ECE 480 Final Project

Importing Python Libraries

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from matplotlib.patches import Ellipse
from sklearn.mixture import GaussianMixture
import seaborn as sns
import pandas as pd
import math
import os
```

Setting Important variables

In [ ]:
```
cluster_count = 5
mfccs = (1,3) #Can go from 1 to 13
mfcc_range = 3
```

Deciphering data and getting values

Import Data

In [ ]:
```
#Import the data by reading the text file /Spoken\ Arabic\ Digit/Train_Arabi
def import_data(file_path):
    #go through the text file line by line
    with open(file_path) as f:
        data = []
        for line in f:
            if not line.strip(): #if the line is empty
                #array of 13 0s
                line = [0] * 13
            else: #if the line is not empty
                #split the line by spaces
                line = line.split()
                #convert the strings to floats
                line = [float(i) for i in line]

            #append the line to the data array
            data.append(line)
    return data

spokenDigitDataRaw = import_data("Spoken Arabic Digit/Train_Arabic_Digit.txt
#print the first value of the first 40 rows
# print(spokenDigitDataRaw[0:40])
# print (spokenDigitDataRaw[1][0])
```

```
spokenDigitDataBlocks = []
index = 0
new_block = False

for cepstralValues in spokenDigitDataRaw:
   if cepstralValues[0] == cepstralValues[1] == 0:
            index += 1
            new_block = True
   else:
        if new_block:
            spokenDigitDataBlocks.append([])
            new_block = False
        spokenDigitDataBlocks[index-1].append(cepstralValues)

print(len(spokenDigitDataBlocks))
print(len(spokenDigitDataBlocks[0]))
print(len(spokenDigitDataBlocks[0][0]))
print(spokenDigitDataBlocks[0][0][0])
```

```
6600
38
13
-0.81101
```

Importing the Testing Data

In [ ]:
```
spokenDigitDataTestRaw = import_data("Spoken Arabic Digit/Test_Arabic_Digit.
spokenDigitDataTestBlocks = []
index = 0
new_block = False

for cepstralValues in spokenDigitDataTestRaw:
    if cepstralValues[0] == cepstralValues[1] == 0:
        index += 1
        new_block = True
    else:
        if new_block:
            spokenDigitDataTestBlocks.append([])
            new_block = False
        spokenDigitDataTestBlocks[index-1].append(cepstralValues)

print(len(spokenDigitDataTestBlocks))
print(len(spokenDigitDataTestBlocks[0]))
print(len(spokenDigitDataTestBlocks[0][0]))
print(spokenDigitDataTestBlocks[0][0][0])
```

```
2200
28
13
1.2572
```

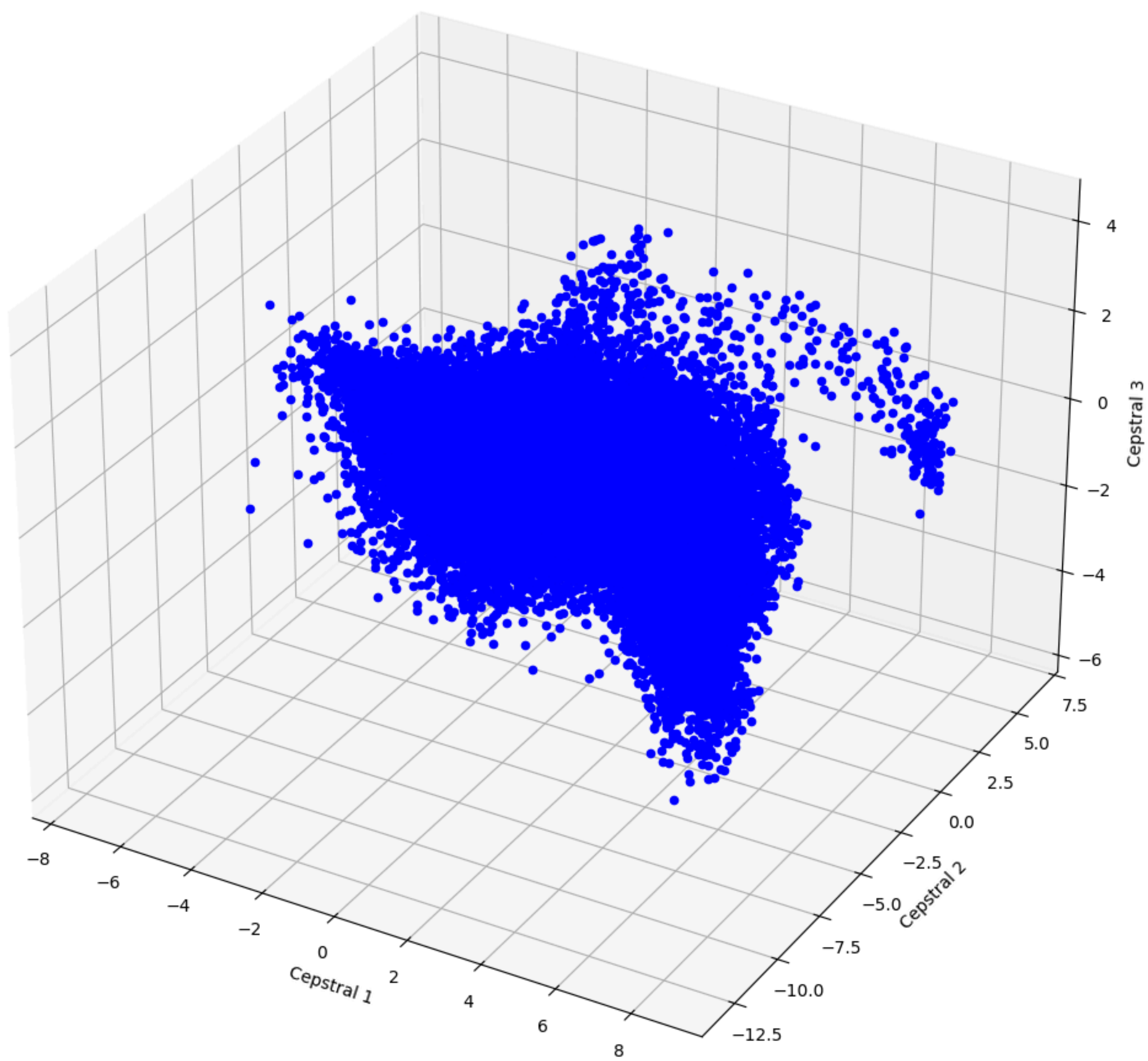
Graph Data

Graphing first three cepstrals of each digit

In [ ]:
```
#graph the first three cepstral cofeeicients of the first utterance

def extractCepstrals(block,first,last):
    cepstrals = []
    for j in range(first,last+1):
        cepstrals.append([])
    for i in range(len(block)):
        for j in range(first,last+1):
            cepstrals[j-first].append(block[i][j-first])

    return cepstrals

def extractAndPlotCepstrals(blocks,first,last):
    #create a figure
    plt.figure()
    cepstral_colors = sns.color_palette("viridis",last-first+1)
    if len(blocks) == 1: plt.figure(figsize=(10,3))
    else:
        plt.figure(figsize=(len(blocks)*2,len(blocks)))
        plt.suptitle("Cepstral Coefficient Plots")
    for block in blocks:
        cepstrals = extractCepstrals(block,first,last)
        if len(blocks) == 1: plt.subplot(1,1,1)
        else: plt.subplot( int(len(blocks)/2), int(len(blocks)/2) - 1 , bloc
        for i in range(len(cepstrals)):
            plt.plot(cepstrals[i], label = "Cepstral " + str((i+first)), col
            plt.title("Digit " + str(blocks.index(block)))
            plt.xlabel("Frame")
            plt.ylabel("Cepstral Value")
        plt.legend()
    plt.tight_layout()
    plt.show()


blocksToPlot = []
for i in range(10):
    blocksToPlot.append(spokenDigitDataBlocks[i*660])

# extractAndPlotCepstrals(blocksToPlot,1,3)
#Just Plotting 0
# extractAndPlotCepstrals([spokenDigitDataBlocks[0]],1,3)

#plot in 3d
def plot_block3D(blocks,first,last):
    fig = plt.figure()
    #size of the figure
    fig.set_size_inches(15,10)
    ax = fig.add_subplot(111, projection='3d')
    for block in blocks:
        cepstrals = extractCepstrals(block,first,last)
        for i in range(len(cepstrals[0])):
            ax.scatter(cepstrals[0][i],cepstrals[1][i],cepstrals[2][i], c =
    ax.set_xlabel("Cepstral 1")
```

```
    ax.set_ylabel("Cepstral 2")
    ax.set_zlabel("Cepstral 3")

    # fig.suptitle("3D Plot of Cepstral Coefficients for Digit 0")
    fig.tight_layout()
    plt.show()

data = []
for i in range(660):
    data.append(spokenDigitDataBlocks[i])
print(len(data))
plot_block3D(data,1,3)
```

660

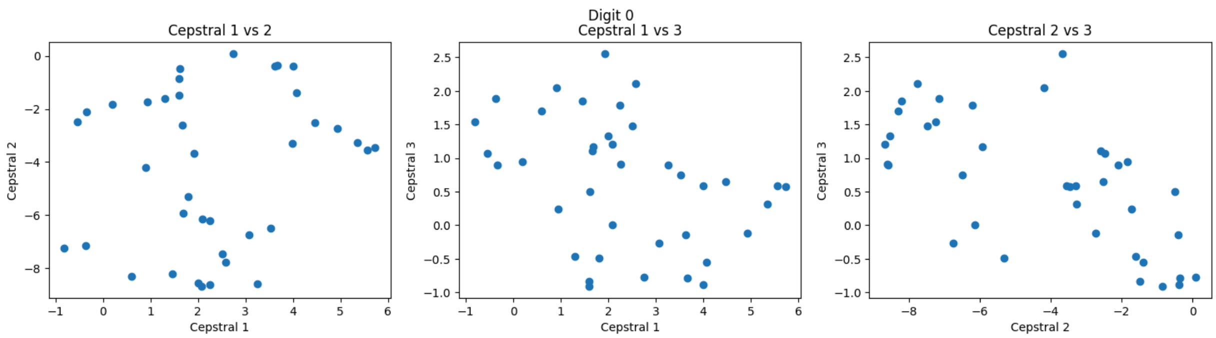


Scatter plots of each cepstral vs the other cepstrals (1-3) for each digit

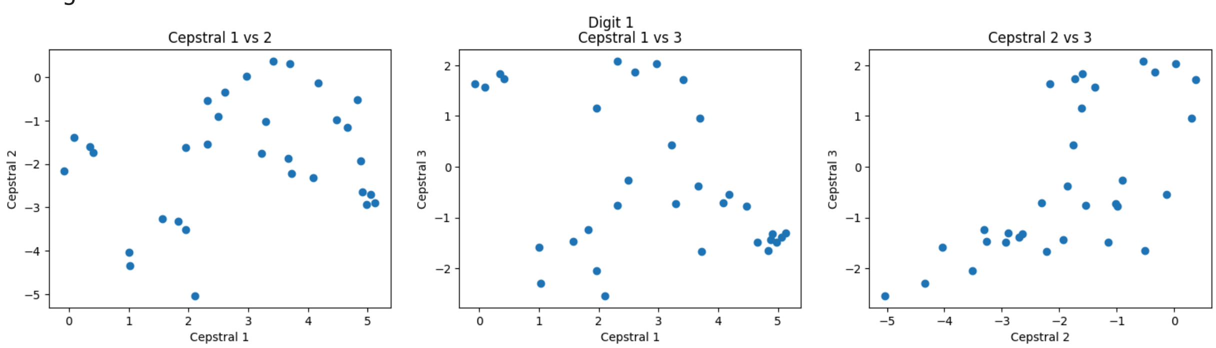
In [ ]:
```
def plotCepstral1v2v3 (block, suptitle = "Cepstral Coefficient Plots"):
    cepstrals = extractCepstrals(block,1,3)
    #1x3 subplot
```

