NAME : DHANANJOY SHAW
SECTION : IT A2
ROLL NUMBER : 002211001086
SUBJECT : ML LAB
GITHUB:
https://github.com/DhananjoyShaw/ML_LAB

GOOGLE COLLAB

Installing Libraries

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

In [ ]:
```
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.12/dist-pack
ages (0.0.7)
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/python
3.12/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-p
ackages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
t-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packa
ges (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
```

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

178 rows × 13 columns

In [ ]:
```
y
```

Out[ ]:

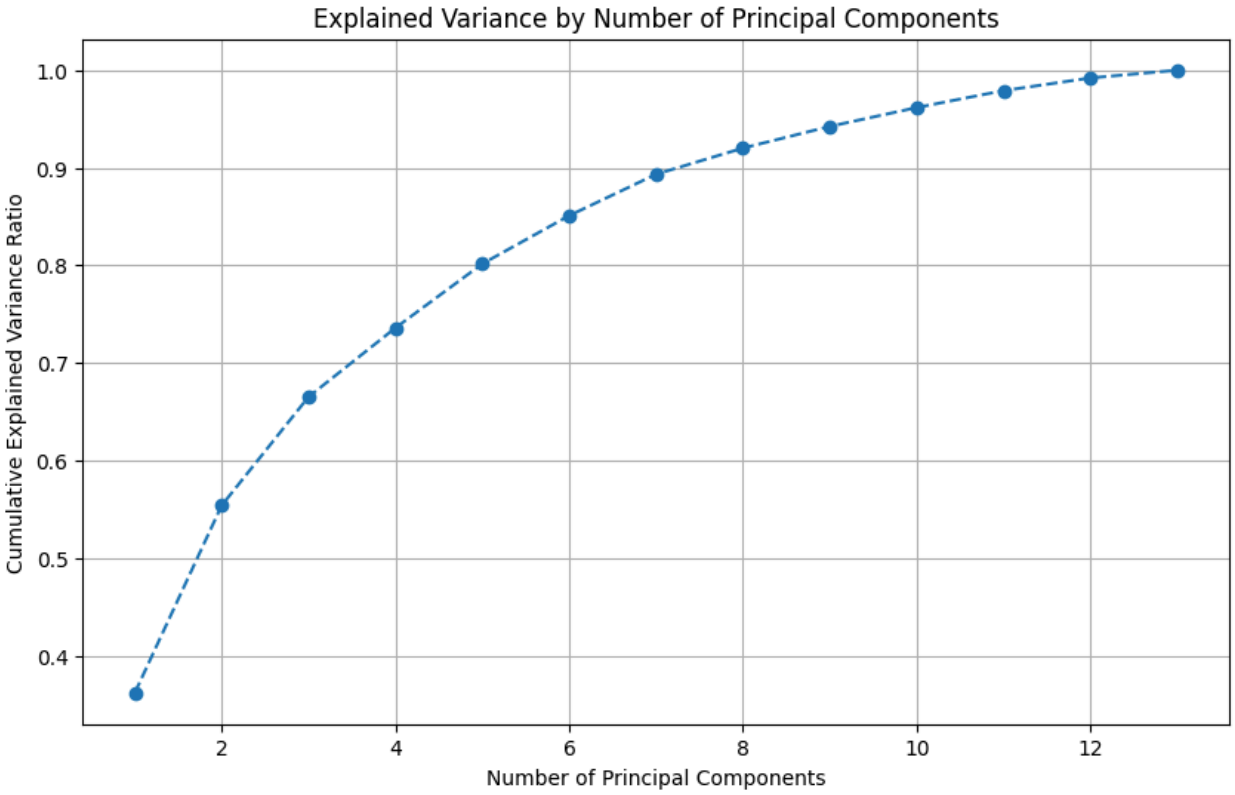
	class
0	1
1	1
2	1
3	1
4	1
...	...
173	3
174	3
175	3
176	3
177	3

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 th
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:
```

	0	1	2	3	4	5	6	
0	3.316751	1.443463	-0.165739	-0.215631	0.693043	0.223880	0.596427	-0.065
1	2.209465	-0.333393	-2.026457	-0.291358	-0.257655	0.927120	0.053776	-1.024
2	2.516740	1.031151	0.982819	0.724902	-0.251033	-0.549276	0.424205	0.344
3	3.757066	2.756372	-0.176192	0.567983	-0.311842	-0.114431	-0.383337	-0.643
4	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_m
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, r
   svc = SVC(kernel=kernel, probability=True, random_state=42) # Add probabili
   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_divisi
   recall = recall_score(y_test, y_pred, average='weighted') # Calculate weigh
   f1 = f1_score(y_test, y_pred, average='weighted') # Calculate weighted F1-s
```

```
    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 Evalution Table and Graphs

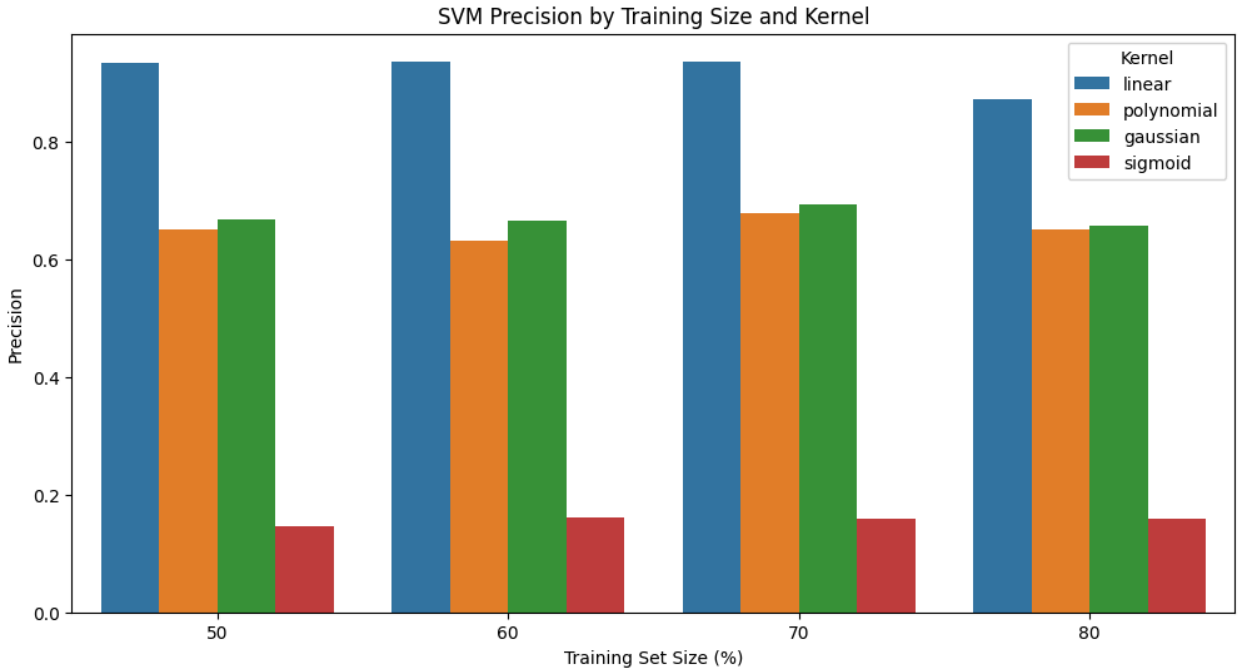
In [ ]:

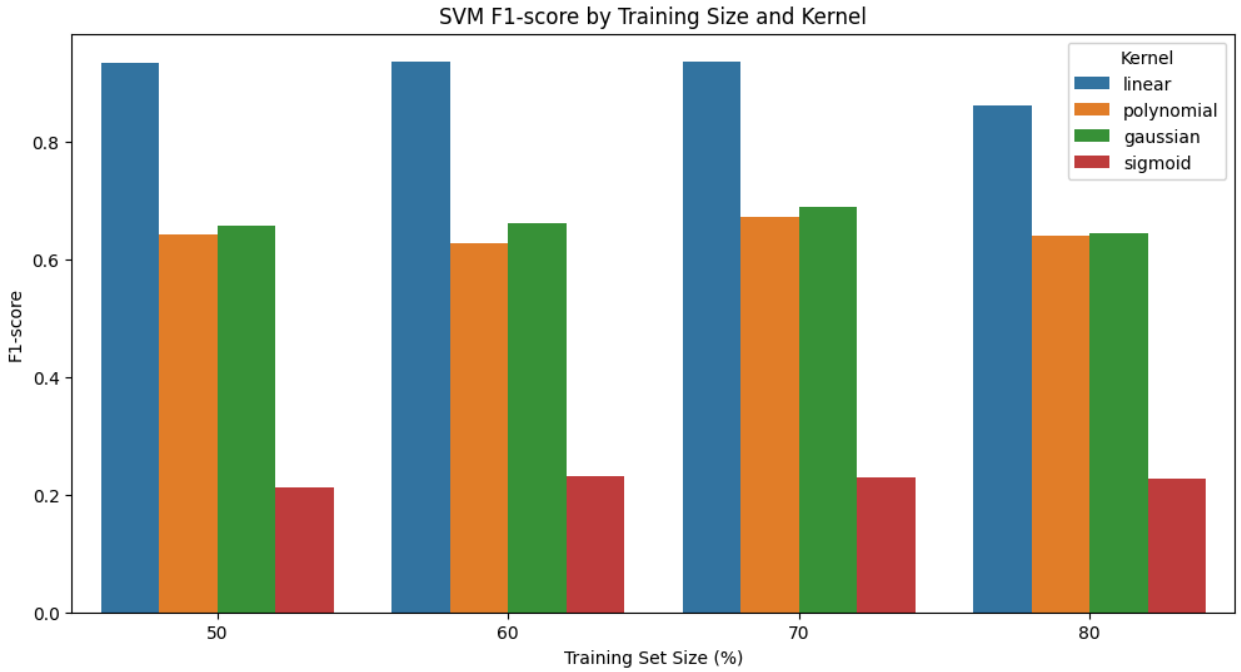
```
df = pd.DataFrame(results)
display(df) # Display the DataFrame as a table

# 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
Kernel
linear
polynomial
gaussian
sigmoid
Accuracy
0.0
0.2
0.4
0.6
0.8
50
60
70
80
Training Set Size (%)


SVM Precision by Training Size and Kernel
Kernel
linear
polynomial
gaussian
sigmoid
Precision
0.0
0.2
0.4
0.6
0.8
50
60
70
80
Training Set Size (%)

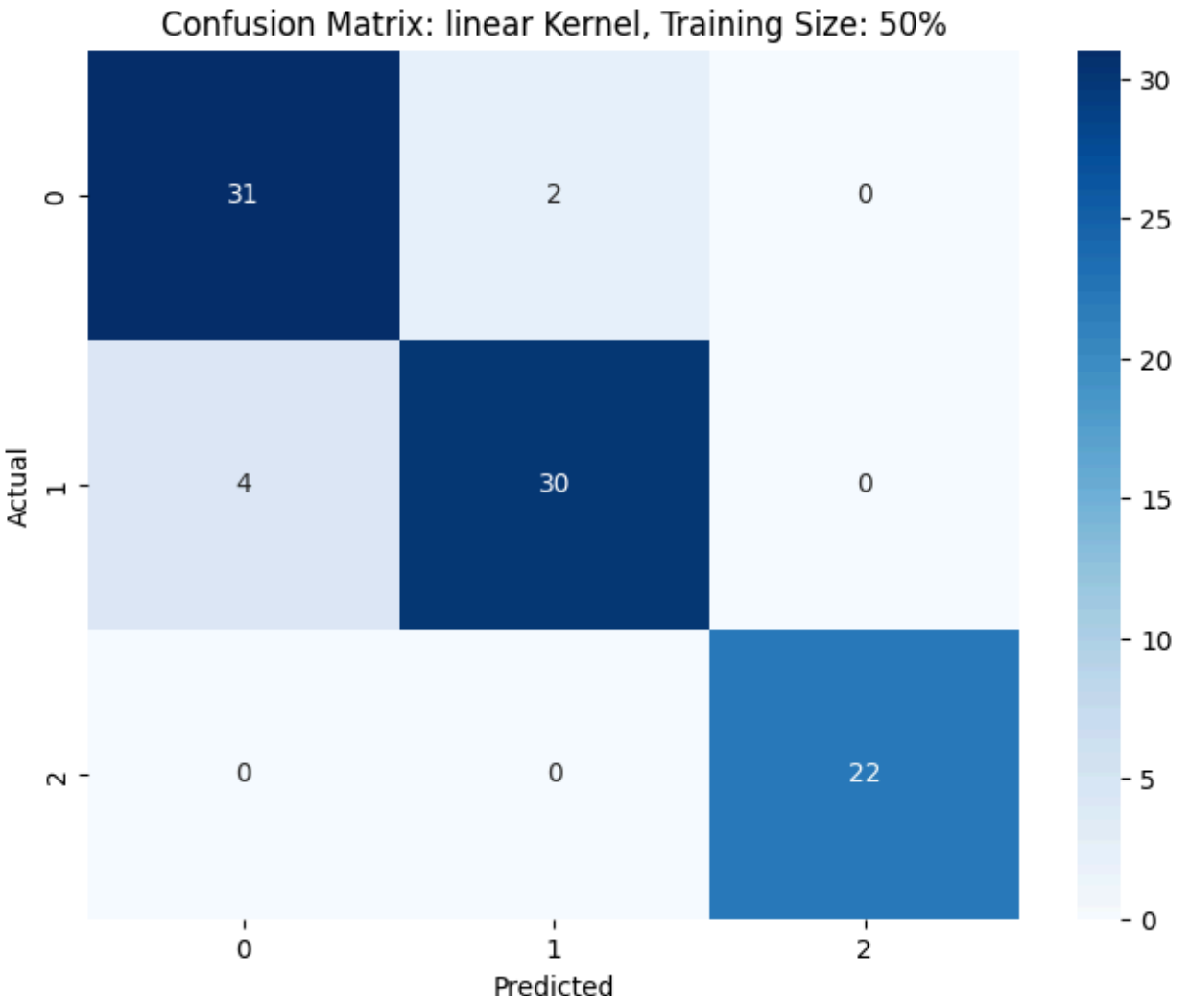


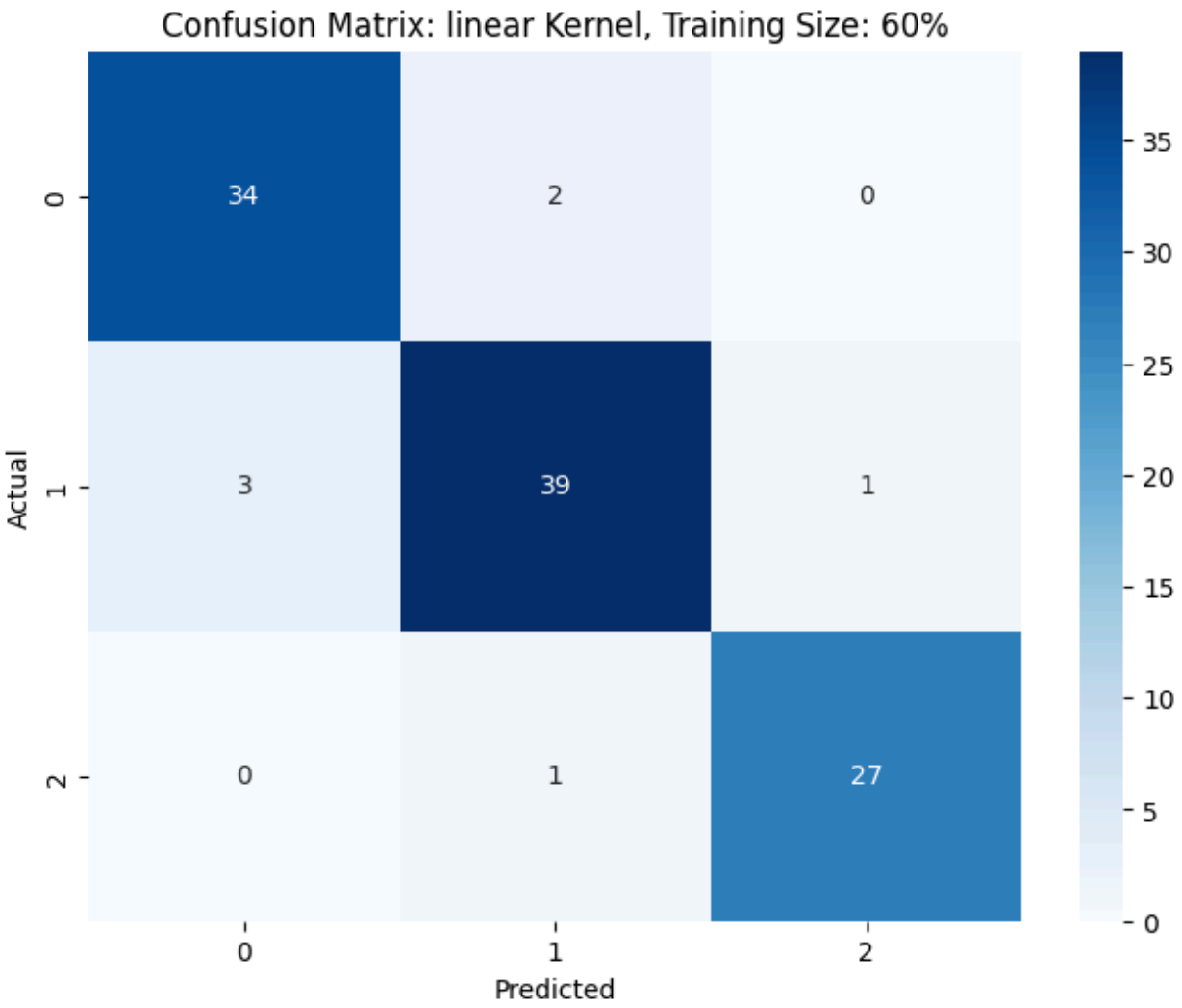
Show Confusion Matrices

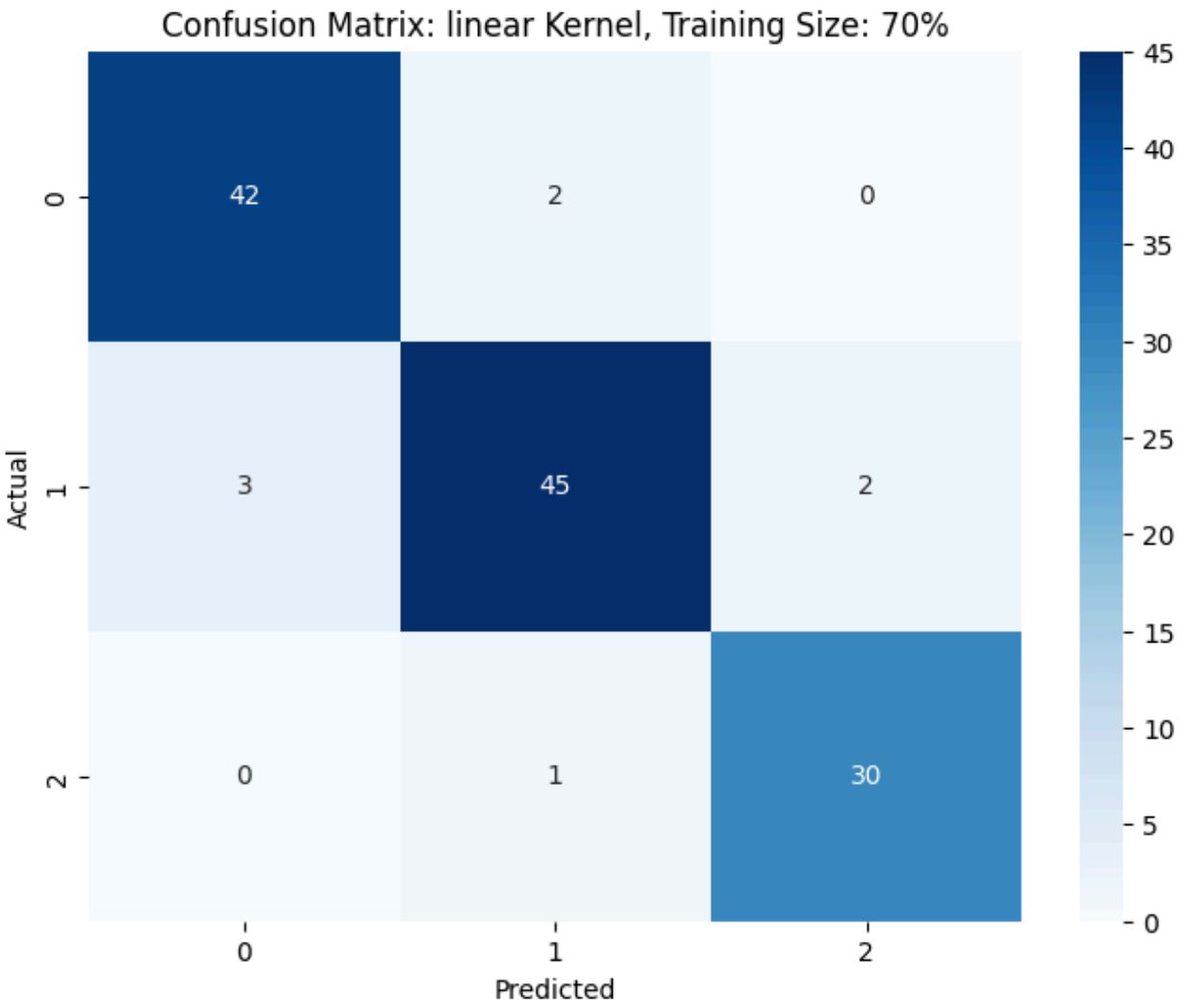
In [ ]:

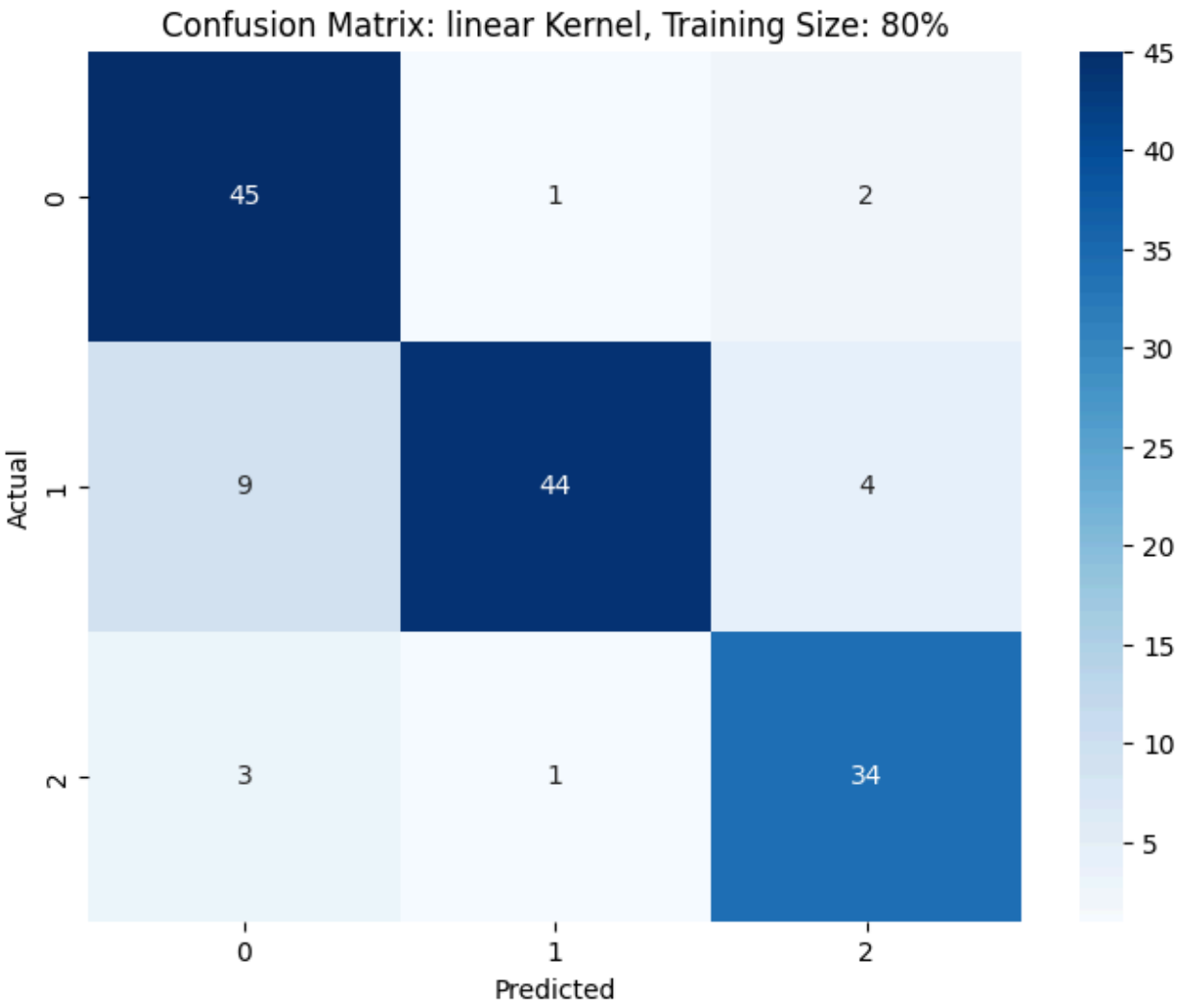
```
import matplotlib.pyplot as plt
import seaborn as sns

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))
```

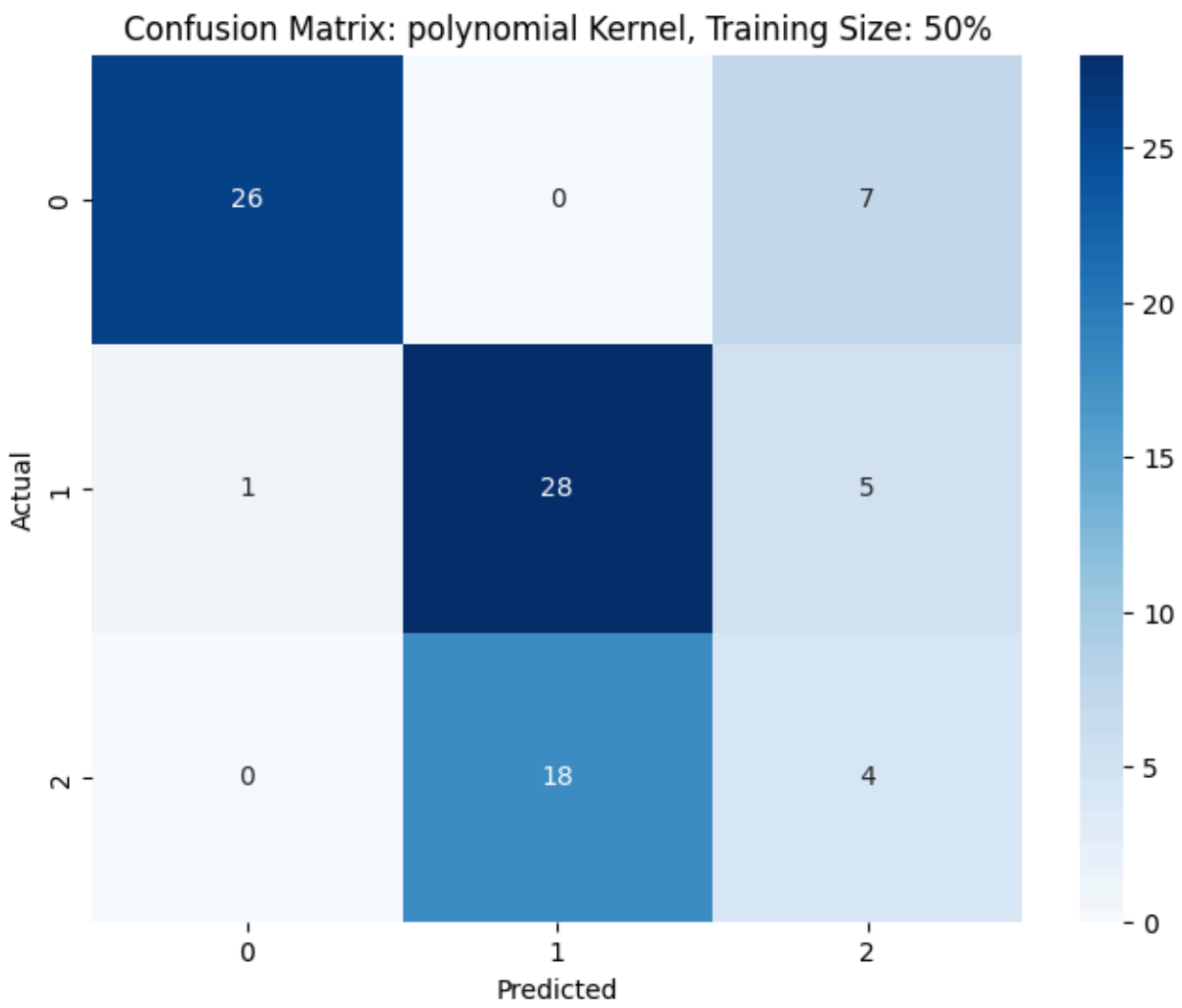
```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix: {kernel_name} Kernel, Training Size: {traini
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

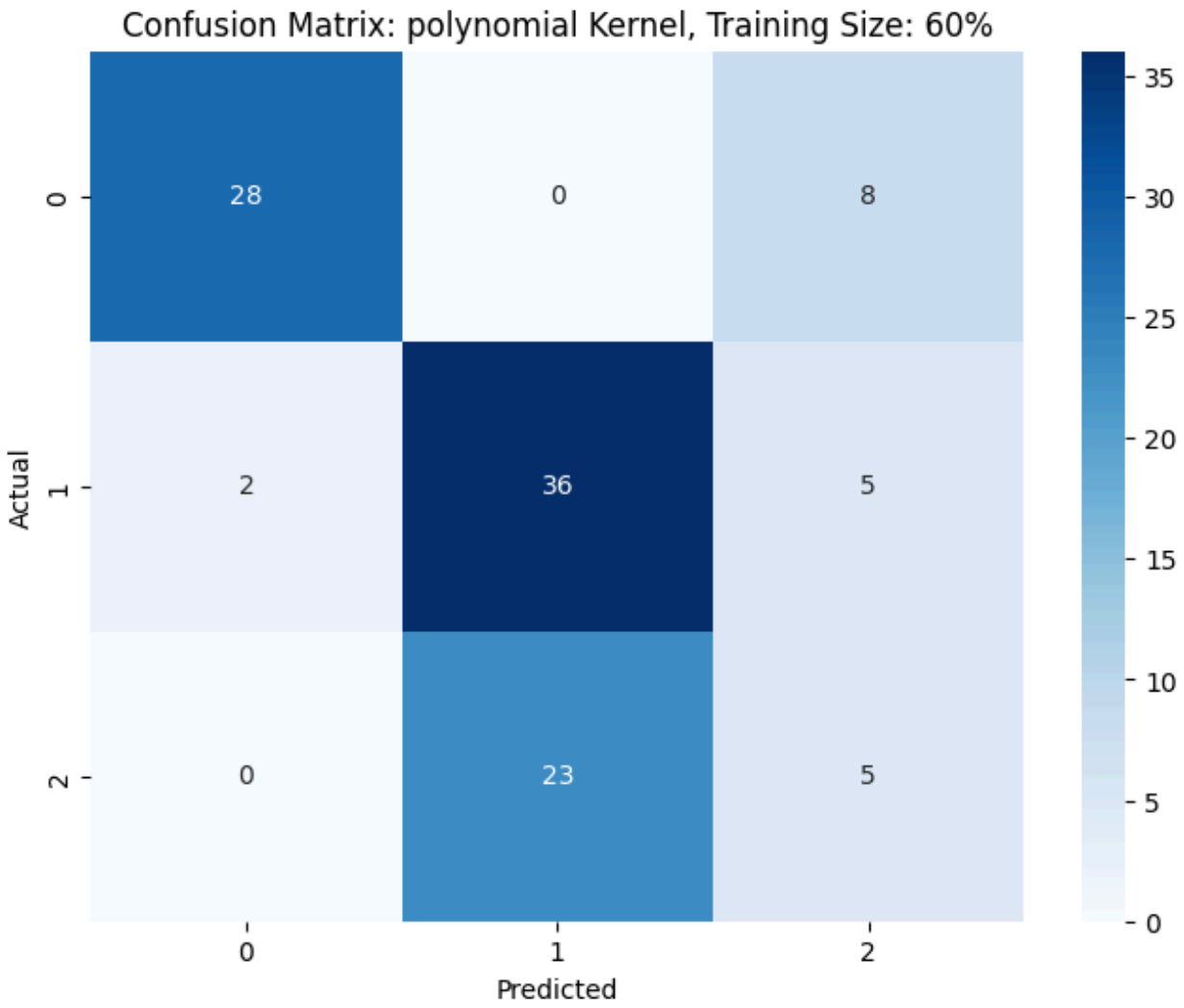


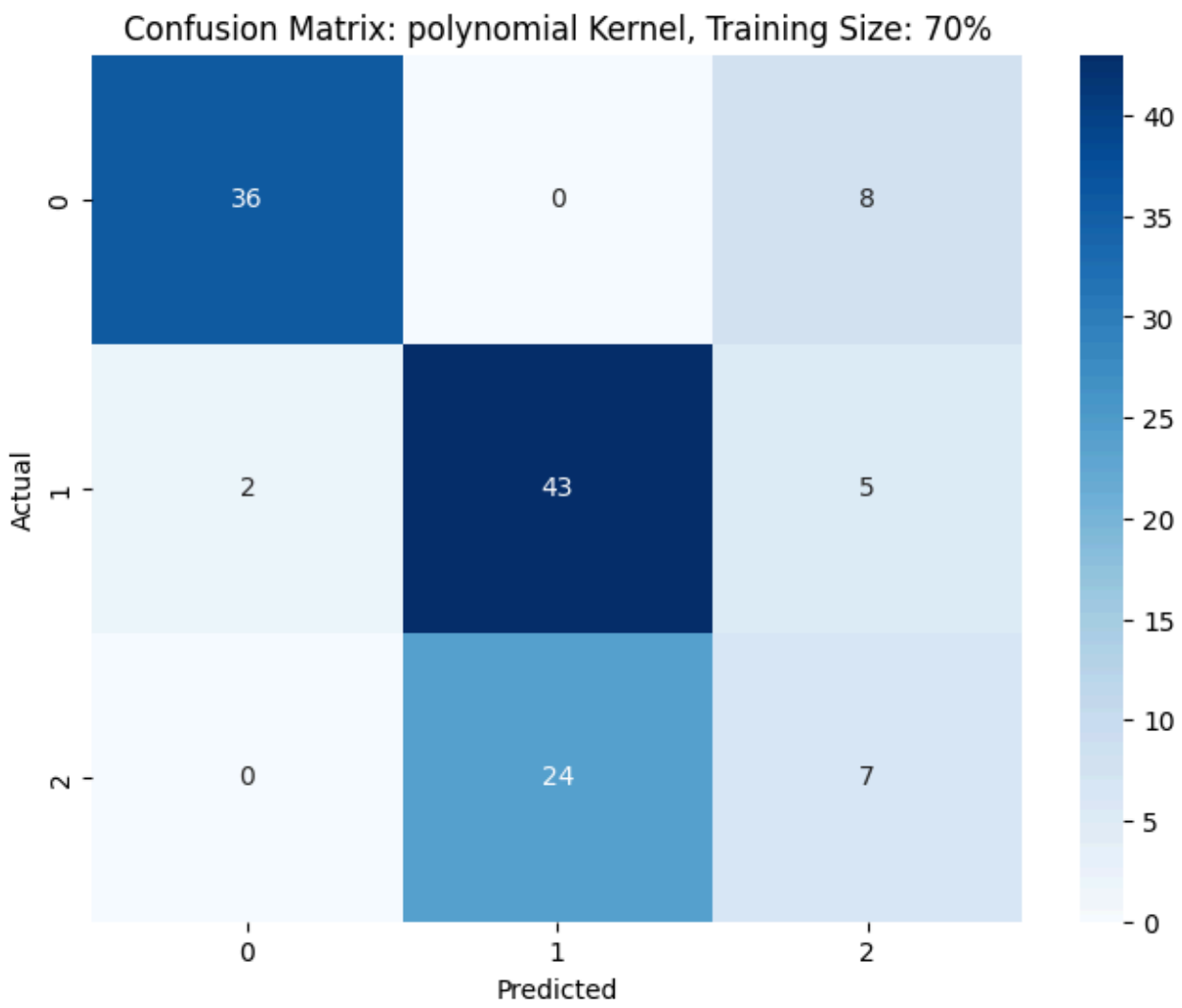

Confusion Matrix: linear Kernel, Training Size: 60%
Actual
Predicted
0
1
2
34
2
0
3
39
1
0
1
27
35
30
25
20
15
10
5
0

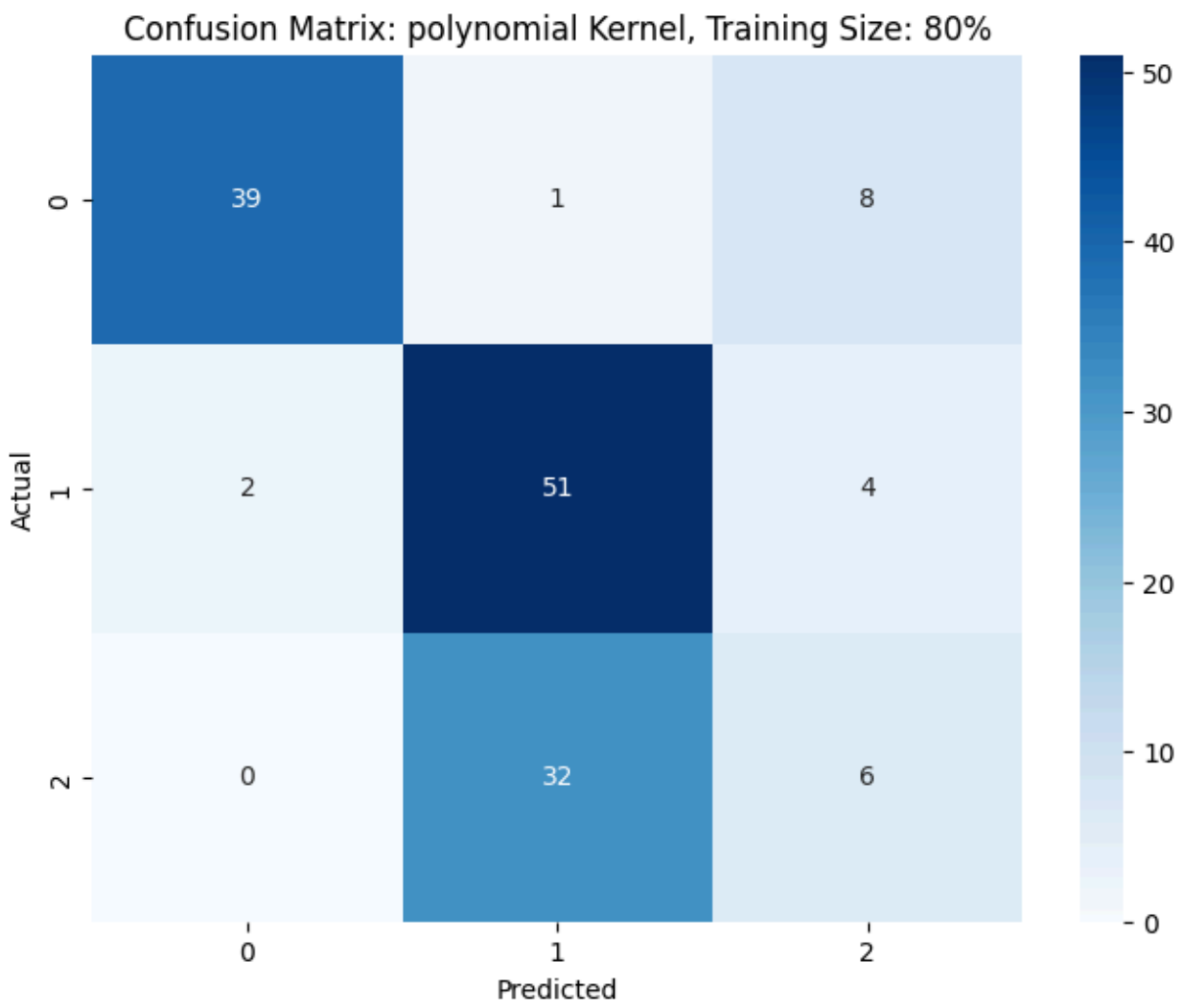

Confusion Matrix: linear Kernel, Training Size: 70%
Actual
Predicted
0
1
2
42
2
0
3
45
2
0
1
30
45
40
35
30
25
20
15
10
5
0

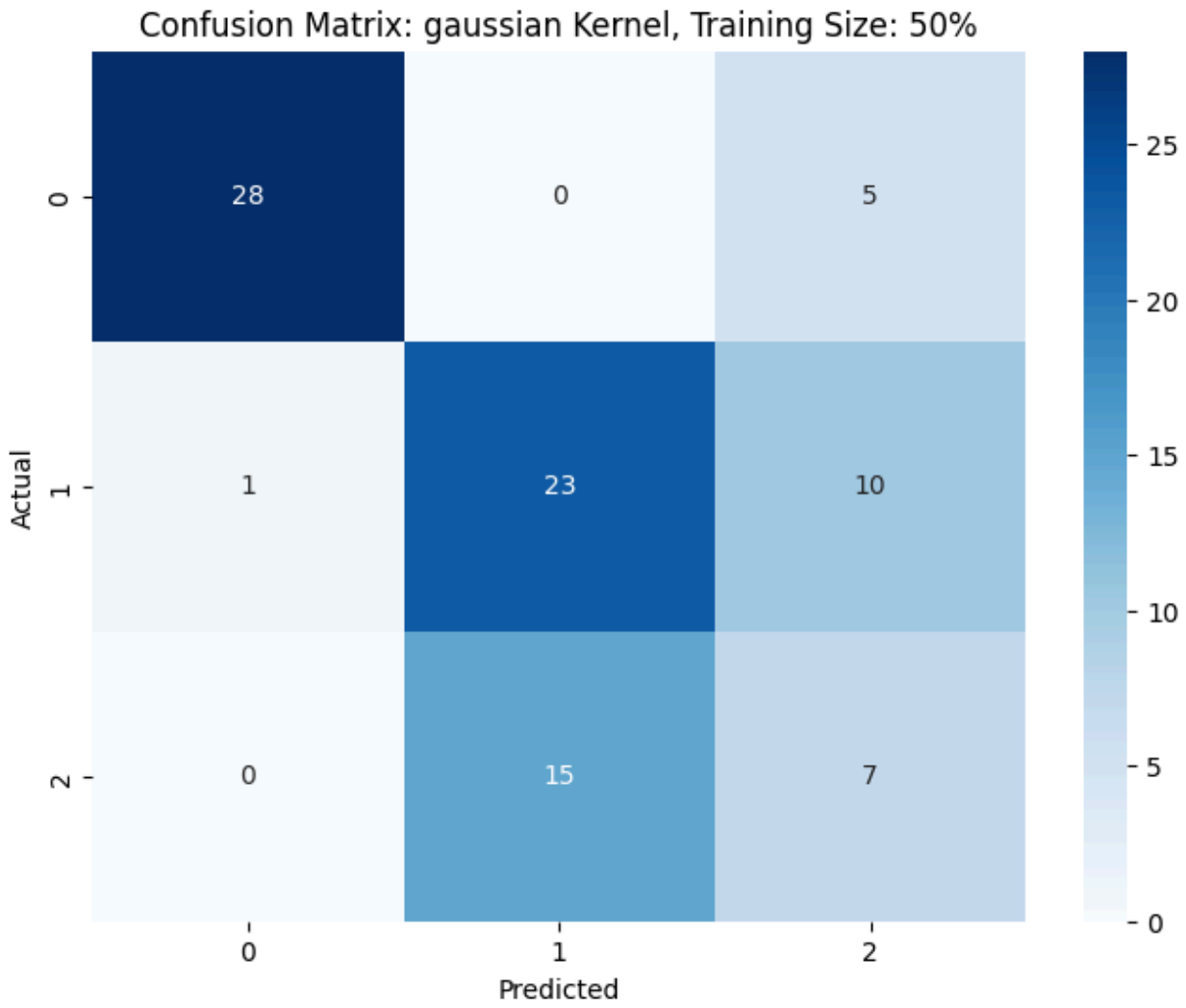

Confusion Matrix: linear Kernel, Training Size: 80%
Actual
Predicted
0
1
2
45
1
2
9
44
4
3
1
34
45
40
35
30
25
20
15
10
5

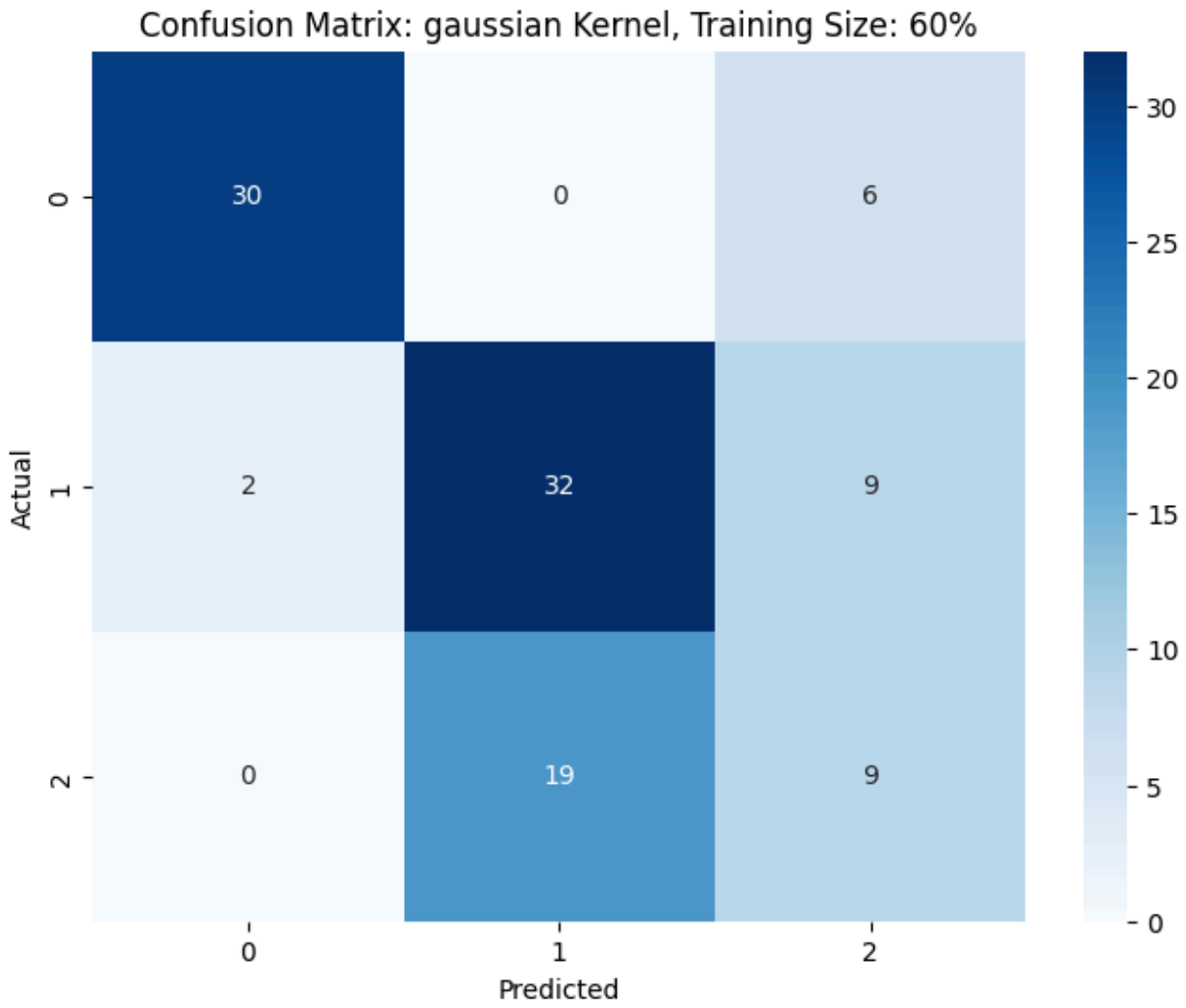

Confusion Matrix: polynomial Kernel, Training Size: 50%
26
0
7
1
28
5
0
18
4
Actual
0
1
2
Predicted
25
20
15
10
5
0

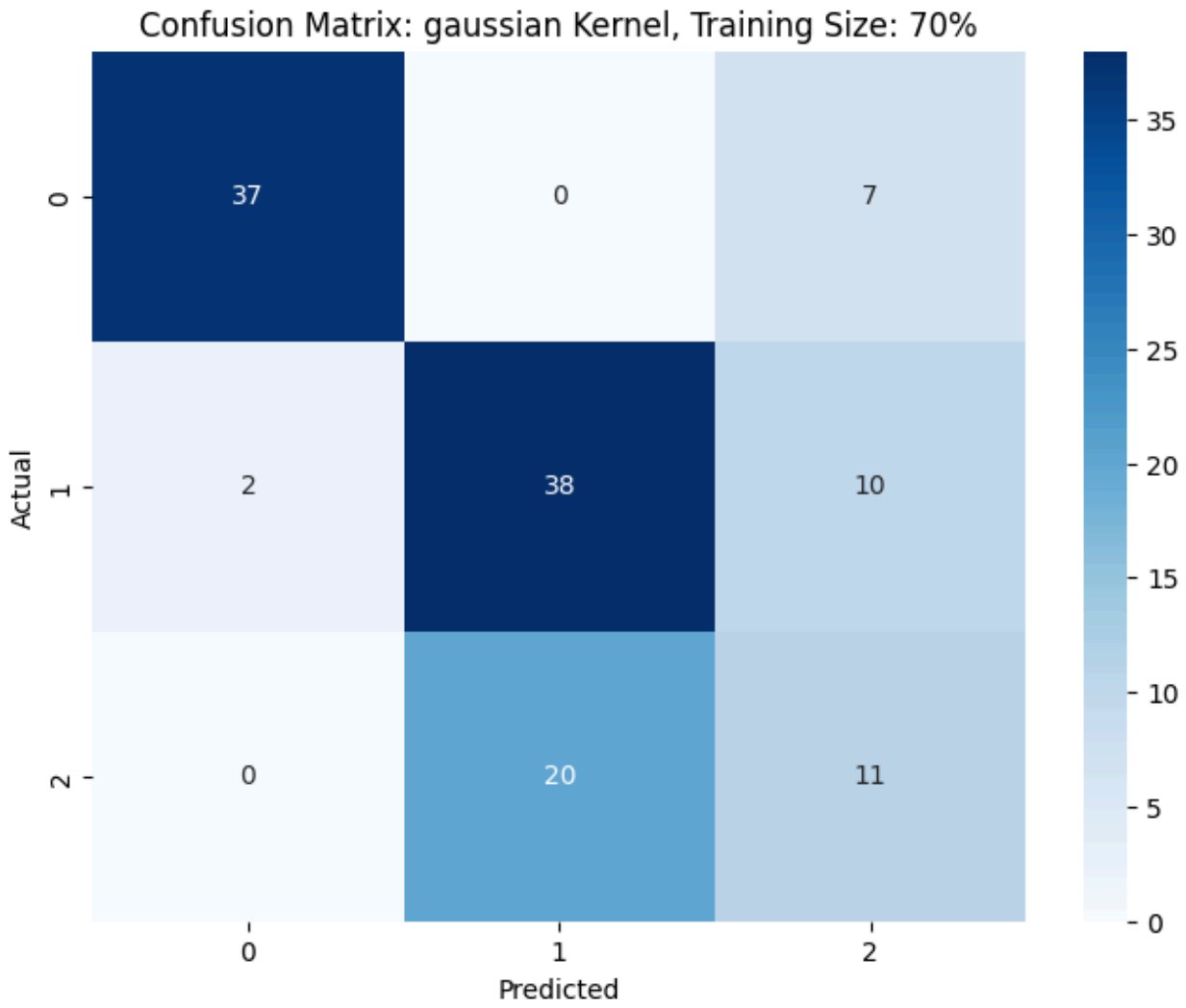

Confusion Matrix: polynomial Kernel, Training Size: 60%
28
0
8
2
36
5
0
23
5
Actual
Predicted
0
1
2
0
1
2
35
30
25
20
15
10
5
0

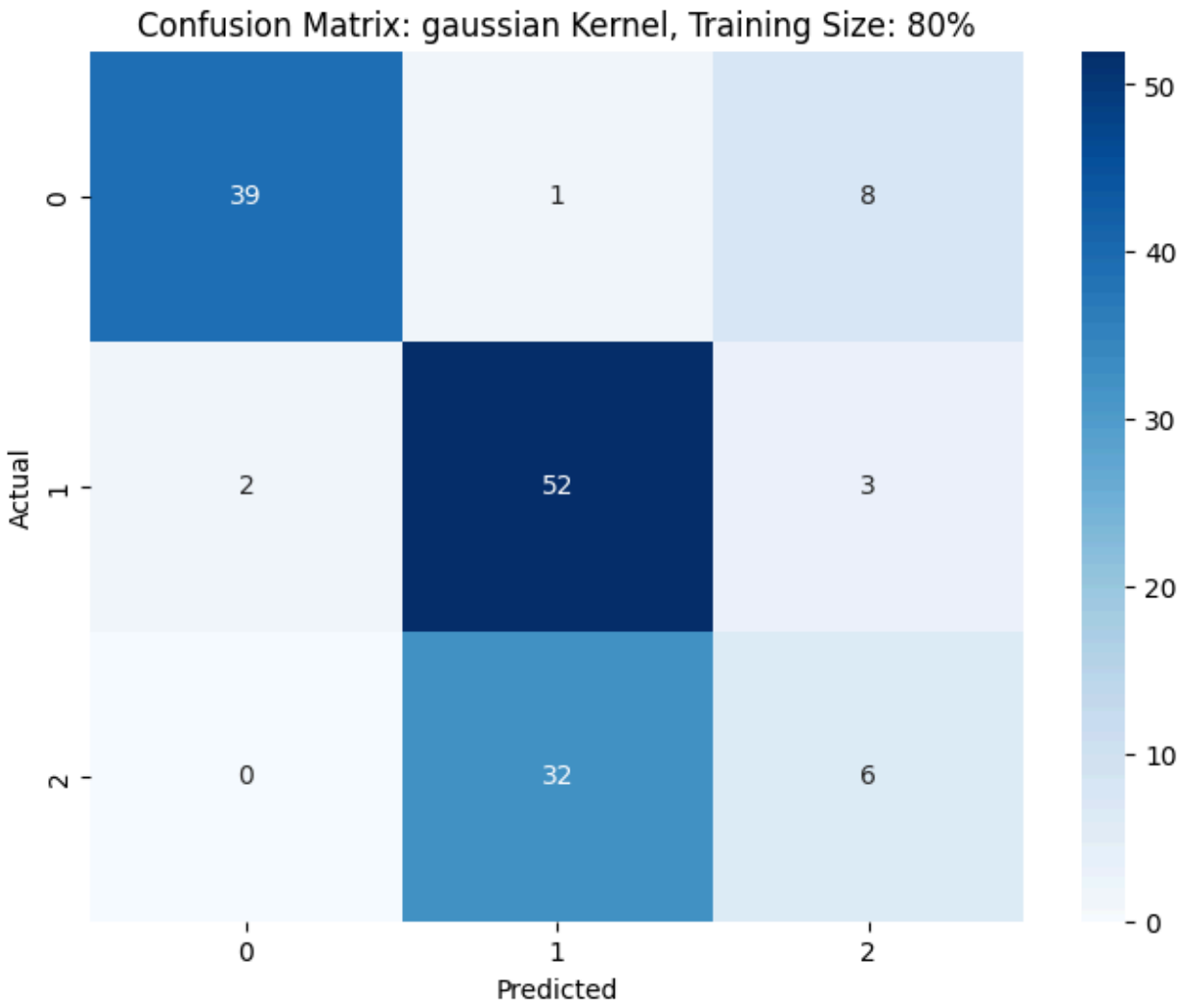

Confusion Matrix: polynomial Kernel, Training Size: 70%
Actual
Predicted
0
1
2
36
0
8
2
43
5
0
24
7
40
35
30
25
20
15
10
5
0

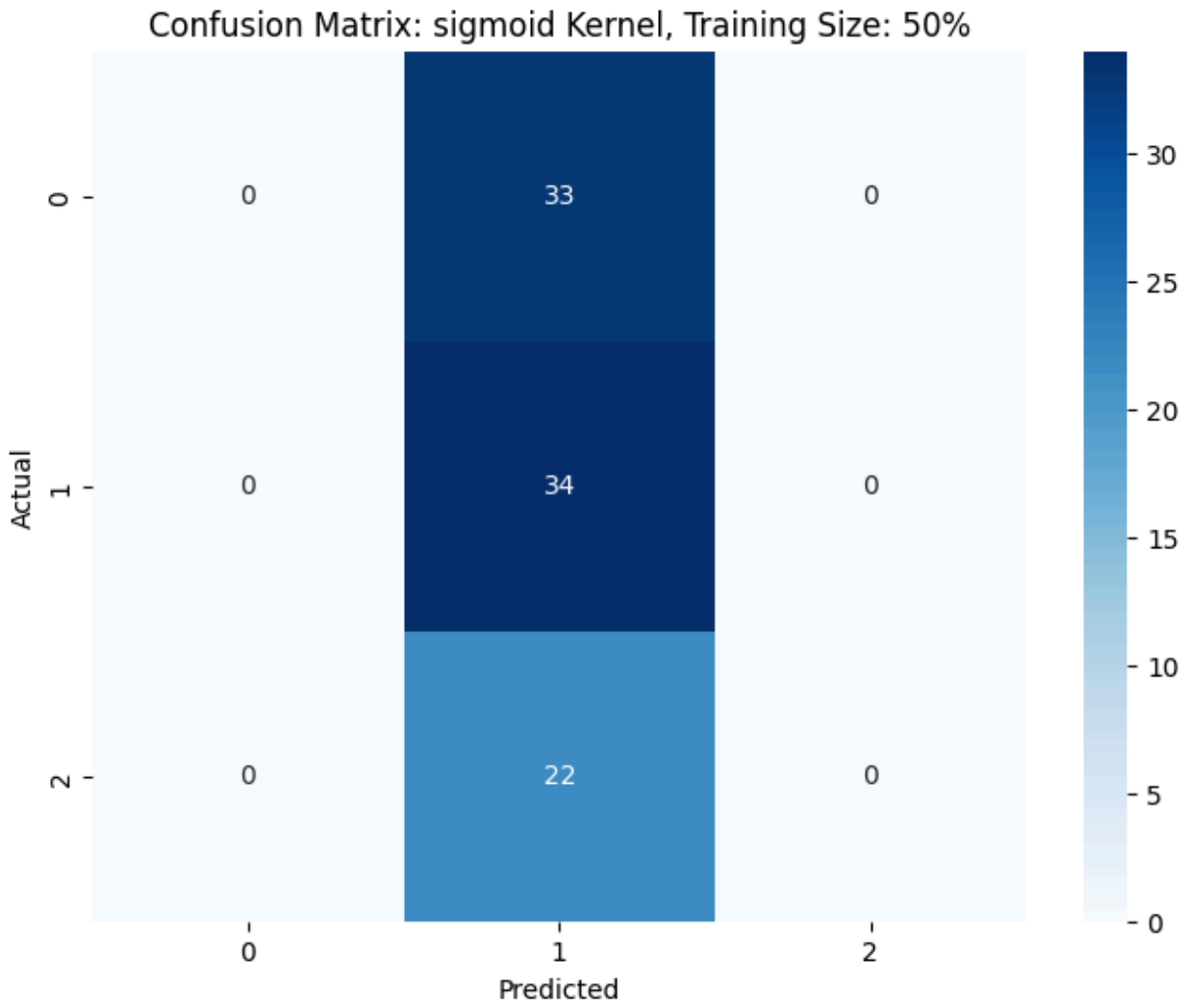

Confusion Matrix: polynomial Kernel, Training Size: 80%
39
1
8
2
51
4
0
32
6
Actual
Predicted
0
1
2
50
40
30
20
10
0

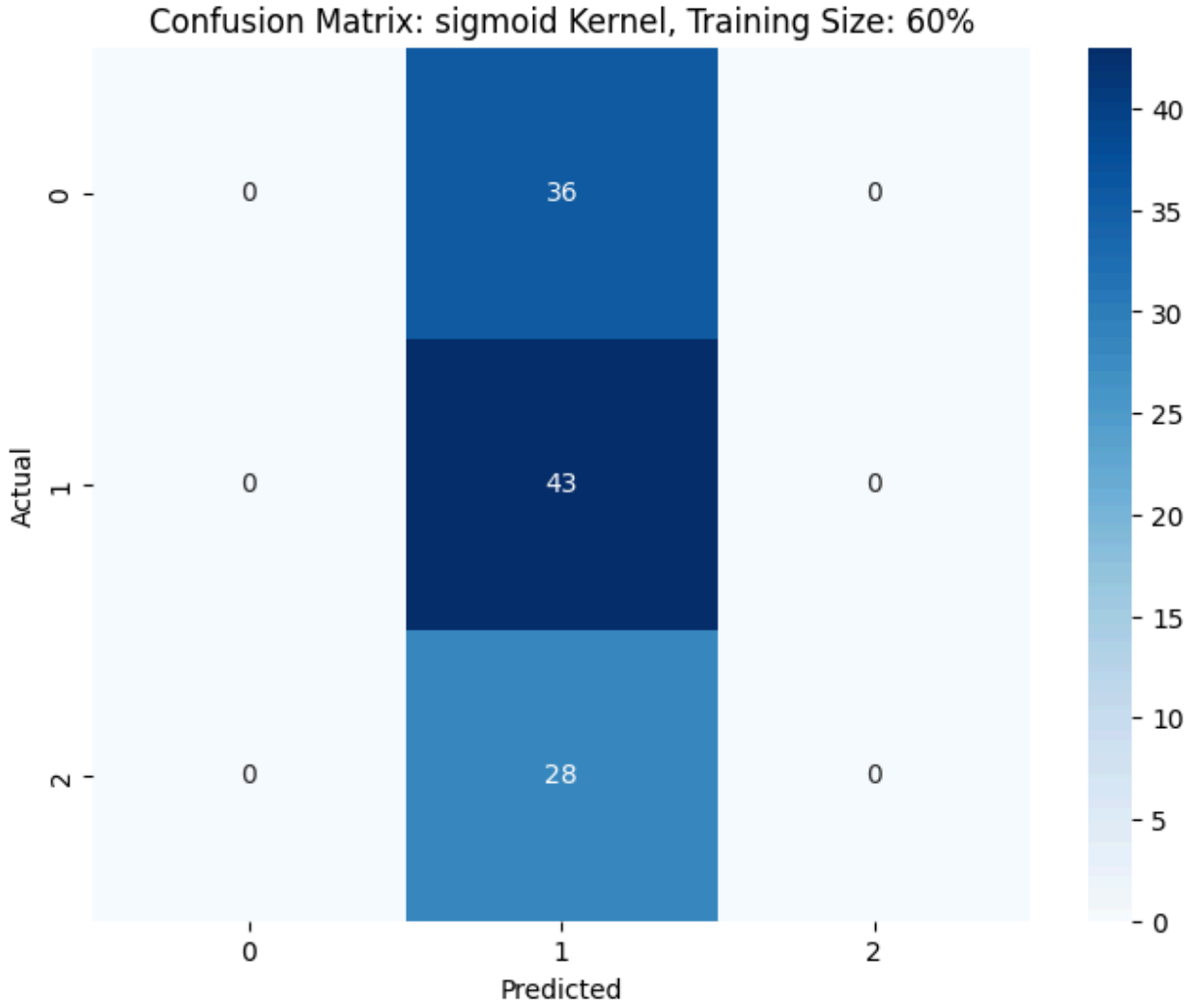

Confusion Matrix: gaussian Kernel, Training Size: 50%
28
0
5
1
23
10
0
15
7
Actual
Predicted
0
1
2
0
1
2
0
5
10
15
20
25

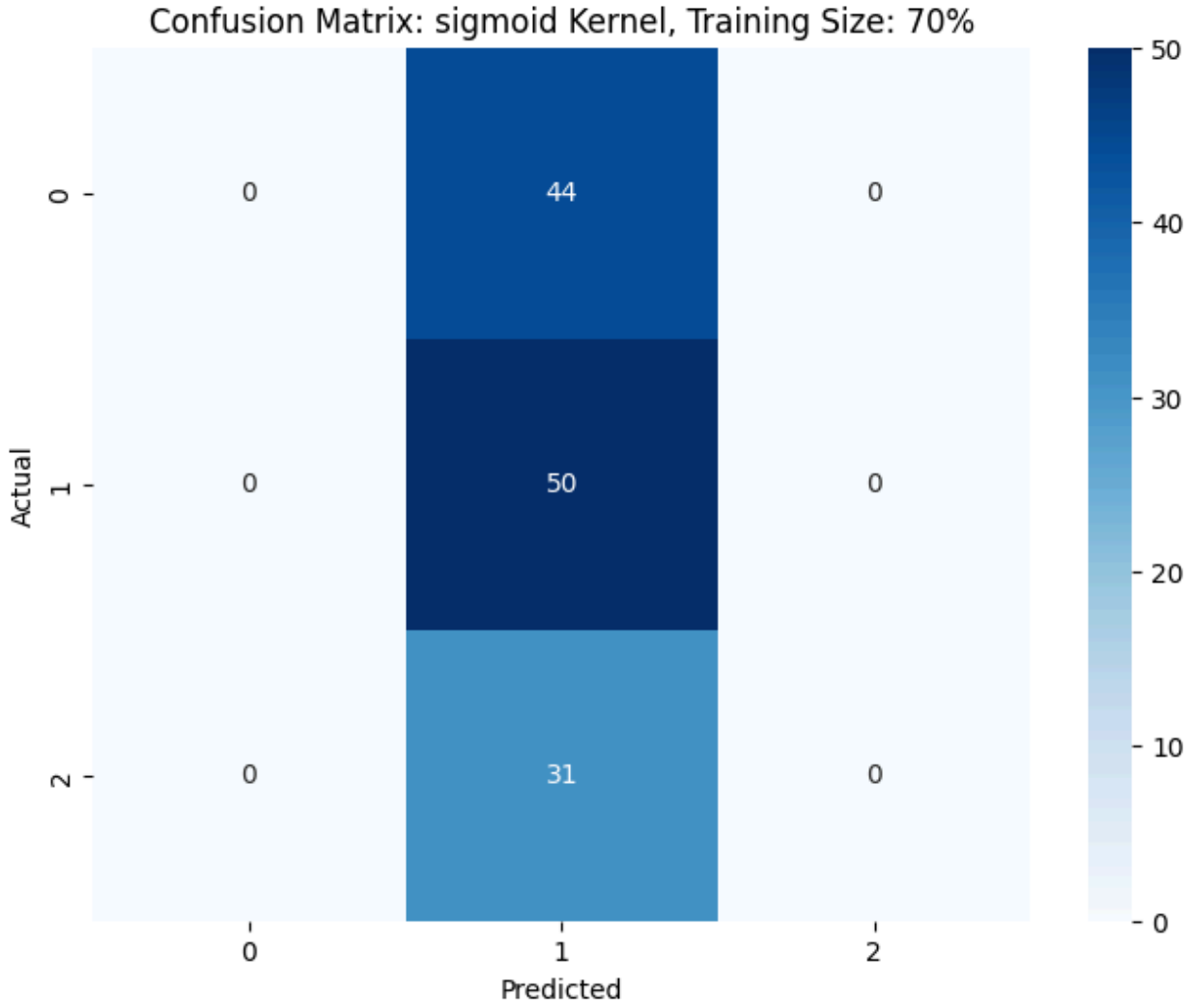

Confusion Matrix: gaussian Kernel, Training Size: 60%
30
0
6
2
32
9
0
19
9
Actual
Predicted
0
1
2
0
1
2
30
25
20
15
10
5
0

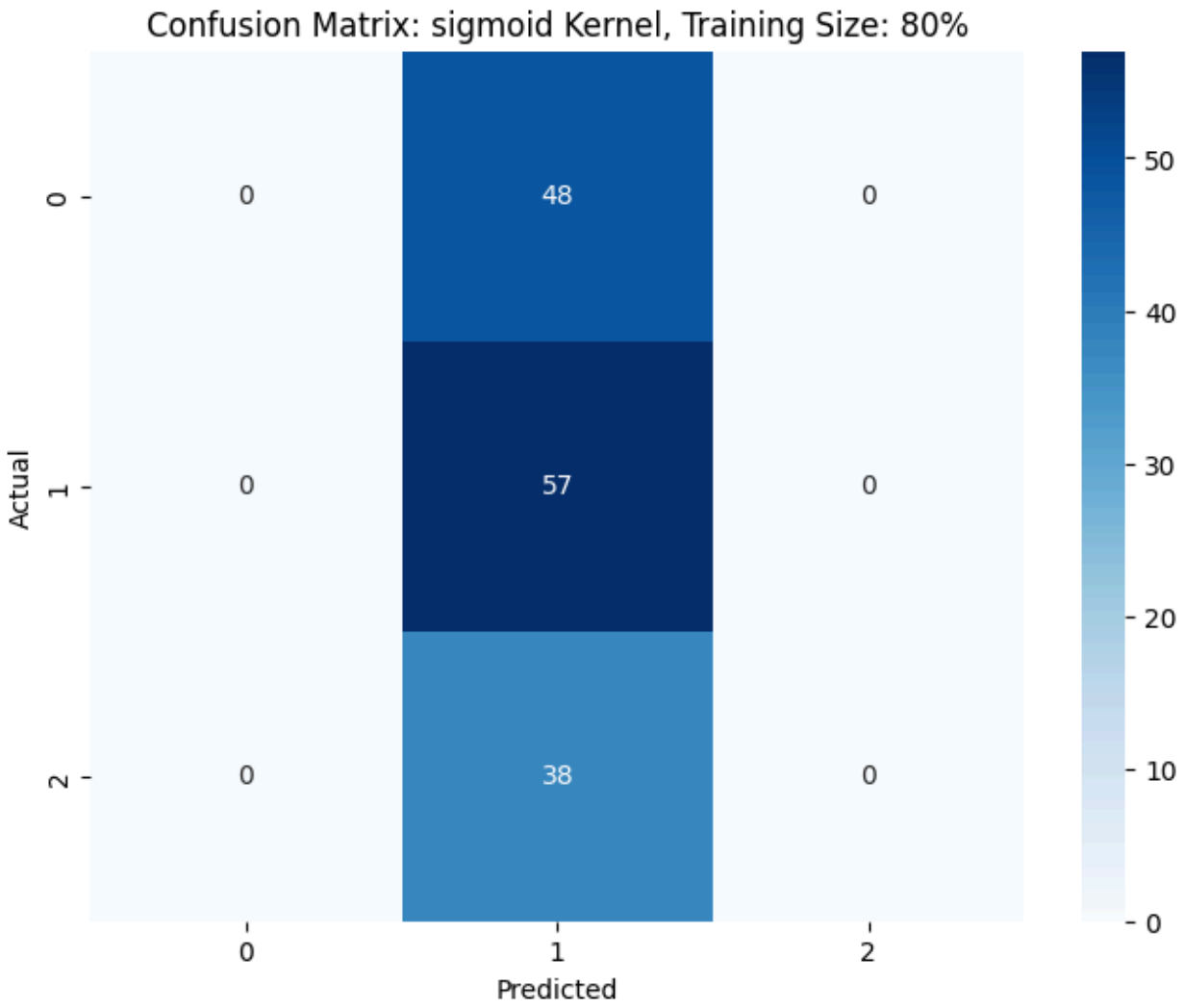

Confusion Matrix: gaussian Kernel, Training Size: 70%
37
0
7
2
38
10
0
20
11
Actual
Predicted
0
1
2
0
1
2
35
30
25
20
15
10
5
0


Confusion Matrix: gaussian Kernel, Training Size: 80%
Actual
Predicted
0
1
2
39
1
8
2
52
3
0
32
6
50
40
30
20
10
0


Confusion Matrix: sigmoid Kernel, Training Size: 50%
Actual
Predicted
0
1
2
0
33
0
0
34
0
0
22
0
30
25
20
15
10
5
0


Confusion Matrix: sigmoid Kernel, Training Size: 60%
Actual
Predicted
0
1
2
0
36
0
0
43
0
0
28
0
40
35
30
25
20
15
10
5
0

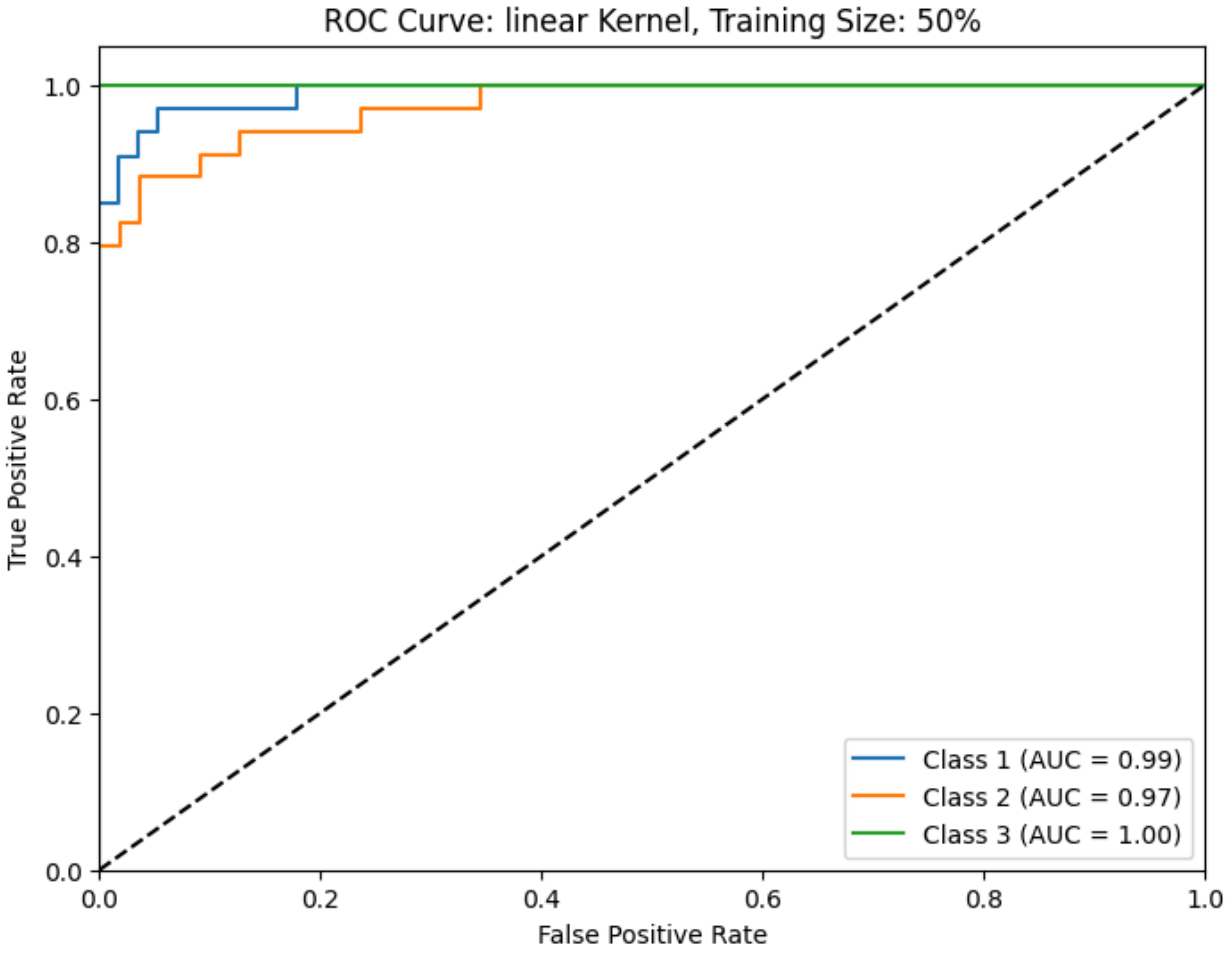

Confusion Matrix: sigmoid Kernel, Training Size: 70%
Actual
Predicted
0
1
2
0
44
0
0
50
0
0
31
0
50
40
30
20
10
0

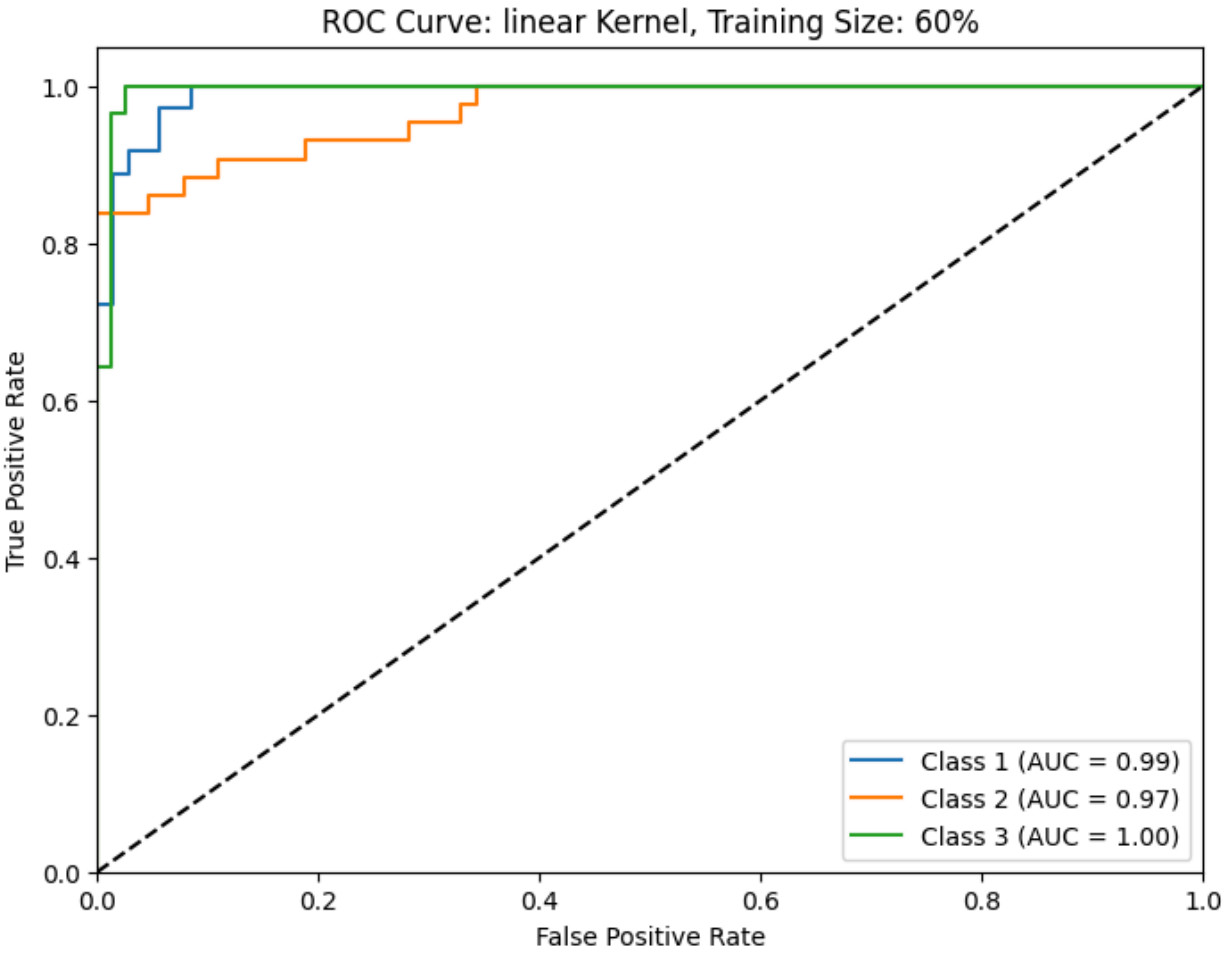


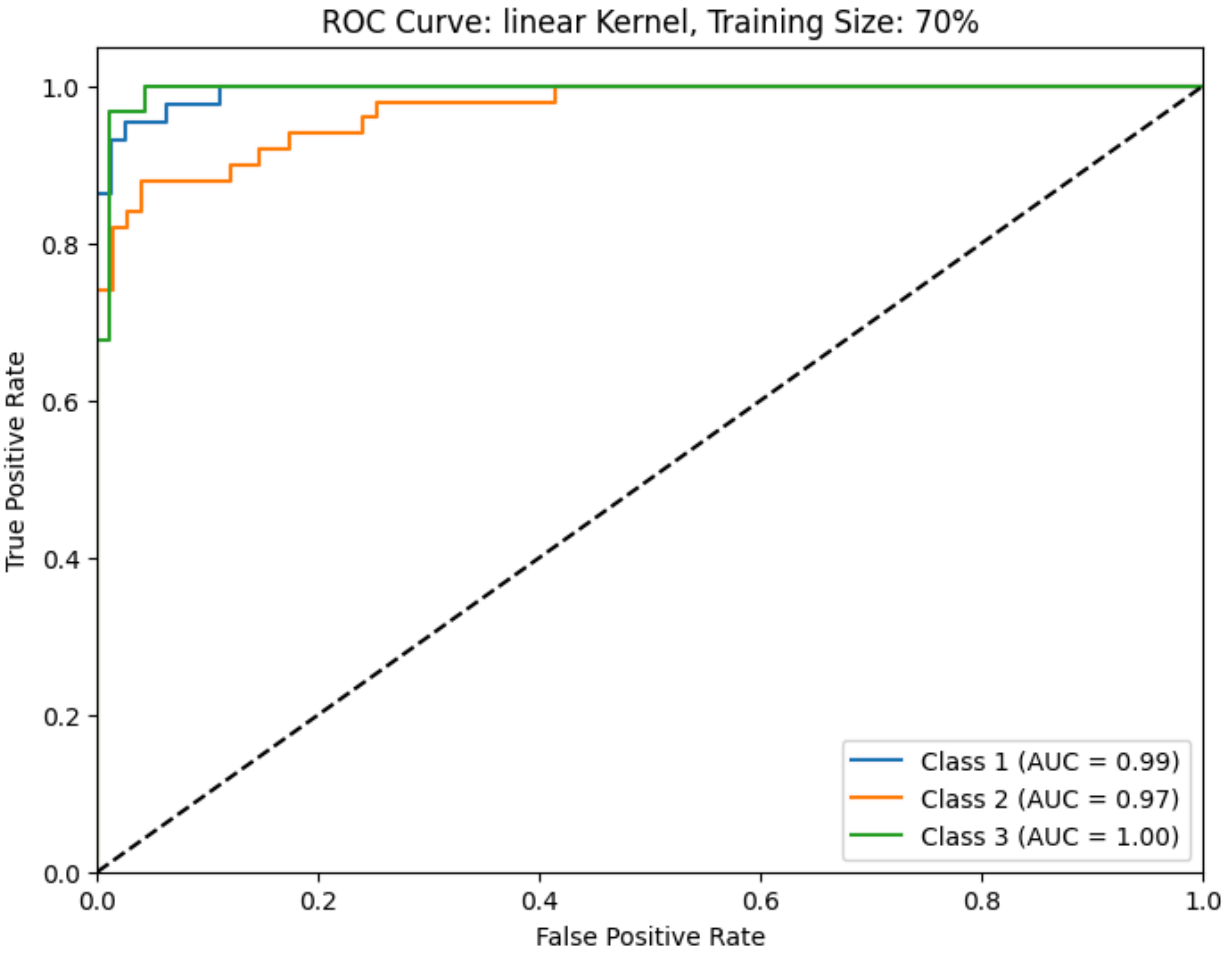
ROC and AOC curve

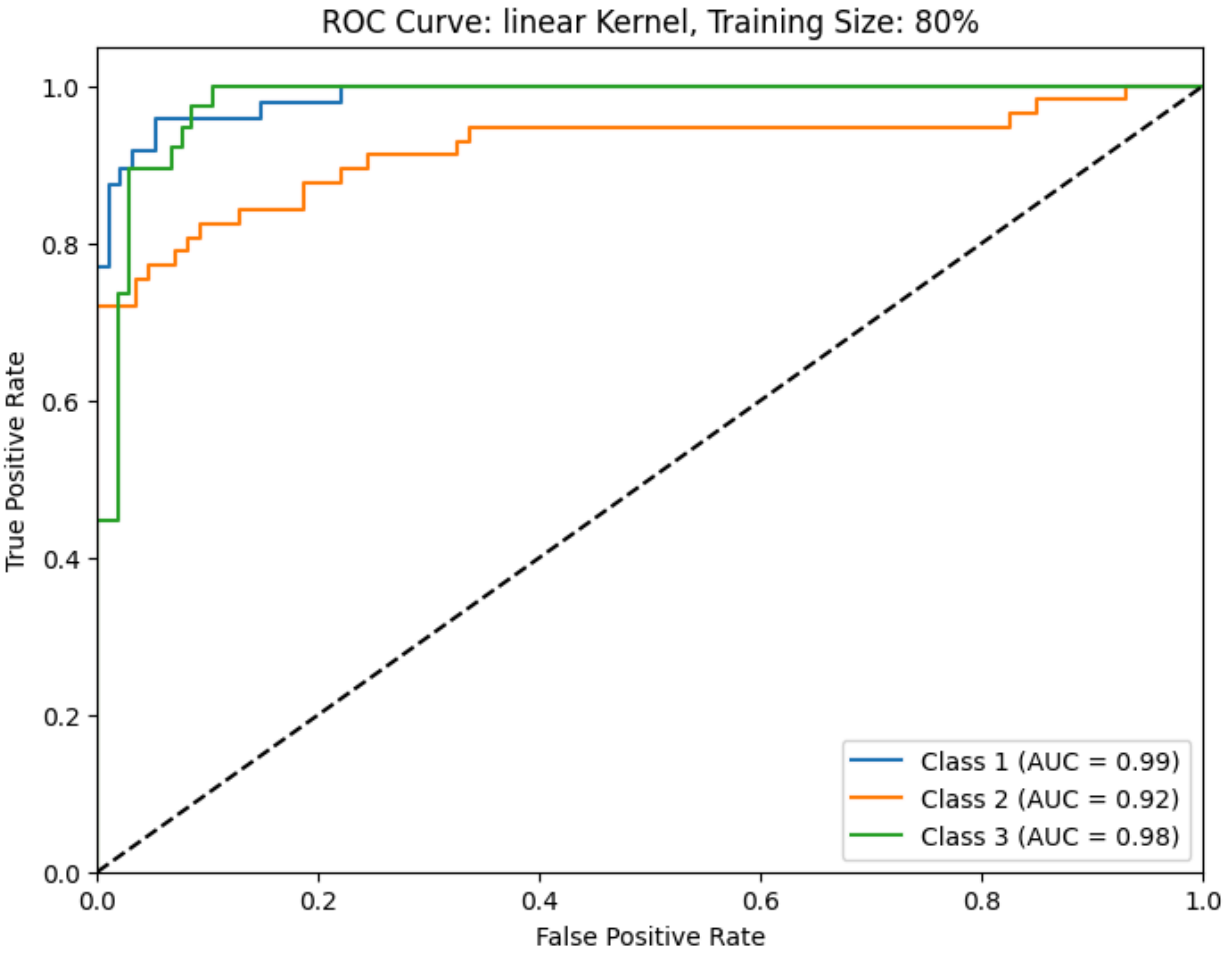
In [ ]:
```
from sklearn.metrics import roc_curve, auc

# 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()
```

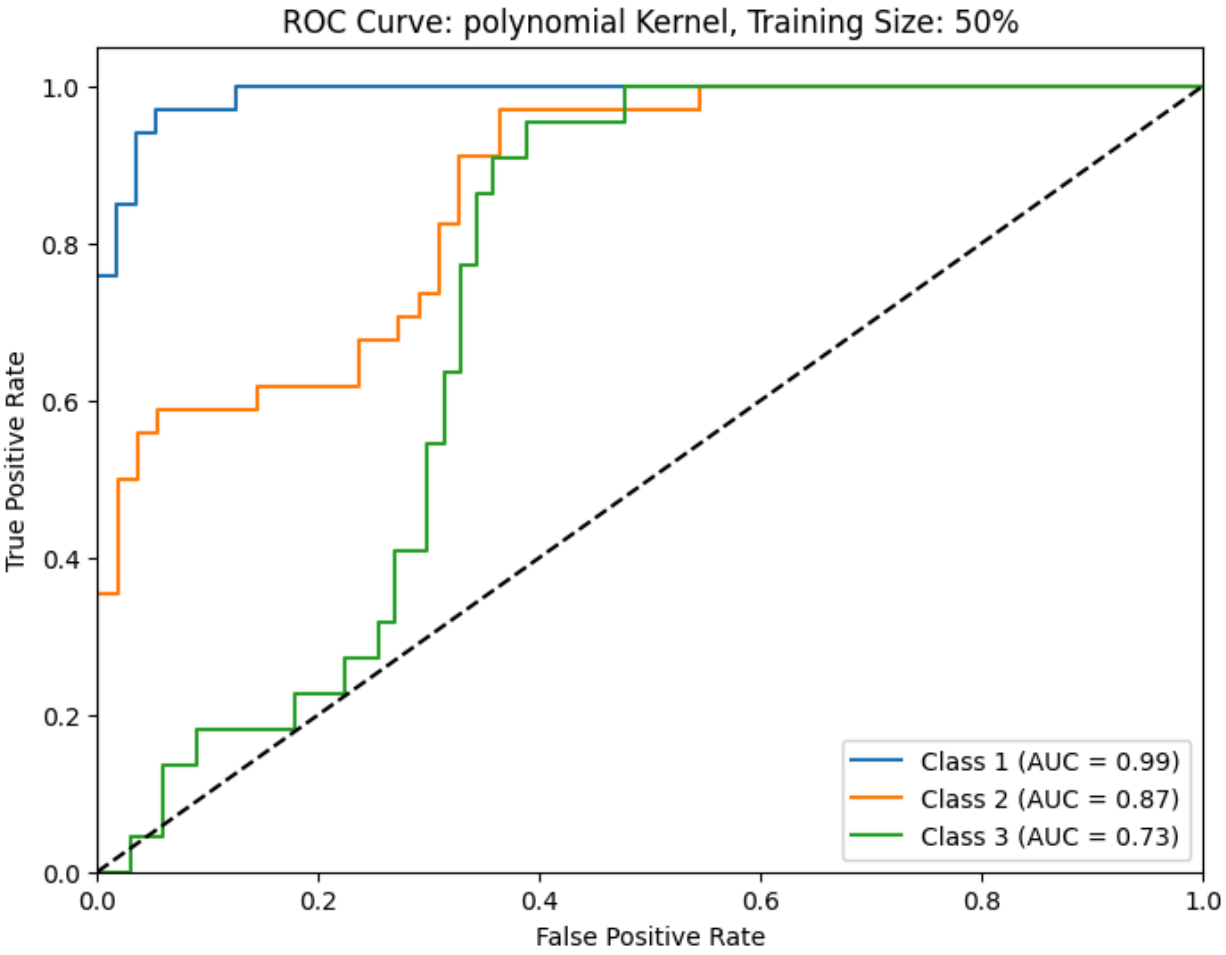
```
    roc_auc = dict()
    n_classes = len(np.unique(y_test))

