Assignment #1
Face Recognition

Seif El Din Wael Salem 18010832
Ziad Hassan Mahmoud 18010720
Mohamed Mostafa Badran 18011621

Generate the Data Matrix and the Label vector:

```
df = pd.DataFrame()
for s in range(1,41):
    for i in range(1,11):
        img = PIL.Image.open("/kaggle/input//att-database-of-faces/s{}/{}.pgm".format(s,i))
        img = img.getdata()
        img = np.array(img)
        df = df.append({'data': img, 'label': s }, ignore_index=True)
        # print(img)
df['label'] = (df['label']).apply(np.int64)
```

```
                                                  data  label
0    [48, 49, 45, 47, 49, 57, 39, 42, 53, 49, 53, 6...      1
1    [60, 60, 62, 53, 48, 51, 61, 60, 71, 68, 78, 7...      1
2    [39, 44, 53, 37, 61, 48, 61, 45, 35, 40, 40, 3...      1
3    [63, 53, 35, 36, 33, 34, 31, 35, 39, 43, 56, 7...      1
4    [64, 76, 80, 53, 34, 72, 60, 66, 66, 50, 47, 4...      1
..                                                 ...    ...
395  [123, 121, 126, 122, 127, 127, 123, 124, 123, ...     40
396  [129, 127, 133, 124, 131, 129, 130, 129, 127, ...     40
397  [125, 119, 124, 125, 124, 121, 123, 125, 123, ...     40
398  [119, 120, 120, 118, 120, 121, 121, 116, 120, ...     40
399  [125, 124, 124, 126, 123, 125, 127, 123, 124, ...     40
```

Split the Dataset into Training and Test sets:

```
test_data = df.iloc[::2]
train_data = df.iloc[1::2]
print(test_data)
print(train_data)
```

```
                                                  data  label
0    [48, 49, 45, 47, 49, 57, 39, 42, 53, 49, 53, 6...      1
2    [39, 44, 53, 37, 61, 48, 61, 45, 35, 40, 40, 3...      1
4    [64, 76, 80, 53, 34, 72, 60, 66, 66, 50, 47, 4...      1
6    [41, 47, 47, 46, 44, 49, 48, 58, 61, 49, 58, 6...      1
8    [42, 41, 44, 46, 48, 39, 37, 37, 33, 37, 53, 5...      1
..                                                 ...    ...
390  [119, 121, 123, 121, 120, 123, 121, 120, 123, ...     40
392  [131, 125, 126, 131, 125, 129, 125, 127, 127, ...     40
394  [128, 125, 125, 129, 128, 132, 125, 133, 125, ...     40
396  [129, 127, 133, 124, 131, 129, 130, 129, 127, ...     40
398  [119, 120, 120, 118, 120, 121, 121, 116, 120, ...     40

[200 rows x 2 columns]
                                                  data  label
1    [60, 60, 62, 53, 48, 51, 61, 60, 71, 68, 78, 7...      1
3    [63, 53, 35, 36, 33, 34, 31, 35, 39, 43, 56, 7...      1
5    [43, 50, 41, 58, 78, 83, 67, 48, 44, 46, 62, 7...      1
7    [44, 43, 32, 32, 30, 30, 38, 40, 48, 66, 54, 4...      1
9    [34, 34, 33, 32, 38, 40, 39, 49, 54, 57, 65, 7...      1
..                                                 ...    ...
391  [127, 131, 128, 129, 127, 128, 127, 128, 128, ...     40
393  [130, 123, 127, 125, 126, 126, 127, 125, 125, ...     40
395  [123, 121, 126, 122, 127, 127, 123, 124, 123, ...     40
397  [125, 119, 124, 125, 124, 121, 123, 125, 123, ...     40
399  [125, 124, 124, 126, 123, 125, 127, 123, 124, ...     40

[200 rows x 2 columns]
```