```
    plt.figure()
    plt.figure(figsize=(18,4))
    plt.suptitle(suptitle)
    thingToGraph = [[1,2],[1,3],[2,3]]
    for pairs in thingToGraph:
        plt.subplot(1,3, thingToGraph.index(pairs)+1)
        plt.scatter(cepstrals[pairs[0]-1],cepstrals[pairs[1]-1])
        plt.title("Cepstral " + str(pairs[0]) + " vs " + str(pairs[1]))
        plt.xlabel("Cepstral " + str(pairs[0]))
        plt.ylabel("Cepstral " + str(pairs[1]))
    plt.show()

for i in range(10):
    plotCepstral1v2v3(spokenDigitDataBlocks[i*660], "Digit " + str(i))
```

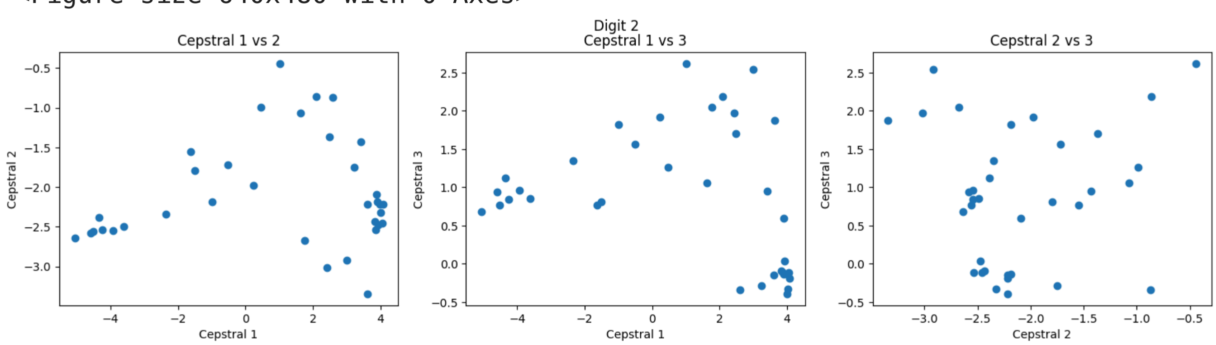
<Figure size 640x480 with 0 Axes>



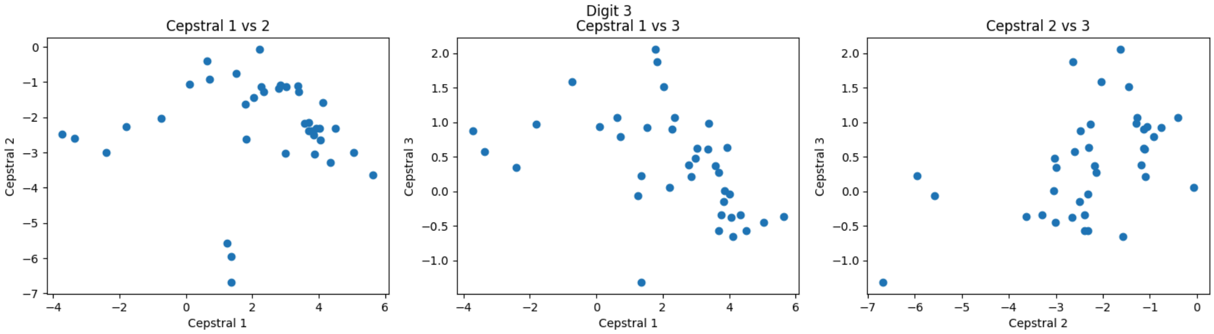
<Figure size 640x480 with 0 Axes>



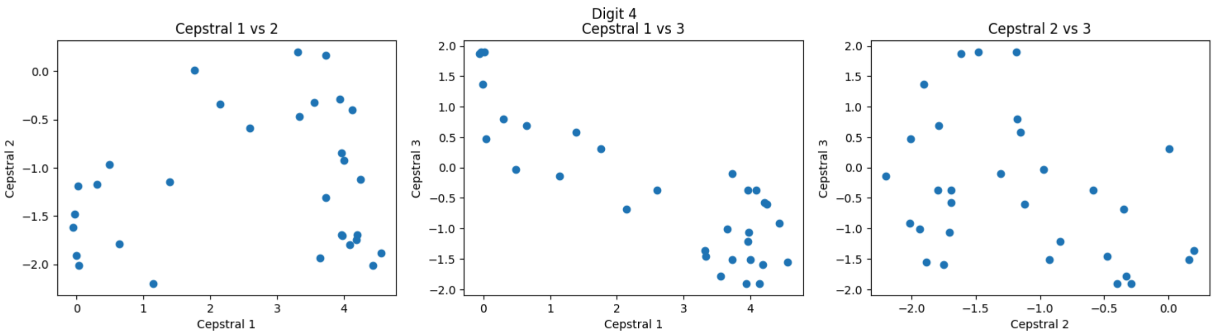
<Figure size 640x480 with 0 Axes>



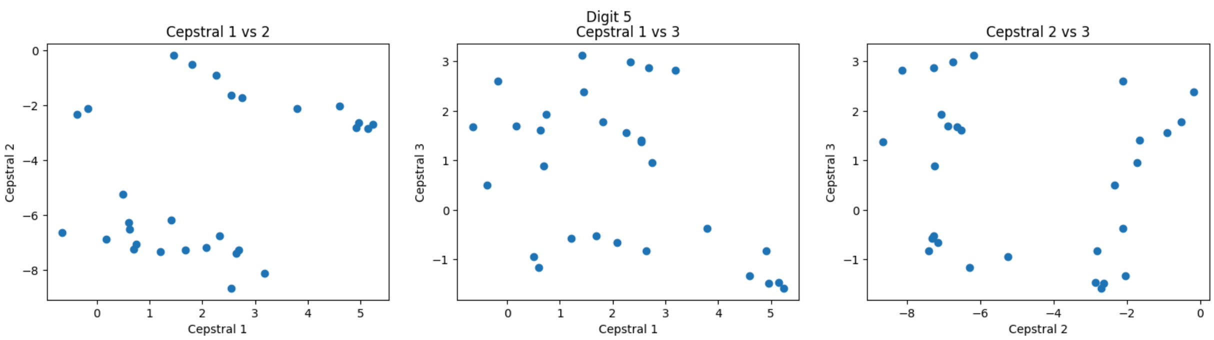
<Figure size 640x480 with 0 Axes>



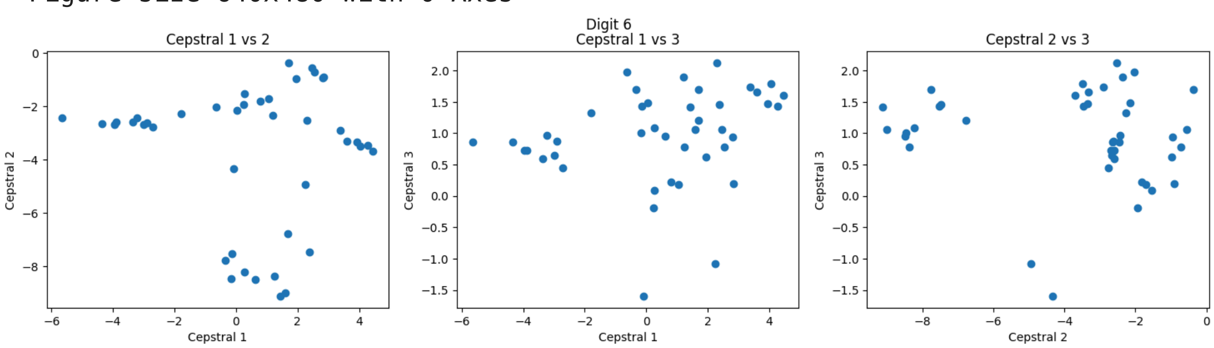
<Figure size 640x480 with 0 Axes>



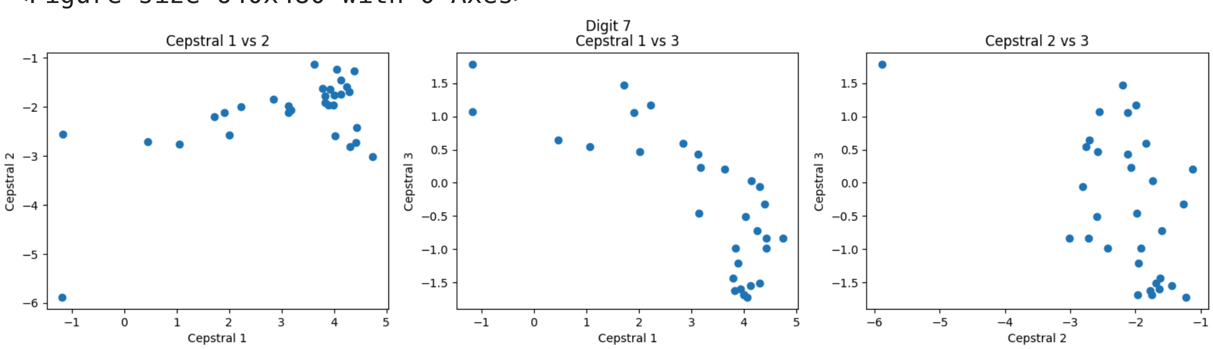
<Figure size 640x480 with 0 Axes>



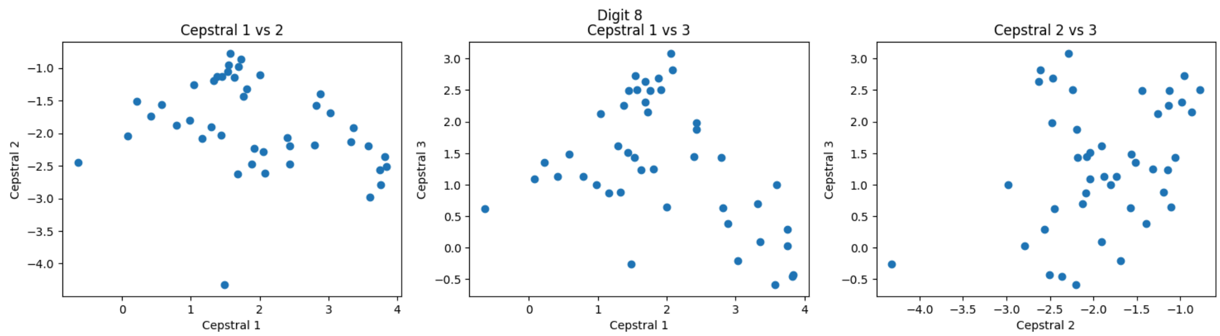
<Figure size 640x480 with 0 Axes>



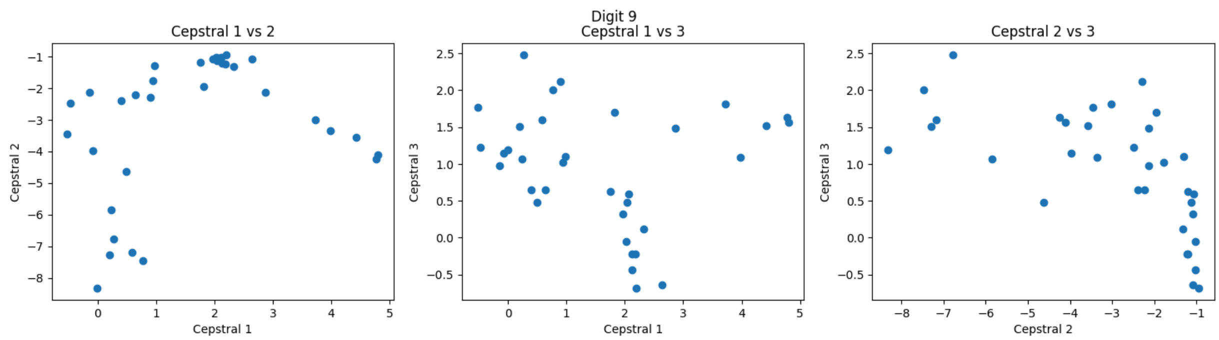
<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>




