

# **Machine Learning**

## **Lab A3**

**ASIM KUMAR HANSDA**

**ROLL NO - 002211001136**

**ASSIGNMENT - 2**

---

**Github Link:**

<https://github.com/cryptasim/MACHINE-LEARNING-LAB>



Open in Colab

# Libraries

```
In [ ]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

```
In [ ]: !pip install ucimlrepo  
  
Collecting ucimlrepo  
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)  
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from ucimlrepo) (2.2.2)  
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.12/dist-packages (from ucimlrepo) (2025.8.3)  
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)  
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)  
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)  
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)  
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)  
  Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)  
Installing collected packages: ucimlrepo  
Successfully installed ucimlrepo-0.0.7
```

# Wine Dataset

## Prepare Dataset

```
In [ ]: from ucimlrepo import fetch_ucirepo  
  
wine = fetch_ucirepo(id=109)
```

```
In [ ]: X = wine.data.features  
y = wine.data.targets
```

```
In [ ]: X
```

Out[ ]:

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavonoids
<b>0</b>	14.23	1.71	2.43		15.6	127	2.80
<b>1</b>	13.20	1.78	2.14		11.2	100	2.65
<b>2</b>	13.16	2.36	2.67		18.6	101	2.80
<b>3</b>	14.37	1.95	2.50		16.8	113	3.85
<b>4</b>	13.24	2.59	2.87		21.0	118	2.80
...	...	...	...		...	...	...
<b>173</b>	13.71	5.65	2.45		20.5	95	1.68
<b>174</b>	13.40	3.91	2.48		23.0	102	1.80
<b>175</b>	13.27	4.28	2.26		20.0	120	1.59
<b>176</b>	13.17	2.59	2.37		20.0	120	1.65
<b>177</b>	14.13	4.10	2.74		24.5	96	2.05

178 rows × 13 columns

In [ ]:

Out[ ]:

	class
<b>0</b>	1
<b>1</b>	1
<b>2</b>	1
<b>3</b>	1
<b>4</b>	1
...	...
<b>173</b>	3
<b>174</b>	3
<b>175</b>	3
<b>176</b>	3
<b>177</b>	3

178 rows × 1 columns

# Principal Component Analysis (PCA)

```
In [ ]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standardize the data before applying PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

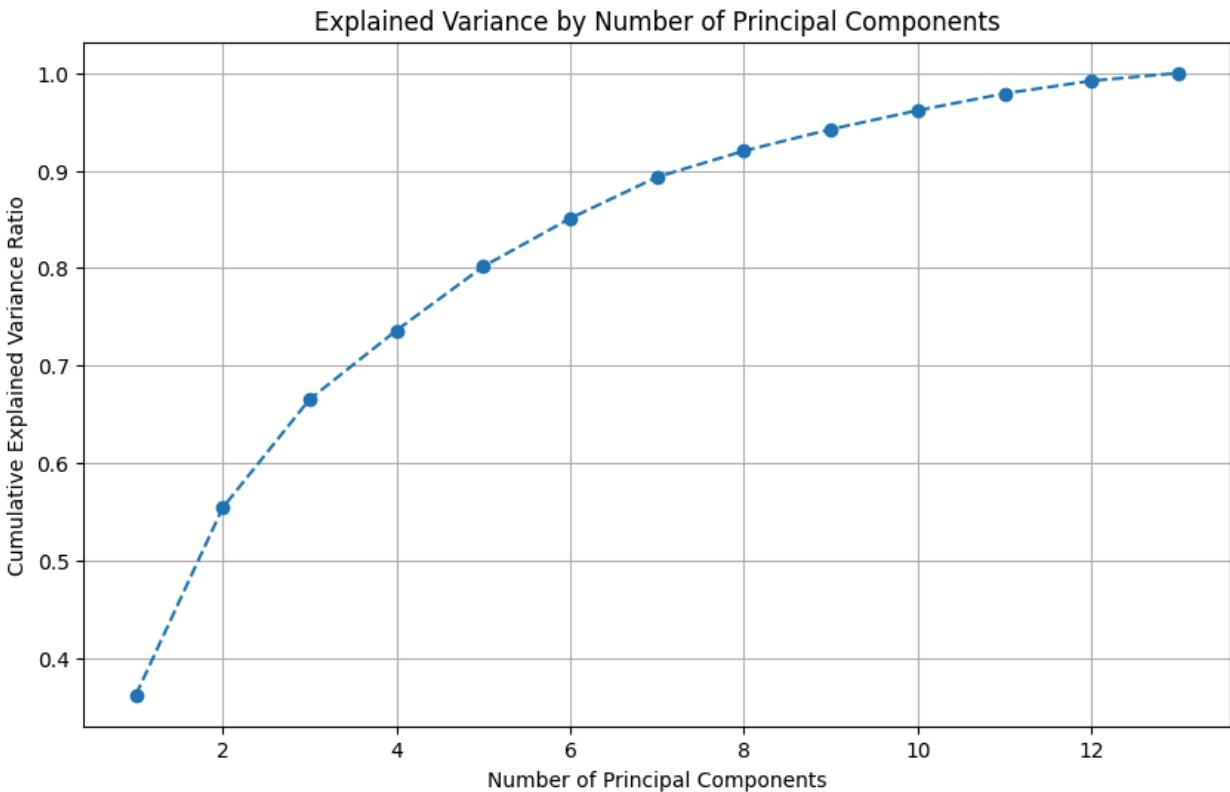
# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained variance ratio by each component:", explained_variance_ratio)

# Cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
print("Cumulative explained variance:", cumulative_explained_variance)

# Plot explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), cumulative_explained_var
plt.title('Explained Variance by Number of Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```

```
Explained variance ratio by each component: [0.36198848 0.1920749  0.11123631
0.0706903  0.06563294 0.04935823
0.04238679 0.02680749 0.02222153 0.01930019 0.01736836 0.01298233
0.00795215]
Cumulative explained variance: [0.36198848 0.55406338 0.66529969 0.73598999 0.8
0162293 0.85098116
0.89336795 0.92017544 0.94239698 0.96169717 0.97906553 0.99204785
1.        ]
```



Based on the cumulative explained variance plot, you can decide how many principal components to retain to capture a desired amount of variance. For example, to retain 95% of the variance, you would choose the number of components where the cumulative explained variance is close to 0.95.

Let's apply PCA with a chosen number of components (e.g., based on the plot).

```
In [ ]: # Choose the number of components (e.g., to retain 95% variance)
# Based on the plot, let's assume we choose 10 components to retain most of the variance
n_components = 10 # You can change this based on your analysis of the plot

pca = PCA(n_components=n_components)
X_pca_reduced = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced.shape[1]}")
print("\nData after PCA:")
display(pd.DataFrame(X_pca_reduced).head())
```

Original number of features: 13  
 Reduced number of features after PCA: 10

Data after PCA:

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	
<b>0</b>	3.316751	1.443463	-0.165739	-0.215631	0.693043	0.223880	0.596427	-0.065
<b>1</b>	2.209465	-0.333393	-2.026457	-0.291358	-0.257655	0.927120	0.053776	-1.024
<b>2</b>	2.516740	1.031151	0.982819	0.724902	-0.251033	-0.549276	0.424205	0.344
<b>3</b>	3.757066	2.756372	-0.176192	0.567983	-0.311842	-0.114431	-0.383337	-0.643
<b>4</b>	1.008908	0.869831	2.026688	-0.409766	0.298458	0.406520	0.444074	-0.416

## SVM

### Different SVM models

```
In [ ]: from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]
kernels = {
    'linear': 'linear',
    'polynomial': 'poly',
    'gaussian': 'rbf',
    'sigmoid': 'sigmoid'
}
```

### Calculate values for different test size

```
In [ ]: results = []
confusion_matrices = []
trained_models = [] # Add a list to store trained models

for kernel_name, kernel in kernels.items():
    for size in training_sizes:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)
        svc = SVC(kernel=kernel, probability=True, random_state=42) # Add probability=True
        svc.fit(X_train, y_train.values.ravel())
        y_pred = svc.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
        recall = recall_score(y_test, y_pred, average='weighted') # Calculate weighted recall
        f1 = f1_score(y_test, y_pred, average='weighted') # Calculate weighted F1-score
```

```

cm = confusion_matrix(y_test, y_pred) # Calculate confusion matrix

results.append({
    "Training size":int (size*100),
    "Accuracy":acc,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1,
    "Kernel":kernel_name
})
confusion_matrices.append({ # Store confusion matrix with metadata
    "Training size":int (size*100),
    "Kernel":kernel_name,
    "Confusion Matrix":cm
})
trained_models.append({ # Store the trained model with metadata
    "Training size":int (size*100),
    "Kernel":kernel_name,
    "Model": svc,
    "X_test": X_test,
    "y_test": y_test
})

```

## Print Evaluation Table and Graphs

```

In [ ]: df = pd.DataFrame(results)
display(df) # Display the DataFrame as a table

# Plot Accuracy
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy', hue='Kernel')
plt.title('SVM Accuracy by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

# Plot Precision
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision', hue='Kernel')
plt.title('SVM Precision by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

# Plot Recall
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall', hue='Kernel')
plt.title('SVM Recall by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

```

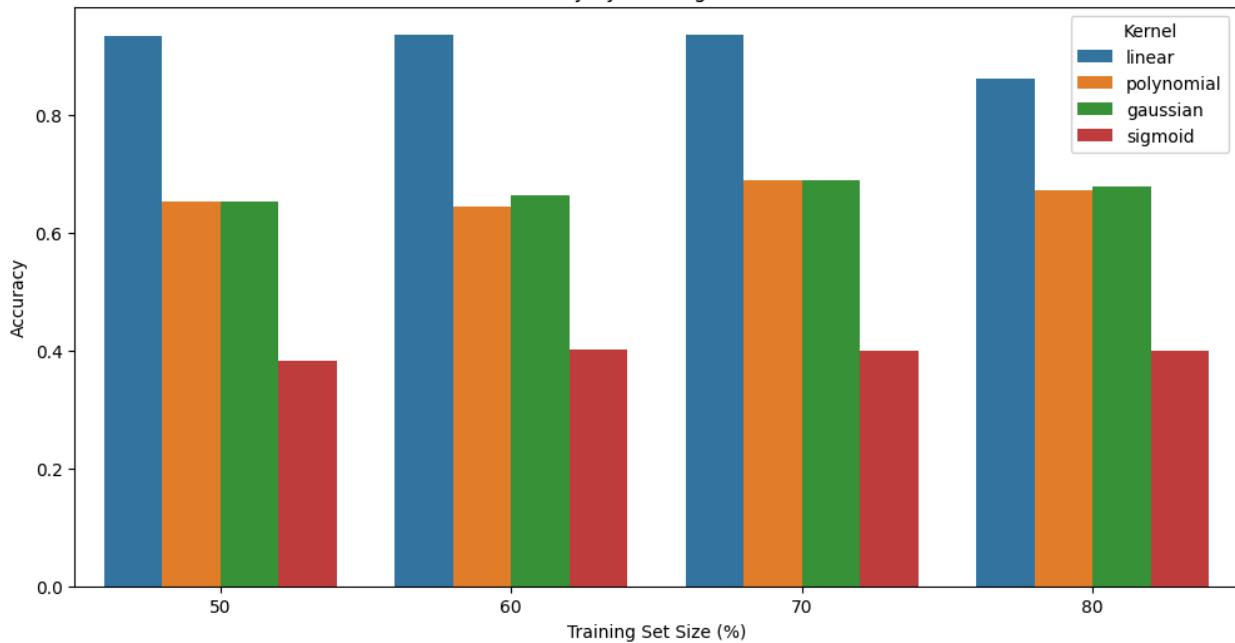
```

# Plot F1-score
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score', hue='Kernel')
plt.title('SVM F1-score by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()

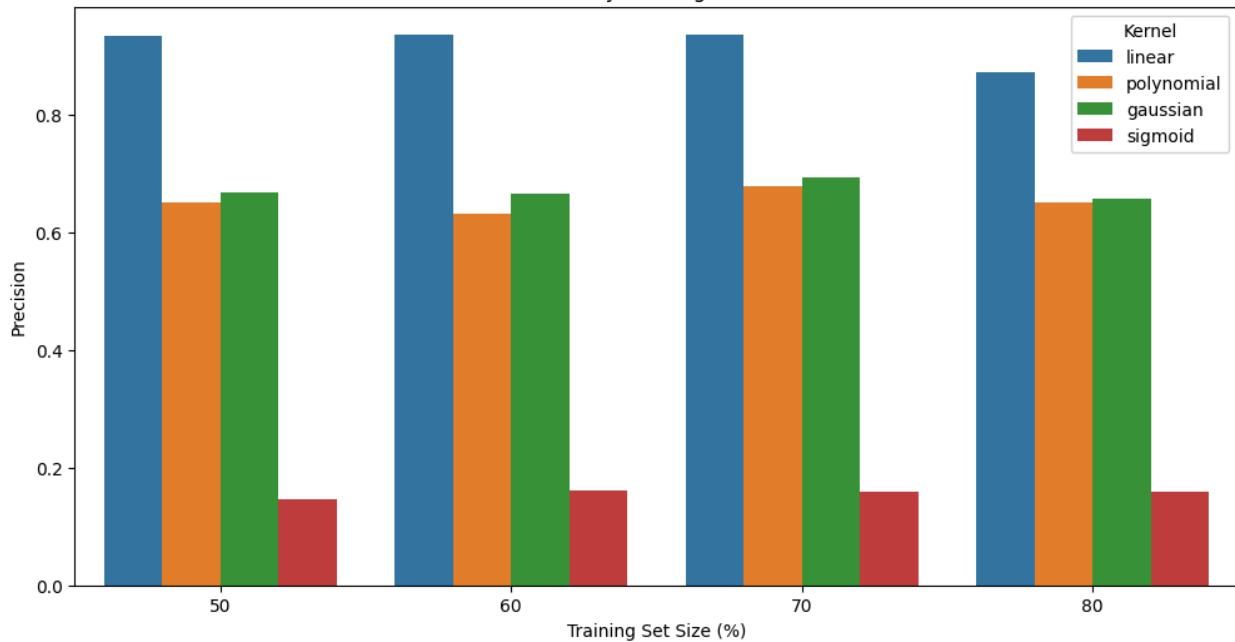
```

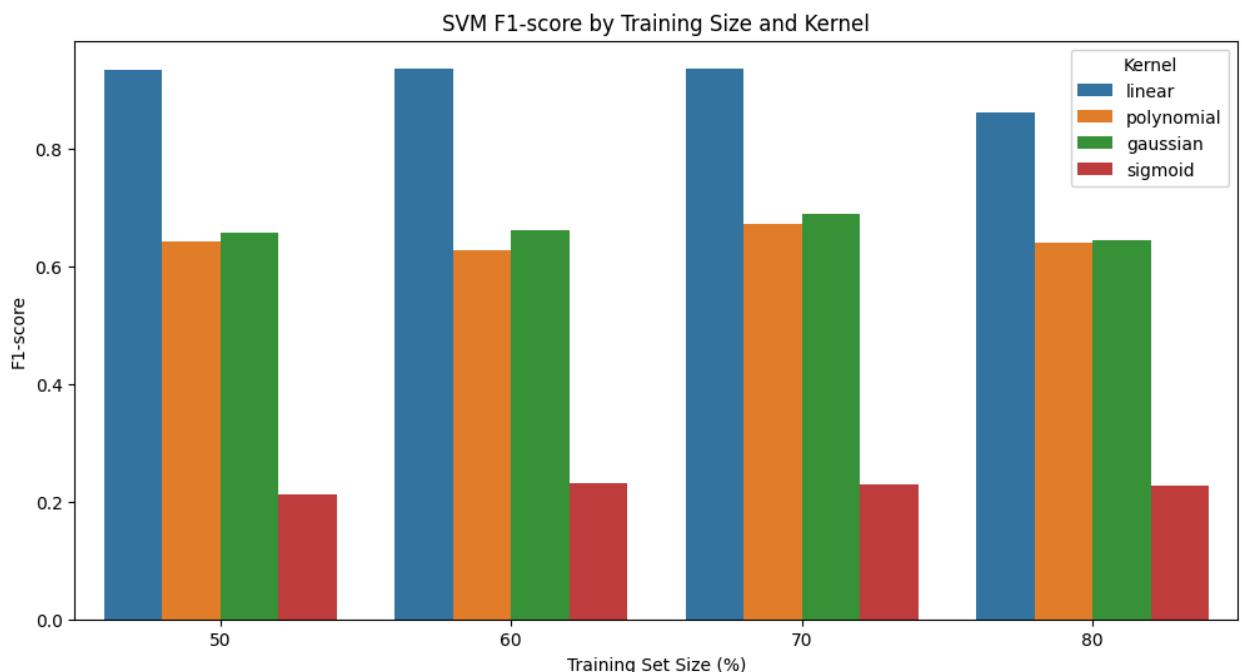
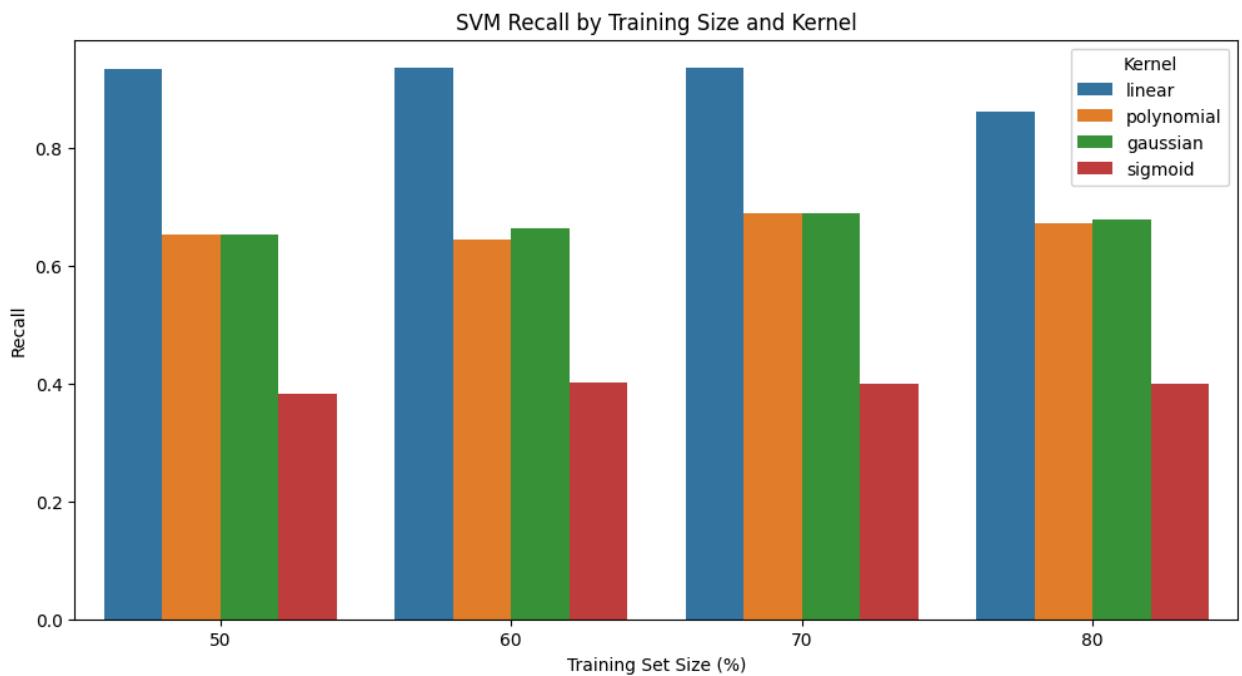
	Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.932584	0.933748	0.932584	0.932554	linear
1	60	0.934579	0.934670	0.934579	0.934515	linear
2	70	0.936000	0.936033	0.936000	0.935762	linear
3	80	0.860140	0.872143	0.860140	0.859931	linear
4	50	0.651685	0.651387	0.651685	0.640804	polynomial
5	60	0.644860	0.631917	0.644860	0.626031	polynomial
6	70	0.688000	0.676990	0.688000	0.671169	polynomial
7	80	0.671329	0.649877	0.671329	0.639471	polynomial
8	50	0.651685	0.667877	0.651685	0.657626	gaussian
9	60	0.663551	0.665705	0.663551	0.661062	gaussian
10	70	0.688000	0.693446	0.688000	0.687787	gaussian
11	80	0.678322	0.656929	0.678322	0.644090	gaussian
12	50	0.382022	0.145941	0.382022	0.211199	sigmoid
13	60	0.401869	0.161499	0.401869	0.230405	sigmoid
14	70	0.400000	0.160000	0.400000	0.228571	sigmoid
15	80	0.398601	0.158883	0.398601	0.227203	sigmoid

SVM Accuracy by Training Size and Kernel



SVM Precision by Training Size and Kernel





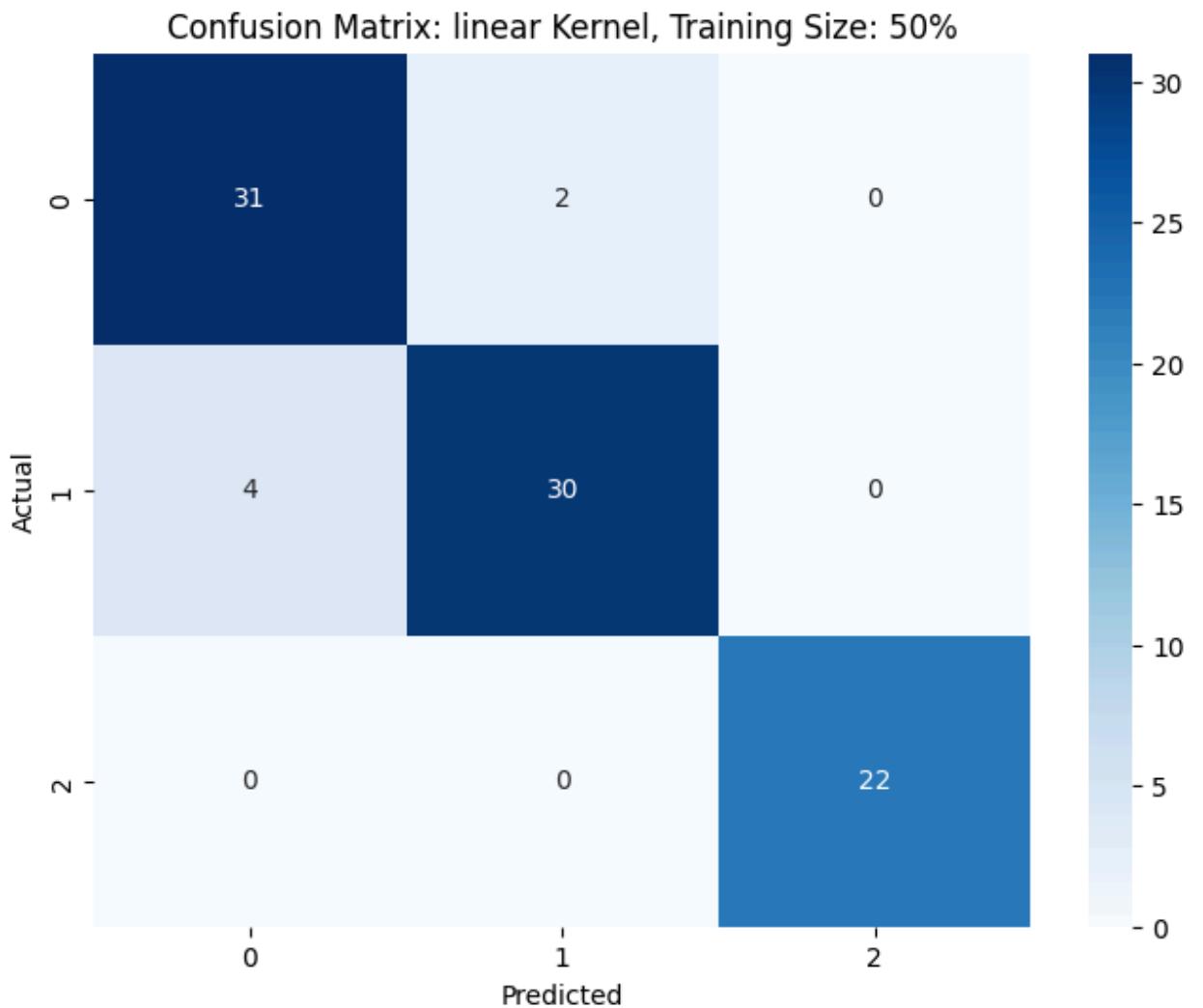
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

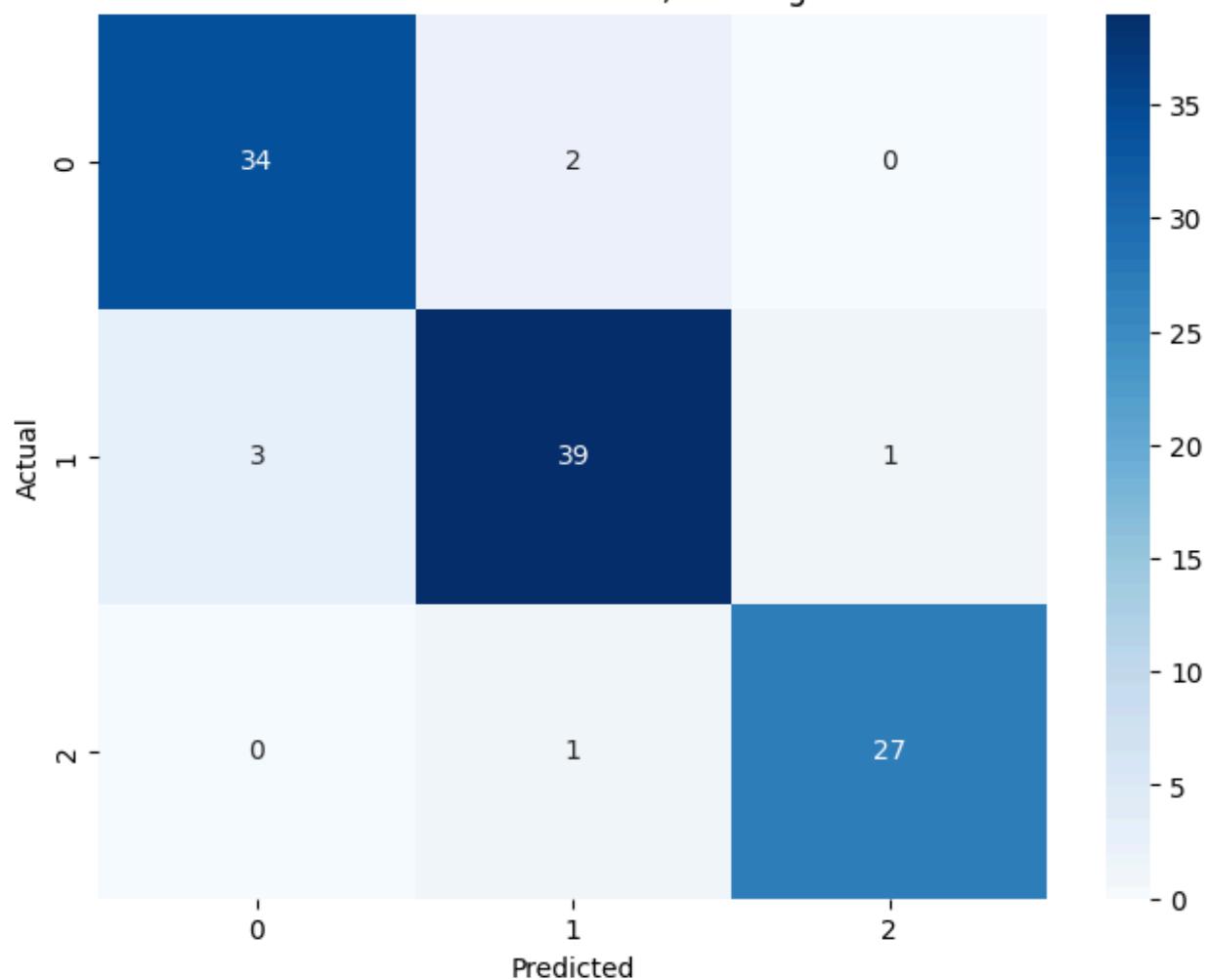
for cm_data in confusion_matrices:
    cm = cm_data["Confusion Matrix"]
    kernel_name = cm_data["Kernel"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
```

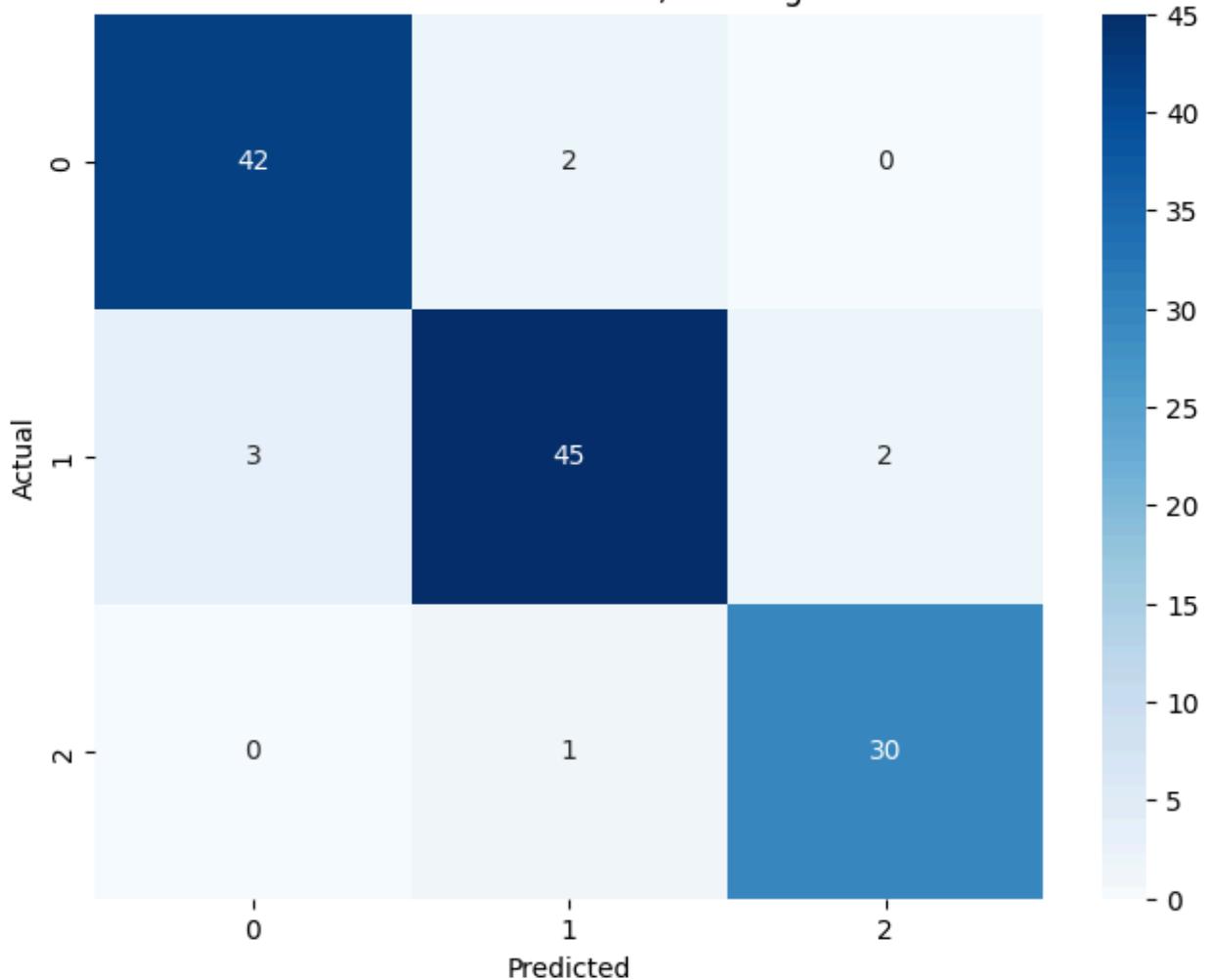
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix: {kernel_name} Kernel, Training Size: {train_size}%')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



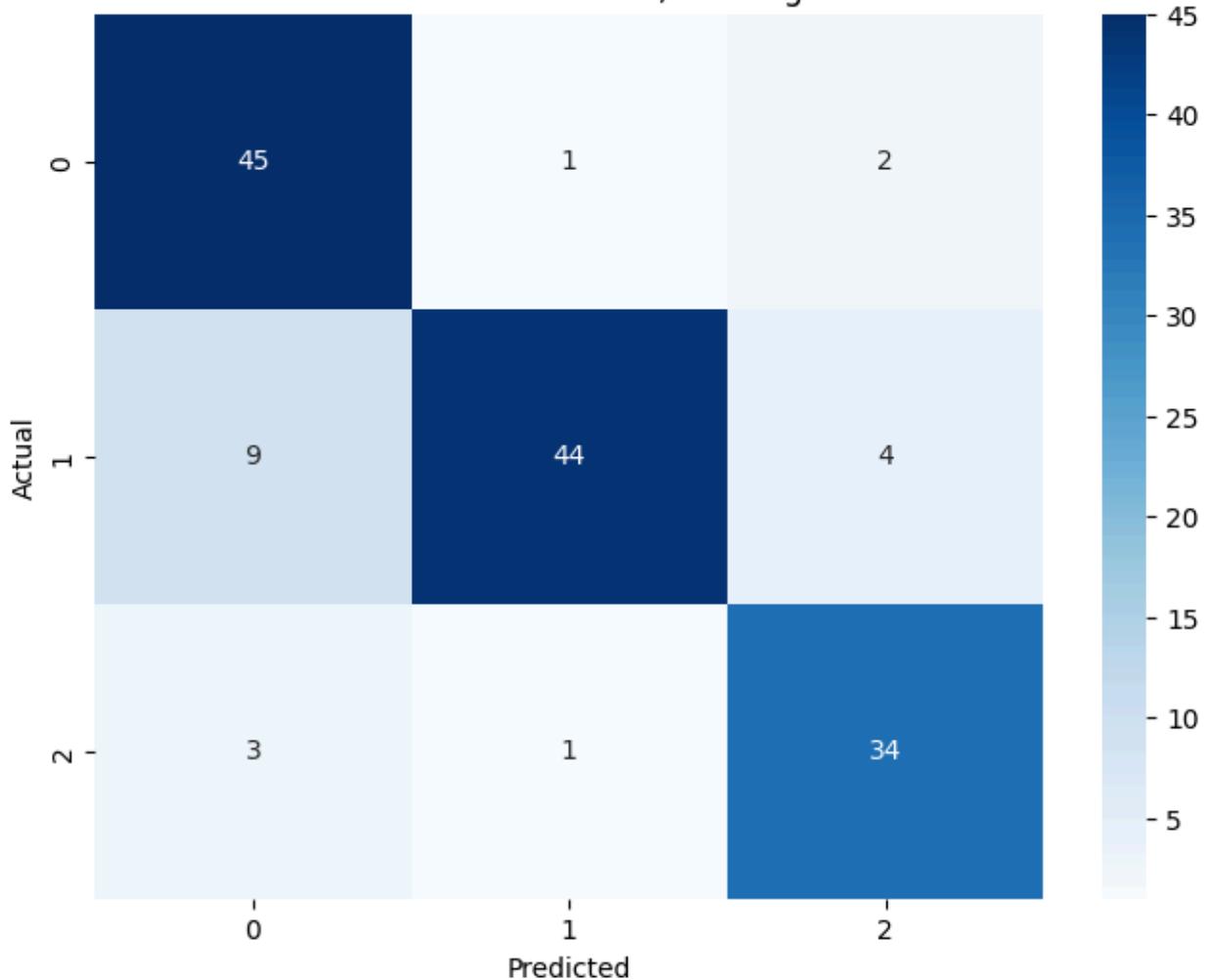
Confusion Matrix: linear Kernel, Training Size: 60%



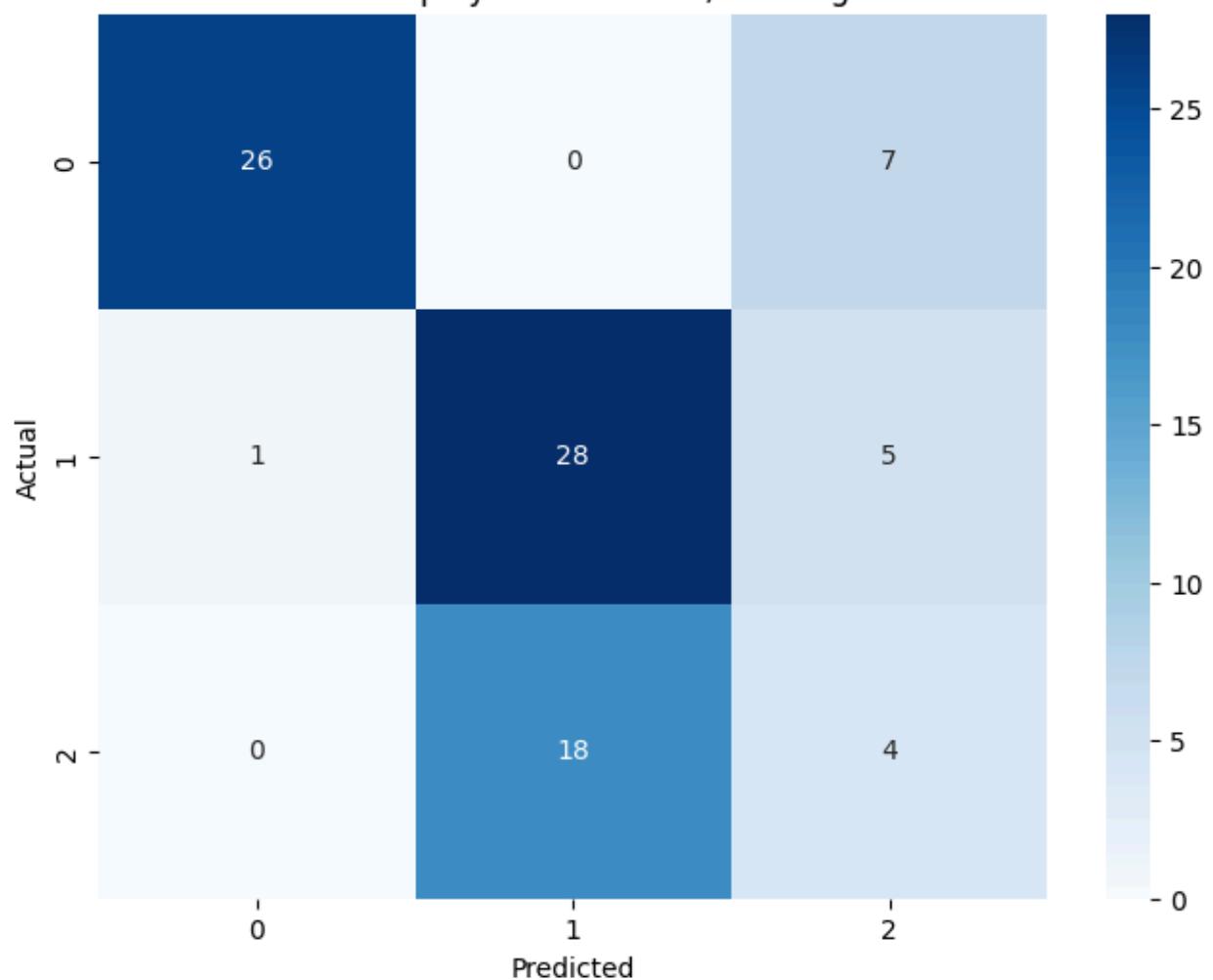
Confusion Matrix: linear Kernel, Training Size: 70%



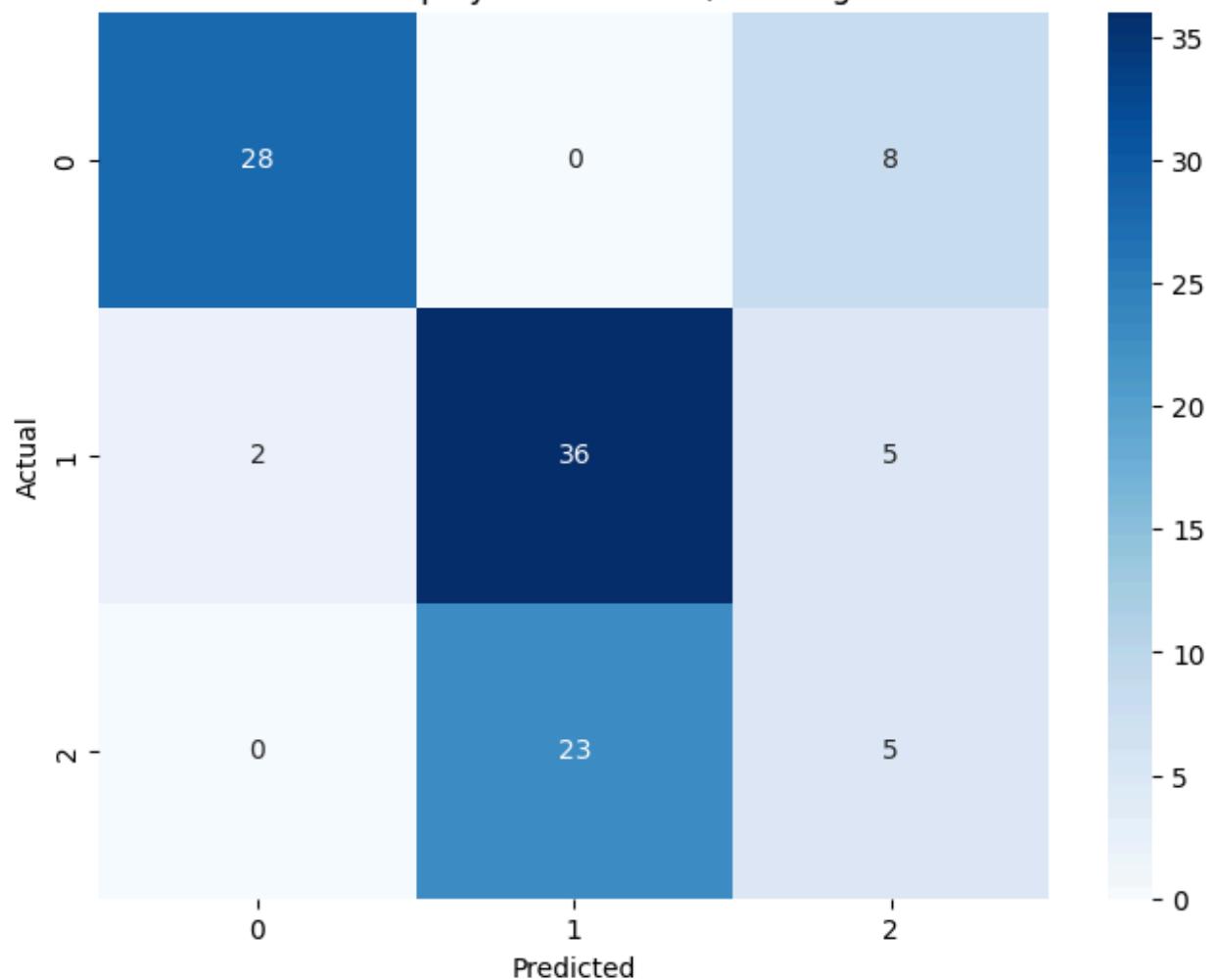
Confusion Matrix: linear Kernel, Training Size: 80%



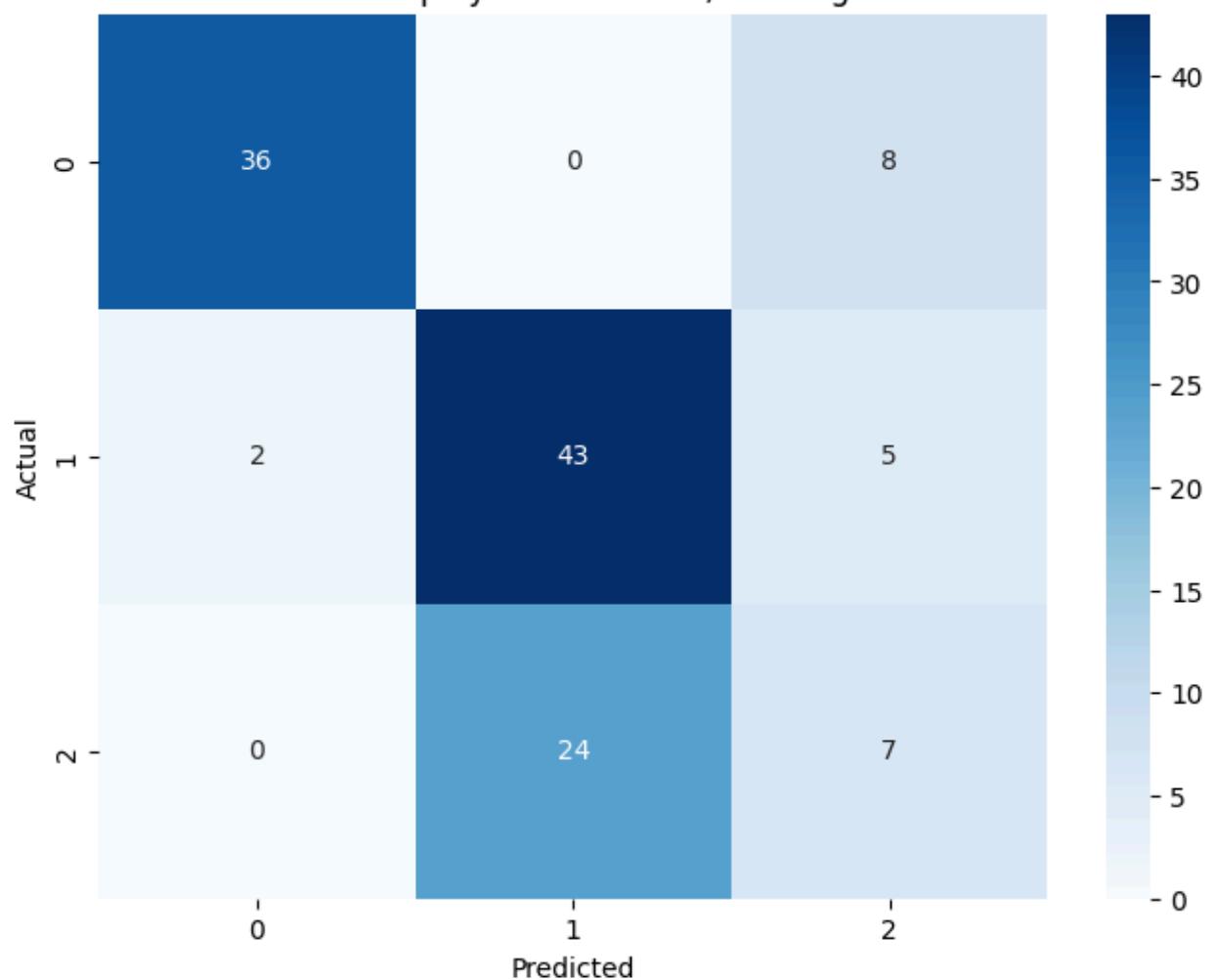
Confusion Matrix: polynomial Kernel, Training Size: 50%



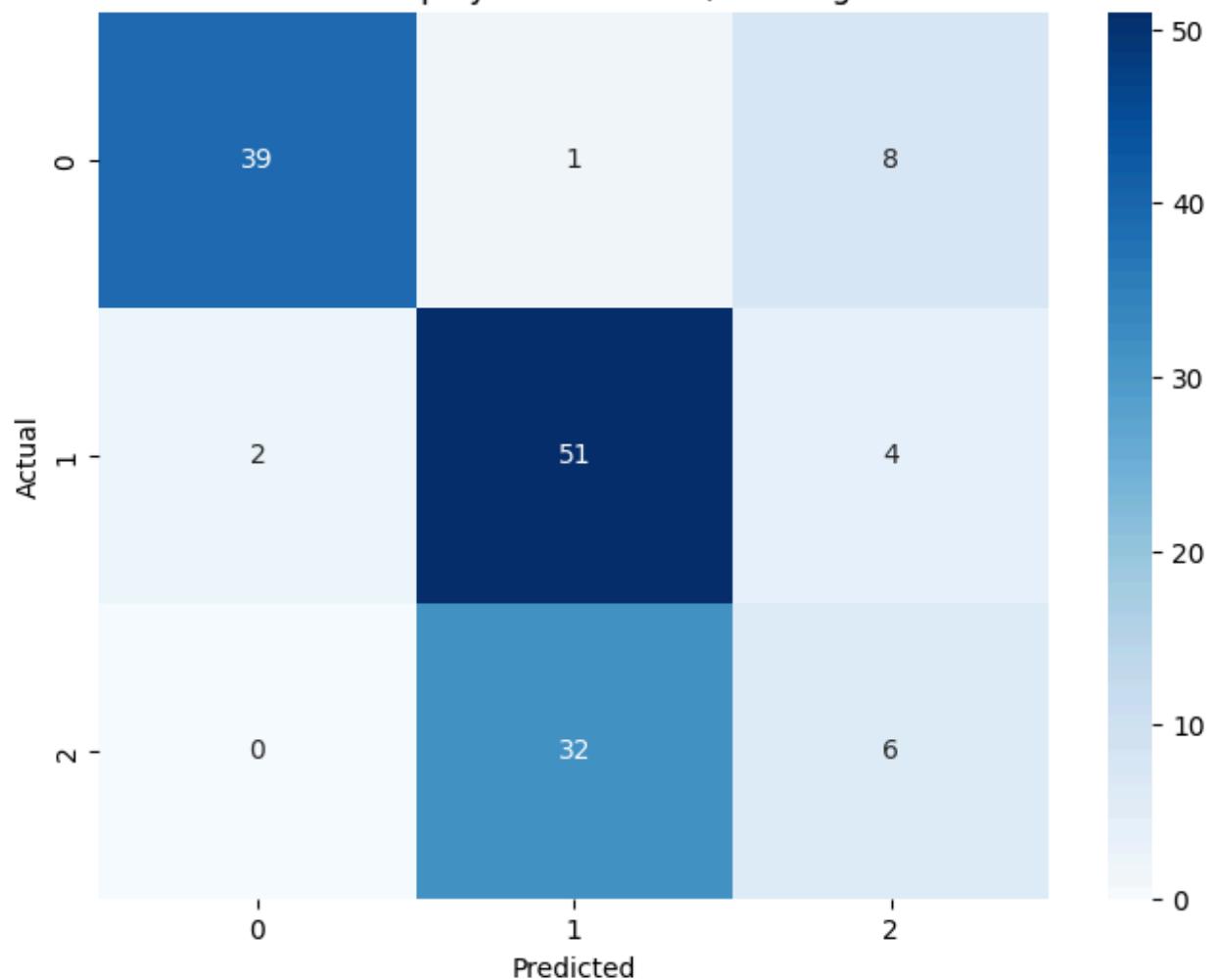
Confusion Matrix: polynomial Kernel, Training Size: 60%



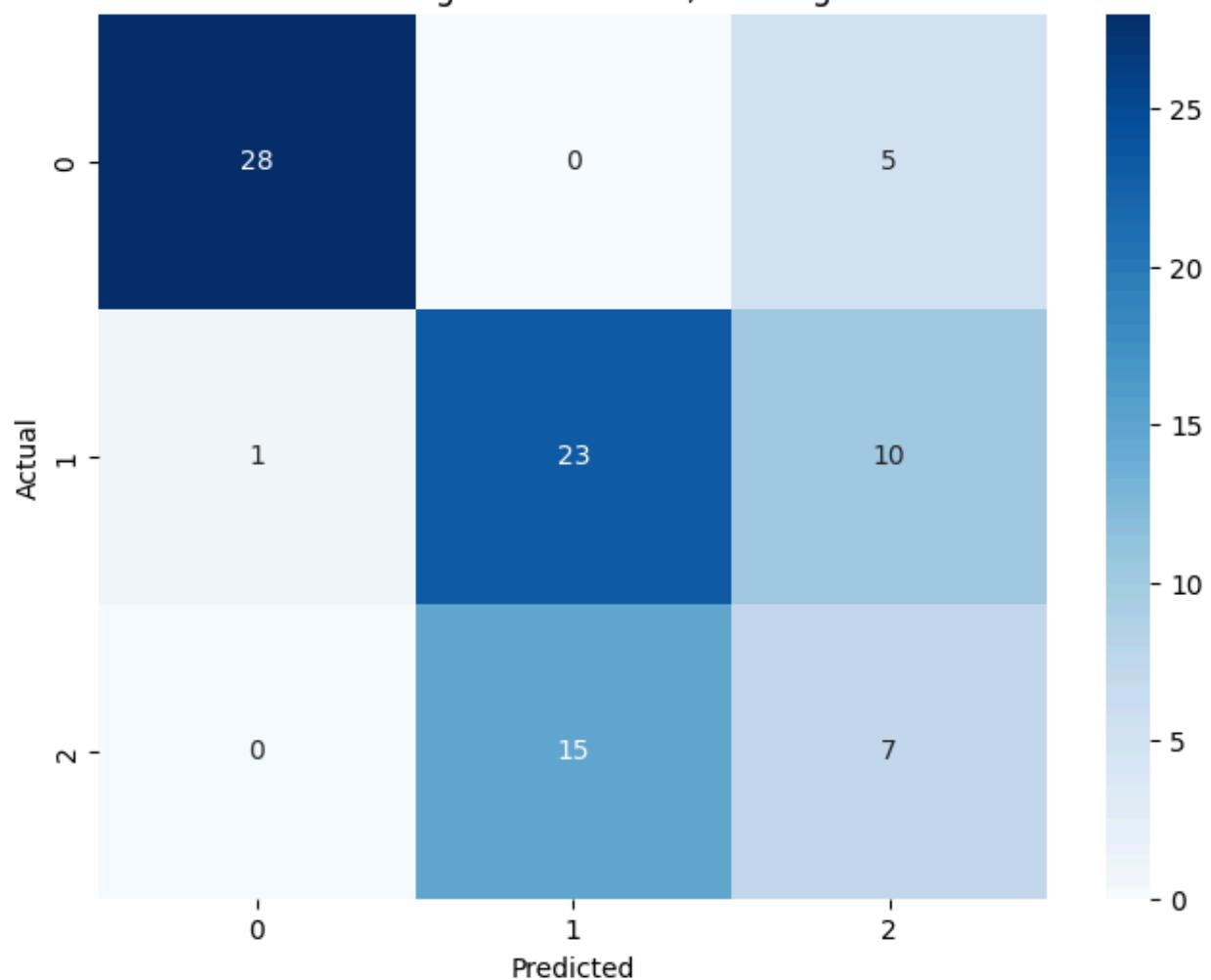
Confusion Matrix: polynomial Kernel, Training Size: 70%



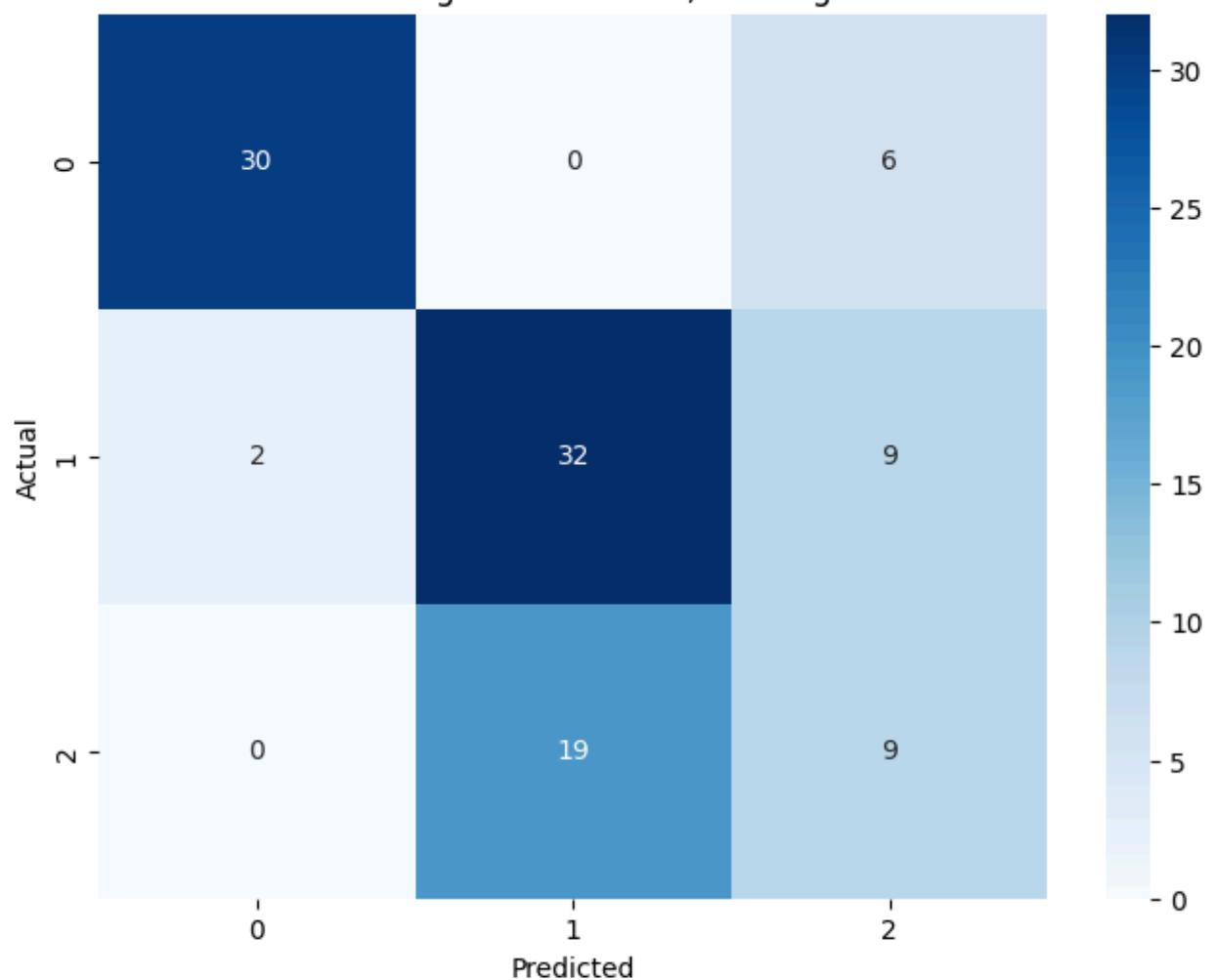
Confusion Matrix: polynomial Kernel, Training Size: 80%



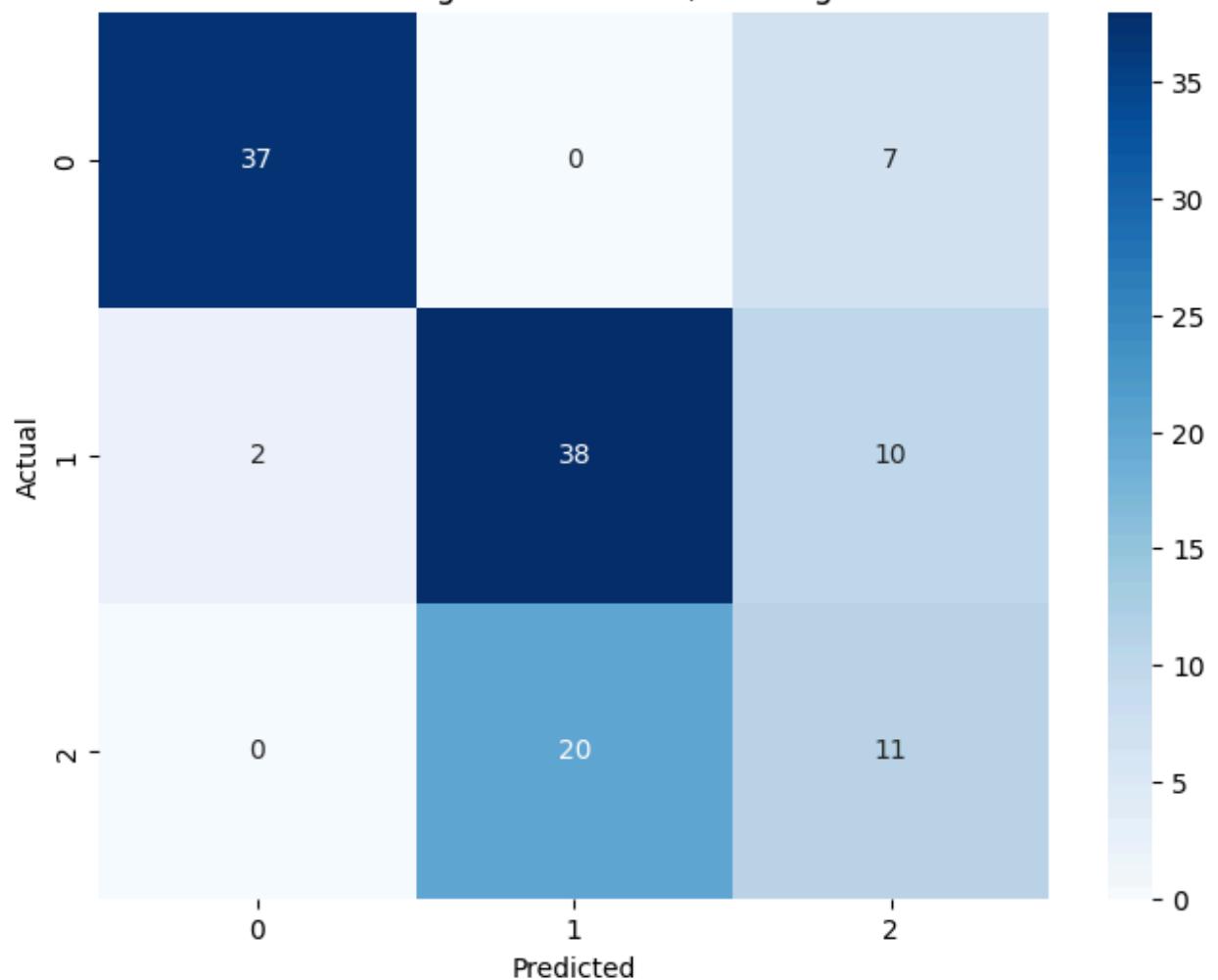
Confusion Matrix: gaussian Kernel, Training Size: 50%



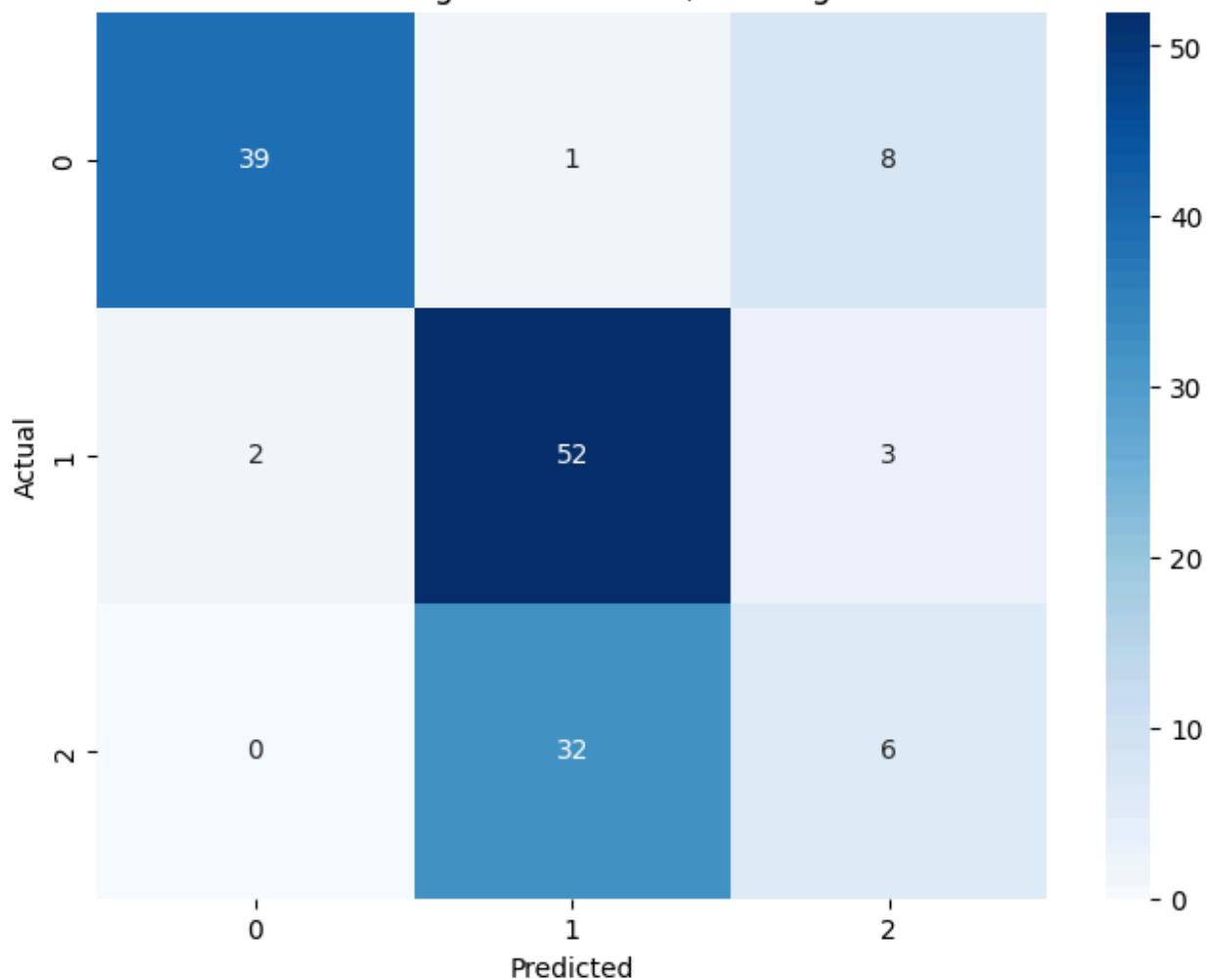
Confusion Matrix: gaussian Kernel, Training Size: 60%



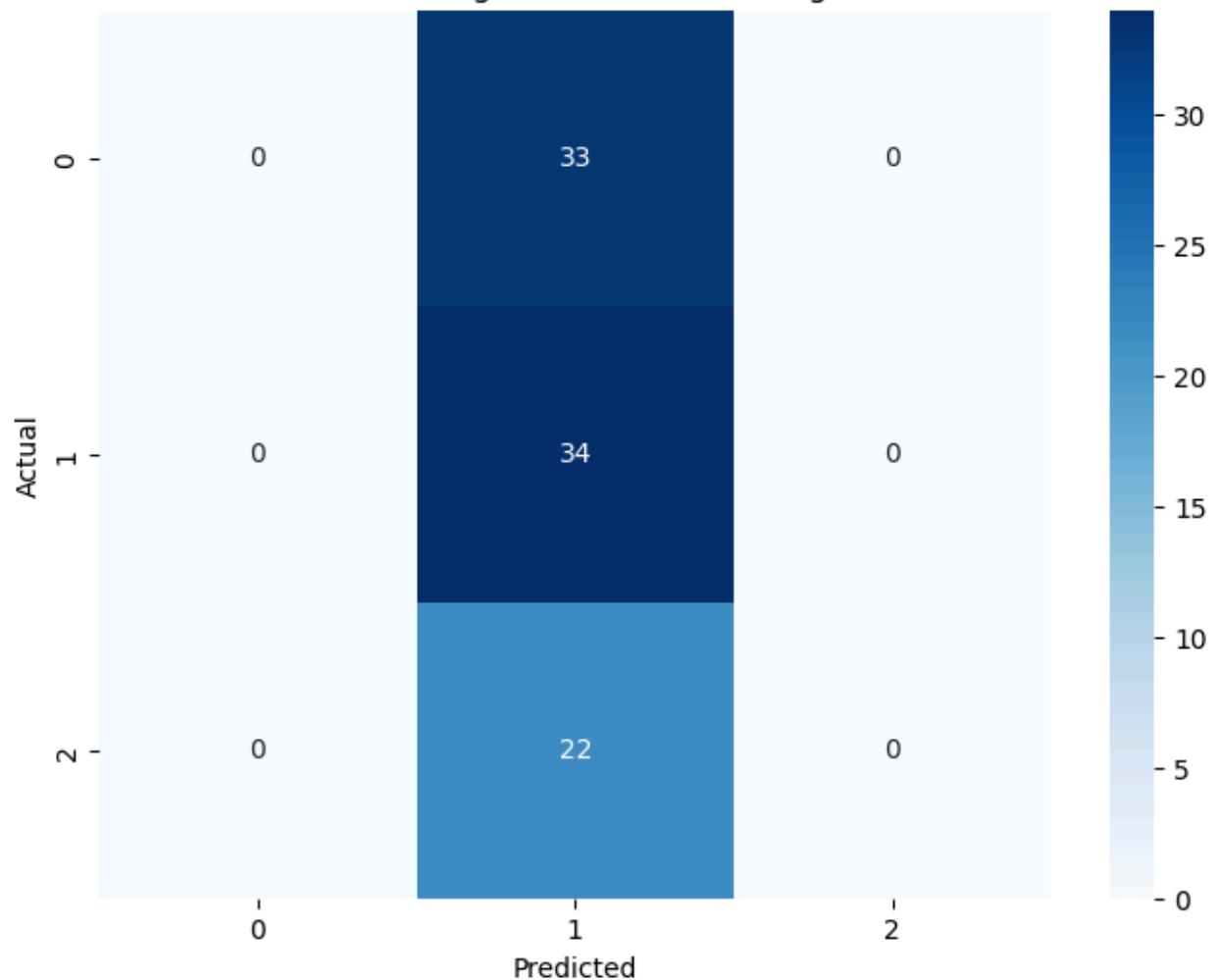
Confusion Matrix: gaussian Kernel, Training Size: 70%



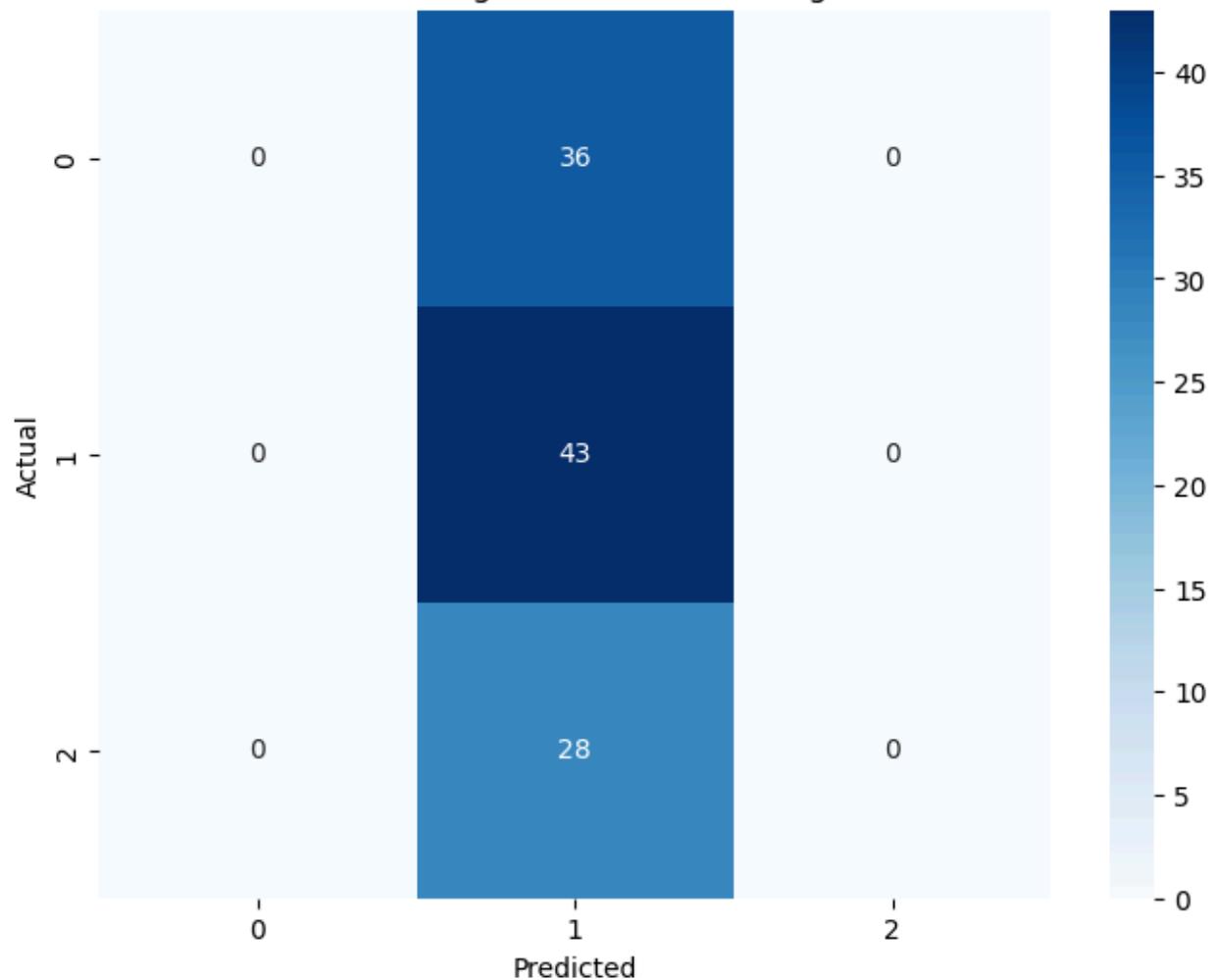
Confusion Matrix: gaussian Kernel, Training Size: 80%



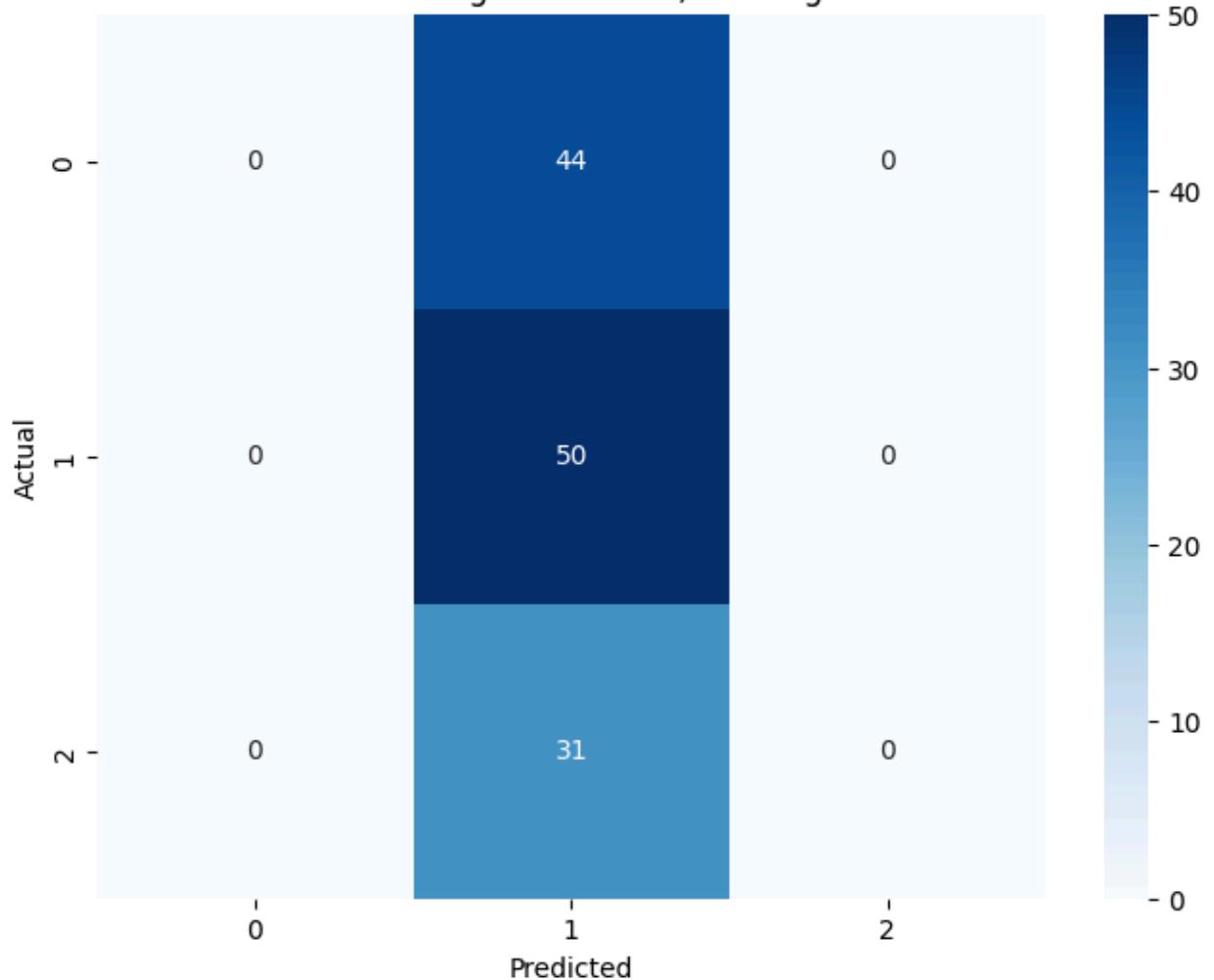
Confusion Matrix: sigmoid Kernel, Training Size: 50%

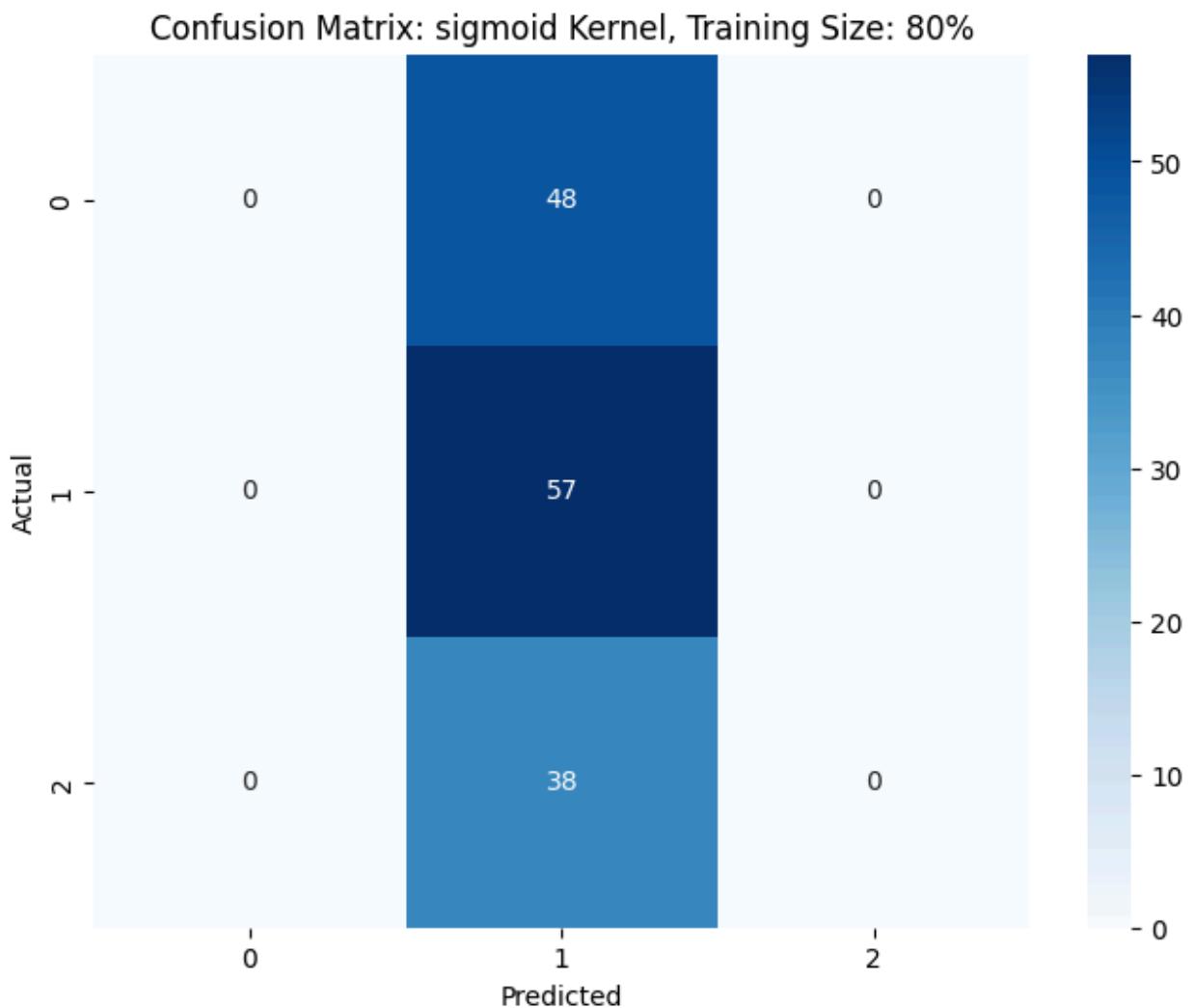


Confusion Matrix: sigmoid Kernel, Training Size: 60%



Confusion Matrix: sigmoid Kernel, Training Size: 70%





## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    kernel_name = model_data["Kernel"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
```

```

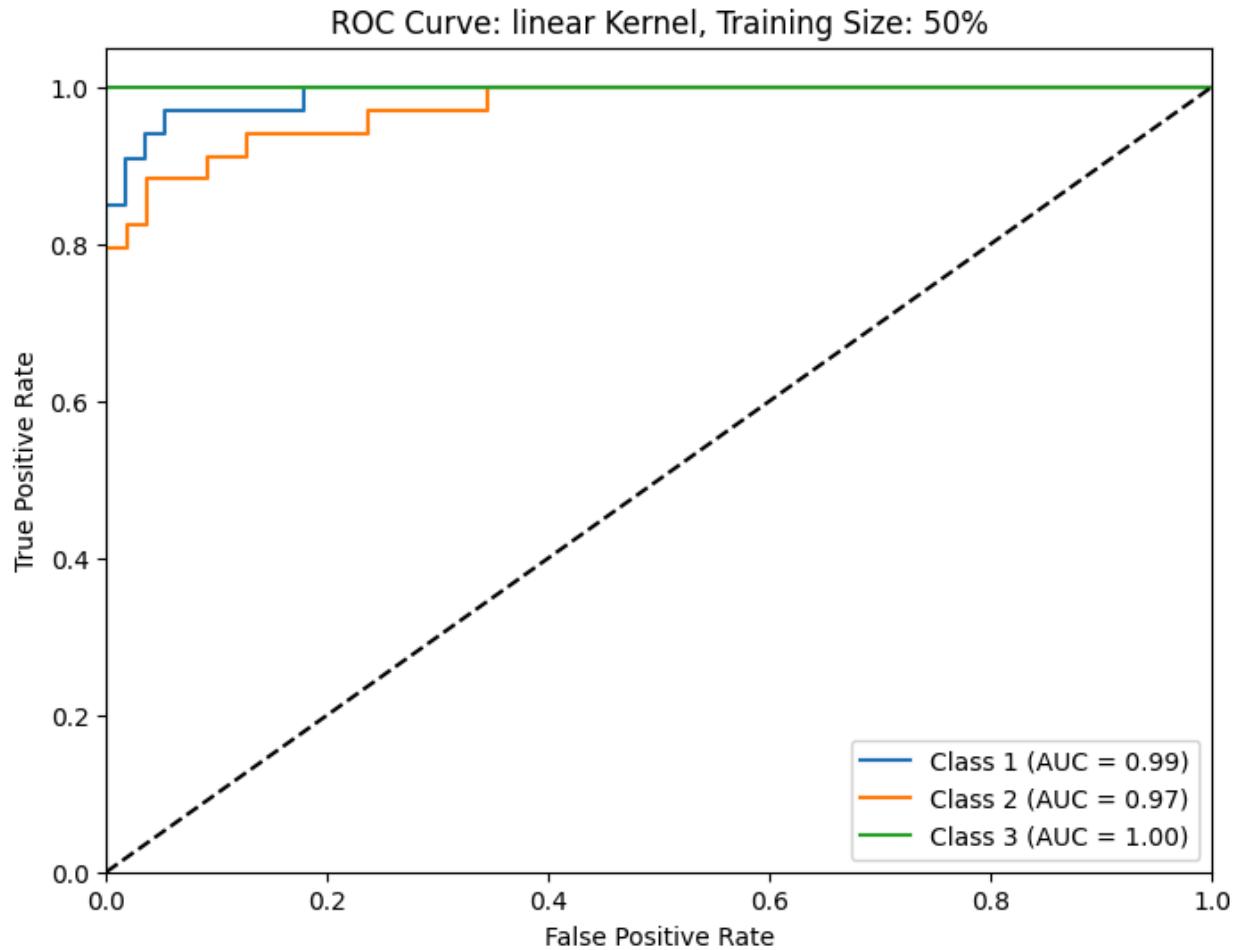
roc_auc = dict()
n_classes = len(np.unique(y_test))

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
    roc_auc[i] = auc(fpr[i], tpr[i])

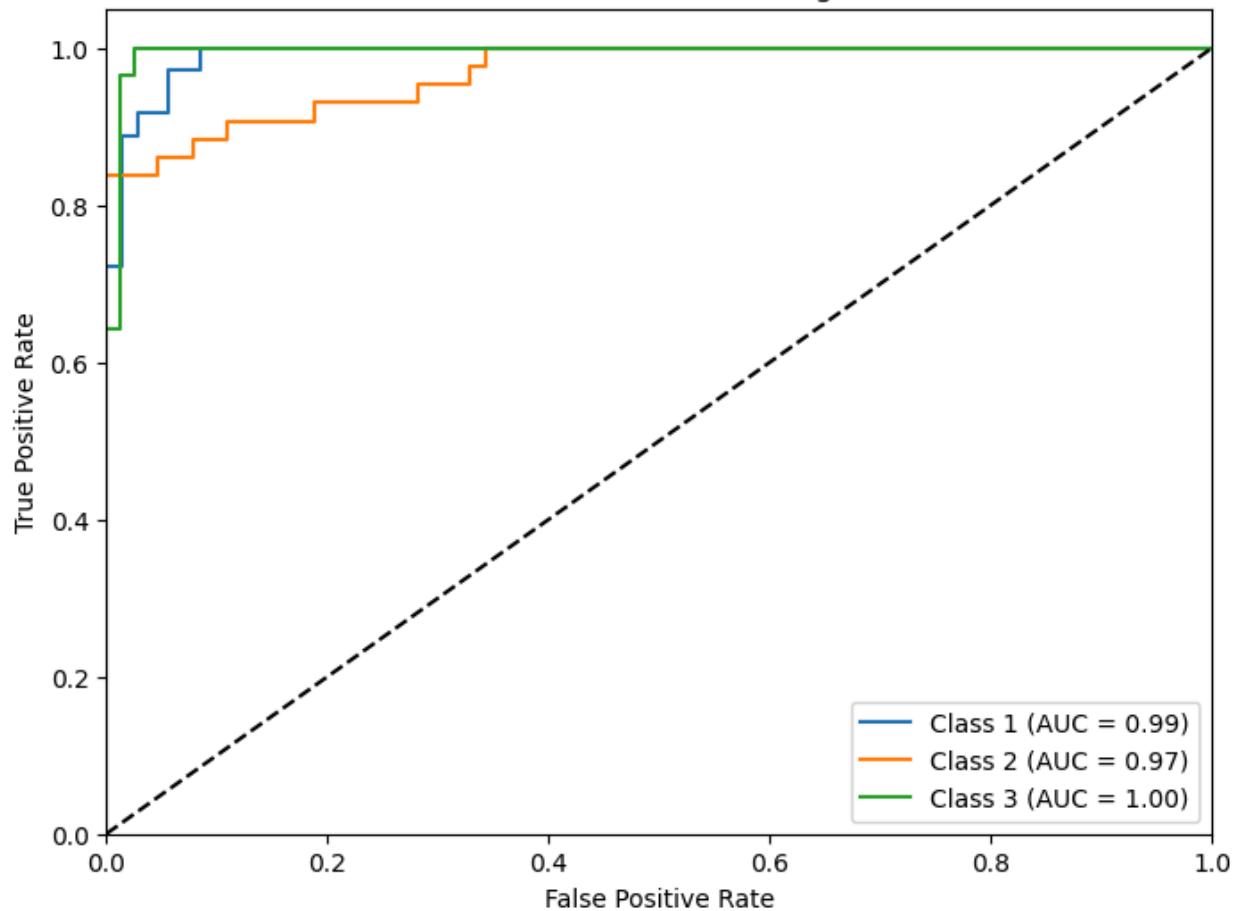
# Plot ROC curve for each class
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve: {kernel_name} Kernel, Training Size: {training_size}')
plt.legend(loc="lower right")
plt.show()

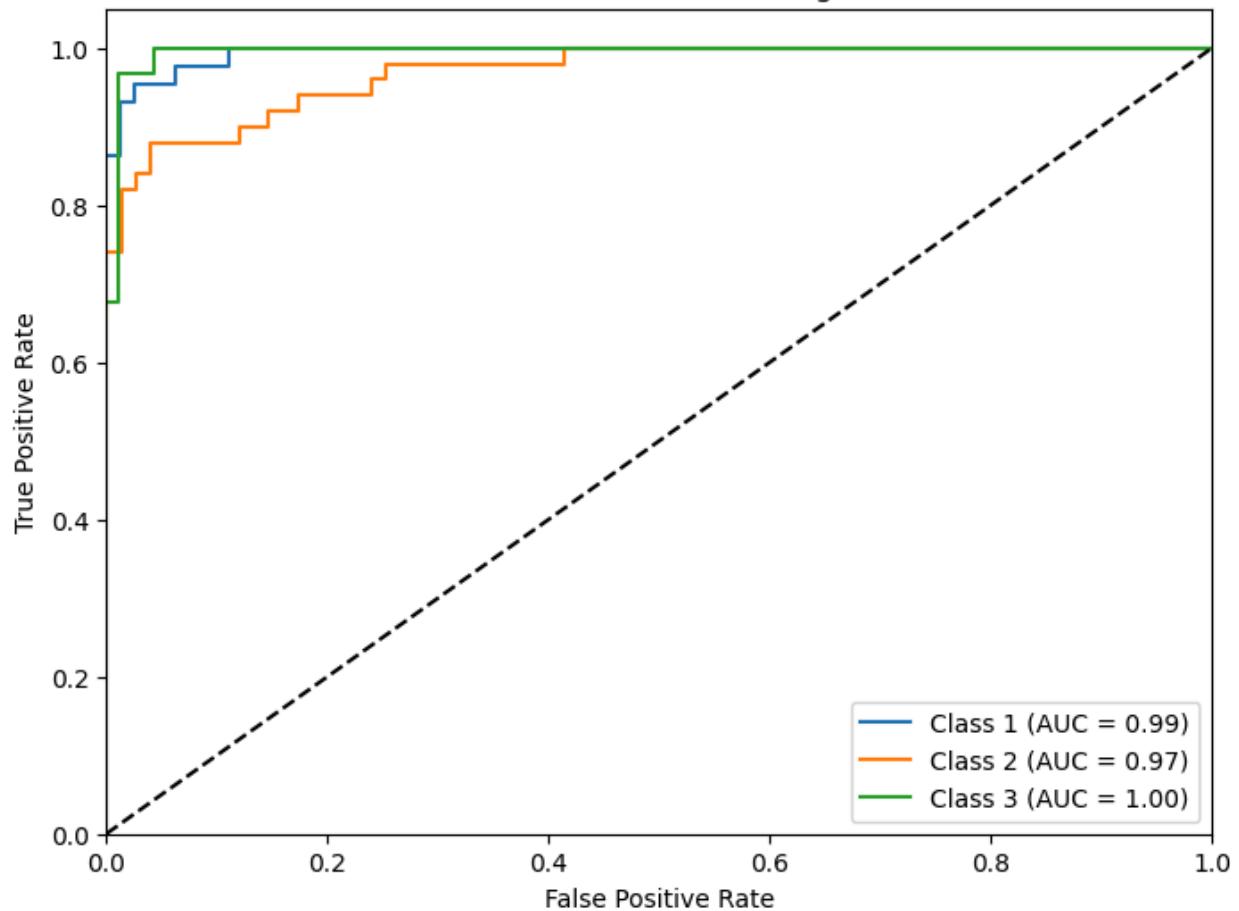
```



