horizontal_class_df DataFrame:")
7  random_20_rows_horizontal
```

Random 20 rows from the horizontal_class_df DataFrame:

Out[20]:

	Actual Horizontal Class	Predicted Horizontal Class
160	2	1
33	2	2
132	2	2
4	3	3
234	3	3
2	3	3
211	3	3
109	1	1
100	2	2
55	1	1
65	3	3
47	3	3
119	1	1
246	5	5
216	3	1
107	3	3
185	3	3
112	3	3
25	2	2
6	5	5

```python
In [21]:    1   # Extract the actual and predicted horizontal class from the DataFrame
            2   actual_classes = random_20_rows_horizontal['Actual Horizontal Class'].toli
            3   predicted_classes = random_20_rows_horizontal['Predicted Horizontal Class'
            4
            5   # Define the layout of the plate and strip labels
            6   strip_labels = ['5', '4', '3', '2', '1']  # Reverse order to label from bo
            7
            8   # Create a new figure
            9   fig, ax = plt.subplots(figsize=(8, 6))
           10
           11   # Plot horizontal strips and assign numbers
           12   for i, label in enumerate(strip_labels):
           13       strip_y = (4 - i) / 5  # Adjusting the y-coordinate to represent botto
           14       rect_strip = plt.Rectangle((0, strip_y), 1, 1 / 5, fill=False, edgecol
           15       ax.add_patch(rect_strip)
           16       # Add strip label at the left corner of each strip
           17       ax.text(0.02, strip_y + 0.02, label, ha='center', va='center', fontsiz
           18
           19   # Set axis limits and labels
           20   ax.set_xlim(0, 1)
           21   ax.set_ylim(0, 1)
           22   ax.set_xticks([])
           23   ax.set_yticks([])
           24   ax.set_aspect('equal')
           25
           26   # Add title
           27   plt.title('Damage Plate classify as 5 Horizontal Strips')
           28
           29   # Initialize lists to store input data for red and blue alphabets
           30   red_alphabets = []
           31   blue_alphabets = []
           32
           33   # Take 5 inputs for red alphabets
           34   for i, strip_num in enumerate(actual_classes):
           35       letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
           36       red_alphabets.append((letter, strip_num))
           37
           38   # Take 5 inputs for blue alphabets
           39   for i, strip_num in enumerate(predicted_classes):
           40       letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
           41       blue_alphabets.append((letter, strip_num))
           42
           43   # Function to mark alphabets on the plate
           44   def mark_alphabets(alphabets, color):
           45       for letter, strip_num in alphabets:
           46           if 1 <= strip_num <= 5:
           47               strip_y = (strip_num - 1) / 5  # y-coordinate of the selected
           48               # Generate random x and y coordinates inside the selected stri
           49               random_x_coordinate = np.random.uniform(0.1, 0.9)  # Adjust th
           50               random_y_coordinate = np.random.uniform(strip_y, strip_y + 0.2
           51               # Marking the alphabet with specified color
           52               ax.text(random_x_coordinate, random_y_coordinate, letter, ha='
           53
           54   # Create text labels outside the plot
           55   offset_x = 1.08  # Adjust x-offset for label placement
```

```
56  offset_y = 0.9  # Adjust y-offset for label placement
57
58  label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
59  label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l
60
61  # Mark red alphabets
62  mark_alphabets(red_alphabets, 'red')
63
64  # Mark blue alphabets
65  mark_alphabets(blue_alphabets, 'blue')
66
67  # Show the plot
68  plt.show()
```


Damage Plate classify as 5 Horizontal Strips
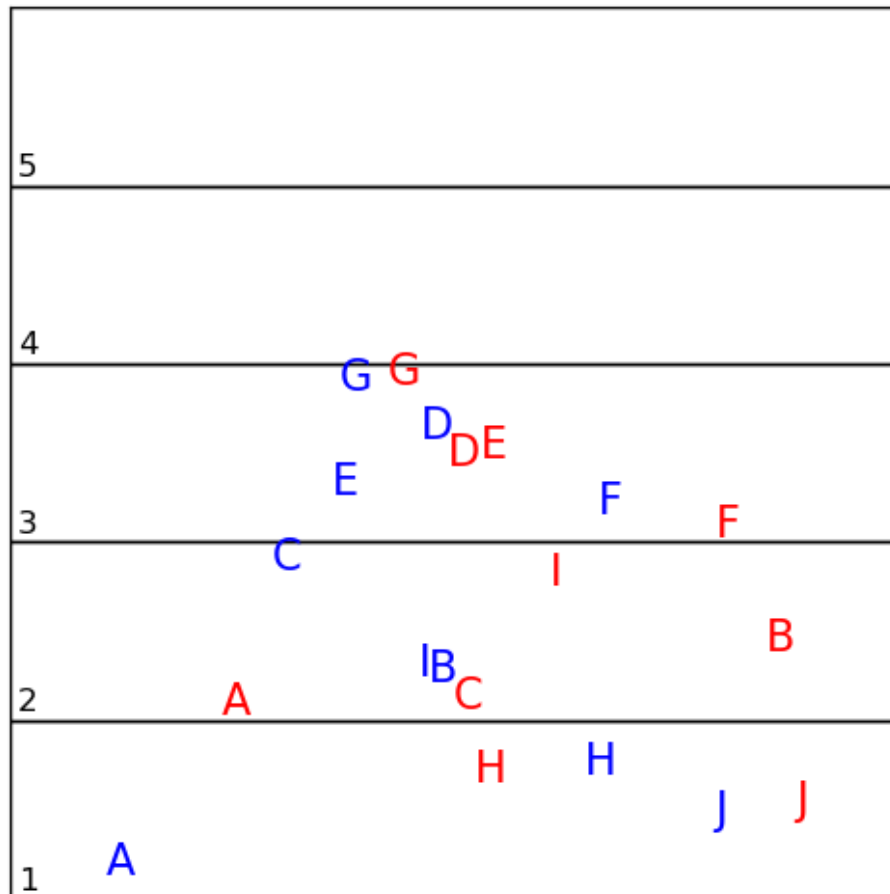
```
1  # Selecting actual and predicted classes for the vertical class
2  vertical_class_df = combined_results_df[['Actual Vertical Class', 'Predict
3
4  # Select 20 random rows for demonstration
5  random_20_rows_vertical = vertical_class_df.sample(n=20)
6
7  print("Random 20 rows from the vertical_class_df DataFrame:")
8  random_20_rows_vertical
```

Random 20 rows from the vertical_class_df DataFrame:

	Actual Vertical Class	Predicted Vertical Class
182	2	2
43	1	5
87	5	5
248	1	1
252	5	3
50	4	4
184	3	3
164	5	5
244	4	5
235	3	2
90	1	2
168	2	2
119	3	3
99	2	2
130	2	1
107	3	3
108	1	1
15	4	4
189	2	2
226	3	3