Classification using PCA:

a. Define the alpha = {0.8,0.85,0.9,0.95}

```
for alpha in np.arange(0.8,0.96,0.05):
    tot = 0
    k = 0
    while tot/sum(eignvalues) < alpha:
        tot += eignvalues[k]
        k += 1
    k

    x = (eignvectors[:,0])*z[10]
    for j in range(1,k):
        x += (eignvectors[:,j])*z[10]

    new_df = pd.DataFrame()
    for s in range(1,41):
        for i in range(1,11):
            x = (eignvectors[:,0])*(df.iloc[(s-1)*10 + i-1]['data']-mean)
            for j in range(1,k):
                x += (eignvectors[:,j])*(df.iloc[(s-1)*10 + i-1]['data']-mean)
            x = np.array(x)
            new_df = new_df.append({'data': x, 'label': s }, ignore_index=True)


    new_test_data = new_df.iloc[::2]
    new_train_data = new_df.iloc[1::2]

    knn = KNeighborsClassifier(n_neighbors=1)

    knn.fit(list(new_train_data['data']), list(new_train_data['label']))
    ac  = knn.score( list(new_test_data['data']), list(new_test_data['label']))
    acc = acc.append({'acc': ac, 'alpha': alpha  }, ignore_index=True)
print(acc)
```

b. Project the training set, and test sets separately using the same projection matrix.

```
new_df = pd.DataFrame()
for s in range(1,41):
    for i in range(1,11):
        x = (eignvectors[:,0])*(df.iloc[(s-1)*10 + i-1]['data']-mean)
        for j in range(1,k):
            x += (eignvectors[:,j])*(df.iloc[(s-1)*10 + i-1]['data']-mean)
        x = np.array(x)
        new_df = new_df.append({'data': x, 'label': s }, ignore_index=True)


new_test_data = new_df.iloc[::2]
new_train_data = new_df.iloc[1::2]
```

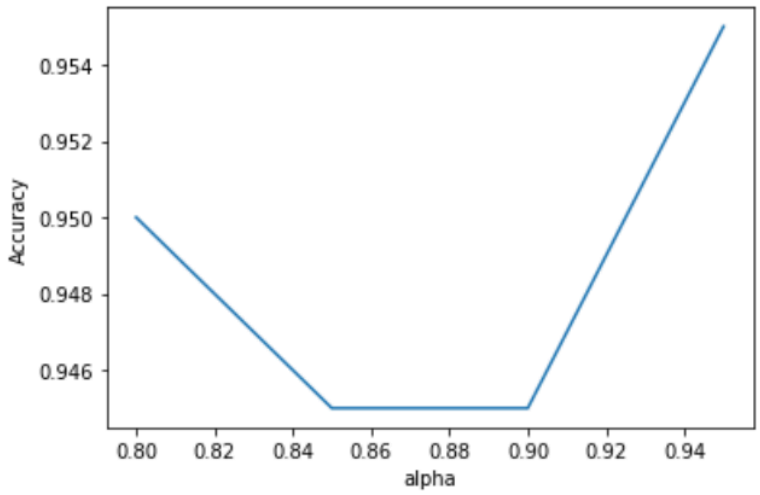
c. Use a simple classifier (first Nearest Neighbor to determine the class labels).

```
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(list(new_train_data['data']), list(new_train_data['label']))
ac  = knn.score( list(new_test_data['data']), list(new_test_data['label']))
acc = acc.append({'acc': ac, 'alpha': alpha  }, ignore_index=True)
```

d. Report Accuracy for every value of alpha separately.

```
    acc  alpha
0  0.950   0.80
1  0.945   0.85
2  0.945   0.90
3  0.955   0.95
Text(0, 0.5, 'Accuracy')
```



e. Can you find a relation between alpha and classification accuracy

-alpha is worst between 0.85 and 0.9

-better above 0.9 and below 0.85

-there is no linear relation that appears between alpha and accuracy

Classification Using LDA:

a. Use the pseudo code below for LDA.

```
ui = []
for i in range(1,41):
    data = (train_data[train_data['label']==i]['data']).to_numpy()
    data = np.stack(data)
    ui.append(np.mean(data, axis=0))

data = (train_data['data']).to_numpy()
data = np.stack(data)
u = np.mean(data, axis=0)
sb = 10 * np.outer((ui[0] - u) , (ui[0] - u ))
for i in range (1,40):
    sb += 10 * np.outer((ui[i] - u) , (ui[i] - u ))
data = (train_data[train_data['label']==1]['data']).to_numpy()
data = np.stack(data)
z =  data - ui[0]

for i in range(2,41):
    data = (train_data[train_data['label']==i]['data']).to_numpy()
    data = np.stack(data)
    z =  np.append(z, data - ui[i-1],axis = 0)
s = np.dot(np.transpose(z[0:5]),z[0:5])
for i in range(2,41):
    s += np.dot(np.transpose(z[(i-1)*5:i*5]),z[(i-1)*5:i*5])
si = np.linalg.inv(s)
sib = np.dot(si,sb)
eignvalues1,eignvectors1 = np.linalg.eigh(sib)
eignvalues1
u = eignvectors[:,:39]
new_df = pd.DataFrame()
for s in range(1,41):
    for i in range(1,11):
        x = np.dot(np.transpose(u),df['data'].iloc[(s-1)*10 + i-1])
        new_df = new_df.append({'data': x, 'label': s }, ignore_index=True)
```