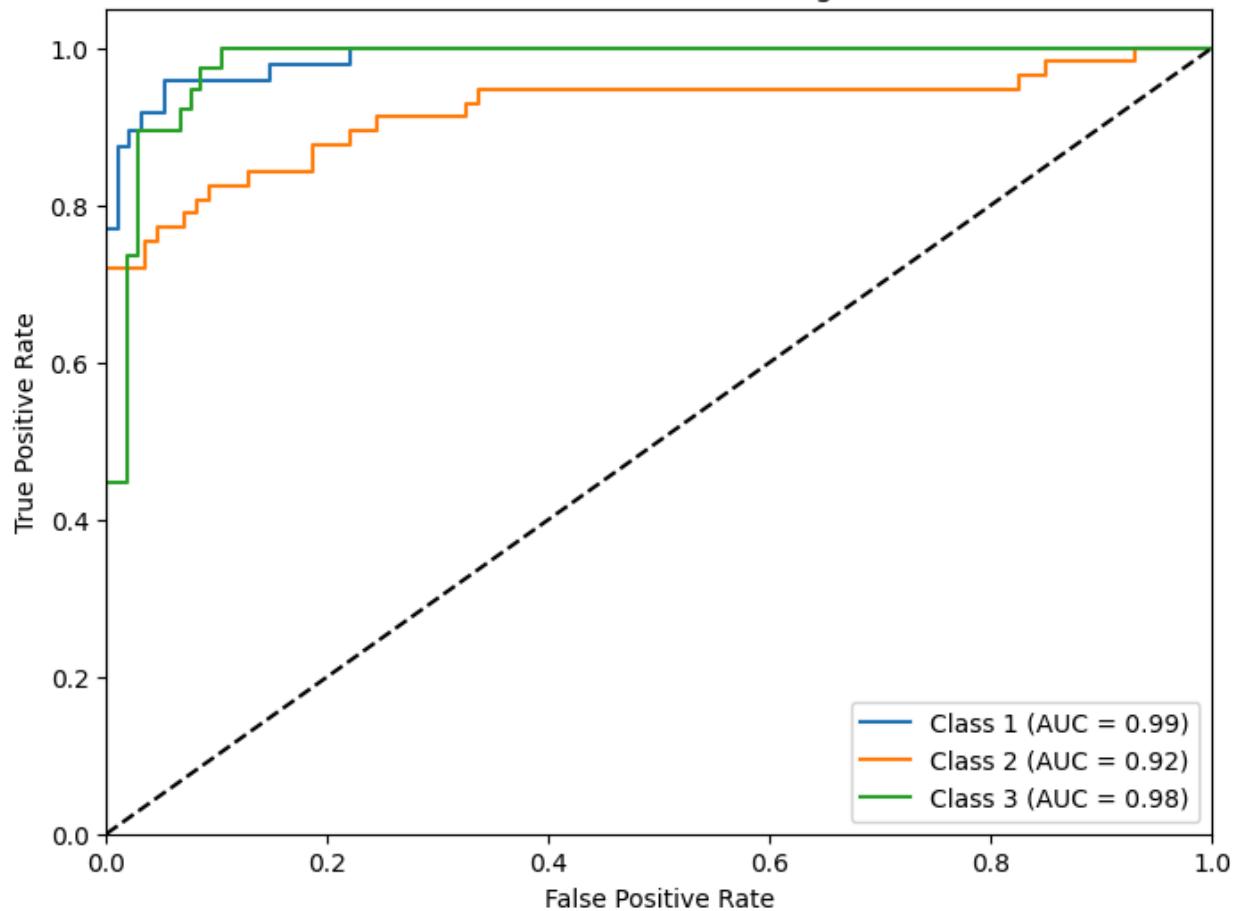
ROC Curve: linear Kernel, Training Size: 60%



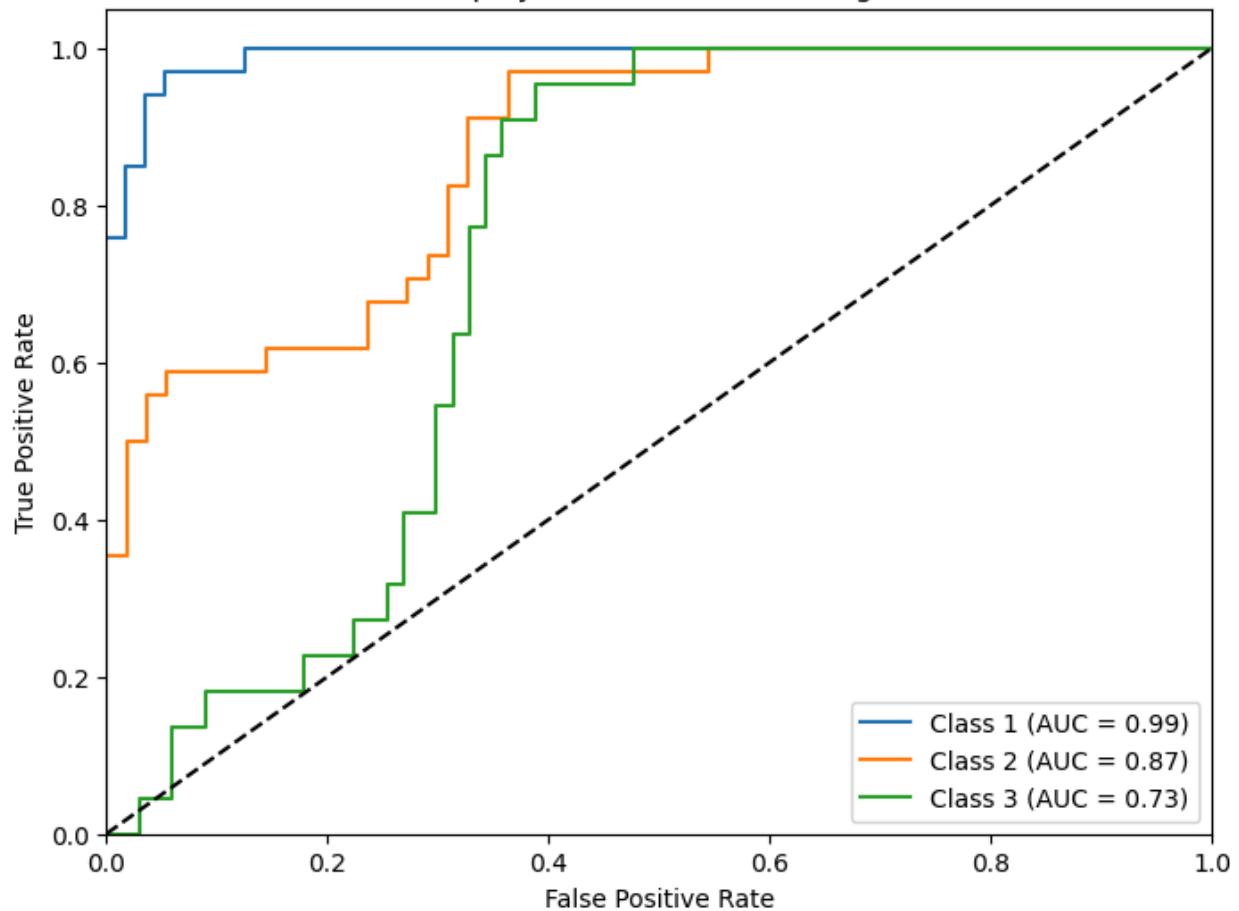
ROC Curve: linear Kernel, Training Size: 70%



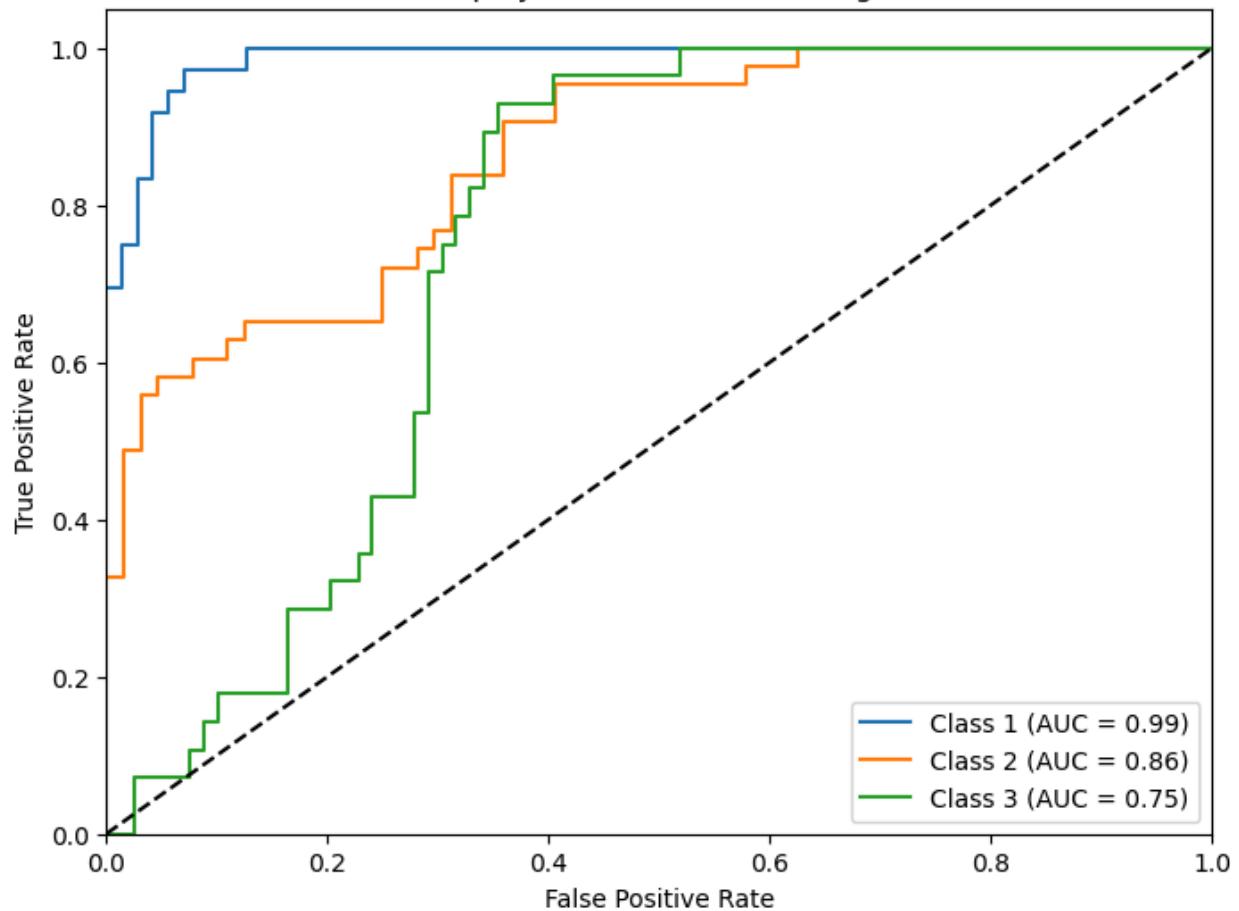
ROC Curve: linear Kernel, Training Size: 80%



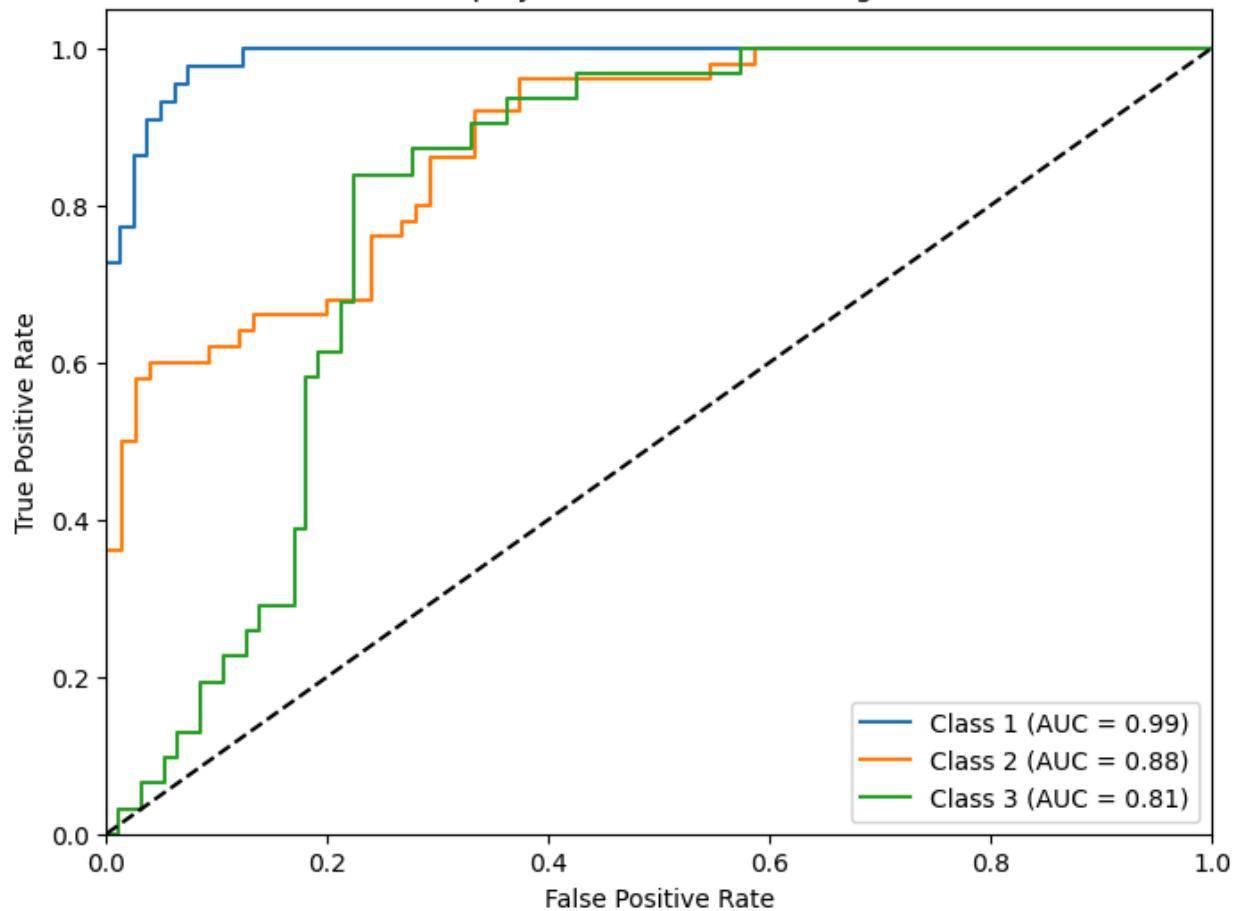
ROC Curve: polynomial Kernel, Training Size: 50%



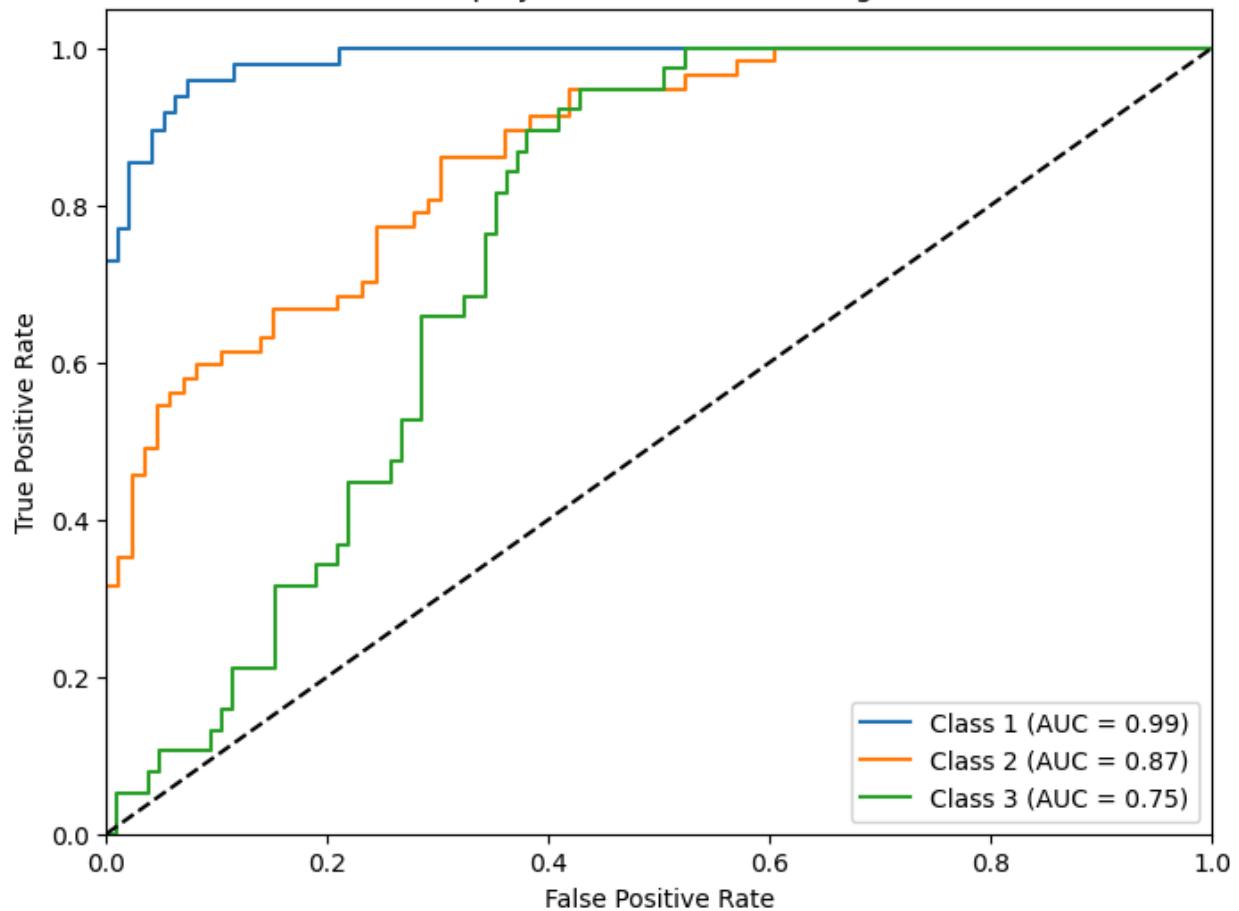
ROC Curve: polynomial Kernel, Training Size: 60%



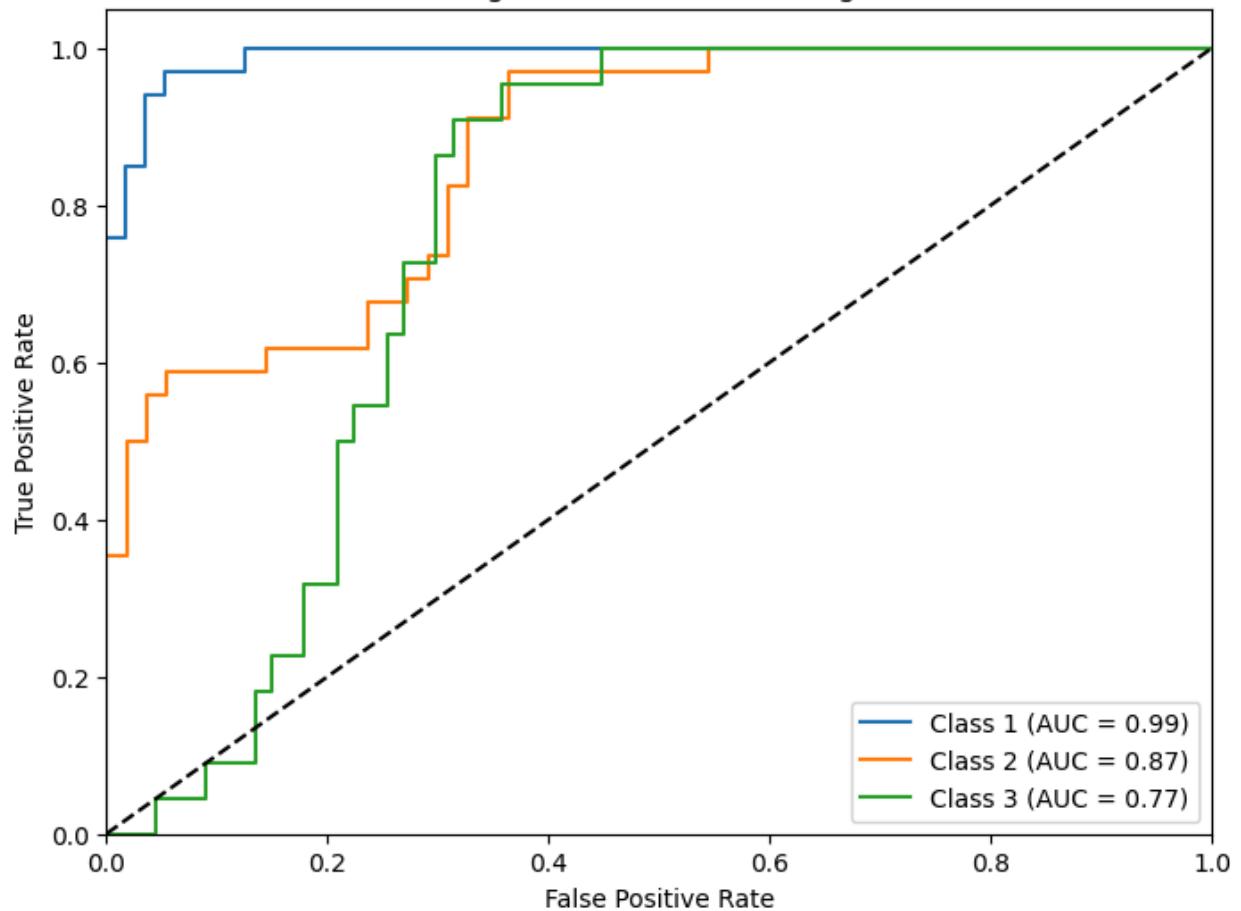
ROC Curve: polynomial Kernel, Training Size: 70%



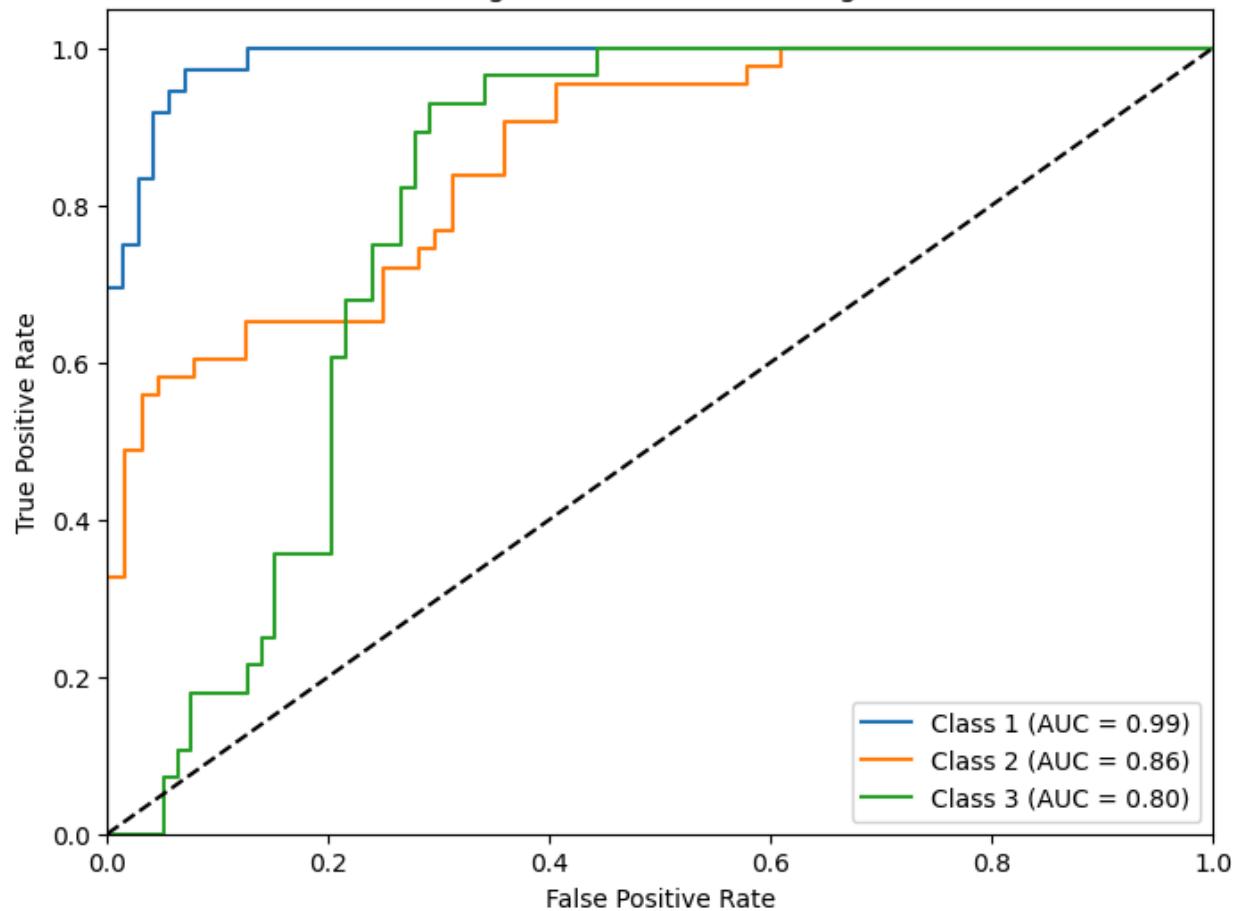
ROC Curve: polynomial Kernel, Training Size: 80%



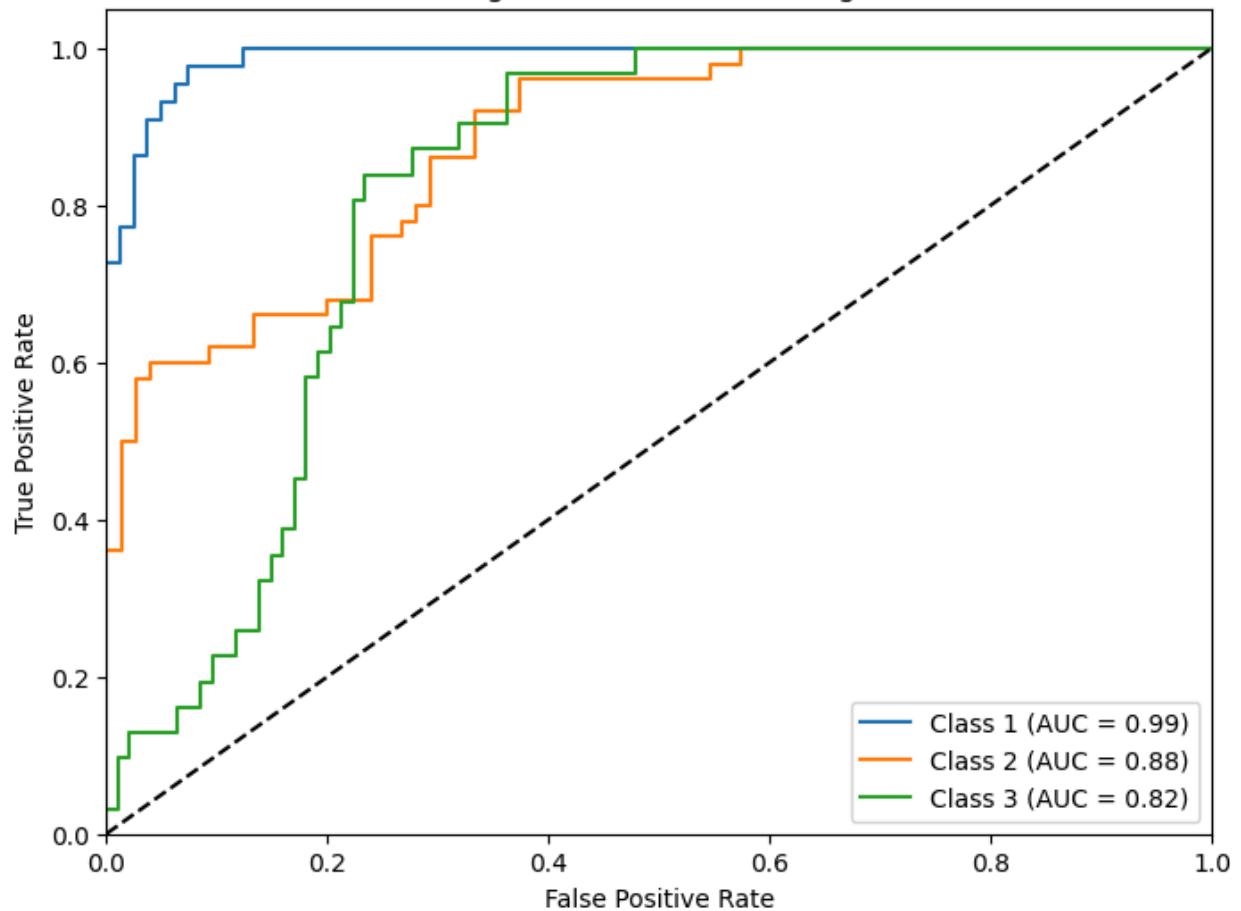
ROC Curve: gaussian Kernel, Training Size: 50%



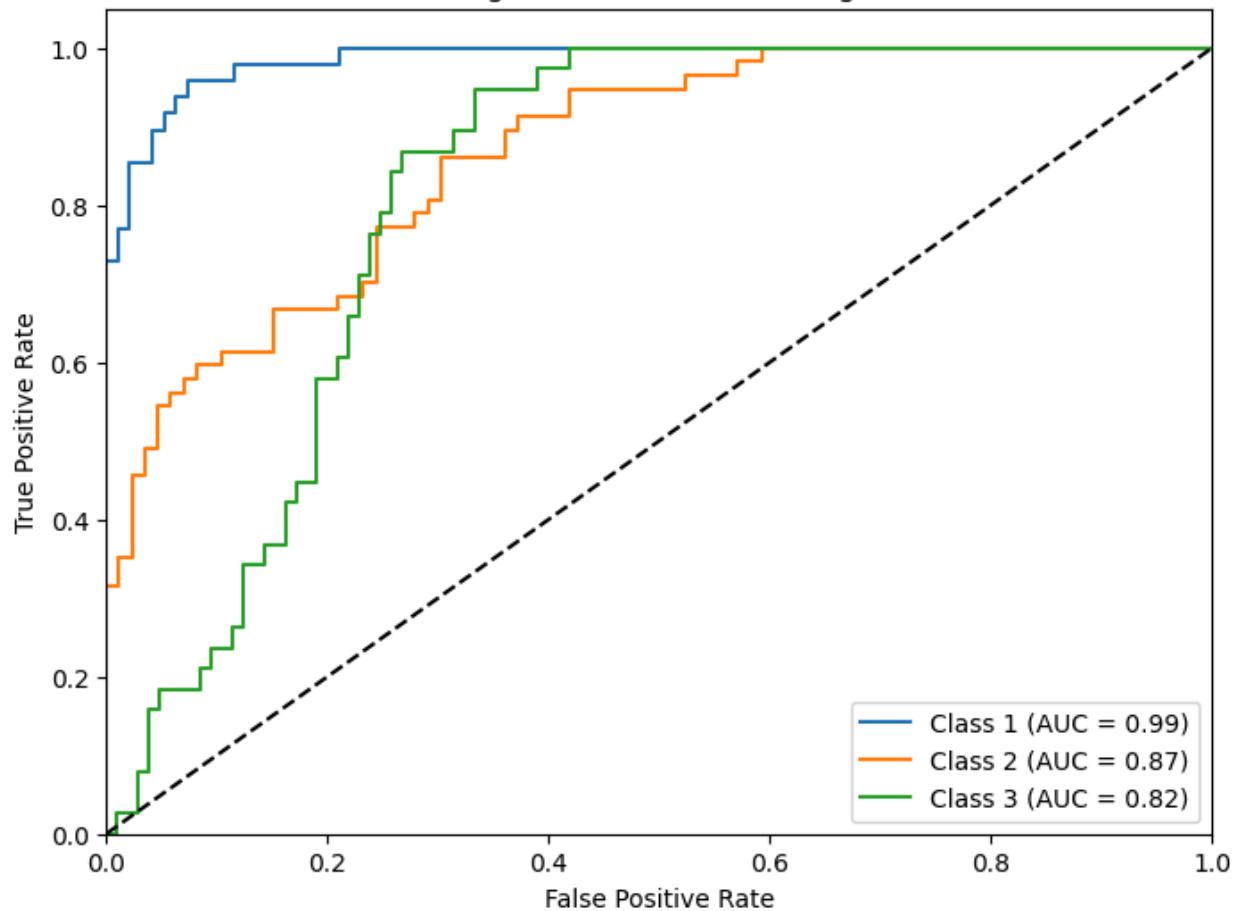
ROC Curve: gaussian Kernel, Training Size: 60%



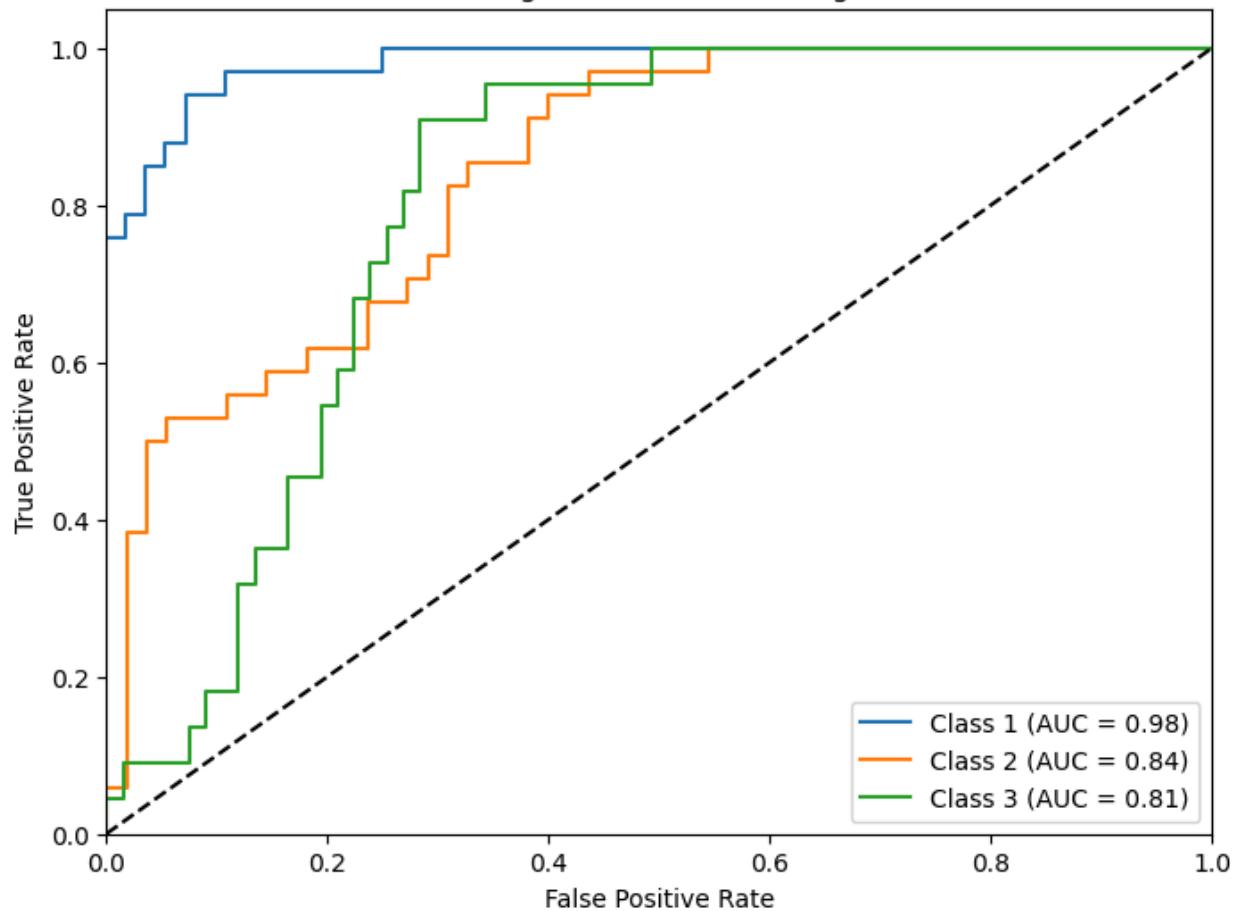
ROC Curve: gaussian Kernel, Training Size: 70%



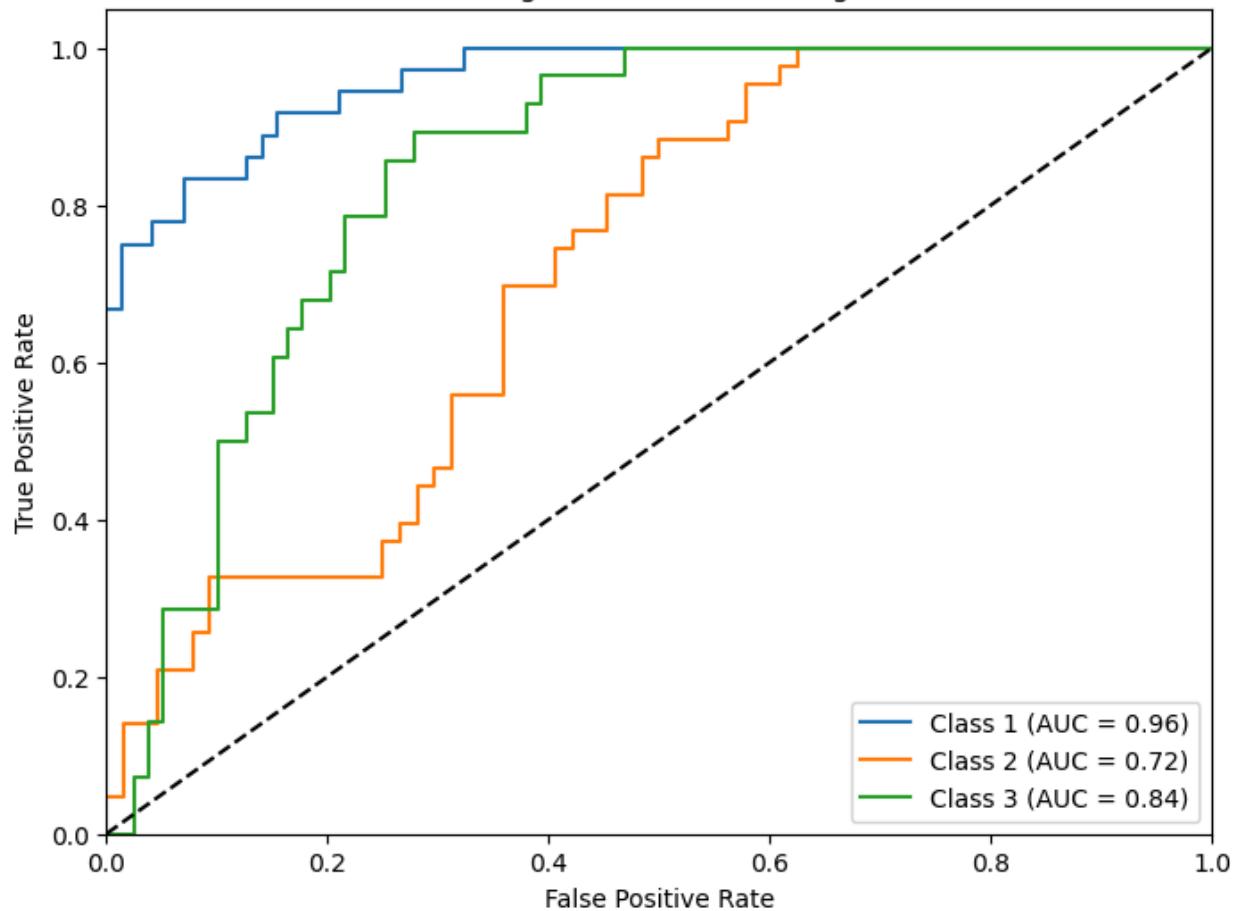
ROC Curve: gaussian Kernel, Training Size: 80%



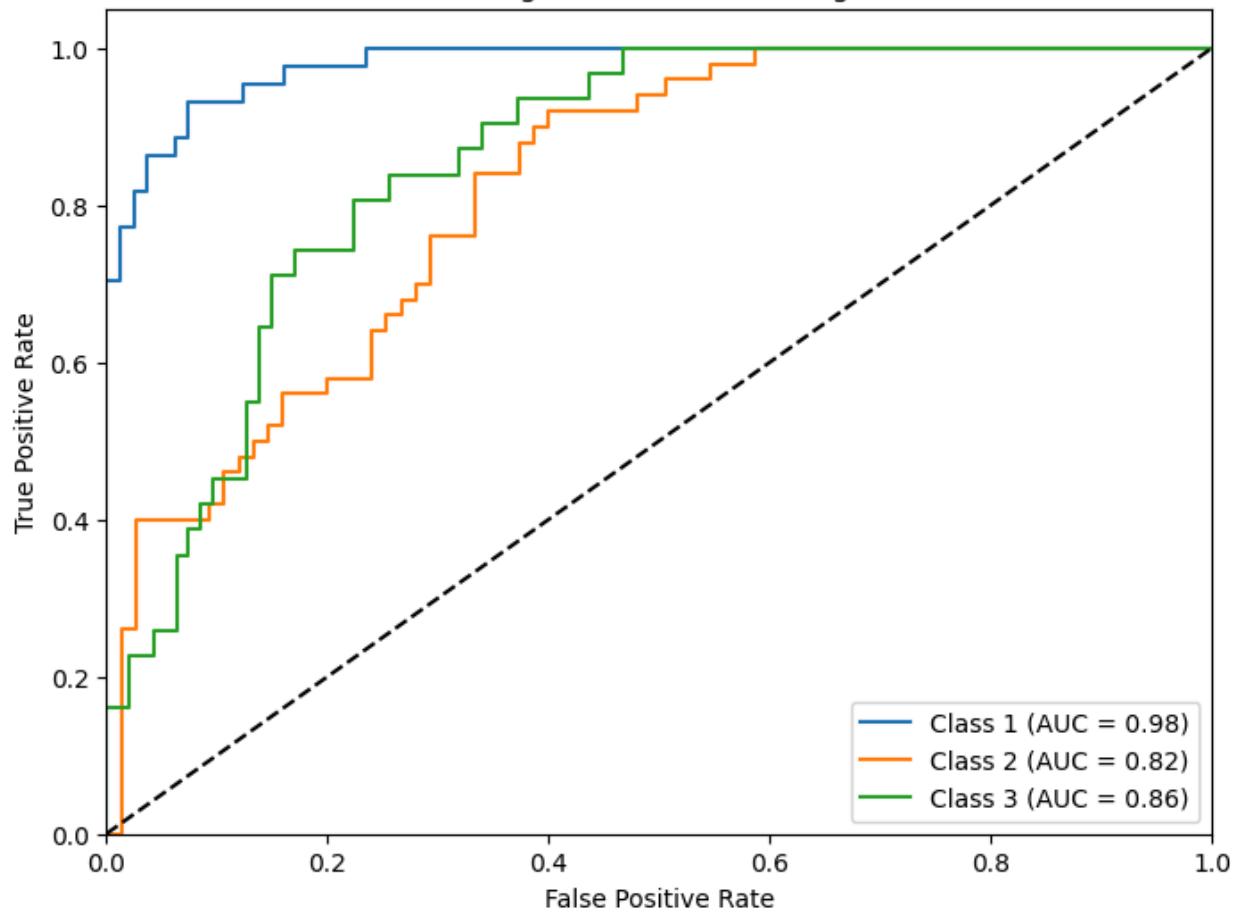
ROC Curve: sigmoid Kernel, Training Size: 50%

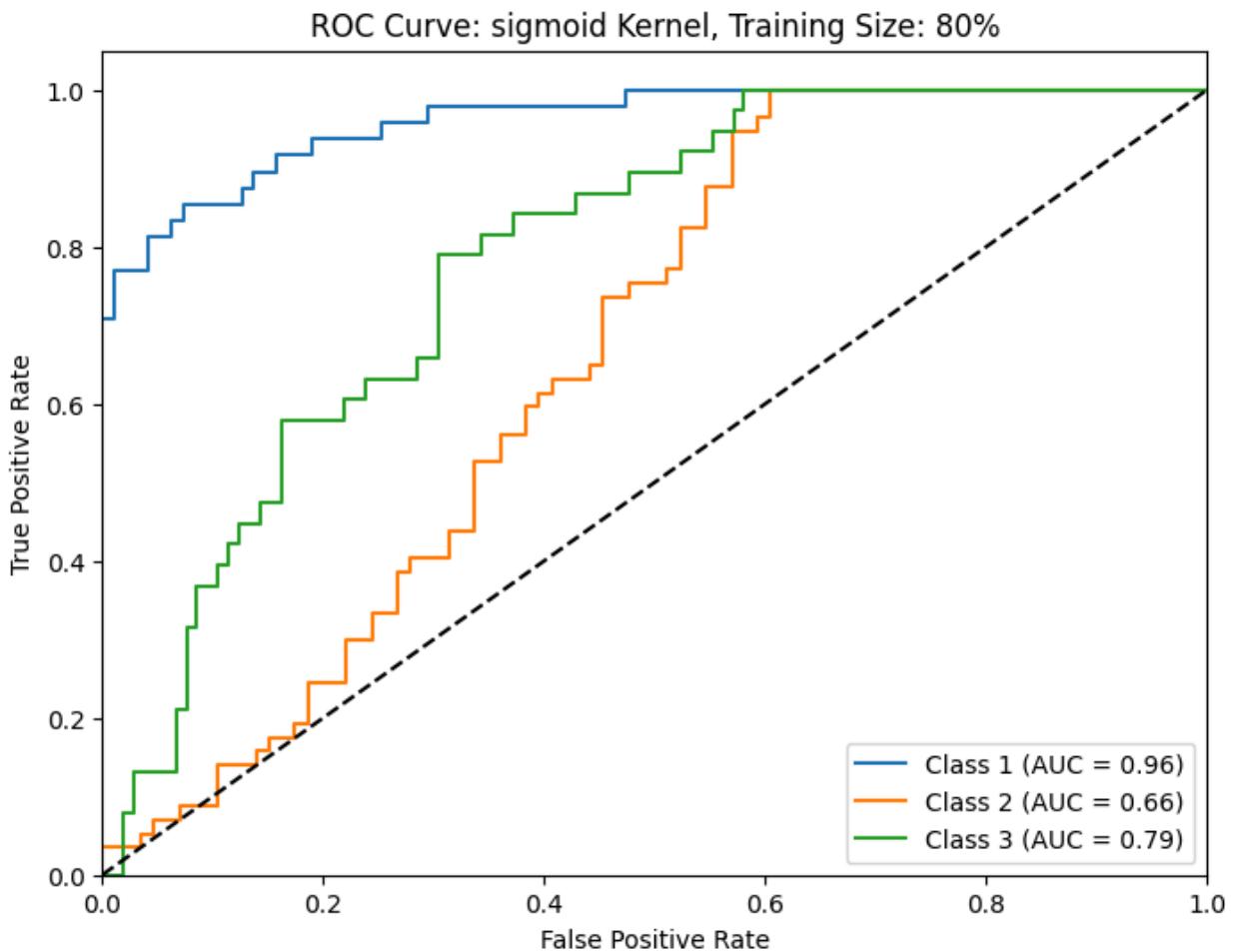


ROC Curve: sigmoid Kernel, Training Size: 60%



ROC Curve: sigmoid Kernel, Training Size: 70%





Calculate values for different test size using PCA-reduced data

```
In [ ]: results_pca = []
confusion_matrices_pca = []

for kernel_name, kernel in kernels.items():
    for size in training_sizes:
        X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca_reduced,
            svc_pca = SVC(kernel=kernel, probability=True, random_state=42) # Add probability
            svc_pca.fit(X_train_pca, y_train.values.ravel())
            y_pred_pca = svc_pca.predict(X_test_pca)

            acc_pca = accuracy_score(y_test, y_pred_pca)
            precision_pca = precision_score(y_test, y_pred_pca, average='weighted', zero_division=1)
            recall_pca = recall_score(y_test, y_pred_pca, average='weighted') # Calculate weighted recall
            f1_pca = f1_score(y_test, y_pred_pca, average='weighted') # Calculate weighted F1 score

            cm_pca = confusion_matrix(y_test, y_pred_pca) # Calculate confusion matrix

            results_pca.append({
                "Training size":int (size*100),
                "Kernel":kernel_name,
                "Accuracy":acc_pca,
                "Precision":precision_pca,
                "Recall":recall_pca,
                "F1 Score":f1_pca
            })
```

```

        "Accuracy":acc_pca,
        "Precision": precision_pca,
        "Recall": recall_pca,
        "F1-score": f1_pca,
        "Kernel":kernel_name
    })
confusion_matrices_pca.append({ # Store confusion matrix with metadata
    "Training size":int (size*100),
    "Kernel":kernel_name,
    "Confusion Matrix":cm_pca
})

```

## Print Accuracy, Precision, Recall, and F1-score Table and Graphs for PCA-Reduced Data

```

In [ ]: df_pca = pd.DataFrame(results_pca)
display(df_pca) # Display the DataFrame as a table

# Plot Accuracy
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Accuracy', hue='Kernel')
plt.title('SVM Accuracy by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

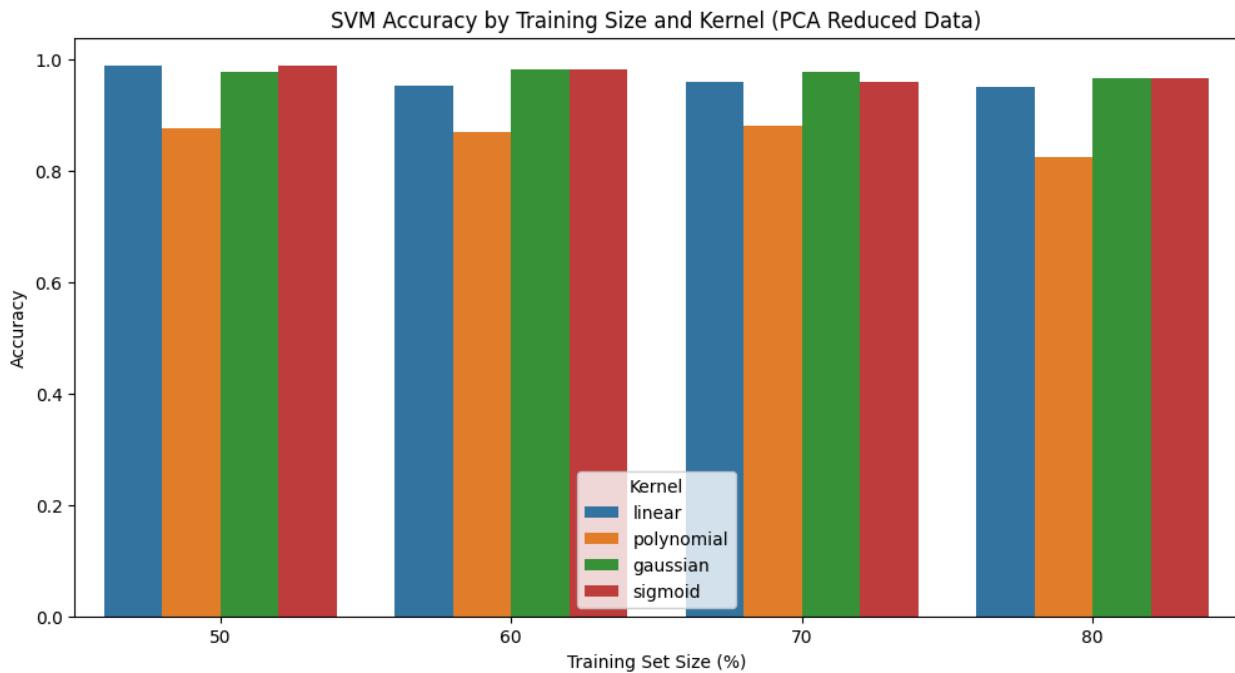
# Plot Precision
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Precision', hue='Kernel')
plt.title('SVM Precision by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

# Plot Recall
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Recall', hue='Kernel')
plt.title('SVM Recall by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

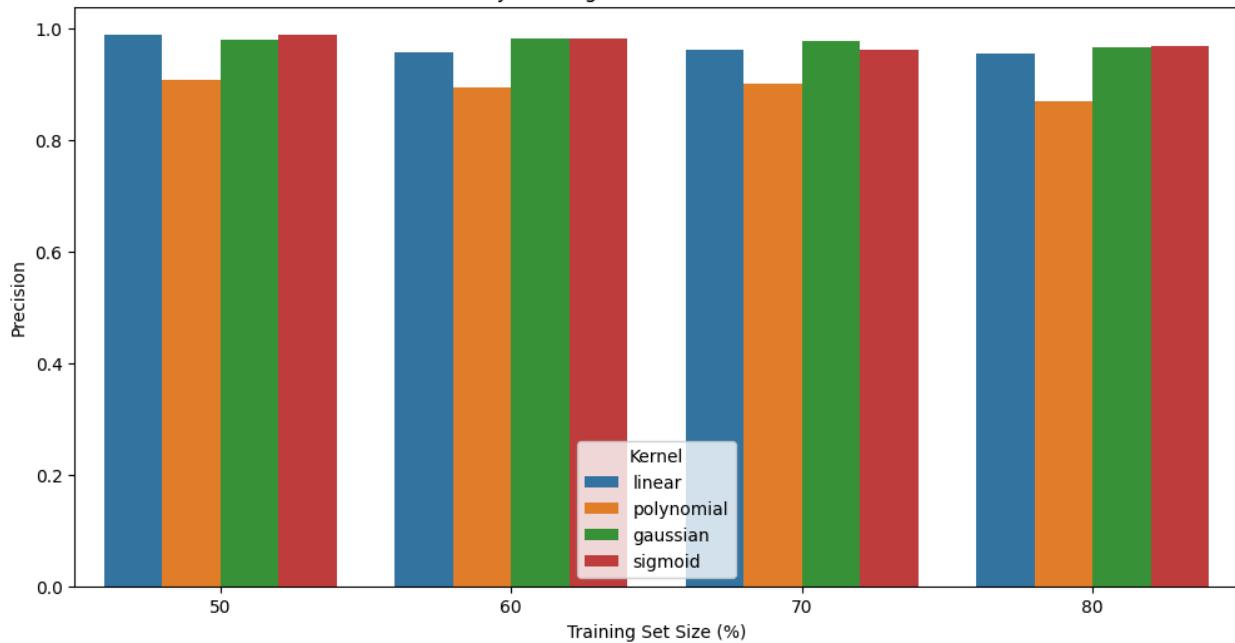
# Plot F1-score
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='F1-score', hue='Kernel')
plt.title('SVM F1-score by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()

```

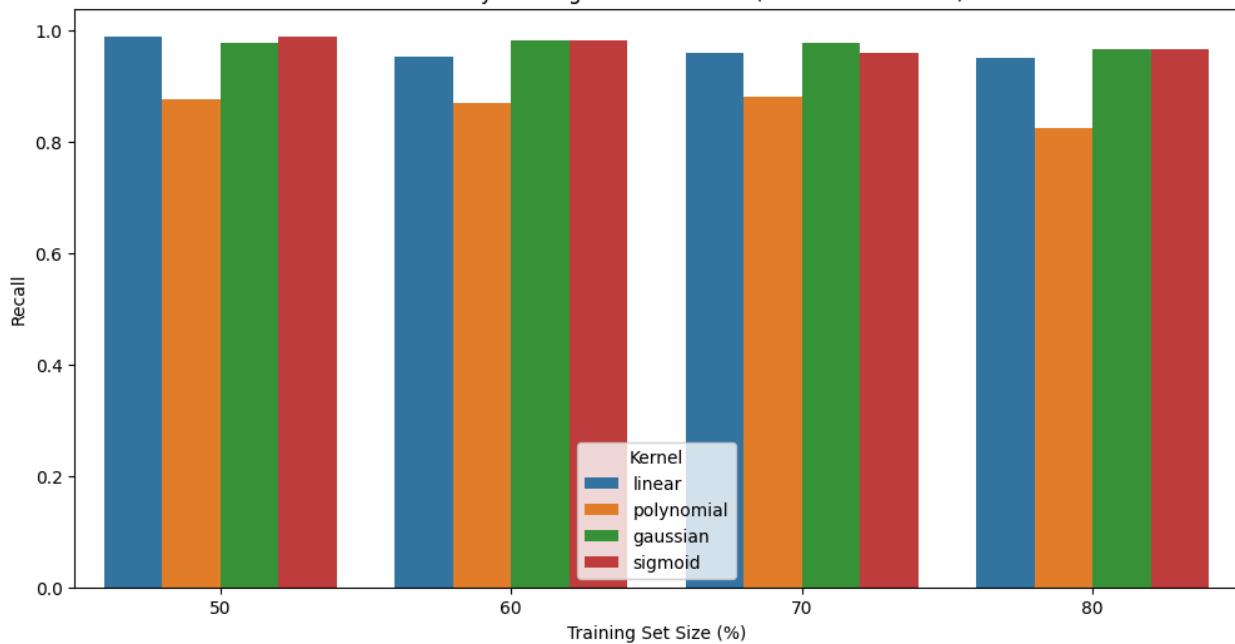
<b>Training size</b>		<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Kernel</b>
<b>0</b>	50	0.988764	0.989085	0.988764	0.988715	linear
<b>1</b>	60	0.953271	0.956072	0.953271	0.953166	linear
<b>2</b>	70	0.960000	0.962150	0.960000	0.960011	linear
<b>3</b>	80	0.951049	0.954736	0.951049	0.950406	linear
<b>4</b>	50	0.876404	0.906617	0.876404	0.874846	polynomial
<b>5</b>	60	0.869159	0.892997	0.869159	0.869891	polynomial
<b>6</b>	70	0.880000	0.900444	0.880000	0.879207	polynomial
<b>7</b>	80	0.825175	0.870248	0.825175	0.825552	polynomial
<b>8</b>	50	0.977528	0.978777	0.977528	0.977632	gaussian
<b>9</b>	60	0.981308	0.982139	0.981308	0.981370	gaussian
<b>10</b>	70	0.976000	0.976806	0.976000	0.976083	gaussian
<b>11</b>	80	0.965035	0.965834	0.965035	0.965202	gaussian
<b>12</b>	50	0.988764	0.989095	0.988764	0.988764	sigmoid
<b>13</b>	60	0.981308	0.981308	0.981308	0.981308	sigmoid
<b>14</b>	70	0.960000	0.961607	0.960000	0.959935	sigmoid
<b>15</b>	80	0.965035	0.967366	0.965035	0.964838	sigmoid

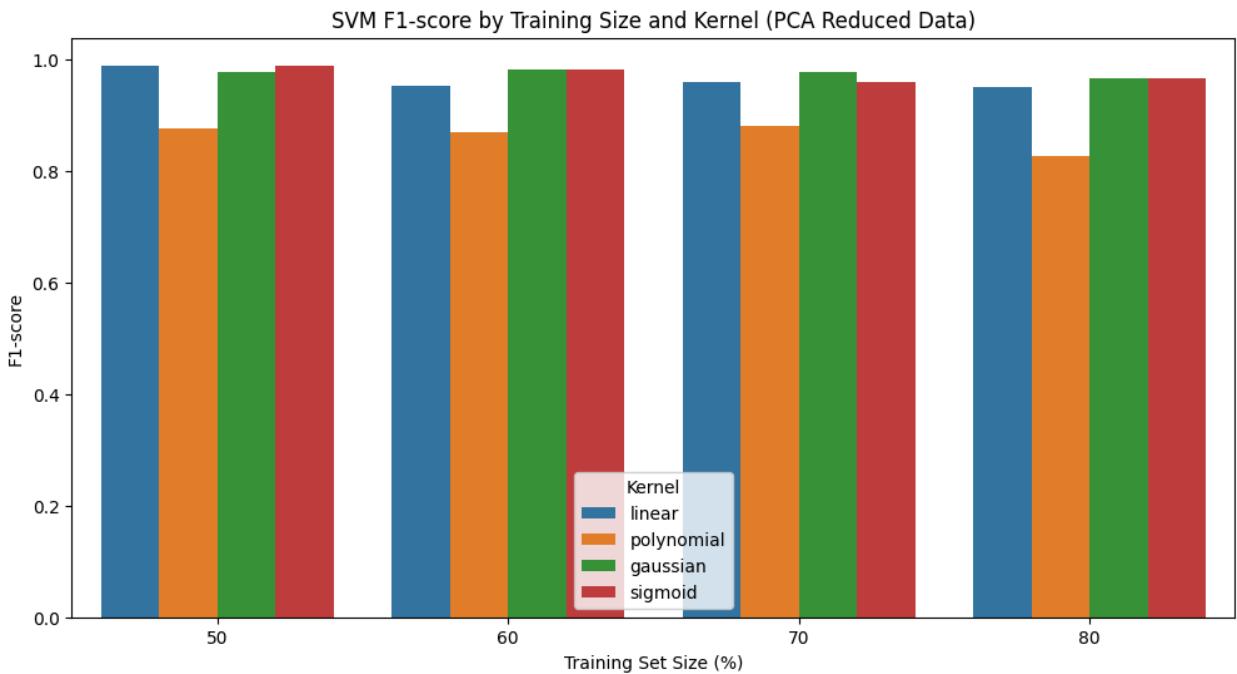


SVM Precision by Training Size and Kernel (PCA Reduced Data)



SVM Recall by Training Size and Kernel (PCA Reduced Data)

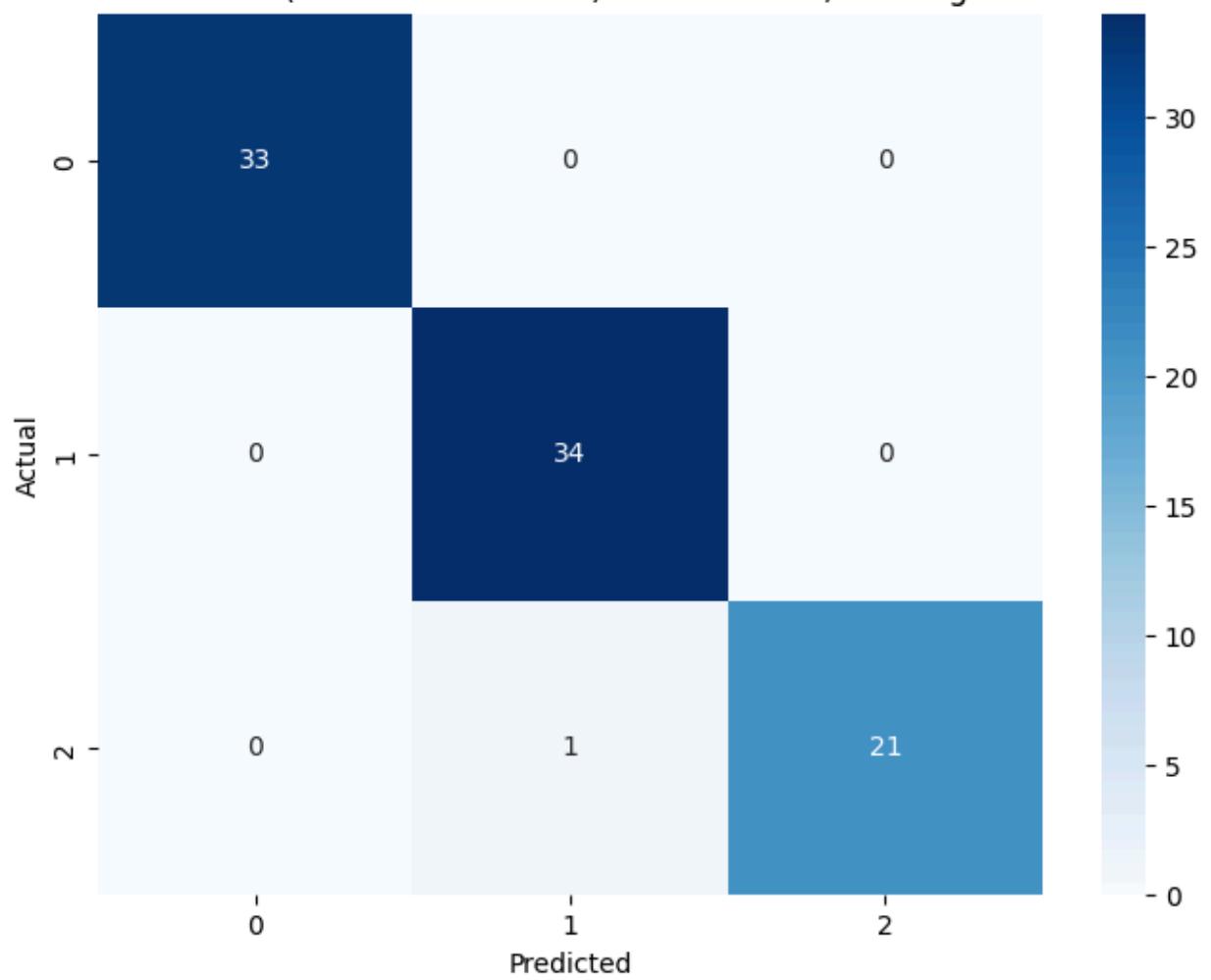




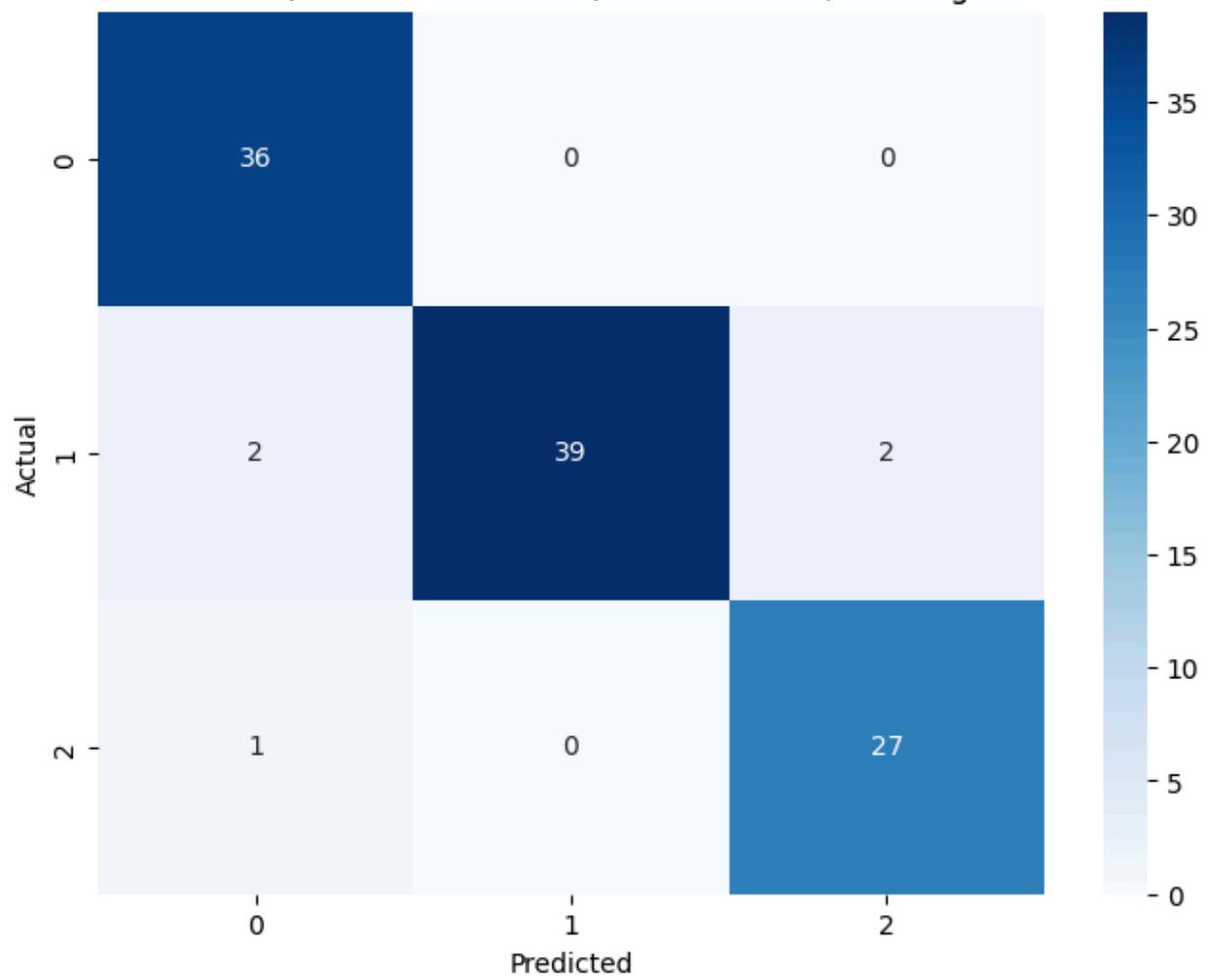
## Show Confusion Matrices for PCA-Reduced Data

```
In [ ]: for cm_data in confusion_matrices_pca:  
    cm = cm_data["Confusion Matrix"]  
    kernel_name = cm_data["Kernel"]  
    training_size = cm_data["Training size"]  
  
    plt.figure(figsize=(8, 6))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  
    plt.title(f'Confusion Matrix (PCA Reduced Data): {kernel_name} Kernel, Tra  
    plt.xlabel('Predicted')  
    plt.ylabel('Actual')  
    plt.show()
```

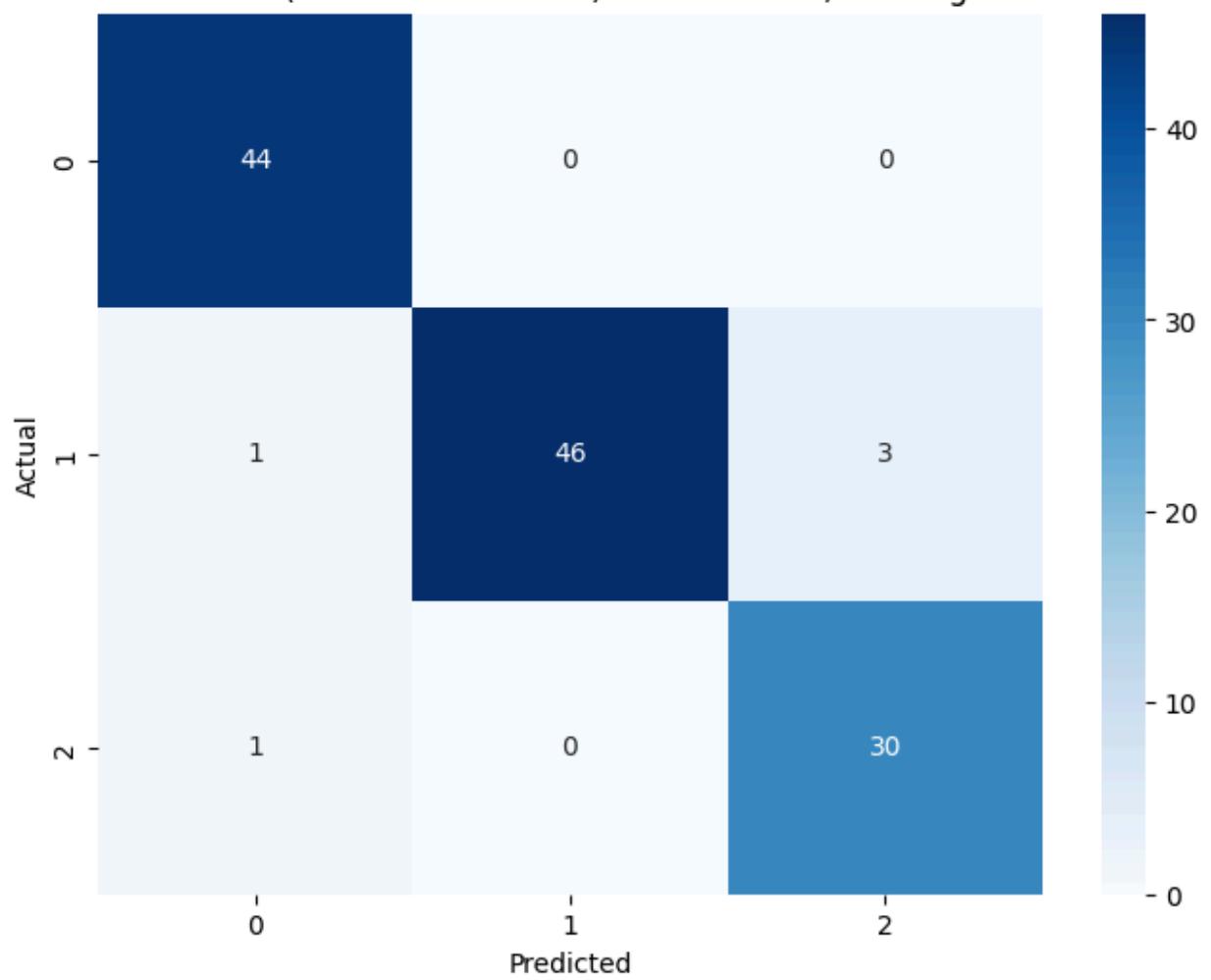
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 50%



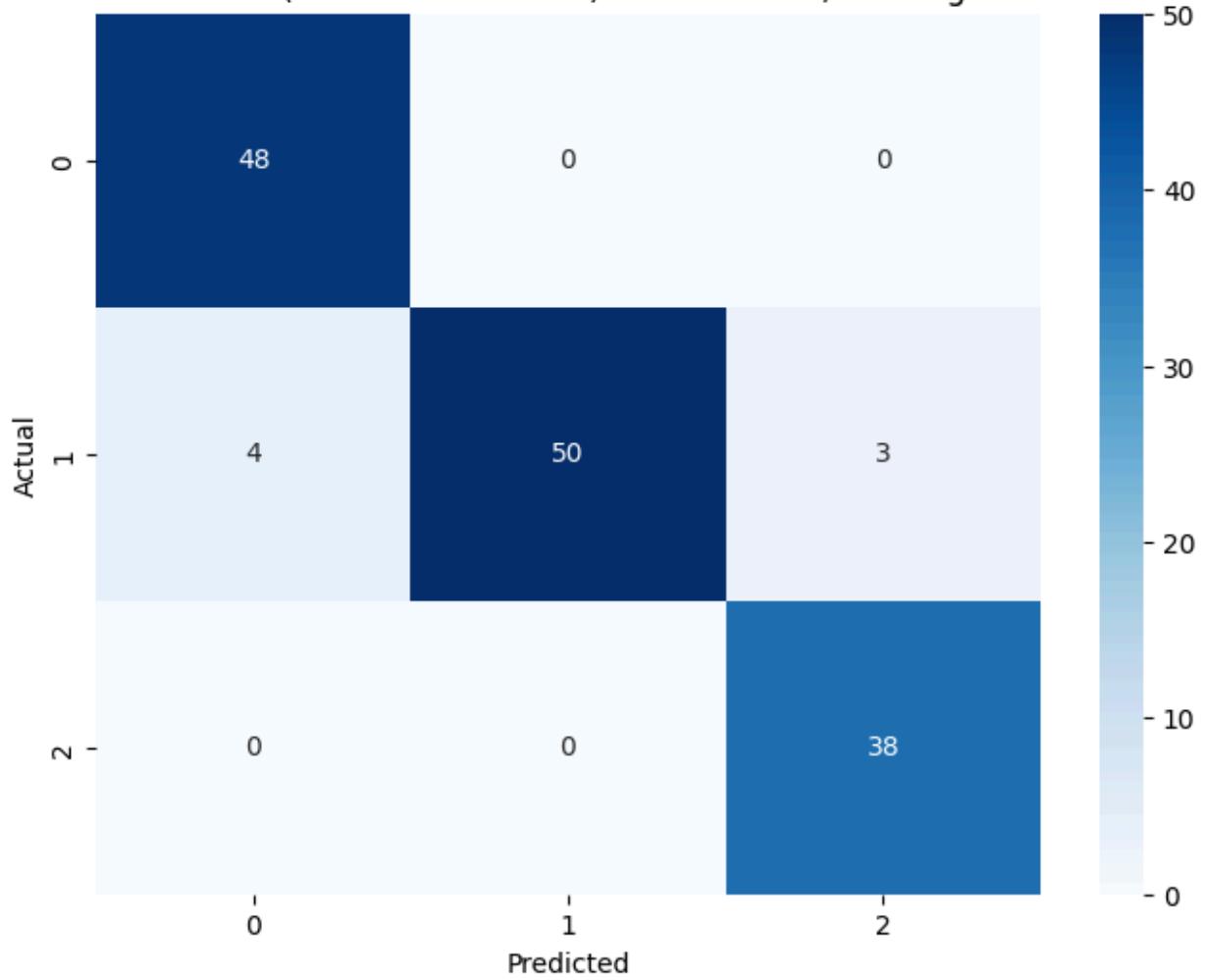
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 60%



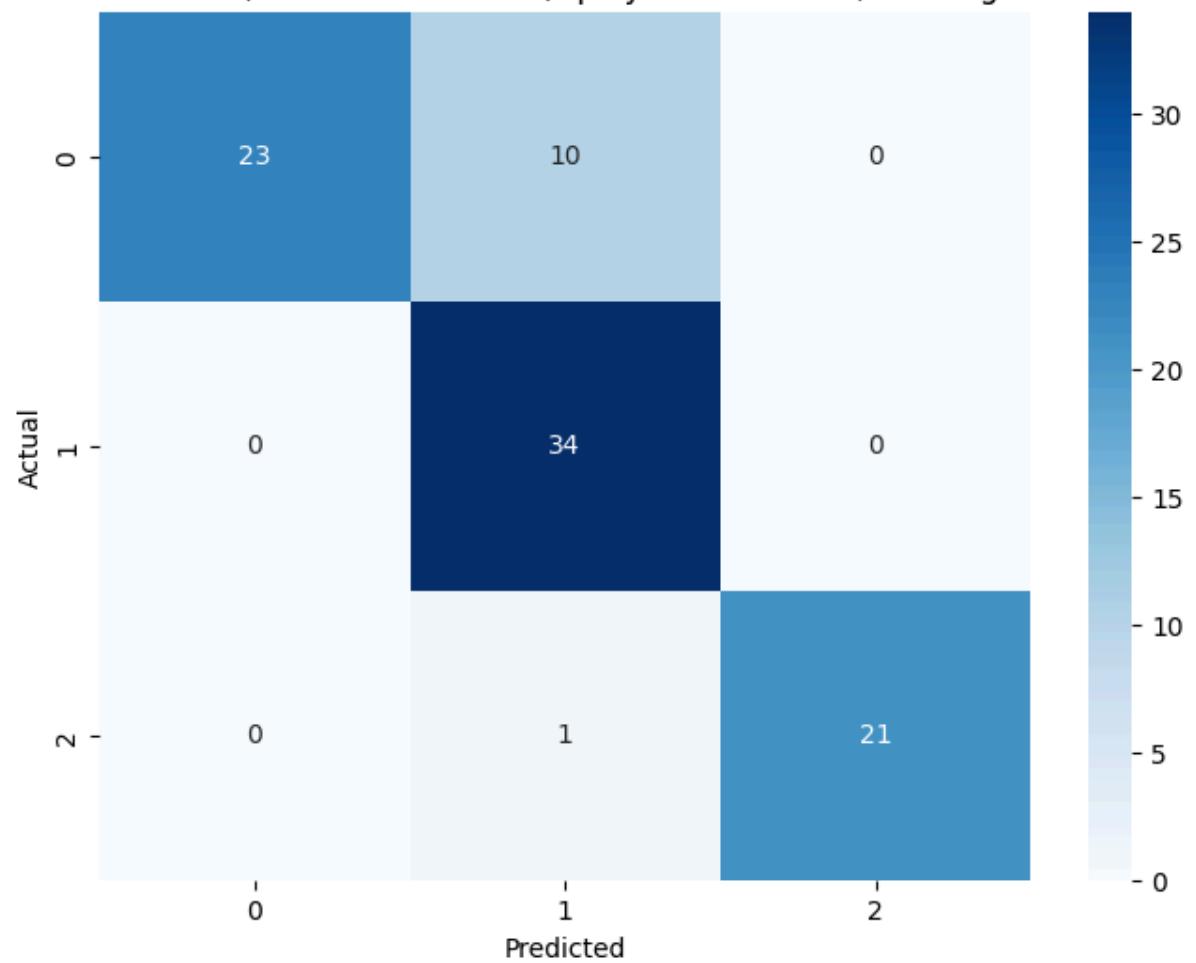
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 70%



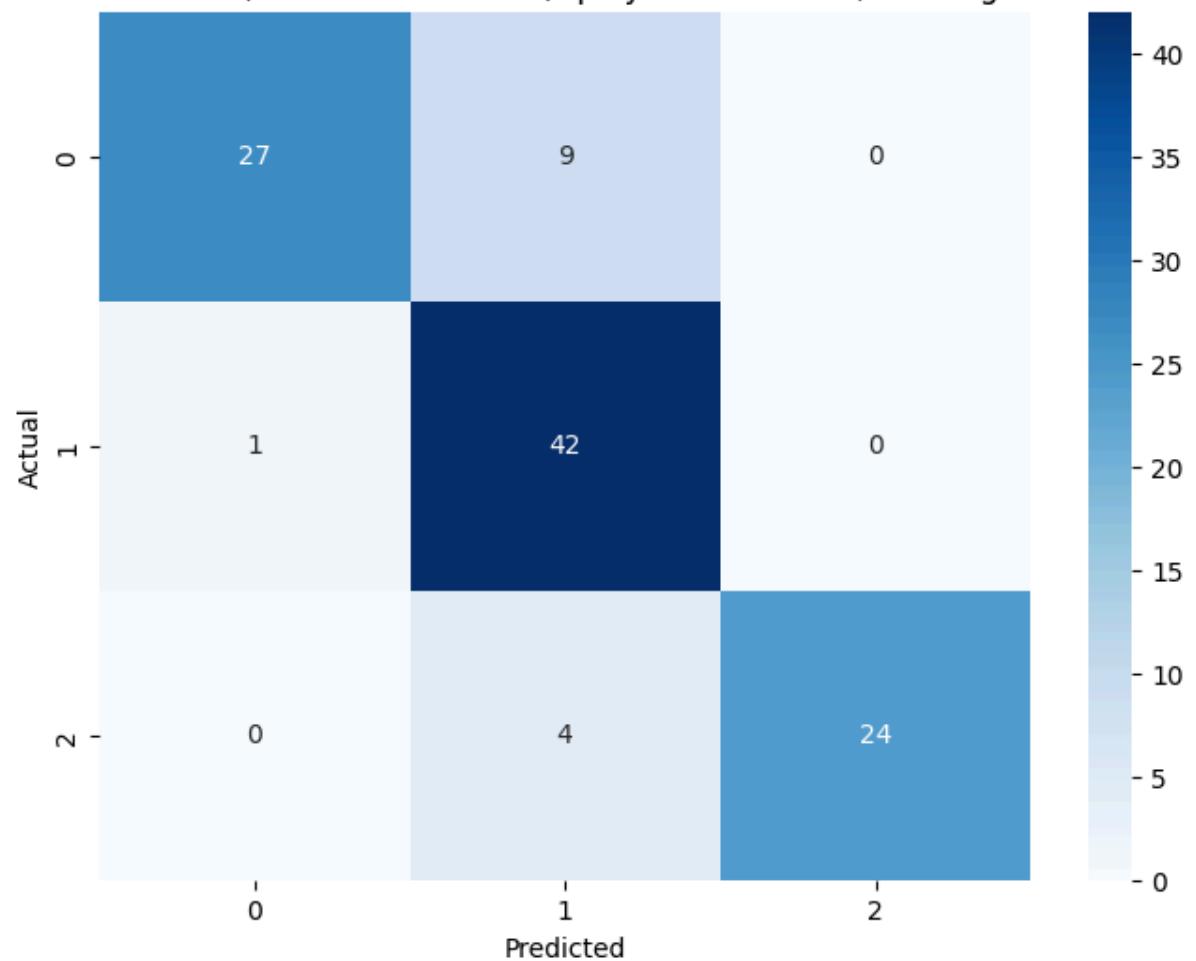
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 80%



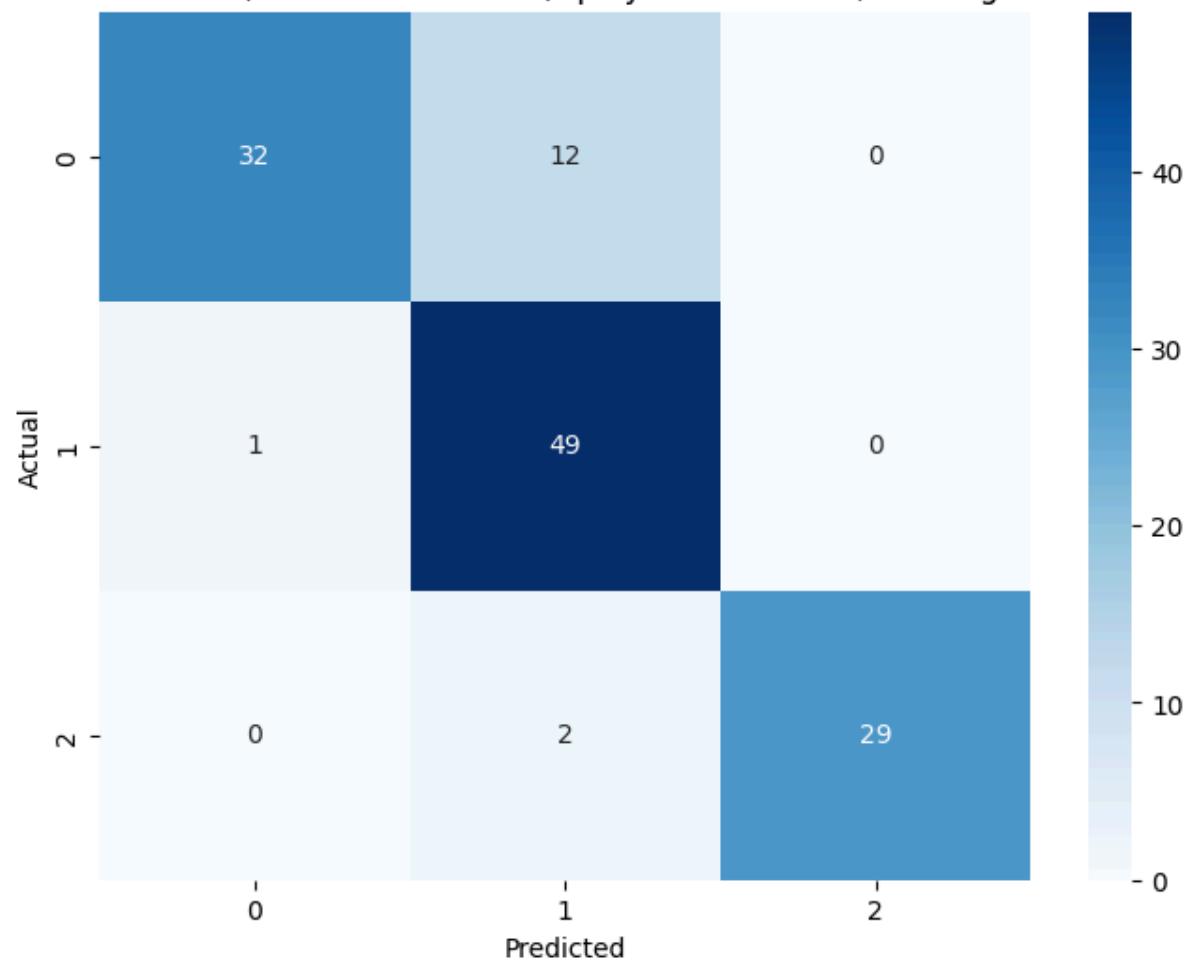
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 50%



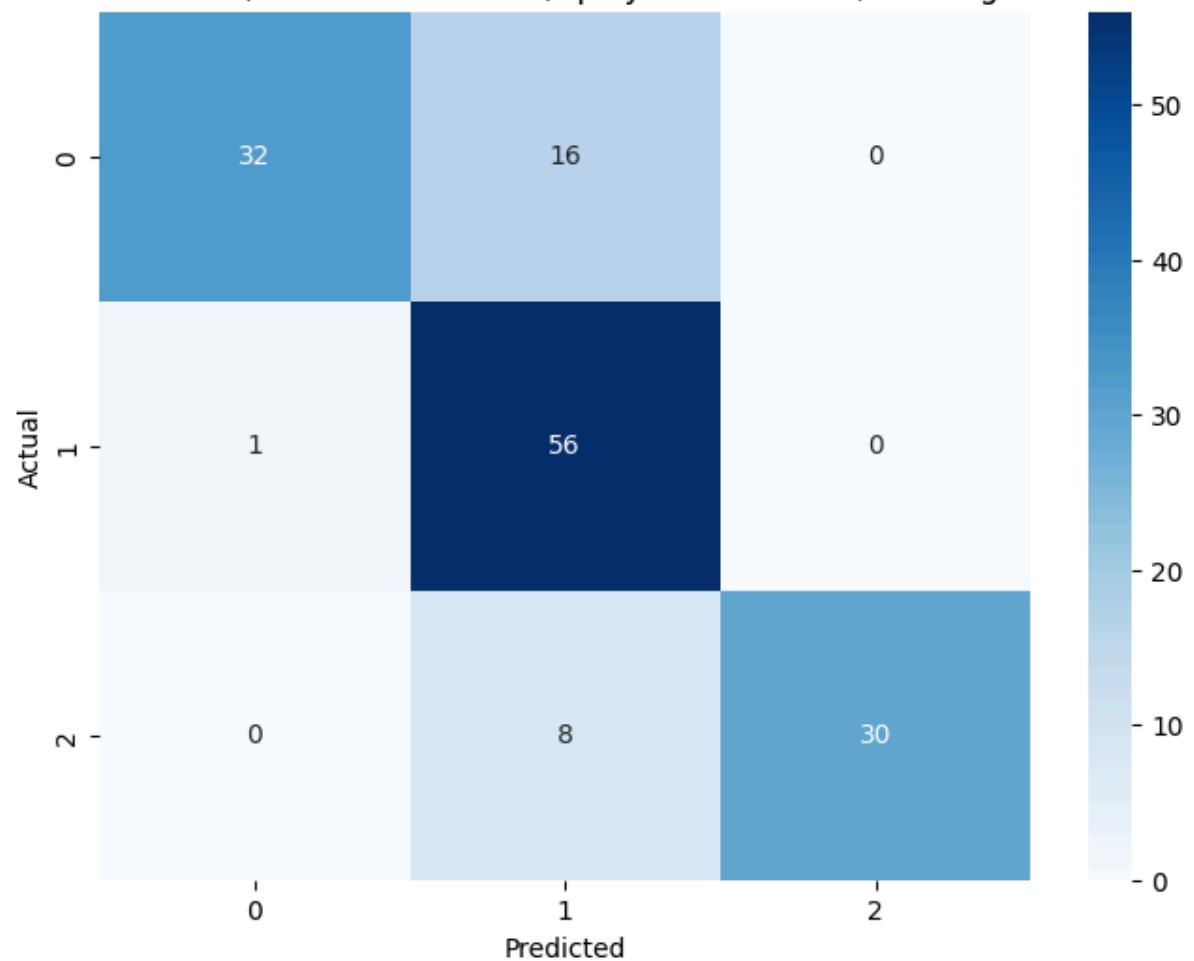
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 60%



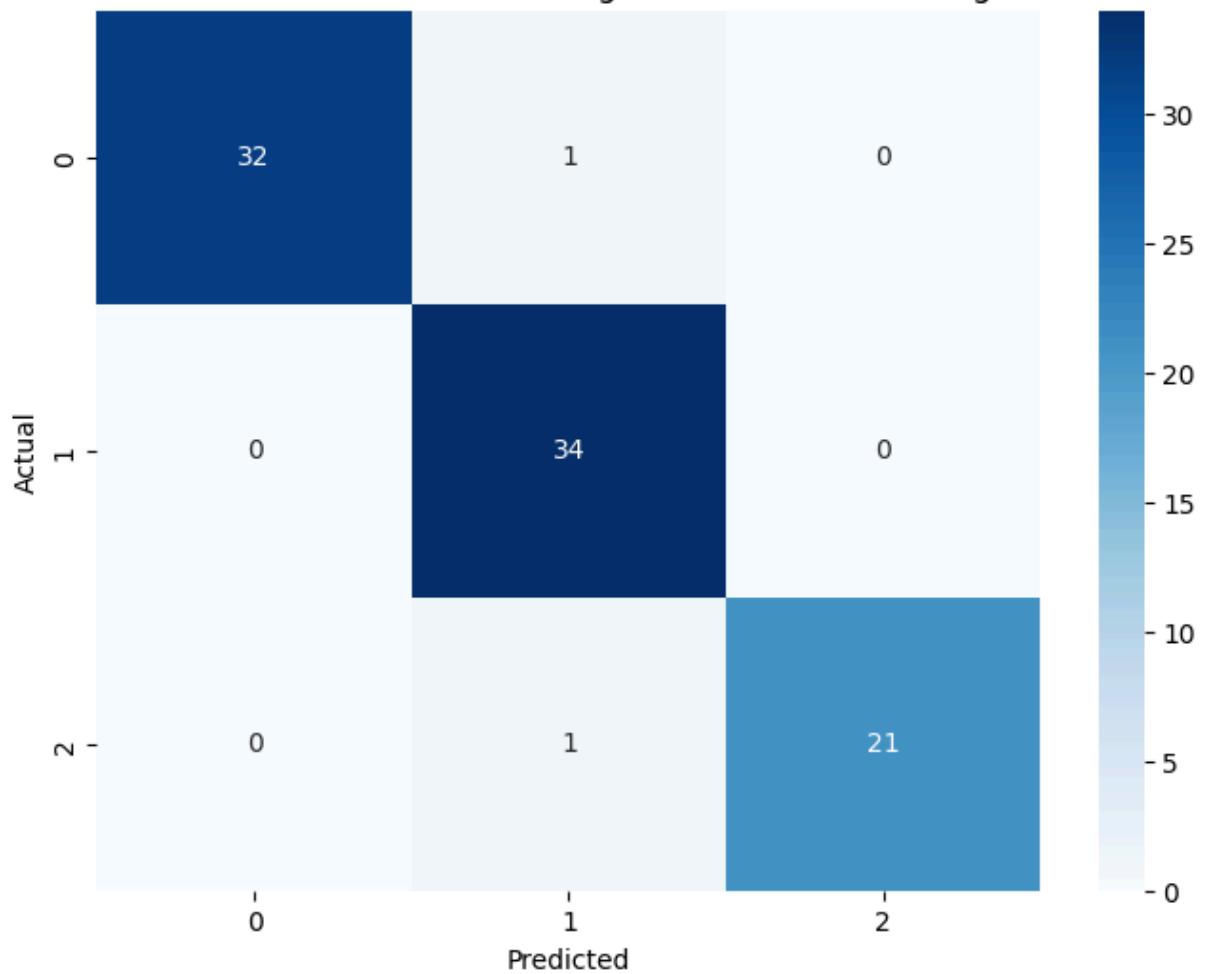
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 70%