```python
In [23]:   1  # Extract the actual and predicted vertical class from the DataFrame
           2  actual_classes = random_20_rows_vertical['Actual Vertical Class'].tolist()
           3  predicted_classes = random_20_rows_vertical['Predicted Vertical Class'].to
           4
           5  # Define the layout of the plate and strip labels
           6  strip_labels = ['1', '2', '3', '4', '5']  # Label from bottom to top for v
           7
           8  # Create a new figure
           9  fig, ax = plt.subplots(figsize=(8, 6))
          10
          11  # Plot vertical strips and assign numbers
          12  for i, label in enumerate(strip_labels):
          13      strip_x = i / 5  # Adjusting the x-coordinate to represent leftmost st
          14      rect_strip = plt.Rectangle((strip_x, 0), 1 / 5, 1, fill=False, edgecol
          15      ax.add_patch(rect_strip)
          16      # Add strip label at the bottom corner of each strip
          17      ax.text(strip_x + 0.02, 0.02, label, ha='center', va='center', fontsiz
          18
          19  # Set axis limits and labels
          20  ax.set_xlim(0, 1)
          21  ax.set_ylim(0, 1)
          22  ax.set_xticks([])
          23  ax.set_yticks([])
          24  ax.set_aspect('equal')
          25
          26  # Add title
          27  plt.title('Damage Plate classify as 5 Vertical Strips')
          28
          29  # Initialize lists to store input data for red and blue alphabets
          30  red_alphabets = []
          31  blue_alphabets = []
          32
          33  # Take 5 inputs for red alphabets
          34  for i, strip_num in enumerate(actual_classes):
          35      letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
          36      red_alphabets.append((letter, strip_num))
          37
          38  # Take 5 inputs for blue alphabets
          39  for i, strip_num in enumerate(predicted_classes):
          40      letter = chr(ord('A') + i)  # Convert index to corresponding alphabet
          41      blue_alphabets.append((letter, strip_num))
          42
          43  # Function to mark alphabets on the plate
          44  def mark_alphabets(alphabets, color):
          45      for letter, strip_num in alphabets:
          46          if 1 <= strip_num <= 5:
          47              strip_x = (strip_num - 1) / 5  # x-coordinate of the selected
          48              # Generate random x and y coordinates inside the selected stri
          49              random_x_coordinate = np.random.uniform(strip_x, strip_x + 0.2
          50              random_y_coordinate = np.random.uniform(0.1, 0.9)  # Adjust th
          51              # Marking the alphabet with specified color
          52              ax.text(random_x_coordinate, random_y_coordinate, letter, ha='
          53
          54  # Create text labels outside the plot
          55  offset_x = 1.08  # Adjust x-offset for label placement
```

```
56  offset_y = .95   # Adjust y-offset for label placement
57
58  label_text1 = plt.text(offset_x, offset_y, "\u2022 Actual", ha='left', va=
59  label_text2 = plt.text(offset_x, offset_y - 0.1, "\u2022 Predicted", ha='l
60
61  # Mark red alphabets
62  mark_alphabets(red_alphabets, 'red')
63
64  # Mark blue alphabets
65  mark_alphabets(blue_alphabets, 'blue')
66
67  # Show the plot
68  plt.show()
```


Damage Plate classify as 5 Vertical Strips