Intermediate Machine Learning: Assignment 3

Deadline

Assignment 3 is due Wednesday, October 30 by 11:59pm. Late work will not be accepted as per the course policies (see the Syllabus and Course policies on Canvas).

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged.

You should start early so that you have time to get help if you're stuck. The drop-in office hours schedule can be found on Canvas. You can also post questions or start discussions on Ed Discussion. The assignment may look long at first glance, but the problems are broken up into steps that should help you to make steady progress.

Submission

Submit your assignment as a pdf file on Gradescope, and as a notebook (.ipynb) on Canvas. You can access Gradescope through Canvas on the left-side of the class home page. The problems in each homework assignment are numbered. Note: When submitting on Gradescope, please select the correct pages of your pdf that correspond to each problem. This will allow graders to more easily find your complete solution to each problem.

To produce the .pdf, please do the following in order to preserve the cell structure of the notebook:

Go to "File" at the top-left of your Jupyter Notebook Under "Download as", select "HTML (.html)" After the .html has downloaded, open it and then select "File" and "Print" (note you will not actually be printing) From the print window, select the option to save as a .pdf

Topics

- Variational autoencoders
- Undirected graphs
- The graphical lasso

This assignment will also help to solidify your Python and Jupyter notebook skills.

Problem 1: Face time (35 points)

In this problem, we will implement a "shoestring" version of this amazing fake face generator, using a variational autoencoder (VAE). Building a generator like the one featured

in the article can take a tremendous amount of computational resources, time, and parameter tuning. In this problem we will build a basic version to illustrate the main concepts, and help you to become more famililar with VAEs. Here is an outline of the process that we'll step you through:

Problem outline:

- Load data
- Create face groups based on attributes
- Construct the VAE
- Define the loss function and train the VAE (Problem 1.1)
- Encode and reconstruct faces (Problem 1.2)
- Visualize the latent space (Problem 1.3)
- Morph between faces (Problem 1.4)
- Shift attributes of faces (Problem 1.5)
- Generate new faces (Problem 1.6)
- Analyze the effect of the scaling factor in the loss function (Problem 1.7, optional)

In the next cell we load the packages that we'll need. If you don't have one or more of these, you can install them with `!pip install <package_name>` in the cell, or outside the notebook with `conda install -c conda-forge <package_name>`

In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import os
import glob
import pandas as pd
import random
import numpy as np
import imageio
from tqdm import tqdm

from PIL import Image
from sklearn.model_selection import train_test_split
from scipy.stats import norm
import tensorflow

from sklearn.covariance import GraphicalLasso
import seaborn as sns
import networkx as nx