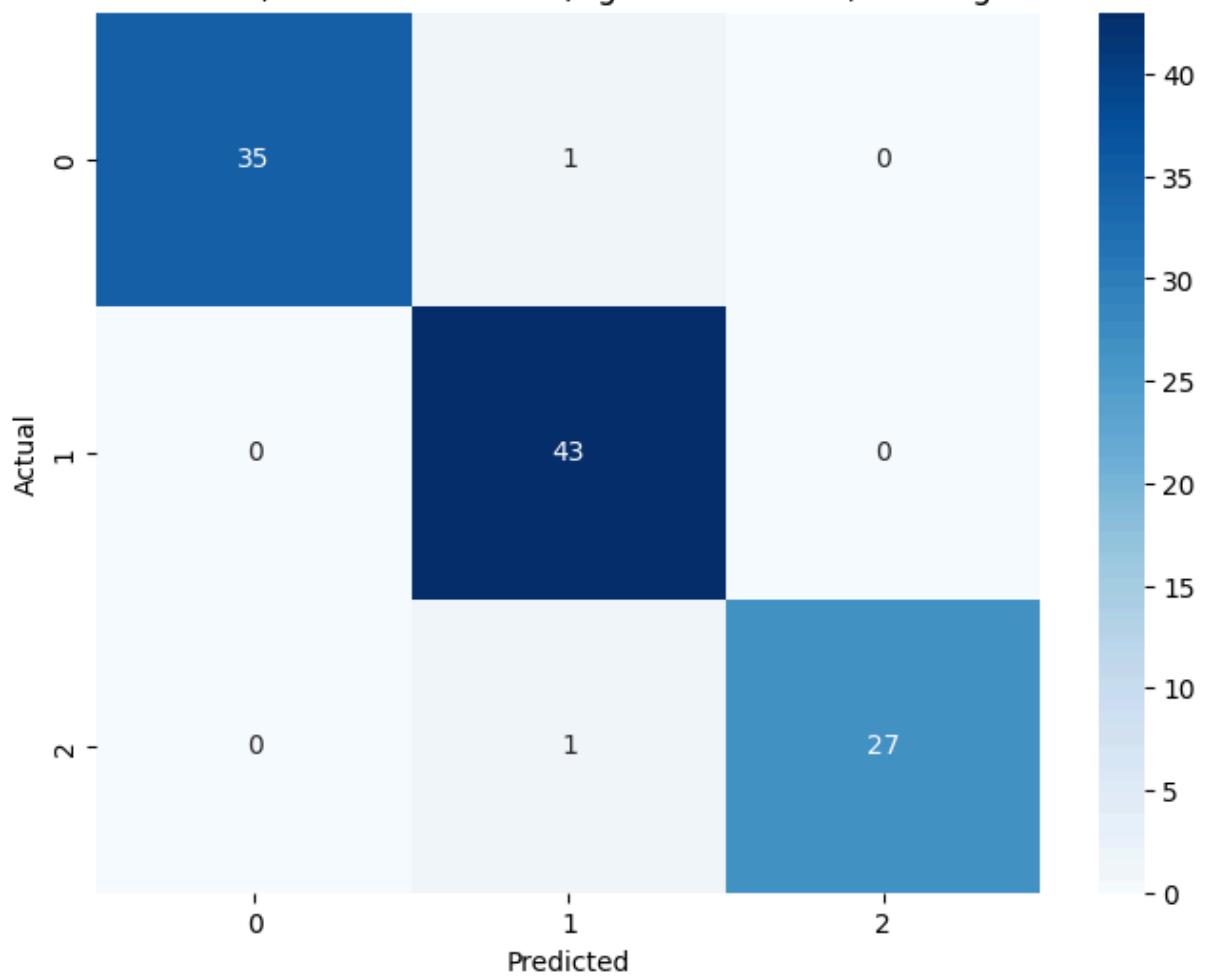
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 80%



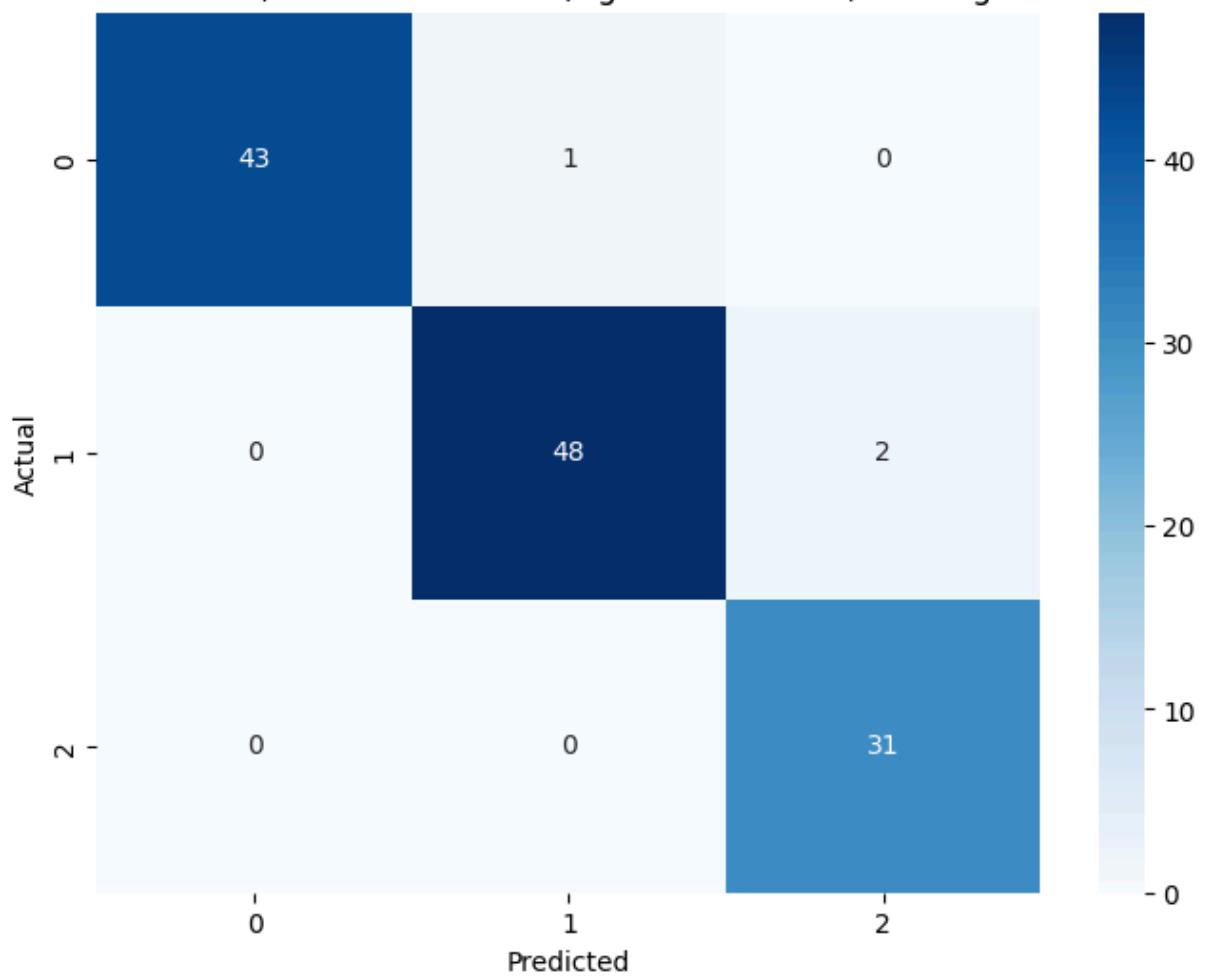
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 50%



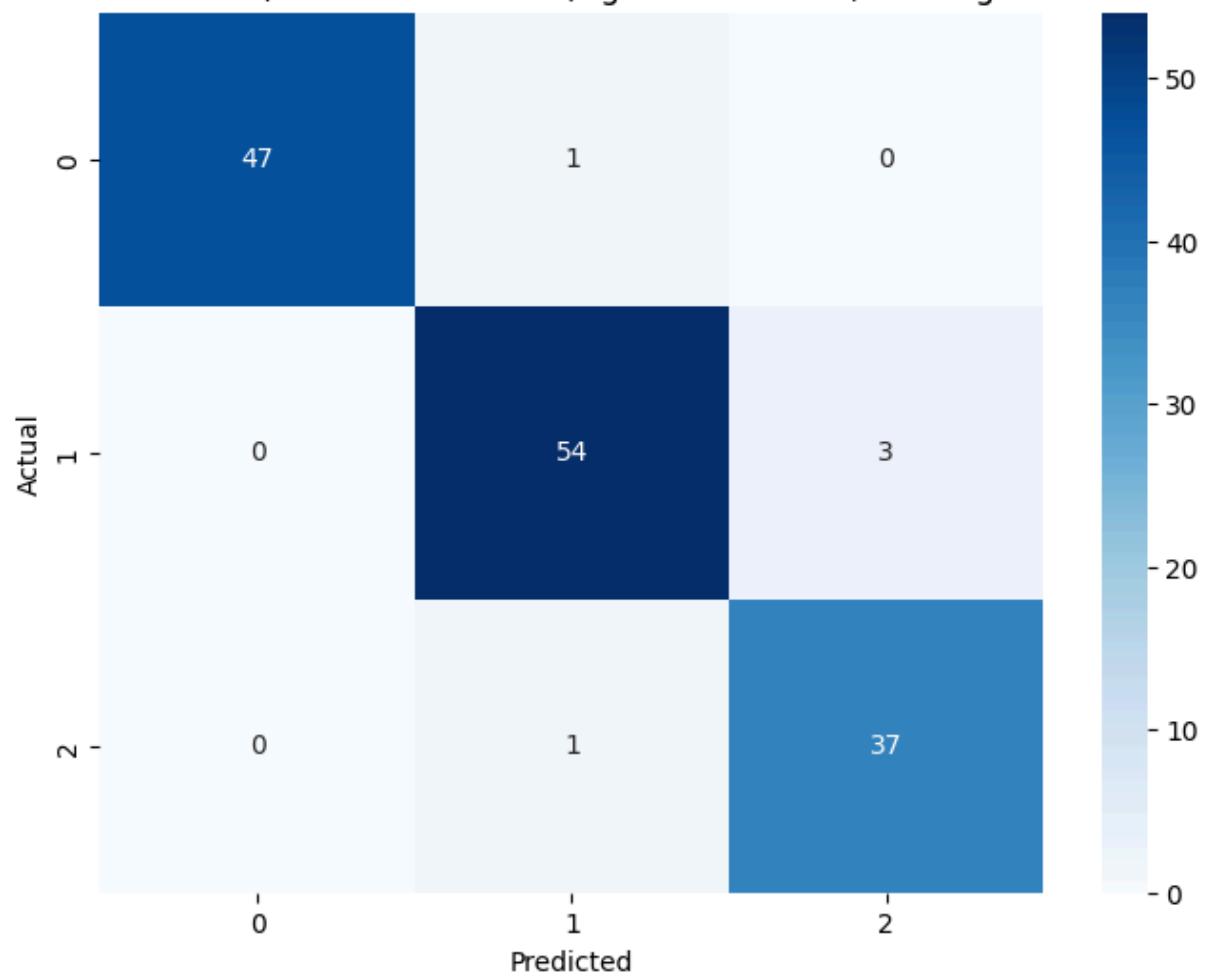
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 60%



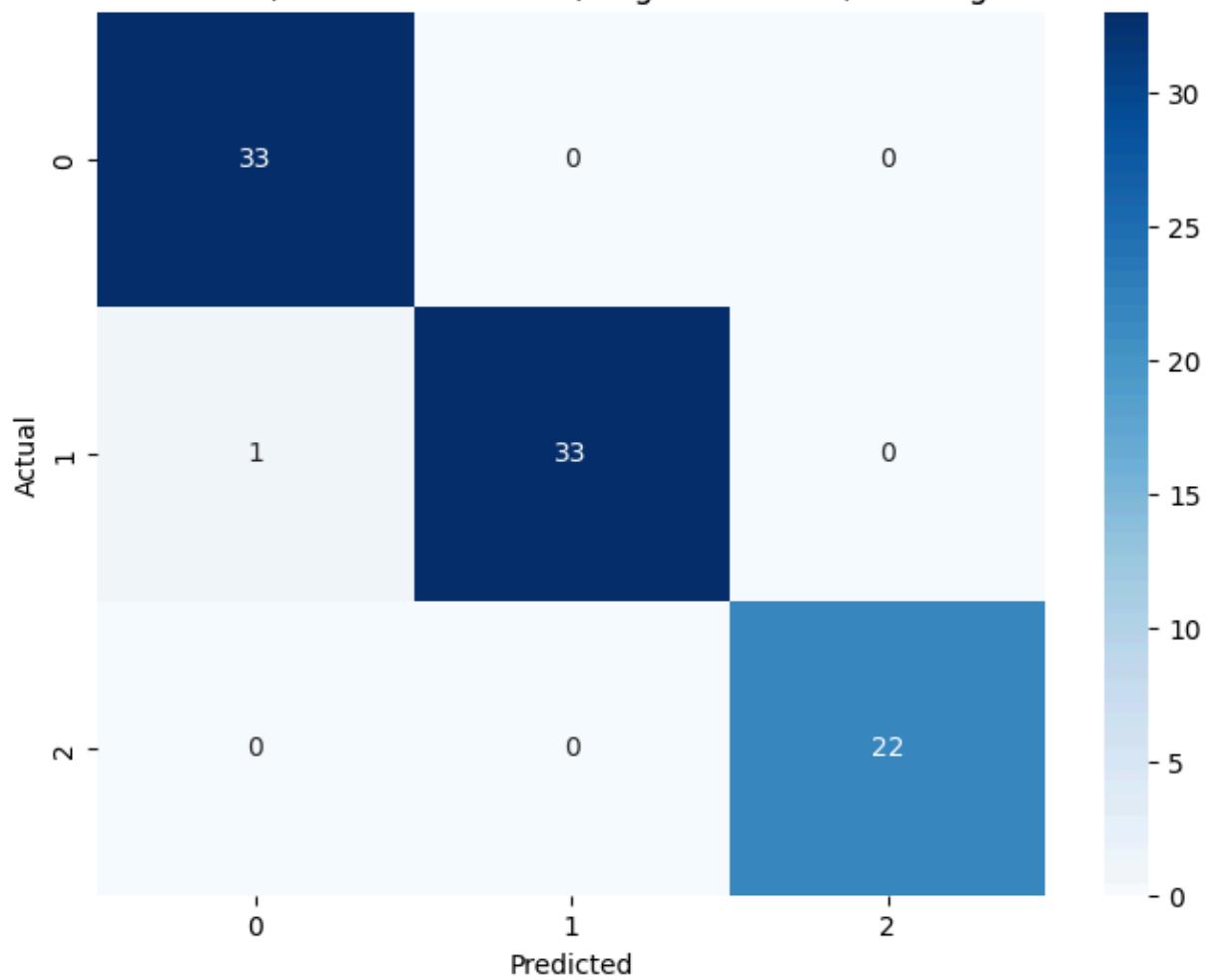
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 70%



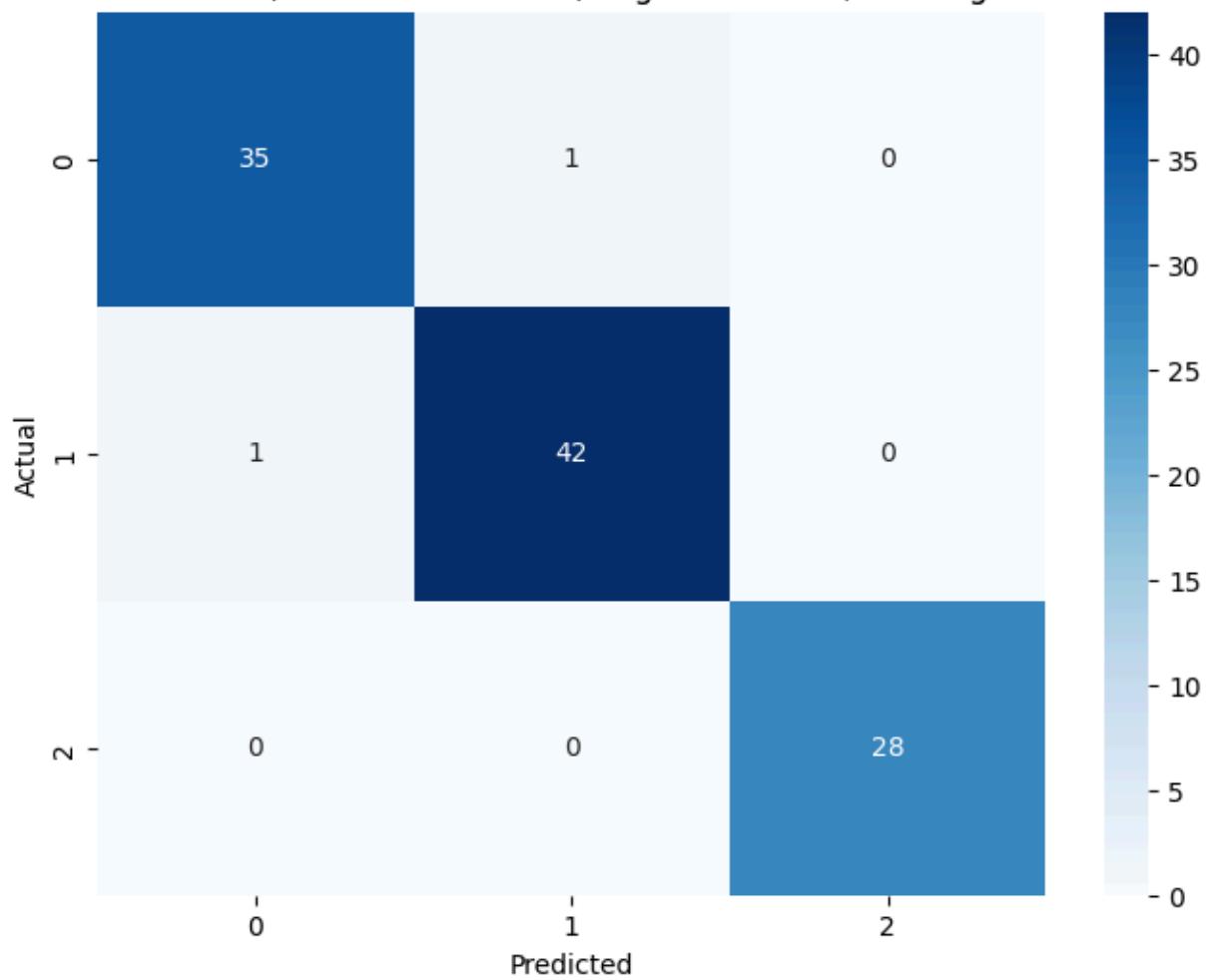
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 80%



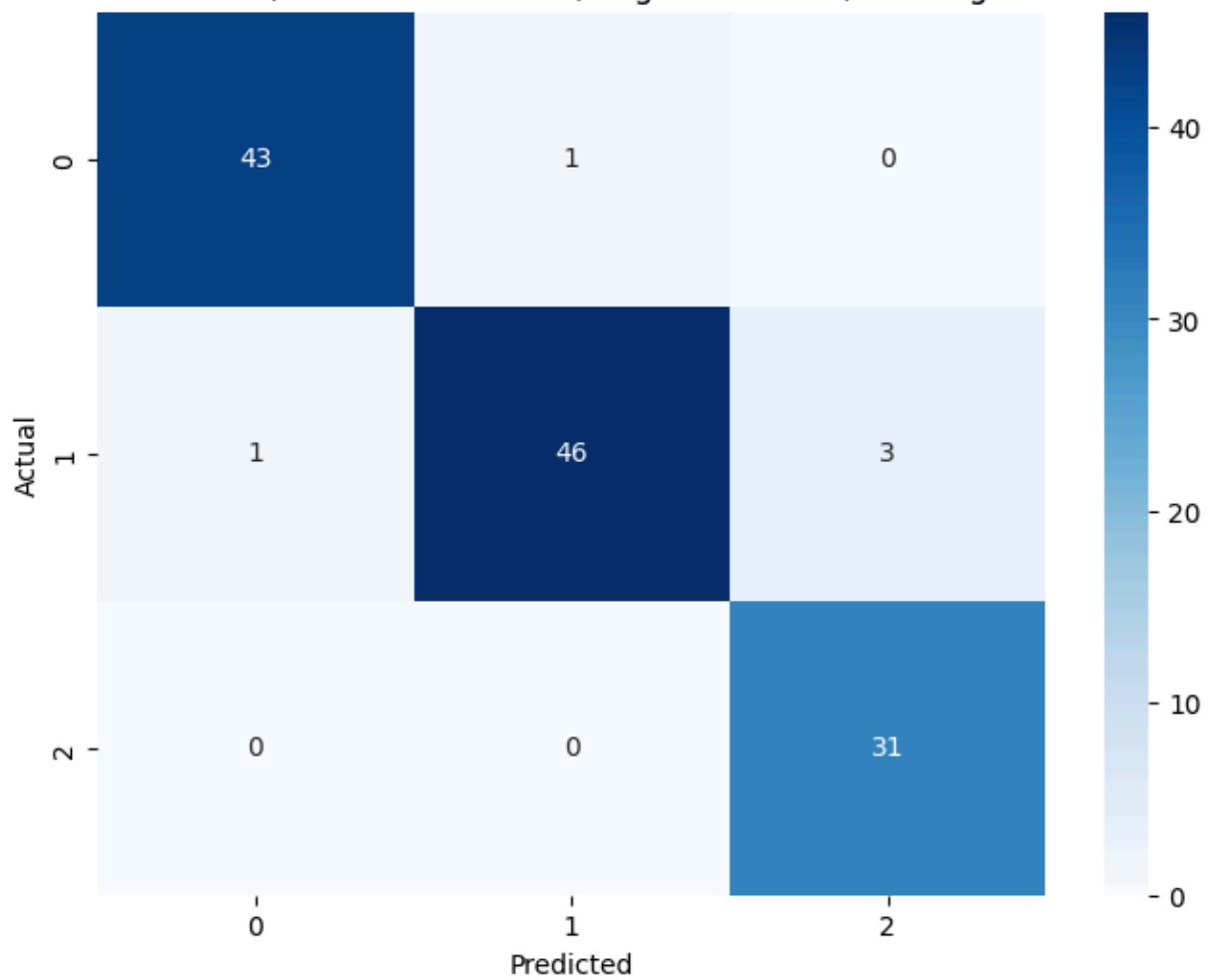
Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 50%



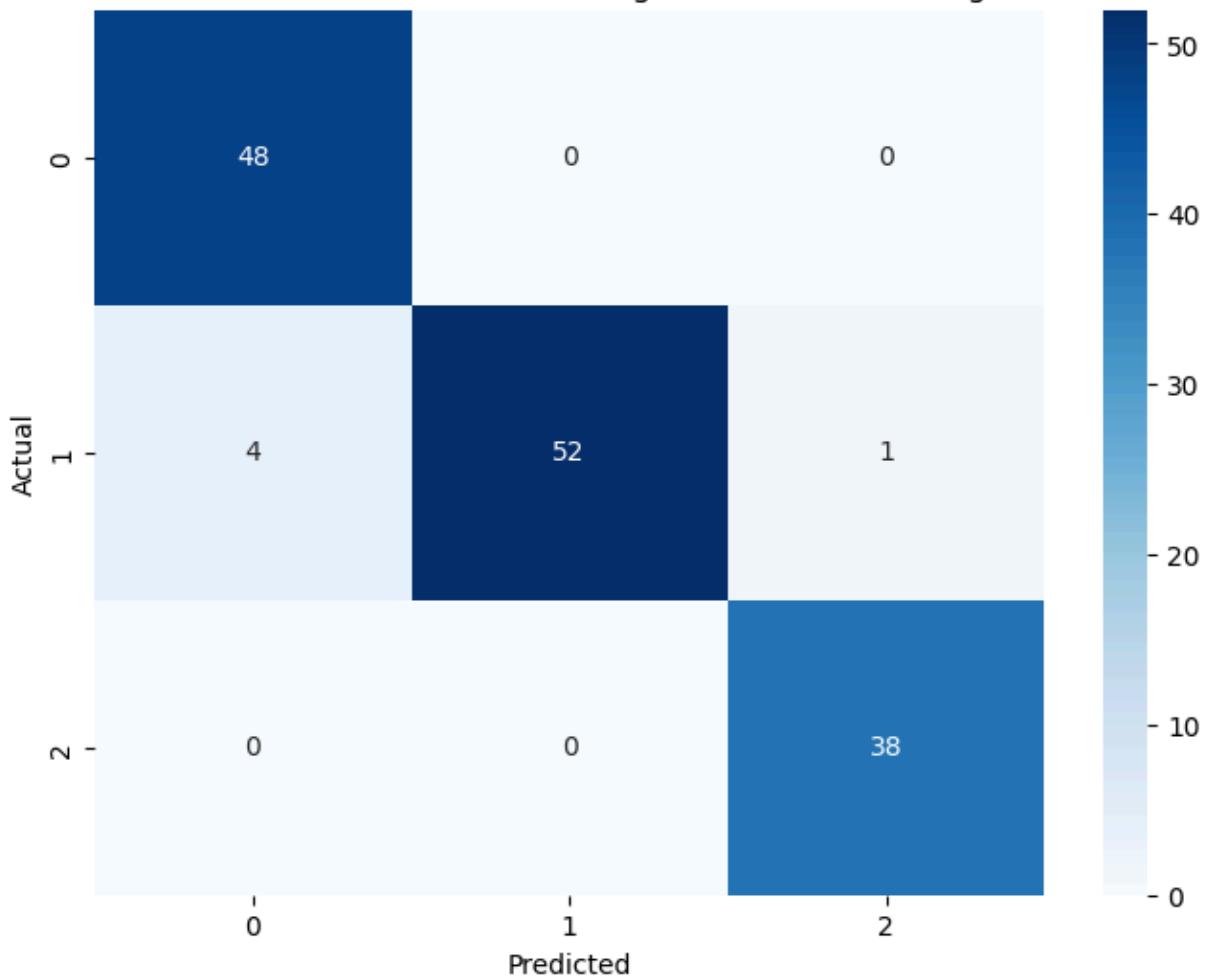
Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 60%



Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 70%



Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 80%



## MLP

Calculate values for different test size

```
In [ ]: from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]

results_mlp = []
confusion_matrices_mlp = []
trained_models_mlp = []

for size in training_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size,
```

```

# Initialize MLPClassifier
# You might need to tune the hyperparameters like hidden_layer_sizes, acti
mlp = MLPClassifier(max_iter=1500, momentum=0.7, learning_rate_init=0.0005

mlp.fit(X_train, y_train.values.ravel())
y_pred = mlp.predict(X_test)

acc = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_divis
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

cm = confusion_matrix(y_test, y_pred)

results_mlp.append({
    "Training size": int(size * 100),
    "Accuracy": acc,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1
})
confusion_matrices_mlp.append({
    "Training size": int(size * 100),
    "Confusion Matrix": cm
})
trained_models_mlp.append({ # Store the trained model with metadata
    "Training size":int (size*100),
    "Model": mlp,
    "X_test": X_test,
    "y_test": y_test
})

```

## Print Evaluation Table and Graphs

```

In [ ]: df = pd.DataFrame(results_mlp)
display(df) # Display the DataFrame as a table

# Plot Accuracy
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy')
plt.title('SVM Accuracy by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

# Plot Precision
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision')
plt.title('SVM Precision by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

```

```

# Plot Recall
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall')
plt.title('SVM Recall by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

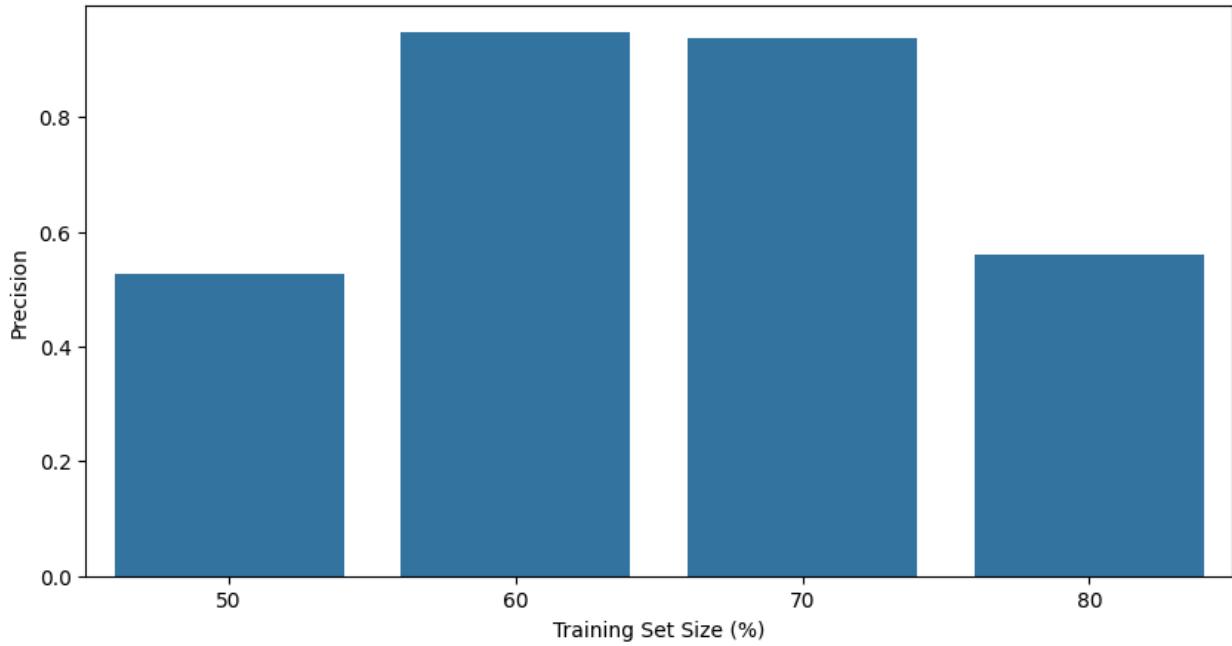
# Plot F1-score
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score')
plt.title('SVM F1-score by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()

```

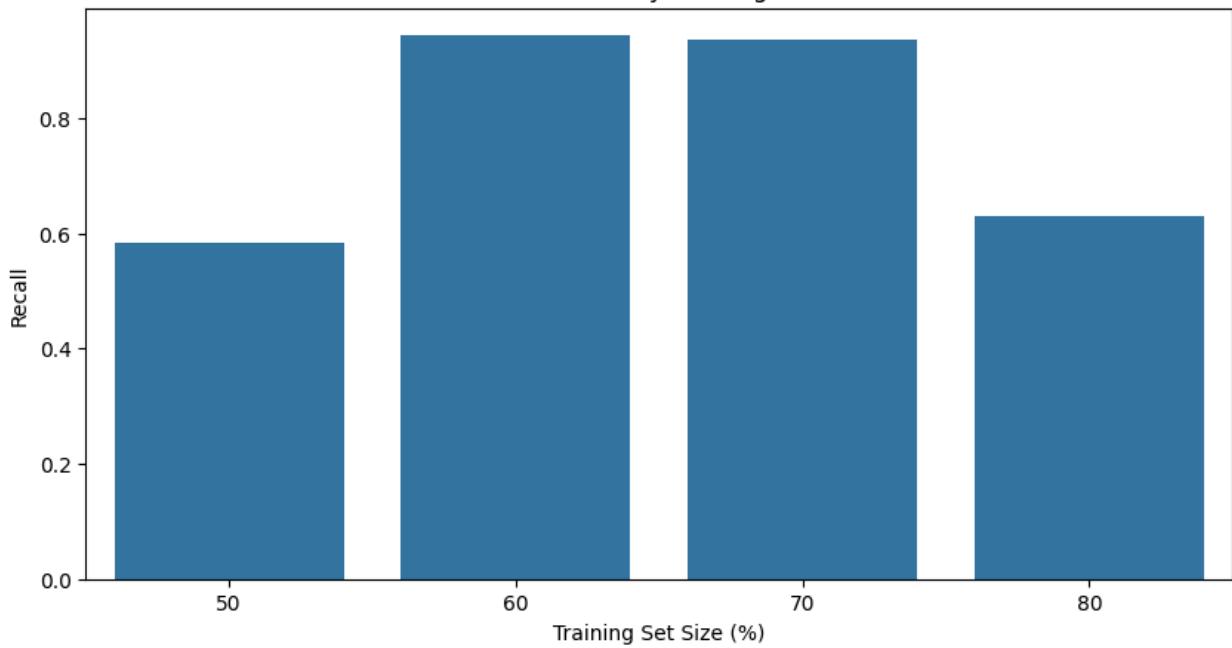
	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.584270	0.525580	0.584270	0.506239
1	60	0.943925	0.947556	0.943925	0.944307
2	70	0.936000	0.937158	0.936000	0.936255
3	80	0.629371	0.559791	0.629371	0.580560



SVM Precision by Training Size



SVM Recall by Training Size





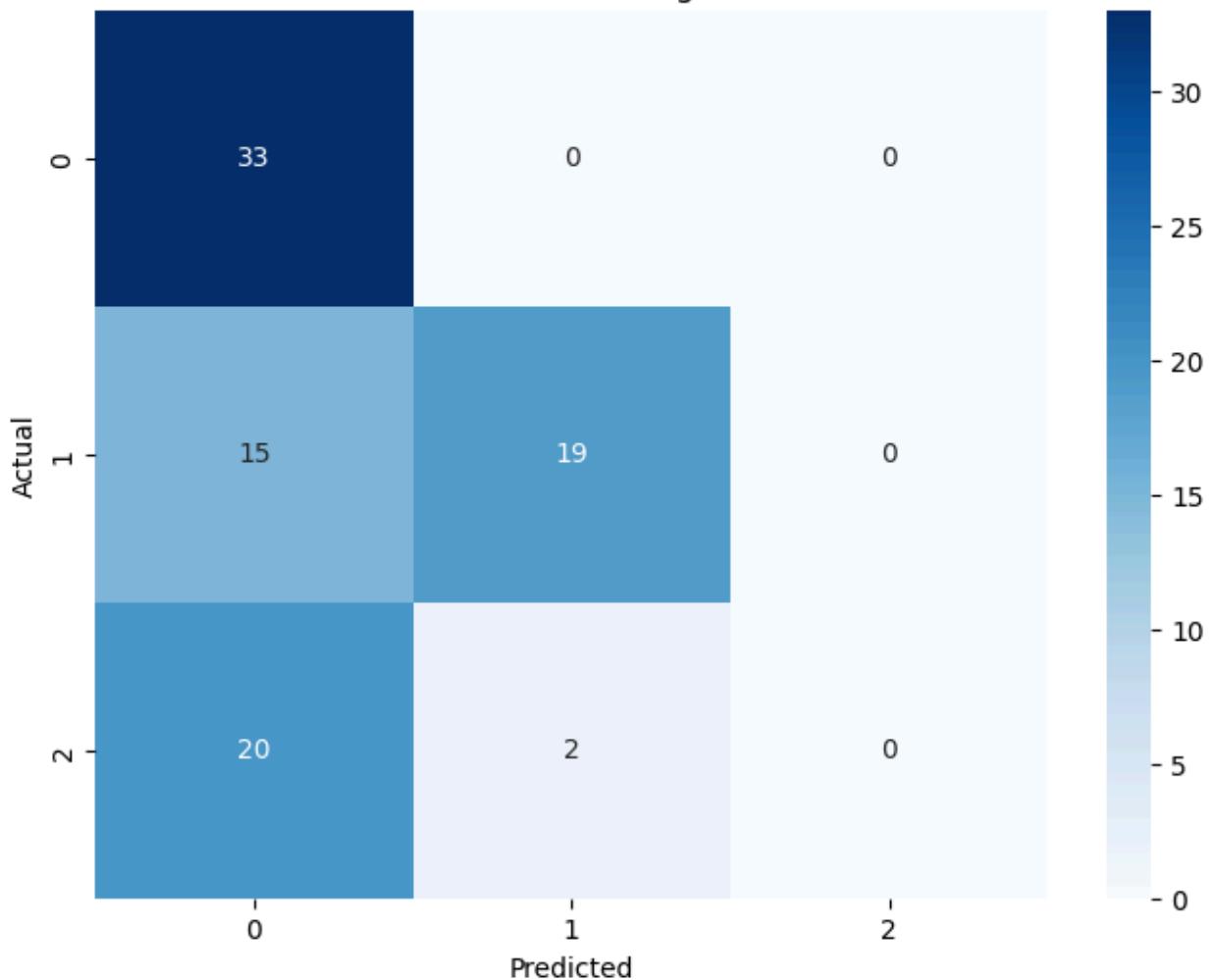
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

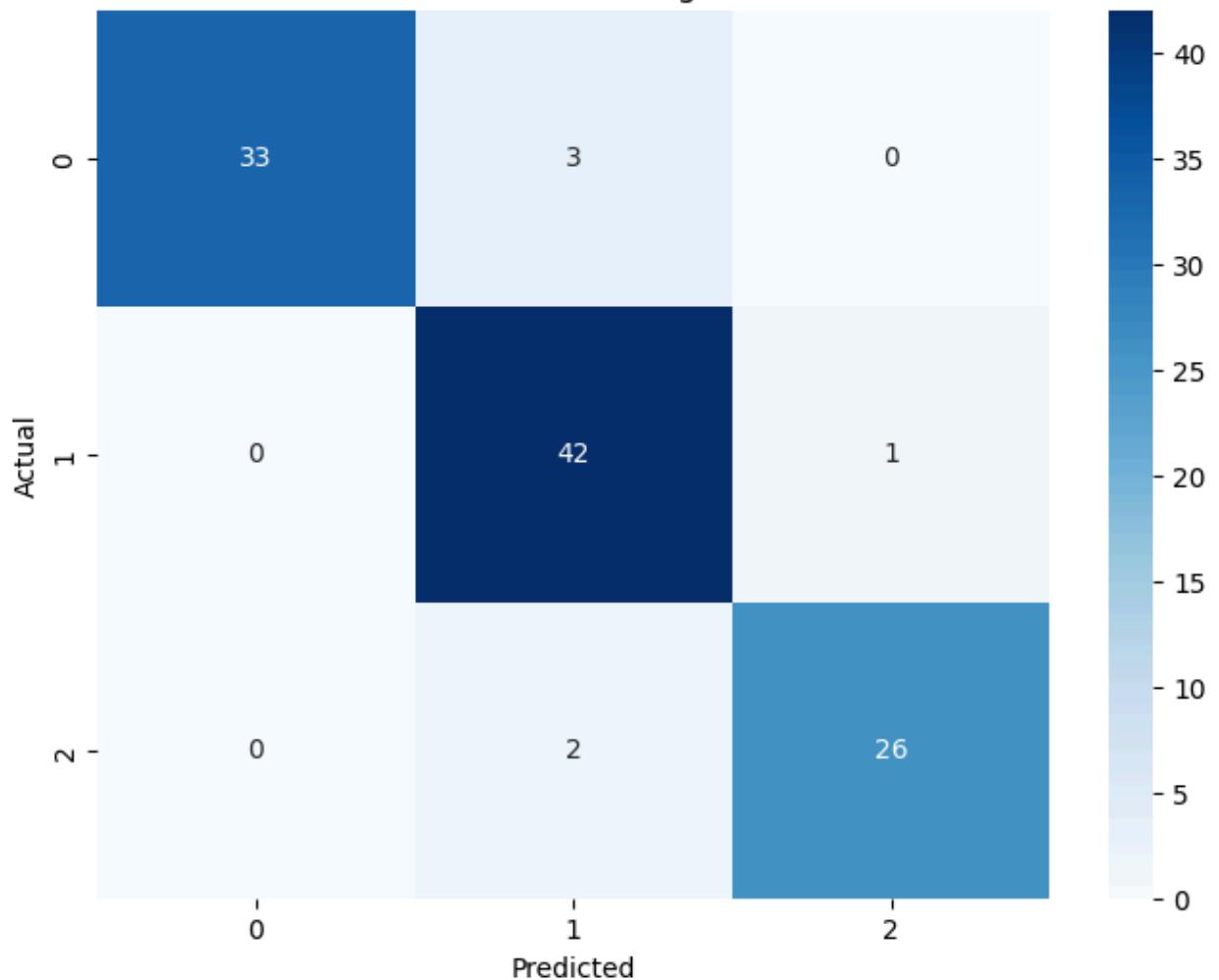
for cm_data in confusion_matrices_mlp:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix: Training Size: {training_size}%')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

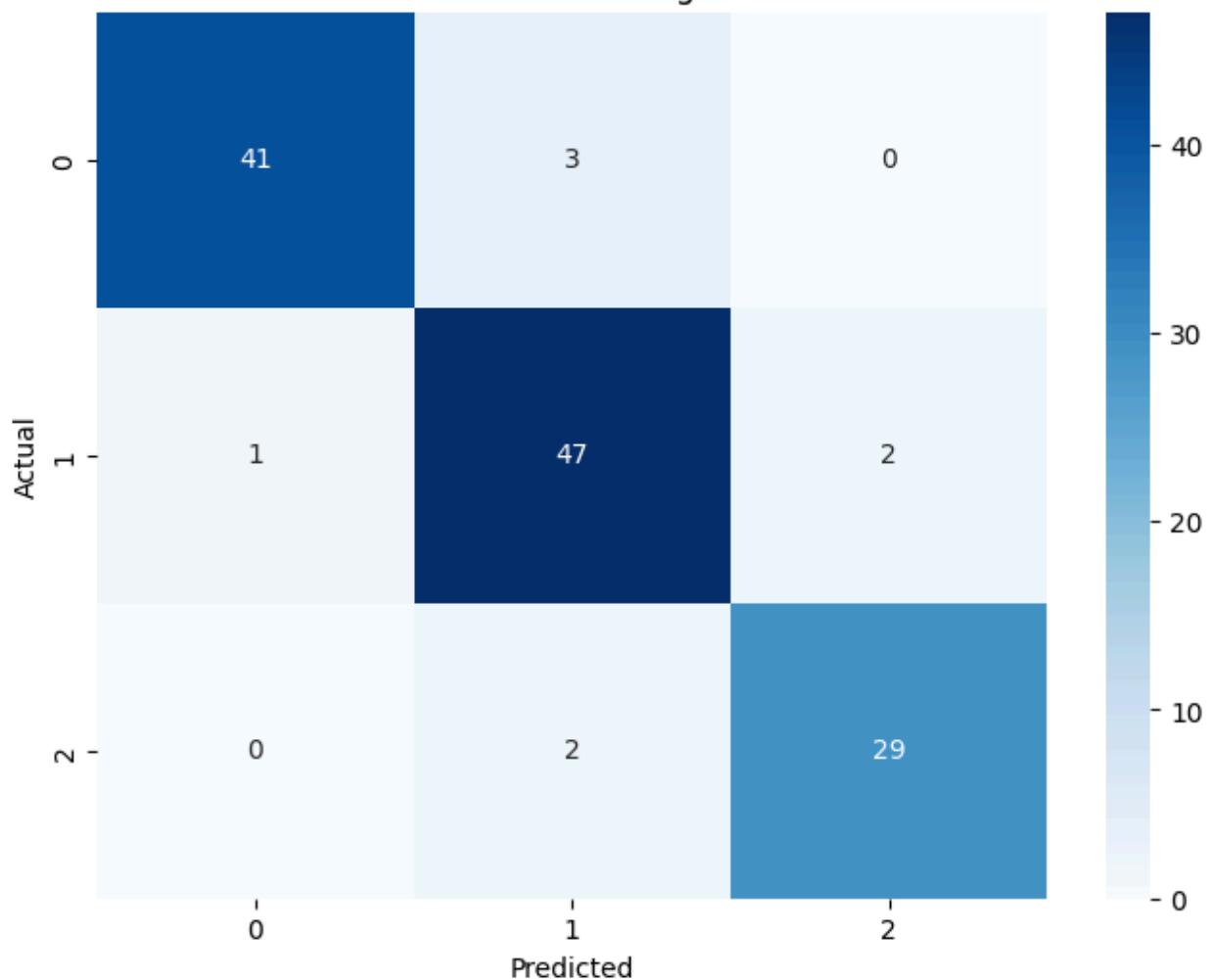
Confusion Matrix: Training Size: 50%

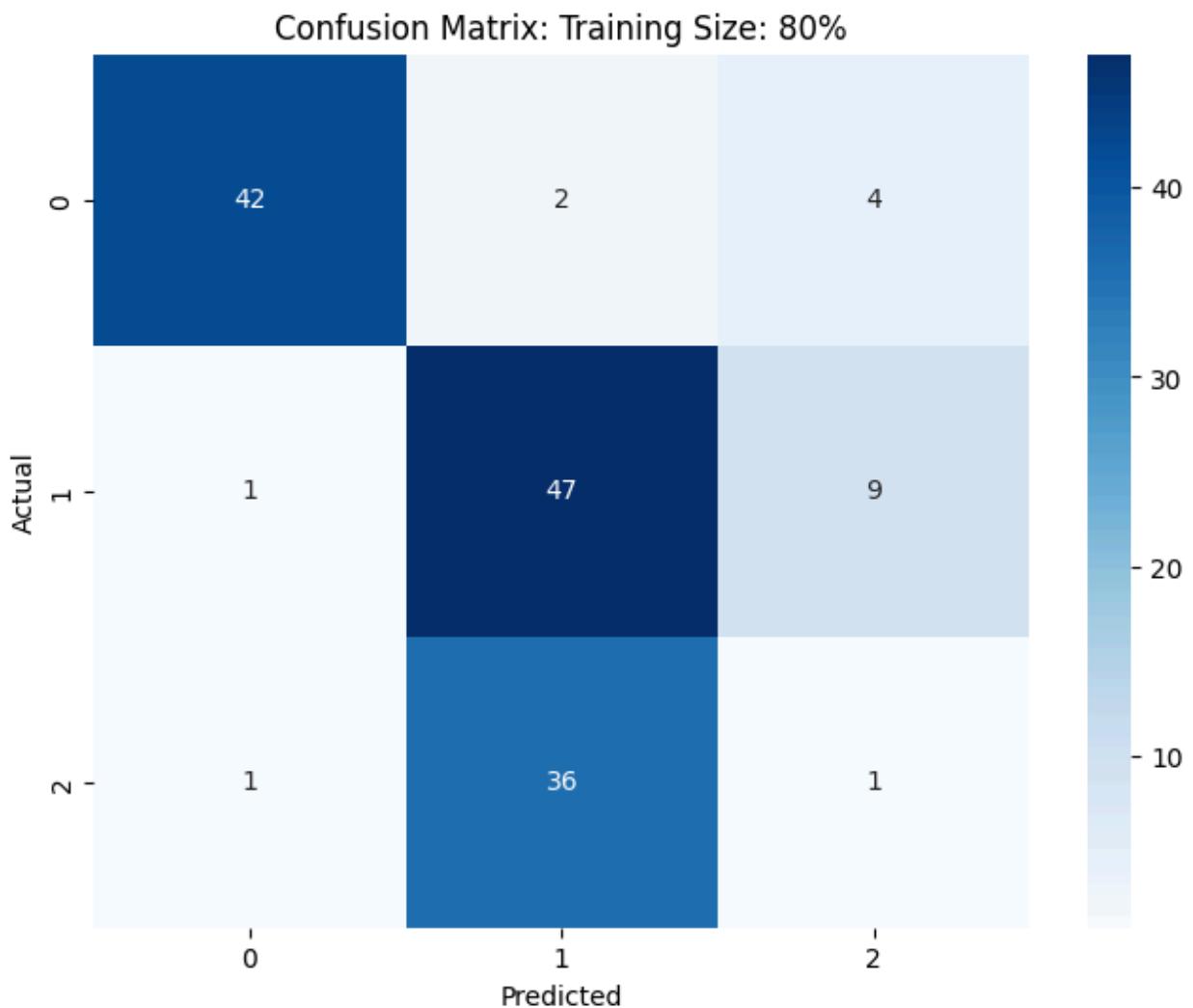


Confusion Matrix: Training Size: 60%



Confusion Matrix: Training Size: 70%



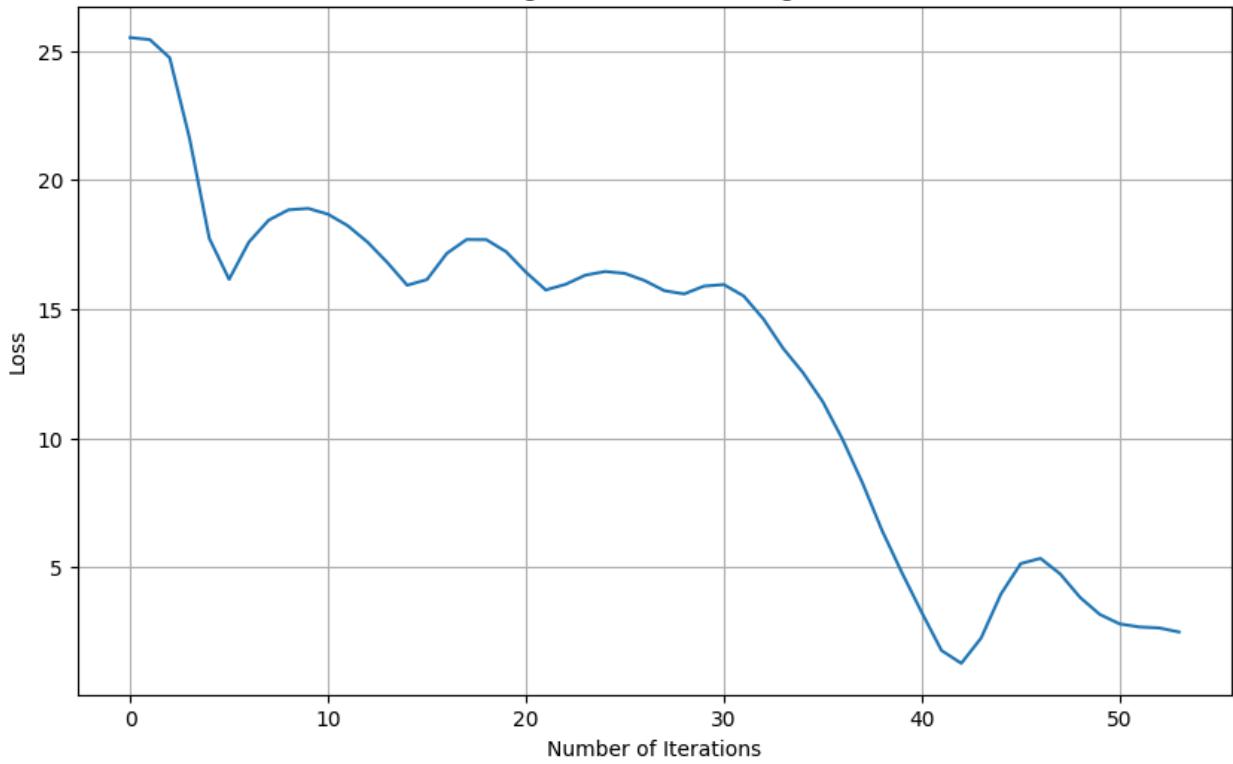


## Training and Loss generation curve

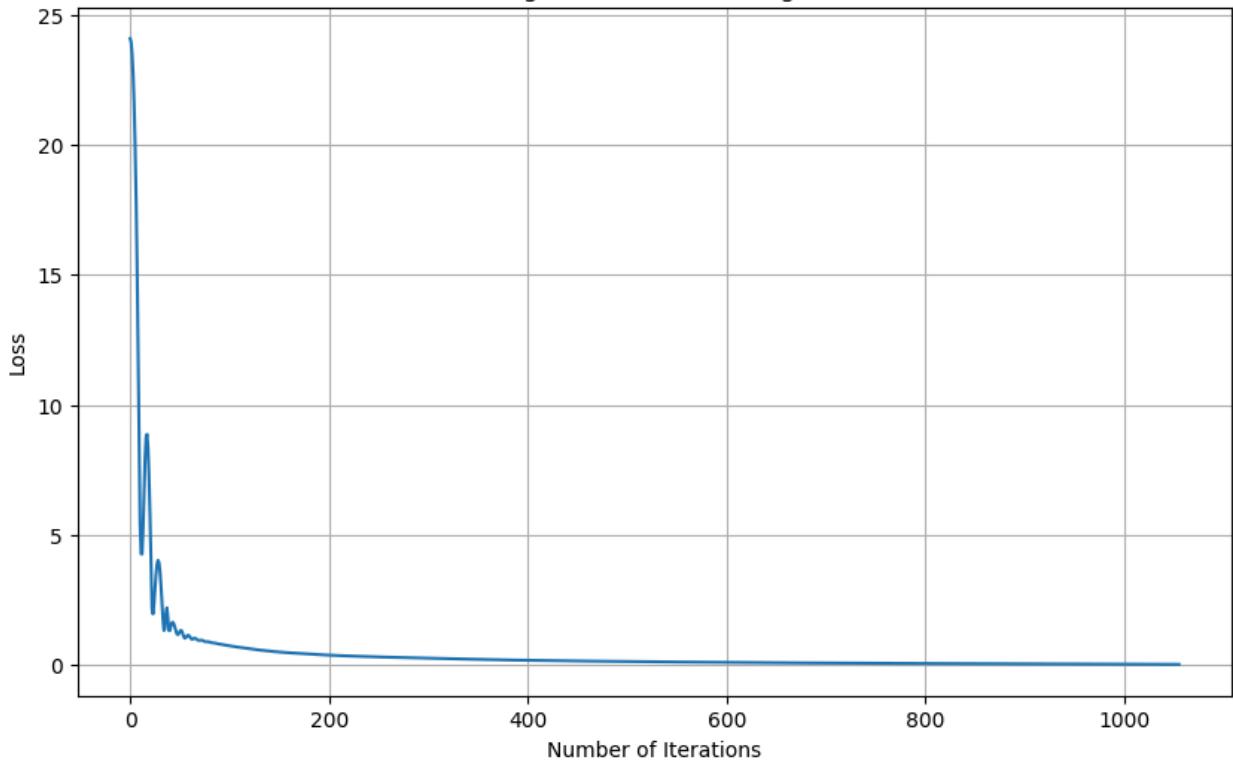
```
In [ ]: # Plot training loss curve for each trained MLP model
for model_data in trained_models_mlp:
    model = model_data["Model"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(10, 6))
    plt.plot(model.loss_curve_)
    plt.title(f'MLP Training Loss Curve: Training Size: {training_size}%')
    plt.xlabel('Number of Iterations')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()
```

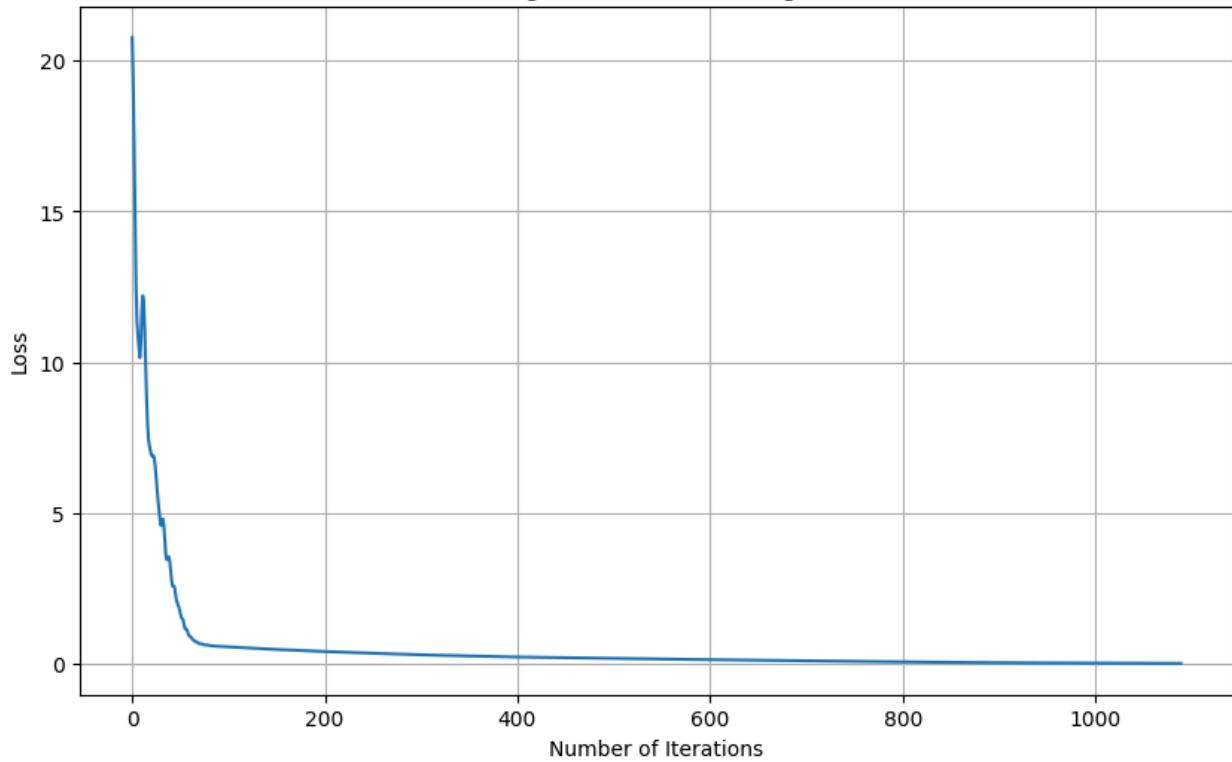
MLP Training Loss Curve: Training Size: 50%



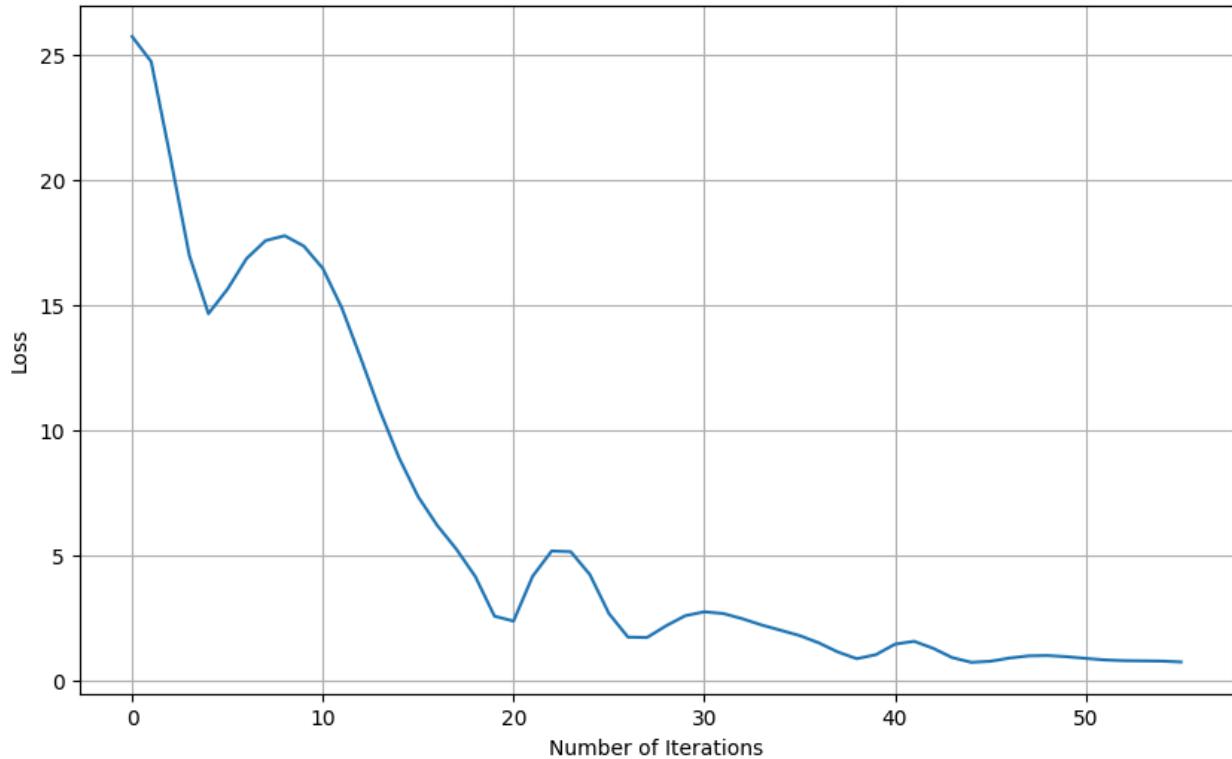
MLP Training Loss Curve: Training Size: 60%



MLP Training Loss Curve: Training Size: 70%



MLP Training Loss Curve: Training Size: 80%



## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc
```

```

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models_mlp:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes = len(np.unique(y_test))

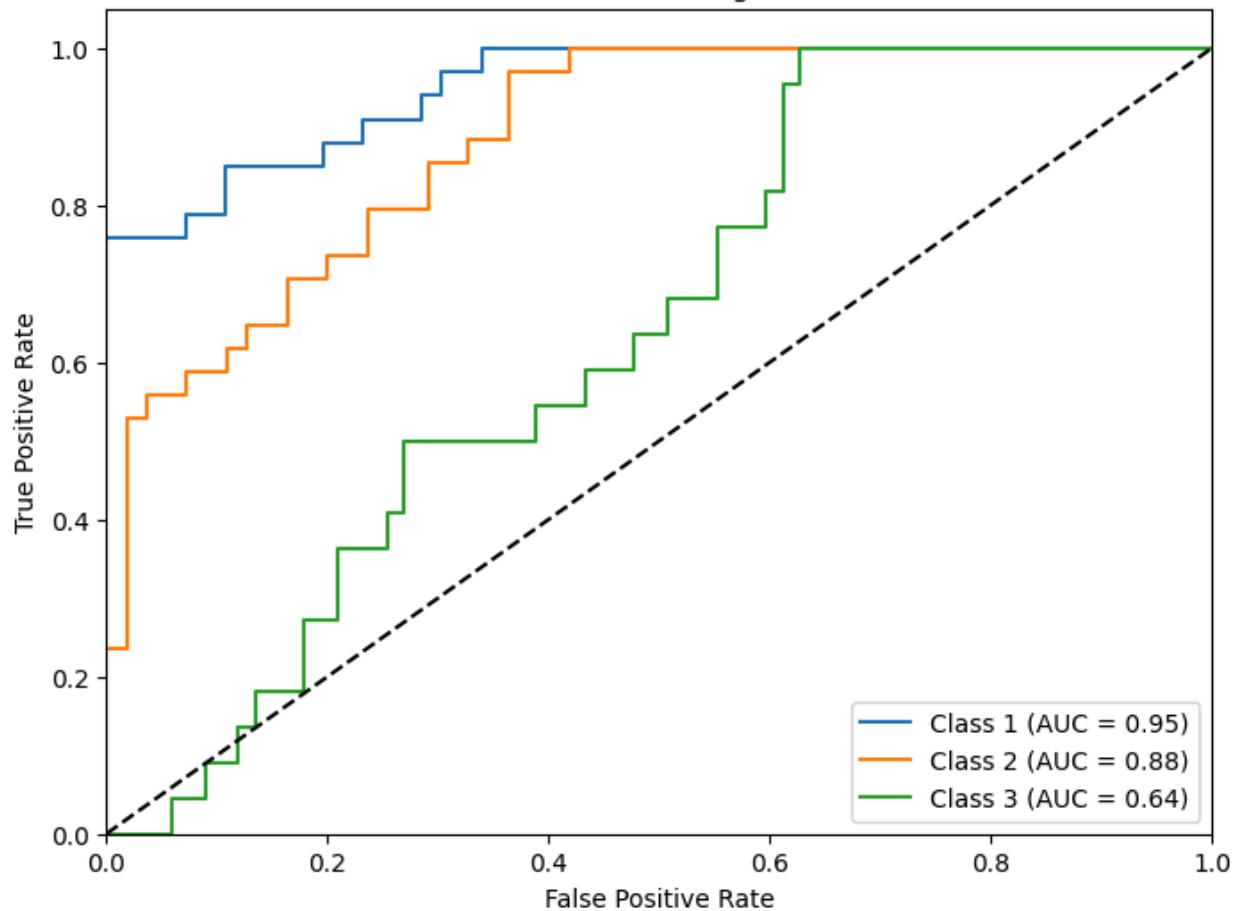
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curve for each class
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

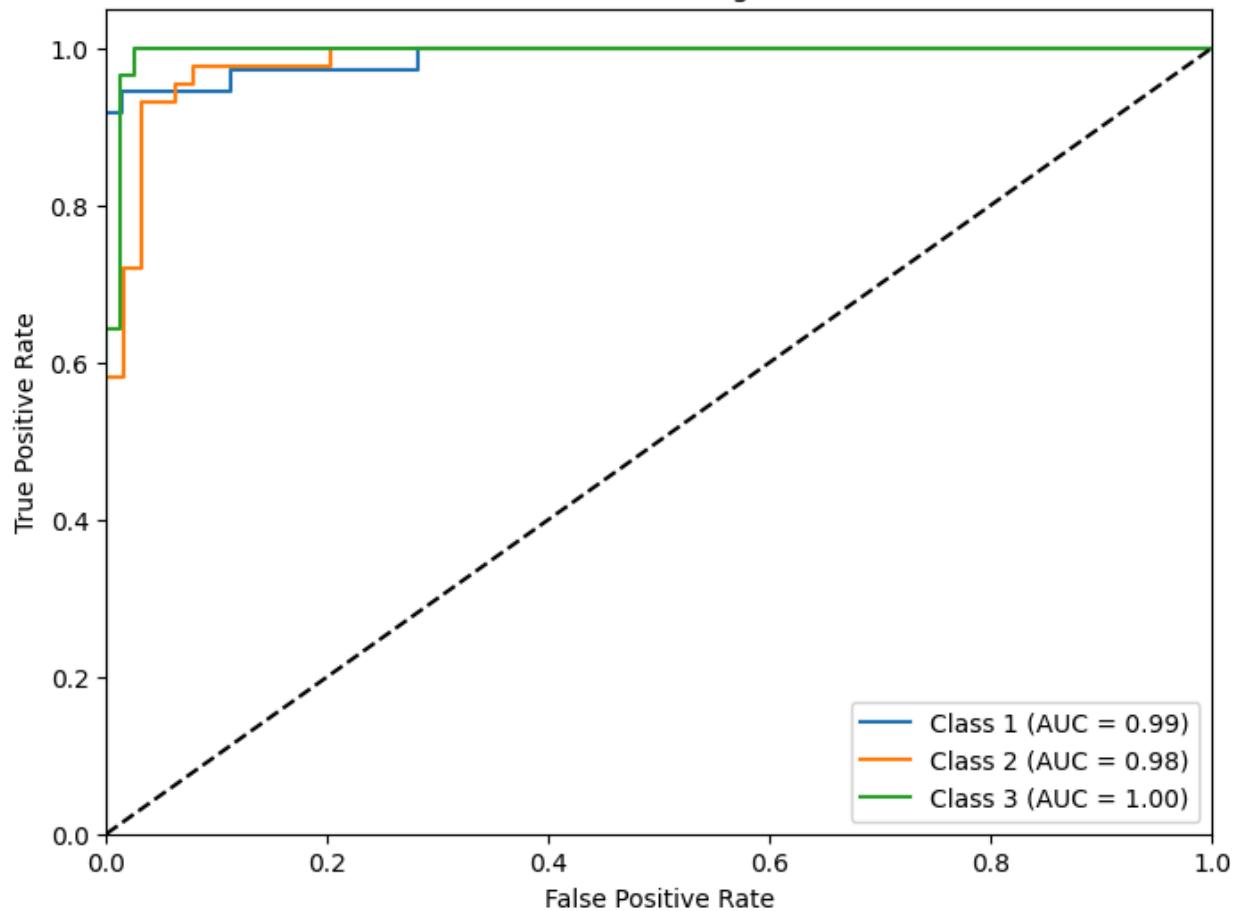
    plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for Training Size: {training_size}%') # Add title for each plot
    plt.legend(loc="lower right")
    plt.show()

```

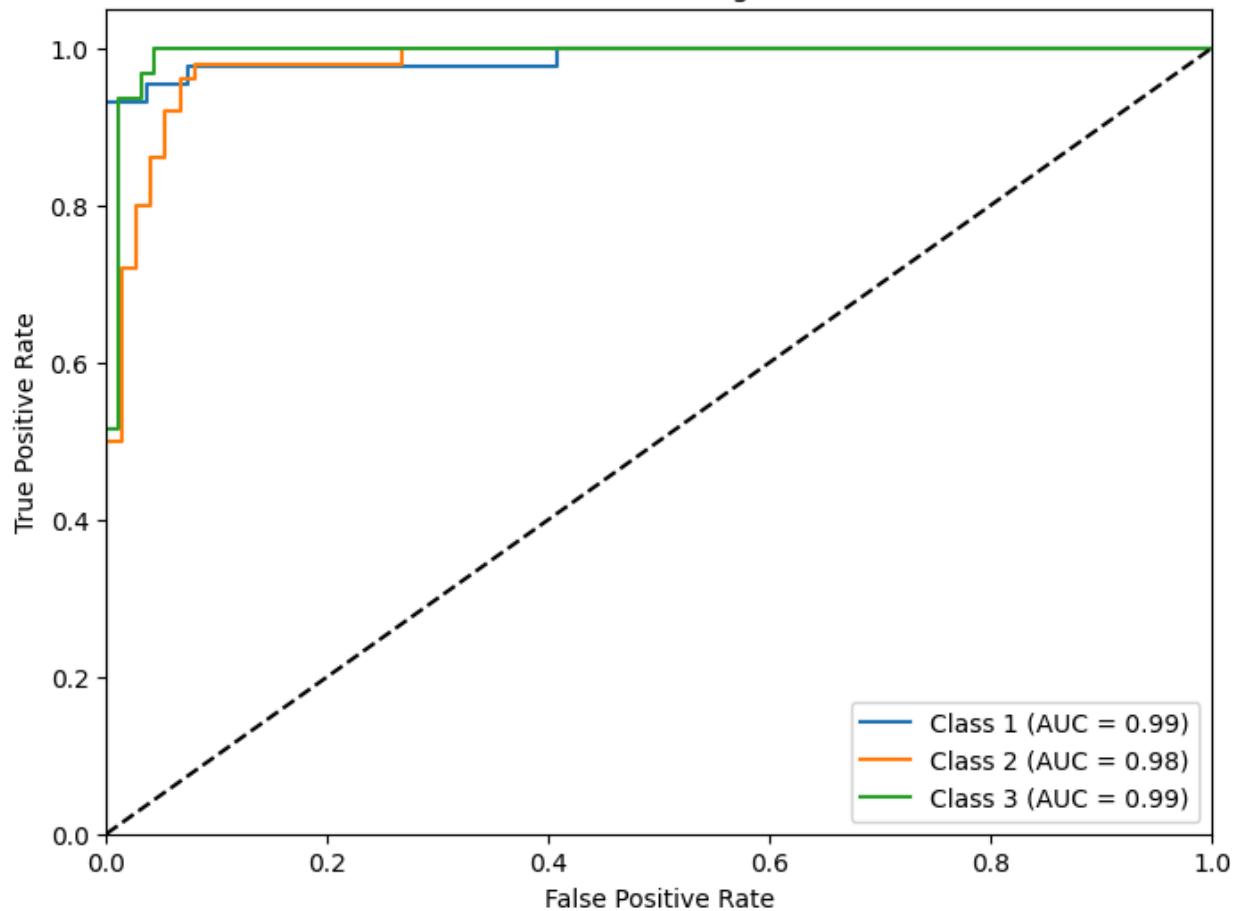
ROC Curve for Training Size: 50%

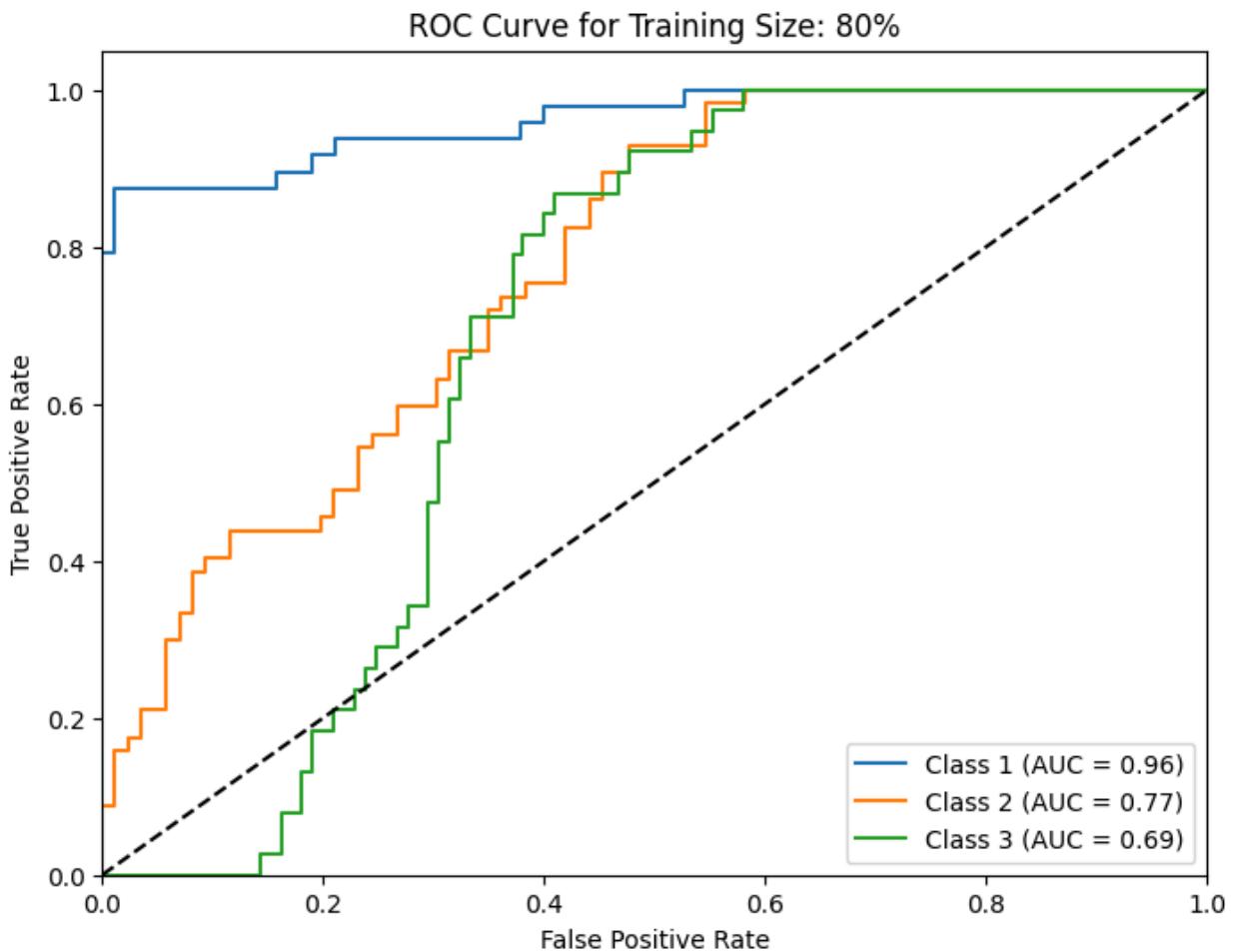


ROC Curve for Training Size: 60%



ROC Curve for Training Size: 70%





## MLP with PCA-Reduced Data

```
In [ ]: # Apply the existing PCA transformation to the scaled data
X_pca_reduced_mlp = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced_mlp.shape[1]}")
print("\nData after PCA for Random Forest:")
display(pd.DataFrame(X_pca_reduced_mlp).head())
```

Original number of features: 13  
 Reduced number of features after PCA: 10

Data after PCA for Random Forest:

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	
<b>0</b>	3.316751	1.443463	-0.165739	-0.215631	0.693043	0.223880	0.596427	-0.065
<b>1</b>	2.209465	-0.333393	-2.026457	-0.291358	-0.257655	0.927120	0.053776	-1.024
<b>2</b>	2.516740	1.031151	0.982819	0.724902	-0.251033	-0.549276	0.424205	0.344
<b>3</b>	3.757066	2.756372	-0.176192	0.567983	-0.311842	-0.114431	-0.383337	-0.643
<b>4</b>	1.008908	0.869831	2.026688	-0.409766	0.298458	0.406520	0.444074	-0.416

## Calculate MLP performance with PCA-Reduced Data for different test sizes

```
In [ ]: results_mlp_pca = []
confusion_matrices_mlp_pca = []
trained_models_mlp_pca = []

for size in training_sizes:
    X_train_pca_mlp, X_test_pca_mlp, y_train, y_test = train_test_split(X_pca_reduced, y, test_size=size/100)
    mlp_pca = MLPClassifier(max_iter=1500, momentum=0.7, learning_rate_init=0.001)
    mlp_pca.fit(X_train_pca_mlp, y_train.values.ravel())
    y_pred_pca_mlp = mlp_pca.predict(X_test_pca_mlp)

    acc_pca_mlp = accuracy_score(y_test, y_pred_pca_mlp)
    precision_pca_mlp = precision_score(y_test, y_pred_pca_mlp, average='weighted')
    recall_pca_mlp = recall_score(y_test, y_pred_pca_mlp, average='weighted')
    f1_pca_mlp = f1_score(y_test, y_pred_pca_mlp, average='weighted')

    cm_pca_mlp = confusion_matrix(y_test, y_pred_pca_mlp)

    results_mlp_pca.append({
        "Training size":int (size*100),
        "Accuracy":acc_pca_mlp,
        "Precision": precision_pca_mlp,
        "Recall": recall_pca_mlp,
        "F1-score": f1_pca_mlp
    })

    confusion_matrices_mlp_pca.append({
        "Training size":int (size*100),
        "Confusion Matrix":cm_pca_mlp
    })

    trained_models_mlp_pca.append({
        "Training size":int (size*100),
        "Model": mlp_pca
    })
```

## Print MLP Performance Table and Graphs with PCA-Reduced Data

```
In [ ]: df_mlp_pca = pd.DataFrame(results_mlp_pca)
display(df_mlp_pca)

# Plot Accuracy
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Accuracy')
plt.title('Random Forest Accuracy by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.show()

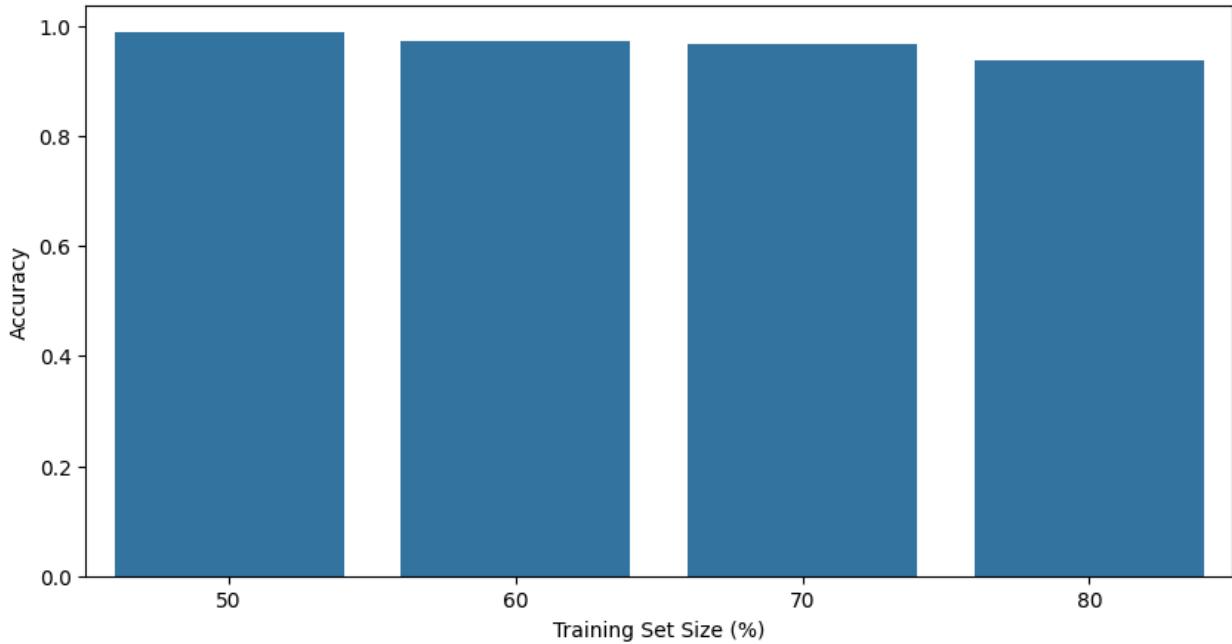
# Plot Precision
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Precision')
plt.title('Random Forest Precision by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Precision')
plt.show()

# Plot Recall
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Recall')
plt.title('Random Forest Recall by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Recall')
plt.show()

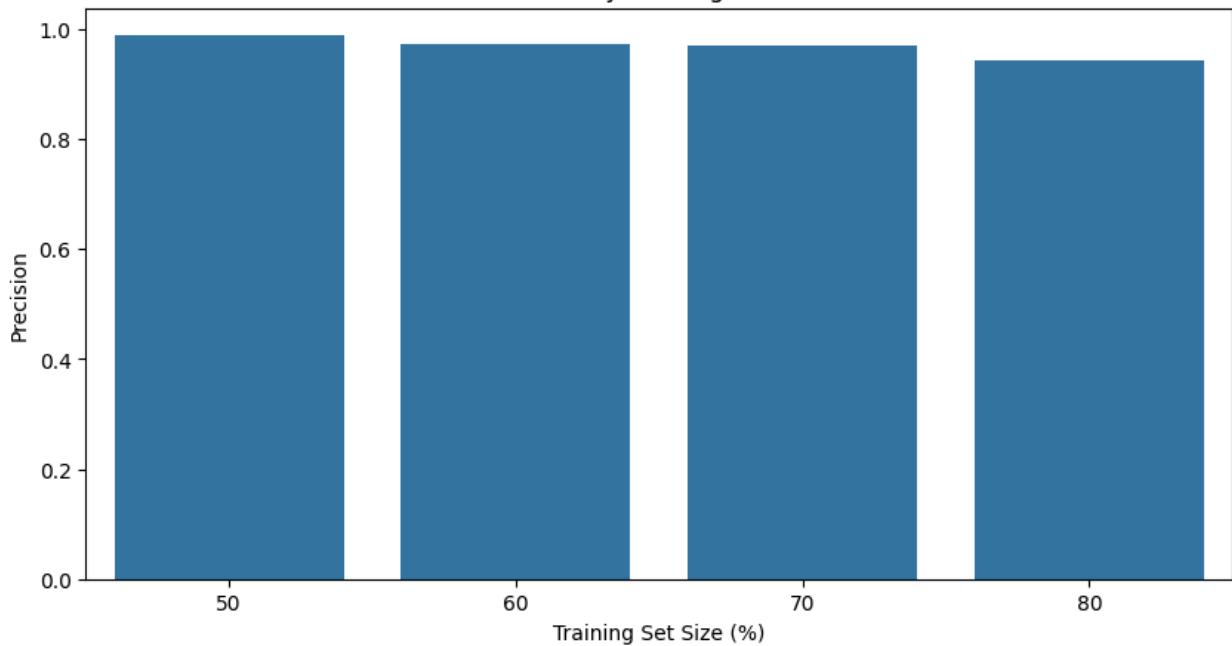
# Plot F1-score
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='F1-score')
plt.title('Random Forest F1-score by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('F1-score')
plt.show()
```

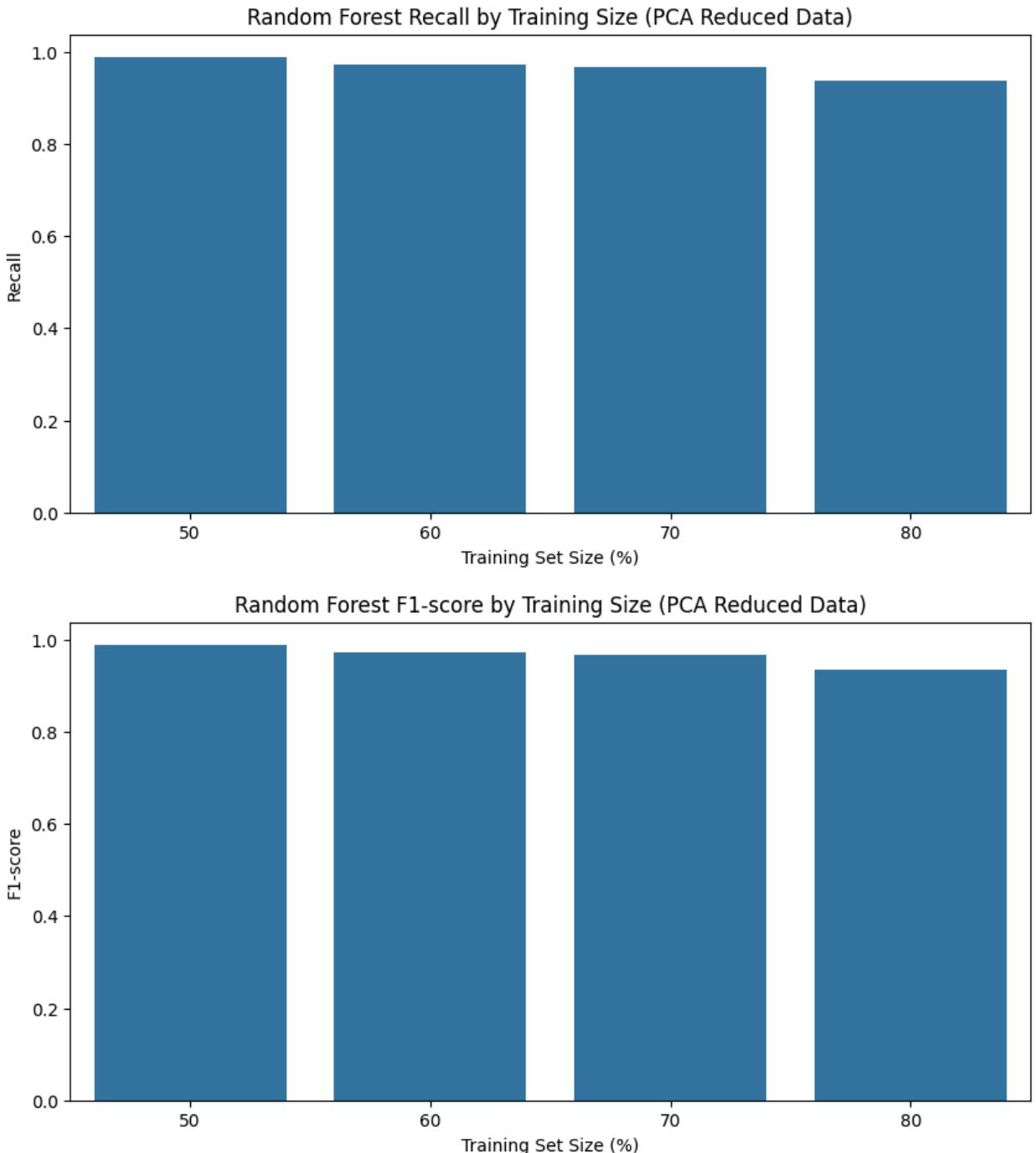
	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.988764	0.989085	0.988764	0.988759
1	60	0.971963	0.972986	0.971963	0.972054
2	70	0.968000	0.969784	0.968000	0.968181
3	80	0.937063	0.943280	0.937063	0.936004

Random Forest Accuracy by Training Size (PCA Reduced Data)



Random Forest Precision by Training Size (PCA Reduced Data)





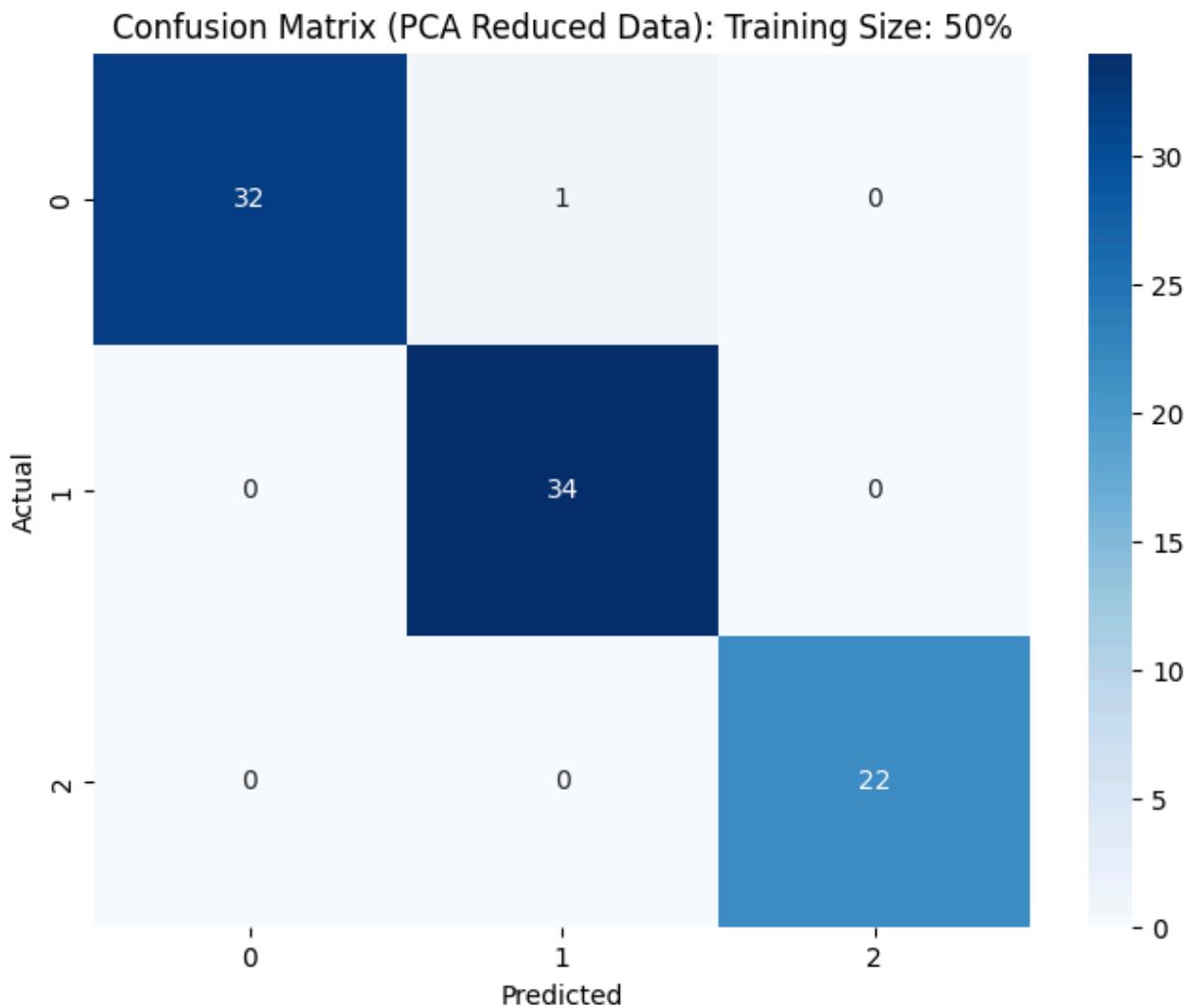
## Show Confusion Matrices for MLP with PCA-Reduced Data

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

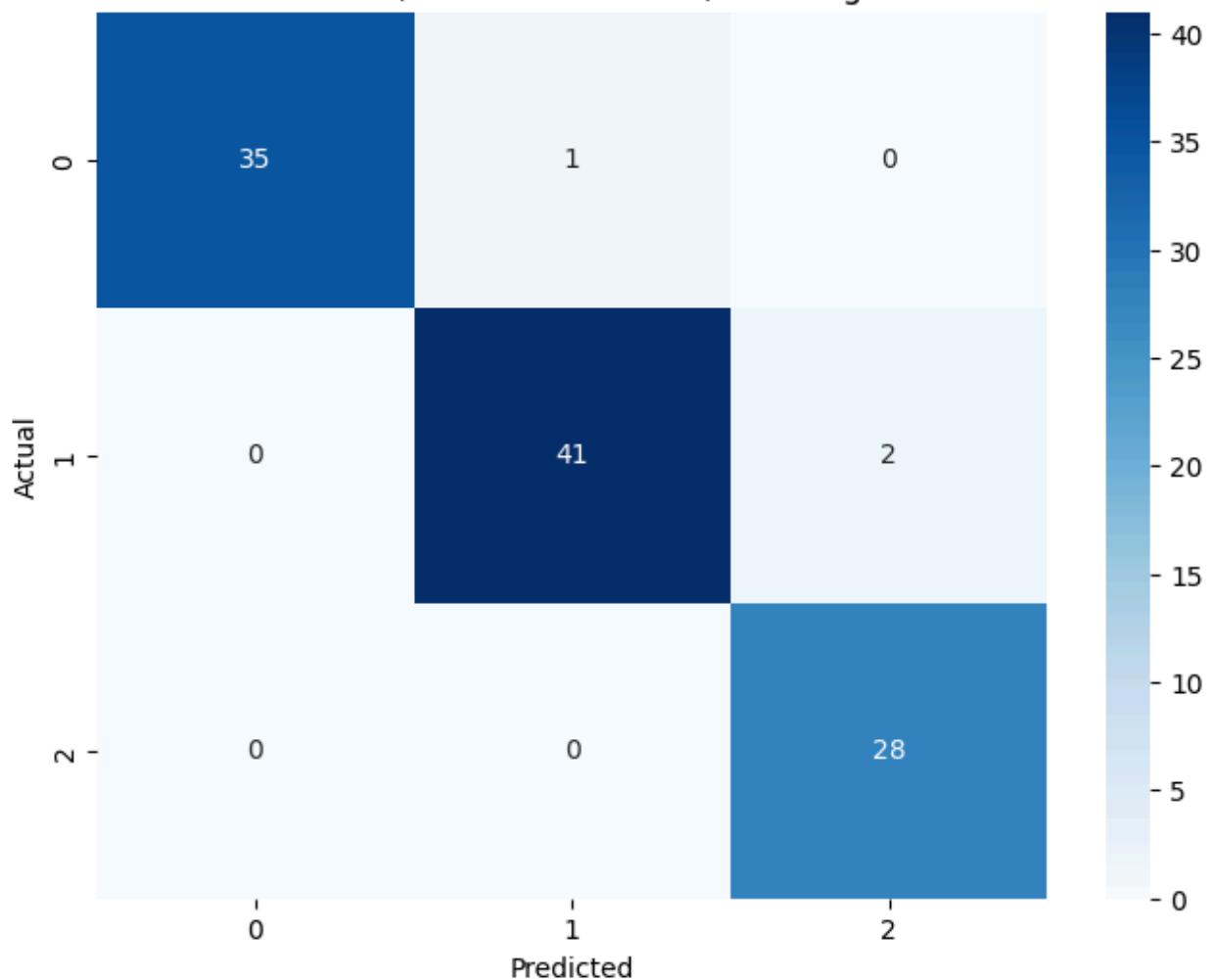
for cm_data in confusion_matrices_mlp_pca:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
```