    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()
```

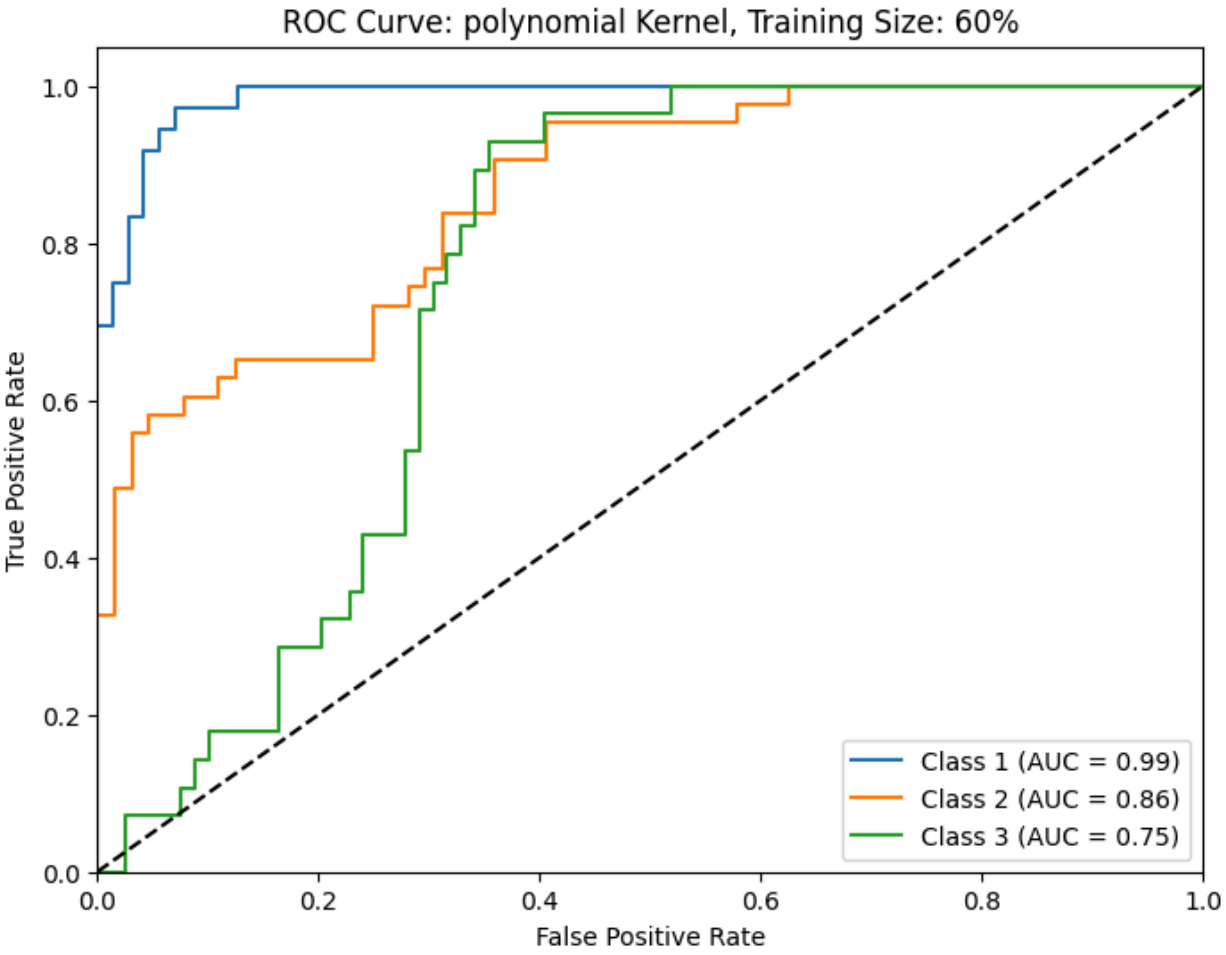


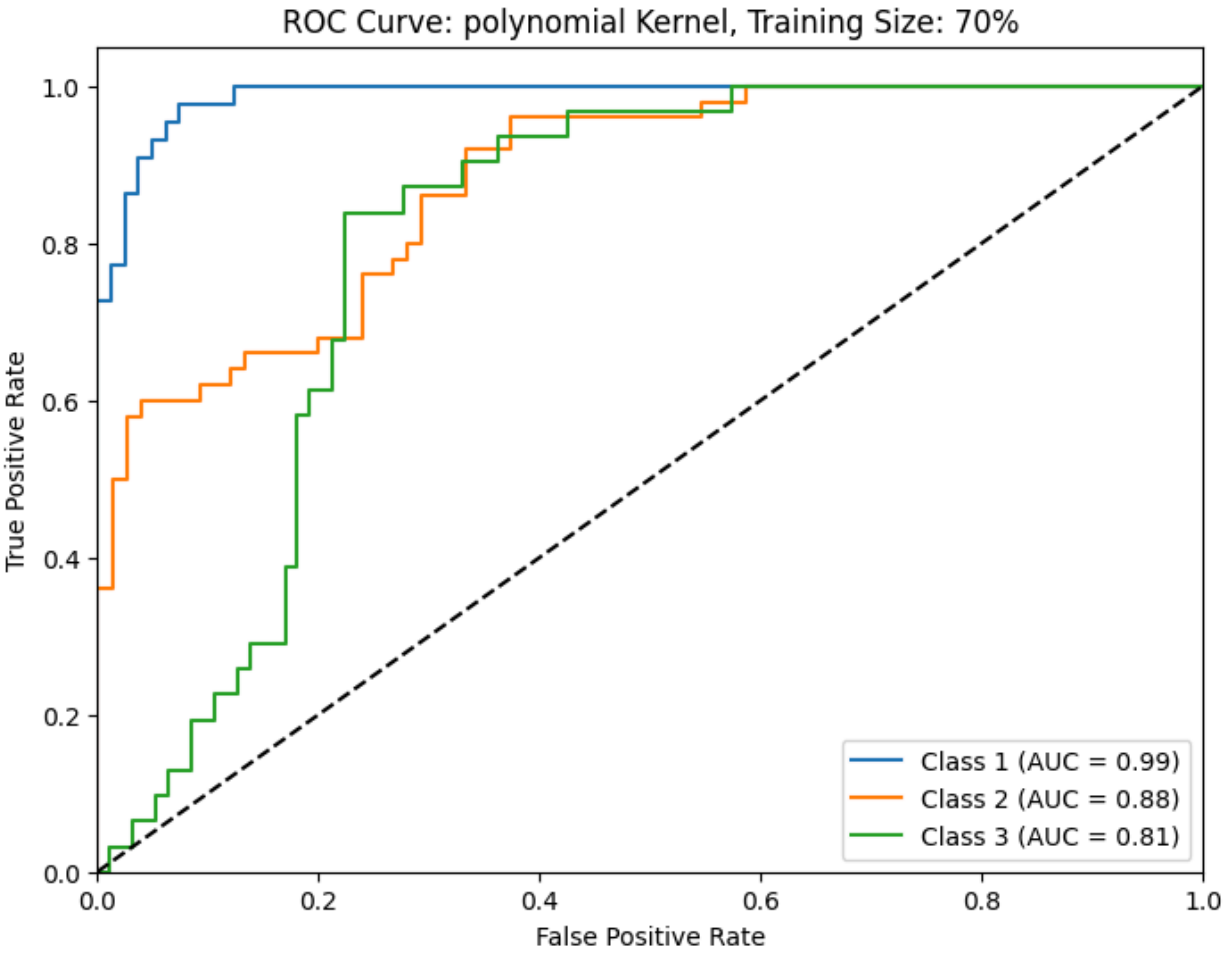

ROC Curve: linear Kernel, Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.97)
Class 3 (AUC = 1.00)

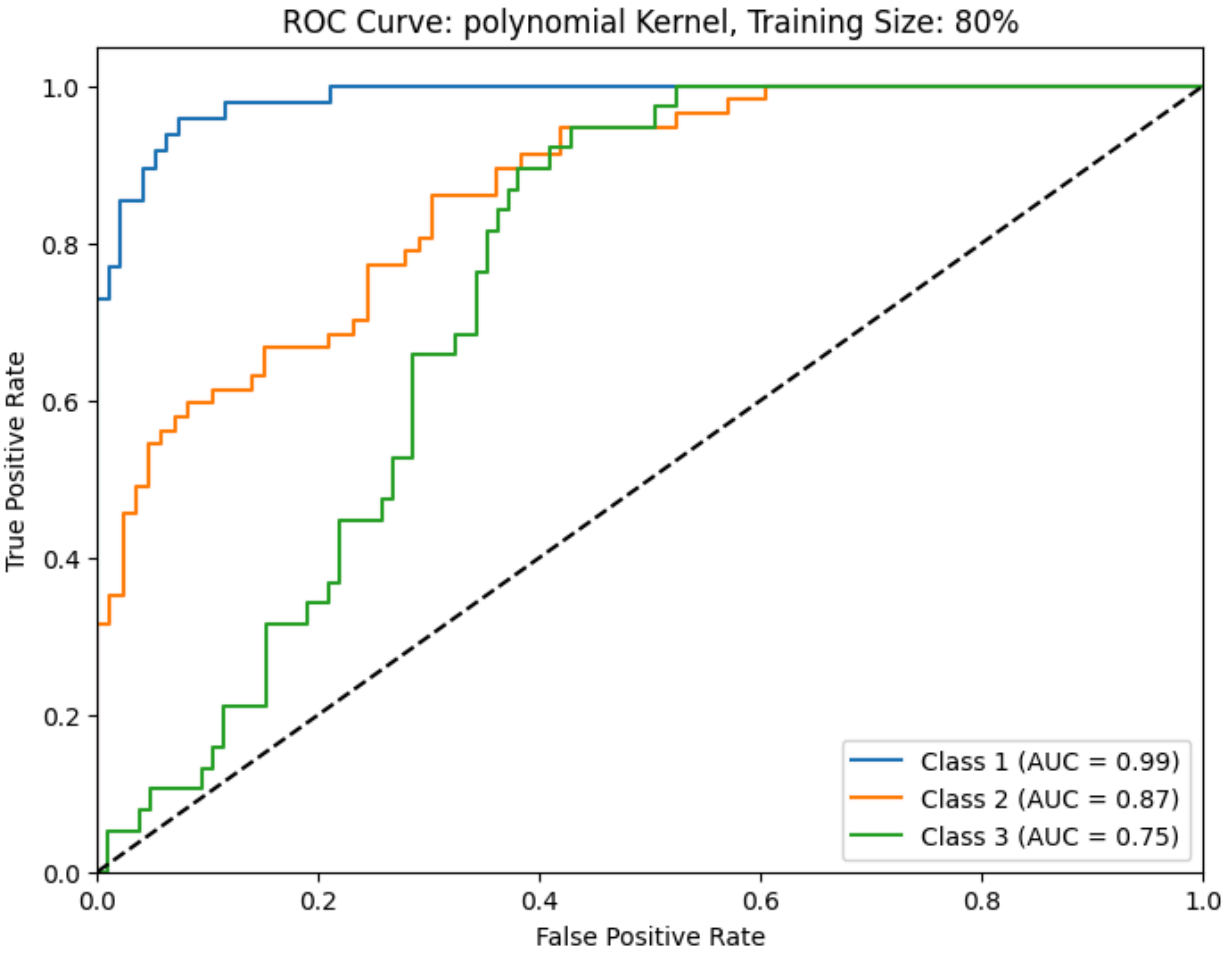

ROC Curve: linear Kernel, Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.97)
Class 3 (AUC = 1.00)

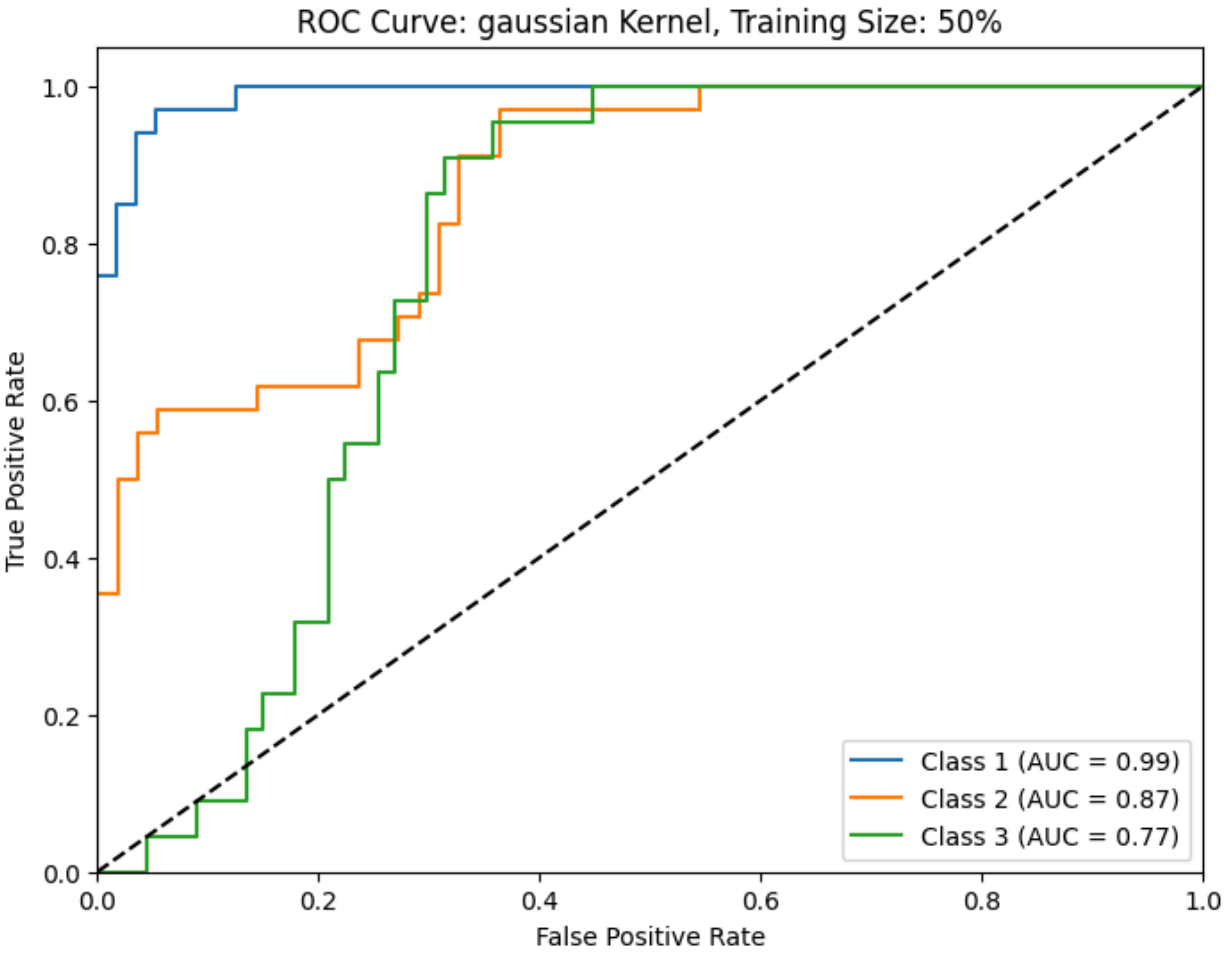

ROC Curve: linear Kernel, Training Size: 80%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.92)
Class 3 (AUC = 0.98)

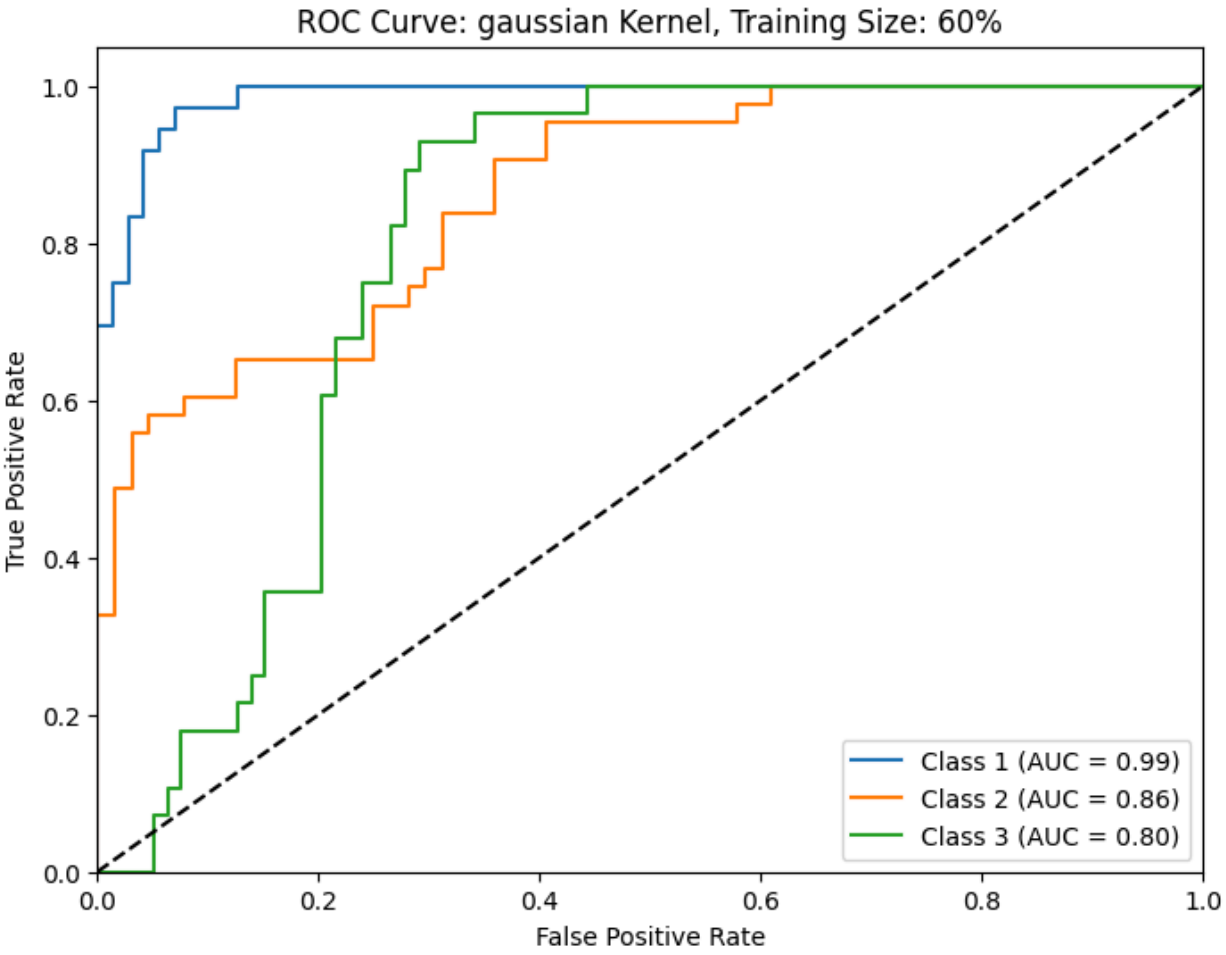

ROC Curve: polynomial Kernel, Training Size: 50%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.87)
Class 3 (AUC = 0.73)

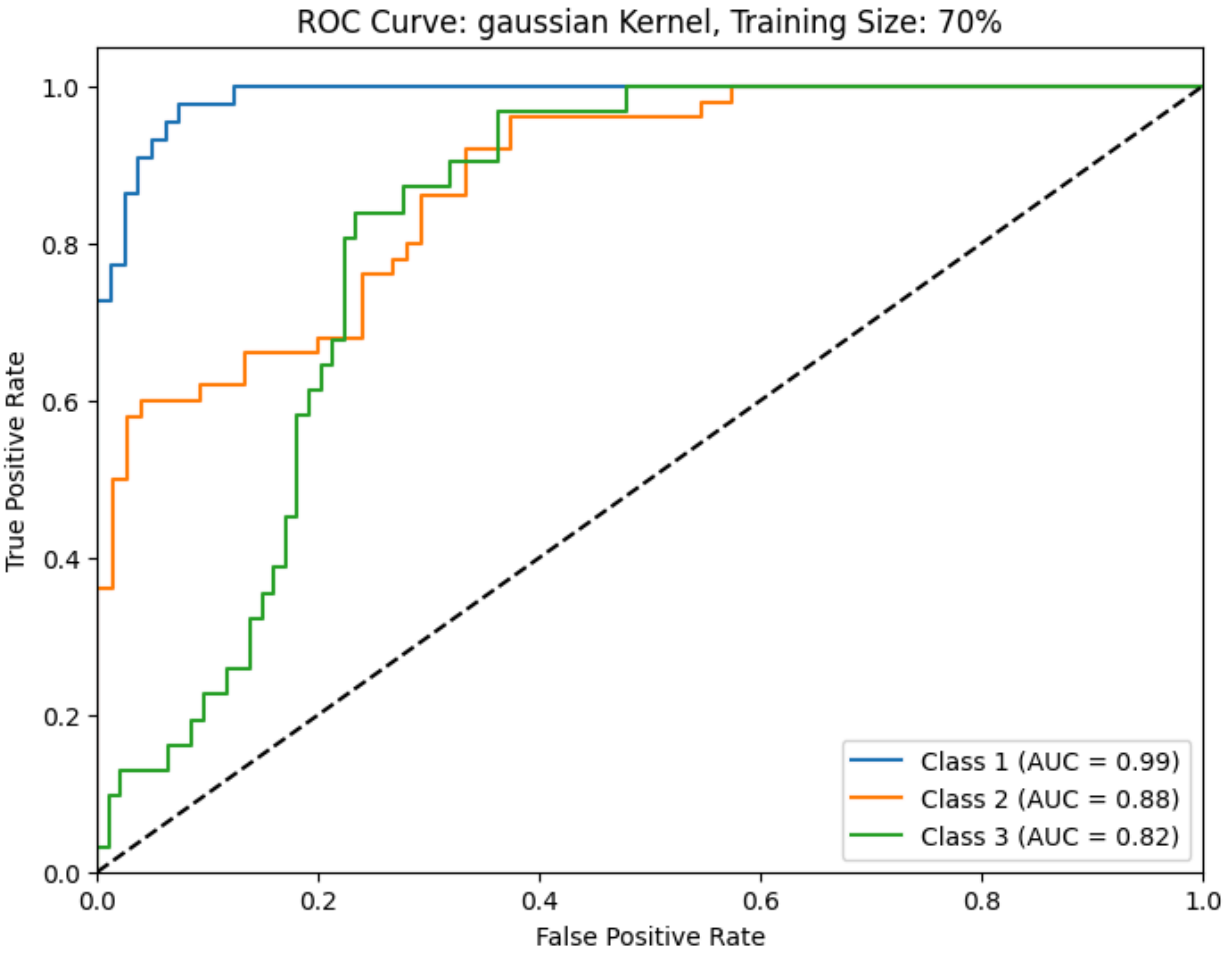

ROC Curve: polynomial Kernel, Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.86)
Class 3 (AUC = 0.75)

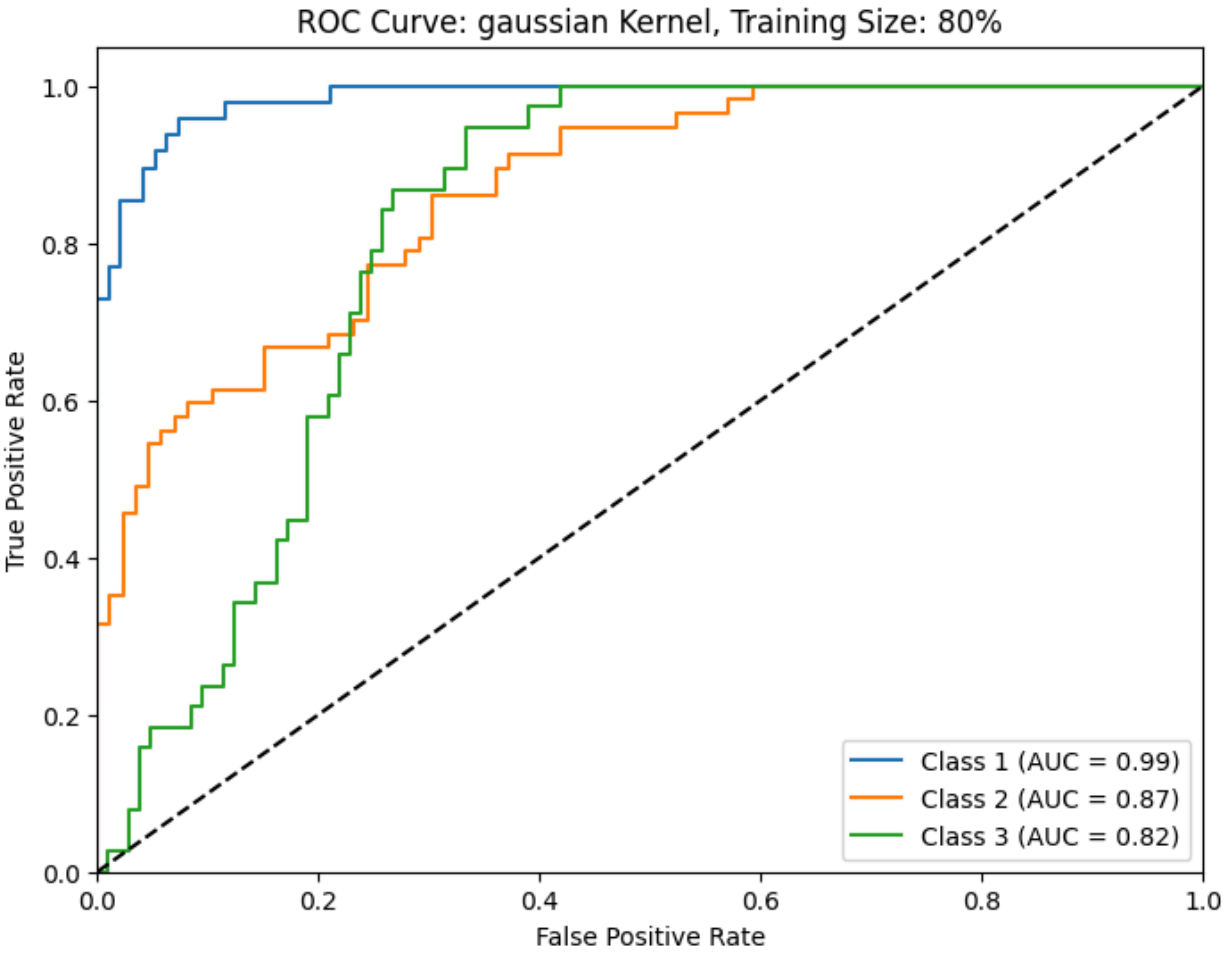

ROC Curve: polynomial Kernel, Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.88)
Class 3 (AUC = 0.81)

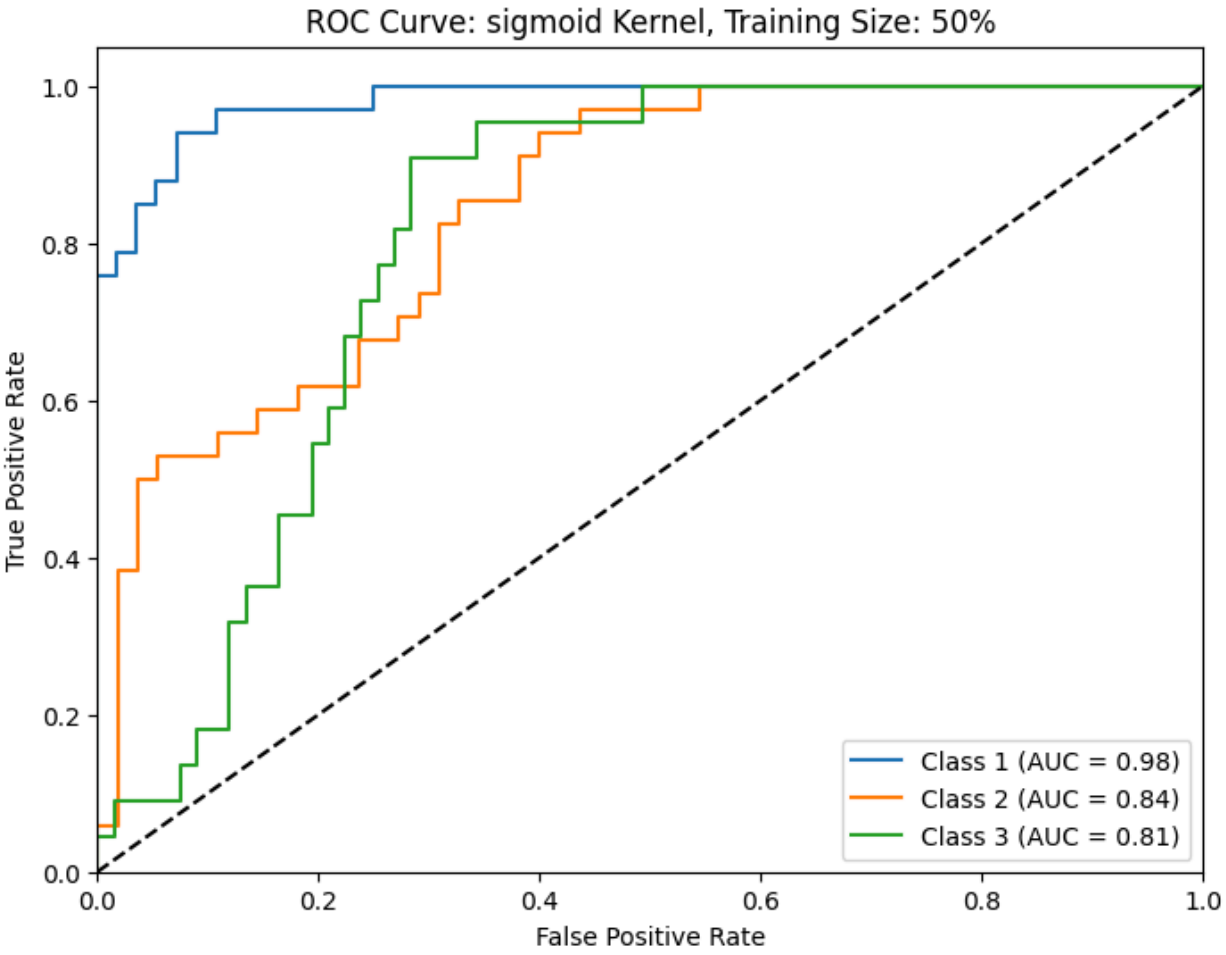

ROC Curve: polynomial Kernel, Training Size: 80%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.87)
Class 3 (AUC = 0.75)

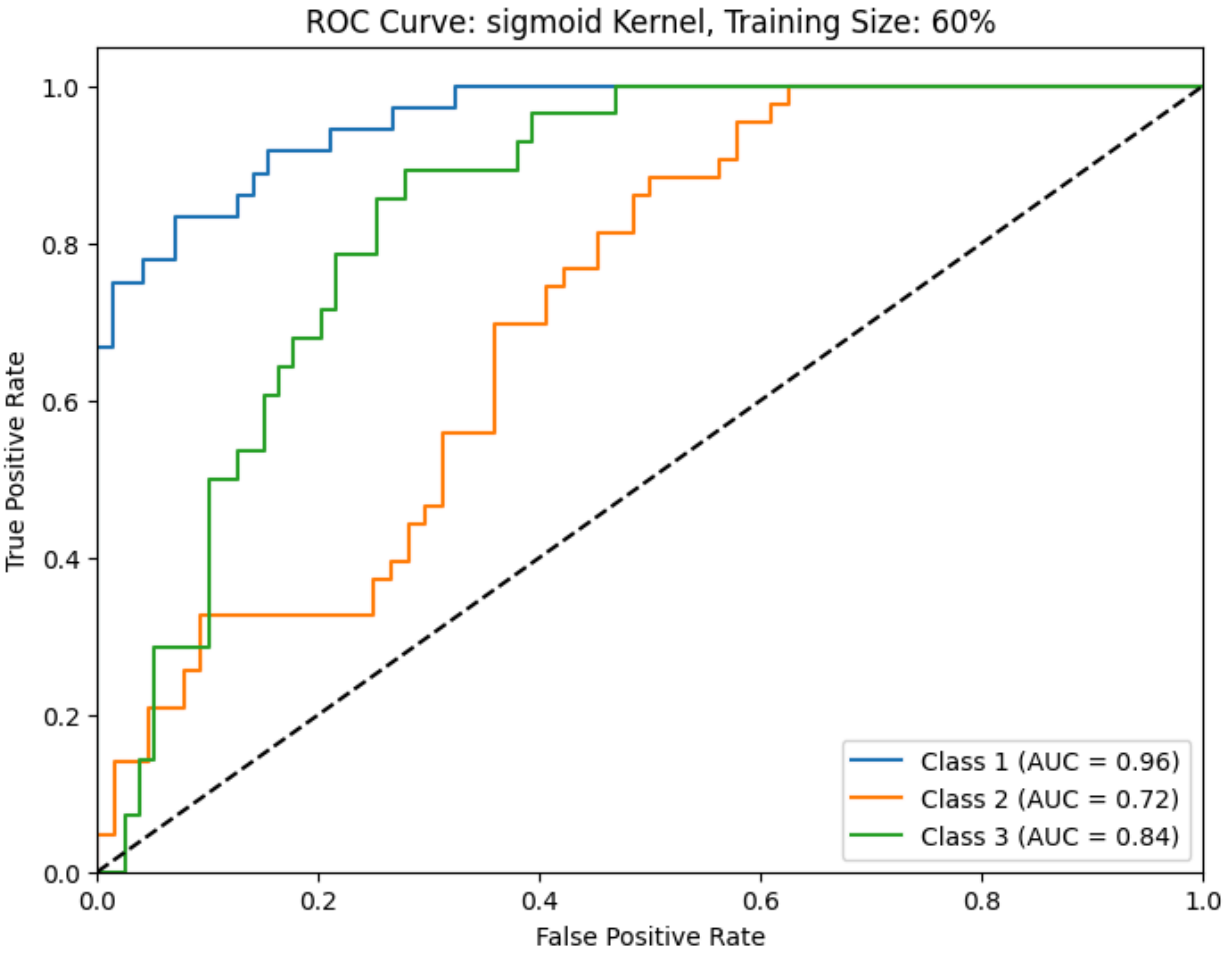

ROC Curve: gaussian Kernel, Training Size: 50%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.87)
Class 3 (AUC = 0.77)

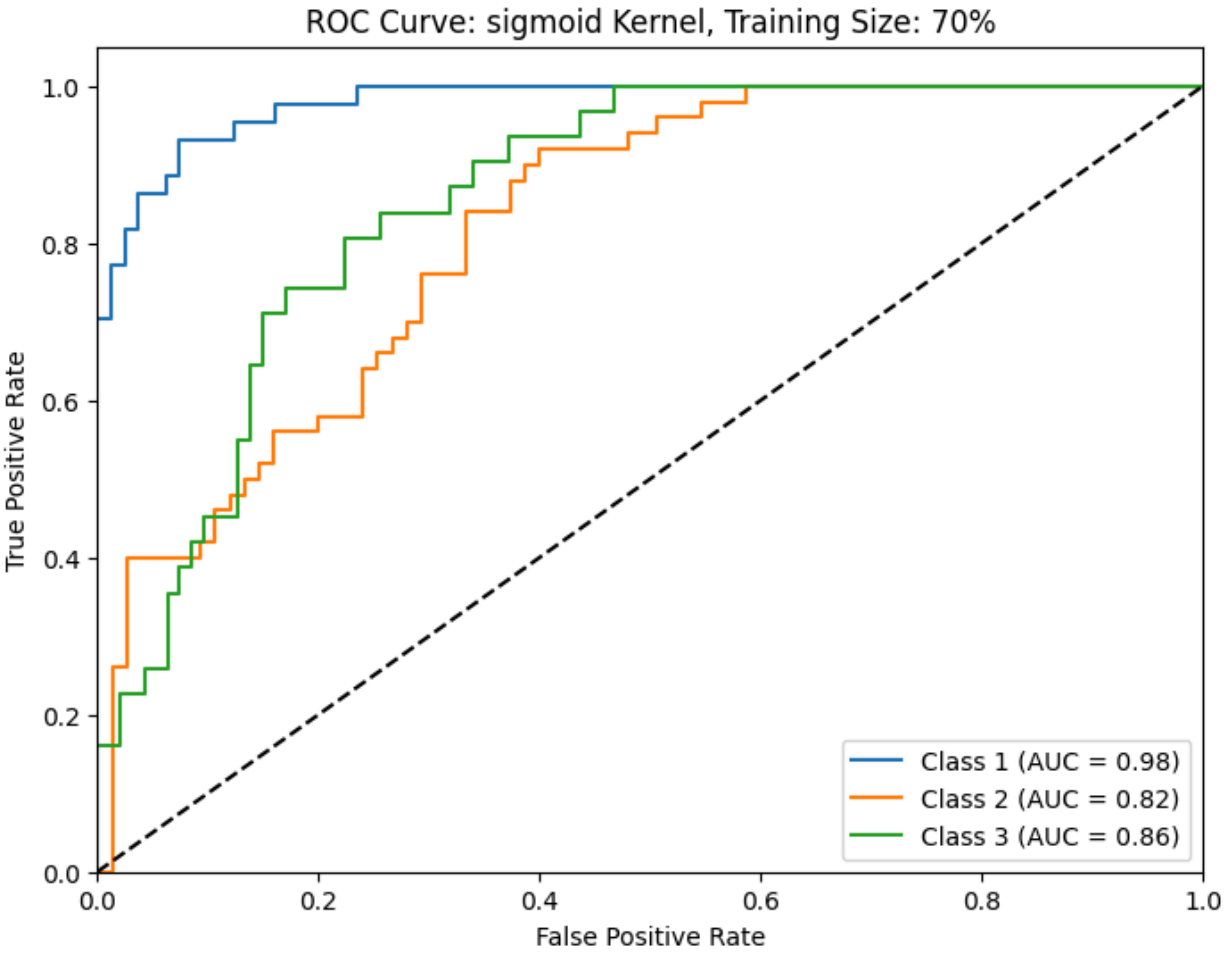

ROC Curve: gaussian Kernel, Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.86)
Class 3 (AUC = 0.80)

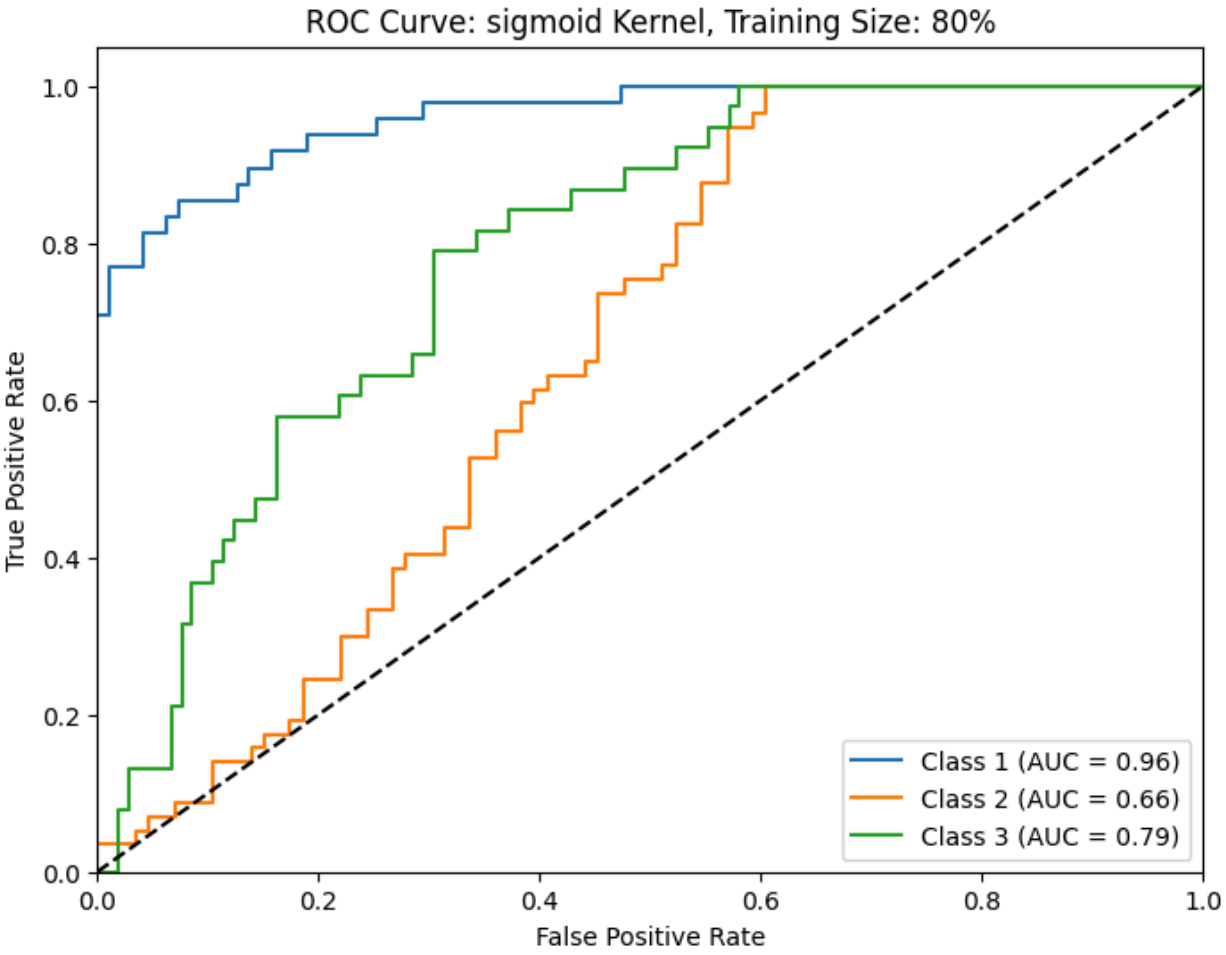

ROC Curve: gaussian Kernel, Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.88)
Class 3 (AUC = 0.82)


ROC Curve: gaussian Kernel, Training Size: 80%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.87)
Class 3 (AUC = 0.82)


ROC Curve: sigmoid Kernel, Training Size: 50%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.98)
Class 2 (AUC = 0.84)
Class 3 (AUC = 0.81)


ROC Curve: sigmoid Kernel, Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.96)
Class 2 (AUC = 0.72)
Class 3 (AUC = 0.84)


ROC Curve: sigmoid Kernel, Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.98)
Class 2 (AUC = 0.82)
Class 3 (AUC = 0.86)



Calculate values for different test size using PCA-reduced data

In [ ]:

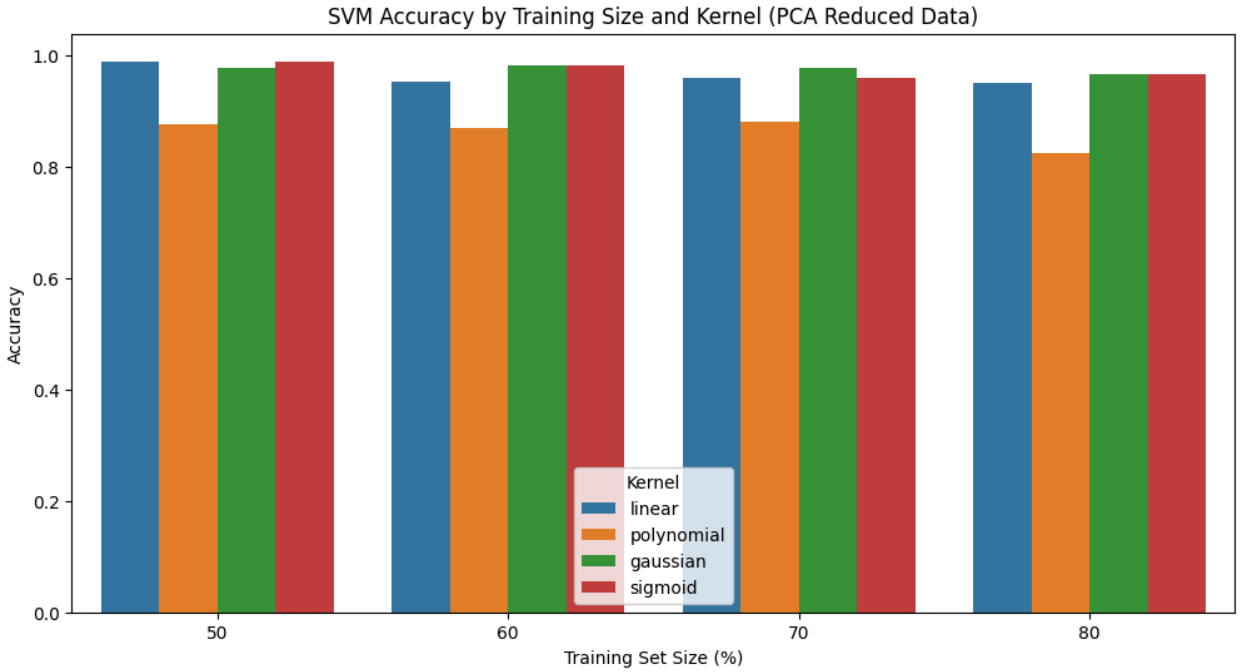
```
results_pca = []
confusion_matrices_pca = []

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 proba
   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', zer
   recall_pca = recall_score(y_test, y_pred_pca, average='weighted') # Calcula
   f1_pca = f1_score(y_test, y_pred_pca, average='weighted') # Calculate weigh


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

   results_pca.append({
       "Training size":int (size*100),
```

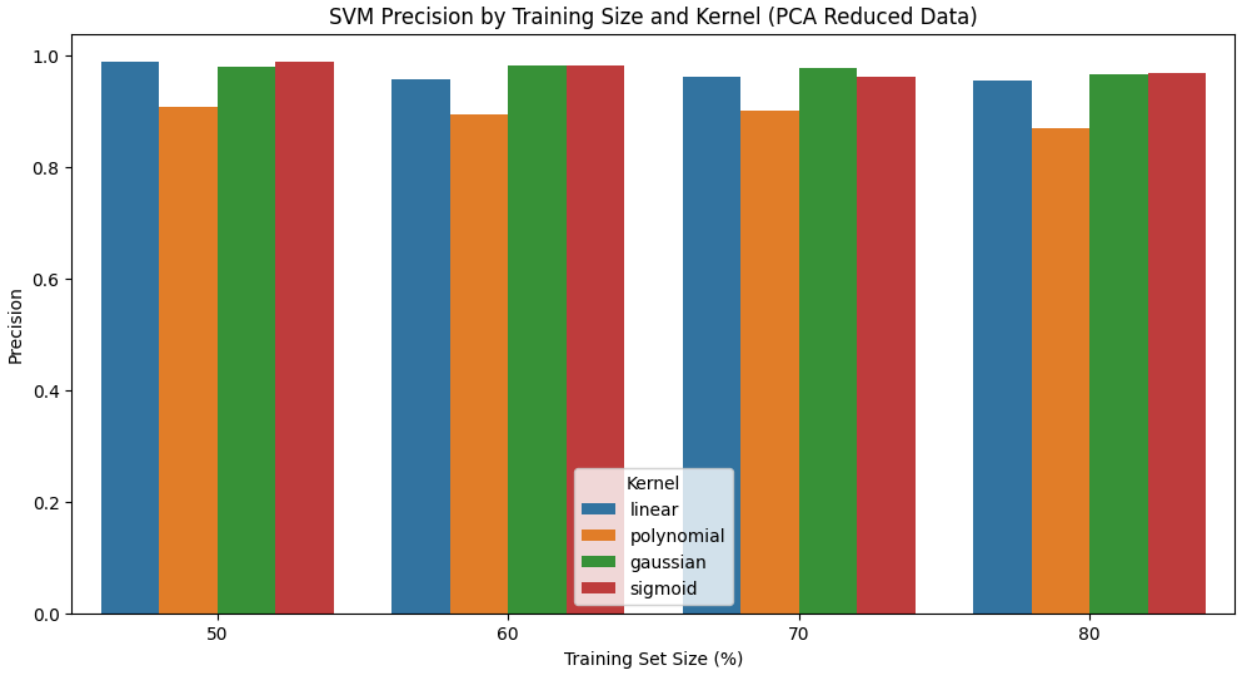
```
        "Accuracy":acc_pca,
        "Precision": precision_pca,
        "Recall": recall_pca,
        "F1-score": f1_pca,
        "Kernel":kernel_name
    })
    confusion_matrices_pca.append({ # Store confusion matrix with metadata
        "Training size":int (size*100),
        "Kernel":kernel_name,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

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

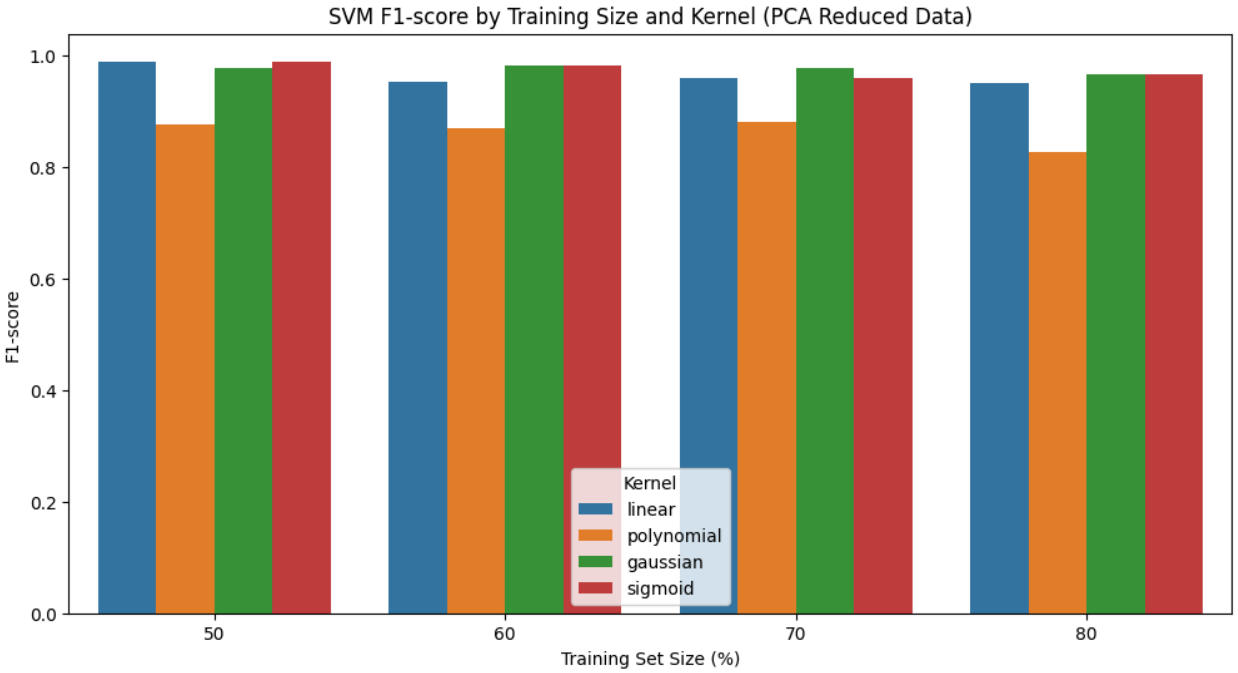
In [ ]:
```
df_pca = pd.DataFrame(results_pca)
display(df_pca) # Display the DataFrame as a table

# 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
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)') # Ad
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 t
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
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	0.988764	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



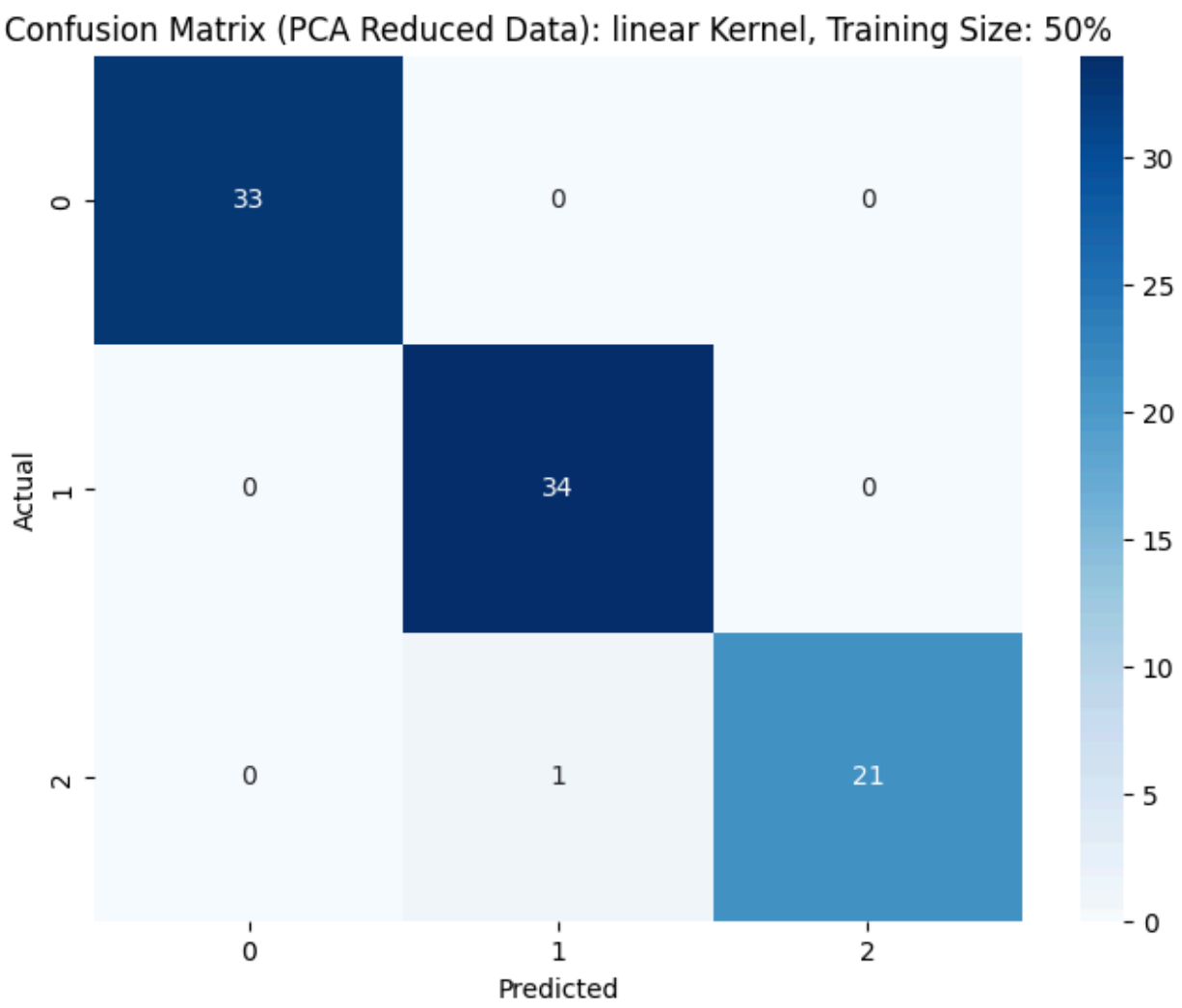

SVM Precision by Training Size and Kernel (PCA Reduced Data)
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid

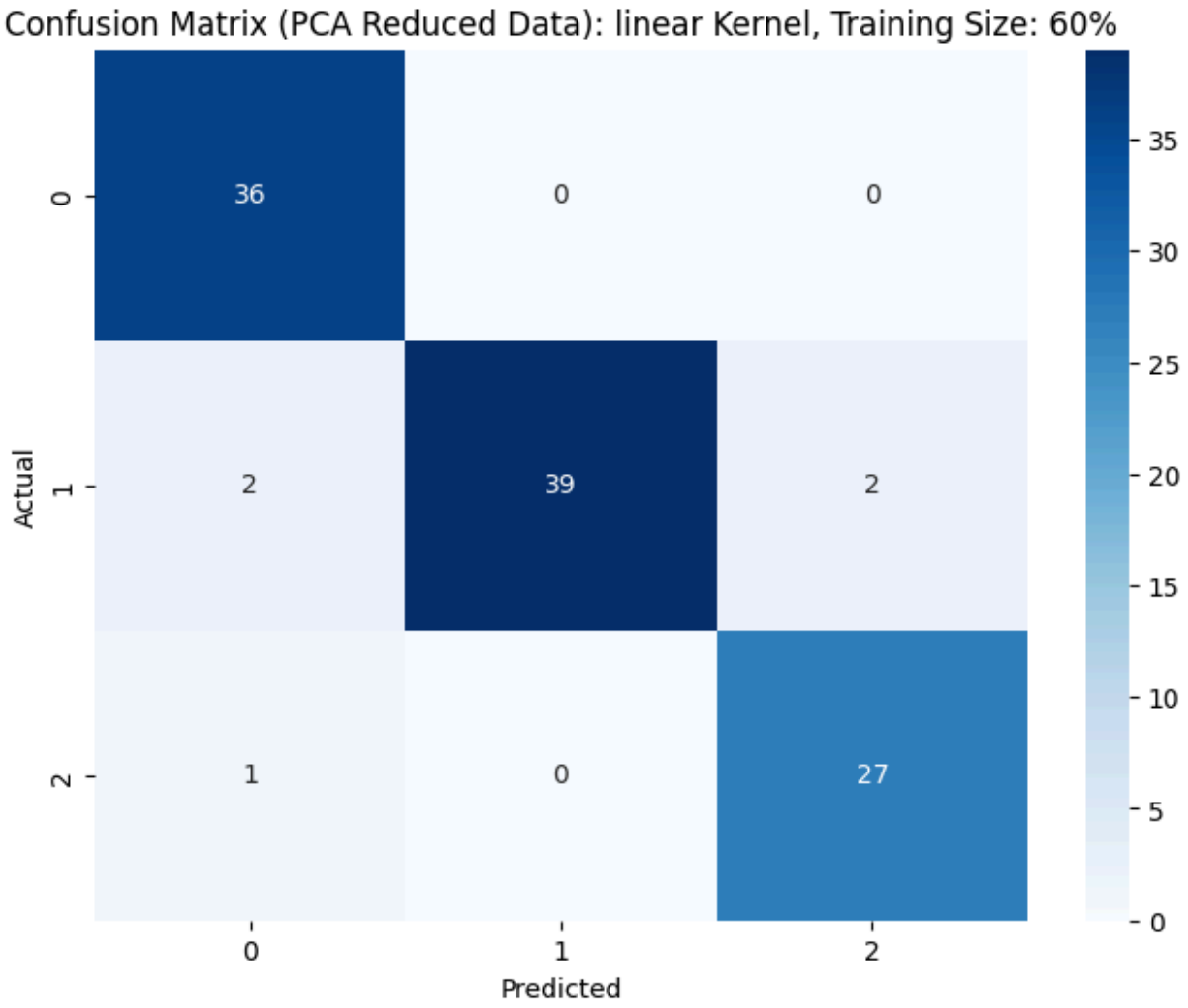

SVM Recall by Training Size and Kernel (PCA Reduced Data)
Recall
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid

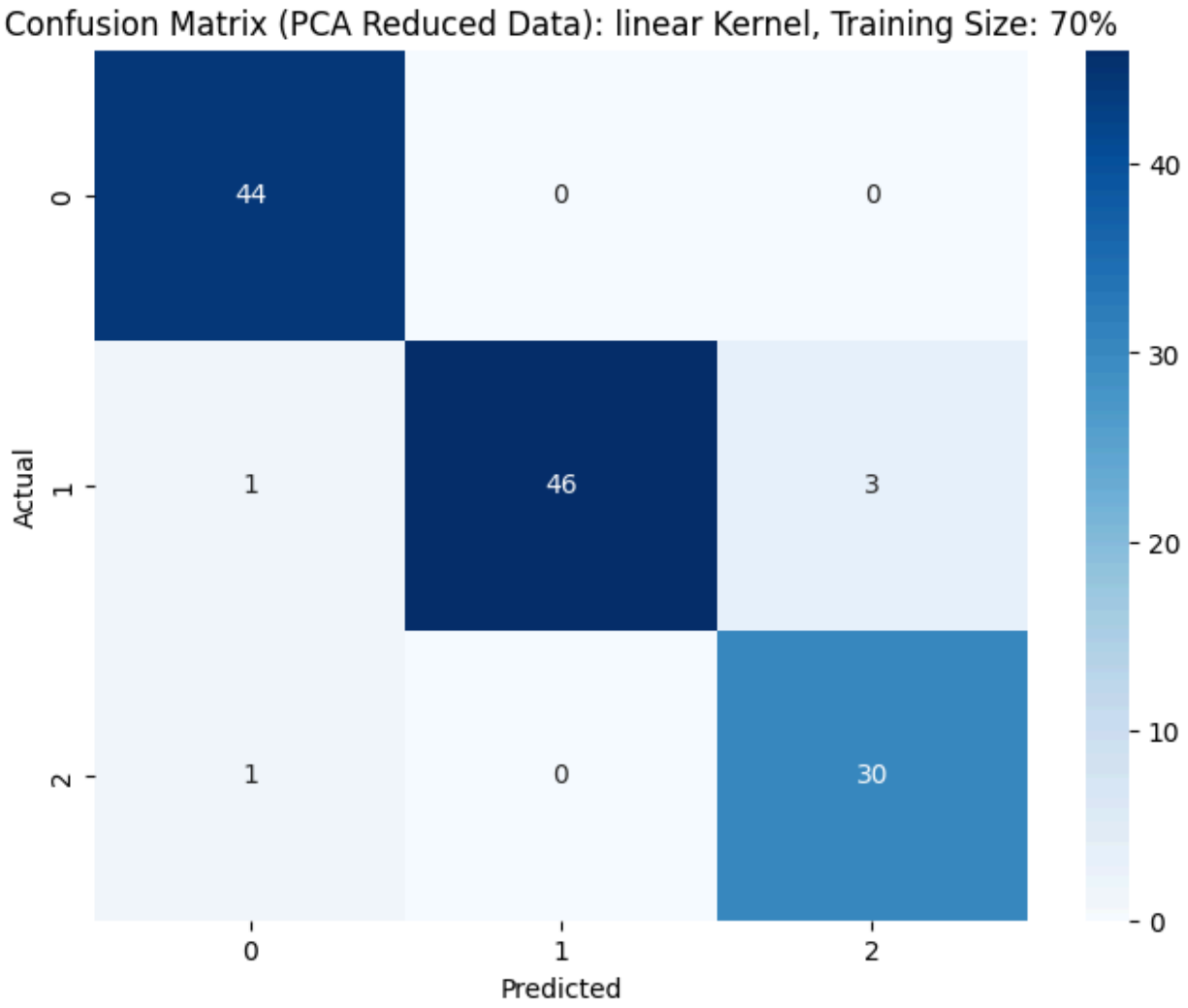


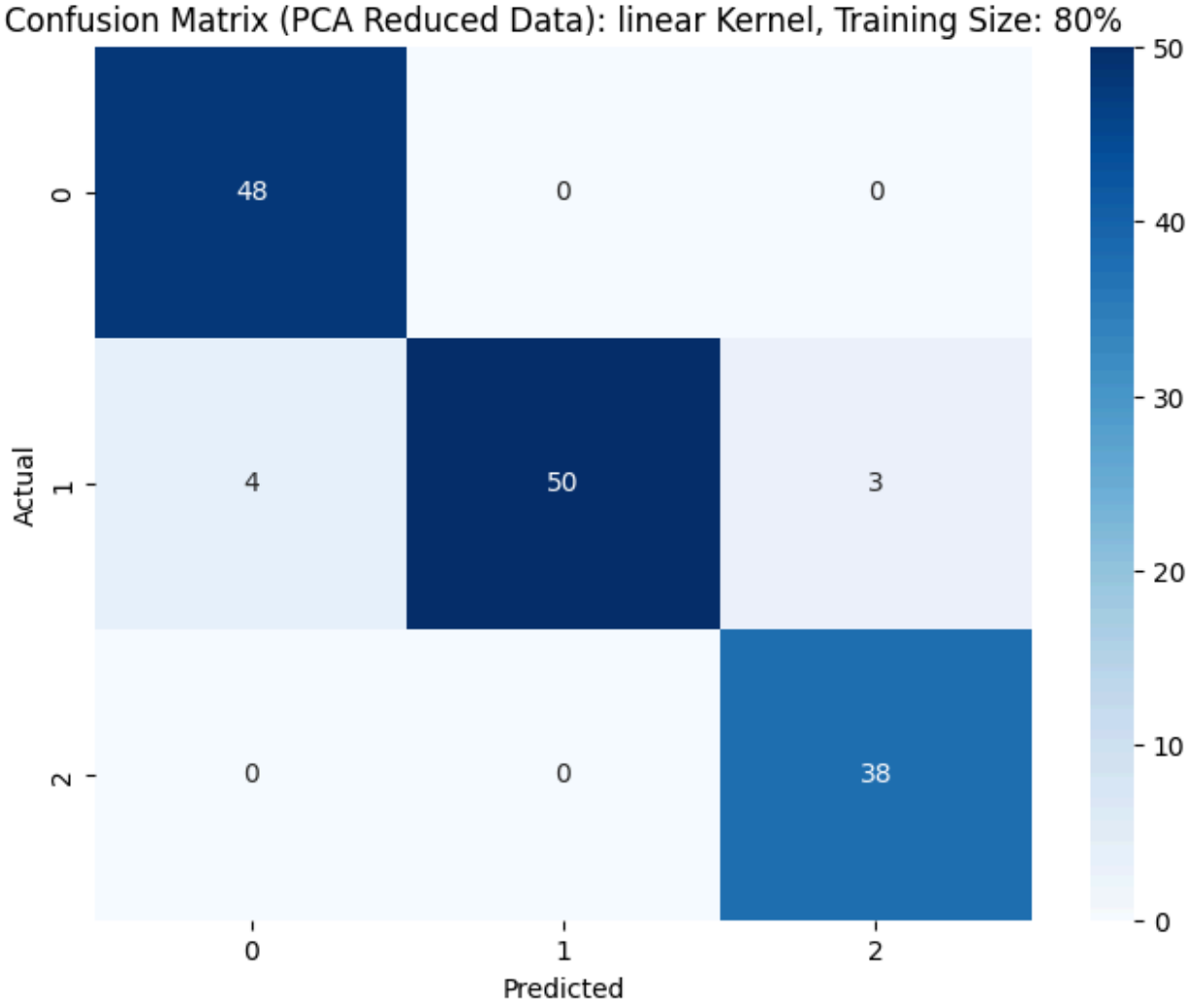
Show Confusion Matrices for PCA-Reduced Data

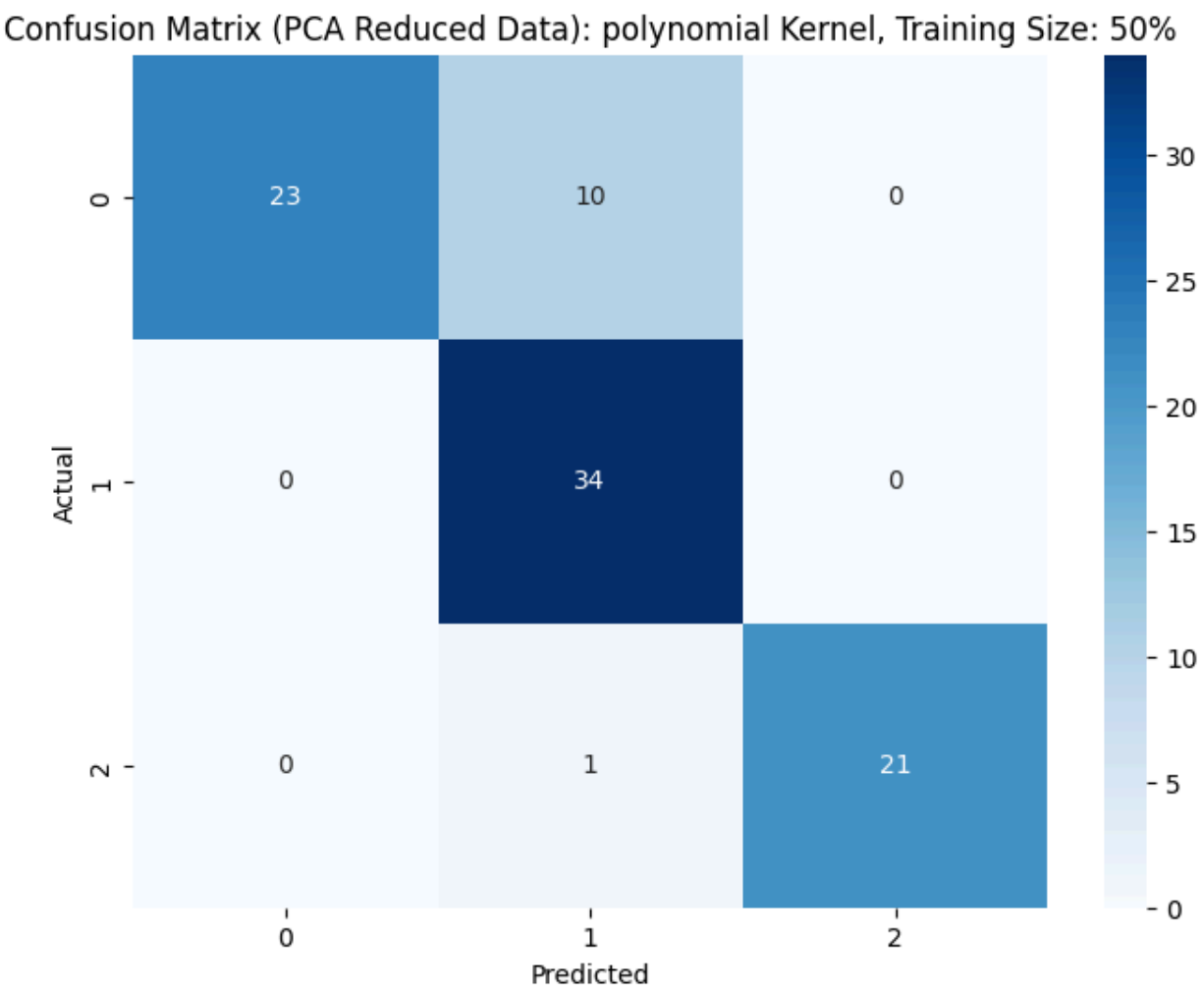
In [ ]:
```
for cm_data in confusion_matrices_pca:
    cm = cm_data["Confusion Matrix"]
    kernel_name = cm_data["Kernel"]
    training_size = cm_data["Training size"]

    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()
```

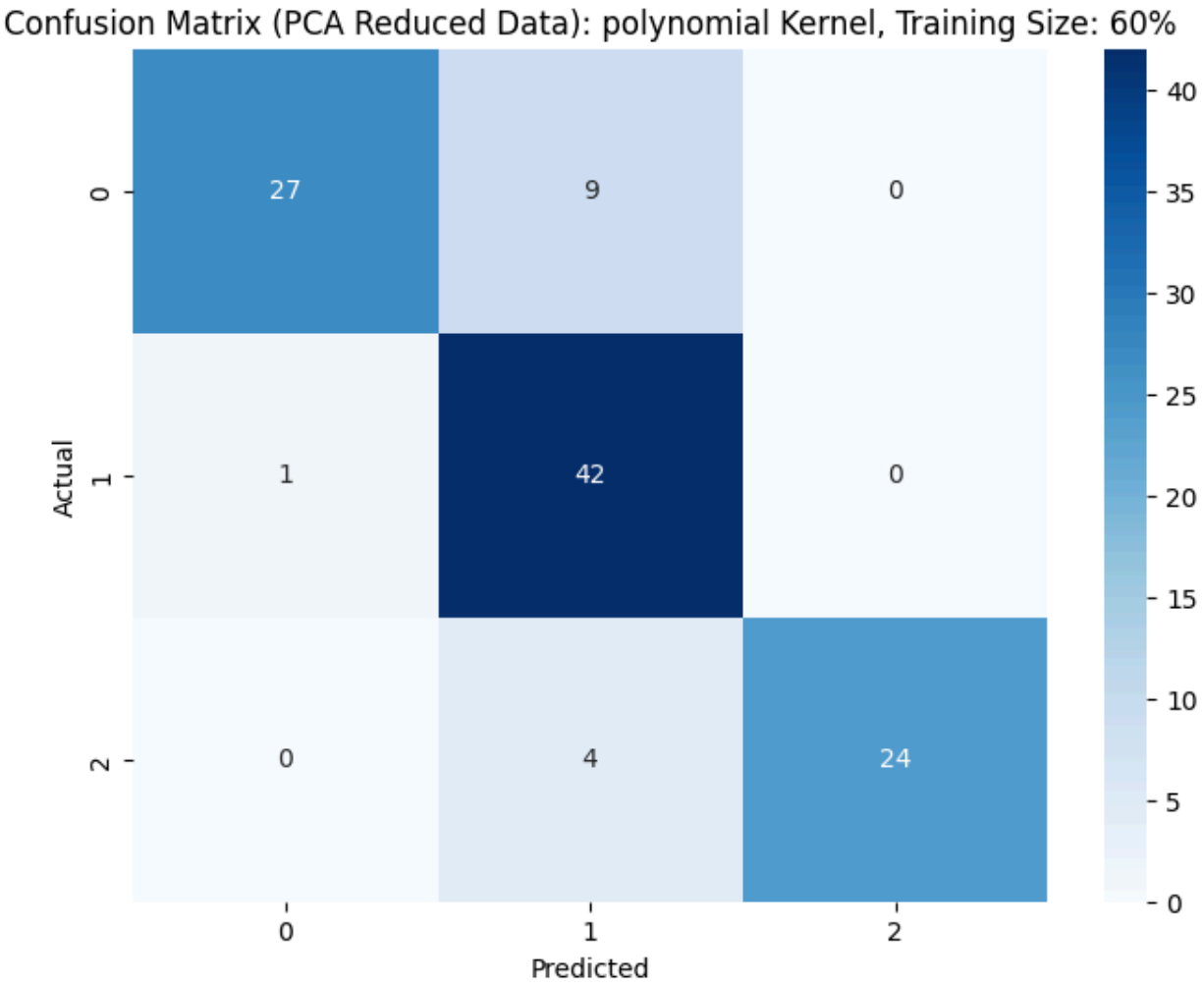

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 50%
33
0
0
0
34
0
0
1
21
Actual
0
1
2
Predicted
30
25
20
15
10
5
0

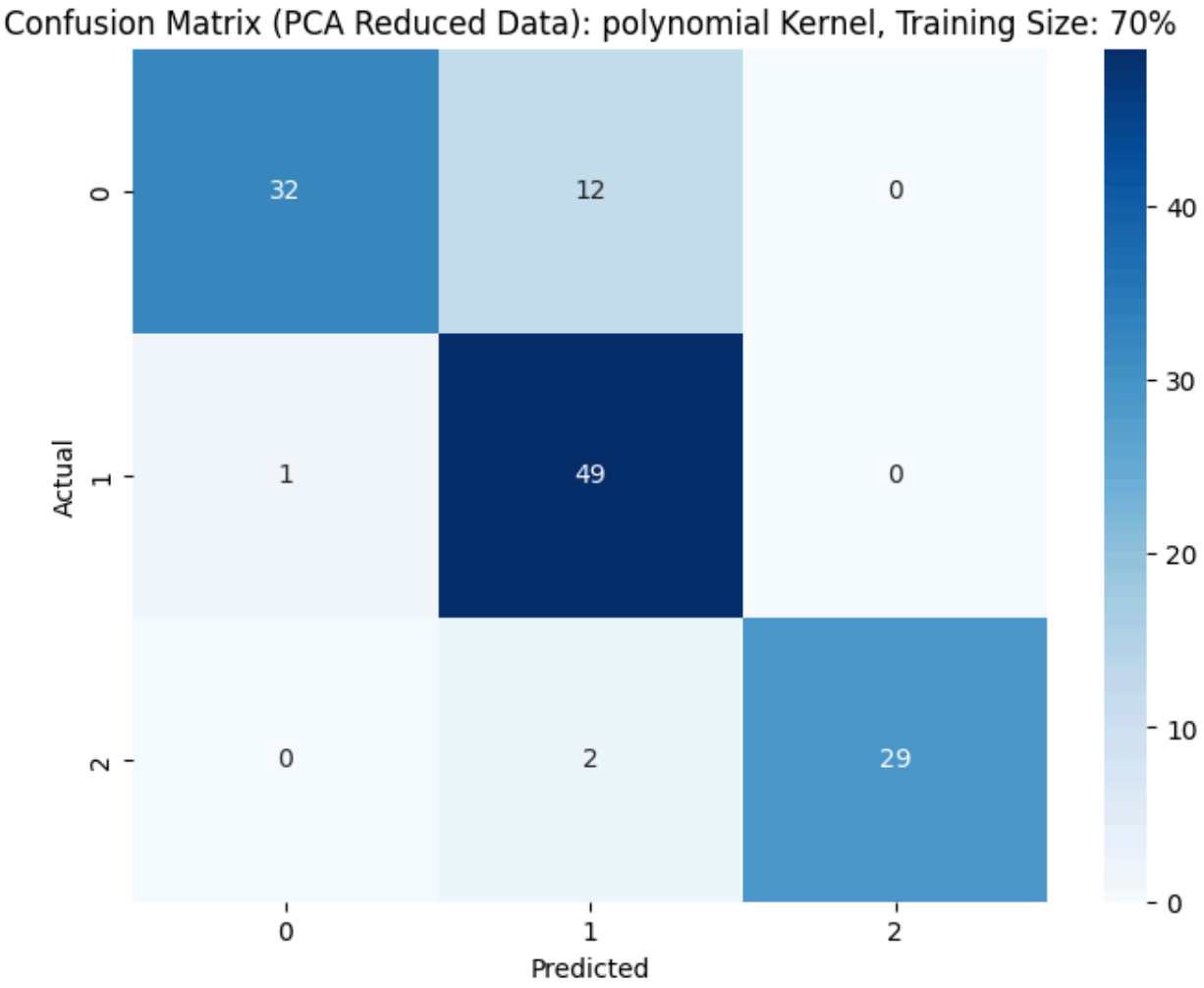

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 60%
36
0
0
2
39
2
1
0
27
Actual
0
1
2
Predicted
35
30
25
20
15
10
5
0

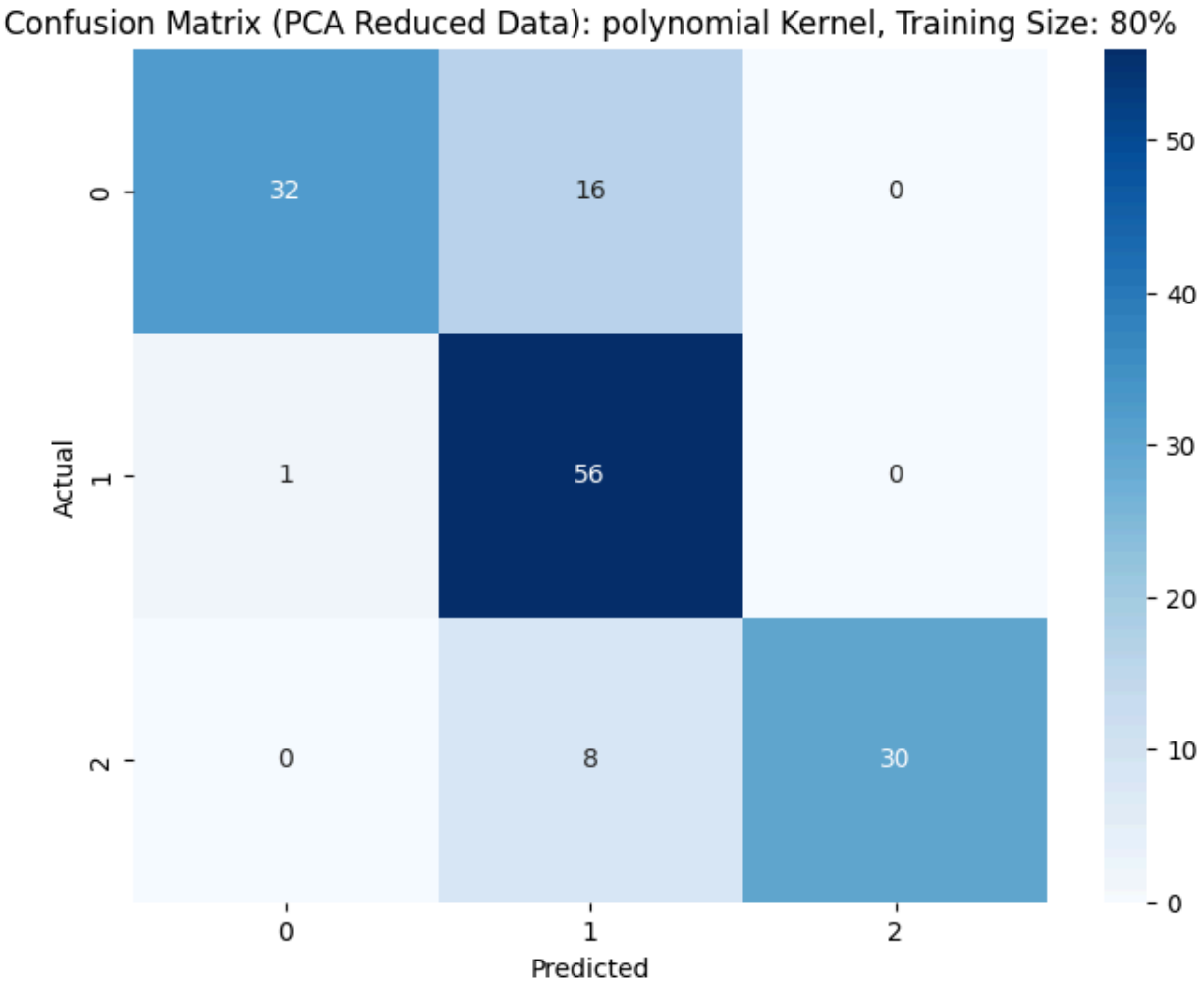

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 70%
Actual
Predicted
0
1
2
44
0
0
1
46
3
1
0
30
40
30
20
10
0

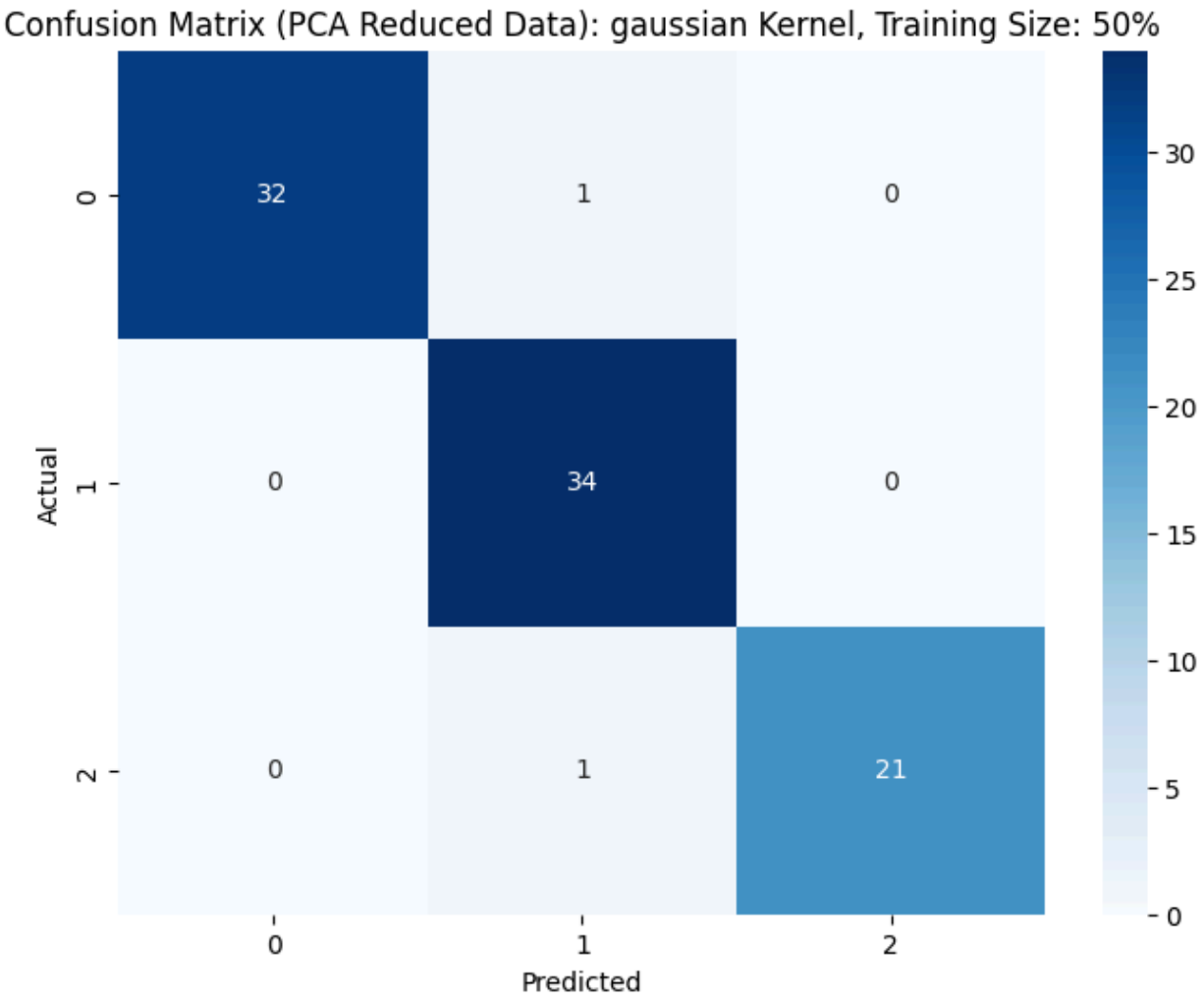

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 80%
48
0
0
4
50
3
0
0
38
Actual
Predicted
0
1
2
50
40
30
20
10
0

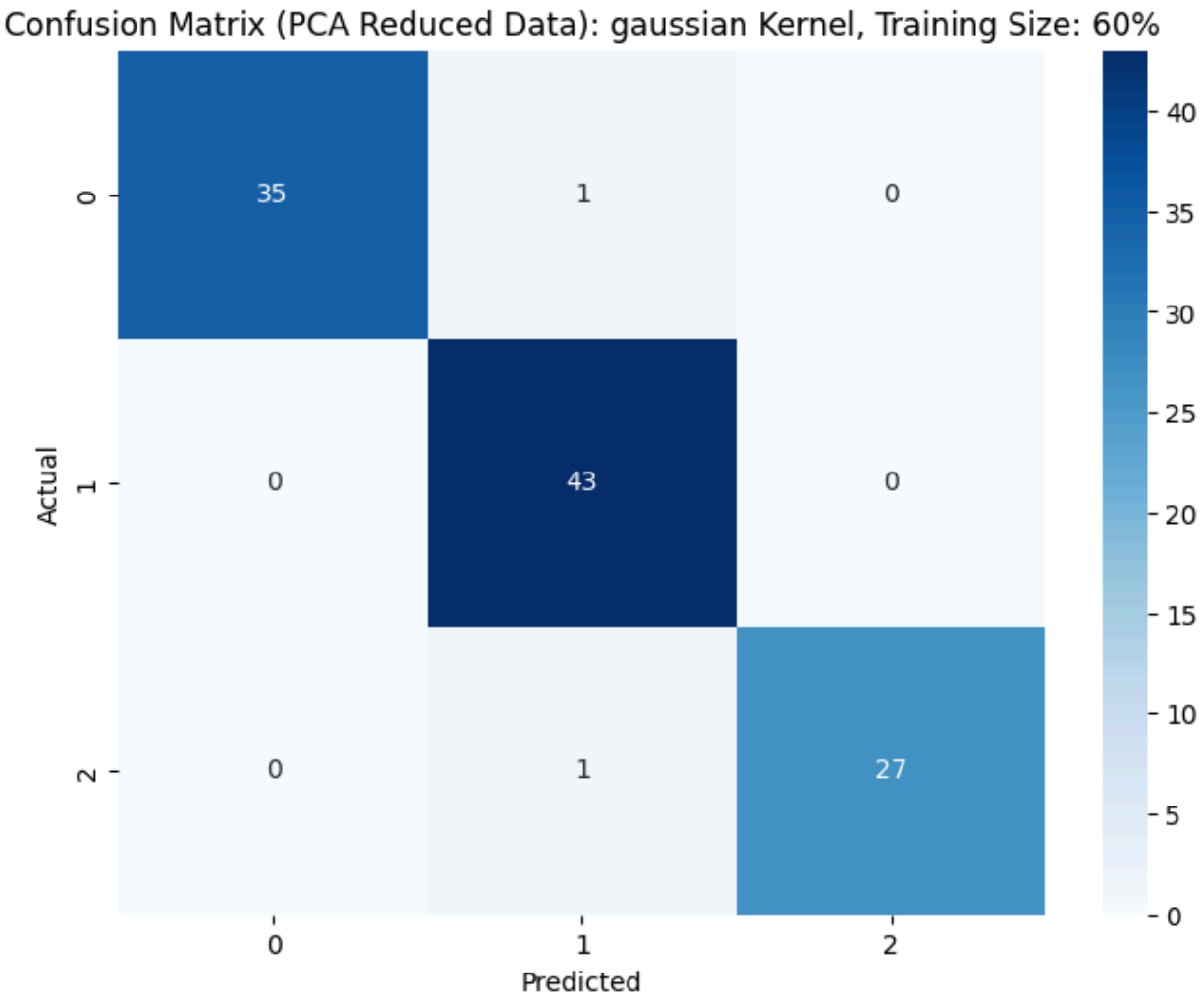

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 50%
Actual
Predicted
0
1
2
23
10
0
0
34
0
0
1
21
30
25
20
15
10
5
0

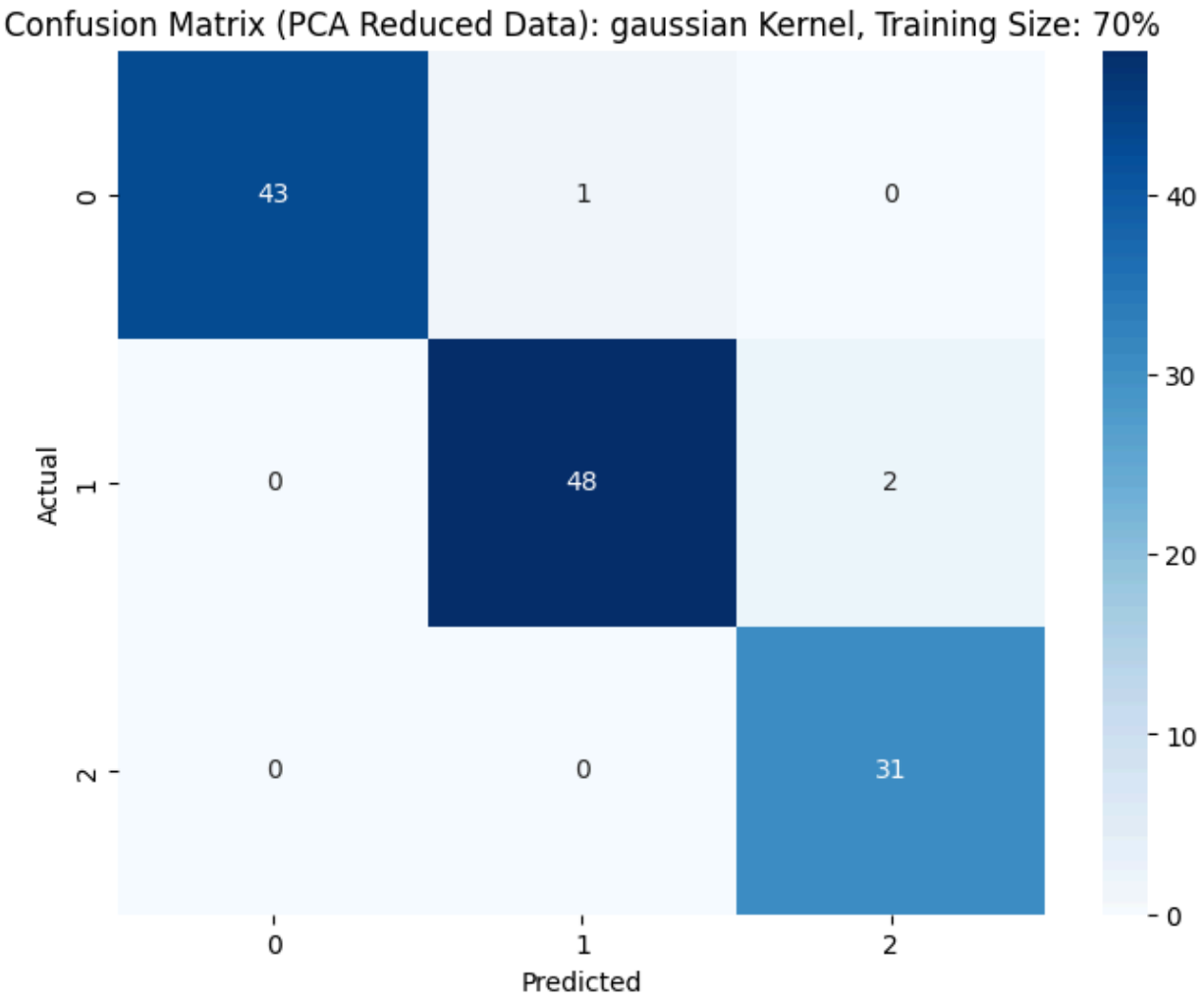

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 60%
27
9
0
1
42
0
0
4
24
Actual
0
1
2
Predicted
0
1
2
40
35
30
25
20
15
10
5
0

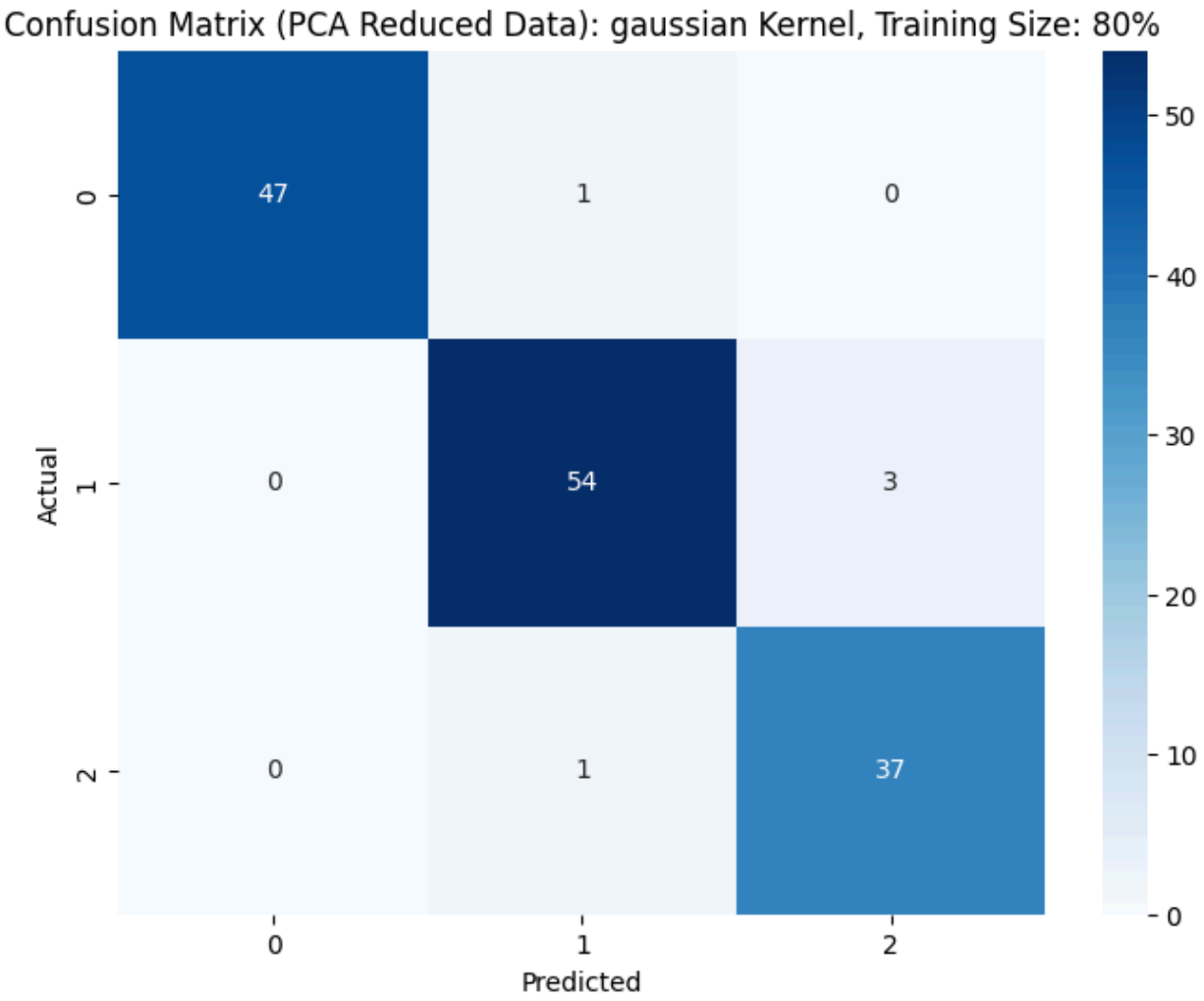

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 70%
32
12
0
1
49
0
0
2
29
Actual
0
1
2
Predicted
40
30
20
10
0

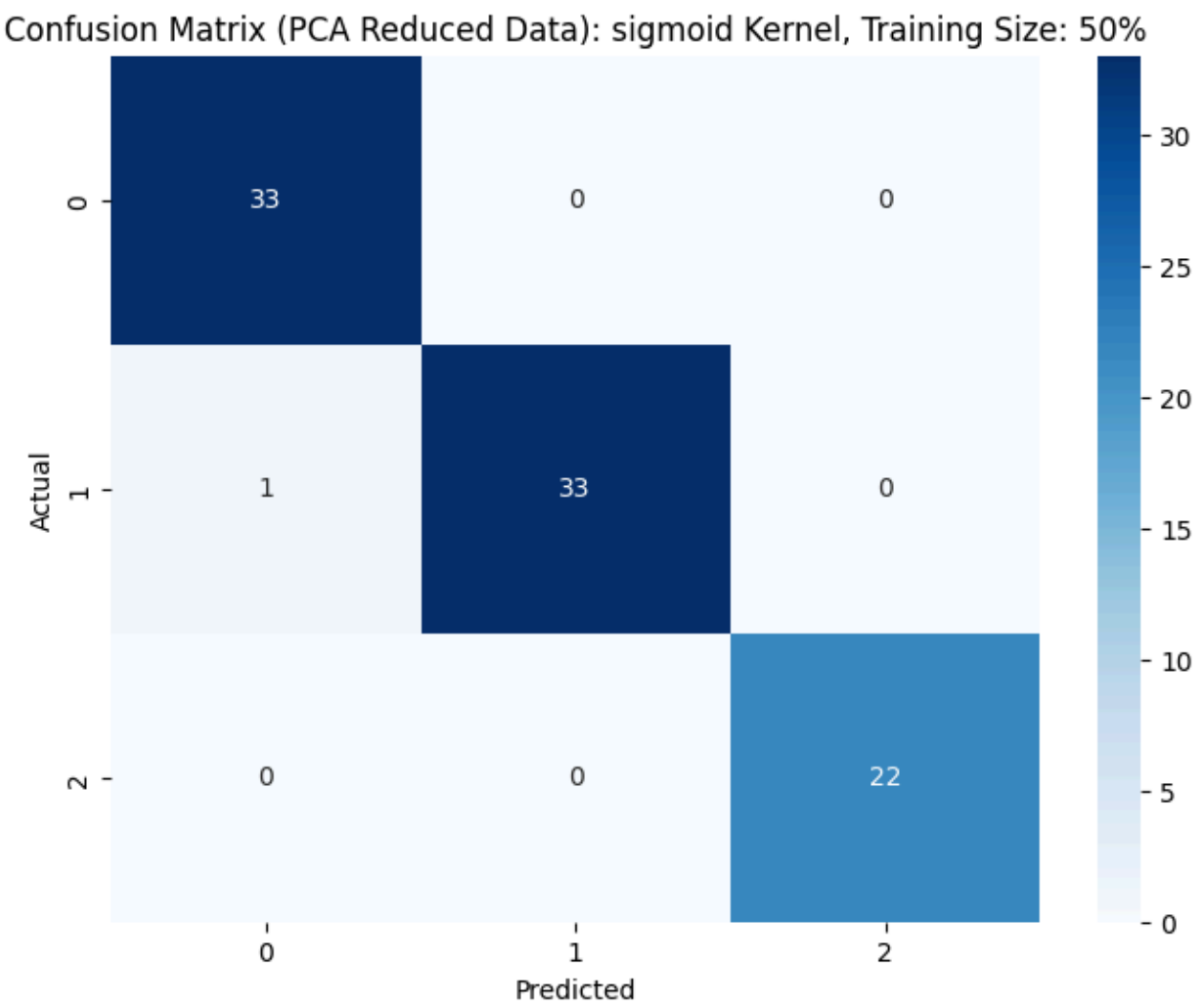

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 80%
32
16
0
1
56
0
0
8
30
Actual
0
1
2
Predicted
0
1
2
50
40
30
20
10
0

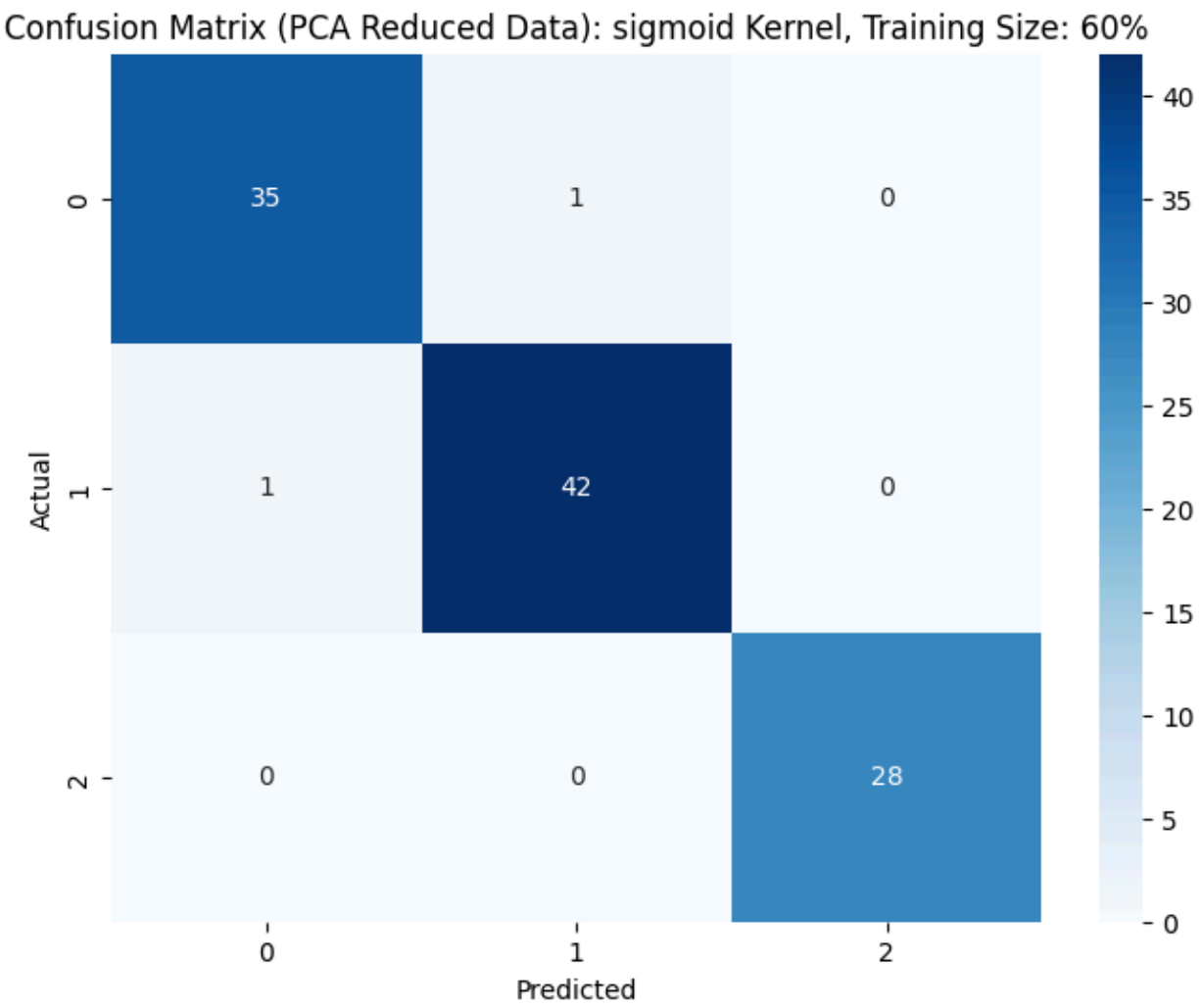

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 50%
Actual
Predicted
0
1
2
32
1
0
0
34
0
0
1
21
30
25
20
15
10
5
0

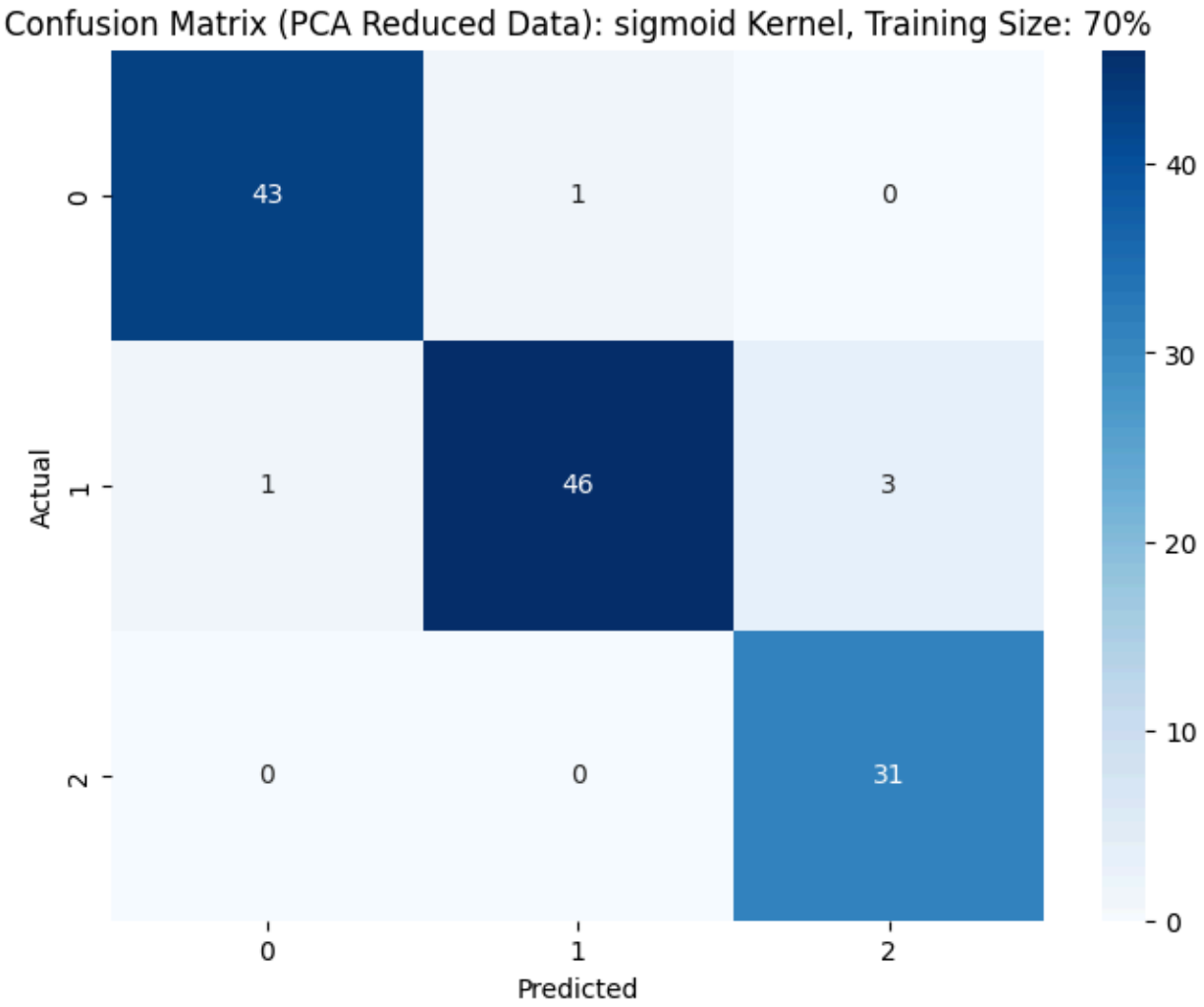

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 60%
Actual
Predicted
0
1
2
35
1
0
0
43
0
0
1
27
40
35
30
25
20
15
10
5
0

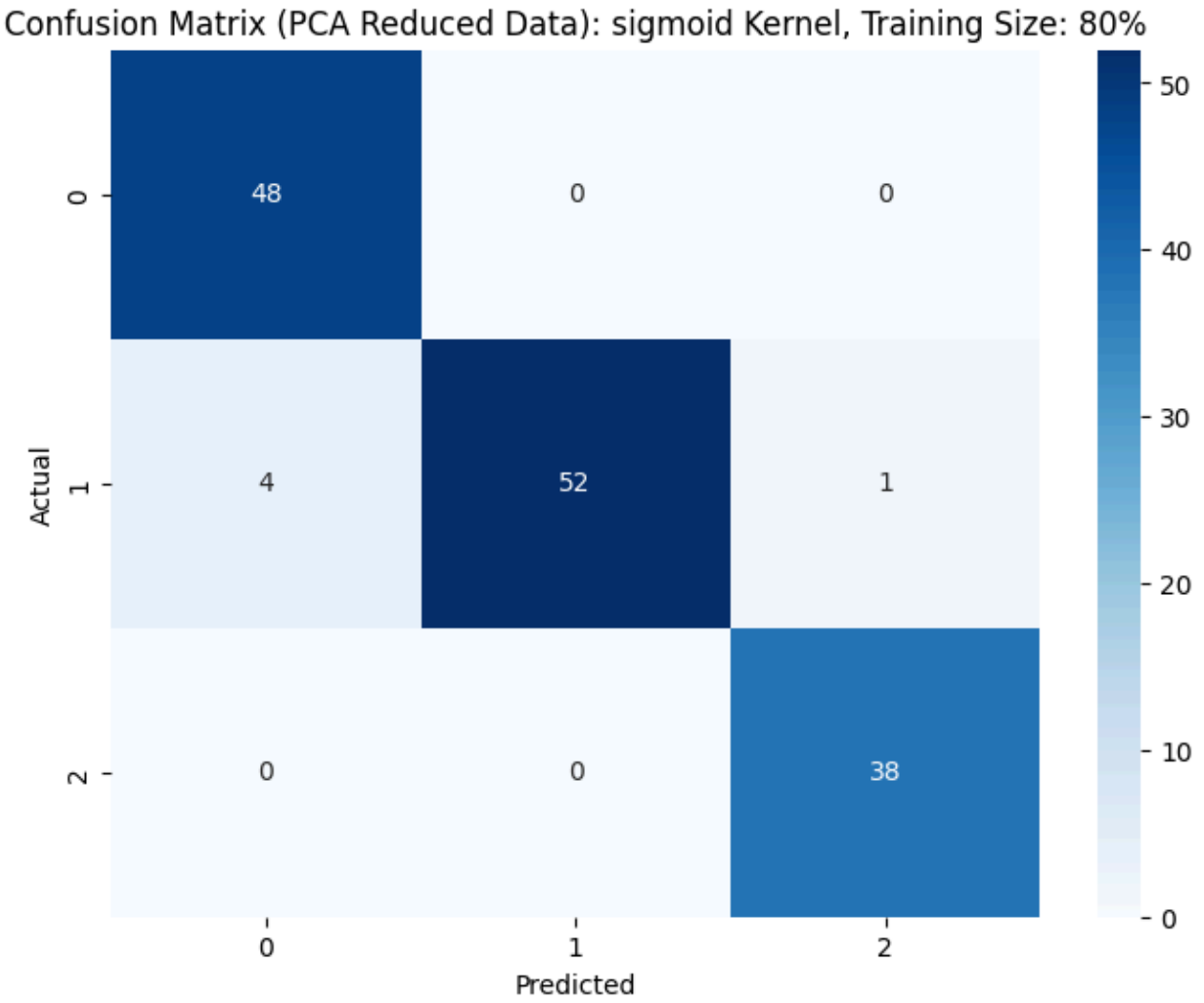

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 70%
43
1
0
0
48
2
0
0
31
Actual
0
1
2
Predicted
40
30
20
10
0


Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 80%
47
1
0
0
54
3
0
1
37
Actual
0
1
2
Predicted
50
40
30
20
10
0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 50%
Actual
Predicted
0
1
2
33
0
0
1
33
0
0
0
22
30
25
20
15
10
5
0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 60%
35
1
0
1
42
0
0
0
28
Actual
0
1
2
Predicted
40
35
30
25
20
15
10
5
0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 70%
43
1
0
1
46
3
0
0
31
Actual
0
1
2
Predicted
40
30
20
10
0



===== 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
    })
```