b. Project the training set, and test sets separately using the same projection matrix U. You will have 39 dimensions in the new space.

```
u = eignvectors[:,:39]
new_df = pd.DataFrame()
for s in range(1,41):
    for i in range(1,11):
        x = np.dot(np.transpose(u),df['data'].iloc[(s-1)*10 + i-1])
        new_df = new_df.append({'data': x, 'label': s }, ignore_index=True)
```

c. Use a simple classifier (first Nearest Neighbor to determine the class labels).

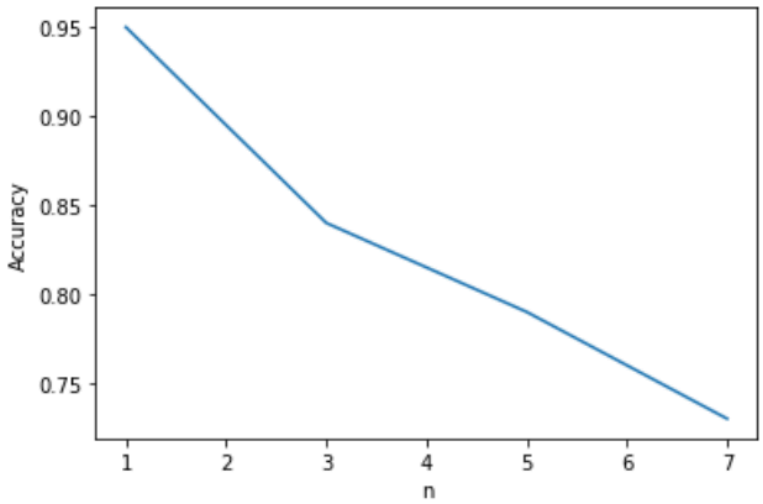
```
new_test_data = new_df.iloc[::2]
new_train_data = new_df.iloc[1::2]

knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(list(new_train_data['data']), list(new_train_data['label']))
print(knn.score( list(new_test_data['data']), list(new_test_data['label'])))
print(classification_report(list(new_test_data['label']), knn.predict( list(new_test_data['data']))))
```

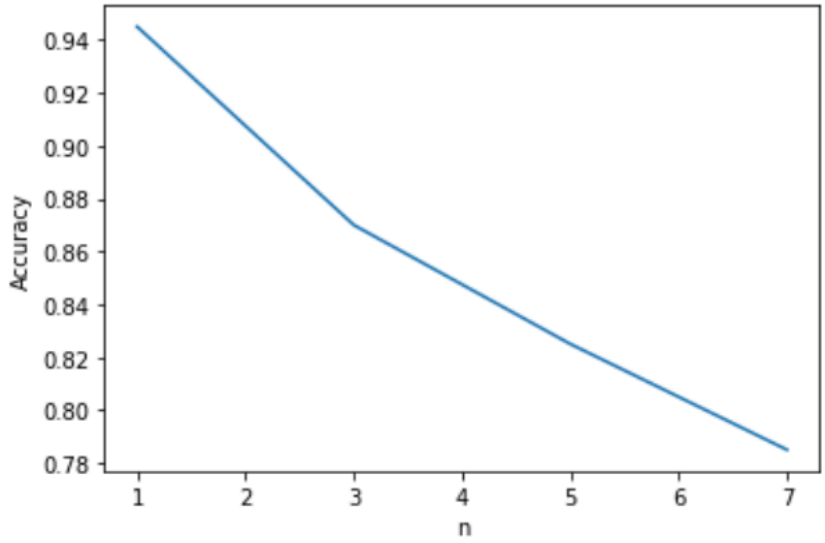
d. Report Accuracy for the Multiclass LDA on the face recognition dataset.

```
    acc    n
0  0.95  1.0
1  0.84  3.0
2  0.79  5.0
3  0.73  7.0
Text(0, 0.5, 'Accuracy')
```



e. Compare the results to PCA results.

```
     acc    n
0  0.945  1.0
1  0.870  3.0
2  0.825  5.0
3  0.785  7.0
Text(0, 0.5, 'Accuracy')
```



from the last 2 graphs we see LDA has higher accuracy than PCA in some values of k and some less accuracy