tensorflow.compat.v1.disable_eager_execution()
```

Loading the data

Labeled Faces in the Wild (LFW) is a database of face photographs. The images are placed in the folder lfw-deepfunneled. lfw_attributes.txt is a document including a set of attributes

associated for each image, such as 'Male', 'Smile', 'Bold', etc. All the features are numerical and large positive values indicate that the keywords well describe the photo; large negative values indicate that the keywords don't fit the photo.

For this problem, we will keep only the middle parts of the photos to avoid complex backgrounds.

Download the data from the cloud at these URLs:

https://sds365.s3.amazonaws.com/lfw/lfw-deepfunneled.zip
https://sds365.s3.amazonaws.com/lfw/lfw_attributes.txt

Once you have the data, unzip it, and place it in a directory that we will call "YOUR_PATH" below.

Run all the cells in this section to load the data.

Note: Please write down the entire path instead of using something like '~/Desktop/datasets/' to avoid unnecessary compiling errors. Also, if you choose to use Colab to do your homework. We need to download the data into the same directory as your code. You may also need

"from google.colab import drive"

"drive.mount('/content/drive')"

to enable using paths in Google Drive before starting your code below. But personally, I would suggest using jupyter notebook to run the code locally instead of using Google Colab since the latter may take longer time.

In [3]:
```
# Change these path names to correspond with your directory
DATASET_PATH ="/home/jacob/classes/SDS365/assn3/lfw-deepfunneled/lfw-deepfunneled/"
ATTRIBUTES_PATH = "/home/jacob/classes/SDS365/assn3/lfw_attributes.txt"
```

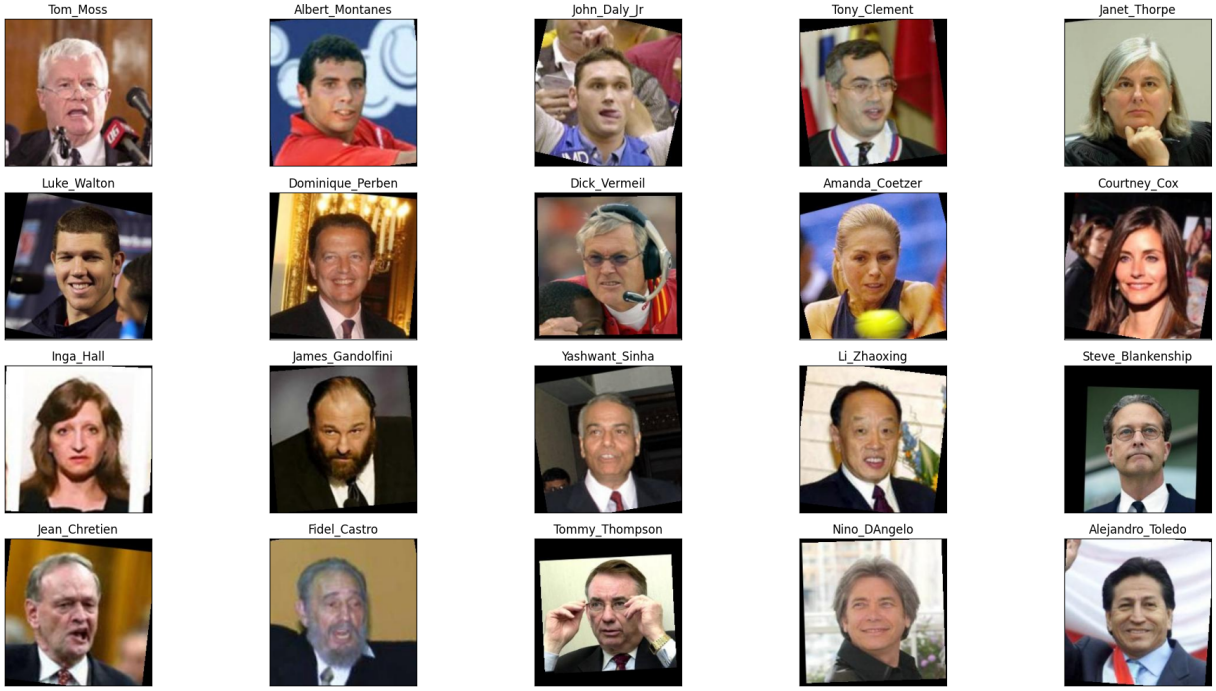
In [4]:
```
# Make sure the above path is correct before running this cell
dataset = []
for path in glob.iglob(os.path.join(DATASET_PATH, "**", "*.jpg")):
    person = path.split("/")[-2]
    dataset.append({"person":person, "path": path})