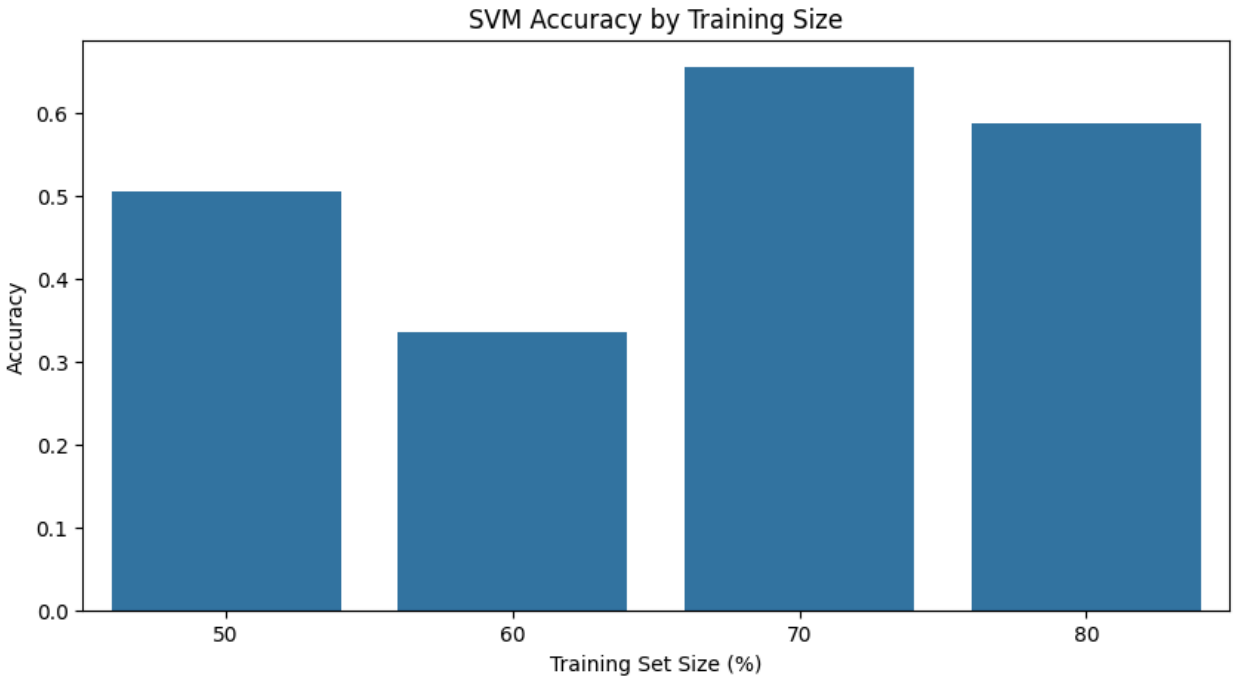
Print Evalution Table and Graphs

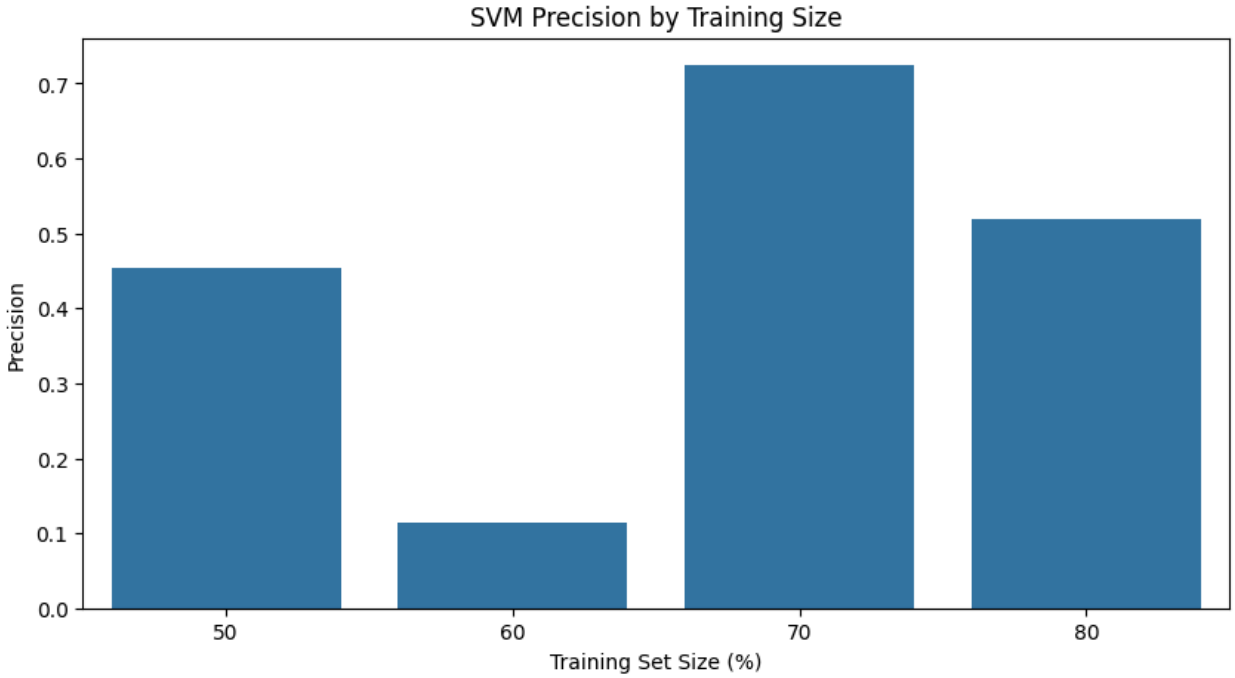
In [ ]:

```
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	0.505618	0.454262	0.505618	0.441859
1	60	0.336449	0.113198	0.336449	0.169401
2	70	0.656000	0.724503	0.656000	0.678092
3	80	0.587413	0.519894	0.587413	0.505192



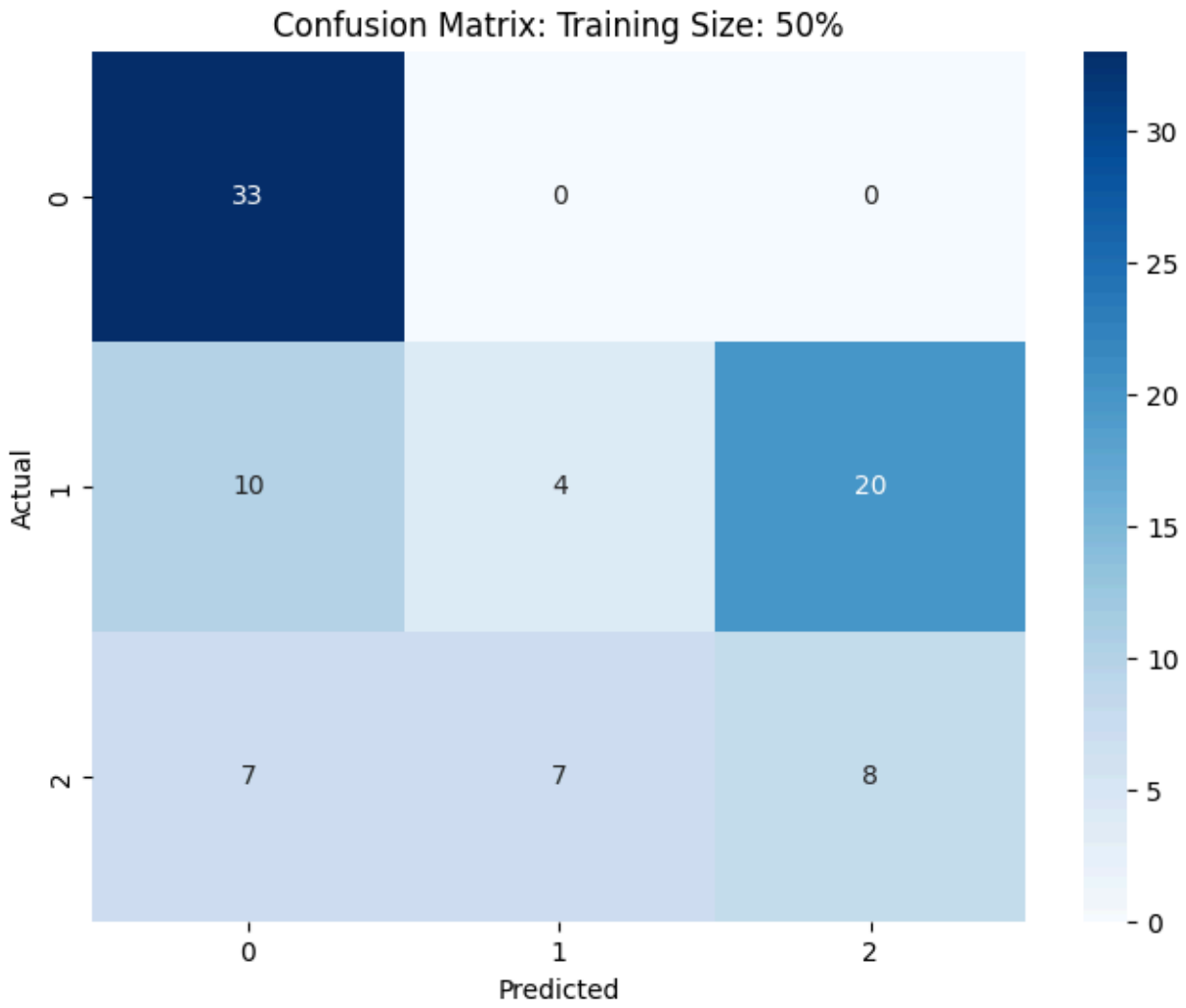

SVM Precision by Training Size
Precision
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
50
60
70
80
Training Set Size (%)

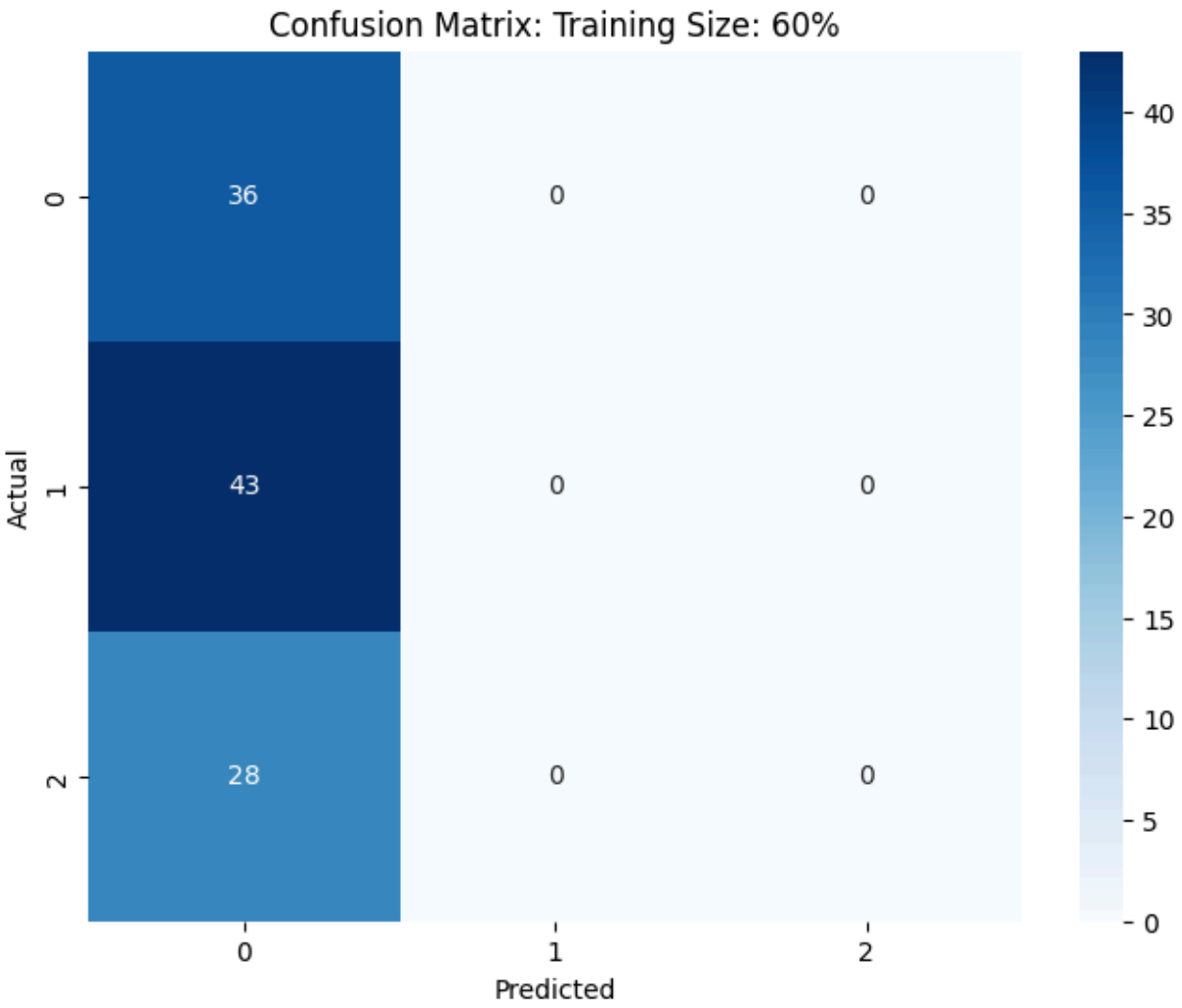

SVM Recall by Training Size
Recall
0.0
0.1
0.2
0.3
0.4
0.5
0.6
50
60
70
80
Training Set Size (%)

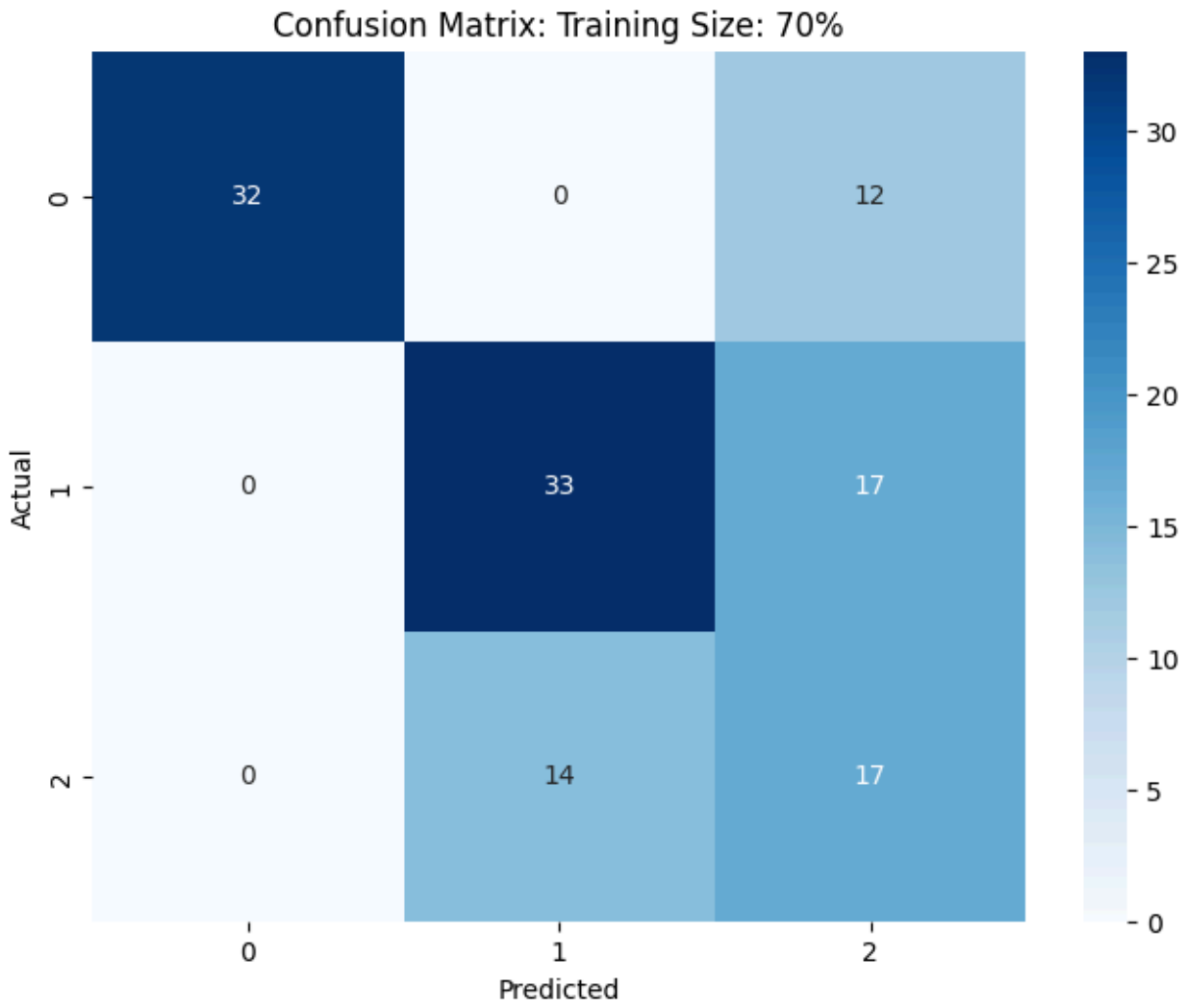
Show Confusion Matrices

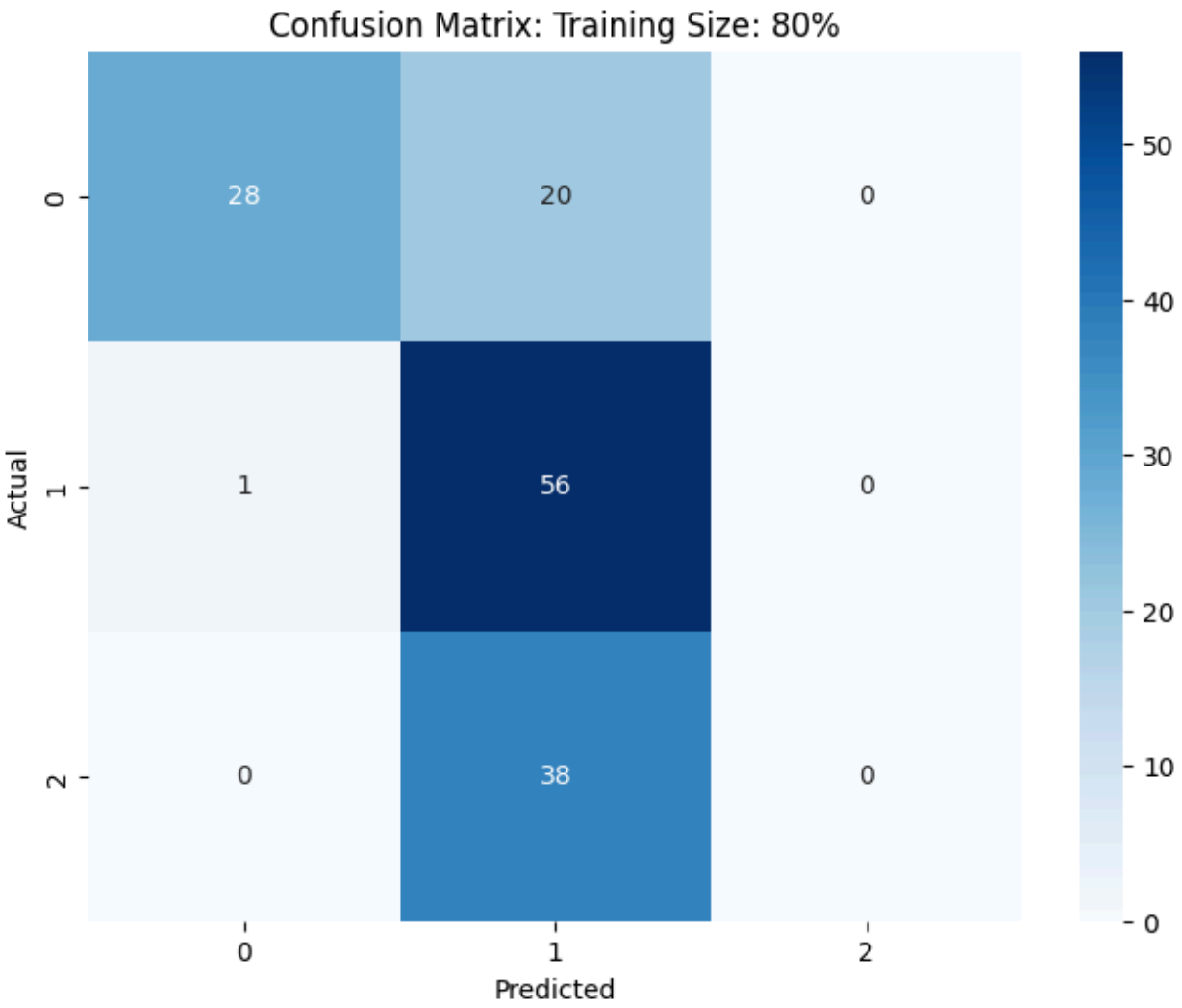
In [ ]:

```
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%
33
0
0
10
4
20
7
7
8
Actual
Predicted
0
1
2
0
1
2
30
25
20
15
10
5
0

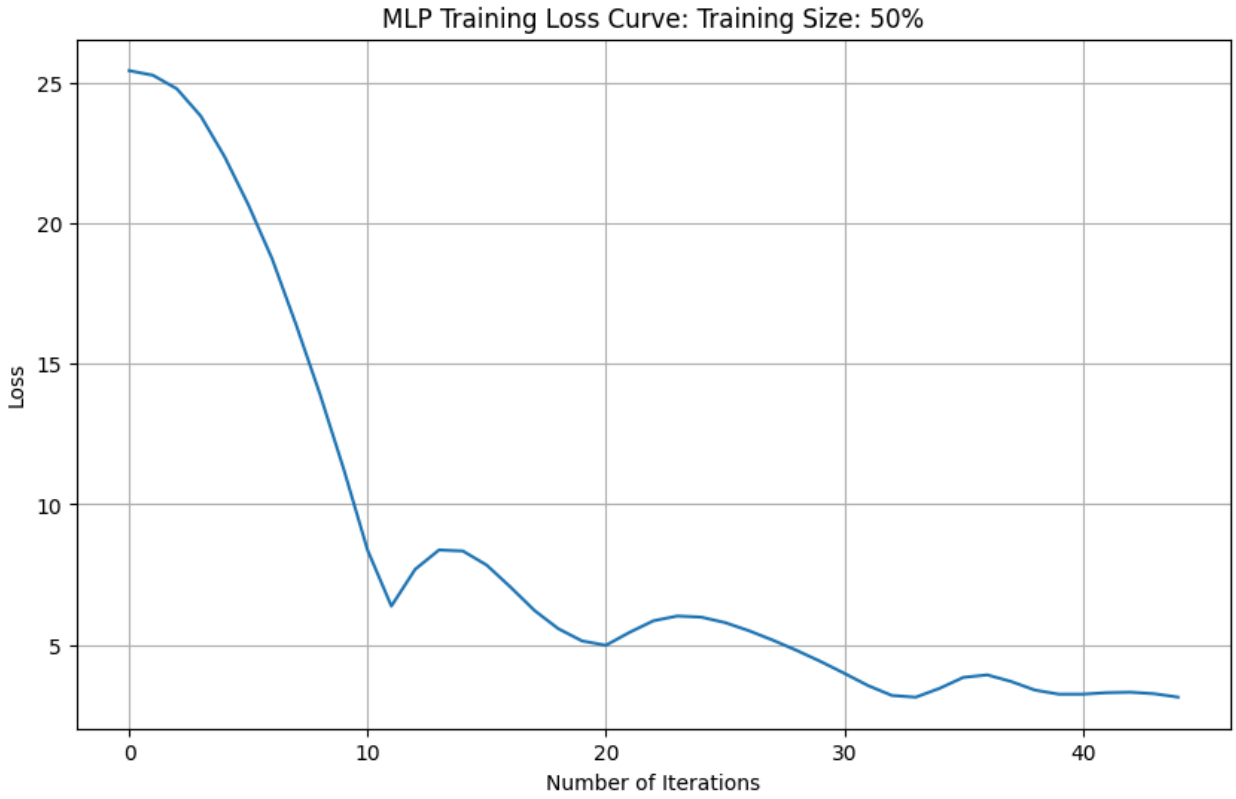

Confusion Matrix: Training Size: 60%
36
0
0
43
0
0
28
0
0
Actual
0
1
2
Predicted
40
35
30
25
20
15
10
5
0

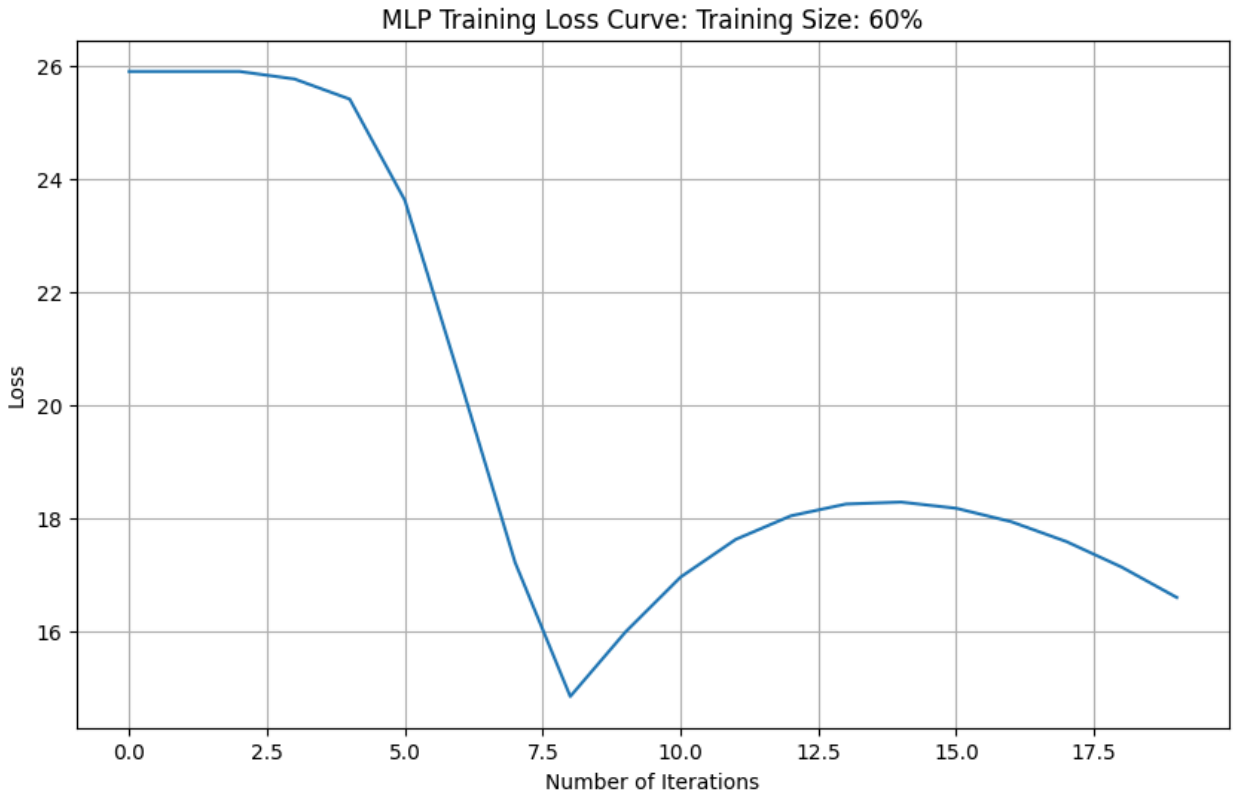

Confusion Matrix: Training Size: 70%
32
0
12
0
33
17
0
14
17
Actual
Predicted
0
1
2
30
25
20
15
10
5
0

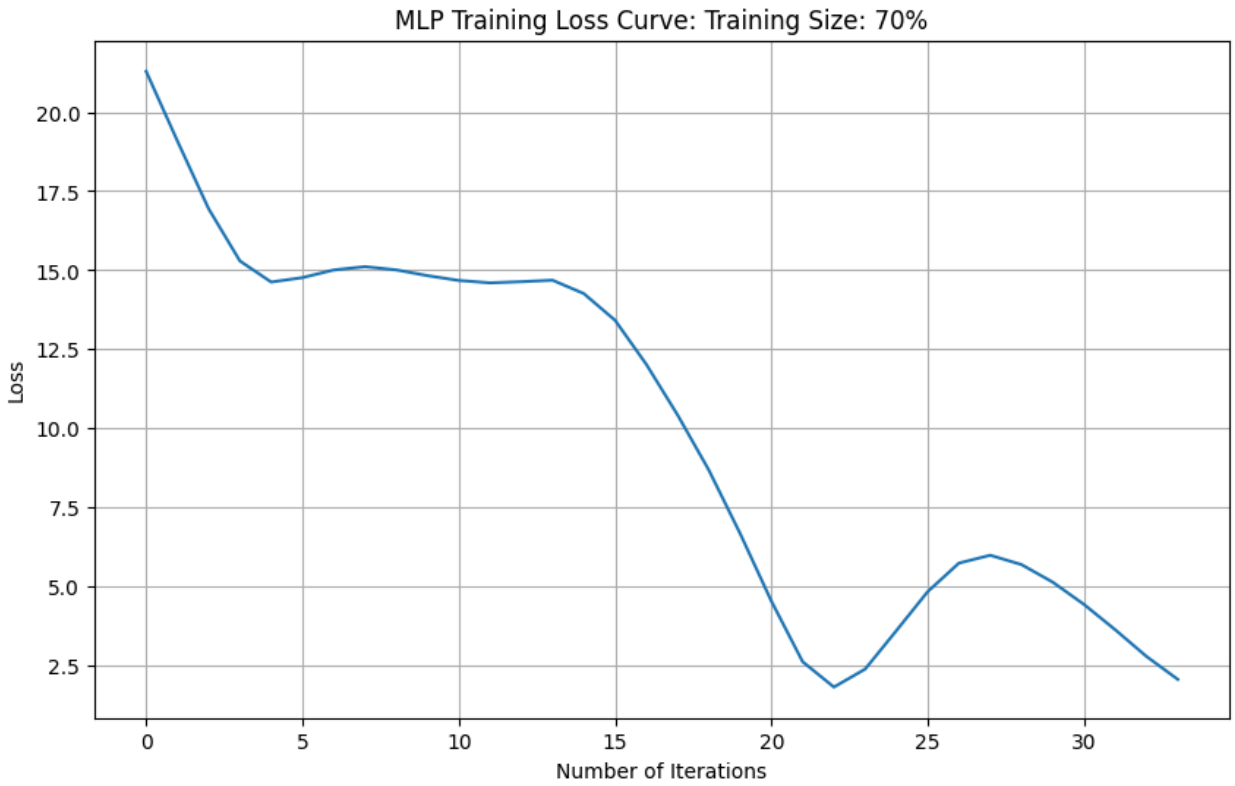


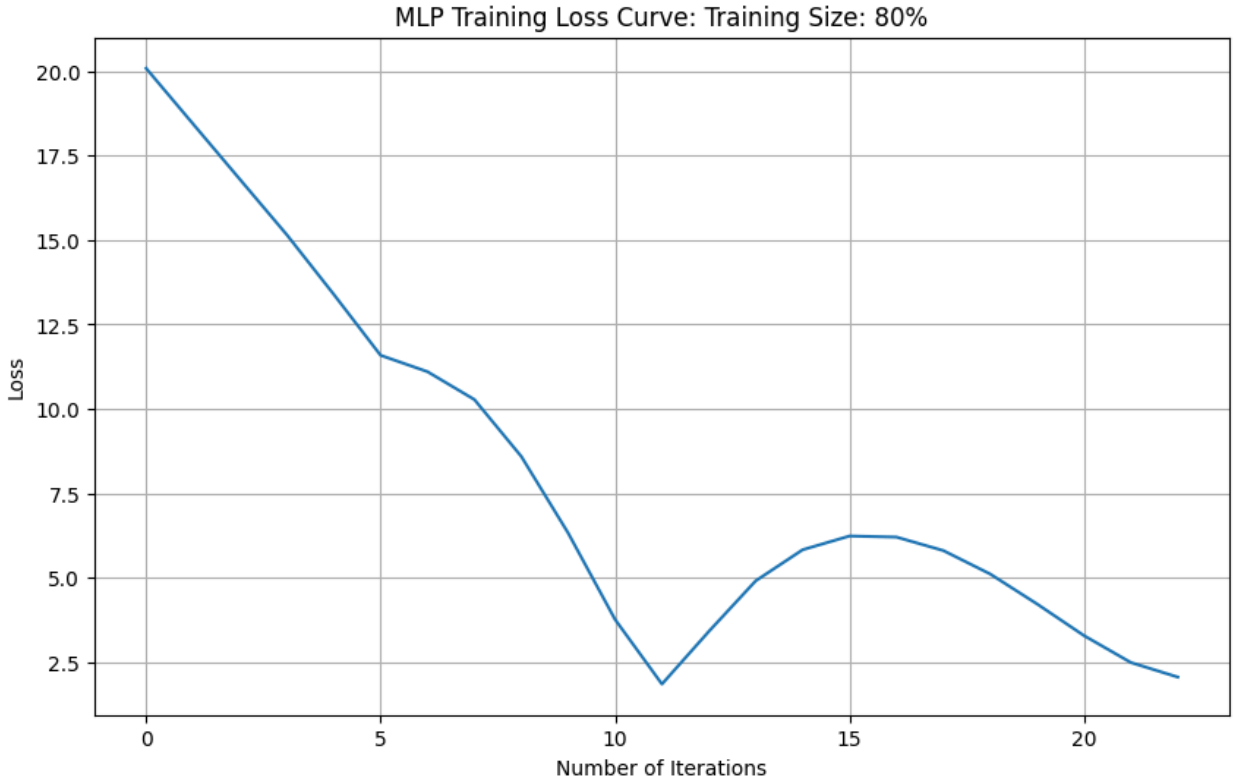
Training and Loss generation curve

In [ ]:
```python
# 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%
Loss
Number of Iterations
25
20
15
10
5
0
10
20
30
40


MLP Training Loss Curve: Training Size: 60%
Loss
Number of Iterations
26
24
22
20
18
16
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

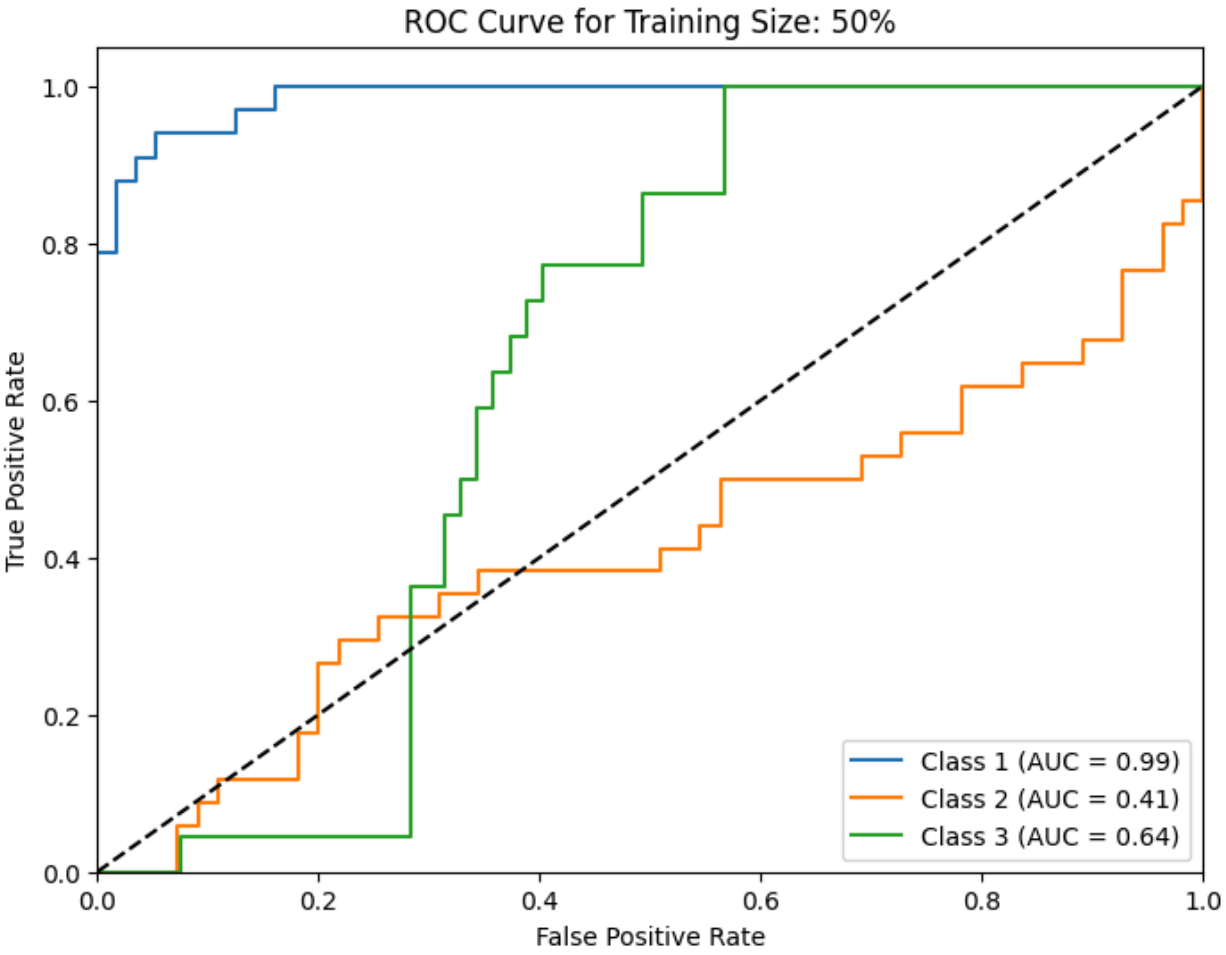


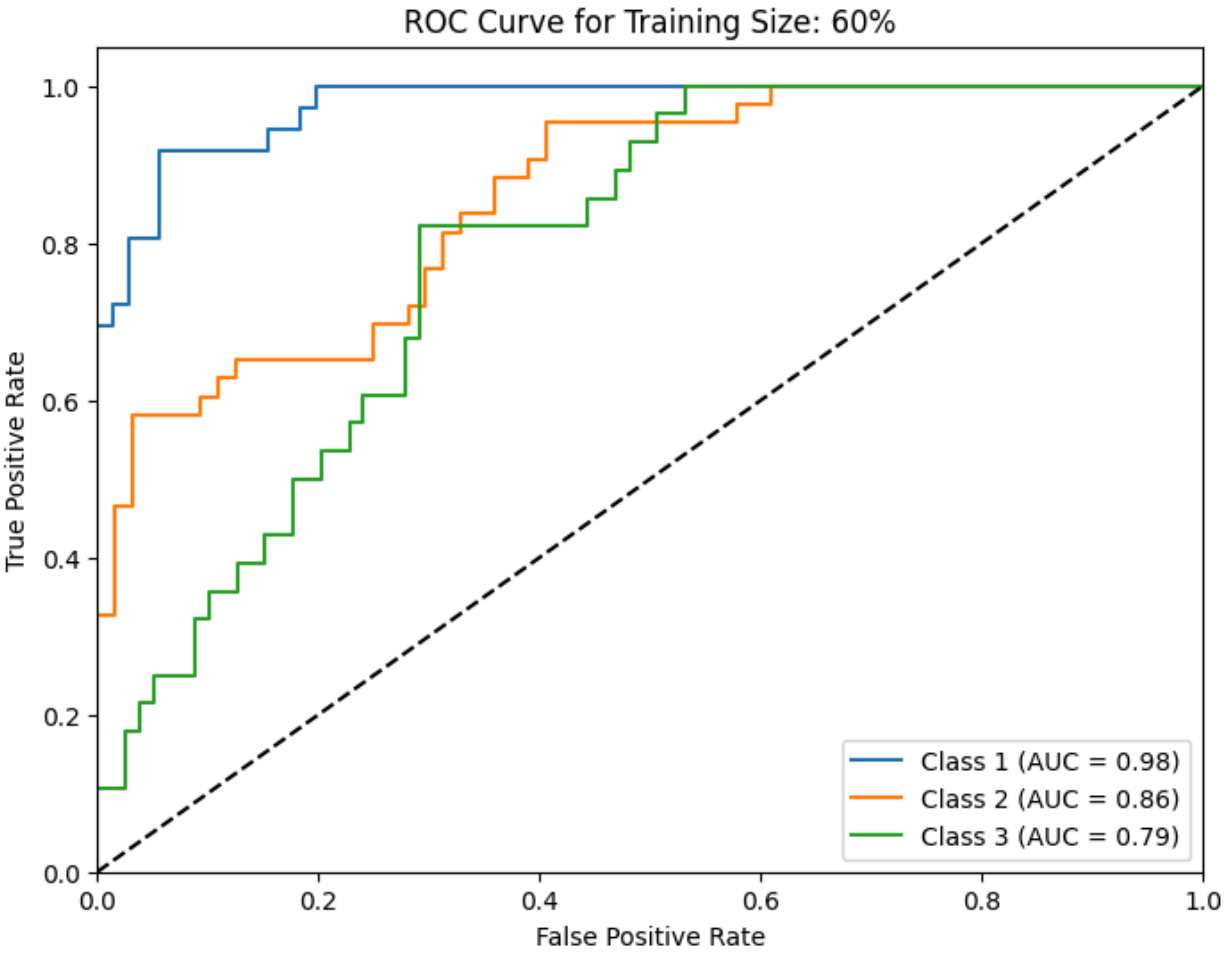


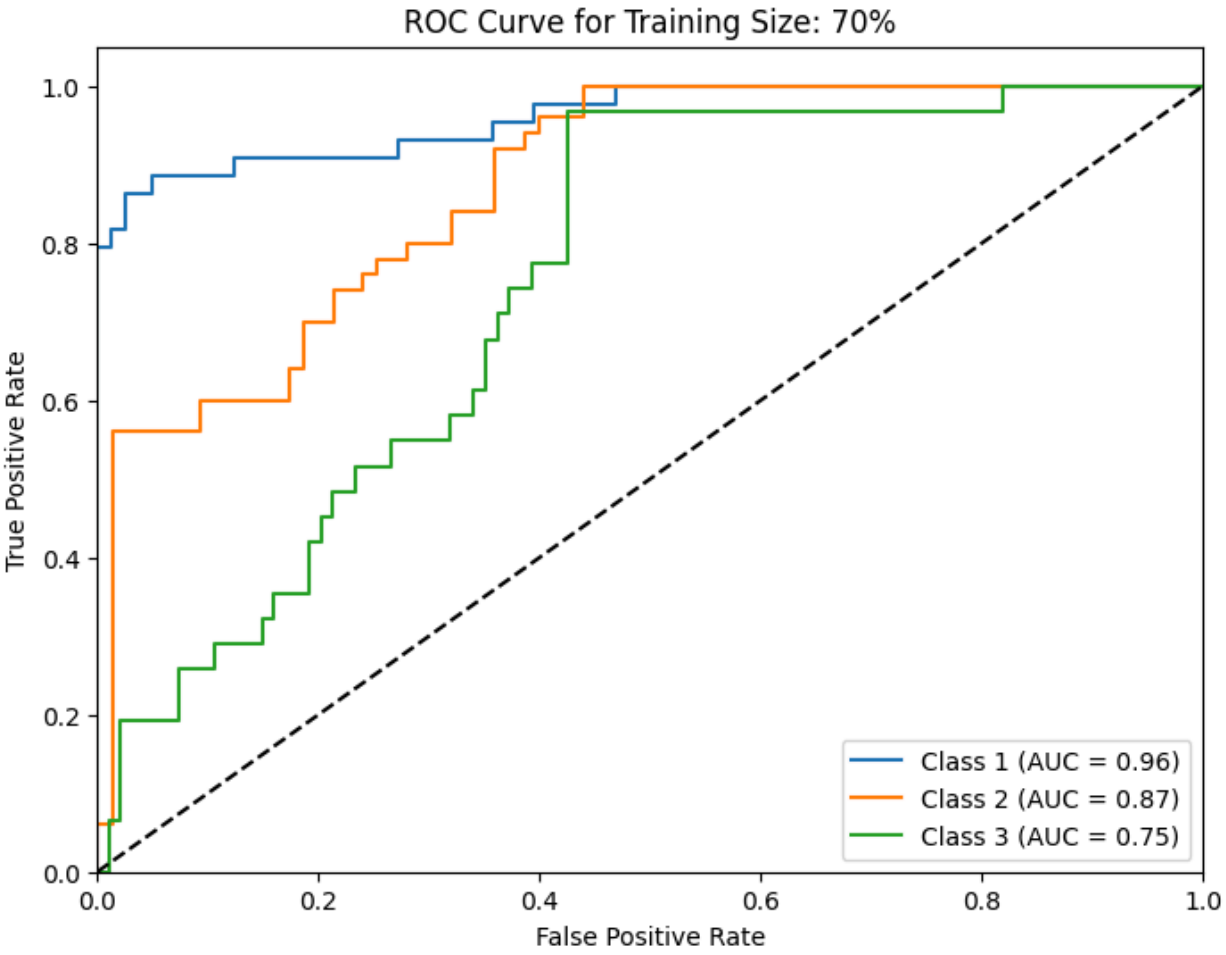
ROC and AOC curve

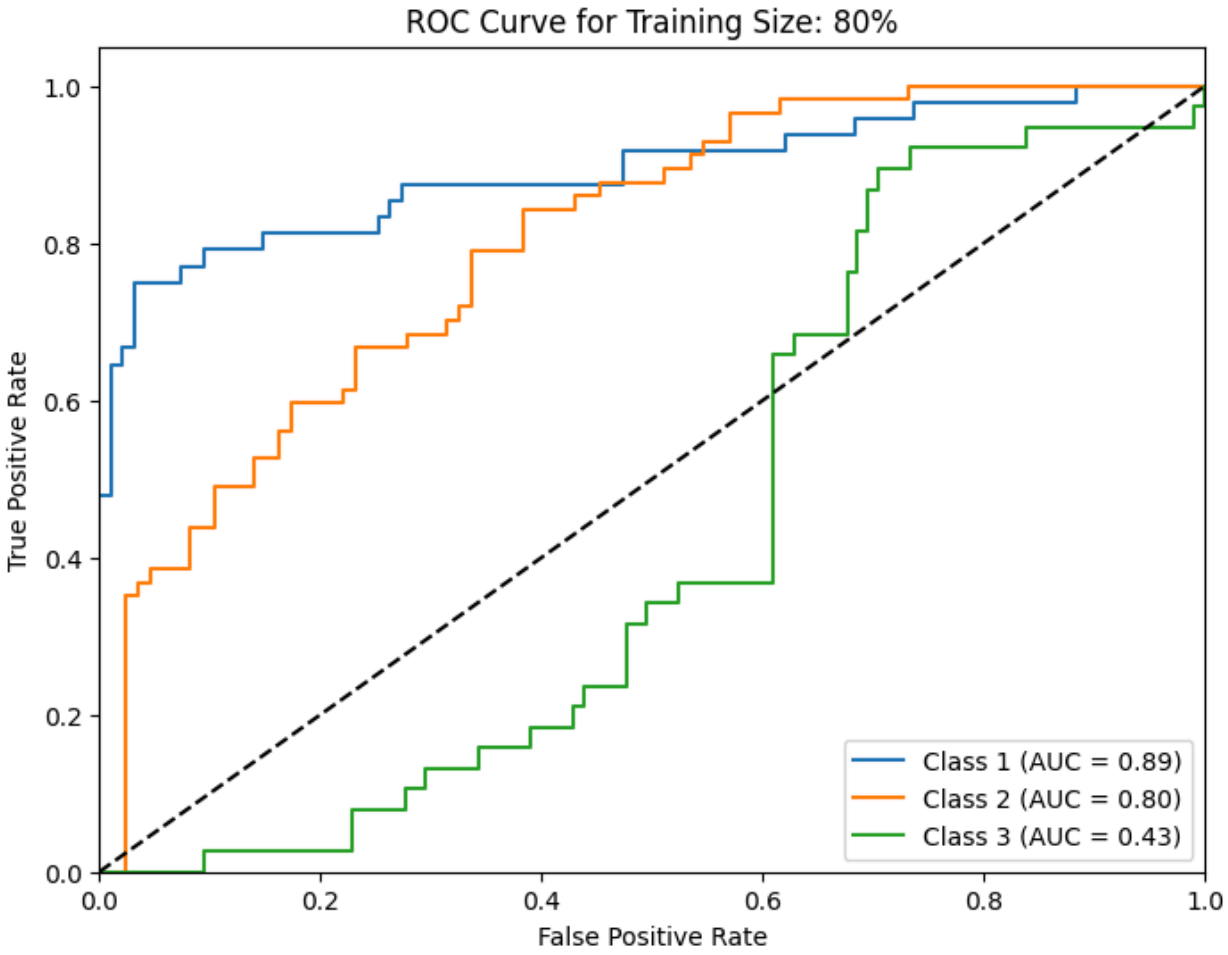
In [ ]:
```
from sklearn.metrics import roc_curve, auc
```

```python
# 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 fo
    plt.legend(loc="lower right")
    plt.show()
```


ROC Curve for Training Size: 50%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.99)
Class 2 (AUC = 0.41)
Class 3 (AUC = 0.64)


ROC Curve for Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.98)
Class 2 (AUC = 0.86)
Class 3 (AUC = 0.79)


ROC Curve for Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 0.96)
Class 2 (AUC = 0.87)
Class 3 (AUC = 0.75)



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:
```

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

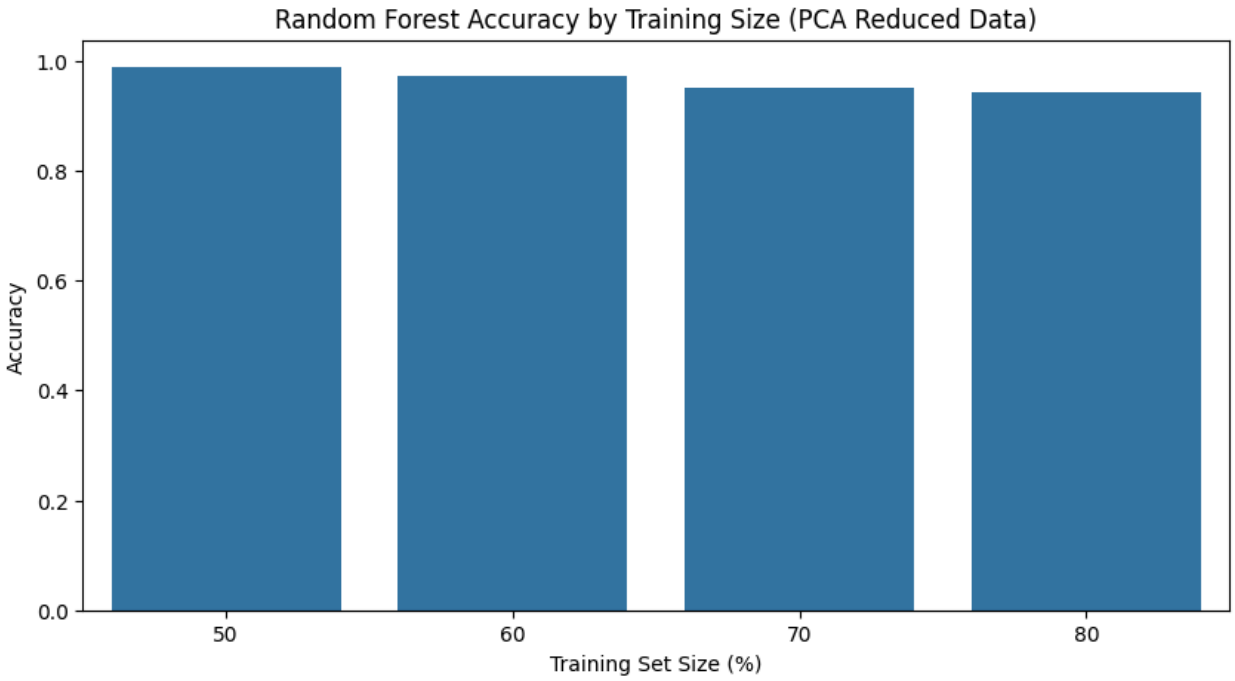
Calculate MLP performance with PCA-Reduced Data for different test sizes

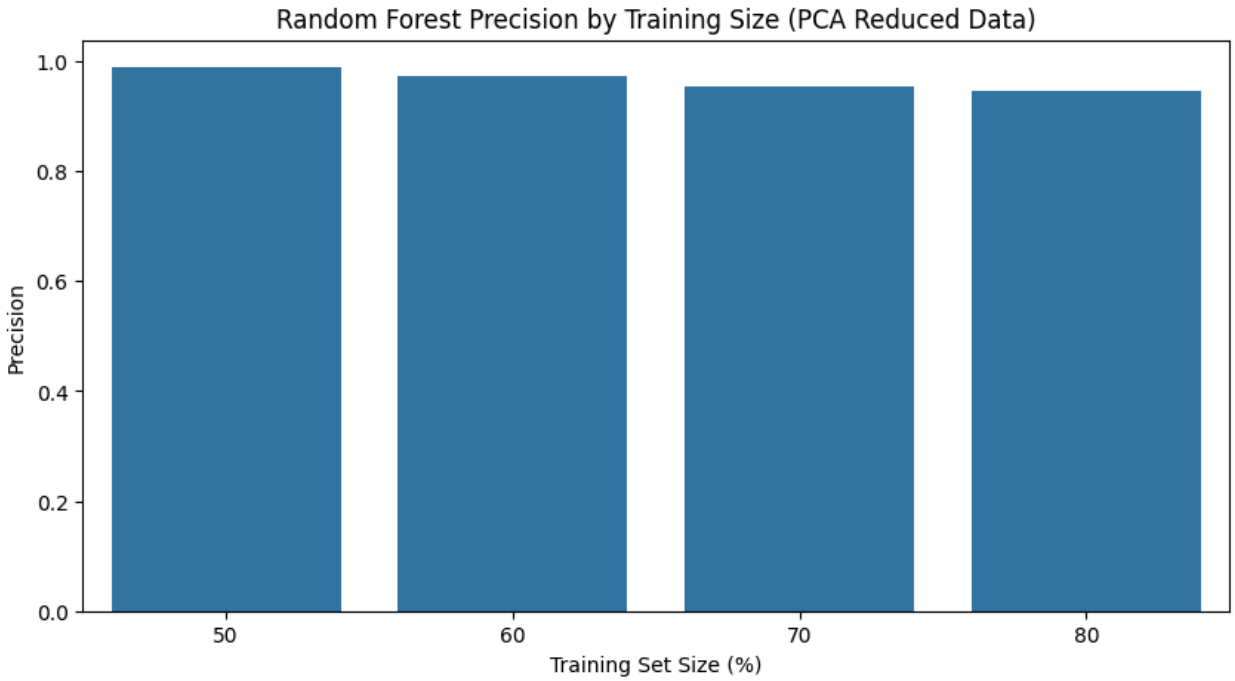
In [ ]:
```
results_mlp_pca = []
confusion_matrices_mlp_pca = []
trained_models_mlp_pca = []


for size in training_sizes:
  X_train_pca_mlp, X_test_pca_mlp, y_train, y_test = train_test_split(X_pca_re
  mlp_pca = MLPClassifier(max_iter=1500, momentum=0.7, learning_rate_init=0.00

  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='weighte
  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
  })
```

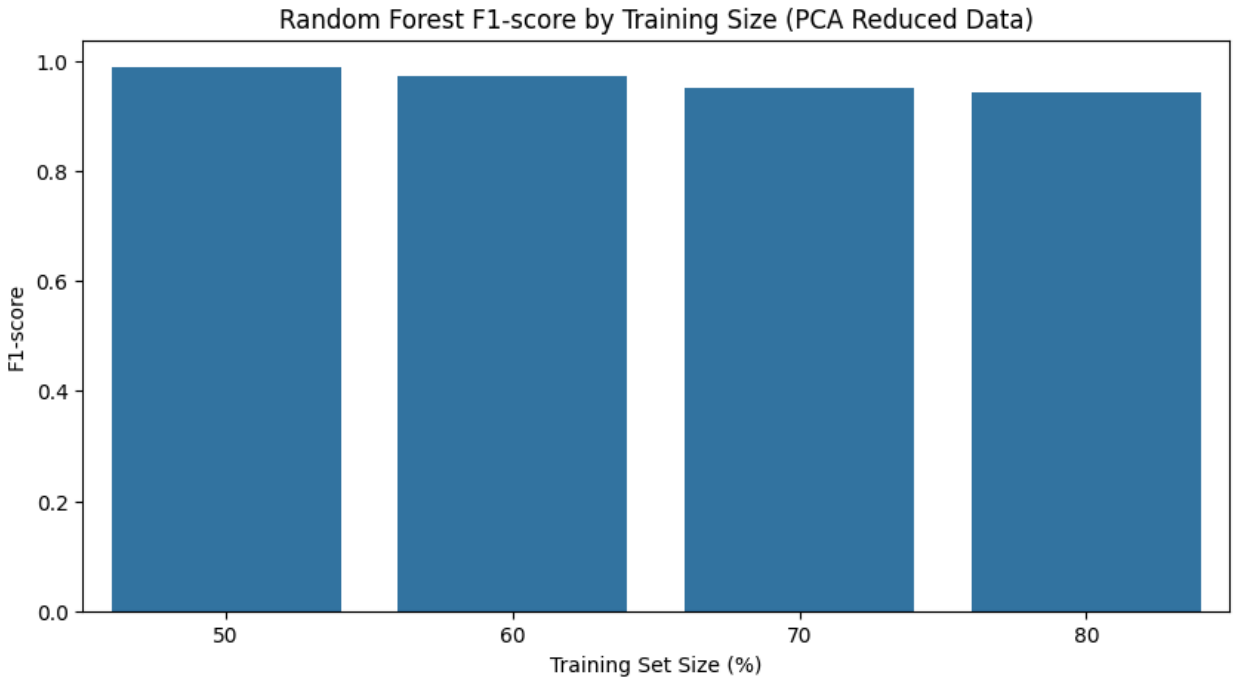
Print MLP Performance Table and Graphs with PCA-Reduced Data

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	0.973461	0.971963	0.971842
2	70	0.952000	0.954961	0.952000	0.951970
3	80	0.944056	0.946736	0.944056	0.943487


Random Forest Accuracy by Training Size (PCA Reduced Data)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

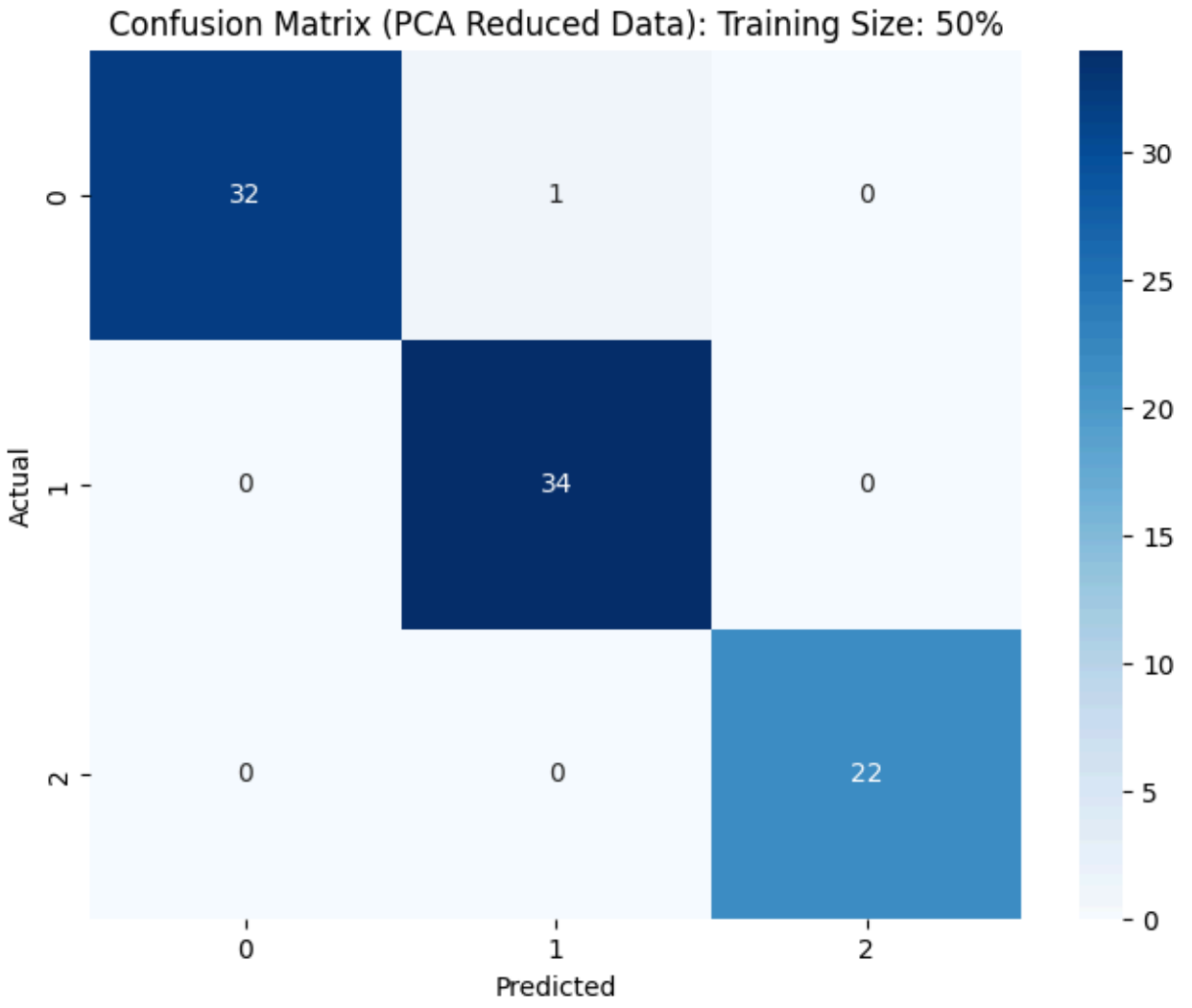

Random Forest Precision by Training Size (PCA Reduced Data)
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

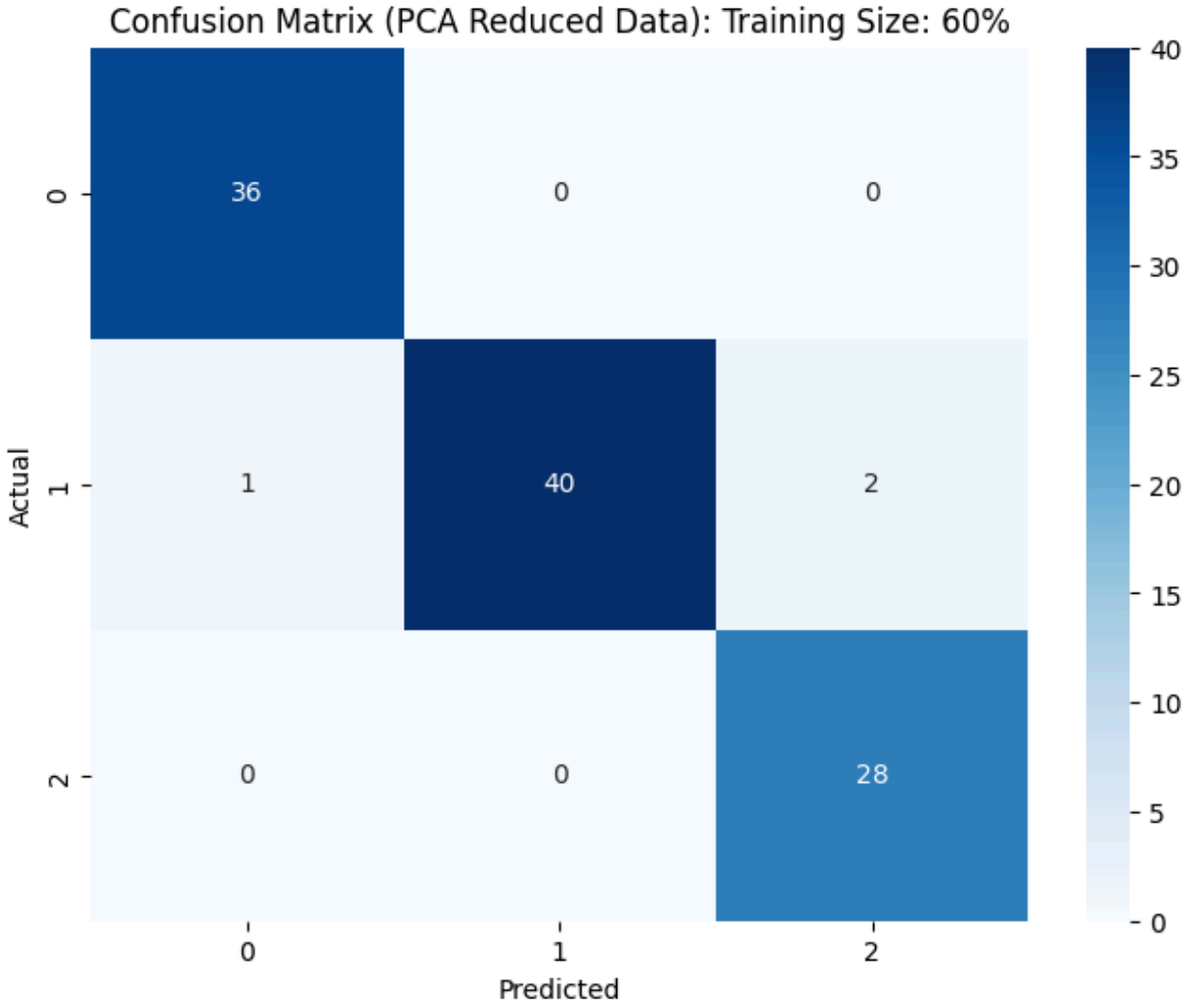


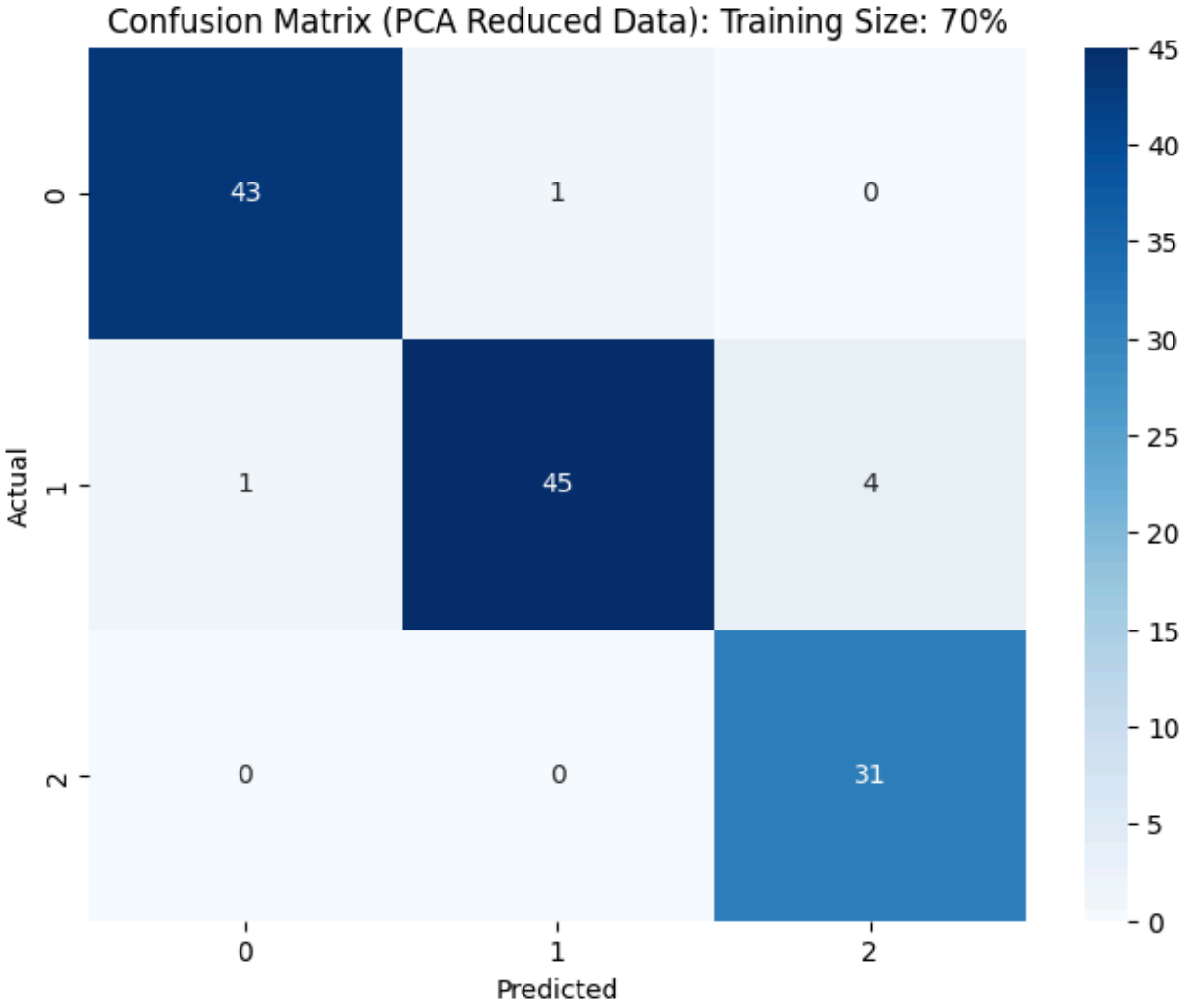
Show Confusion Matrices for MLP with PCA-Reduced Data

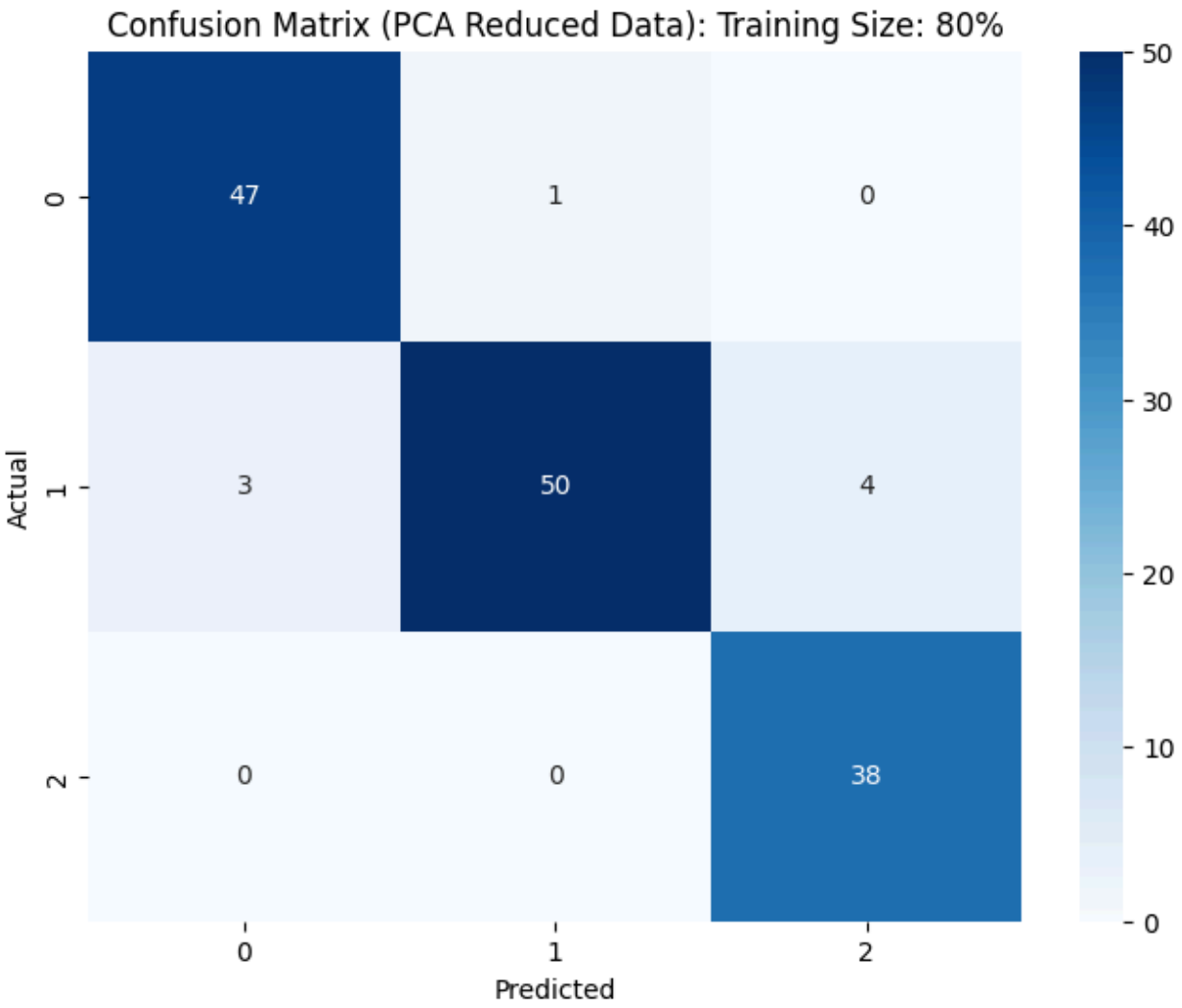
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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_
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```



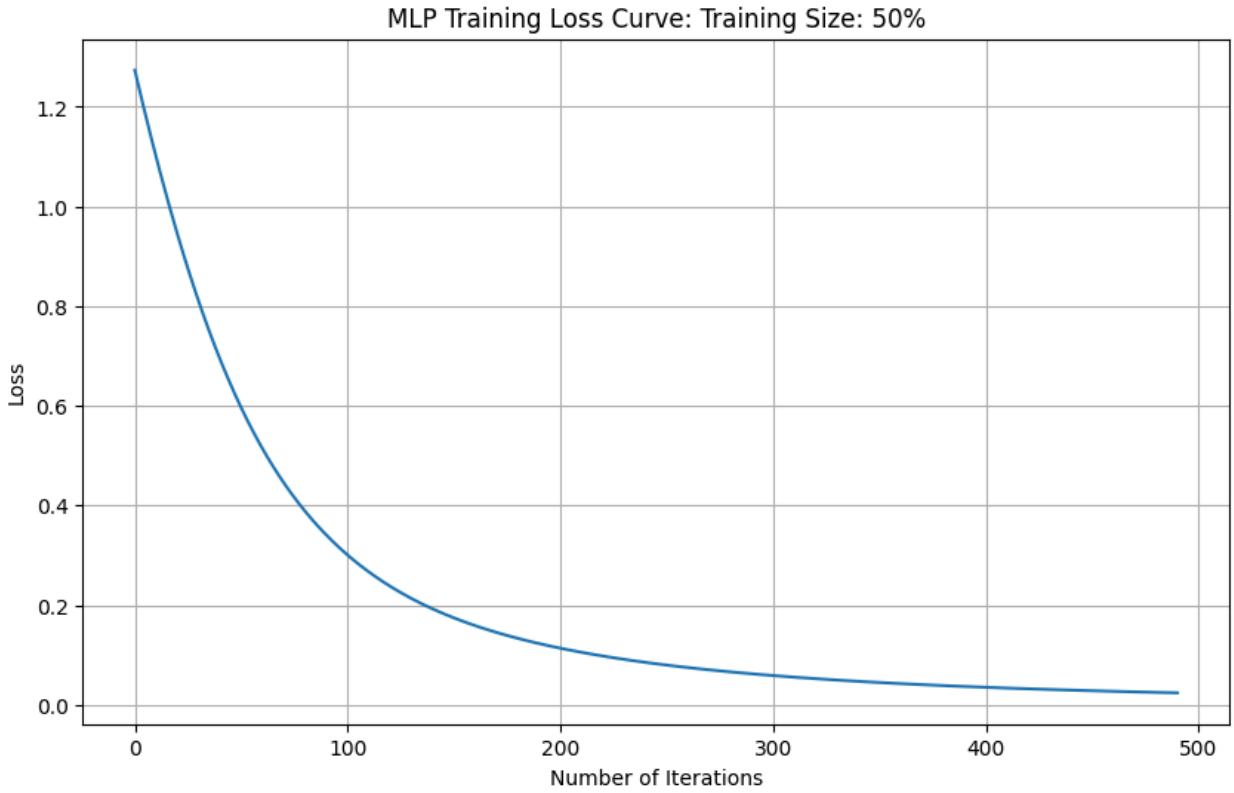

Confusion Matrix (PCA Reduced Data): Training Size: 60%
Actual
Predicted
0
1
2
36
0
0
1
40
2
0
0
28
40
35
30
25
20
15
10
5
0

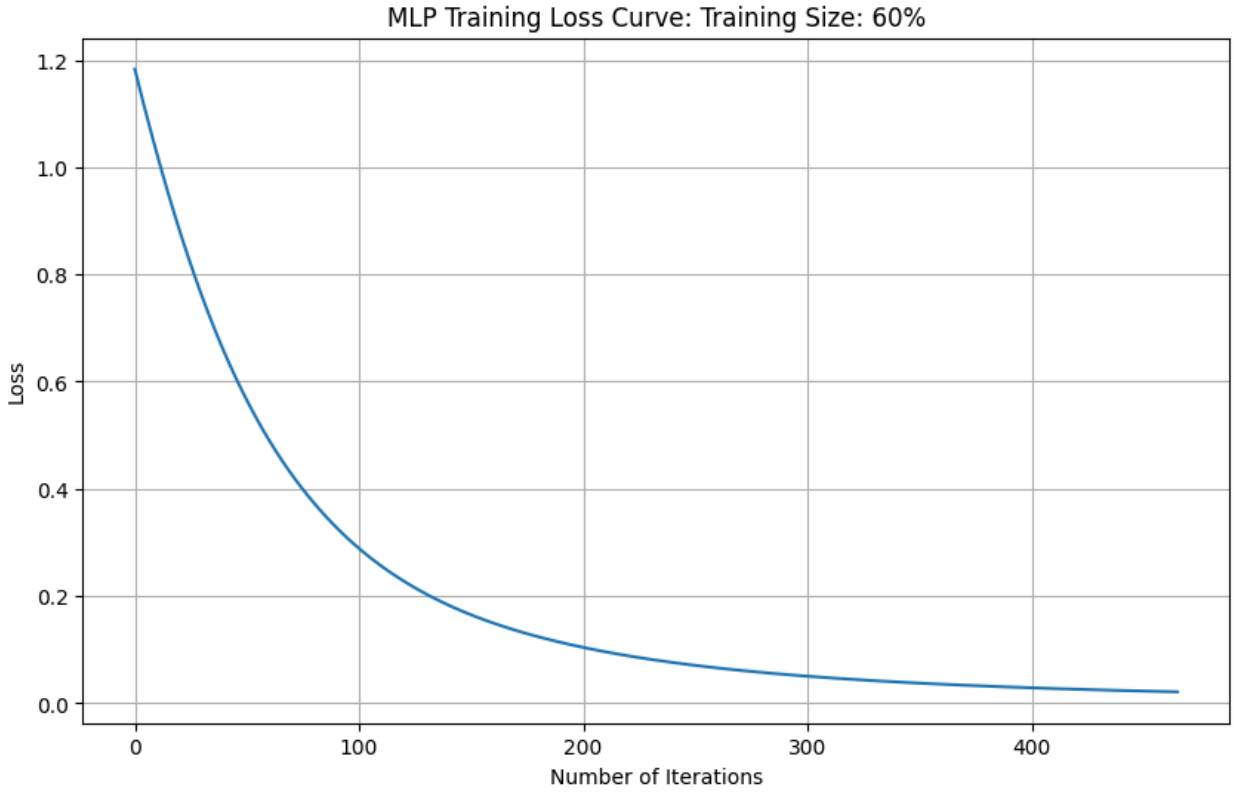

Confusion Matrix (PCA Reduced Data): Training Size: 70%
Actual
Predicted
0
1
2
43
1
0
1
45
4
0
0
31
45
40
35
30
25
20
15
10
5
0