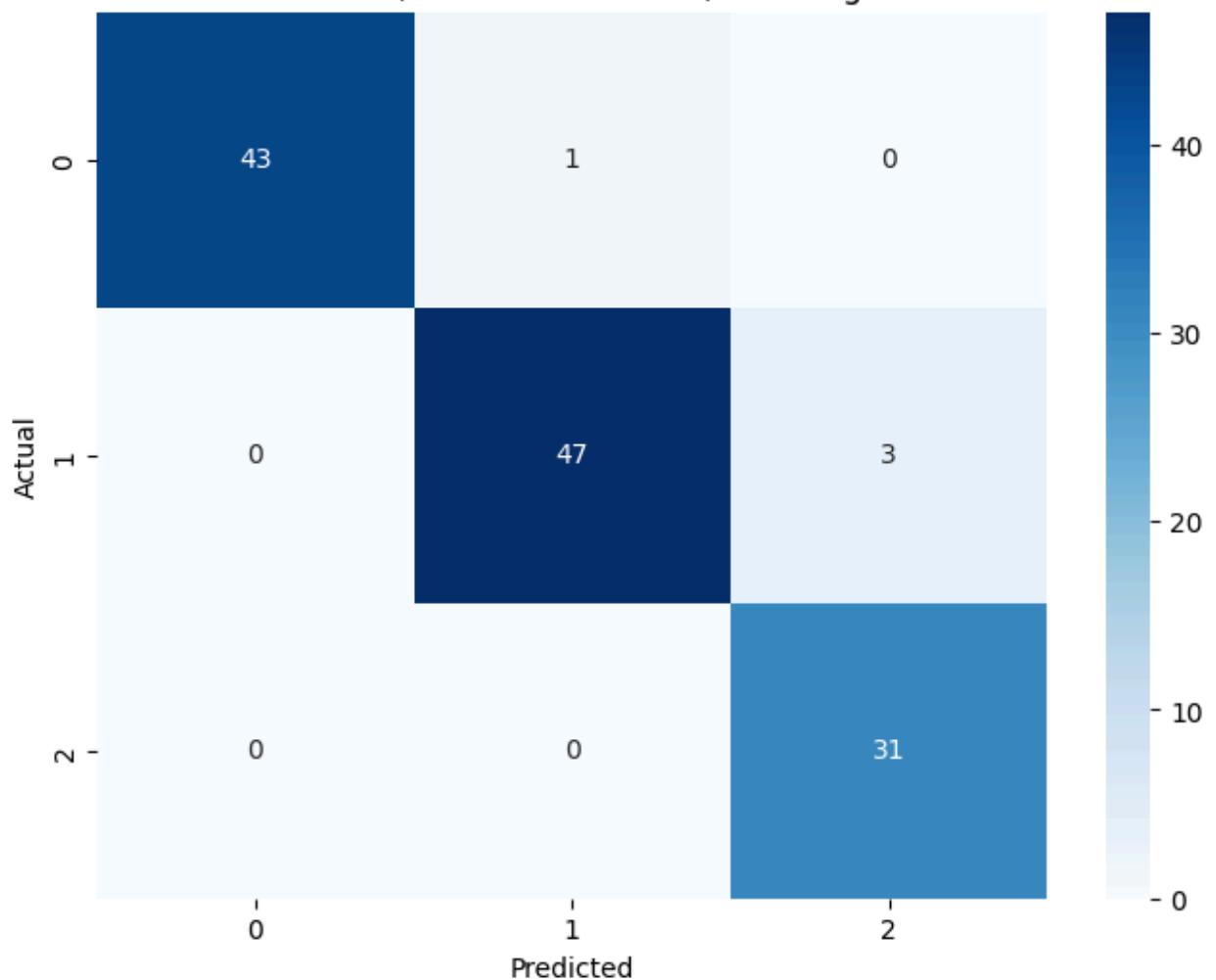
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix (PCA Reduced Data): Training Size: {training_size}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

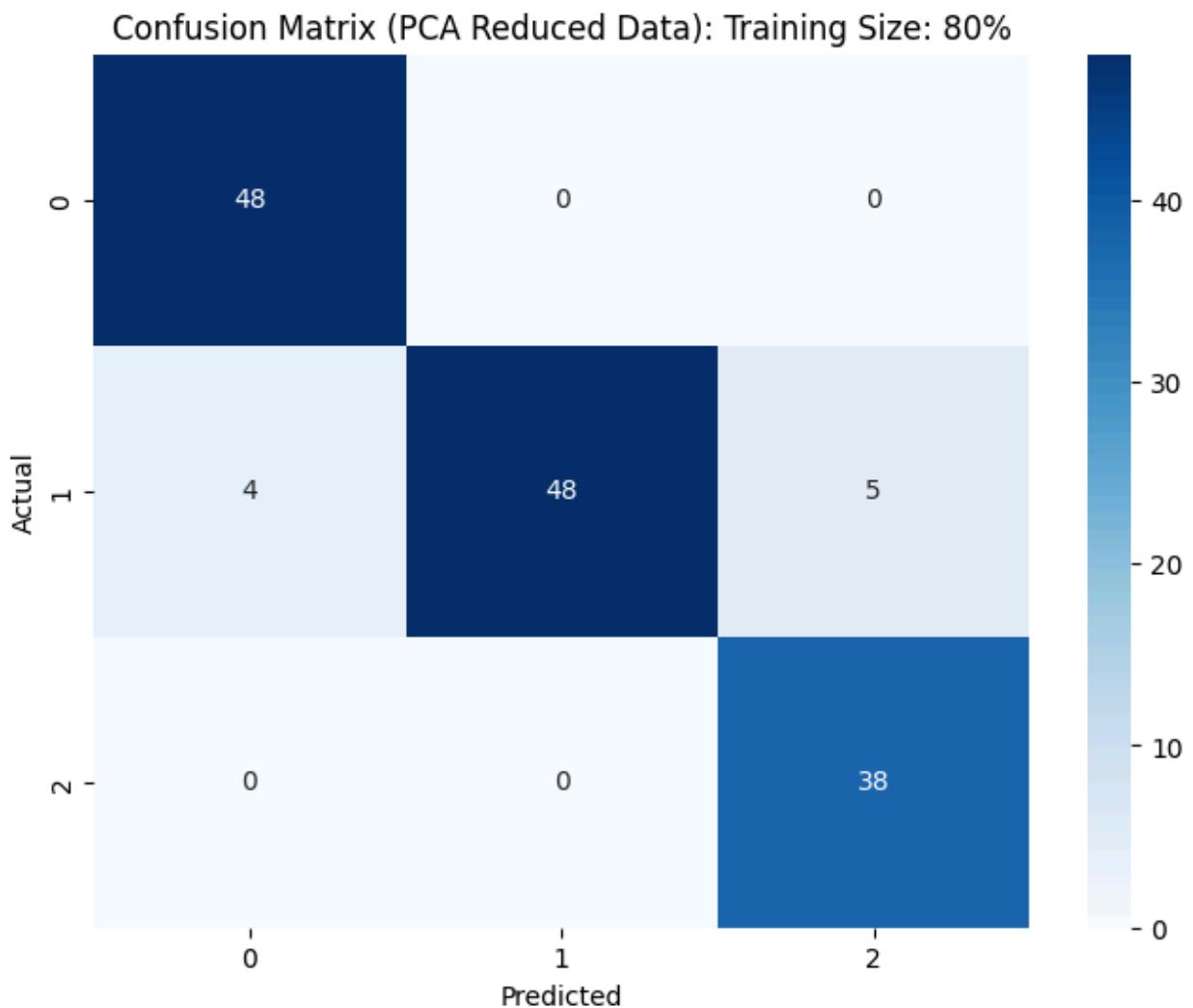


Confusion Matrix (PCA Reduced Data): Training Size: 60%



Confusion Matrix (PCA Reduced Data): Training Size: 70%



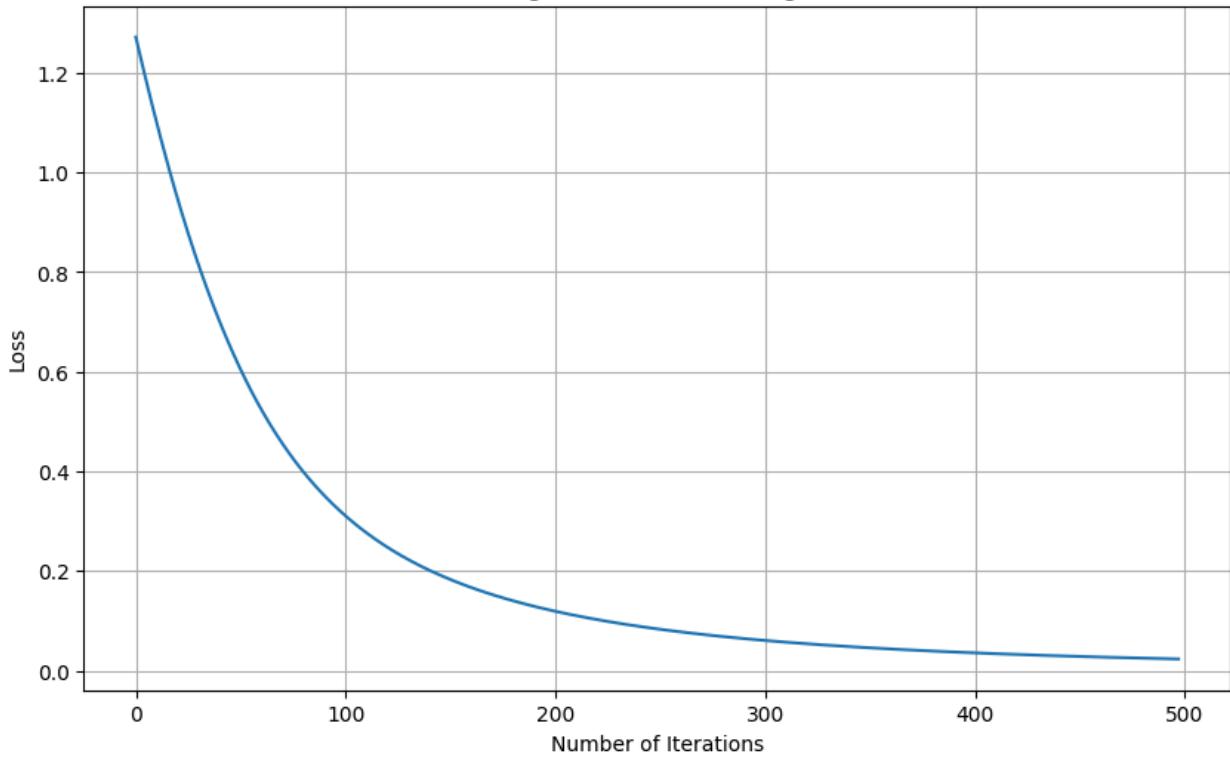


## Training Loss generation curve with PCA-Reduced Data

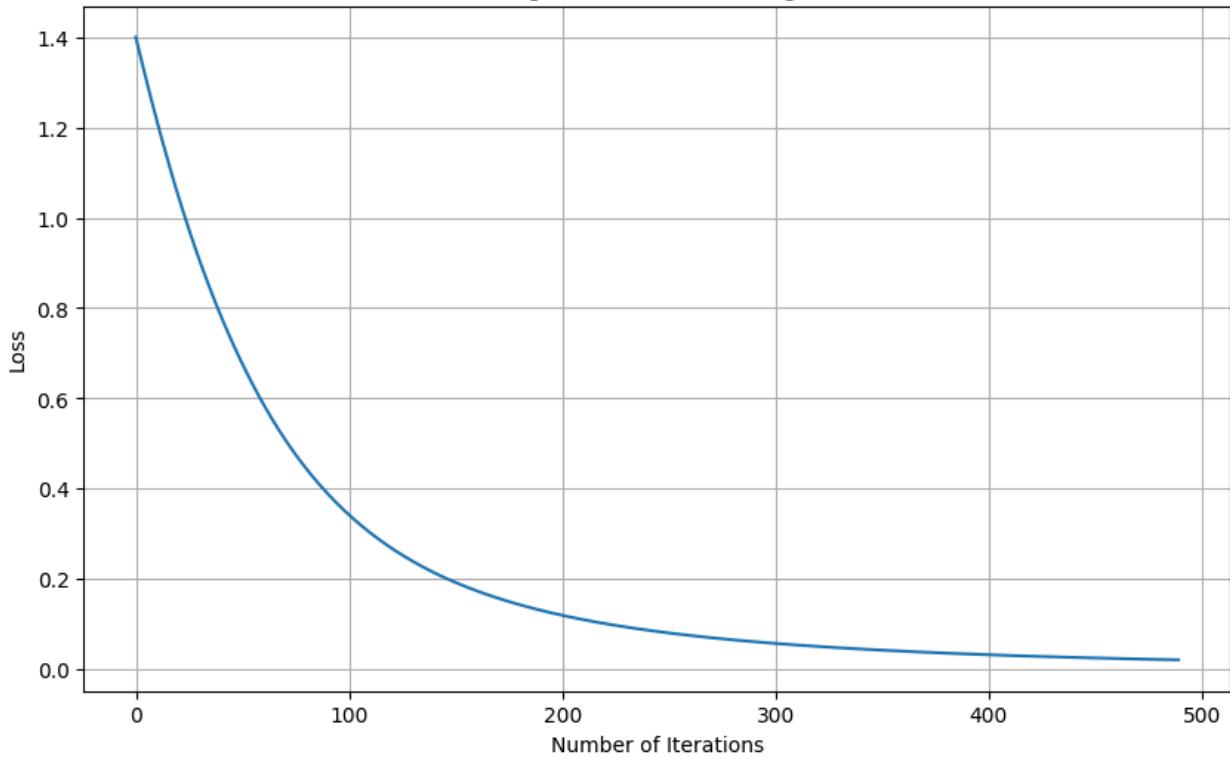
```
In [ ]: # Plot training loss curve for each trained MLP model
for model_data in trained_models_mlp_pca:
    model = model_data["Model"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(10, 6))
    plt.plot(model.loss_curve_)
    plt.title(f'MLP Training Loss Curve: Training Size: {training_size}%')
    plt.xlabel('Number of Iterations')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()
```

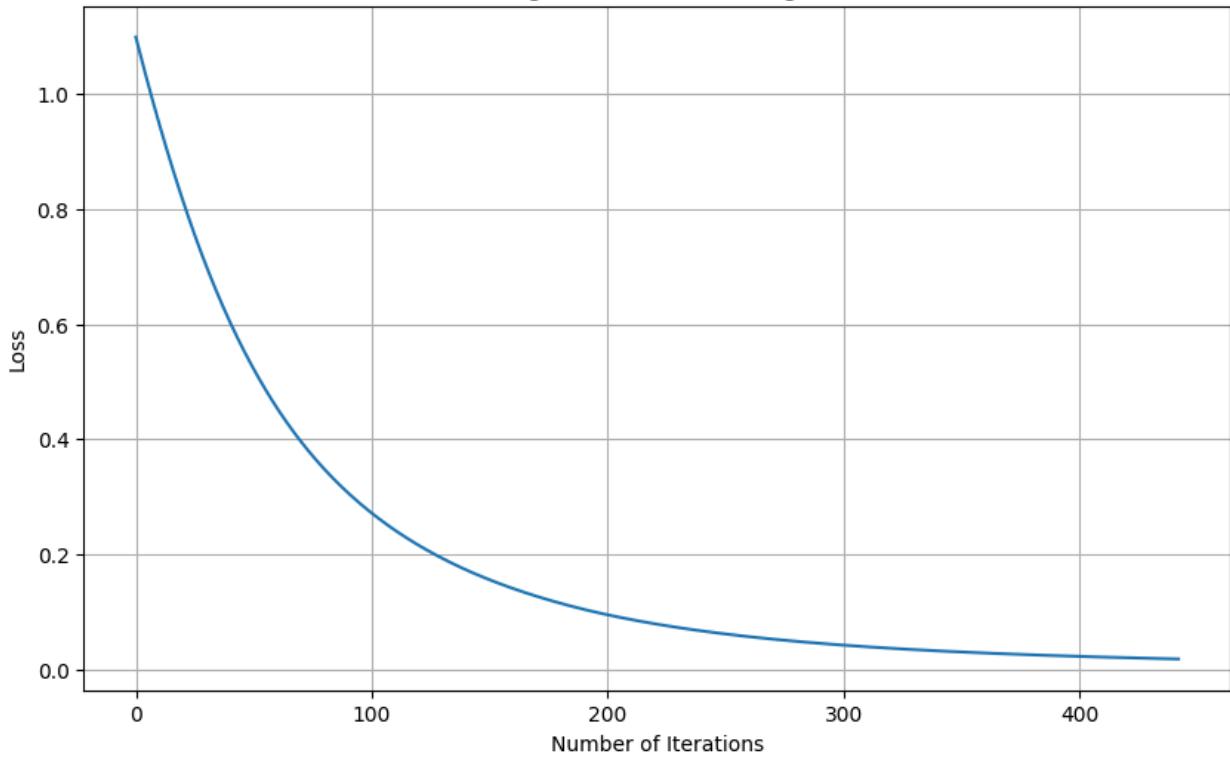
MLP Training Loss Curve: Training Size: 50%



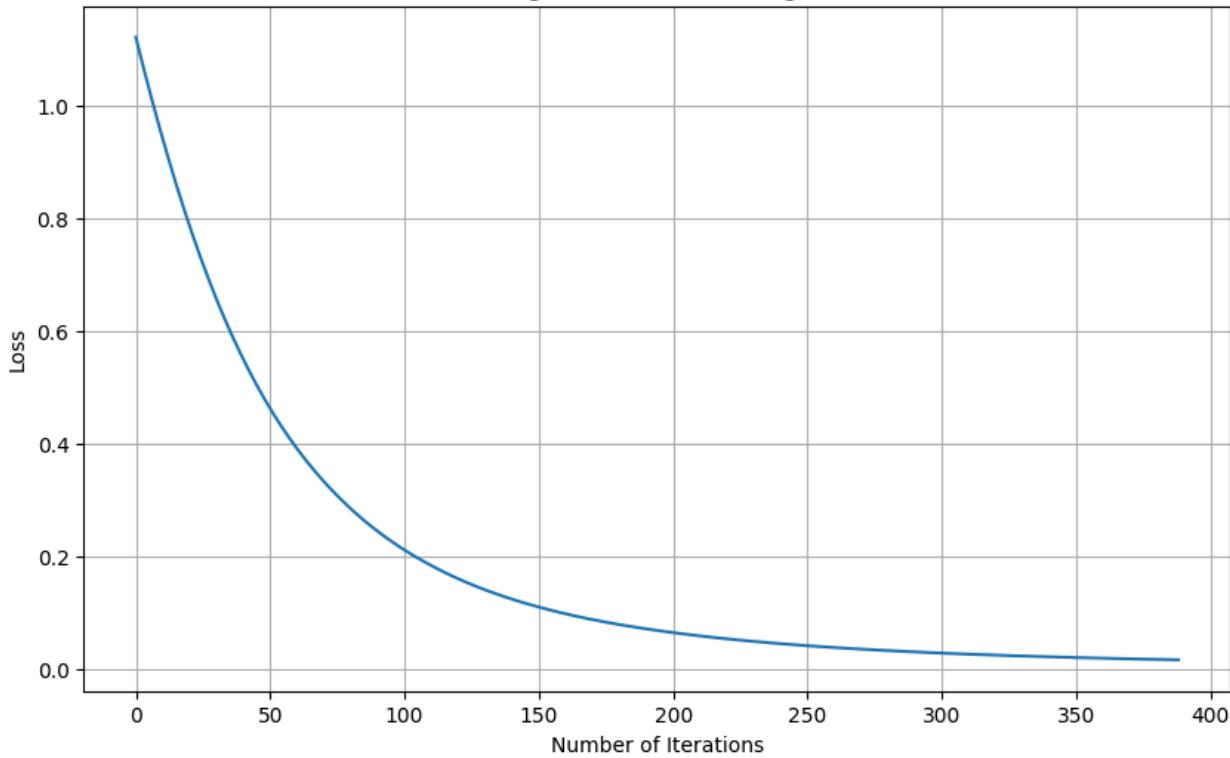
MLP Training Loss Curve: Training Size: 60%



MLP Training Loss Curve: Training Size: 70%



MLP Training Loss Curve: Training Size: 80%



# Random Forest

## Calculate values for different test size

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]

results_rf = []
confusion_matrices_rf = []
trained_models_rf = [] # Add a list to store trained models

for size in training_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)
    rf = RandomForestClassifier(random_state=42)

    rf.fit(X_train, y_train.values.ravel())
    y_pred_test = rf.predict(X_test)
    y_pred_train = rf.predict(X_train)

    acc_test = accuracy_score(y_test, y_pred_test)
    precision_test = precision_score(y_test, y_pred_test, average='weighted', zero_division=0)
    recall_test = recall_score(y_test, y_pred_test, average='weighted') # Calculate weighted recall
    f1_test = f1_score(y_test, y_pred_test, average='weighted') # Calculate weighted F1-score

    acc_train = accuracy_score(y_train, y_pred_train)
    precision_train = precision_score(y_train, y_pred_train, average='weighted', zero_division=0)
    recall_train = recall_score(y_train, y_pred_train, average='weighted')
    f1_train = f1_score(y_train, y_pred_train, average='weighted')

    cm = confusion_matrix(y_test, y_pred_test) # Calculate confusion matrix

    results_rf.append({
        "Training size":int (size*100),
        "Accuracy":acc_test,
        "Precision":precision_test,
        "Recall": recall_test,
        "F1-score": f1_test
    })

    confusion_matrices_rf.append({ # Store confusion matrix with metadata
        "Training size":int (size*100),
        "Confusion Matrix":cm
    })
```

```
})

trained_models_rf.append({ # Store the trained model with metadata
    "Training size":int (size*100),
    "Model": rf,
    "X_test": X_test,
    "y_test": y_test
})
```

## Print Evaluation Table and Graphs

```
In [ ]: df = pd.DataFrame(results_rf)
display(df) # Display the DataFrame as a table

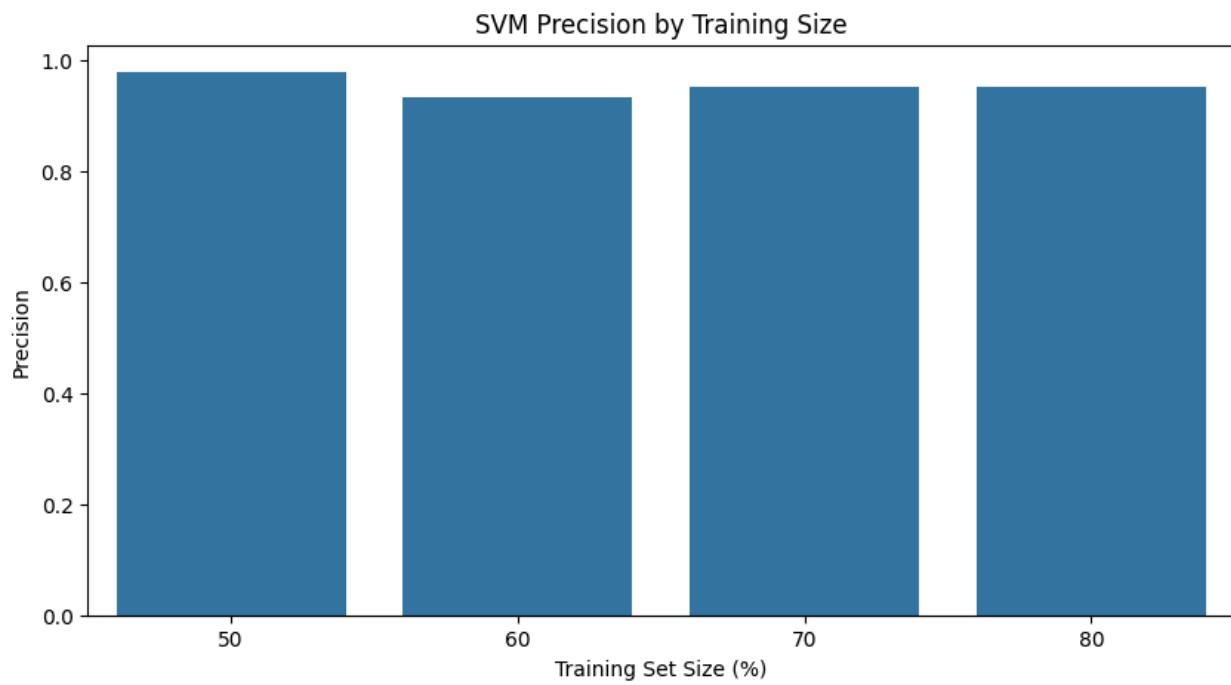
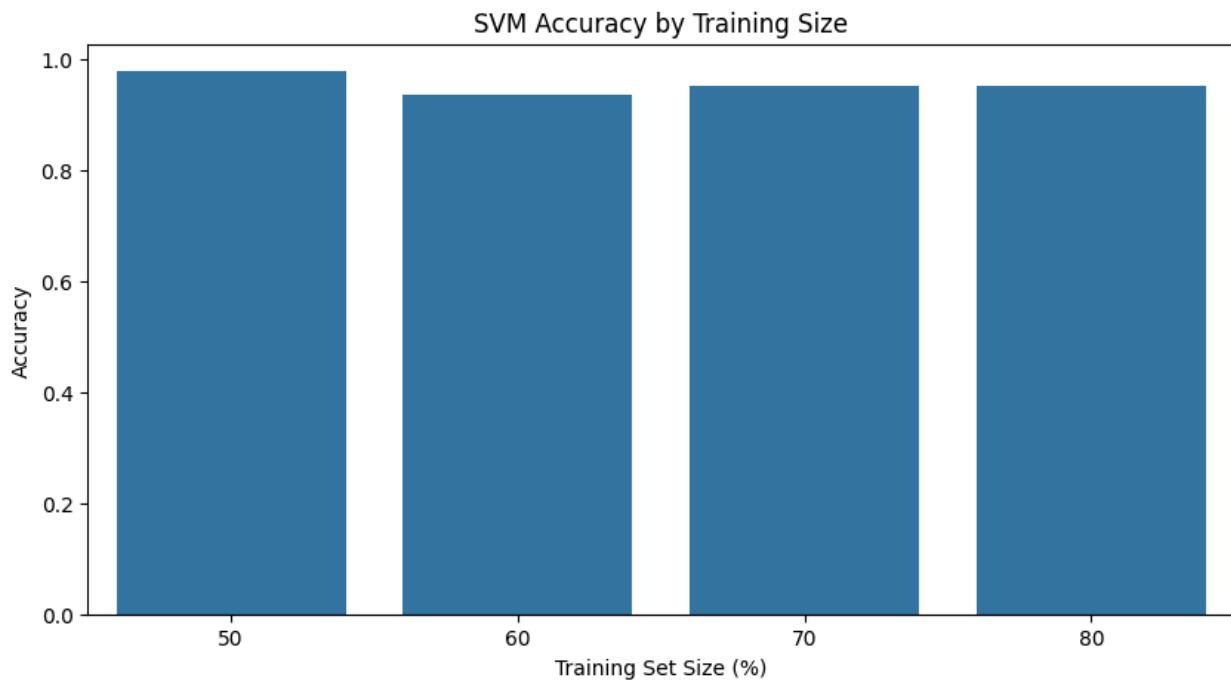
# Plot Accuracy
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy')
plt.title('SVM Accuracy by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

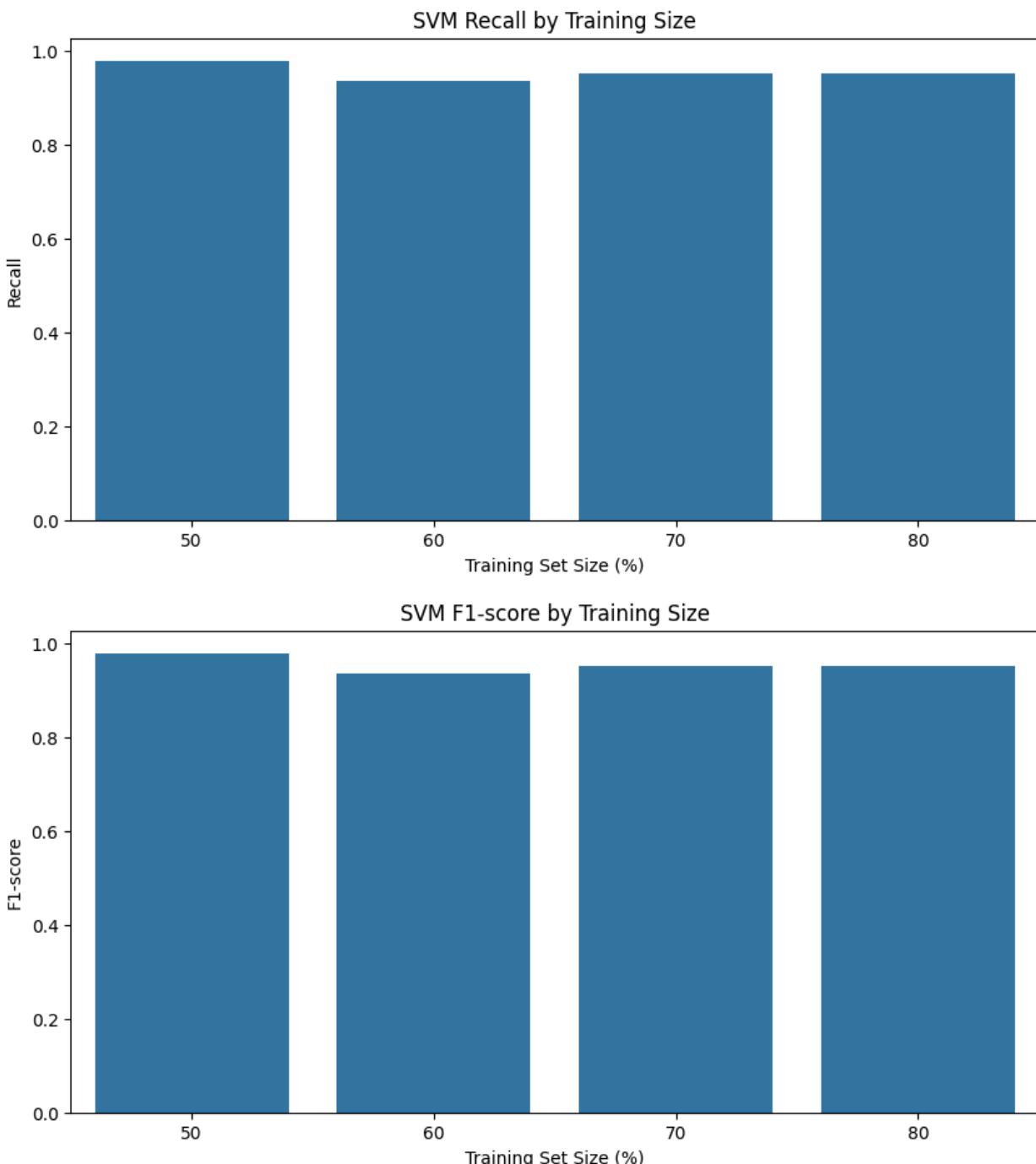
# Plot Precision
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision')
plt.title('SVM Precision by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

# Plot Recall
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall')
plt.title('SVM Recall by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

# Plot F1-score
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score')
plt.title('SVM F1-score by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()
```

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.977528	0.978777	0.977528
<b>1</b>	60	0.934579	0.934670	0.934579
<b>2</b>	70	0.952000	0.952117	0.952000
<b>3</b>	80	0.951049	0.952787	0.951049





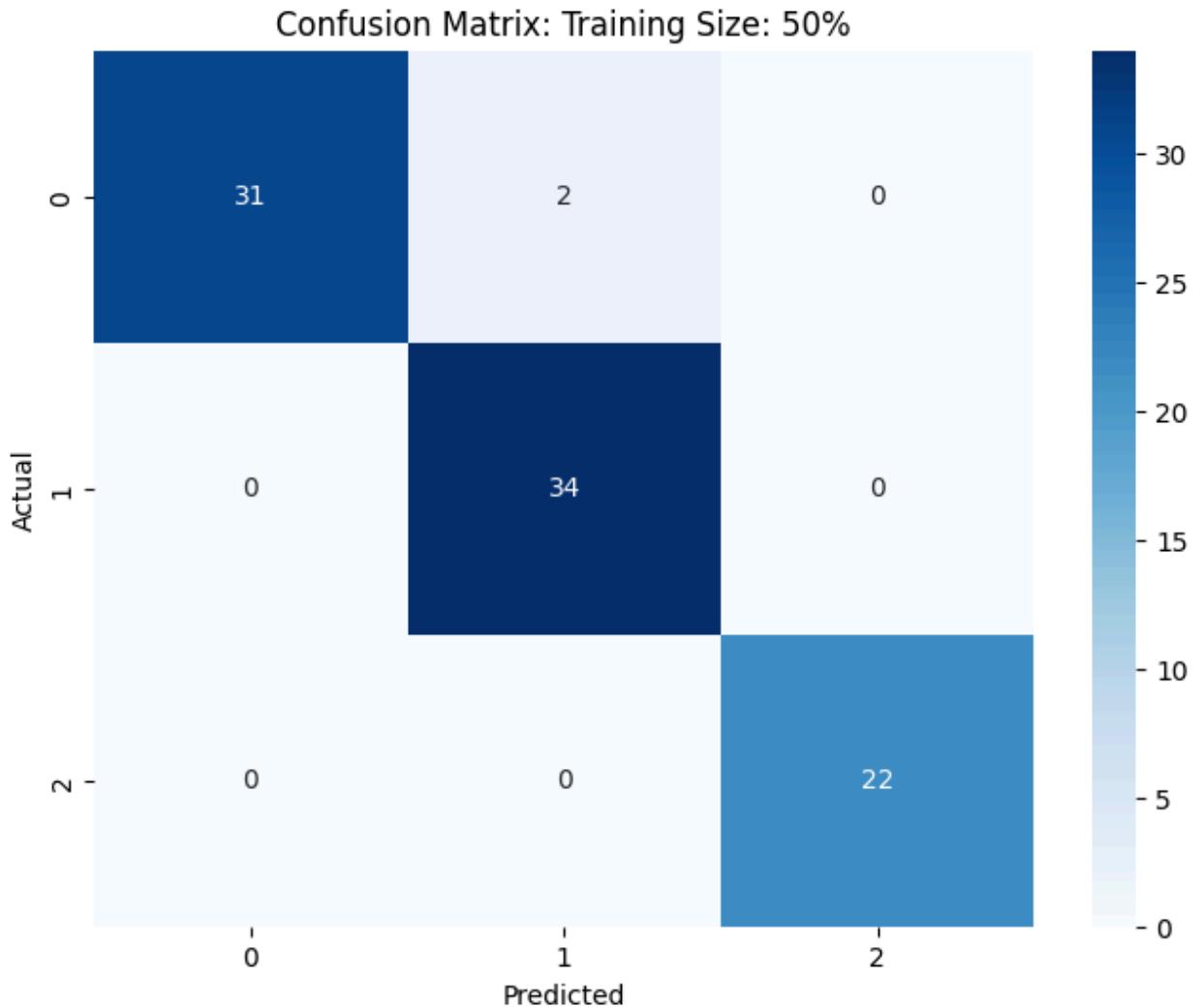
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

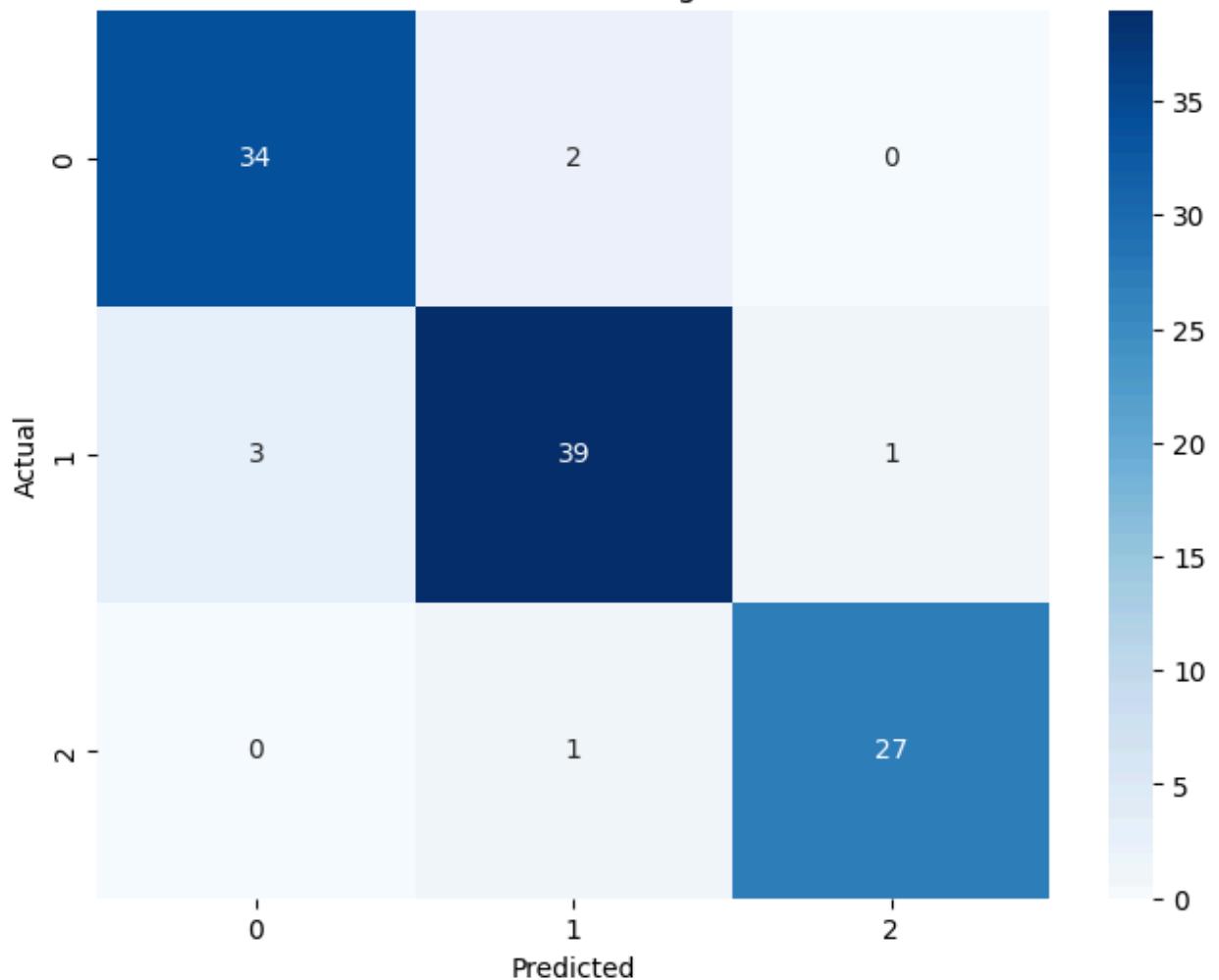
for cm_data in confusion_matrices_rf:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
```

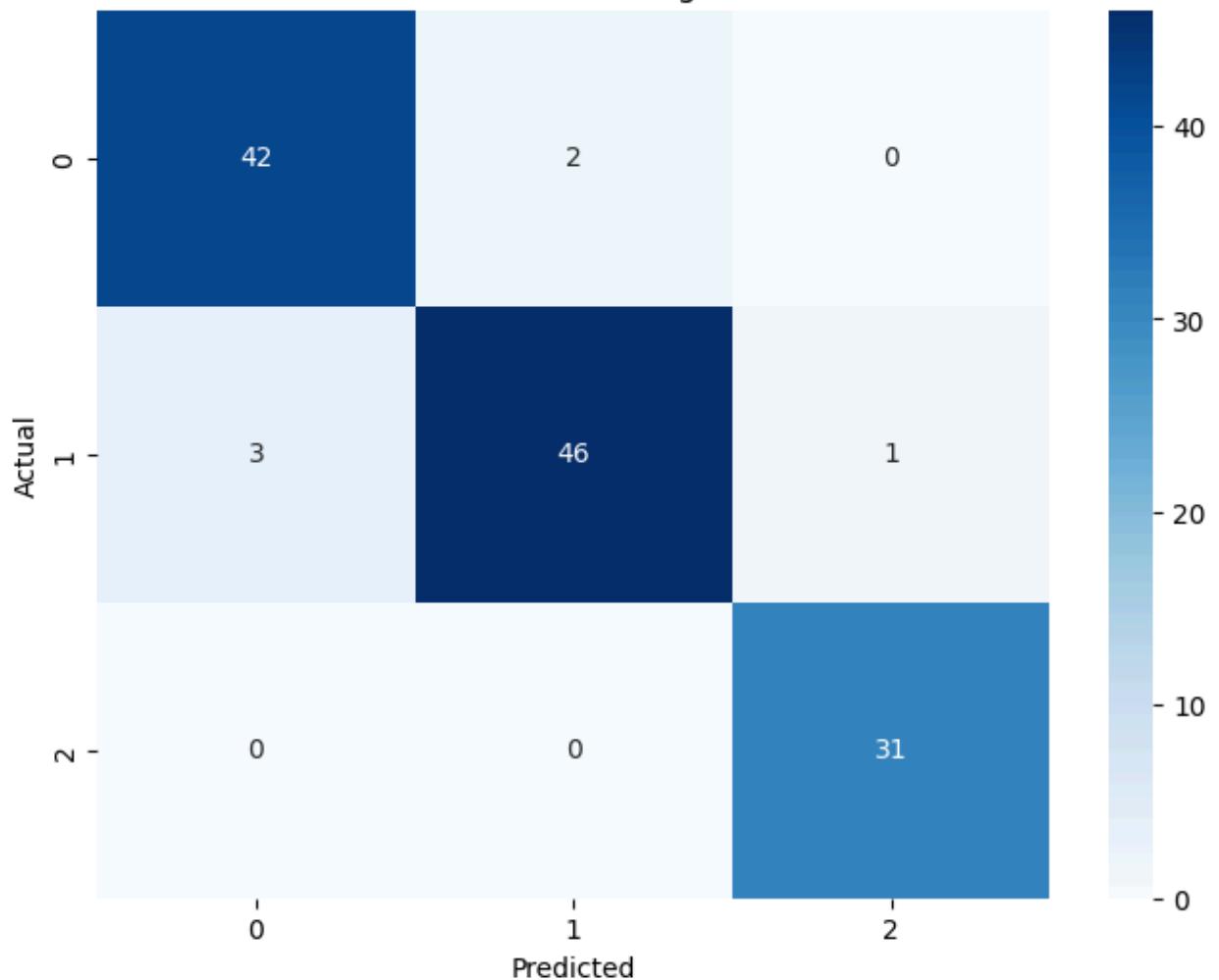
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix: Training Size: {training_size}%')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

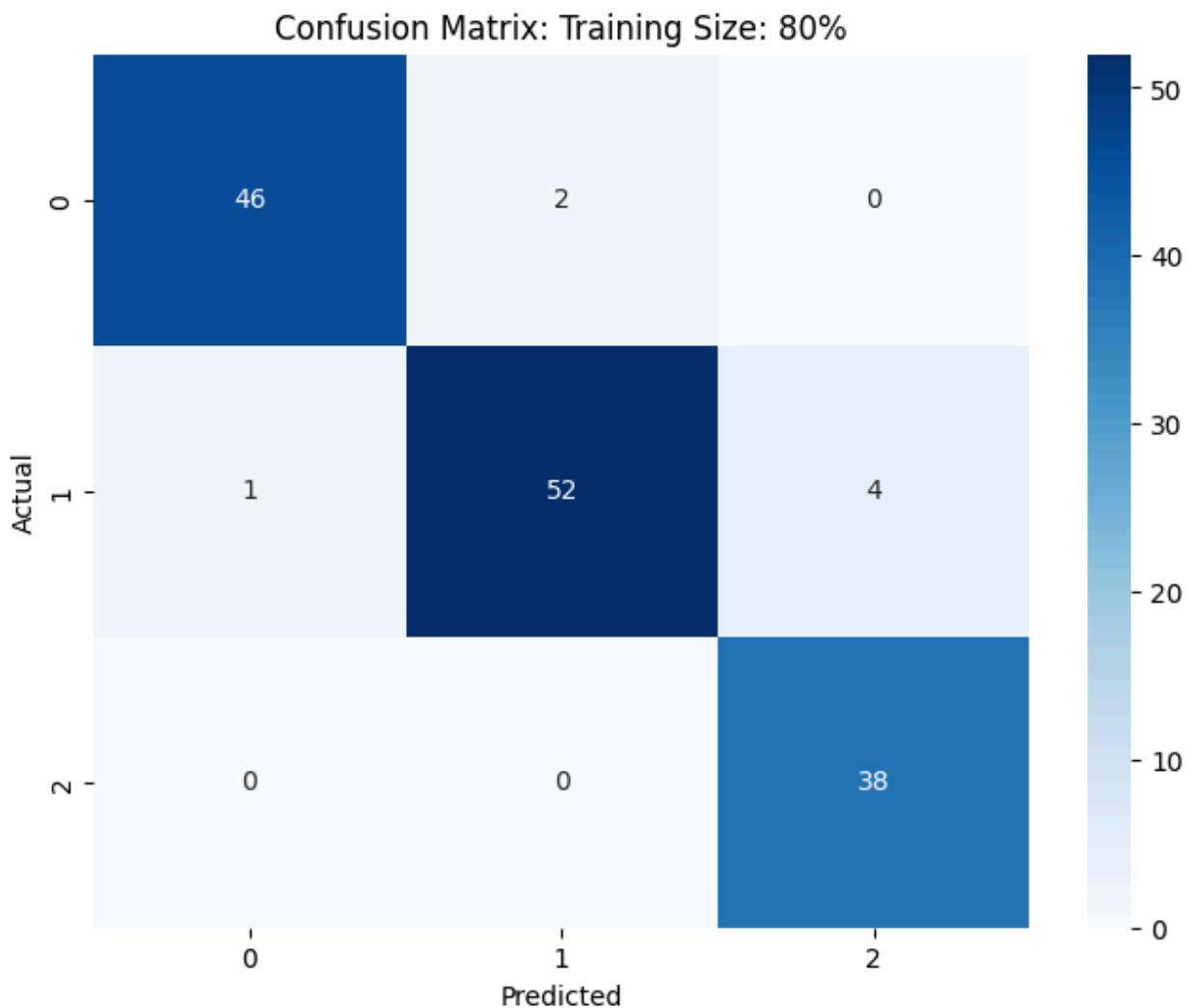


Confusion Matrix: Training Size: 60%



Confusion Matrix: Training Size: 70%





## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models_rf:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
```

```

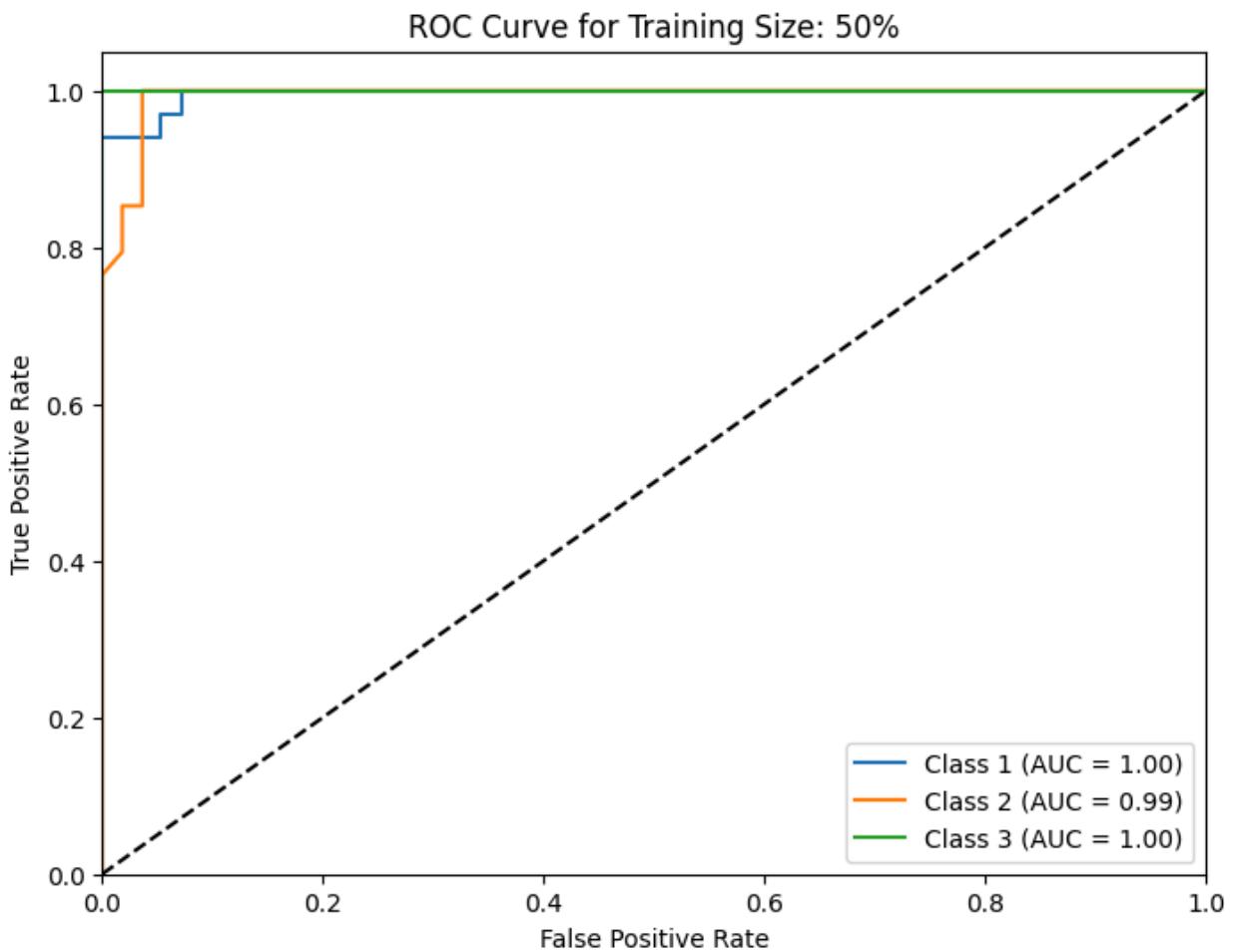
n_classes = len(np.unique(y_test))

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
    roc_auc[i] = auc(fpr[i], tpr[i])

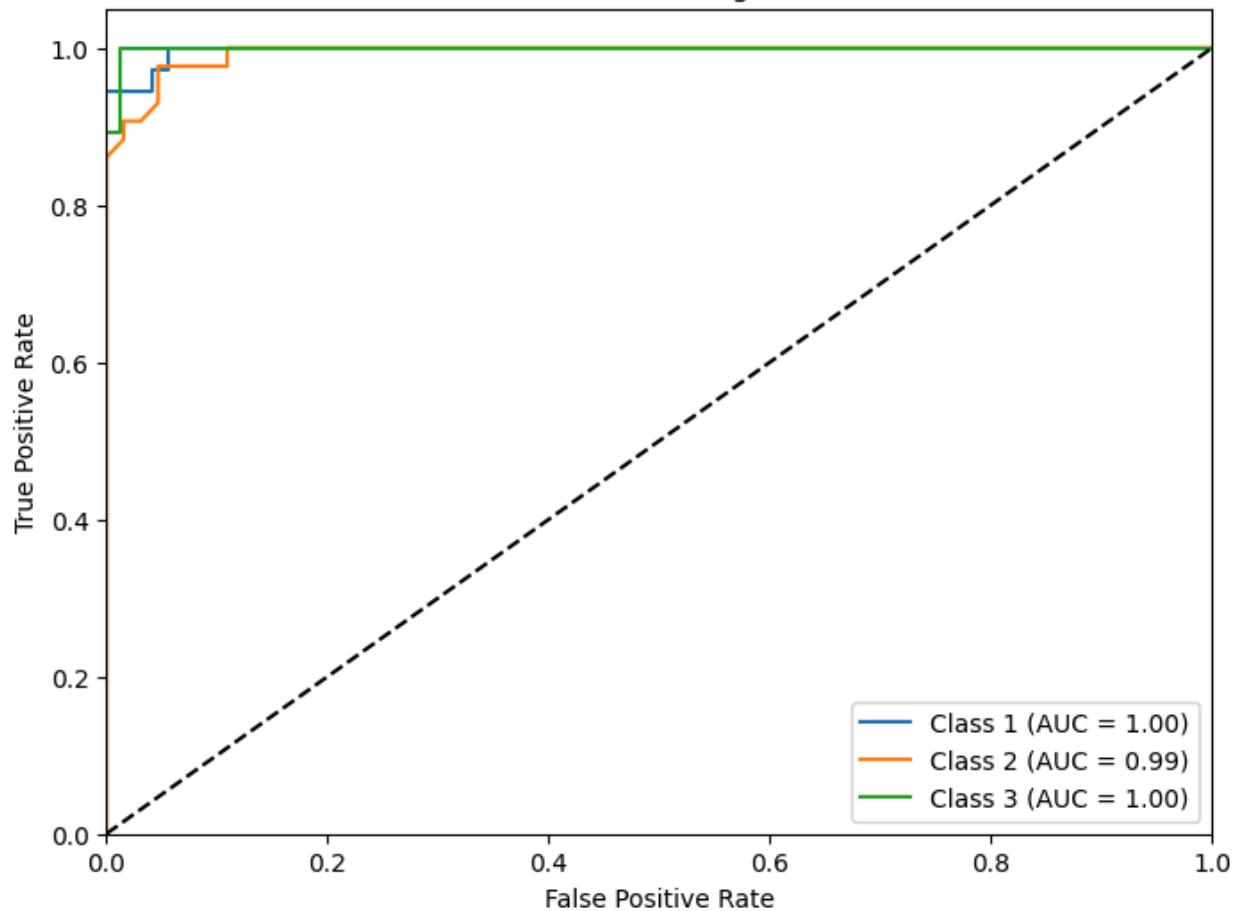
# Plot ROC curve for each class
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Training Size: {training_size}%') # Add title for plot
plt.legend(loc="lower right")
plt.show()

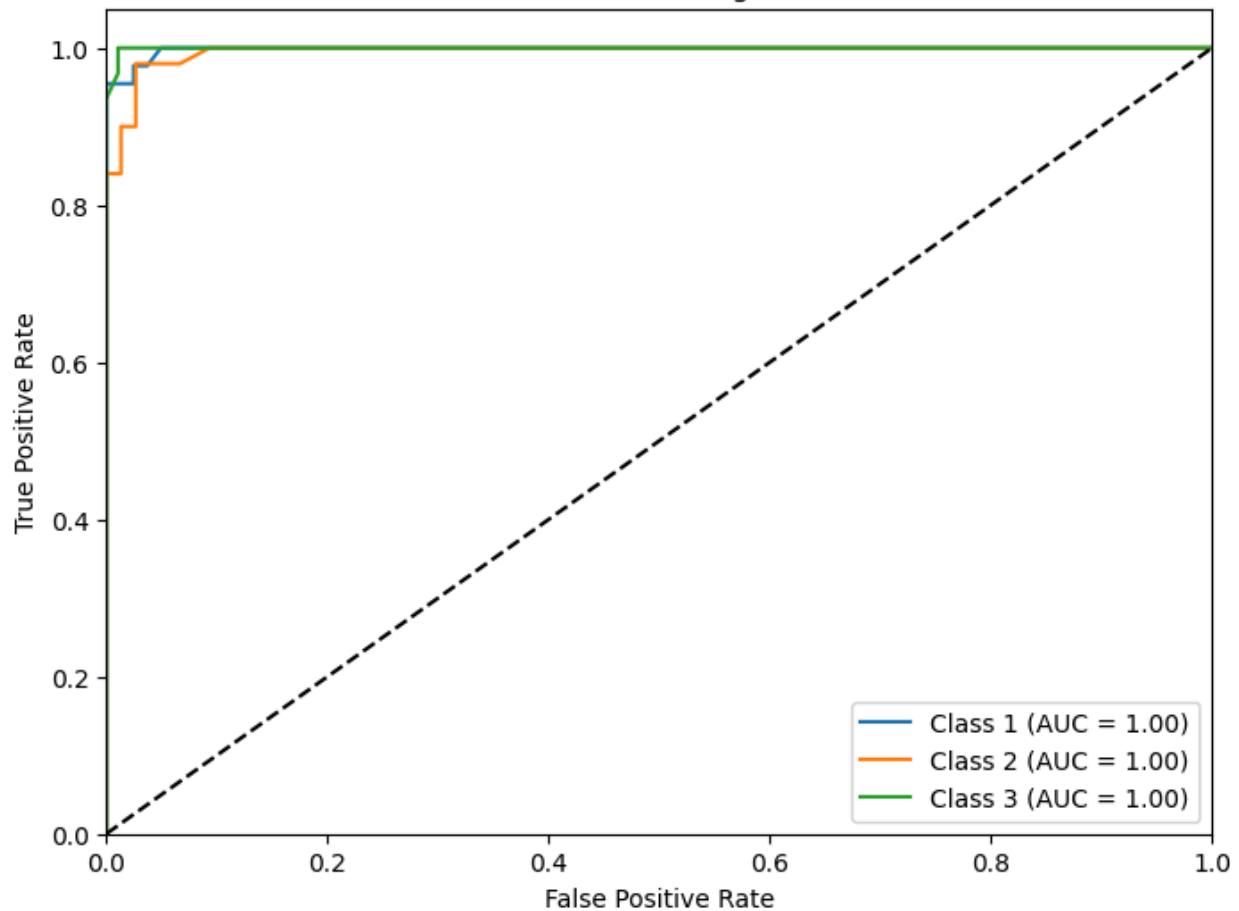
```

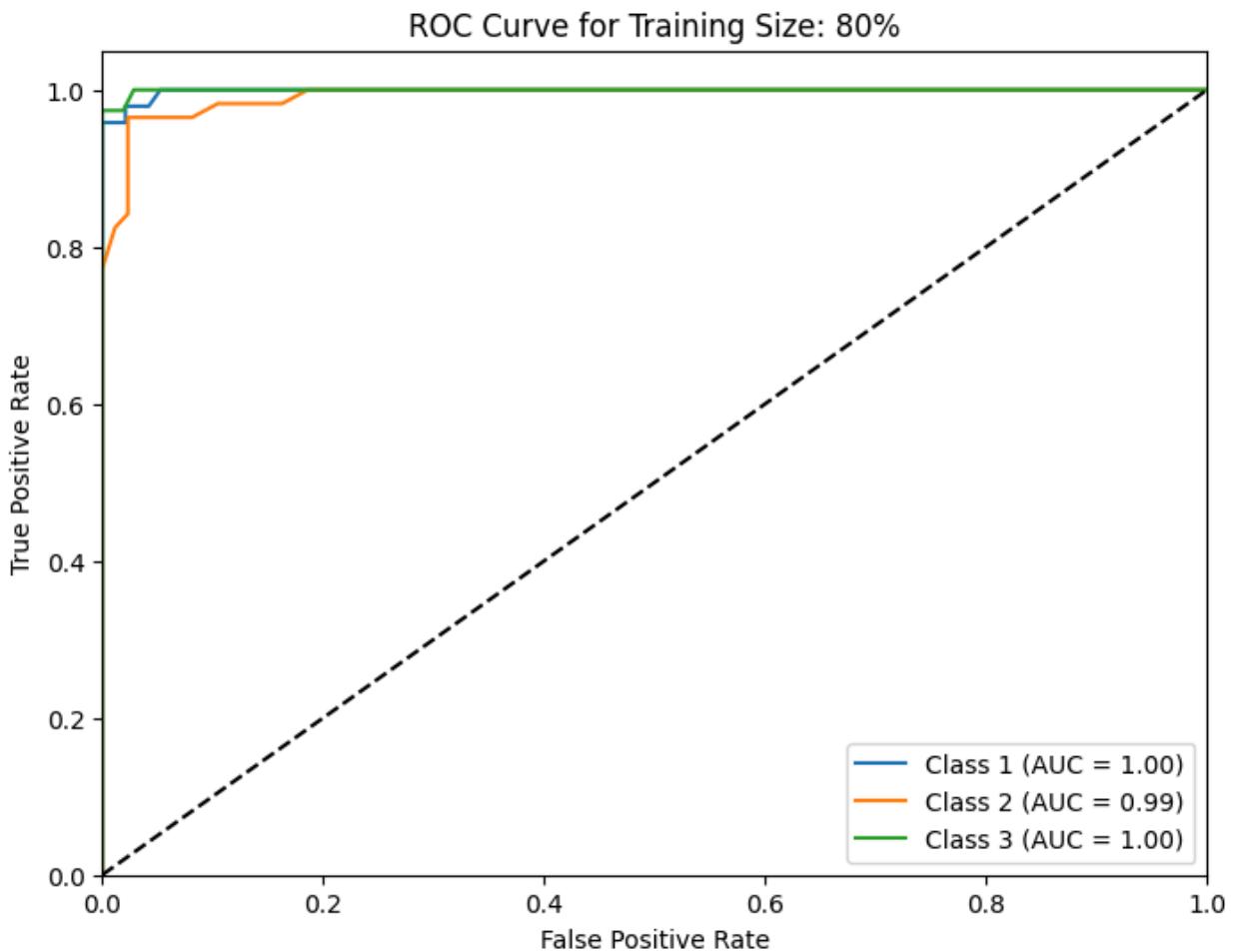


ROC Curve for Training Size: 60%



ROC Curve for Training Size: 70%





## Random Forest with PCA-Reduced Data

```
In [ ]: # Apply the existing PCA transformation to the scaled data
X_pca_reduced_rf = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced_rf.shape[1]}")
print("\nData after PCA for Random Forest:")
display(pd.DataFrame(X_pca_reduced_rf).head())
```

Original number of features: 13  
 Reduced number of features after PCA: 10

Data after PCA for Random Forest:

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	
<b>0</b>	3.316751	1.443463	-0.165739	-0.215631	0.693043	0.223880	0.596427	-0.065
<b>1</b>	2.209465	-0.333393	-2.026457	-0.291358	-0.257655	0.927120	0.053776	-1.024
<b>2</b>	2.516740	1.031151	0.982819	0.724902	-0.251033	-0.549276	0.424205	0.344
<b>3</b>	3.757066	2.756372	-0.176192	0.567983	-0.311842	-0.114431	-0.383337	-0.643
<b>4</b>	1.008908	0.869831	2.026688	-0.409766	0.298458	0.406520	0.444074	-0.416

## Calculate Random Forest performance with PCA-Reduced Data for different test sizes

```
In [ ]: results_rf_pca = []
confusion_matrices_rf_pca = []

for size in training_sizes:
    X_train_pca_rf, X_test_pca_rf, y_train, y_test = train_test_split(X_pca_reduced, random_state=42)
    rf_pca = RandomForestClassifier(random_state=42)

    rf_pca.fit(X_train_pca_rf, y_train.values.ravel())
    y_pred_pca_rf = rf_pca.predict(X_test_pca_rf)

    acc_pca_rf = accuracy_score(y_test, y_pred_pca_rf)
    precision_pca_rf = precision_score(y_test, y_pred_pca_rf, average='weighted')
    recall_pca_rf = recall_score(y_test, y_pred_pca_rf, average='weighted')
    f1_pca_rf = f1_score(y_test, y_pred_pca_rf, average='weighted')

    cm_pca_rf = confusion_matrix(y_test, y_pred_pca_rf)

    results_rf_pca.append({
        "Training size":int (size*100),
        "Accuracy":acc_pca_rf,
        "Precision": precision_pca_rf,
        "Recall": recall_pca_rf,
        "F1-score": f1_pca_rf
    })

    confusion_matrices_rf_pca.append({
        "Training size":int (size*100),
        "Confusion Matrix":cm_pca_rf
    })
```

## Print Random Forest Performance Table and Graphs with PCA-Reduced Data

```
In [ ]: df_rf_pca = pd.DataFrame(results_rf_pca)
display(df_rf_pca)
```

```

# Plot Accuracy
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Accuracy')
plt.title('Random Forest Accuracy by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.show()

# Plot Precision
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Precision')
plt.title('Random Forest Precision by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Precision')
plt.show()

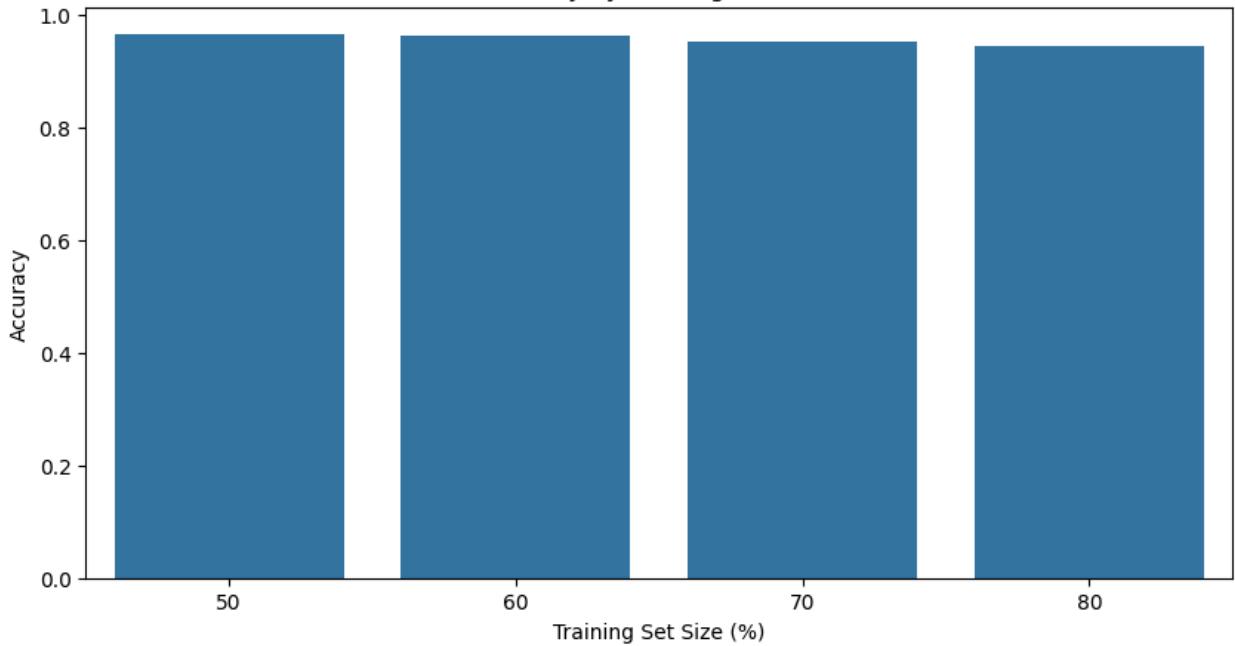
# Plot Recall
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Recall')
plt.title('Random Forest Recall by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Recall')
plt.show()

# Plot F1-score
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='F1-score')
plt.title('Random Forest F1-score by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('F1-score')
plt.show()

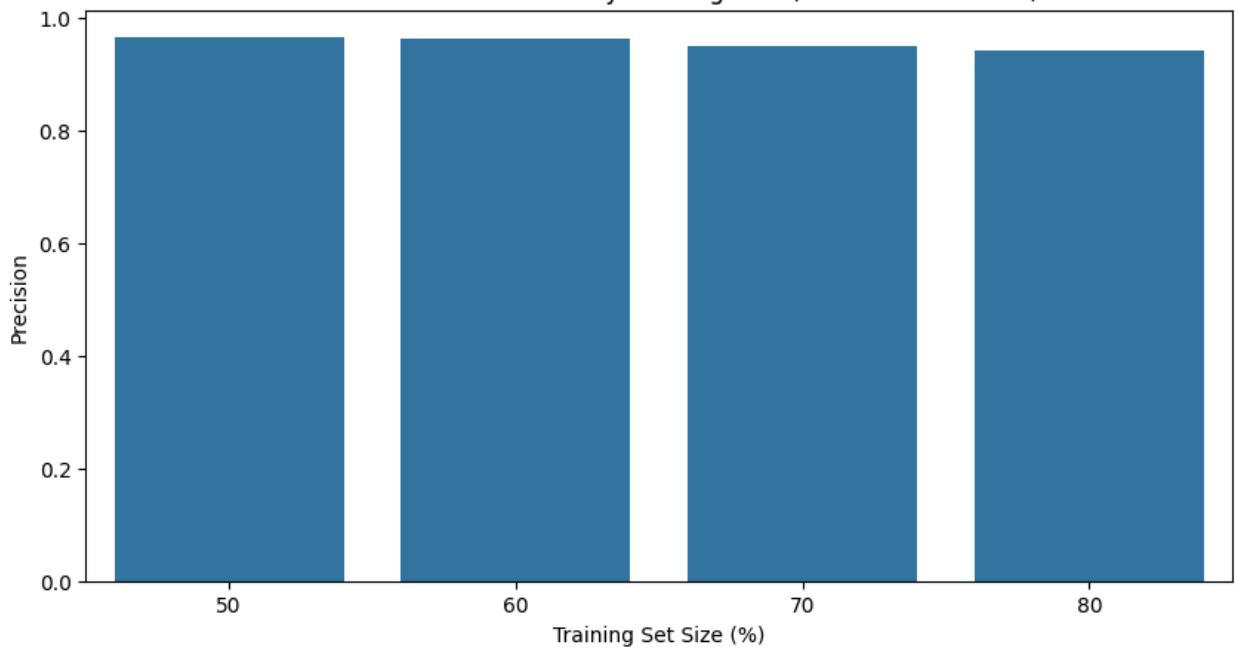
```

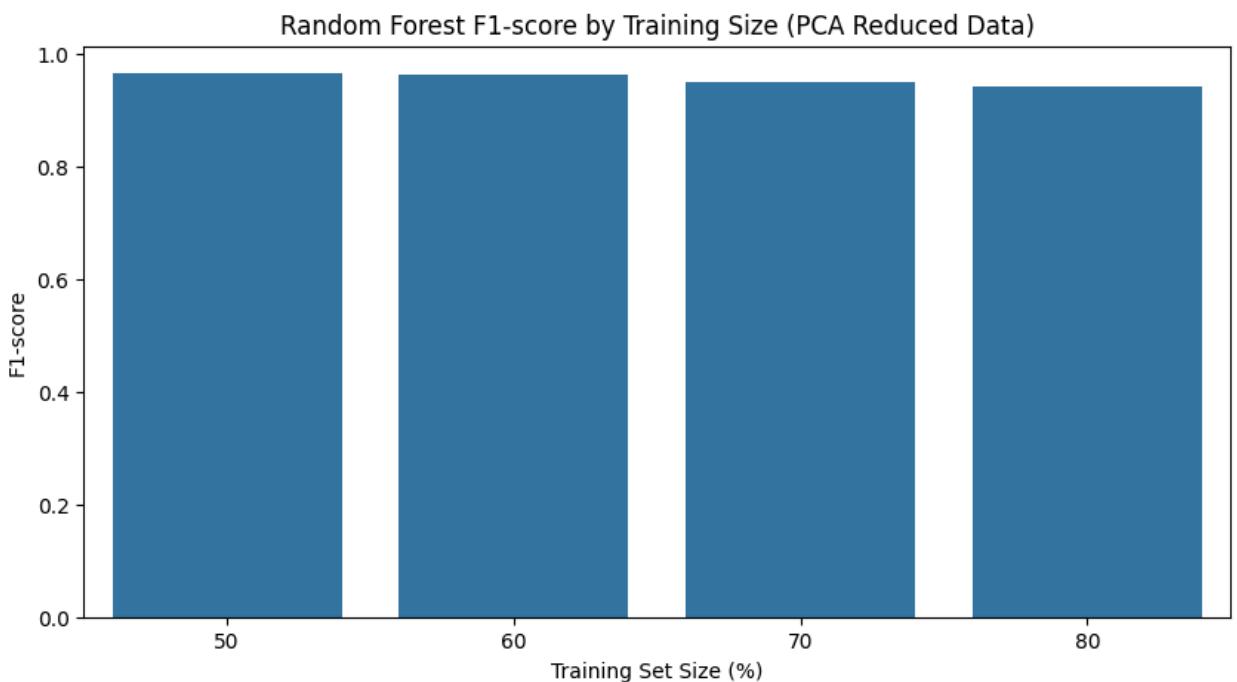
	<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.966292	0.966583	0.966292	0.966277
<b>1</b>	60	0.962617	0.963637	0.962617	0.962677
<b>2</b>	70	0.952000	0.951878	0.952000	0.951834
<b>3</b>	80	0.944056	0.943992	0.944056	0.943736

Random Forest Accuracy by Training Size (PCA Reduced Data)



Random Forest Precision by Training Size (PCA Reduced Data)



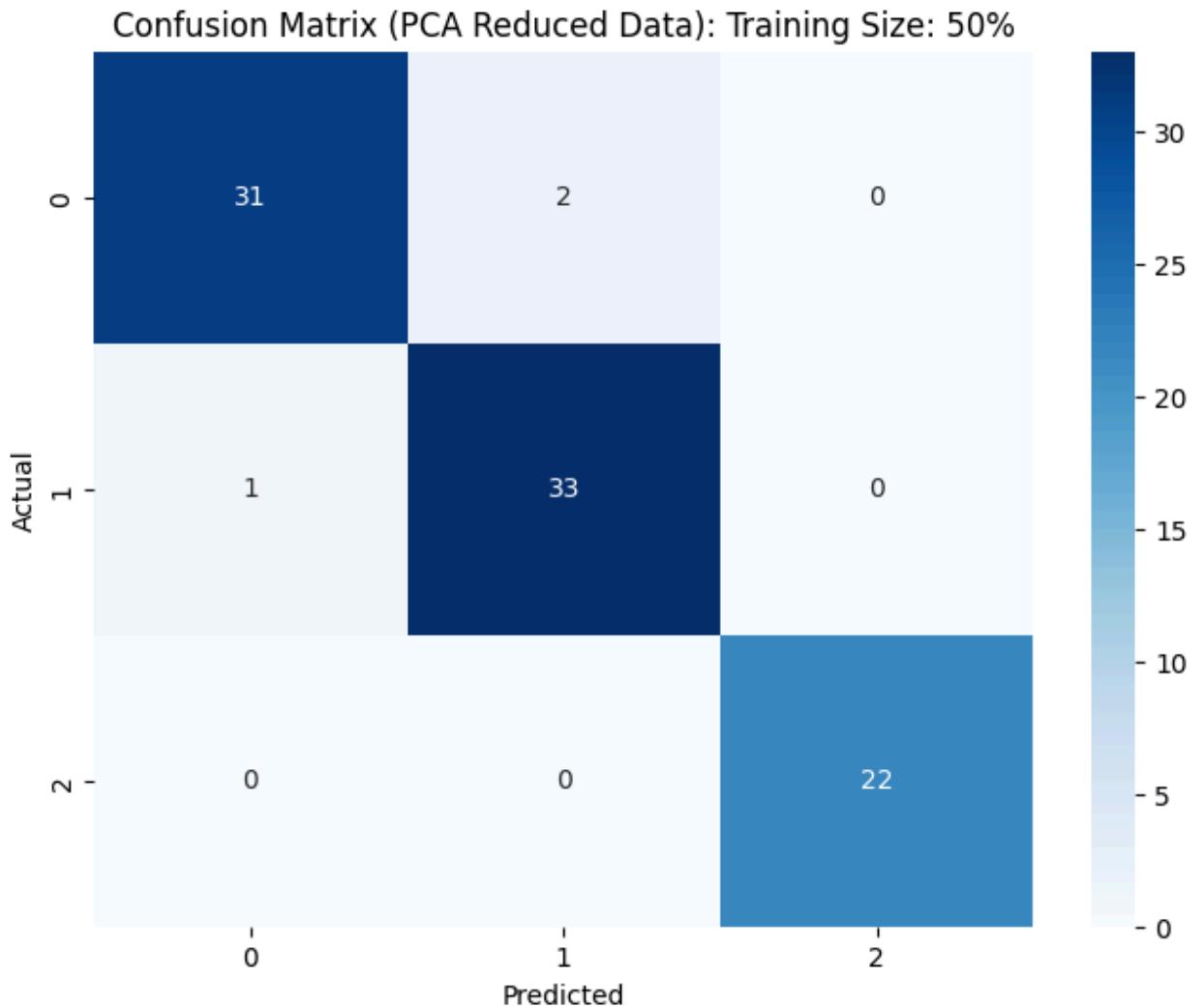


Show Confusion Matrices for Random Forest with PCA-Reduced Data

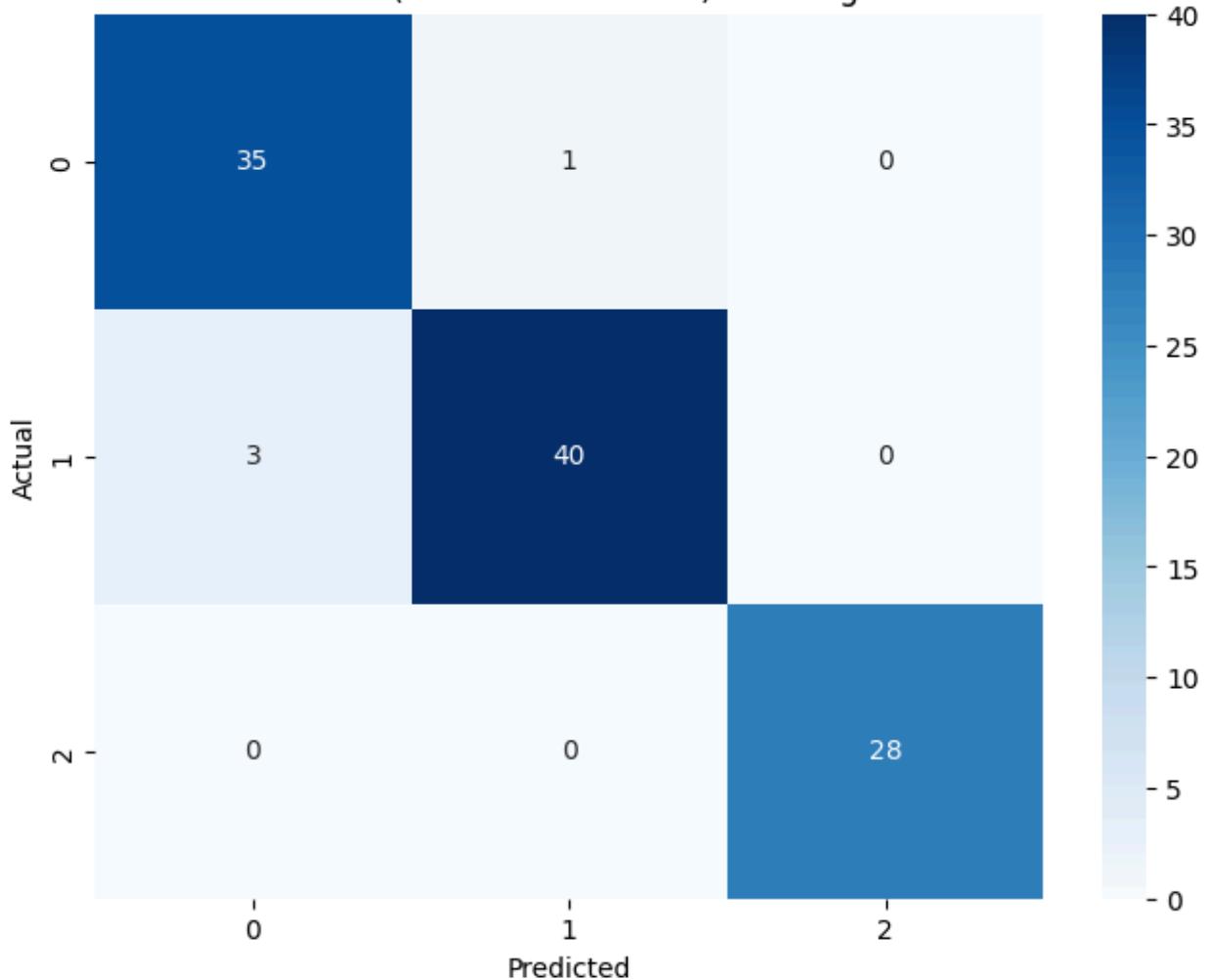
```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

for cm_data in confusion_matrices_rf_pca:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]
```

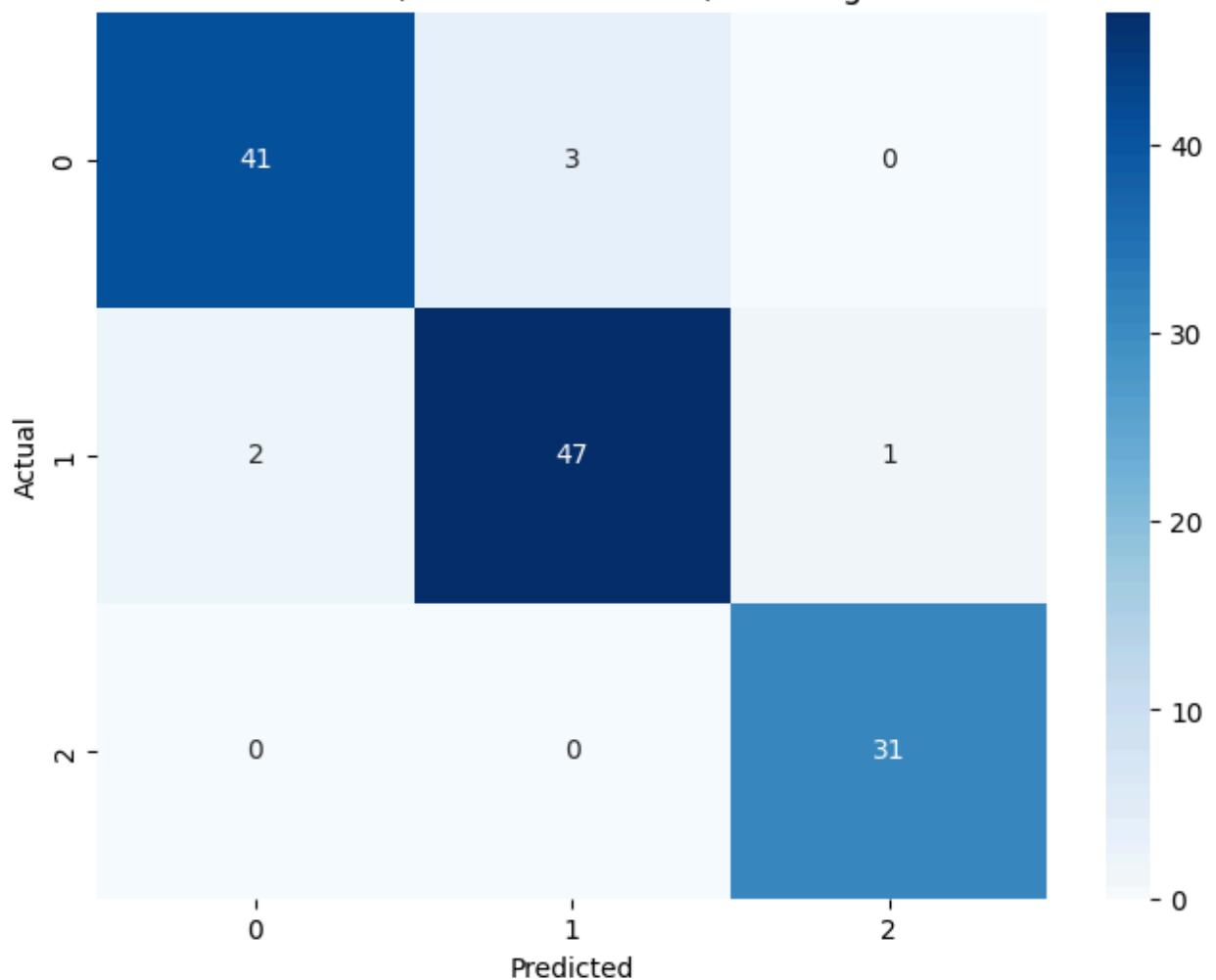
```
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix (PCA Reduced Data): Training Size: {training_size}%')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

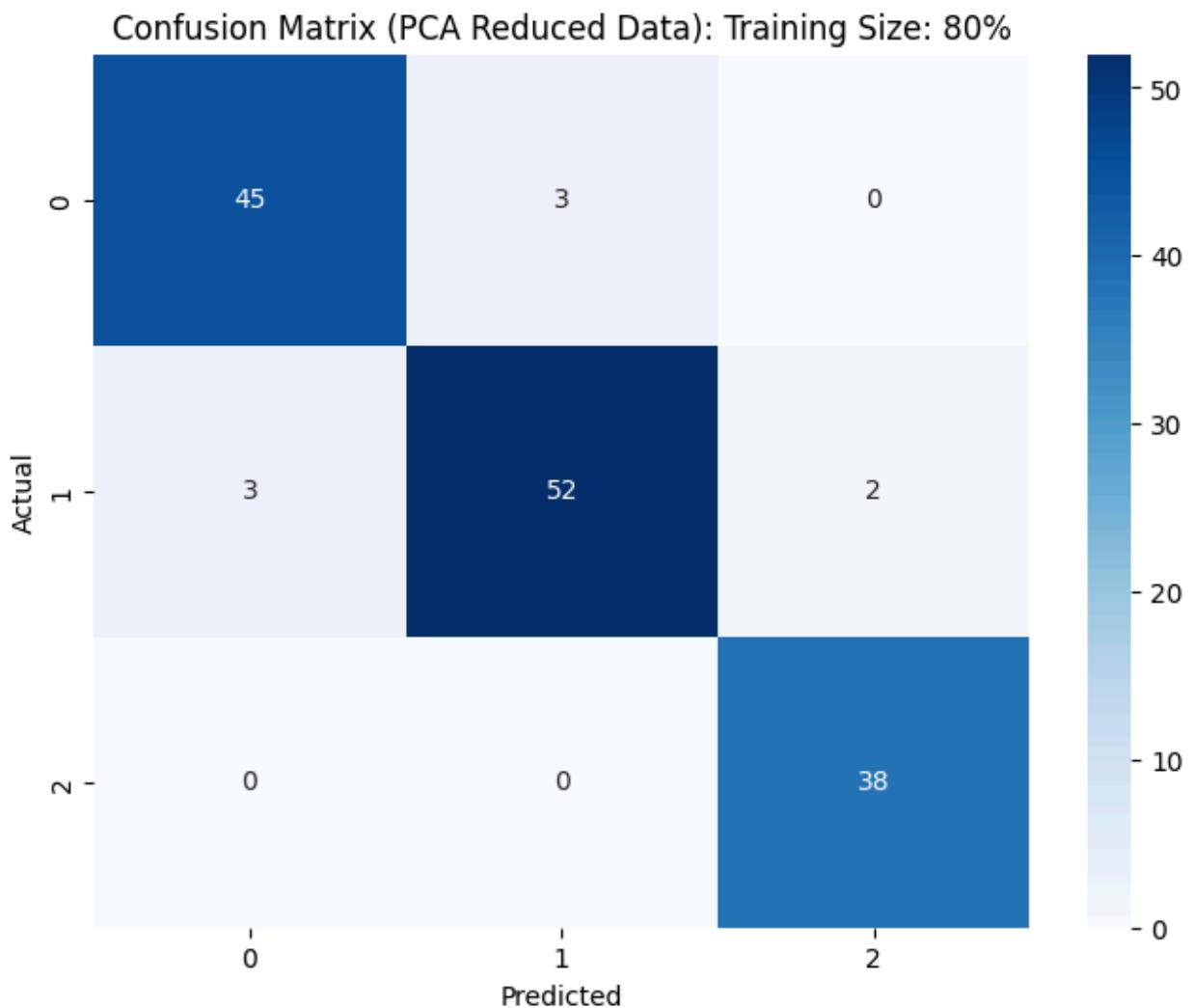


Confusion Matrix (PCA Reduced Data): Training Size: 60%



Confusion Matrix (PCA Reduced Data): Training Size: 70%





## Hand Written Digit Dataset

### Prepare Dataset

```
In [ ]: from ucimlrepo import fetch_ucirepo  
  
# fetch dataset  
digits = fetch_ucirepo(id=80)
```

```
In [ ]: X = digits.data.features  
y = digits.data.targets
```

```
In [ ]: X
```

Out[ ]:

	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7
0	0	1	6	15	12	1	1
1	0	0	10	16	6	0	0
2	0	0	8	15	16	13	13
3	0	0	0	3	11	16	16
4	0	0	5	14	4	0	0
...	...	...	...	...	...	...	...
5615	0	0	4	10	13	6	6
5616	0	0	6	16	13	11	11
5617	0	0	1	11	15	1	1
5618	0	0	2	10	7	0	0
5619	0	0	10	14	8	1	1

5620 rows × 64 columns

In [ ]: y

Out[ ]:

	class
0	0
1	0
2	7
3	4
4	6
...	...
5615	9
5616	0
5617	8
5618	9
5619	8

5620 rows × 1 columns

# Principal Component Analysis (PCA)

```
In [ ]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standardize the data before applying PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained variance ratio by each component:", explained_variance_ratio)

# Cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
print("Cumulative explained variance:", cumulative_explained_variance)

# Plot explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), cumulative_explained_var
plt.title('Explained Variance by Number of Principal Components')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
```

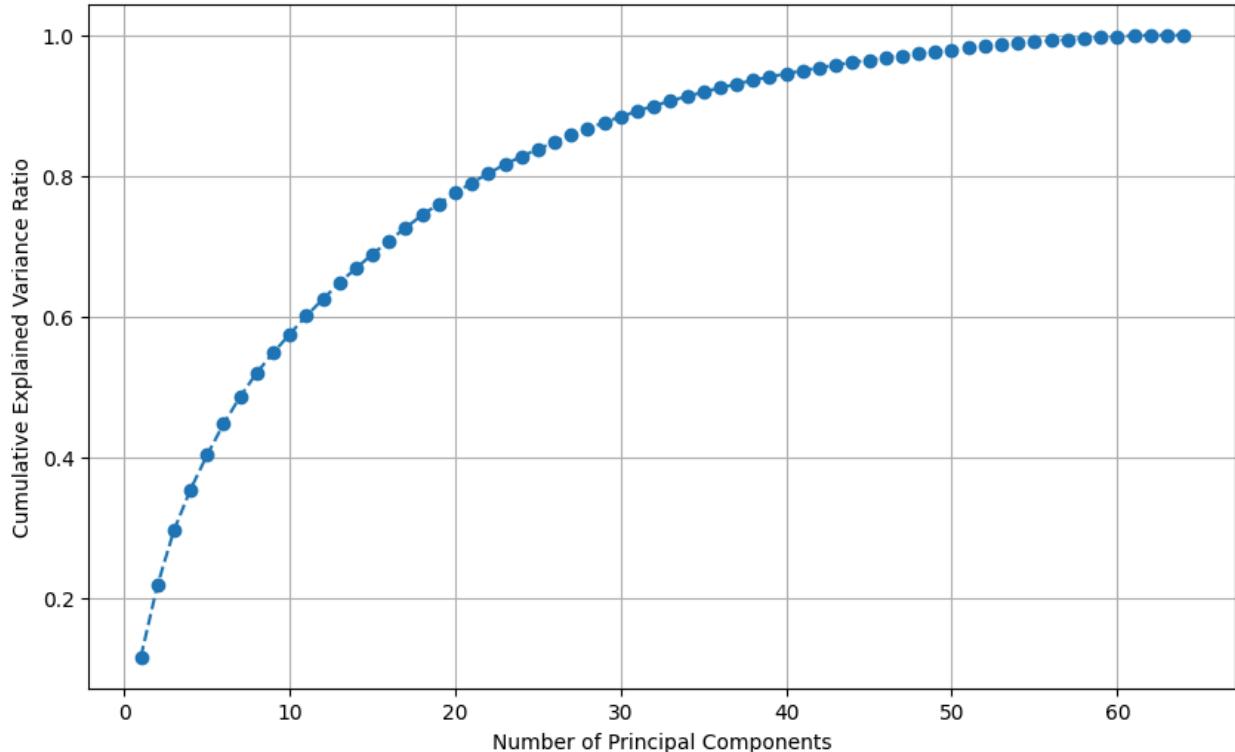
```

Explained variance ratio by each component: [1.16058580e-01 1.01701536e-01 7.80
219607e-02 5.83857531e-02
4.87287824e-02 4.44316461e-02 3.85877681e-02 3.39688103e-02
2.84702728e-02 2.66407663e-02 2.57952555e-02 2.39640483e-02
2.29641894e-02 2.09663662e-02 1.98540773e-02 1.92591302e-02
1.83958755e-02 1.79110669e-02 1.61101921e-02 1.53670187e-02
1.44033255e-02 1.35133284e-02 1.26344227e-02 1.13733091e-02
1.02700335e-02 1.01363699e-02 9.84299671e-03 9.24277181e-03
8.78744597e-03 8.23774719e-03 8.05148259e-03 7.30720466e-03
7.12056062e-03 6.30496265e-03 6.16040321e-03 6.03570820e-03
5.55208908e-03 5.13331216e-03 4.94934613e-03 4.52420547e-03
4.38713196e-03 4.10618712e-03 3.93404572e-03 3.51002669e-03
3.44296187e-03 3.27169186e-03 3.15703391e-03 2.87722127e-03
2.65990842e-03 2.60422421e-03 2.50353416e-03 2.41975789e-03
2.21433164e-03 2.06632155e-03 1.89531541e-03 1.84897966e-03
1.67357847e-03 1.57841578e-03 1.43360959e-03 1.26219371e-03
1.04637788e-03 9.43029486e-04 1.47912361e-18 0.00000000e+00]

Cumulative explained variance: [0.11605858 0.21776012 0.29578208 0.35416783 0.4
0289661 0.44732826
0.48591603 0.51988484 0.54835511 0.57499588 0.60079113 0.62475518
0.64771937 0.66868574 0.68853981 0.70779894 0.72619482 0.74410589
0.76021608 0.7755831 0.78998642 0.80349975 0.81613417 0.82750748
0.83777752 0.84791389 0.85775688 0.86699965 0.8757871 0.88402485
0.89207633 0.89938353 0.90650409 0.91280906 0.91896946 0.92500517
0.93055726 0.93569057 0.94063992 0.94516412 0.94955125 0.95365744
0.95759149 0.96110151 0.96454448 0.96781617 0.9709732 0.97385042
0.97651033 0.97911455 0.98161809 0.98403785 0.98625218 0.9883185
0.99021382 0.9920628 0.99373637 0.99531479 0.9967484 0.99801059
0.99905697 1. 1. 1. ]

```

Explained Variance by Number of Principal Components



Based on the cumulative explained variance plot, you can decide how many principal components to retain to capture a desired amount of variance. For example, to retain 95% of the variance, you would choose the number of components where the cumulative explained variance is close to 0.95.

