Name: Dileep Naidu Maripi

1.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
file_path = '/content/drive/My Drive/allFaces.mat'
import scipy.io
data = scipy.io.loadmat(file_path)
```

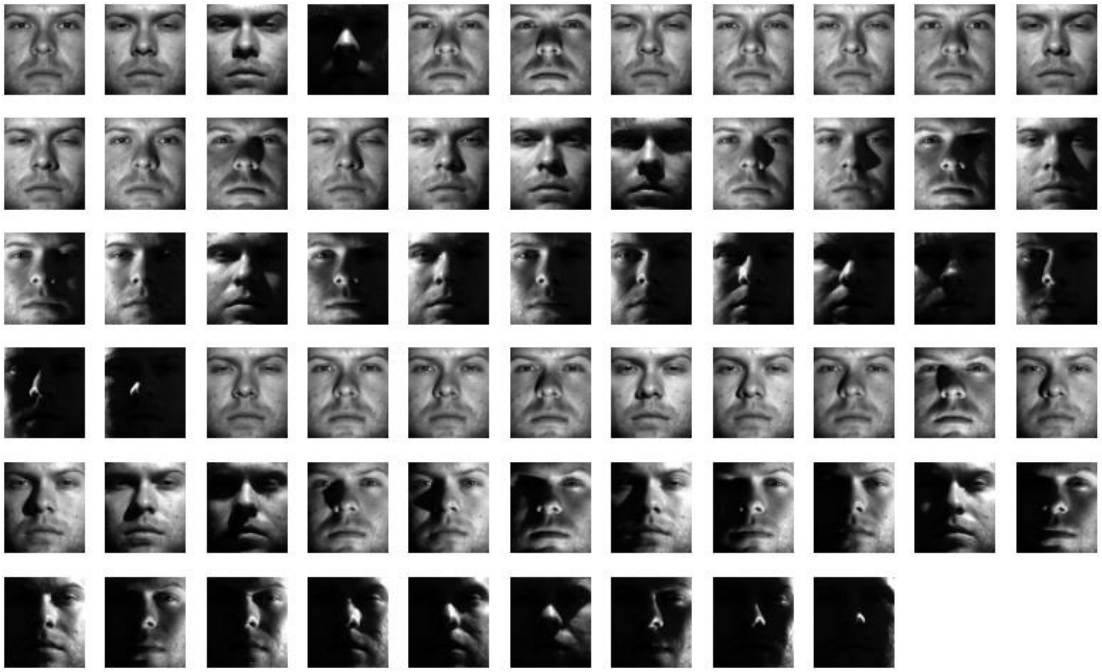
Part 1

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

faces = data['faces']
nfaces = data['nfaces']

n, m = 192, 168

# Plot to show 36 people
plt.figure(figsize=(8, 8))

for i in range(36):
    start_index = int(np.sum(nfaces[0, :i]))
    face_image = np.reshape(faces[:, start_index], (m, n)).T

    plt.subplot(6, 6, i+1)
    plt.imshow(face_image, cmap='gray')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

faces = data['faces']
nfaces = data['nfaces'][0]

for i in range(2):
    start_index = int(np.sum(nfaces[:i]))
    end_index = int(np.sum(nfaces[:i+1]))

    person_faces = faces[:, start_index:end_index]
    print("Images of Person ",i+1)
    # Plot each face
    plt.figure(figsize=(8,5))
```

```
    for j in range(nfaces[i]):
        face_image = np.reshape(person_faces[:, j], (m, n)).T
        plt.subplot(6,11, j+1)
        plt.imshow(face_image, cmap='gray')
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
Images of Person  1
```