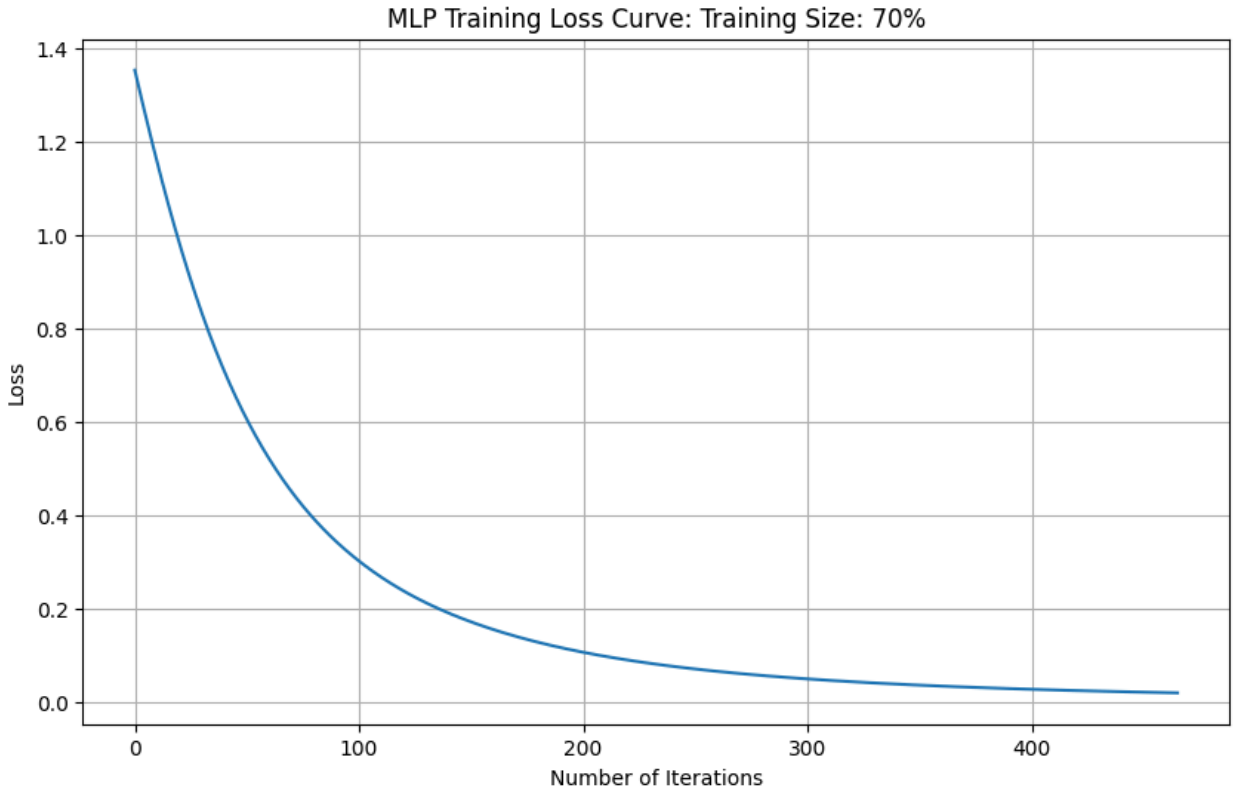


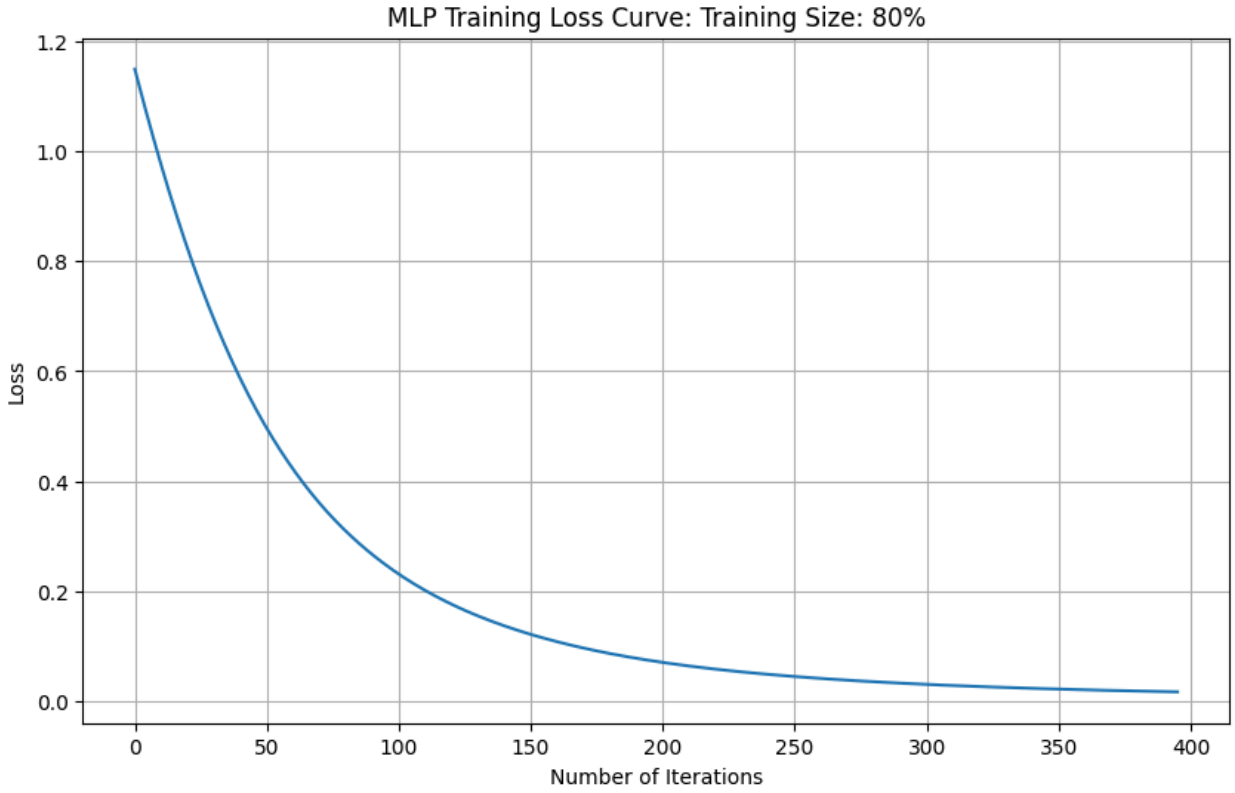
Training Loss generation curve with PCA-Reduced Data

In [ ]:
```python
# 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%
Loss
Number of Iterations
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
100
200
300
400
500


MLP Training Loss Curve: Training Size: 60%
Loss
Number of Iterations
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
100
200
300
400


MLP Training Loss Curve: Training Size: 70%
Loss
Number of Iterations
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
100
200
300
400


MLP Training Loss Curve: Training Size: 80%
Loss
Number of Iterations
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
50
100
150
200
250
300
350
400

===== 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_m
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, ra
  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', ze
  recall_test = recall_score(y_test, y_pred_test, average='weighted') # Calcul
  f1_test = f1_score(y_test, y_pred_test, average='weighted') # Calculate weig

  acc_train = accuracy_score(y_train, y_pred_train)
  precision_train = precision_score(y_train, y_pred_train, average='weighted',
  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
```

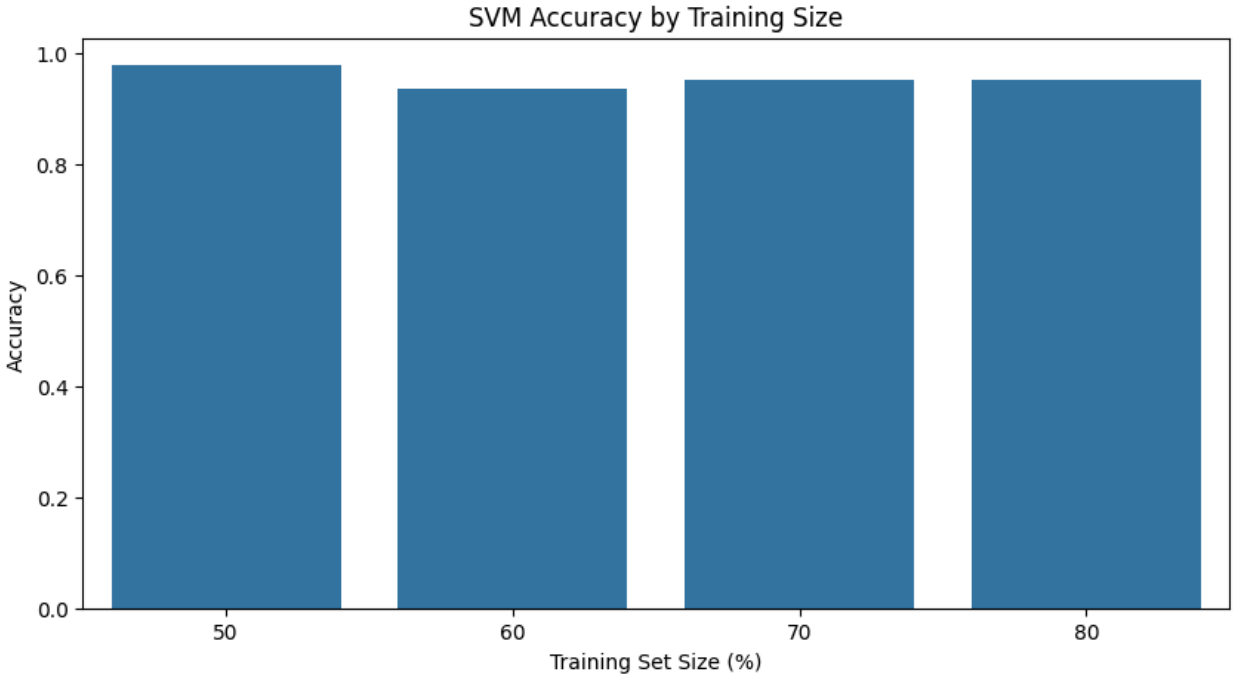
```
  })

  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
  })
```

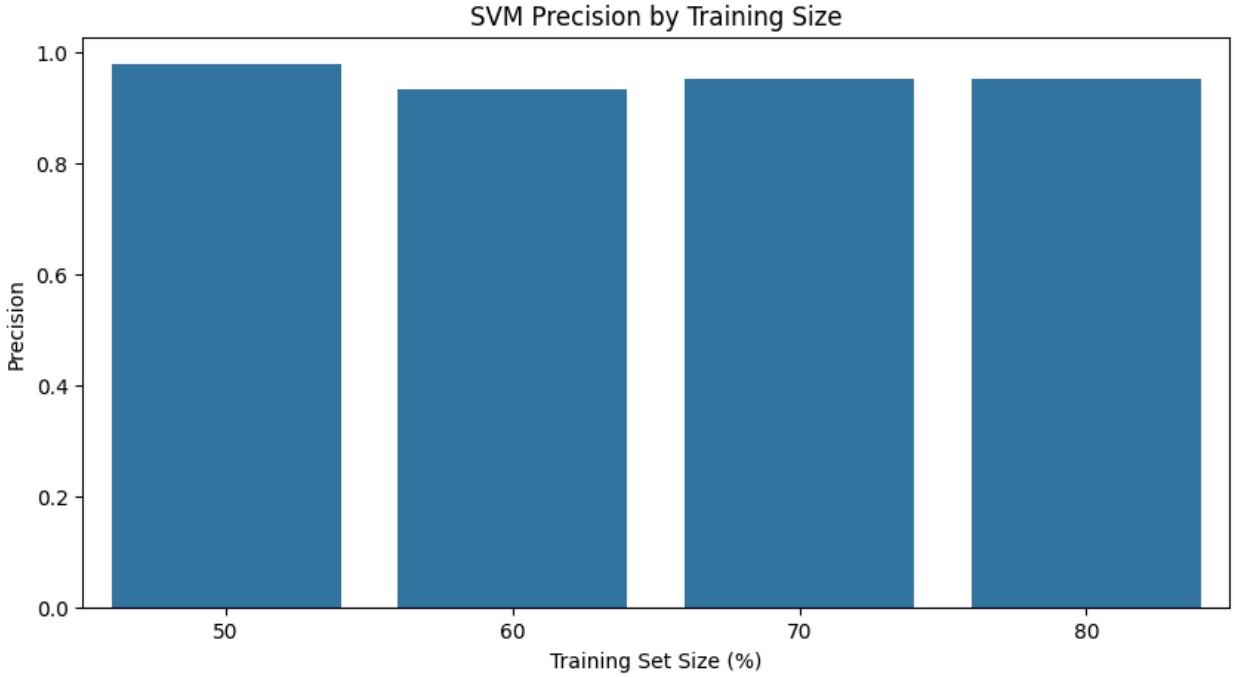
Print Evalution Table and Graphs

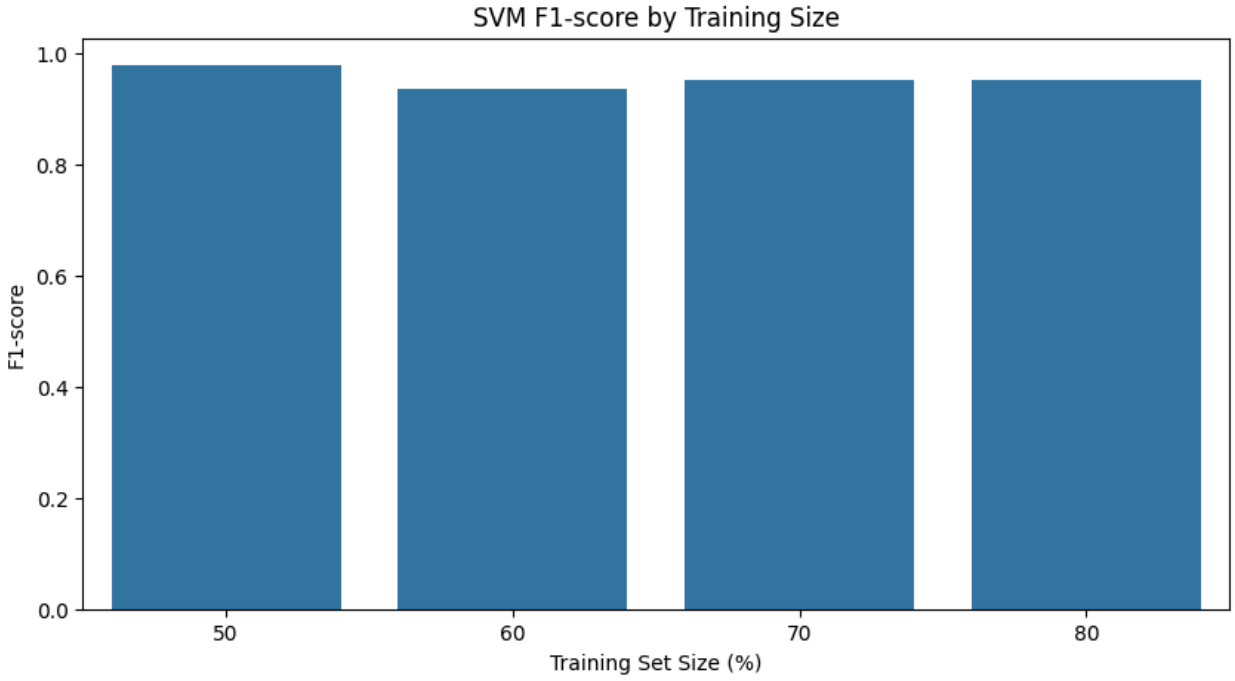
In [ ]:

```
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()
```

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.977528	0.978777	0.977528	0.977498
1	60	0.934579	0.934670	0.934579	0.934515
2	70	0.952000	0.952117	0.952000	0.951798
3	80	0.951049	0.952787	0.951049	0.950976



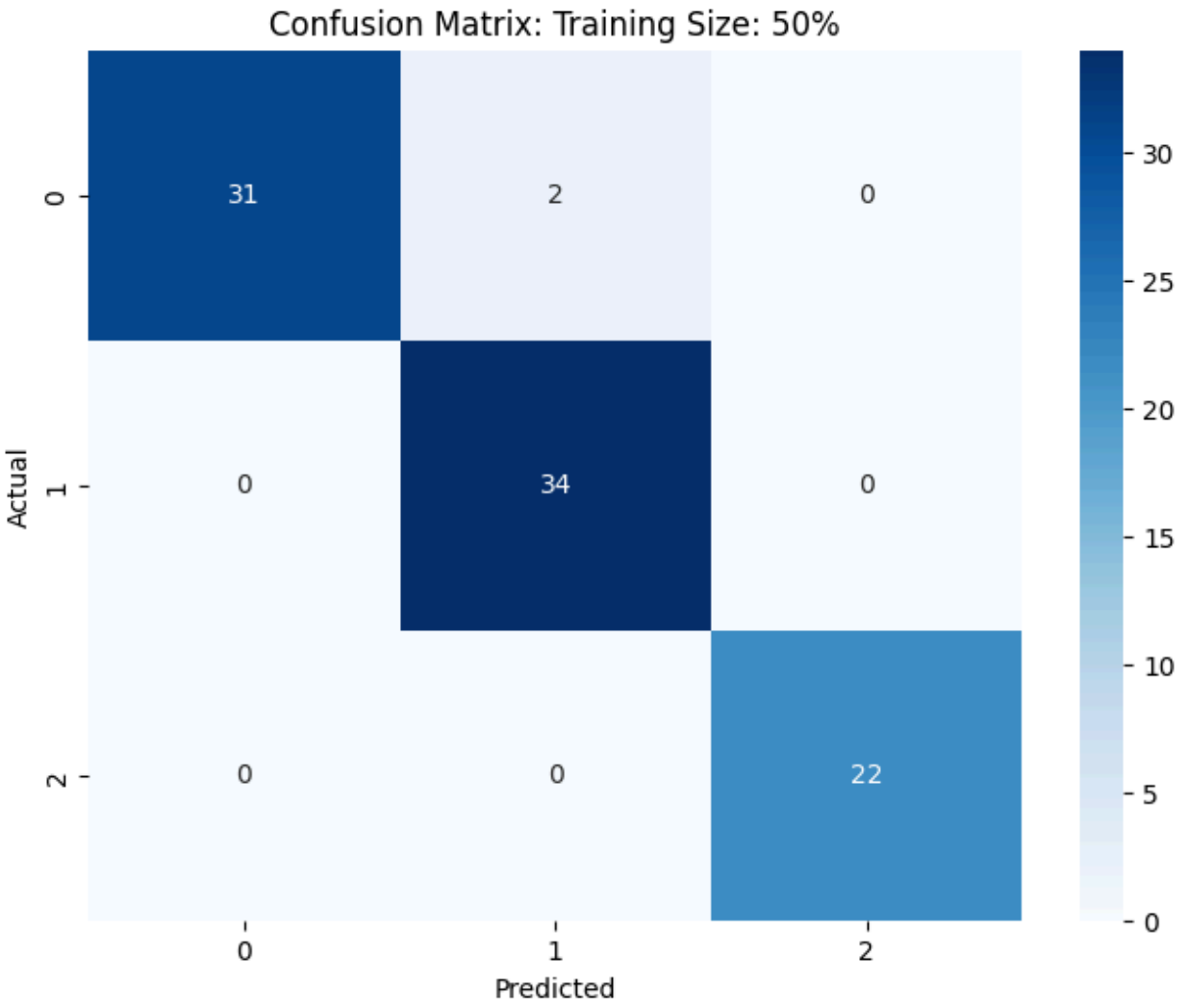


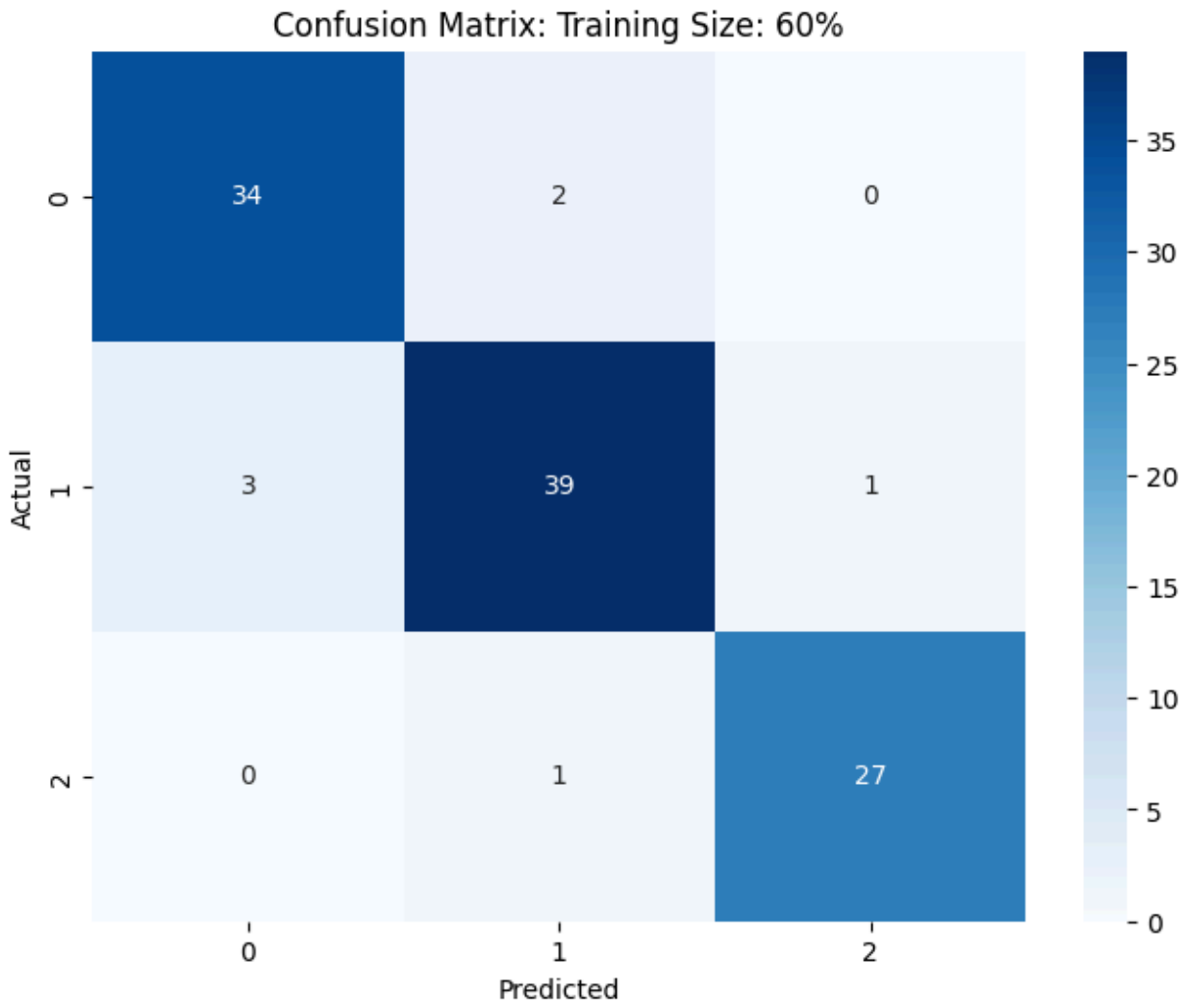


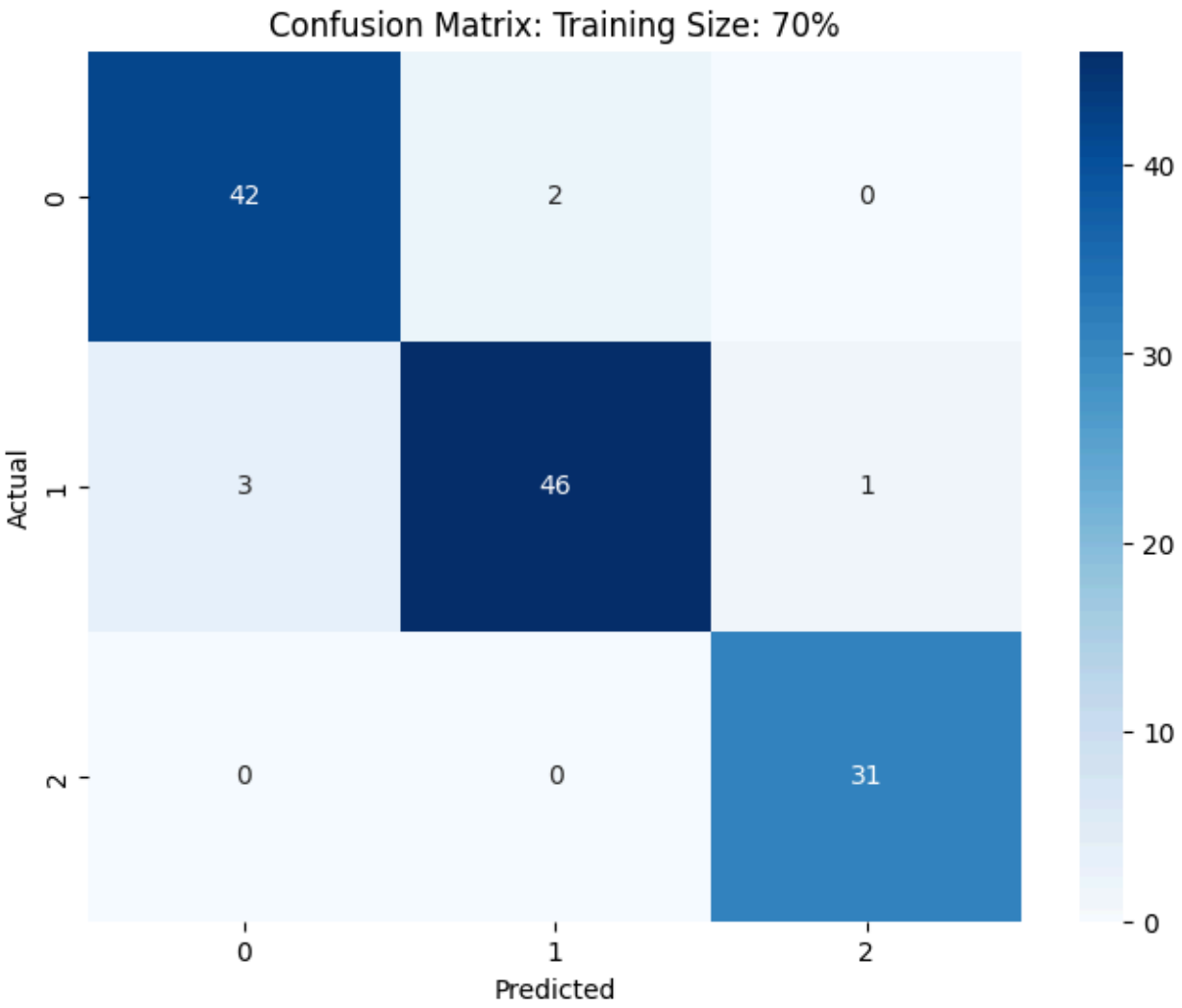
Show Confusion Matrices

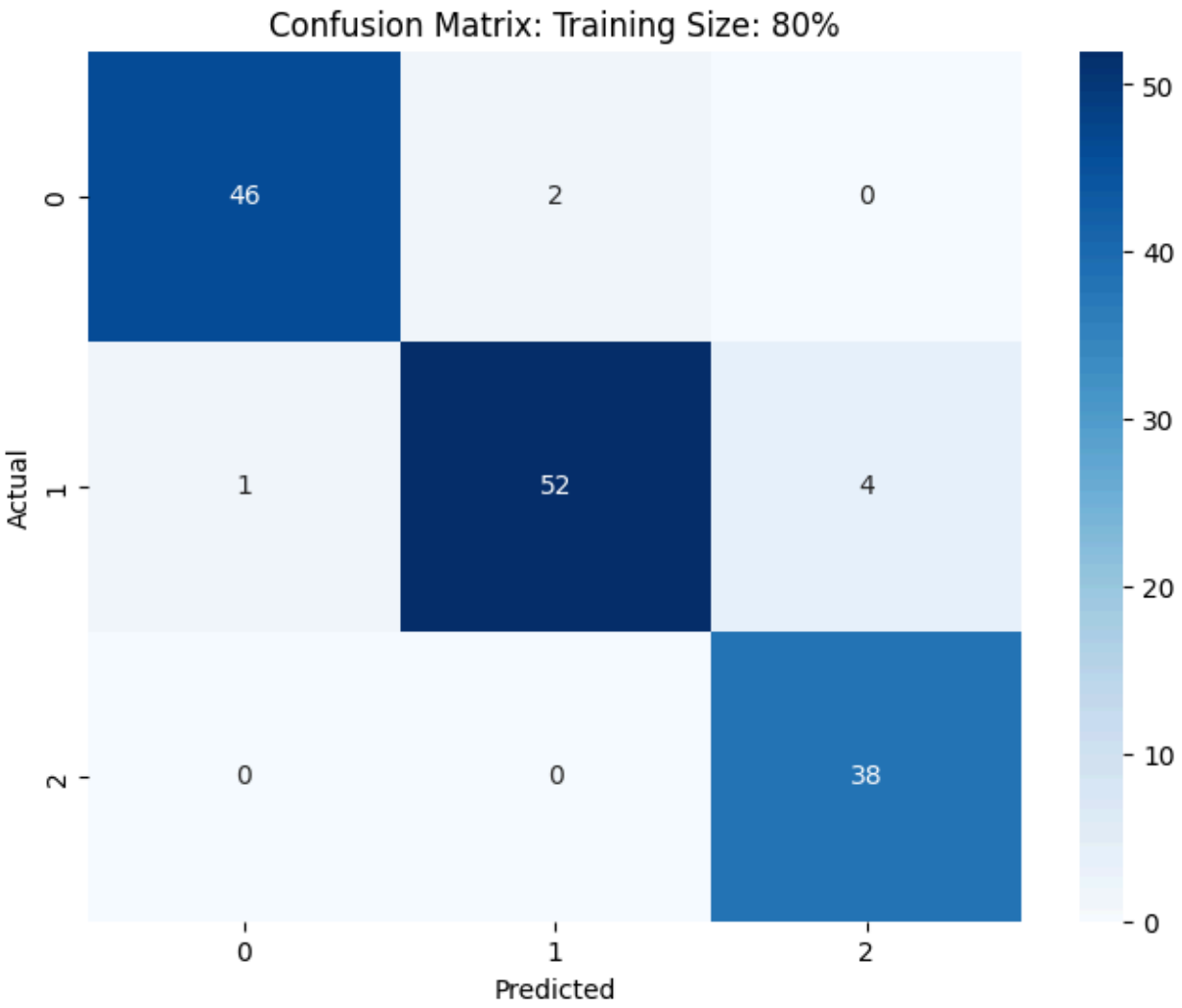
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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%
34
2
0
3
39
1
0
1
27
Actual
Predicted
0
1
2
35
30
25
20
15
10
5
0

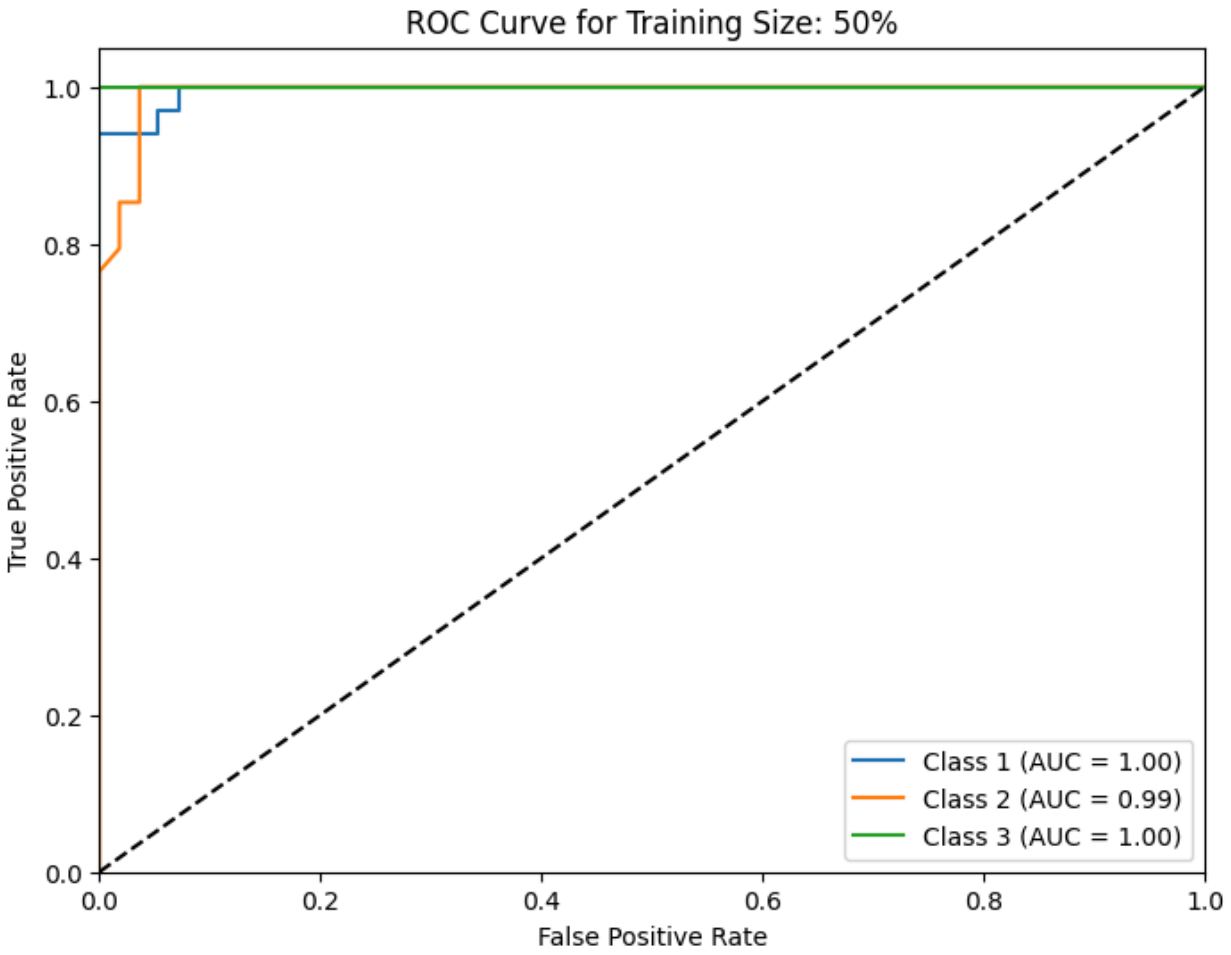

Confusion Matrix: Training Size: 70%
42
2
0
3
46
1
0
0
31
Actual
0
1
2
Predicted
40
30
20
10
0

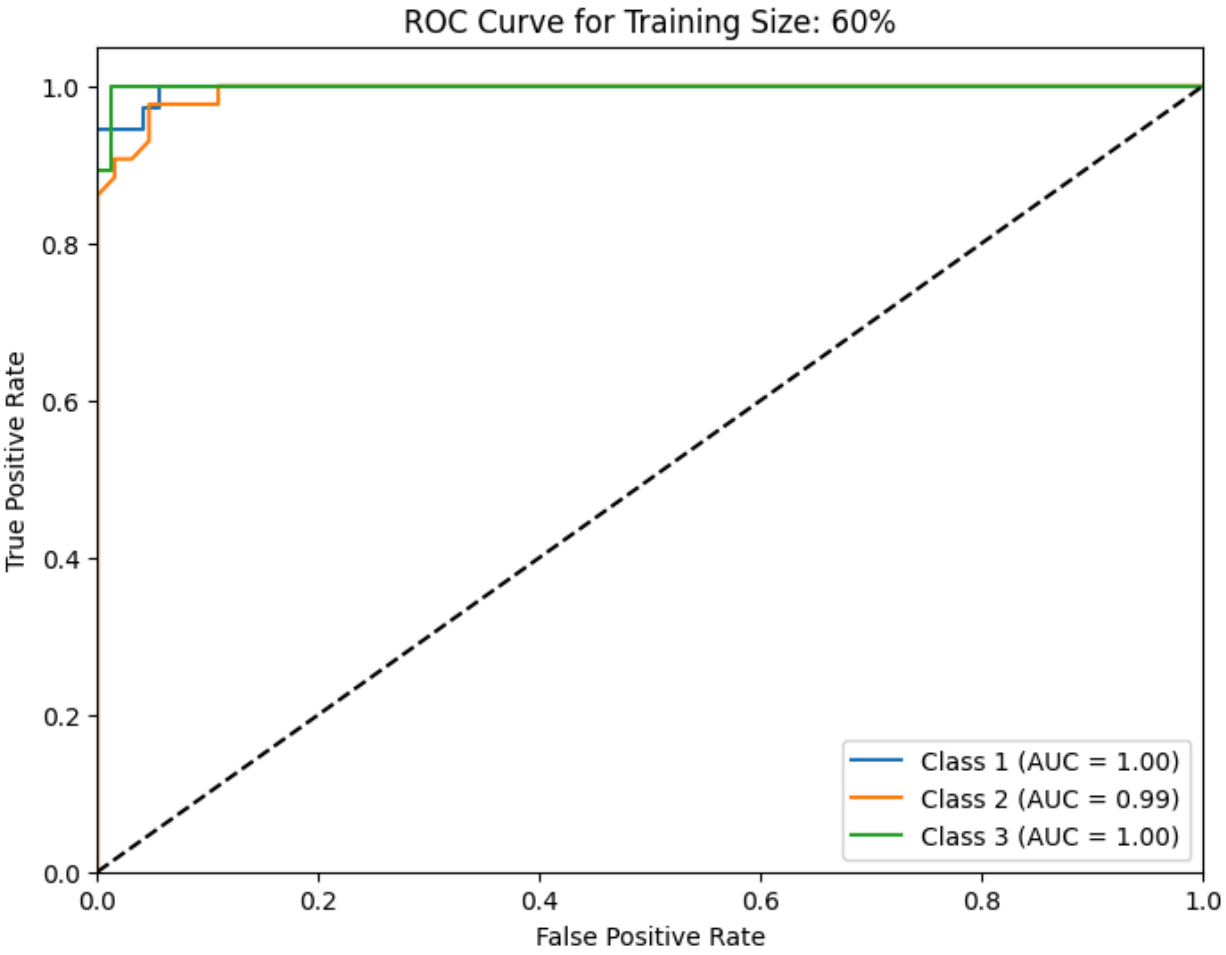


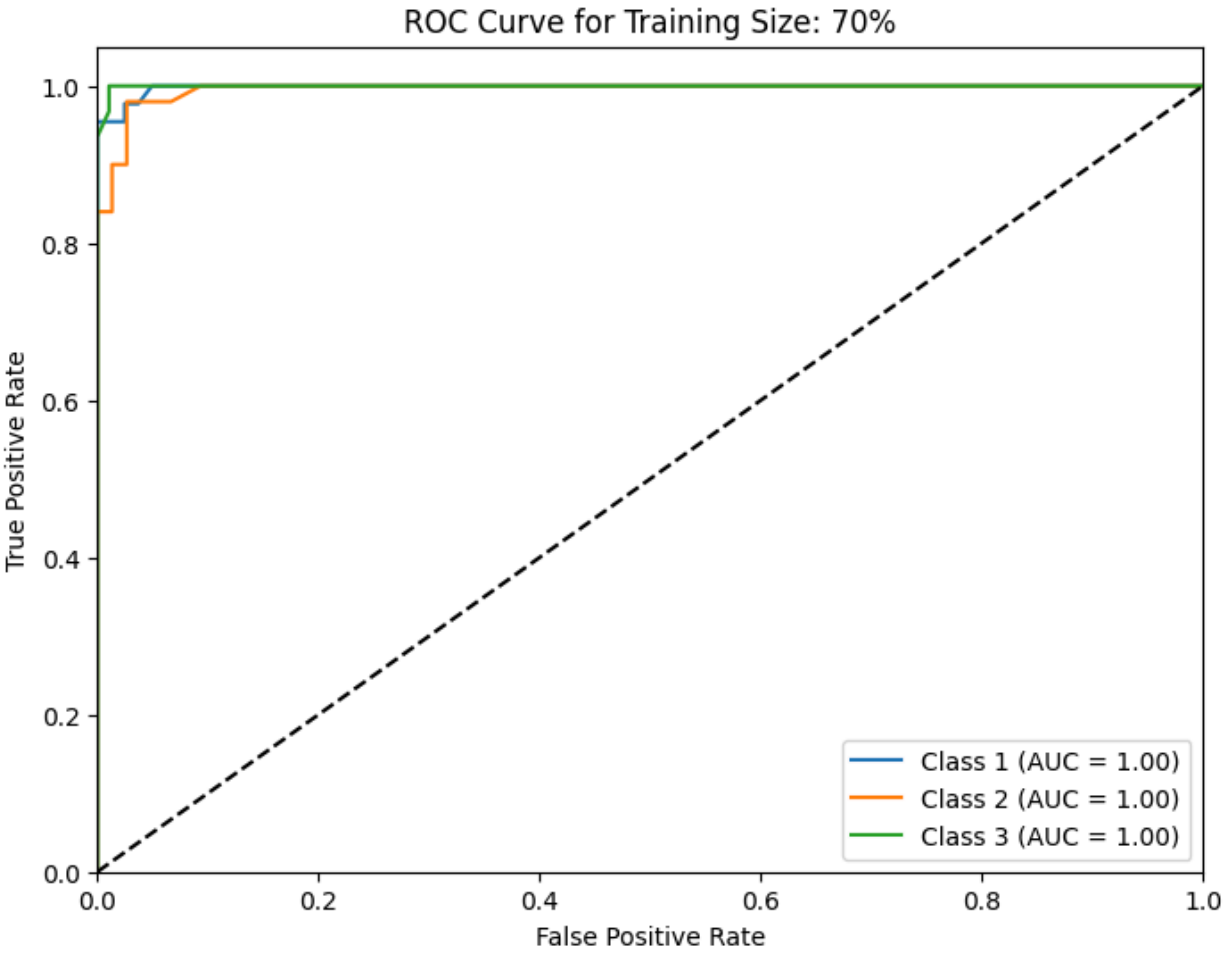
ROC and AOC curve

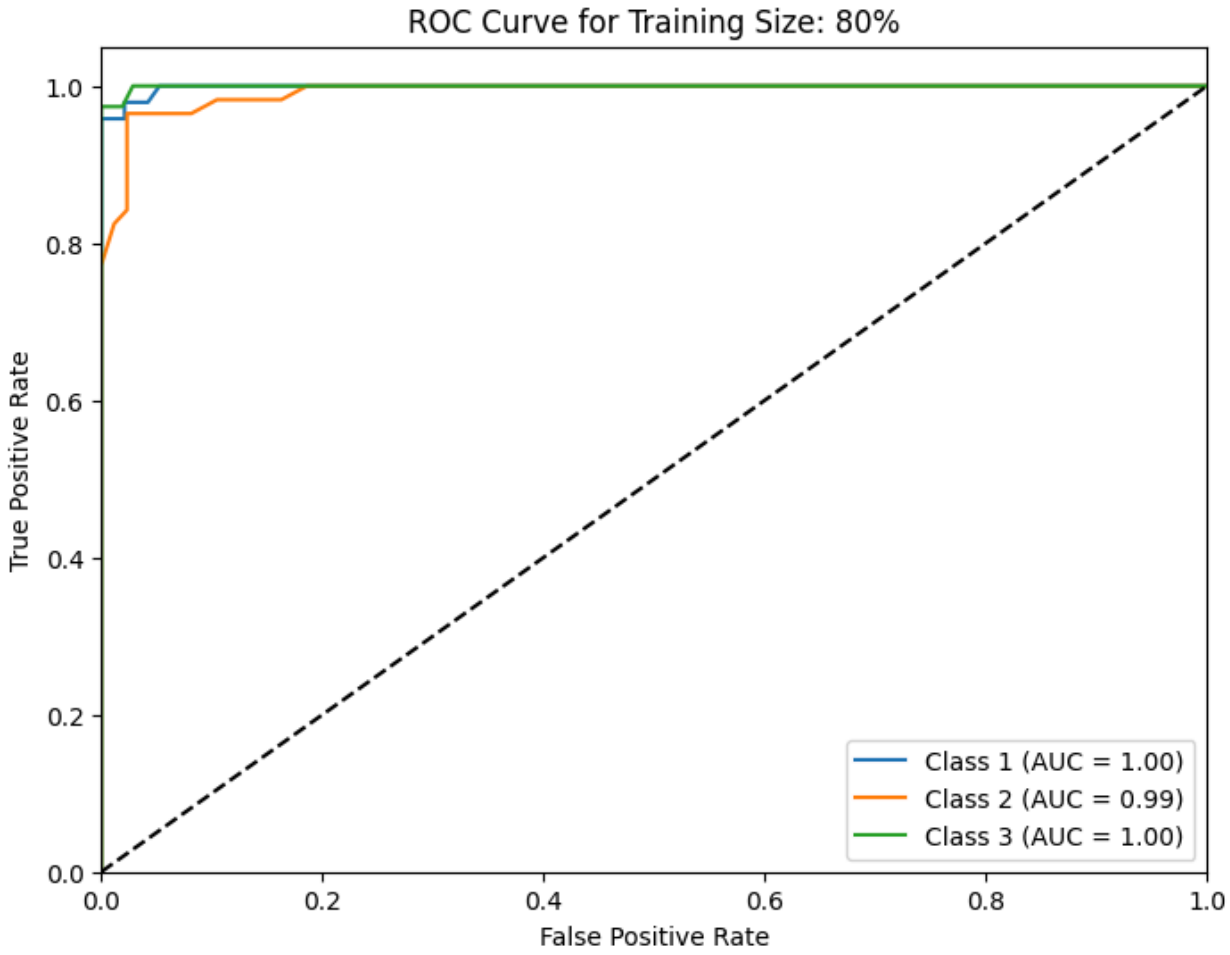
In [ ]:
```
from sklearn.metrics import roc_curve, auc

# 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 fo
    plt.legend(loc="lower right")
    plt.show()
```




ROC Curve for Training Size: 60%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 1.00)
Class 2 (AUC = 0.99)
Class 3 (AUC = 1.00)


ROC Curve for Training Size: 70%
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class 1 (AUC = 1.00)
Class 2 (AUC = 1.00)
Class 3 (AUC = 1.00)



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:
```

	0	1	2	3	4	5	6	
0	3.316751	1.443463	-0.165739	-0.215631	0.693043	0.223880	0.596427	-0.065
1	2.209465	-0.333393	-2.026457	-0.291358	-0.257655	0.927120	0.053776	-1.024
2	2.516740	1.031151	0.982819	0.724902	-0.251033	-0.549276	0.424205	0.344
3	3.757066	2.756372	-0.176192	0.567983	-0.311842	-0.114431	-0.383337	-0.643
4	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

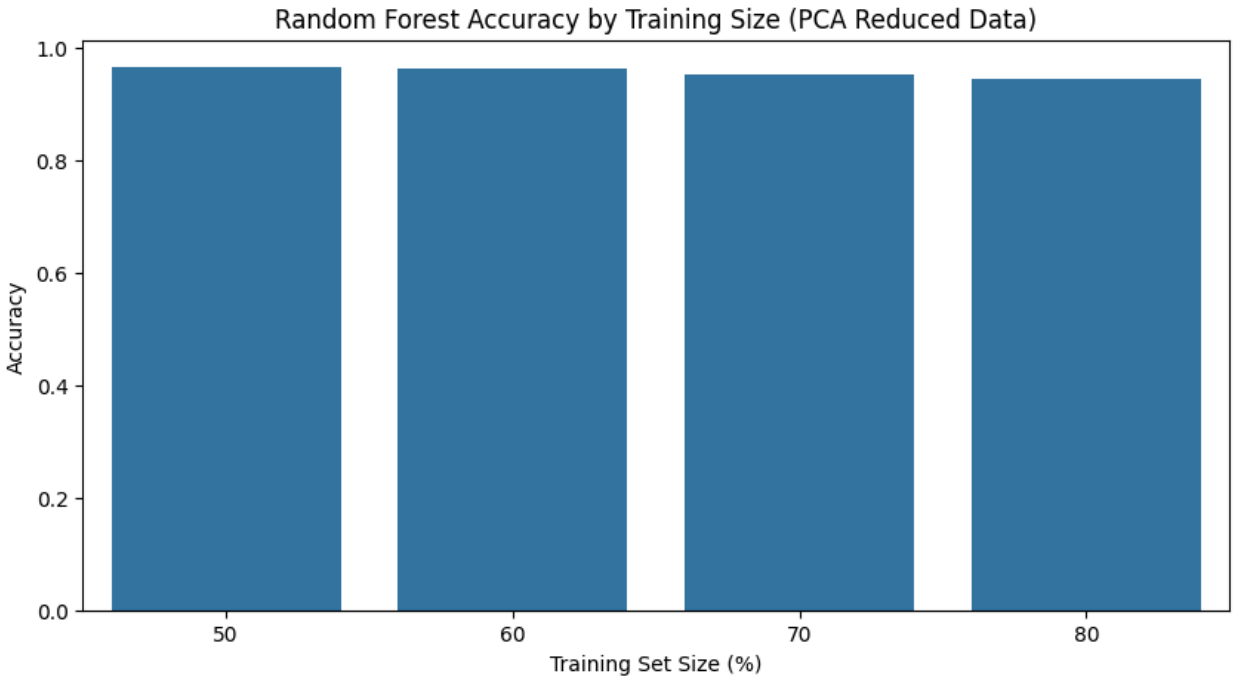
In [ ]:
```
results_rf_pca = []
confusion_matrices_rf_pca = []


for size in training_sizes:
  X_train_pca_rf, X_test_pca_rf, y_train, y_test = train_test_split(X_pca_redu
  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
  })
```

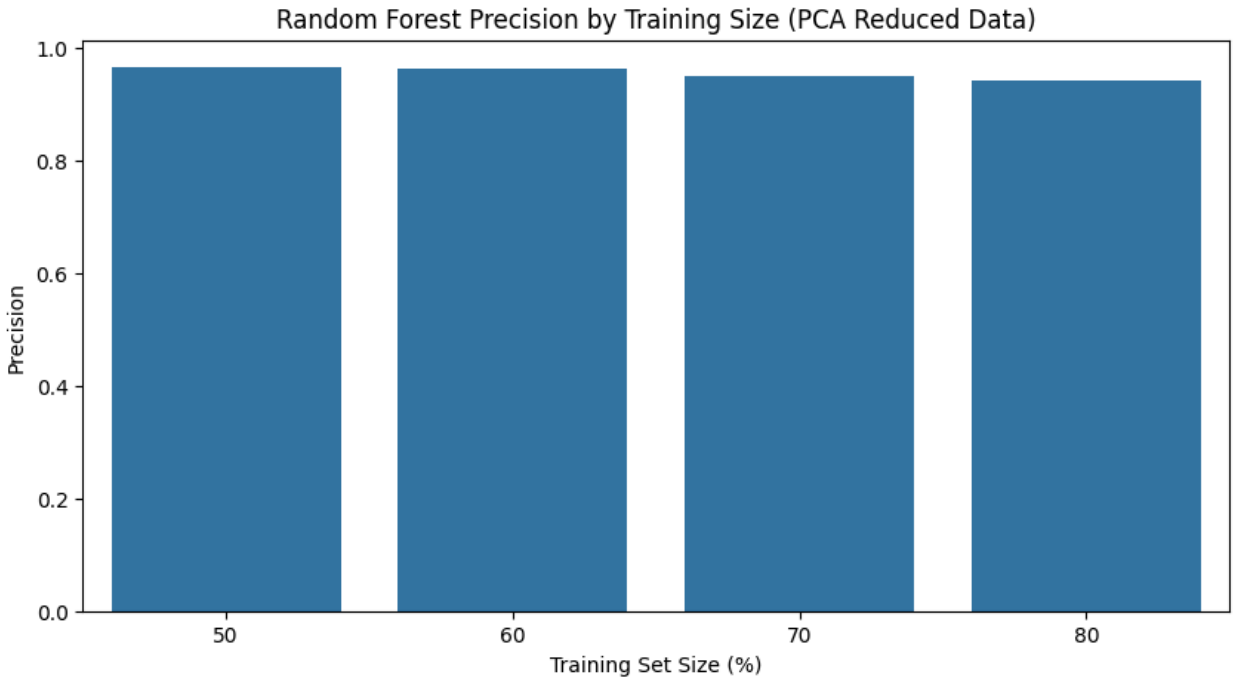
Print Random Forest Performance Table and Graphs with PCA-Reduced Data

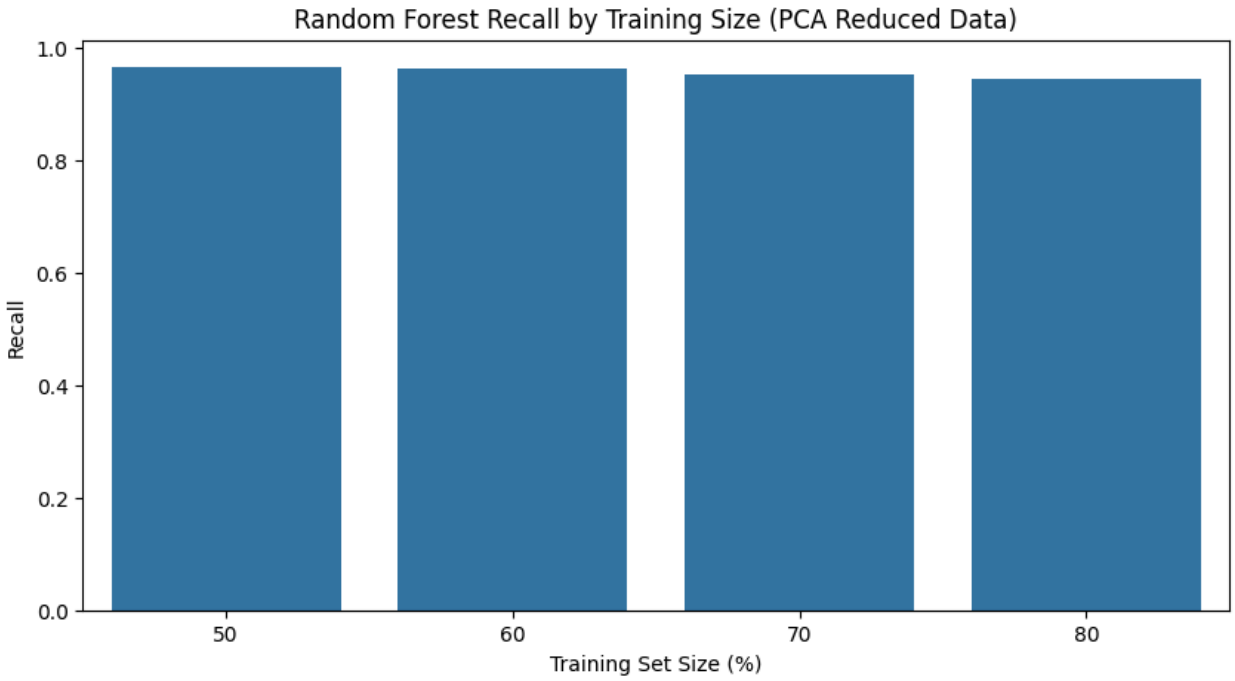
In [ ]:
```
df_rf_pca = pd.DataFrame(results_rf_pca)
display(df_rf_pca)
```

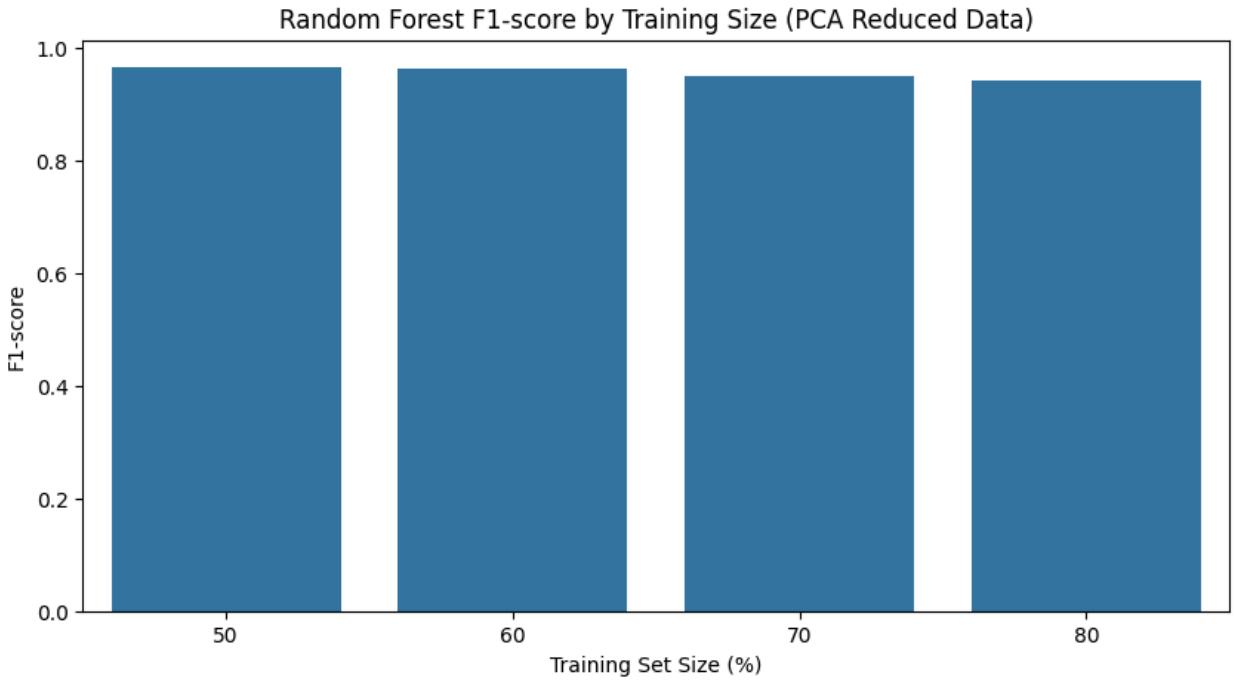
```
# Plot Accuracy
plt.figure(figsize=(10, 5))
sns.barplot(data=df_rf_pca, x='Training size', y='Accuracy')
plt.title('Random Forest Accuracy by Training Size (PCA Reduced Data)')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.show()

# 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()
```

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


Random Forest Accuracy by Training Size (PCA Reduced Data)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)


Random Forest Precision by Training Size (PCA Reduced Data)
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

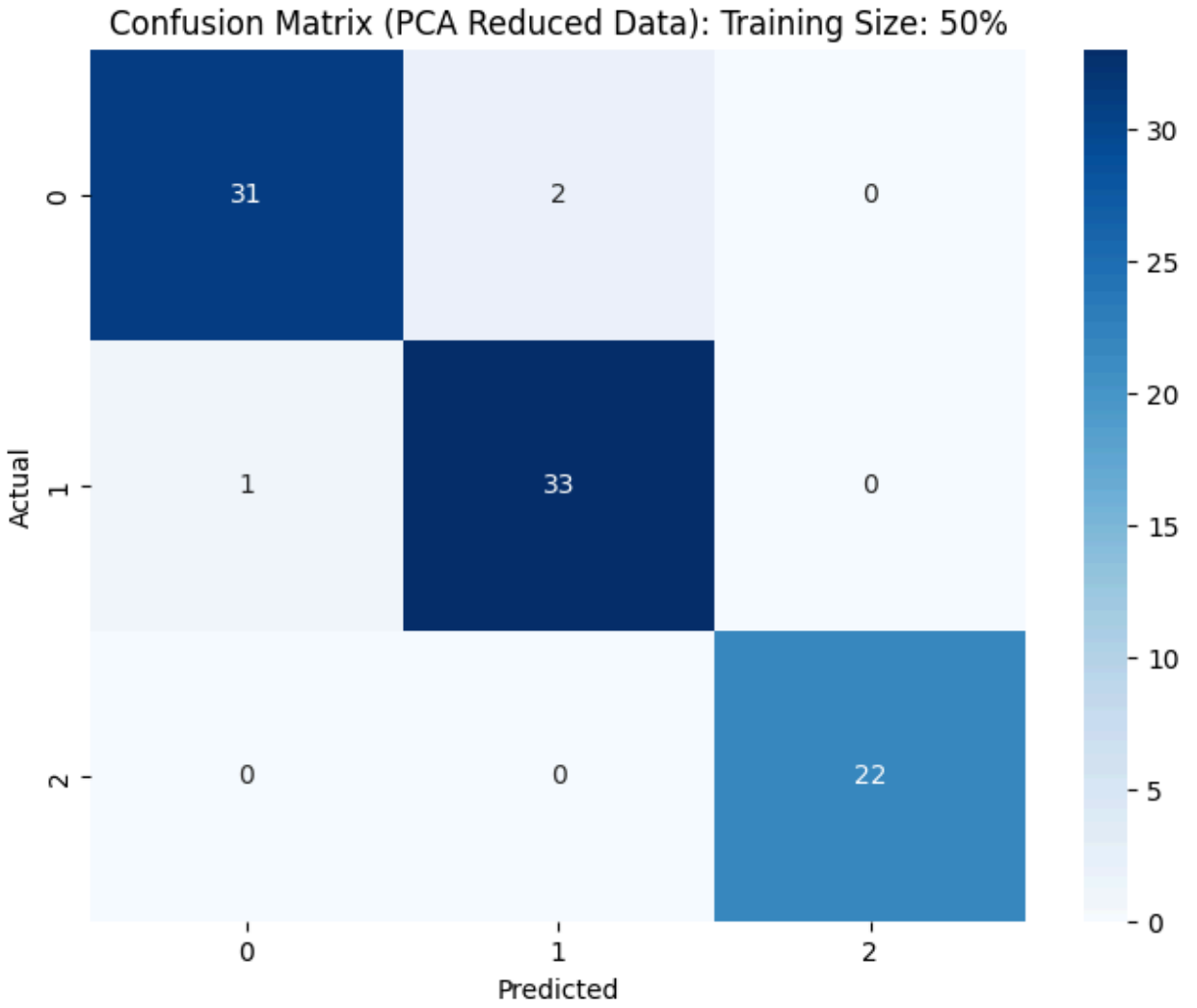


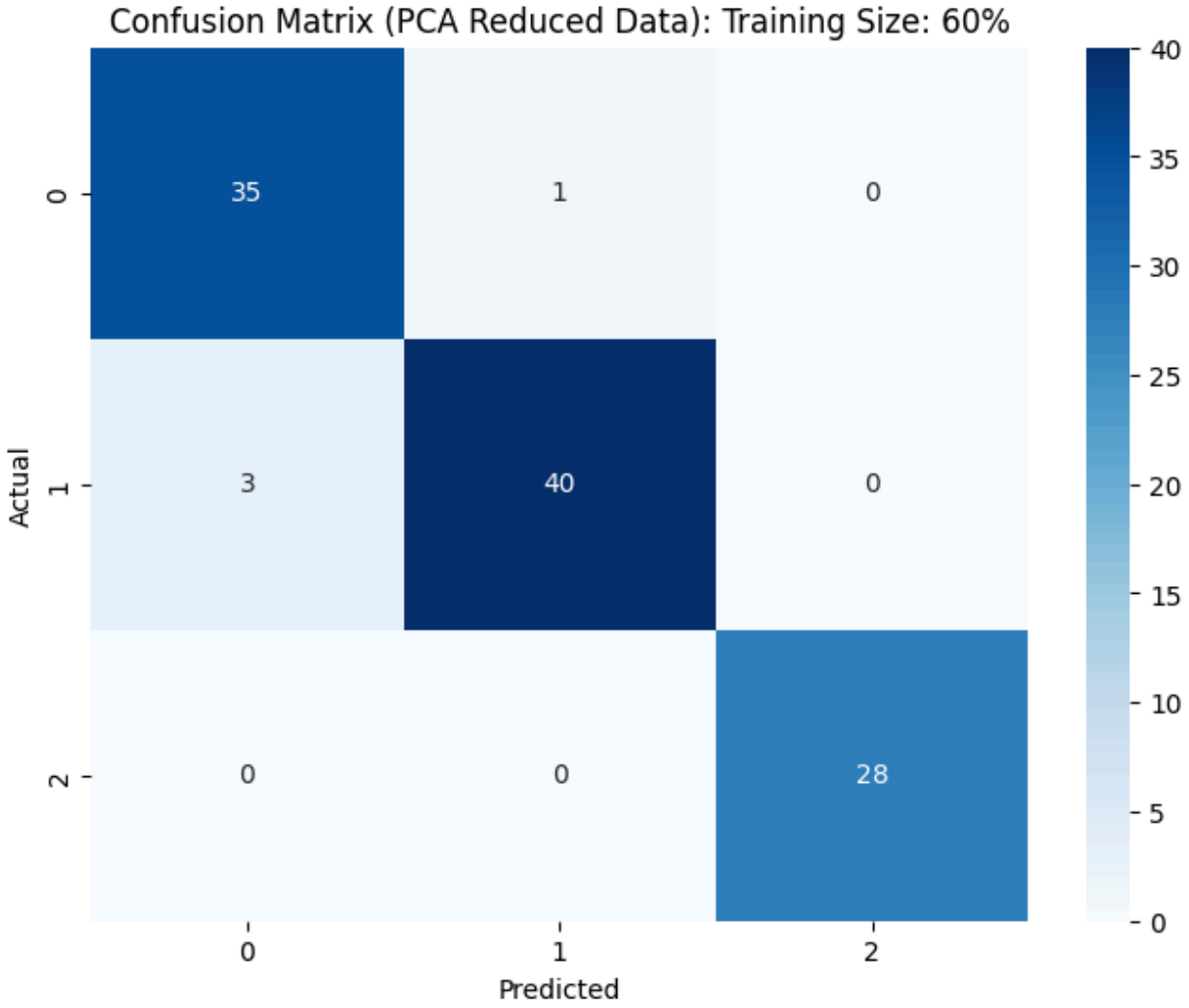


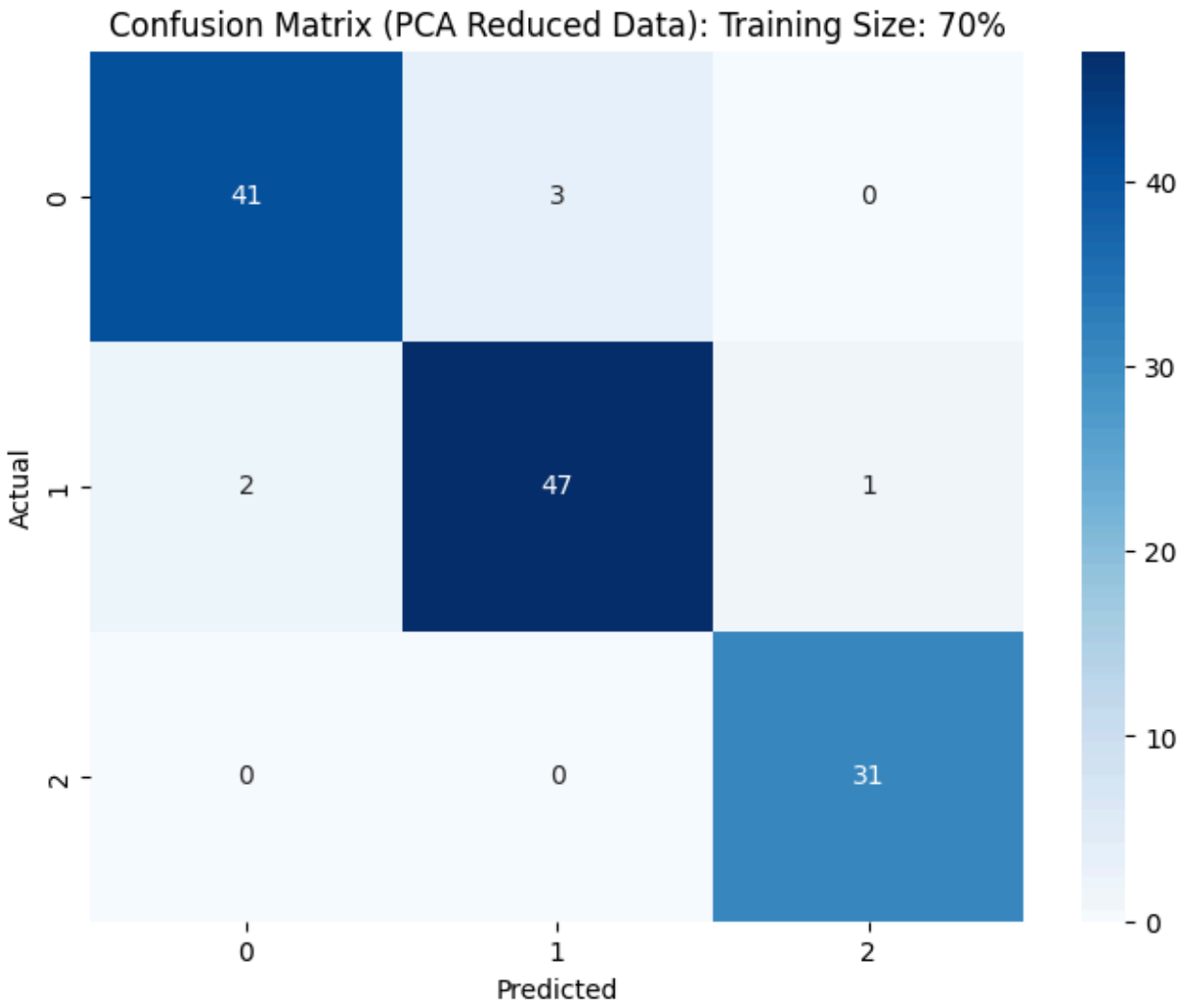
Show Confusion Matrices for Random Forest with PCA-Reduced Data

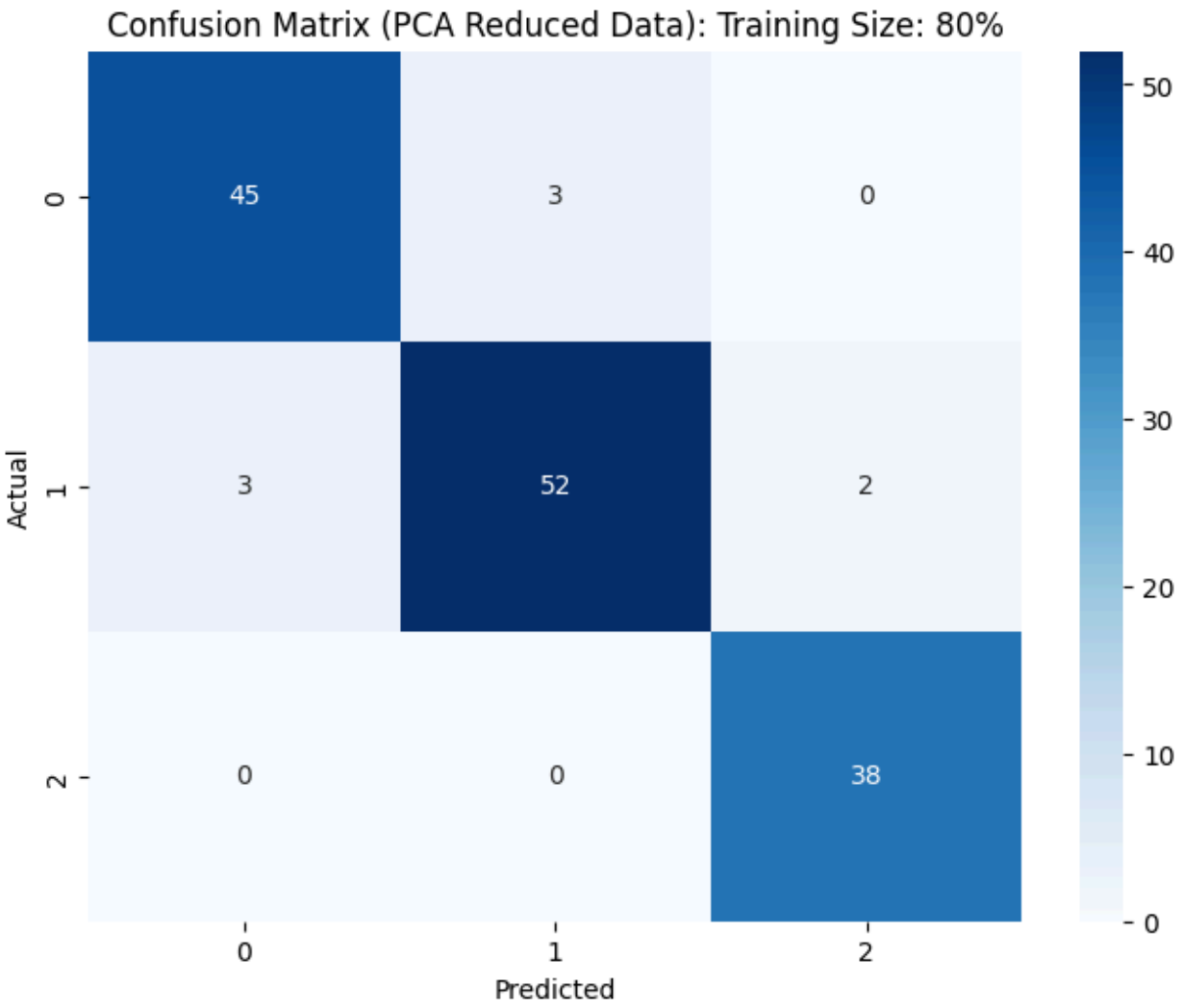
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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_
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```




Confusion Matrix (PCA Reduced Data): Training Size: 60%
35
1
0
3
40
0
0
0
28
Actual
Predicted
0
1
2
40
35
30
25
20
15
10
5
0


Confusion Matrix (PCA Reduced Data): Training Size: 70%
41
3
0
2
47
1
0
0
31
Actual
0
1
2
Predicted
40
30
20
10
0



Machine Learning Model Performance Report

Wine Dataset Evaluation

1. Support Vector Machine (SVM)

This section evaluates how SVM models perform on the Wine dataset, focusing on the influence of kernel choice, training set size, and dimensionality reduction using PCA.

Effect of Training Size and Kernel Selection:

SVM models were trained using four kernel functions—linear, polynomial, gaussian, and sigmoid—on different proportions of the dataset (50%, 60%, 70%, and 80%).

Performance was assessed using Accuracy, Precision, Recall, and F1-score.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for each kernel here]

Results Summary:

[Insert table of SVM results on Original Wine Data here]

Across all training sizes, the linear kernel consistently produced the strongest results. Both polynomial and gaussian kernels delivered fluctuating performance depending on the training set size, while the sigmoid kernel lagged behind across all scenarios. Increasing the training size generally enhanced the performance for every kernel, highlighting the importance of larger datasets for model generalization.

Influence of PCA Feature Reduction:

PCA was used to reduce the feature dimensionality of the Wine dataset while retaining most of its variance. The SVM models were then retrained on this reduced representation.

[Insert table of SVM results on PCA-Reduced Data here]

The results show that PCA positively influenced the performance of the SVM models, especially when using the linear and gaussian kernels. Metrics such as accuracy, precision, and recall improved, indicating that the lower-dimensional representation made patterns in the data more accessible to the models.

Confusion Matrices:

[Insert Confusion Matrices for SVM on Original Data here]

[Insert Confusion Matrices for SVM on PCA-Reduced Data here]

The confusion matrices reveal that PCA, particularly in combination with the linear kernel, resulted in a higher proportion of correct classifications compared to the original feature space.

ROC and AUC Curves:

[Insert ROC and AUC curves for SVM on Original Data here]

The ROC and AUC plots reinforce these findings, with the linear kernel achieving the highest AUC values and PCA further enhancing the separation between classes.

---

2. Multilayer Perceptron (MLP)

This section examines how MLP models perform on the Wine dataset, with emphasis on training size effects, learning behavior, and PCA-based dimensionality reduction.

Effect of Training Size:

The MLP model was trained with dataset splits of 50%, 60%, 70%, and 80% to observe changes in performance as training size increased.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for MLP here]

Results Summary:

[Insert table of MLP results on Original Wine Data here]

On the original dataset, the MLP's performance varied across training sizes without showing a consistent upward trend. This suggests that the chosen hyperparameters may not have been fully optimized for the dataset.

Training Loss Curve:

[Insert MLP Training Loss Curves here]

The loss curves provide insight into the convergence process. While the loss generally decreased over time, the curves indicate potential room for improvement through hyperparameter tuning or extending training epochs.

Effect of PCA Feature Reduction:

The MLP model was also tested on PCA-transformed features.

[Insert table of MLP results on PCA-Reduced Data here]

With PCA, the MLP achieved a notable boost in performance. Both accuracy and other metrics improved significantly, suggesting that dimensionality reduction minimized noise and allowed the model to focus on the most important patterns in the data.

Confusion Matrices:

[Insert Confusion Matrices for MLP on Original Data here]

[Insert Confusion Matrices for MLP on PCA-Reduced Data here]

The confusion matrices show far fewer misclassifications on the PCA-reduced dataset compared to the original, underscoring PCA's positive effect.

ROC and AUC Curves:

[Insert ROC and AUC curves for MLP on Original Data here]

When PCA was applied, the ROC and AUC curves showed higher AUC values, confirming that the MLP could distinguish classes more effectively with reduced dimensions.

---

3. Random Forest

This section reviews the Random Forest model's performance on the Wine dataset, considering both original and PCA-transformed features.

Effect of Training Size:

Random Forest was evaluated on training splits of 50%, 60%, 70%, and 80%.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for Random Forest here]

Results Summary:

[Insert table of Random Forest results on Original Wine Data here]

The Random Forest model produced strong and stable results across all training sizes on the original dataset. Its robustness suggests that the algorithm is naturally well-suited to this classification task.

Effect of PCA Feature Reduction:

The same model was then trained on PCA-reduced data.

[Insert table of Random Forest results on PCA-Reduced Data here]

Unlike with SVM and MLP, PCA slightly reduced the performance of Random Forest. This outcome suggests that the original features carried richer discriminatory information for the tree-based method, and dimensionality reduction removed some of that useful detail.

Confusion Matrices:

[Insert Confusion Matrices for Random Forest on Original Data here]

[Insert Confusion Matrices for Random Forest on PCA-Reduced Data here]

The confusion matrices confirm this observation, showing fewer misclassifications in the original dataset compared to PCA-reduced features.

ROC and AUC Curves:

[Insert ROC and AUC curves for Random Forest on Original Data here]

The ROC and AUC plots further emphasize that Random Forest maintained stronger class separation on the original dataset compared to PCA-reduced data.

Installing 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/python
3.12/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-p
ackages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
t-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packa
ges (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
```

Hand Wirtten 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	Att
0	0	1	6	15	12	1	
1	0	0	10	16	6	0	
2	0	0	8	15	16	13	
3	0	0	0	3	11	16	
4	0	0	5	14	4	0	
...	...	...	...	...	...	...	
5615	0	0	4	10	13	6	
5616	0	0	6	16	13	11	
5617	0	0	1	11	15	1	
5618	0	0	2	10	7	0	
5619	0	0	10	14	8	1	

5620 rows × 64 columns

In [ ]:
```
y
```

Out[ ]:

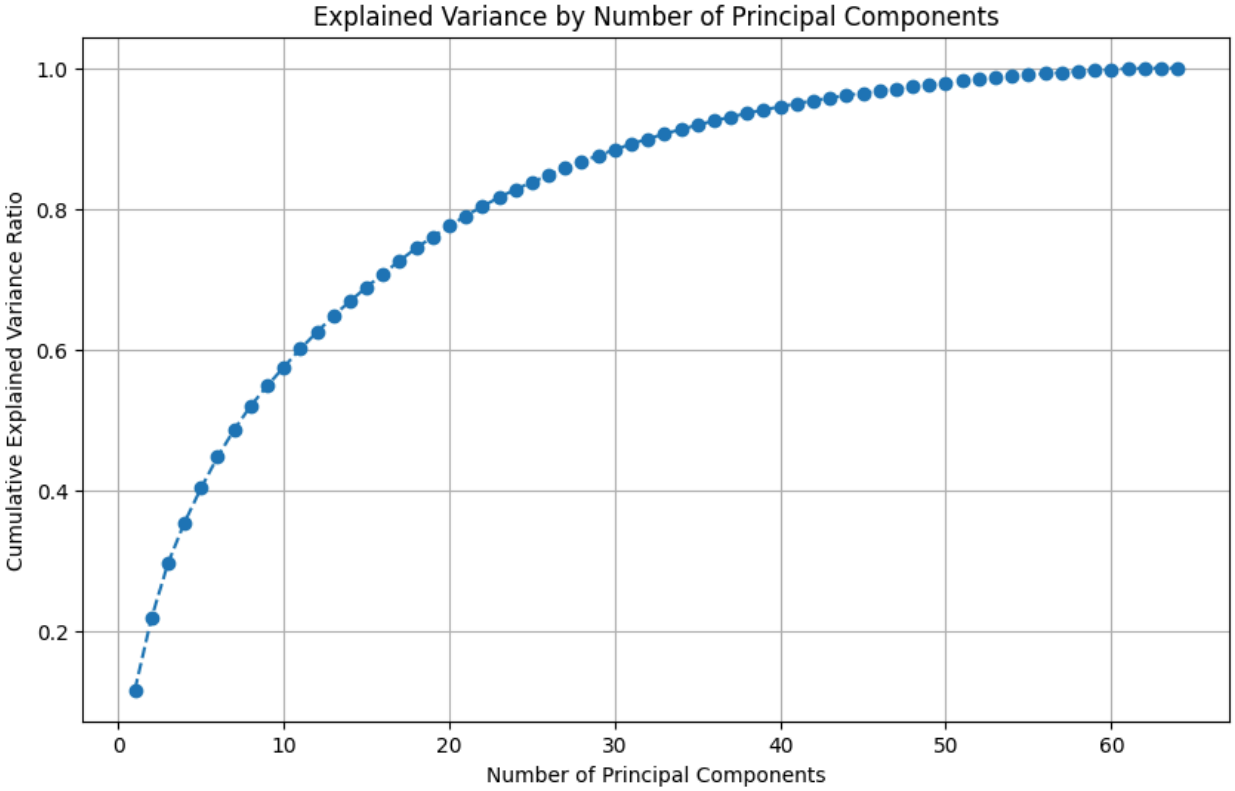
	class
0	0
1	0
2	7
3	4
4	6
...	...
5615	9
5616	0
5617	8
5618	9
5619	8

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.         ]
```



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 th
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_m
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

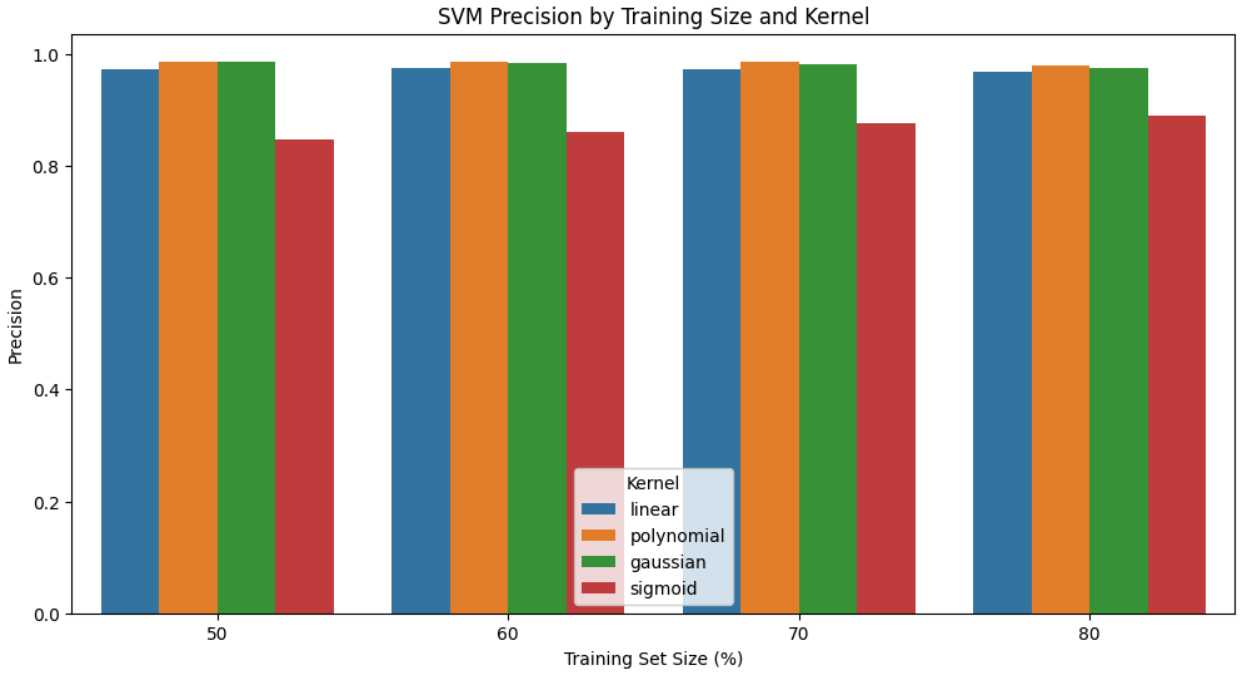
In [ ]:
```
results = []
confusion_matrices = []
trained_models = [] # Add a list to store trained models

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, r
   svc = SVC(kernel=kernel, probability=True, random_state=42) # Add probabili
   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_divisi
   recall = recall_score(y_test, y_pred, average='weighted') # Calculate weigh
   f1 = f1_score(y_test, y_pred, average='weighted') # Calculate weighted F1-s