Let's apply PCA with a chosen number of components (e.g., based on the plot).

```
In [ ]: # Choose the number of components (e.g., to retain 95% variance)
# Based on the plot, let's assume we choose 10 components to retain most of the
n_components = 50 # You can change this based on your analysis of the plot

pca = PCA(n_components=n_components)
X_pca_reduced = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced.shape[1]}")
print("\nData after PCA:")
display(pd.DataFrame(X_pca_reduced).head())
```

Original number of features: 64  
Reduced number of features after PCA: 50

Data after PCA:

	0	1	2	3	4	5	6	
0	-0.296633	-1.446450	-3.863044	-3.223597	0.773631	0.631849	0.163733	0.474
1	-0.156717	-3.065688	-5.811656	-3.232099	1.064670	0.031353	0.270302	2.091
2	-0.753929	3.261949	0.779656	-1.108976	0.311024	-0.814548	-4.307315	0.841
3	-4.226579	1.900320	-0.441486	1.329548	-0.149220	2.710149	2.128209	0.206
4	0.582983	-3.490539	-1.602212	-1.052587	-1.360737	0.406474	-0.662578	-2.572

5 rows × 50 columns

## SVM

### Different SVM models

```
In [ ]: from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]
kernels = {
```

```
'linear': 'linear',
'polynomial': 'poly',
'gaussian': 'rbf',
'sigmoid': 'sigmoid'
}
```

## Calculate values for different test size

```
In [ ]: results = []
confusion_matrices = []
trained_models = [] # Add a list to store trained models

for kernel_name, kernel in kernels.items():
    for size in training_sizes:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)
        svc = SVC(kernel=kernel, probability=True, random_state=42) # Add probability=True
        svc.fit(X_train, y_train.values.ravel())
        y_pred = svc.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
        recall = recall_score(y_test, y_pred, average='weighted') # Calculate weighted recall
        f1 = f1_score(y_test, y_pred, average='weighted') # Calculate weighted F1-score

        cm = confusion_matrix(y_test, y_pred) # Calculate confusion matrix

        results.append({
            "Training size":int (size*100),
            "Accuracy":acc,
            "Precision": precision,
            "Recall": recall,
            "F1-score": f1,
            "Kernel":kernel_name
        })
        confusion_matrices.append({ # Store confusion matrix with metadata
            "Training size":int (size*100),
            "Kernel":kernel_name,
            "Confusion Matrix":cm
        })
        trained_models.append({ # Store the trained model with metadata
            "Training size":int (size*100),
            "Kernel":kernel_name,
            "Model": svc,
            "X_test": X_test,
            "y_test": y_test
        })
    }
```

## Print Evaluation Table and Graphs

```
In [ ]: df = pd.DataFrame(results)
```

```
display(df) # Display the DataFrame as a table

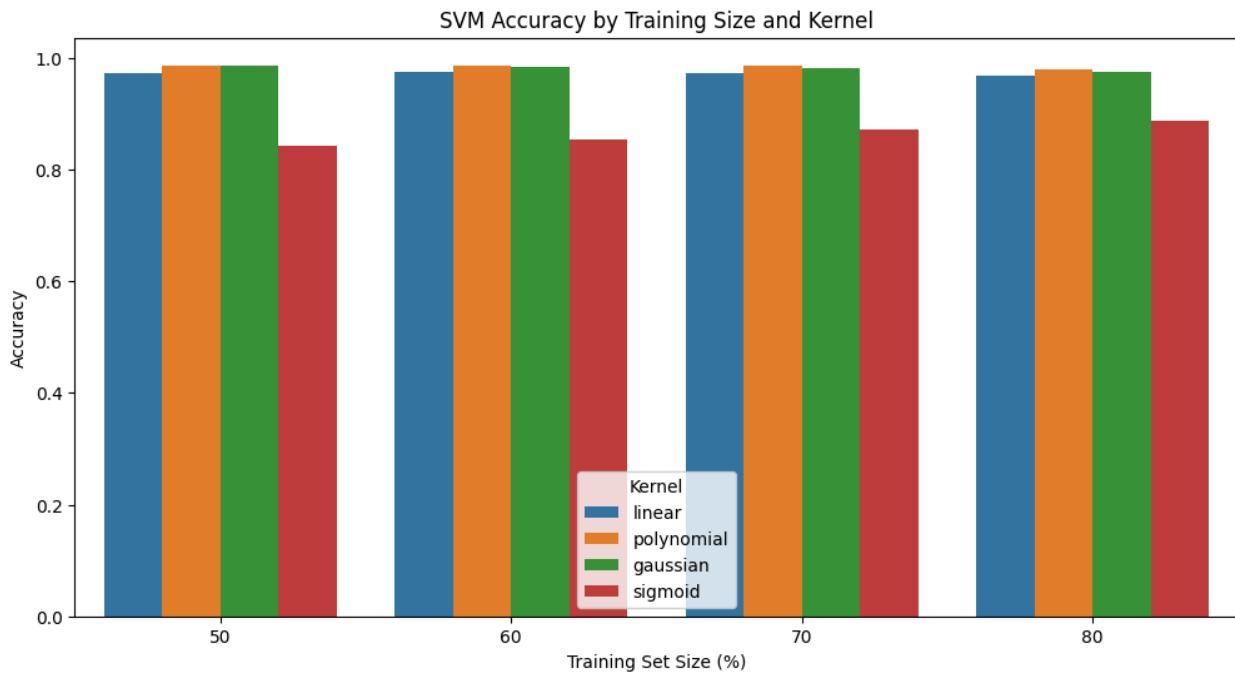
# Plot Accuracy
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy', hue='Kernel')
plt.title('SVM Accuracy by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

# Plot Precision
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision', hue='Kernel')
plt.title('SVM Precision by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

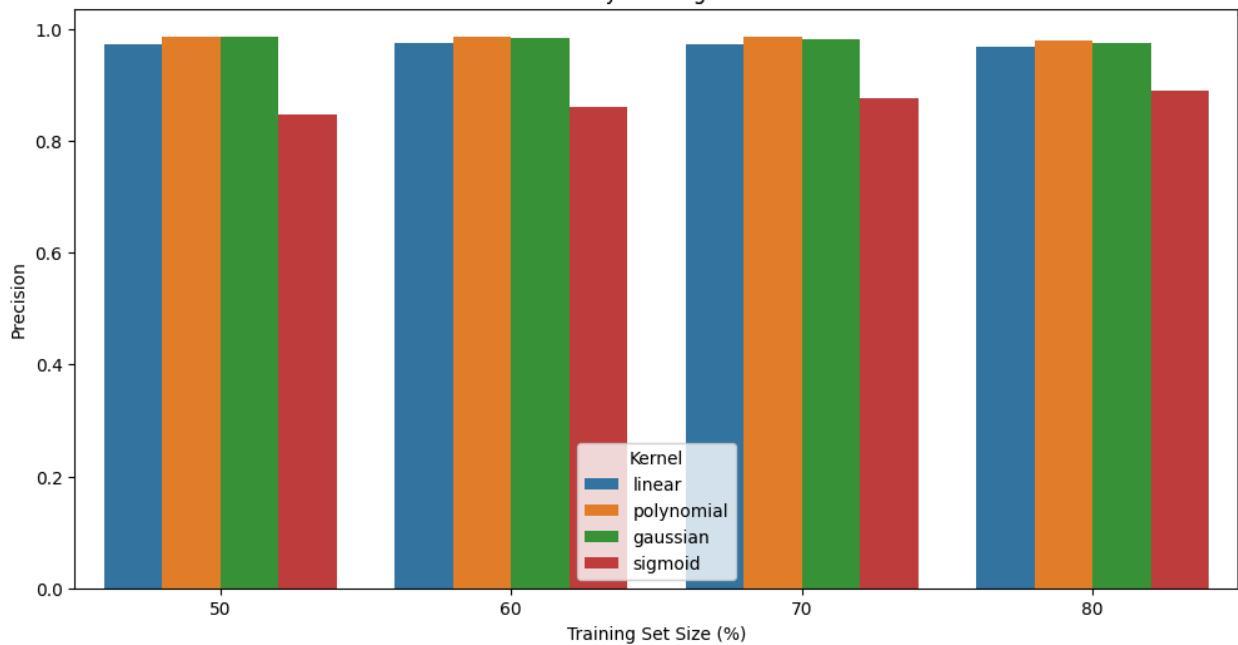
# Plot Recall
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall', hue='Kernel')
plt.title('SVM Recall by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

# Plot F1-score
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score', hue='Kernel')
plt.title('SVM F1-score by Training Size and Kernel') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()
```

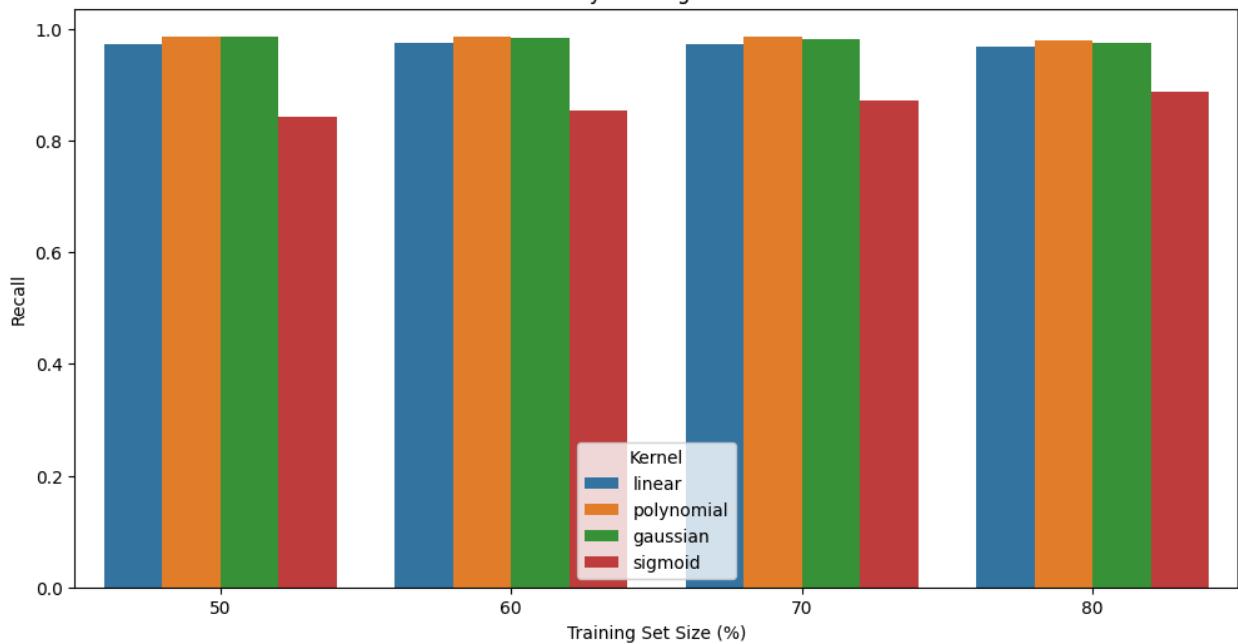
Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.972954	0.973049	0.972954	0.972962
1	60	0.974792	0.974869	0.974792	0.974799
2	70	0.971530	0.971745	0.971530	0.971538
3	80	0.968194	0.968383	0.968194	0.968173
4	50	0.985053	0.985087	0.985053	0.985055
5	60	0.986655	0.986672	0.986655	0.986647
6	70	0.985765	0.985794	0.985765	0.985765
7	80	0.979760	0.979809	0.979760	0.979765
8	50	0.986121	0.986187	0.986121	0.986130
9	60	0.983096	0.983148	0.983096	0.983091
10	70	0.982461	0.982488	0.982461	0.982443
11	80	0.974867	0.975072	0.974867	0.974875
12	50	0.841993	0.848099	0.841993	0.843197
13	60	0.854686	0.859502	0.854686	0.855680
14	70	0.872140	0.875307	0.872140	0.873104
15	80	0.886566	0.888754	0.886566	0.887225



SVM Precision by Training Size and Kernel



SVM Recall by Training Size and Kernel





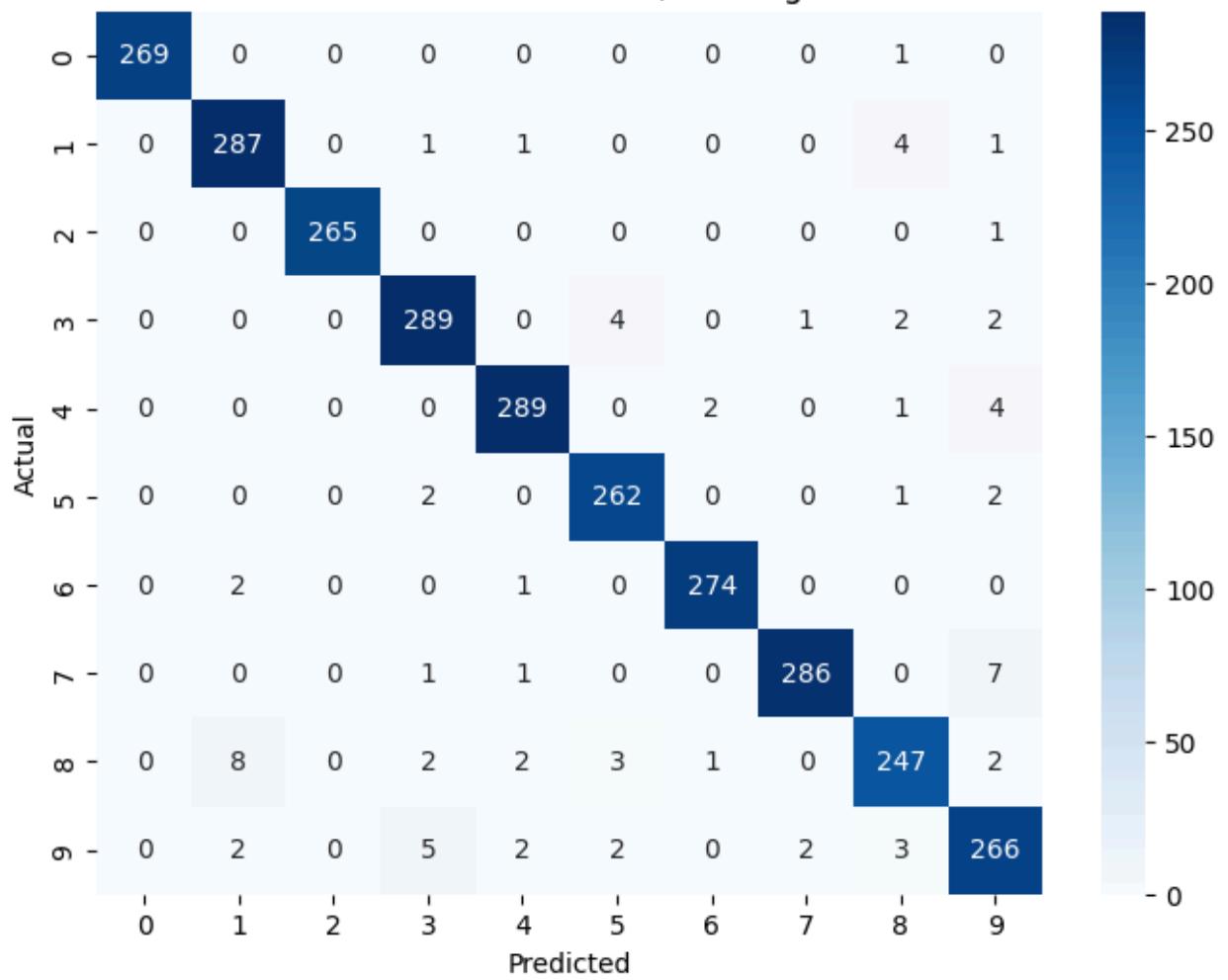
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

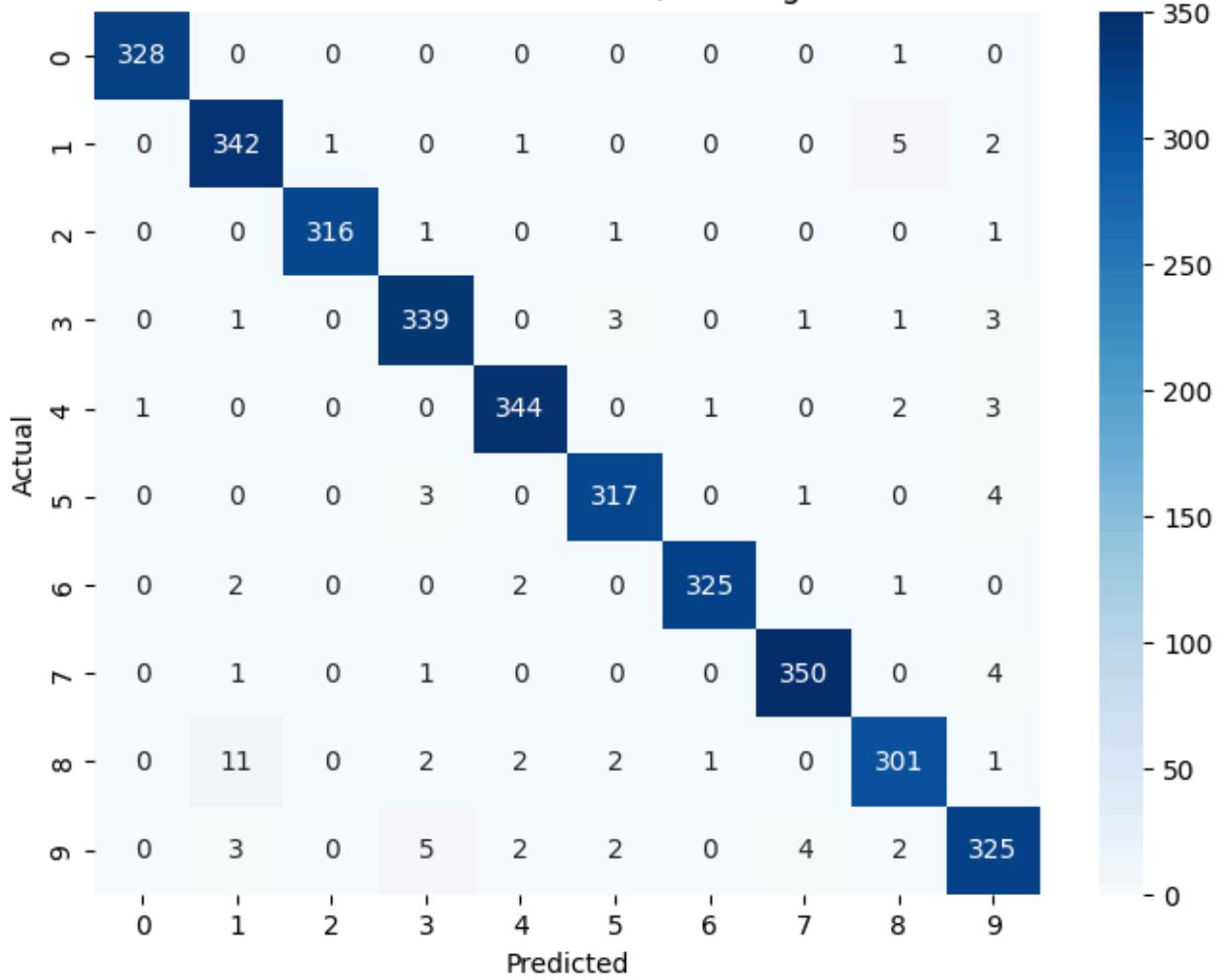
for cm_data in confusion_matrices:
    cm = cm_data["Confusion Matrix"]
    kernel_name = cm_data["Kernel"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix: {kernel_name} Kernel, Training Size: {training_size}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

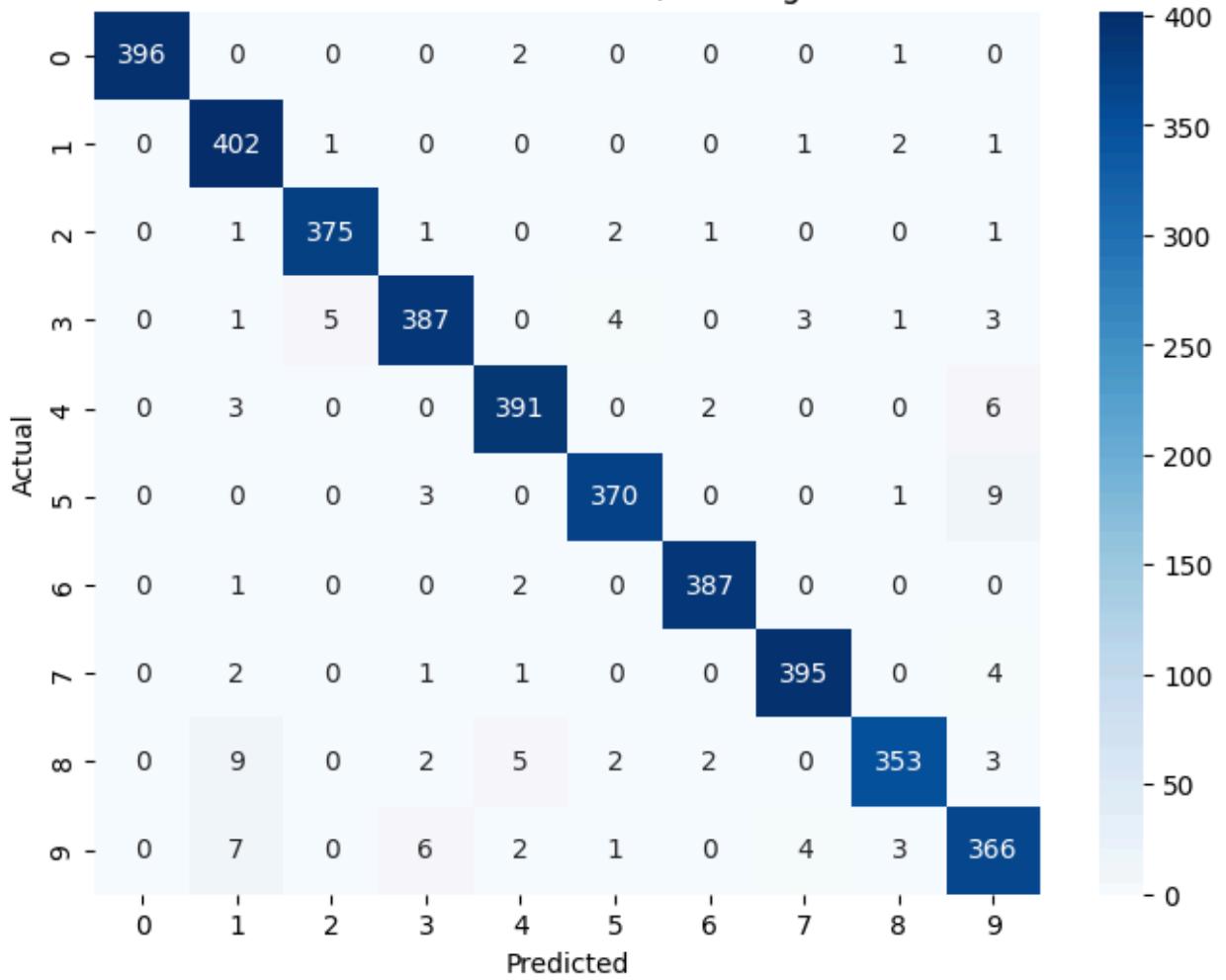
Confusion Matrix: linear Kernel, Training Size: 50%



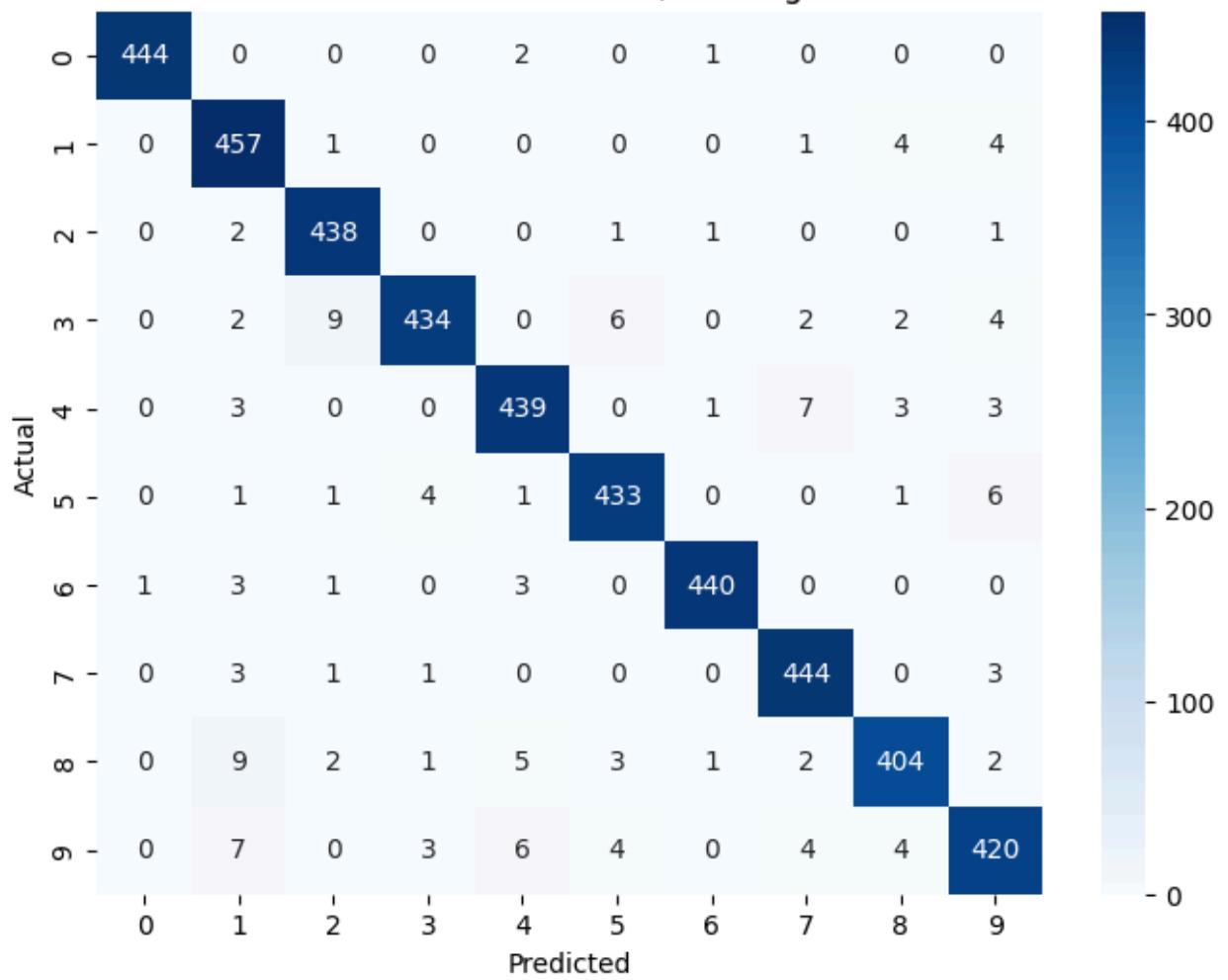
Confusion Matrix: linear Kernel, Training Size: 60%

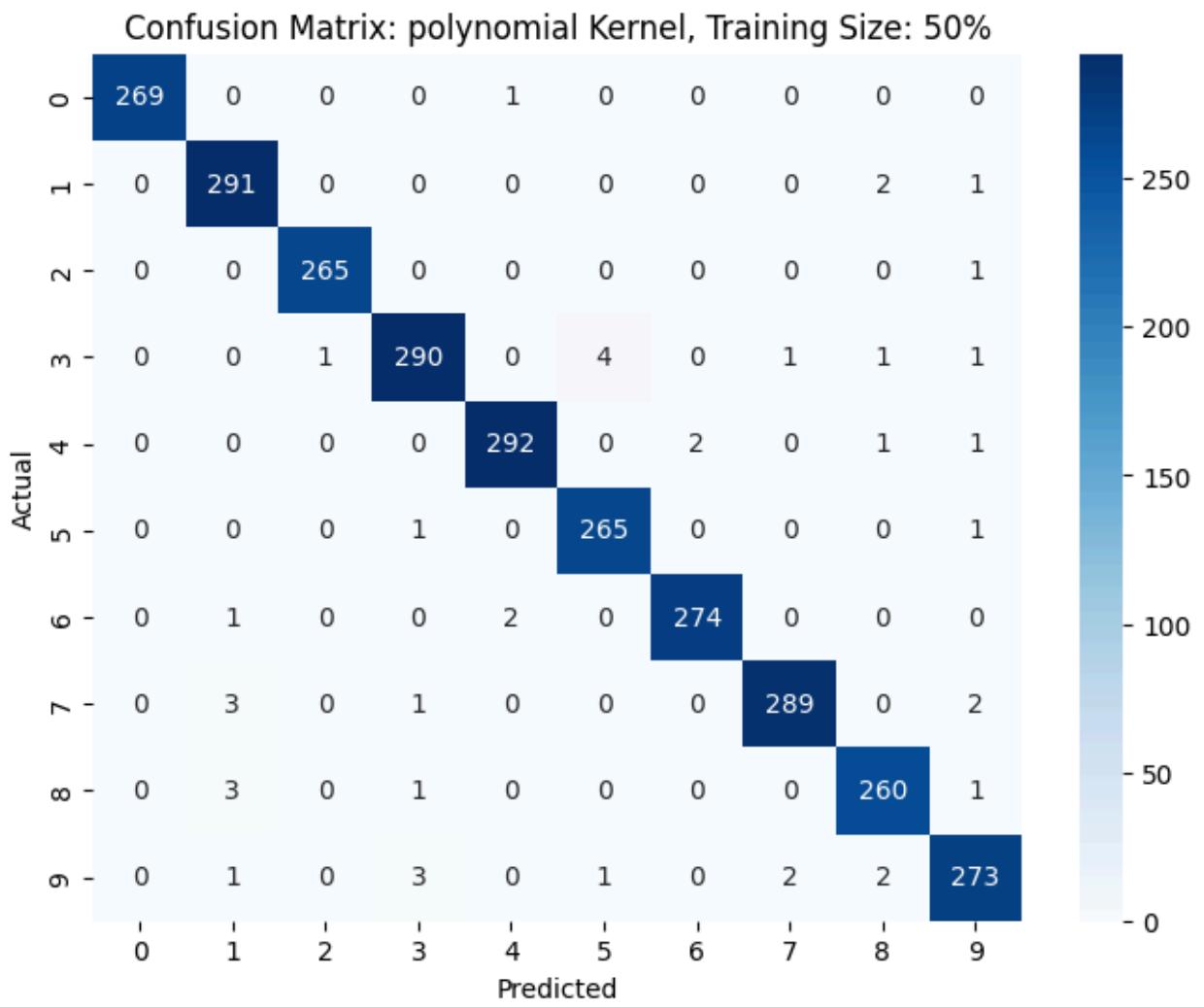


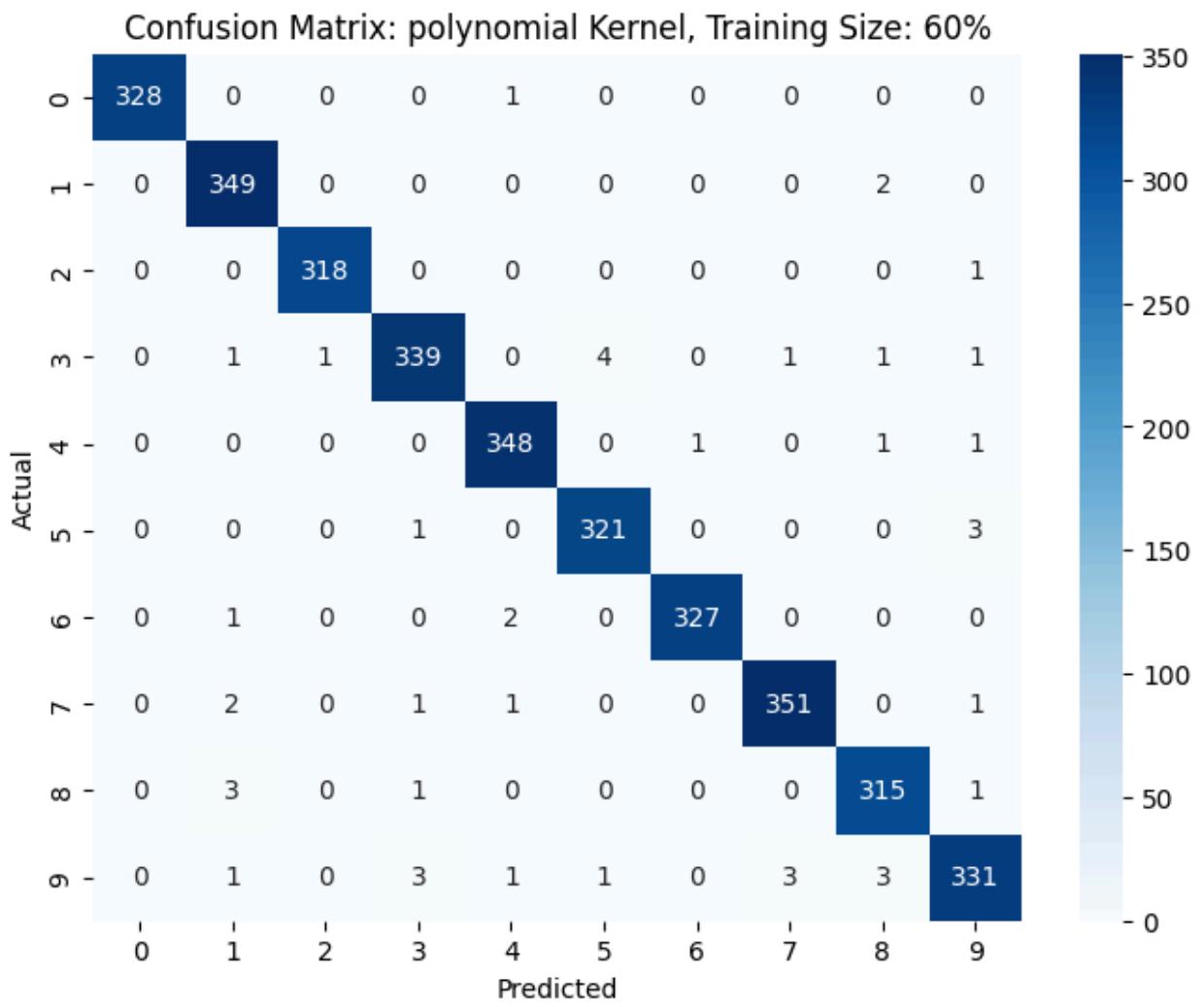
Confusion Matrix: linear Kernel, Training Size: 70%

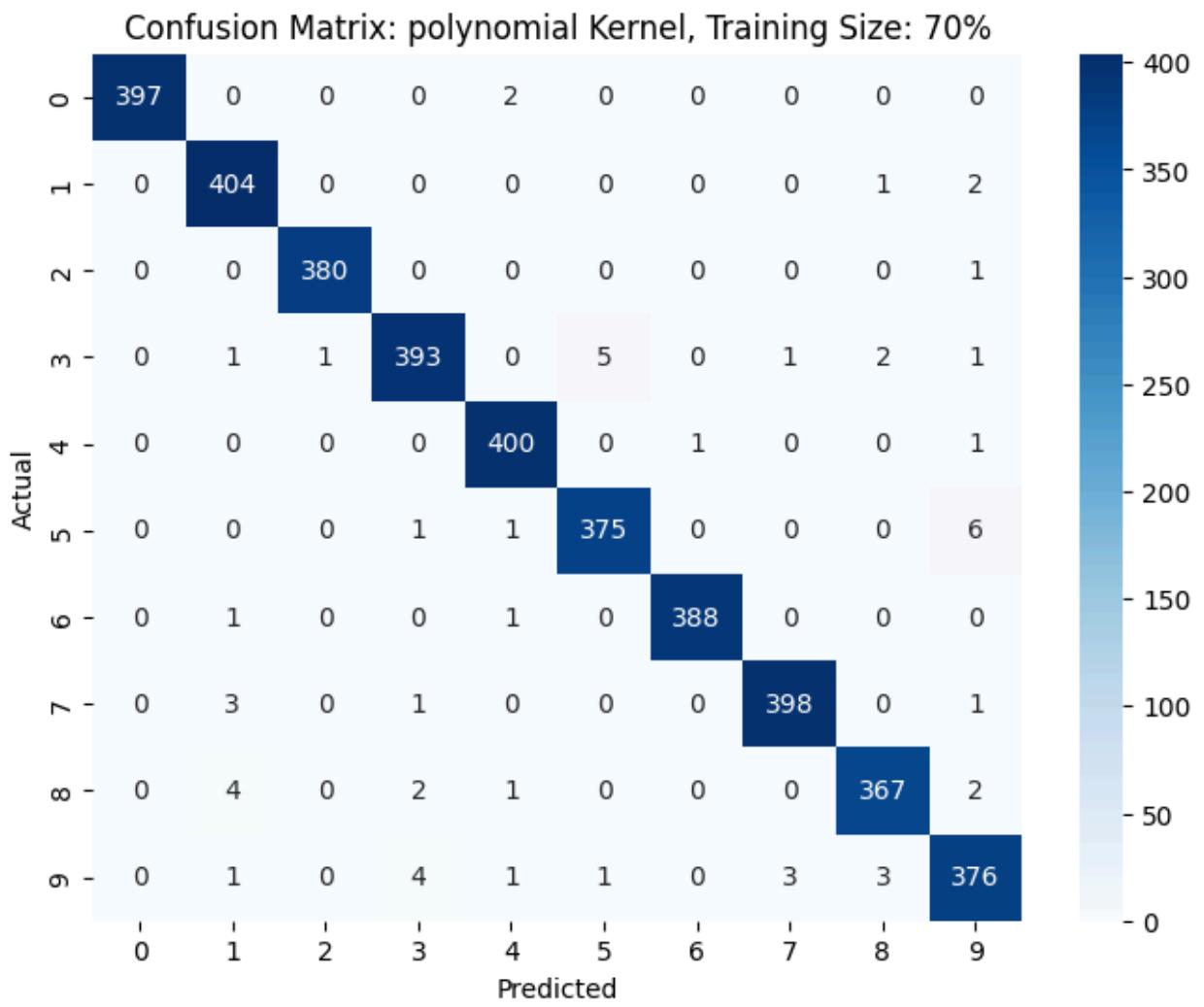


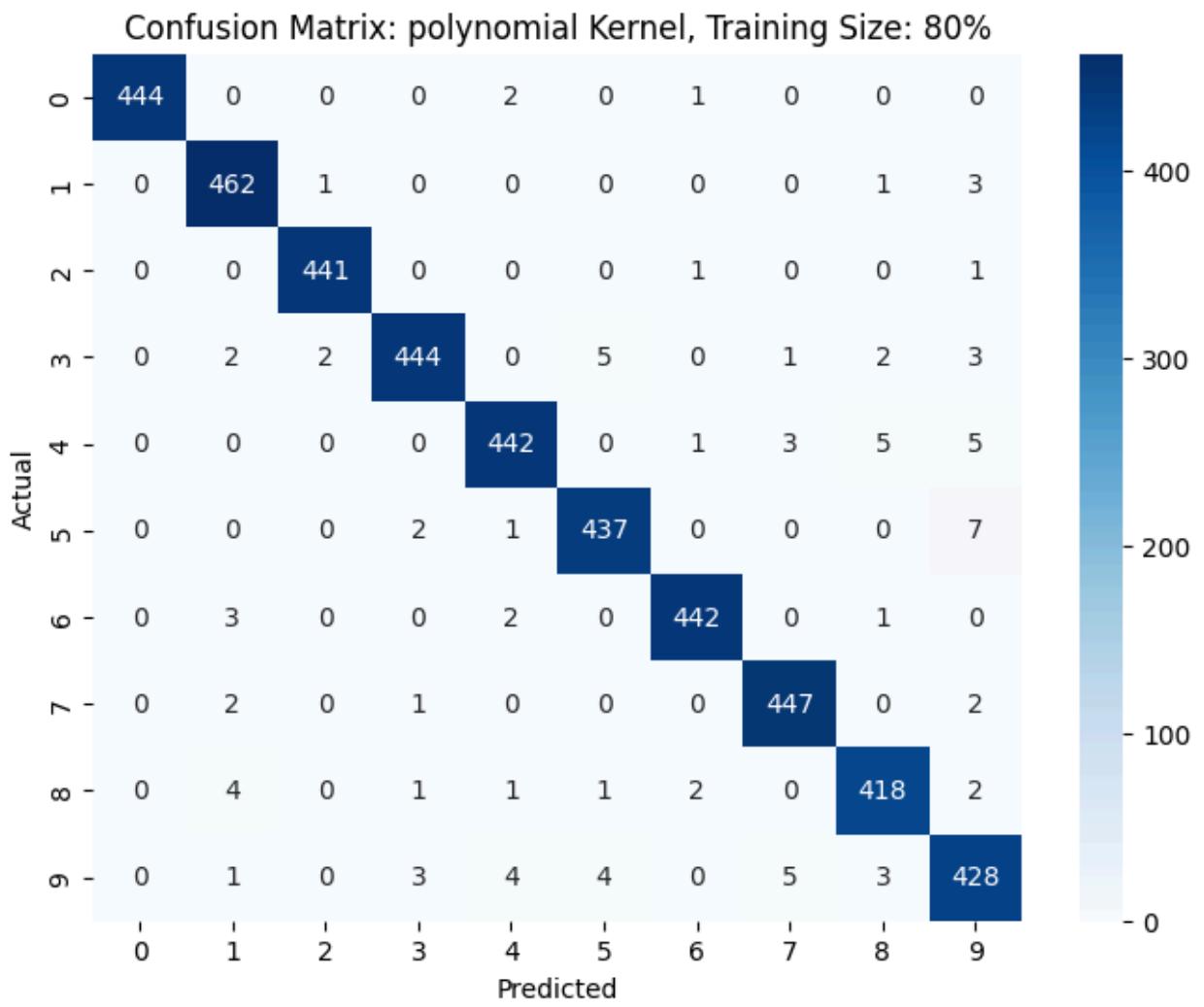
Confusion Matrix: linear Kernel, Training Size: 80%



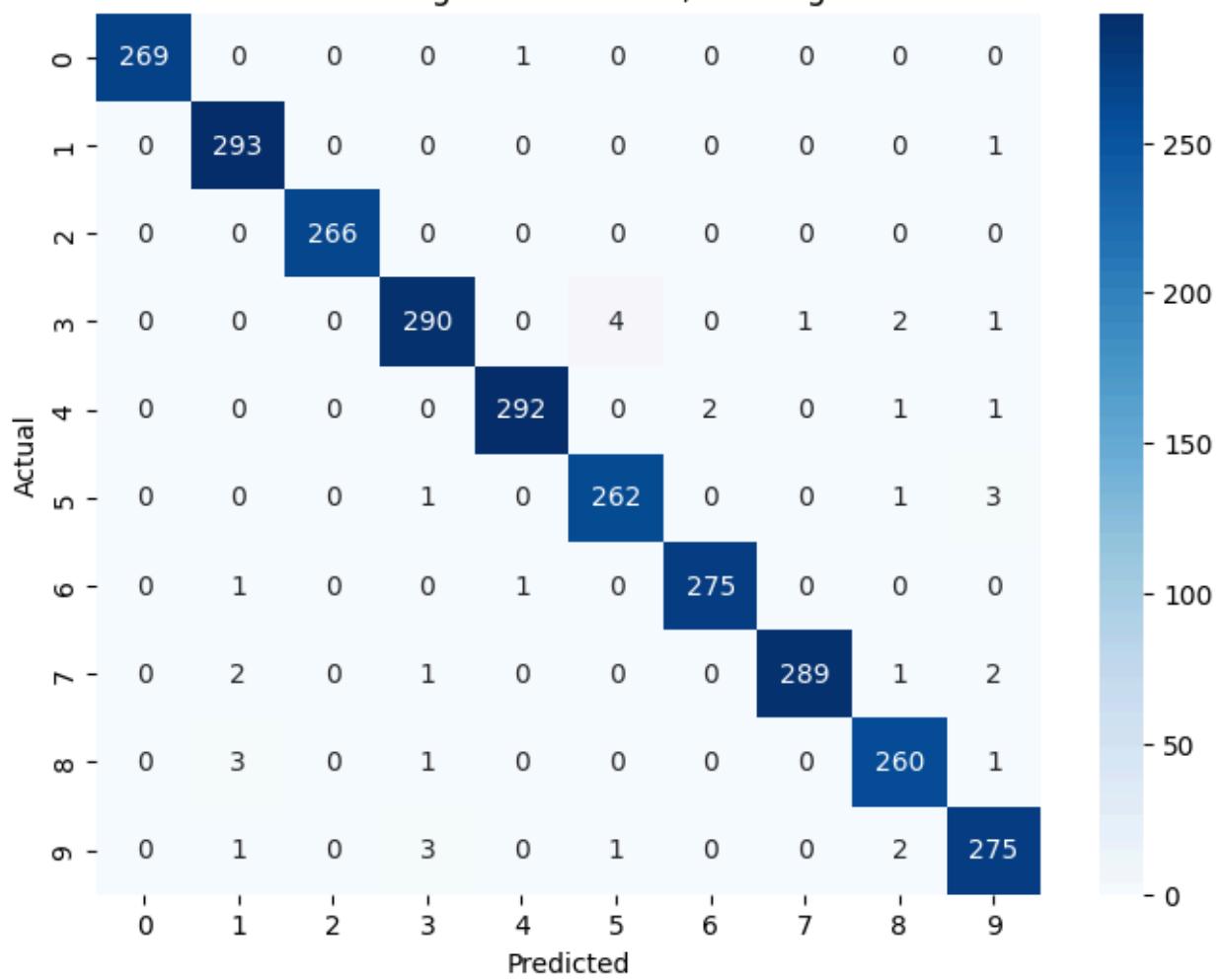


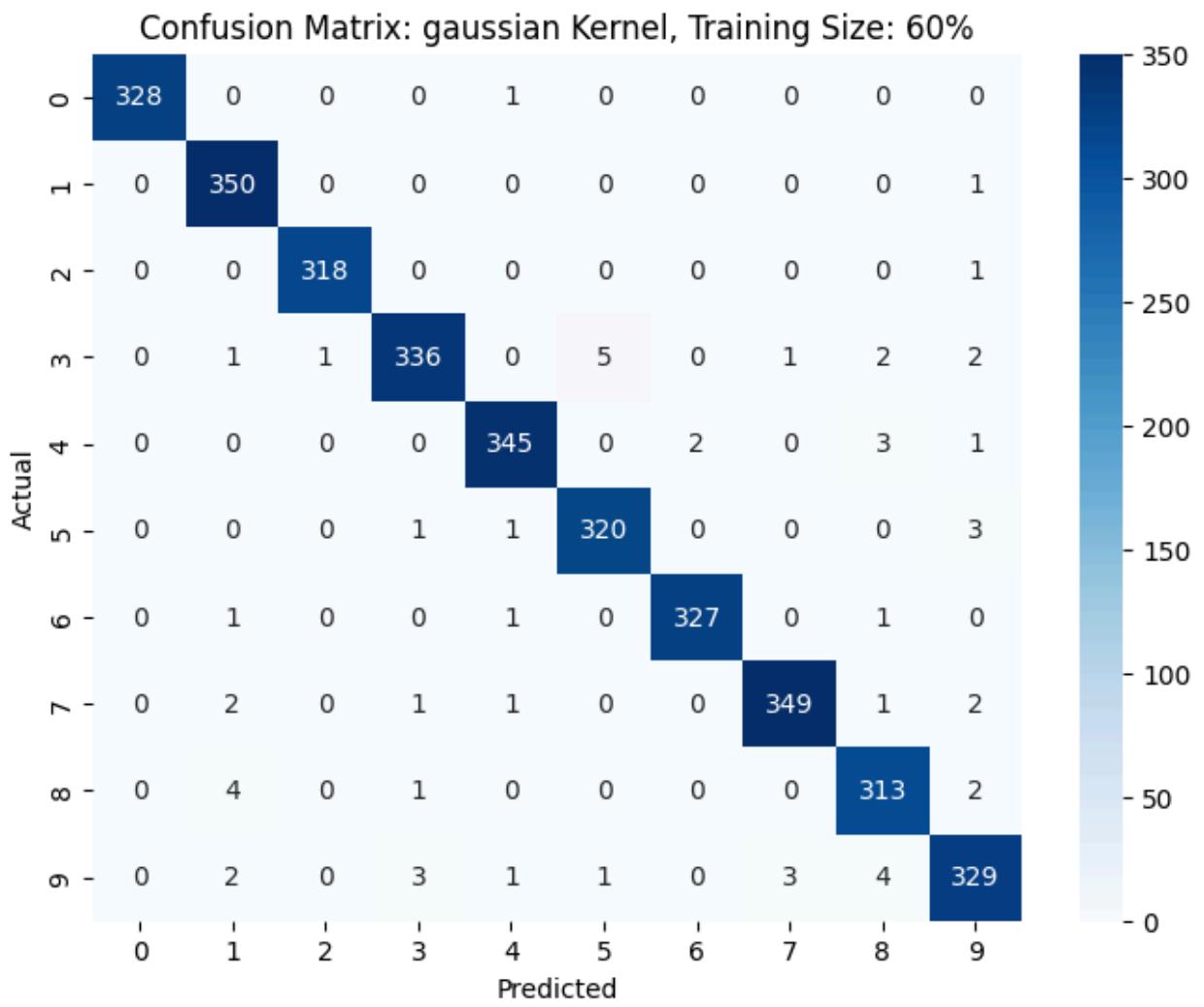




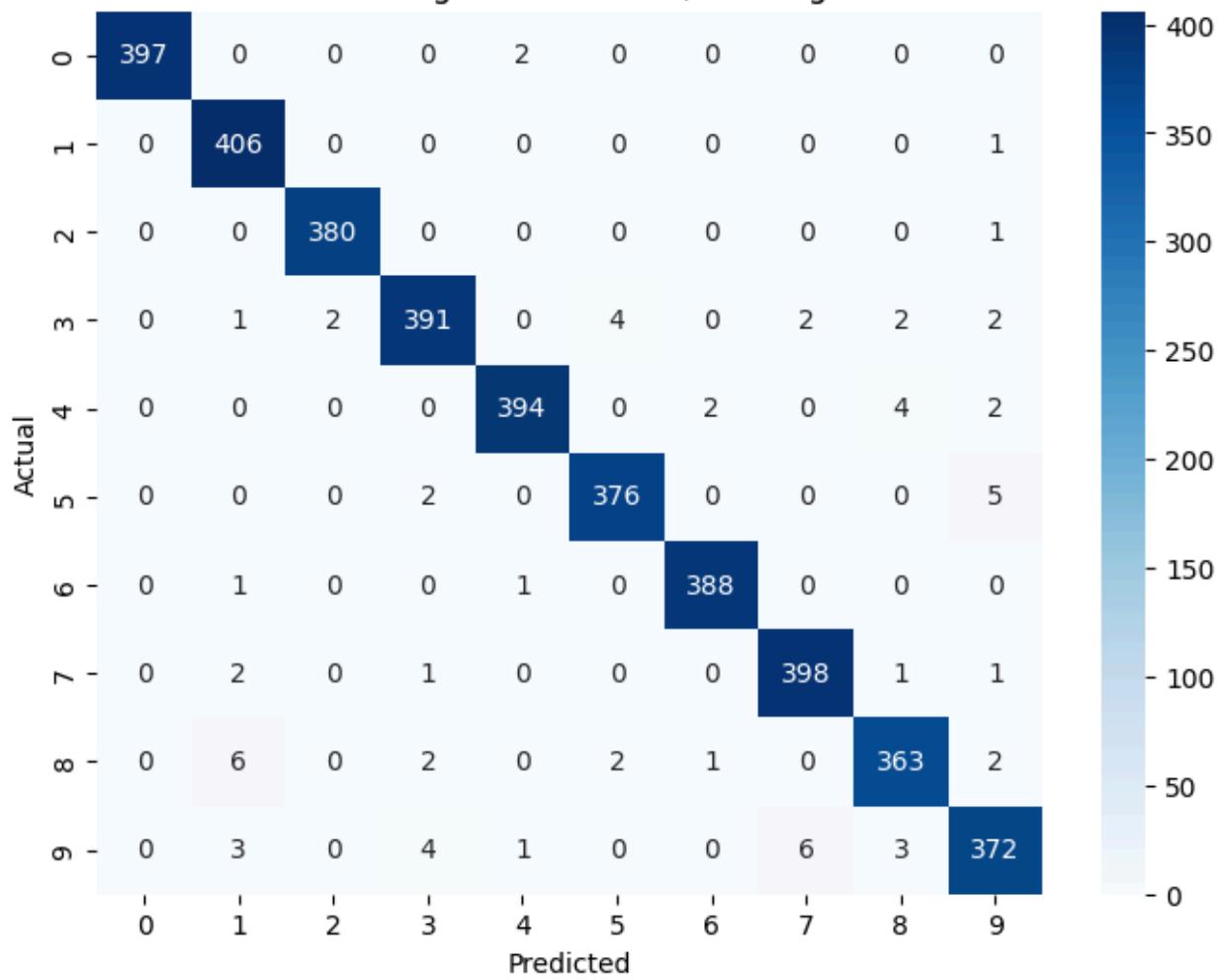


Confusion Matrix: gaussian Kernel, Training Size: 50%

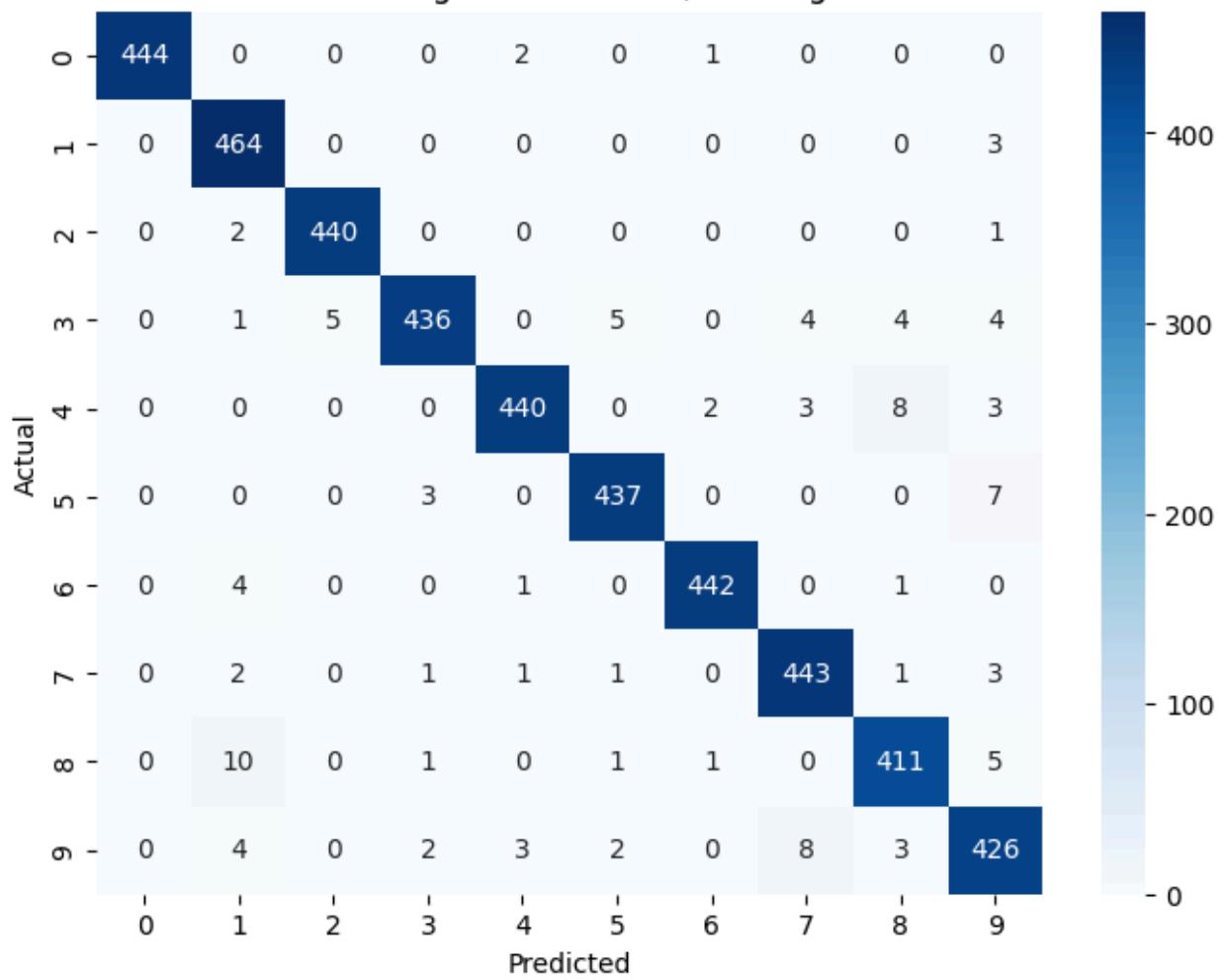




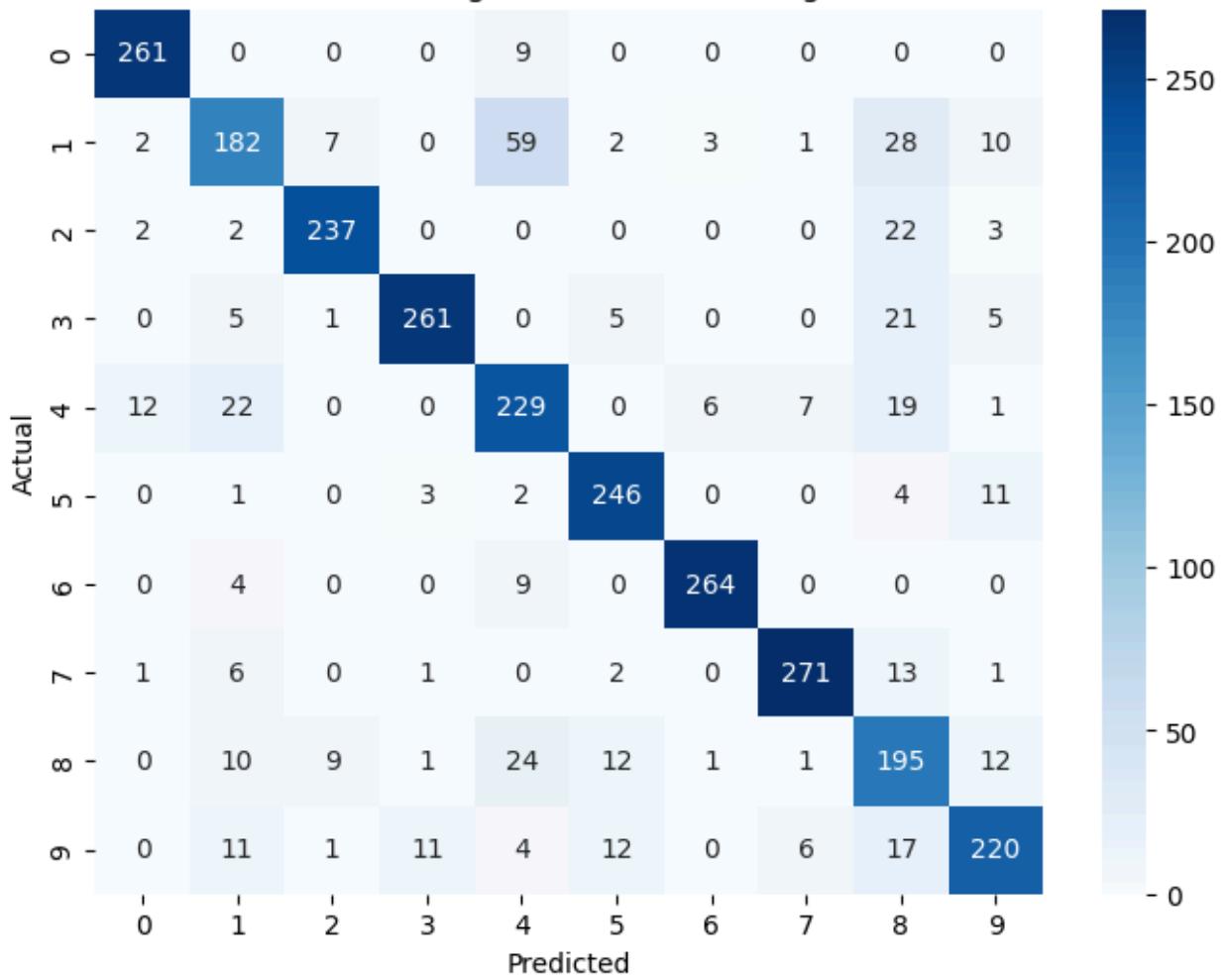
Confusion Matrix: gaussian Kernel, Training Size: 70%



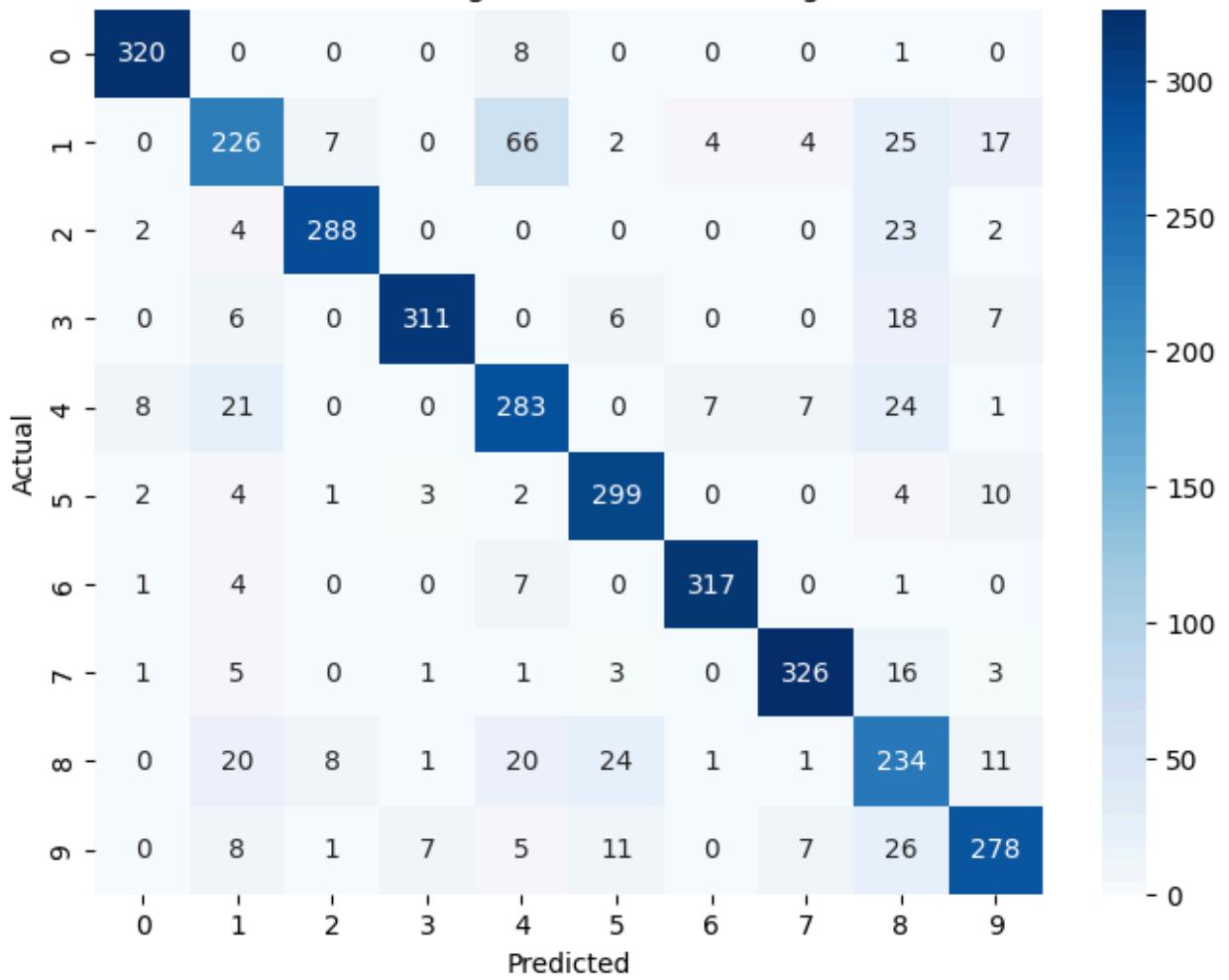
Confusion Matrix: gaussian Kernel, Training Size: 80%



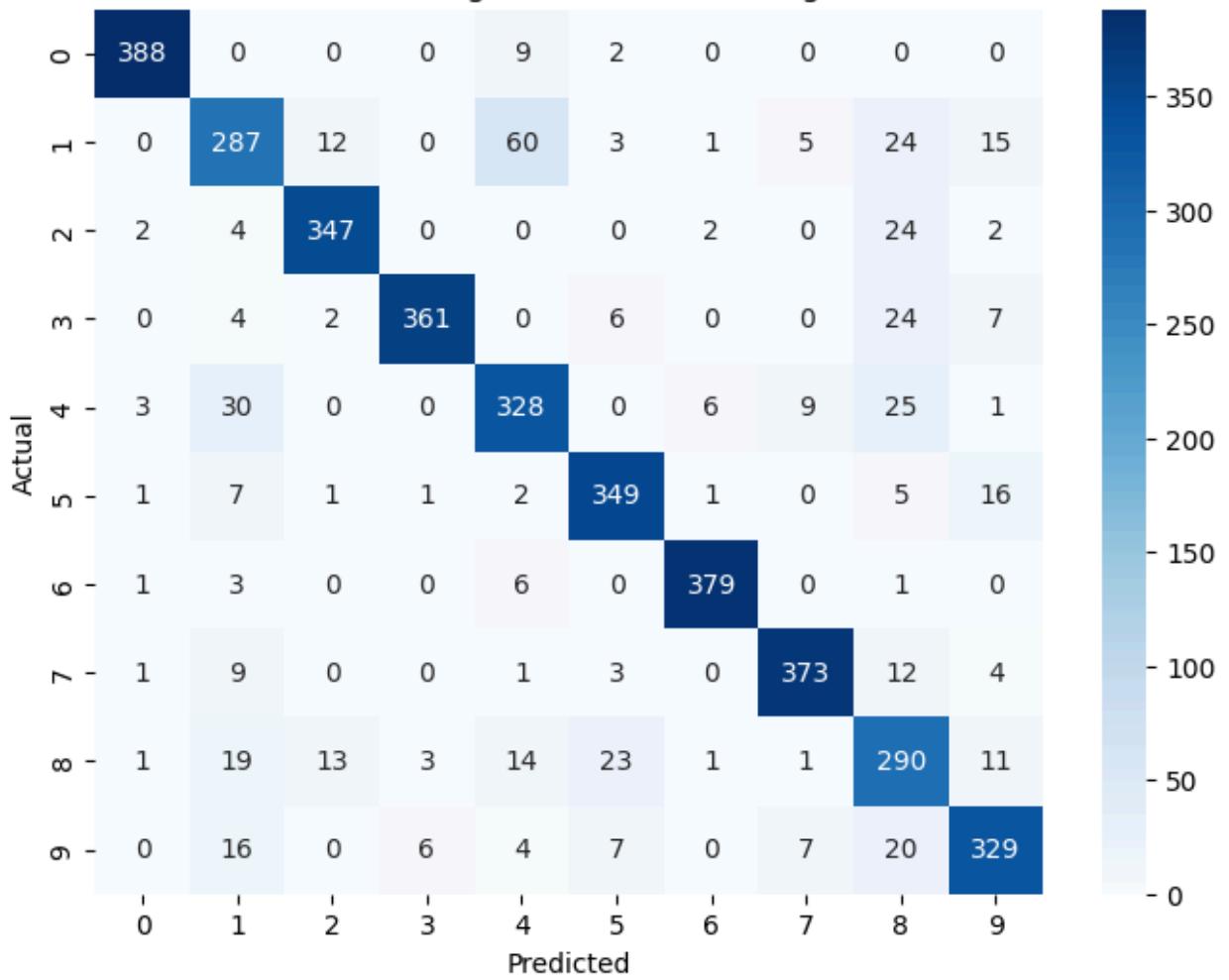
Confusion Matrix: sigmoid Kernel, Training Size: 50%

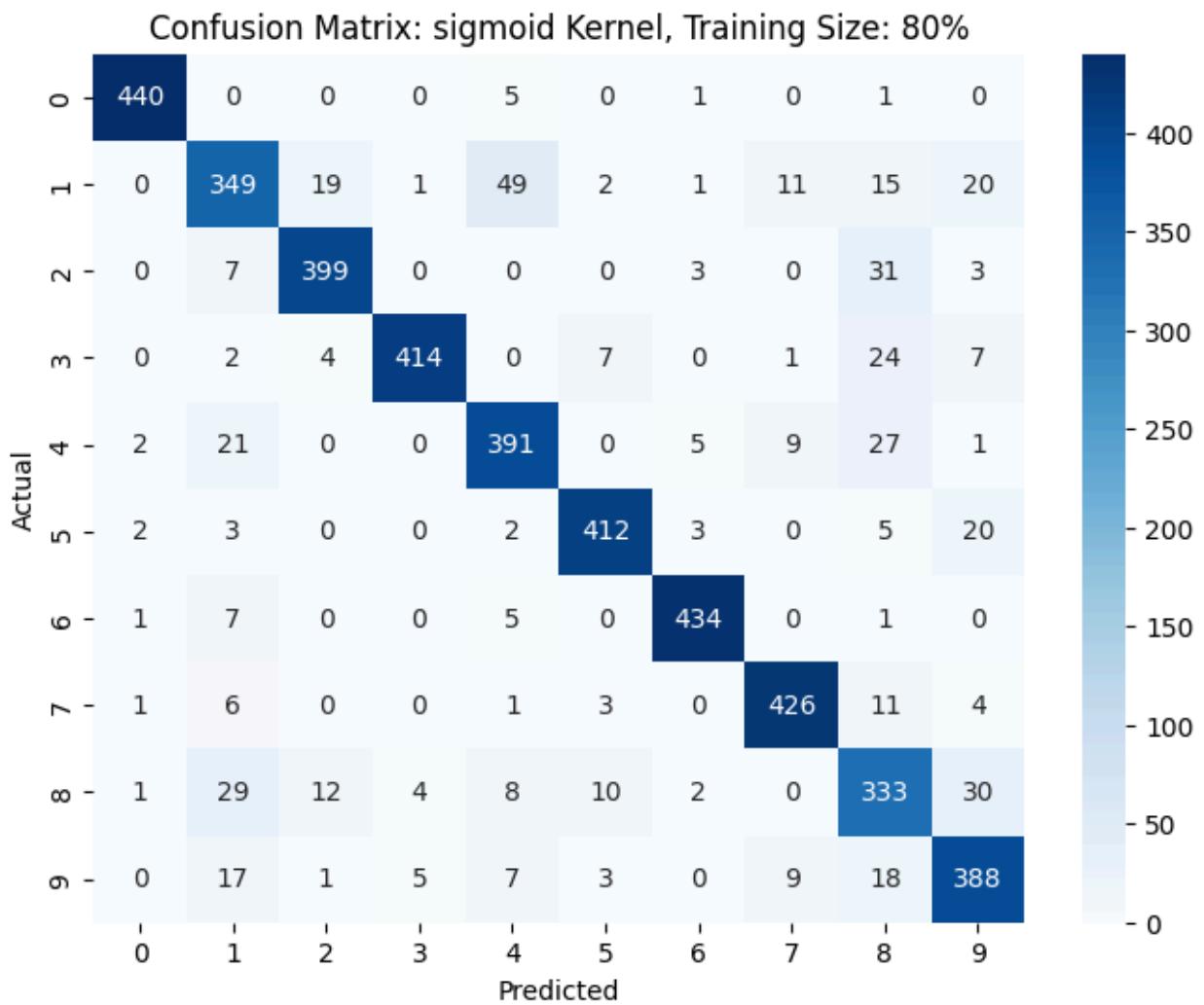


Confusion Matrix: sigmoid Kernel, Training Size: 60%



Confusion Matrix: sigmoid Kernel, Training Size: 70%





## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    kernel_name = model_data["Kernel"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
```

```

roc_auc = dict()
n_classes = len(np.unique(y_test))

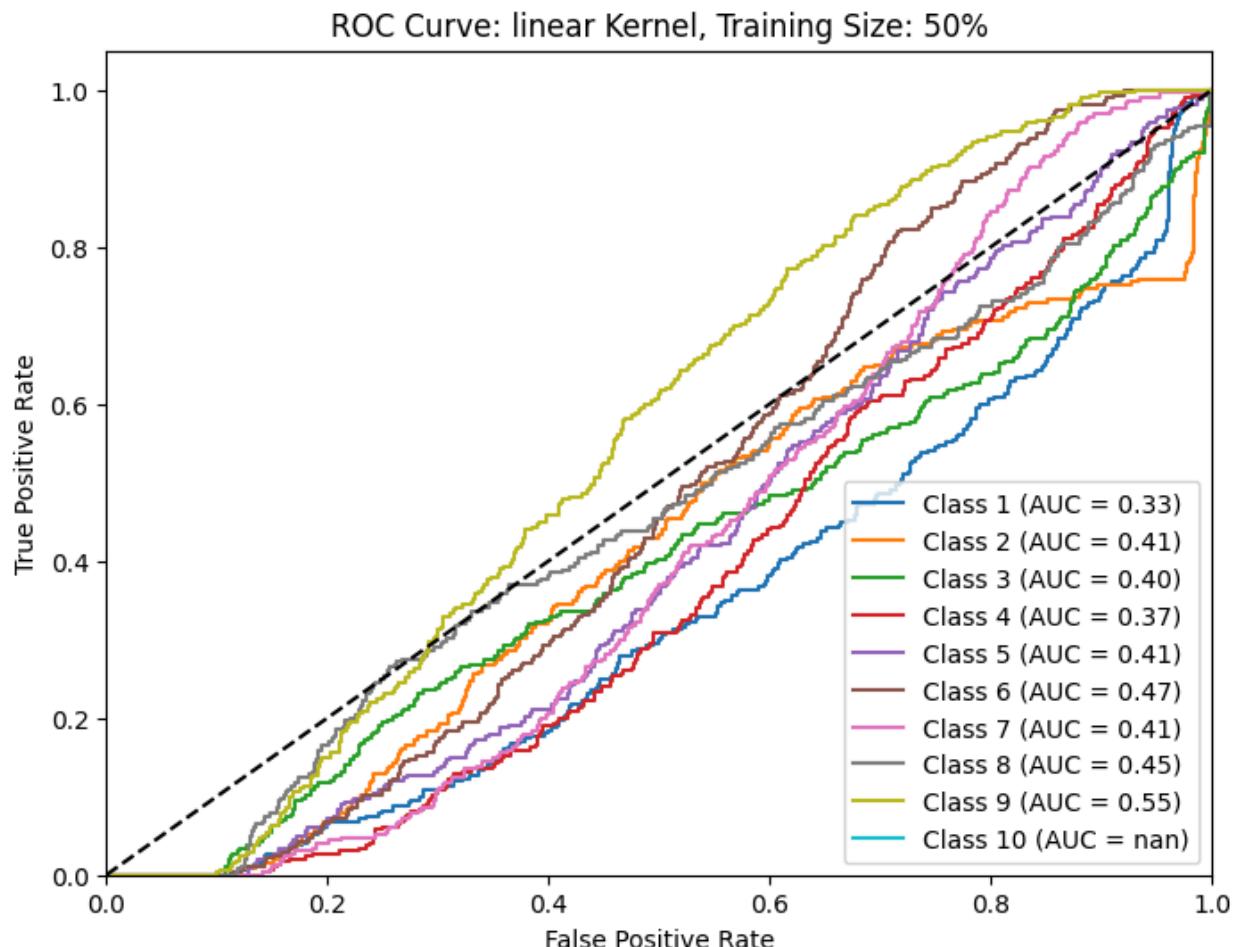
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve: {kernel_name} Kernel, Training Size: {training_size}')
plt.legend(loc="lower right")
plt.show()