Classifier Tuning:

a. Set the number of neighbors in the K-NN classifier to 1,3,5,7.

```
acc = pd.DataFrame()

for n in range (1,9,2):

    knn = KNeighborsClassifier(n_neighbors=n)

    knn.fit(list(new_train_data['data']), list(new_train_data['label']))
    ac  = knn.score( list(new_test_data['data']), list(new_test_data['label']))
    acc = acc.append({'acc': ac, 'n': n  }, ignore_index=True)
print(acc)
```

b. Tie breaking at your preferred strategy.

at n=1 the best accuracy of both LDA and PCA

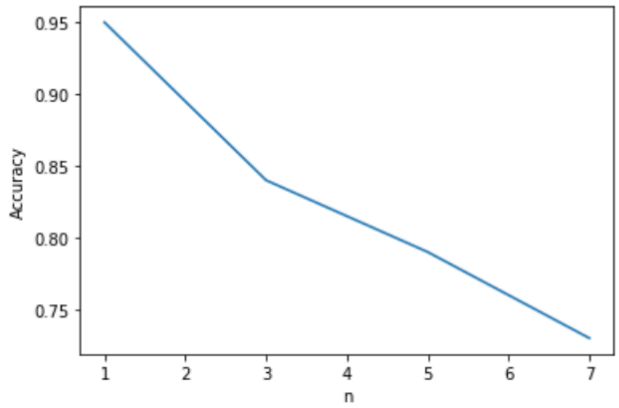
LDA is at 0.95 while PCA is at 0.945

therefore LDA is better

c. Plot the performance measure (accuracy) against the K value.

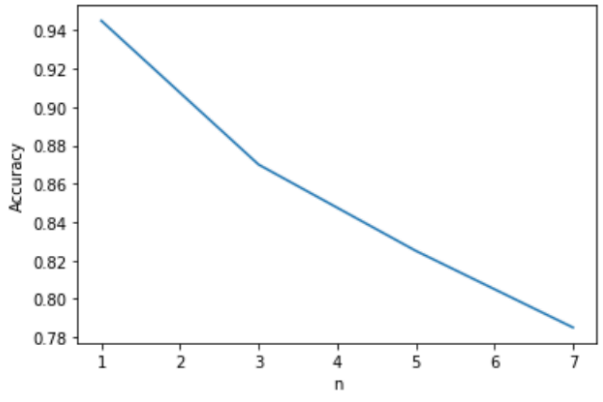
LDA

```
   acc    n
0  0.95  1.0
1  0.84  3.0
2  0.79  5.0
3  0.73  7.0
Text(0, 0.5, 'Accuracy')
```



PCA

```
    acc    n
0  0.945  1.0
1  0.870  3.0
2  0.825  5.0
3  0.785  7.0
Text(0, 0.5, 'Accuracy')
```



Compare vs Non-Face Images:

a. Download non-face images and make them of the same size 92x112.

and try to solve the classification problem faces vs. Non-faces.