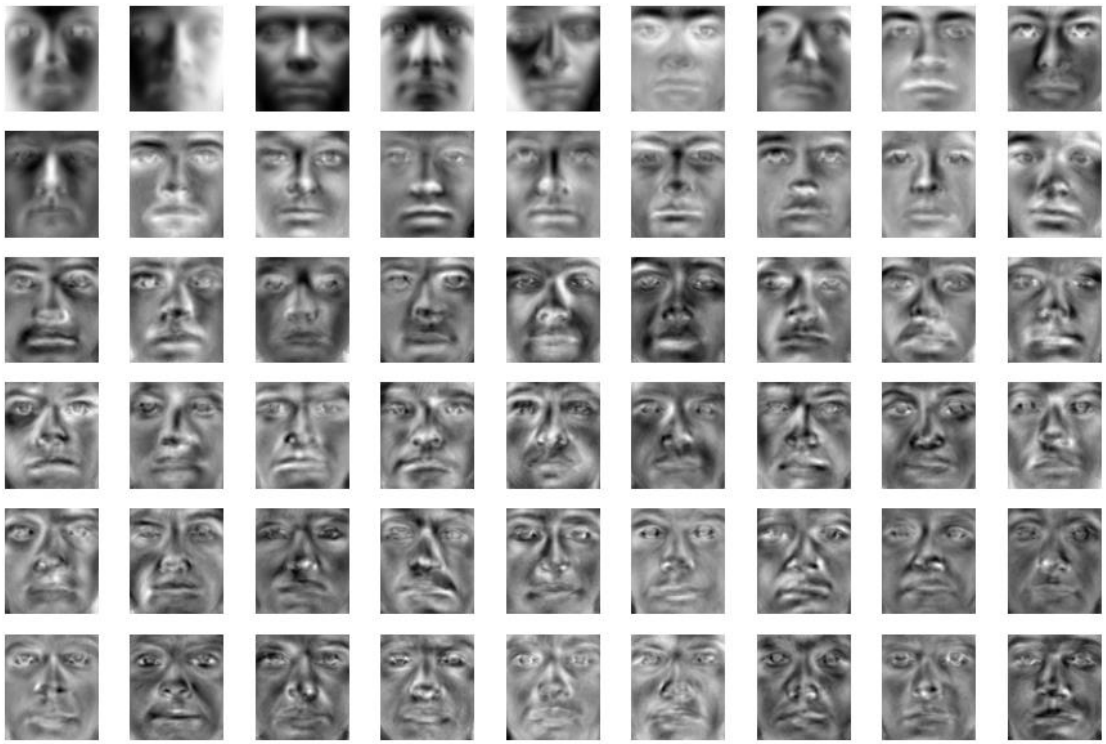
```
Images of Person  2
```

Part 2

```
import numpy as np
import scipy.io

faces = data['faces']
nfaces = data['nfaces']

end_index_36 = int(np.sum(nfaces[0, :36]))
all_faces_1_to_36 = faces[:, :end_index_36]

average_face = np.mean(all_faces_1_to_36, axis=1)

X = all_faces_1_to_36 - average_face[:, np.newaxis]

# Plot
plt.figure(figsize=(5, 5))
plt.imshow(np.reshape(average_face, (168, 192)).T, cmap='gray')
plt.title('Average Face')
plt.axis('off')
plt.show()
```

Part 3

```
U, S, Vt = np.linalg.svd(X, full_matrices=False)

# Plotting the first 54 eigenfaces
plt.figure(figsize=(9, 6))
for i in range(54):
    eigenface_image = np.reshape(U[:, i], (168, 192)).T
    plt.subplot(6, 9, i+1)
    plt.imshow(eigenface_image, cmap='gray')
    plt.axis('off')

plt.tight_layout()
plt.show()
```



Question: Carefully investigate these eigen faces. Do you think each eigen face is trying to detect certain features?

Answer: Eigenfaces are like filters that pick up different features from face images. The first few eigenfaces capture big changes, like light and shadow, or the general shape of a face. As we look at more eigenfaces, they start to show smaller details, like expressions or nose shapes. Towards the end, they might just show noise or tiny details. So, yes, each eigenface is highlighting different face features, from major ones to tiny details

Question: Compute the inner product of the first and the 5th eigen faces (do the computations with vectors, before reshaping to 2-dimensional pictures). Are these eigen faces orthogonal? How about the 10th eigen face and 15th eigen face? Are they all orthogonal?

Answer:

```
# Compute inner products
inner_product_1_5 = np.dot(U[:, 0], U[:, 4])
inner_product_10_15 = np.dot(U[:, 9], U[:, 14])

print("Inner product of 1st and 5th eigenface:", inner_product_1_5)
print("Inner product of 10th and 15th eigenface:", inner_product_10_15)
```

```
Inner product of 1st and 5th eigenface: 4.440892098500626e-16
Inner product of 10th and 15th eigenface: -8.326672684688674e-17
```

