In [147]:

```
# Running the Libraries
from pdf2image import convert_from_path # for converting PDF to JPG
from matplotlib.image import imread
import numpy as np
import pandas as pd
import scipy as sc
import os
import matplotlib.pyplot as plt
from pylab import *
from skimage.transform import resize  # for resizing the images
import statistics as stats

from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import recall_score, f1_score, precision_score
from keras.models import load_model
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Bat
from keras.models import Sequential
from pdf2image.exceptions import (PDFInfoNotInstalledError, PDFPageCountE

# Packages for ANOVA Table
import statsmodels.api as sm
from statsmodels.formula.api import ols
import scikit_posthocs as sp
from scipy.stats import f_oneway
from scipy.stats import levene

#Chi-Square
from bioinfokit.analys import stat, get_data

# Fisher-Exact Test
import rpy2.robjects.numpy2ri
from rpy2.robjects.packages import importr
rpy2.robjects.numpy2ri.activate()

# KNN
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt

# CNN
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow import keras
from sklearn.model_selection import GridSearchCV
from scikeras.wrappers import KerasClassifier

# K-Means
from sklearn.cluster import KMeans

# K-Medoids
from sklearn_extra.cluster import KMedoids
```

We will import the dataset containg the FEV1, GOLD Stage, COPD Score, & Adjusted Clock Drawing Test Score for 1,716 unique Subject IDs.

In [16]:
```
Dataset_1 = pd.read_csv('C:/Users/12563/Documents/Res. Methods in Math &
Dataset_1
```

Out[16]:

	sid	fev1pp_final	goldstage_final_V2	COPD_P3	Score
0	10015T	51.1	2	1	2
1	10017X	55.3	2	1	2
2	10031R	39.5	2	0	2
3	10032T	58.8	2	1	0
4	10049K	59.5	2	1	2
...	...	...	...	...	...
1712	23634Q	73.9	1	0	2
1713	23696M	72.4	2	1	2
1714	23785L	51.7	2	1	2
1715	23793K	108.6	0	0	2
1716	24637B	119.4	2	0	2

1717 rows × 5 columns

In [17]:
```
# Rename column in a dataframe
Dataset_1.rename(columns = {'sid':'Clocks' }, inplace = True)
```

We will perform the K-Nearest Neighbors Algorithm to solve the classification problem of weather or not a particular person (Subject ID) has COPD or not.

In [4]:
```
df_knn = pd.DataFrame(Dataset_1, columns = [ 'fev1pp_final', 'goldstage_
df_knn
```

Out[4]:

	fev1pp_final	goldstage_final_V2	COPD_P3	Score
0	51.1	2	1	2
1	55.3	2	1	2
2	39.5	2	0	2
3	58.8	2	1	0
4	59.5	2	1	2
...	...	...	...	...
1712	73.9	1	0	2
1713	72.4	2	1	2
1714	51.7	2	1	2
1715	108.6	0	0	2
1716	119.4	2	0	2

1717 rows × 4 columns

In [6]:
```
X_1 = df_knn.drop("COPD_P3", axis = 1)
X_1 = X_1.values
y_1 = df_knn['COPD_P3']
y_1 = y_1.values
```

In [7]:
```
# Split the data into train and test sets
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y_1, te
```

In [8]:
```
# KNN is a supervised machine learning method
# Trying to predict weather or not someone will have COPD based
# on their CDT Score (the predictor)
knn_model_1 = KNeighborsRegressor(n_neighbors=2)
knn_model_1.fit(X_train_1, y_train_1)
```

Out[8]:
```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=2)
```

In [9]:
```
# Calculate RMSE for Training Data
train_preds_1 = knn_model_1.predict(X_train_1)
mse_1 = mean_squared_error(y_train_1, train_preds_1)
rmse_1 = sqrt(mse_1)
rmse_1
```

Out[9]: 0.30741029992207347

In [10]:

```
# Calculate RMSE for Testing Data
test_preds_1_t = knn_model_1.predict(X_test_1)
mse_1_t = mean_squared_error(y_test_1, test_preds_1_t)
rmse_1_test = sqrt(mse_1_t)
rmse_1_test
```

Out[10]: 0.4985443928356902

In [11]:

```
# Calculate Accurary
y_pred_1 = knn_model_1.predict(X_test_1)

# Evaluate the training & testing Accuracy
#
print('Training accuracy score: %.3f' % knn_model_1.score(X_train_1, y_t
print('Test accuracy score: %.3f' % knn_model_1.score(X_test_1, y_test_1
```

```
Training accuracy score: 0.593
Test accuracy score: -0.132
```

Let us first import the Excel file containg 19,485 unqiue Subject IDs (SIDs) along with the assocaited Adjusted Clock Drawing Test Score (either 0 (High-Risk for Dementia) or 2 (not at High-Risk for Dementia)).

In [12]:

```
# Reading clock scores from CSV score file

df = pd.read_csv('C:/Users/12563/Desktop/CDT/Adjudicated Mini-Cog Clock 
scores1 = df.shape
df = df[df['Adjudicated Clock Score'].notnull()]
scores2 = df.shape
df['Adjudicated Clock Score'] = df['Adjudicated Clock Score'].apply(lamb
scores3 = df.shape
```

Now, we will create the Clock_List data frame that contains the SIDs of the Clock Drawing Test Images with the corresponding Adjusted Clock Score from the Excel file.

In [13]:

```
# Reading the image files from the source folder
path = r'C:/Users/12563/Documents/Test Clocks 2/'
files = os.listdir(path)


# The code below extracts the patient_id from the image names and match
# add it to the file name (format : score_patientID)
scores = []
name_sel = []

nummatch = 0
numdiscard = 0

for i in files:

    # Depending on the image file names different split patterns are nee
    f_name_prim = os.path.basename(i).split('.')[0].split(' ')[0].split(
#     f_name_prim = os.path.basename(i).split('.')[0].split(' ')[0].spli
#     f_name_prim = os.path.basename(i).split(' ')[1].split('.')[0]
#     f_name_prim = os.path.basename(i).split(' ')[1].split('-')[1].spli
#     f_name_prim = os.path.basename(i).split(' ')[1].split('#')[1].spli
#     f_name_prim = os.path.basename(i).split(' ')[1].split('.')[0]
#     f_name_prim = os.path.basename(i).split(' ')[0].split('_')[1]
#     f_name_prim = os.path.basename(i).split(' ')[1].split('.')[0]
#     f_name_prim = os.path.basename(i).split('_')[1].split('_')[0]

    for k in range(df.shape[0]):
        if f_name_prim  == df.iloc[k,0]:
            nummatch += 1
            scores.append(df.iloc[k,1])
            name_sel.append(f_name_prim )

#             os.rename(os.path.join(path, i), os.path.join(path, ''.joi
        else:
            numdiscard += 1


Clock_list = pd.DataFrame([name_sel, scores], index = ['Clocks', 'Scores
Clock_list = Clock_list.T

# saving the results in the target folder
Clock_list.to_csv(path + 'Clock_list.csv')
Clock_list
```

Out[13]:

	Clocks	Scores
0	10071D	0
1	10098X	0
2	10111P	0
3	10286Y	0
4	10292T	0

	Clocks	Scores
...	...	...
538	30352X	2
539	30357H	2
540	30361Y	2
541	30363C	2
542	30368M	2

543 rows × 2 columns

Furthermore, we will create a new data frame that will merge the Clock_List Data Frame with the excel file that contains the assoicated FEV1 Score, GOLD Stage, & COPD Score for 370 of the the SIDs in which we have the Clock Drawing Test image & the associated score.