   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 Evalution Table and Graphs

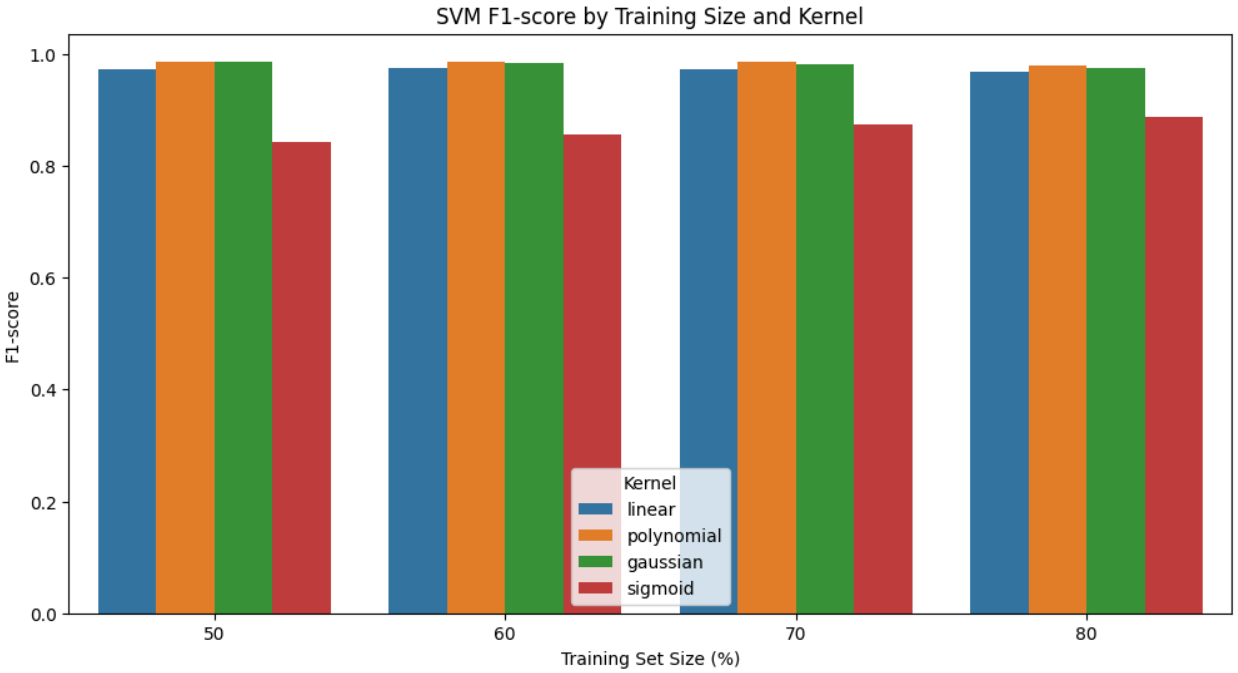
In [ ]:
```
df = pd.DataFrame(results)
```

```
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	linear
1	60	0.974792	0.974869	0.974792	0.974799	linear
2	70	0.971530	0.971745	0.971530	0.971538	linear
3	80	0.968194	0.968383	0.968194	0.968173	linear
4	50	0.985053	0.985087	0.985053	0.985055	polynomial
5	60	0.986655	0.986672	0.986655	0.986647	polynomial
6	70	0.985765	0.985794	0.985765	0.985765	polynomial
7	80	0.979760	0.979809	0.979760	0.979765	polynomial
8	50	0.986121	0.986187	0.986121	0.986130	gaussian
9	60	0.983096	0.983148	0.983096	0.983091	gaussian
10	70	0.982461	0.982488	0.982461	0.982443	gaussian
11	80	0.974867	0.975072	0.974867	0.974875	gaussian
12	50	0.841993	0.848099	0.841993	0.843197	sigmoid
13	60	0.854686	0.859502	0.854686	0.855680	sigmoid
14	70	0.872140	0.875307	0.872140	0.873104	sigmoid
15	80	0.886566	0.888754	0.886566	0.887225	sigmoid


SVM Precision by Training Size and Kernel
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid

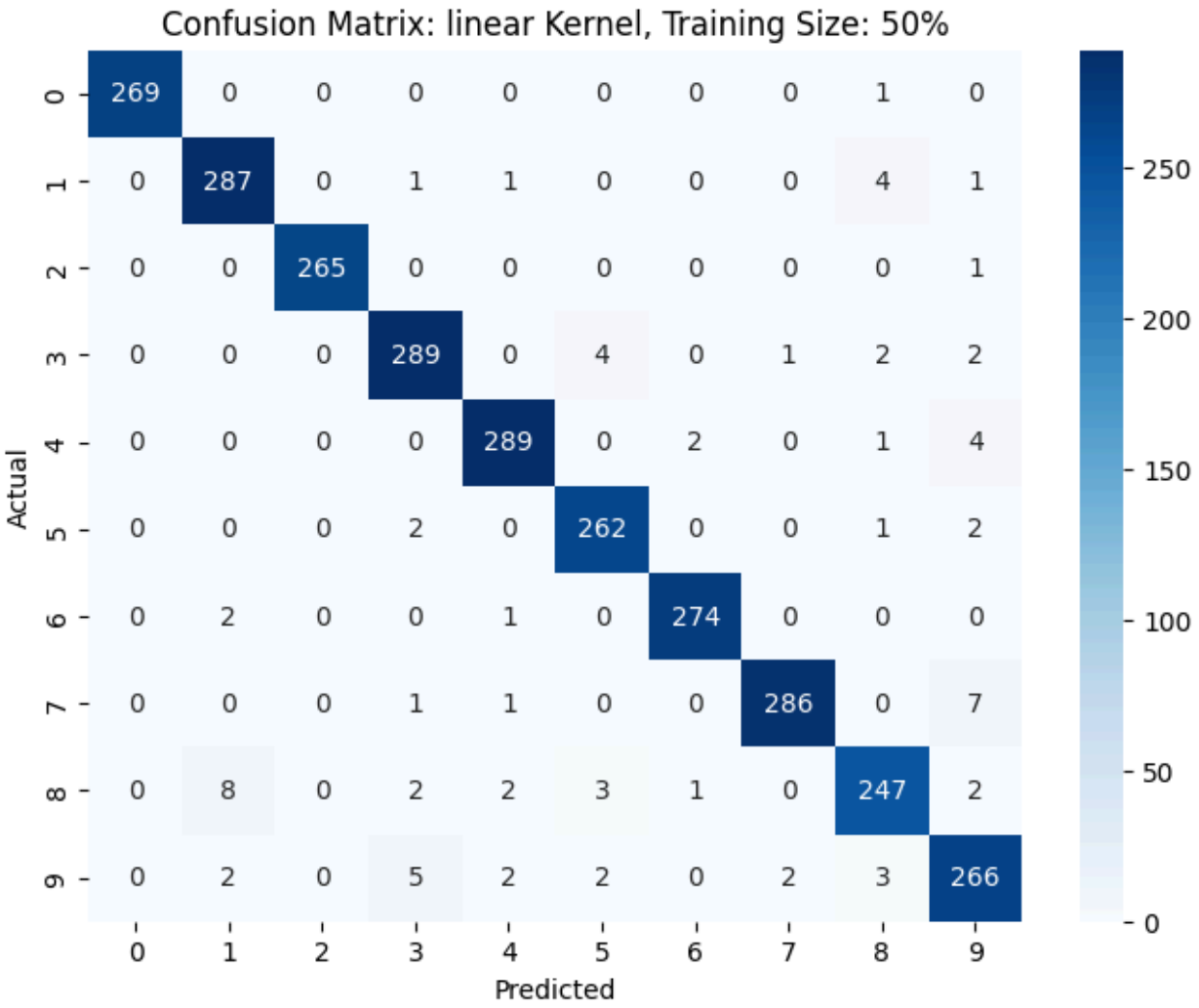

SVM Recall by Training Size and Kernel
Recall
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid

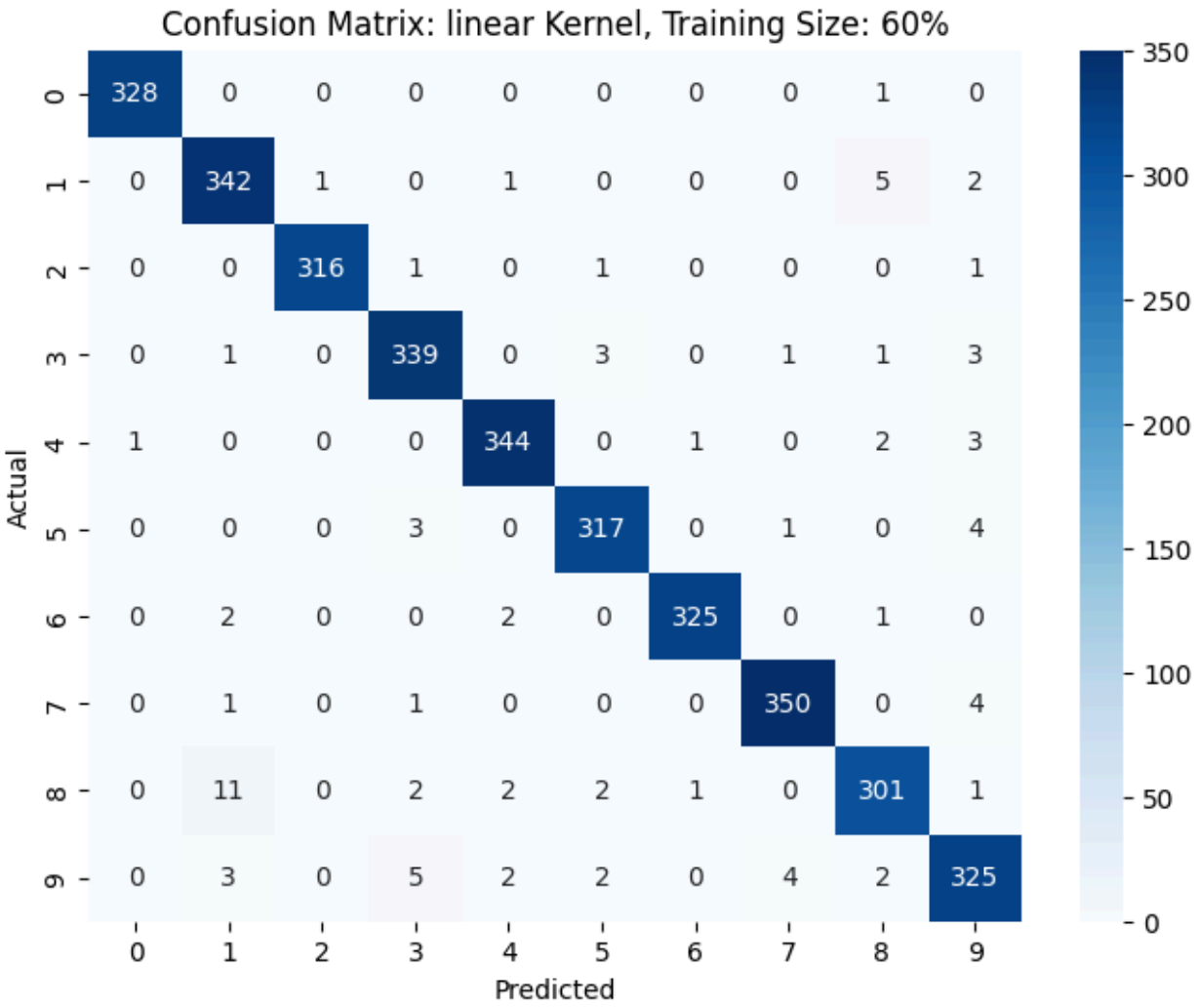


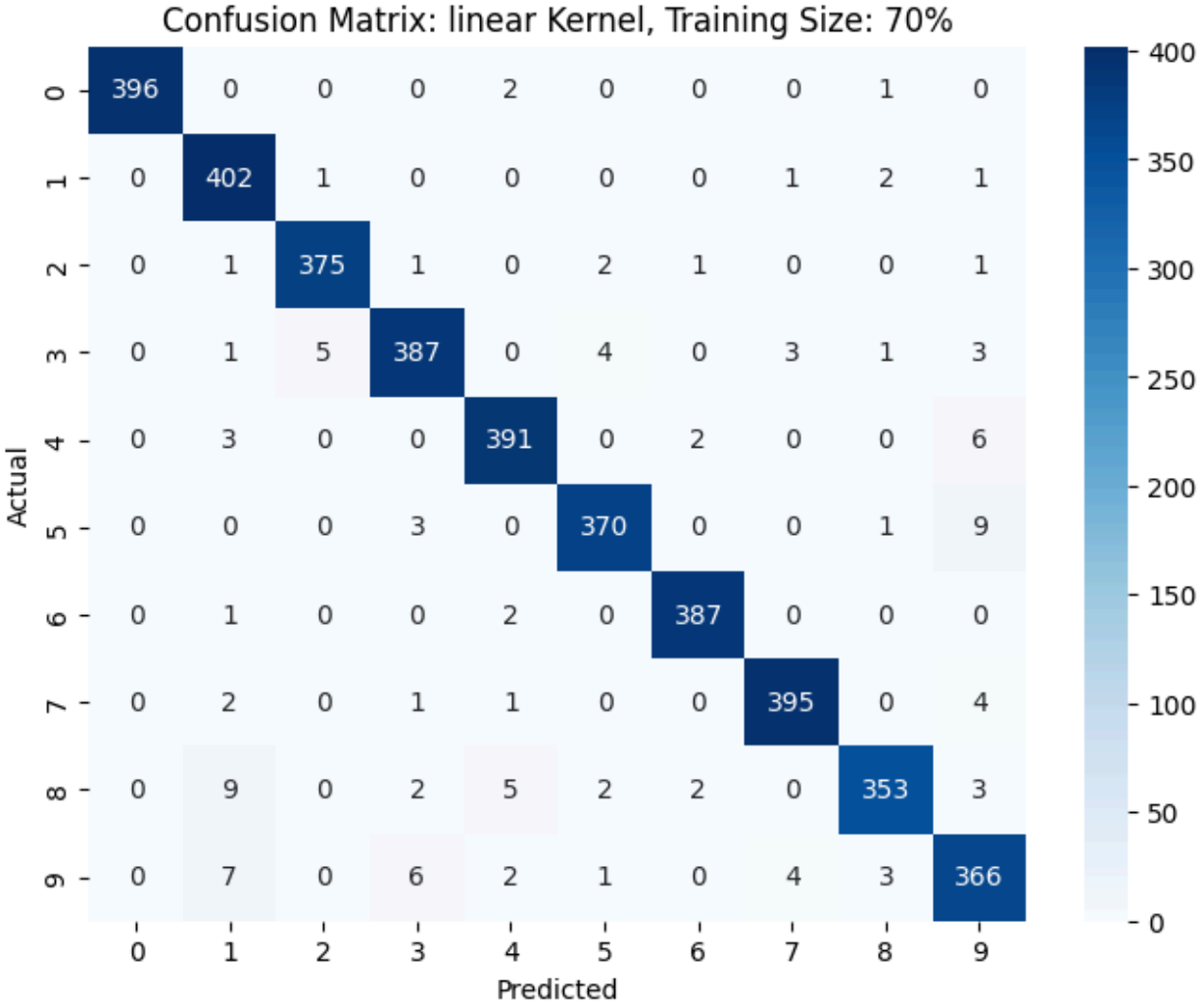
Show Confusion Matrices

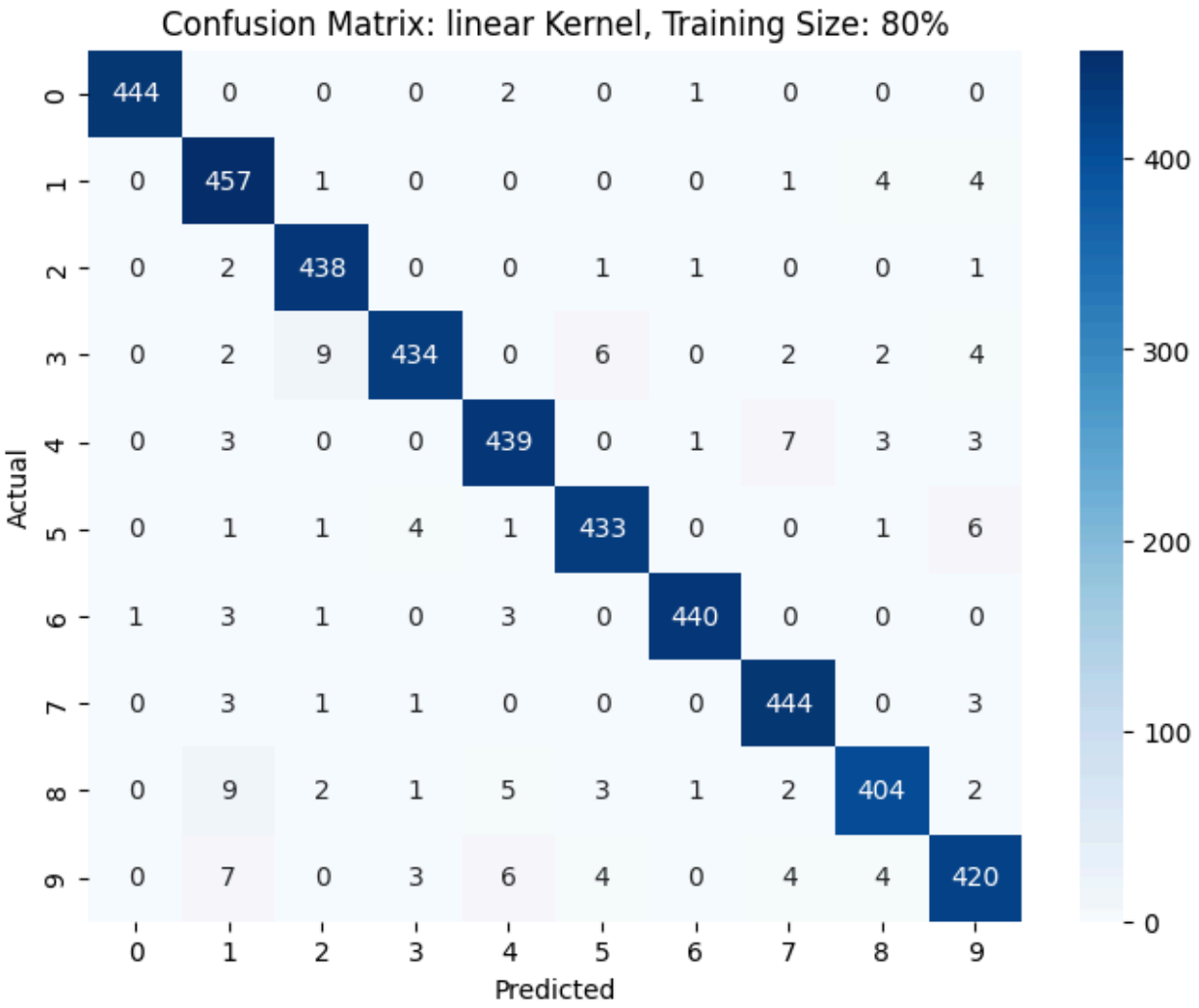
In [ ]:

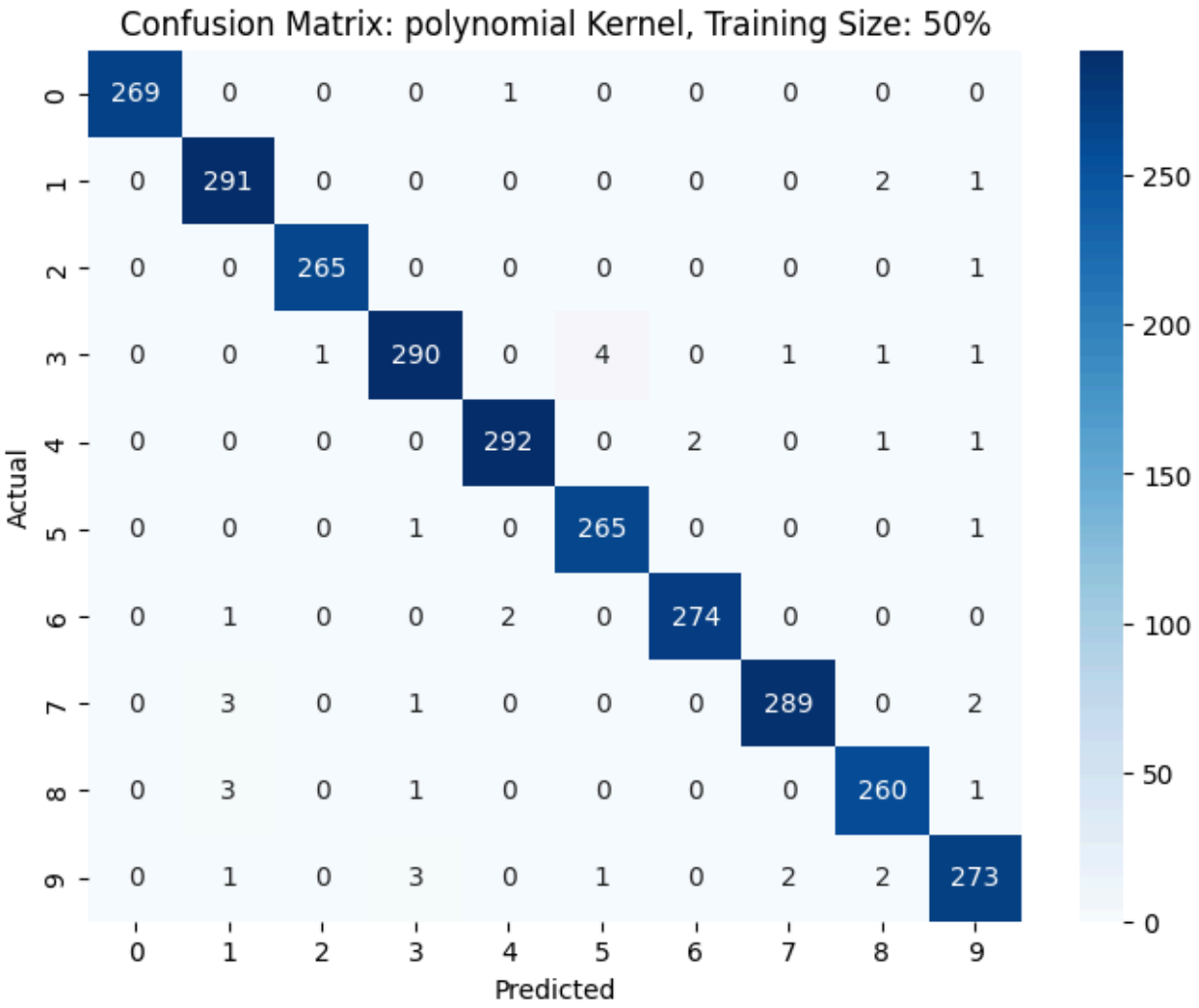
```
import matplotlib.pyplot as plt
import seaborn as sns

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: {traini
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

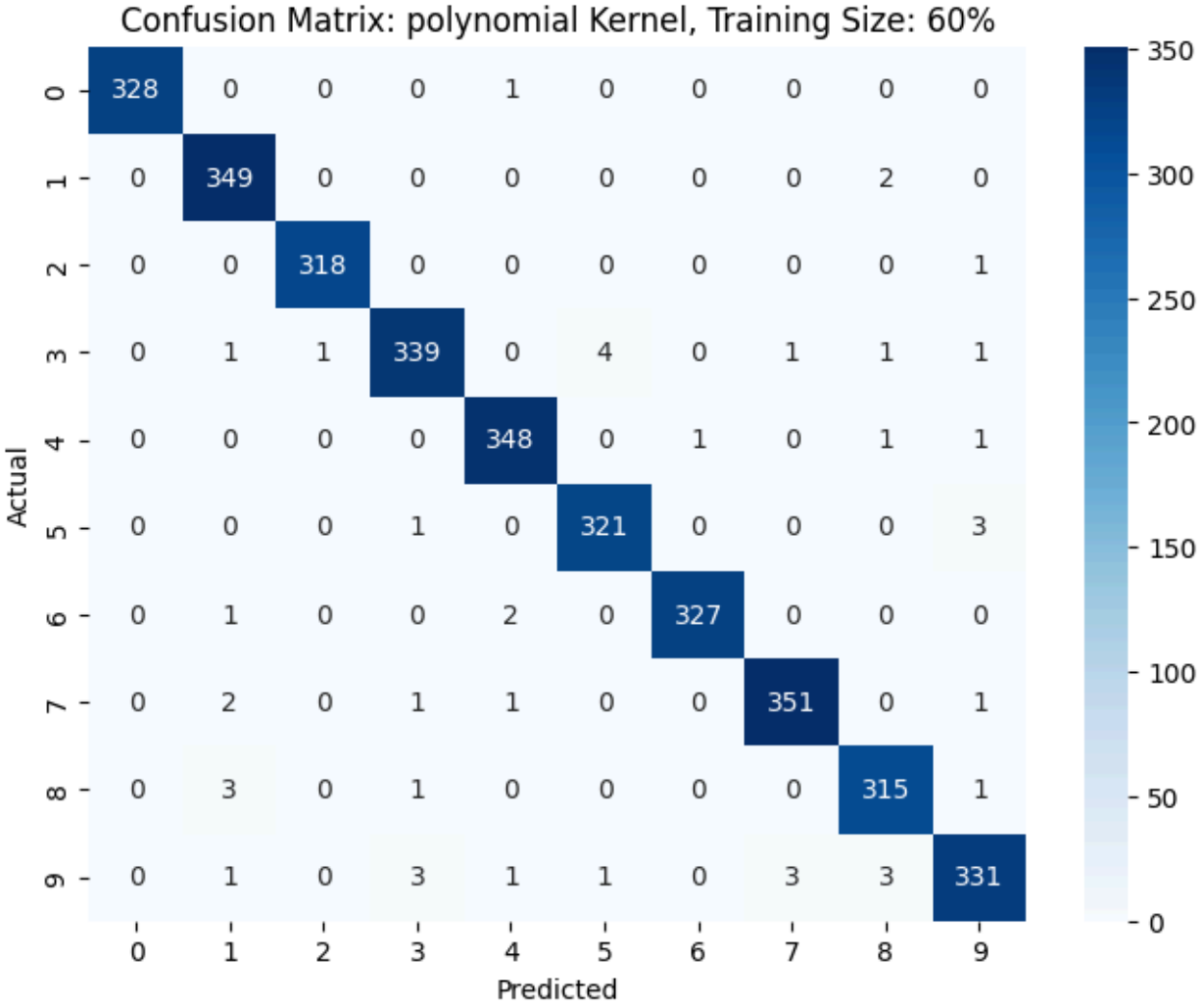

Confusion Matrix: linear Kernel, Training Size: 50%
Actual
Predicted
0 269 0 0 0 0 0 0 0 1 0
1 0 287 0 1 1 0 0 0 4 1
2 0 0 265 0 0 0 0 0 0 1
3 0 0 0 289 0 4 0 1 2 2
4 0 0 0 0 289 0 2 0 1 4
5 0 0 0 2 0 262 0 0 1 2
6 0 2 0 0 1 0 274 0 0 0
7 0 0 0 1 1 0 0 286 0 7
8 0 8 0 2 2 3 1 0 247 2
9 0 2 0 5 2 2 0 2 3 266
0 1 2 3 4 5 6 7 8 9
250
200
150
100
50
0

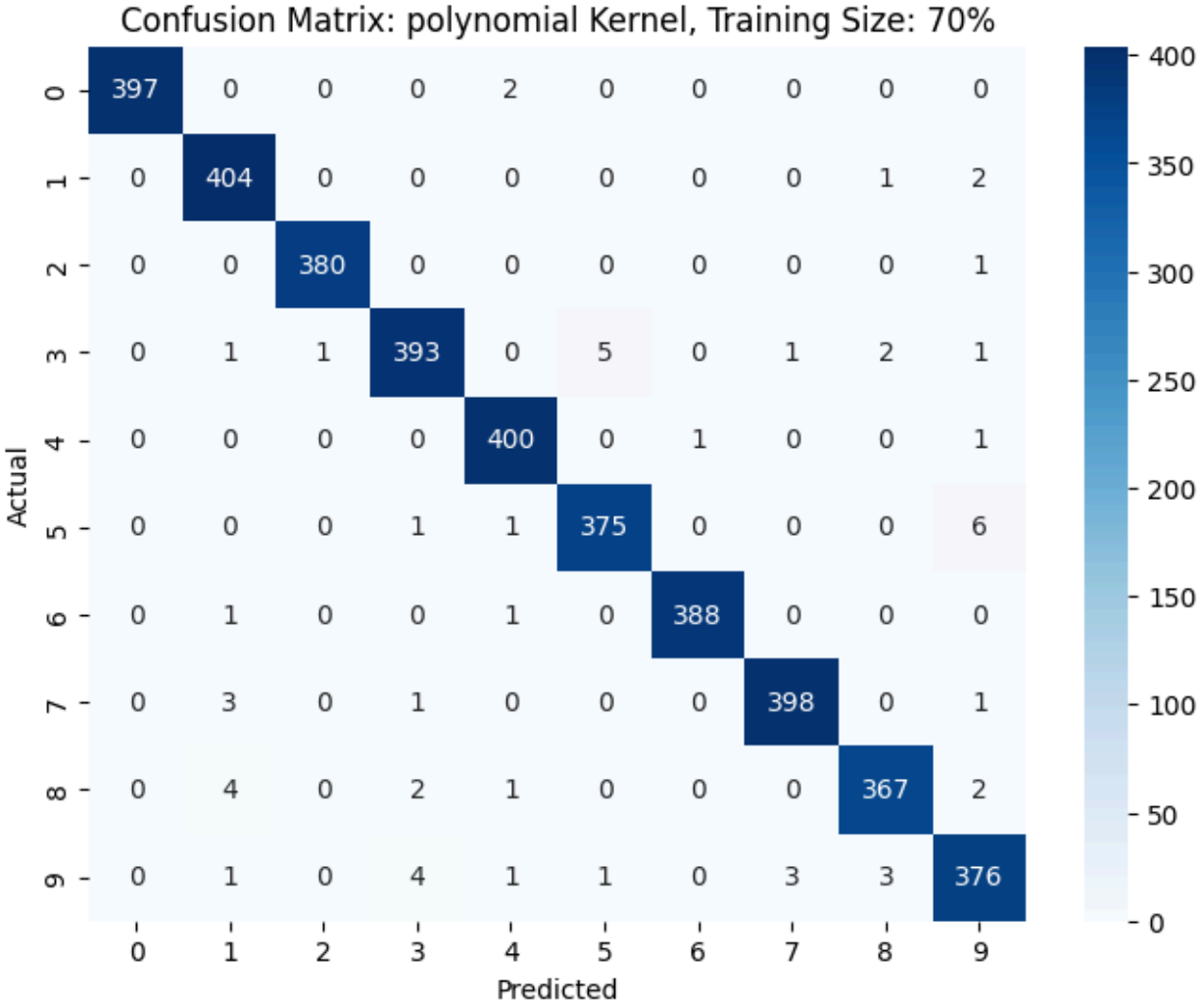

Confusion Matrix: linear Kernel, Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
328 0 0 0 0 0 0 0 1 0
0 342 1 0 1 0 0 0 5 2
0 0 316 1 0 1 0 0 0 1
0 1 0 339 0 3 0 1 1 3
1 0 0 0 344 0 1 0 2 3
0 0 0 3 0 317 0 1 0 4
0 2 0 0 2 0 325 0 1 0
0 1 0 1 0 0 0 350 0 4
0 11 0 2 2 2 1 0 301 1
0 3 0 5 2 2 0 4 2 325
350 300 250 200 150 100 50 0

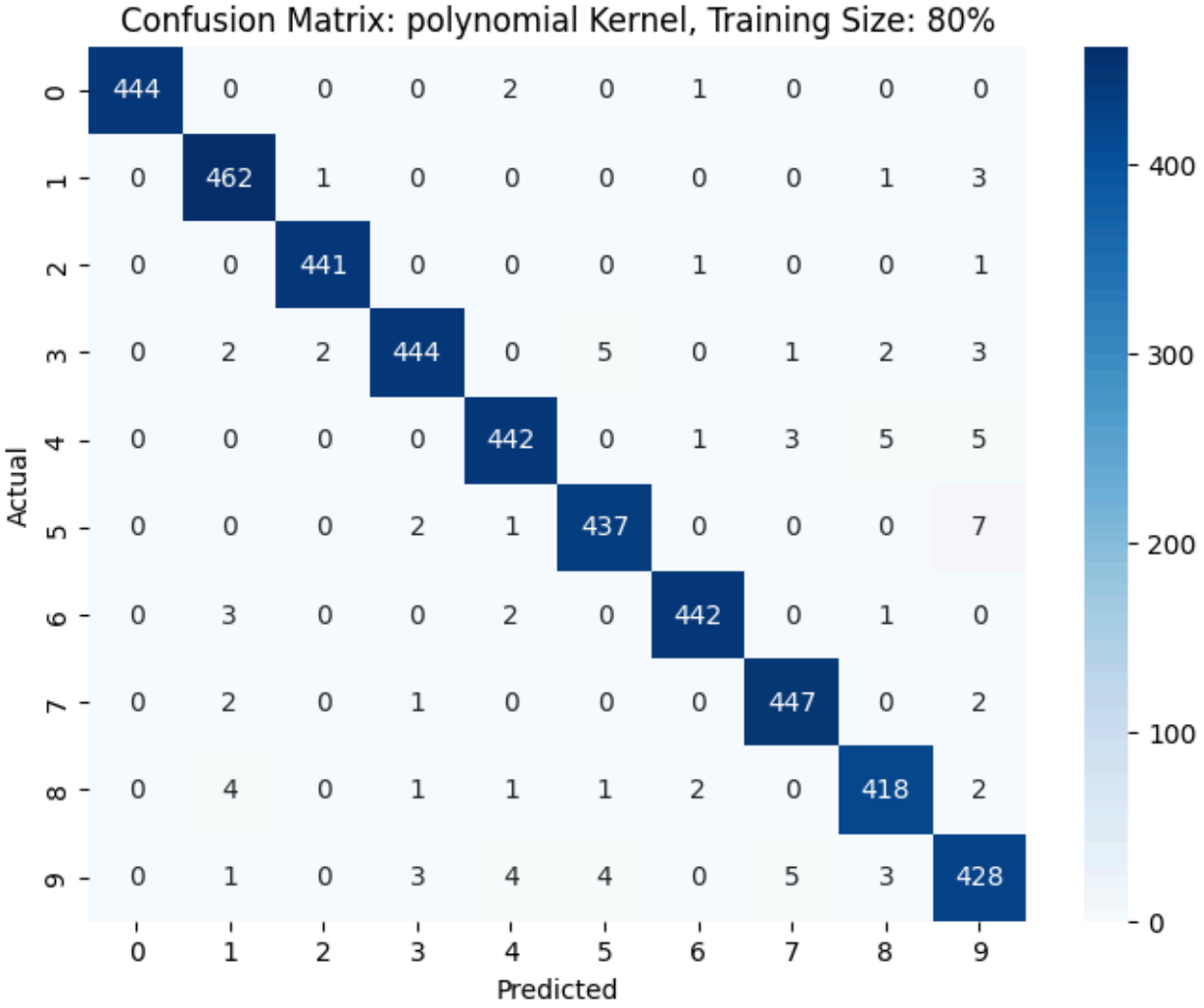

Confusion Matrix: linear Kernel, Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 396 0 0 0 2 0 0 0 1 0
1 0 402 1 0 0 0 0 1 2 1
2 0 1 375 1 0 2 1 0 0 1
3 0 1 5 387 0 4 0 3 1 3
4 0 3 0 0 391 0 2 0 0 6
5 0 0 0 3 0 370 0 0 1 9
6 0 1 0 0 2 0 387 0 0 0
7 0 2 0 1 1 0 0 395 0 4
8 0 9 0 2 5 2 2 0 353 3
9 0 7 0 6 2 1 0 4 3 366
400
350
300
250
200
150
100
50
0

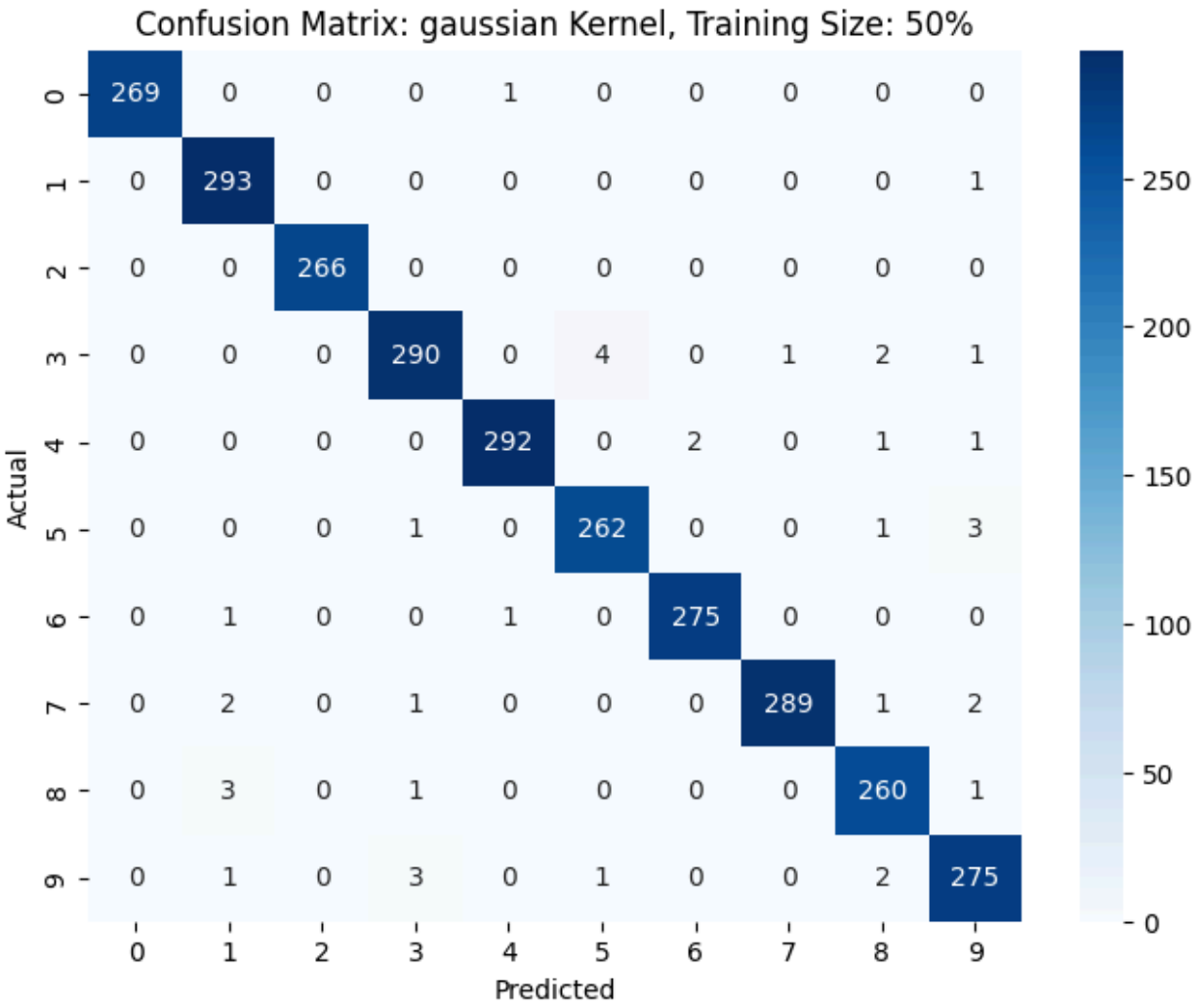

Confusion Matrix: linear Kernel, Training Size: 80%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
444 0 0 0 2 0 1 0 0 0
0 457 1 0 0 0 0 1 4 4
0 2 438 0 0 1 1 0 0 1
0 2 9 434 0 6 0 2 2 4
0 3 0 0 439 0 1 7 3 3
0 1 1 4 1 433 0 0 1 6
1 3 1 0 3 0 440 0 0 0
0 3 1 1 0 0 0 444 0 3
0 9 2 1 5 3 1 2 404 2
0 7 0 3 6 4 0 4 4 420
400
300
200
100
0

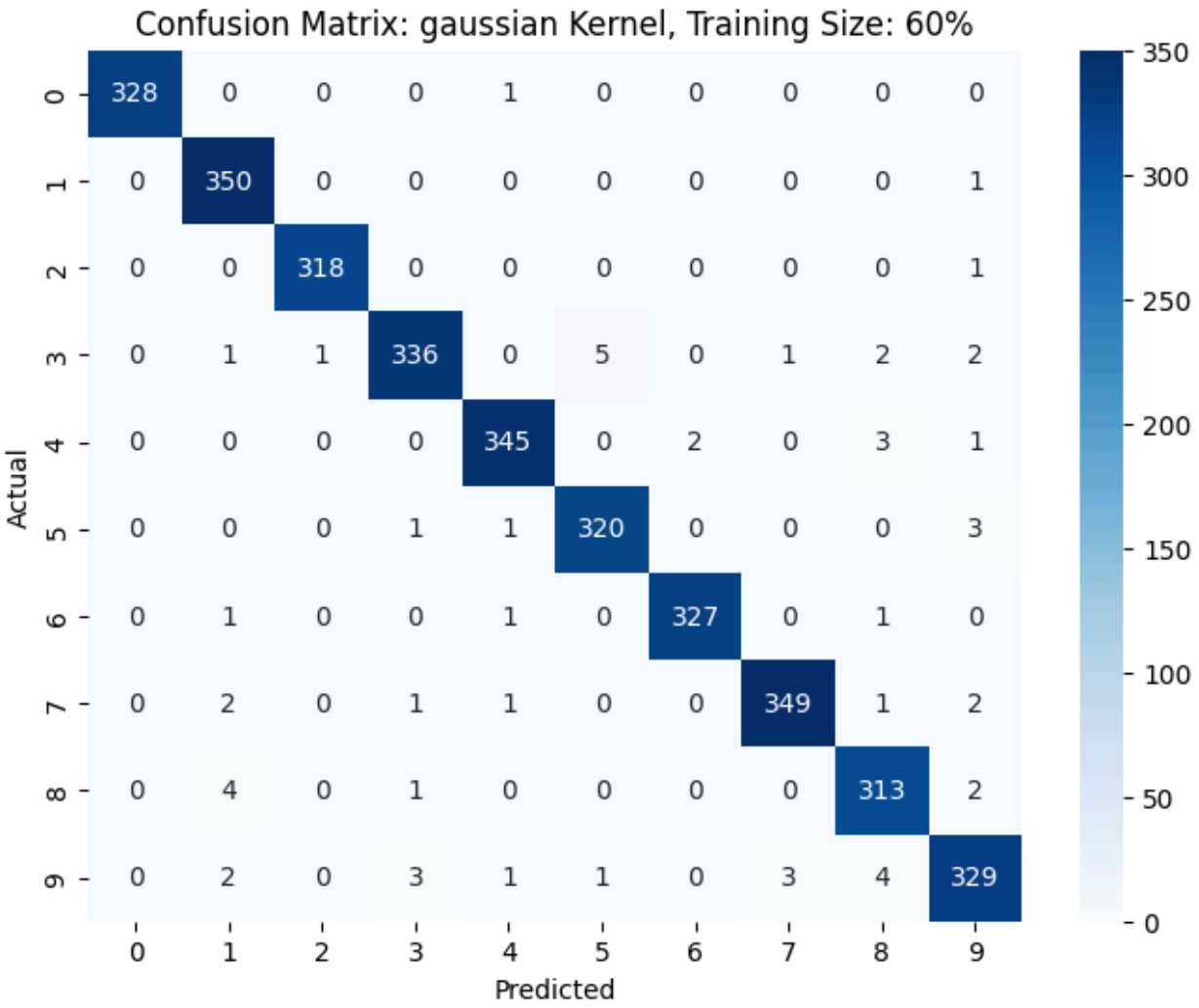

Confusion Matrix: polynomial Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
269 0 0 0 1 0 0 0 0 0
0 291 0 0 0 0 0 0 2 1
0 0 265 0 0 0 0 0 0 1
0 0 1 290 0 4 0 1 1 1
0 0 0 0 292 0 2 0 1 1
0 0 0 1 0 265 0 0 0 1
0 1 0 0 2 0 274 0 0 0
0 3 0 1 0 0 0 289 0 2
0 3 0 1 0 0 0 0 260 1
0 1 0 3 0 1 0 2 2 273
250
200
150
100
50
0

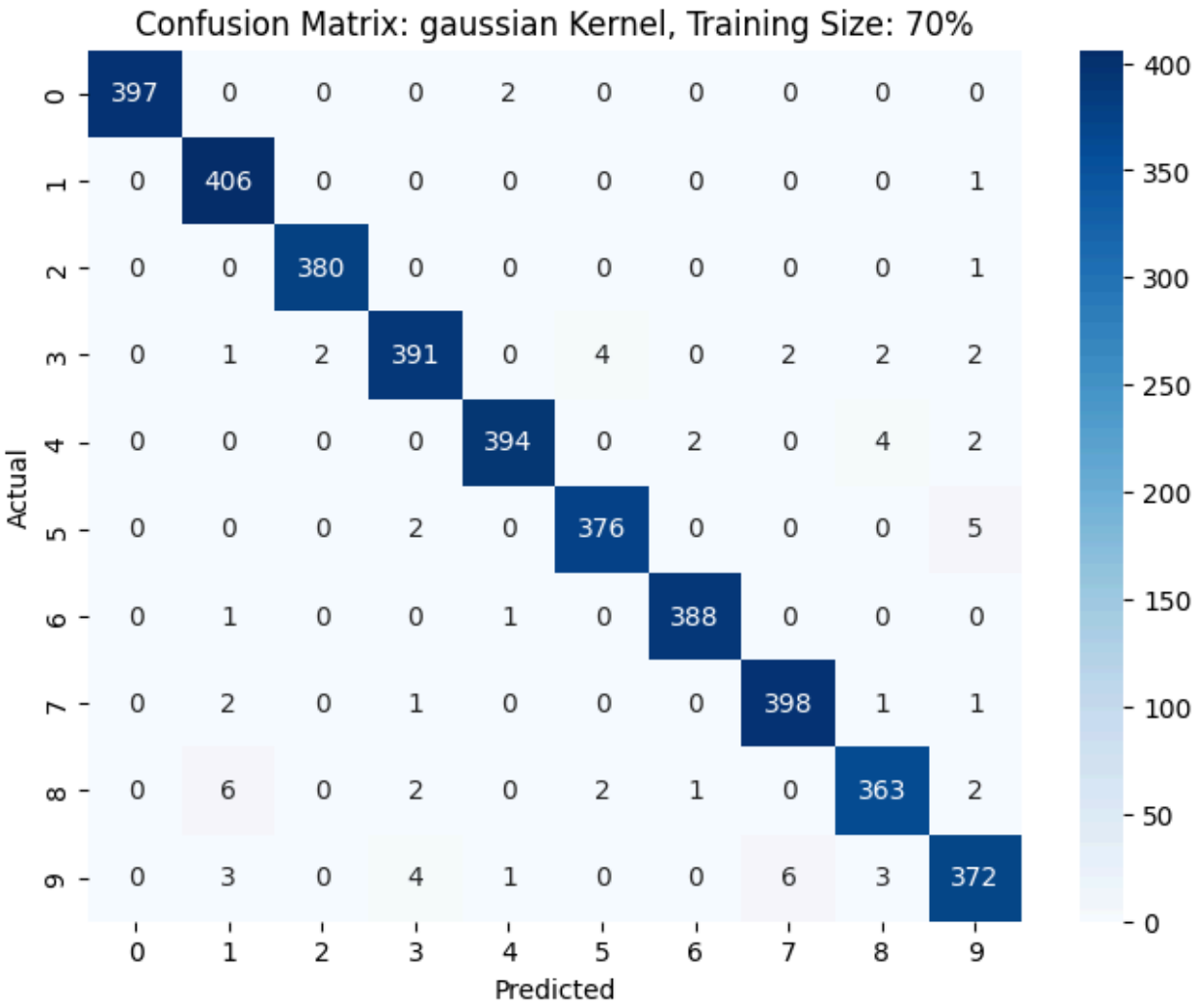

Confusion Matrix: polynomial Kernel, Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
328 0 0 0 1 0 0 0 0 0
0 349 0 0 0 0 0 0 2 0
0 0 318 0 0 0 0 0 0 1
0 1 1 339 0 4 0 1 1 1
0 0 0 0 348 0 1 0 1 1
0 0 0 1 0 321 0 0 0 3
0 1 0 0 2 0 327 0 0 0
0 2 0 1 1 0 0 351 0 1
0 3 0 1 0 0 0 0 315 1
0 1 0 3 1 1 0 3 3 331
350 300 250 200 150 100 50 0

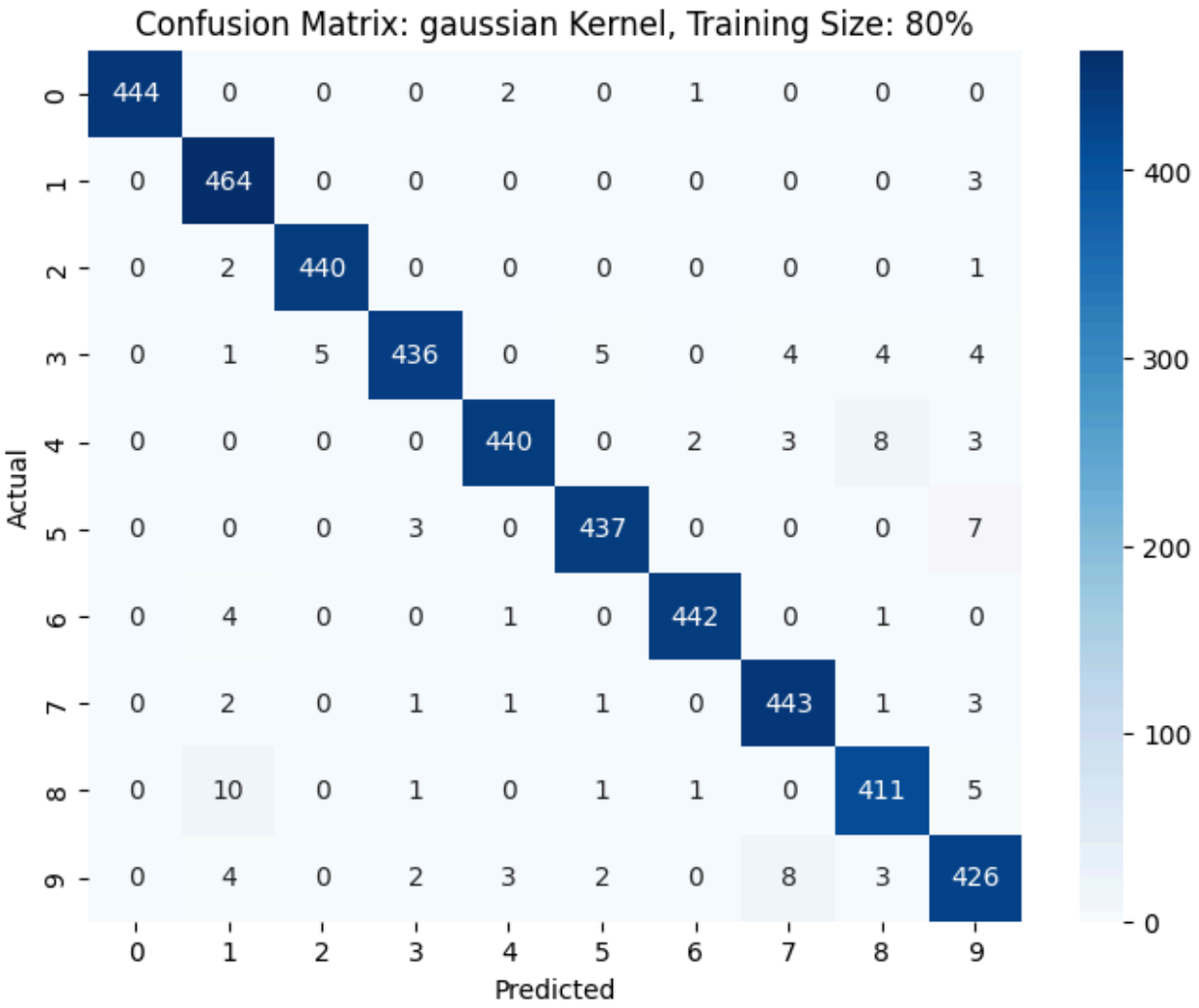

Confusion Matrix: polynomial Kernel, Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 397 0 0 0 2 0 0 0 0 0
1 0 404 0 0 0 0 0 0 1 2
2 0 0 380 0 0 0 0 0 0 1
3 0 1 1 393 0 5 0 1 2 1
4 0 0 0 0 400 0 1 0 0 1
5 0 0 0 1 1 375 0 0 0 6
6 0 1 0 0 1 0 388 0 0 0
7 0 3 0 1 0 0 0 398 0 1
8 0 4 0 2 1 0 0 0 367 2
9 0 1 0 4 1 1 0 3 3 376
400 350 300 250 200 150 100 50 0

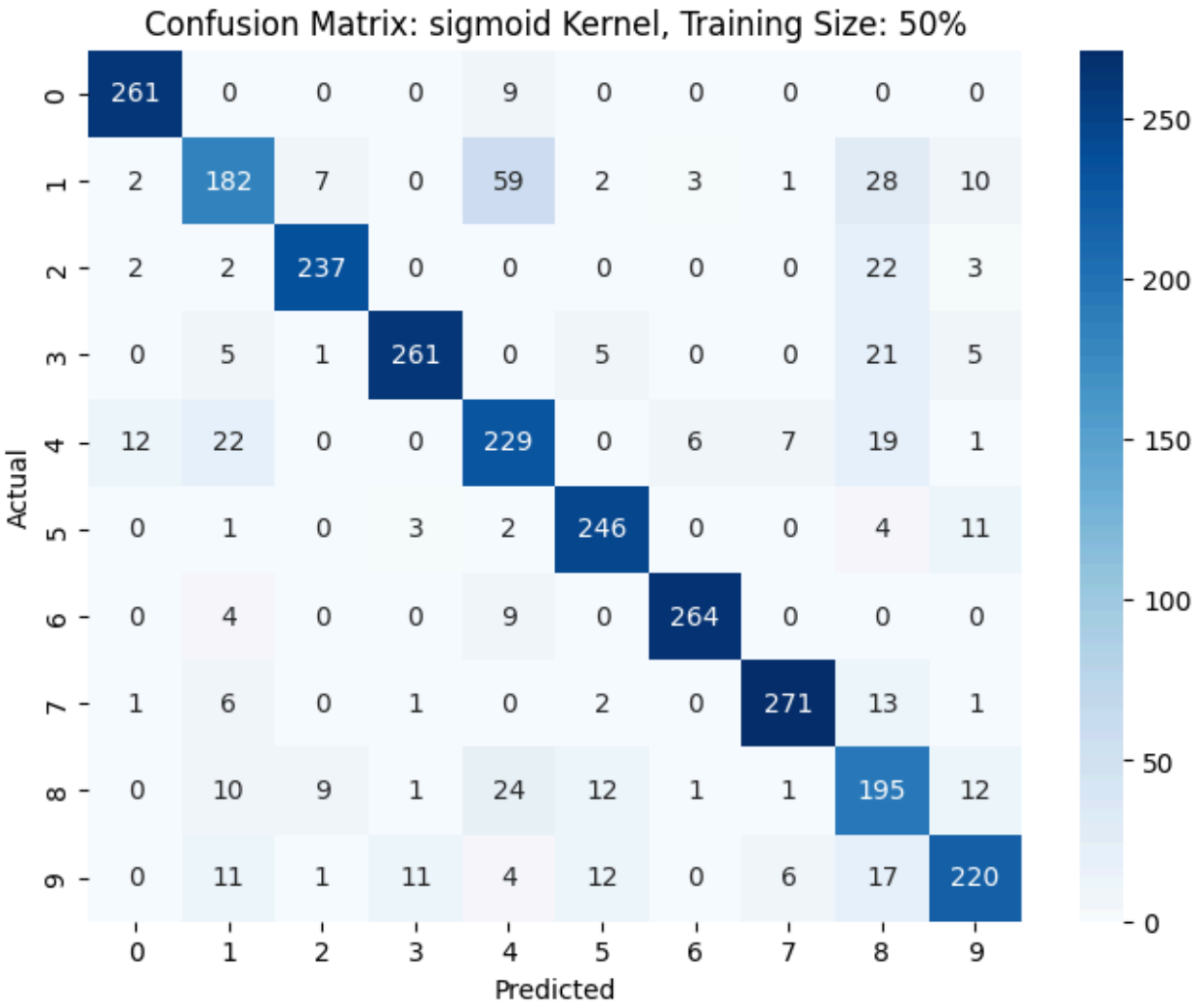

Confusion Matrix: polynomial Kernel, Training Size: 80%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
444 0 0 0 2 0 1 0 0 0
0 462 1 0 0 0 0 0 1 3
0 0 441 0 0 0 1 0 0 1
0 2 2 444 0 5 0 1 2 3
0 0 0 0 442 0 1 3 5 5
0 0 0 2 1 437 0 0 0 7
0 3 0 0 2 0 442 0 1 0
0 2 0 1 0 0 0 447 0 2
0 4 0 1 1 1 2 0 418 2
0 1 0 3 4 4 0 5 3 428
400
300
200
100
0

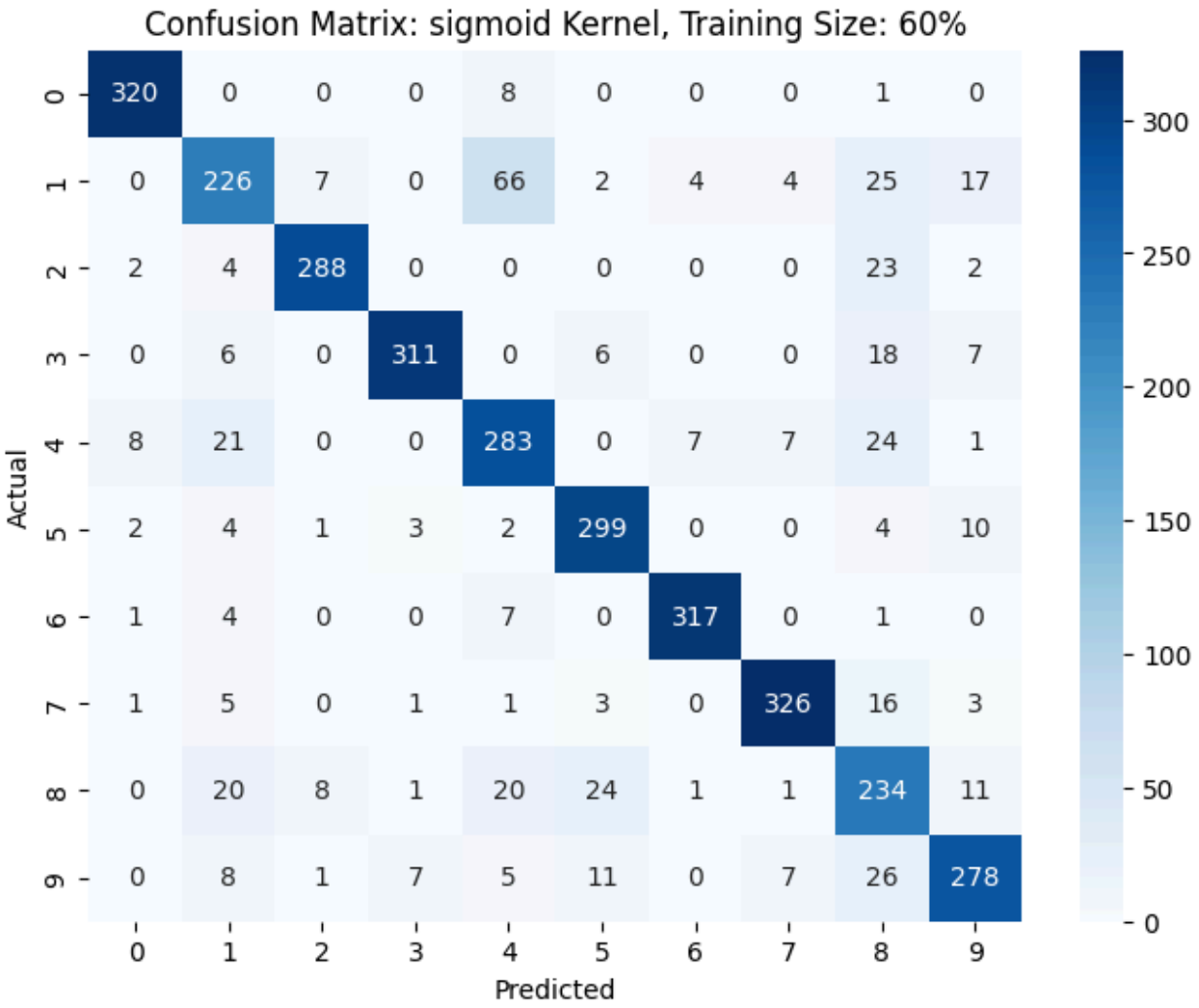

Confusion Matrix: gaussian Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
269 0 0 0 1 0 0 0 0 0
0 293 0 0 0 0 0 0 0 1
0 0 266 0 0 0 0 0 0 0
0 0 0 290 0 4 0 1 2 1
0 0 0 0 292 0 2 0 1 1
0 0 0 1 0 262 0 0 1 3
0 1 0 0 1 0 275 0 0 0
0 2 0 1 0 0 0 289 1 2
0 3 0 1 0 0 0 0 260 1
0 1 0 3 0 1 0 0 2 275
250
200
150
100
50
0

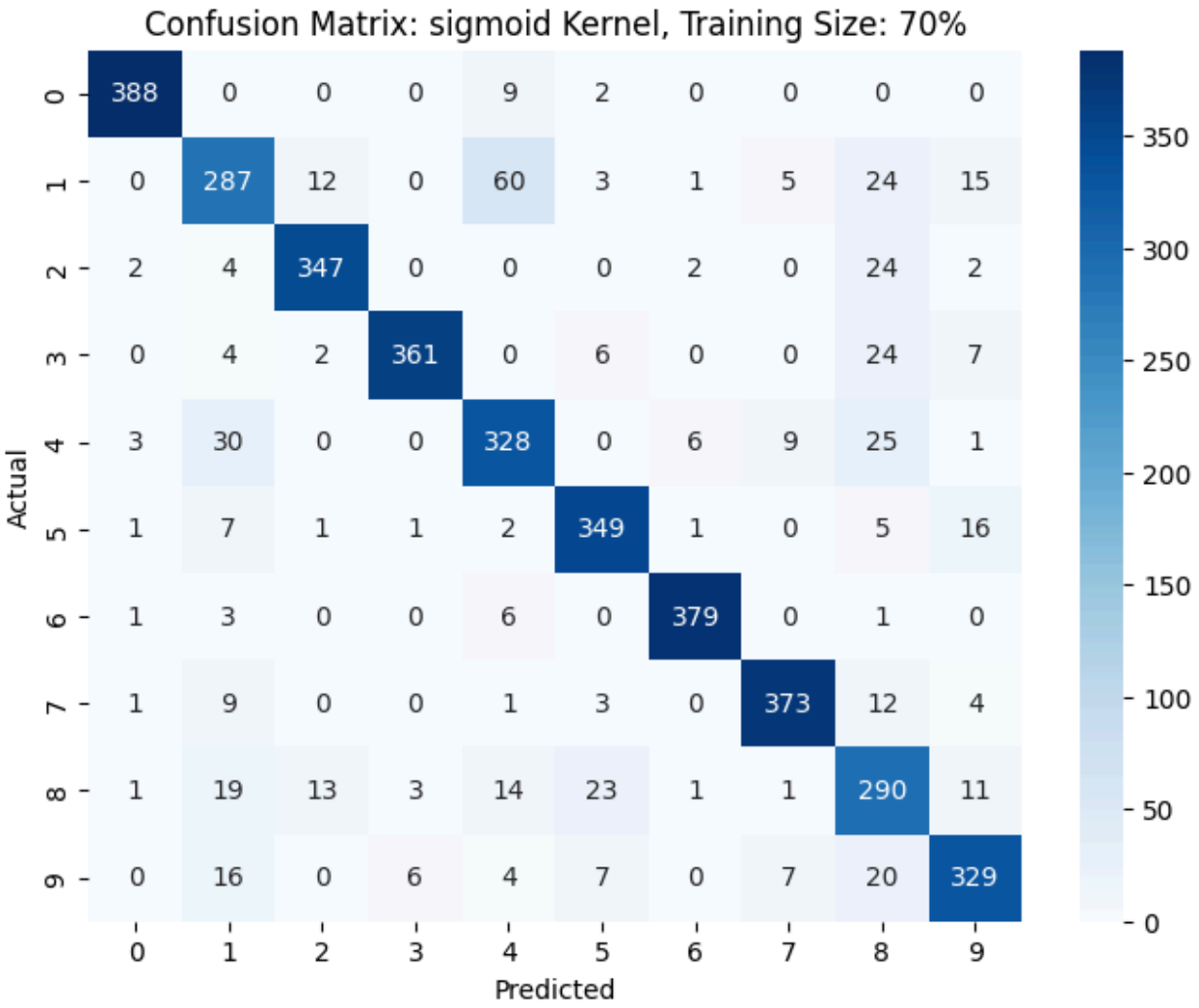

Confusion Matrix: gaussian Kernel, Training Size: 60%
Actual
Predicted
0 328 0 0 0 1 0 0 0 0 0
1 0 350 0 0 0 0 0 0 0 1
2 0 0 318 0 0 0 0 0 0 1
3 0 1 1 336 0 5 0 1 2 2
4 0 0 0 0 345 0 2 0 3 1
5 0 0 0 1 1 320 0 0 0 3
6 0 1 0 0 1 0 327 0 1 0
7 0 2 0 1 1 0 0 349 1 2
8 0 4 0 1 0 0 0 0 313 2
9 0 2 0 3 1 1 0 3 4 329
0 1 2 3 4 5 6 7 8 9
350
300
250
200
150
100
50
0

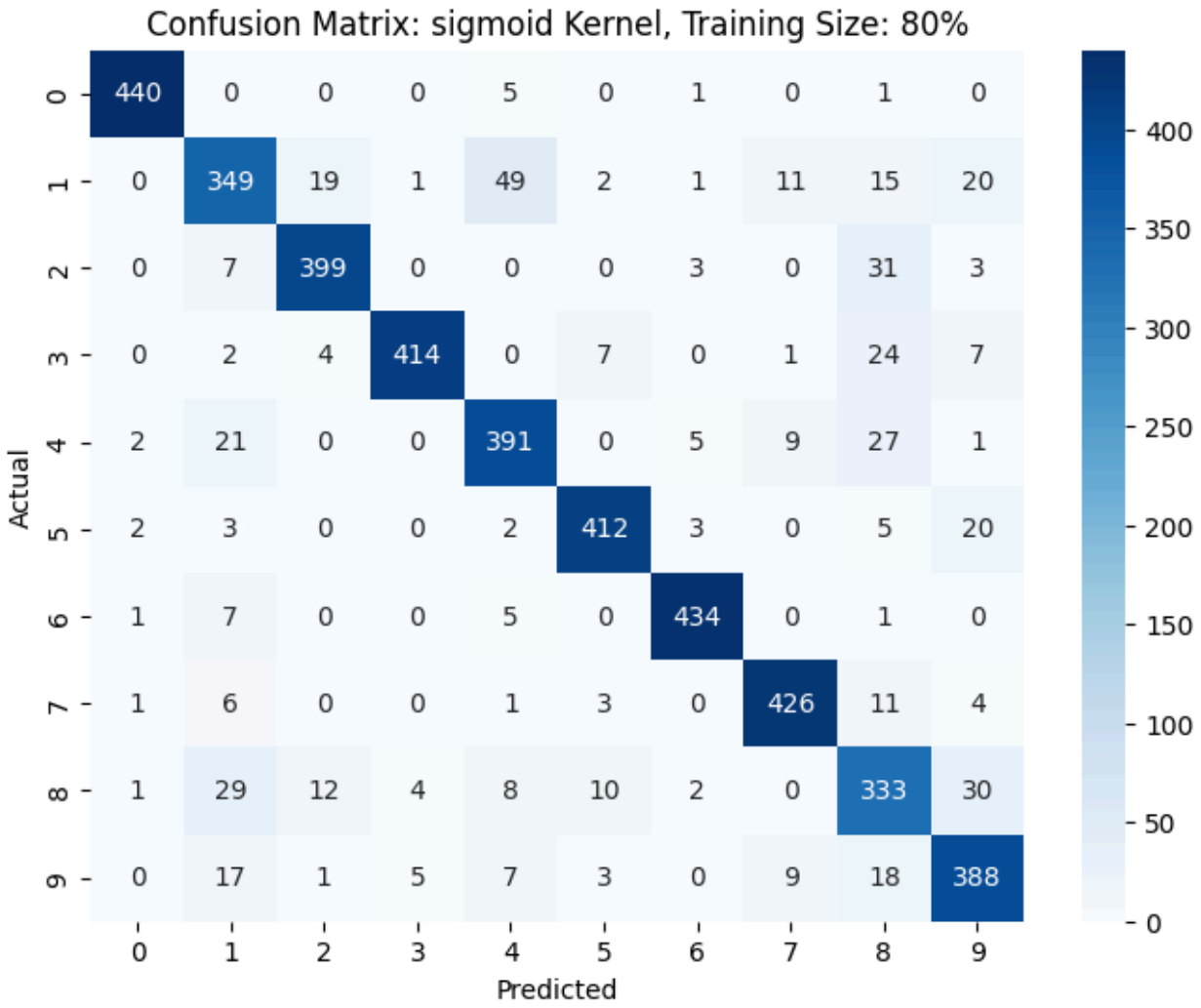

Confusion Matrix: gaussian Kernel, Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 397 0 0 0 2 0 0 0 0 0
1 0 406 0 0 0 0 0 0 0 1
2 0 0 380 0 0 0 0 0 0 1
3 0 1 2 391 0 4 0 2 2 2
4 0 0 0 0 394 0 2 0 4 2
5 0 0 0 2 0 376 0 0 0 5
6 0 1 0 0 1 0 388 0 0 0
7 0 2 0 1 0 0 0 398 1 1
8 0 6 0 2 0 2 1 0 363 2
9 0 3 0 4 1 0 0 6 3 372
400 350 300 250 200 150 100 50 0


Confusion Matrix: gaussian Kernel, Training Size: 80%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
444 0 0 0 2 0 1 0 0 0
0 464 0 0 0 0 0 0 0 3
0 2 440 0 0 0 0 0 0 1
0 1 5 436 0 5 0 4 4 4
0 0 0 0 440 0 2 3 8 3
0 0 0 3 0 437 0 0 0 7
0 4 0 0 1 0 442 0 1 0
0 2 0 1 1 1 0 443 1 3
0 10 0 1 0 1 1 0 411 5
0 4 0 2 3 2 0 8 3 426
400
300
200
100
0


Confusion Matrix: sigmoid Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
261 0 0 0 9 0 0 0 0 0
2 182 7 0 59 2 3 1 28 10
2 2 237 0 0 0 0 0 22 3
0 5 1 261 0 5 0 0 21 5
12 22 0 0 229 0 6 7 19 1
0 1 0 3 2 246 0 0 4 11
0 4 0 0 9 0 264 0 0 0
1 6 0 1 0 2 0 271 13 1
0 10 9 1 24 12 1 1 195 12
0 11 1 11 4 12 0 6 17 220
250
200
150
100
50
0


Confusion Matrix: sigmoid Kernel, Training Size: 60%
Actual
Predicted
0 320 0 0 0 8 0 0 0 1 0
1 0 226 7 0 66 2 4 4 25 17
2 2 4 288 0 0 0 0 0 23 2
3 0 6 0 311 0 6 0 0 18 7
4 8 21 0 0 283 0 7 7 24 1
5 2 4 1 3 2 299 0 0 4 10
6 1 4 0 0 7 0 317 0 1 0
7 1 5 0 1 1 3 0 326 16 3
8 0 20 8 1 20 24 1 1 234 11
9 0 8 1 7 5 11 0 7 26 278
0 1 2 3 4 5 6 7 8 9
300
250
200
150
100
50
0


Confusion Matrix: sigmoid Kernel, Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
388 0 0 0 9 2 0 0 0 0
0 287 12 0 60 3 1 5 24 15
2 4 347 0 0 0 2 0 24 2
0 4 2 361 0 6 0 0 24 7
3 30 0 0 328 0 6 9 25 1
1 7 1 1 2 349 1 0 5 16
1 3 0 0 6 0 379 0 1 0
1 9 0 0 1 3 0 373 12 4
1 19 13 3 14 23 1 1 290 11
0 16 0 6 4 7 0 7 20 329
350
300
250
200
150
100
50
0



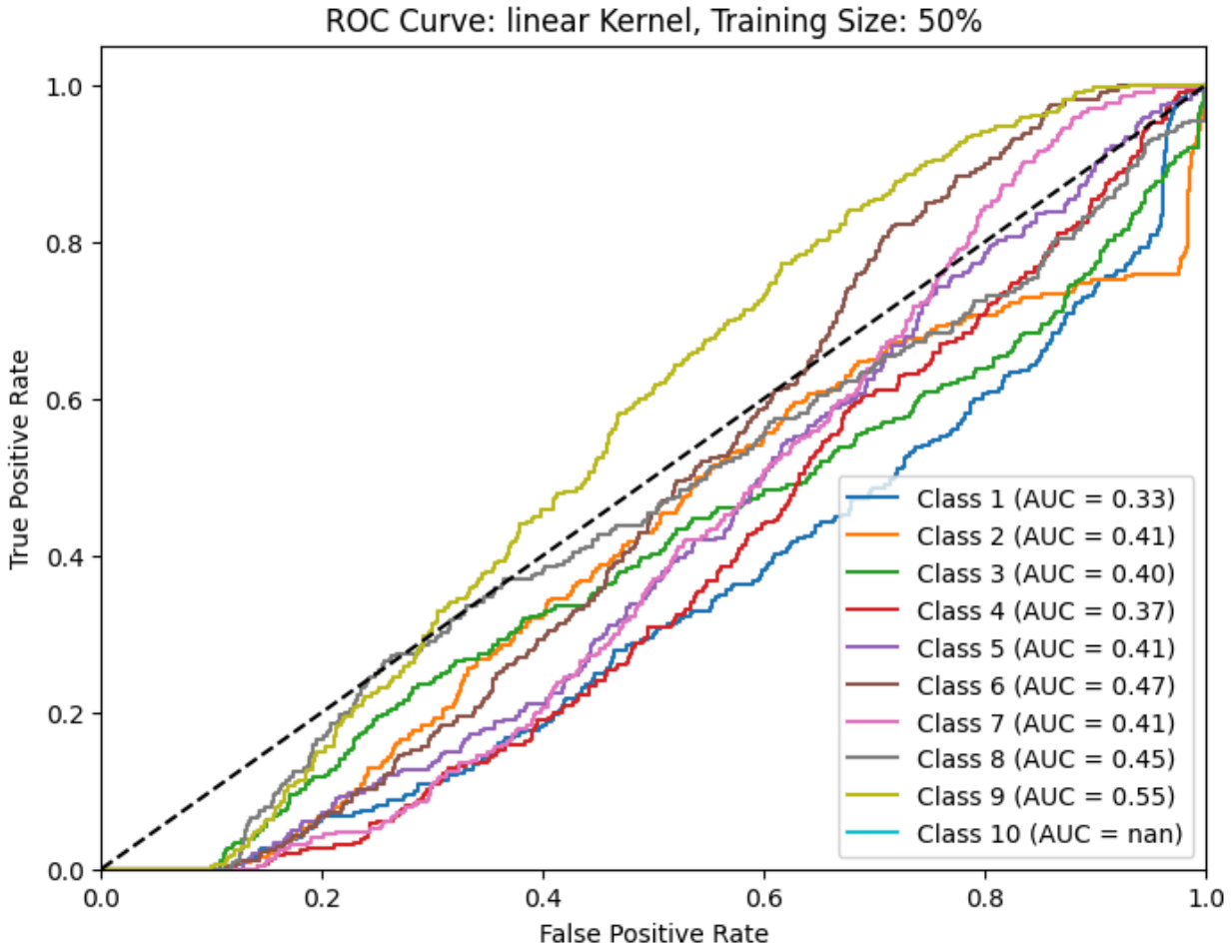
ROC and AOC curve

In [ ]:

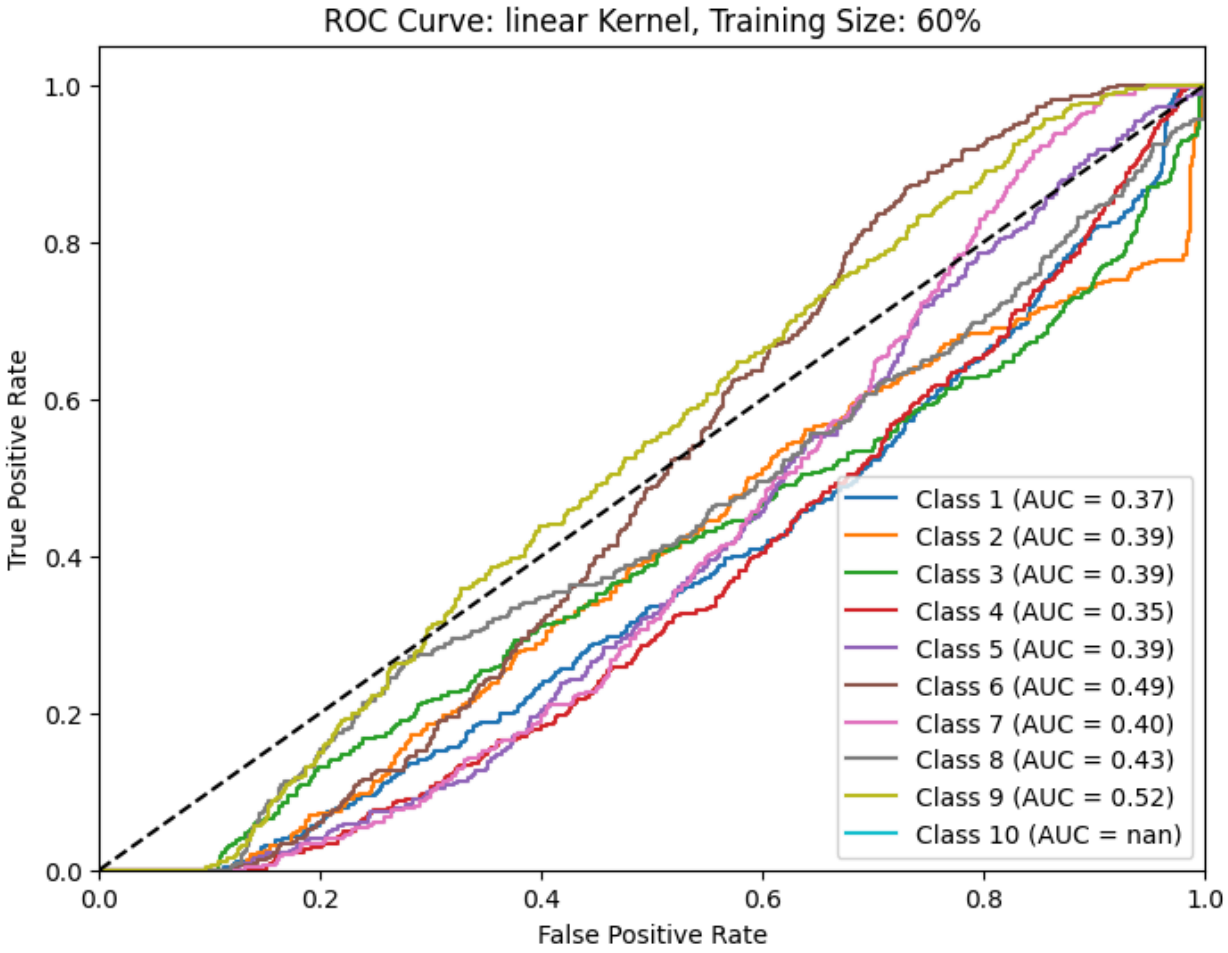
```
from sklearn.metrics import roc_curve, auc

# 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()
```

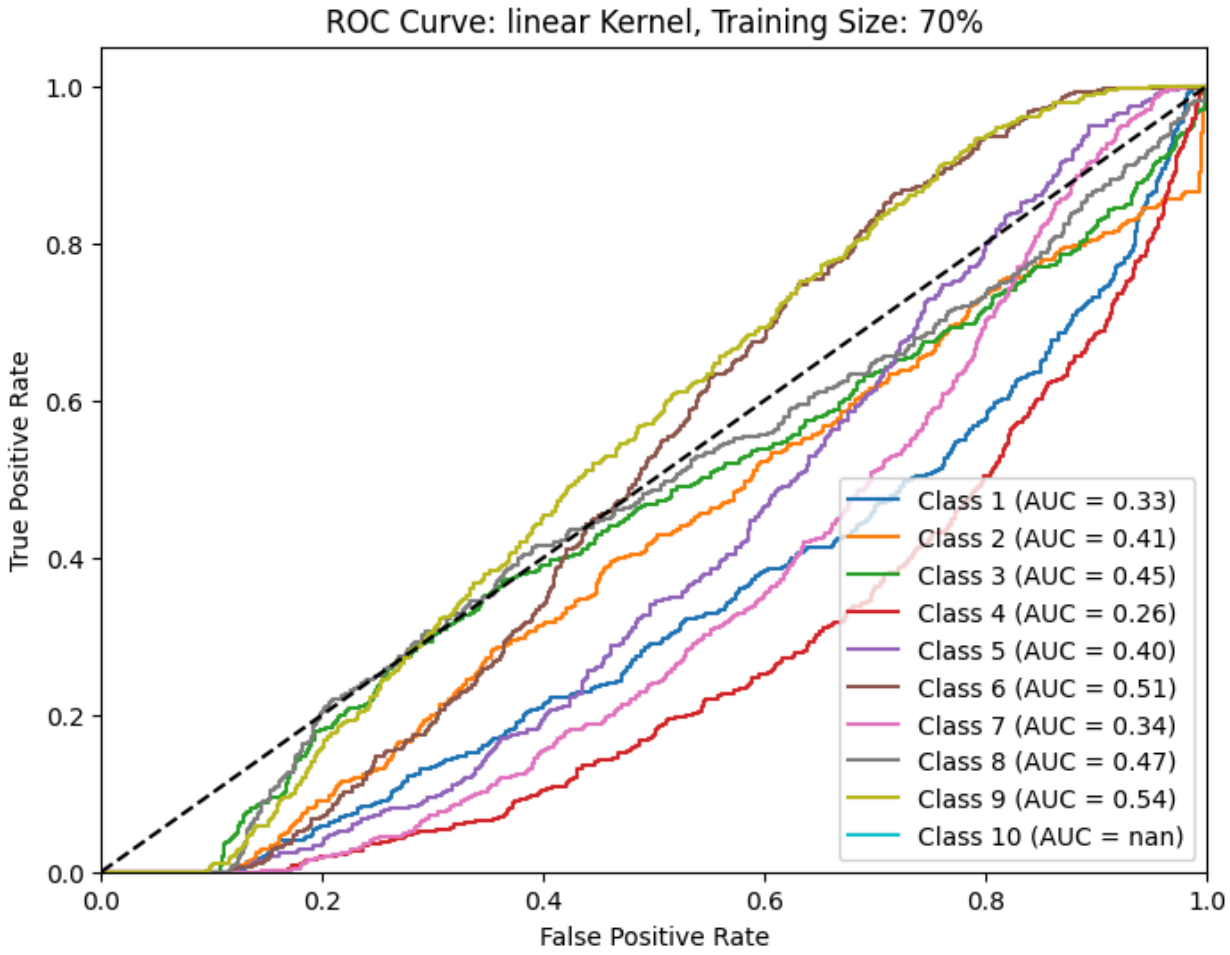
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



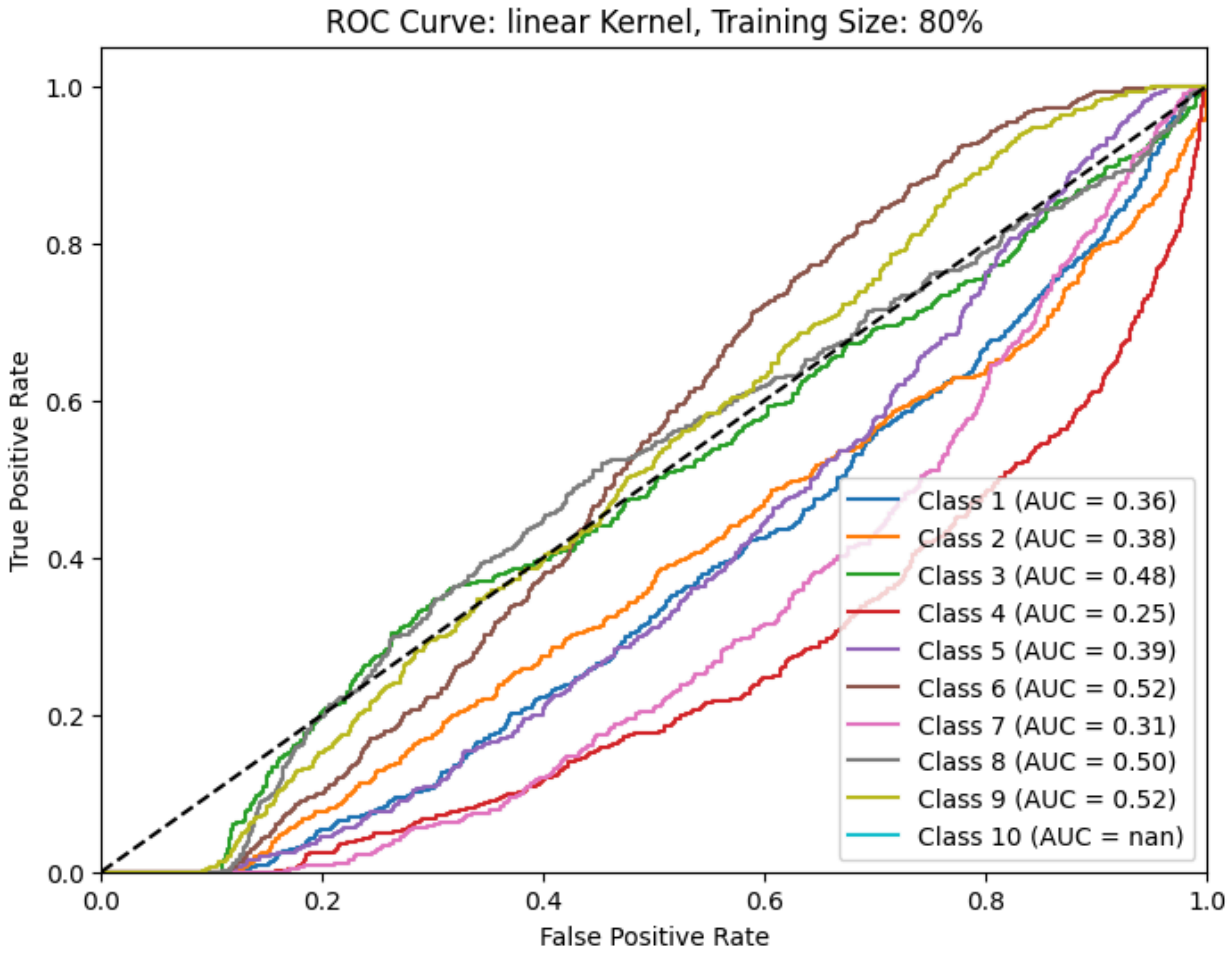
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