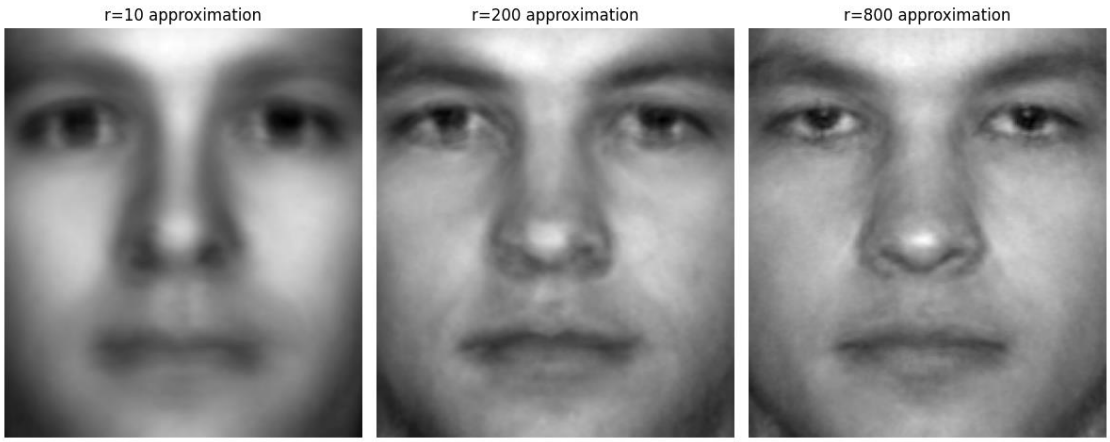
As the inner product is so close to *Zero* we can say that these eigen faces are orthogonal

Part 4

```
start_index_37 = int(np.sum(nfaces[0, :36]))
V = faces[:, start_index_37]

alphas = np.dot(U.T, V)

r = 5
approximated_image = average_face + np.dot(U[:, :r], alphas[:r])

approximated_image_reshaped = np.reshape(approximated_image, (m, n)).T

# Plot the approximation
plt.imshow(approximated_image_reshaped, cmap='gray')
plt.axis('off')
plt.title('r=5 approximation')
plt.show()
```


```
def approximate_image(r, U, alphas, average_face):
    approximated = average_face + np.dot(U[:, :r], alphas[:r])
    return np.reshape(approximated, (m, n)).T

r_values = [10, 200, 800]

plt.figure(figsize=(12, 5))
plt.subplot(1, 3, 1)
```

```
for idx, r in enumerate(r_values, 2):
    approximated_image_reshaped = approximate_image(r, U, alphas,
average_face)
    # Plot the approximation
    plt.subplot(1, 3, idx-1)
    plt.imshow(approximated_image_reshaped, cmap='gray')
    plt.axis('off')
    plt.title(f'r={r} approximation')

plt.tight_layout()
plt.show()
```

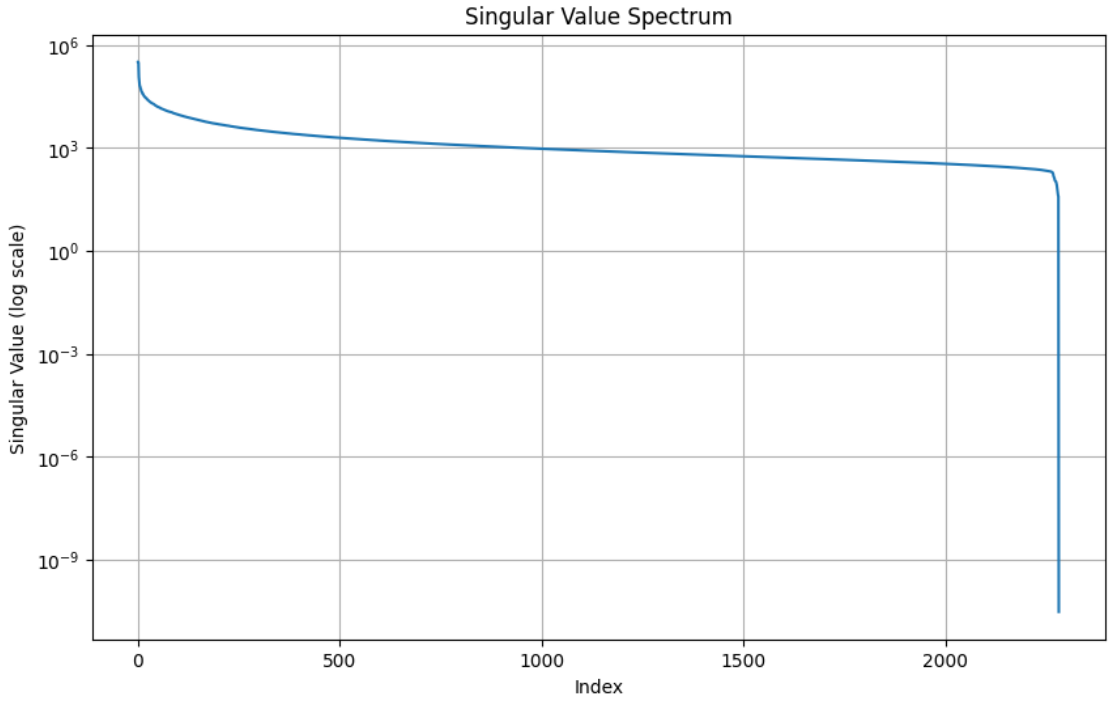


Question: Do you think you can find a good approximation of the picture with r=100?