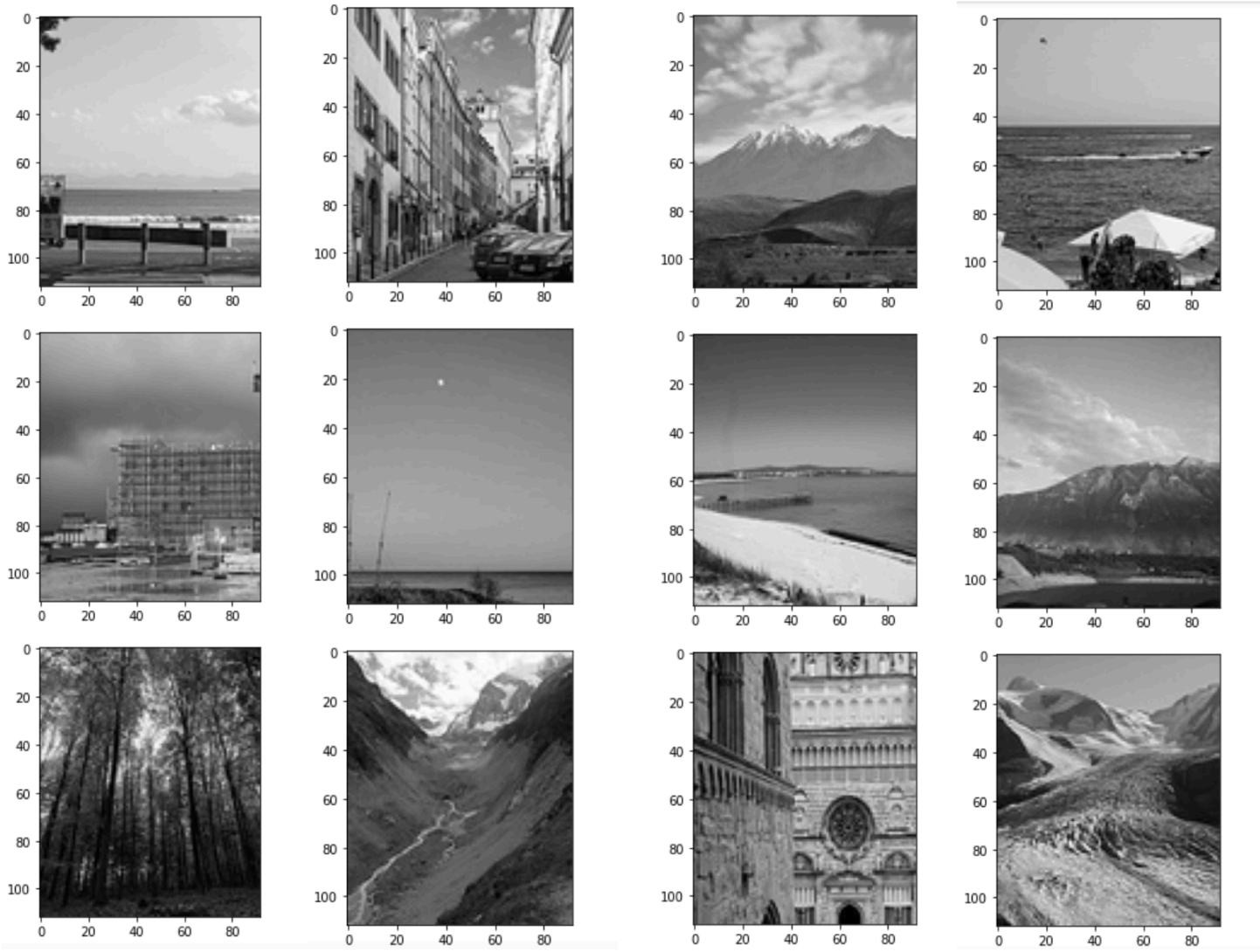
```
n_nonfaces = 400

for i in range(n_nonfaces):
    img = PIL.Image.open("../input/scene-classification/train-scene classification/train/{}.jpg".format(i))
    img = ImageOps.grayscale(img)
    img = img.resize((92,112), Image.ANTIALIAS)
    img = img.getdata()
    img = np.array(img)
    df2 = df2.append({'data': img, 'label': 1 }, ignore_index=True)
```

i. Show failure and success cases

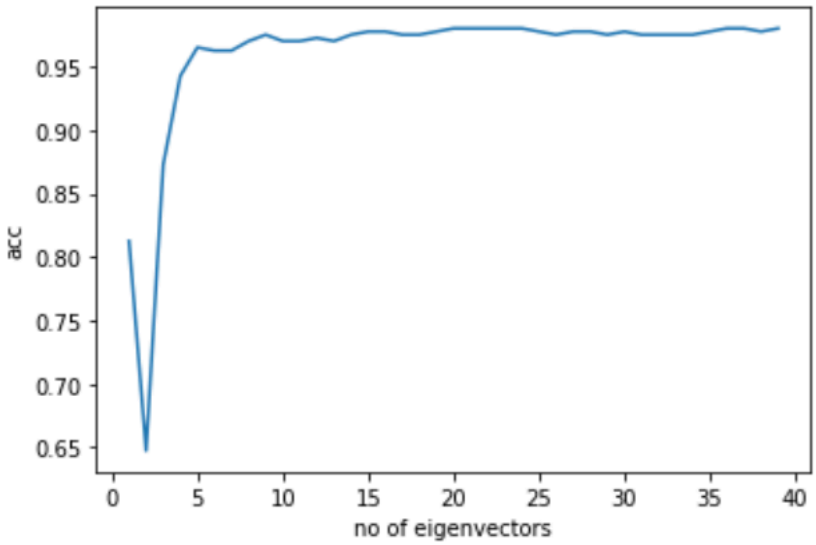
failure cases:

success cases:



ii. How many dominant eigenvectors will you use for the LDA solution?