In [18]:
```
Dataset_2 = Clock_list.merge(Dataset_1,on = 'Clocks')
Dataset_2
```

Out[18]:

	Clocks	Scores	fev1pp_final	goldstage_final_V2	COPD_P3	Score
0	10071D	0	42.7	2	1	0
1	10098X	0	120.3	2	0	0
2	10111P	0	19.6	2	1	0
3	10286Y	0	77.7	2	0	0
4	10292T	0	35.6	2	1	0
...	...	...	...	...	...	...
365	23619U	2	83.6	2	0	2
366	23634Q	2	73.9	1	0	2
367	23696M	2	72.4	2	1	2
368	23785L	2	51.7	2	1	2
369	24637B	2	119.4	2	0	2

370 rows × 6 columns

We will now perform Statistical Analysis on Dataset 2, this Data Frame created by Merging Dataset 1 & Clock_List.

```
We begin the statistical Analysis by performing the Chi-Squared Test for Independence on the Adjusted Clock Drawing Test Score by the Self-Reported COPD Score
```

In [20]:

```
# Chi-Square Test

df_use_2_copd = pd.DataFrame(Dataset_2, columns = ['Score', 'COPD_P3'])
df_use_2_copd
```

Out[20]:

	Score	COPD_P3
0	0	1
1	0	0
2	0	1
3	0	0
4	0	1
...	...	...
365	2	0
366	2	0
367	2	1
368	2	1
369	2	0

370 rows × 2 columns

In [22]:

```
# create contingency table
data_crosstab_2_copd = pd.crosstab(df_use_2_copd['Score'],
                                   df_use_2_copd['COPD_P3'],
                                  )

data_crosstab_2_copd
```

Out[22]:

COPD_P3	0	1
Score		
0	31	26
2	218	95

In [23]:

```
# Chi-Squared Test

res_2 = stat()
res_2.chisq(df=data_crosstab_2_copd)
# output
print(res_2.summary)
```

```
Chi-squared test for independence

Test              Df    Chi-square    P-value
--------------  ----  ------------  ---------
Pearson            1       4.43385  0.0352329
Log-likelihood     1       4.25976  0.0390255
```

We will now observe the Expected Frequency Counts for the Chi-Squared Test for Independence executed on the Adjusted Clock Drawing Test Score by the Self-Reported COPD Score variable.

In [24]:

```
print(res_2.expected_df)

# Need at least 5, preferably 15
# Assumptions met.
```

```
Expected frequency counts

           0         1
--  --------  --------
 0   38.3595   18.6405
 1  210.641   102.359
```

Chi-Squared Test for Independence on the Adjusted Clock Drawing Test Score by the Final GOLD Baseline Variable

In [28]:

```
# Chi-Square Test

df_use_2_gold = pd.DataFrame(Dataset_2, columns = ['Score', 'goldstage_f
df_use_2_gold
```

Out[28]:

	Score	goldstage_final_V2
0	0	2
1	0	2
2	0	2
3	0	2
4	0	2
...	...	...
365	2	2
366	2	1
367	2	2
368	2	2
369	2	2

370 rows × 2 columns

In [30]:

```
# create contingency table
data_crosstab_2_gold = pd.crosstab(df_use_2_gold['Score'],
                                   df_use_2_gold['goldstage_final_V2'],
                                  )

data_crosstab_2_gold
```

Out[30]:

goldstage_final_V2	0	1	2
Score			
0	20	6	31
2	141	37	135

In [31]:

```
# Chi-Squared Test

res_2_gold = stat()
res_2_gold.chisq(df=data_crosstab_2_gold)
# output
print(res_2_gold.summary)
```

```
Chi-squared test for independence

Test              Df    Chi-square    P-value
--------------  ----  ------------  ---------
Pearson            2       2.53034   0.282192
Log-likelihood     2       2.52584   0.282827
```

```
Finally, we will perform the K-Nearest Neighbors Algorithm to solve the
classification problem of a patients Clock Drawing Test Score.
```

In [38]:

```
df_knn_2 = pd.DataFrame(Dataset_2, columns = ['fev1pp_final', 'goldstage_
df_knn_2
```

Out[38]:

	fev1pp_final	goldstage_final_V2	COPD_P3	Score
0	42.7	2	1	0
1	120.3	2	0	0
2	19.6	2	1	0
3	77.7	2	0	0
4	35.6	2	1	0
...	...	...	...	...
365	83.6	2	0	2
366	73.9	1	0	2
367	72.4	2	1	2
368	51.7	2	1	2
369	119.4	2	0	2

370 rows × 4 columns

In [45]:

```
X_2 = df_knn_2.drop("Score", axis = 1)
X_2 = X_2.values
y_2 = df_knn_2['Score']
y_2 = y_2.values
```

In [46]:

```
rain and test sets
_2_score, y_train_2_score, y_test_2_score = train_test_split(X_2, y_2, test_s
```

In [47]:
```
# KNN is a supervised machine learning method
# Trying to predict weather or not someone will have COPD based
# on their CDT Score (the predictor)
knn_model_2_score = KNeighborsRegressor(n_neighbors=2)
knn_model_2_score.fit(X_train_2_score, y_train_2_score)
```

Out[47]:
```
KNeighborsRegressor

KNeighborsRegressor(n_neighbors=2)
```

In [48]:
```
# Calculate RMSE for Training Data
train_preds_2_score = knn_model_2_score.predict(X_train_2_score)
mse_2_score = mean_squared_error(y_train_2_score, train_preds_2_score)
rmse_2_score = sqrt(mse_2_score)
rmse_2_score
```

Out[48]: 0.48629876257963733

In [49]:
```
# Calculate RMSE for Testing Data
test_preds_2_test_s = knn_model_2_score.predict(X_test_2_score)
mse_2_test_s = mean_squared_error(y_test_2_score, test_preds_2_test_s)
rmse_2_test_s = sqrt(mse_2_test_s)
rmse_2_test_s
```

Out[49]: 0.86991767240168

In [50]:
```
# Calculate Accurary
y_pred_2_s = knn_model_2_score.predict(X_test_2_score)

# Evaluate the training & testing Accuracy
#
print('Training accuracy score: %.3f' % knn_model_2_score.score(X_train_
print('Test accuracy score: %.3f' % knn_model_2_score.score(X_test_2_sco
```

```
Training accuracy score: 0.541
Test accuracy score: -0.392
```

```
Analysis of the Factor Variables on the Adjusted Clock Drawing Test Score
```

Now, we will pixelate the 565 Clock Drawing Test images

In [57]:

```
# pixalating PDF clock images and convering them to data matrix

# Reading in Images with an Adjusted Clock Drawing Test Score
# of 0

path = r'C:/Users/12563/Documents/Test Clocks 2/'
#path = r'C:/Users/12563/Documents/Clocks_0/'
files = os.listdir(path)
dmat1 = np.zeros((len(files), 288, 288))

for index, file in enumerate(files):
    f_name_prim = os.path.basename(file)
    print(f_name_prim)
    print(path)
    print(path+f_name_prim)

    _, ext = os.path.splitext(file)
    if ext.lower() == '.pdf':
        # pdf processing
        #print(file)
        images = convert_from_path(path + f_name_prim, poppler_path = r"C
        image = np.array(images[0])
        y,x,_ = image.shape

        # cropping the clocks from the images
        #print(image.shape)
         # saving the clocks in the target folder
        #fig, ax = plt.subplots()
        #plt.title('File name: {}, Index: {}'.format(file, index))
        #ax.imshow(image, cmap='gray')
        #Cropping
        # I decided on the cropping bounds by looking at printed images
        image = image[350:1250,350:1350]
        #print(image.shape)
         # saving the clocks in the target folder
        fig, ax = plt.subplots()
        plt.title('File name: {}, Index: {}'.format(file, index))
        ax.imshow(image, cmap='gray')
        print()

        # Grayscaling the images
        image = np.mean(image, axis = 2)

        # resizing the cropped clocks
        image_resized = resize(image, (288, 288))

        dmat1[index] = image_resized

        # saving the clocks into the target folder
        fig, ax = plt.subplots()
        plt.title('File name: {}, Index: {}'.format(file, index))
        ax.imshow(image_resized, cmap='gray')
        #fig.savefig(path + f_name_prim + '.jpeg')

        # saving the datamatrix in the target folder
        #np.save(path + 'dmat2', dmat2)
```

```
57
58      elif ext.lower() == '.jpeg':
59          # jpeg processing
60
61
62          image = imread(path + f_name_prim)
63
64          # cropping the clocks from the images
65          image = image[550:2400, 550:2400]
66
67          # Grayscaling the images
68          image = np.mean(image, axis = 2)
69
70          # resizing the cropped clocks
71          image_resized = resize(image, (288, 288))
72
73          dmat1[index] = image_resized
74
75          # saving the clocks in the target folder
76          fig, ax = plt.subplots()
77          plt.title('File name: {}, Index: {}'.format(file, index))
78          ax.imshow(image, cmap='gray')
79
80          #fig.savefig(path + f_name_prim)
81      #else:
82          #print(file, ext)
83
84          #if file == "JPEG":
85              #continue
86          #print(file)
87          #images = convert_from_path(path + f_name_prim, poppler_path = r'
88          #image = np.array(images[0])
89          #y,x,_ = image.shape
```

```
10071D.PDF
C:/Users/12563/Documents/Test Clocks 2/
C:/Users/12563/Documents/Test Clocks 2/10071D.PDF

10098X.PDF
C:/Users/12563/Documents/Test Clocks 2/
C:/Users/12563/Documents/Test Clocks 2/10098X.PDF

10111P.PDF
C:/Users/12563/Documents/Test Clocks 2/
C:/Users/12563/Documents/Test Clocks 2/10111P.PDF

10286Y.jpeg
C:/Users/12563/Documents/Test Clocks 2/
C:/Users/12563/Documents/Test Clocks 2/10286Y.jpeg
10292T.PDF
C:/Users/12563/Documents/Test Clocks 2/
C:/Users/12563/Documents/Test Clocks 2/10292T.PDF
```