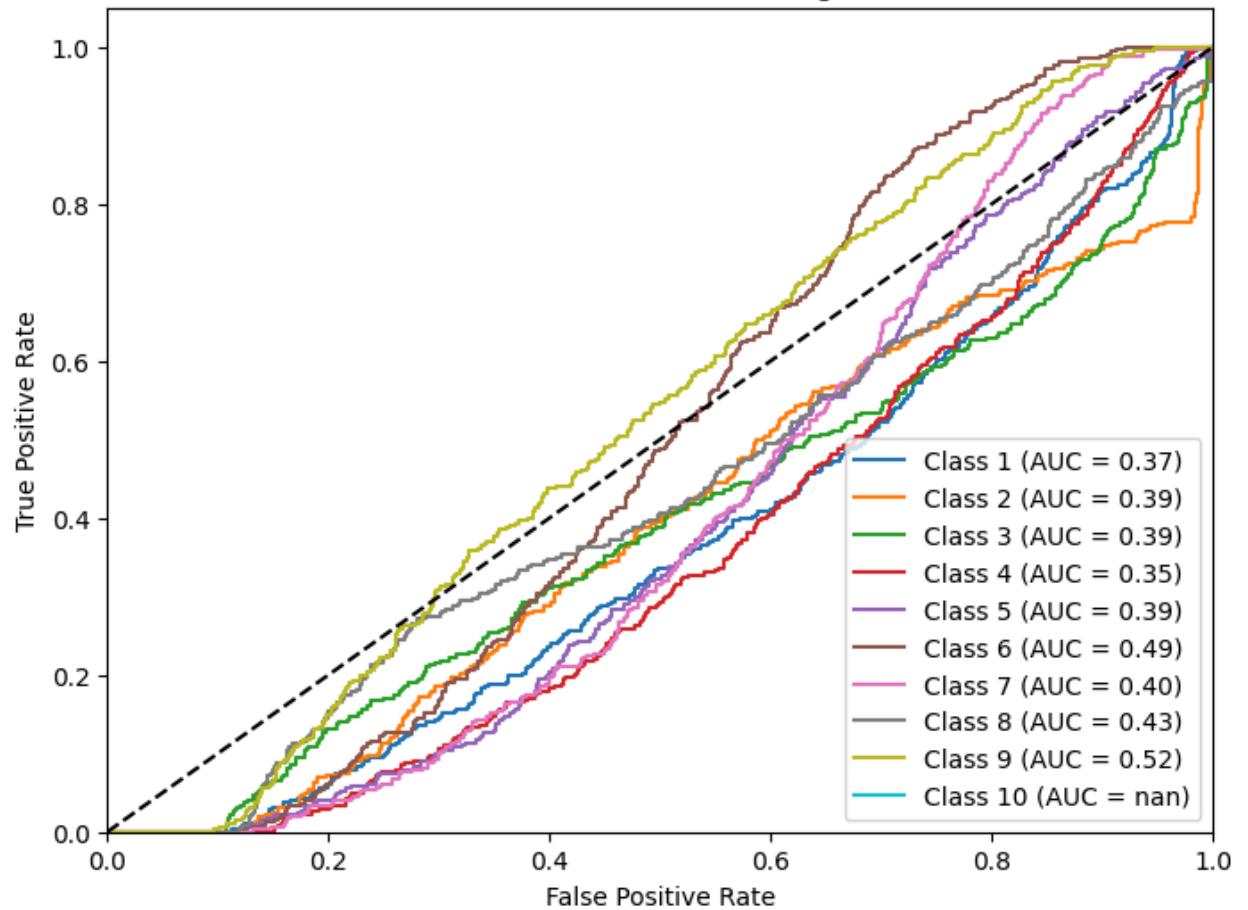
```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/\_ranking.py:1188: UndefinedMetricWarning: No positive samples in y\_true, true positive value should be meaningless  
 warnings.warn(



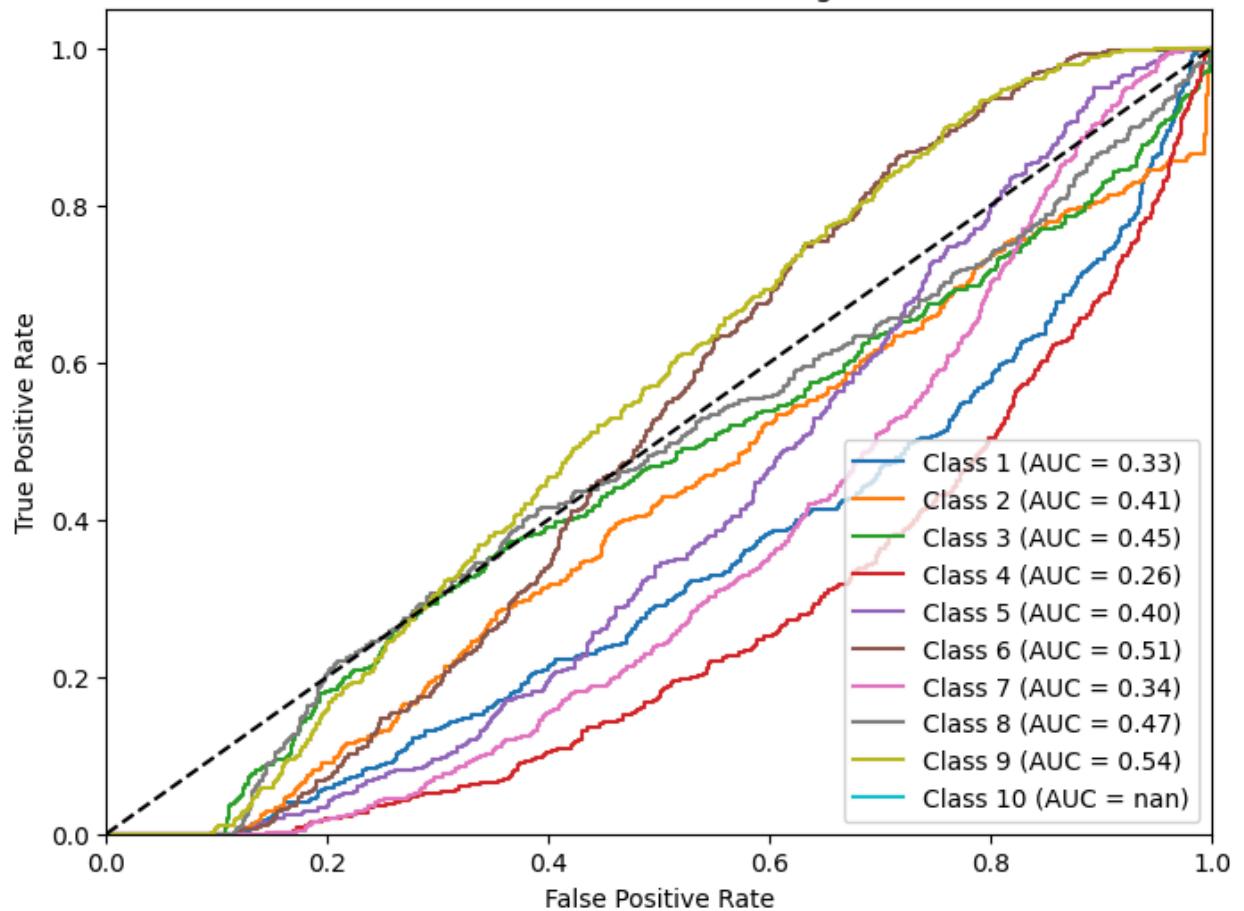
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: linear Kernel, Training Size: 60%



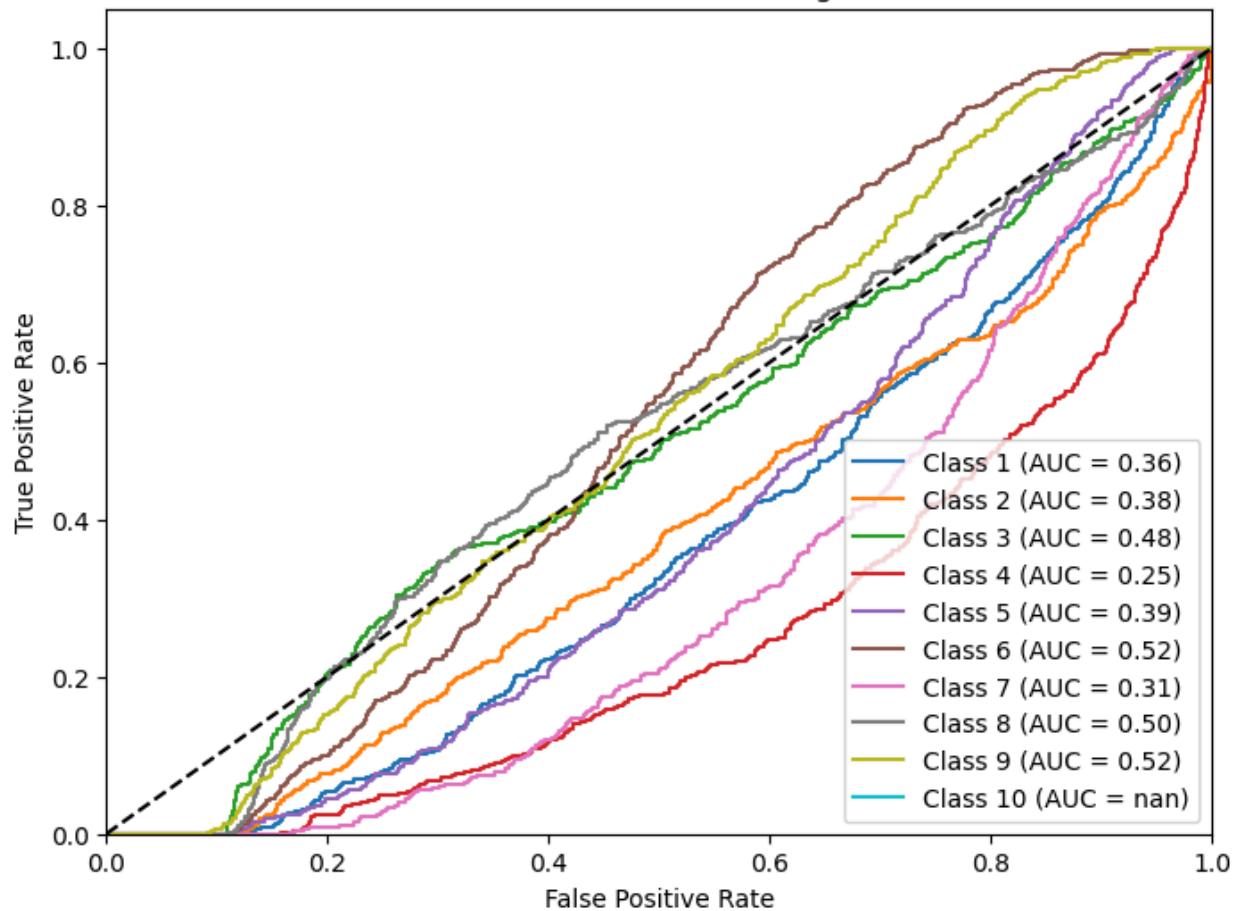
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: linear Kernel, Training Size: 70%



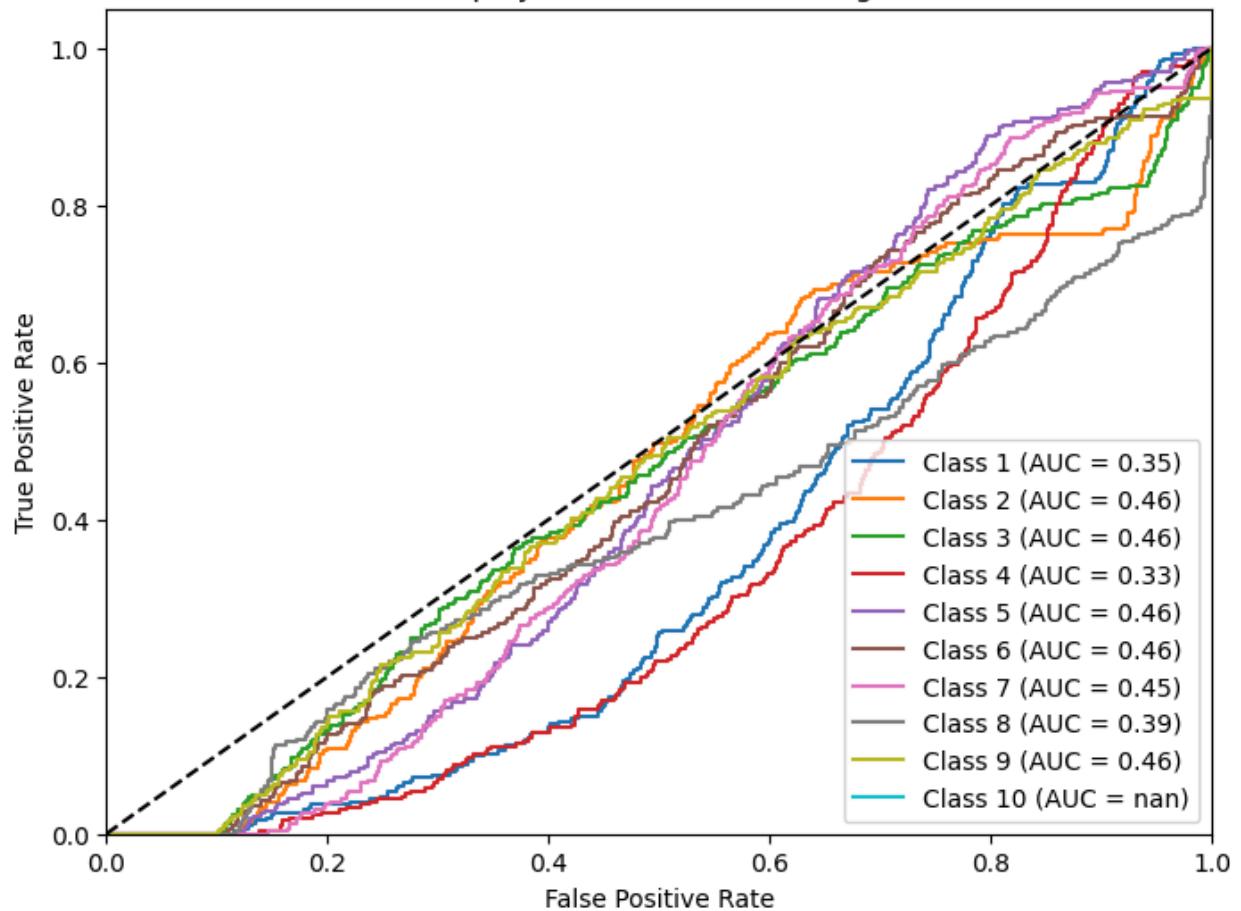
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: linear Kernel, Training Size: 80%



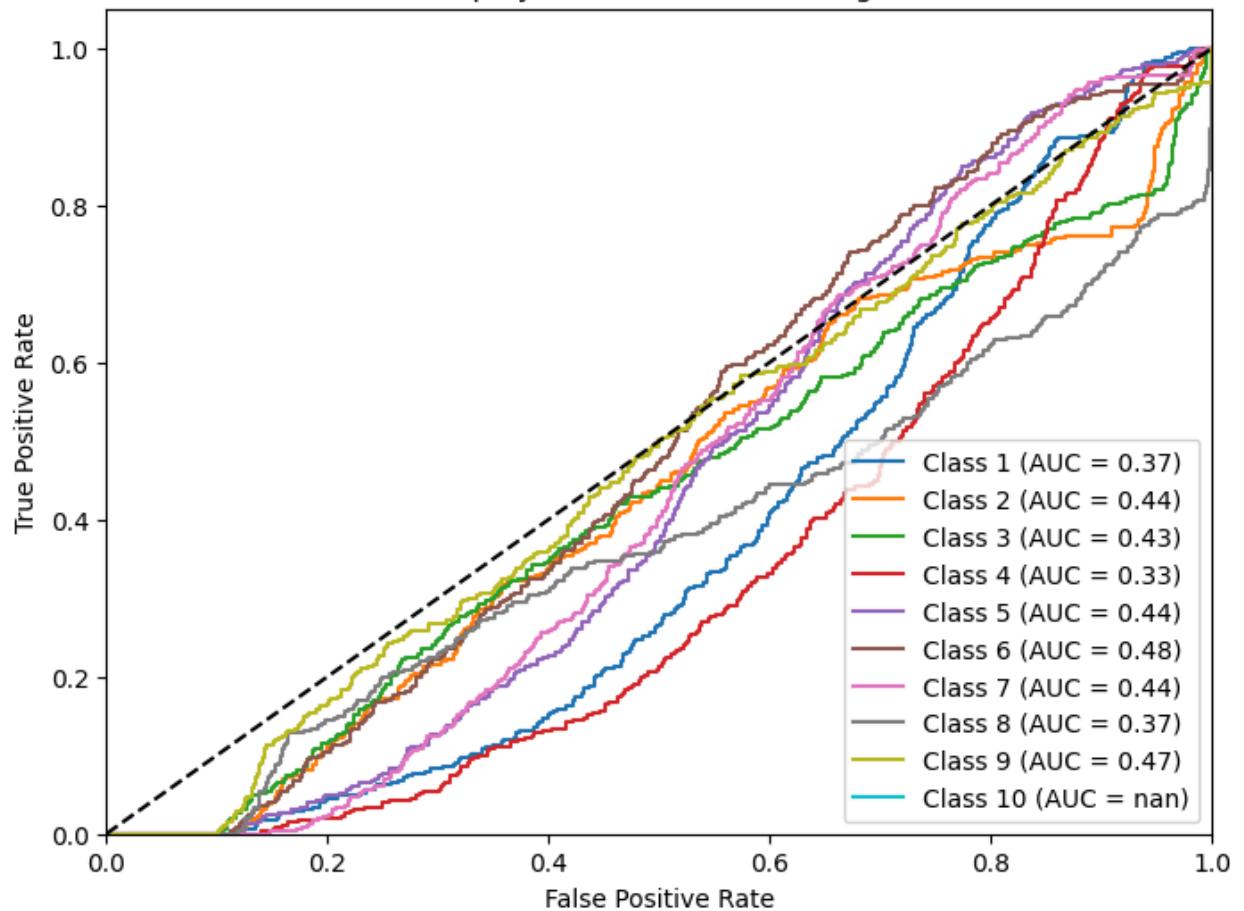
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: polynomial Kernel, Training Size: 50%



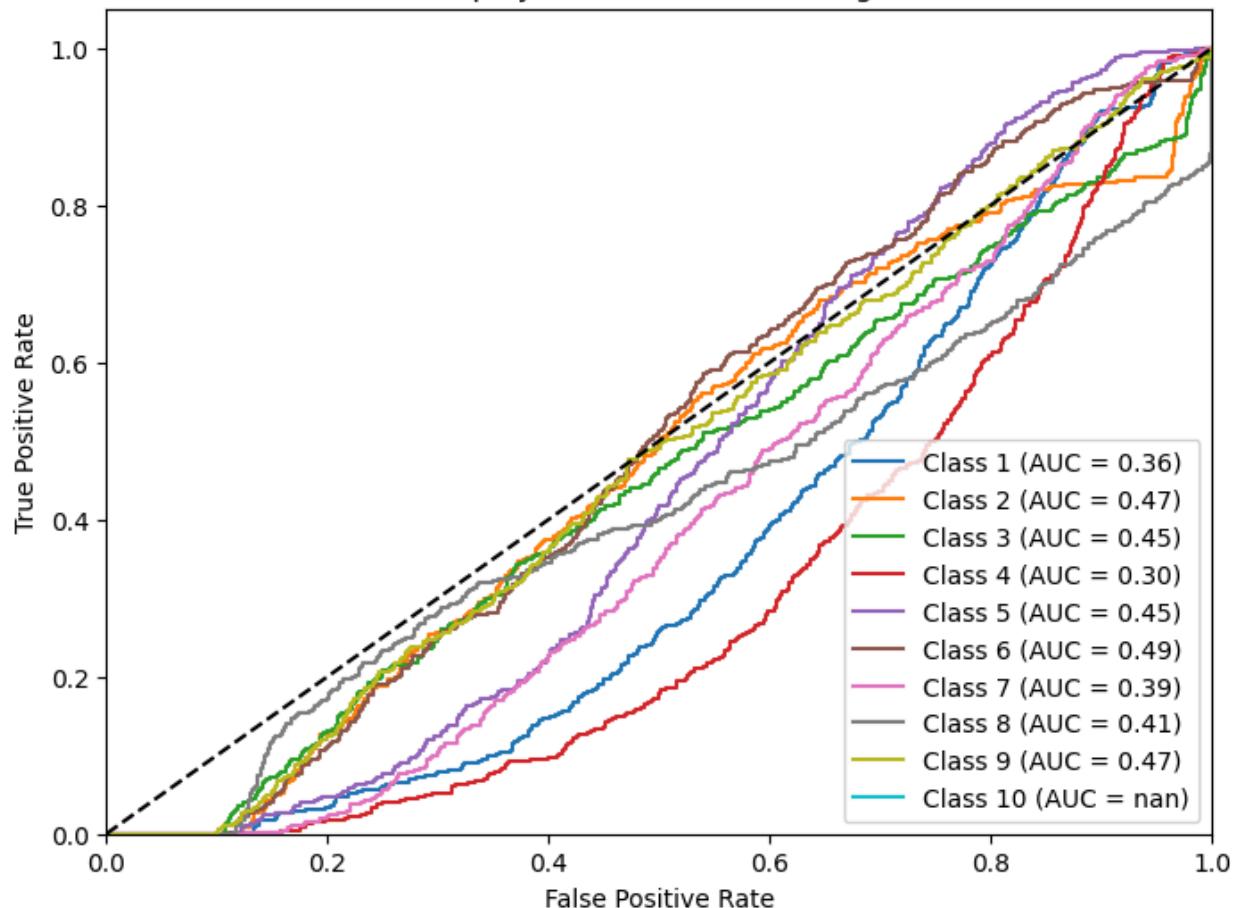
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: polynomial Kernel, Training Size: 60%



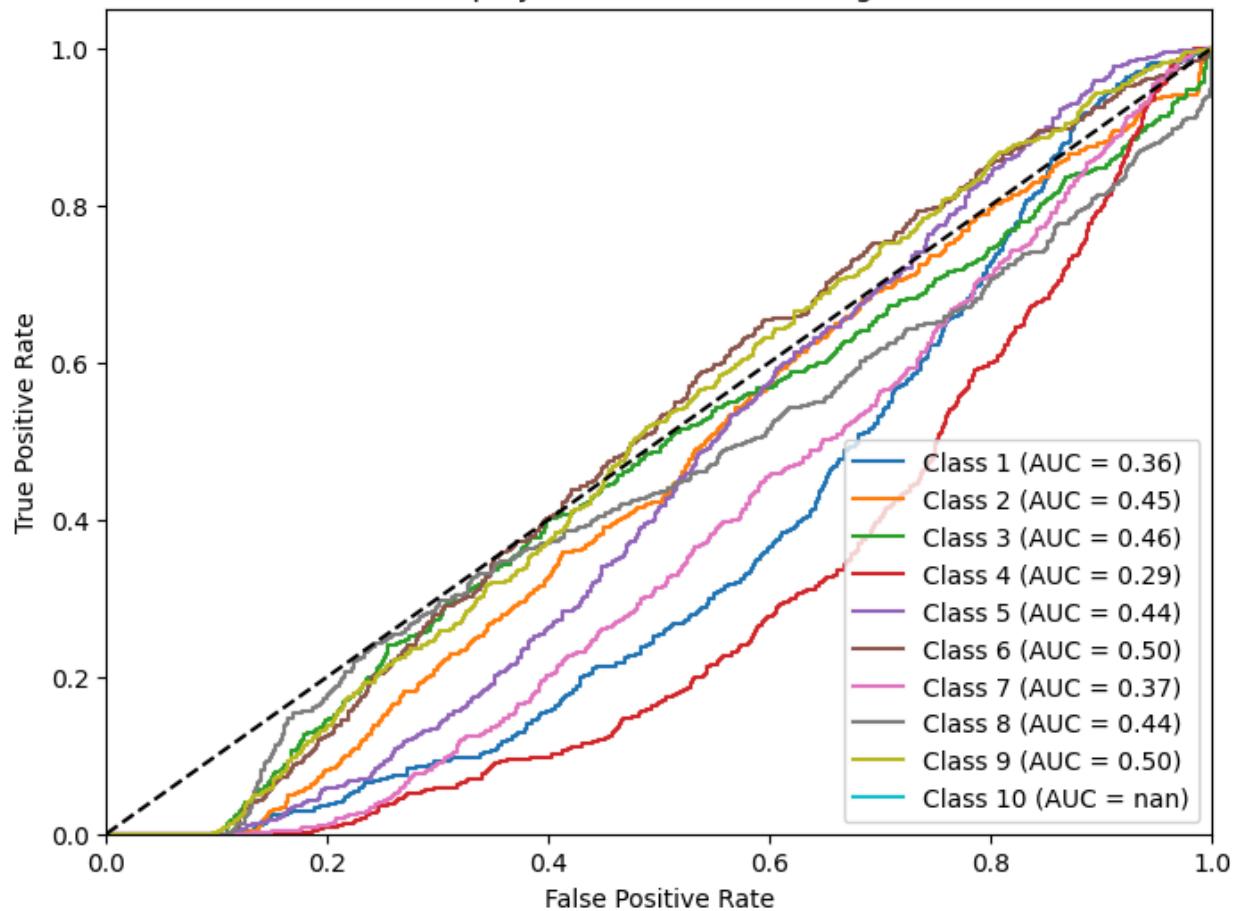
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: polynomial Kernel, Training Size: 70%



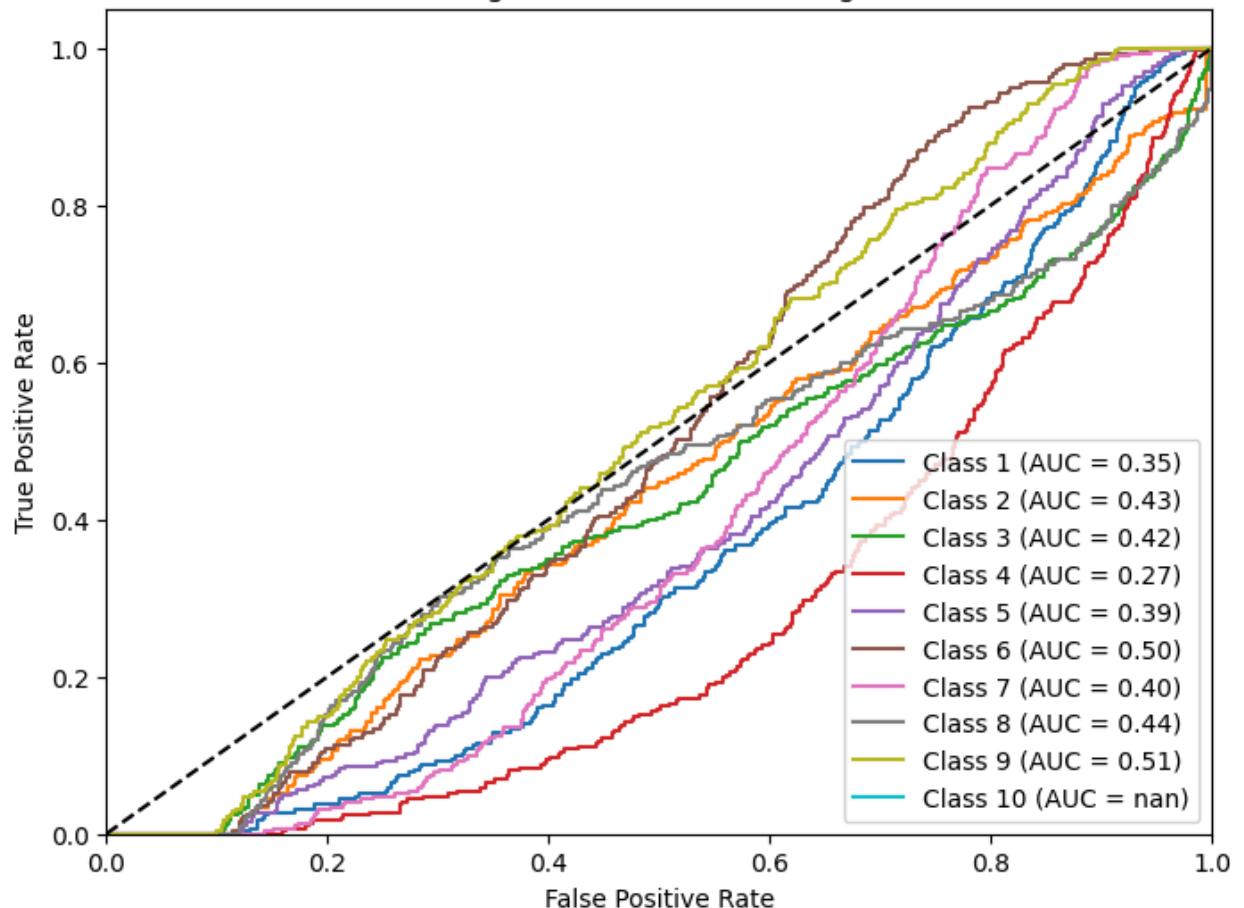
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: polynomial Kernel, Training Size: 80%



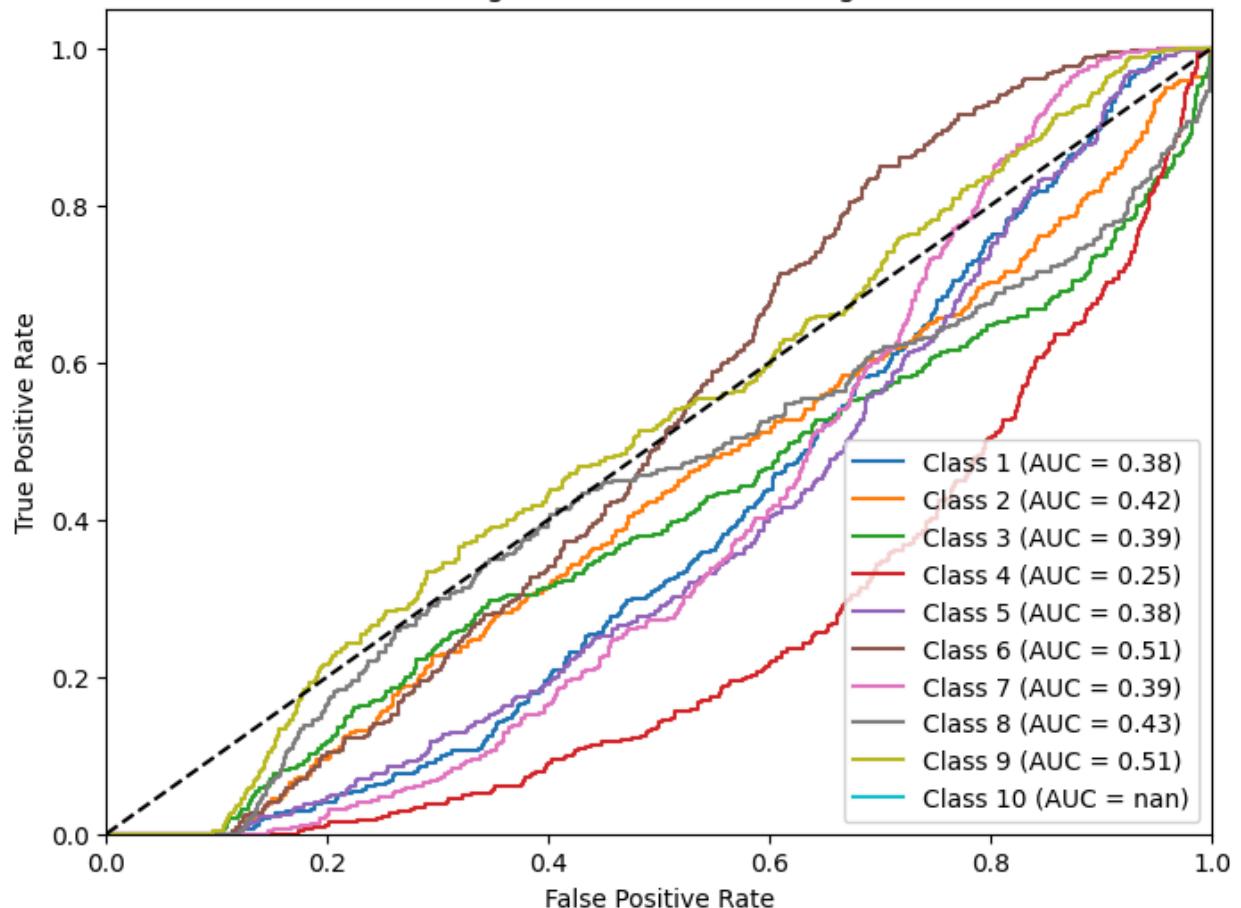
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: gaussian Kernel, Training Size: 50%



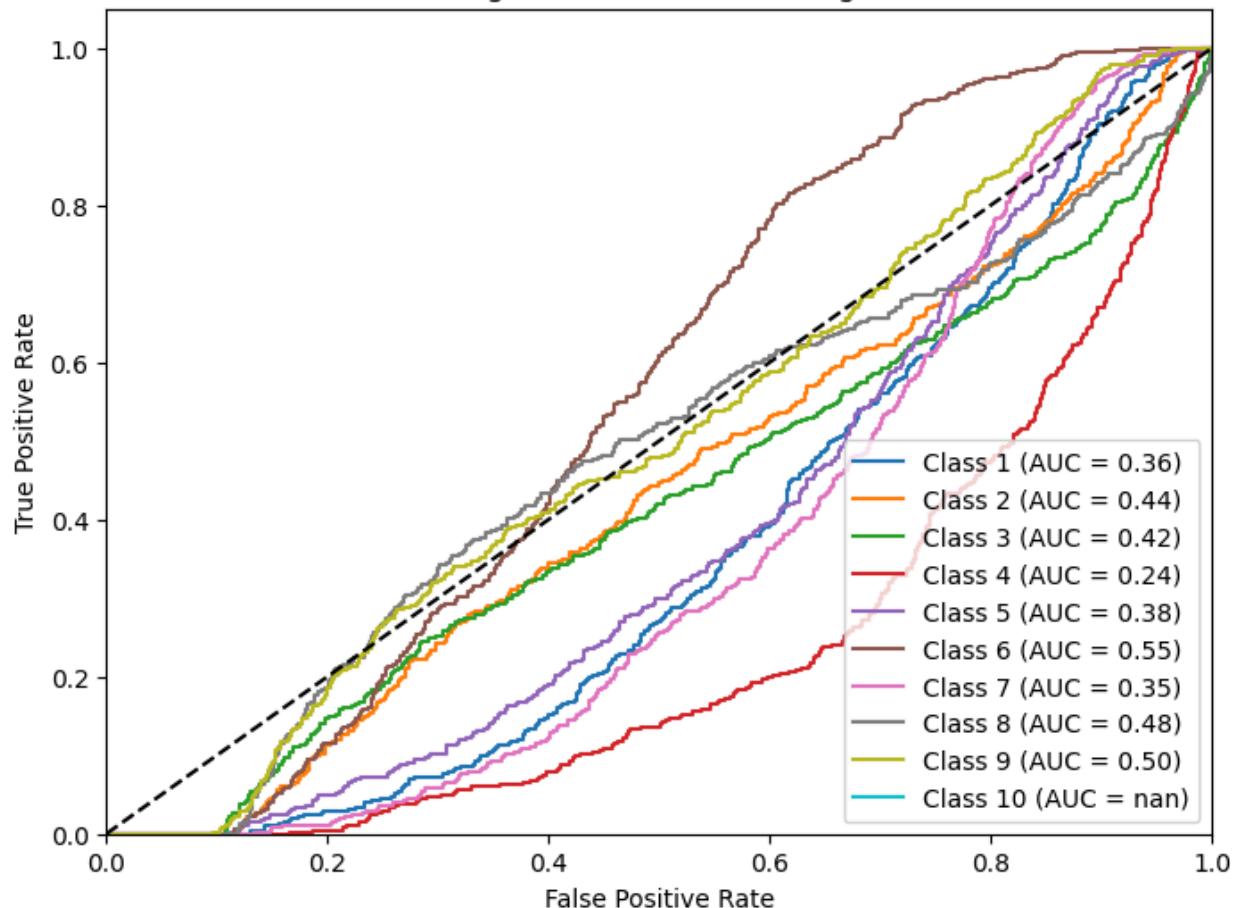
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: gaussian Kernel, Training Size: 60%



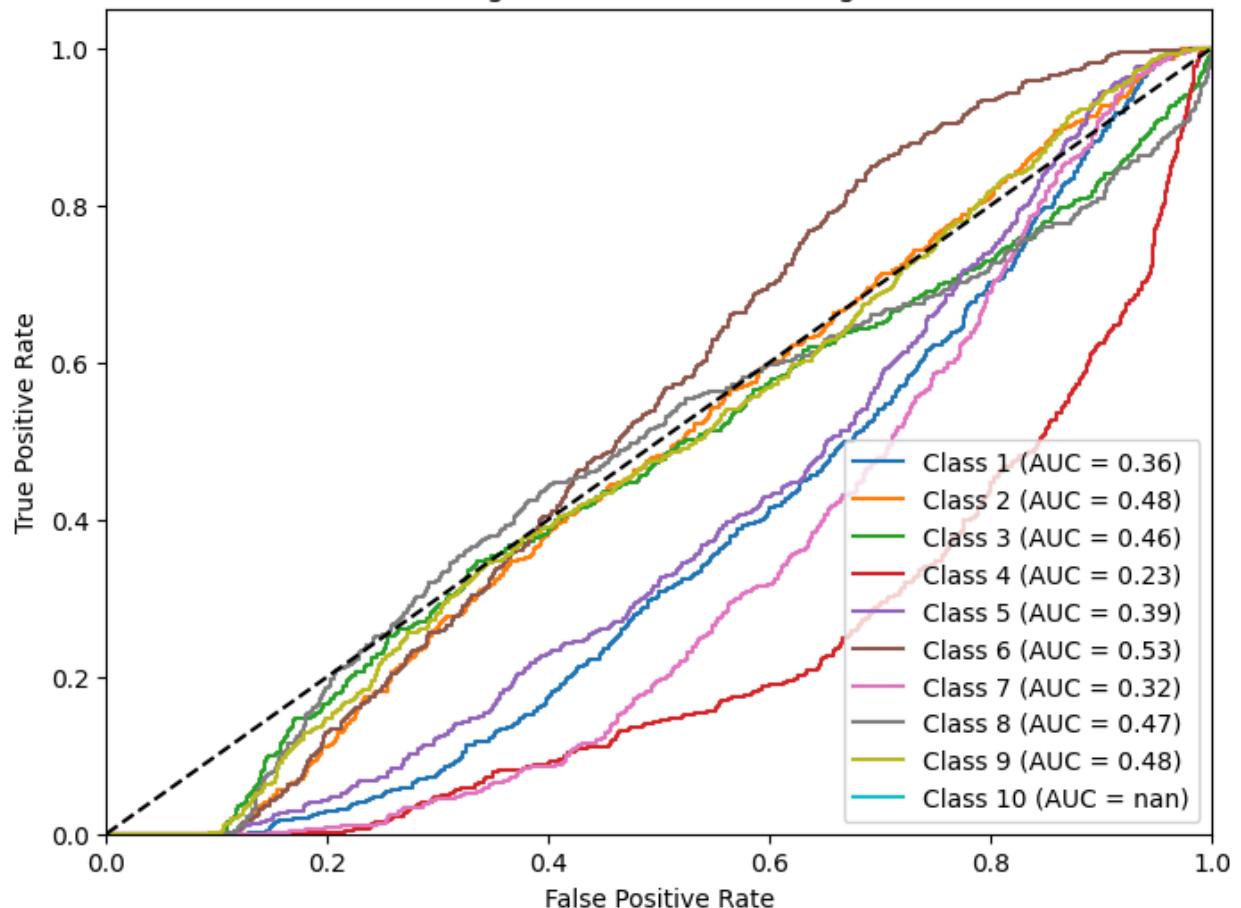
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: gaussian Kernel, Training Size: 70%



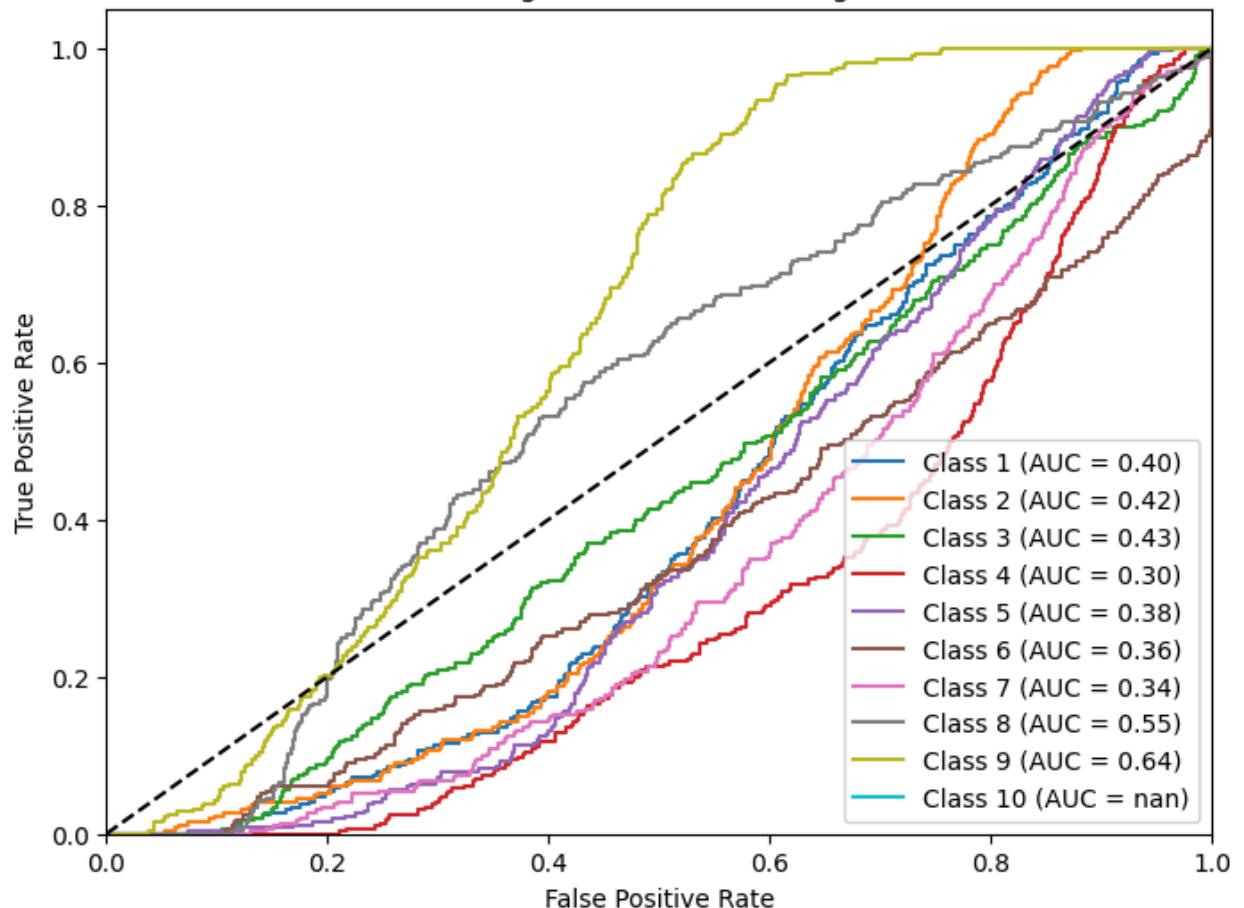
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: gaussian Kernel, Training Size: 80%



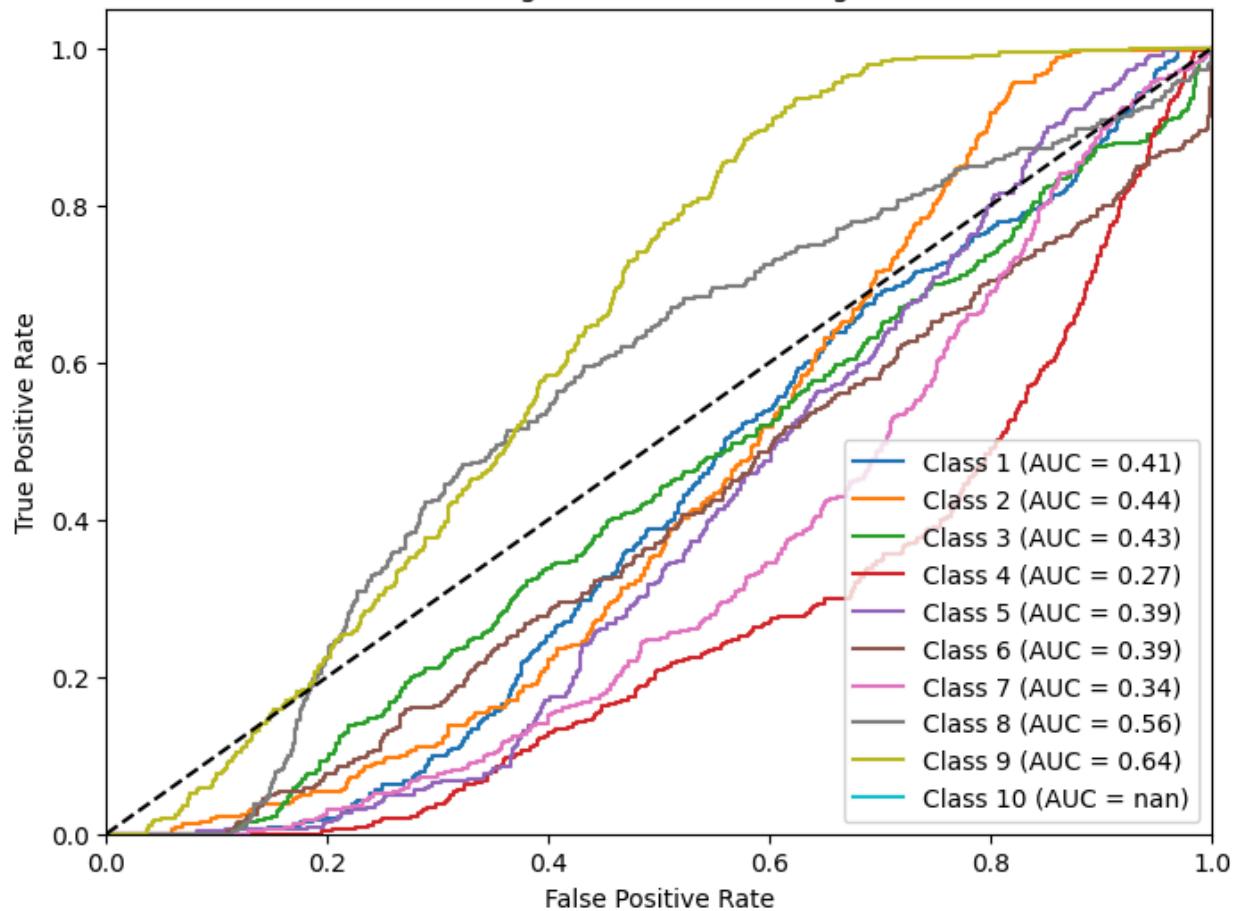
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: sigmoid Kernel, Training Size: 50%



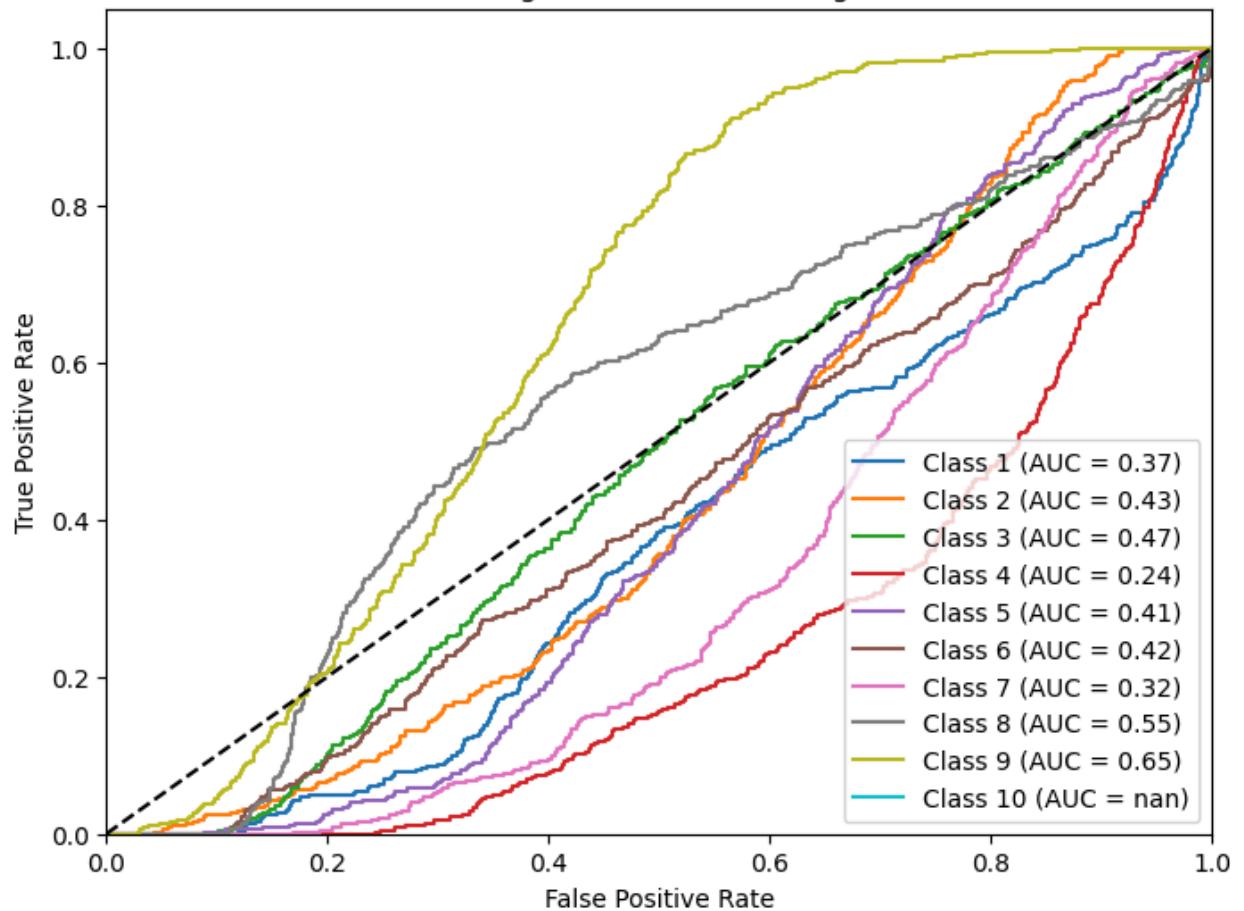
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: sigmoid Kernel, Training Size: 60%

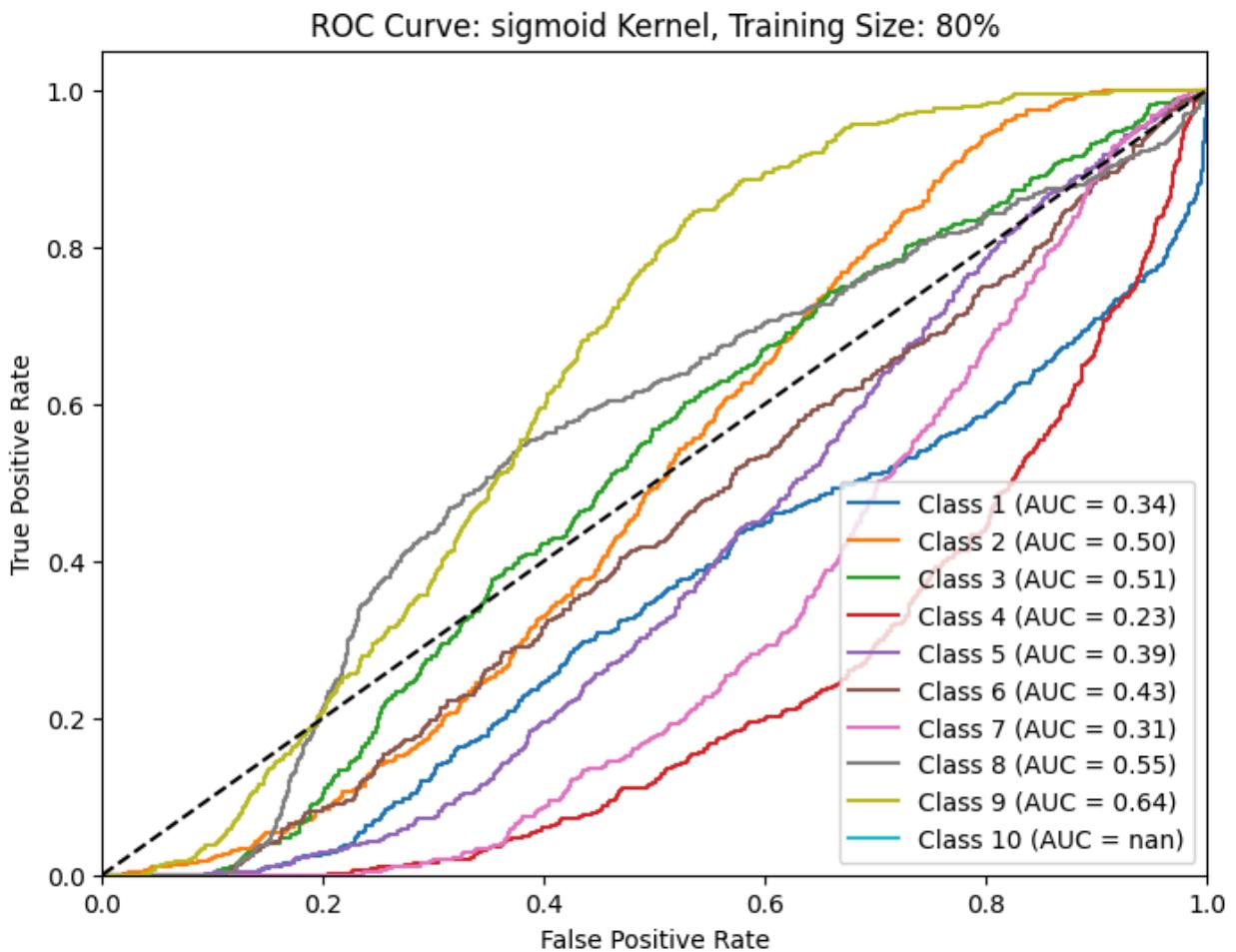


```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve: sigmoid Kernel, Training Size: 70%



```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```



Calculate values for different test size using PCA-reduced data

```
In [ ]: results_pca = []
confusion_matrices_pca = []

for kernel_name, kernel in kernels.items():
    for size in training_sizes:
        X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca_reduced,
            svc_pca = SVC(kernel=kernel, probability=True, random_state=42) # Add probability
            svc_pca.fit(X_train_pca, y_train.values.ravel())
            y_pred_pca = svc_pca.predict(X_test_pca)

            acc_pca = accuracy_score(y_test, y_pred_pca)
            precision_pca = precision_score(y_test, y_pred_pca, average='weighted', zero_division=0)
            recall_pca = recall_score(y_test, y_pred_pca, average='weighted') # Calculate weighted recall
            f1_pca = f1_score(y_test, y_pred_pca, average='weighted') # Calculate weighted F1 score

            cm_pca = confusion_matrix(y_test, y_pred_pca) # Calculate confusion matrix

            results_pca.append({
                "Training size":int (size*100),
                "Kernel":kernel_name,
                "Accuracy":acc_pca,
                "Precision":precision_pca,
                "Recall":recall_pca,
                "F1 Score":f1_pca,
                "Confusion Matrix":cm_pca
            })
        
```

```

        "Accuracy":acc_pca,
        "Precision": precision_pca,
        "Recall": recall_pca,
        "F1-score": f1_pca,
        "Kernel":kernel_name
    })
confusion_matrices_pca.append({ # Store confusion matrix with metadata
    "Training size":int (size*100),
    "Kernel":kernel_name,
    "Confusion Matrix":cm_pca
})

```

## Print Accuracy, Precision, Recall, and F1-score Table and Graphs for PCA-Reduced Data

```

In [ ]: df_pca = pd.DataFrame(results_pca)
display(df_pca) # Display the DataFrame as a table

# Plot Accuracy
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Accuracy', hue='Kernel')
plt.title('SVM Accuracy by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

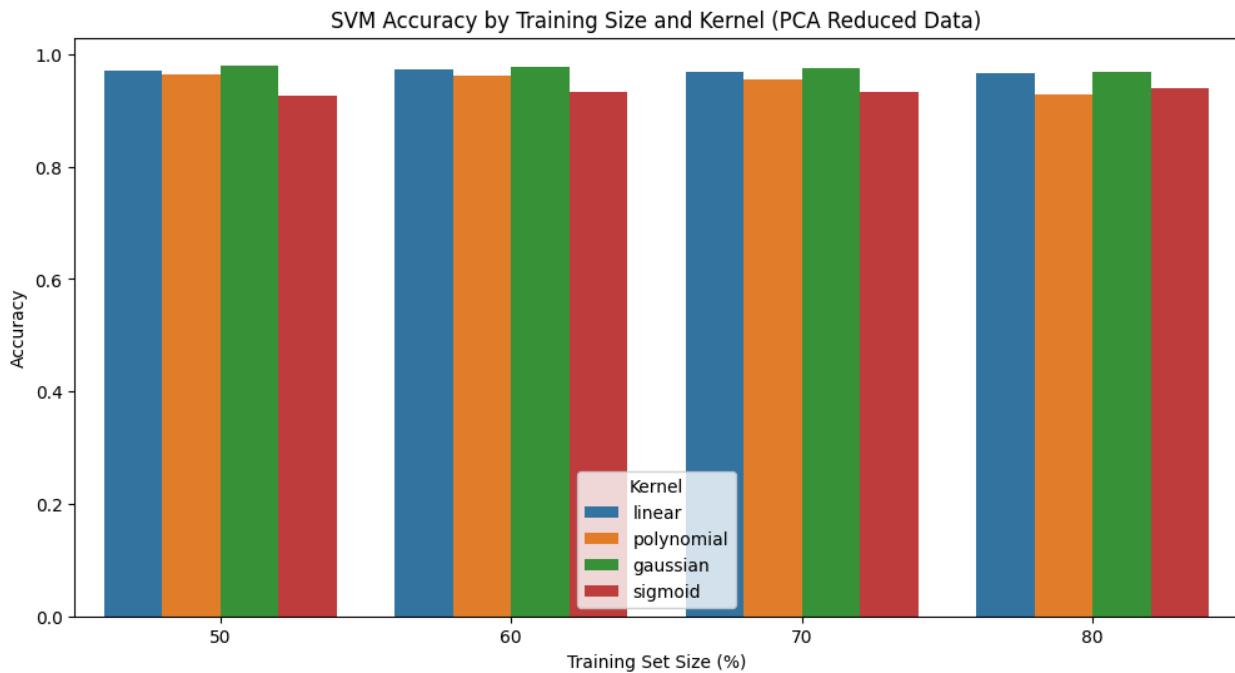
# Plot Precision
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Precision', hue='Kernel')
plt.title('SVM Precision by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

# Plot Recall
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='Recall', hue='Kernel')
plt.title('SVM Recall by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

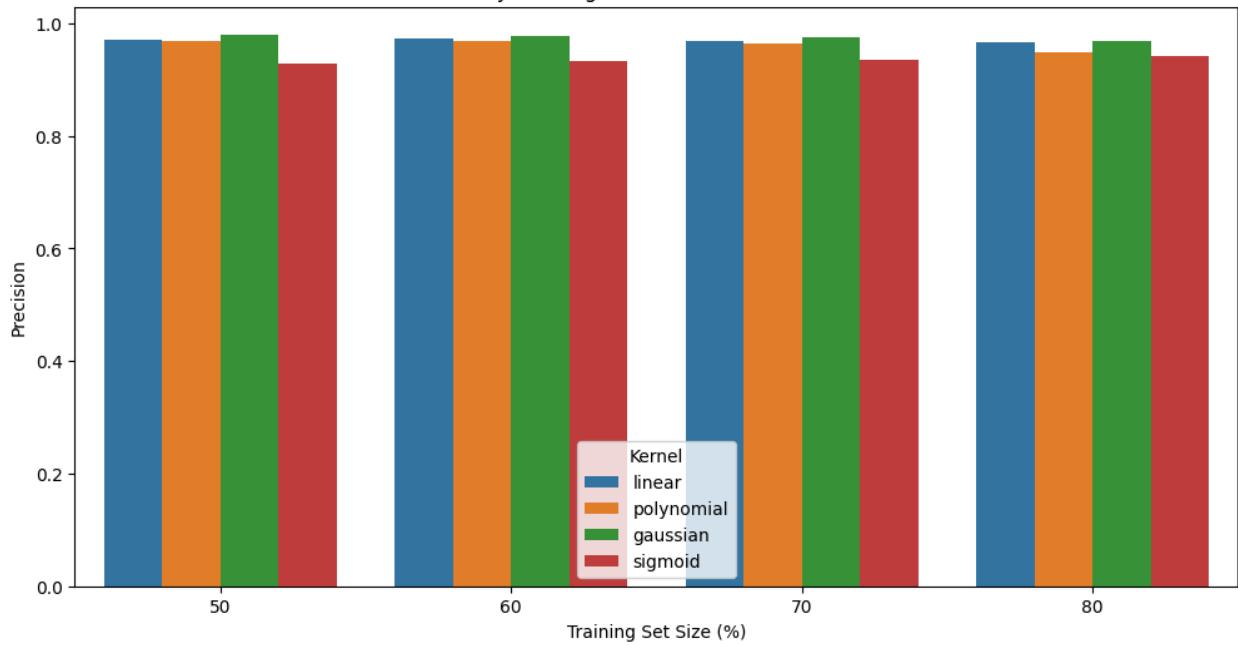
# Plot F1-score
plt.figure(figsize=(12, 6)) # Increase figure size
sns.barplot(data=df_pca, x='Training size', y='F1-score', hue='Kernel')
plt.title('SVM F1-score by Training Size and Kernel (PCA Reduced Data)') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()

```

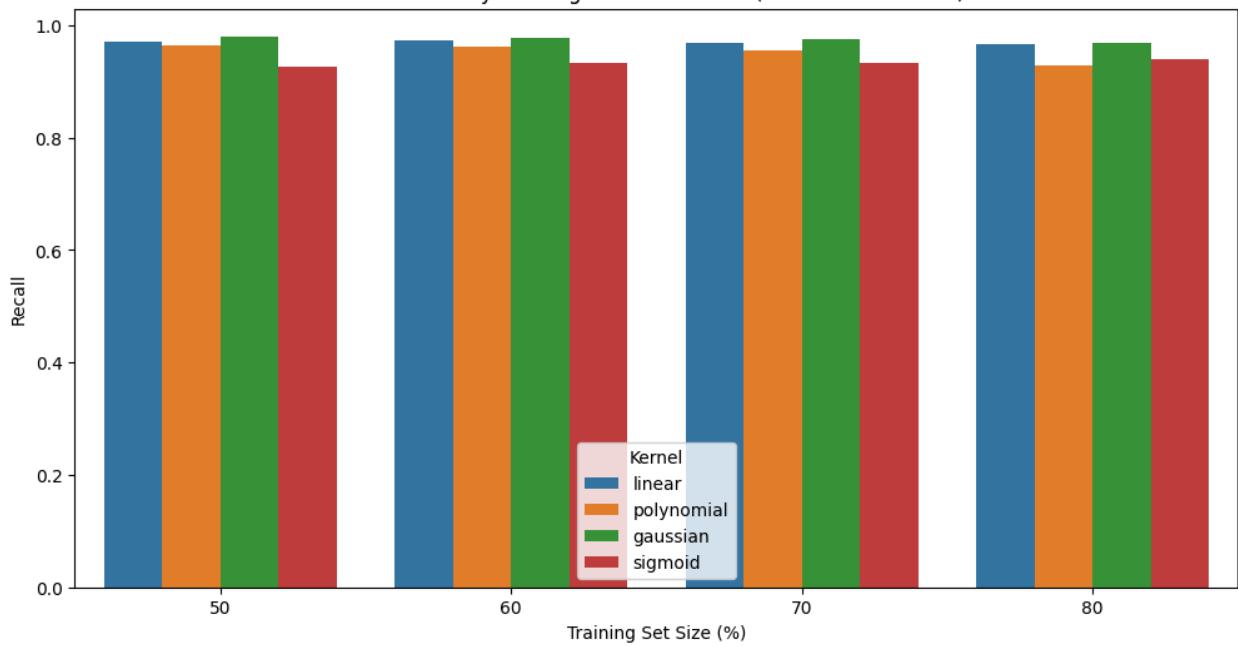
<b>Training size</b>		<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Kernel</b>
<b>0</b>	50	0.970819	0.970937	0.970819	0.970850	linear
<b>1</b>	60	0.972716	0.972695	0.972716	0.972687	linear
<b>2</b>	70	0.968734	0.968882	0.968734	0.968751	linear
<b>3</b>	80	0.966192	0.966314	0.966192	0.966197	linear
<b>4</b>	50	0.964413	0.968400	0.964413	0.965236	polynomial
<b>5</b>	60	0.962930	0.967787	0.962930	0.963943	polynomial
<b>6</b>	70	0.956279	0.963473	0.956279	0.957801	polynomial
<b>7</b>	80	0.927491	0.948847	0.927491	0.932636	polynomial
<b>8</b>	50	0.980427	0.980517	0.980427	0.980437	gaussian
<b>9</b>	60	0.977165	0.977299	0.977165	0.977172	gaussian
<b>10</b>	70	0.975089	0.975237	0.975089	0.975076	gaussian
<b>11</b>	80	0.968416	0.968740	0.968416	0.968425	gaussian
<b>12</b>	50	0.927402	0.927971	0.927402	0.927522	sigmoid
<b>13</b>	60	0.933571	0.933986	0.933571	0.933645	sigmoid
<b>14</b>	70	0.933910	0.934523	0.933910	0.934033	sigmoid
<b>15</b>	80	0.940169	0.940999	0.940169	0.940326	sigmoid

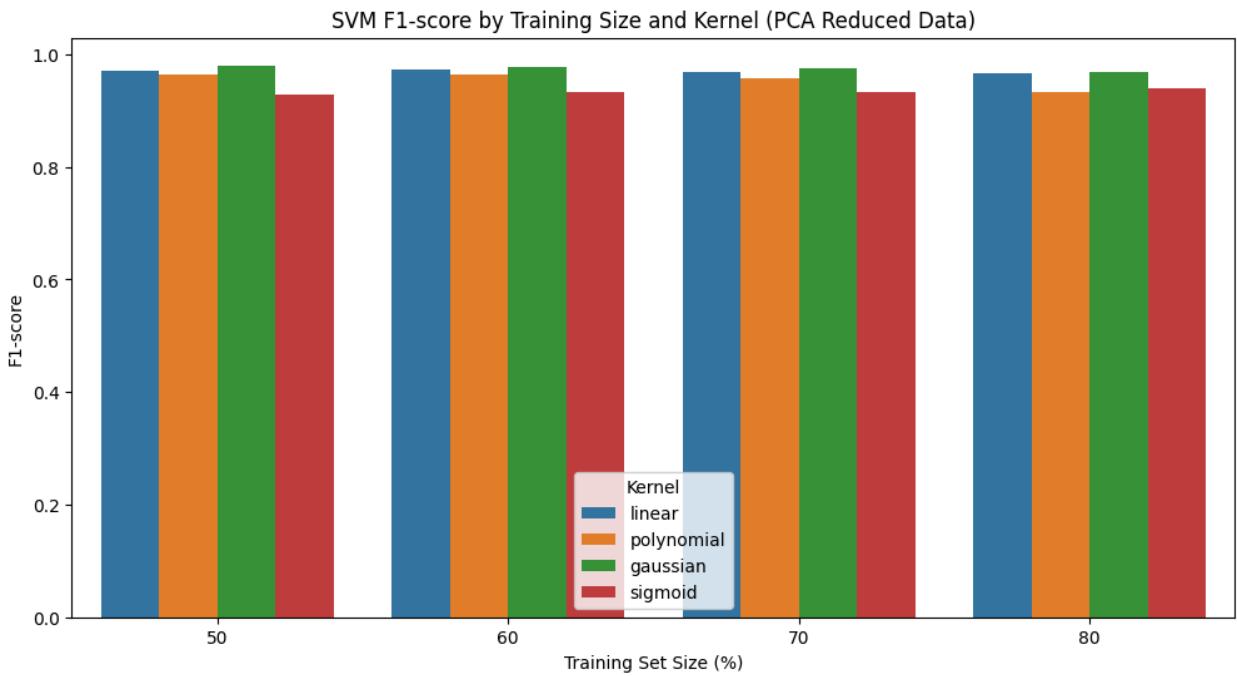


SVM Precision by Training Size and Kernel (PCA Reduced Data)



SVM Recall by Training Size and Kernel (PCA Reduced Data)

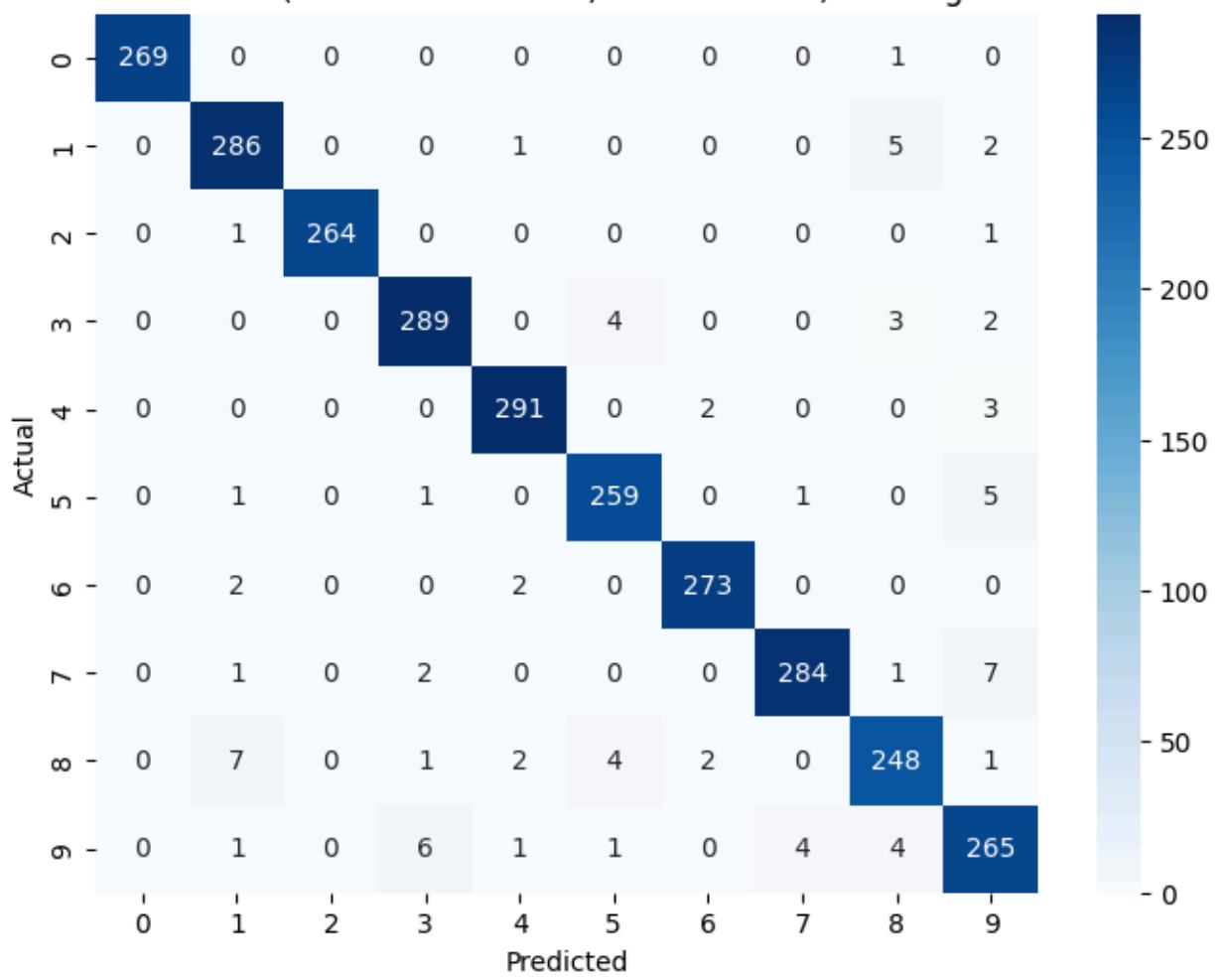




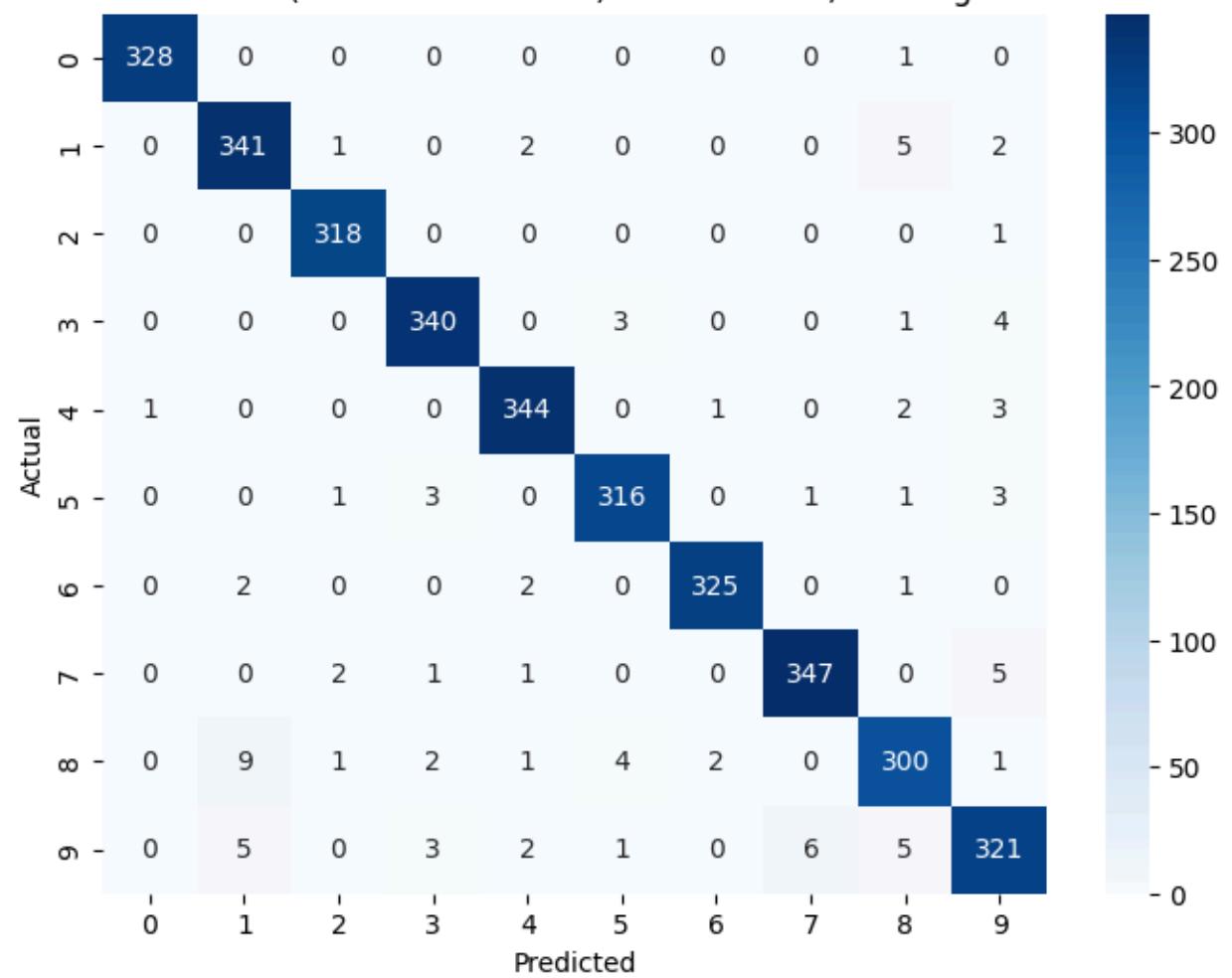
## Show Confusion Matrices for PCA-Reduced Data

```
In [ ]: for cm_data in confusion_matrices_pca:  
    cm = cm_data["Confusion Matrix"]  
    kernel_name = cm_data["Kernel"]  
    training_size = cm_data["Training size"]  
  
    plt.figure(figsize=(8, 6))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  
    plt.title(f'Confusion Matrix (PCA Reduced Data): {kernel_name} Kernel, Tra  
    plt.xlabel('Predicted')  
    plt.ylabel('Actual')  
    plt.show()
```

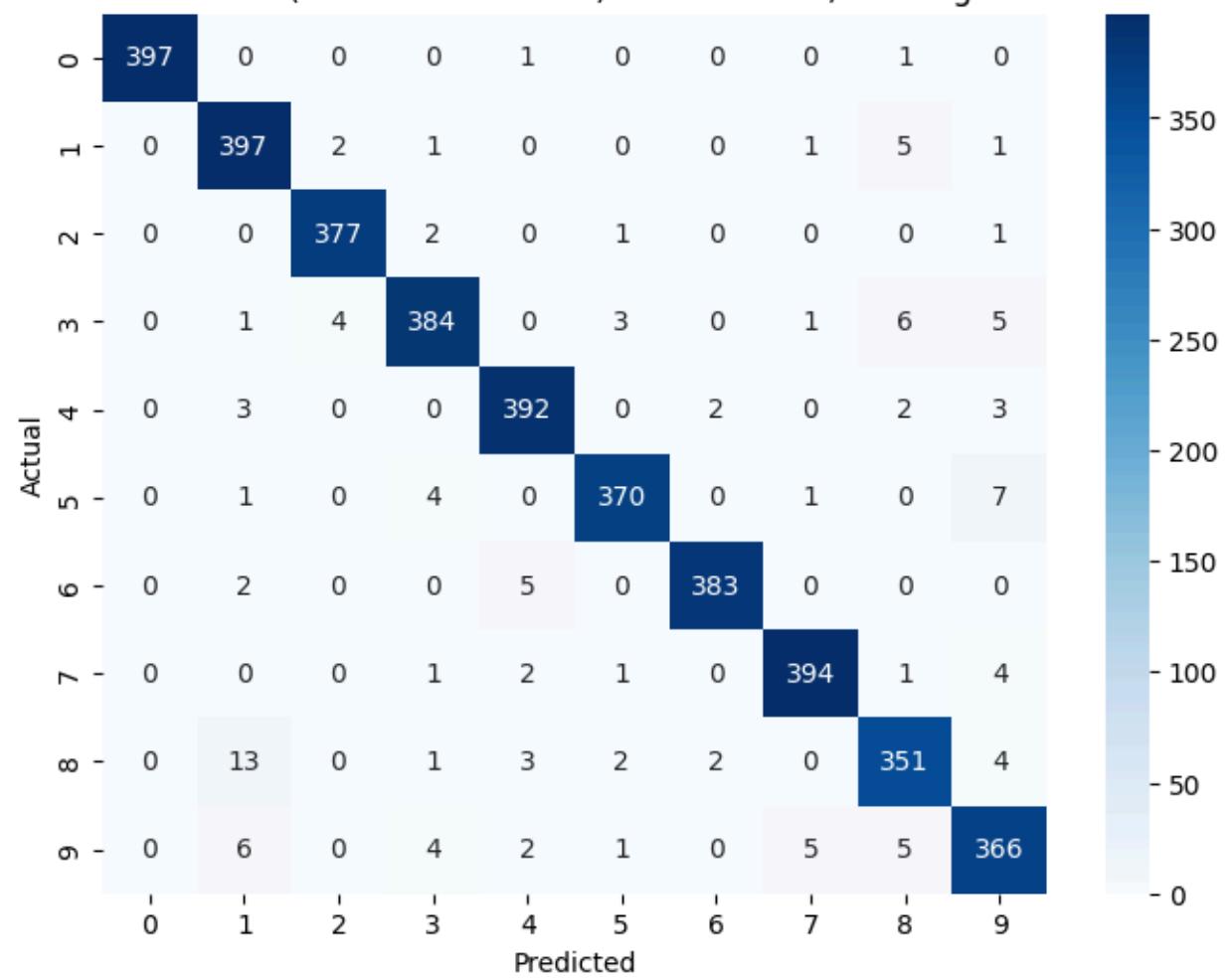
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 50%



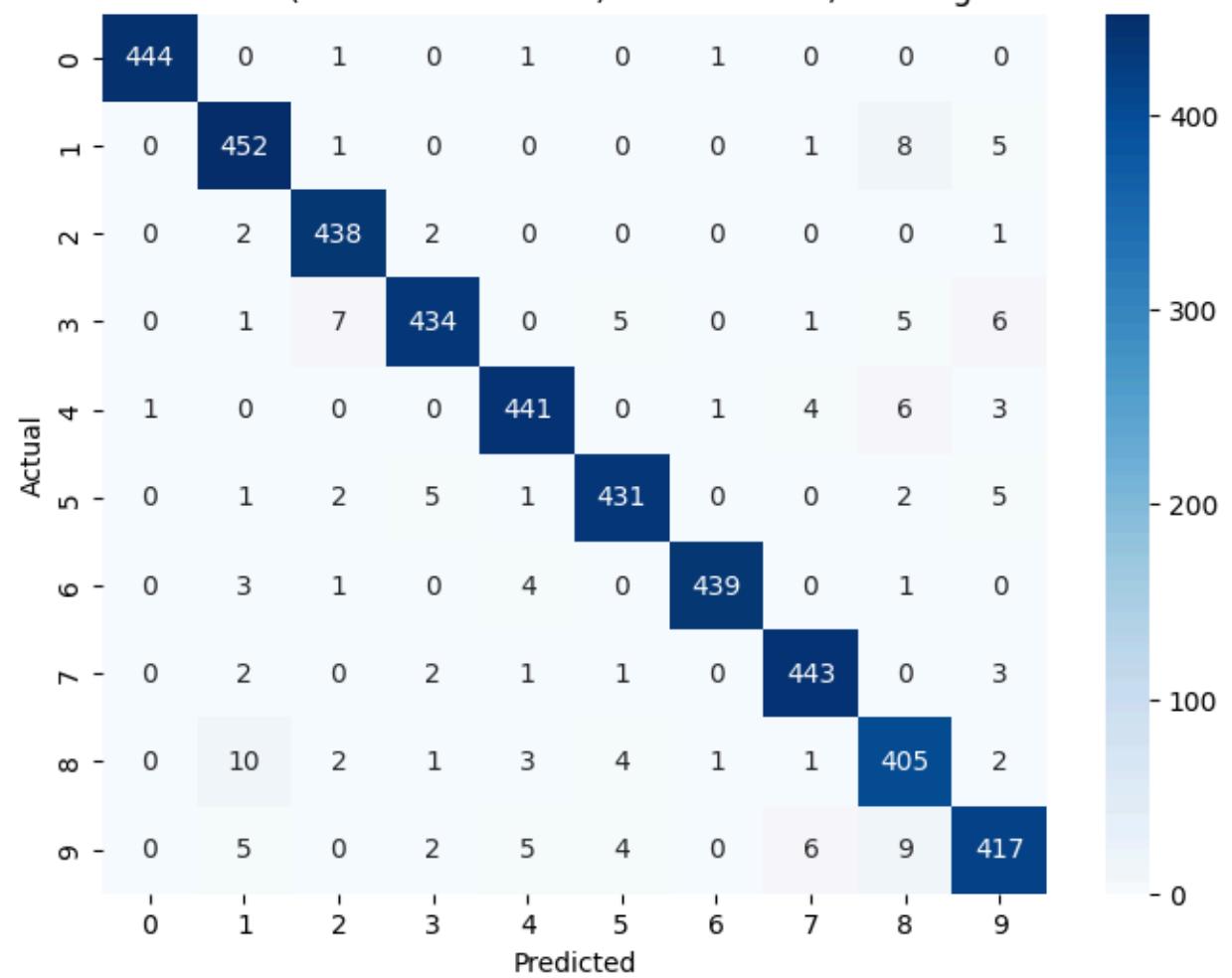
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 60%



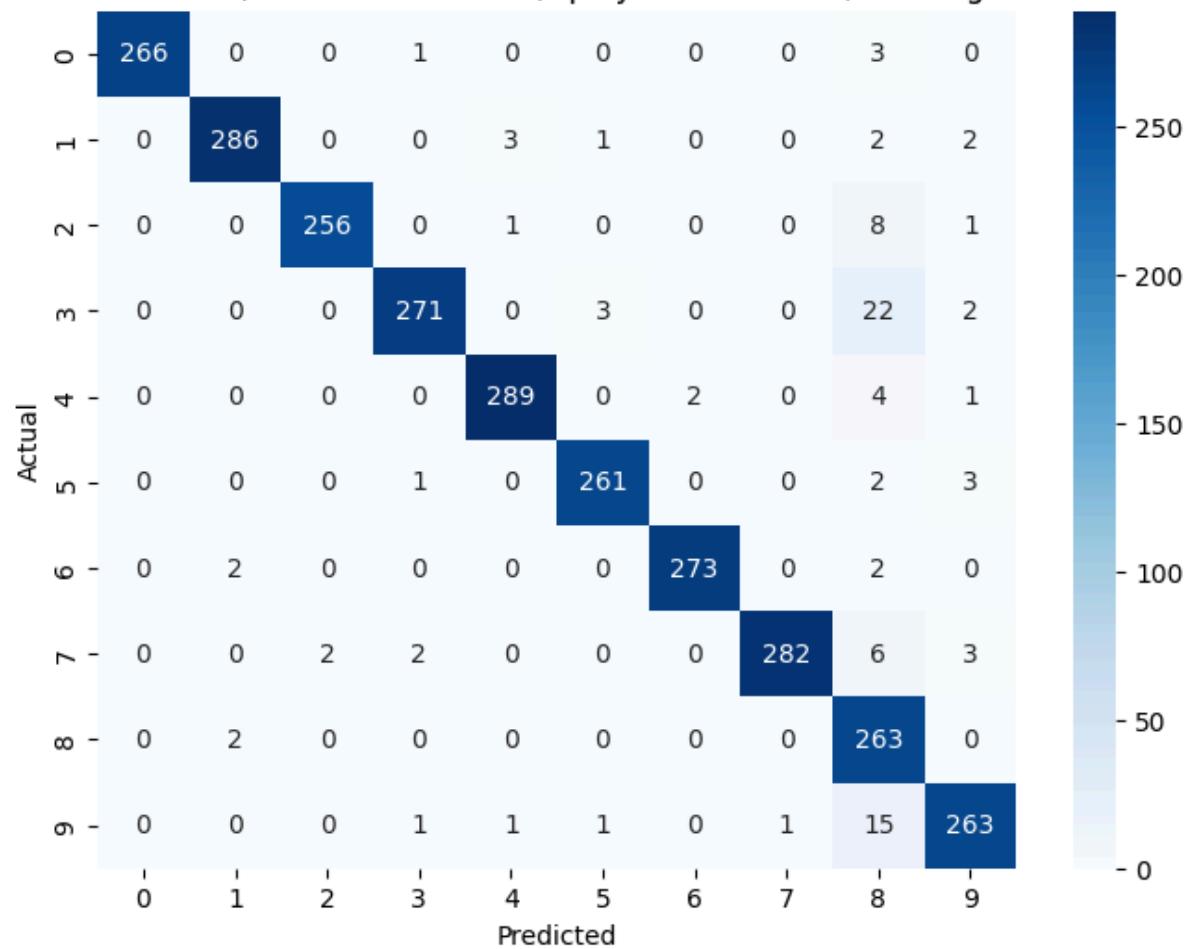
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 70%



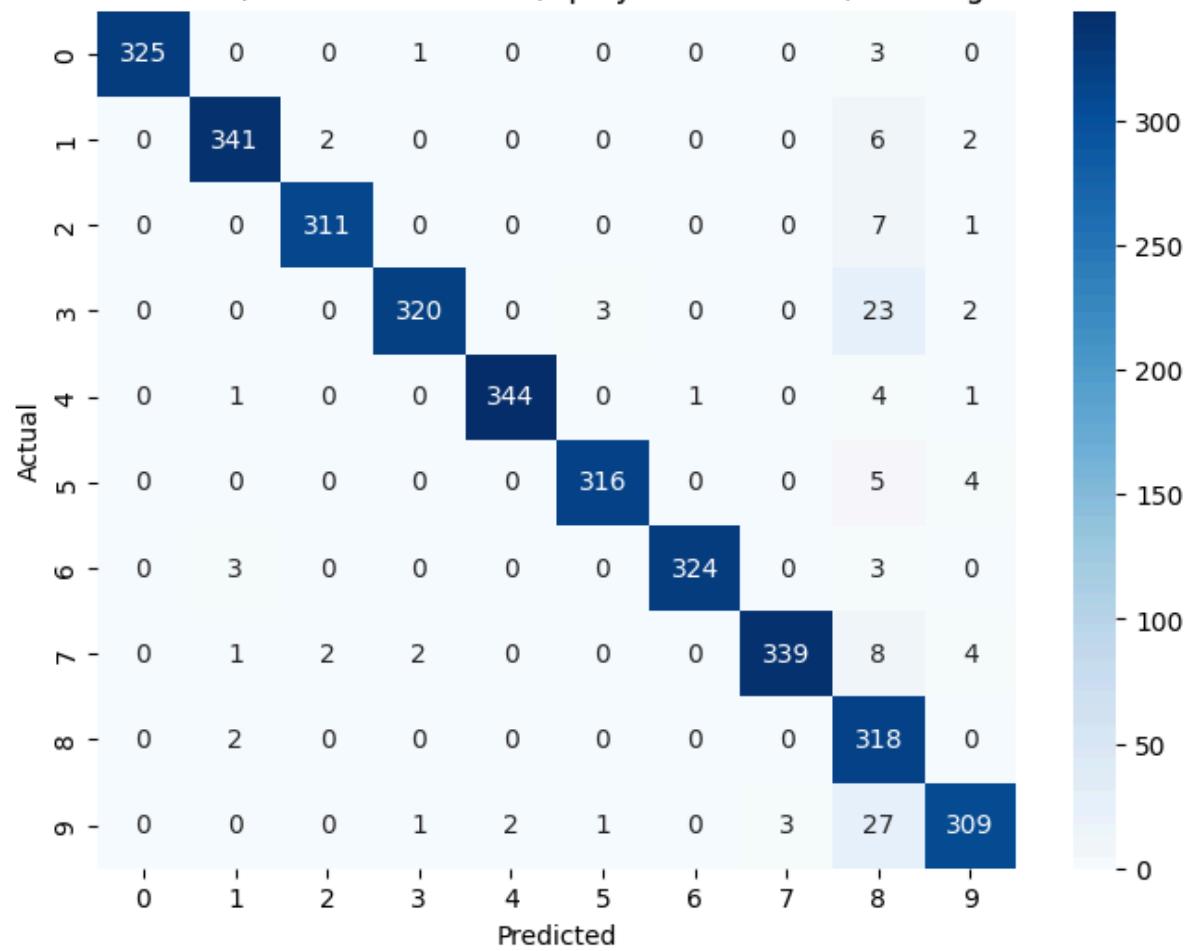
Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 80%



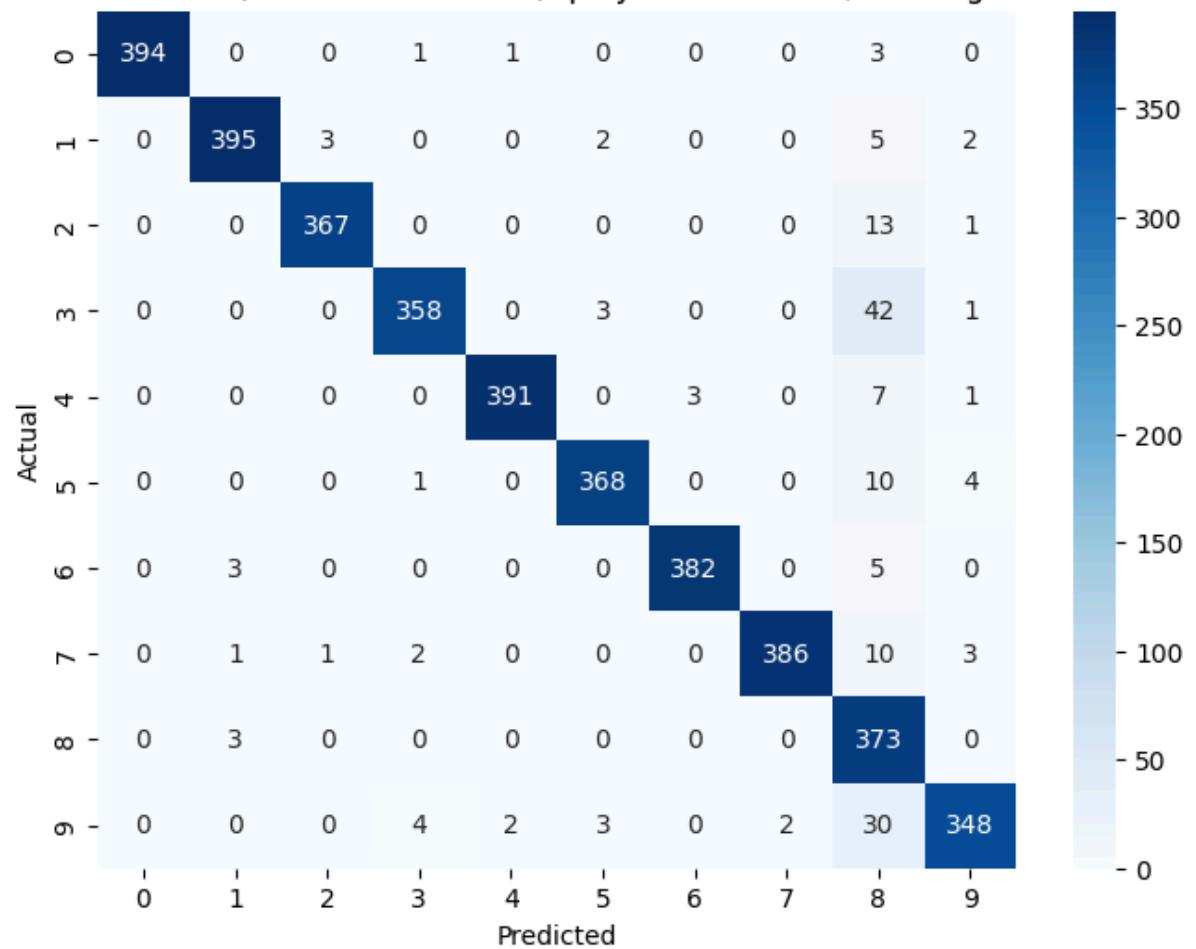
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 50%



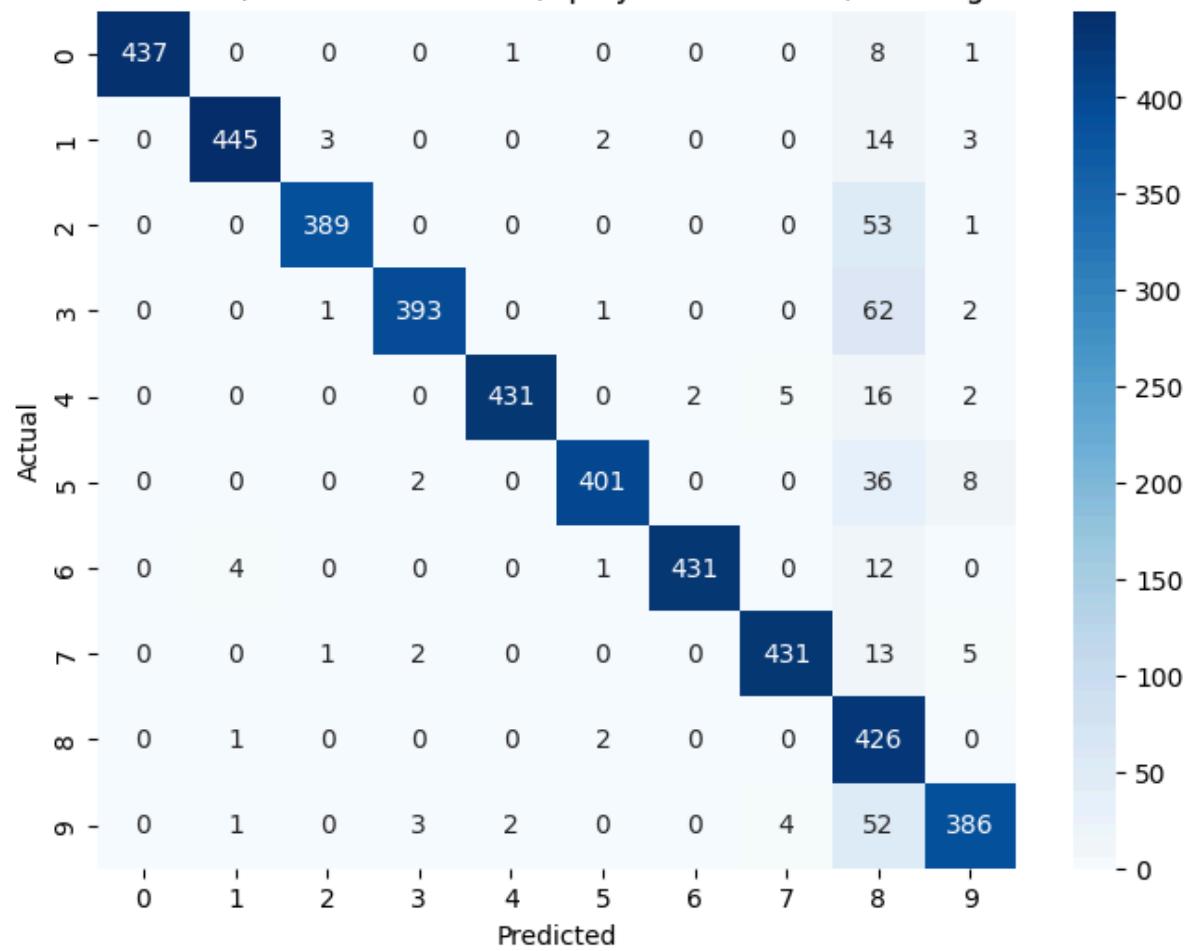
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 60%



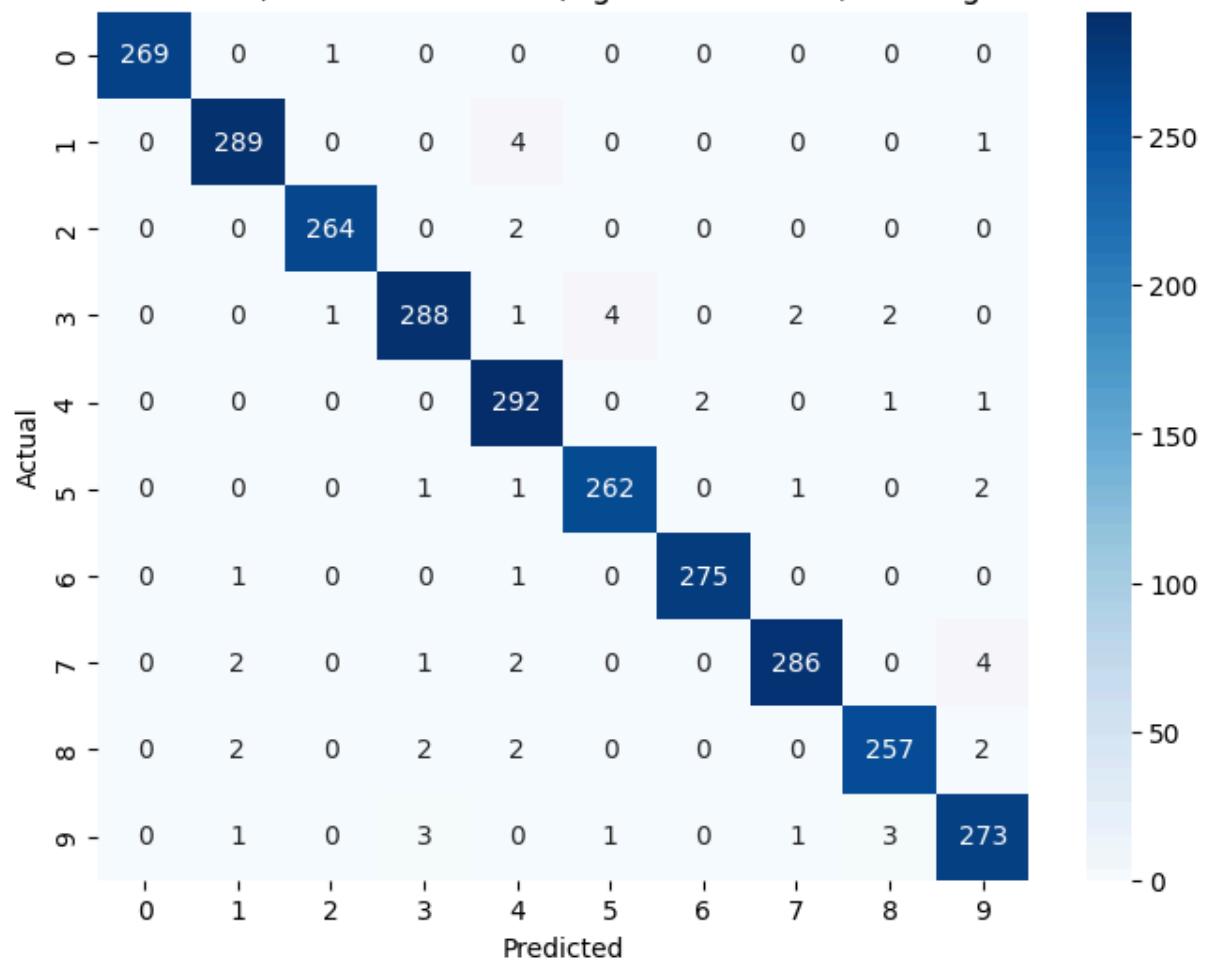
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 70%



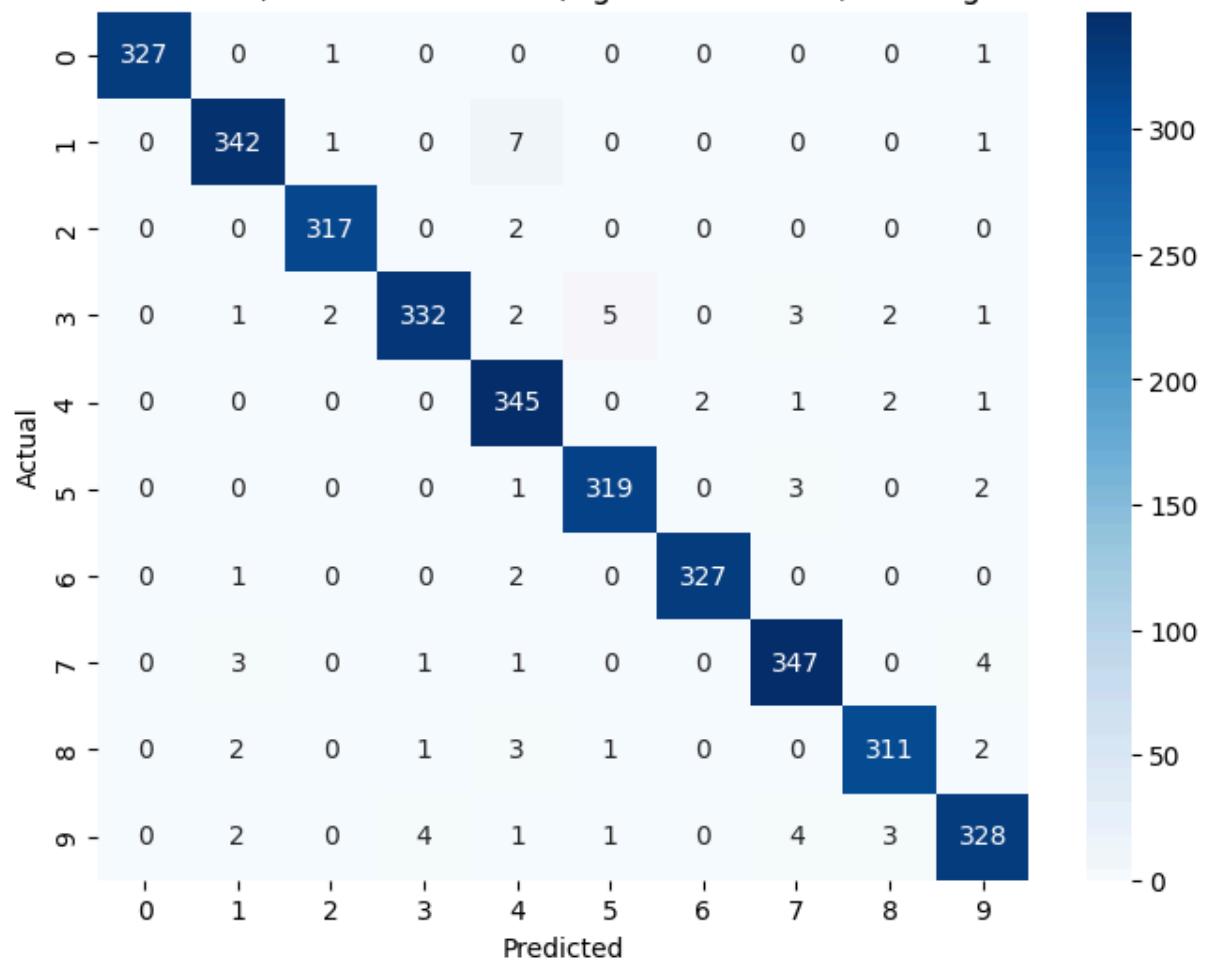
Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 80%



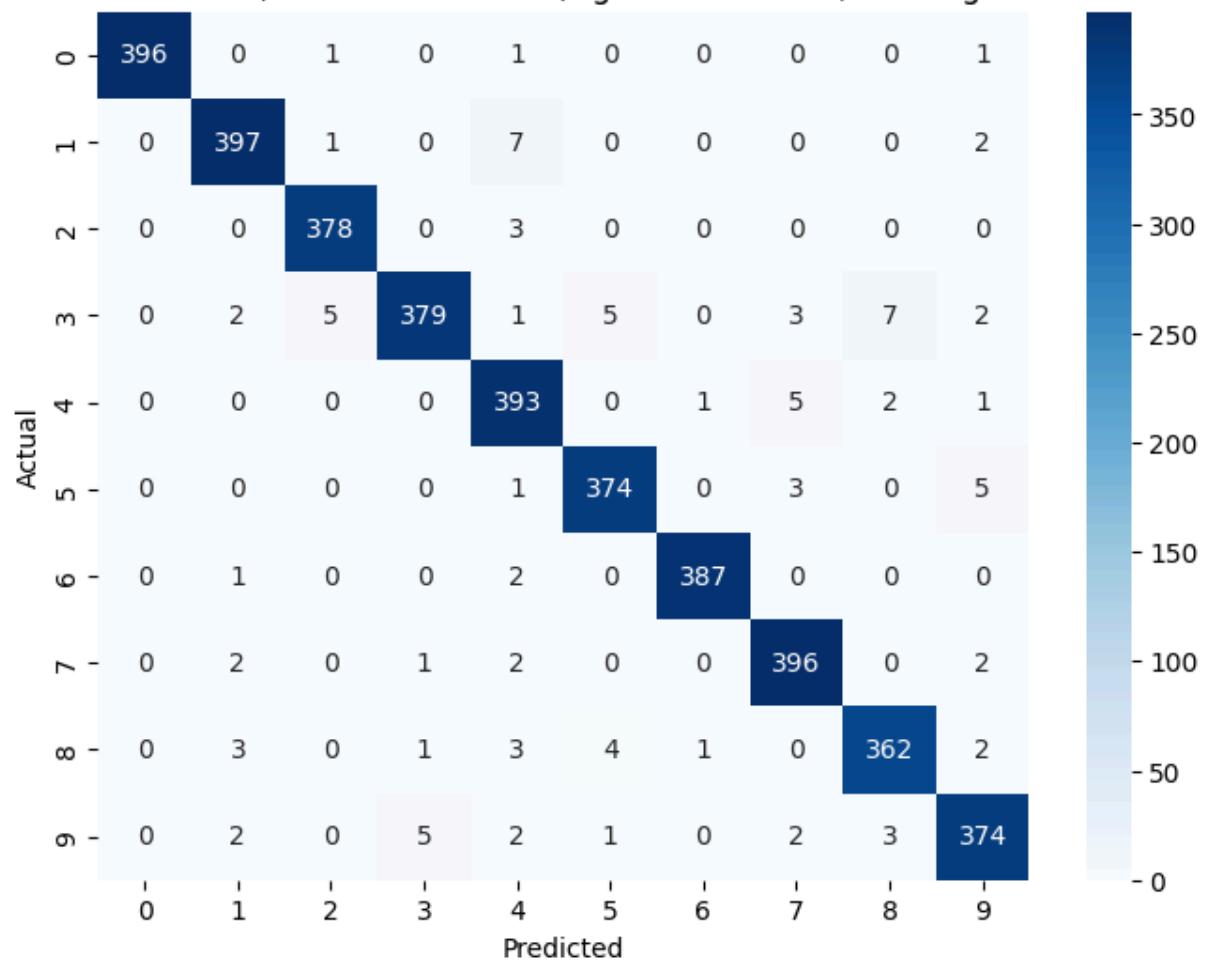
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 50%



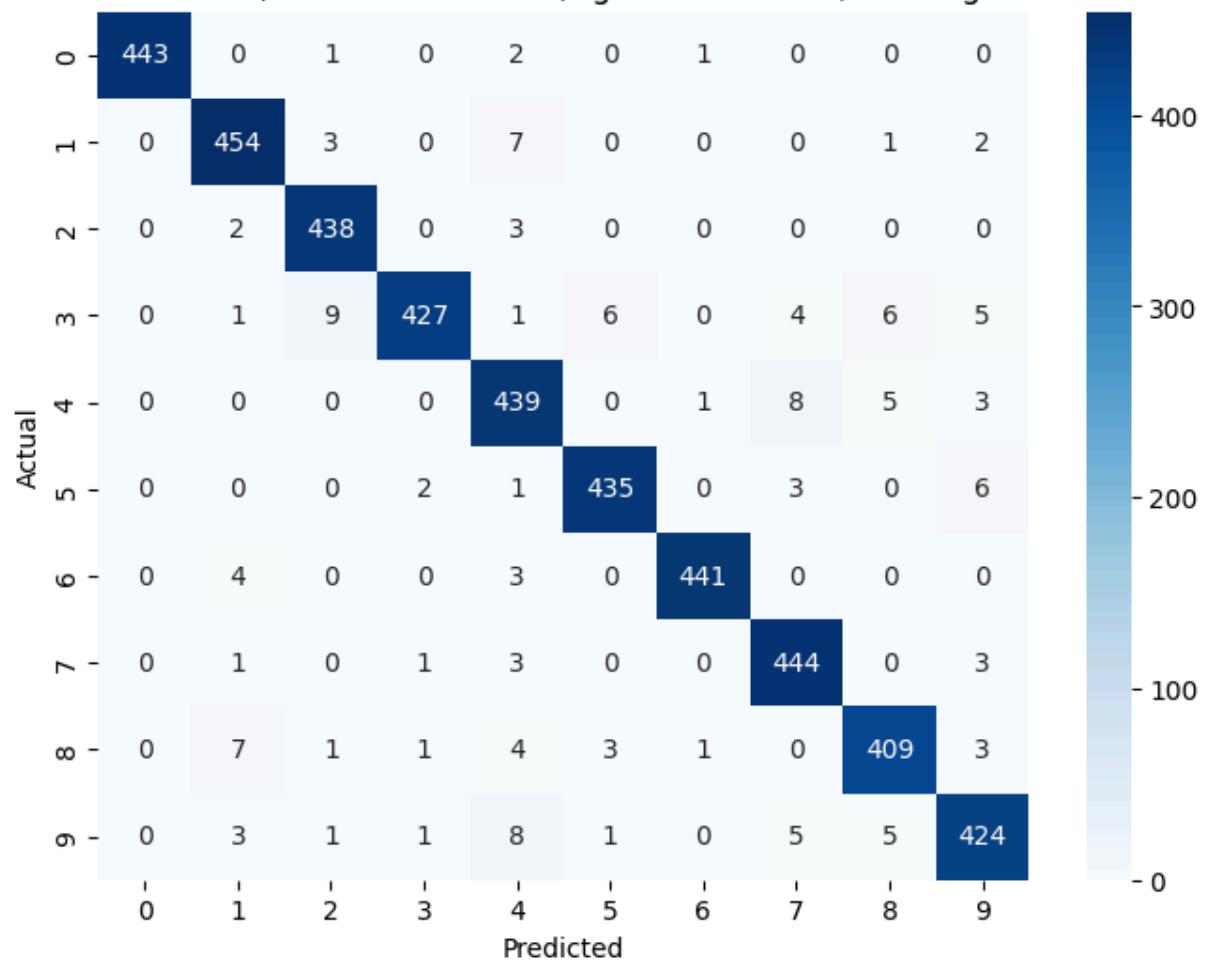
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 60%



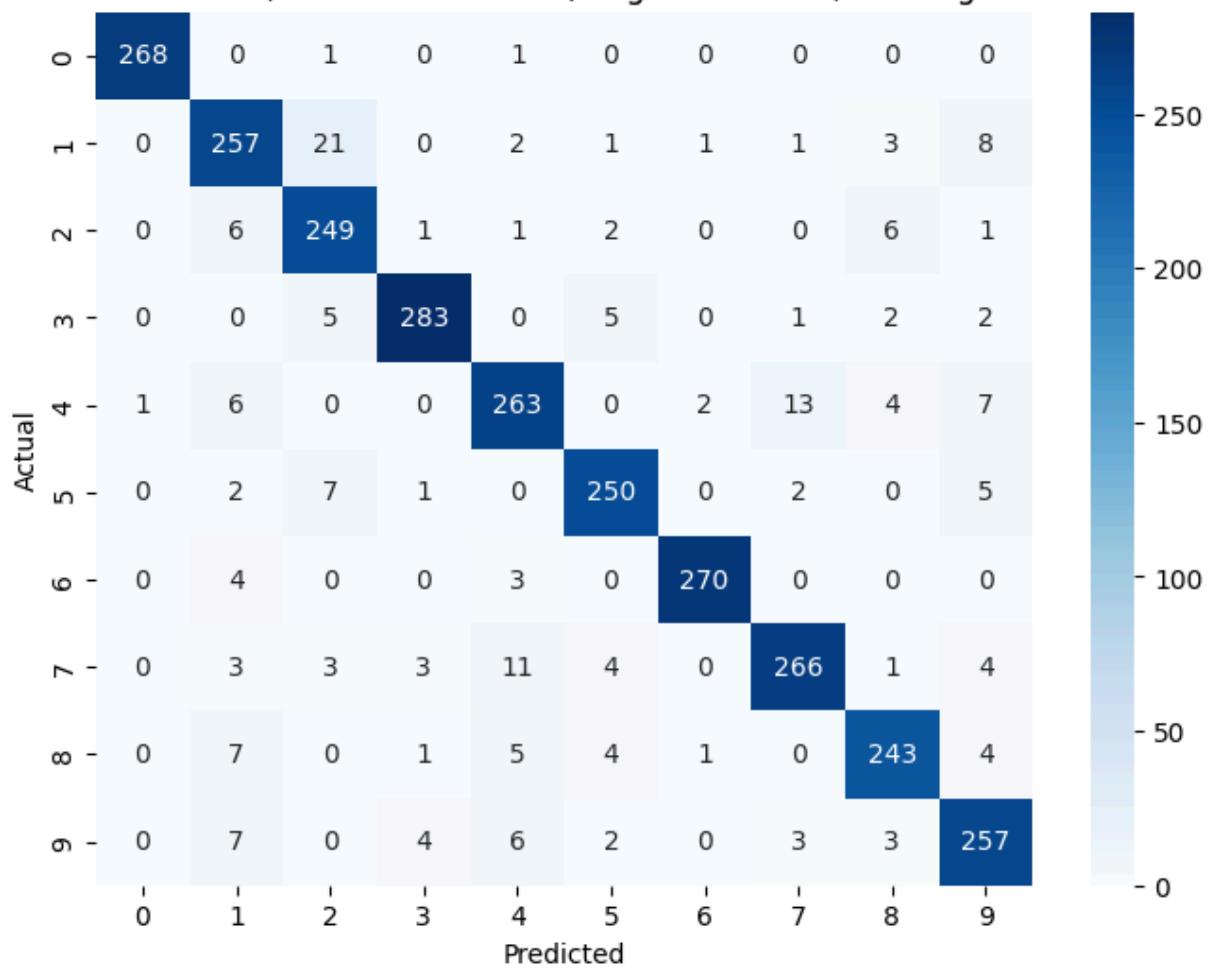
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 70%



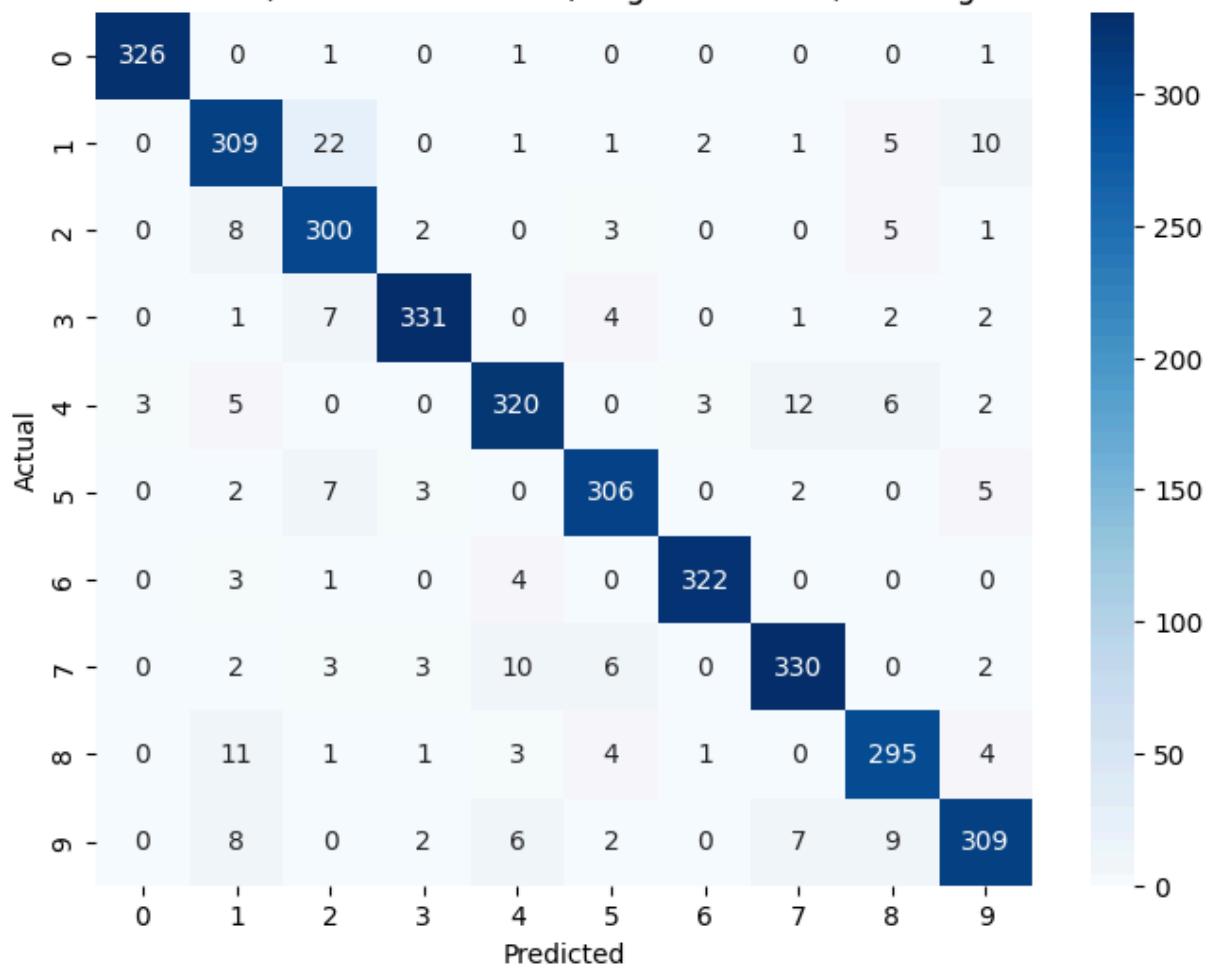
Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 80%



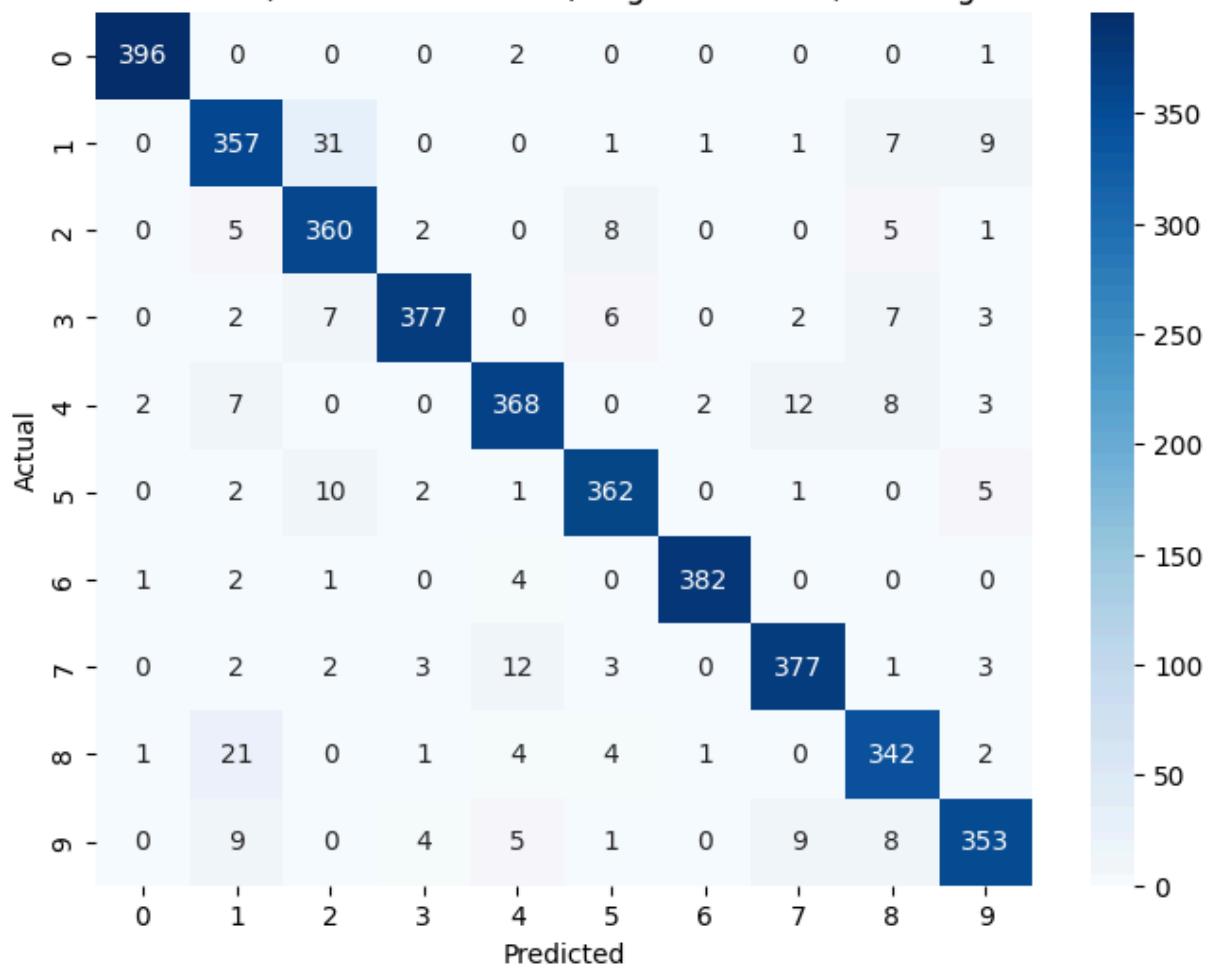
Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 50%



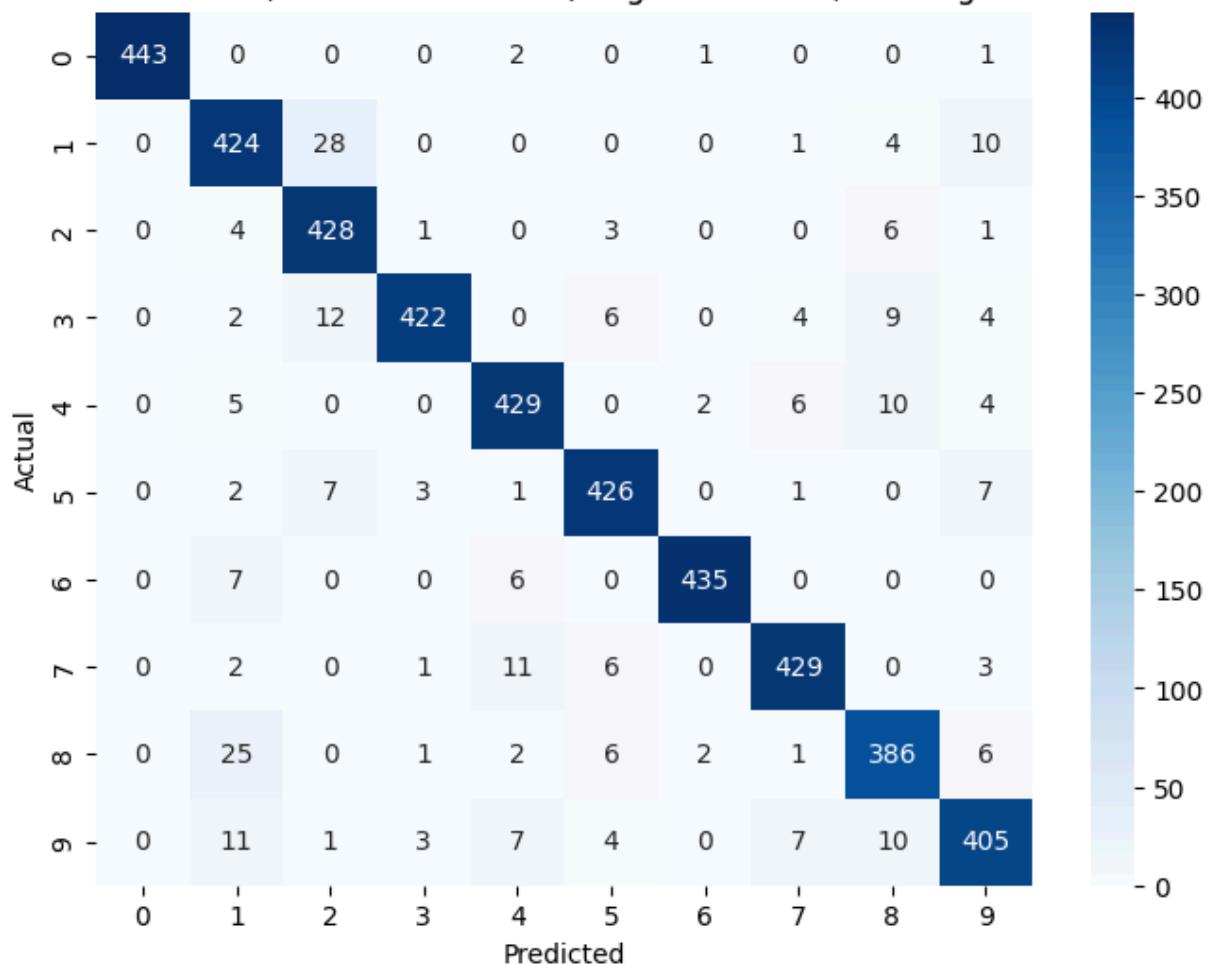
Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 60%



Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 70%



Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 80%



## MLP

Calculate values for different test size