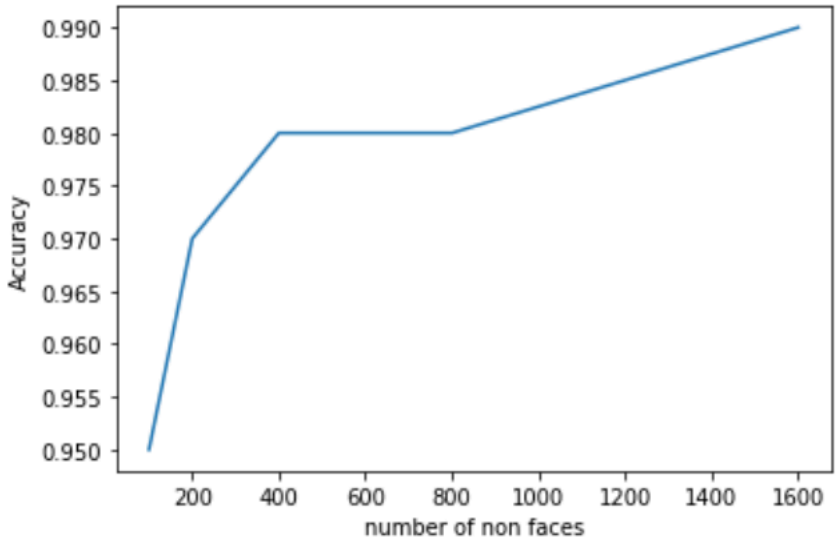
	acc	n
0	0.8125	1.0
1	0.6475	2.0
2	0.8725	3.0
3	0.9425	4.0
4	0.9650	5.0
5	0.9625	6.0
6	0.9625	7.0
7	0.9700	8.0
8	0.9750	9.0
9	0.9700	10.0
10	0.9700	11.0
11	0.9725	12.0
12	0.9700	13.0
13	0.9750	14.0
14	0.9775	15.0
15	0.9775	16.0
16	0.9750	17.0
17	0.9750	18.0
18	0.9775	19.0
19	0.9800	20.0
20	0.9800	21.0
21	0.9800	22.0
22	0.9800	23.0
23	0.9800	24.0
24	0.9775	25.0
25	0.9750	26.0
26	0.9775	27.0
27	0.9775	28.0
28	0.9750	29.0
29	0.9775	30.0
30	0.9750	31.0
31	0.9750	32.0
32	0.9750	33.0
33	0.9750	34.0
34	0.9775	35.0
35	0.9800	36.0
36	0.9800	37.0
37	0.9775	38.0
38	0.9800	39.0



from above graph we see best number of dominant eigenvectors is 10 where after it no improvement happen to accuracy

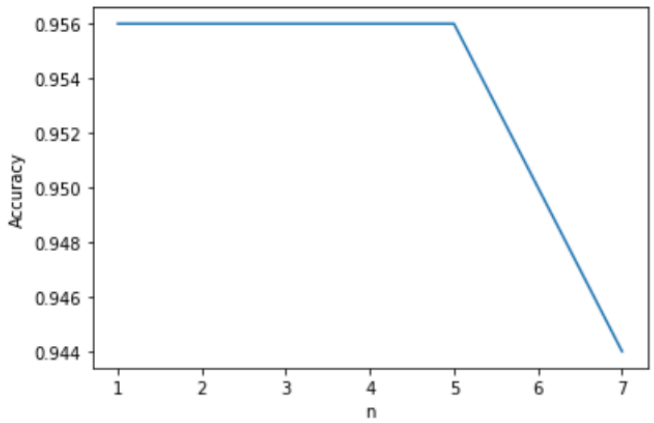
iii. Plot the accuracy vs the number of non-faces images while fixing the number of face images.

```
Text(0, 0.5, 'Accuracy')
```



for n = 100

```
   acc    n
0  0.956  1.0
1  0.956  3.0
2  0.956  5.0
3  0.944  7.0
Text(0, 0.5, 'Accuracy')
```