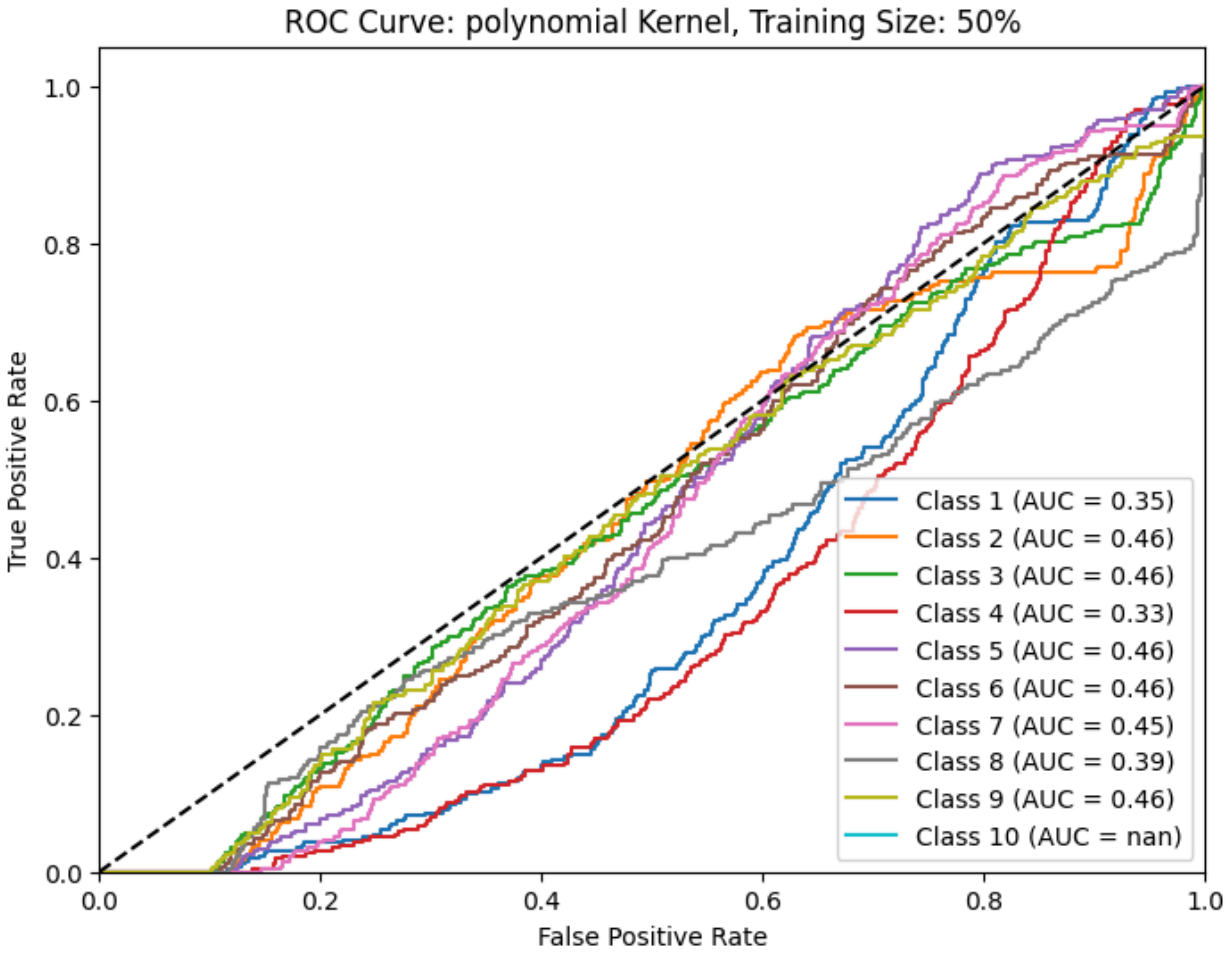

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



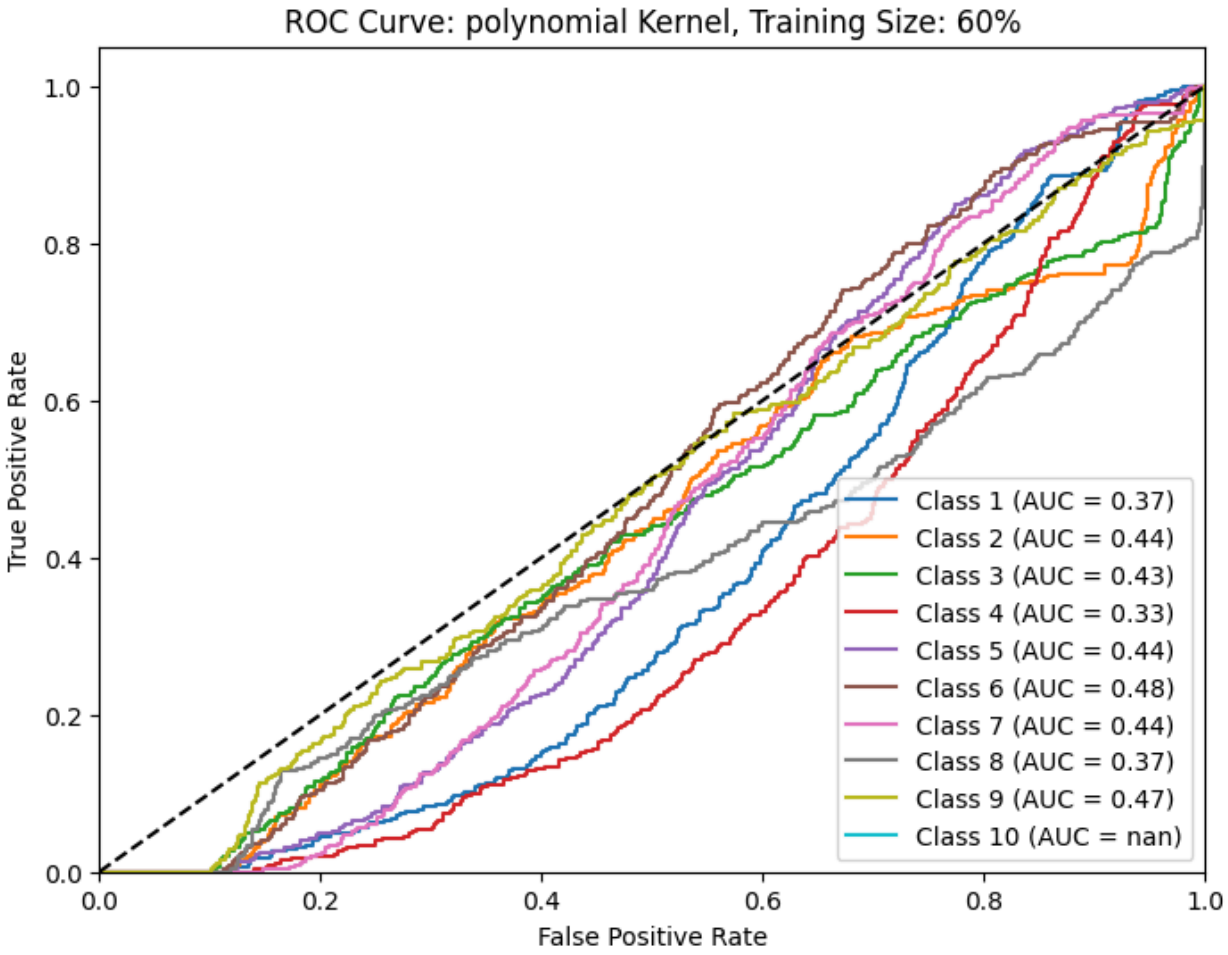

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



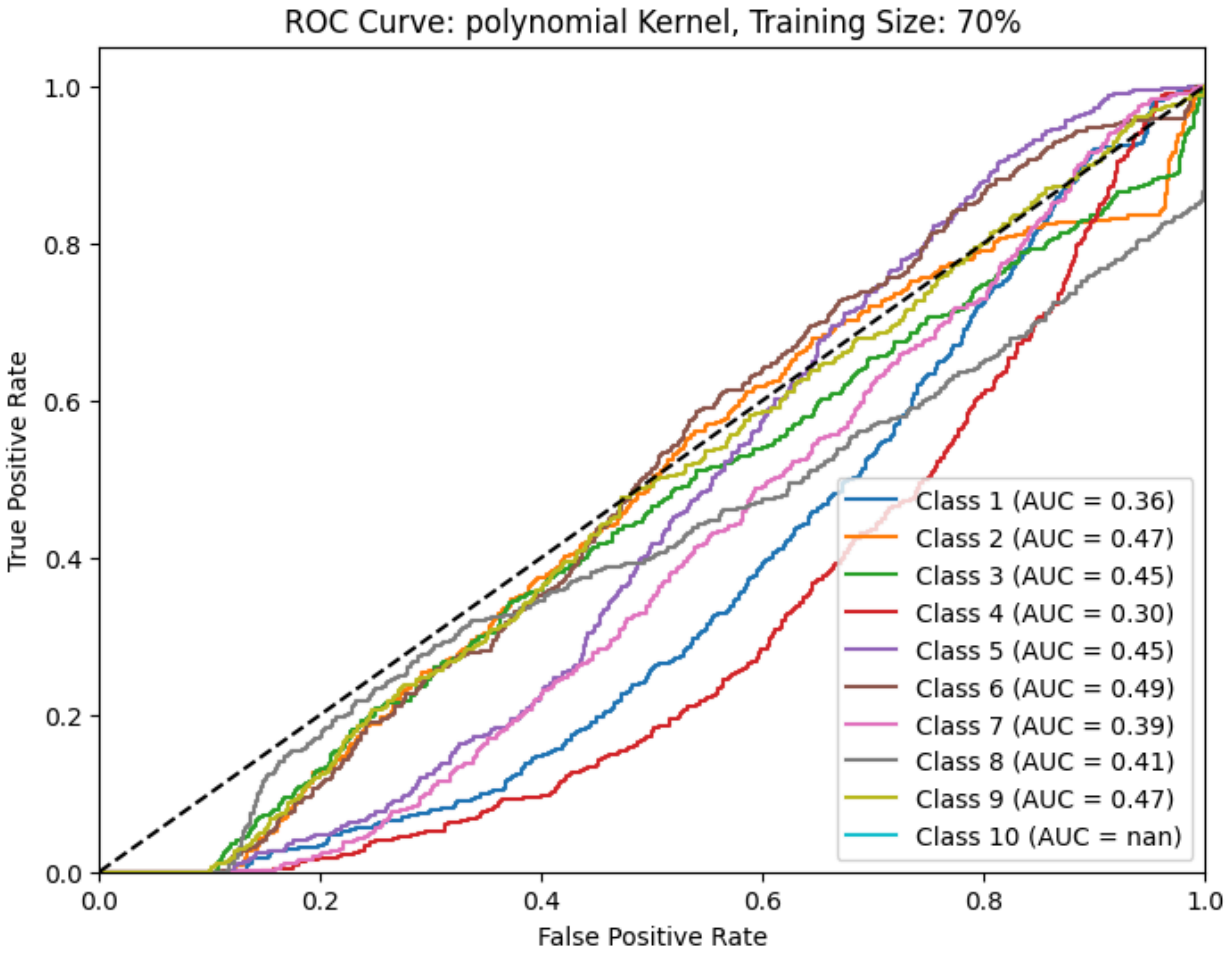

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



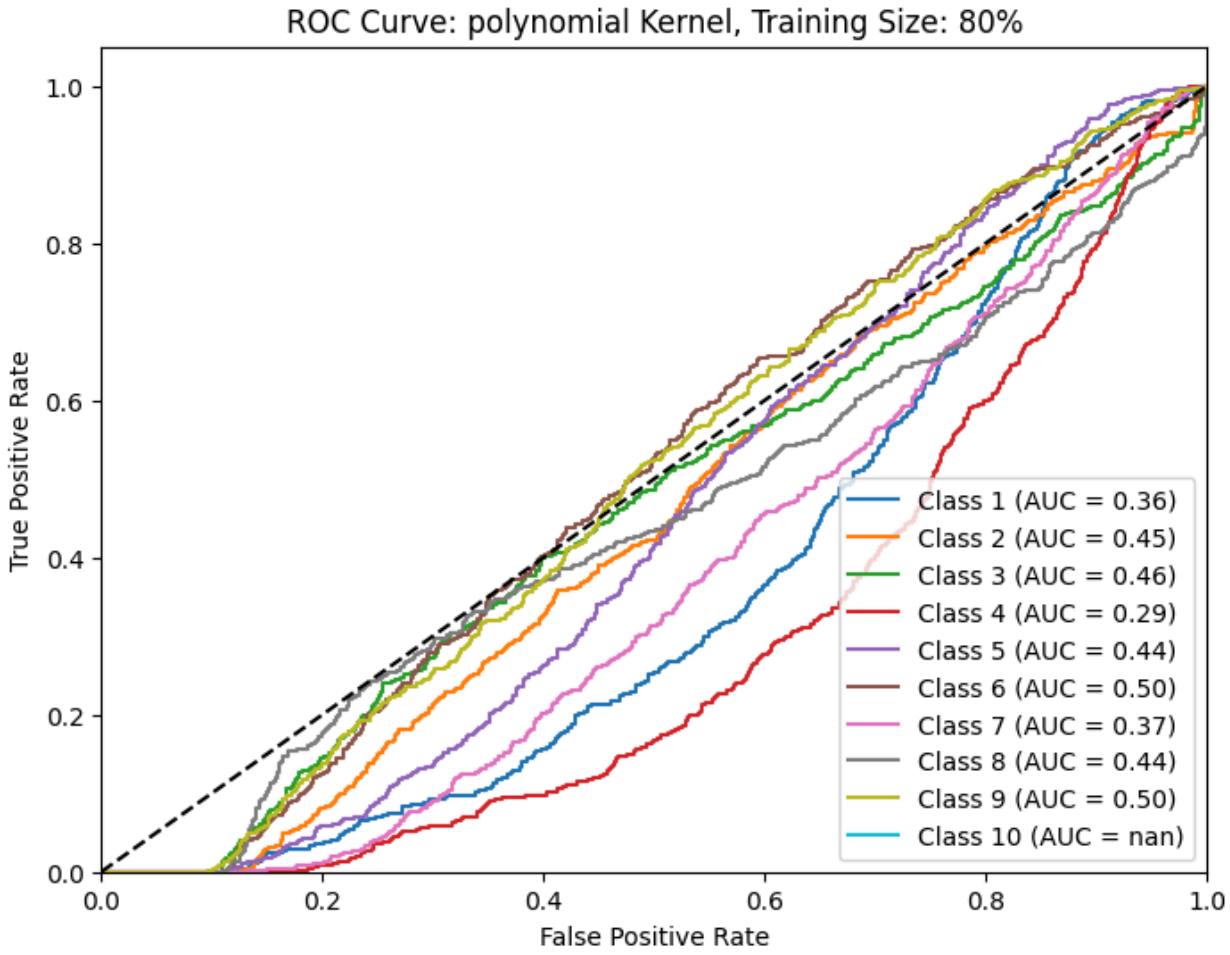

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



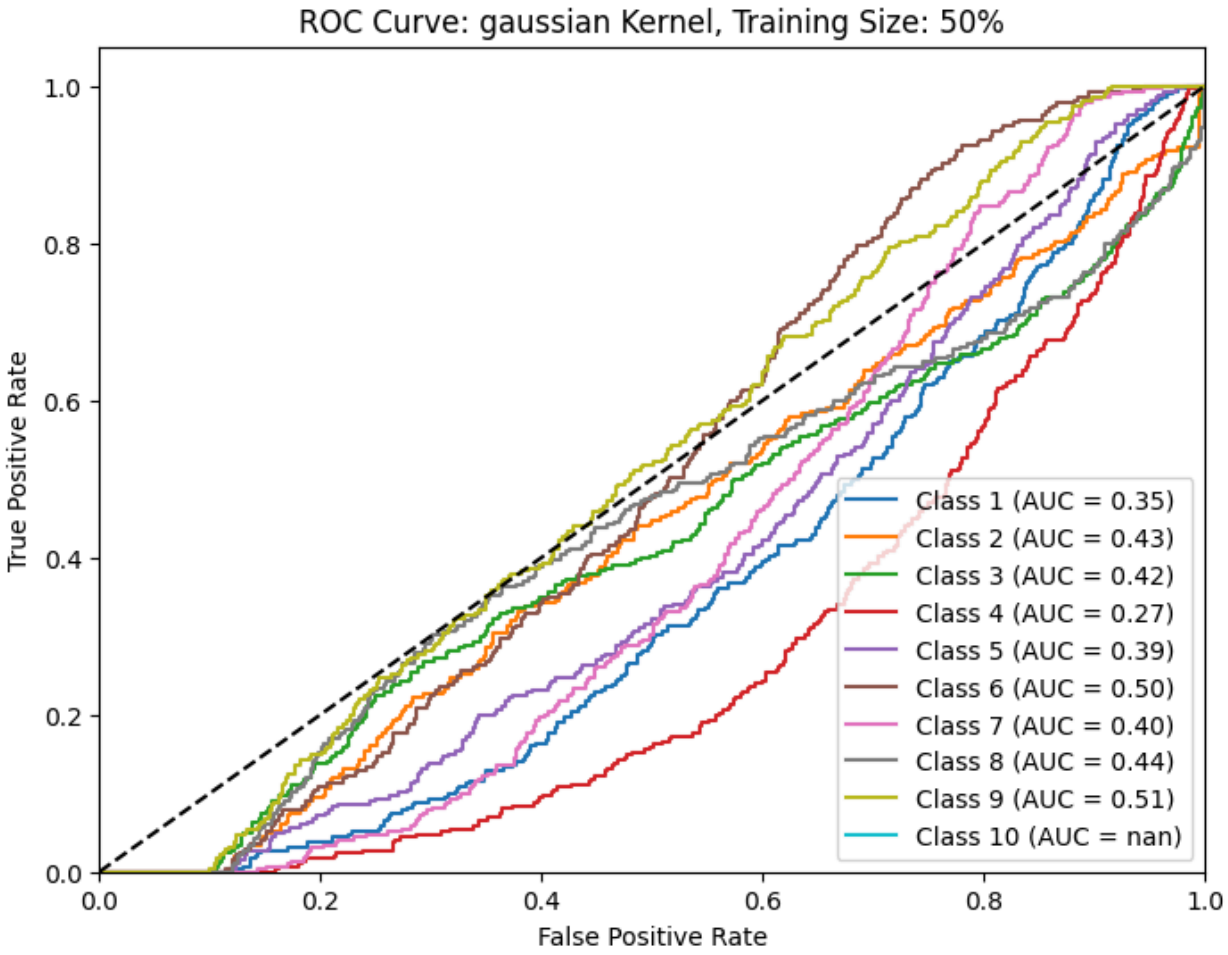

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



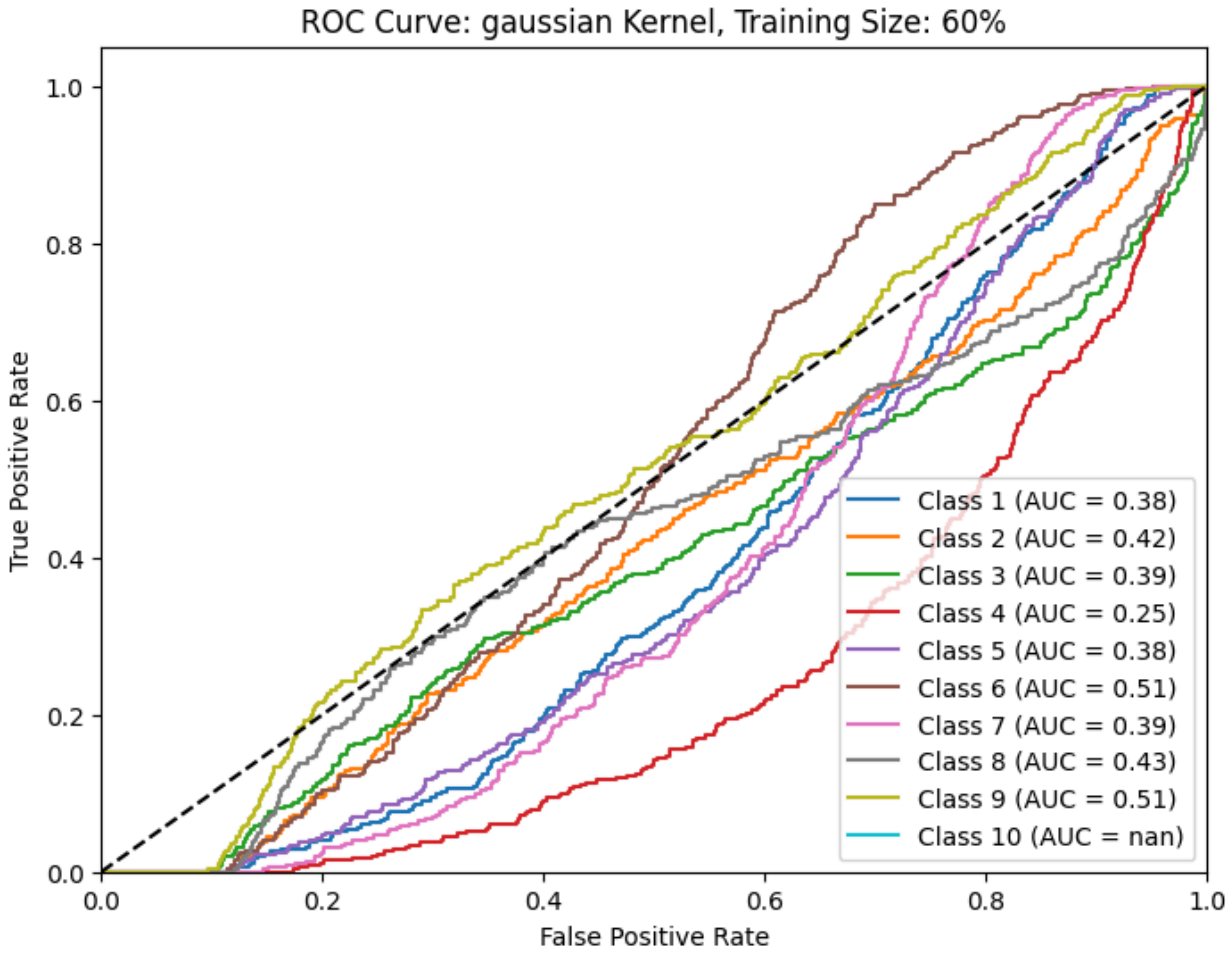

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



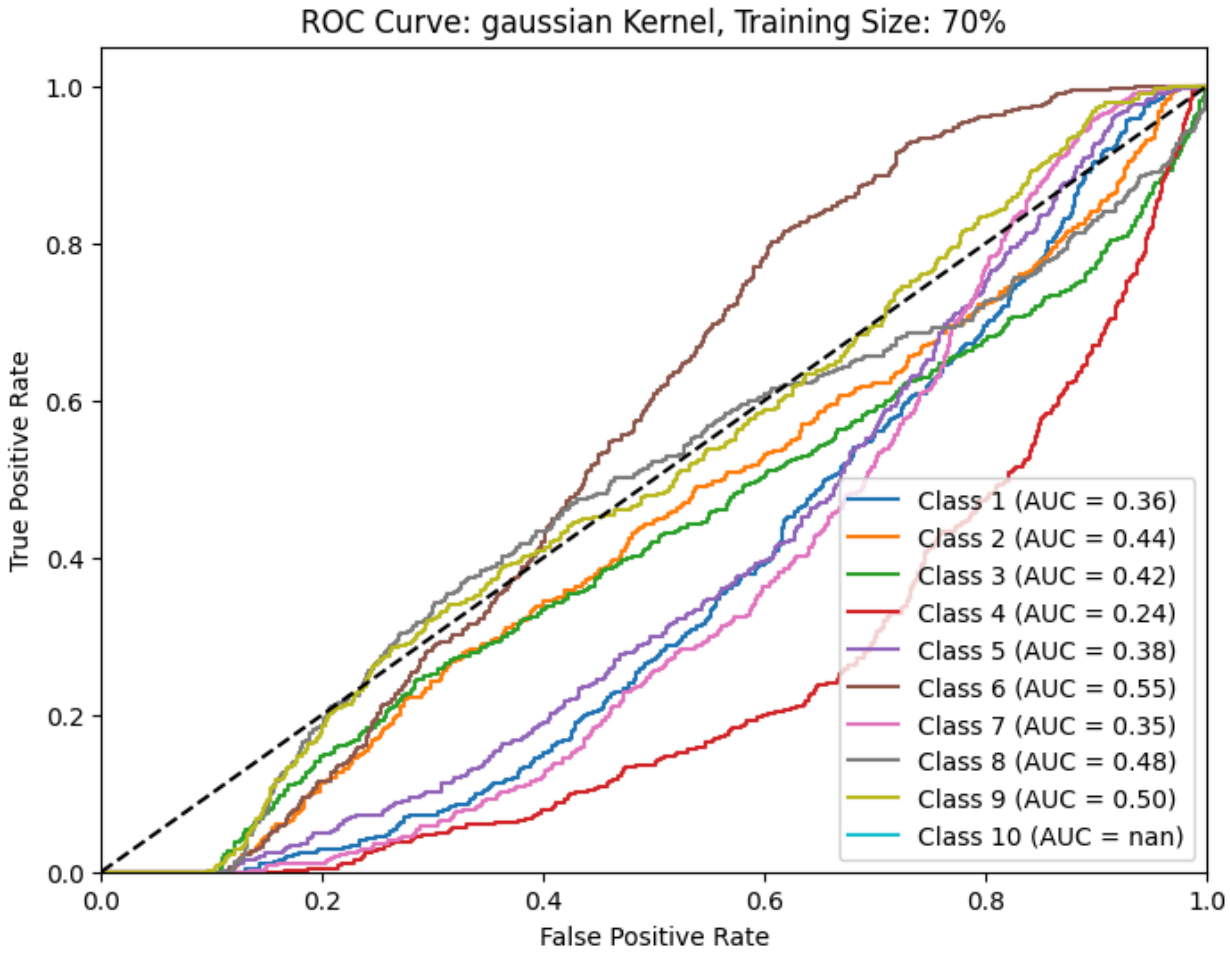

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



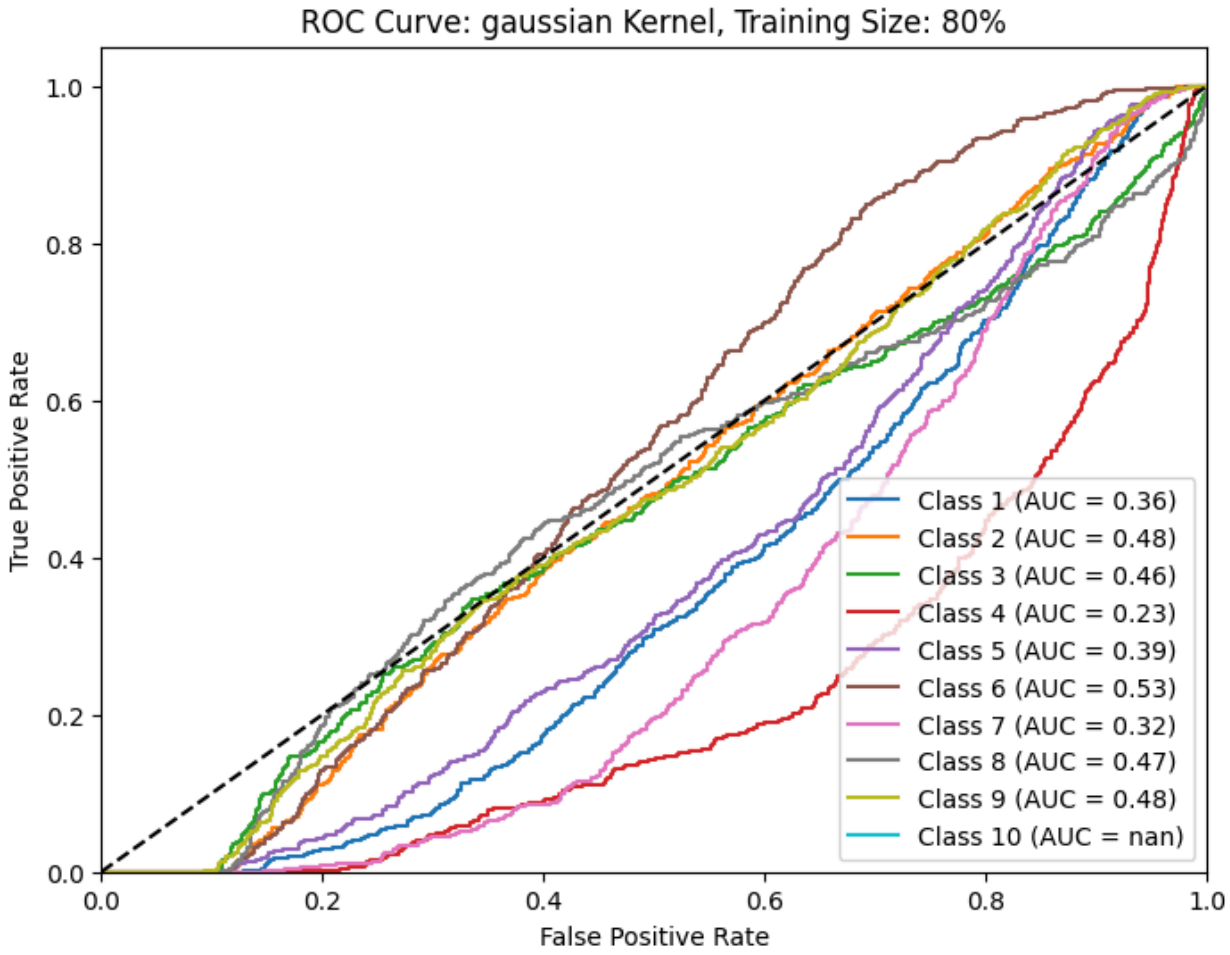

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



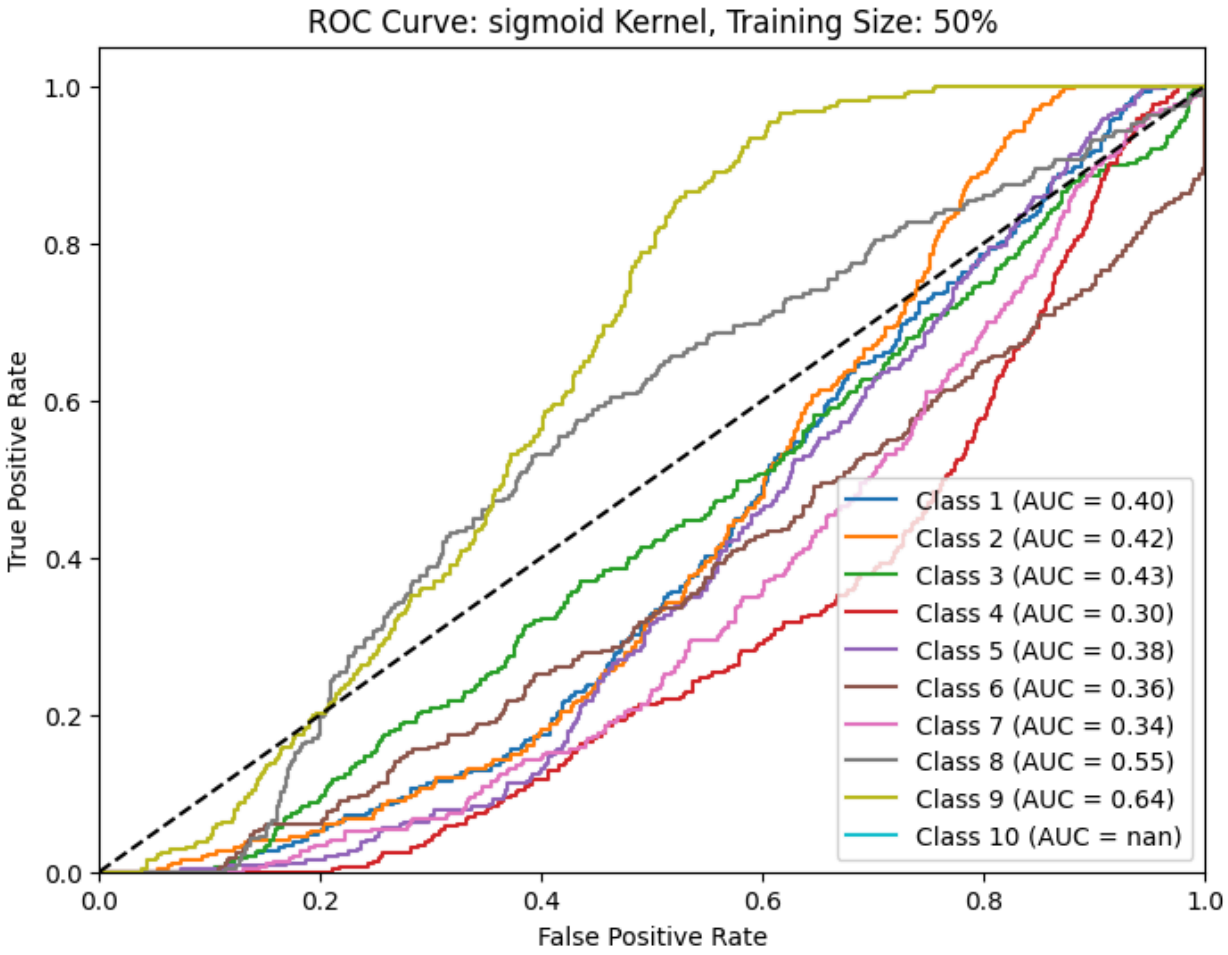

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



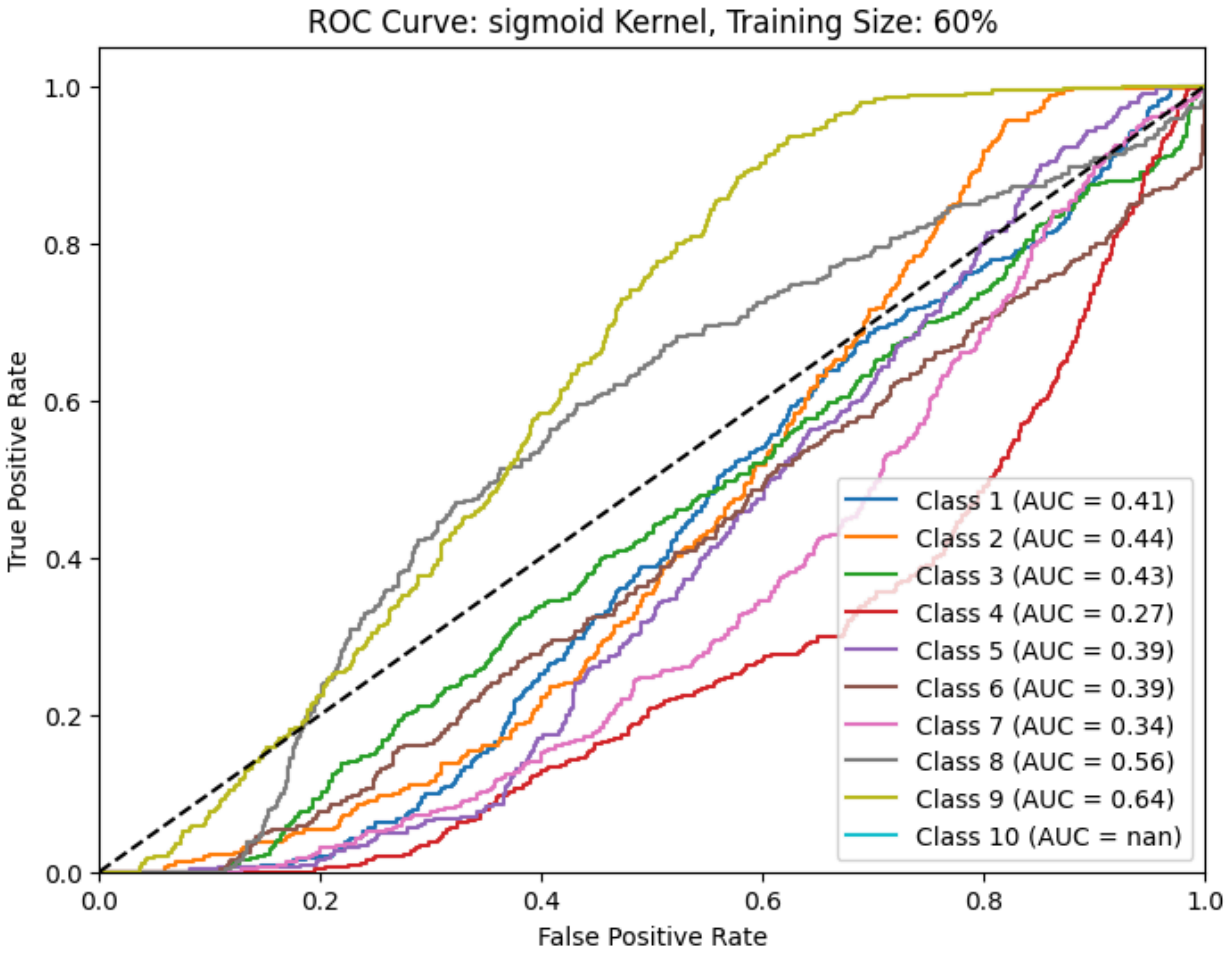

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



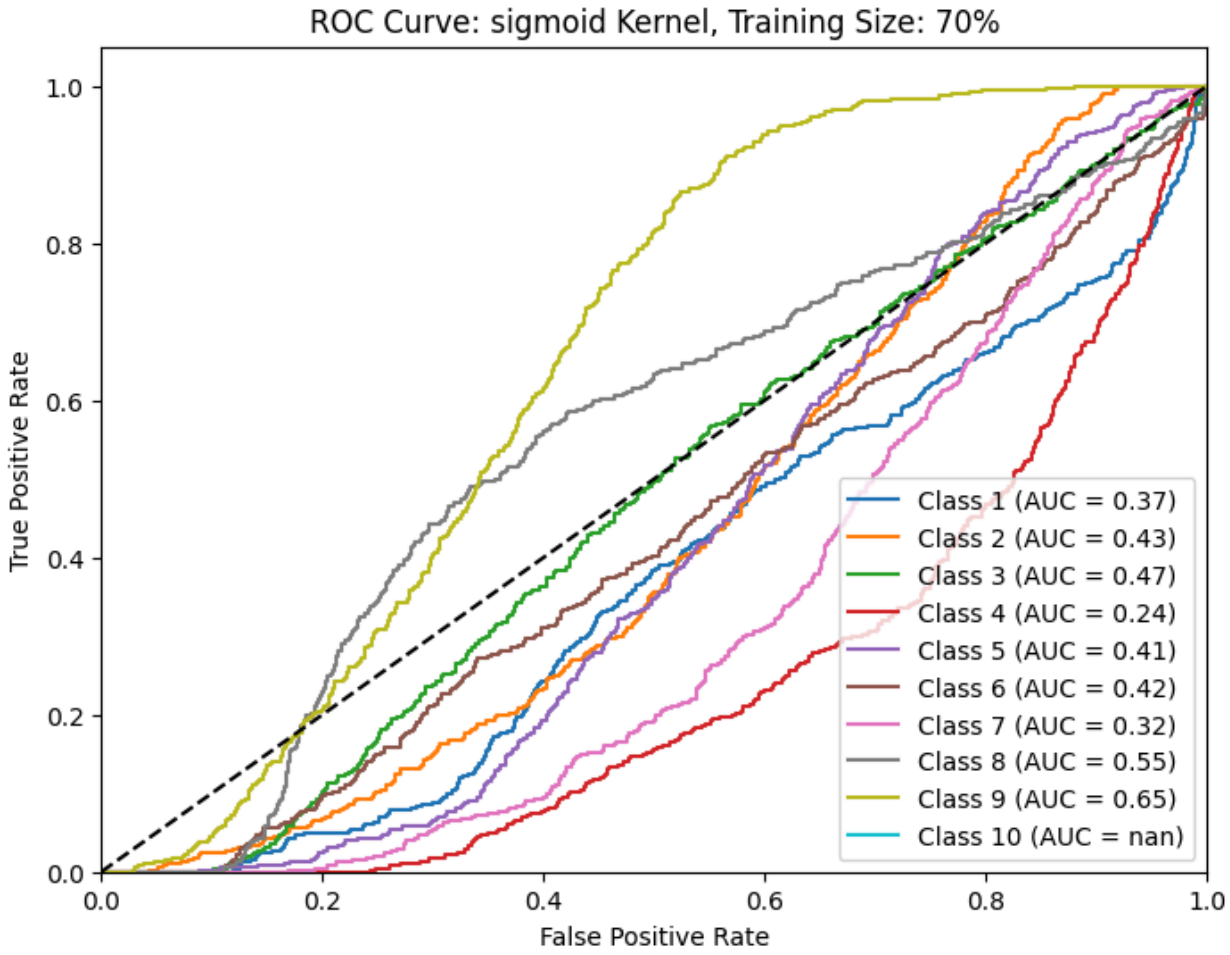

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



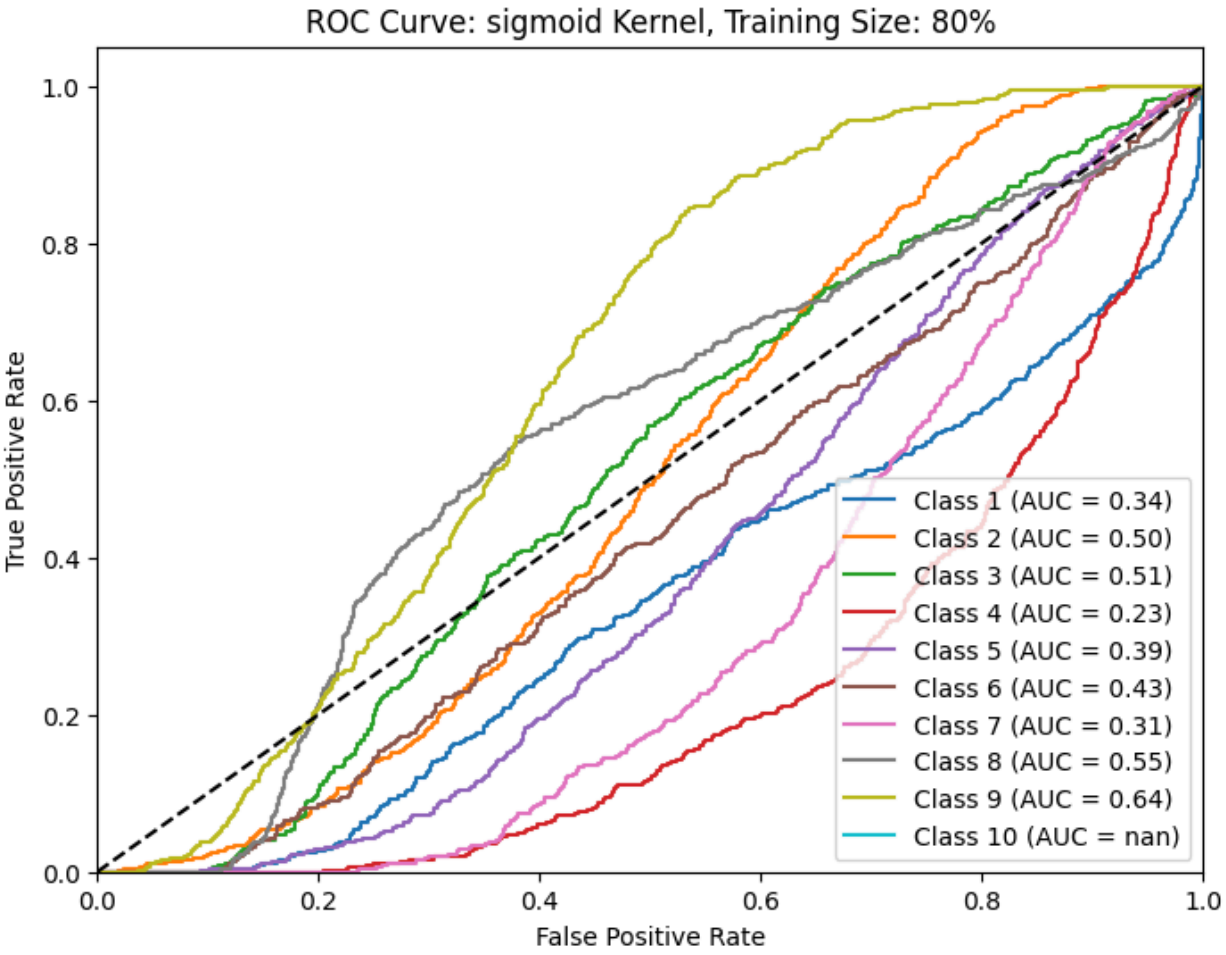

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```




```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: 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: Undef
inedMetricWarning: 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 [ ]:

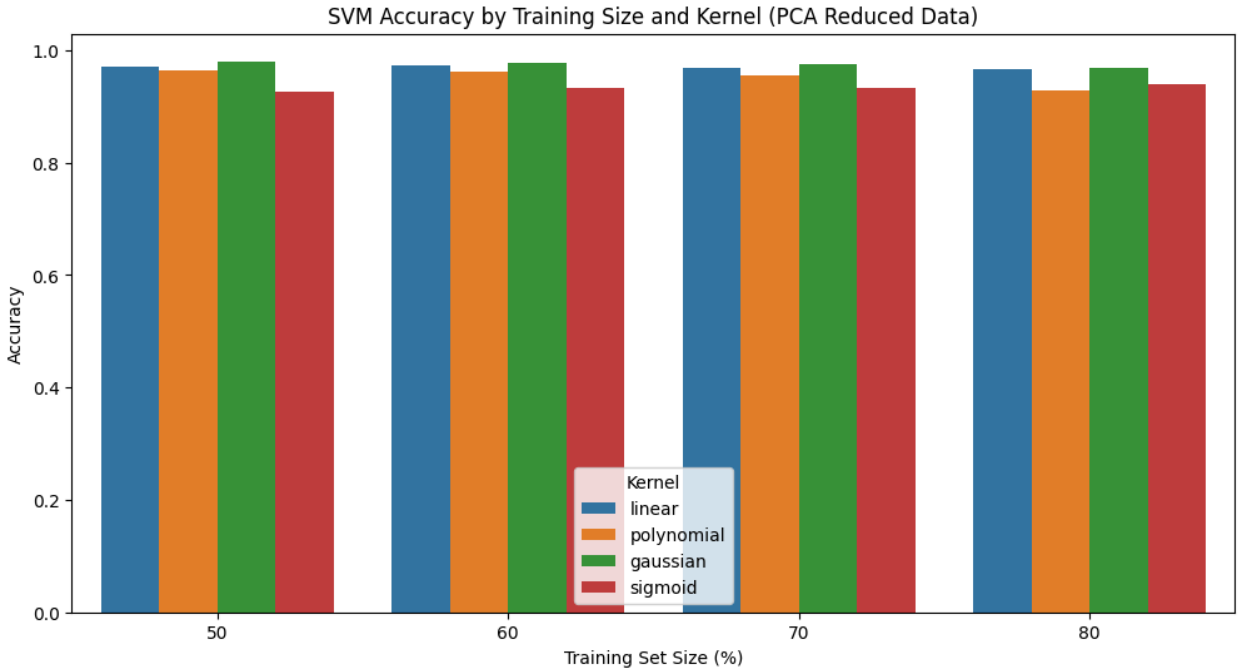
```
results_pca = []
confusion_matrices_pca = []

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 proba
   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', zer
   recall_pca = recall_score(y_test, y_pred_pca, average='weighted') # Calcula
   f1_pca = f1_score(y_test, y_pred_pca, average='weighted') # Calculate weigh


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

   results_pca.append({
       "Training size":int (size*100),
```

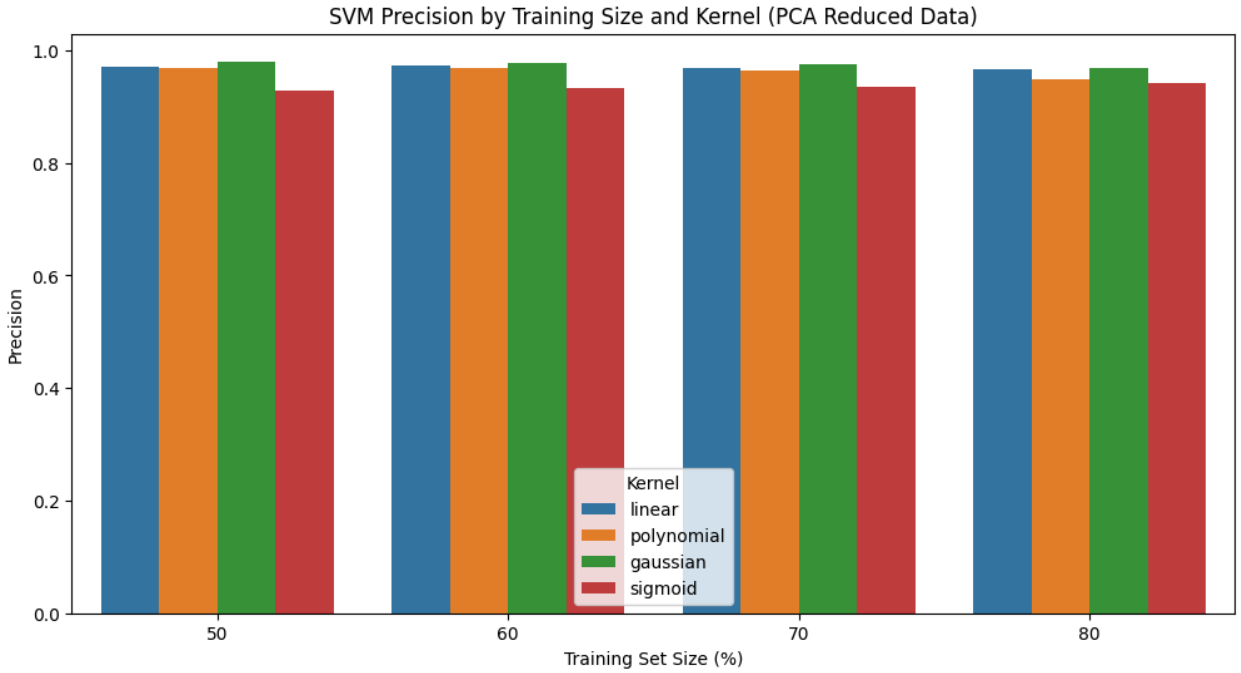
```
        "Accuracy":acc_pca,
        "Precision": precision_pca,
        "Recall": recall_pca,
        "F1-score": f1_pca,
        "Kernel":kernel_name
    })
    confusion_matrices_pca.append({ # Store confusion matrix with metadata
        "Training size":int (size*100),
        "Kernel":kernel_name,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

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

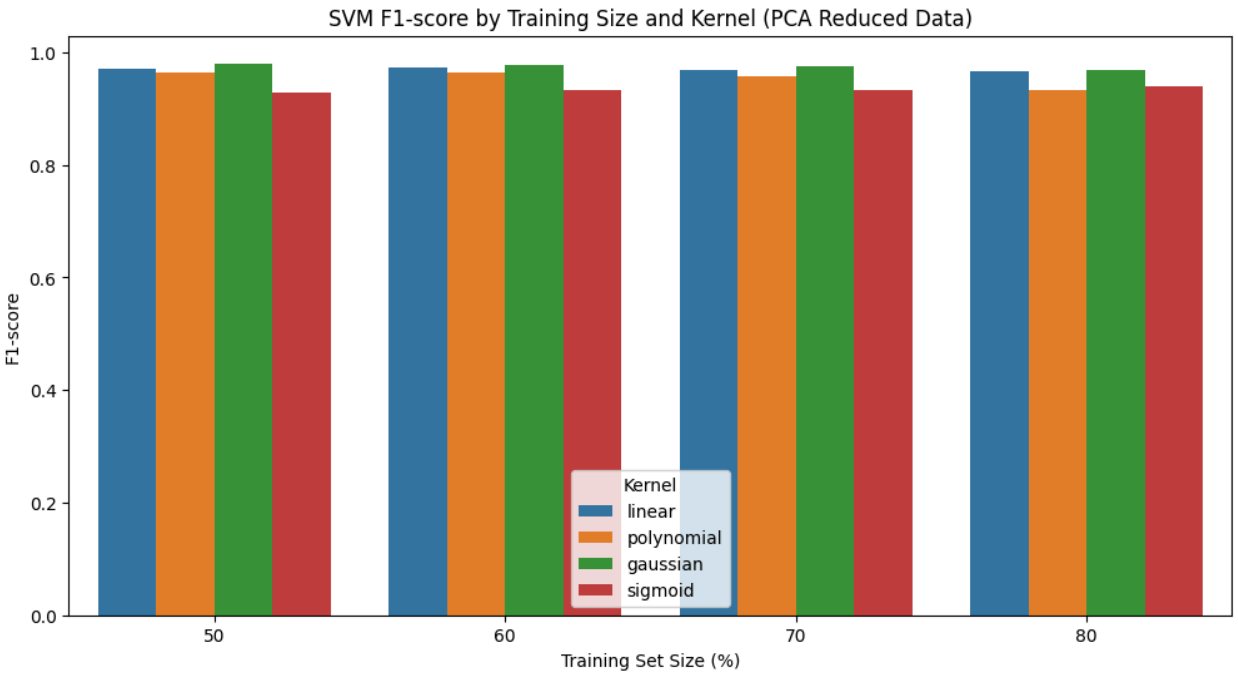
In [ ]:
```
df_pca = pd.DataFrame(results_pca)
display(df_pca) # Display the DataFrame as a table

# 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
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)') # Ad
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 t
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
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	0.970819	0.970937	0.970819	0.970850	linear
1	60	0.972716	0.972695	0.972716	0.972687	linear
2	70	0.968734	0.968882	0.968734	0.968751	linear
3	80	0.966192	0.966314	0.966192	0.966197	linear
4	50	0.964413	0.968400	0.964413	0.965236	polynomial
5	60	0.962930	0.967787	0.962930	0.963943	polynomial
6	70	0.956279	0.963473	0.956279	0.957801	polynomial
7	80	0.927491	0.948847	0.927491	0.932636	polynomial
8	50	0.980427	0.980517	0.980427	0.980437	gaussian
9	60	0.977165	0.977299	0.977165	0.977172	gaussian
10	70	0.975089	0.975237	0.975089	0.975076	gaussian
11	80	0.968416	0.968740	0.968416	0.968425	gaussian
12	50	0.927402	0.927971	0.927402	0.927522	sigmoid
13	60	0.933571	0.933986	0.933571	0.933645	sigmoid
14	70	0.933910	0.934523	0.933910	0.934033	sigmoid
15	80	0.940169	0.940999	0.940169	0.940326	sigmoid



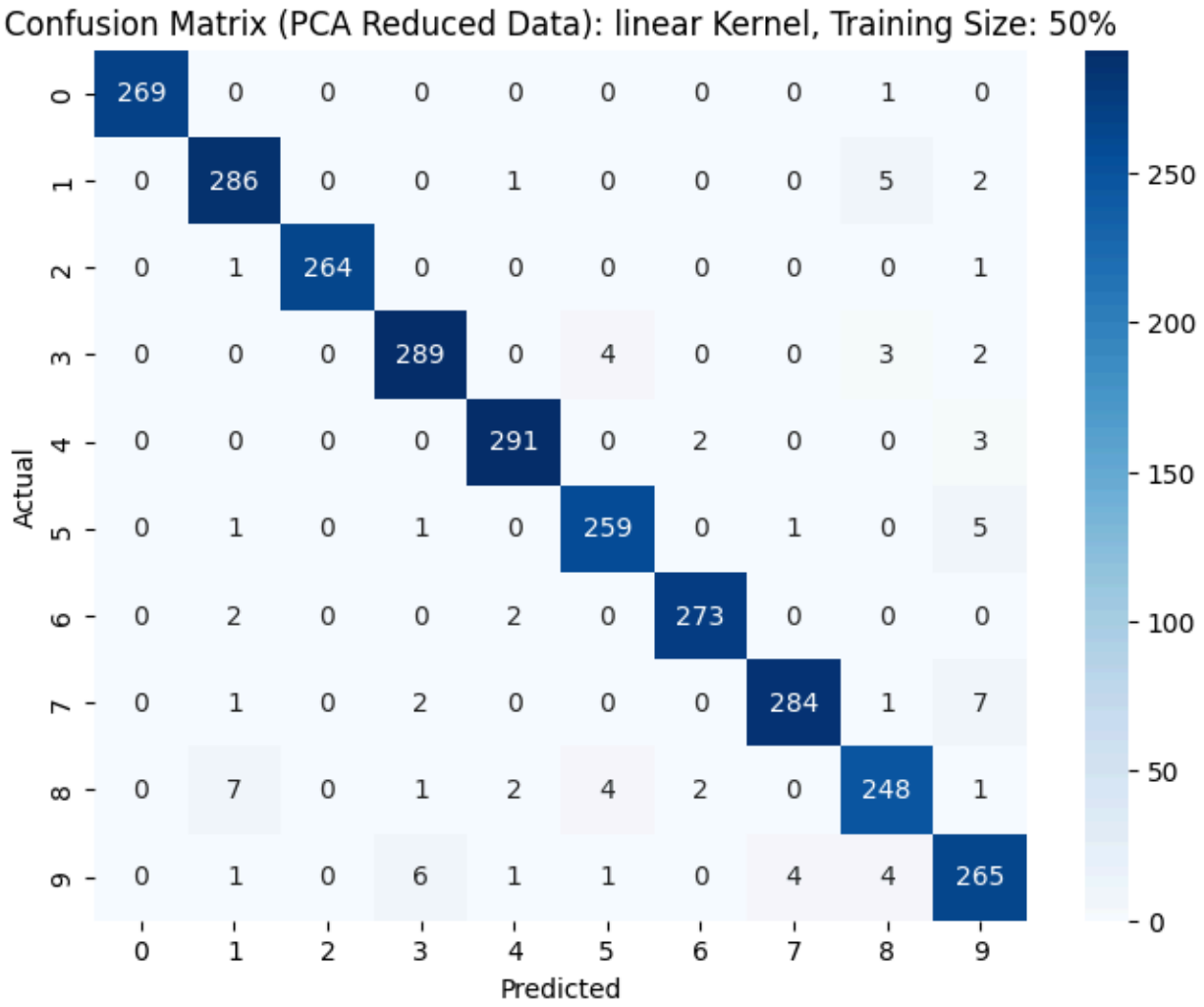

SVM Precision by Training Size and Kernel (PCA Reduced Data)
Precision
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid
0.0
0.2
0.4
0.6
0.8
1.0
50
60
70
80

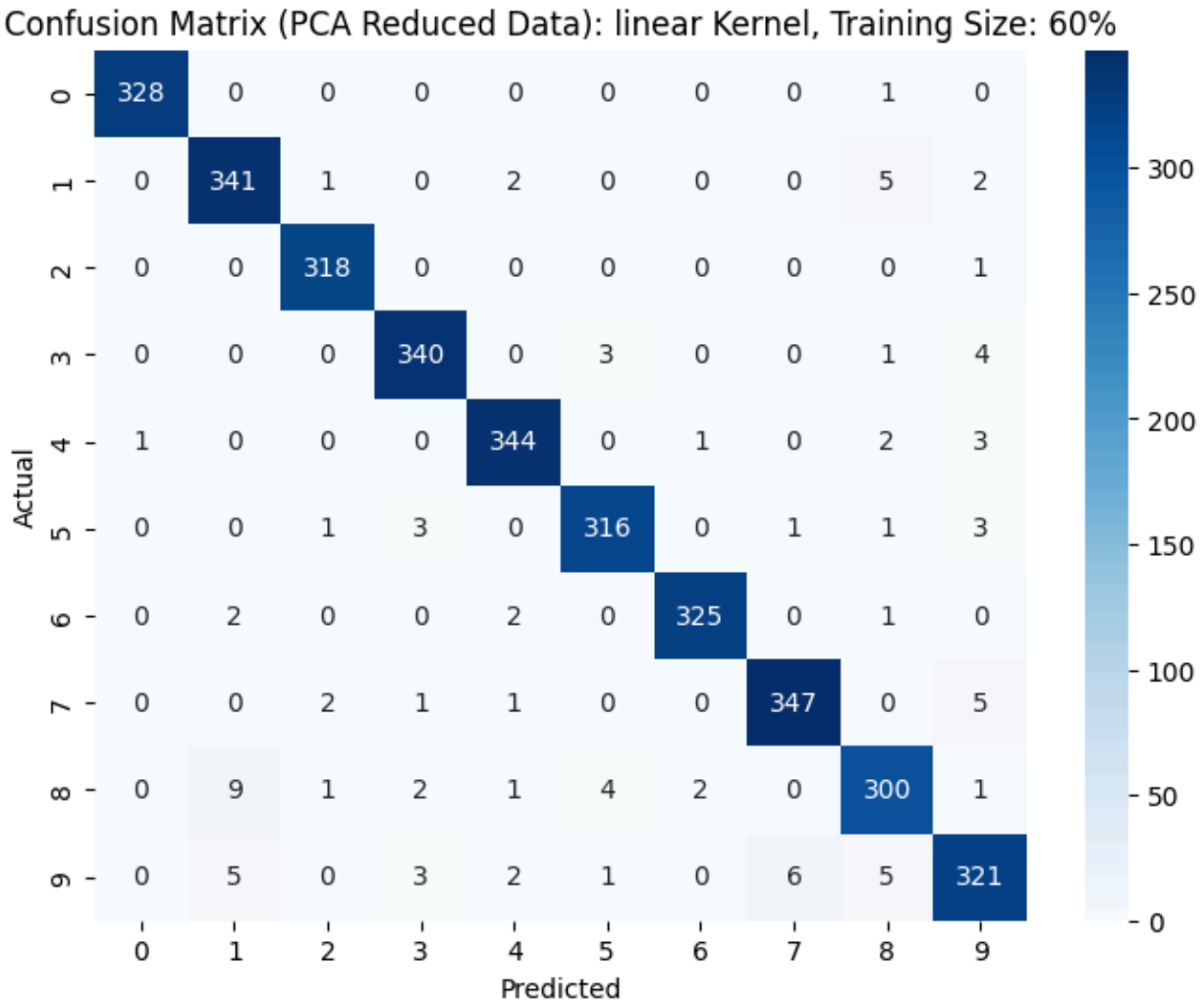

SVM Recall by Training Size and Kernel (PCA Reduced Data)
Recall
Training Set Size (%)
Kernel
linear
polynomial
gaussian
sigmoid
0.0
0.2
0.4
0.6
0.8
1.0
50
60
70
80

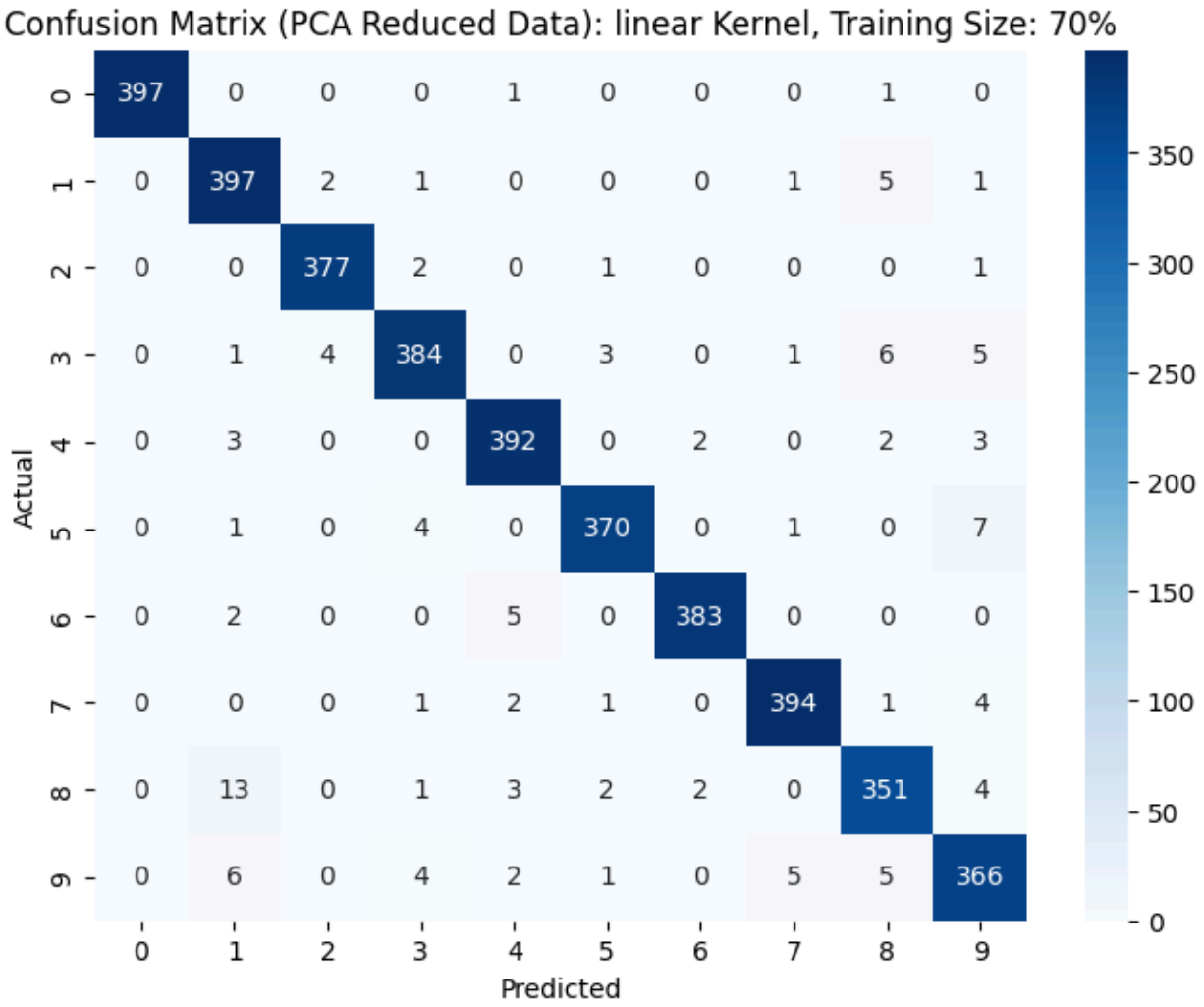


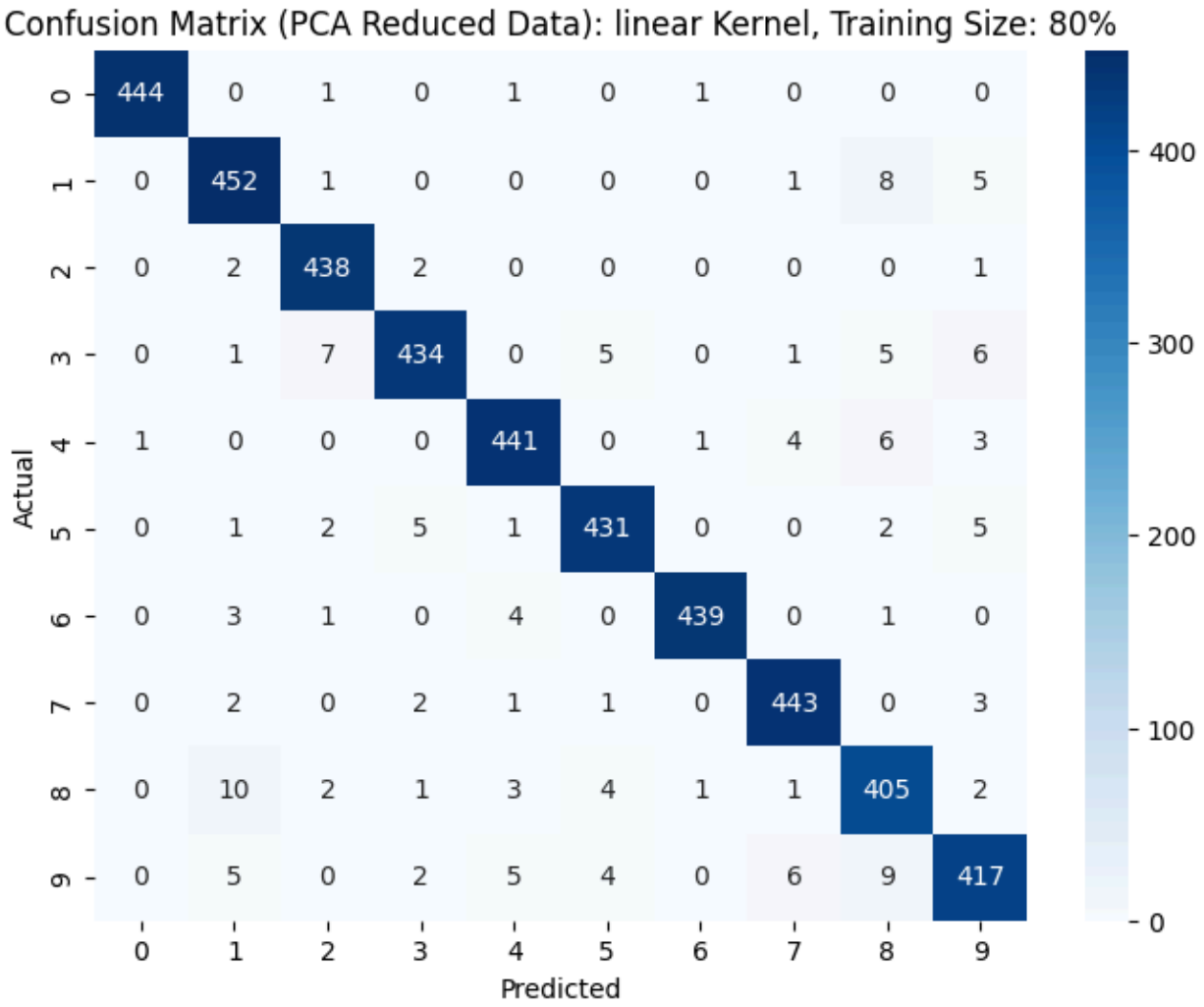
Show Confusion Matrices for PCA-Reduced Data

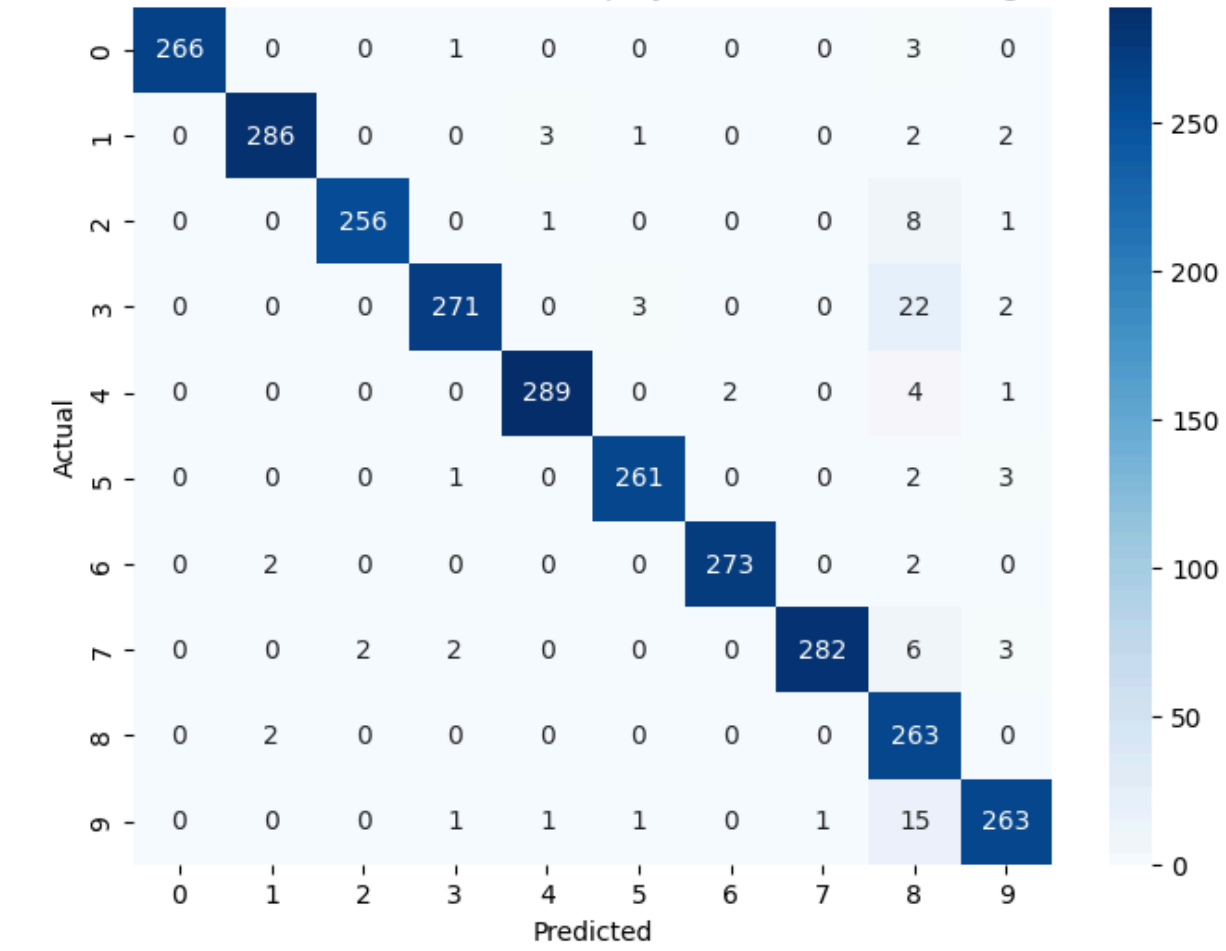
In [ ]:
```
for cm_data in confusion_matrices_pca:
    cm = cm_data["Confusion Matrix"]
    kernel_name = cm_data["Kernel"]
    training_size = cm_data["Training size"]

    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()
```

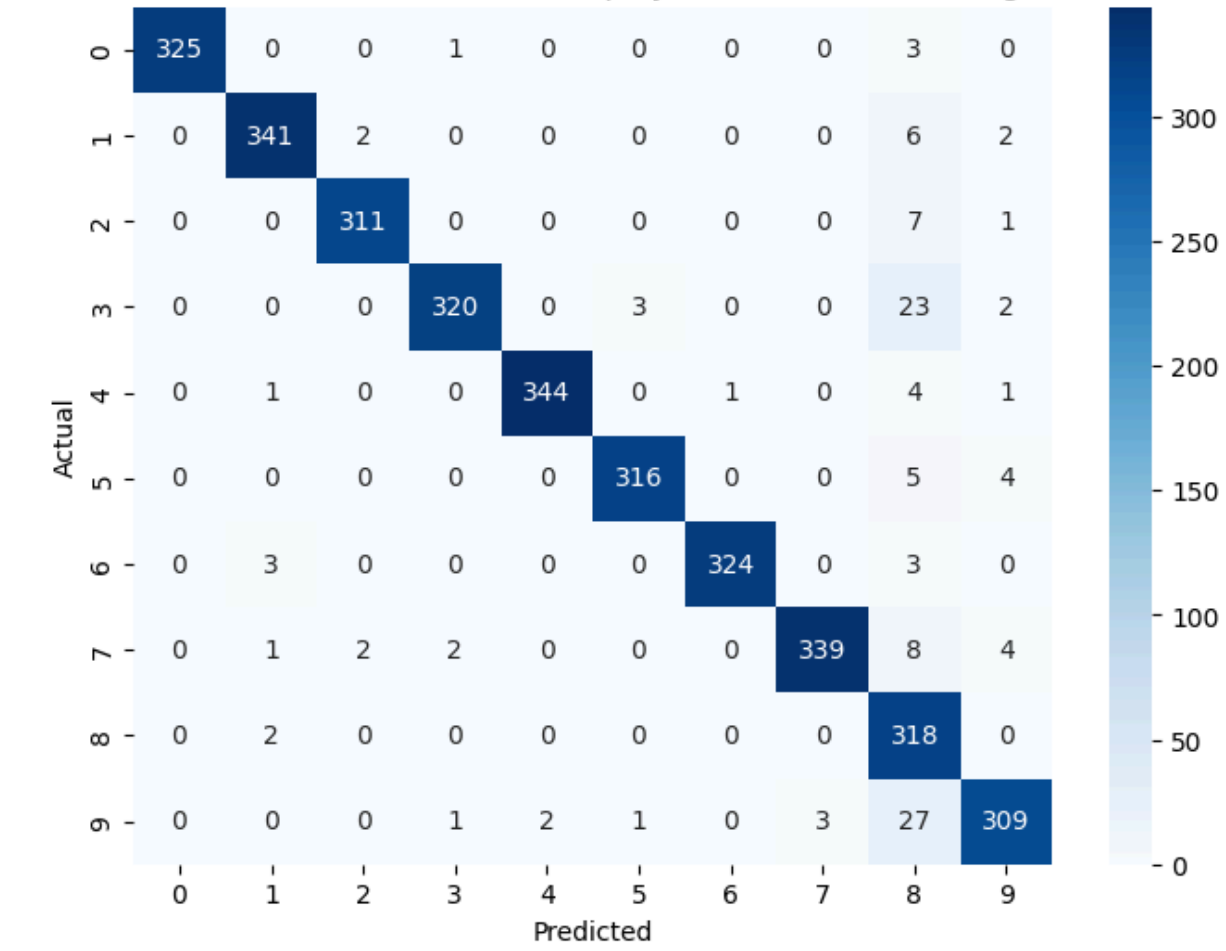

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
269 0 0 0 0 0 0 0 1 0
0 286 0 0 1 0 0 0 5 2
0 1 264 0 0 0 0 0 0 1
0 0 0 289 0 4 0 0 3 2
0 0 0 0 291 0 2 0 0 3
0 1 0 1 0 259 0 1 0 5
0 2 0 0 2 0 273 0 0 0
0 1 0 2 0 0 0 284 1 7
0 7 0 1 2 4 2 0 248 1
0 1 0 6 1 1 0 4 4 265
250
200
150
100
50
0

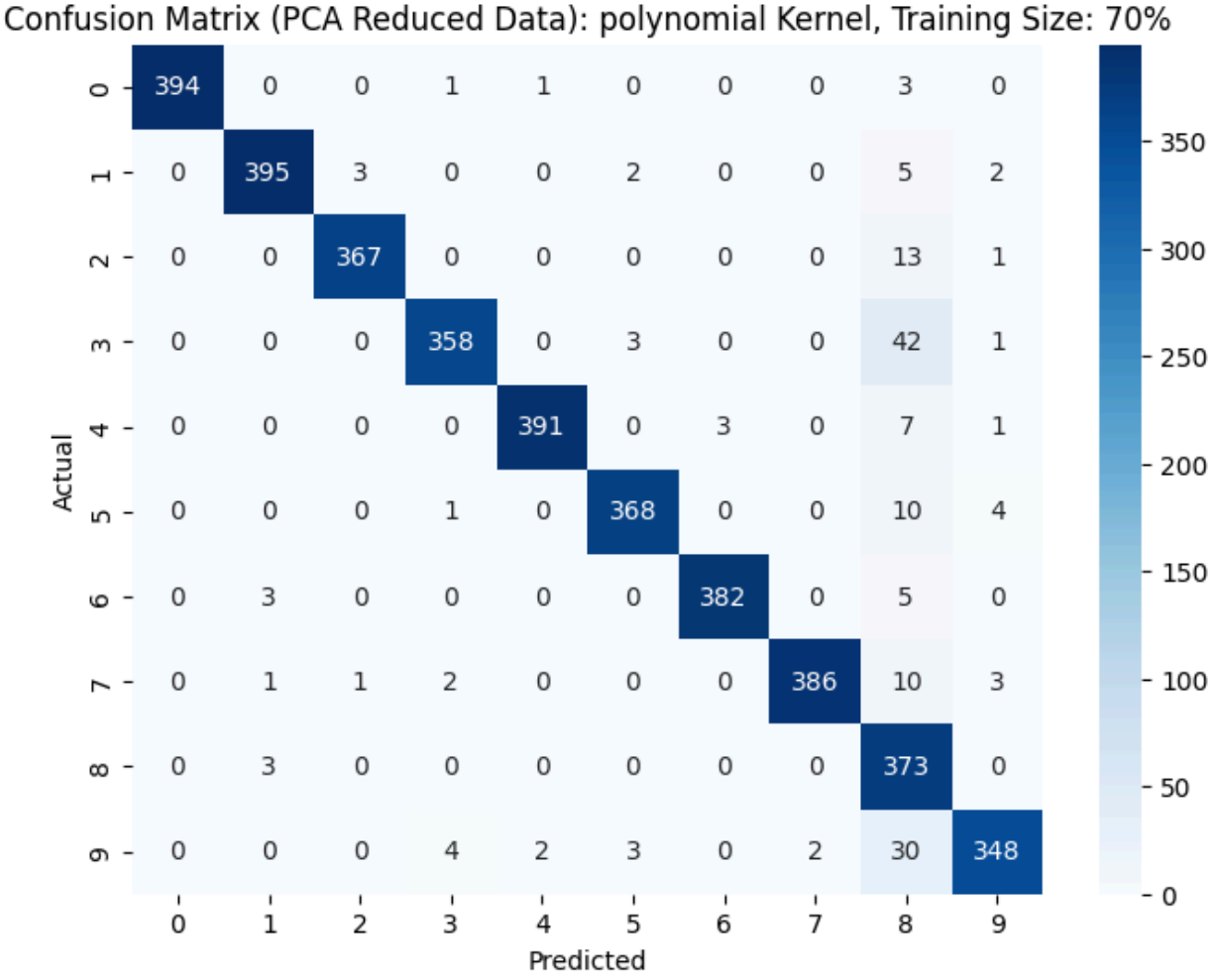

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 60%
Actual
Predicted
0 328 0 0 0 0 0 0 0 1 0
1 0 341 1 0 2 0 0 0 5 2
2 0 0 318 0 0 0 0 0 0 1
3 0 0 0 340 0 3 0 0 1 4
4 1 0 0 0 344 0 1 0 2 3
5 0 0 1 3 0 316 0 1 1 3
6 0 2 0 0 2 0 325 0 1 0
7 0 0 2 1 1 0 0 347 0 5
8 0 9 1 2 1 4 2 0 300 1
9 0 5 0 3 2 1 0 6 5 321
0 1 2 3 4 5 6 7 8 9
300
250
200
150
100
50
0

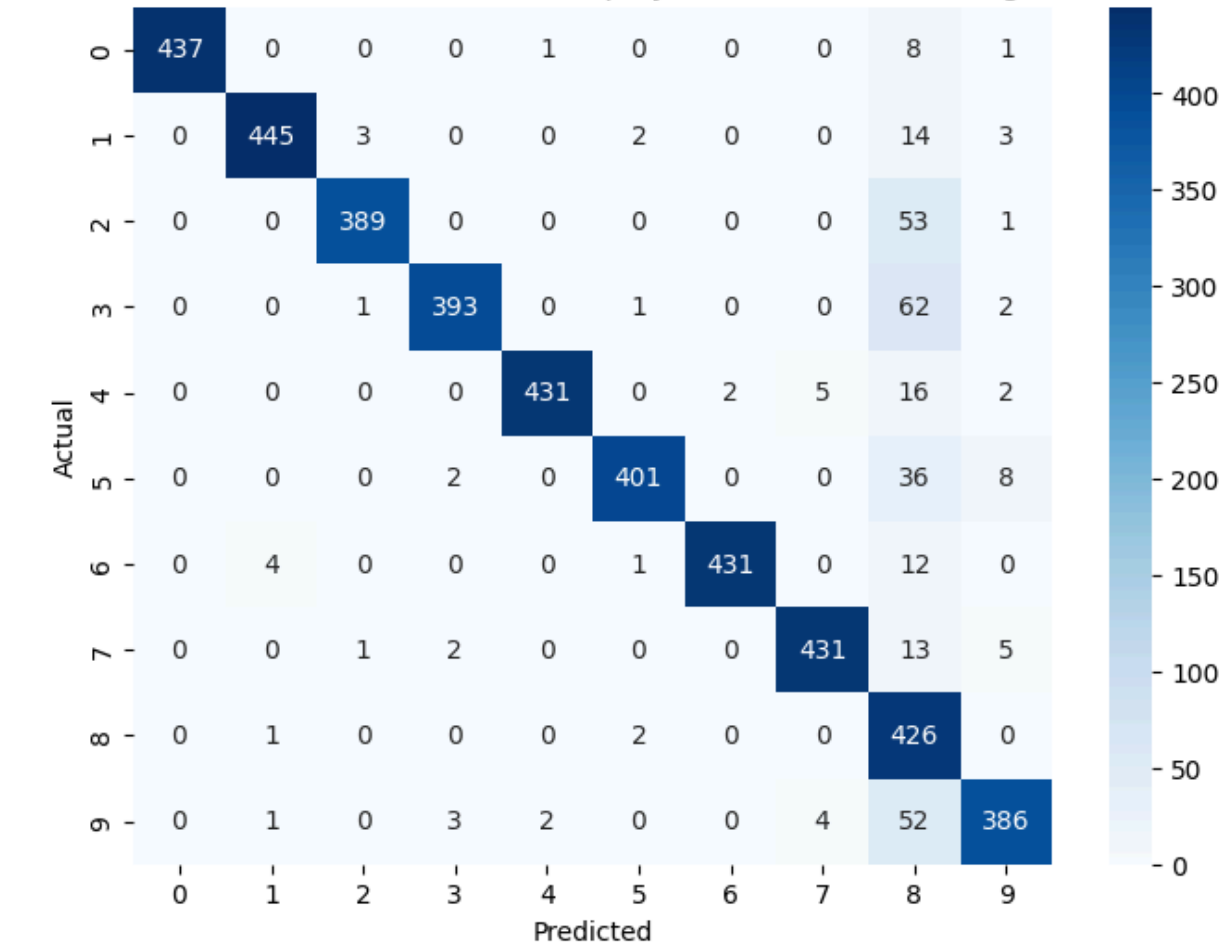

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 70%
Actual
Predicted
0 397 0 0 0 1 0 0 0 1 0
1 0 397 2 1 0 0 0 1 5 1
2 0 0 377 2 0 1 0 0 0 1
3 0 1 4 384 0 3 0 1 6 5
4 0 3 0 0 392 0 2 0 2 3
5 0 1 0 4 0 370 0 1 0 7
6 0 2 0 0 5 0 383 0 0 0
7 0 0 0 1 2 1 0 394 1 4
8 0 13 0 1 3 2 2 0 351 4
9 0 6 0 4 2 1 0 5 5 366
0 1 2 3 4 5 6 7 8 9
350 300 250 200 150 100 50 0

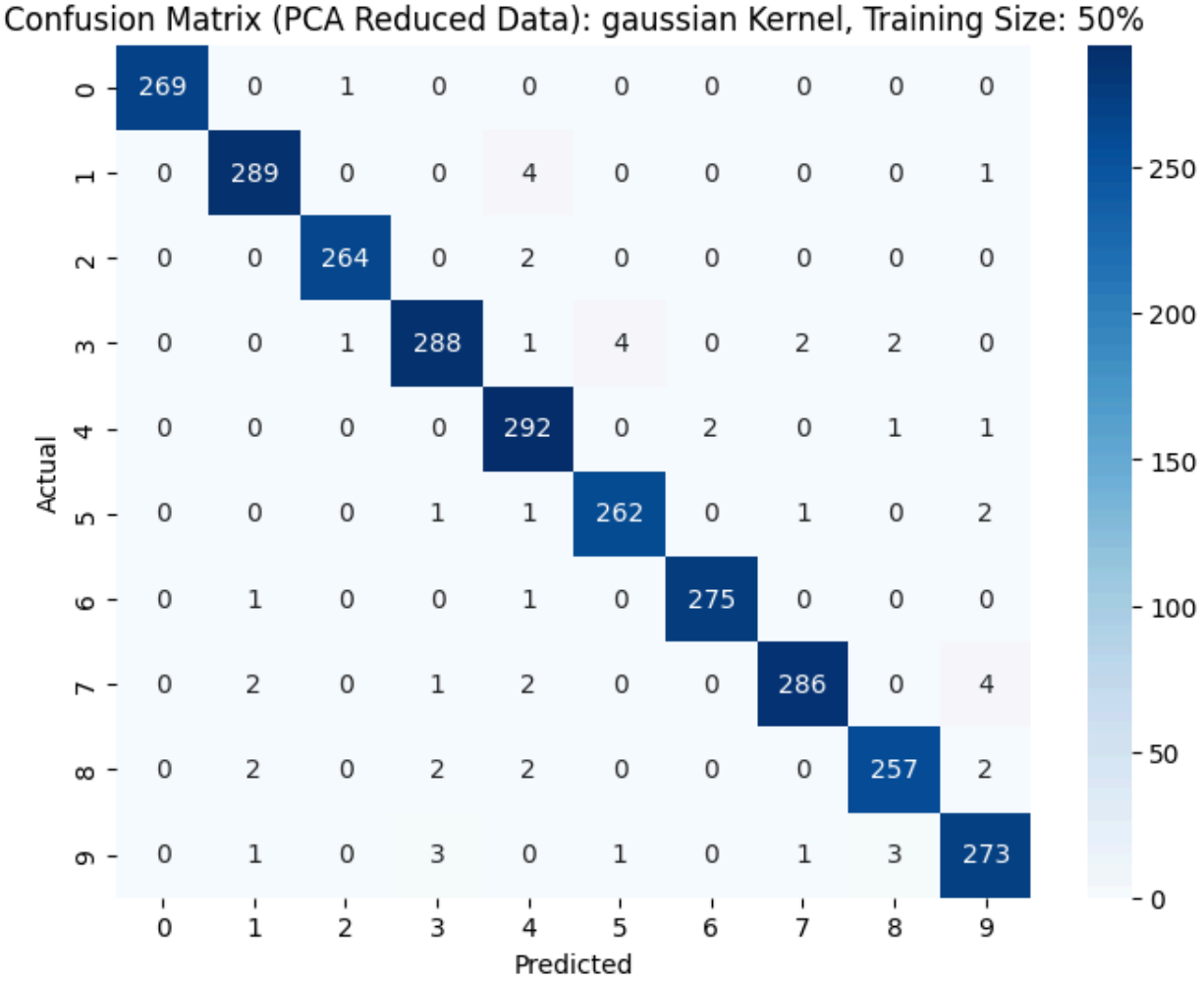

Confusion Matrix (PCA Reduced Data): linear Kernel, Training Size: 80%
Actual
Predicted
0 444 0 1 0 1 0 1 0 0 0
1 0 452 1 0 0 0 0 1 8 5
2 0 2 438 2 0 0 0 0 0 1
3 0 1 7 434 0 5 0 1 5 6
4 1 0 0 0 441 0 1 4 6 3
5 0 1 2 5 1 431 0 0 2 5
6 0 3 1 0 4 0 439 0 1 0
7 0 2 0 2 1 1 0 443 0 3
8 0 10 2 1 3 4 1 1 405 2
9 0 5 0 2 5 4 0 6 9 417
0 1 2 3 4 5 6 7 8 9
400
300
200
100
0

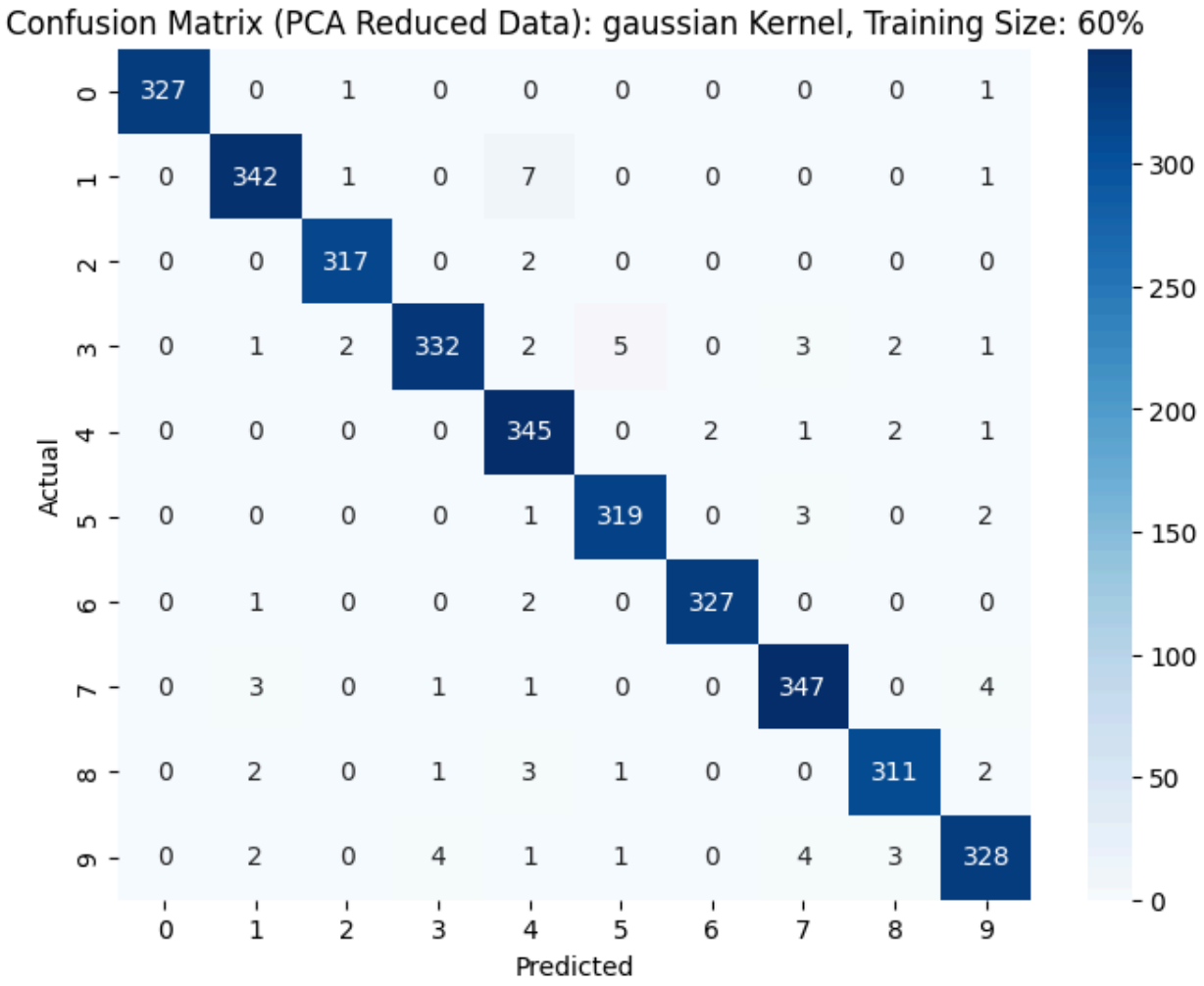

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
266 0 0 1 0 0 0 0 3 0
0 286 0 0 3 1 0 0 2 2
0 0 256 0 1 0 0 0 8 1
0 0 0 271 0 3 0 0 22 2
0 0 0 0 289 0 2 0 4 1
0 0 0 1 0 261 0 0 2 3
0 2 0 0 0 0 273 0 2 0
0 0 2 2 0 0 0 282 6 3
0 2 0 0 0 0 0 0 263 0
0 0 0 1 1 1 0 1 15 263
250
200
150
100
50
0

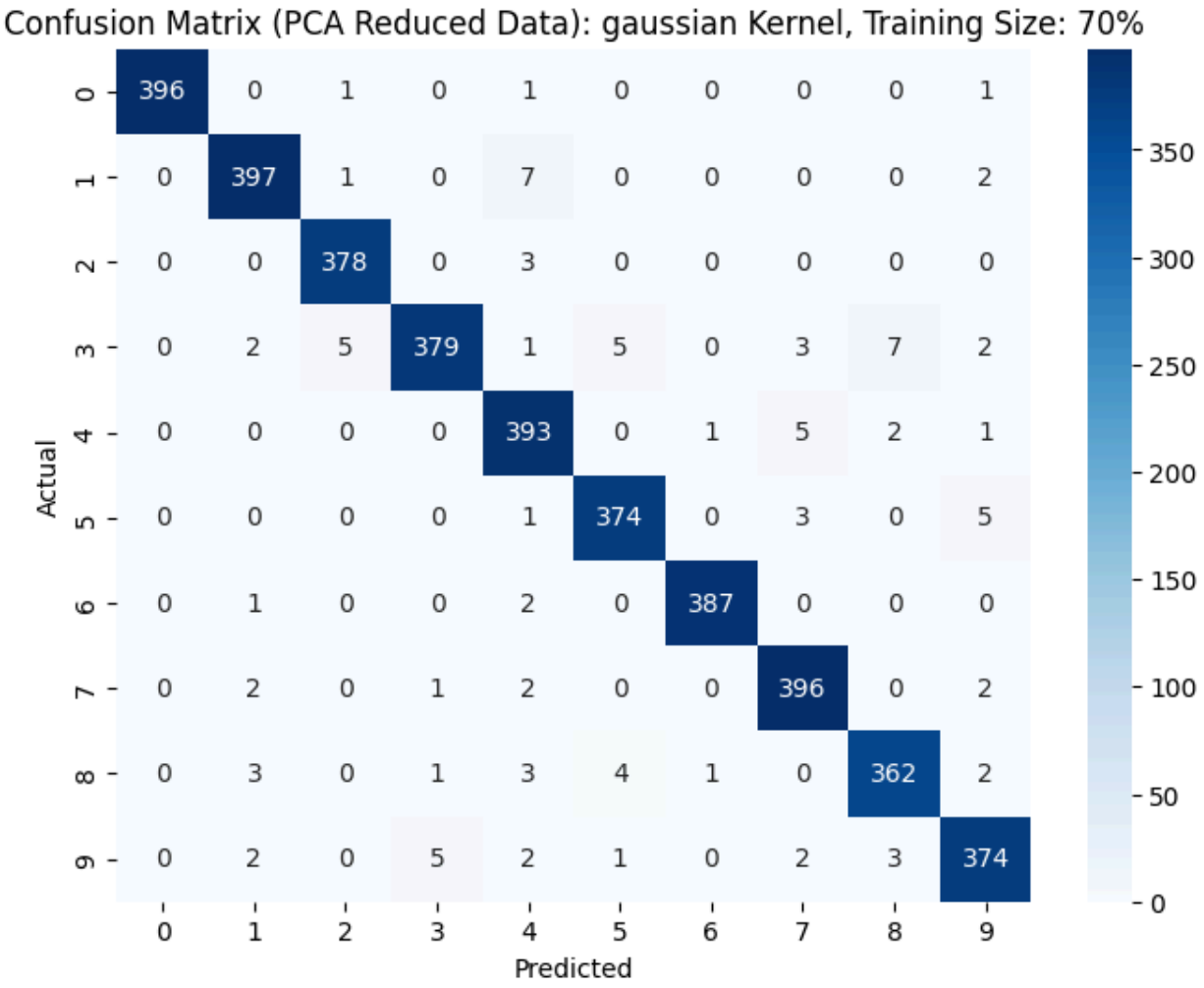

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 60%
Actual
Predicted
0 325 0 0 1 0 0 0 0 3 0
1 0 341 2 0 0 0 0 0 6 2
2 0 0 311 0 0 0 0 0 7 1
3 0 0 0 320 0 3 0 0 23 2
4 0 1 0 0 344 0 1 0 4 1
5 0 0 0 0 0 316 0 0 5 4
6 0 3 0 0 0 0 324 0 3 0
7 0 1 2 2 0 0 0 339 8 4
8 0 2 0 0 0 0 0 0 318 0
9 0 0 0 1 2 1 0 3 27 309
0 1 2 3 4 5 6 7 8 9
300
250
200
150
100
50
0

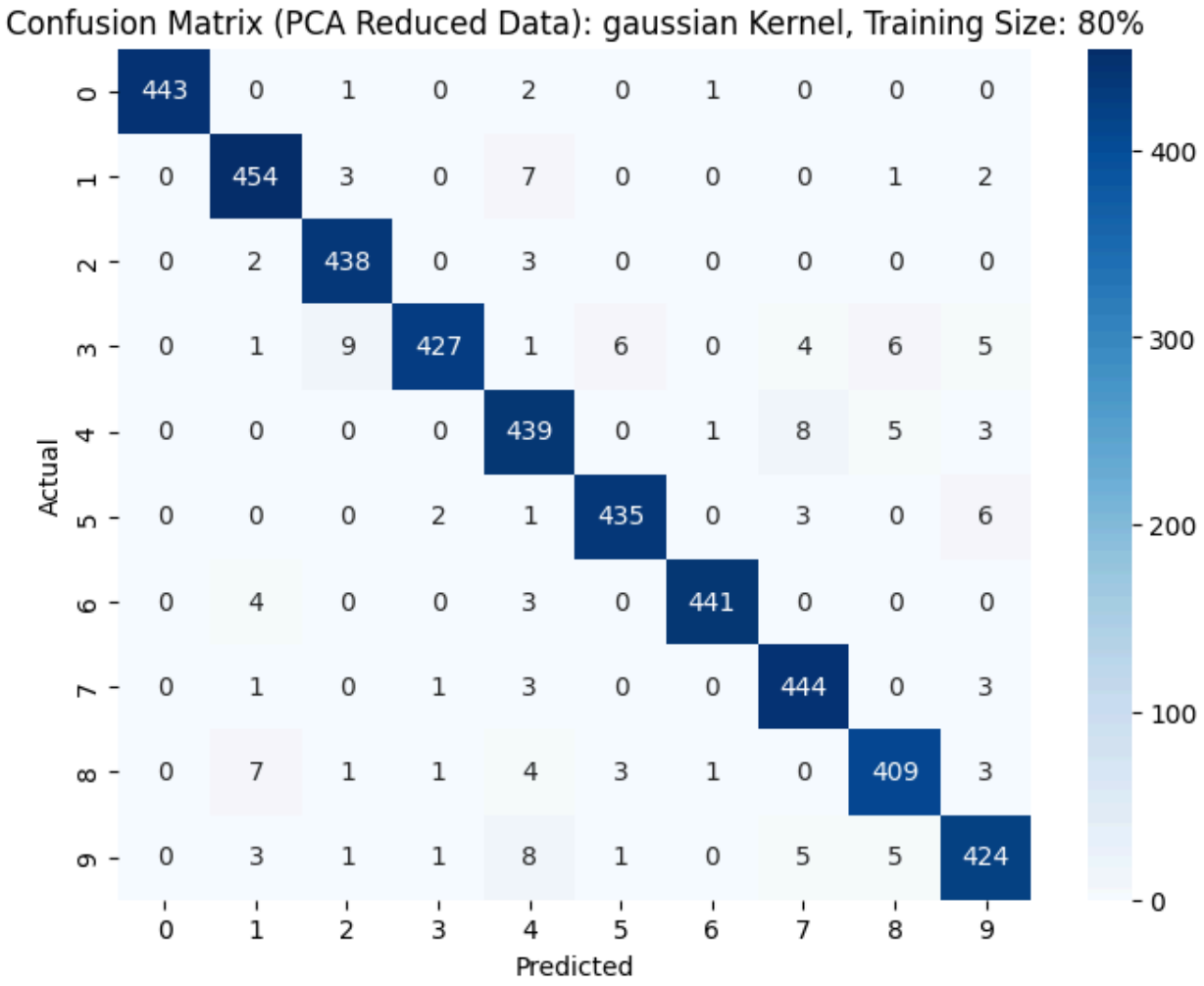

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 70%
Actual
Predicted
0 394 0 0 1 1 0 0 0 3 0
1 0 395 3 0 0 2 0 0 5 2
2 0 0 367 0 0 0 0 0 13 1
3 0 0 0 358 0 3 0 0 42 1
4 0 0 0 0 391 0 3 0 7 1
5 0 0 0 1 0 368 0 0 10 4
6 0 3 0 0 0 0 382 0 5 0
7 0 1 1 2 0 0 0 386 10 3
8 0 3 0 0 0 0 0 0 373 0
9 0 0 0 4 2 3 0 2 30 348
0 1 2 3 4 5 6 7 8 9
350
300
250
200
150
100
50
0

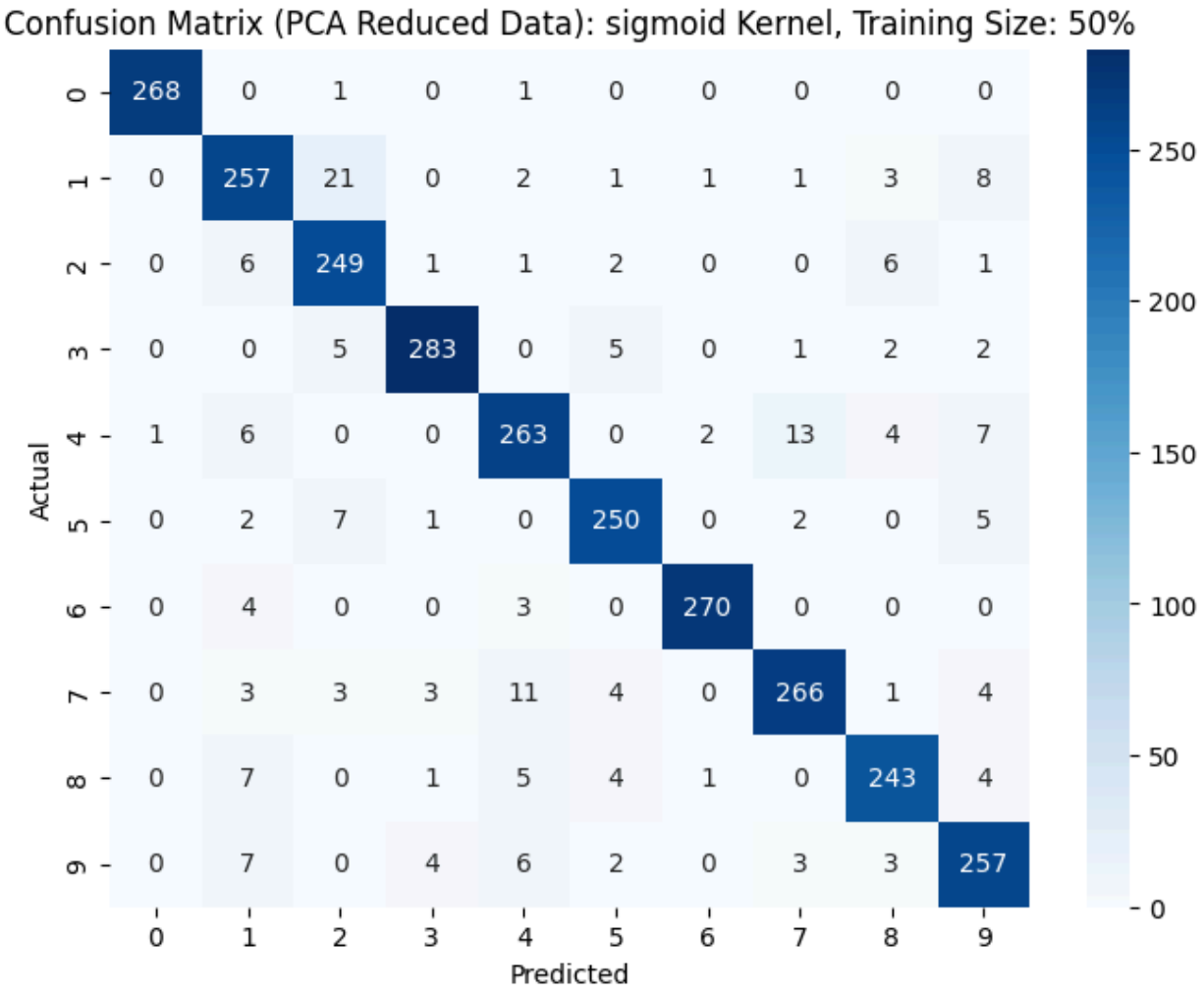

Confusion Matrix (PCA Reduced Data): polynomial Kernel, Training Size: 80%
Actual
Predicted
0 437 0 0 0 1 0 0 0 8 1
1 0 445 3 0 0 2 0 0 14 3
2 0 0 389 0 0 0 0 0 53 1
3 0 0 1 393 0 1 0 0 62 2
4 0 0 0 0 431 0 2 5 16 2
5 0 0 0 2 0 401 0 0 36 8
6 0 4 0 0 0 1 431 0 12 0
7 0 0 1 2 0 0 0 431 13 5
8 0 1 0 0 0 2 0 0 426 0
9 0 1 0 3 2 0 0 4 52 386
0 1 2 3 4 5 6 7 8 9
400
350
300
250
200
150
100
50
0

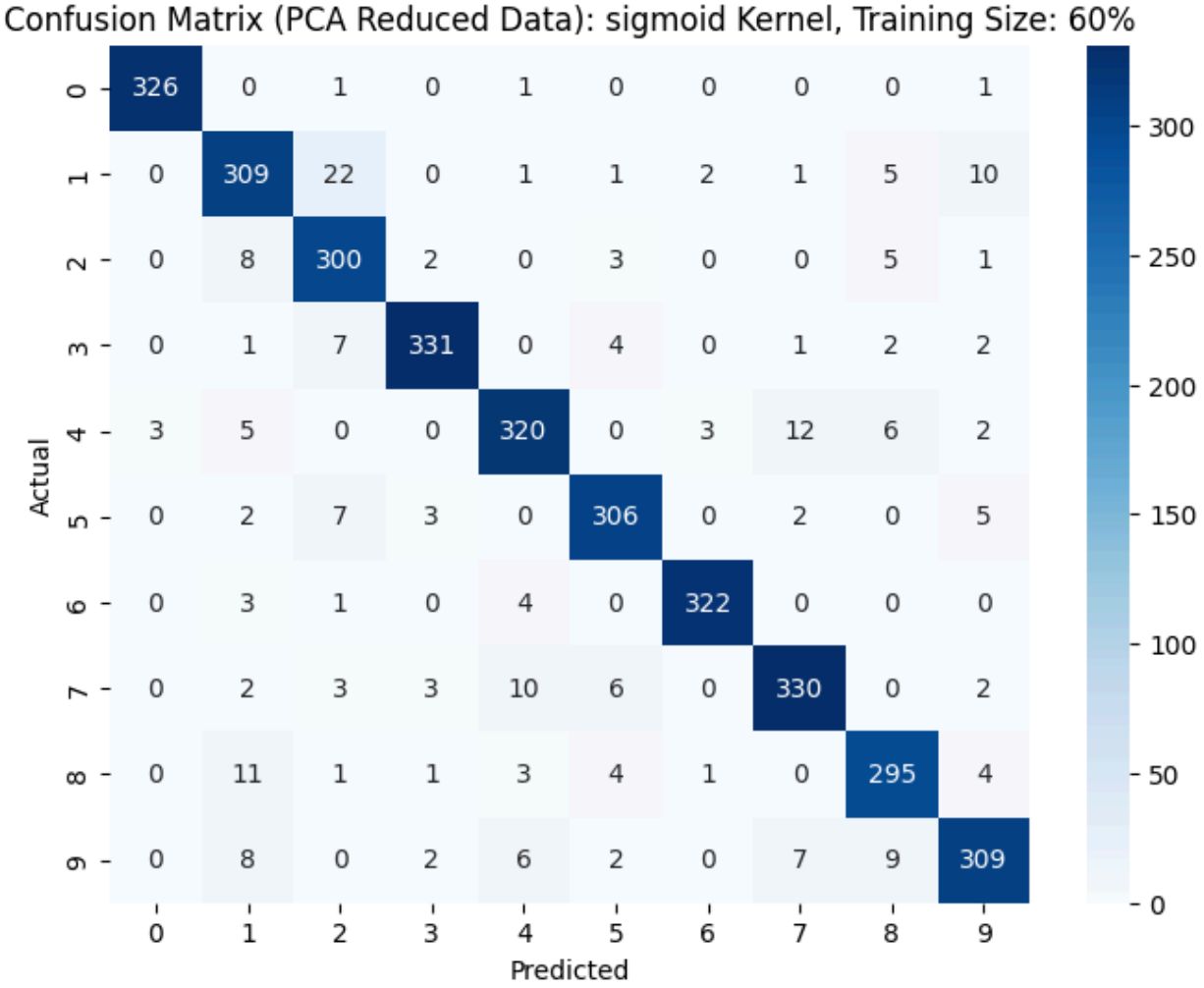

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 50%
Actual
Predicted
0 269 0 1 0 0 0 0 0 0 0
1 0 289 0 0 4 0 0 0 0 1
2 0 0 264 0 2 0 0 0 0 0
3 0 0 1 288 1 4 0 2 2 0
4 0 0 0 0 292 0 2 0 1 1
5 0 0 0 1 1 262 0 1 0 2
6 0 1 0 0 1 0 275 0 0 0
7 0 2 0 1 2 0 0 286 0 4
8 0 2 0 2 2 0 0 0 257 2
9 0 1 0 3 0 1 0 1 3 273
0 1 2 3 4 5 6 7 8 9
250
200
150
100
50
0

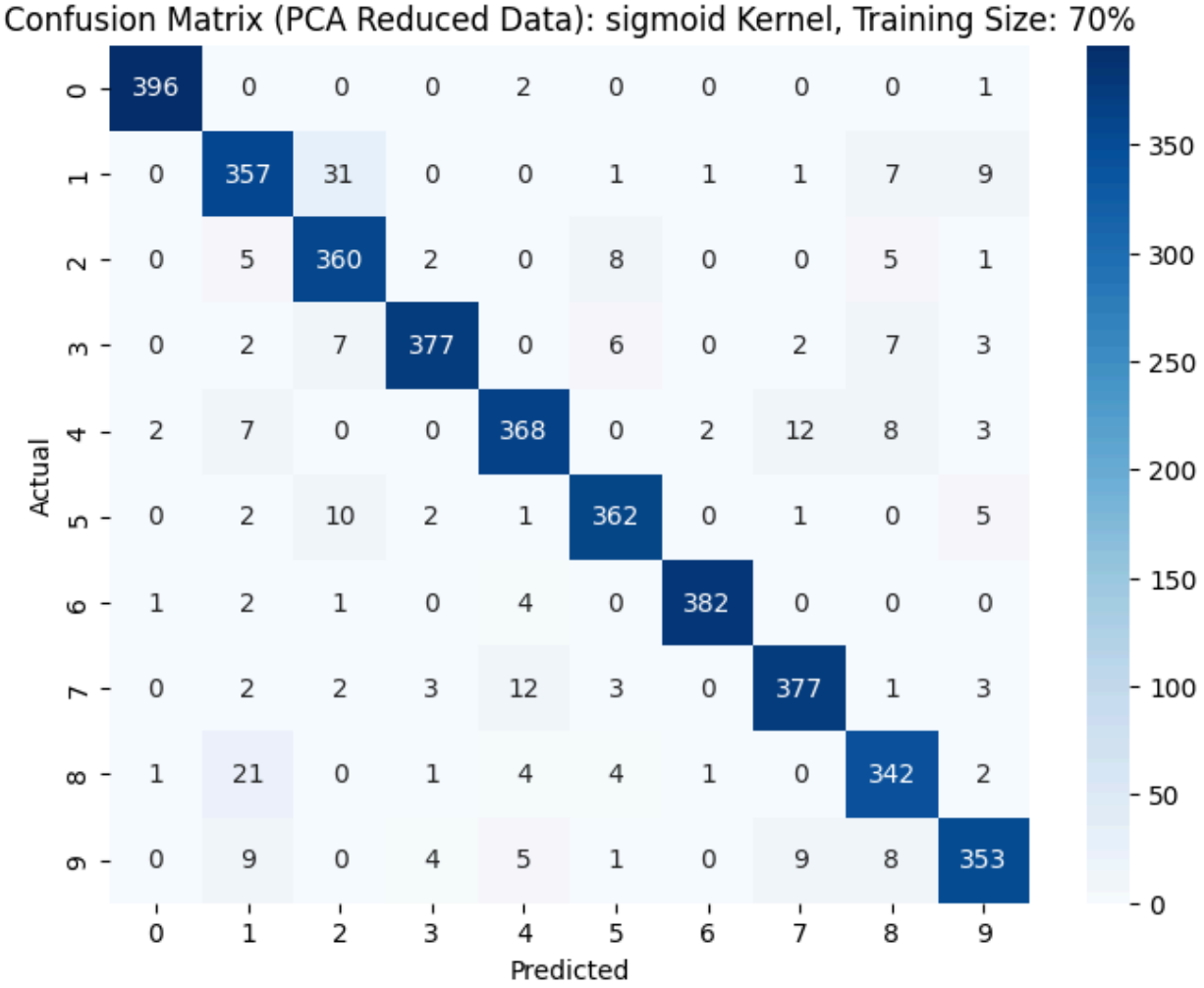

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 327 0 1 0 0 0 0 0 0 1
1 0 342 1 0 7 0 0 0 0 1
2 0 0 317 0 2 0 0 0 0 0
3 0 1 2 332 2 5 0 3 2 1
4 0 0 0 0 345 0 2 1 2 1
5 0 0 0 0 1 319 0 3 0 2
6 0 1 0 0 2 0 327 0 0 0
7 0 3 0 1 1 0 0 347 0 4
8 0 2 0 1 3 1 0 0 311 2
9 0 2 0 4 1 1 0 4 3 328
300
250
200
150
100
50
0

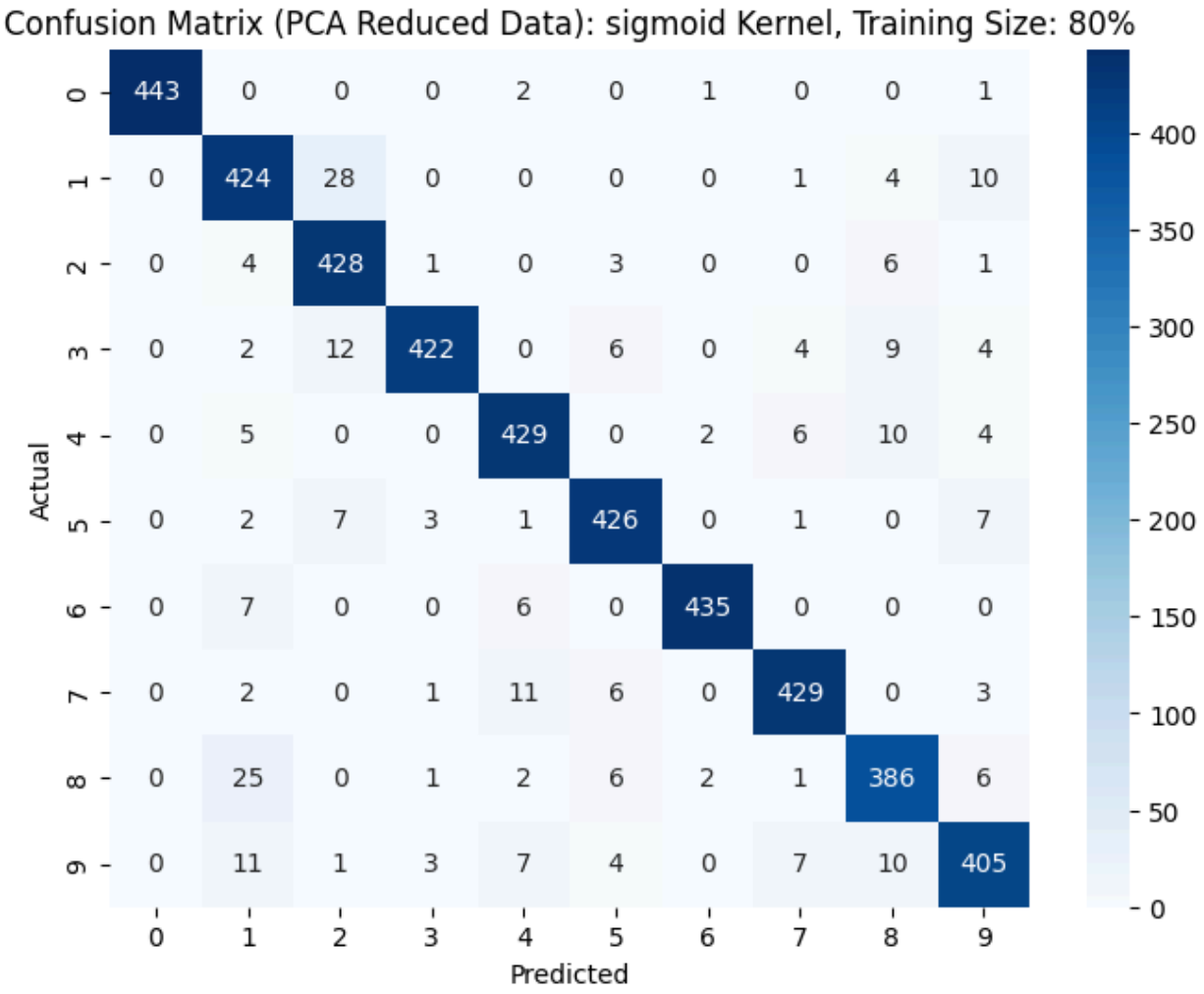

Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 70%
Actual
Predicted
0 396 0 1 0 1 0 0 0 0 1
1 0 397 1 0 7 0 0 0 0 2
2 0 0 378 0 3 0 0 0 0 0
3 0 2 5 379 1 5 0 3 7 2
4 0 0 0 0 393 0 1 5 2 1
5 0 0 0 0 1 374 0 3 0 5
6 0 1 0 0 2 0 387 0 0 0
7 0 2 0 1 2 0 0 396 0 2
8 0 3 0 1 3 4 1 0 362 2
9 0 2 0 5 2 1 0 2 3 374
0 1 2 3 4 5 6 7 8 9
350
300
250
200
150
100
50
0


Confusion Matrix (PCA Reduced Data): gaussian Kernel, Training Size: 80%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
443 0 1 0 2 0 1 0 0 0
0 454 3 0 7 0 0 0 1 2
0 2 438 0 3 0 0 0 0 0
0 1 9 427 1 6 0 4 6 5
0 0 0 0 439 0 1 8 5 3
0 0 0 2 1 435 0 3 0 6
0 4 0 0 3 0 441 0 0 0
0 1 0 1 3 0 0 444 0 3
0 7 1 1 4 3 1 0 409 3
0 3 1 1 8 1 0 5 5 424
400
300
200
100
0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 50%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
268 0 1 0 1 0 0 0 0 0
0 257 21 0 2 1 1 1 3 8
0 6 249 1 1 2 0 0 6 1
0 0 5 283 0 5 0 1 2 2
1 6 0 0 263 0 2 13 4 7
0 2 7 1 0 250 0 2 0 5
0 4 0 0 3 0 270 0 0 0
0 3 3 3 11 4 0 266 1 4
0 7 0 1 5 4 1 0 243 4
0 7 0 4 6 2 0 3 3 257
250
200
150
100
50
0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
326 0 1 0 1 0 0 0 0 1
0 309 22 0 1 1 2 1 5 10
0 8 300 2 0 3 0 0 5 1
0 1 7 331 0 4 0 1 2 2
3 5 0 0 320 0 3 12 6 2
0 2 7 3 0 306 0 2 0 5
0 3 1 0 4 0 322 0 0 0
0 2 3 3 10 6 0 330 0 2
0 11 1 1 3 4 1 0 295 4
0 8 0 2 6 2 0 7 9 309
300 250 200 150 100 50 0


Confusion Matrix (PCA Reduced Data): sigmoid Kernel, Training Size: 70%
Actual
Predicted
0 396 0 0 0 2 0 0 0 0 1
1 0 357 31 0 0 1 1 1 7 9
2 0 5 360 2 0 8 0 0 5 1
3 0 2 7 377 0 6 0 2 7 3
4 2 7 0 0 368 0 2 12 8 3
5 0 2 10 2 1 362 0 1 0 5
6 1 2 1 0 4 0 382 0 0 0
7 0 2 2 3 12 3 0 377 1 3
8 1 21 0 1 4 4 1 0 342 2
9 0 9 0 4 5 1 0 9 8 353
0 1 2 3 4 5 6 7 8 9
350 300 250 200 150 100 50 0



===== 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
    })
```