```
In [ ]: from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]

results_mlp = []
confusion_matrices_mlp = []
trained_models_mlp = []

for size in training_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size,
# Initialize MLPClassifier
```

```

# You might need to tune the hyperparameters like hidden_layer_sizes, acti
mlp = MLPClassifier(max_iter=1000)

mlp.fit(X_train, y_train.values.ravel())
y_pred = mlp.predict(X_test)

acc = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_divis
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

cm = confusion_matrix(y_test, y_pred)

results_mlp.append({
    "Training size": int(size * 100),
    "Accuracy": acc,
    "Precision": precision,
    "Recall": recall,
    "F1-score": f1
})
confusion_matrices_mlp.append({
    "Training size": int(size * 100),
    "Confusion Matrix": cm
})
trained_models_mlp.append({ # Store the trained model with metadata
    "Training size": int(size*100),
    "Model": mlp,
    "X_test": X_test,
    "y_test": y_test
})

```

## Print Evaluation Table and Graphs

```

In [ ]: df = pd.DataFrame(results_mlp)
display(df) # Display the DataFrame as a table

# Plot Accuracy
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy')
plt.title('SVM Accuracy by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

# Plot Precision
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision')
plt.title('SVM Precision by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

```

```

# Plot Recall
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall')
plt.title('SVM Recall by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

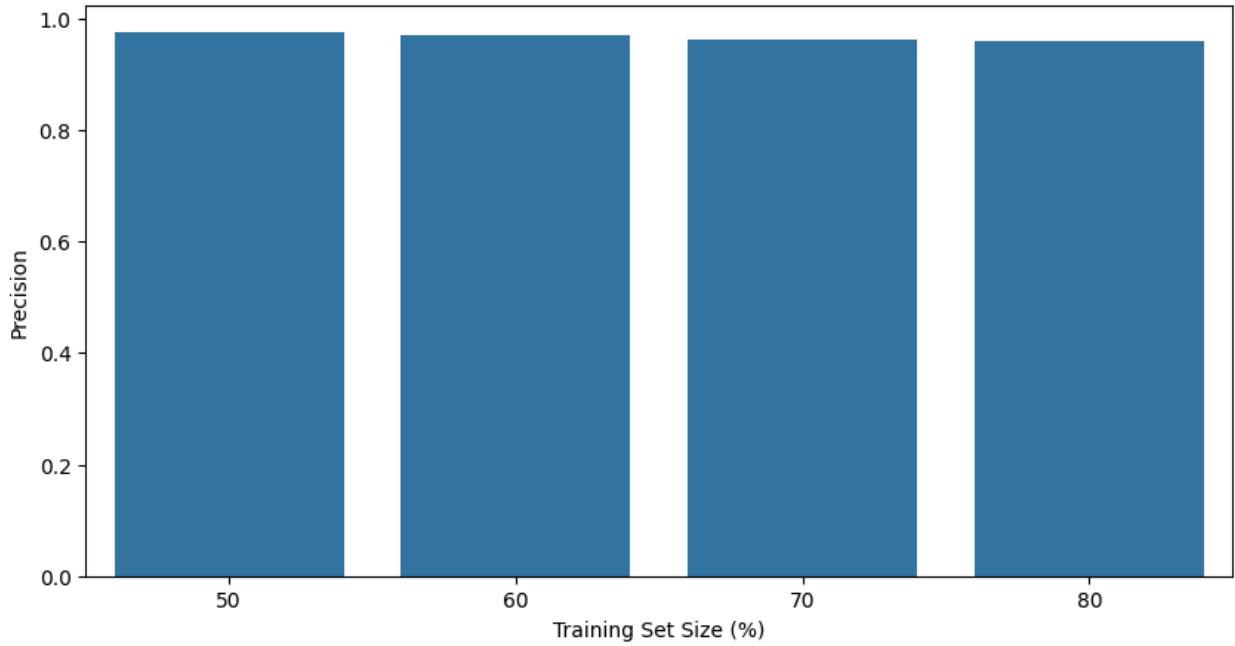
# Plot F1-score
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score')
plt.title('SVM F1-score by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()

```

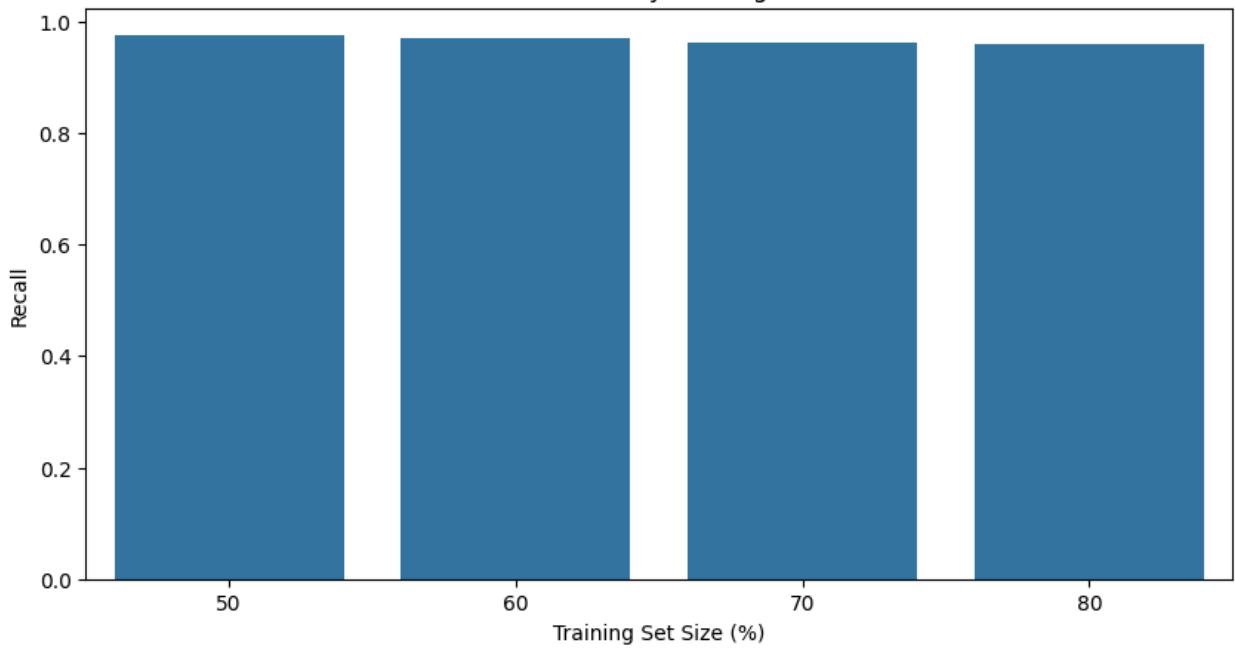
	<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.975089	0.975259	0.975089	0.975121
<b>1</b>	60	0.969158	0.969391	0.969158	0.969207
<b>2</b>	70	0.961617	0.961873	0.961617	0.961637
<b>3</b>	80	0.958852	0.959329	0.958852	0.958933



SVM Precision by Training Size



SVM Recall by Training Size





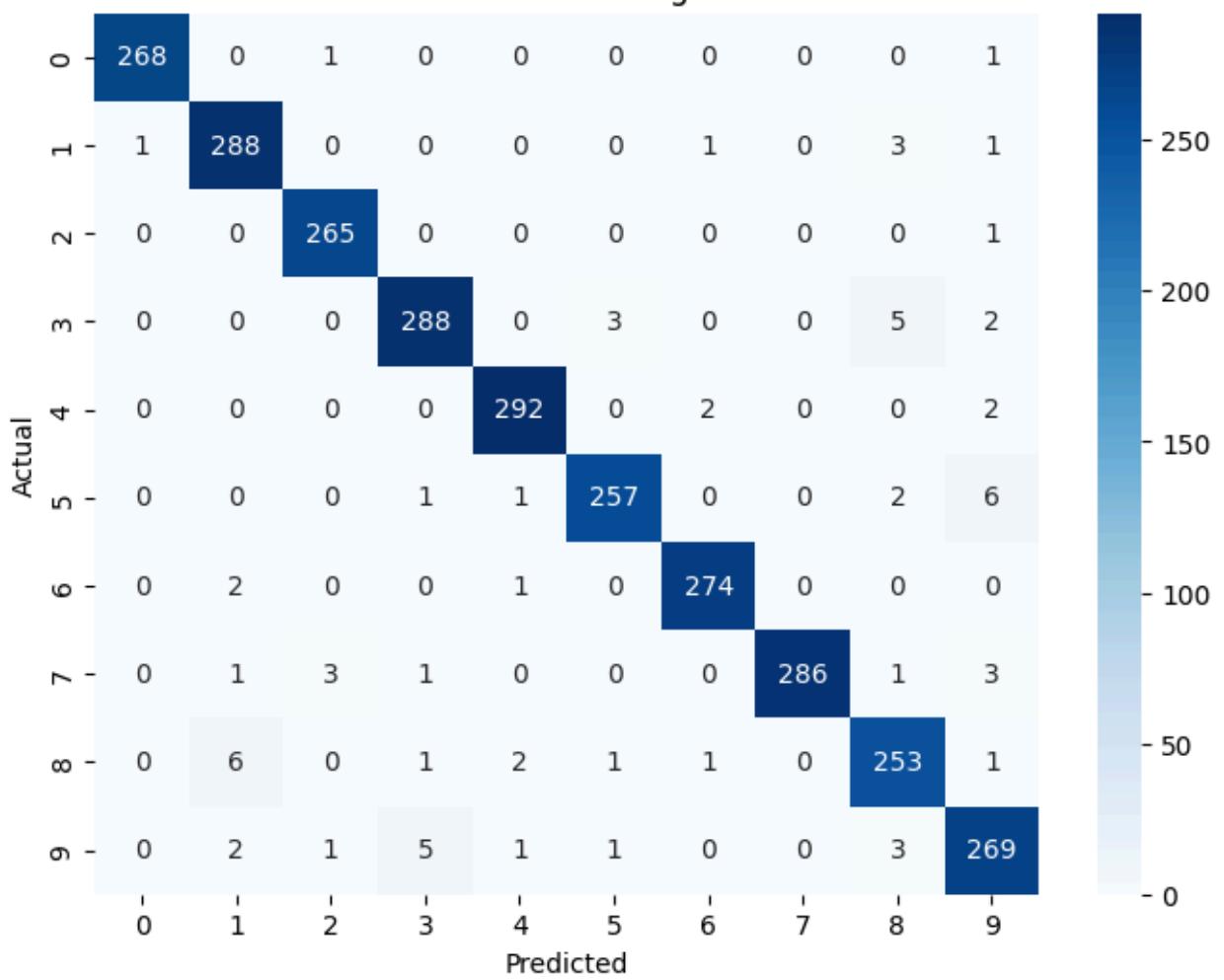
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

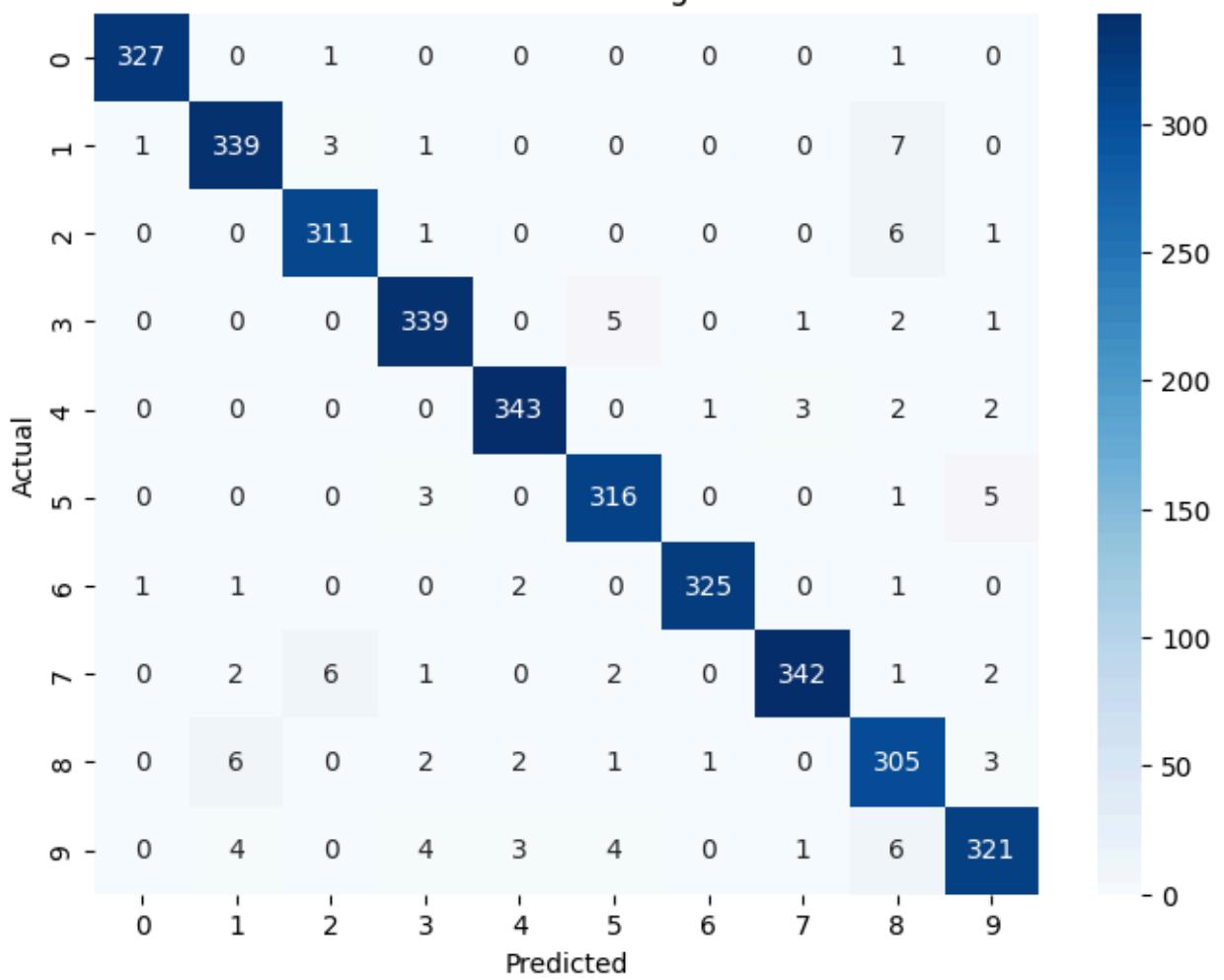
for cm_data in confusion_matrices_mlp:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix: Training Size: {training_size}%')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

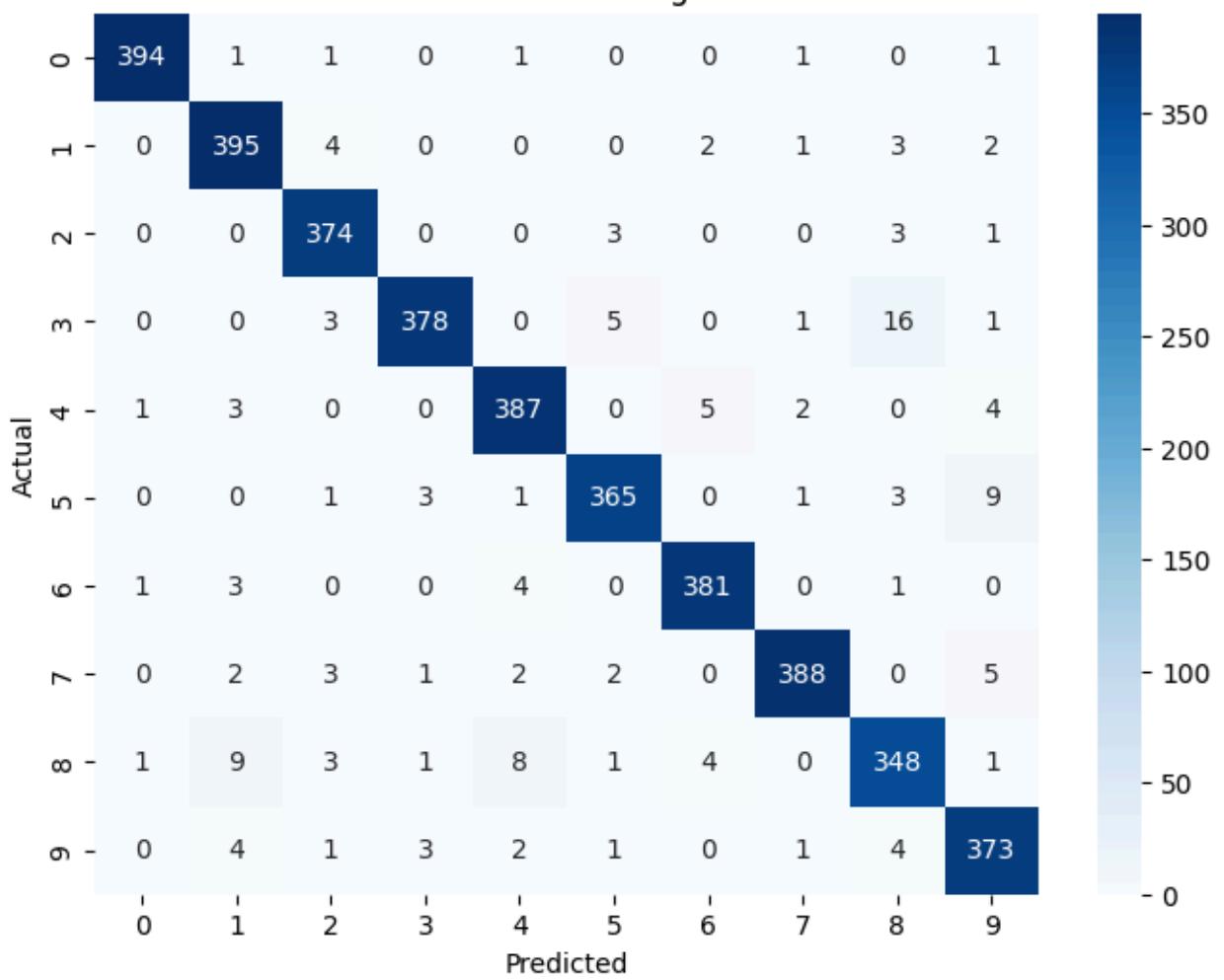
Confusion Matrix: Training Size: 50%

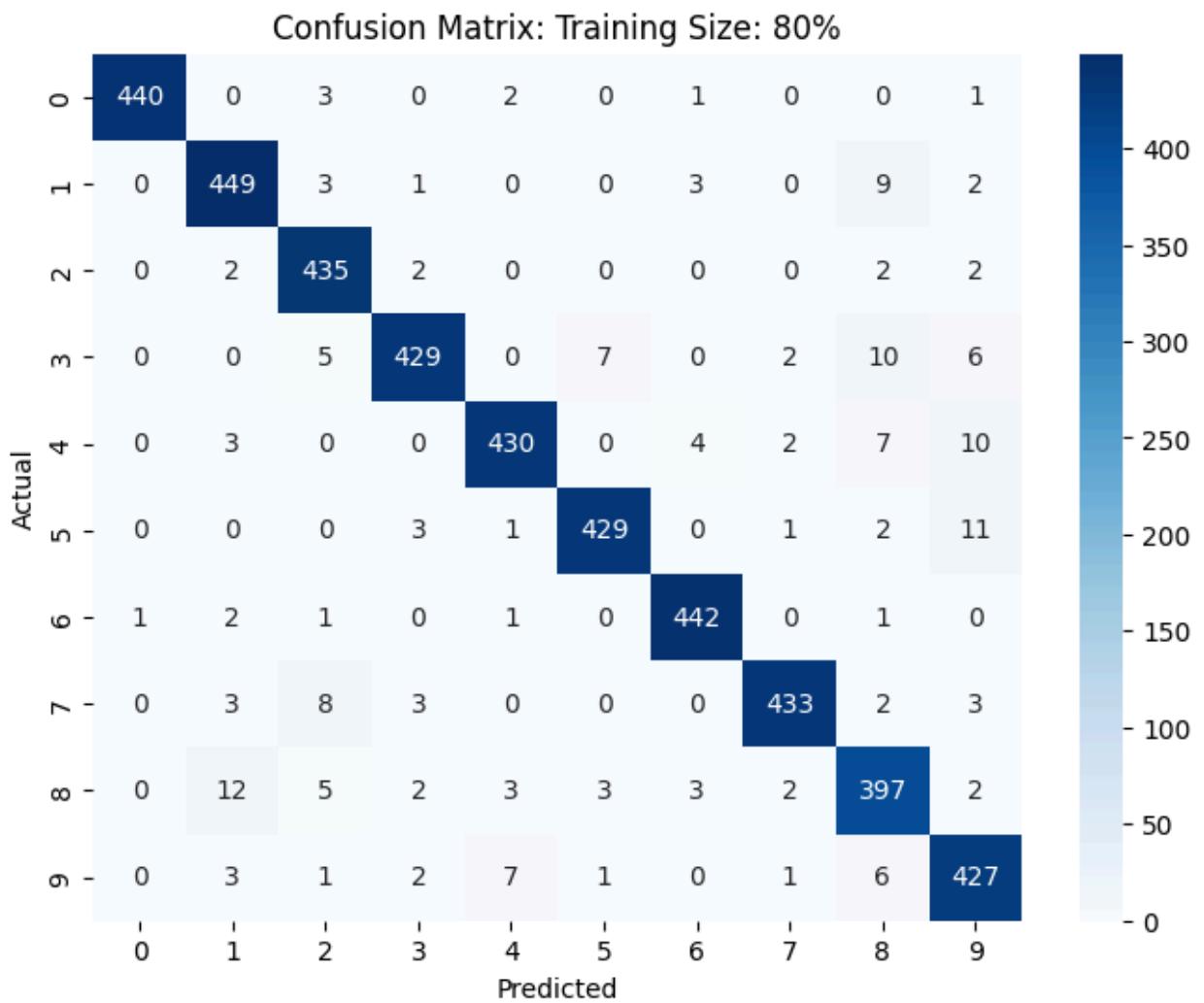


Confusion Matrix: Training Size: 60%



Confusion Matrix: Training Size: 70%



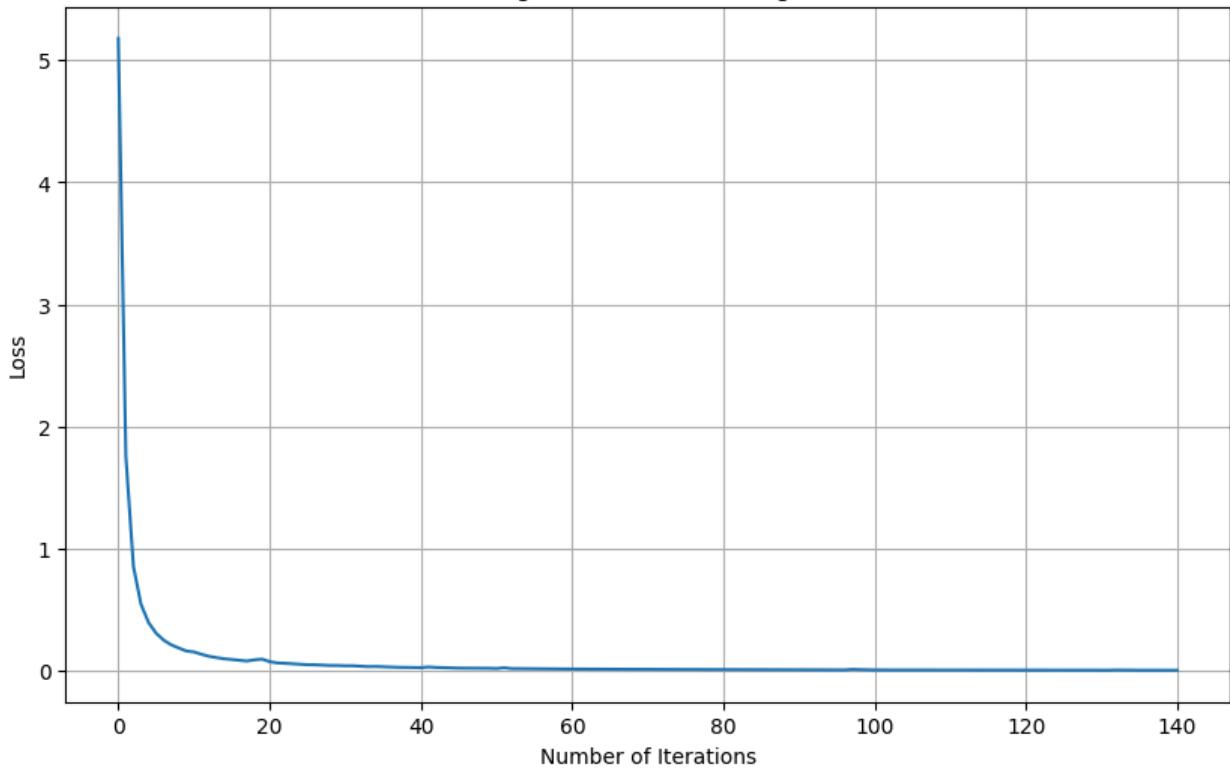


## Training Loss generation curve

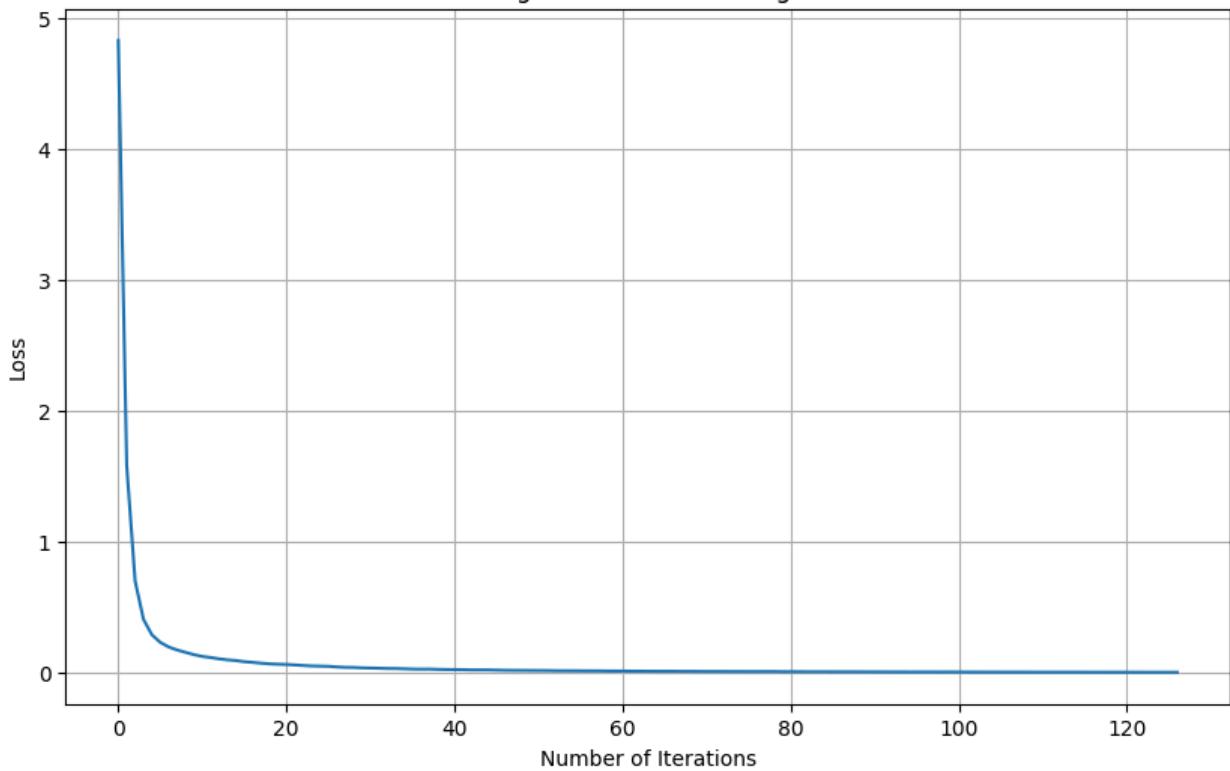
```
In [ ]: # Plot training loss curve for each trained MLP model
for model_data in trained_models_mlp:
    model = model_data["Model"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(10, 6))
    plt.plot(model.loss_curve_)
    plt.title(f'MLP Training Loss Curve: Training Size: {training_size}%')
    plt.xlabel('Number of Iterations')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()
```

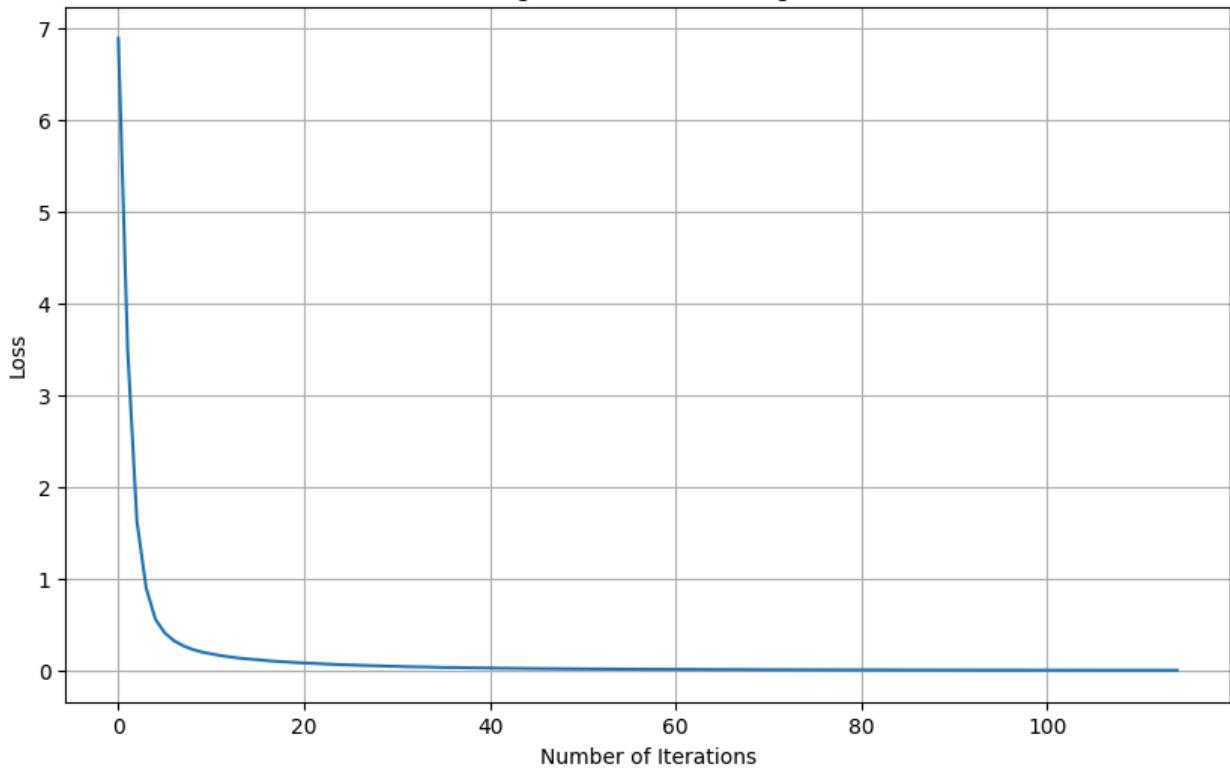
MLP Training Loss Curve: Training Size: 50%



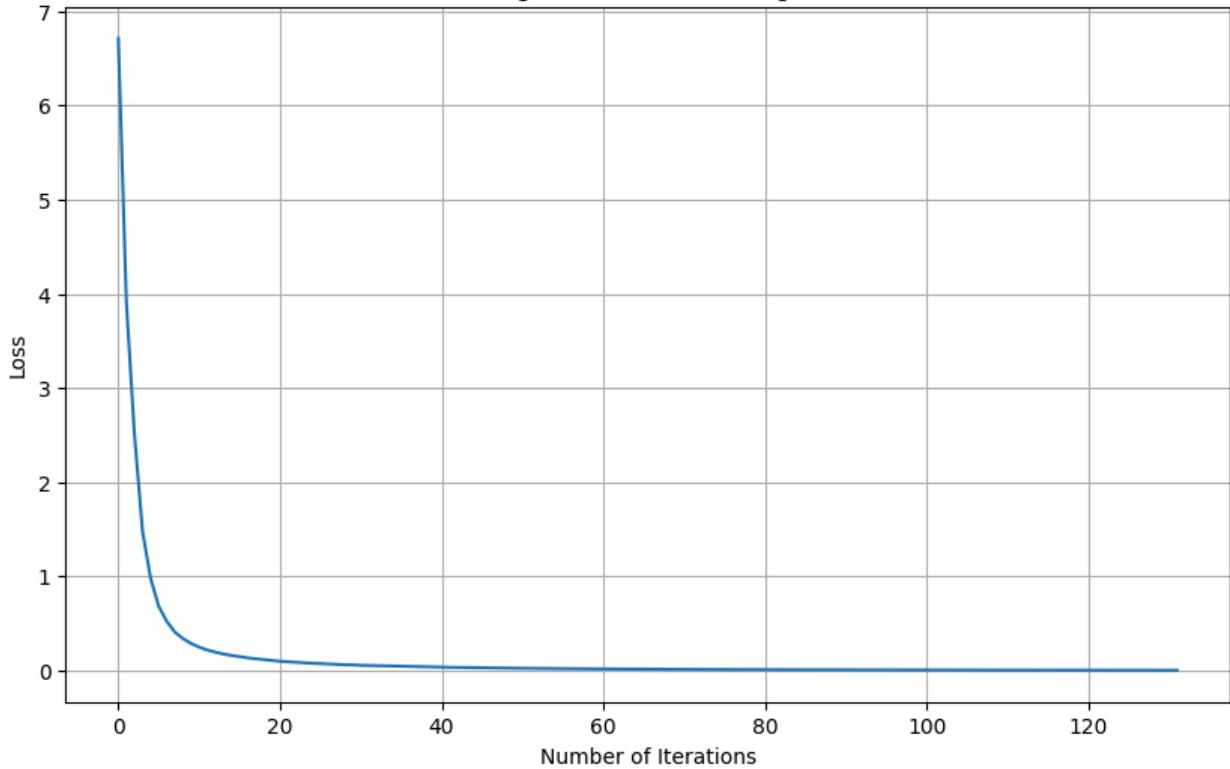
MLP Training Loss Curve: Training Size: 60%



MLP Training Loss Curve: Training Size: 70%



MLP Training Loss Curve: Training Size: 80%



## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc
```

```

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models_mlp:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes = len(np.unique(y_test))

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curve for each class
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

    plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for Training Size: {training_size}%') # Add title for plot
    plt.legend(loc="lower right")
    plt.show()

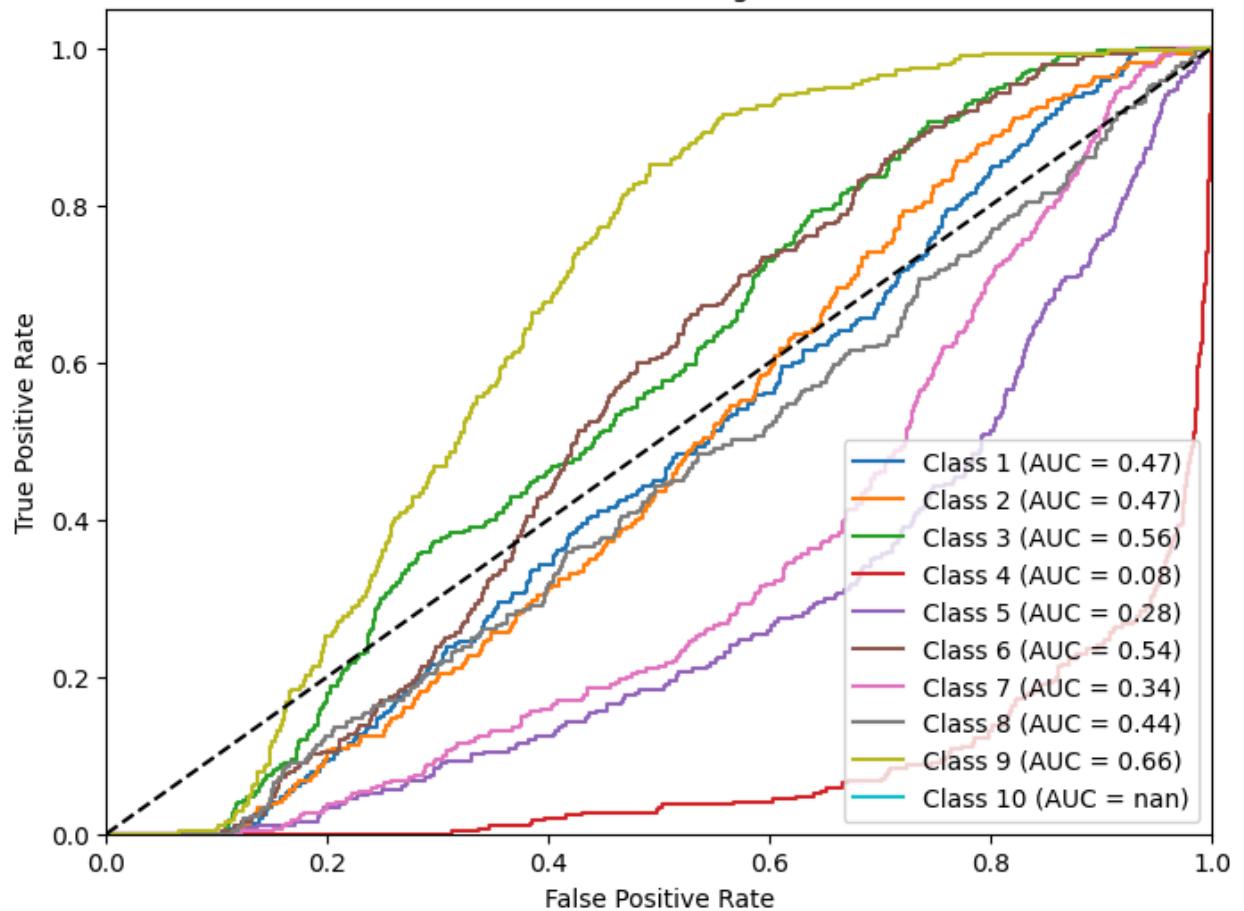
```

```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(

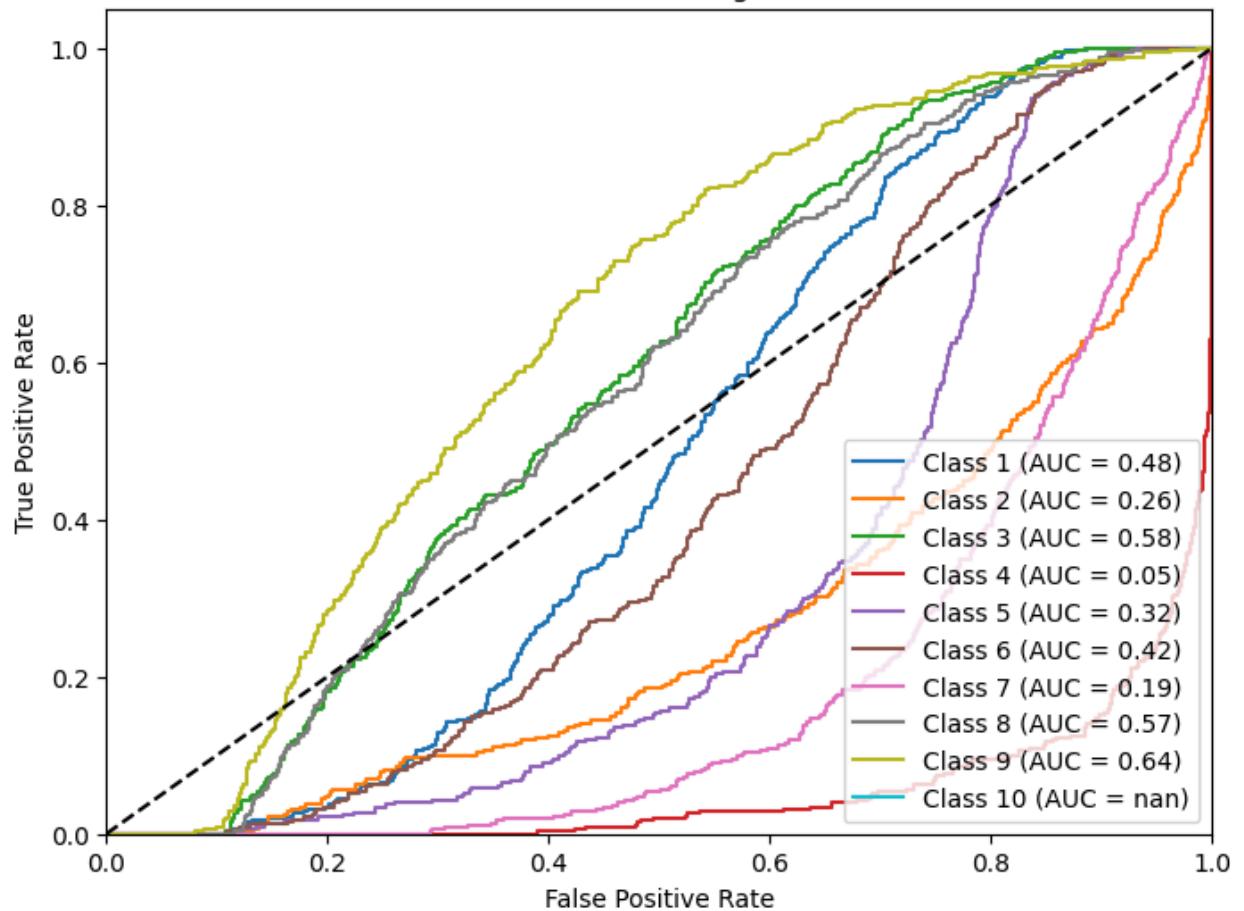
```

ROC Curve for Training Size: 50%



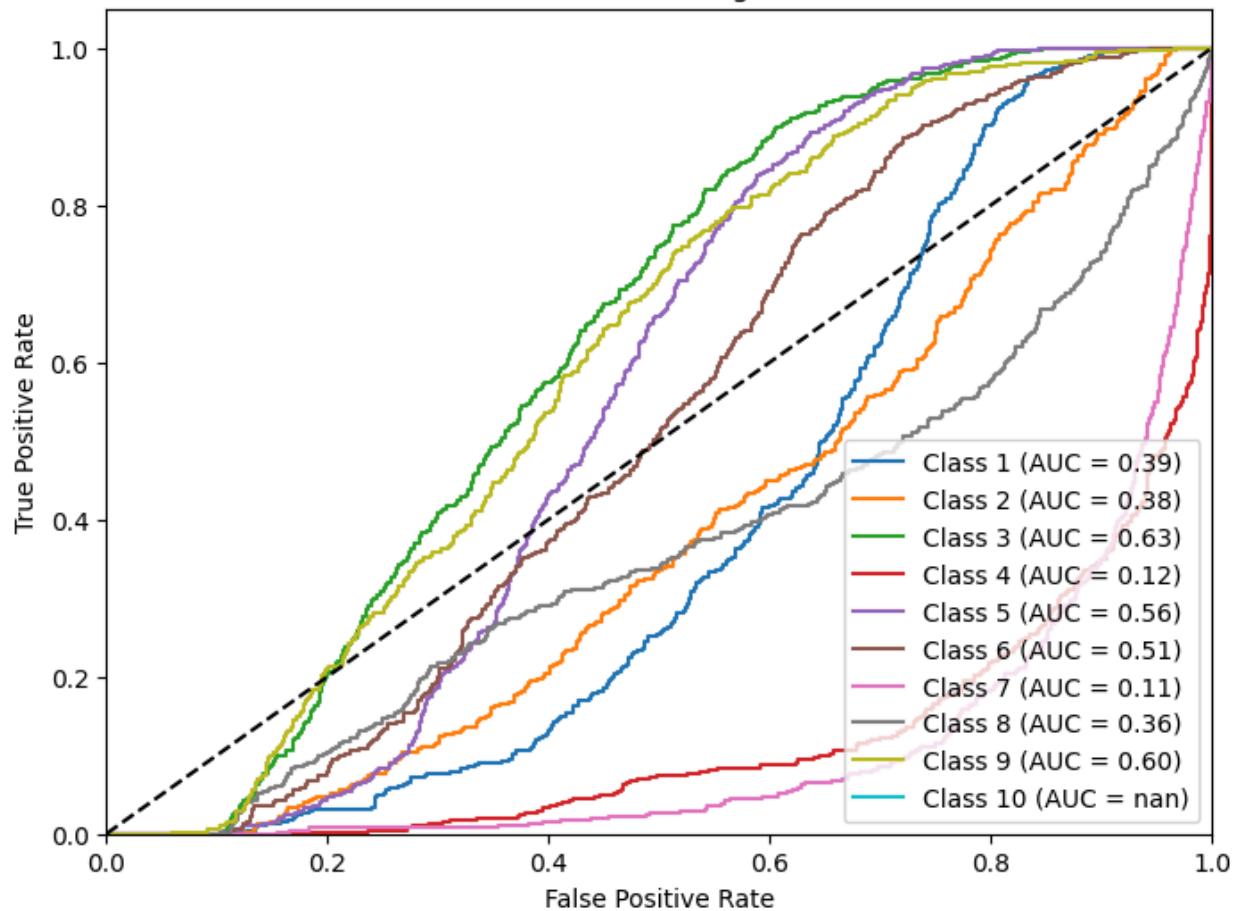
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve for Training Size: 60%

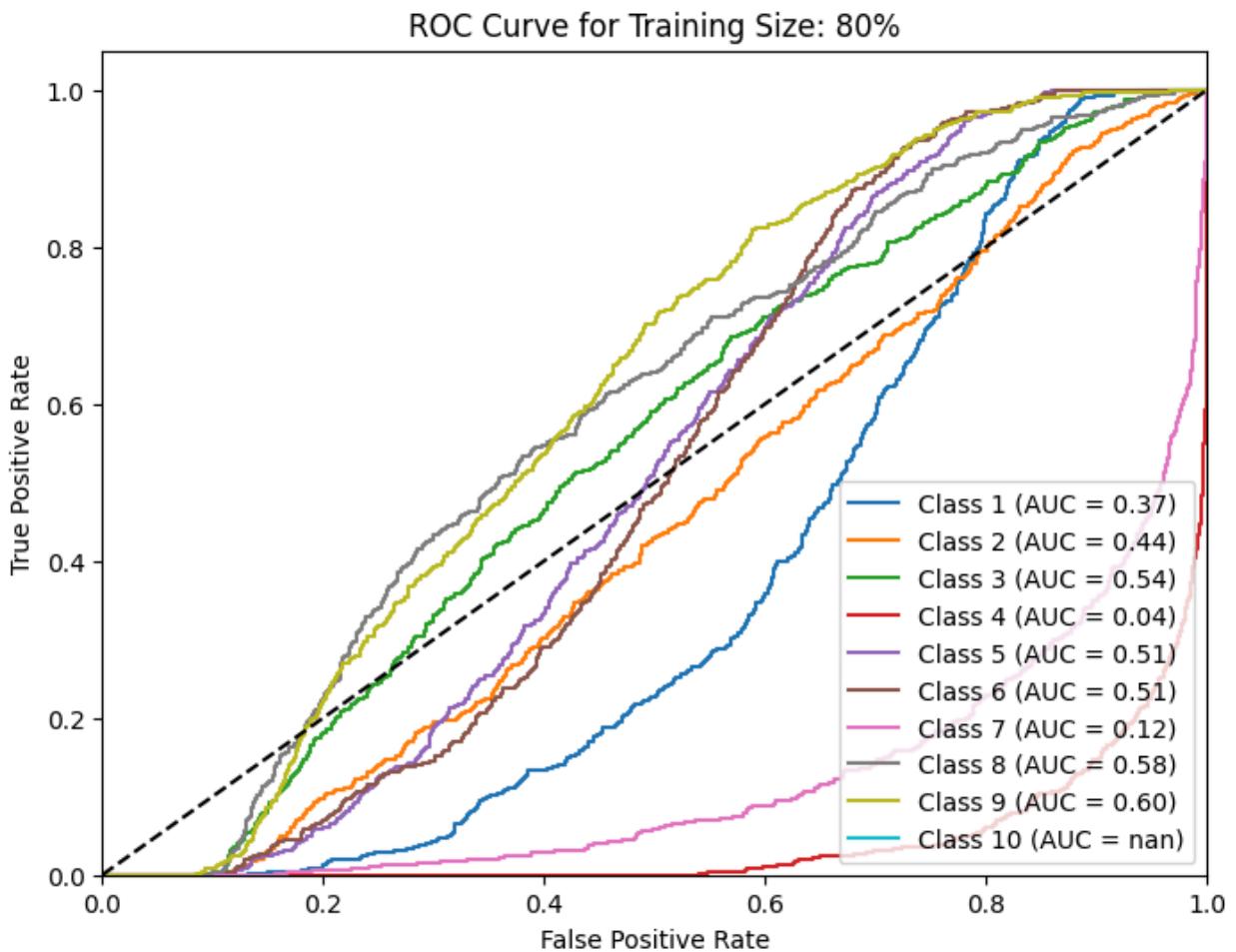


```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve for Training Size: 70%



```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```



## MLP with PCA-Reduced Data

```
In [ ]: # Apply the existing PCA transformation to the scaled data
X_pca_reduced_mlp = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced_mlp.shape[1]}")
print("\nData after PCA for Random Forest:")
display(pd.DataFrame(X_pca_reduced_mlp).head())
```

Original number of features: 64  
 Reduced number of features after PCA: 50

Data after PCA for Random Forest:

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	
<b>0</b>	-0.296633	-1.446450	-3.863044	-3.223597	0.773631	0.631849	0.163733	0.474
<b>1</b>	-0.156717	-3.065688	-5.811656	-3.232099	1.064670	0.031353	0.270302	2.091
<b>2</b>	-0.753929	3.261949	0.779656	-1.108976	0.311024	-0.814548	-4.307315	0.841
<b>3</b>	-4.226579	1.900320	-0.441486	1.329548	-0.149220	2.710149	2.128209	0.206
<b>4</b>	0.582983	-3.490539	-1.602212	-1.052587	-1.360737	0.406474	-0.662578	-2.572

5 rows × 50 columns

## Calculate MLP performance with PCA-Reduced Data for different test sizes

```
In [ ]: results_mlp_pca = []
confusion_matrices_mlp_pca = []
trained_models_mlp_pca = []

for size in training_sizes:
    X_train_pca_mlp, X_test_pca_mlp, y_train, y_test = train_test_split(X_pca_reduced, test_size=size)
    mlp_pca = MLPClassifier(max_iter=1000)

    mlp_pca.fit(X_train_pca_mlp, y_train.values.ravel())
    y_pred_pca_mlp = mlp_pca.predict(X_test_pca_mlp)

    acc_pca_mlp = accuracy_score(y_test, y_pred_pca_mlp)
    precision_pca_mlp = precision_score(y_test, y_pred_pca_mlp, average='weighted')
    recall_pca_mlp = recall_score(y_test, y_pred_pca_mlp, average='weighted')
    f1_pca_mlp = f1_score(y_test, y_pred_pca_mlp, average='weighted')

    cm_pca_mlp = confusion_matrix(y_test, y_pred_pca_mlp)

    results_mlp_pca.append({
        "Training size":int (size*100),
        "Accuracy":acc_pca_mlp,
        "Precision": precision_pca_mlp,
        "Recall": recall_pca_mlp,
        "F1-score": f1_pca_mlp
    })

    confusion_matrices_mlp_pca.append({
        "Training size":int (size*100),
        "Confusion Matrix":cm_pca_mlp
    })

    trained_models_mlp_pca.append({
        "Training size":int (size*100),
        "Model": mlp_pca,
```

```
 })
```

## Print MLP Performance Table and Graphs with PCA-Reduced Data

```
In [ ]: df_mlp_pca = pd.DataFrame(results_mlp_pca)
display(df_mlp_pca)

# Plot Accuracy
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Accuracy')
plt.title('Random Forest Accuracy by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.show()

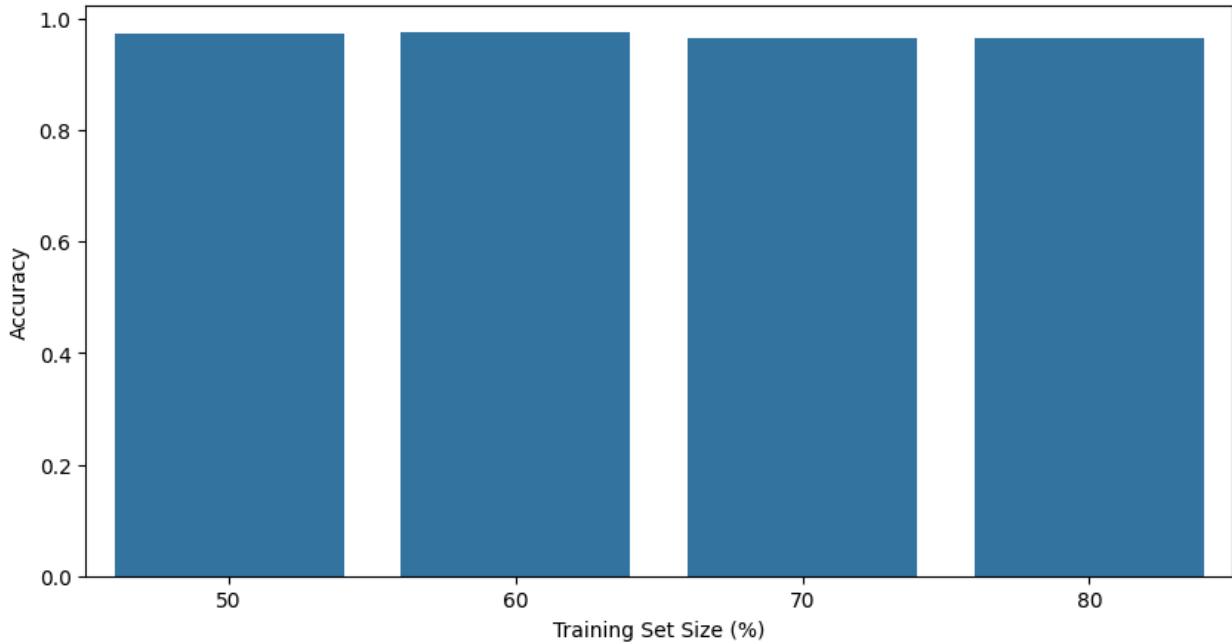
# Plot Precision
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Precision')
plt.title('Random Forest Precision by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Precision')
plt.show()

# Plot Recall
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='Recall')
plt.title('Random Forest Recall by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Recall')
plt.show()

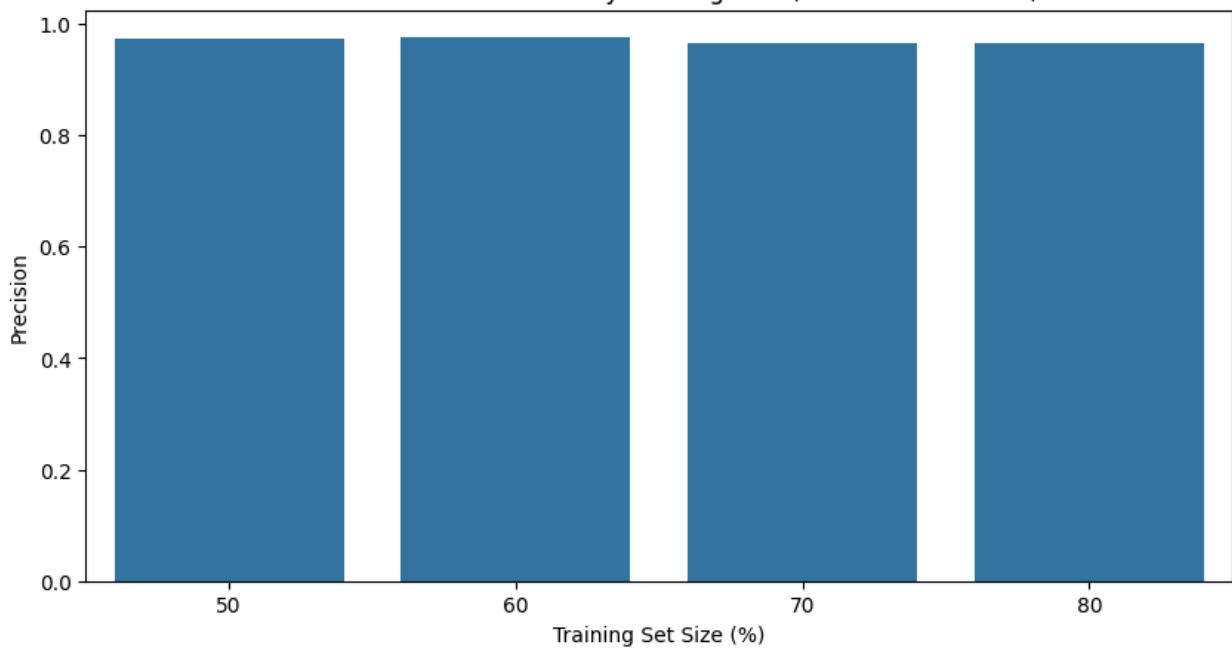
# Plot F1-score
plt.figure(figsize=(10, 5))
sns.barplot(data=df_mlp_pca, x='Training size', y='F1-score')
plt.title('Random Forest F1-score by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('F1-score')
plt.show()
```

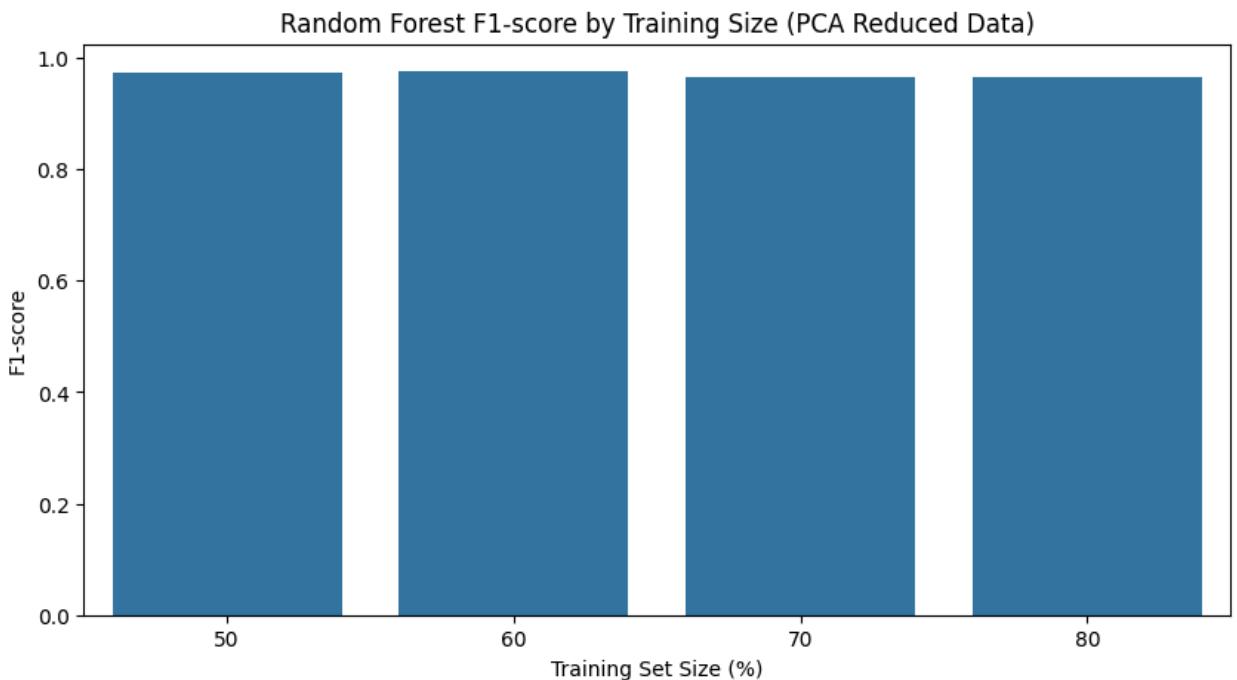
	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.973665	0.973913	0.973665	0.973740
1	60	0.975979	0.976062	0.975979	0.975998
2	70	0.966446	0.966648	0.966446	0.966444
3	80	0.965970	0.966241	0.965970	0.966018

Random Forest Accuracy by Training Size (PCA Reduced Data)



Random Forest Precision by Training Size (PCA Reduced Data)





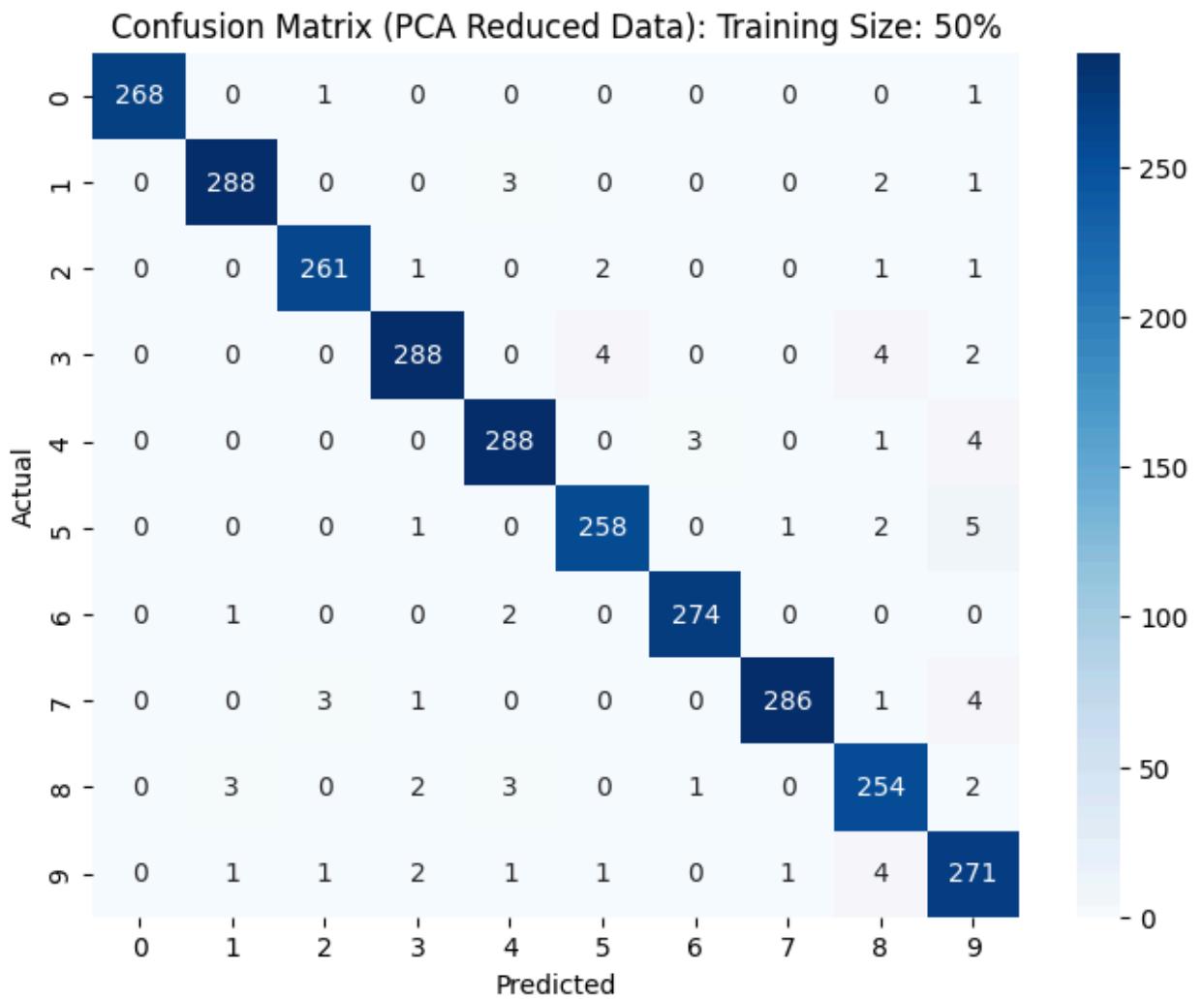
## Show Confusion Matrices for MLP with PCA-Reduced Data

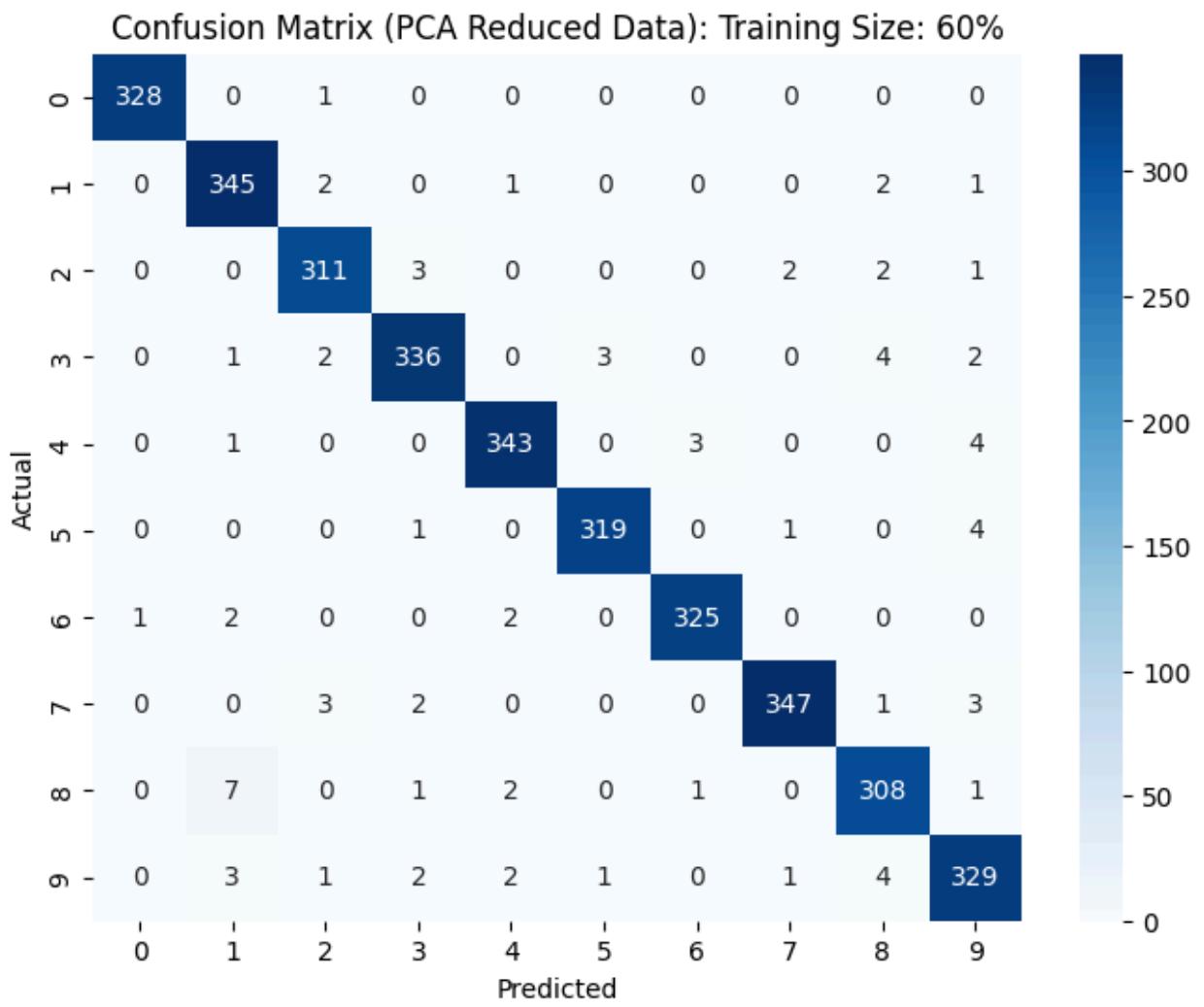
```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

for cm_data in confusion_matrices_mlp_pca:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

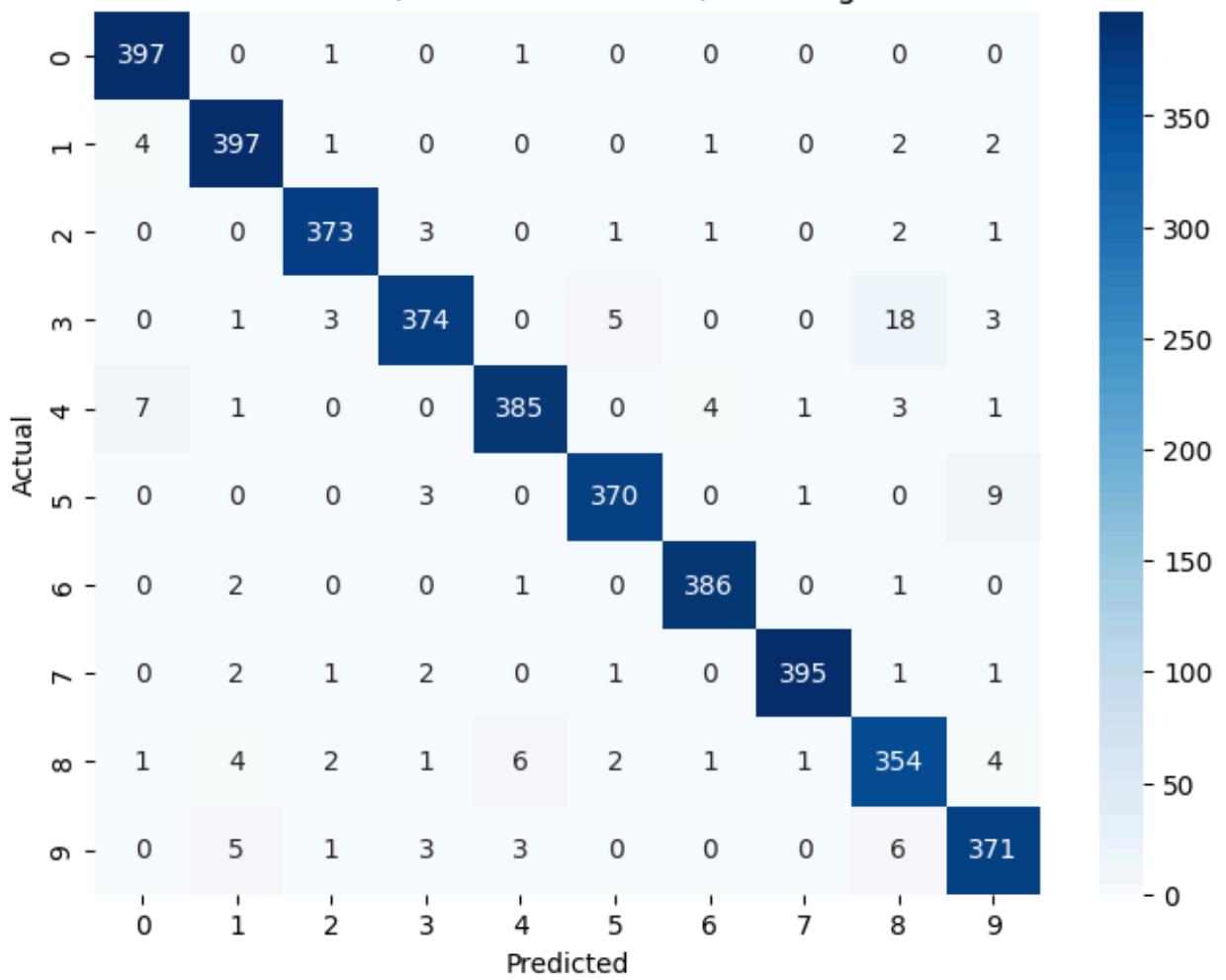
    plt.figure(figsize=(8, 6))
```