```
<Figure size 640x480 with 0 Axes>
```



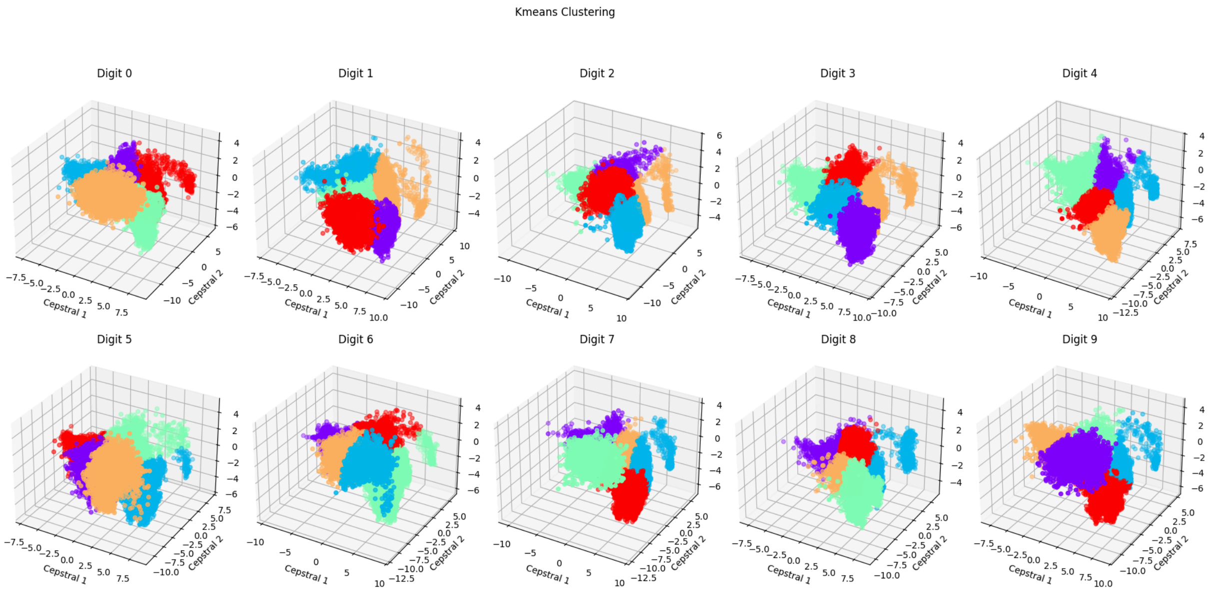
New dataframe of collected cepstrals for each digit

In [ ]:
```
collected_cepstrals = [] #[[[cepstral1],[cepstral2],[cepstral3]],[[cepstral1

def find_collected_cepstrals(mfcc_lower = 1, mfcc_upper = 3):
    collected_cepstrals = []
    digit = -1
    mfcc_range = mfcc_upper - mfcc_lower + 1
    for block in spokenDigitDataBlocks:
        #restart every 660 blocks
        if spokenDigitDataBlocks.index(block) % 660 == 0:
            digit += 1
            # print(digit, ' ', spokenDigitDataBlocks.index(block))
            list_to_append = []
            for i in range(mfcc_range):
                list_to_append.append([])
            collected_cepstrals.append(list_to_append)

        cepstrals = extractCepstrals(block, mfcc_lower, mfcc_upper)
        for i in range(mfcc_range):
            for value in cepstrals[i]:
                    collected_cepstrals[digit][i].append(value)

    return collected_cepstrals

collected_cepstrals = find_collected_cepstrals(mfccs[0], mfccs[1])
```

Kmeans

Kmeans of cepstral data

In [ ]:
```
#apply kmeans clustering to the data
def kmeansClustering(data, clusters):
    kmeans = KMeans(n_clusters = clusters)
    kmeans.fit(data)
    return kmeans

def plotKmeansClusters(kmeans, data, title = "Kmeans Clustering"):
    plt.figure()
    plt.figure(figsize=(10,10))
    plt.title(title)
    plt.scatter(data[:,0], data[:,1], c = kmeans.labels_, cmap = 'rainbow')
    plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1],
    plt.show()

def plotKmeansClusters3D(kmeans, data, title = "Kmeans Clustering"):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(data[:,0], data[:,1], data[:,2], c = kmeans.labels_, cmap = '
    ax.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], k
    #label the axes
    ax.set_xlabel('Cepstral 1')
    ax.set_ylabel('Cepstral 2')
    ax.set_zlabel('Cepstral 3')
    plt.title(title)
    plt.show()

def plotAllKmeansClusters3D(kmeans_list, data_list, title = "Kmeans Clusteri
    fig = plt.figure(figsize=(20,10))
    fig.suptitle(title)
    for i in range(10):
        data = np.array(data_list[i]).T
        ax = fig.add_subplot(2, 5, i+1, projection='3d')
        ax.scatter(data[:,0], data[:,1], data[:,2], c = kmeans_list[i].label
        ax.scatter(kmeans_list[i].cluster_centers_[:,0] ,kmeans_list[i].clus
        ax.set_xlabel('Cepstral 1')
        ax.set_ylabel('Cepstral 2')
        ax.set_zlabel('Cepstral 3')
        ax.set_title("\nDigit " + str(i))
    fig.tight_layout(rect=[0.05, 0.03, 1, 0.95])
    plt.show()

#apply kmeans to our collected cepstrals
kmeans_cepstrals = []
for i in range(10):
    kmeans = kmeansClustering(np.array(collected_cepstrals[i]).T, cluster_cc
    kmeans_cepstrals.append(kmeans)

if mfcc_range == 3: plotAllKmeansClusters3D(kmeans_cepstrals, collected_ceps
```



Getting liklihood of each digit being another digit

In [ ]:
```
## find the kmeans score of the individual digit data
def kmeansScore(kmeans, data):
    return kmeans.score(data)

def find_kmeans_liklihoods(kmeans_list, data):
    scores = []
    for i in range(10):
        scores.append(kmeansScore(kmeans_list[i], data))
    return scores

def predict_kmeans_digit(kmeans_list, data):
    scores = find_kmeans_liklihoods(kmeans_list, data)
    return scores.index(max(scores))

def make_kmeans_confusion_matrix(kmeans_list, data, mfcc_range = (1,3)):
    total_predictions = 0
    prediction_matrix = np.zeros((10,10))
    utterance_amounts = int(len(data)/10)

    for i in range(10):
        for block in data[i*utterance_amounts:i*utterance_amounts+int(uttera
            total_predictions += 1
            prediction = predict_kmeans_digit(kmeans_list, np.array(extractC
            prediction_matrix[i][prediction] += 1

    return prediction_matrix

def plot_confusion_matrix(confusion_matrix):
    #plot the confusion matrix as a heatmap
    plt.figure()
    plt.figure(figsize=(10,10))
    plt.imshow(confusion_matrix, interpolation='nearest')
    plt.title('Confusion Matrix')
    plt.xticks(np.arange(10))
```

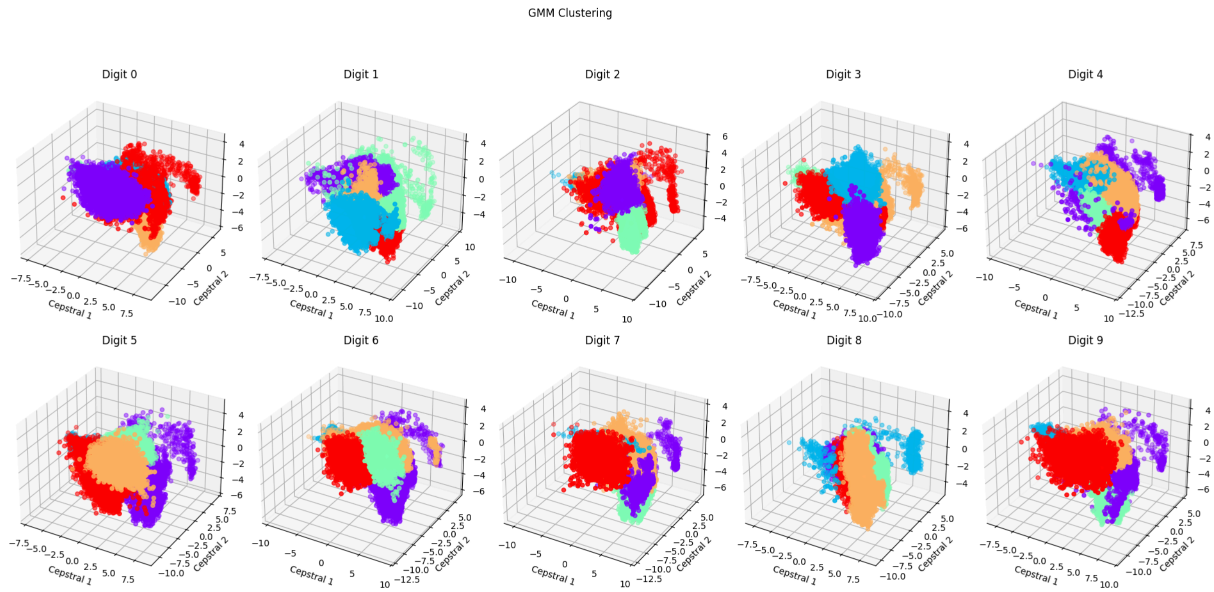
```
    plt.yticks(np.arange(10))
    plt.colorbar()
    plt.show()
```

EM GMM

Expectation Maximization gmm of cepstral data

In [ ]:
```
#Apply GMM clustering to the data
def GMMClustering(data, clusters, covariance_type = 'full'):
    gmm = GaussianMixture(n_components = clusters, covariance_type = covaria
    gmm.fit(data)
    return gmm

def plotGMMClusters(gmm, data, title = "GMM Clustering"):
    plt.figure()
    plt.figure(figsize=(10,10))
    plt.title(title)
    plt.scatter(data[:,0], data[:,1], c = gmm.predict(data), cmap = 'rainbow
    plt.scatter(gmm.means_[:,0] ,gmm.means_[:,1], color = 'black')
    plt.show()

def plotGMMClusters3D(gmm, data, title = "GMM Clustering"):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(data[:,0], data[:,1], data[:,2], c = gmm.predict(data), cmap
    ax.scatter(gmm.means_[:,0] ,gmm.means_[:,1], gmm.means_[:,2], color = 'b
    #label the axes
    ax.set_xlabel('Cepstral 1')
    ax.set_ylabel('Cepstral 2')
    ax.set_zlabel('Cepstral 3')
    plt.title(title)
    plt.show()

def plotAllGMMClusters3D(gmm_list, data_list, title = "GMM Clustering"):
    fig = plt.figure(figsize=(20,10))
    fig.suptitle(title)
    for i in range(10):
        data = np.array(data_list[i]).T
        ax = fig.add_subplot(2, 5, i+1, projection='3d')
        ax.scatter(data[:,0], data[:,1], data[:,2], c = gmm_list[i].predict(
        ax.scatter(gmm_list[i].means_[:,0] ,gmm_list[i].means_[:,1], gmm_lis
        ax.set_xlabel('Cepstral 1')
        ax.set_ylabel('Cepstral 2')
        ax.set_zlabel('Cepstral 3')
        ax.set_title("\nDigit " + str(i))
    fig.tight_layout(rect=[0.05, 0.03, 1, 0.95])
    plt.show()