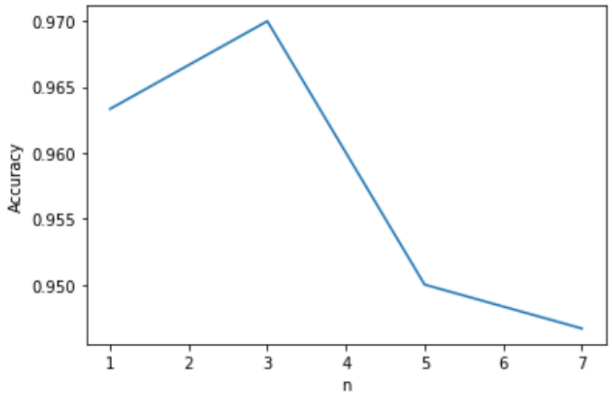

```
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       200
         1.0       1.00      0.78      0.88        50

    accuracy                           0.96       250
   macro avg       0.97      0.89      0.92       250
weighted avg       0.96      0.96      0.95       250
```

for n = 200

```
        acc    n
0  0.963333  1.0
1  0.970000  3.0
2  0.950000  5.0
3  0.946667  7.0
Text(0, 0.5, 'Accuracy')
```



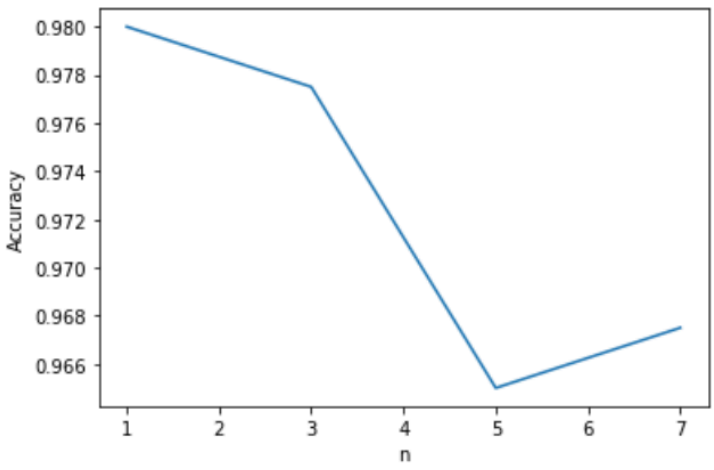

```
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       200
         1.0       1.00      0.91      0.95       100

    accuracy                           0.97       300
   macro avg       0.98      0.96      0.97       300
weighted avg       0.97      0.97      0.97       300
```

for n = 400

```
      acc    n
0  0.9800  1.0
1  0.9775  3.0
2  0.9650  5.0
3  0.9675  7.0
Text(0, 0.5, 'Accuracy')
```

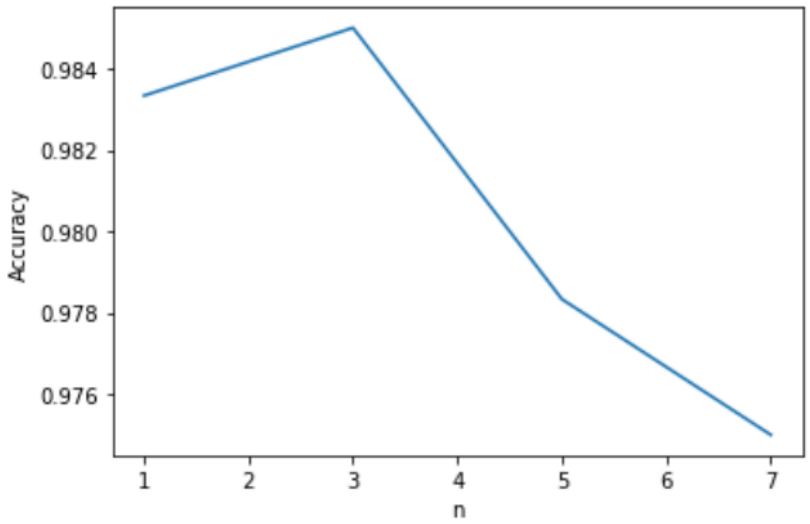


	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	200
1.0	1.00	0.96	0.98	200
accuracy			0.98	400
macro avg	0.98	0.98	0.98	400
weighted avg	0.98	0.98	0.98	400