Now, we will rotate the images in need of rotation as well as delete images that need to be deleted.

In [62]:

```
# This code removes the noise and rotates the images

# Images that need rotation (Indexes via printing in the cell above)
S_rot = [31, 216, 445, 554]

# Noise images that will be removed, Clock not shown on images
S_noise = [4, 12, 14, 15, 27,33, 35, 41, 44, 47, 51, 52, 93, 103, 125, 1
           251, 259, 270, 271, 274, 278, 281, 285, 287, 288, 293, 296, 3
          449, 450, 451, 467, 468, 526, 559, 561]

#S_rot = []
#S_noise= [4, 12, 13, 34, 46, 65]
# function for rotating the JPG images

def img_rotate (df, rotate) :
    for i in rotate:
        df[i] = np.rot90(df[i],3)
    return(df)

dmat1 = img_rotate (dmat1, S_rot)


# removing noise images
dmat1_nonoise = np.delete(dmat1, S_noise , axis = 0)
```

In [63]:

```
# Loading X(clock matrix) and Y (labels) data (already suffeled and noise
X = np.load('/Users/12563/Desktop/Batch 1 (2)/Xall_nonoi_shfl_use_0.npy'
y = np.load('/Users/12563/Desktop/Batch 1 (2)/Yall_nonoi_shfl_use_0.npy'
```

In [64]:

```
# Creating array of the files to create
files = np.array(files)

files_nonoise = np.delete(files, S_noise, axis = 0)
```

Now, we are going to create a data frame that consists of the Subject ID's in which we have succefully pixeled their Clock Drawing Test Image.

In [65]:

```
# Generating the List of scores associated with the files_nonoise
score_list = np.zeros(files_nonoise.shape[0])

blank = []

# for loop to match score of df_join to Pixelated clock images
for indices in range(files_nonoise.shape[0]):
    ext, _ = os.path.splitext(files_nonoise[indices])
    #print(ext)
    if ext in Clock_list['Clocks'].tolist():
        score_update = Clock_list[Clock_list['Clocks'] == ext]['Scores']
        print(score_update)
        score_list[indices] = score_update
    else:
        blank.append(indices)

# Cleaned Image Data
dmat1_delete = np.delete(dmat1_nonoise, blank, axis = 0)

# Cleaned Score Data
Scores_delete = np.delete(score_list, blank, axis = 0)
```

```
0
0
0
0
2
2
2
0
2
2
2
0
0
2
0
2
2
2
2
```

In [87]:

```
# Deleting the SIDs of the files that were delted
files_delete = np.delete(files_nonoise, blank, axis = 0)

# Convert the NumPy Array to Pandas data Frame
files_delete = pd.DataFrame(files_delete)

# Rename column in a dataframe
files_delete.columns = ['Clocks']
```

In [88]:

```
# We need to get rid of the .PDF, .pdf, & .jpeg
# Strip .pdf
files_delete['Clocks'] = files_delete['Clocks'].str.rstrip('.pdf')
# Strip .PDF
files_delete['Clocks'] = files_delete['Clocks'].str.rstrip('.PDF')
# Strip .jpeg
files_delete['Clocks'] = files_delete['Clocks'].str.rstrip('.jpeg')

files_delete
```

Out[88]:

	Clocks
0	10071
1	10098X
2	10111
3	10286Y
4	10294X
...	...
480	30352X
481	30357H
482	30361Y
483	30363C
484	30368M

485 rows × 1 columns

In [86]:

```
files_delete
```

In [92]:

```
# Rename column in a dataframe
df_factors.rename(columns = {'sid':'Clocks' }, inplace = True)

```

In [93]:

```
df = files_delete.merge(df_factors,on = 'Clocks')
df
```

Out[93]:

	Clocks	F1compP2	F2compP2	F3compP2	F4compP2
0	10098X	0.396002	0.868078	0.451179	-0.729966
1	10286Y	0.792556	-0.630638	0.904748	-0.785195
2	10294X	1.988744	0.705892	1.891007	0.835416
3	10348U	-0.819512	-0.414752	-0.379585	-2.070531
4	10528W	1.043932	-0.255540	0.788534	0.838686
...	...	...	...	...	...
325	24160C	-0.985130	-0.566078	-1.393219	-0.892061
326	24196X	-1.222794	-1.192649	-0.664929	-0.932192
327	24205Y	0.067990	-2.257025	-0.196353	-1.515259
328	24555Z	0.356886	-0.348367	0.187666	0.204181
329	25340J	-0.577903	-1.390396	-0.710814	0.085333

330 rows × 5 columns

In [94]:

```
dataset_factors_use = df.merge(Clock_list,on = 'Clocks')
dataset_factors_use
```

Out[94]:

	Clocks	F1compP2	F2compP2	F3compP2	F4compP2	Scores
0	10098X	0.396002	0.868078	0.451179	-0.729966	0
1	10286Y	0.792556	-0.630638	0.904748	-0.785195	0
2	10294X	1.988744	0.705892	1.891007	0.835416	2
3	10348U	-0.819512	-0.414752	-0.379585	-2.070531	2
4	10528W	1.043932	-0.255540	0.788534	0.838686	2
...	...	...	...	...	...	...
328	24160C	-0.985130	-0.566078	-1.393219	-0.892061	2
329	24196X	-1.222794	-1.192649	-0.664929	-0.932192	2
330	24205Y	0.067990	-2.257025	-0.196353	-1.515259	2
331	24555Z	0.356886	-0.348367	0.187666	0.204181	2
332	25340J	-0.577903	-1.390396	-0.710814	0.085333	2

333 rows × 6 columns

In [97]:
```
df_box = df.drop('Clocks', axis = 1)
df_box
```

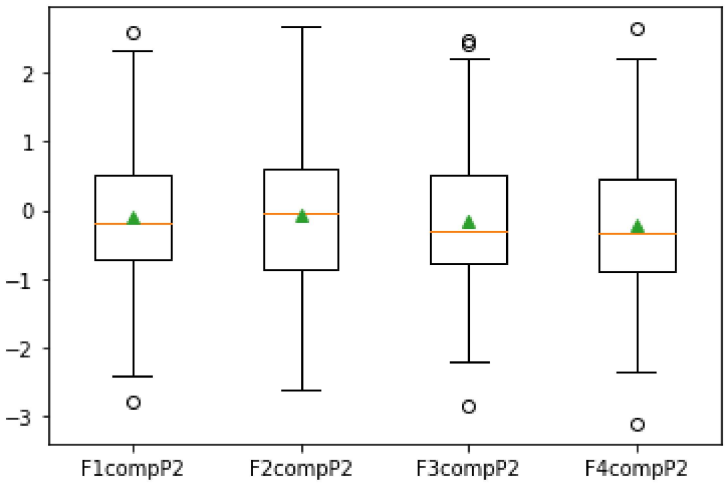
Out[97]:

	F1compP2	F2compP2	F3compP2	F4compP2
0	0.396002	0.868078	0.451179	-0.729966
1	0.792556	-0.630638	0.904748	-0.785195
2	1.988744	0.705892	1.891007	0.835416
3	-0.819512	-0.414752	-0.379585	-2.070531
4	1.043932	-0.255540	0.788534	0.838686
...	...	...	...	...
325	-0.985130	-0.566078	-1.393219	-0.892061
326	-1.222794	-1.192649	-0.664929	-0.932192
327	0.067990	-2.257025	-0.196353	-1.515259
328	0.356886	-0.348367	0.187666	0.204181
329	-0.577903	-1.390396	-0.710814	0.085333

330 rows × 4 columns

In [98]:
```
# Creating plot
plt.boxplot(df_box, showmeans = True)
plt.xticks([1, 2, 3, 4], ['F1compP2', 'F2compP2', 'F3compP2', 'F4compP2'
```

Out[98]:
```
([<matplotlib.axis.XTick at 0x234daf6a370>,
  <matplotlib.axis.XTick at 0x234daf6a400>,
  <matplotlib.axis.XTick at 0x234cd9cefd0>,
  <matplotlib.axis.XTick at 0x234b8acec70>],
 [Text(1, 0, 'F1compP2'),
  Text(2, 0, 'F2compP2'),
  Text(3, 0, 'F3compP2'),
  Text(4, 0, 'F4compP2')])
```



In [99]:
```
# T-test Factor 1
sc.stats.ttest_ind(dataset_factors_use['Scores'], dataset_factors_use['F
```

Out[99]:
```
Ttest_indResult(statistic=17.992744029470185, pvalue=2.9525947380683684e-59)
```

In [100]:
```
# T-test Factor 2
sc.stats.ttest_ind(dataset_factors_use['Scores'], dataset_factors_use['F
```

Out[100]:
```
Ttest_indResult(statistic=16.599459858946588, pvalue=5.094635135350925e-52)
```

In [101]:
```
# T-test Factor 3
sc.stats.ttest_ind(dataset_factors_use['Scores'], dataset_factors_use['F
```

Out[101]:
```
Ttest_indResult(statistic=18.668544585336004, pvalue=7.720186926657116e-63)
```

In [102]:
```
# T-test Factor 4
sc.stats.ttest_ind(dataset_factors_use['Scores'], dataset_factors_use['F
```

Out[102]:
```
Ttest_indResult(statistic=18.664412707937306, pvalue=8.121916090616809e-63)
```

In [103]:
```
df_fac_adj = pd.read_csv('C:/Users/12563/Documents/Res. Methods in Math &
df_fac_adj

```

Out[103]:

	sid	age_final	gender_final	race_final	education_finalV1	income_final	kidney_dise
0	10015T	69.9	1	1	3	5	
1	10017X	77.6	1	1	3	4	
2	10031R	71.7	1	1	3	2	
3	10032T	71.1	2	1	3	2	
4	10049K	60.0	1	2	1	1	
...	...	...	...	...	...	...	
1712	23634Q	66.1	2	1	1	3	
1713	23696M	68.7	2	1	3	2	
1714	23785L	66.9	1	1	3	1	
1715	23793K	55.8	2	1	3	5	
1716	24637B	68.1	1	1	3	2	

1717 rows × 8 columns

In [106]:
```
df_fac_adj = df_fac_adj.rename(columns = {'sid' : 'Clocks'})
```

In [109]:
```
df_fac_adj_ana = dataset_factors_use.merge(df_fac_adj,on = 'Clocks')
df_fac_adj_ana
```

Out[109]:

	Clocks	F1compP2	F2compP2	F3compP2	F4compP2	Scores	age_final	gender_final
0	10098X	0.396002	0.868078	0.451179	-0.729966	0	88.3	2
1	10286Y	0.792556	-0.630638	0.904748	-0.785195	0	59.6	2
2	10348U	-0.819512	-0.414752	-0.379585	-2.070531	2	84.1	1
3	10528W	1.043932	-0.255540	0.788534	0.838686	2	85.7	1
4	10647E	1.361600	0.373160	1.250930	0.534924	2	77.8	2
...	...	...	...	...	...	...	...	...
223	23467X	-0.456152	-1.794621	-0.599533	-0.607022	2	62.9	2
224	23478C	-0.905084	1.815225	-0.993261	-0.841087	2	59.8	2
225	23619U	0.723385	0.281395	0.804737	-0.071033	2	62.5	2
226	23634Q	-0.533470	-0.131728	-0.553965	0.532302	2	66.1	2
227	23785L	0.771995	1.026923	1.211564	1.812055	2	66.9	1

228 rows × 13 columns

In [112]:
```
B1 = pd.read_csv('C:/Users/12563/Documents/Res. Methods in Math & Stats/
B1 = B1.rename(columns = {'sid' : 'Clocks'})
```

In [113]:
```
df_b = B1.merge(df_fac_adj_ana,on = 'Clocks')
df_b
```

Out[113]:

compP2	F3compP2	F4compP2	Scores	age_final	gender_final	race_final	education_finalV1	ir
.868078	0.451179	-0.729966	0	88.3	2	1	3	
.630638	0.904748	-0.785195	0	59.6	2	2	3	
.414752	-0.379585	-2.070531	2	84.1	1	1	3	
.255540	0.788534	0.838686	2	85.7	1	1	3	
.373160	1.250930	0.534924	2	77.8	2	1	3	
...	...	...	...	...	...	...	...	
.794621	-0.599533	-0.607022	2	62.9	2	1	3	
.815225	-0.993261	-0.841087	2	59.8	2	1	3	
.281395	0.804737	-0.071033	2	62.5	2	1	3	
.131728	-0.553965	0.532302	2	66.1	2	1	1	
.026923	1.211564	1.812055	2	66.9	1	1	3	

In [115]:

```
# F1 ANOVA
# w/o GOLD
F1_anova_model = ols('F1compP2 ~ C(Scores) + age_final + C(gender_final)
anova_table_F1 = sm.stats.anova_lm(F1_anova_model, typ=1)
anova_table_F1
```

Out[115]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Scores)	1.0	7.081691	7.081691	10.137368	0.001668
C(gender_final)	1.0	2.616250	2.616250	3.745135	0.054273
C(race_final)	1.0	1.154399	1.154399	1.652510	0.200001
C(education_finalV1)	2.0	3.000253	1.500126	2.147416	0.119284
C(income_final)	5.0	3.659889	0.731978	1.047819	0.390533
C(kidney_disease_final)	1.0	3.185581	3.185581	4.560126	0.033856
age_final	1.0	12.580973	12.580973	18.009533	0.000033
Residual	215.0	150.193184	0.698573	NaN	NaN

In [118]:

```
# F2 ANOVA
F2_anova_model = ols('F2compP2 ~ C(Scores) + age_final + C(gender_final)
anova_table_F2 = sm.stats.anova_lm(F2_anova_model, typ=1)
anova_table_F2
```

Out[118]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Scores)	1.0	1.254987	1.254987	1.488475	0.223790
C(gender_final)	1.0	1.714421	1.714421	2.033386	0.155328
C(race_final)	1.0	0.337690	0.337690	0.400517	0.527495
C(education_finalV1)	2.0	2.649832	1.324916	1.571414	0.210128
C(income_final)	5.0	8.449333	1.689867	2.004263	0.079221
C(kidney_disease_final)	1.0	0.012088	0.012088	0.014338	0.904801
age_final	1.0	1.270923	1.270923	1.507375	0.220883
Residual	215.0	181.274321	0.843136	NaN	NaN

In [120]:

```
# F3 ANOVA
# Gas-Trapping
F3_anova_model = ols('F3compP2 ~ C(Scores) + age_final + C(gender_final)
anova_table_F3 = sm.stats.anova_lm(F3_anova_model, typ=1)
anova_table_F3
```