#apply GMM to our collected cepstrals
gmm_cepstrals = []
for i in range(10):
    gmm = GMMClustering(np.array(collected_cepstrals[i]).T, cluster_count, '
```

```
    gmm_cepstrals.append(gmm)

if mfcc_range == 3: plotAllGMMClusters3D(gmm_cepstrals, collected_cepstrals)
```



Getting liklihood of each digit being another digit

In [ ]:
```
## find the GMM score of the individual digit data
def GMMscore(gmm, data):
    return gmm.score(data)

def find_GMM_liklihoods(gmm_list, data):
    scores = []
    for i in range(10):
        scores.append(GMMscore(gmm_list[i], data))
    return scores

def predict_GMM_digit(gmm_list, data):
    scores = find_GMM_liklihoods(gmm_list, data)
    return scores.index(max(scores))

def make_GMM_confusion_matrix(gmm_list, data, mfcc_range = (1,3)):
    total_predictions = 0
    prediction_matrix = np.zeros((10,10))
    utterance_amounts = int(len(data)/10)

    for i in range(10):
        for block in data[i*utterance_amounts:i*utterance_amounts+int(uttera
            total_predictions += 1
            prediction = predict_GMM_digit(gmm_list, np.array(extractCepstra
            prediction_matrix[i][prediction] += 1

    return prediction_matrix
```

Visualizing Data

Plotting Functions

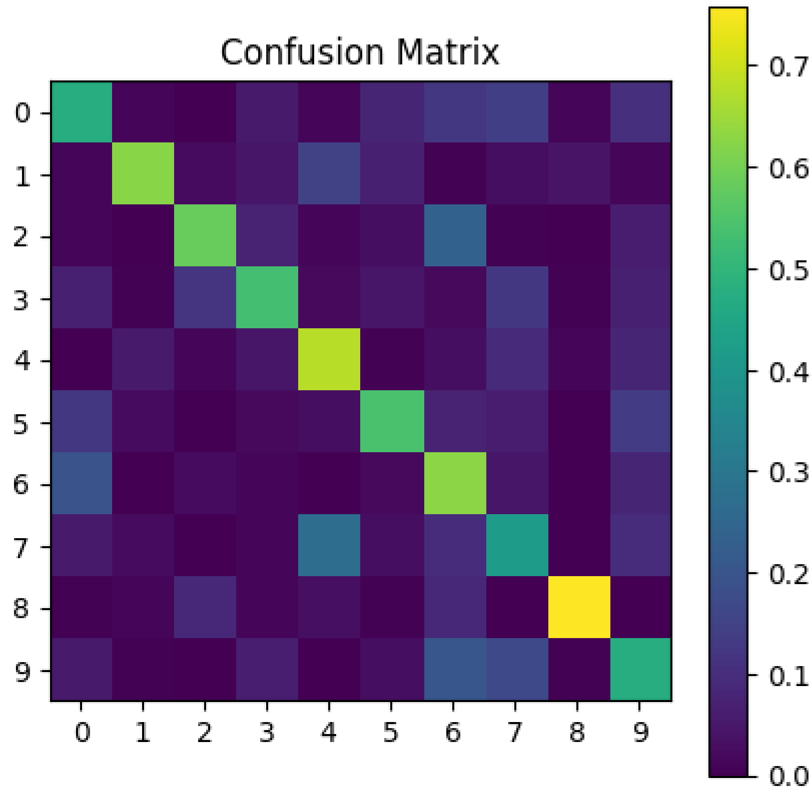
In [ ]:
```
def plot_confusion_matrix(confusion_matrix):
    #plot the confusion matrix as a heatmap
    plt.figure()
    plt.figure(figsize=(5,5))
    plt.imshow(confusion_matrix, interpolation='nearest')
    plt.title('Confusion Matrix')
    plt.xticks(np.arange(10))
    plt.yticks(np.arange(10))
    plt.colorbar()
    plt.show()

def plot_multiple_confusion_matrices(confusion_matrix_list, title_list, titl
    #dynamic number of subplots based on the number of confusion matrices
    subplots = len(confusion_matrix_list)
    if subplots <= 5: fig = plt.figure(figsize=(5*subplots,5))
    else: fig = plt.figure(figsize=(5*subplots,15))
    fig.suptitle(title, fontsize=35)
    for i in range(subplots):
        if subplots <= 5: ax = fig.add_subplot(1, subplots, i+1)
        else: ax = fig.add_subplot(2, int(subplots/2), i+1)
        ax.imshow(confusion_matrix_list[i], interpolation='nearest')
        ax.set_title(title_list[i])
        ax.set_xticks(np.arange(10))
        ax.set_yticks(np.arange(10))
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')
    fig.tight_layout(rect=[0.05, 0.03, 1, 0.95])
    plt.show()

def compute_accuracy(confusion_matrix):
    correct_predictions = 0
    total_predictions = 0
    for i in range(10):
        correct_predictions += confusion_matrix[i][i]
        total_predictions += sum(confusion_matrix[i])
    return correct_predictions/total_predictions
```

Plot Kmeans Confusion Matrix

In [ ]:
```
#apply Kmeans to our collected cepstrals
def find_kmeans_confusion_matrix(num_clusters, testData = True, cepstral_dat
    kmeans_cepstrals = []
    for i in range(10):
        kmeans = kmeansClustering(np.array(cepstral_data[i]).T, num_clusters
        kmeans_cepstrals.append(kmeans)

    if testData: digitBlocks = spokenDigitDataTestBlocks
    else: digitBlocks = spokenDigitDataBlocks
    kmeans_prediction_matrix = make_kmeans_confusion_matrix(kmeans_cepstrals
    #normalize the confusion matrix
    kmeans_prediction_matrix = kmeans_prediction_matrix / kmeans_prediction_
    return kmeans_prediction_matrix
```

```
kmeans_prediction_matrix = find_kmeans_confusion_matrix(cluster_count, colle
#display the confusion matrix as a heatmap
plot_confusion_matrix(kmeans_prediction_matrix)
```

<Figure size 640x480 with 0 Axes>



Plot EM GMM Confusion Matrix

Confusion Matrix Function

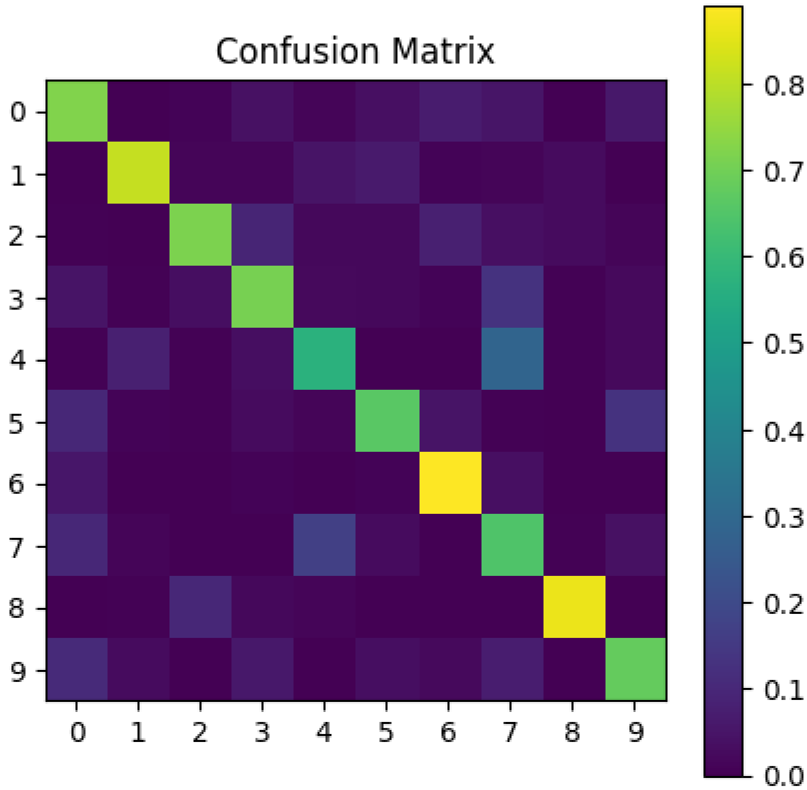
In [ ]:
```
def find_GMM_confusion_matrix(num_clusters, covariance_type, testData = True
    gmm_cepstrals = []
    for i in range(10):
        gmm = GMMClustering(np.array(cepstral_data[i]).T, num_clusters, cova
        gmm_cepstrals.append(gmm)

    if testData: digitBlocks = spokenDigitDataTestBlocks
    else: digitBlocks = spokenDigitDataBlocks
    gmm_prediction_matrix = make_GMM_confusion_matrix(gmm_cepstrals, digitBl
    #normalize the confusion matrix
    gmm_prediction_matrix = gmm_prediction_matrix / gmm_prediction_matrix.su
    return gmm_prediction_matrix
```

With Full Covariance

In [ ]:
```
#apply full covairance GMM to our collected cepstrals
gmm_prediction_matrix_full = find_GMM_confusion_matrix(cluster_count, 'full'
```