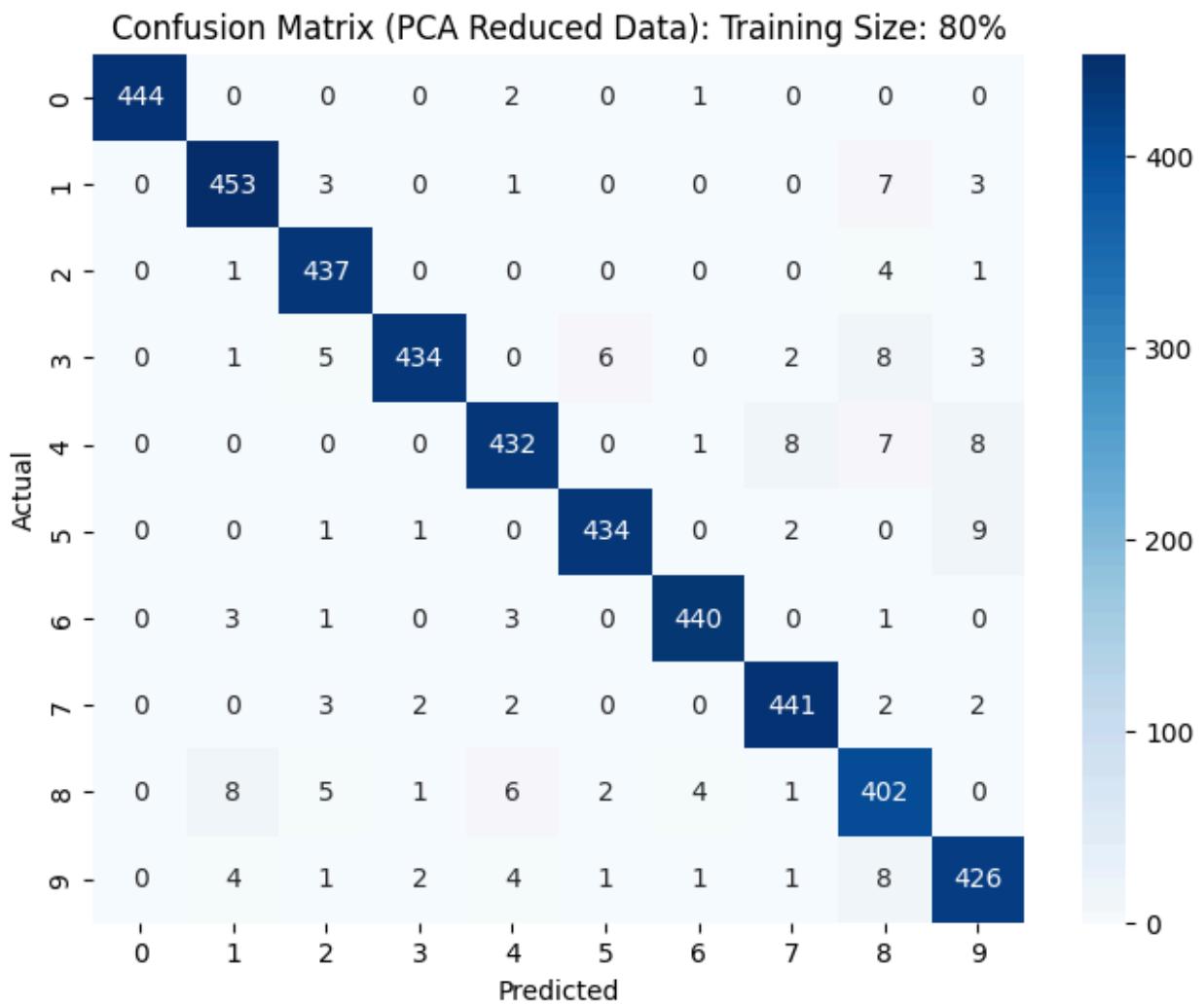
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix (PCA Reduced Data): Training Size: {training_size}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```





Confusion Matrix (PCA Reduced Data): Training Size: 70%



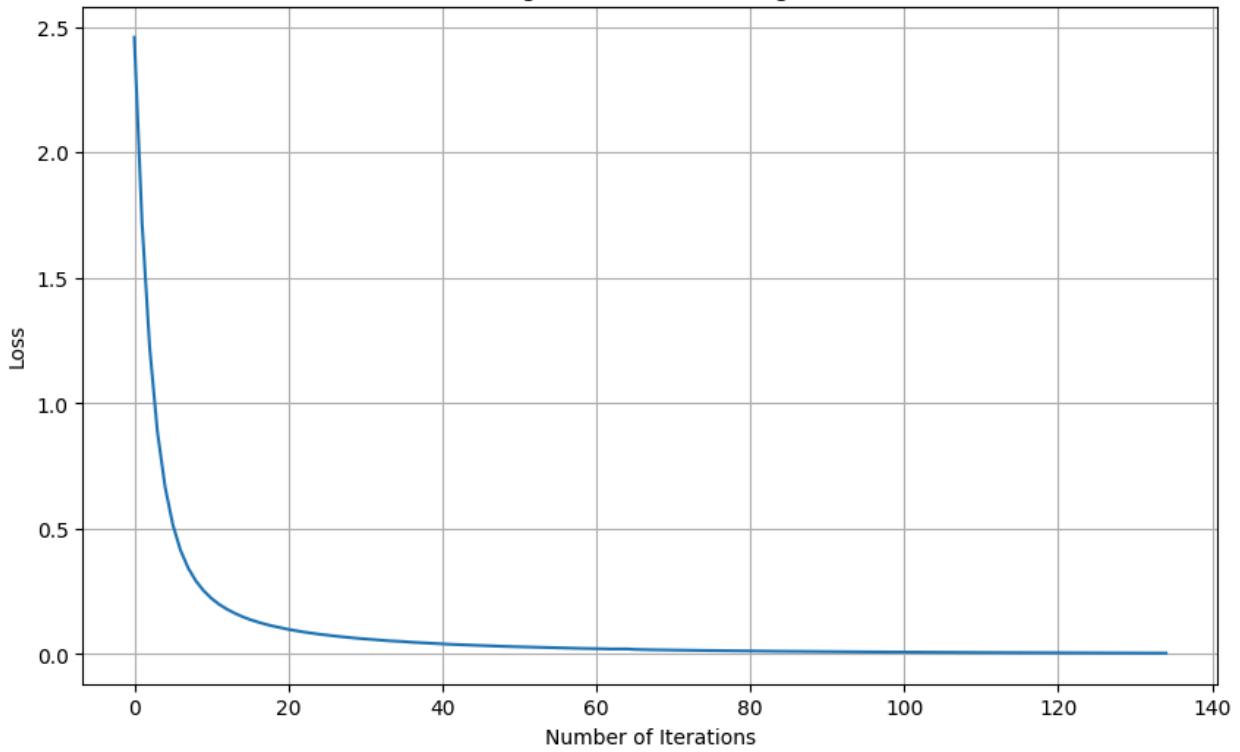


## Training Loss generation curve with PCA data

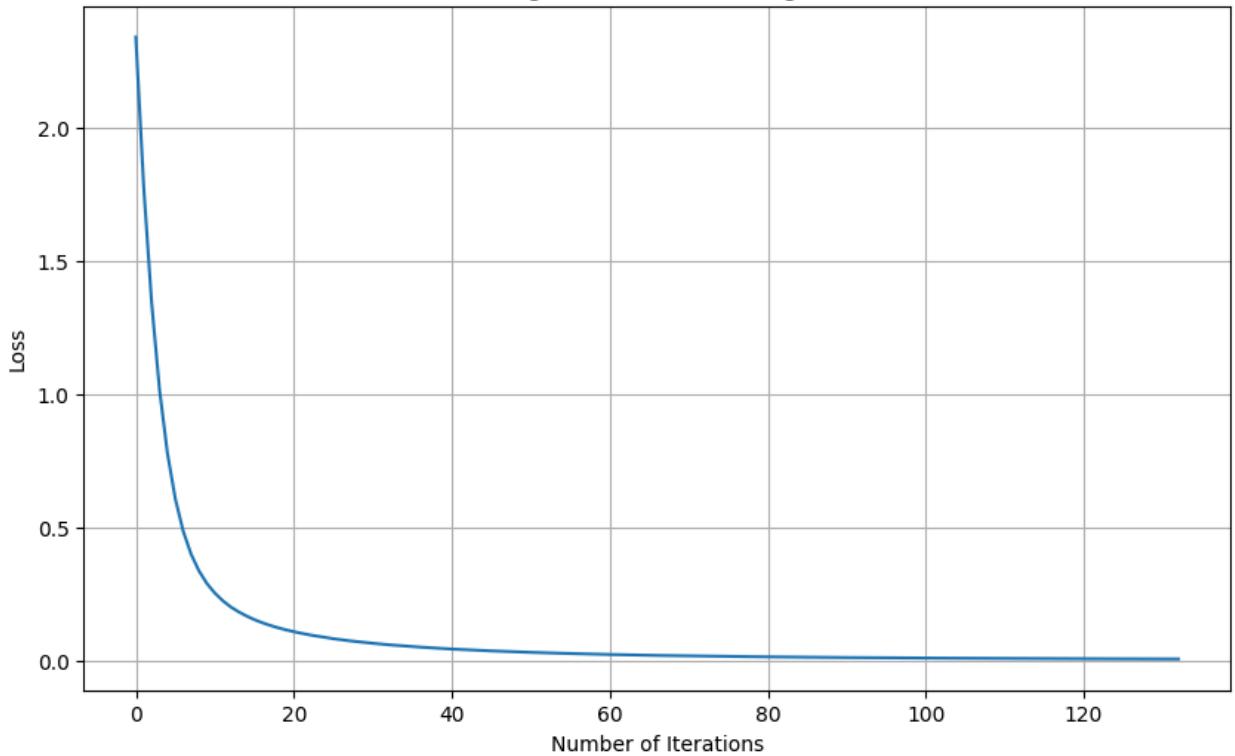
```
In [ ]: # Plot training loss curve for each trained MLP model
for model_data in trained_models_mlp_pca:
    model = model_data["Model"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(10, 6))
    plt.plot(model.loss_curve_)
    plt.title(f'MLP Training Loss Curve: Training Size: {training_size}%')
    plt.xlabel('Number of Iterations')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()
```

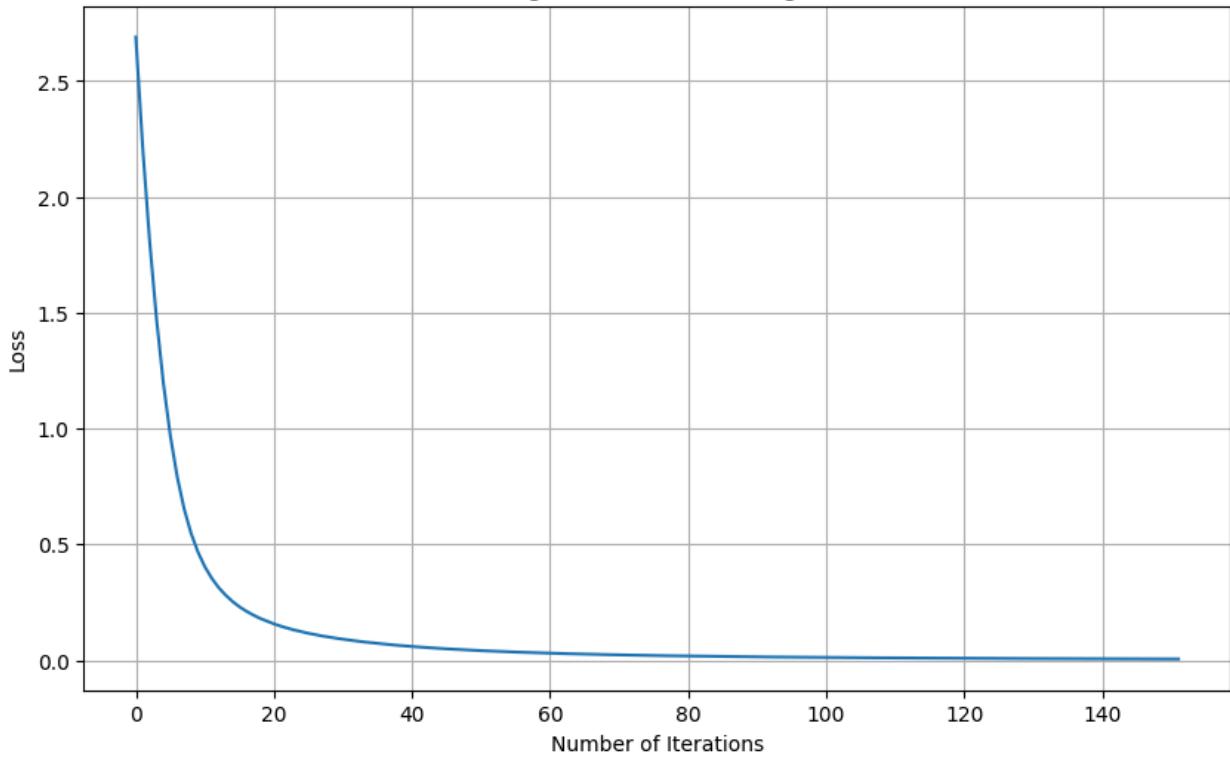
MLP Training Loss Curve: Training Size: 50%



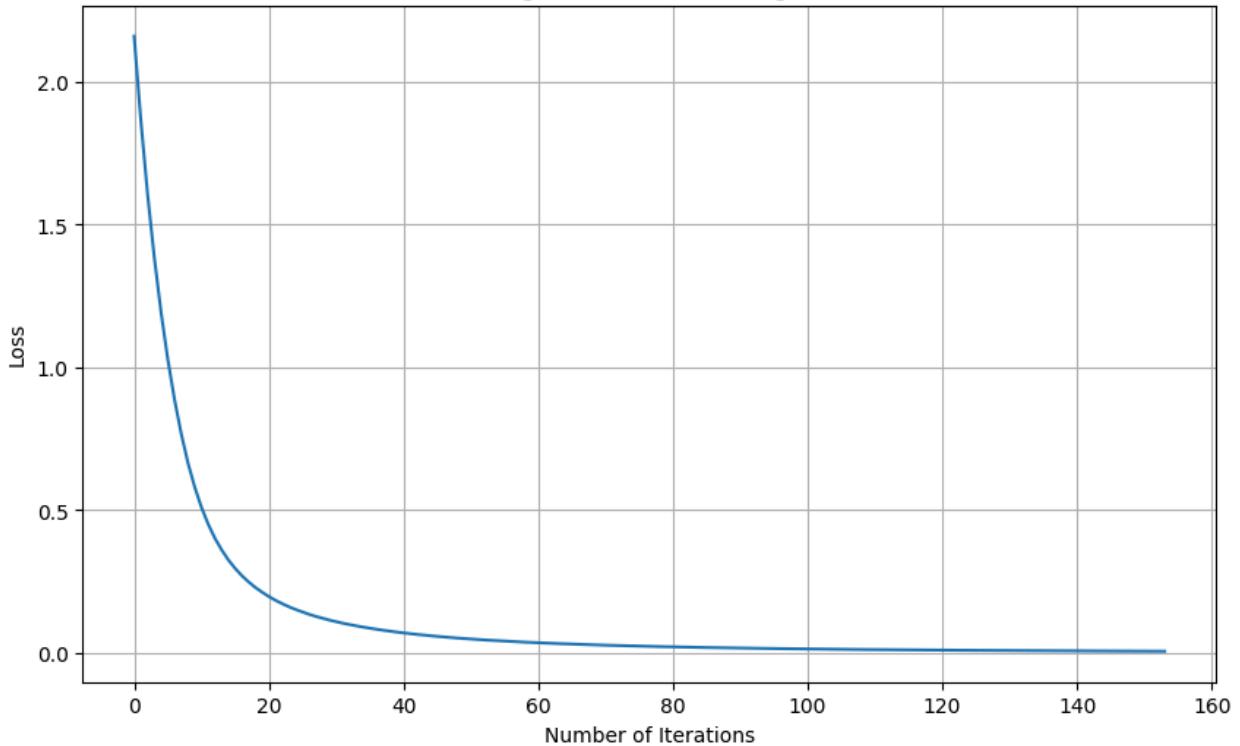
MLP Training Loss Curve: Training Size: 60%



MLP Training Loss Curve: Training Size: 70%



MLP Training Loss Curve: Training Size: 80%



# Random Forest

## Calculate values for different test size

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

training_sizes = [0.5, 0.6, 0.7, 0.8]

results_rf = []
confusion_matrices_rf = []
trained_models_rf = [] # Add a list to store trained models

for size in training_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size, random_state=42)
    rf = RandomForestClassifier(random_state=42)

    rf.fit(X_train, y_train.values.ravel())
    y_pred_test = rf.predict(X_test)
    y_pred_train = rf.predict(X_train)

    acc_test = accuracy_score(y_test, y_pred_test)
    precision_test = precision_score(y_test, y_pred_test, average='weighted', zero_division=0)
    recall_test = recall_score(y_test, y_pred_test, average='weighted') # Calculate weighted recall
    f1_test = f1_score(y_test, y_pred_test, average='weighted') # Calculate weighted F1-score

    acc_train = accuracy_score(y_train, y_pred_train)
    precision_train = precision_score(y_train, y_pred_train, average='weighted', zero_division=0)
    recall_train = recall_score(y_train, y_pred_train, average='weighted')
    f1_train = f1_score(y_train, y_pred_train, average='weighted')

    cm = confusion_matrix(y_test, y_pred_test) # Calculate confusion matrix

    results_rf.append({
        "Training size":int (size*100),
        "Accuracy":acc_test,
        "Precision":precision_test,
        "Recall": recall_test,
        "F1-score": f1_test
    })

    confusion_matrices_rf.append({ # Store confusion matrix with metadata
        "Training size":int (size*100),
        "Confusion Matrix":cm
    })
```

```
})

trained_models_rf.append({ # Store the trained model with metadata
    "Training size":int (size*100),
    "Model": rf,
    "X_test": X_test,
    "y_test": y_test
})
```

## Print Evaluation Table and Graphs

```
In [ ]: df = pd.DataFrame(results_rf)
display(df) # Display the DataFrame as a table

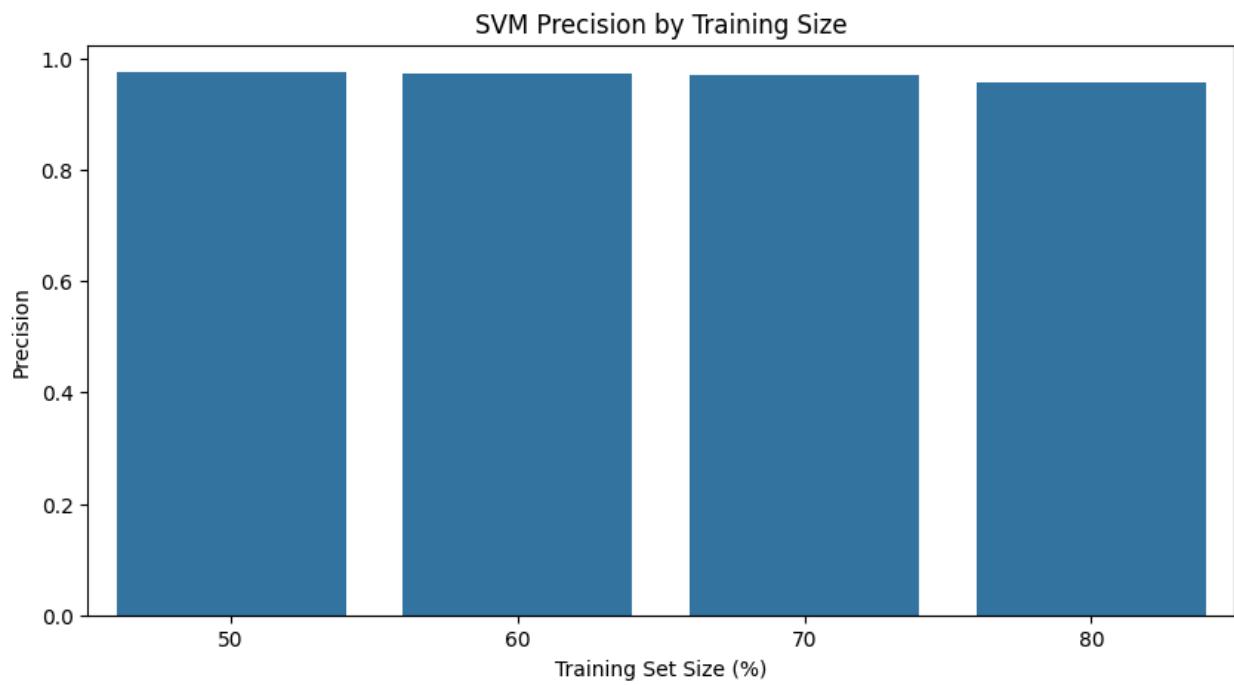
# Plot Accuracy
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Accuracy')
plt.title('SVM Accuracy by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Accuracy') # Add y-axis label
plt.show()

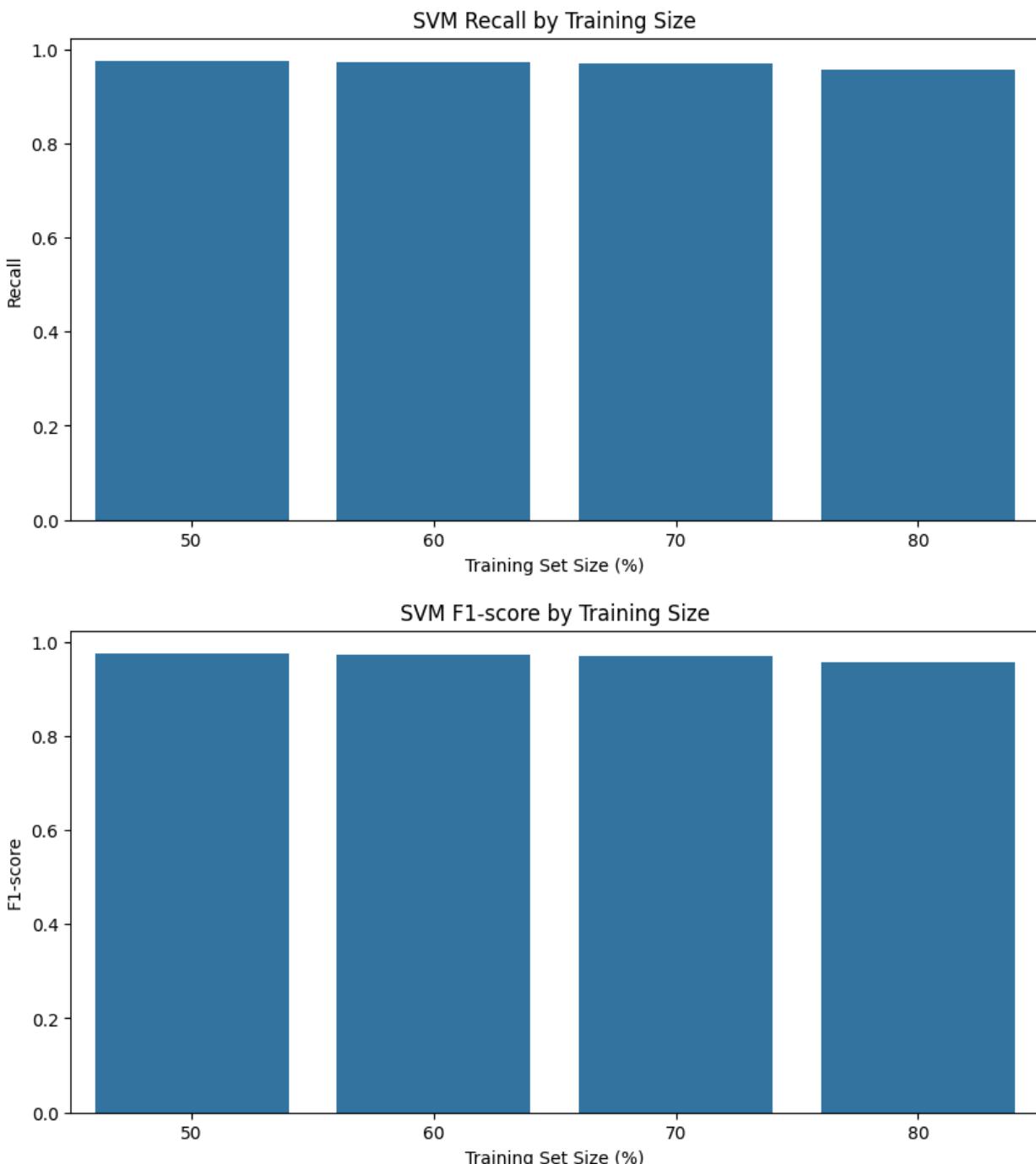
# Plot Precision
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Precision')
plt.title('SVM Precision by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Precision') # Add y-axis label
plt.show()

# Plot Recall
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall')
plt.title('SVM Recall by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

# Plot F1-score
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='F1-score')
plt.title('SVM F1-score by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('F1-score') # Add y-axis label
plt.show()
```

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.974733	0.975024	0.974733
<b>1</b>	60	0.972716	0.972770	0.972716
<b>2</b>	70	0.968480	0.968704	0.968480
<b>3</b>	80	0.956628	0.957079	0.956628





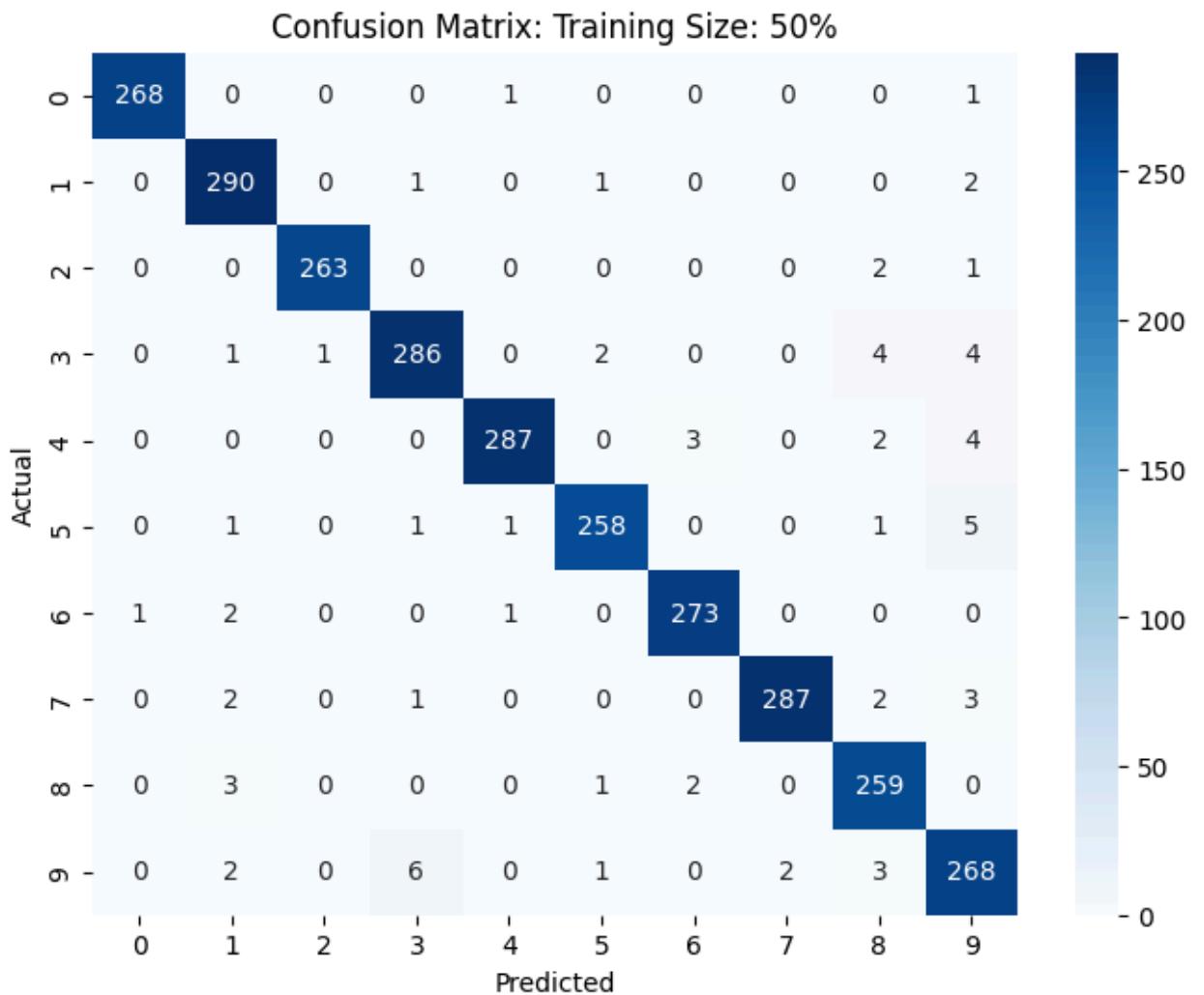
## Show Confusion Matrices

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

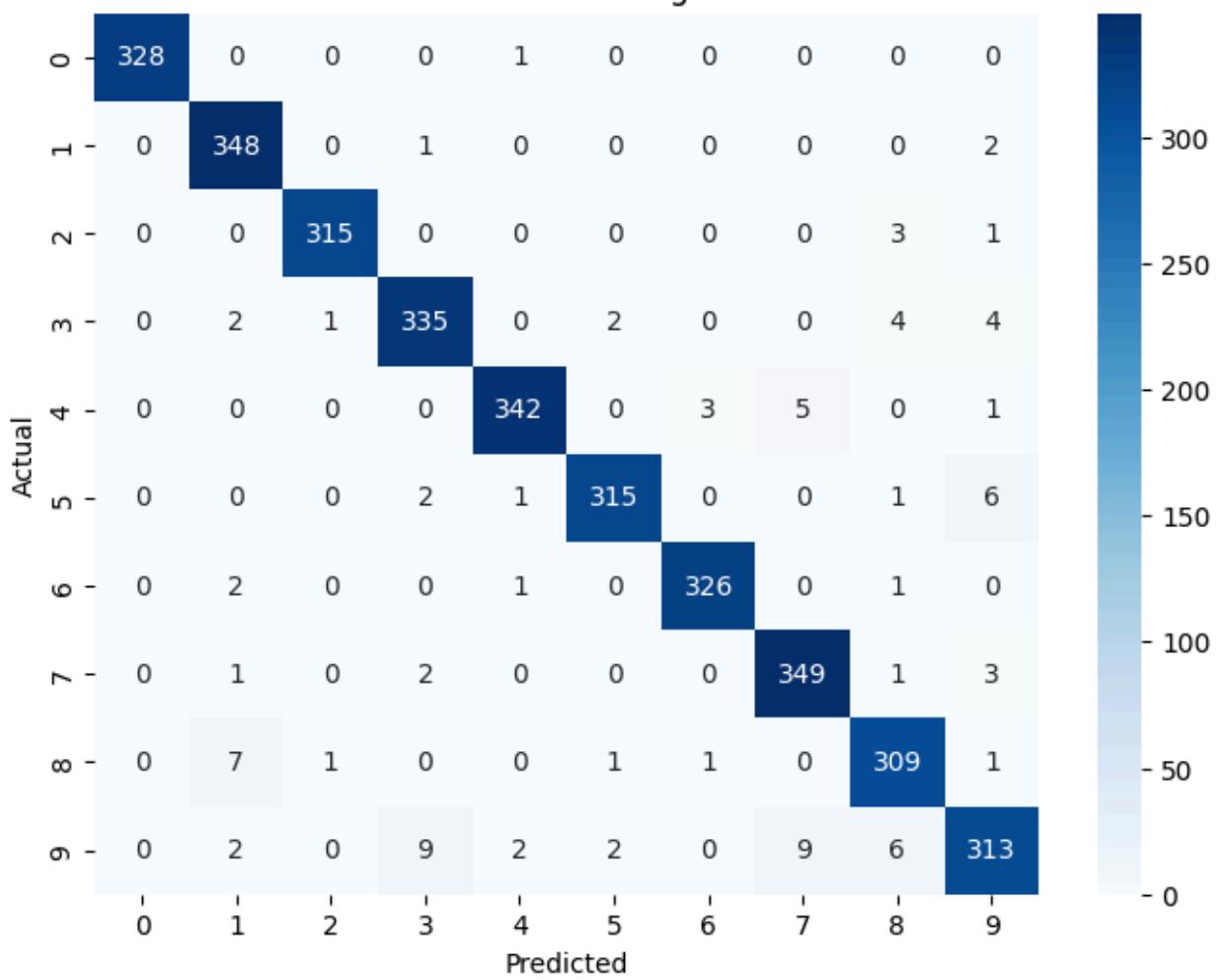
for cm_data in confusion_matrices_rf:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]

    plt.figure(figsize=(8, 6))
```

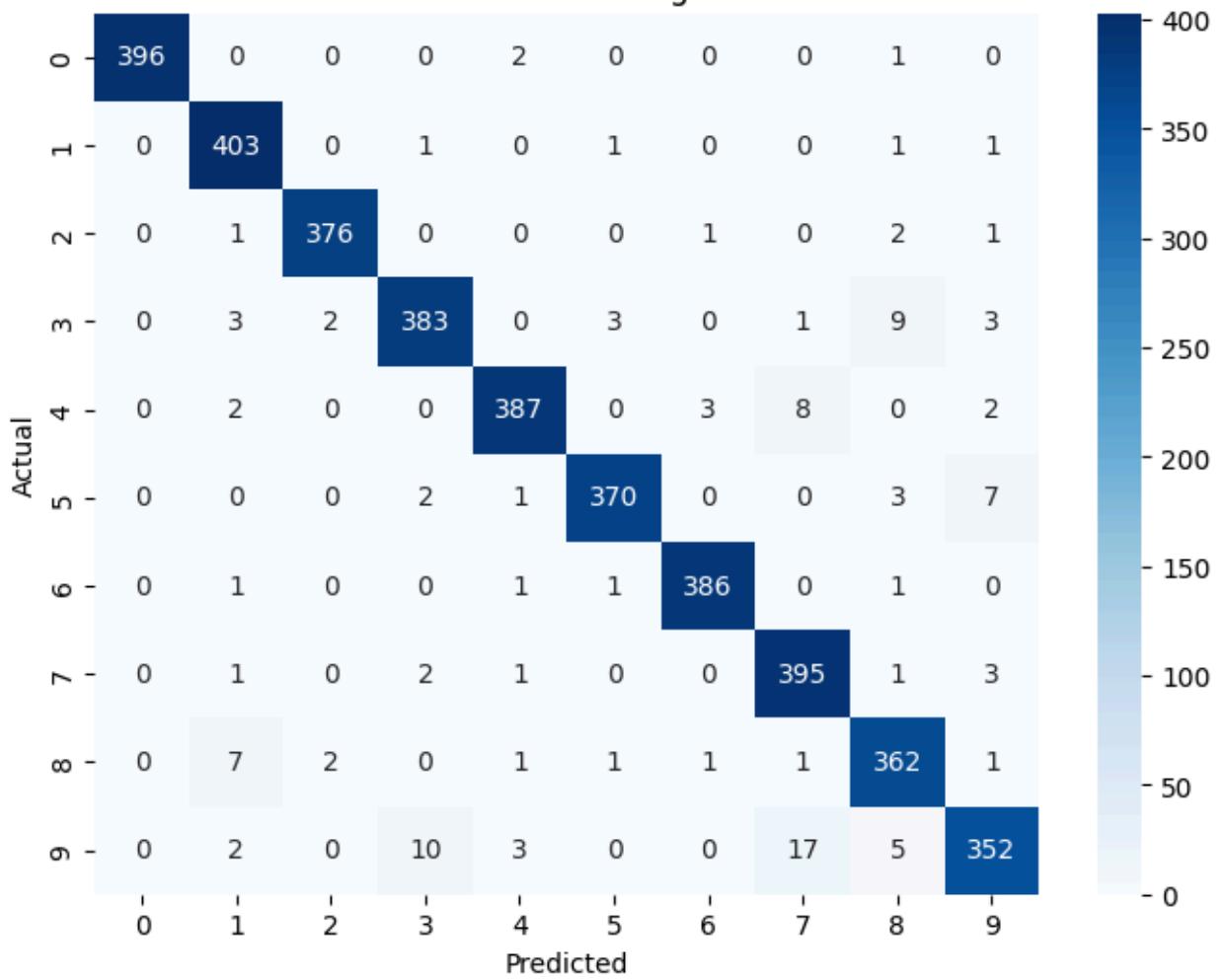
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix: Training Size: {training_size}%')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

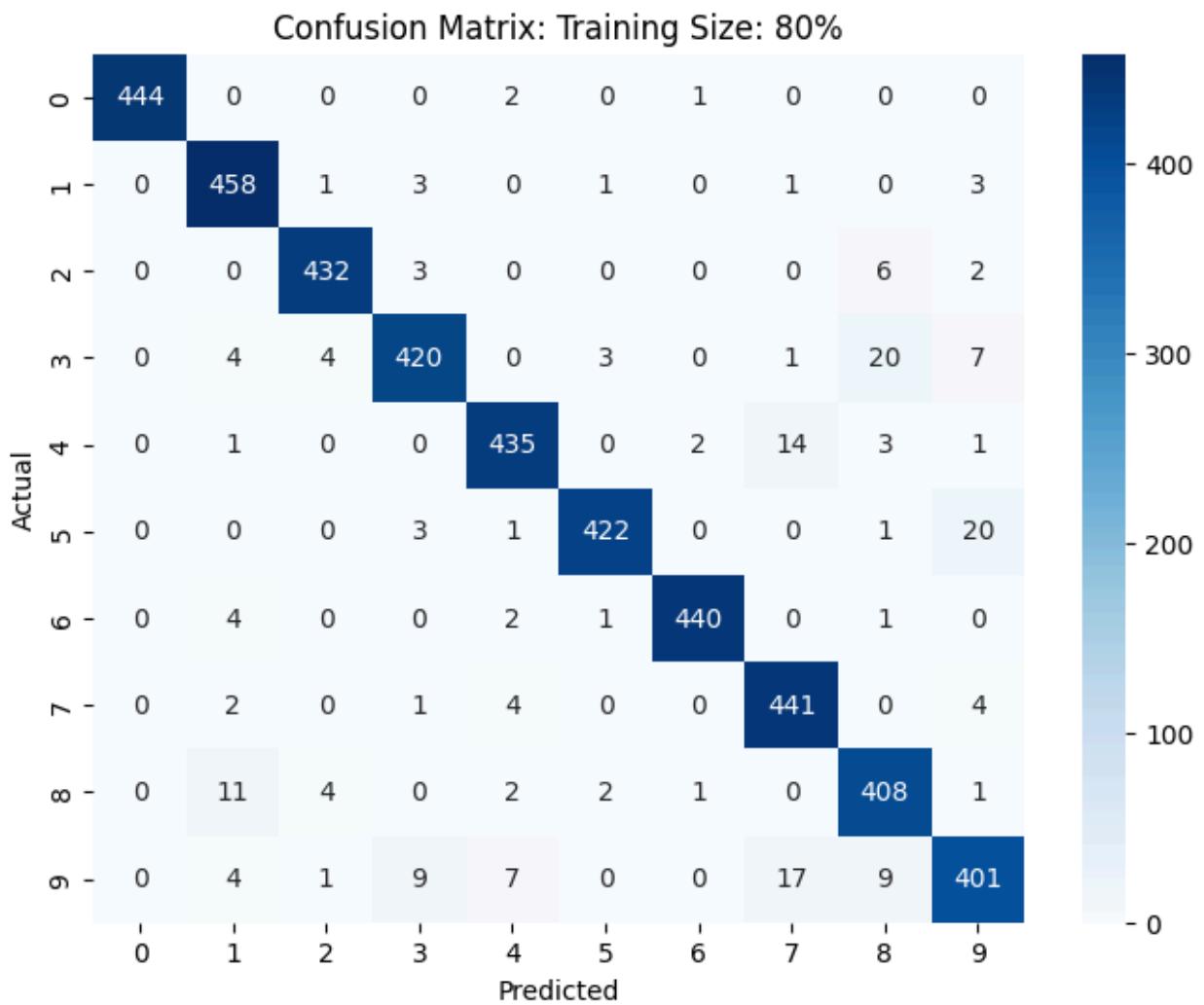


Confusion Matrix: Training Size: 60%



Confusion Matrix: Training Size: 70%





## ROC and AOC curve

```
In [ ]: from sklearn.metrics import roc_curve, auc

# Since this is a multi-class problem, we'll use a One-vs-Rest (OvR) approach
for model_data in trained_models_rf:
    model = model_data["Model"]
    X_test = model_data["X_test"]
    y_test = model_data["y_test"]
    training_size = model_data["Training size"]

    plt.figure(figsize=(8, 6)) # Create a new figure for each plot

    # Get predicted probabilities for each class
    y_prob = model.predict_proba(X_test)

    # Compute ROC curve and AUC for each class using OvR
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
```

```

n_classes = len(np.unique(y_test))

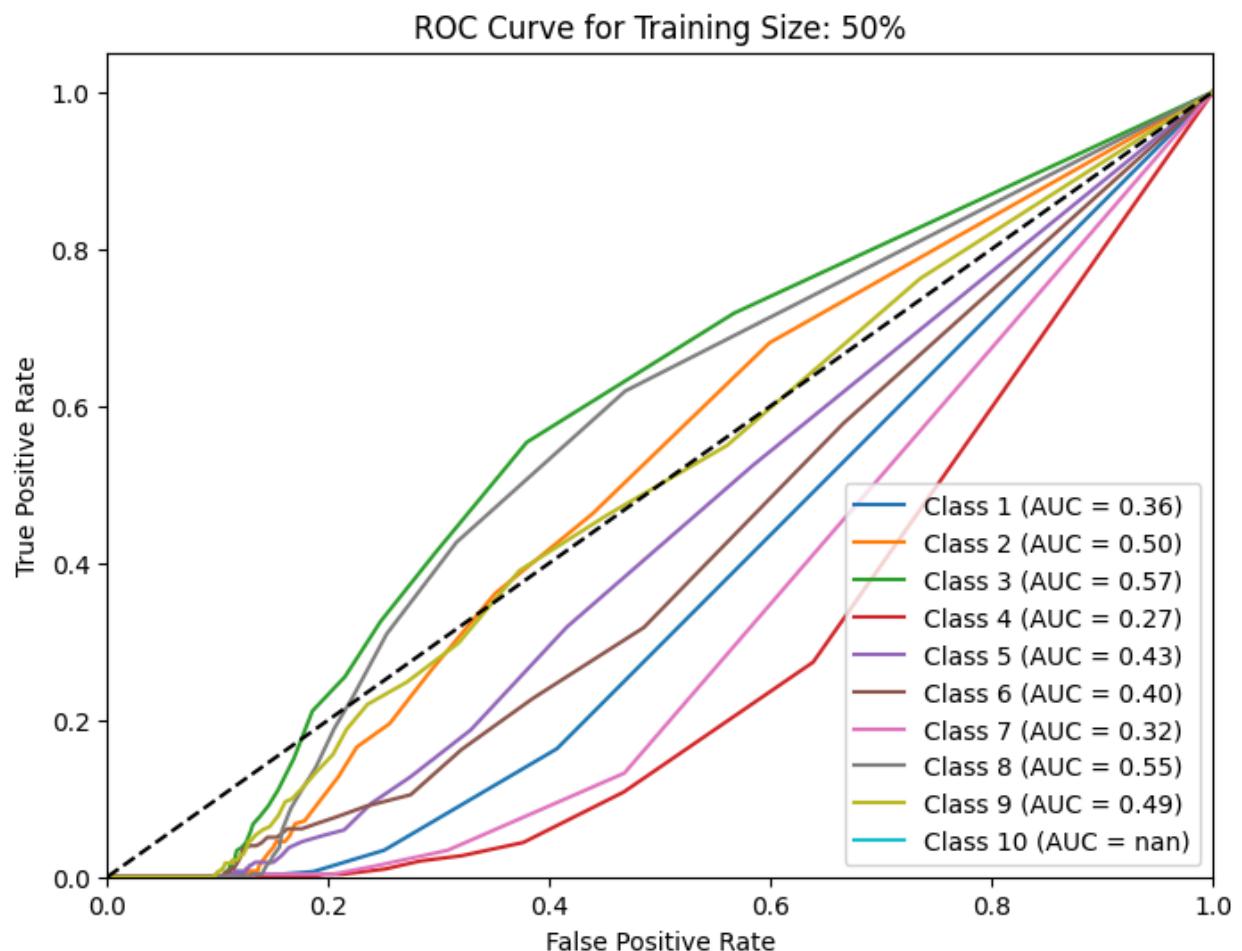
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=i+1)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {i+1} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Plot random chance line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve for Training Size: {training_size}%') # Add title for plot
plt.legend(loc="lower right")
plt.show()

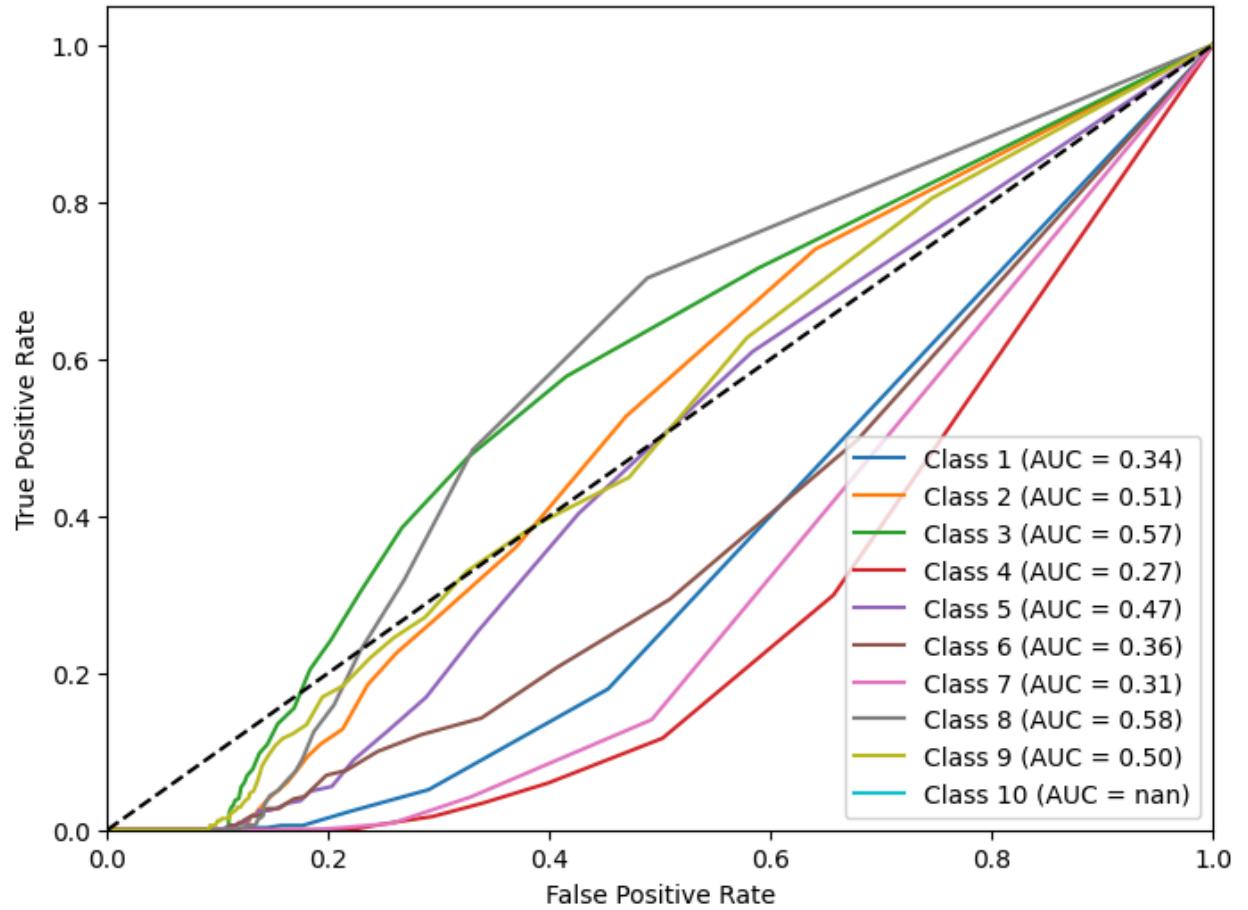
```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/\_ranking.py:1188: UndefinedMetricWarning: No positive samples in y\_true, true positive value should be meaningless  
 warnings.warn(



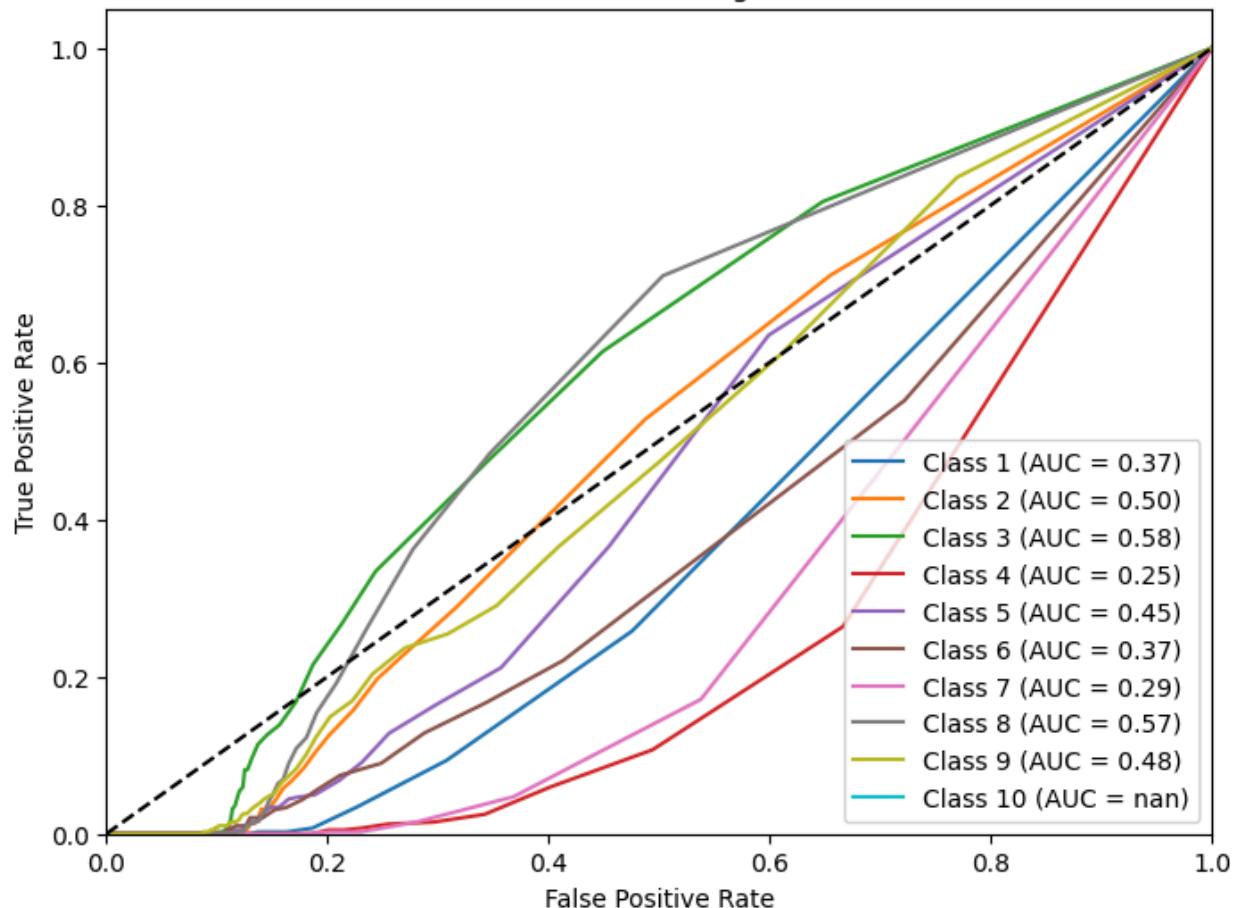
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve for Training Size: 60%

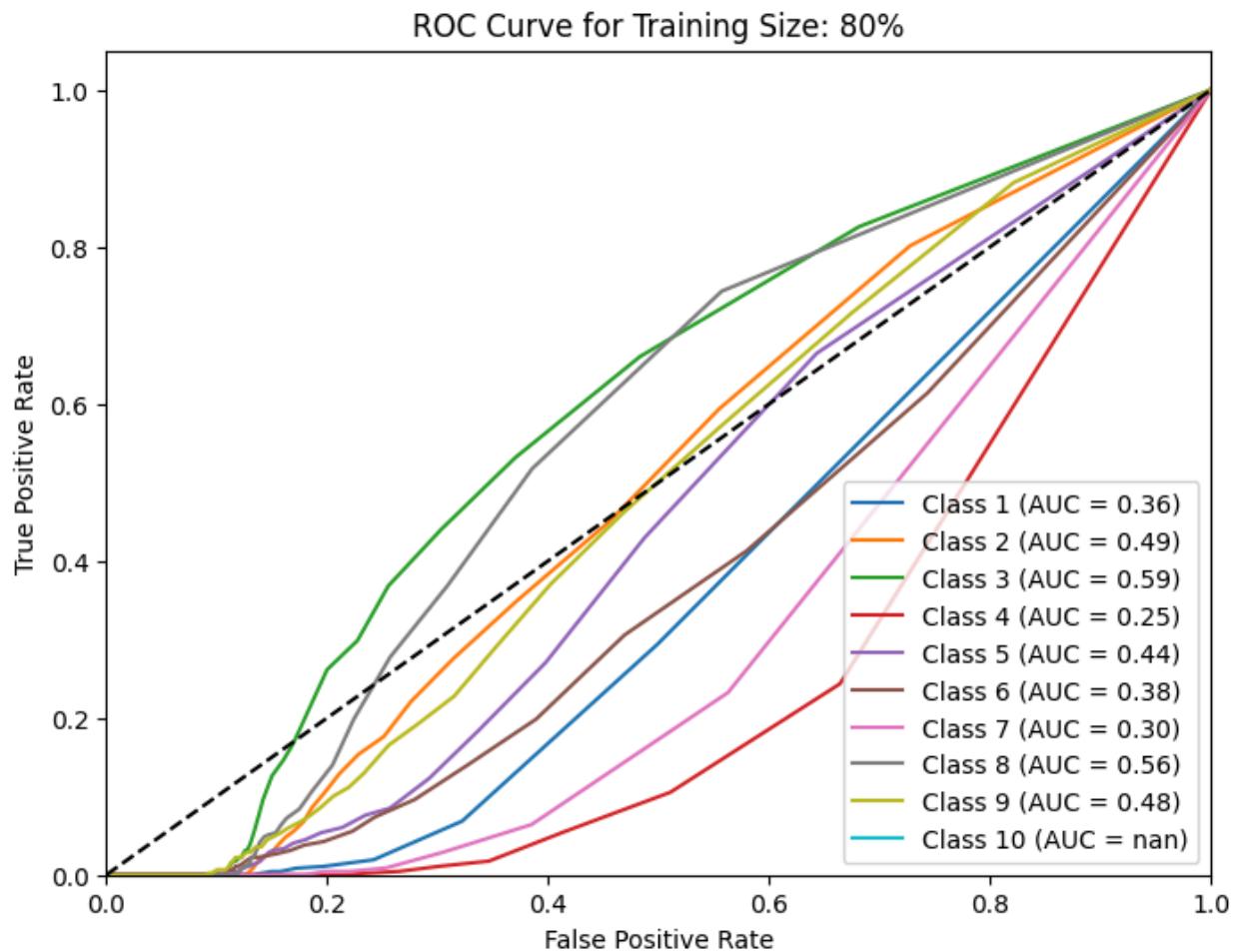


```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```

ROC Curve for Training Size: 70%



```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  warnings.warn(
```



## Random Forest with PCA-Reduced Data

```
In [ ]: # Apply the existing PCA transformation to the scaled data
X_pca_reduced_rf = pca.fit_transform(X_scaled)

print(f"\nOriginal number of features: {X.shape[1]}")
print(f"Reduced number of features after PCA: {X_pca_reduced_rf.shape[1]}")
print("\nData after PCA for Random Forest:")
display(pd.DataFrame(X_pca_reduced_rf).head())
```

Original number of features: 64  
 Reduced number of features after PCA: 50

Data after PCA for Random Forest:

	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	
<b>0</b>	-0.296633	-1.446450	-3.863044	-3.223597	0.773631	0.631849	0.163733	0.474
<b>1</b>	-0.156717	-3.065688	-5.811656	-3.232099	1.064670	0.031353	0.270302	2.091
<b>2</b>	-0.753929	3.261949	0.779656	-1.108976	0.311024	-0.814548	-4.307315	0.841
<b>3</b>	-4.226579	1.900320	-0.441486	1.329548	-0.149220	2.710149	2.128209	0.206
<b>4</b>	0.582983	-3.490539	-1.602212	-1.052587	-1.360737	0.406474	-0.662578	-2.572

5 rows × 50 columns

## Calculate Random Forest performance with PCA-Reduced Data for different test sizes

```
In [ ]: results_rf_pca = []
confusion_matrices_rf_pca = []

for size in training_sizes:
    X_train_pca_rf, X_test_pca_rf, y_train, y_test = train_test_split(X_pca_reduced, random_state=42)

    rf_pca.fit(X_train_pca_rf, y_train.values.ravel())
    y_pred_pca_rf = rf_pca.predict(X_test_pca_rf)

    acc_pca_rf = accuracy_score(y_test, y_pred_pca_rf)
    precision_pca_rf = precision_score(y_test, y_pred_pca_rf, average='weighted')
    recall_pca_rf = recall_score(y_test, y_pred_pca_rf, average='weighted')
    f1_pca_rf = f1_score(y_test, y_pred_pca_rf, average='weighted')

    cm_pca_rf = confusion_matrix(y_test, y_pred_pca_rf)

    results_rf_pca.append({
        "Training size":int (size*100),
        "Accuracy":acc_pca_rf,
        "Precision":precision_pca_rf,
        "Recall":recall_pca_rf,
        "F1-score":f1_pca_rf
    })

    confusion_matrices_rf_pca.append({
        "Training size":int (size*100),
        "Confusion Matrix":cm_pca_rf
    })
```

## Print Random Forest Performance Table and Graphs with PCA-Reduced Data

```
In [ ]: df_rf_pca = pd.DataFrame(results_rf_pca)
display(df_rf_pca)

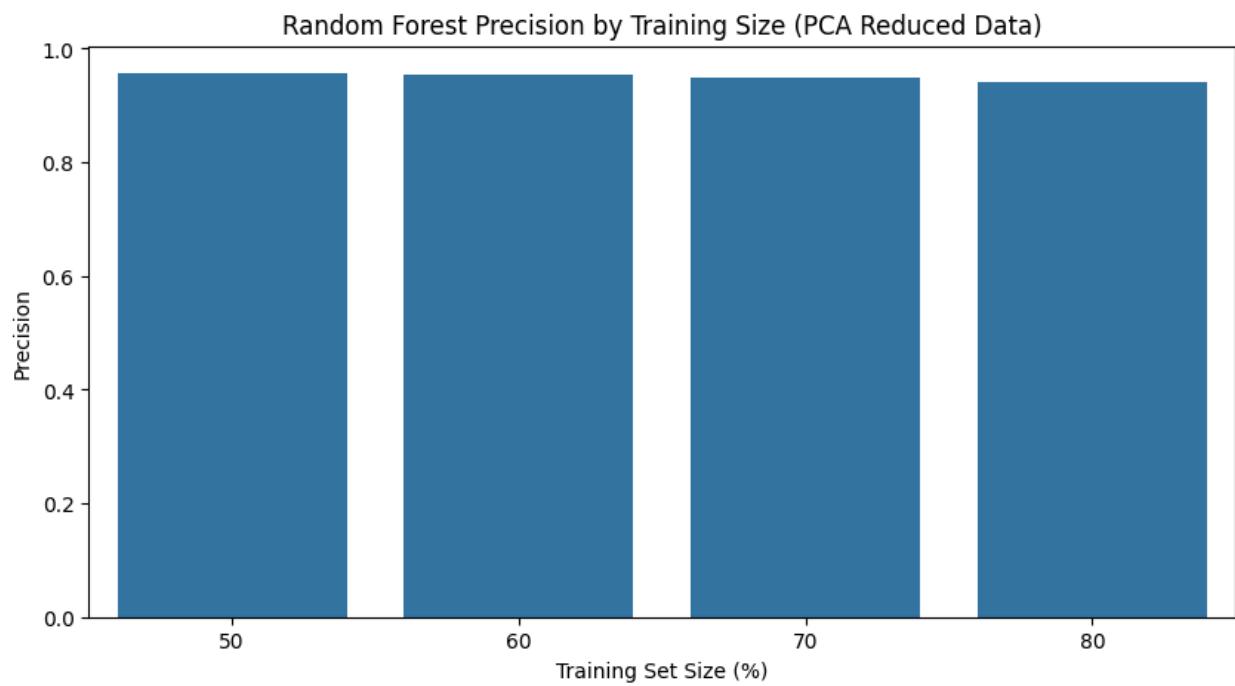
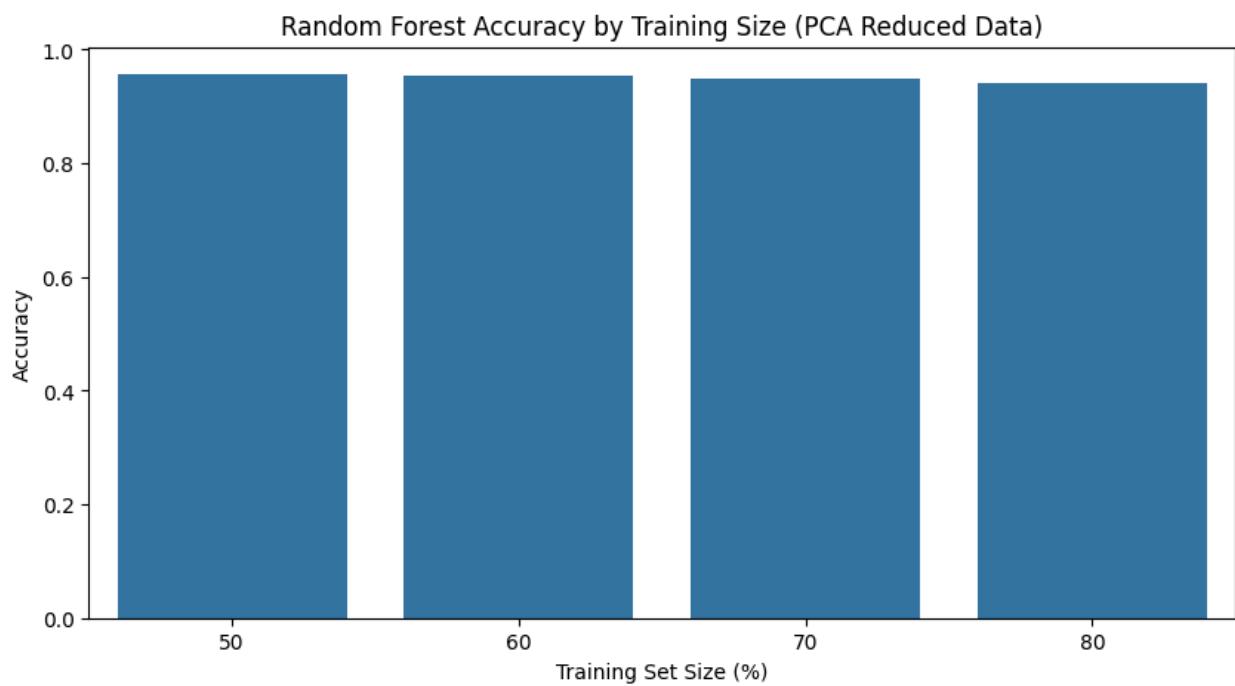
# Plot Accuracy
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Accuracy')
plt.title('Random Forest Accuracy by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.show()

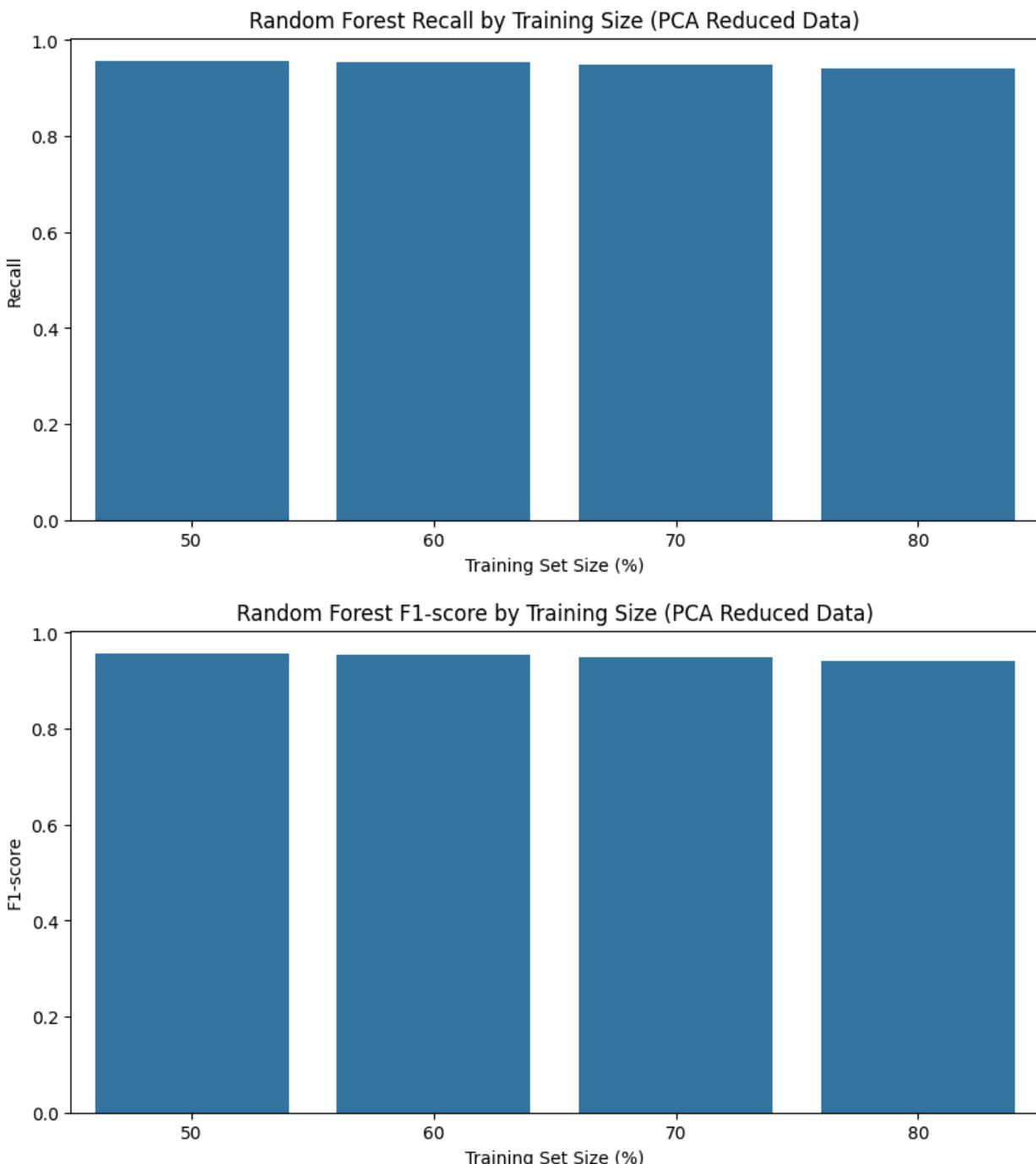
# Plot Precision
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Precision')
plt.title('Random Forest Precision by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Precision')
plt.show()

# Plot Recall
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Recall')
plt.title('Random Forest Recall by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Recall')
plt.show()

# Plot F1-score
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='F1-score')
plt.title('Random Forest F1-score by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('F1-score')
plt.show()
```

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.956584	0.956721	0.956584	0.956593
1	60	0.953440	0.953564	0.953440	0.953439
2	70	0.949670	0.949986	0.949670	0.949693
3	80	0.940169	0.940719	0.940169	0.940303





## Show Confusion Matrices for Random Forest with PCA-Reduced Data

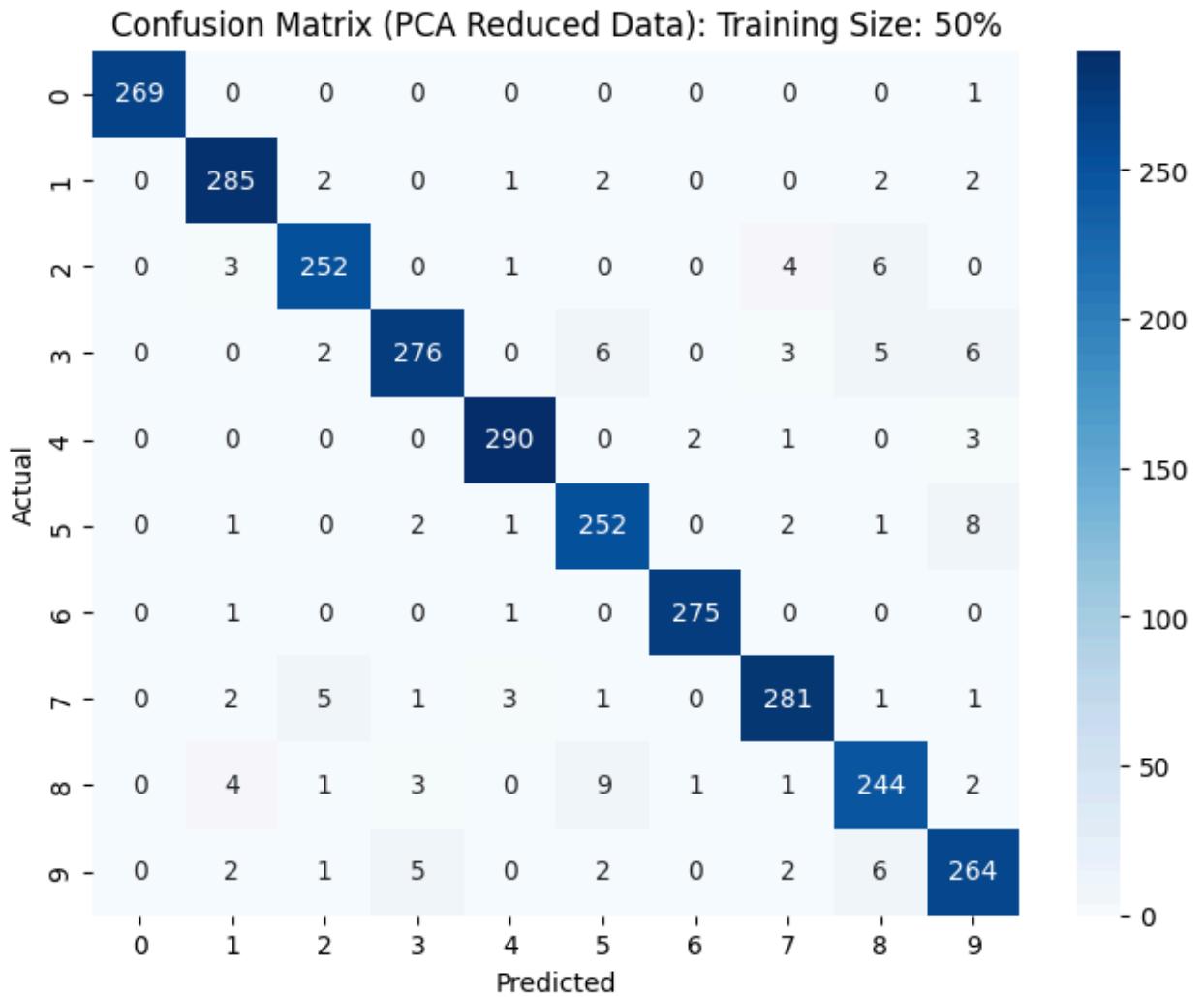
```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns

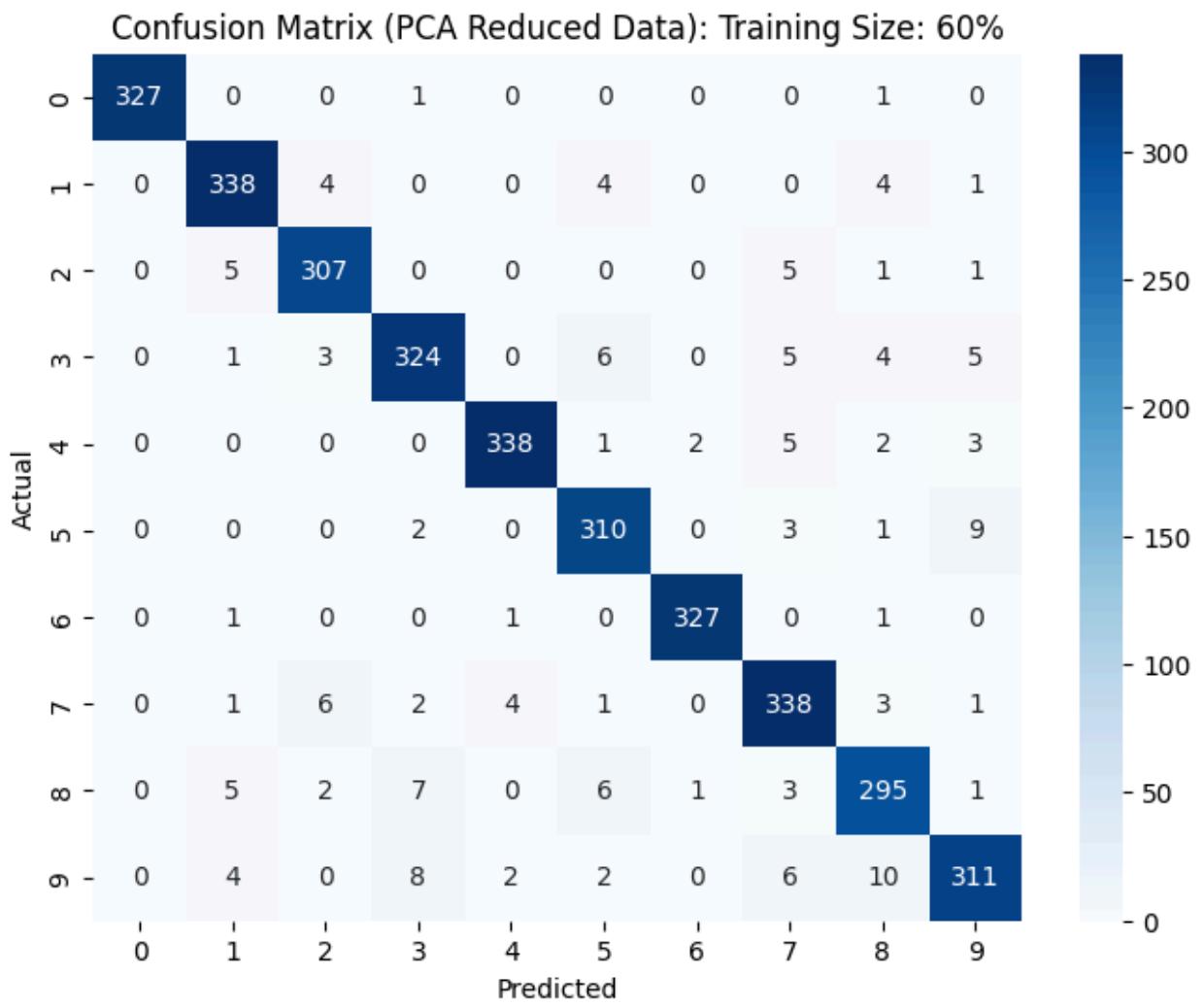
for cm_data in confusion_matrices_rf_pca:
    cm = cm_data["Confusion Matrix"]
    training_size = cm_data["Training size"]
```

```

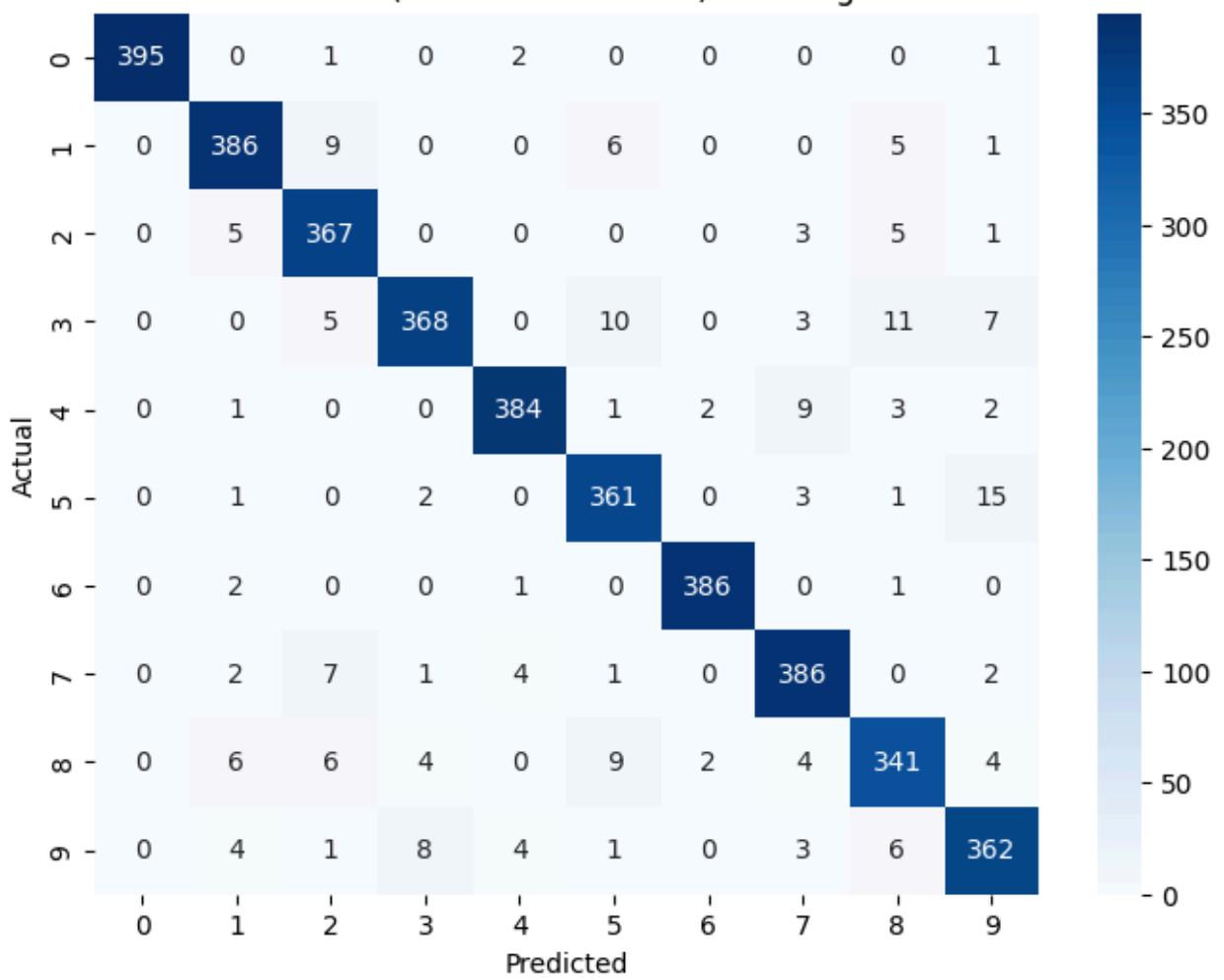
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix (PCA Reduced Data): Training Size: {training_size}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

```

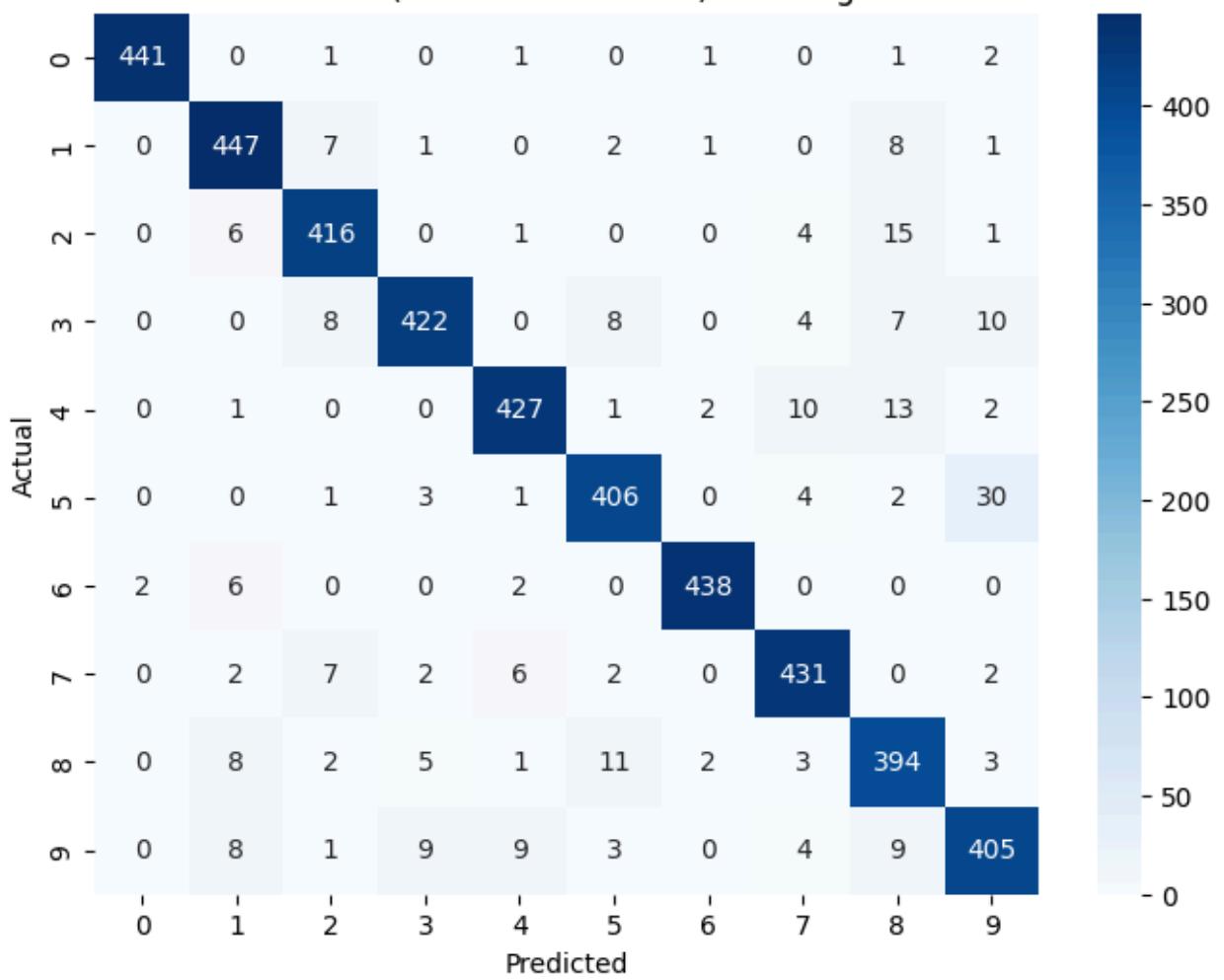




Confusion Matrix (PCA Reduced Data): Training Size: 70%



Confusion Matrix (PCA Reduced Data): Training Size: 80%



# Machine Learning Lab A3

ASIM KUMAR HANSDA

ROLL NO - 002211001136

ASSIGNMENT - 2

---

Github Link: <https://github.com/cryptasim/MACHINE-LEARNING-LAB>

## Machine Learning Model Performance Report

### Wine Dataset Analysis

#### 1. Support Vector Machine (SVM)

This section analyzes the performance of various SVM models on the Wine dataset, exploring the impact of different kernels and training set sizes, and the effect of Principal Component Analysis (PCA) on the model's performance.

#### Impact of Training Size and Kernel:

The performance of the SVM models was evaluated using different kernels (linear, polynomial, gaussian, and sigmoid) and training set sizes (50%, 60%, 70%, and 80%). The key metrics considered were Accuracy, Precision, Recall, and F1-score.

## Analysis of Results:

Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.932584	0.933748	0.932584	0.932554
1	60	0.934579	0.934670	0.934579	0.934515
2	70	0.936000	0.936033	0.936000	0.935762
3	80	0.860140	0.872143	0.860140	0.859931
4	50	0.651685	0.651387	0.651685	0.640804
5	60	0.644860	0.631917	0.644860	0.626031
6	70	0.688000	0.676990	0.688000	0.671169
7	80	0.671329	0.649877	0.671329	0.639471
8	50	0.651685	0.667877	0.651685	0.657626
9	60	0.663551	0.665705	0.663551	0.661062
10	70	0.688000	0.693446	0.688000	0.687787
11	80	0.678322	0.656929	0.678322	0.644090
12	50	0.382022	0.145941	0.382022	0.211199
13	60	0.401869	0.161499	0.401869	0.230405
14	70	0.400000	0.160000	0.400000	0.228571
15	80	0.398601	0.158883	0.398601	0.227203

Based on the results, the linear kernel generally performed the best across all training sizes on the original data. The polynomial and gaussian kernels showed varying performance, while the sigmoid kernel consistently had the lowest scores. As the training size increased, the performance metrics generally improved for all kernels, indicating that more data leads to better model generalization.

## Effect of PCA Feature Reduction:

Principal Component Analysis (PCA) was applied to the Wine dataset to reduce the number of features while retaining a significant portion of the variance. The SVM models were then trained and evaluated on this PCA-reduced data.

	Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.988764	0.989085	0.988764	0.988715	linear
1	60	0.953271	0.956072	0.953271	0.953166	linear
2	70	0.960000	0.962150	0.960000	0.960011	linear
3	80	0.951049	0.954736	0.951049	0.950406	linear
4	50	0.876404	0.906617	0.876404	0.874846	polynomial
5	60	0.869159	0.892997	0.869159	0.869891	polynomial
6	70	0.880000	0.900444	0.880000	0.879207	polynomial
7	80	0.825175	0.870248	0.825175	0.825552	polynomial
8	50	0.977528	0.978777	0.977528	0.977632	gaussian
9	60	0.981308	0.982139	0.981308	0.981370	gaussian
10	70	0.976000	0.976806	0.976000	0.976083	gaussian
11	80	0.965035	0.965834	0.965035	0.965202	gaussian
12	50	0.988764	0.989095	0.988764	0.988764	sigmoid
13	60	0.981308	0.981308	0.981308	0.981308	sigmoid
14	70	0.960000	0.961607	0.960000	0.959935	sigmoid
15	80	0.965035	0.967366	0.965035	0.964838	sigmoid

Comparing the performance metrics before and after PCA, it can be observed that PCA had a positive impact on the performance of the SVM models, particularly for the linear and gaussian kernels. The accuracy and other metrics generally improved with PCA, suggesting

that reducing the dimensionality of the data helped the SVM models to better capture the underlying patterns.

## 2. Multilayer Perceptron (MLP)

This section analyzes the performance of the MLP model on the Wine dataset, examining the influence of training set size, the training loss, and the effect of PCA.

### Impact of Training Size:

The MLP model was evaluated with different training set sizes (50%, 60%, 70%, and 80%).

### Analysis of Results:

Training size	Accuracy	Precision	Recall	F1-score
0	50	0.584270	0.525580	0.584270
1	60	0.943925	0.947556	0.943925
2	70	0.936000	0.937158	0.936000
3	80	0.629371	0.559791	0.629371

The MLP model showed varied performance across different training sizes on the original data. There isn't a clear trend of performance consistently improving with increased training size, suggesting

that the default hyperparameters might not be optimally tuned for this dataset.

### Effect of PCA Feature Reduction:

The MLP model was also trained and evaluated on the PCA-reduced Wine dataset.

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.977528	0.978777	0.977528
<b>1</b>	60	0.934579	0.934670	0.934579
<b>2</b>	70	0.952000	0.952117	0.952000
<b>3</b>	80	0.951049	0.952787	0.951049

Training the MLP model on PCA-reduced data resulted in a significant improvement in performance compared to the original data. The accuracy and other metrics are notably higher, indicating that reducing the dimensionality and potentially the noise in the data benefited the MLP model's ability to learn the classification task.

## **3. Random Forest**

This section presents the analysis of the Random Forest model's performance on the Wine dataset, considering the effect of training size and PCA.

### Impact of Training Size:

The Random Forest model was evaluated using different training set sizes (50%, 60%, 70%, and 80%).

### Analysis of Results:

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.977528	0.978777	0.977528
<b>1</b>	60	0.934579	0.934670	0.934579
<b>2</b>	70	0.952000	0.952117	0.952000
<b>3</b>	80	0.951049	0.952787	0.951049

The Random Forest model demonstrated high accuracy and other performance metrics on the original Wine dataset across all training sizes. This suggests that Random Forest is well-suited for this dataset and is less sensitive to the specific training size within the tested range.

### Effect of PCA Feature Reduction:

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.966292	0.966583	0.966292
<b>1</b>	60	0.962617	0.963637	0.962617
<b>2</b>	70	0.952000	0.951878	0.952000
<b>3</b>	80	0.944056	0.943992	0.944056

While Random Forest performed well on the original data, applying PCA resulted in a slight decrease in performance metrics. This indicates that for Random Forest on this dataset, the original features provided more discriminatory information than the PCA-reduced features.

# HandWritten Digit Dataset Analysis

## 1. Support Vector Machine (SVM)

This section analyzes the performance of various SVM models on the HandWritten Digit dataset, including the impact of different kernels and training set sizes, and the effect of Principal Component Analysis (PCA).

### Impact of Training Size and Kernel:

The SVM models were evaluated with different kernels (linear, polynomial, gaussian, and sigmoid) and training set sizes (50%, 60%, 70%, and 80%) on the Digit dataset.

### Analysis of Results:

Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.972954	0.973049	0.972954	0.972962
1	60	0.974792	0.974869	0.974792	0.974799
2	70	0.971530	0.971745	0.971530	0.971538
3	80	0.968194	0.968383	0.968194	0.968173
4	50	0.985053	0.985087	0.985053	0.985055
5	60	0.986655	0.986672	0.986655	0.986647
6	70	0.985765	0.985794	0.985765	0.985765
7	80	0.979760	0.979809	0.979760	0.979765
8	50	0.986121	0.986187	0.986121	0.986130
9	60	0.983096	0.983148	0.983096	0.983091
10	70	0.982461	0.982488	0.982461	0.982443
11	80	0.974867	0.975072	0.974867	0.974875
12	50	0.841993	0.848099	0.841993	0.843197
13	60	0.854686	0.859502	0.854686	0.855680
14	70	0.872140	0.875307	0.872140	0.873104
15	80	0.886566	0.888754	0.886566	0.887225

For the Digit dataset, the polynomial and gaussian kernels achieved the highest performance metrics, consistently outperforming the linear and sigmoid kernels. The performance generally remained high across different training sizes for the better-performing kernels.

### Effect of PCA Feature Reduction:

PCA was applied to the Digit dataset to reduce its high dimensionality, and the SVM models were evaluated on the PCA-reduced data.

Training size	Accuracy	Precision	Recall	F1-score	Kernel
0	50	0.970819	0.970937	0.970819	0.970850
1	60	0.972716	0.972695	0.972716	0.972687
2	70	0.968734	0.968882	0.968734	0.968751
3	80	0.966192	0.966314	0.966192	0.966197
4	50	0.964413	0.968400	0.964413	0.965236
5	60	0.962930	0.967787	0.962930	0.963943
6	70	0.956279	0.963473	0.956279	0.957801
7	80	0.927491	0.948847	0.927491	0.932636
8	50	0.980427	0.980517	0.980427	0.980437
9	60	0.977165	0.977299	0.977165	0.977172
10	70	0.975089	0.975237	0.975089	0.975076
11	80	0.968416	0.968740	0.968416	0.968425
12	50	0.927402	0.927971	0.927402	0.927522
13	60	0.933571	0.933986	0.933571	0.933645
14	70	0.933910	0.934523	0.933910	0.934033
15	80	0.940169	0.940999	0.940169	0.940326

Applying PCA to the Digit dataset had a positive impact on the performance of the SVM models, particularly for the gaussian kernel. The accuracy and other metrics improved

with PCA, suggesting that dimensionality reduction helped in handling the complexity of the image data.

## 2. Multilayer Perceptron (MLP)

This section analyzes the performance of the MLP model on the HandWritten Digit dataset, examining the influence of training set size, the training loss, and the effect of PCA.

### Impact of Training Size:

The MLP model was evaluated with different training set sizes (50%, 60%, 70%, and 80%) on the Digit dataset.

### Analysis of Results:

Training size	Accuracy	Precision	Recall	F1-score
0	50	0.975089	0.975259	0.975089
1	60	0.969158	0.969391	0.969158
2	70	0.961617	0.961873	0.961617
3	80	0.958852	0.959329	0.958852

The MLP model performed well on the original Digit dataset, with high accuracy and other metrics across different training sizes. The performance remained relatively consistent, indicating the model's robustness.

### Effect of PCA Feature Reduction:

The MLP model was also trained and evaluated on the PCA-reduced Digit dataset.

Training size	Accuracy	Precision	Recall	F1-score
0	50	0.973665	0.973913	0.973665
1	60	0.975979	0.976062	0.975979
2	70	0.966446	0.966648	0.966446
3	80	0.965970	0.966241	0.965970

Similar to the Wine dataset, applying PCA to the Digit dataset had a positive impact on the MLP model's performance. The accuracy and other metrics improved, suggesting that PCA helped the MLP model to handle the high dimensionality of the image data more effectively.

### 3. Random Forest

This section presents the analysis of the Random Forest model's performance on the Hand Written Digit dataset, considering the effect of training size and PCA.

#### Impact of Training Size:

The Random Forest model was evaluated using different training set sizes (50%, 60%, 70%, and 80%) on the Digit dataset.

## Analysis of Results:

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.974733	0.975024	0.974733
<b>1</b>	60	0.972716	0.972770	0.972716
<b>2</b>	70	0.968480	0.968704	0.968480
<b>3</b>	80	0.956628	0.957079	0.956628

The Random Forest model achieved high performance metrics on the original Digit dataset. The performance was consistent across different training sizes.

## Effect of PCA Feature Reduction:

The Random Forest model was also applied to the PCA-reduced Digit dataset.

<b>Training size</b>	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>0</b>	50	0.956584	0.956721	0.956584
<b>1</b>	60	0.953440	0.953564	0.953440
<b>2</b>	70	0.949670	0.949986	0.949670
<b>3</b>	80	0.940169	0.940719	0.940169

Applying PCA to the Digit dataset resulted in a slight decrease in the performance of the Random Forest model. Similar to the Wine dataset, for Random Forest on this dataset, the original features provided more discriminatory information than the PCA-reduced features.