Out[120]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Scores)	1.0	7.432433	7.432433	10.702248	0.001246
C(gender_final)	1.0	0.700826	0.700826	1.009146	0.316237
C(race_final)	1.0	0.000024	0.000024	0.000034	0.995331
C(education_finalV1)	2.0	6.052799	3.026399	4.357830	0.013956
C(income_final)	5.0	4.369009	0.873802	1.258221	0.283208
C(kidney_disease_final)	1.0	3.222713	3.222713	4.640509	0.032338
age_final	1.0	16.417315	16.417315	23.639926	0.000002
Residual	215.0	149.311918	0.694474	NaN	NaN

In [122]:

```
# F4 ANOVA
# TLC FRC
F4_anova_model = ols('F4compP2 ~ C(Scores) + age_final + C(gender_final)
anova_table_F4 = sm.stats.anova_lm(F4_anova_model, typ=1)
anova_table_F4
```

Out[122]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Scores)	1.0	3.506831	3.506831	4.191373	0.041846
C(gender_final)	1.0	0.002536	0.002536	0.003031	0.956145
C(race_final)	1.0	0.264772	0.264772	0.316456	0.574331
C(education_finalV1)	2.0	14.658253	7.329127	8.759791	0.000220
C(income_final)	5.0	6.207951	1.241590	1.483952	0.196206
C(kidney_disease_final)	1.0	4.268622	4.268622	5.101867	0.024903
age_final	1.0	6.710542	6.710542	8.020457	0.005065
Residual	215.0	179.885834	0.836678	NaN	NaN

In [123]:

```
# pixalating PDF clock images and convering them to data matrix

# Reading in Images with an Adjusted Clock Drawing Test Score
# of 0

#path = r'C:/Users/12563/Documents/Test Clocks 2/'
path = r'C:/Users/12563/Documents/Clocks_0/'
files = os.listdir(path)
dmat1 = np.zeros((len(files), 288, 288))

for index, file in enumerate(files):
    f_name_prim = os.path.basename(file)
    print(f_name_prim)
    print(path)
    print(path+f_name_prim)

    _, ext = os.path.splitext(file)
    if ext.lower() == '.pdf':
        # pdf processing
        #print(file)
        images = convert_from_path(path + f_name_prim, poppler_path = r"(
        image = np.array(images[0])
        y,x,_ = image.shape

        # cropping the clocks from the images
        #print(image.shape)
         # saving the clocks in the target folder
        #fig, ax = plt.subplots()
        #plt.title('File name: {}, Index: {}'.format(file, index))
        #ax.imshow(image, cmap='gray')
        #Cropping
        # I decided on the cropping bounds by looking at printed images
        image = image[350:1250,350:1350]
        #print(image.shape)
         # saving the clocks in the target folder
        fig, ax = plt.subplots()
        plt.title('File name: {}, Index: {}'.format(file, index))
        ax.imshow(image, cmap='gray')
        print()

        # Grayscaling the images
        image = np.mean(image, axis = 2)

        # resizing the cropped clocks
        image_resized = resize(image, (288, 288))

        dmat1[index] = image_resized

        # saving the clocks into the target folder
        fig, ax = plt.subplots()
        plt.title('File name: {}, Index: {}'.format(file, index))
        ax.imshow(image_resized, cmap='gray')
        #fig.savefig(path + f_name_prim + '.jpeg')

        # saving the datamatrix in the target folder
        #np.save(path + 'dmat2', dmat2)
```

```
57
58      elif ext.lower() == '.jpeg':
59          # jpeg processing
60
61
62          image = imread(path + f_name_prim)
63
64          # cropping the clocks from the images
65          image = image[550:2400, 550:2400]
66
67          # Grayscaling the images
68          image = np.mean(image, axis = 2)
69
70          # resizing the cropped clocks
71          image_resized = resize(image, (288, 288))
72
73          dmat1[index] = image_resized
74
75          # saving the clocks in the target folder
76          fig, ax = plt.subplots()
77          plt.title('File name: {}, Index: {}'.format(file, index))
78          ax.imshow(image, cmap='gray')
79
80          #fig.savefig(path + f_name_prim)
81      #else:
82          #print(file, ext)
83
84          #if file == "JPEG":
85              #continue
86          #print(file)
87          #images = convert_from_path(path + f_name_prim, poppler_path = r
88          #image = np.array(images[0])
89          #y,x,_ = image.shape
```

In [124]:

```
# This code removes the noise and rotates the images

# Images that need rotation (Indexes via printing in the cell above)
#S_rot = [31, 216, 445, 554]

# Noise images that will be removed, Clock not shown on images
#S_noise = [4, 12, 14, 15, 27,33, 35, 41, 44, 47, 51, 52, 93, 103, 125, 1
            #251, 259, 270, 271, 274, 278, 281, 285, 287, 288, 293, 296, 1
           #449, 450, 451, 467, 468, 526, 559, 561]

S_rot = []
S_noise= [4, 12, 13, 34, 46, 65]
# function for rotating the JPG images

def img_rotate (df, rotate) :
    for i in rotate:
        df[i] = np.rot90(df[i],3)
    return(df)

dmat1 = img_rotate (dmat1, S_rot)


# removing noise images
dmat1_nonoise = np.delete(dmat1, S_noise , axis = 0)
```

In [126]:

```
# Loading X(clock matrix) and Y (labels) data (already suffeled and noise
X = np.load('/Users/12563/Desktop/Batch 1 (2)/Xall_nonoi_shfl_use_0.npy'
y = np.load('/Users/12563/Desktop/Batch 1 (2)/Yall_nonoi_shfl_use_0.npy'
```

In [140]:

```
# Add an extra Dimension for Channel for CNN
X = X.reshape([len(X), 288, 288, 1])

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2

print (len(y_train[y_train == 0]) , len(y_train[y_train == 1]))
print (len(y_test[y_test == 0]) , len(y_test[y_test == 1]))

# Scaling train and test datasets
X_train = X_train/255
X_test = X_test/255
```

```
52 0
13 0
```

In [141]:

```
# Function for invidual CNN models (2 stuctures: Conv_Pool and Conv_Pool_
def model(activation = 'relu'):
    model = Sequential()
    # Input Image with dimensions (288x288x1) an input image
    # 288 pixels wide by 288 pixels in height with 1 channel
    model.add(Conv2D(32, kernel_size=(3,3), activation= activation , inpu
    model.add(MaxPooling2D(pool_size=(2,2),name="maxpool_1"))
#     model.add(Dropout(0.25))
#     model.add(Conv2D(64, kernel_size=(3, 3), activation= activation ,ne
#     model.add(MaxPooling2D(pool_size=(2,2),name="maxpool_2"))
#     model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(32,activation="relu", name="dense_layer_1"))
    model.add(Dense(2,activation="sigmoid"))
    model.compile(loss="sparse_categorical_crossentropy", metrics=['accu
    return(model)

conv_model =  model(activation = 'relu')
conv_hist = conv_model.fit(X_train,y_train ,epochs= 20,validation_data=(
conv_hist_dict = conv_hist.history
```

```
Epoch 1/20
2/2 [==============================] - 2s 591ms/step - loss: 0.4427 - accur
acy: 0.6346 - val_loss: 0.0030 - val_accuracy: 1.0000
Epoch 2/20
2/2 [==============================] - 1s 519ms/step - loss: 7.7510e-04 - a
ccuracy: 1.0000 - val_loss: 4.4656e-06 - val_accuracy: 1.0000
Epoch 3/20
2/2 [==============================] - 1s 449ms/step - loss: 1.1508e-06 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/20
2/2 [==============================] - 1s 446ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/20
2/2 [==============================] - 1s 511ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/20
2/2 [==============================] - 1s 526ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/20
2/2 [==============================] - 1s 576ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/20
2/2 [==============================] - 1s 548ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/20
2/2 [==============================] - 1s 560ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/20
2/2 [==============================] - 1s 513ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 11/20
2/2 [==============================] - 1s 483ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 12/20
2/2 [==============================] - 1s 482ms/step - loss: 0.0000e+00 - a
```

```
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 13/20
2/2 [==============================] - 1s 505ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 14/20
2/2 [==============================] - 1s 537ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/20
2/2 [==============================] - 1s 479ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 16/20
2/2 [==============================] - 1s 470ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 17/20
2/2 [==============================] - 1s 550ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 18/20
2/2 [==============================] - 1s 480ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 19/20
2/2 [==============================] - 1s 464ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 20/20
2/2 [==============================] - 1s 478ms/step - loss: 0.0000e+00 - a
ccuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

In [144]:

```python
import tensorflow as tf
feature_extractor = tf.keras.Model(
    inputs=conv_model.inputs,
    outputs=[
              conv_model.output,  # < last layer output
              conv_model.layers[2].output # < your convolution layer outpu
    ]
)

model_output, conv_layer_output = feature_extractor(X)
conv_layer_output
```

Out[144]:

```
<tf.Tensor: shape=(65, 654368), dtype=float32, numpy=
array([[  0.      , 113.47755 ,  40.749527, ...,   0.      , 186.66112 ,
          0.      ],
       [  0.      ,  34.348255,  18.957294, ...,   0.      , 158.27054 ,
          0.      ],
       [  0.      , 113.02469 ,  40.945564, ...,   0.      , 185.92625 ,
          0.      ],
       ...,
       [  0.      , 113.92434 ,  40.90993 , ...,   0.      , 187.39598 ,
          0.      ],
       [  0.      , 113.92434 ,  40.90993 , ...,   0.      , 187.39598 ,
          0.      ],
       [  0.      , 110.74818 ,  39.708916, ...,   0.      , 183.49847 ,
          0.      ]], dtype=float32)>
```

In [145]:

```
# Perform K-Means on Conv-Layer Output
model = KMeans(n_clusters=2, random_state=42)
kmeans = model.fit(conv_layer_output)
```

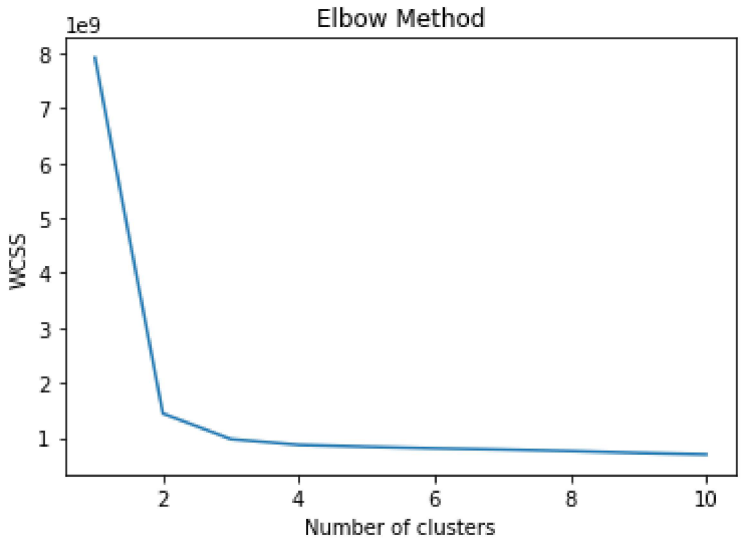
```
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [134]:

```python
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init
    kmeans.fit(conv_layer_output)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

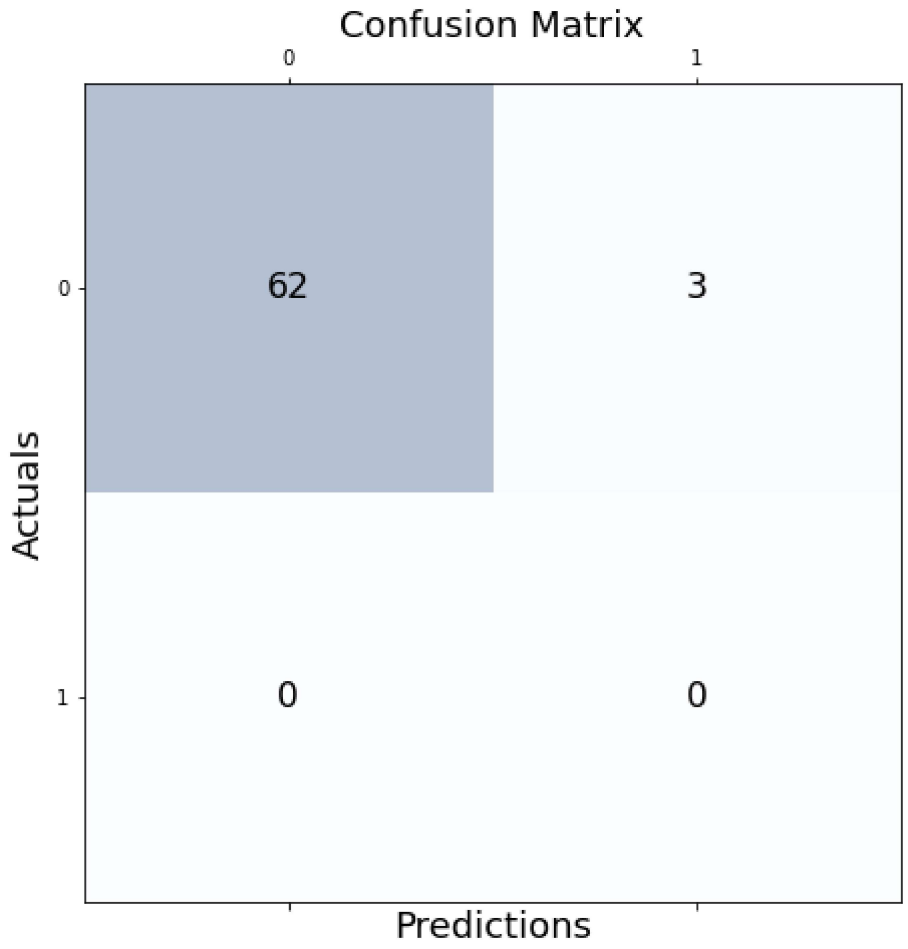
```
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kme
ans.py:1334: UserWarning: KMeans is known to have a memory leak on Window
s with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

```
C:\Users\12563\OneDrive\New folder\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [146]:

```
from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y, kmeans.labels_)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center',
                ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



In [135]:

```
from more_itertools import locate
list_to_check = kmeans.labels_
def find_indices(list_to_check, item_to_find):
    indices = locate(list_to_check,lambda x: x == item_to_find)
    return list(indices)

indices_of_intrest = find_indices(list_to_check, 1)
indices_of_intrest
```

Out[135]: [12, 23, 36, 50, 57]

In [136]:
```
files_nonoise
files_nonoise[indices_of_intrest]
```

Out[136]:
```
array(['10973R.PDF', '11715A.PDF', '12719N.PDF', '13529N.PDF',
       '13840N.pdf'], dtype='<U46')
```

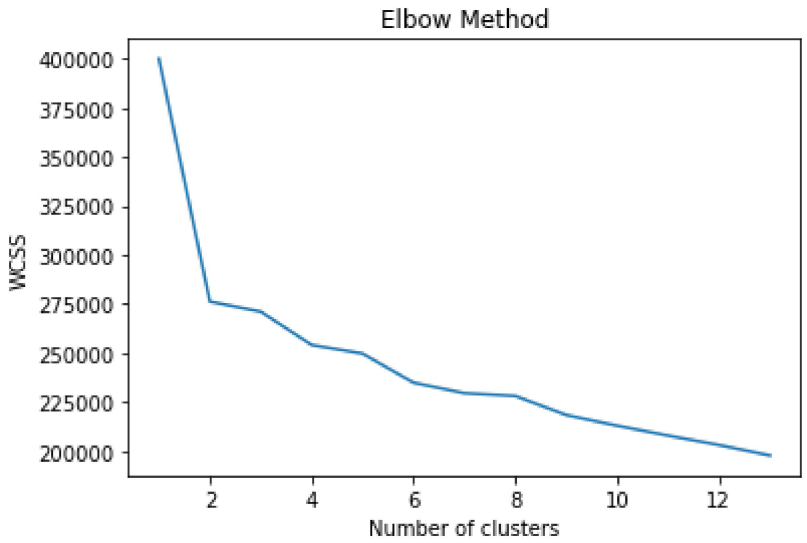
In [137]:
```
# The Lowest SSE Value
kmeans.inertia_
```