```
#display the confusion matrix as a heatmap
plot_confusion_matrix(gmm_prediction_matrix_full)
```

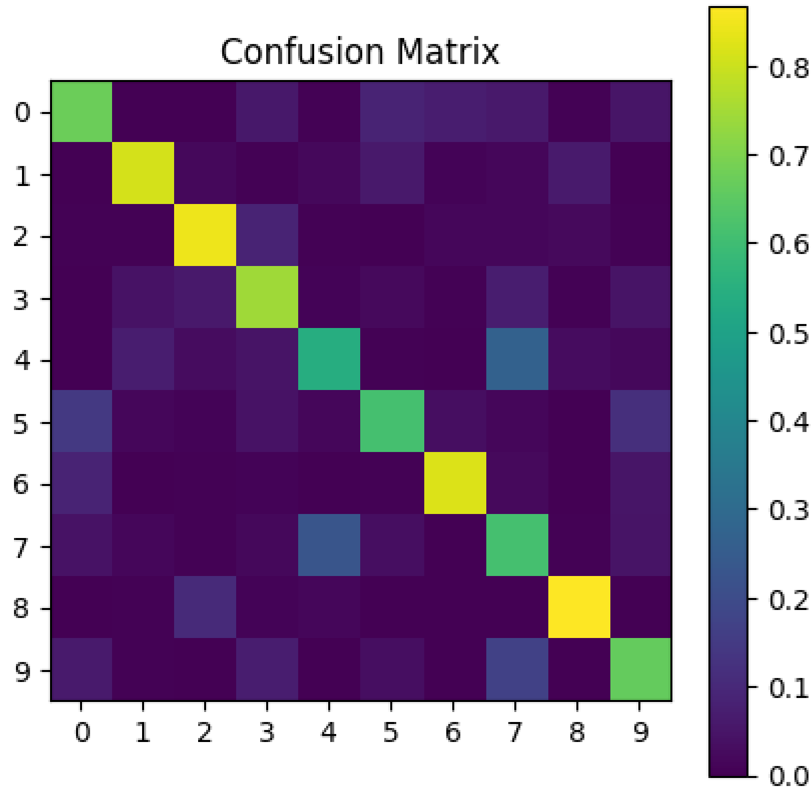
<Figure size 640x480 with 0 Axes>



With Diagonal Covariance

In [ ]:
```
#apply diagonal GMM to our collected cepstrals
gmm_prediction_matrix_diag = find_GMM_confusion_matrix(cluster_count, 'diag'
#display the confusion matrix as a heatmap
plot_confusion_matrix(gmm_prediction_matrix_diag)
```

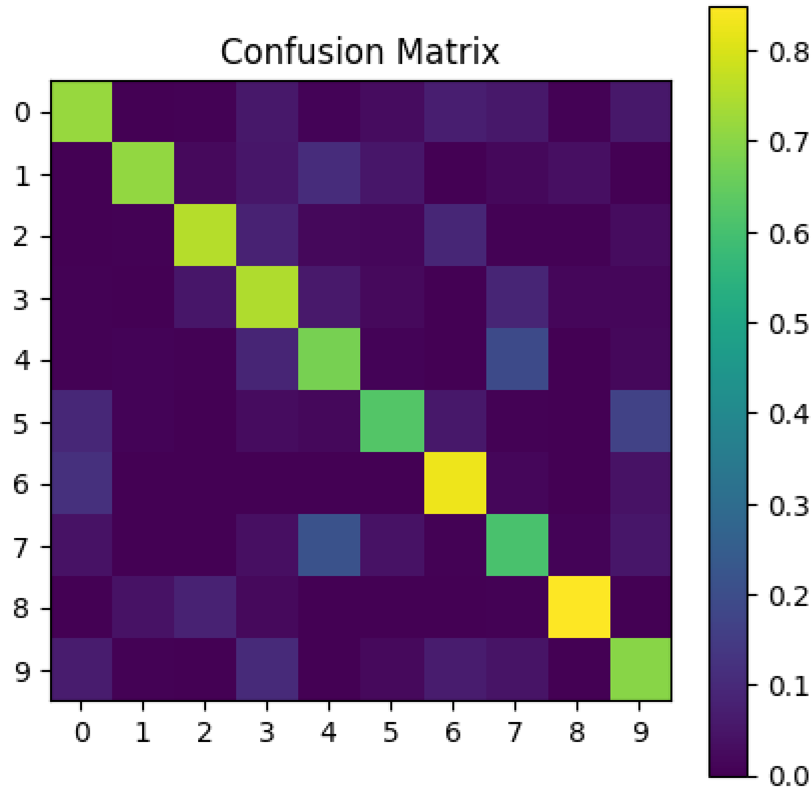
<Figure size 640x480 with 0 Axes>



With Tied Covariance

In [ ]:
```
#apply tied GMM to our collected cepstrals
gmm_prediction_matrix_tied = find_GMM_confusion_matrix(cluster_count, 'tied'
#display the confusion matrix as a heatmap
plot_confusion_matrix(gmm_prediction_matrix_tied)
```

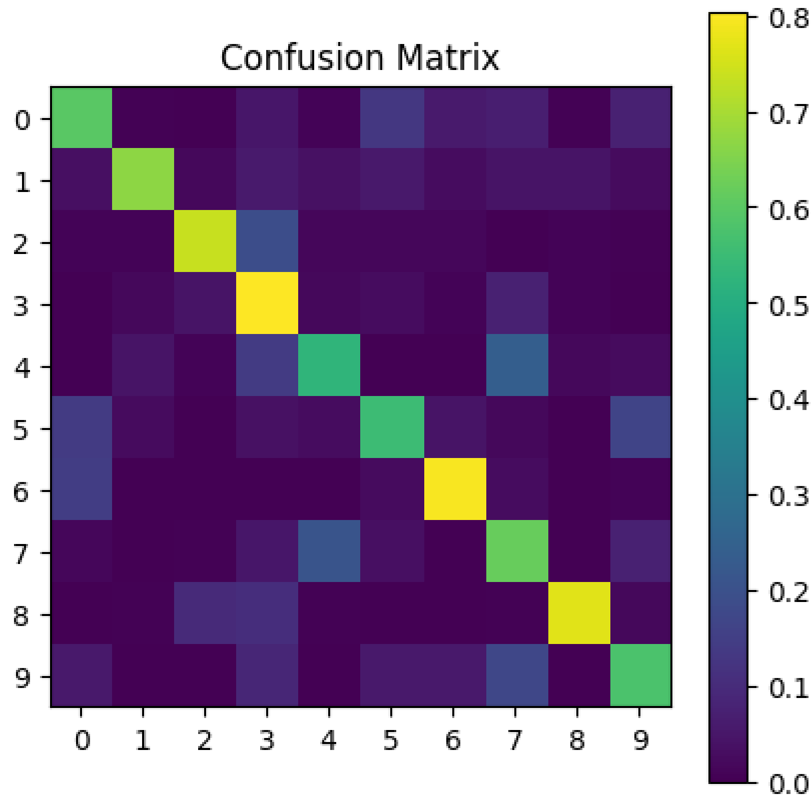
```
<Figure size 640x480 with 0 Axes>
```



With Spherical Covariance

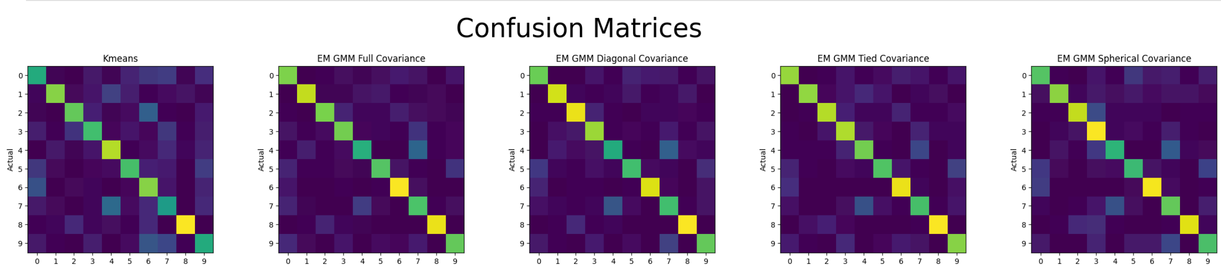
In [ ]:
```
#apply spherical GMM to our collected cepstrals
gmm_prediction_matrix_spherical = find_GMM_confusion_matrix(cluster_count, '
#display the confusion matrix as a heatmap
plot_confusion_matrix(gmm_prediction_matrix_spherical)
```

```
<Figure size 640x480 with 0 Axes>
```

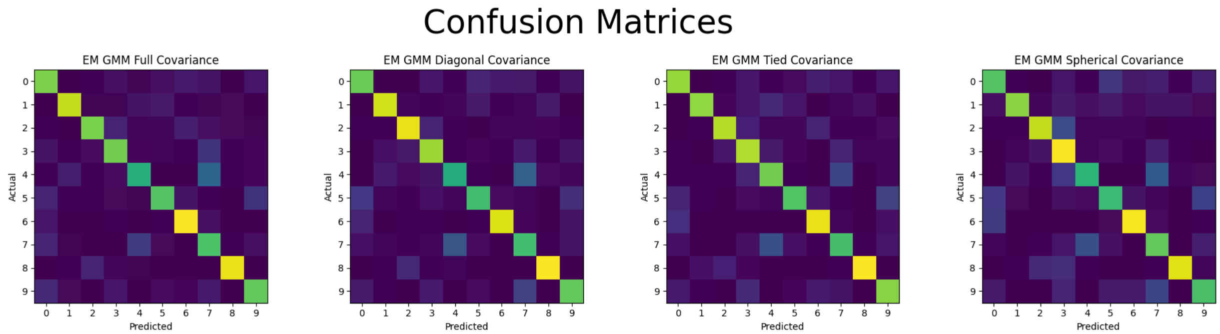


Comparing Confusion Matricies

In [ ]:
```
titles = ["Kmeans", "EM GMM Full Covariance", "EM GMM Diagonal Covariance",
confusion_matrices = [kmeans_prediction_matrix, gmm_prediction_matrix_full,
plot_multiple_confusion_matrices(confusion_matrices, titles)

#compute the accuracy of each model
for i in range(len(confusion_matrices)):
    #up to 2 decimal places
    accuracy = round(compute_accuracy(confusion_matrices[i])*100, 2)
    print("Accuracy of ", titles[i], " model: ", accuracy, "%")

#without kmeans
titles = ["EM GMM Full Covariance", "EM GMM Diagonal Covariance", "EM GMM Ti
confusion_matrices = [gmm_prediction_matrix_full, gmm_prediction_matrix_diag
plot_multiple_confusion_matrices(confusion_matrices, titles)
```




```
Accuracy of  Kmeans  model:  57.05 %
Accuracy of  EM GMM Full Covariance  model:  72.73 %
Accuracy of  EM GMM Diagonal Covariance  model:  72.0 %
Accuracy of  EM GMM Tied Covariance  model:  72.18 %
Accuracy of  EM GMM Spherical Covariance  model:  66.45 %
```



Comparing different cluster counts

Accuracy function

In [ ]:
```
def find_counts_accuracies(counts):
    kmeans_accuracies = []
    gmm_accuracies_full = []
    gmm_accuracies_diag = []
    gmm_accuracies_tied = []
    gmm_accuracies_spherical = []
    for i in counts:
        kmeans_prediction_matrix = find_kmeans_confusion_matrix(i)
        gmm_prediction_matrix_full = find_GMM_confusion_matrix(i, 'full')
        gmm_prediction_matrix_diag = find_GMM_confusion_matrix(i, 'diag')
        gmm_prediction_matrix_tied = find_GMM_confusion_matrix(i, 'tied')
        gmm_prediction_matrix_spherical = find_GMM_confusion_matrix(i, 'sphe
        kmeans_accuracies.append(compute_accuracy(kmeans_prediction_matrix))
        gmm_accuracies_full.append(compute_accuracy(gmm_prediction_matrix_fu
        gmm_accuracies_diag.append(compute_accuracy(gmm_prediction_matrix_di
        gmm_accuracies_tied.append(compute_accuracy(gmm_prediction_matrix_ti
        gmm_accuracies_spherical.append(compute_accuracy(gmm_prediction_matr
    return kmeans_accuracies, gmm_accuracies_full, gmm_accuracies_diag, gmm_
```

Testing Different Counts

In [ ]:
```
low_counts = [2,3,4,5,6,7,8,9,10]
kmeans_accuracies_low, gmm_accuracies_full_low, gmm_accuracies_diag_low, gmm
```

In [ ]:
```
high_counts = [2,5,10,15,20,25,30,35,40,45,50]
kmeans_accuracies_high, gmm_accuracies_full_high, gmm_accuracies_diag_high,
```

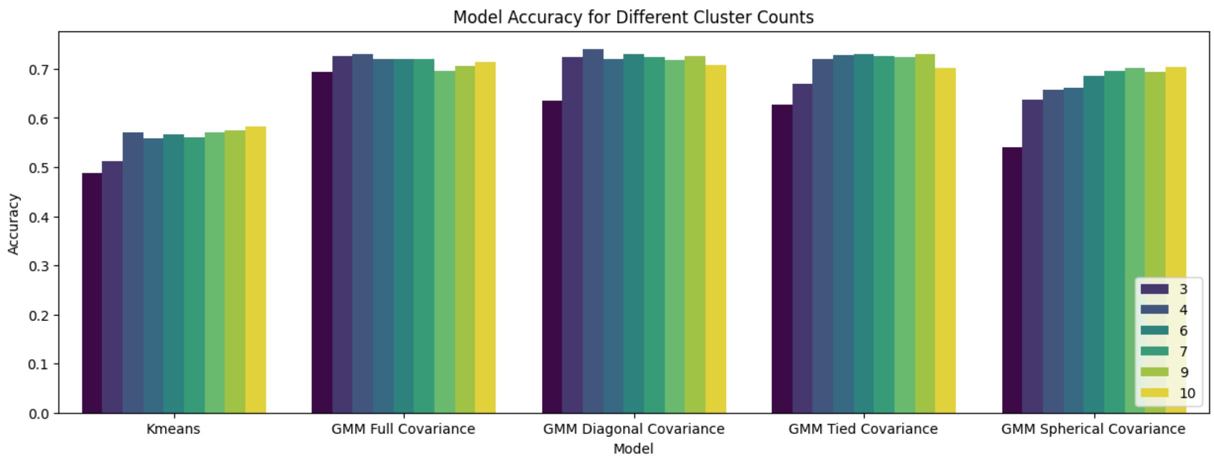
Plotting accuracies of the cluster counts

In [ ]:
```
def plot_bar_graph_accuracies(counts, kmeans_accuracies, gmm_accuracies_full
    #bar graph of the accuracy of each model for different cluster counts (u
    data = {'Cluster Count': counts, 'Kmeans': kmeans_accuracies, 'GMM Full 
    df = pd.DataFrame(data)
    df = df.melt('Cluster Count', var_name='Model', value_name='Accuracy')
    plt.figure(figsize=(15, 5))
    sns.barplot(x='Model', y='Accuracy', hue='Cluster Count', data=df, palet
    plt.title('Model Accuracy for Different Cluster Counts')
    # Move the legend to the bottom right
```