Answer: Using the first 100 eigenfaces captures the main features of the face well. So, with r = 100 , we can get a pretty good approximation of the original picture. The more eigenfaces we use, the closer the approximation will be to the original image, but the first 100 already get us quite close.

Part 5

```
# Plotting singular values
plt.figure(figsize=(10,6))
plt.semilogy(S)
plt.xlabel('Index')
plt.ylabel('Singular Value (log scale)')
plt.title('Singular Value Spectrum')
plt.grid(True)
plt.show()
```



Question: Do you see a good point for truncation?

Answer: Yes. Around curve becomes horizontal between the values 500 to 1000. So I think index around 800 would be a good enough point for truncation.

Part 6

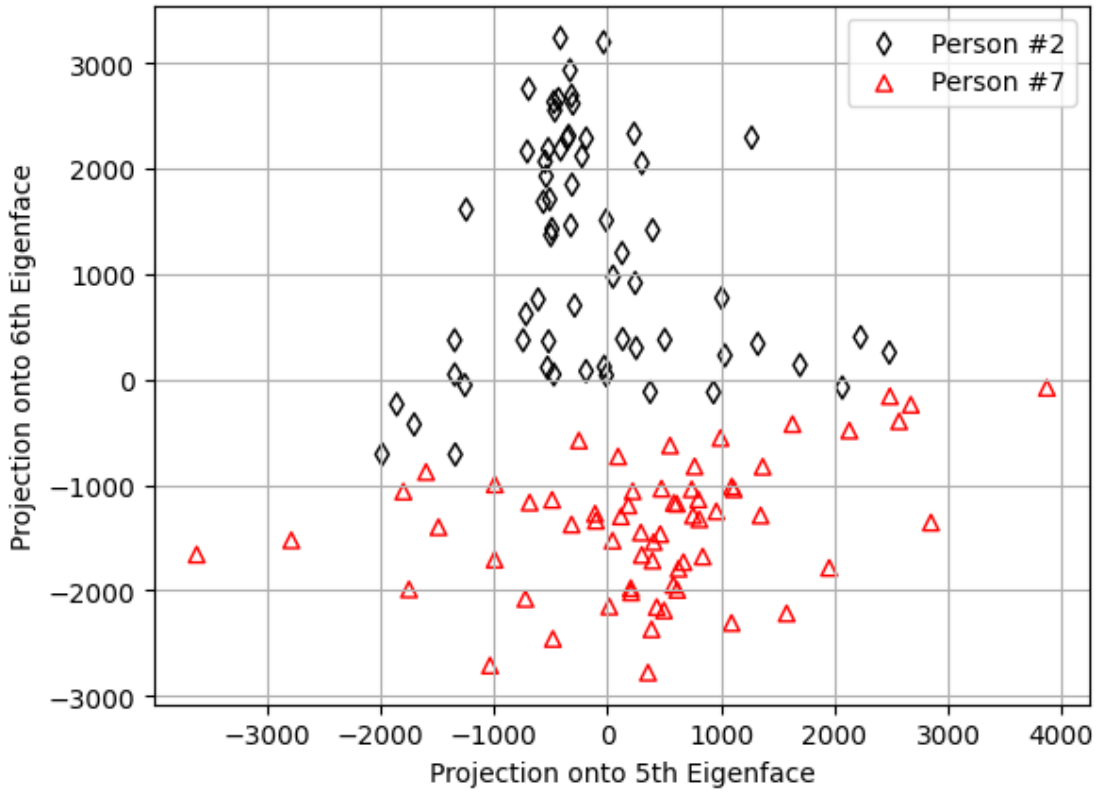
```
import numpy as np
import scipy.io


faces = data['faces']
nfaces = data['nfaces'][0]

start_col_person2 = sum(nfaces[:1])
end_col_person2 = start_col_person2 + nfaces[1]

start_col_person7 = sum(nfaces[:6])
end_col_person7 = start_col_person7 + nfaces[6]

images_person2 = [faces[:, i].reshape(192, 168) for i in
range(start_col_person2, end_col_person2)]
images_person7 = [faces[:, i].reshape(192, 168) for i in
range(start_col_person7, end_col_person7)]

# Coefficients for Person #2
alpha_5_person2 = [np.dot(img.flatten(), U[:, 4]) for img in images_person2]
alpha_6_person2 = [np.dot(img.flatten(), U[:, 5]) for img in images_person2]
```

```
print(alpha_5_person2)
print(alpha_6_person2)
```

```
[-222.95969654407554, -313.2782038665887, -413.11495902535717, -
740.7840805850287, -320.91181912028645, -287.3963956432432, -
461.3785308261795, -310.2091045156584, -351.7043857842541, -
7.782441339349873, -301.9279058135712, -184.78936192440415,
136.00448082644573, -339.57432814640697, -692.7551752369704, -
704.5500976082067, 246.27551936676176, 130.7800357925737, 935.5503529782068,
-562.6817533730841, 1036.1809673762427, 1009.6343094001304, -
1241.814408213123, 2069.909342155462, 50.944704155634085, 2229.8913080017187,
-7.845601825506474, 1325.7522306717858, -516.5049402703086, -
1343.7109031529576, 2483.5700461255906, 1697.0685951706953,
377.9923944696399, -410.0615278995308, -538.0556471797859, -
519.4101367943085, -496.96684461172913, -327.55776796266855, -
429.50032102845853, -507.0395422235997, -526.5199949598102, -
546.9626699778137, -469.93523195179705, -31.505378905447287,
305.20186072122306, -715.3094524389392, -487.0559542812046, -
1338.4007396939655, 236.8735134420442, -1253.4035439132547, -
608.649143723622, 1271.8983003603928, -1982.801488582084, 399.6640140054799,