Out[137]:
```
694187557.1665838
```

K-Medoids

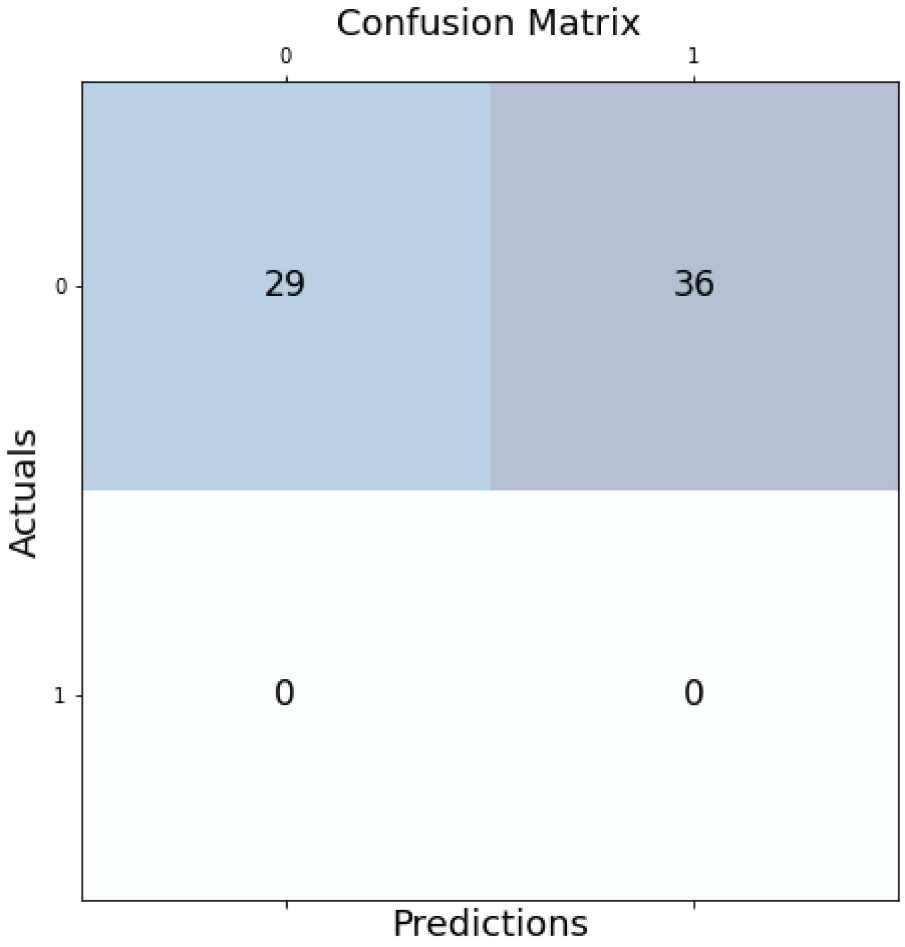
In [148]:
```
# 2-Clusters

kmedoids = KMedoids(n_clusters=2, random_state=0).fit(conv_layer_output)
```

In [149]:
```
wcss = []
for i in range(1, 14):
    kmed = KMedoids(n_clusters=i, init='k-medoids++', max_iter=300, rand
    kmed.fit(conv_layer_output)
    wcss.append(kmed.inertia_)
plt.plot(range(1, 14), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [150]:

```
from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y, kmedoids.labels_)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center',
                ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



In [151]:

```
# The Lowest SSE Value
# A good model is one with low inertia AND a small number
# of clusters

kmedoids.inertia_
```

Out[151]: 393572.34

In [158]:

```
df_ = pd.read_csv("C:/Users/12563/Documents/Res. Methods in Math & Stats/
df_
```

Out[158]:

	sid	COPD_P3
0	10015T	1
1	10017X	1
2	10031R	0
3	10032T	1
4	10049K	1
...	...	...
1712	23634Q	0
1713	23696M	1
1714	23785L	1
1715	23793K	0
1716	24637B	0

1717 rows × 2 columns

In [159]:

```
from more_itertools import locate
list_to_check = kmedoids.labels_
def find_indices(list_to_check, item_to_find):
    indices = locate(list_to_check,lambda x: x == item_to_find)
    return list(indices)

indices_of_intrest = find_indices(list_to_check, 0)
indices_of_intrest
```

Out[159]:

```
[0,
 3,
 7,
 11,
 13,
 16,
 17,
 18,
 21,
 22,
 25,
 27,
 28,
 29,
 31,
 33,
 36,
 38,
 44,
 46
```

In [163]:

```
df_p = pd.DataFrame(files_nonoise[indices_of_intrest], columns = ['sid'])

# We need to get rid of the .PDF, .pdf, & .jpeg
# Strip .pdf
# Remove Leading and Treailing spaces
df_p['sid'] = df_p['sid'].str.strip('.pdf')
# Remove Leading and Treailing spaces
df_p['sid'] = df_p['sid'].str.strip('.PDF')
# Remove Leading and Treailing spaces
df_p['sid'] = df_p['sid'].str.strip('.jpeg')
df_p
#email_
```

Out[163]:

	sid
0	10071
1	10286Y
2	10482Y
3	10796V
4	11001N
5	11239U
6	11240
7	11246R
8	11292Y
9	11308N
10	11724B
11	11750C
12	11944
13	12258
14	12330L
15	12426Y
16	12719N
17	12857Z
18	13215Q
19	13334Y
20	13484R
21	13529N
22	13610W
23	13666X
24	13765Z
25	13848D
26	13937C

	sid
27	13954C
28	14006M

In [165]:
```
# Obtaining the Clock Drawing images obtaing a label of 1
indices_of_intrest_1 = find_indices(list_to_check, 1)
indices_of_intrest_1
```

Out[165]:
```
[1,
 2,
 4,
 5,
 6,
 8,
 9,
 10,
 12,
 14,
 15,
 19,
 20,
 23,
 24,
 26,
 30,
 32,
 34,
 35,
 37,
 39,
 40,
 41,
 42,
 43,
 45,
 47,
 48,
 51,
 53,
 56,
 57,
 59,
 62,
 63]
```

In [166]:

```
df_p_1 = pd.DataFrame(files_nonoise[indices_of_intrest_1], columns = ['s

# We need to get rid of the .PDF, .pdf, & .jpeg
# Strip .pdf
# Remove Leading and Treailing spaces
df_p_1['sid'] = df_p_1['sid'].str.strip('.pdf')
# Remove Leading and Treailing spaces
df_p_1['sid'] = df_p_1['sid'].str.strip('.PDF')
# Remove Leading and Treailing spaces
df_p_1['sid'] = df_p_1['sid'].str.strip('.jpeg')
df_p_1
#email_
```

Out[166]:

	sid
0	10098X
1	10111
2	10294X
3	10348U
4	10473X
5	10528W
6	10630N
7	10647E
8	10973R
9	11161J
10	11185X
11	11251K
12	11252M
13	11715A
14	11716C
15	11734E
16	12281Y
17	12343U
18	12581K
19	12635H
20	12721A
21	12862S
22	12885E
23	13083Z
24	13105J
25	13118S

	sid
26	13328
27	13378S
28	13384N
29	13550E
30	13629R
31	13824
32	13840N
33	13853W
34	13997U
35	14005K

In [168]:
```
1 fac = pd.read_csv("C:/Users/12563/Documents/Res. Methods in Math & Stats/
```

In [169]:
```
1 # Obtaining Average Value of Factor Scores in Cluster A
2 df_fac = fac.merge(df_p, on = 'sid')
3 df_fac
4
5 df_fac.mean()
```

```
C:\Users\12563\AppData\Local\Temp/ipykernel_19464/2776651171.py:5: FutureWa
rning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_
only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  df_fac.mean()
```

Out[169]:
```
F1compP2    0.234492
F2compP2    0.096026
F3compP2    0.331235
F4compP2    0.365538
dtype: float64
```

In [171]:
```
1 # Obtaining Average Value of Factor Scores in Cluster B
2 df_fac_1 = fac.merge(df_p_1, on = 'sid')
3 df_fac_1
4
5 df_fac_1.mean()
```

```
C:\Users\12563\AppData\Local\Temp/ipykernel_19464/4284279653.py:5: FutureWa
rning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_
only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  df_fac_1.mean()
```