```
    plt.legend(loc='lower right', bbox_to_anchor=(1, 0))
    plt.show()

def plot_line_graph_accuracies(counts, kmeans_accuracies, gmm_accuracies_ful
    #line graph of the accuracy of each model for different cluster counts
    plt.figure(figsize=(15, 5))
    plt.plot(counts, kmeans_accuracies, label = 'Kmeans')
    plt.plot(counts, gmm_accuracies_full, label = 'GMM Full Covariance')
    plt.plot(counts, gmm_accuracies_diag, label = 'GMM Diagonal Covariance')
    plt.plot(counts, gmm_accuracies_tied, label = 'GMM Tied Covariance')
    plt.plot(counts, gmm_accuracies_spherical, label = 'GMM Spherical Covari
    plt.xlabel('Cluster Count')
    plt.ylabel('Accuracy')
    plt.title('Model Accuracy for Different Cluster Counts')
    plt.legend()
    plt.show()
```

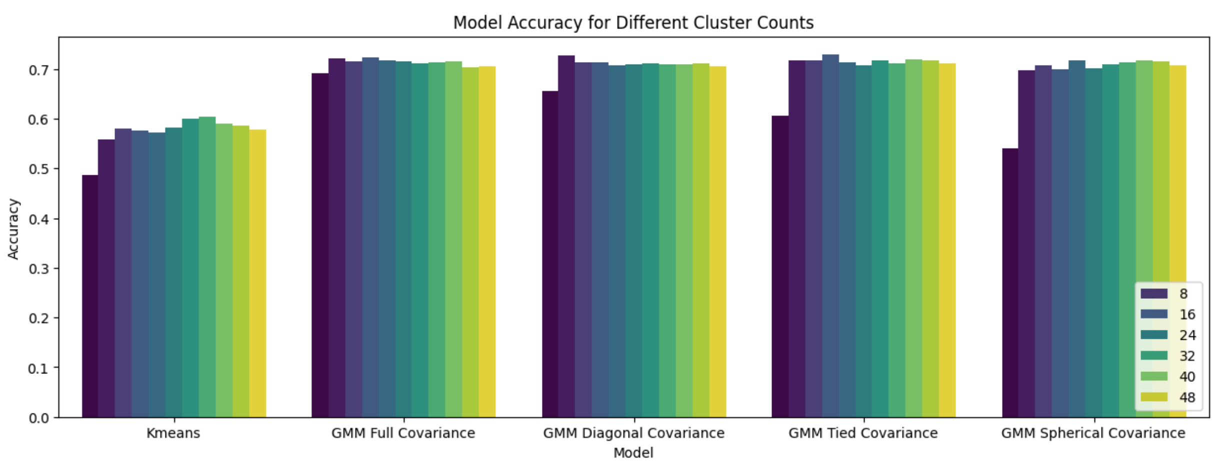
In [ ]:

```
#low count results
plot_bar_graph_accuracies(low_counts, kmeans_accuracies_low, gmm_accuracies_
```



In [ ]:

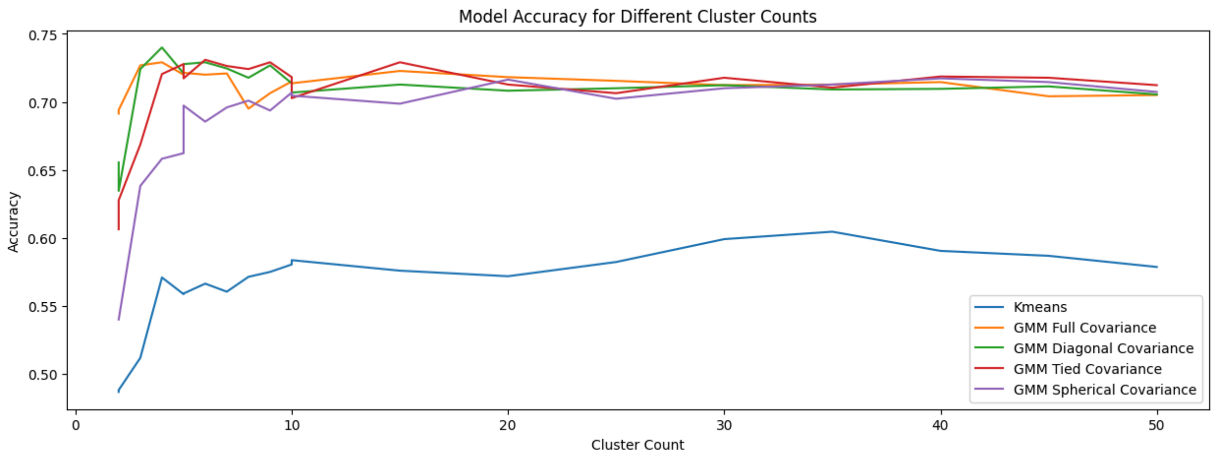
```
#high count results
plot_bar_graph_accuracies(high_counts, kmeans_accuracies_high, gmm_accuracie
```



In [ ]:

```
#combine the low and high count results
all_counts = low_counts + high_counts
all_kmeans_accuracies = kmeans_accuracies_low + kmeans_accuracies_high
all_gmm_accuracies_full = gmm_accuracies_full_low + gmm_accuracies_full_high
```

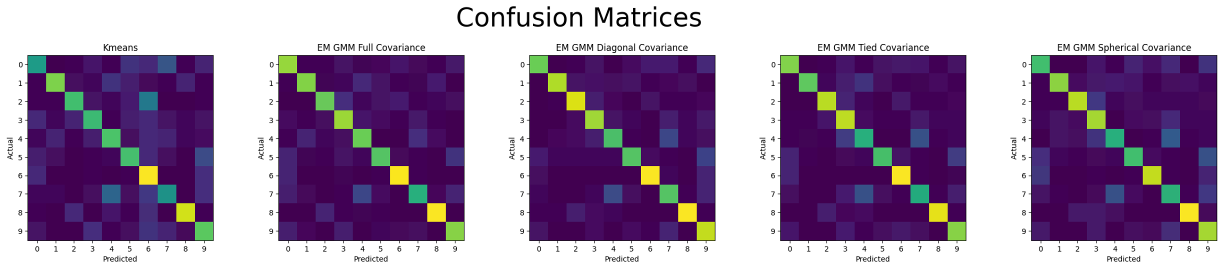
```
all_gmm_accuracies_diag = gmm_accuracies_diag_low + gmm_accuracies_diag_high
all_gmm_accuracies_tied = gmm_accuracies_tied_low + gmm_accuracies_tied_high
all_gmm_accuracies_spherical = gmm_accuracies_spherical_low + gmm_accuracies

#sort the results by cluster count
all_counts, all_kmeans_accuracies, all_gmm_accuracies_full, all_gmm_accuraci

plot_line_graph_accuracies(all_counts, all_kmeans_accuracies, all_gmm_accura
```



Using best performing cluster count from above

In [ ]:
```
cluster_count = 4

#find kmeans confusion matrix
kmeans_prediction_matrix = find_kmeans_confusion_matrix(cluster_count)
#find EM GMM full covariance confusion matrix
gmm_prediction_matrix_full = find_GMM_confusion_matrix(cluster_count, 'full'
#find EM GMM diagonal covariance confusion matrix
gmm_prediction_matrix_diag = find_GMM_confusion_matrix(cluster_count, 'diag'
#find EM GMM tied covariance confusion matrix
gmm_prediction_matrix_tied = find_GMM_confusion_matrix(cluster_count, 'tied'
#find EM GMM spherical covariance confusion matrix
gmm_prediction_matrix_spherical = find_GMM_confusion_matrix(cluster_count, '
```

In [ ]:
```
titles = ["Kmeans", "EM GMM Full Covariance", "EM GMM Diagonal Covariance",
confusion_matrices = [kmeans_prediction_matrix, gmm_prediction_matrix_full,
plot_multiple_confusion_matrices(confusion_matrices, titles)

#compute the accuracy of each model
for i in range(len(confusion_matrices)):
    #up to 2 decimal places
    accuracy = round(compute_accuracy(confusion_matrices[i])*100, 2)
    print("Accuracy of ", titles[i], " model: ", accuracy, "%")
```

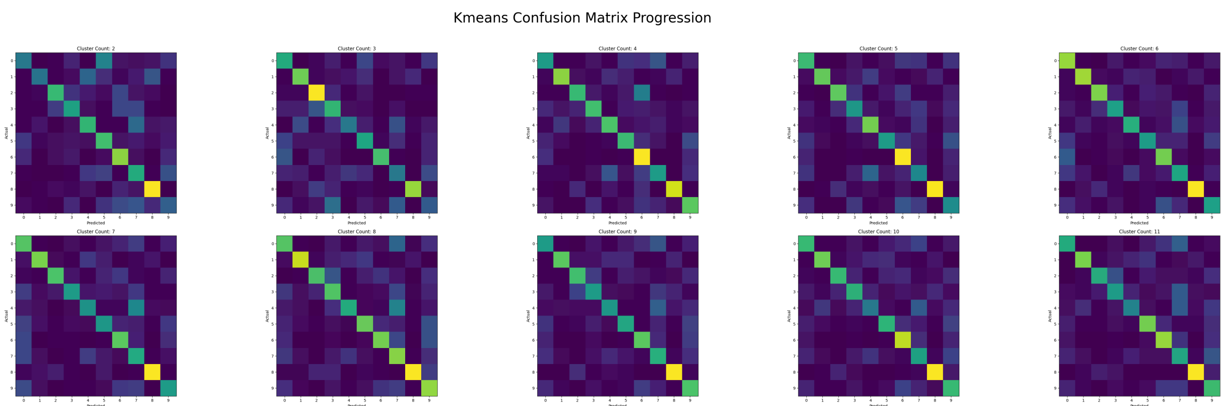