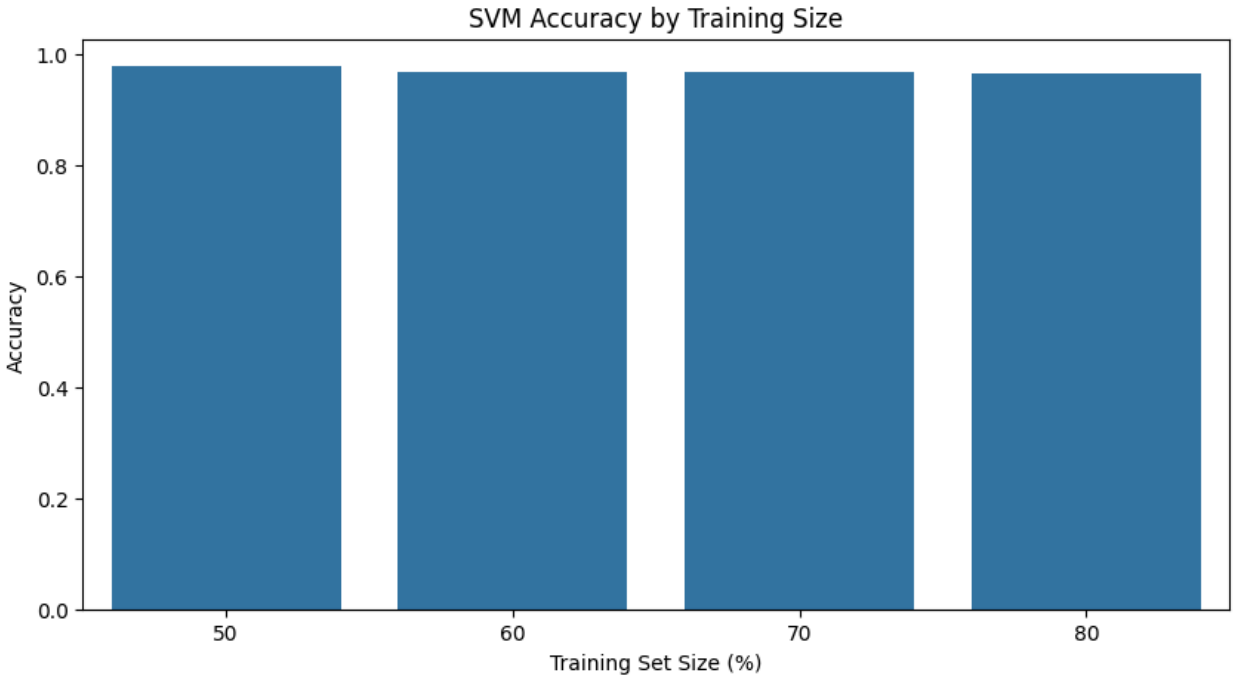
Print Evalution Table and Graphs

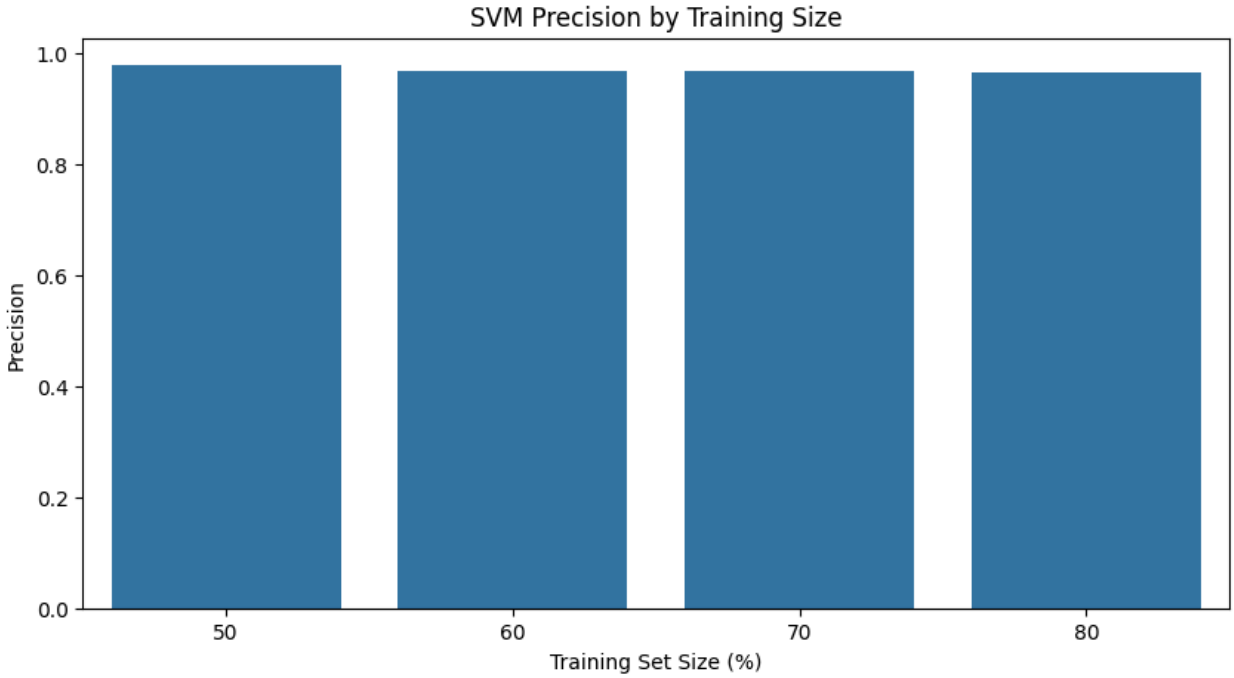
In [ ]:

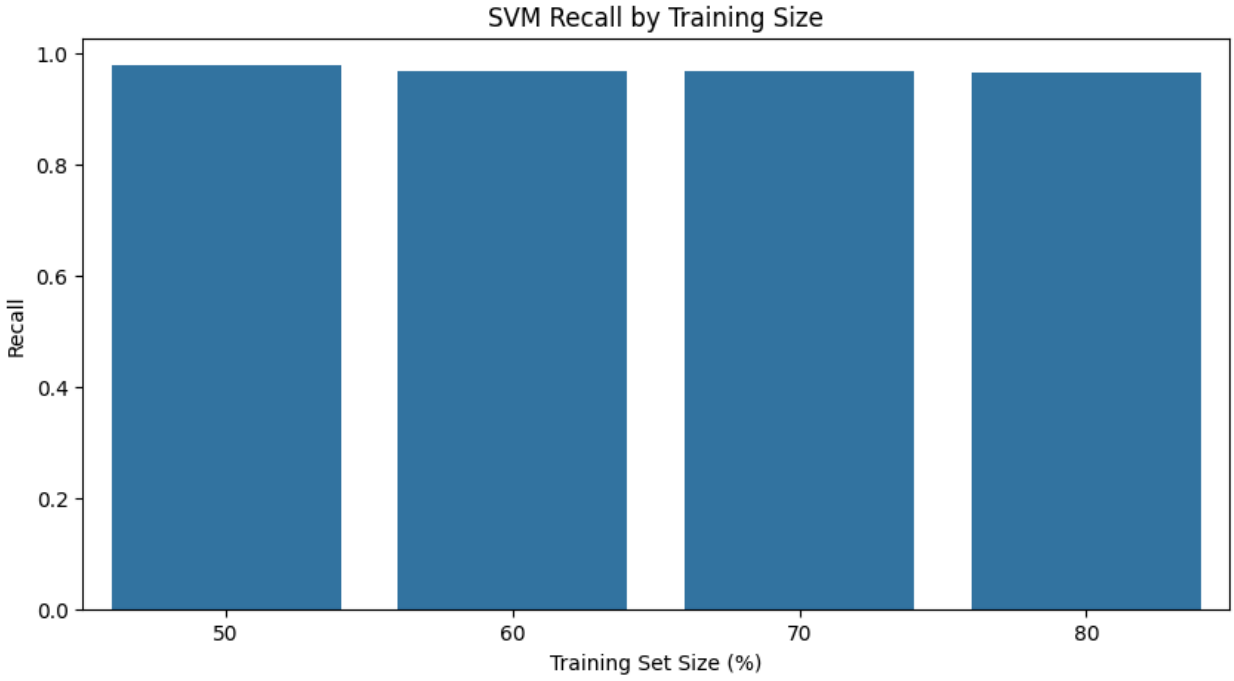
```
df = pd.DataFrame(results_mlp)
display(df) # Display the DataFrame as a table

# 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()
```

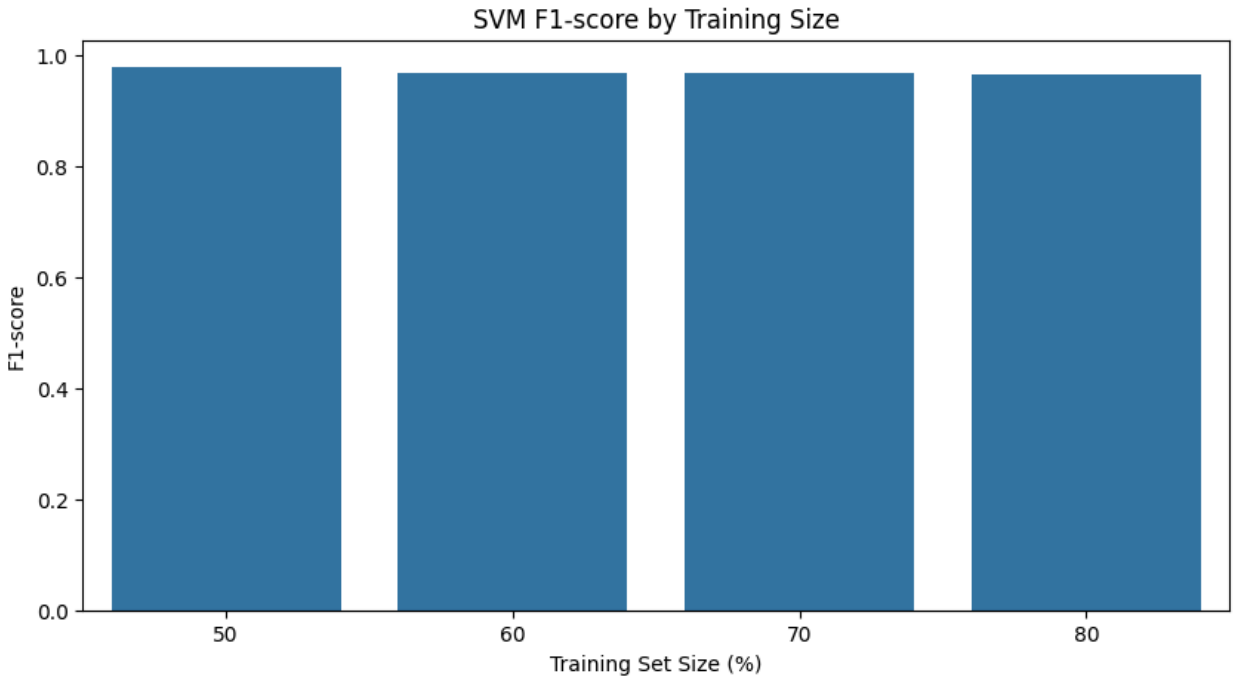
```
# Plot Recall
plt.figure(figsize=(10, 5)) # Increase figure size
sns.barplot(data=df, x='Training size', y='Recall')
plt.title('SVM Recall by Training Size') # Add title
plt.xlabel('Training Set Size (%)') # Add x-axis label
plt.ylabel('Recall') # Add y-axis label
plt.show()

# 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	0.978648	0.978861	0.978648	0.978668
1	60	0.968861	0.969049	0.968861	0.968876
2	70	0.966955	0.967166	0.966955	0.966987
3	80	0.964635	0.965304	0.964635	0.964730




SVM Precision by Training Size
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

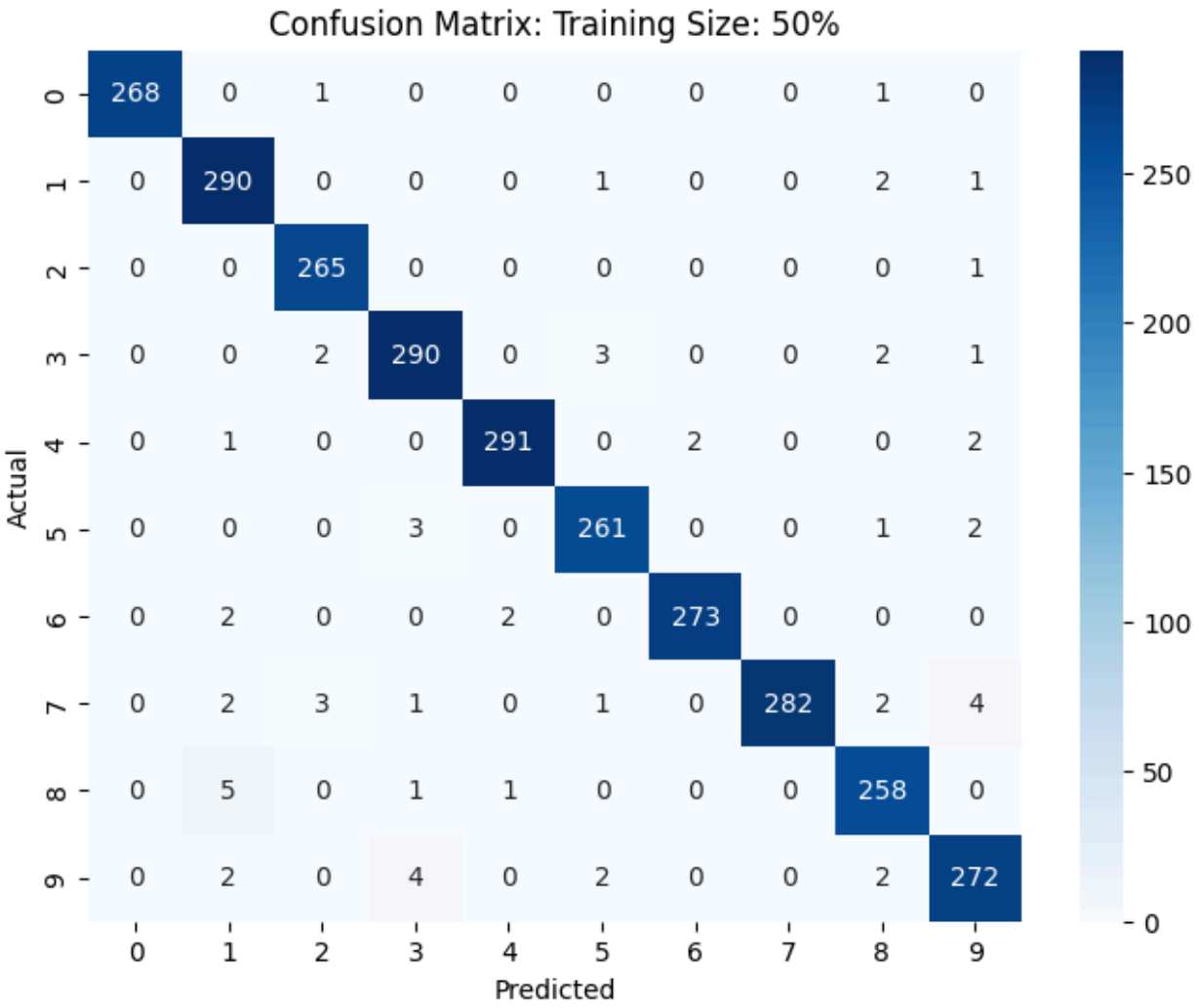

SVM Recall by Training Size
Recall
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

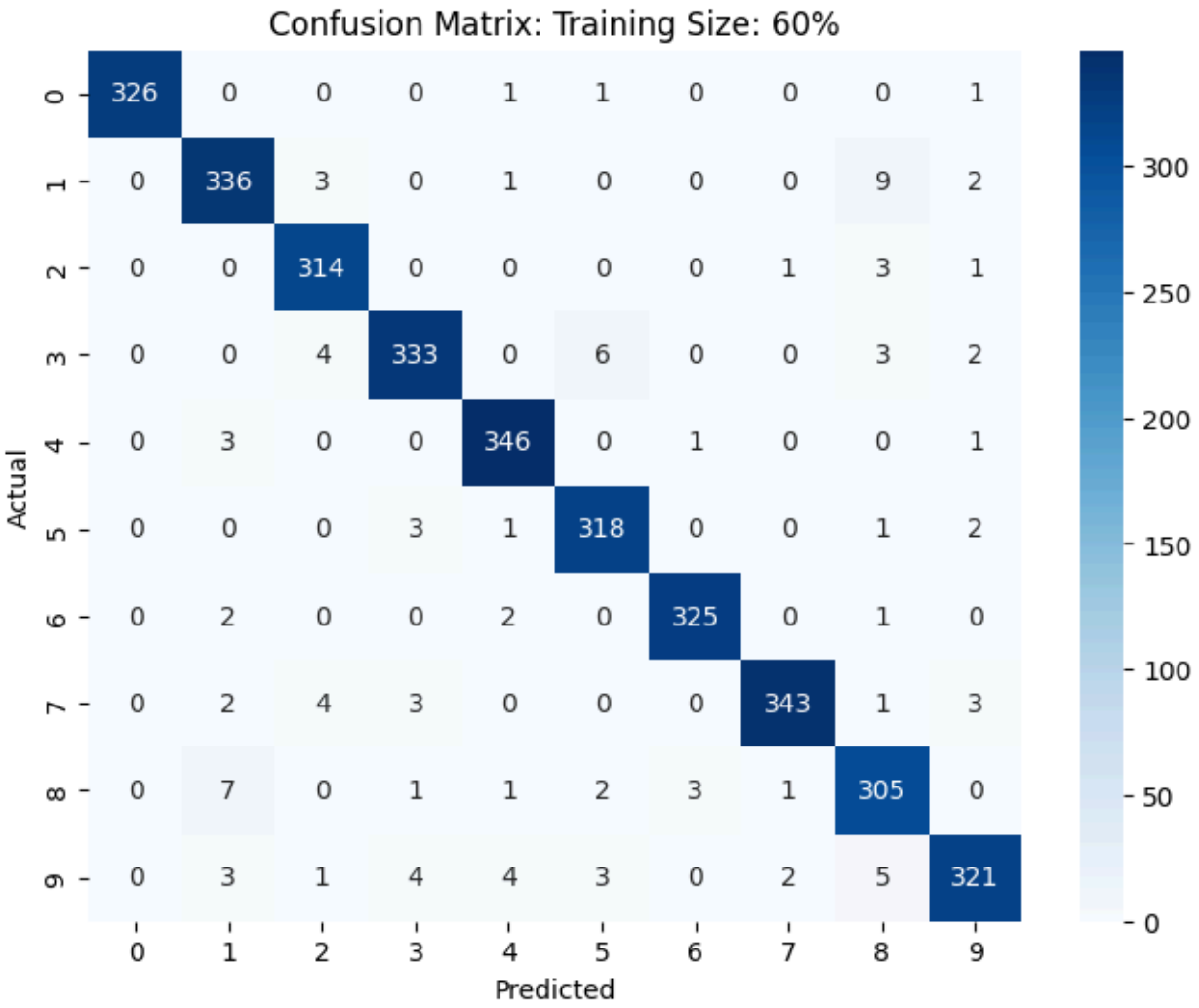


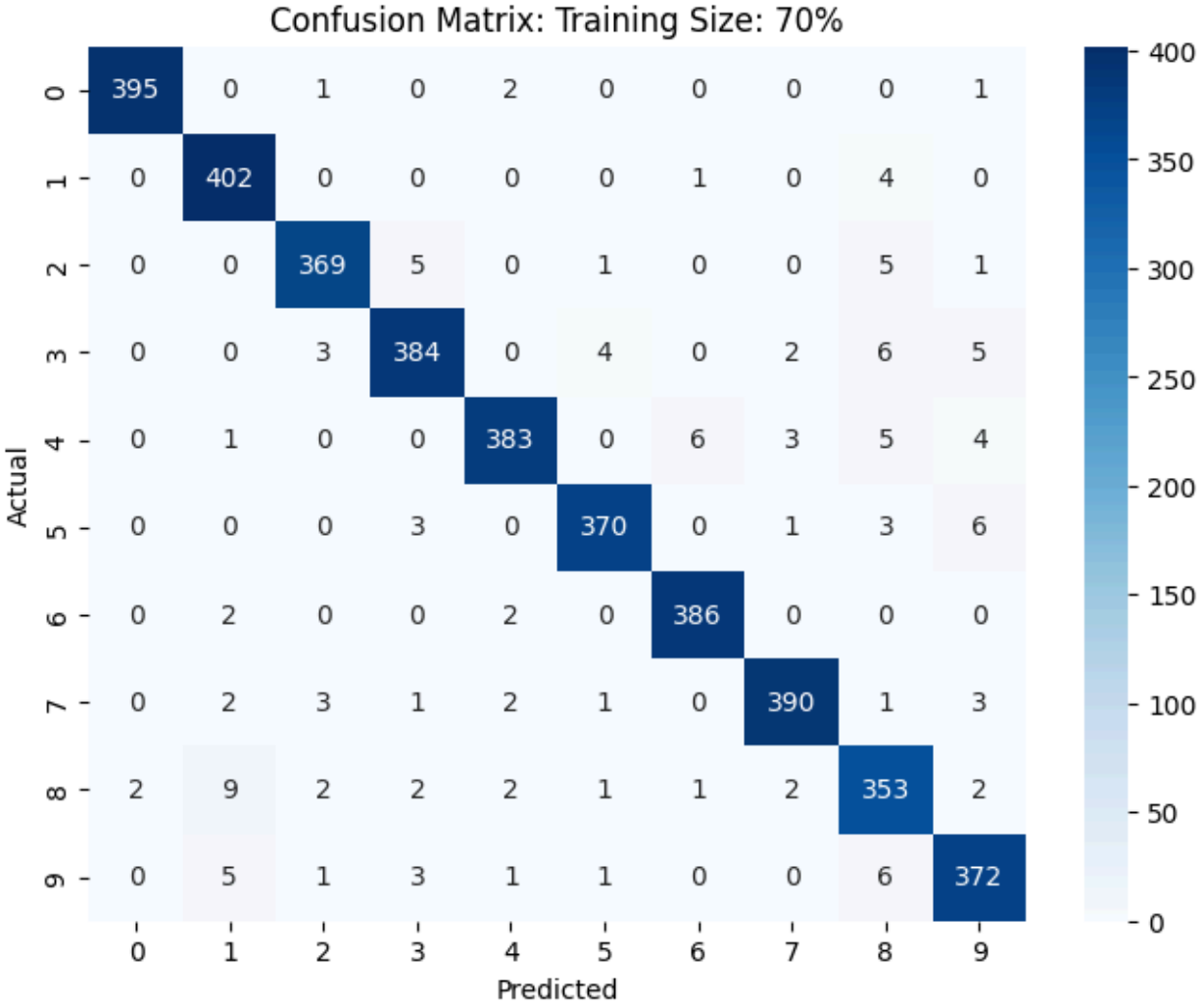
Show Confusion Matrices

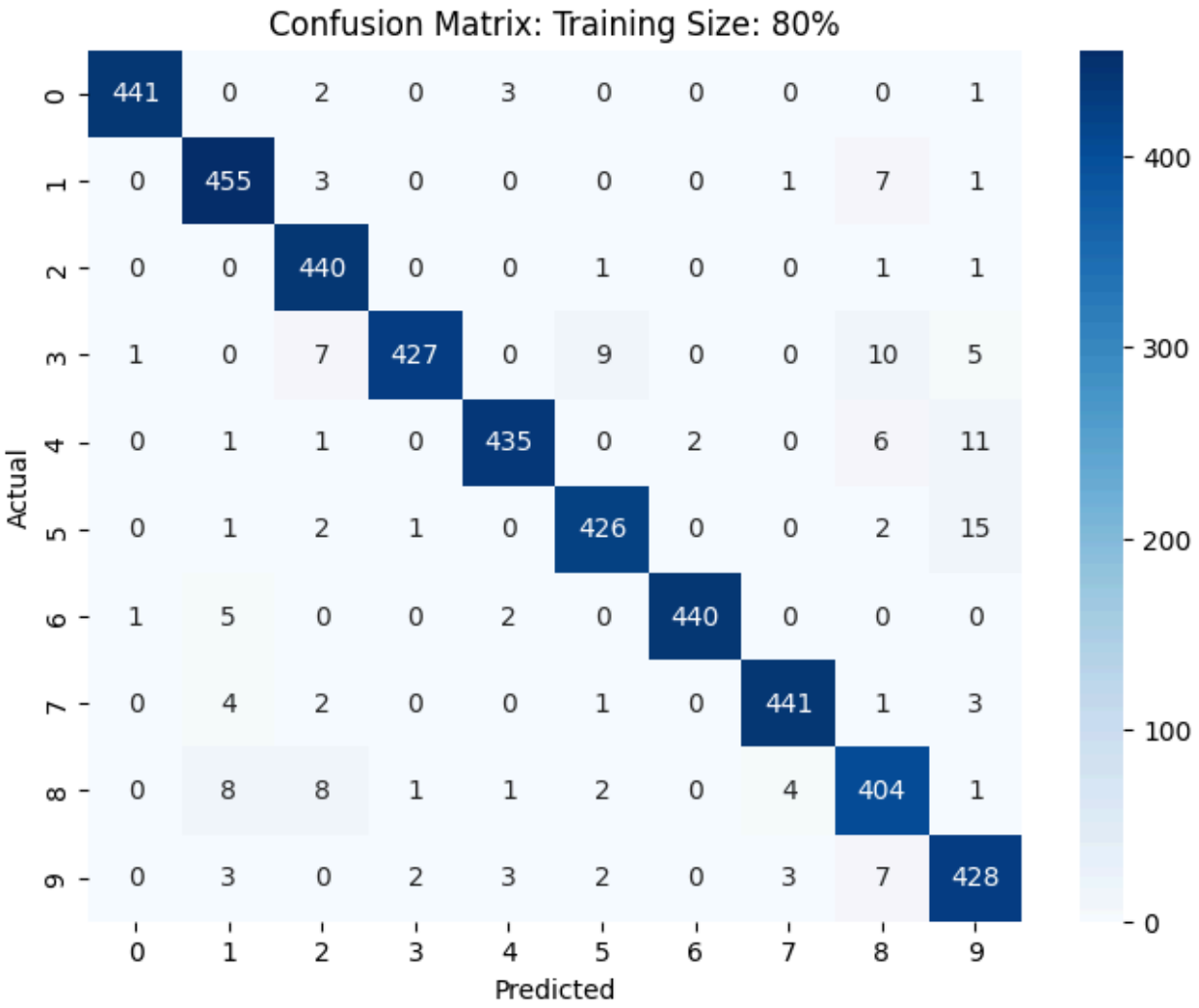
In [ ]:

```
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%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
268 0 1 0 0 0 0 0 1 0
0 290 0 0 0 1 0 0 2 1
0 0 265 0 0 0 0 0 0 1
0 0 2 290 0 3 0 0 2 1
0 1 0 0 291 0 2 0 0 2
0 0 0 3 0 261 0 0 1 2
0 2 0 0 2 0 273 0 0 0
0 2 3 1 0 1 0 282 2 4
0 5 0 1 1 0 0 0 258 0
0 2 0 4 0 2 0 0 2 272
250
200
150
100
50
0


Confusion Matrix: Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 326 0 0 0 1 1 0 0 0 1
1 0 336 3 0 1 0 0 0 9 2
2 0 0 314 0 0 0 0 1 3 1
3 0 0 4 333 0 6 0 0 3 2
4 0 3 0 0 346 0 1 0 0 1
5 0 0 0 3 1 318 0 0 1 2
6 0 2 0 0 2 0 325 0 1 0
7 0 2 4 3 0 0 0 343 1 3
8 0 7 0 1 1 2 3 1 305 0
9 0 3 1 4 4 3 0 2 5 321
300
250
200
150
100
50
0

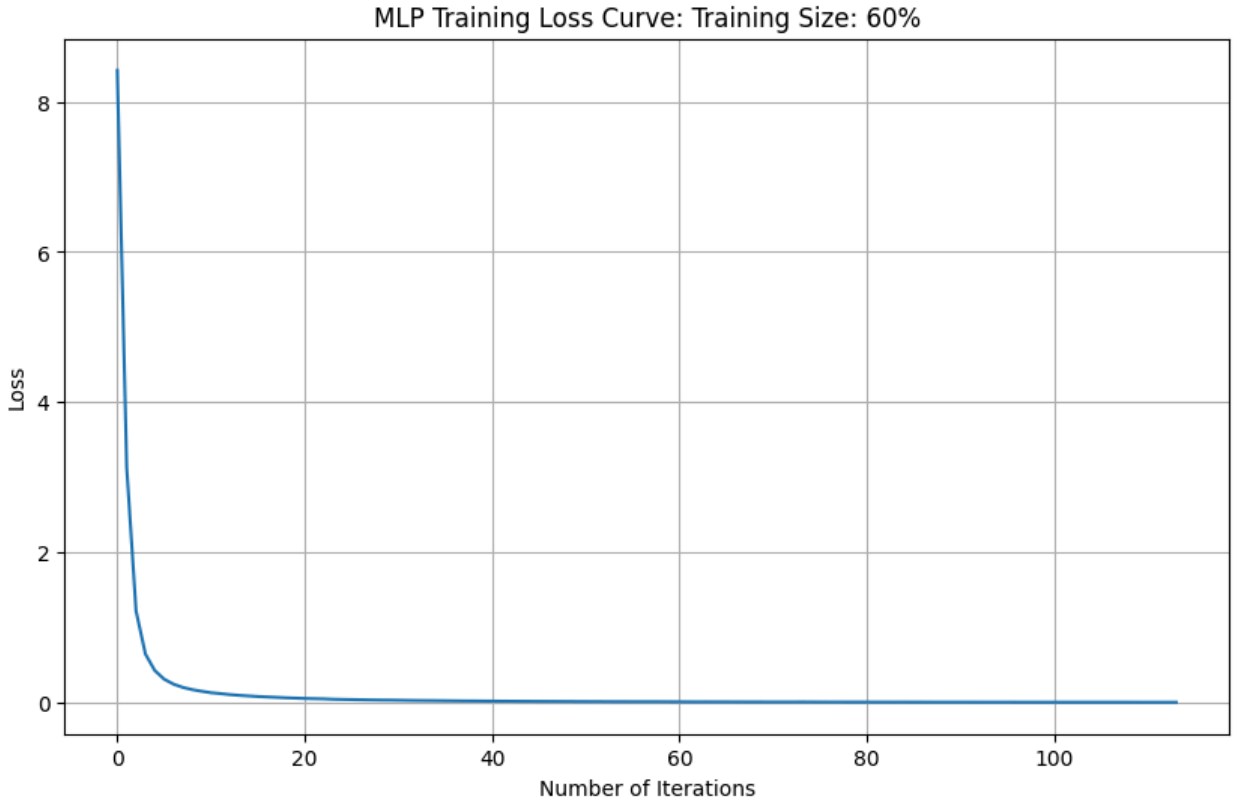

Confusion Matrix: Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 395 0 1 0 2 0 0 0 0 1
1 0 402 0 0 0 0 1 0 4 0
2 0 0 369 5 0 1 0 0 5 1
3 0 0 3 384 0 4 0 2 6 5
4 0 1 0 0 383 0 6 3 5 4
5 0 0 0 3 0 370 0 1 3 6
6 0 2 0 0 2 0 386 0 0 0
7 0 2 3 1 2 1 0 390 1 3
8 2 9 2 2 2 1 1 2 353 2
9 0 5 1 3 1 1 0 0 6 372
400
350
300
250
200
150
100
50
0

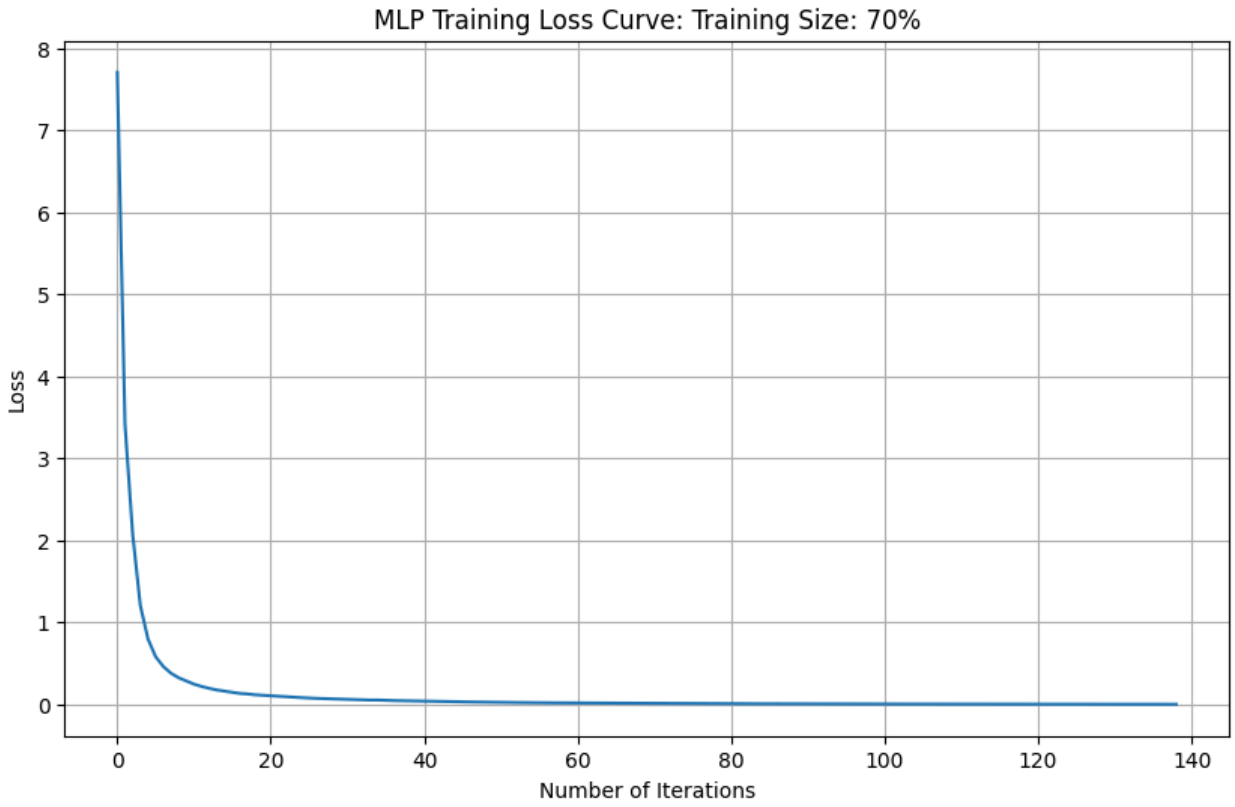


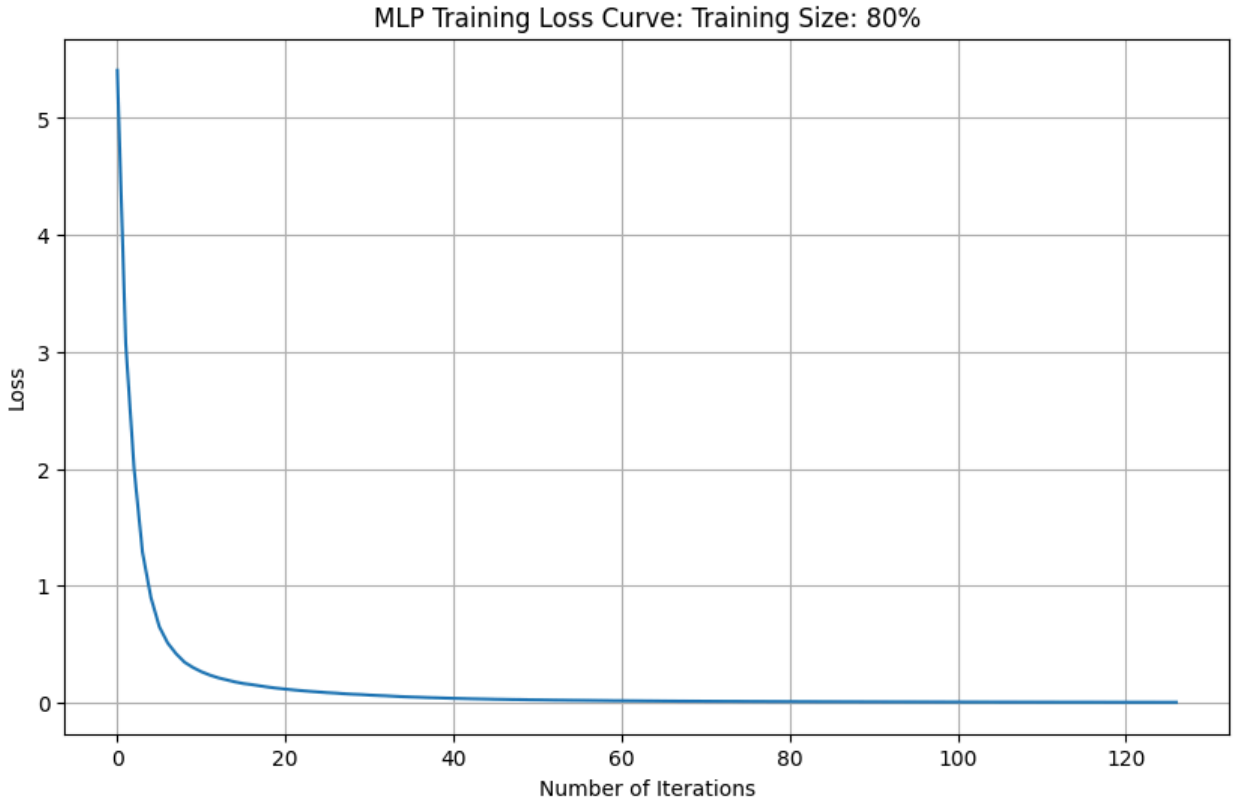
Training Loss generation curve

In [ ]:
```python
# 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%
Loss
Number of Iterations
0
1
2
3
4
5
6
7
0
20
40
60
80
100
120
140


MLP Training Loss Curve: Training Size: 60%
Loss
Number of Iterations
0
2
4
6
8
0
20
40
60
80
100



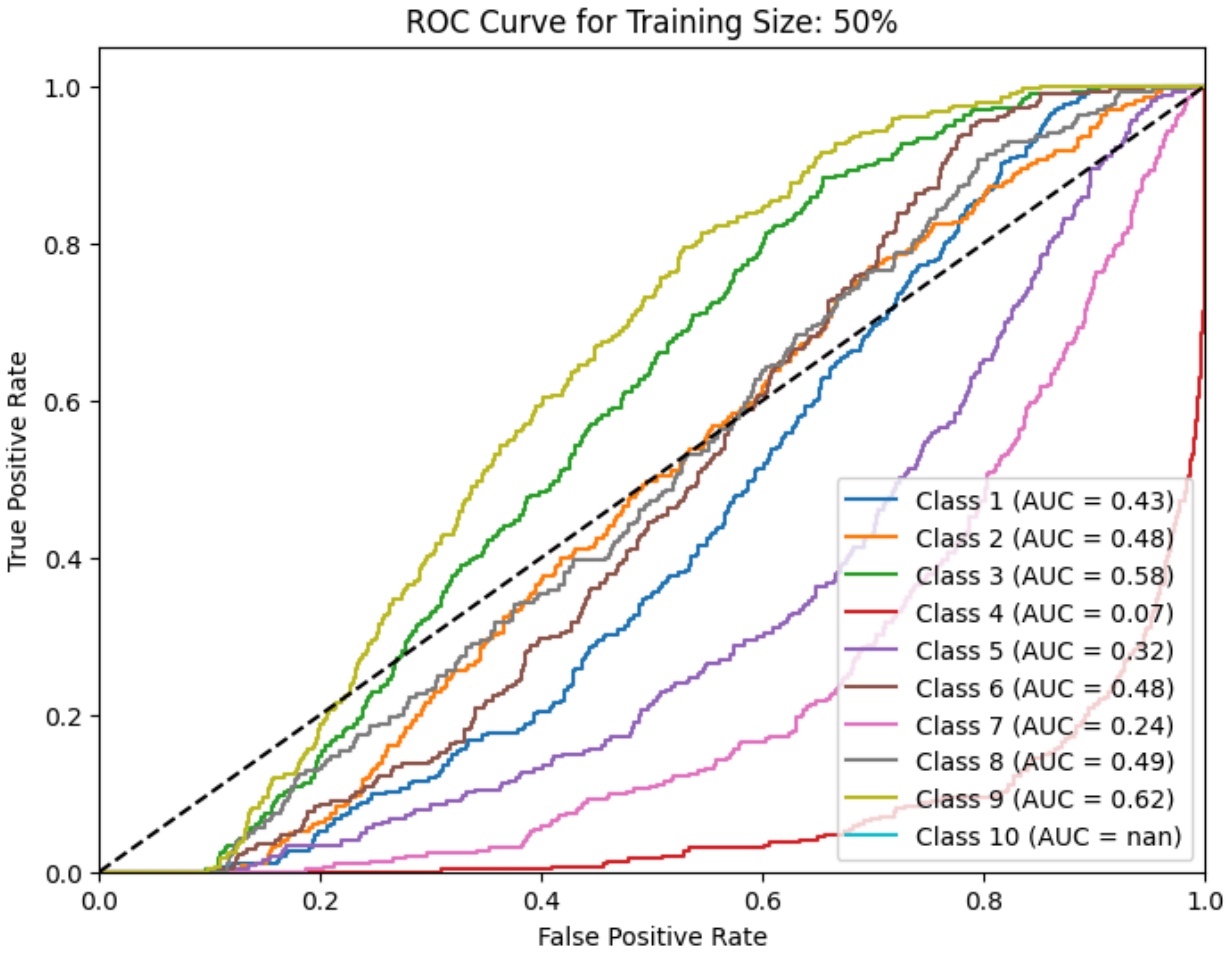


ROC and AOC curve

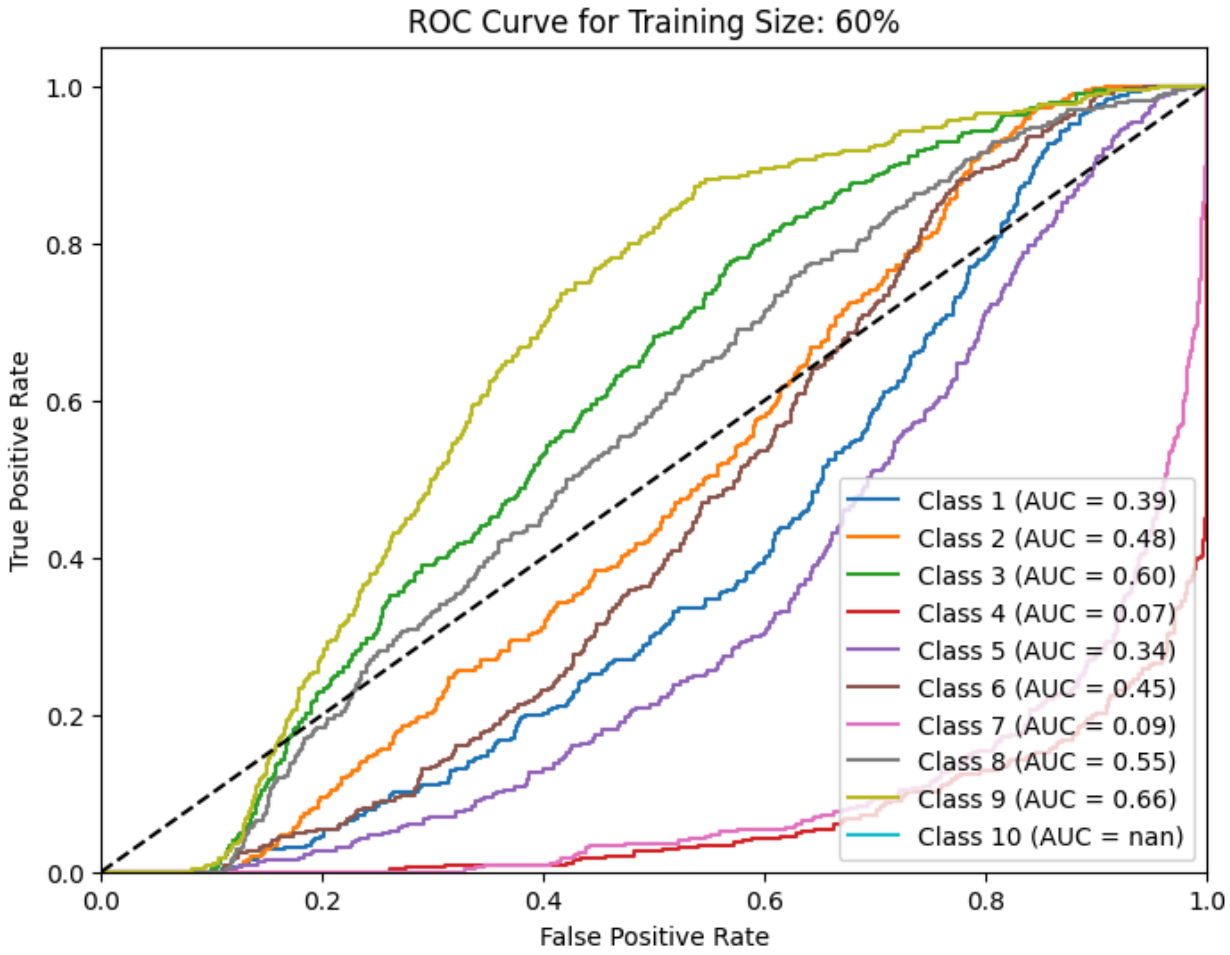
In [ ]:
```
from sklearn.metrics import roc_curve, auc
```

```python
# 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 fo
    plt.legend(loc="lower right")
    plt.show()
```

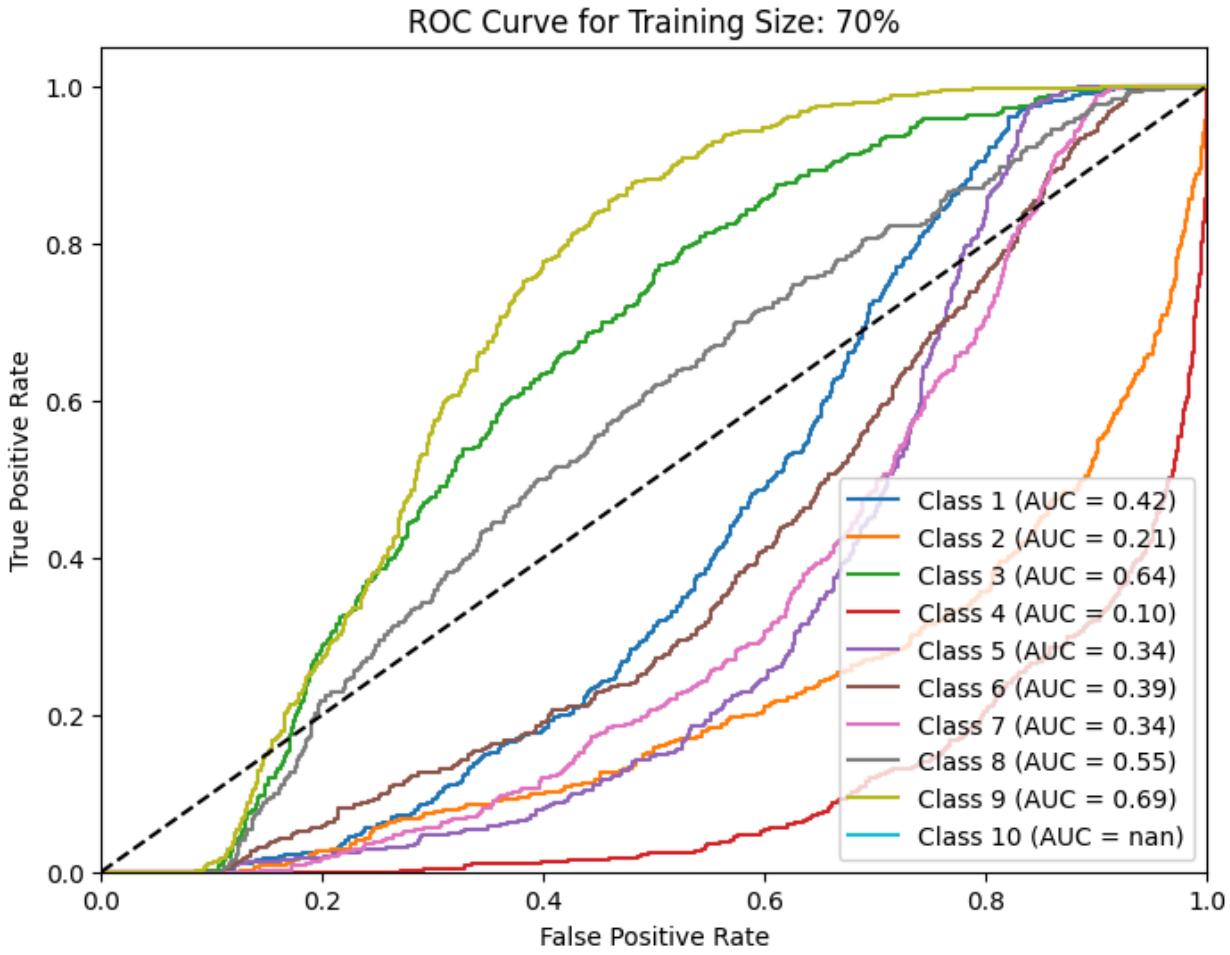
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



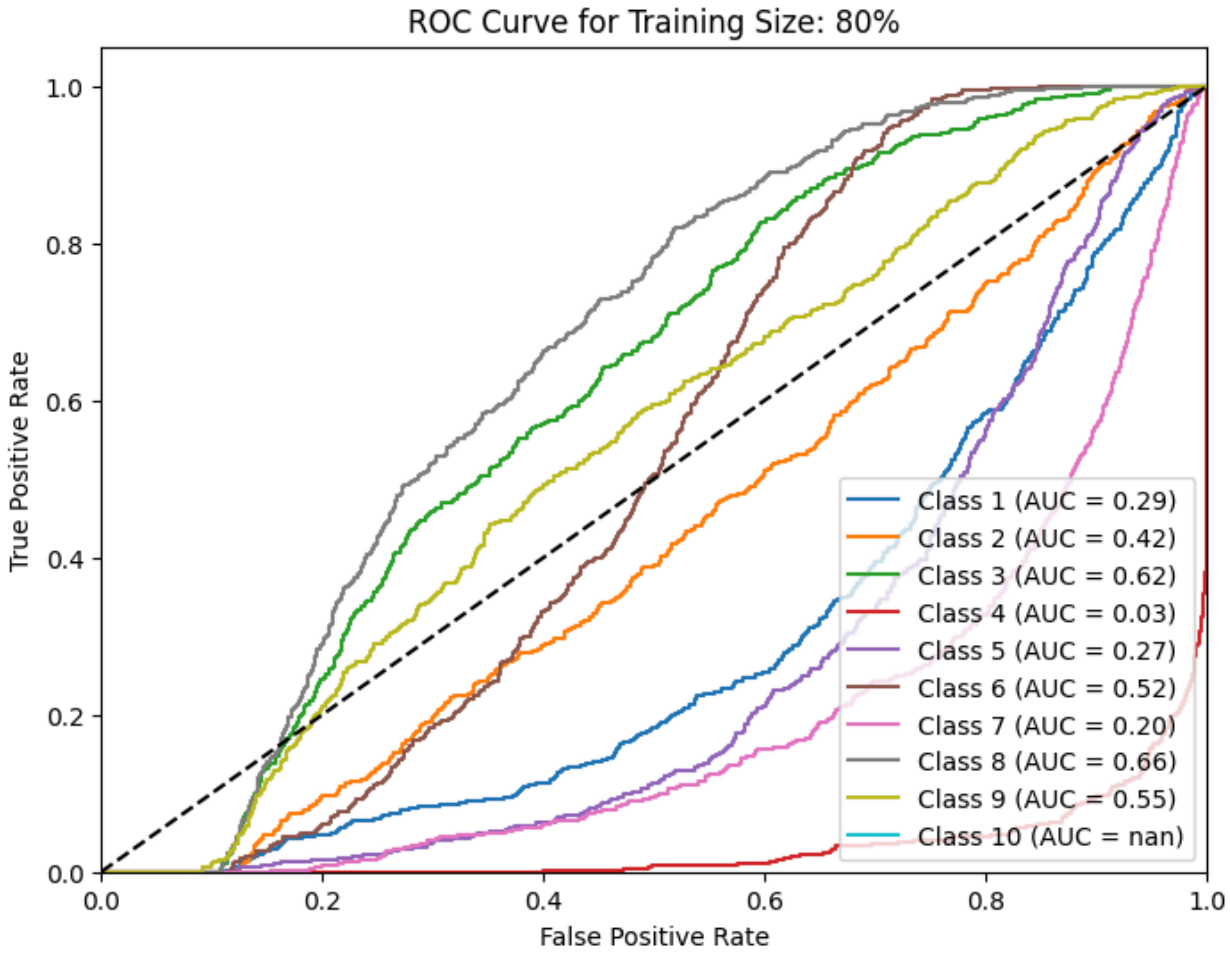

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```




```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: 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: Undef
inedMetricWarning: 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:
```

	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

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

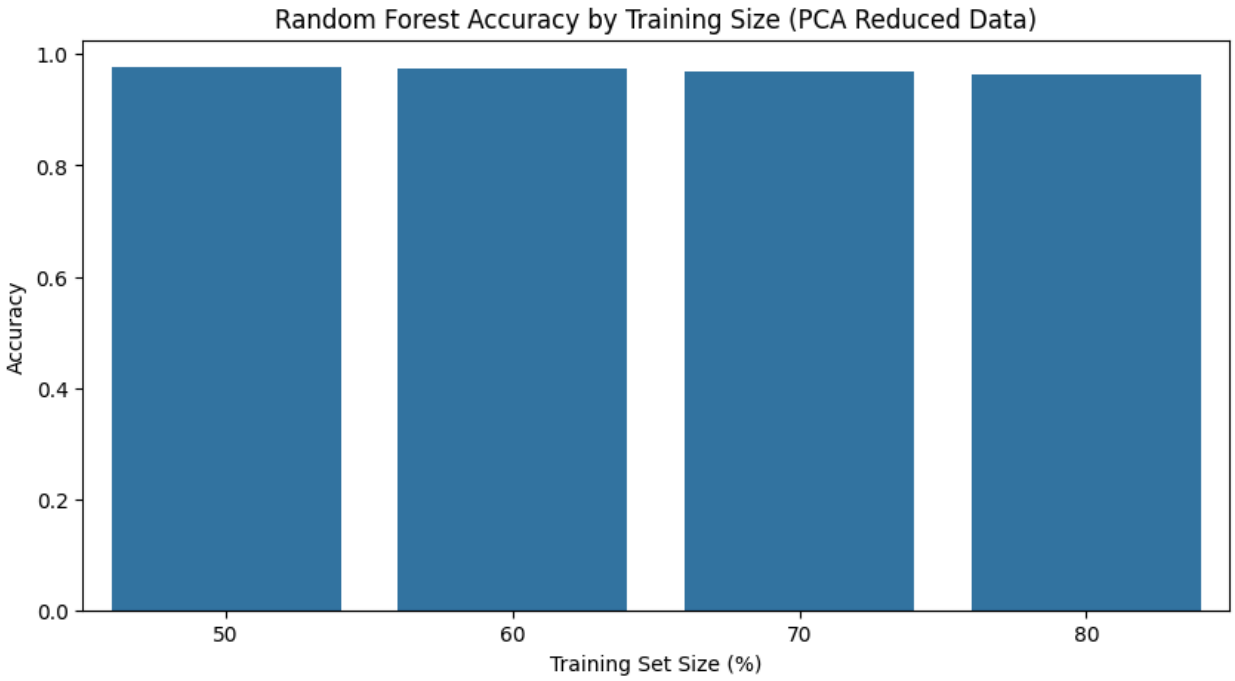
In [ ]:
```
results_mlp_pca = []
confusion_matrices_mlp_pca = []
trained_models_mlp_pca = []


for size in training_sizes:
  X_train_pca_mlp, X_test_pca_mlp, y_train, y_test = train_test_split(X_pca_re
  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='weighte
  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,
```

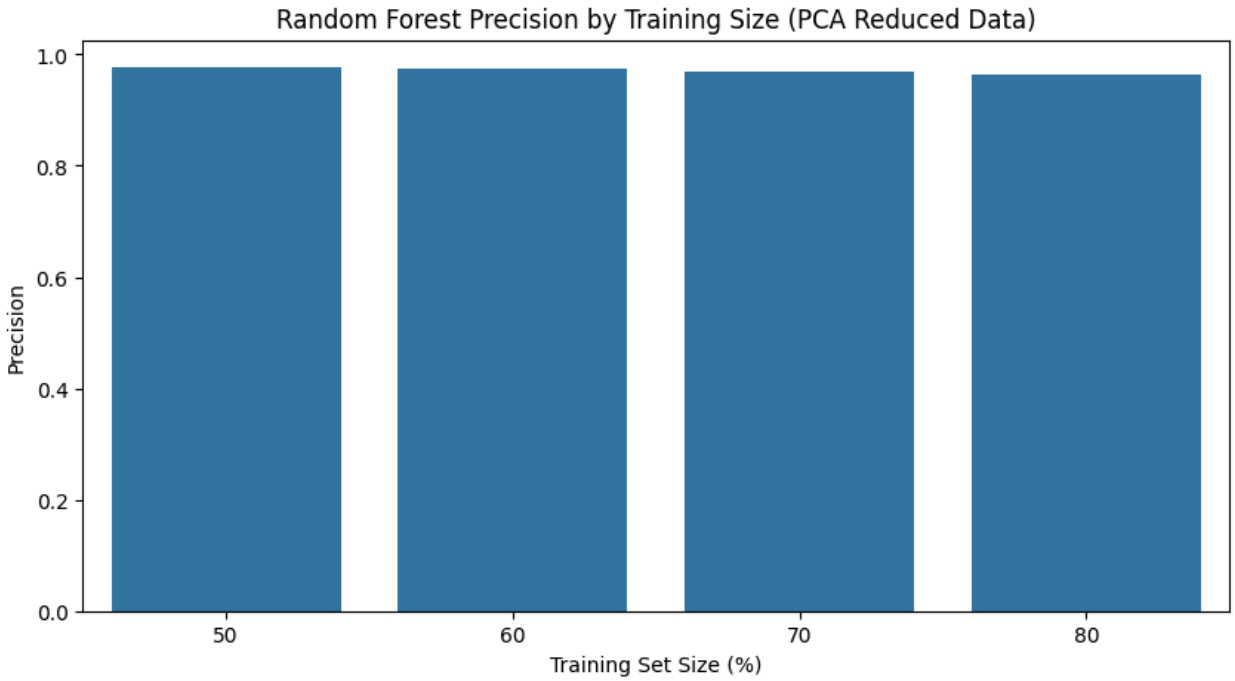
```
})
```

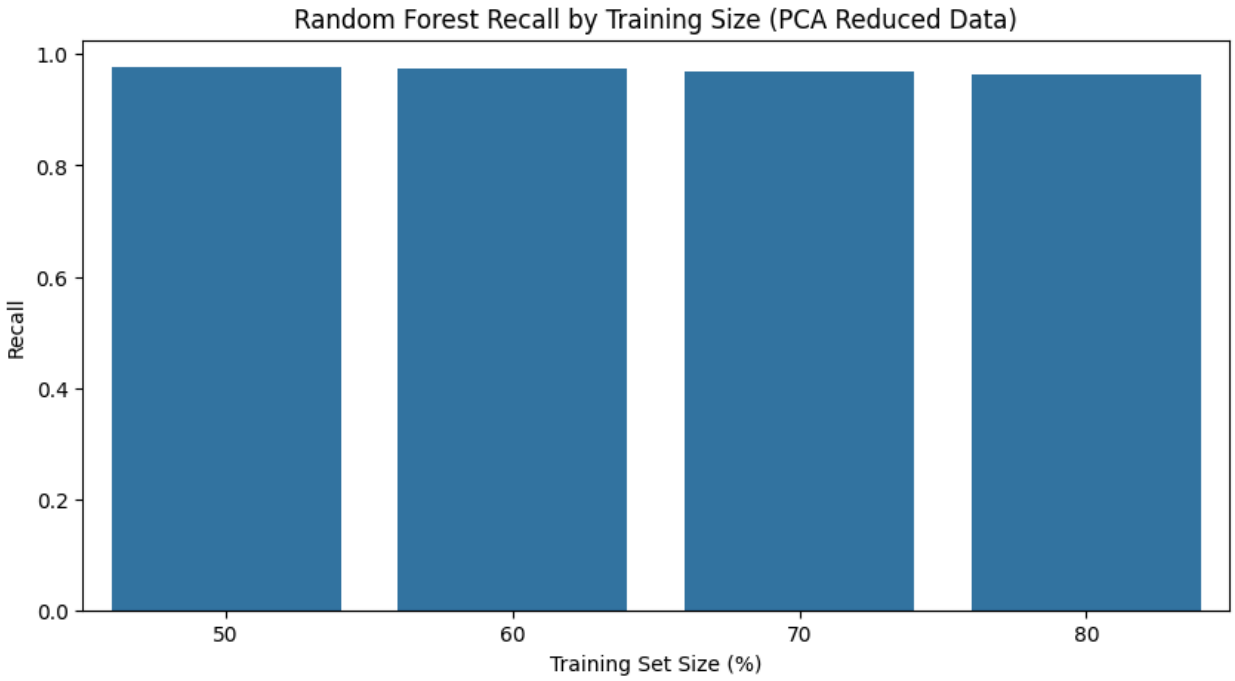
Print MLP Performance Table and Graphs with PCA-Reduced Data

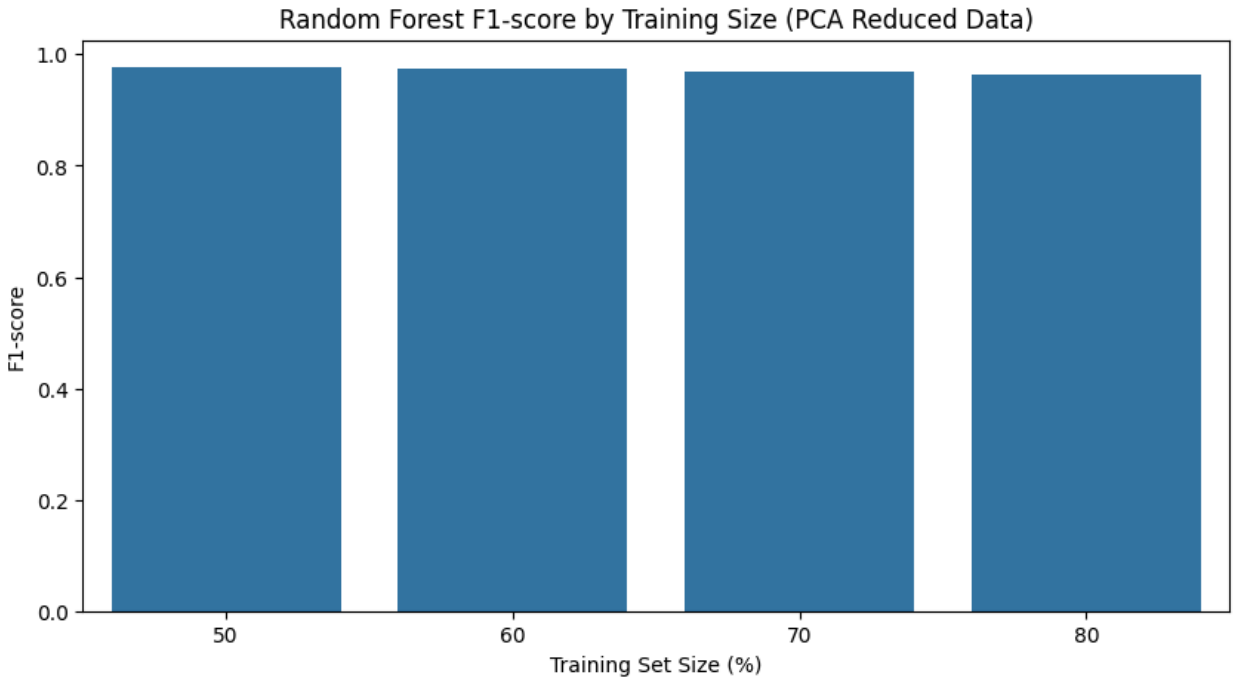
In [ ]:
```
df_mlp_pca = pd.DataFrame(results_mlp_pca)
display(df_mlp_pca)

# 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	0.976868	0.976951	0.976868	0.976883
1	60	0.973310	0.973406	0.973310	0.973321
2	70	0.968480	0.968640	0.968480	0.968490
3	80	0.962856	0.963075	0.962856	0.962863


Random Forest Accuracy by Training Size (PCA Reduced Data)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)


Random Forest Precision by Training Size (PCA Reduced Data)
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

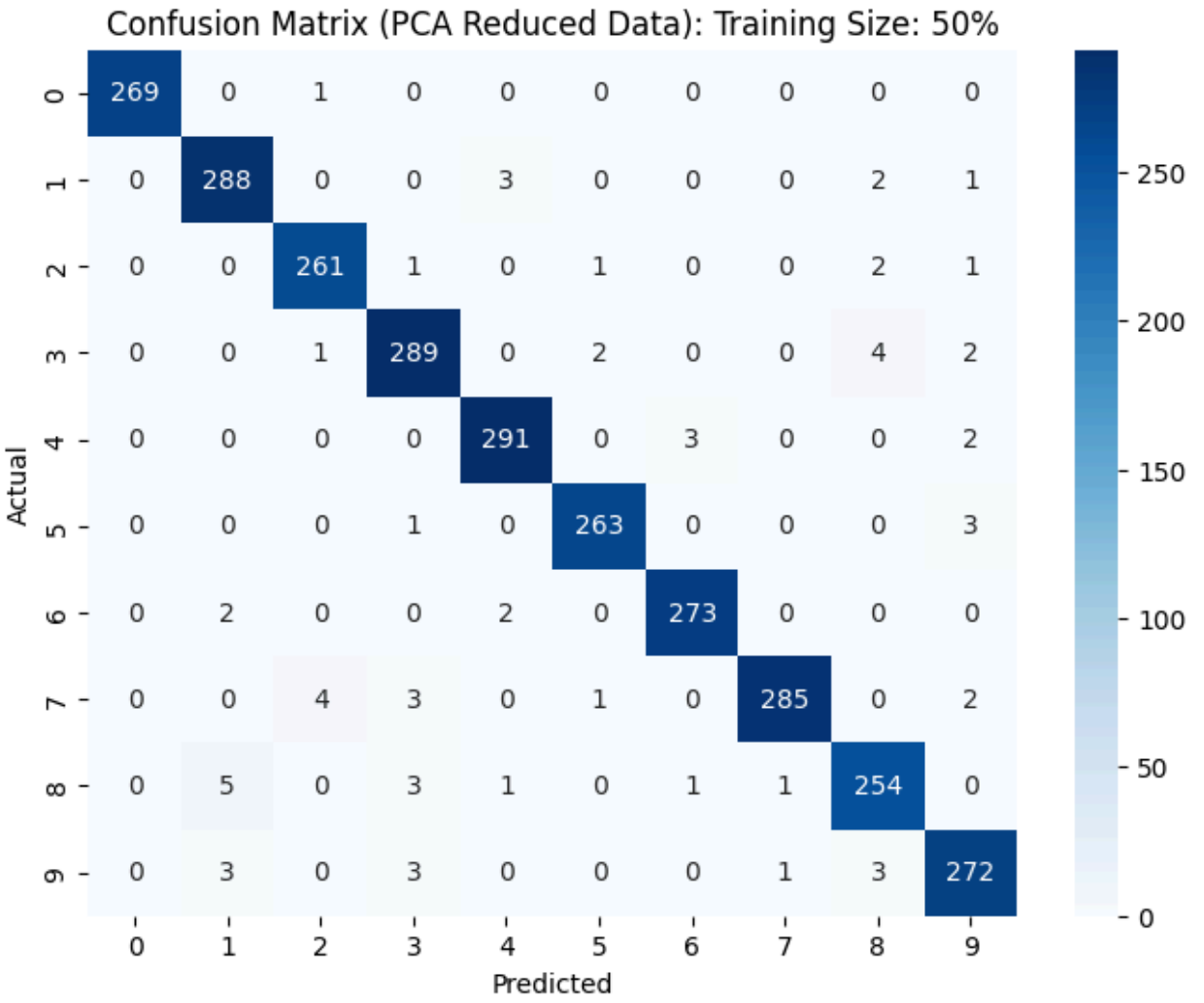


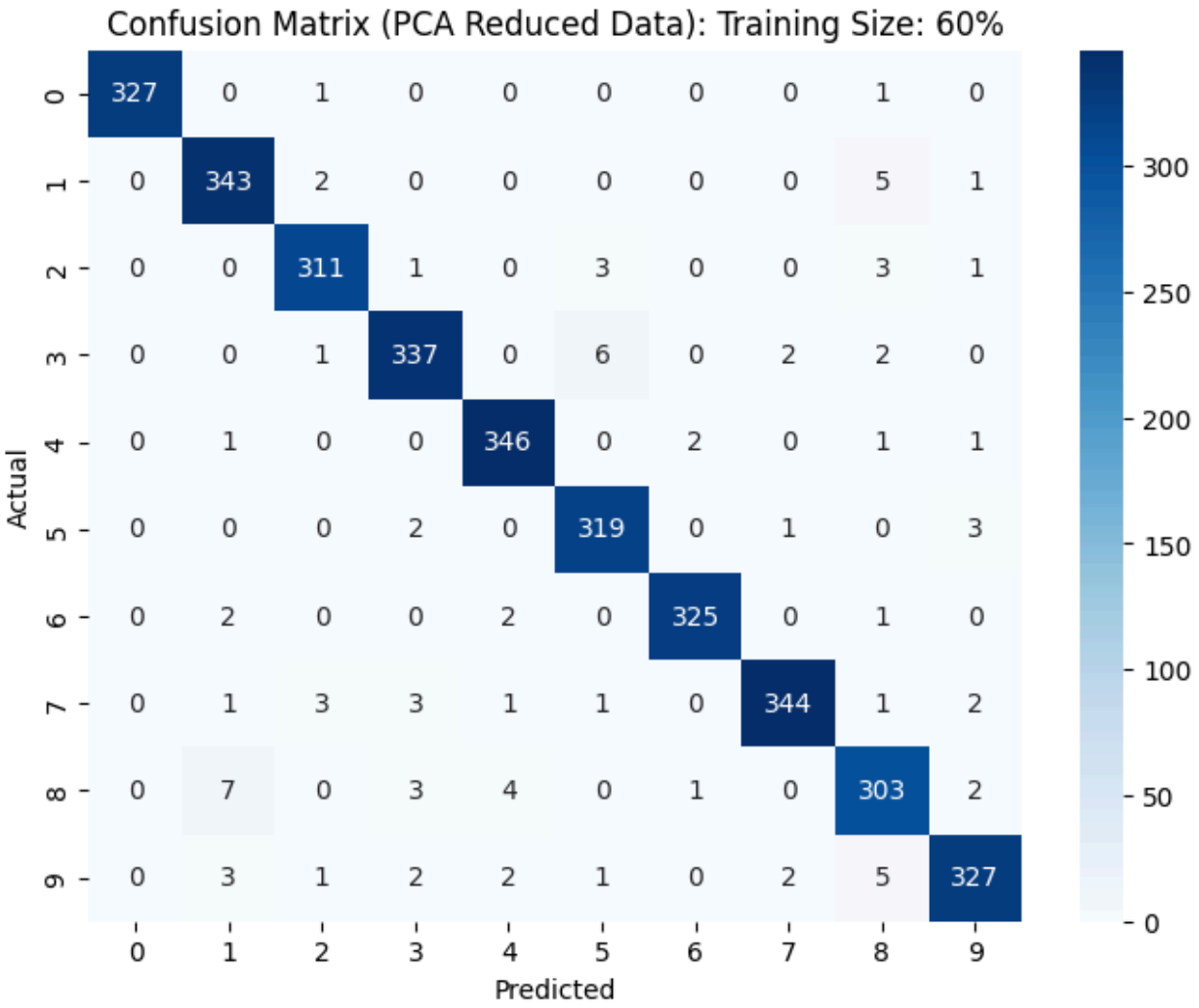


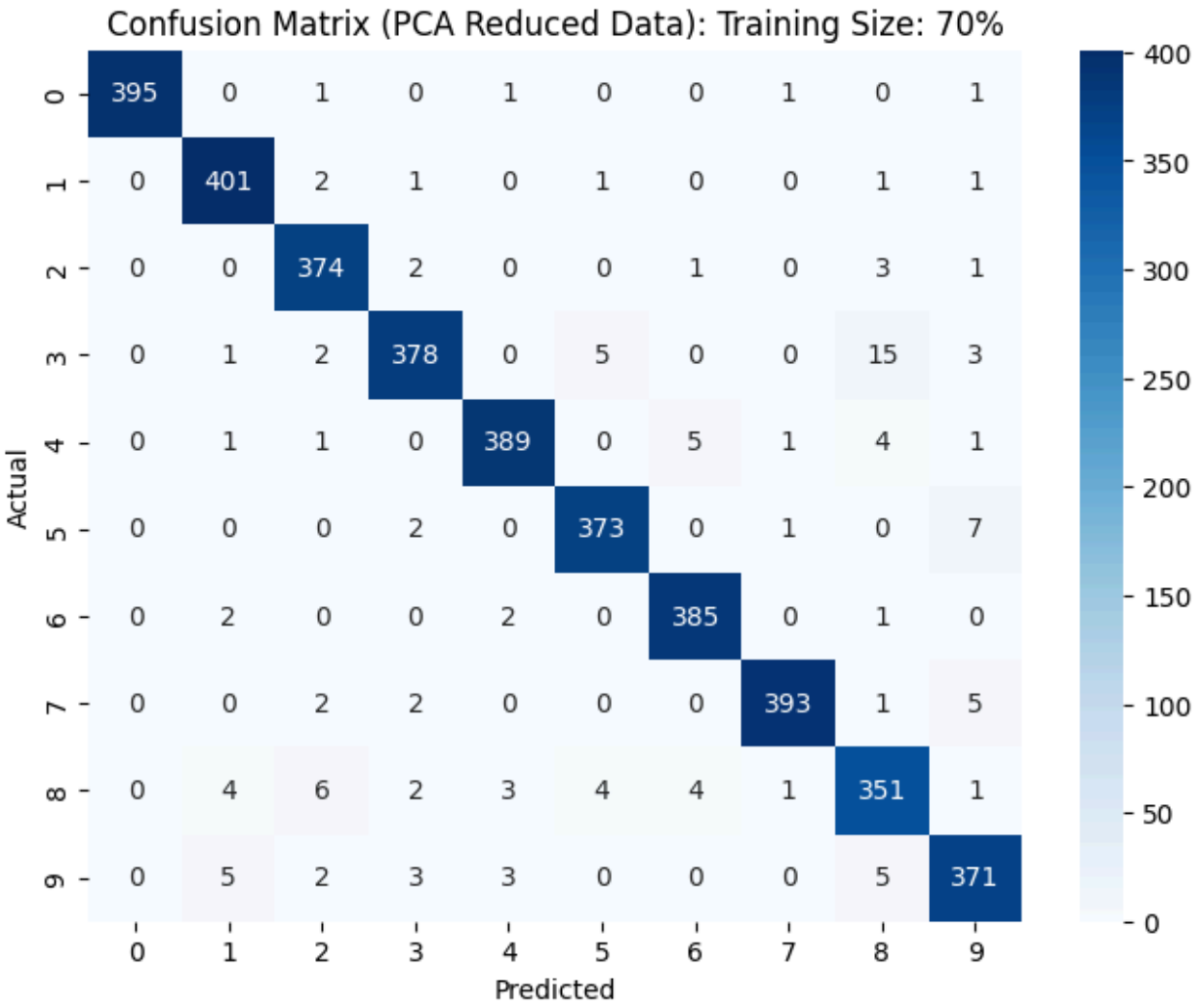
Show Confusion Matrices for MLP with PCA-Reduced Data

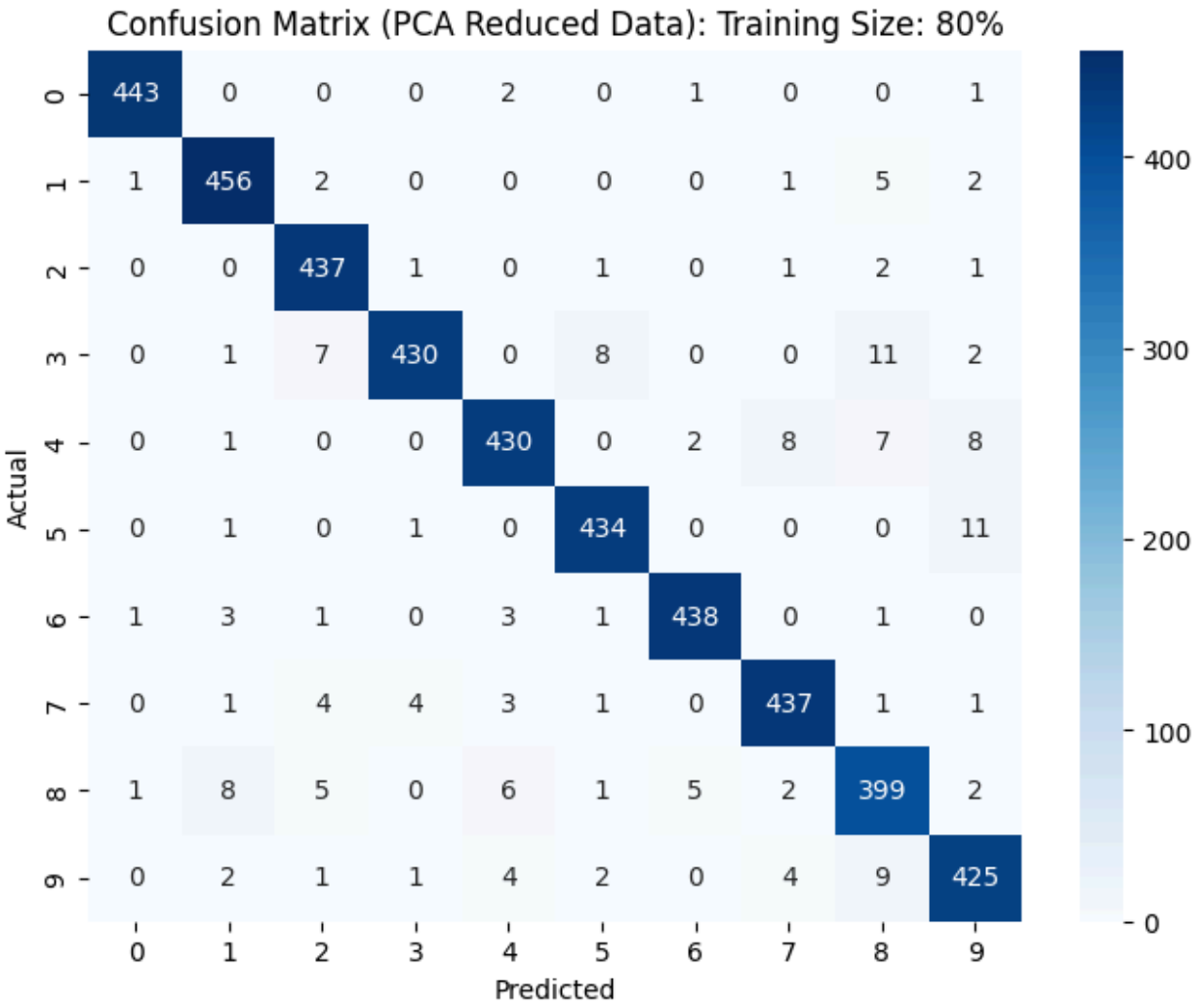
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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_
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```