Out[171]:
```
F1compP2   -0.210619
F2compP2    0.417673
F3compP2   -0.250752
F4compP2   -0.190061
dtype: float64
```

In [172]:
```
gold = pd.read_csv('C:/Users/12563/Documents/Res. Methods in Math & Stat
```

In [175]:
```
# Obtaining Percentage of people with FINAL Golde Baseline 1-4 Cluster A
df_gold = gold.merge(df_p, on = 'sid')
df_gold

(np.sum(df_gold['goldstage_final_V2'] != 0)/29) *100
```

Out[175]: 41.37931034482759

In [181]:
```
# Obtaining Percentage of people with Self-Reported COPD Cluster A
df_copd = gold.merge(df_p, on = 'sid')
df_copd

(np.sum(df_copd['COPD_P3'] == 1)/29) *100
```

Out[181]: 24.137931034482758

In [174]:
```
# Obtaining Percentage of people with FINAL Golde Baseline 1-4 Cluster B
df_gold_1 = gold.merge(df_p_1, on = 'sid')
df_gold_1

(np.sum(df_gold_1['goldstage_final_V2'] != 0)/36) *100
```

Out[174]: 36.11111111111111

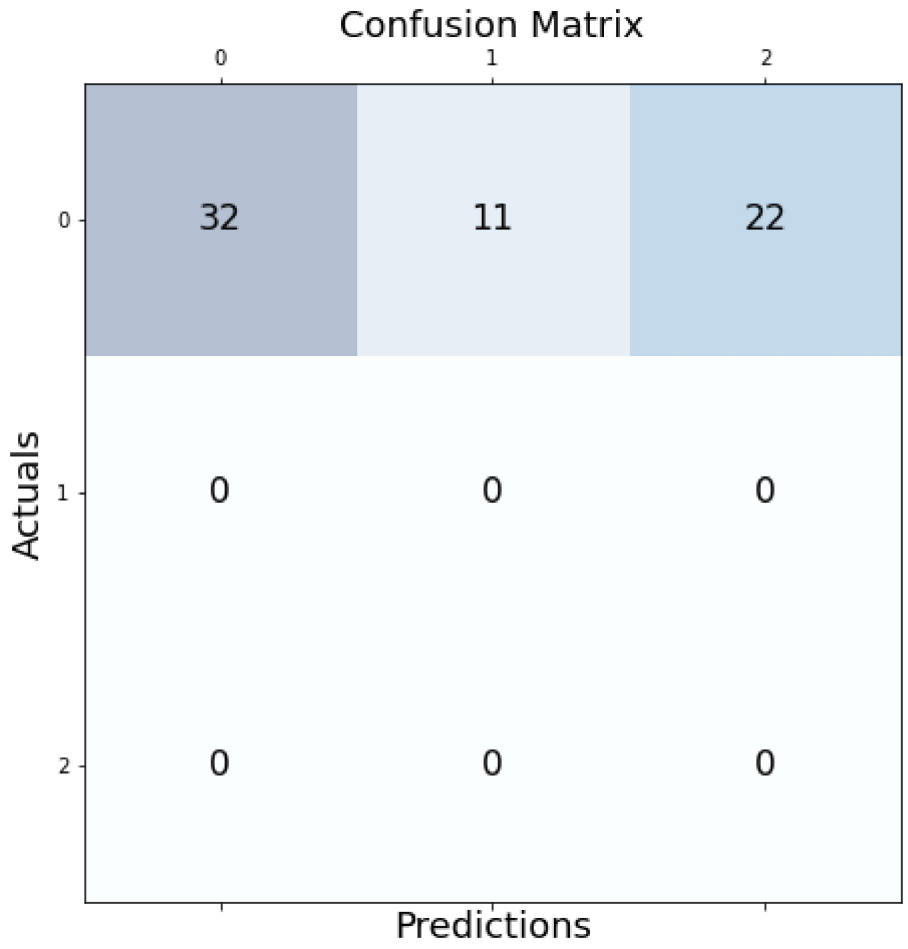
In [180]:
```
# Obtaining Percentage of people with Self-Reported COPD Cluster A
df_copd_1 = gold.merge(df_p_1, on = 'sid')
df_copd_1

(np.sum(df_copd_1['COPD_P3'] == 1)/29) *100
```

Out[180]: 10.344827586206897

In [152]:
```
# 3 Clusters
kmedoids_3 = KMedoids(n_clusters=3, random_state=0).fit(conv_layer_outpu
```

In [153]:

```
from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y, kmedoids_3.labels_)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center',
                ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



In [154]:

```
# The Lowest SSE Value
# A good model is one with low inertia AND a small number
# of clusters

kmedoids_3.inertia_
```

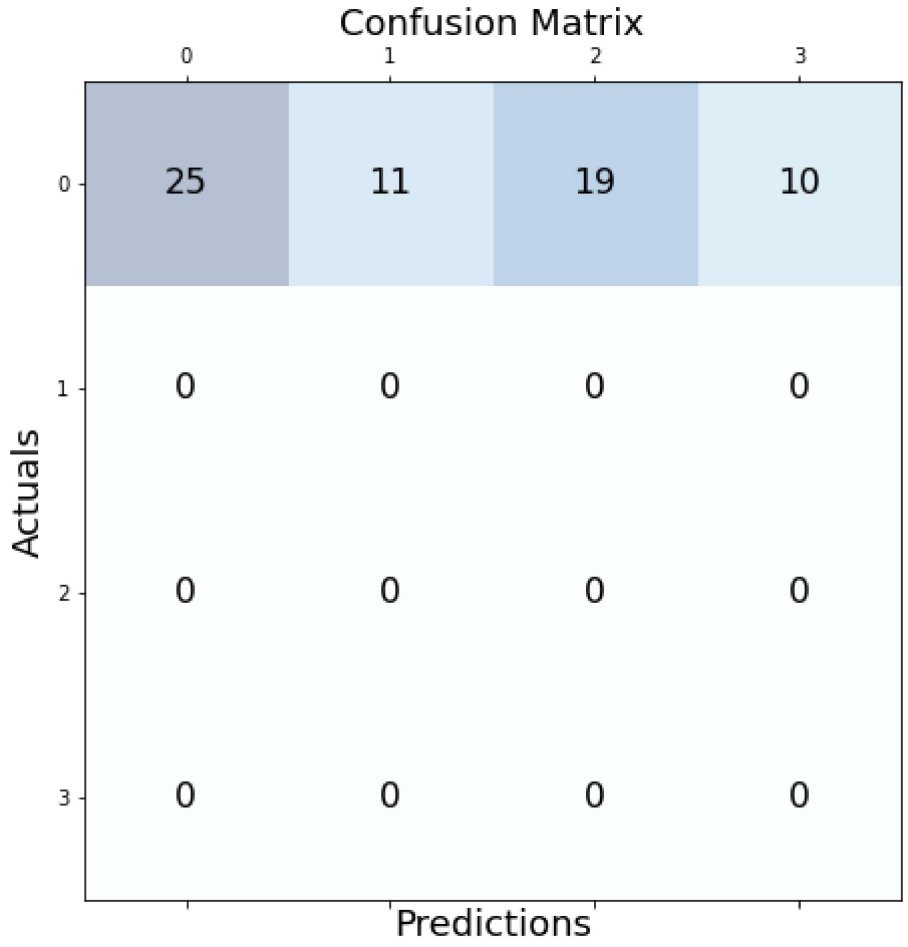
Out[154]: 387474.84

In [155]:

```
# 4 Clusters

kmedoids_4 = KMedoids(n_clusters=4, random_state=0).fit(conv_layer_outpu
```

In [156]:

```
from sklearn.metrics import confusion_matrix
conf_matrix=confusion_matrix(y, kmedoids_4.labels_)

fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center',
                ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



In [157]:

```
# The Lowest SSE Value
# A good model is one with low inertia AND a small number
# of clusters

kmedoids_4.inertia_
```

Out[157]: 383652.62

In [ ]:

```

```