```
Accuracy of  Kmeans  model:  57.32 %
Accuracy of  EM GMM Full Covariance  model:  72.95 %
Accuracy of  EM GMM Diagonal Covariance  model:  74.09 %
Accuracy of  EM GMM Tied Covariance  model:  72.09 %
Accuracy of  EM GMM Spherical Covariance  model:  66.27 %
```

Testing Impact of Cluster Count on Kmeans

In [ ]:
```python
kmeans_counts = [2,3,4,5,6,7,8,9,10,11]
kmeans_predictions = []
for i in kmeans_counts:
    kmeans_prediction_matrix = find_kmeans_confusion_matrix(i)
    kmeans_predictions.append(kmeans_prediction_matrix)
```

In [ ]:
```python
#graph the kmeans confusion matrices
titles = ["Cluster Count: " + str(i) for i in kmeans_counts]
plot_multiple_confusion_matrices(kmeans_predictions, titles, title = "Kmeans
```



Comparing different MFCC ranges

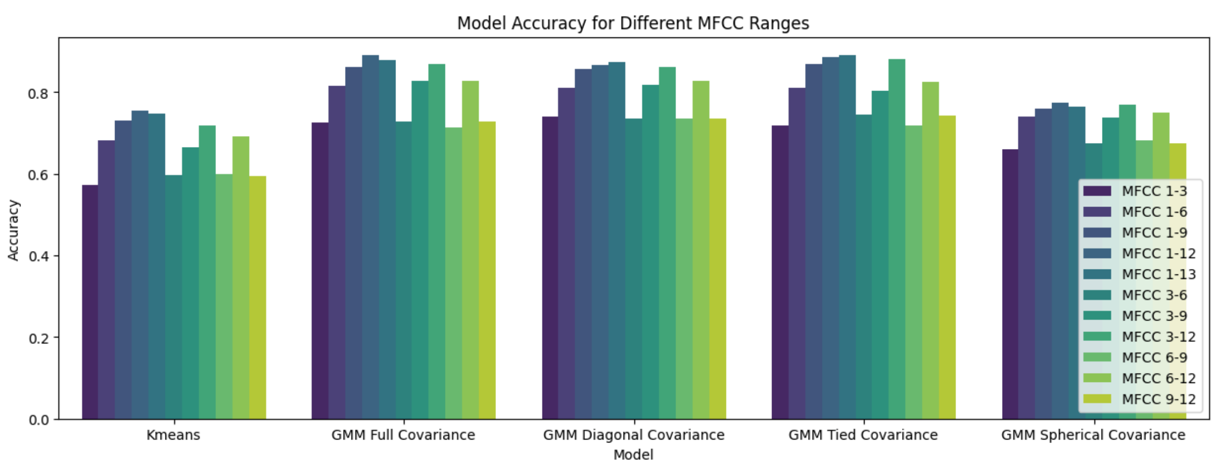
Comparing Wide Variety of MFCC Ranges

In [ ]:
```python
def find_mfcc_accuracies(mfcc_ranges, cluster_count = 4):
    kmeans_accuracies = []
    gmm_accuracies_full = []
    gmm_accuracies_diag = []
    gmm_accuracies_tied = []
    gmm_accuracies_spherical = []
    for range in mfcc_ranges:
        collected_cepstrals = find_collected_cepstrals(range[0], range[1])
        kmeans_prediction_matrix = find_kmeans_confusion_matrix(cluster_cour
        gmm_prediction_matrix_full = find_GMM_confusion_matrix(cluster_count
        gmm_prediction_matrix_diag = find_GMM_confusion_matrix(cluster_count
```

```
        gmm_prediction_matrix_tied = find_GMM_confusion_matrix(cluster_count
        gmm_prediction_matrix_spherical = find_GMM_confusion_matrix(cluster_
        kmeans_accuracies.append(compute_accuracy(kmeans_prediction_matrix))
        gmm_accuracies_full.append(compute_accuracy(gmm_prediction_matrix_fu
        gmm_accuracies_diag.append(compute_accuracy(gmm_prediction_matrix_di
        gmm_accuracies_tied.append(compute_accuracy(gmm_prediction_matrix_ti
        gmm_accuracies_spherical.append(compute_accuracy(gmm_prediction_matr
    return kmeans_accuracies, gmm_accuracies_full, gmm_accuracies_diag, gmm_

def plot_bar_graph_accuracies_mfccs(mfccs, kmeans_accuracies, gmm_accuracies
    #bar graph of the accuracy of each model for different cluster counts (u
    data = {'Cluster Count': mfccs, 'Kmeans': kmeans_accuracies, 'GMM Full C
    df = pd.DataFrame(data)
    df = df.melt('Cluster Count', var_name='Model', value_name='Accuracy')
    plt.figure(figsize=(15, 5))
    sns.barplot(x='Model', y='Accuracy', hue='Cluster Count', data=df, palet
    plt.title('Model Accuracy for Different MFCC Ranges')
    # Move the legend to the bottom right
    plt.legend(loc='lower right', bbox_to_anchor=(1, 0))
    plt.show()
```

In [ ]:
```
mfccs = [[1,3],[1,6],[1,9],[1,12],[1,13],[3,6],[3,9],[3,12],[6,9],[6,12],[9,
kmeans_accuracies, gmm_accuracies_full, gmm_accuracies_diag, gmm_accuracies_
```

In [ ]:
```
mfcc_string = []
for mfcc in mfccs:
    range_string = 'MFCC ' + str(mfcc[0]) + '-' + str(mfcc[1])
    mfcc_string.append(range_string)

#bar graph of the accuracy of each model for different MFCC ranges
plot_bar_graph_accuracies_mfccs(mfcc_string, kmeans_accuracies, gmm_accuraci
```



In [ ]:
```
# Initialize variables to track maximum accuracy and corresponding indices
max_kmeans_idx = 0
max_kmeans_accuracy = 0

max_gmm_full_idx = 0
max_gmm_full = 0

max_gmm_diag_idx = 0
max_gmm_diag = 0
```

```
max_gmm_tied_idx = 0
max_gmm_tied = 0

max_gmm_spherical_idx = 0
max_gmm_spherical = 0

# Iterate over the MFCCs and calculate the accuracies
for i in range(len(mfccs)):
    kmeans_accuracy = kmeans_accuracies[i] * 100
    gmm_full_accuracy = gmm_accuracies_full[i] * 100
    gmm_diag_accuracy = gmm_accuracies_diag[i] * 100
    gmm_tied_accuracy = gmm_accuracies_tied[i] * 100
    gmm_spherical_accuracy = gmm_accuracies_spherical[i] * 100

    # Update maximum KMeans accuracy
    if kmeans_accuracy > max_kmeans_accuracy:
        max_kmeans_accuracy = kmeans_accuracy
        max_kmeans_idx = i

    # Update maximum GMM Full accuracy
    if gmm_full_accuracy > max_gmm_full:
        max_gmm_full = gmm_full_accuracy
        max_gmm_full_idx = i

    # Update maximum GMM Diagonal accuracy
    if gmm_diag_accuracy > max_gmm_diag:
        max_gmm_diag = gmm_diag_accuracy
        max_gmm_diag_idx = i

    # Update maximum GMM Tied accuracy
    if gmm_tied_accuracy > max_gmm_tied:
        max_gmm_tied = gmm_tied_accuracy
        max_gmm_tied_idx = i

    # Update maximum GMM Spherical accuracy
    if gmm_spherical_accuracy > max_gmm_spherical:
        max_gmm_spherical = gmm_spherical_accuracy
        max_gmm_spherical_idx = i

# Print the highest accuracies and corresponding MFCC ranges
print("Best KMeans Accuracy: ", round(max_kmeans_accuracy, 2), "%, with MFCC
print("Best GMM Full Accuracy: ", round(max_gmm_full, 2), "%, with MFCC rang
print("Best GMM Diagonal Accuracy: ", round(max_gmm_diag, 2), "%, with MFCC
print("Best GMM Tied Accuracy: ", round(max_gmm_tied, 2), "%, with MFCC rang
print("Best GMM Spherical Accuracy: ", round(max_gmm_spherical, 2), "%, with
```

```
Best KMeans Accuracy:  75.55 %, with MFCC range [1, 12]
Best GMM Full Accuracy:  89.09 %, with MFCC range [1, 12]
Best GMM Diagonal Accuracy:  87.36 %, with MFCC range [1, 13]
Best GMM Tied Accuracy:  89.05 %, with MFCC range [1, 13]
Best GMM Spherical Accuracy:  77.55 %, with MFCC range [1, 12]
```

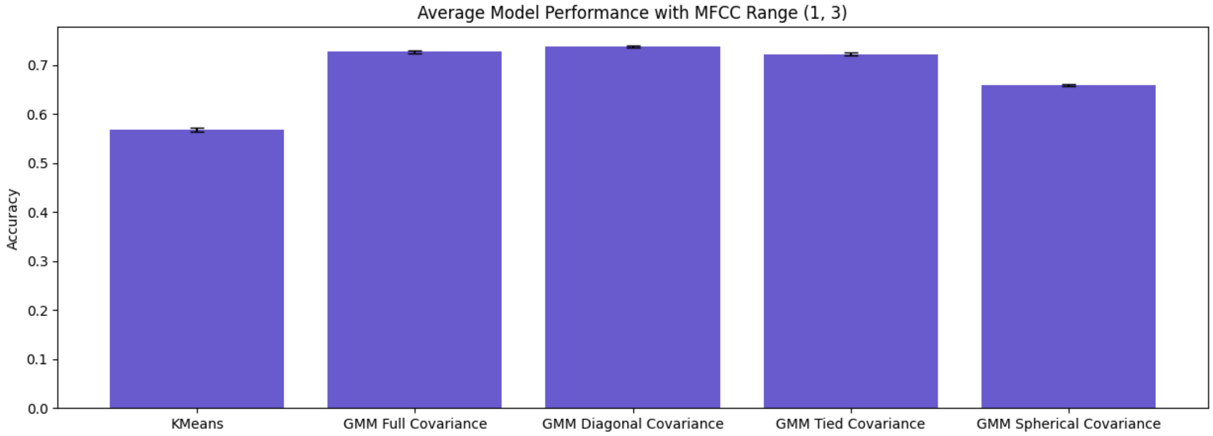
Comparing model performance averages

In [ ]:
```
def find_average_kmeans_performance(cluster_count, mfcc_range, iterations):
    total_accuracy = 0
    accuracies = []
    for i in range(iterations):
        collected_cepstrals = find_collected_cepstrals(mfcc_range[0], mfcc_r
        kmeans_prediction_matrix = find_kmeans_confusion_matrix(cluster_cour
        accuracy = compute_accuracy(kmeans_prediction_matrix)
        total_accuracy += accuracy
        accuracies.append(accuracy)
    mean = total_accuracy/iterations
    stdev = np.std(accuracies)
    best_accuracy = max(accuracies)
    return mean, stdev, best_accuracy