-1853.6678094141332, -1342.8555473101903, -186.2574793707491,
506.17809361956506, 254.98693058507894, -1699.9836689032725, -
470.0373918912839, -26.480733554650993]
[2116.629210710098, 2690.1930332278594, 3237.8069746590068,
371.9022150891707, 1461.226706715649, 701.0668641578949, 2550.340127338158,
1847.1155110825268, 2282.358471250667, 1509.387686359129, 2615.1314620832104,
2284.6098218138477, 383.2774764335091, 2309.8469551532758, 2754.523500633301,
2161.693579568938, 915.1136808572364, 1198.643280019759, -120.68851701508163,
1683.6560755861015, 228.50118679960121, 772.4312988747389,
1611.4081215009003, -76.02809627397869, 975.8417816582829, 401.3840805026273,
39.9584475542747, 337.9942893820944, 362.1196701065692, 372.1728470485902,
255.98867773699146, 137.47856112303924, -117.99433248927765,
2181.187050139366, 1927.5277991479597, 2183.967011044795, 1369.1708353267616,
2931.136516232949, 2658.885015971977, 1708.7742860250223, 112.28905546983391,
2065.4551846716913, 2630.6393592607433, 3197.388379857549,
2051.5039335001493, 619.9286247882114, 1426.2547112680743, -
710.3256107280787, 2329.6923705286904, -52.949952695091724,
759.4684907472788, 2292.846067979319, -708.9467626996875, 1416.1717108935186,
-235.6535850162938, 47.513540019134055, 79.96330196136722, 376.0551809966093,
298.81911317476363, -425.6513044724193, 48.36904233311589,
120.63587938626257]
```

```
# Coefficients for Person #7
alpha_5_person7 = [np.dot(img.flatten(), U[:, 4]) for img in images_person7]
alpha_6_person7 = [np.dot(img.flatten(), U[:, 5]) for img in images_person7]

# Plot
plt.scatter(alpha_5_person2, alpha_6_person2,label='Person #2', marker =
'd',edgecolor='black', facecolor='none')
plt.scatter(alpha_5_person7, alpha_6_person7,label='Person #7',marker =
```

```
'^',edgecolor='red', facecolor='none')
plt.xlabel('Projection onto 5th Eigenface')
plt.ylabel('Projection onto 6th Eigenface')
plt.legend()
plt.grid()
plt.show()
```



Question: Can you use this graph to implement a good classification algorithm for face recognition? Can you do this with SVM?

Answer: Using the graph, it's possible to implement an effective face recognition algorithm. The plotted points, based on eigenfaces, show how different faces have unique characteristics. When we are using the coefficients of the eigenfaces, I am capturing the most distinctive features of each face with the use of coefficients.Support Vector Machines (SVM) can be used for this task. In simple terms, SVMs find the best boundary that separates different groups of data.