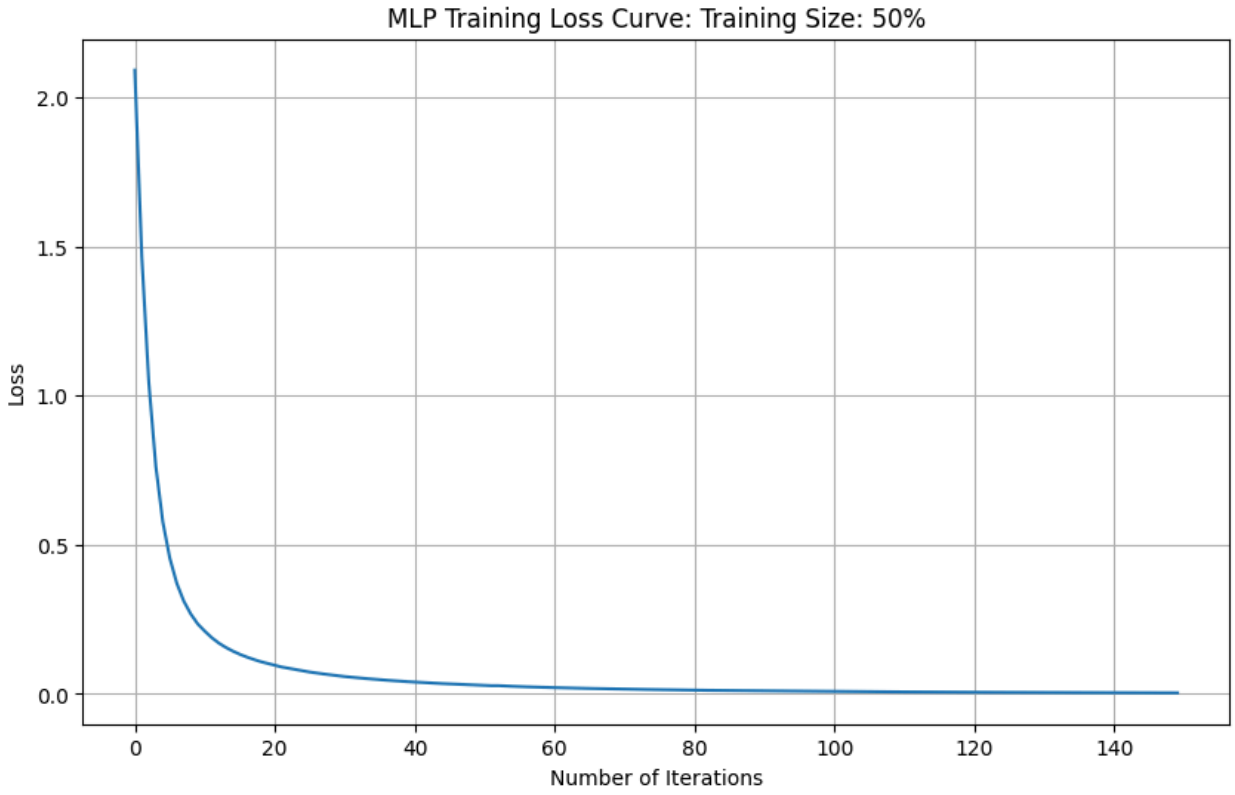

Confusion Matrix (PCA Reduced Data): Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 327 0 1 0 0 0 0 0 1 0
1 0 343 2 0 0 0 0 0 5 1
2 0 0 311 1 0 3 0 0 3 1
3 0 0 1 337 0 6 0 2 2 0
4 0 1 0 0 346 0 2 0 1 1
5 0 0 0 2 0 319 0 1 0 3
6 0 2 0 0 2 0 325 0 1 0
7 0 1 3 3 1 1 0 344 1 2
8 0 7 0 3 4 0 1 0 303 2
9 0 3 1 2 2 1 0 2 5 327
300
250
200
150
100
50
0

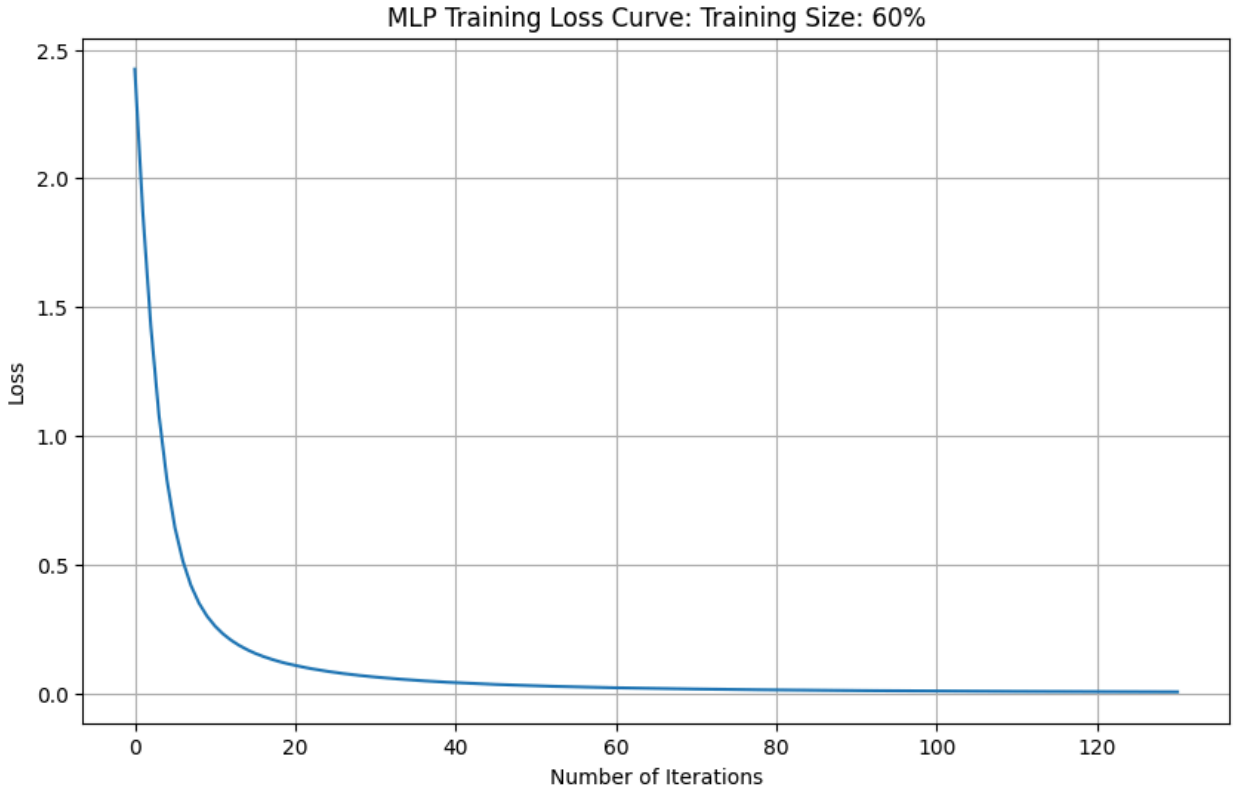

Confusion Matrix (PCA Reduced Data): Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 395 0 1 0 1 0 0 1 0 1
1 0 401 2 1 0 1 0 0 1 1
2 0 0 374 2 0 0 1 0 3 1
3 0 1 2 378 0 5 0 0 15 3
4 0 1 1 0 389 0 5 1 4 1
5 0 0 0 2 0 373 0 1 0 7
6 0 2 0 0 2 0 385 0 1 0
7 0 0 2 2 0 0 0 393 1 5
8 0 4 6 2 3 4 4 1 351 1
9 0 5 2 3 3 0 0 0 5 371
400
350
300
250
200
150
100
50
0

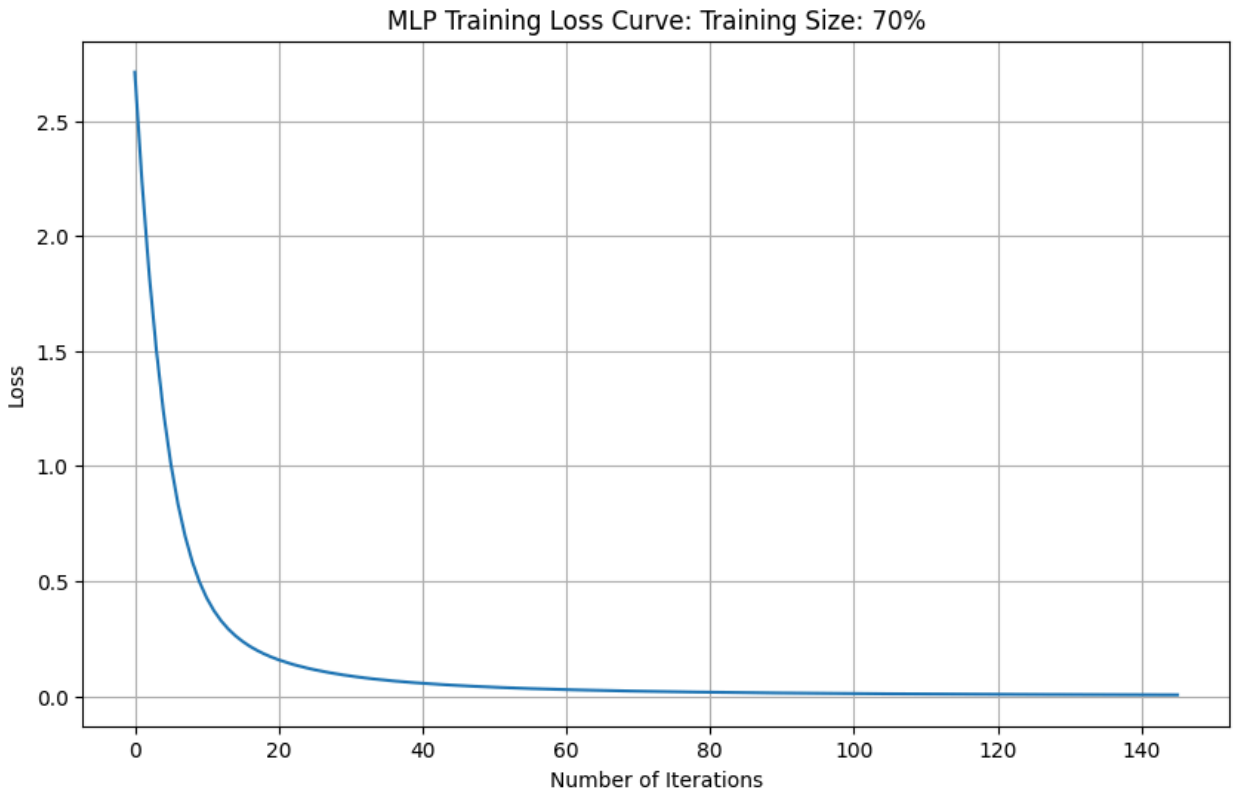


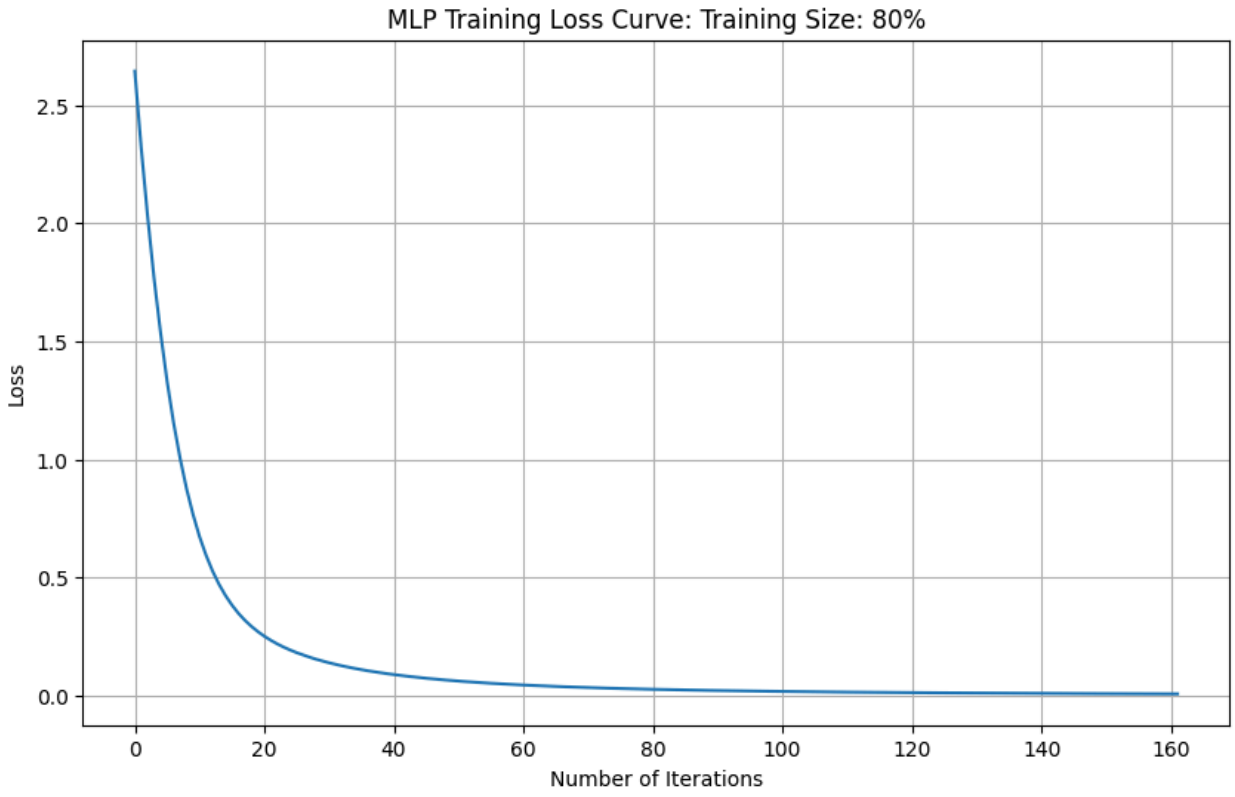
Training Loss generation curve with PCA data

In [ ]:
```python
# 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%
Loss
Number of Iterations
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
120
140


MLP Training Loss Curve: Training Size: 60%
Loss
Number of Iterations
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120


MLP Training Loss Curve: Training Size: 70%
Loss
Number of Iterations
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120
140


MLP Training Loss Curve: Training Size: 80%
Loss
Number of Iterations
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100
120
140
160

===== 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_m
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, ra
  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', ze
  recall_test = recall_score(y_test, y_pred_test, average='weighted') # Calcul
  f1_test = f1_score(y_test, y_pred_test, average='weighted') # Calculate weig

  acc_train = accuracy_score(y_train, y_pred_train)
  precision_train = precision_score(y_train, y_pred_train, average='weighted',
  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
```

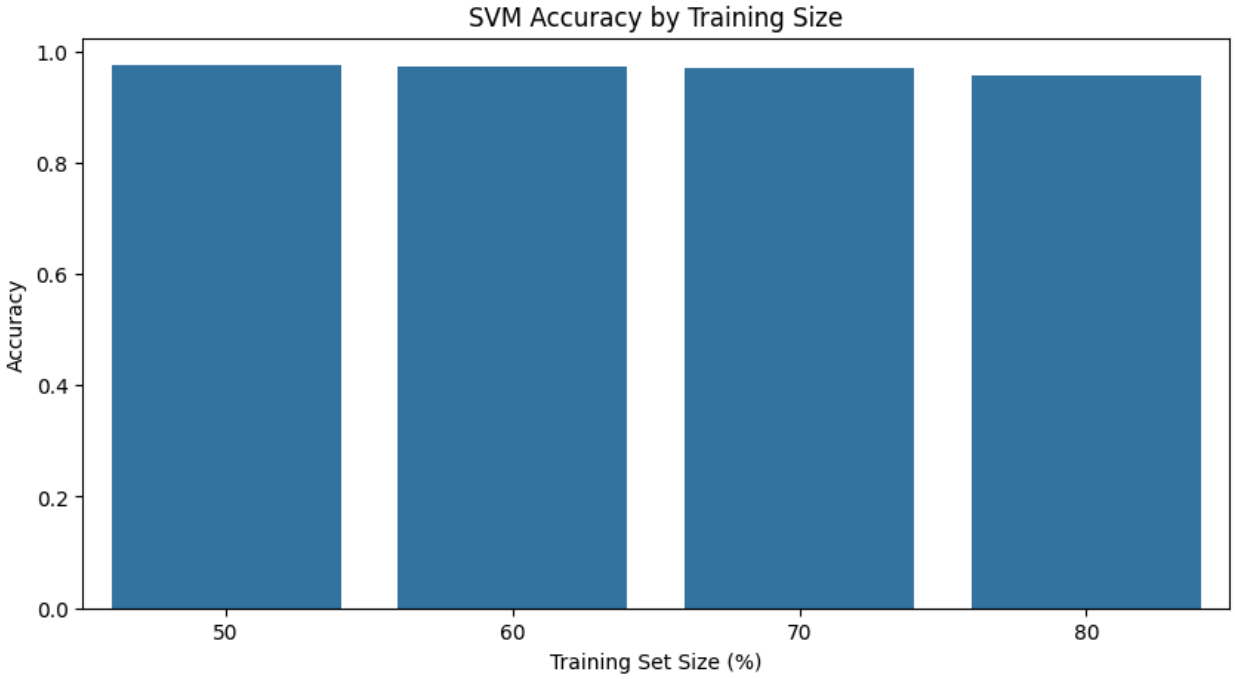
```
  })

  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
  })
```

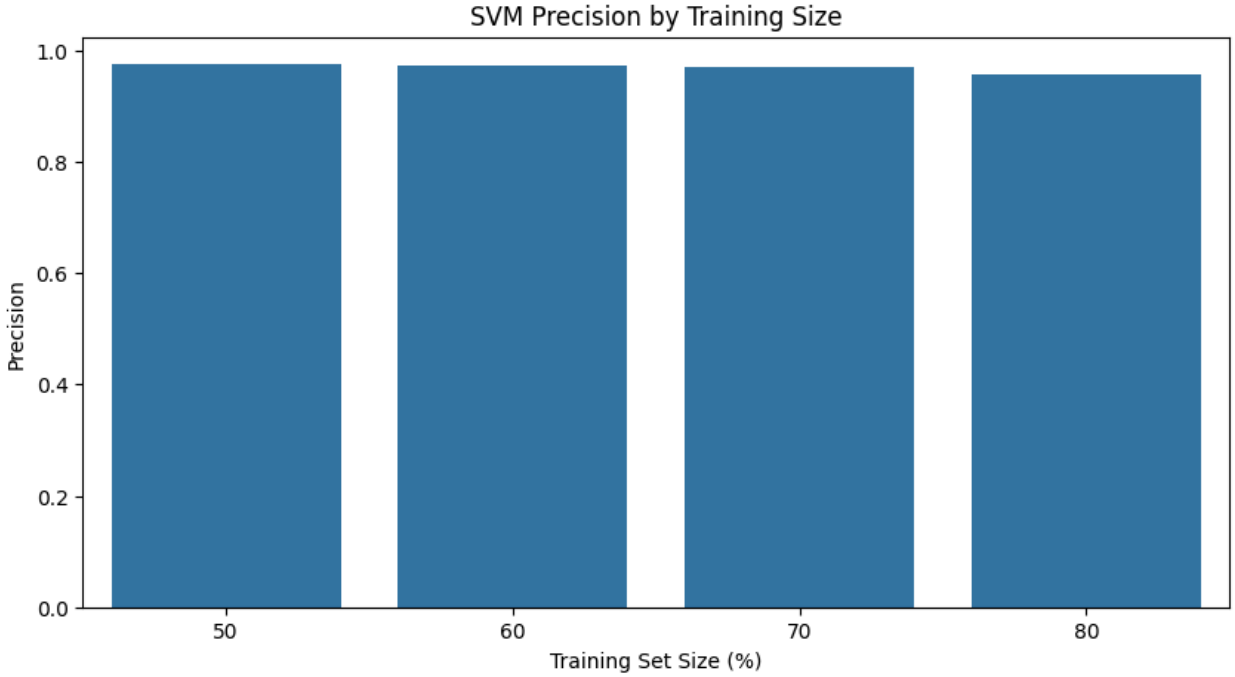
Print Evalution Table and Graphs

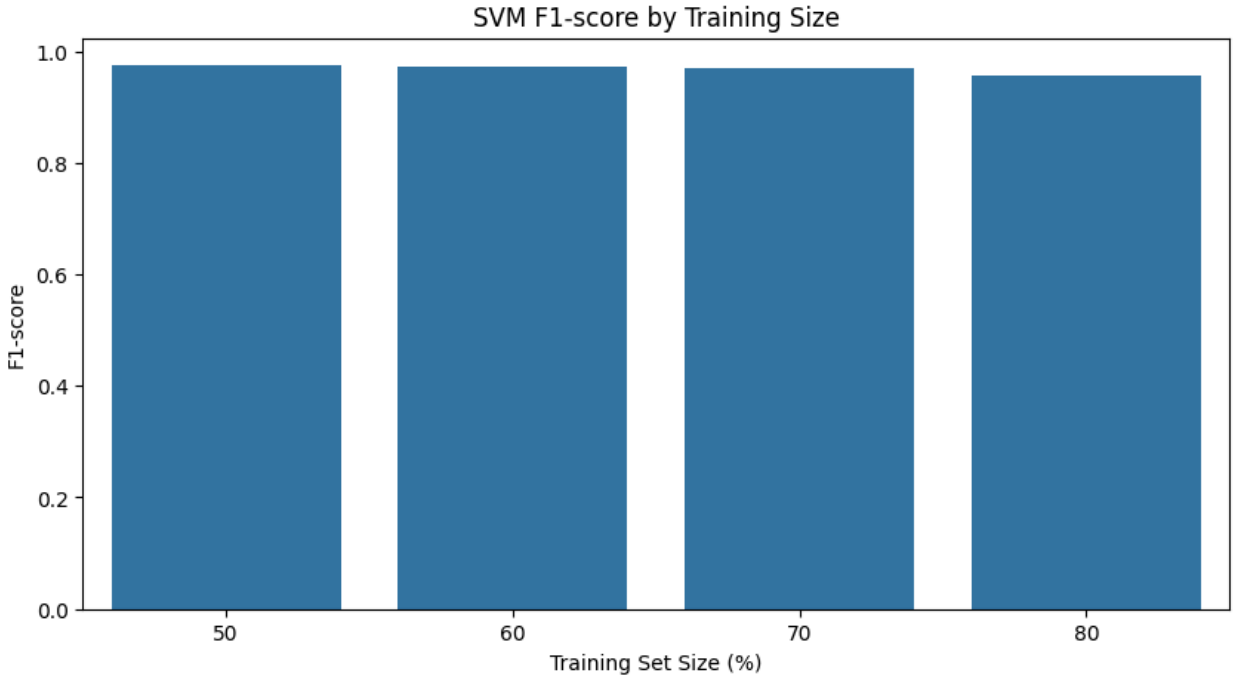
In [ ]:

```
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()
```

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.974733	0.975024	0.974733	0.974802
1	60	0.972716	0.972770	0.972716	0.972664
2	70	0.968480	0.968704	0.968480	0.968424
3	80	0.956628	0.957079	0.956628	0.956624



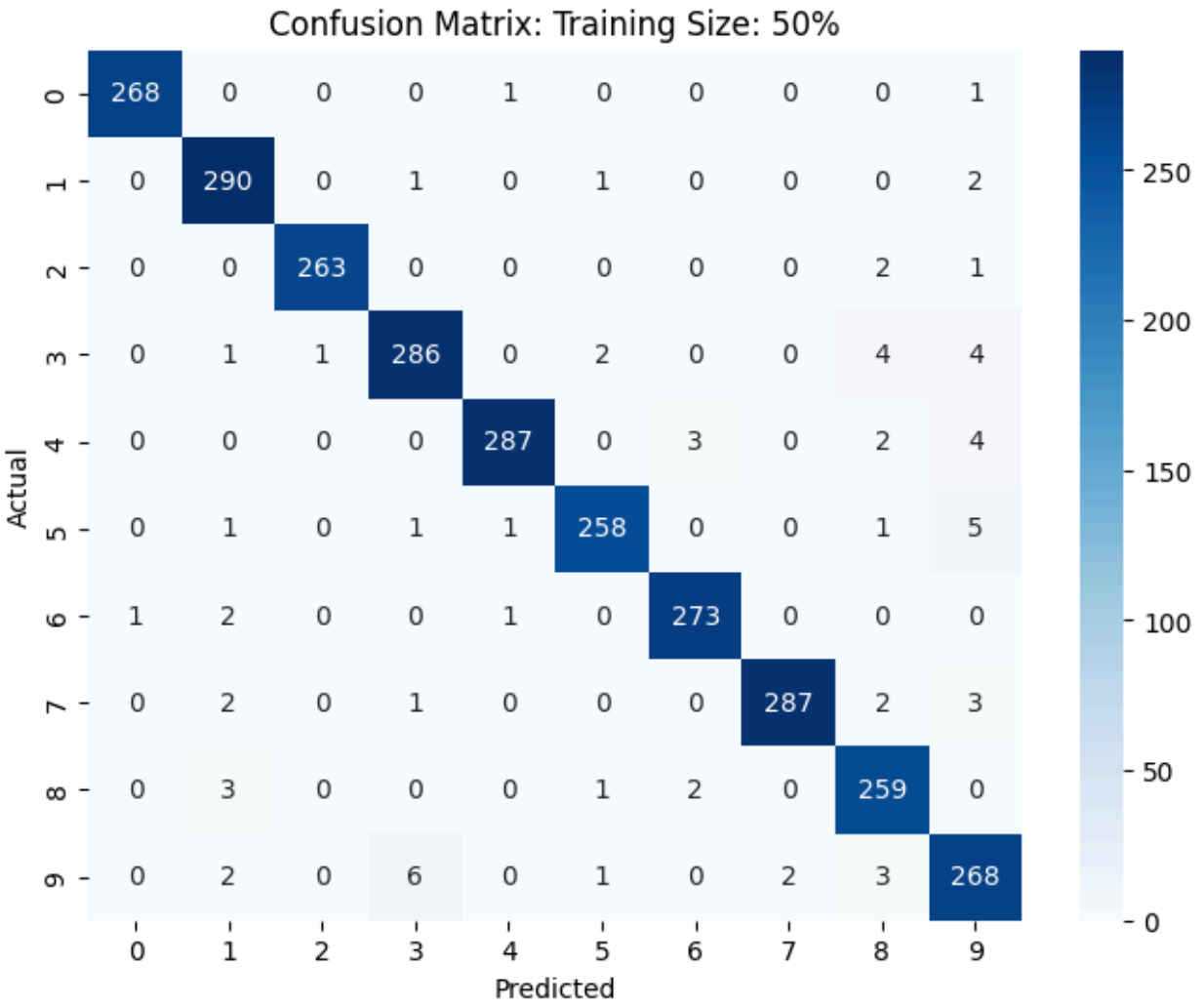


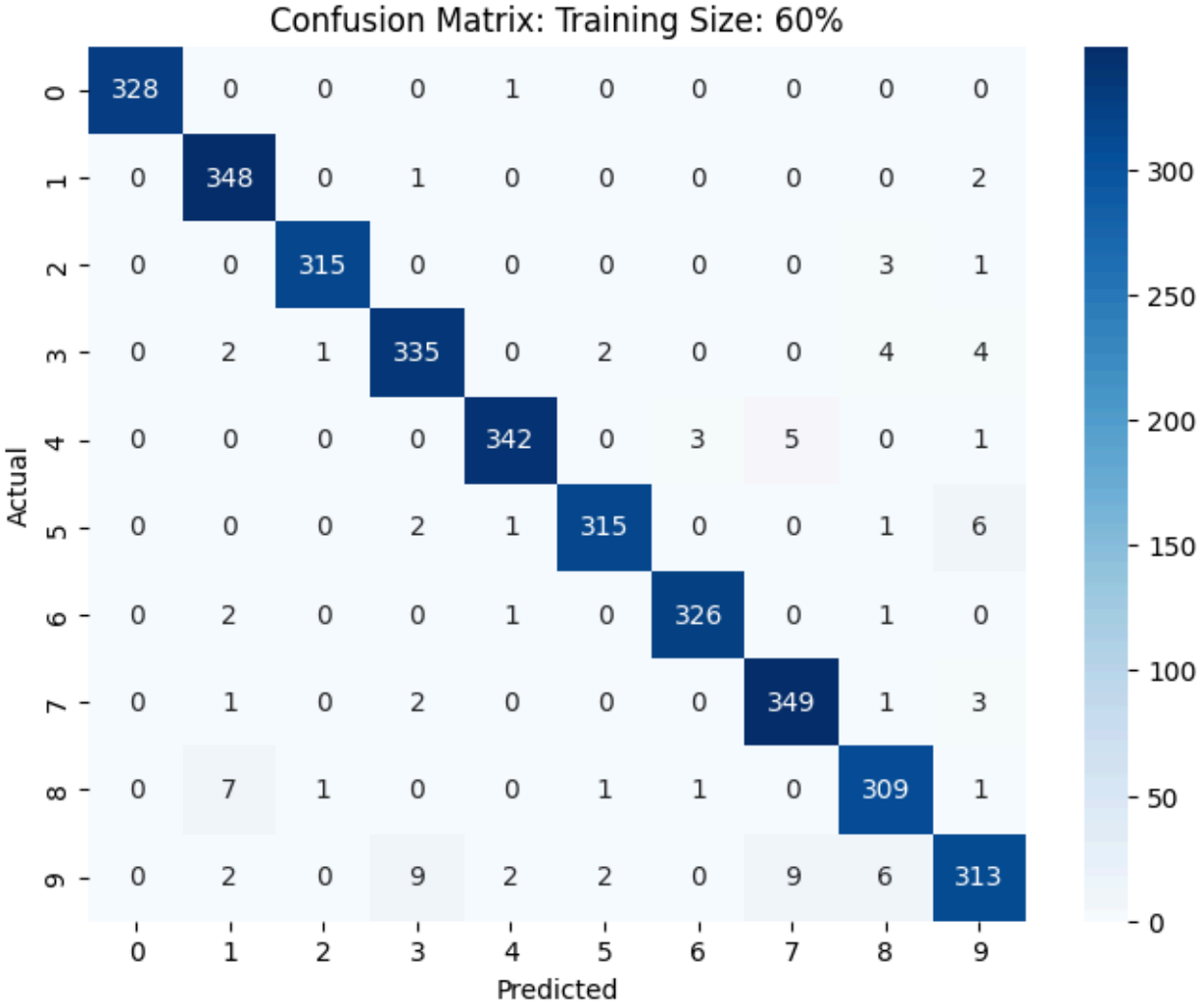


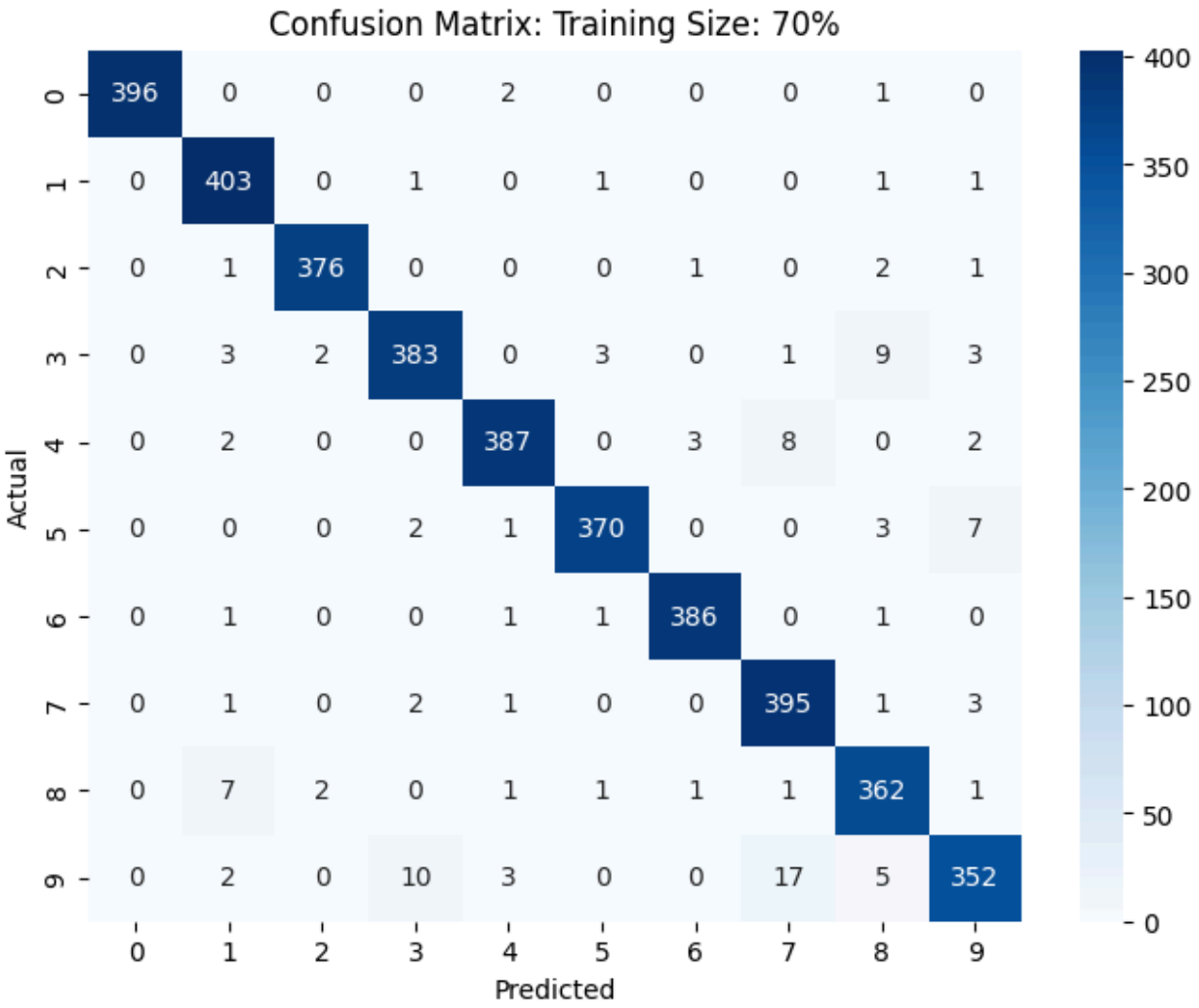
Show Confusion Matrices

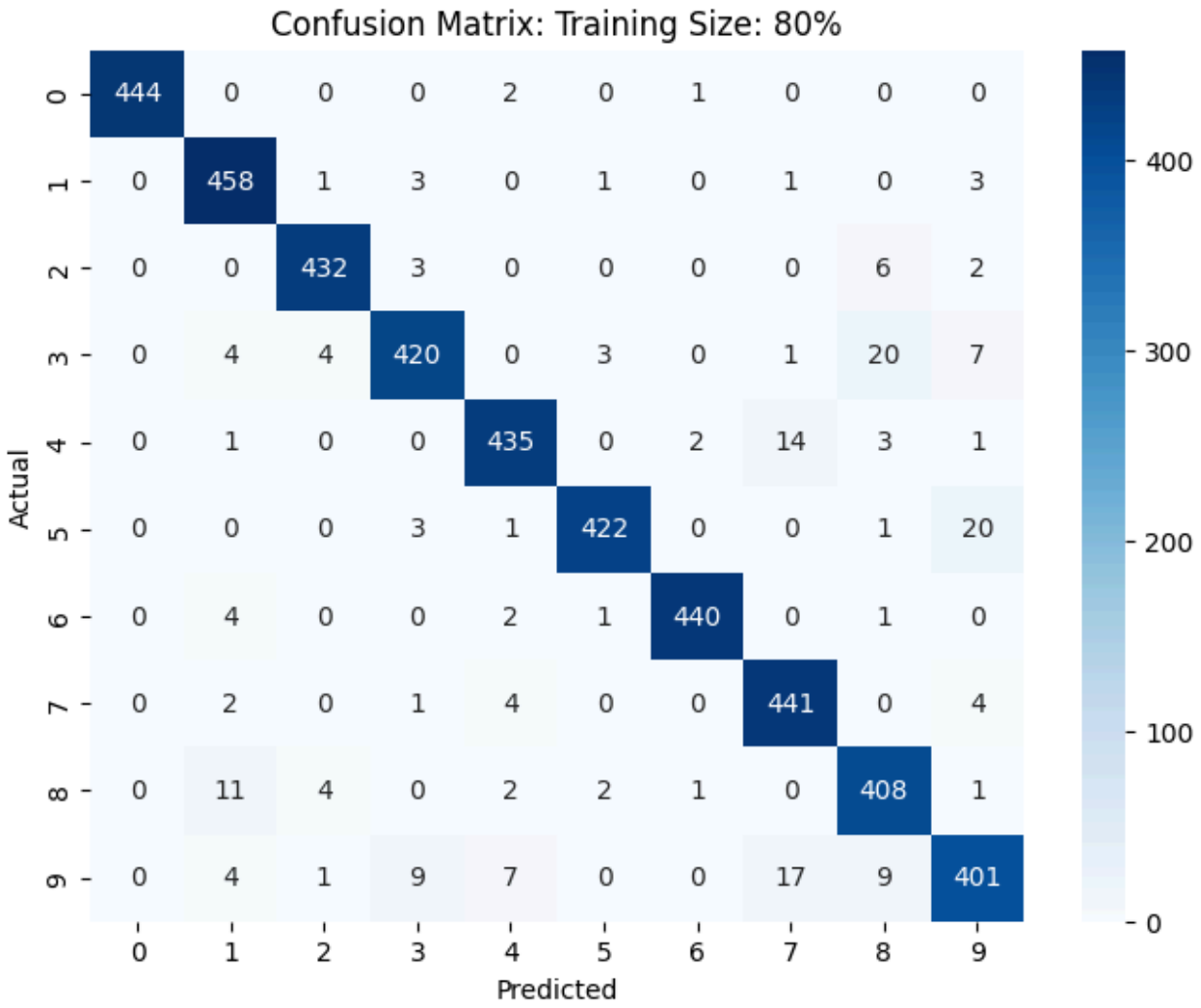
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
328 0 0 0 1 0 0 0 0 0
0 348 0 1 0 0 0 0 0 2
0 0 315 0 0 0 0 0 3 1
0 2 1 335 0 2 0 0 4 4
0 0 0 0 342 0 3 5 0 1
0 0 0 2 1 315 0 0 1 6
0 2 0 0 1 0 326 0 1 0
0 1 0 2 0 0 0 349 1 3
0 7 1 0 0 1 1 0 309 1
0 2 0 9 2 2 0 9 6 313
300
250
200
150
100
50
0


Confusion Matrix: Training Size: 70%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 396 0 0 0 2 0 0 0 1 0
1 0 403 0 1 0 1 0 0 1 1
2 0 1 376 0 0 0 1 0 2 1
3 0 3 2 383 0 3 0 1 9 3
4 0 2 0 0 387 0 3 8 0 2
5 0 0 0 2 1 370 0 0 3 7
6 0 1 0 0 1 1 386 0 1 0
7 0 1 0 2 1 0 0 395 1 3
8 0 7 2 0 1 1 1 1 362 1
9 0 2 0 10 3 0 0 17 5 352
400
350
300
250
200
150
100
50
0



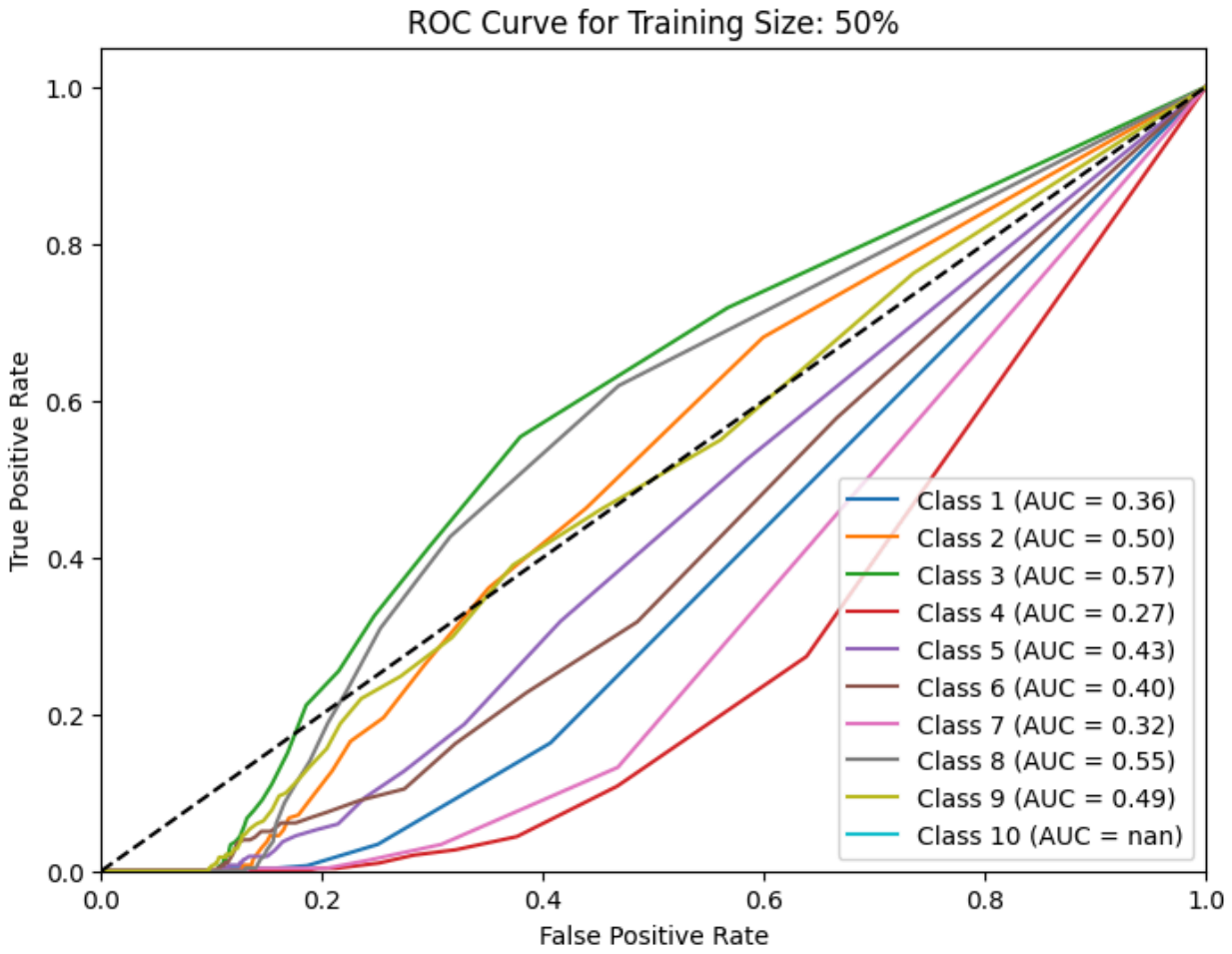
ROC and AOC curve

In [ ]:

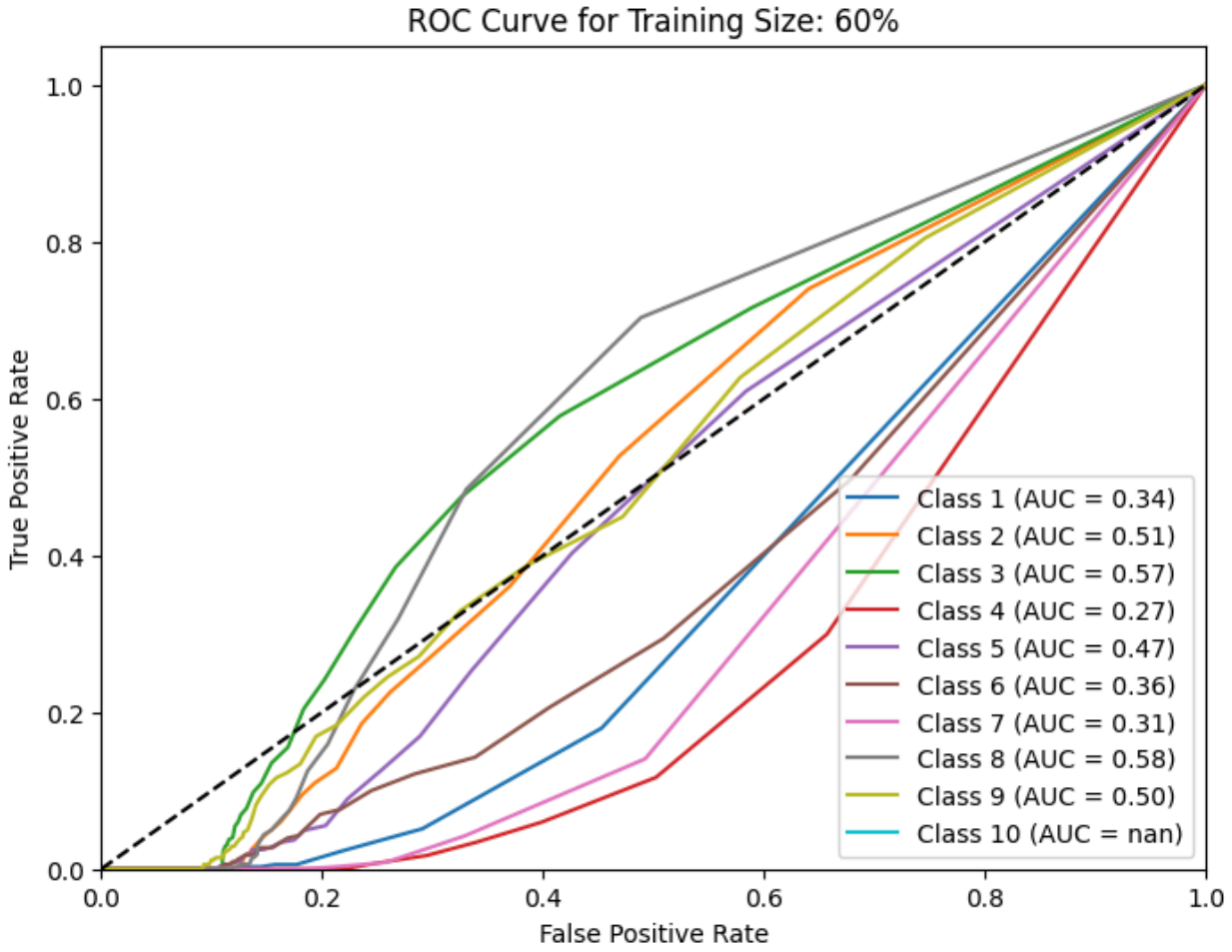
```
from sklearn.metrics import roc_curve, auc

# 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 fo
    plt.legend(loc="lower right")
    plt.show()
```

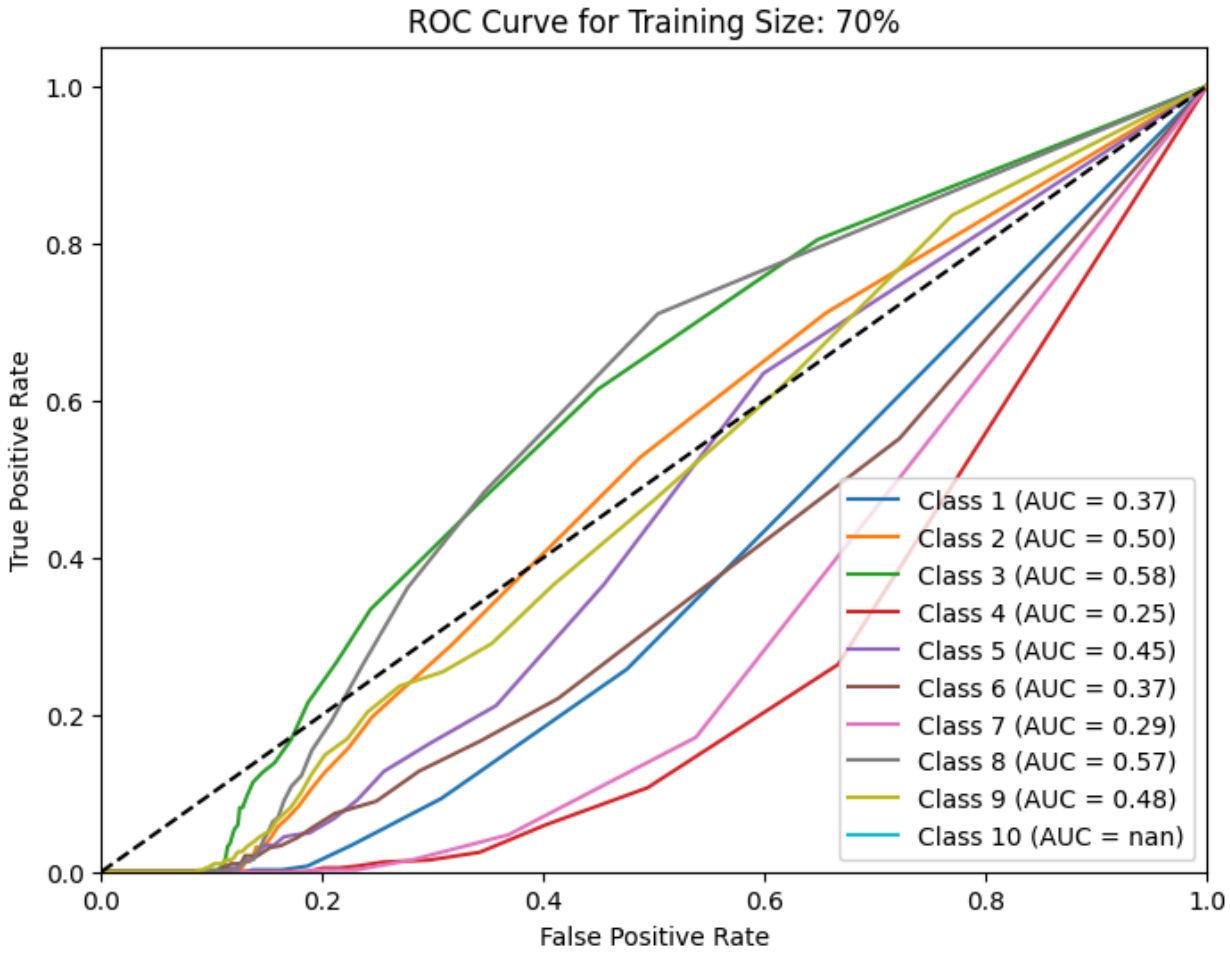
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```



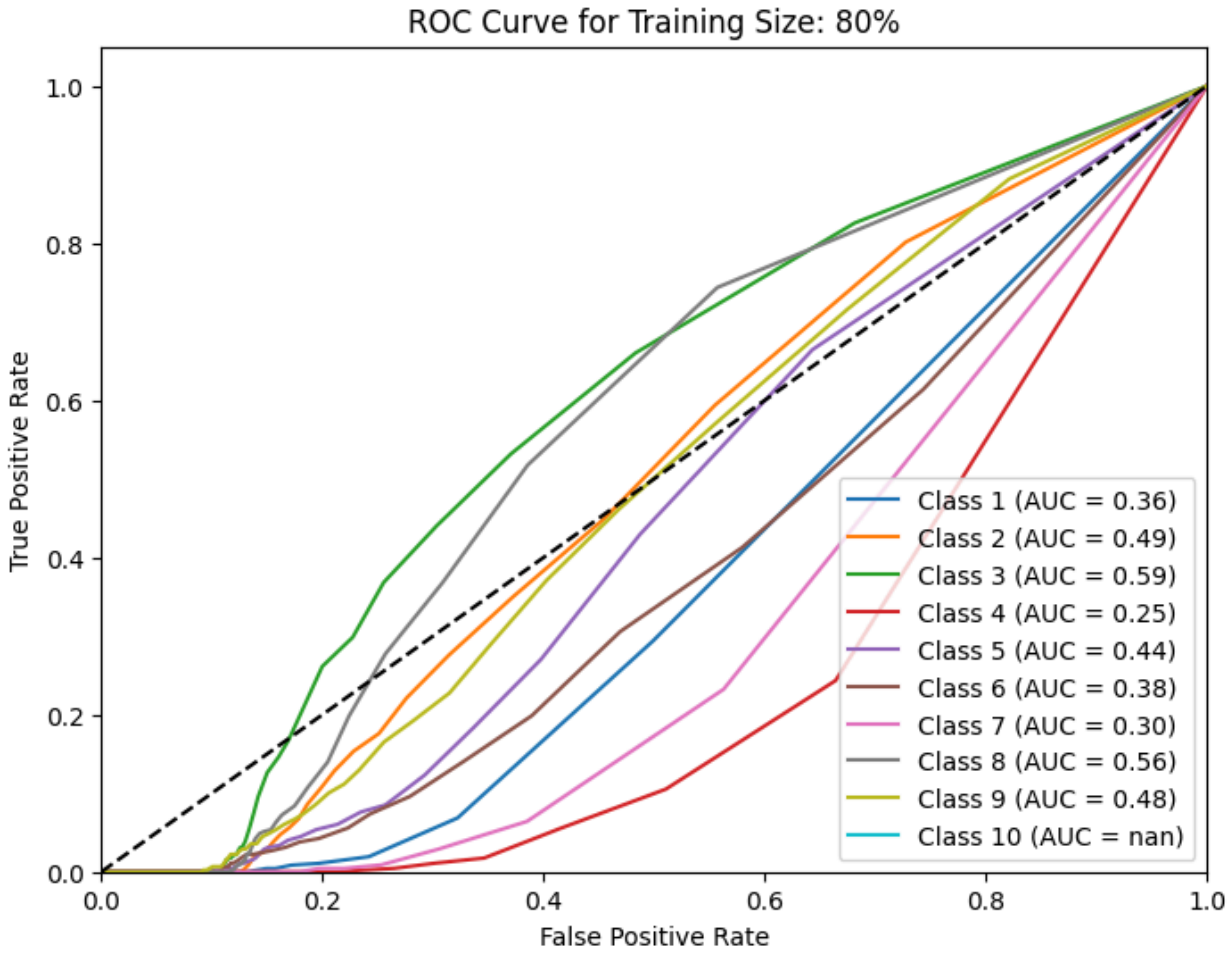
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: No positive samples in y_true, true positive value should be
meaningless
  warnings.warn(
```




```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1188: Undef
inedMetricWarning: 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: Undef
inedMetricWarning: 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:
```

	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

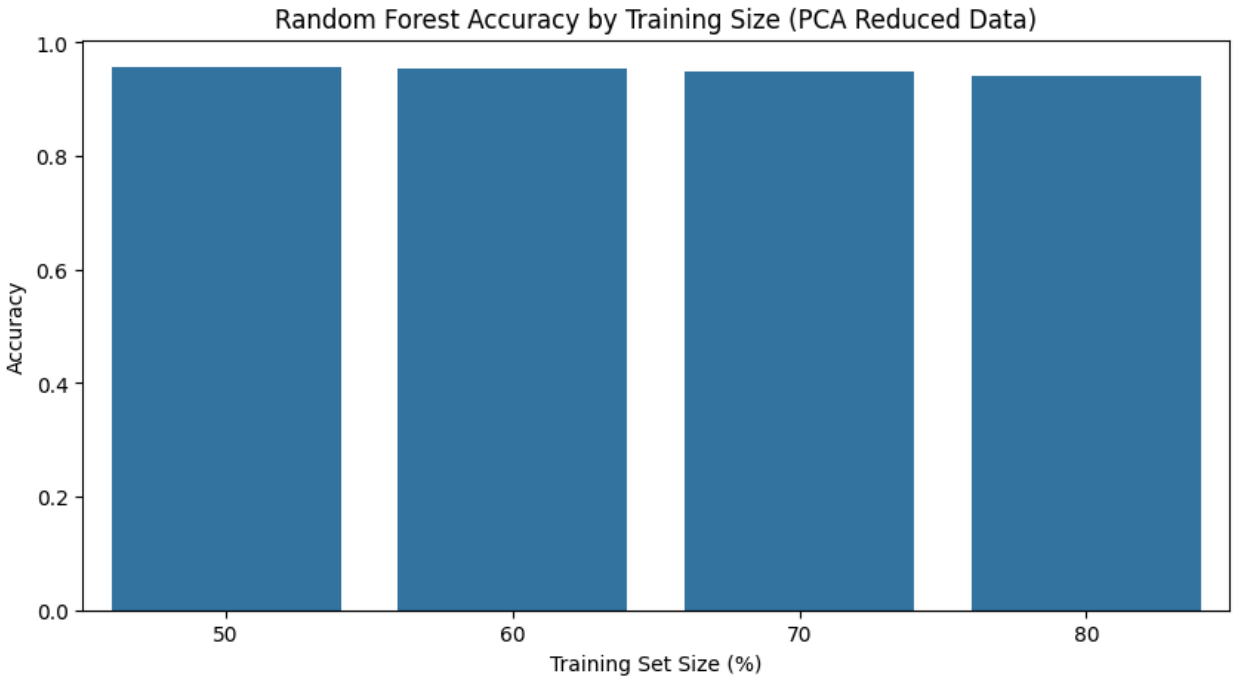
Calculate Random Forest performance with PCA-Reduced Data for different test sizes

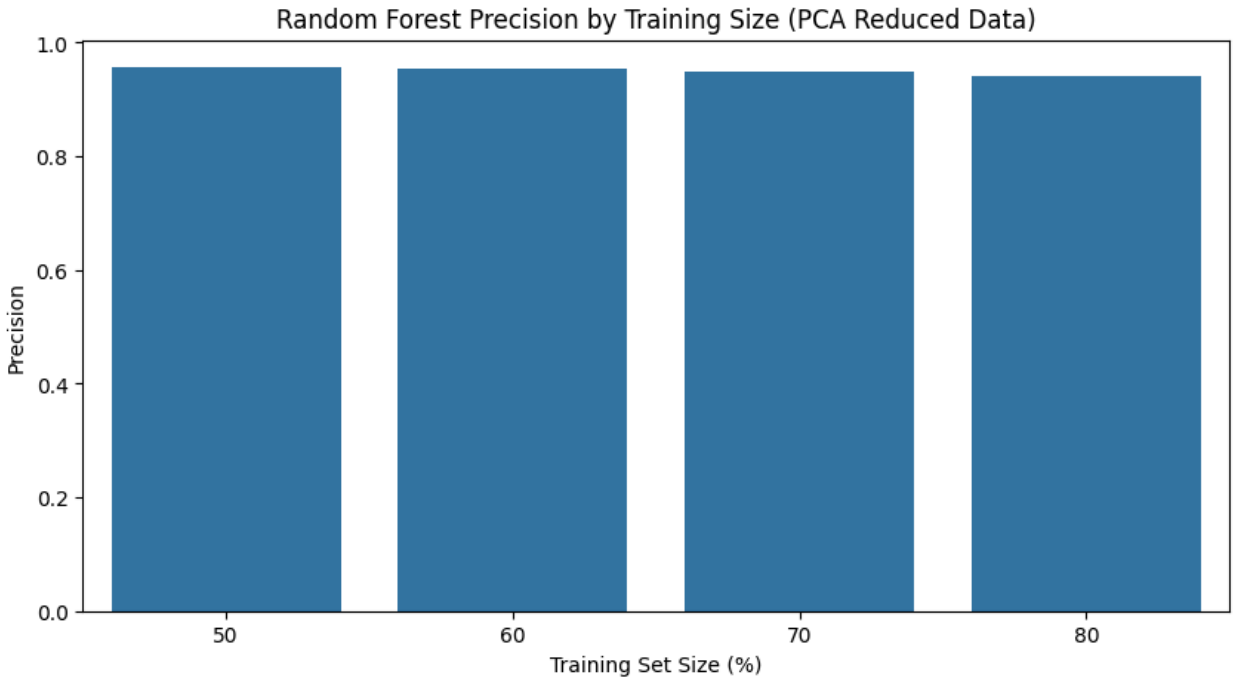
In [ ]:
```
results_rf_pca = []
confusion_matrices_rf_pca = []


for size in training_sizes:
  X_train_pca_rf, X_test_pca_rf, y_train, y_test = train_test_split(X_pca_redu
  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
  })
```

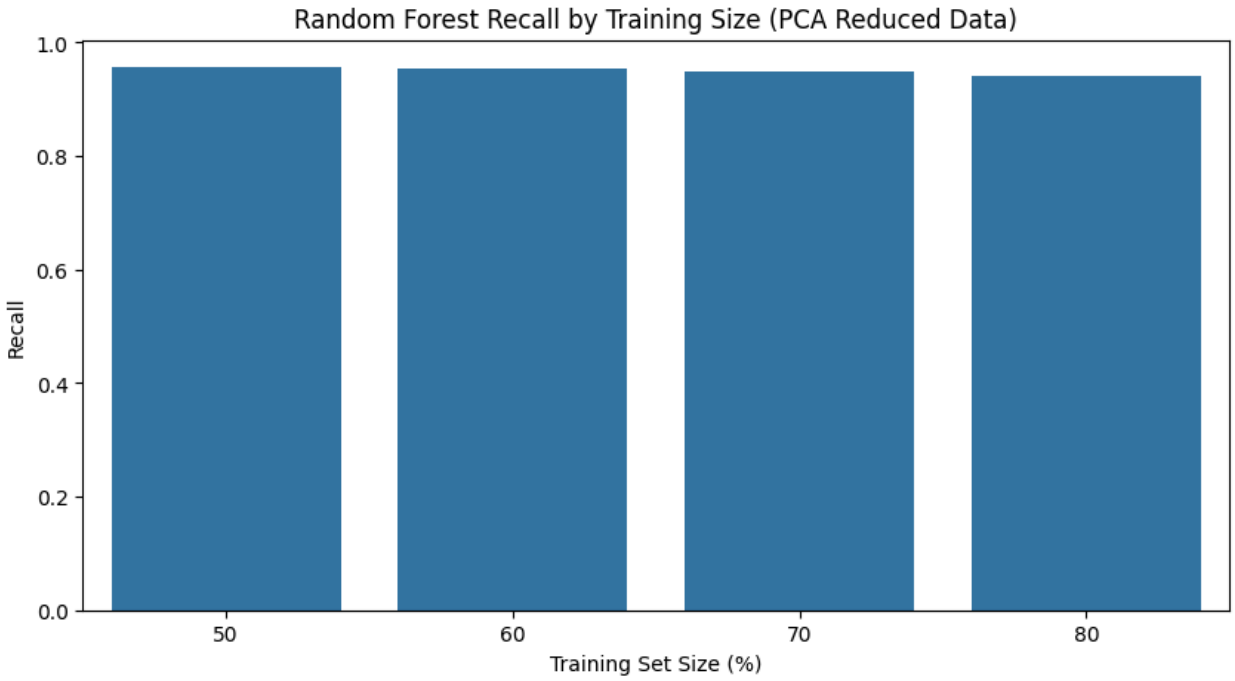
Print Random Forest Performance Table and Graphs with PCA-Reduced Data

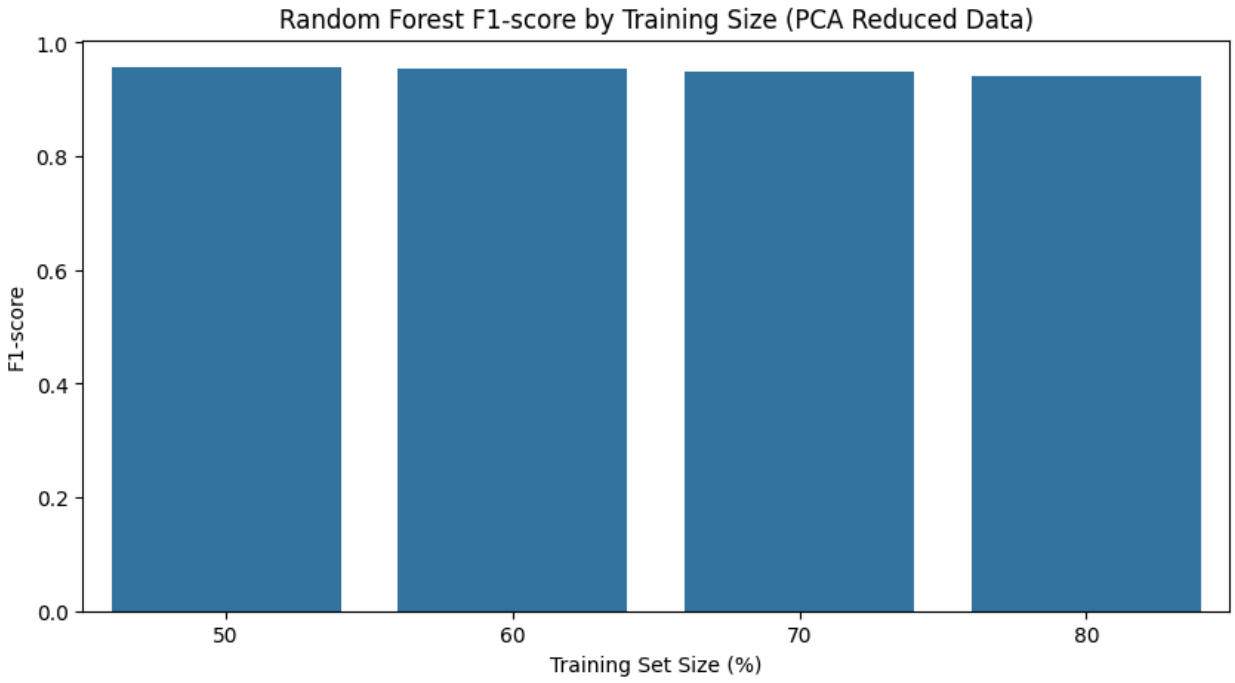
In [ ]:
```
df_rf_pca = pd.DataFrame(results_rf_pca)
display(df_rf_pca)

# 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()
```

	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


Random Forest Accuracy by Training Size (PCA Reduced Data)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)


Random Forest Precision by Training Size (PCA Reduced Data)
Precision
1.0
0.8
0.6
0.4
0.2
0.0
50
60
70
80
Training Set Size (%)

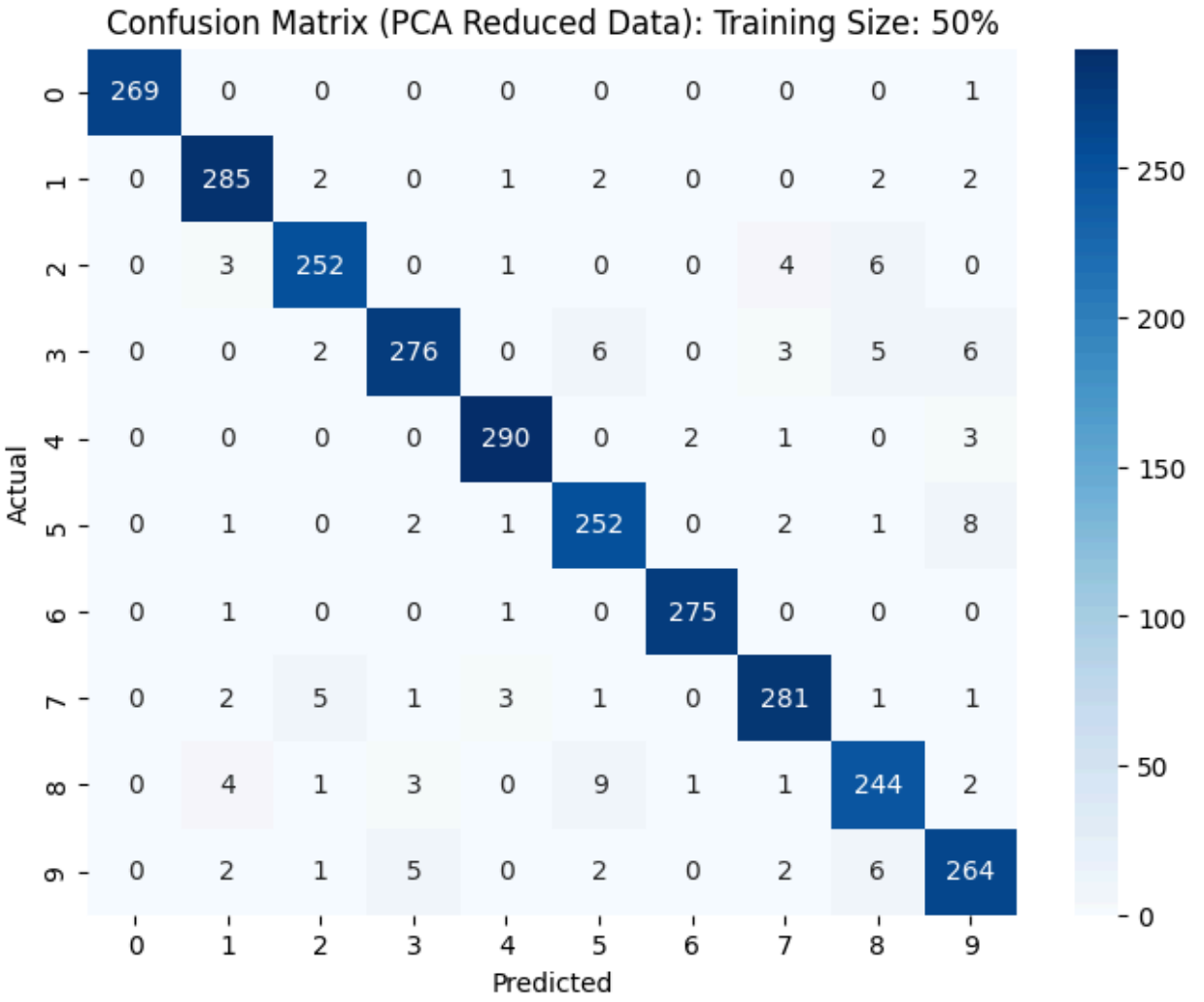


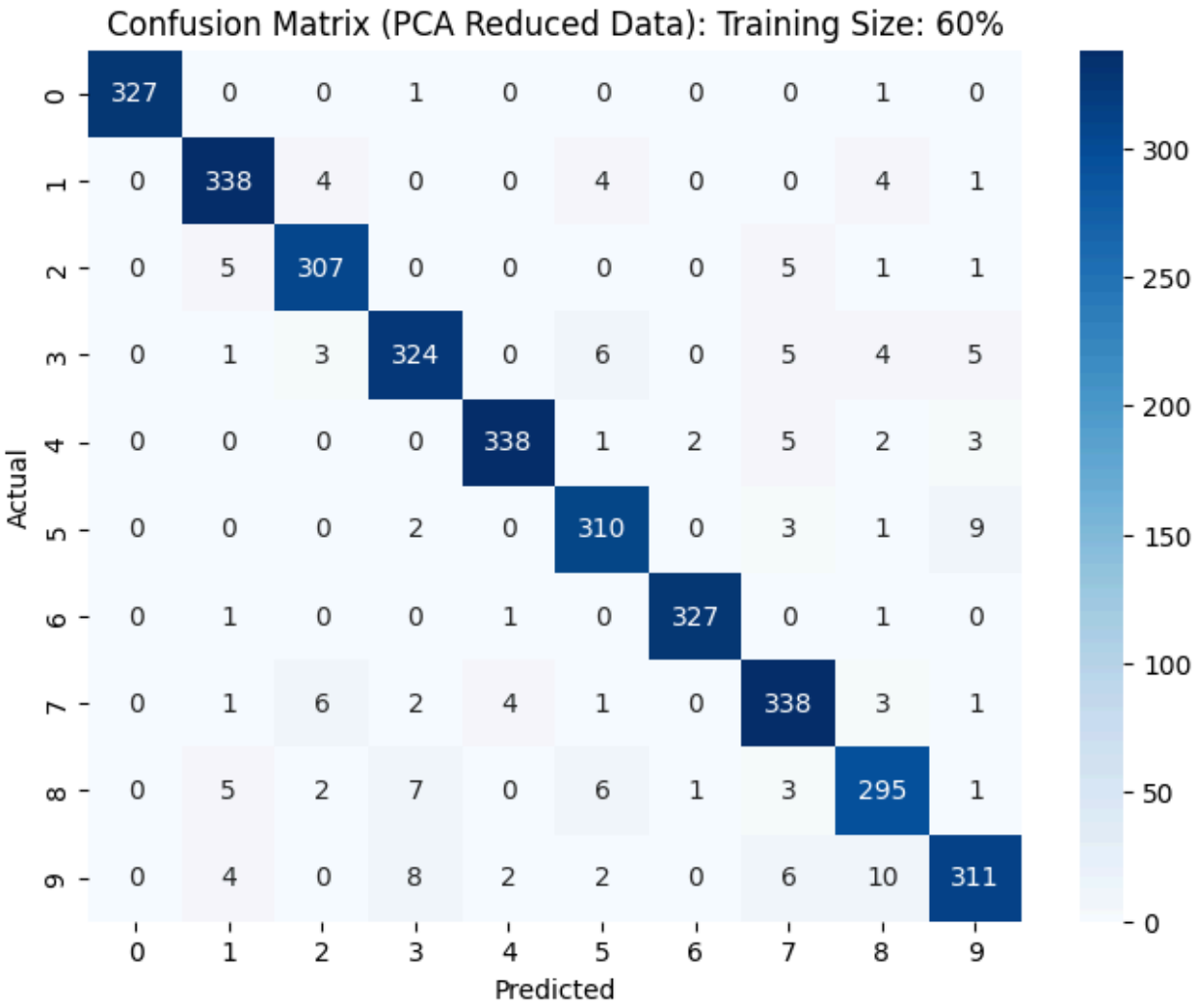


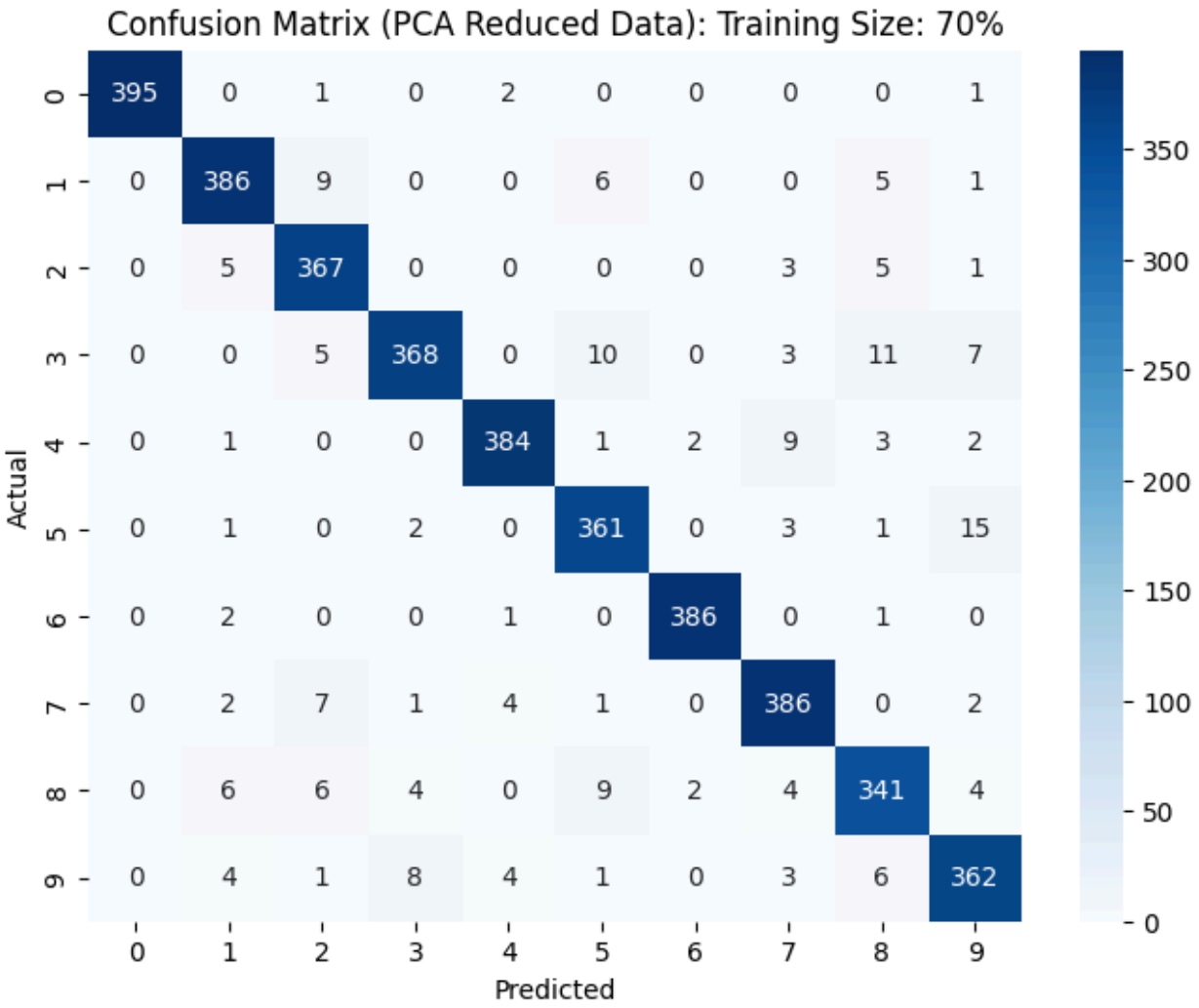
Show Confusion Matrices for Random Forest with PCA-Reduced Data

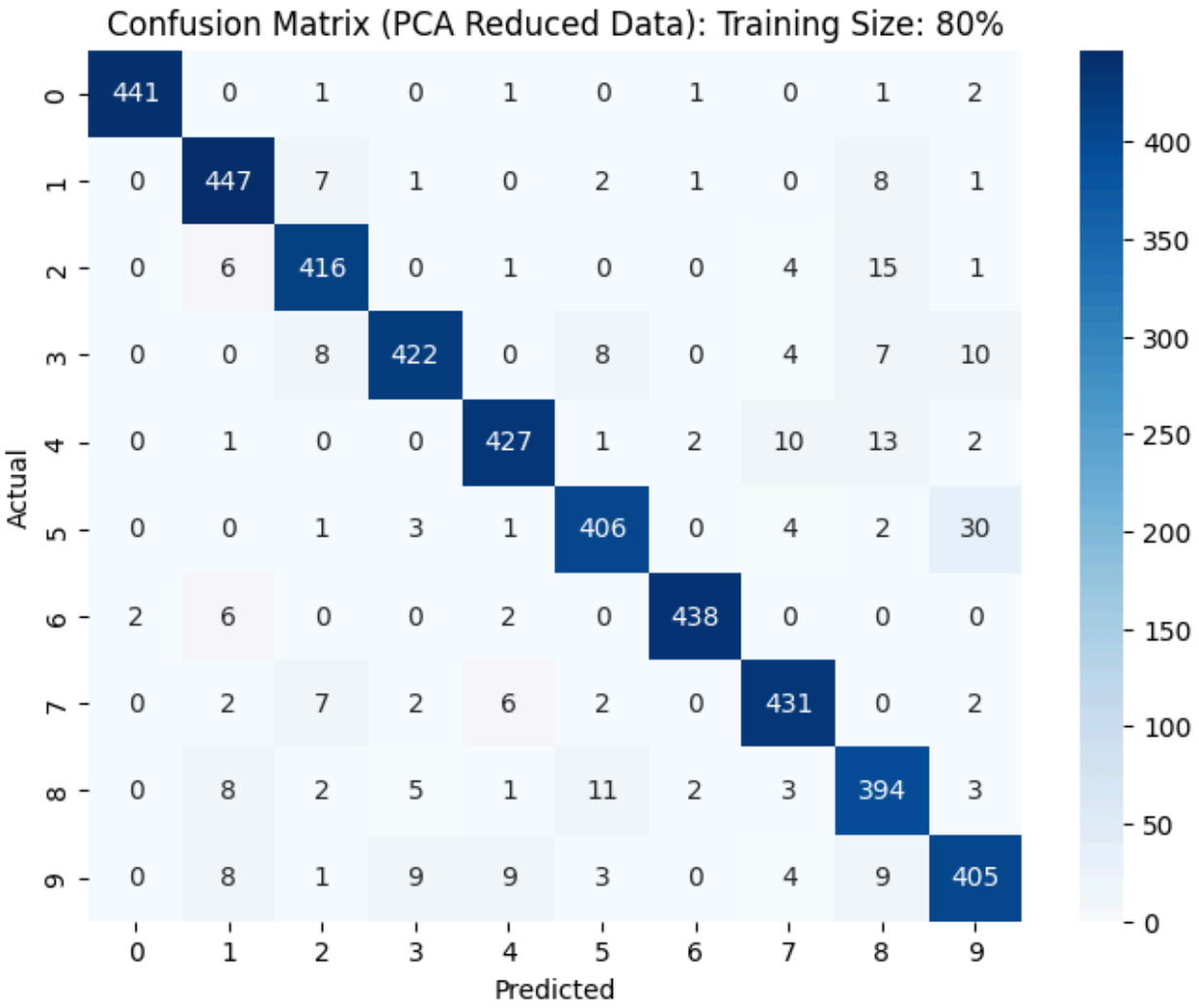
In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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_
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```




Confusion Matrix (PCA Reduced Data): Training Size: 60%
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 327 0 0 1 0 0 0 0 1 0
1 0 338 4 0 0 4 0 0 4 1
2 0 5 307 0 0 0 0 5 1 1
3 0 1 3 324 0 6 0 5 4 5
4 0 0 0 0 338 1 2 5 2 3
5 0 0 0 2 0 310 0 3 1 9
6 0 1 0 0 1 0 327 0 1 0
7 0 1 6 2 4 1 0 338 3 1
8 0 5 2 7 0 6 1 3 295 1
9 0 4 0 8 2 2 0 6 10 311
300
250
200
150
100
50
0


Confusion Matrix (PCA Reduced Data): Training Size: 70%
Actual
Predicted
0 395 0 1 0 2 0 0 0 0 1
1 0 386 9 0 0 6 0 0 5 1
2 0 5 367 0 0 0 0 3 5 1
3 0 0 5 368 0 10 0 3 11 7
4 0 1 0 0 384 1 2 9 3 2
5 0 1 0 2 0 361 0 3 1 15
6 0 2 0 0 1 0 386 0 1 0
7 0 2 7 1 4 1 0 386 0 2
8 0 6 6 4 0 9 2 4 341 4
9 0 4 1 8 4 1 0 3 6 362
0 1 2 3 4 5 6 7 8 9
350
300
250
200
150
100
50
0



Machine Learning Model Performance Report

Handwritten Digit Dataset Evaluation

1. Support Vector Machine (SVM)

This section evaluates the performance of SVM models on the Handwritten Digit dataset, focusing on how different kernels, training sizes, and dimensionality reduction through PCA affect the outcomes.

Influence of Training Size and Kernel Choice:

SVM models were tested with multiple kernel functions (linear, polynomial, gaussian, and sigmoid) across varying training proportions (50%, 60%, 70%, and

80%) on the Digit dataset.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for each kernel on the Digit Dataset here]

Results Overview:

[Insert table of SVM results on Original Digit Dataset here]

For this dataset, the polynomial and gaussian kernels consistently achieved superior performance compared to the linear and sigmoid kernels. Their metrics remained strong and stable across different training sizes.

Effect of PCA Dimensionality Reduction:

PCA was applied to the Digit dataset to reduce its high-dimensional feature space before training SVM models.

[Insert table of SVM results on PCA-Reduced Digit Dataset here]

The application of PCA led to noticeable improvements, particularly with the gaussian kernel. Accuracy and related measures increased, indicating that PCA effectively simplified the feature space while preserving key information.

Confusion Matrices:

[Insert Confusion Matrices for SVM on Original Data here]

[Insert Confusion Matrices for SVM on PCA-Reduced Data here]

The confusion matrices highlight that PCA helped SVM models, especially those using the gaussian kernel, to make more accurate predictions across different digits.

ROC and AUC Curves:

[Insert ROC and AUC curves for SVM on Original Data here]

The ROC and AUC analyses reinforce the stronger performance of polynomial and gaussian kernels, as well as the performance boost after PCA reduction.

---

2. Multilayer Perceptron (MLP)

This section examines the performance of MLP models on the Handwritten Digit dataset, including training size effects, training loss progression, and PCA's influence.

Impact of Training Size:

The MLP model was trained on different portions of the dataset (50%, 60%, 70%, and 80%) to assess the effect of data availability.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for MLP on Digit Dataset here]

Results Overview:

[Insert table of MLP results on Original Digit Dataset here]

Across all training sizes, the MLP delivered high accuracy and strong metrics, showing consistent and reliable performance.

Training Loss Behavior:

[Insert MLP Training Loss Curves for Digit Dataset here]

The training loss curves for the MLP reveal a steady decline, reflecting effective learning and convergence during training.

Effect of PCA Dimensionality Reduction:

MLP was also trained on PCA-transformed features for comparison.

[Insert table of MLP results on PCA-Reduced Digit Dataset here]

PCA enhanced the performance of the MLP, similar to what was observed in other datasets. By lowering dimensionality, the model managed the data complexity more effectively, leading to improved accuracy.

Confusion Matrices:

[Insert Confusion Matrices for MLP on Original Data here]

[Insert Confusion Matrices for MLP on PCA-Reduced Data here]

The confusion matrices confirm that PCA reduced misclassification errors, making predictions more precise.

ROC and AUC Curves:

[Insert ROC and AUC curves for MLP on Original Data here]

On PCA-transformed data, ROC and AUC results show a stronger ability to discriminate between different classes.

3. Random Forest

This section explores the Random Forest model's performance on the Handwritten Digit dataset, both with original features and after applying PCA.

Impact of Training Size:

Random Forest models were trained using subsets of 50%, 60%, 70%, and 80% of the dataset to compare performance stability.

[Insert Bar plots for Accuracy, Precision, Recall, and F1-score vs. Training Size for Random Forest on Digit Dataset here]

Results Overview:

[Insert table of Random Forest results on Original Digit Dataset here]

The Random Forest model consistently produced high-quality predictions on the original data, with little variation across training sizes.

Effect of PCA Dimensionality Reduction:

Random Forest was also trained on PCA-reduced data.

[Insert table of Random Forest results on PCA-Reduced Digit Dataset here]

Unlike SVM and MLP, Random Forest experienced a slight drop in performance when using PCA features. This suggests that the original feature set contained richer discriminatory signals for tree-based methods.

Confusion Matrices:

[Insert Confusion Matrices for Random Forest on Original Data here]

[Insert Confusion Matrices for Random Forest on PCA-Reduced Data here]

The confusion matrices show fewer classification errors when the model used the original dataset compared to PCA-transformed features.

ROC and AUC Curves:

[Insert ROC and AUC curves for Random Forest on Original Data here]

The ROC and AUC plots indicate better performance for the original dataset, with PCA leading to a slight reduction in discriminatory power.

NAME : DHANANJOY SHAW
SECTION : IT A2
ROLL NUMBER : 002211001086
SUBJECT : ML LAB
GITHUB: Assignment2

DOCUMENTATION

Wine Dataset

1. Support Vector Machine (SVM)

This section evaluates how SVM models perform on the Wine dataset, focusing on the influence of kernel choice, training set size, and dimensionality reduction using PCA.

Effect of Training Size and Kernel Selection:

SVM models were trained using four kernel functions—linear, polynomial, gaussian, and sigmoid—on different proportions of the dataset (50%, 60%, 70%, and 80%). Performance was assessed using Accuracy, Precision, Recall, and F1-score.

	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

Across all training sizes, the linear kernel consistently produced the strongest results. Both polynomial and gaussian kernels delivered fluctuating performance depending on the training set size, while the sigmoid kernel lagged behind across all scenarios. Increasing the training size generally enhanced the performance for every kernel, highlighting the importance of larger datasets for model generalization.

Influence of PCA Feature Reduction:

PCA was used to reduce the feature dimensionality of the Wine dataset while retaining most of its variance. The SVM models were then retrained on this reduced representation.

The results show that PCA positively influenced the performance of the SVM models, especially when using the linear and gaussian kernels. Metrics such as accuracy, precision, and recall improved, indicating that the lower-dimensional representation made patterns in the data more accessible to the models.

	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

Confusion Matrices:

- Confusion Matrices for SVM on Original Data here: **See in Google Colab file.**
- Confusion Matrices for SVM on PCA-Reduced Data here: **See in Google Colab file.**

The confusion matrices reveal that PCA, particularly in combination with the linear kernel, resulted in a higher proportion of correct classifications compared to the original feature space.

ROC and AUC Curves:

The ROC and AUC plots reinforce these findings, with the linear kernel achieving the highest AUC values and PCA further enhancing the separation between classes.

---

2. Multilayer Perceptron (MLP)

This section examines how MLP models perform on the Wine dataset, with emphasis on training size effects, learning behavior, and PCA-based dimensionality reduction.

Effect of Training Size:

The MLP model was trained with dataset splits of 50%, 60%, 70%, and 80% to observe changes in performance as training size increased.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.505618	0.454262	0.505618	0.441859
1	60	0.336449	0.113198	0.336449	0.169401
2	70	0.656000	0.724503	0.656000	0.678092
3	80	0.587413	0.519894	0.587413	0.505192

On the original dataset, the MLP's performance varied across training sizes without showing a consistent upward trend. This suggests that the chosen hyperparameters may not have been fully optimized for the dataset.

Training Loss Curve:

Insert MLP Training Loss Curves: **See in Google Colab file.**

The loss curves provide insight into the convergence process. While the loss generally decreased over time, the curves indicate potential room for improvement through hyperparameter tuning or extending training epochs.

Effect of PCA Feature Reduction:

The MLP model was also tested on PCA-transformed features.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.988764	0.989085	0.988764	0.988759
1	60	0.971963	0.973461	0.971963	0.971842
2	70	0.952000	0.954961	0.952000	0.951970
3	80	0.944056	0.946736	0.944056	0.943487

With PCA, the MLP achieved a notable boost in performance. Both accuracy and other metrics improved significantly, suggesting that dimensionality reduction minimized noise and allowed the model to focus on the most important patterns in the data.

Confusion Matrices:

- Insert Confusion Matrices for MLP on Original Data: **See in Google Colab file.**
- Insert Confusion Matrices for MLP on PCA-Reduced Data: **See in Google Colab file.**

The confusion matrices show far fewer misclassifications on the PCA-reduced dataset compared to the original, underscoring PCA's positive effect.

ROC and AUC Curves:

When PCA was applied, the ROC and AUC curves showed higher AUC values, confirming that the MLP could distinguish classes more effectively with reduced dimensions.

---

3. Random Forest

This section reviews the Random Forest model's performance on the Wine dataset, considering both original and PCA-transformed features.

Effect of Training Size:

Random Forest was evaluated on training splits of 50%, 60%, 70%, and 80%.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.977528	0.978777	0.977528	0.977498
1	60	0.934579	0.934670	0.934579	0.934515
2	70	0.952000	0.952117	0.952000	0.951798
3	80	0.951049	0.952787	0.951049	0.950976

The Random Forest model produced strong and stable results across all training sizes on the original dataset. Its robustness suggests that the algorithm is naturally well-suited to this classification task.

Effect of PCA Feature Reduction:

The same model was then trained on PCA-reduced data.

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

Unlike SVM and MLP, PCA slightly reduced the performance of Random Forest. This outcome suggests that the original features carried richer discriminatory information for the tree-based method, and dimensionality reduction removed some of that useful detail.

Confusion Matrices:

- Confusion Matrices for Random Forest on Original Data:**See in Google Colab file.**
- Confusion Matrices for Random Forest on PCA-Reduced Data: **See in Google Colab file.**

The confusion matrices confirm this observation, showing fewer misclassifications in the original dataset compared to PCA-reduced features.

ROC and AUC Curves:

The ROC and AUC plots further emphasize that Random Forest maintained stronger class separation on the original dataset compared to PCA-reduced data.

Handwritten Digit Dataset

1. Support Vector Machine (SVM)

This section evaluates the performance of SVM models on the Handwritten Digit dataset, focusing on how different kernels, training sizes, and dimensionality reduction through PCA affect the outcomes.

Influence of Training Size and Kernel Choice:

SVM models were tested with multiple kernel functions (linear, polynomial, gaussian, and sigmoid) across varying training proportions (50%, 60%, 70%, and 80%) on the Digit dataset.

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

For this dataset, the polynomial and gaussian kernels consistently achieved superior performance compared to the linear and sigmoid kernels. Their metrics remained strong and stable across different training sizes.

Effect of PCA Dimensionality Reduction:

PCA was applied to the Digit dataset to reduce its high-dimensional feature space before training SVM models.

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

The application of PCA led to noticeable improvements, particularly with the gaussian kernel. Accuracy and related measures increased, indicating that PCA effectively simplified the feature space while preserving key information.

Confusion Matrices:

- Confusion Matrices for SVM on Original Data here: **See in Google Colab file.**
- Confusion Matrices for SVM on PCA-Reduced Data here: **See in Google Colab file.**

The confusion matrices highlight that PCA helped SVM models, especially those using the gaussian kernel, to make more accurate predictions across different digits.

ROC and AUC Curves:

The ROC and AUC analyses reinforce the stronger performance of polynomial and gaussian kernels, as well as the performance boost after PCA reduction.

---

2. Multilayer Perceptron (MLP)

This section examines the performance of MLP models on the Handwritten Digit dataset, including training size effects, training loss progression, and PCA's influence.

Impact of Training Size:

The MLP model was trained on different portions of the dataset (50%, 60%, 70%, and 80%) to assess the effect of data availability.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.978648	0.978861	0.978648	0.978668
1	60	0.968861	0.969049	0.968861	0.968876
2	70	0.966955	0.967166	0.966955	0.966987
3	80	0.964635	0.965304	0.964635	0.964730

Across all training sizes, the MLP delivered high accuracy and strong metrics, showing consistent and reliable performance.

Training Loss Behavior:

MLP Training Loss Curves for Digit Dataset here: **See in Google Colab file.**

The training loss curves for the MLP reveal a steady decline, reflecting effective learning and convergence during training.

Effect of PCA Dimensionality Reduction:

MLP was also trained on PCA-transformed features for comparison.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.976868	0.976951	0.976868	0.976883
1	60	0.973310	0.973406	0.973310	0.973321
2	70	0.968480	0.968640	0.968480	0.968490
3	80	0.962856	0.963075	0.962856	0.962863

PCA enhanced the performance of the MLP, similar to what was observed in other datasets. By lowering dimensionality, the model managed the data complexity more effectively, leading to improved accuracy.

Confusion Matrices:

- Confusion Matrices for MLP on Original Data here: **See in Google Colab file.**
- Confusion Matrices for MLP on PCA-Reduced Data here: **See in Google Colab file.**

The confusion matrices confirm that PCA reduced misclassification errors, making predictions more precise.

ROC and AUC Curves:

On PCA-transformed data, ROC and AUC results show a stronger ability to discriminate between different classes.

---

3. Random Forest

This section explores the Random Forest model's performance on the Handwritten Digit dataset, both with original features and after applying PCA.

Impact of Training Size:

Random Forest models were trained using subsets of 50%, 60%, 70%, and 80% of the dataset to compare performance stability.

	Training size	Accuracy	Precision	Recall	F1-score
0	50	0.974733	0.975024	0.974733	0.974802
1	60	0.972716	0.972770	0.972716	0.972664
2	70	0.968480	0.968704	0.968480	0.968424
3	80	0.956628	0.957079	0.956628	0.956624

The Random Forest model consistently produced high-quality predictions on the original data, with little variation across training sizes.

Effect of PCA Dimensionality Reduction:

Random Forest was also trained on PCA-reduced data.

	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

Unlike SVM and MLP, Random Forest experienced a slight drop in performance when using PCA features. This suggests that the original feature set contained richer discriminatory signals for tree-based methods.

Confusion Matrices:

- Confusion Matrices for Random Forest on Original Data here: **See in Google Colab file.**
- Confusion Matrices for Random Forest on PCA-Reduced Data here: **See in Google Colab file.**

The confusion matrices show fewer classification errors when the model used the original dataset compared to PCA-transformed features.

ROC and AUC Curves:

The ROC and AUC plots indicate better performance for the original dataset, with PCA leading to a slight reduction in discriminatory power.