def find_average_gmm_performance(cluster_count, covariance_type, mfcc_range,
    total_accuracy = 0
    accuracies = []
    for i in range(iterations):
        collected_cepstrals = find_collected_cepstrals(mfcc_range[0], mfcc_r
        gmm_prediction_matrix = find_GMM_confusion_matrix(cluster_count, cov
        accuracy = compute_accuracy(gmm_prediction_matrix)
        total_accuracy += accuracy
        accuracies.append(accuracy)
    mean = total_accuracy/iterations
    stdev = np.std(accuracies)
    best_accuracy = max(accuracies)
    return mean, stdev, best_accuracy
```

MFCC Range 1-3

In [ ]:
```
# Find the average performance of KMeans and GMM models
iterations = 10
cluster_count = 4
mfcc_range = (1,3)
kmeans_accuracy_3, kmeans_stdev_3, kmeans_best_3 = find_average_kmeans_perfo
gmm_full_accuracy_3, gmm_full_stdev_3, gmm_full_best_3 = find_average_gmm_pe
gmm_diag_accuracy_3, gmm_diag_stdev_3, gmm_diag_best_3 = find_average_gmm_pe
gmm_tied_accuracy_3, gmm_tied_stdev_3, gmm_tied_best_3 = find_average_gmm_pe
gmm_spherical_accuracy_3, gmm_spherical_stdev_3, gmm_spherical_best_3 = finc
```

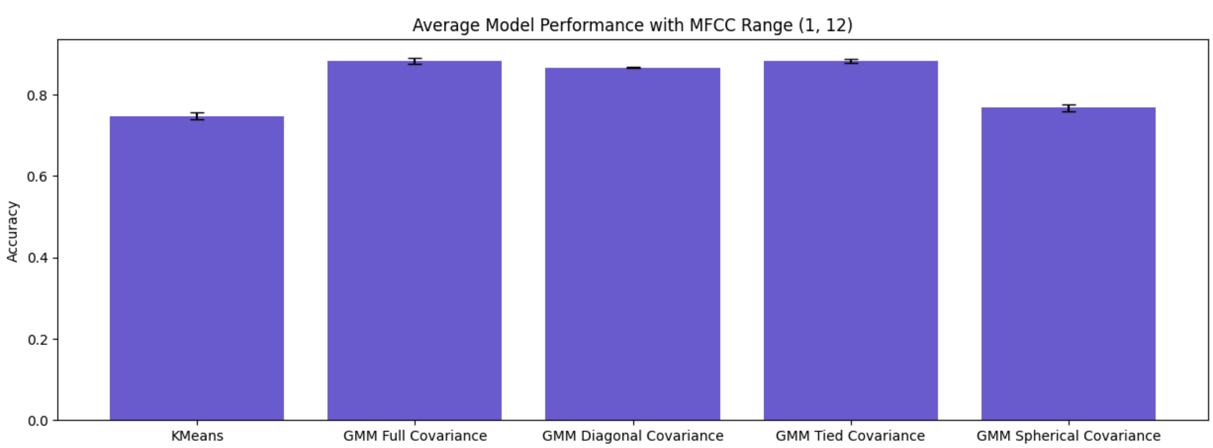
In [ ]:
```
#plot a bar graph of the average performance of each model
models = ['KMeans', 'GMM Full Covariance', 'GMM Diagonal Covariance', 'GMM T
accuracies = [kmeans_accuracy_3, gmm_full_accuracy_3, gmm_diag_accuracy_3, g
stdevs = [kmeans_stdev_3, gmm_full_stdev_3, gmm_diag_stdev_3, gmm_tied_stdev
# Create the bar plot with error bars
plt.figure(figsize=(15, 5))
plt.bar(models, accuracies, yerr=stdevs, capsize=5, color='slateblue')
#set colors
plt.ylabel('Accuracy')
plt.title('Average Model Performance with MFCC Range ' + str(mfcc_range))
plt.show()
```



MFCC Range 1-12

In [ ]:
```
# Find the average performance of KMeans and GMM models with best MFCC range
iterations = 10
cluster_count = 4
mfcc_range = (1,12)
kmeans_accuracy_12, kmeans_stdev_12, best_kmeans_12 = find_average_kmeans_pe
gmm_full_accuracy_12, gmm_full_stdev_12, best_gmm_full_12 = find_average_gmm
gmm_diag_accuracy_12, gmm_diag_stdev_12, best_gmm_diag_12 = find_average_gmm
gmm_tied_accuracy_12, gmm_tied_stdev_12, best_gmm_tied_12 = find_average_gmm
gmm_spherical_accuracy_12, gmm_spherical_stdev_12, best_gmm_spherical_12 = f
```

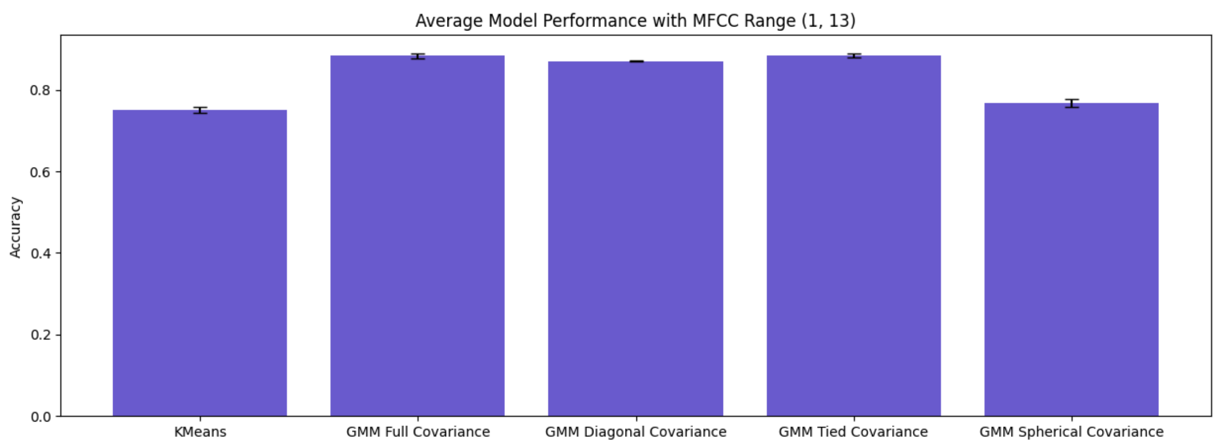
In [ ]:
```
#plot a bar graph of the average performance of each model
models = ['KMeans', 'GMM Full Covariance', 'GMM Diagonal Covariance', 'GMM T
accuracies = [kmeans_accuracy_12, gmm_full_accuracy_12, gmm_diag_accuracy_12
stdevs = [kmeans_stdev_12, gmm_full_stdev_12, gmm_diag_stdev_12, gmm_tied_st
# Create the bar plot with error bars
plt.figure(figsize=(15, 5))
plt.bar(models, accuracies, yerr=stdevs, capsize=5, color='slateblue')
#set colors
plt.ylabel('Accuracy')
plt.title('Average Model Performance with MFCC Range ' + str(mfcc_range))
plt.show()
```



MFCC Range 1-13

In [ ]:
```
# Find the average performance of KMeans and GMM models with best MFCC range
iterations = 10
cluster_count = 4
mfcc_range = (1,13)
kmeans_accuracy_13, kmeans_stdev_13, best_kmeans_13 = find_average_kmeans_pe
gmm_full_accuracy_13, gmm_full_stdev_13, best_gmm_full_13 = find_average_gmm
gmm_diag_accuracy_13, gmm_diag_stdev_13, best_gmm_diag_13 = find_average_gmm
gmm_tied_accuracy_13, gmm_tied_stdev_13, best_gmm_tied_13 = find_average_gmm
gmm_spherical_accuracy_13, gmm_spherical_stdev_13, best_gmm_spherical_13 = f
```

In [ ]:
```
#plot a bar graph of the average performance of each model
models = ['KMeans', 'GMM Full Covariance', 'GMM Diagonal Covariance', 'GMM T
accuracies = [kmeans_accuracy_13, gmm_full_accuracy_13, gmm_diag_accuracy_13
stdevs = [kmeans_stdev_13, gmm_full_stdev_13, gmm_diag_stdev_13, gmm_tied_st
# Create the bar plot with error bars
plt.figure(figsize=(15, 5))
plt.bar(models, accuracies, yerr=stdevs, capsize=5, color='slateblue')
#set colors
plt.ylabel('Accuracy')
plt.title('Average Model Performance with MFCC Range ' + str(mfcc_range))
plt.show()
```



MFCC Range 1-3, 1-12, and 1-13

In [ ]:
```
#Bar Graph of each MFCC accuracy + error bars
mfccs = [(1,3), (1,12), (1,13), (1,3), (1,12), (1,13), (1,3), (1,12), (1,13)
kmeans_accuracies = [kmeans_accuracy_3, kmeans_accuracy_12, kmeans_accuracy_
kmeans_stdevs = [kmeans_stdev_3, kmeans_stdev_12, kmeans_stdev_13]
gmm_full_accuracies = [gmm_full_accuracy_3, gmm_full_accuracy_12, gmm_full_a
gmm_full_stdevs = [gmm_full_stdev_3, gmm_full_stdev_12, gmm_full_stdev_13]
gmm_diag_accuracies = [gmm_diag_accuracy_3, gmm_diag_accuracy_12, gmm_diag_a
gmm_diag_stdevs = [gmm_diag_stdev_3, gmm_diag_stdev_12, gmm_diag_stdev_13]
gmm_tied_accuracies = [gmm_tied_accuracy_3, gmm_tied_accuracy_12, gmm_tied_a
gmm_tied_stdevs = [gmm_tied_stdev_3, gmm_tied_stdev_12, gmm_tied_stdev_13]
gmm_spherical_accuracies = [gmm_spherical_accuracy_3, gmm_spherical_accuracy
gmm_spherical_stdevs = [gmm_spherical_stdev_3, gmm_spherical_stdev_12, gmm_s

# Create a DataFrame for plotting
data = {
    'MFCC Config': [f"{m[0]}-{m[1]}" for m in mfccs],
    'Model': ['KMeans', 'KMeans', 'KMeans',
```

```
                'GMM Full', 'GMM Full', 'GMM Full',
                'GMM Diagonal', 'GMM Diagonal', 'GMM Diagonal',
                'GMM Tied', 'GMM Tied', 'GMM Tied',
                'GMM Spherical', 'GMM Spherical', 'GMM Spherical'],
    'Accuracy': kmeans_accuracies + gmm_full_accuracies + gmm_diag_accuracie
    'Standard Deviation': kmeans_stdevs + gmm_full_stdevs + gmm_diag_stdevs 
}
df = pd.DataFrame(data)
```

In [ ]:

```
# Plotting
plt.figure(figsize=(12, 6))
ax = sns.barplot(data=df, x='MFCC Config', y='Accuracy', hue='Model', palett

# Adding error bars for standard deviation
for i, model in enumerate(df['Model'].unique()):
    subset = df[df['Model'] == model]
    # Using ax.bar and plotting the error bars correctly
    for idx, (accuracy, std_dev) in enumerate(zip(subset['Accuracy'], subset
        ax.errorbar(x=idx + (i - 2) * 0.16,  # Adjusting x position to avoic
                    y=accuracy,
                    yerr=std_dev,
                    fmt='none',
                    capsize=5,
                    color='black',
                    alpha=0.7)

plt.title('Model Accuracies with Standard Deviation for Different MFCC Confi
plt.ylabel('Accuracy')
plt.xlabel('MFCC Configuration')
plt.tight_layout()
plt.legend(title='Model', loc='lower right')
plt.show()
```