for n = 800

	acc	n
0	0.983333	1.0
1	0.985000	3.0
2	0.978333	5.0
3	0.975000	7.0

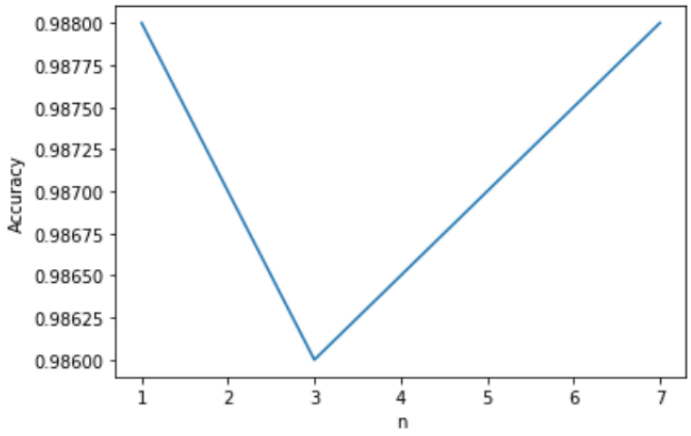
```
Text(0, 0.5, 'Accuracy')
```



	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	200
1.0	1.00	0.98	0.99	400
accuracy			0.98	600
macro avg	0.98	0.99	0.98	600
weighted avg	0.99	0.98	0.99	600

for n = 1600

```
     acc    n
0  0.988  1.0
1  0.986  3.0
2  0.987  5.0
3  0.988  7.0
Text(0, 0.5, 'Accuracy')
```



	precision	recall	f1-score	support
0.0	0.94	1.00	0.97	200
1.0	1.00	0.98	0.99	800
accuracy			0.99	1000
macro avg	0.97	0.99	0.98	1000
weighted avg	0.99	0.99	0.99	1000

iv. Criticize the accuracy measure for large numbers of non-faces images in the training data.

by increasing non faces number we increase overall accuracy of the model without affecting detecting faces accuracy.

this may be due to the large variance of non faces images that can exist unlike faces images which is all nearly the same, so by increasing non faces images we cover larger range of non faces images increasing overall accuracy .

Bonus:

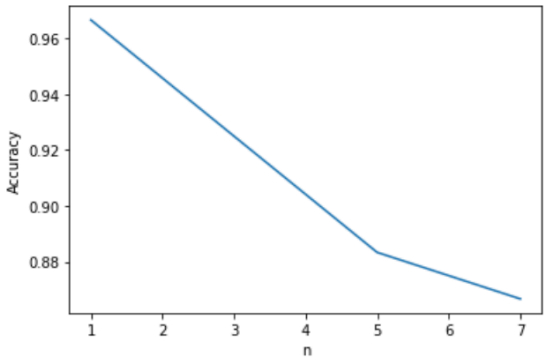
a. Use different Training and Test splits. Change the number of instances per subject to be 7 and keep 3 instances per subject for testing.

```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(df['data'], df['label'], test_size=0.3, stratify=df['label'],random_state=1)
```

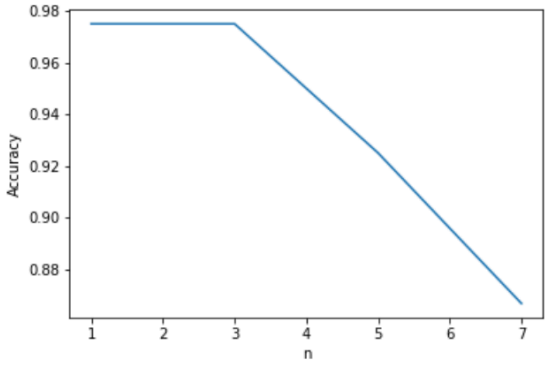
PCA

```
        acc    n
0  0.966667  1.0
1  0.925000  3.0
2  0.883333  5.0
3  0.866667  7.0
Text(0, 0.5, 'Accuracy')
```



LDA

```
        acc    n
0  0.975000  1.0
1  0.975000  3.0
2  0.925000  5.0
3  0.866667  7.0
Text(0, 0.5, 'Accuracy')
```



from the last graphs we see by splitting 7:3 instead of 5:5 for every subject we increase accuracy obtained from all different k values in both PCA and LDA.