dataset = pd.DataFrame(dataset)
dataset = dataset.groupby("person").filter(lambda x: len(x) < 100 )
dataset.head(10)
```

Out[4]:

	person	path
0	Ellen_Saracini	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
1	Lydia_Shum	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
2	Yale_Kamisar	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
3	Pinar_del_Rio	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
4	Larry_Tanenbaum	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
5	Kristen_Rivera	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
6	Matt_Siebrandt	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
7	Ringo_Starr	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
8	Lee_Ann_Knight	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...
9	Tanya_Holyk	/home/jacob/classes/SDS365/assn3/lfw-deepfunne...

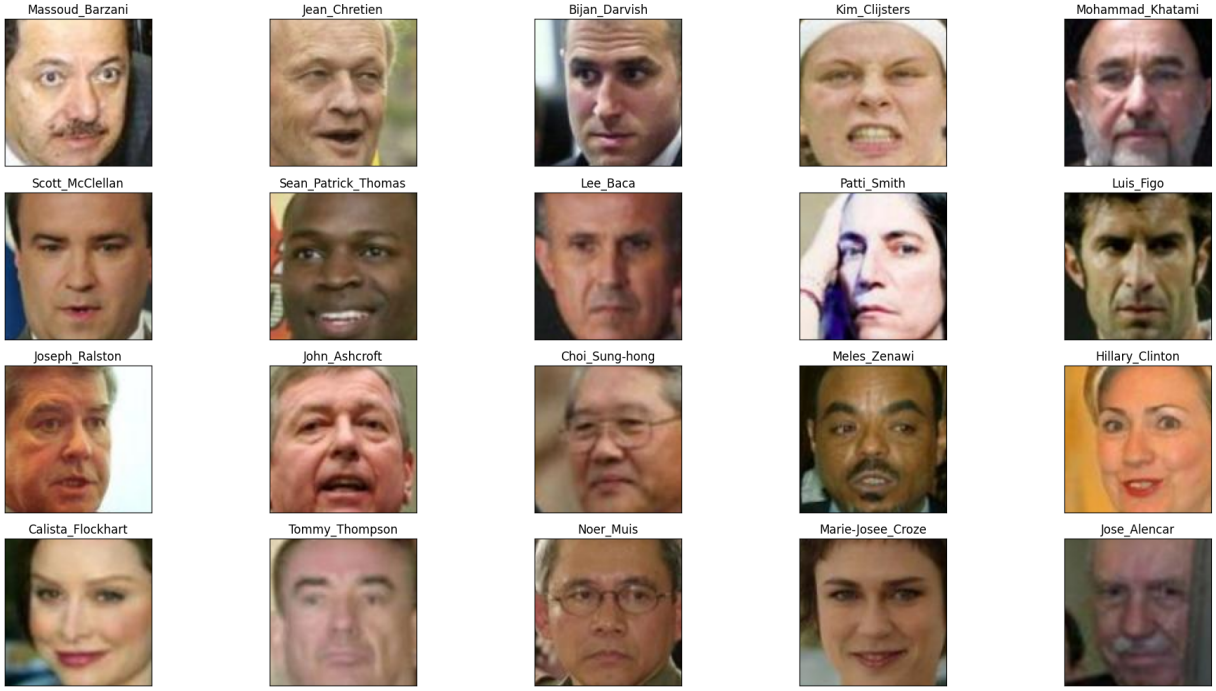
The following cell will display some sample images

In [33]:
```
sampled_id = []

plt.figure(figsize=(20,10))
for i in range(20):
    idx = random.randint(0, len(dataset))
    img = plt.imread(dataset.path.iloc[idx])
    plt.subplot(4, 5, i+1)
    plt.imshow(img)
    plt.title(dataset.person.iloc[idx])
    plt.xticks([])
    plt.yticks([])
    sampled_id.append(idx)
plt.tight_layout()
plt.show()
```



The following cell shows the images with some of the background removed.

In [42]:
```
dx=70
dy=70

plt.figure(figsize=(20,10))
for i in range(20):
    idx = sampled_id[i]
    img = plt.imread(dataset.path.iloc[idx])
    plt.subplot(4, 5, i+1)
    plt.imshow(img[dy:-dy,dx:-dx])
    plt.title(dataset.person.iloc[idx])
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
plt.show()
```



The following function crops the images to 45x45 pixels, which is what we will use in this problem.

In [5]:
```
def fetch_dataset(dx=70, dy=70, dimx=45, dimy=45):

    df_attrs = pd.read_csv(ATTRIBUTES_PATH, sep='\t', skiprows=1,)
    df_attrs = pd.DataFrame(df_attrs.iloc[:,:-1].values, columns = df_attrs.columns

    photo_ids = []
    for dirpath, dirnames, filenames in os.walk(DATASET_PATH):
        for fname in filenames:
            if fname.endswith(".jpg"):
                fpath = os.path.join(dirpath,fname)
                photo_id = fname[:-4].replace('_',' ').split()
                person_id = ' '.join(photo_id[:-1])
                photo_number = int(photo_id[-1])
                photo_ids.append({'person':person_id,'imagenum':photo_number,'photo

    photo_ids = pd.DataFrame(photo_ids)
    df = pd.merge(df_attrs,photo_ids,on=('person','imagenum'))

    assert len(df)==len(df_attrs),"lost some data when merging dataframes"

    all_photos = df['photo_path'].apply(imageio.imread)\
                                 .apply(lambda img:img[dy:-dy,dx:-dx])\
                                 .apply(lambda img: np.array(Image.fromarray(img).re

    all_photos = np.stack(all_photos.values).astype('uint8')
    all_attrs = df.drop(["photo_path","person","imagenum"],axis=1)

    return all_photos,all_attrs
```

The variable `data` has all the face images and the variable `attrs` has all the attributes.

The 8-bit RGB values are converted to values between 0 and 1 for modeling and plotting purposes.

In [6]:
```
data, attrs = fetch_dataset()
data = np.array(data / 255, dtype='float32')
```

Create Face Groups

We can now create groups of faces, by selecting the faces having the highest or lowest scores for each of the attributes. Run all the cells in this section to create and plot some face groups.

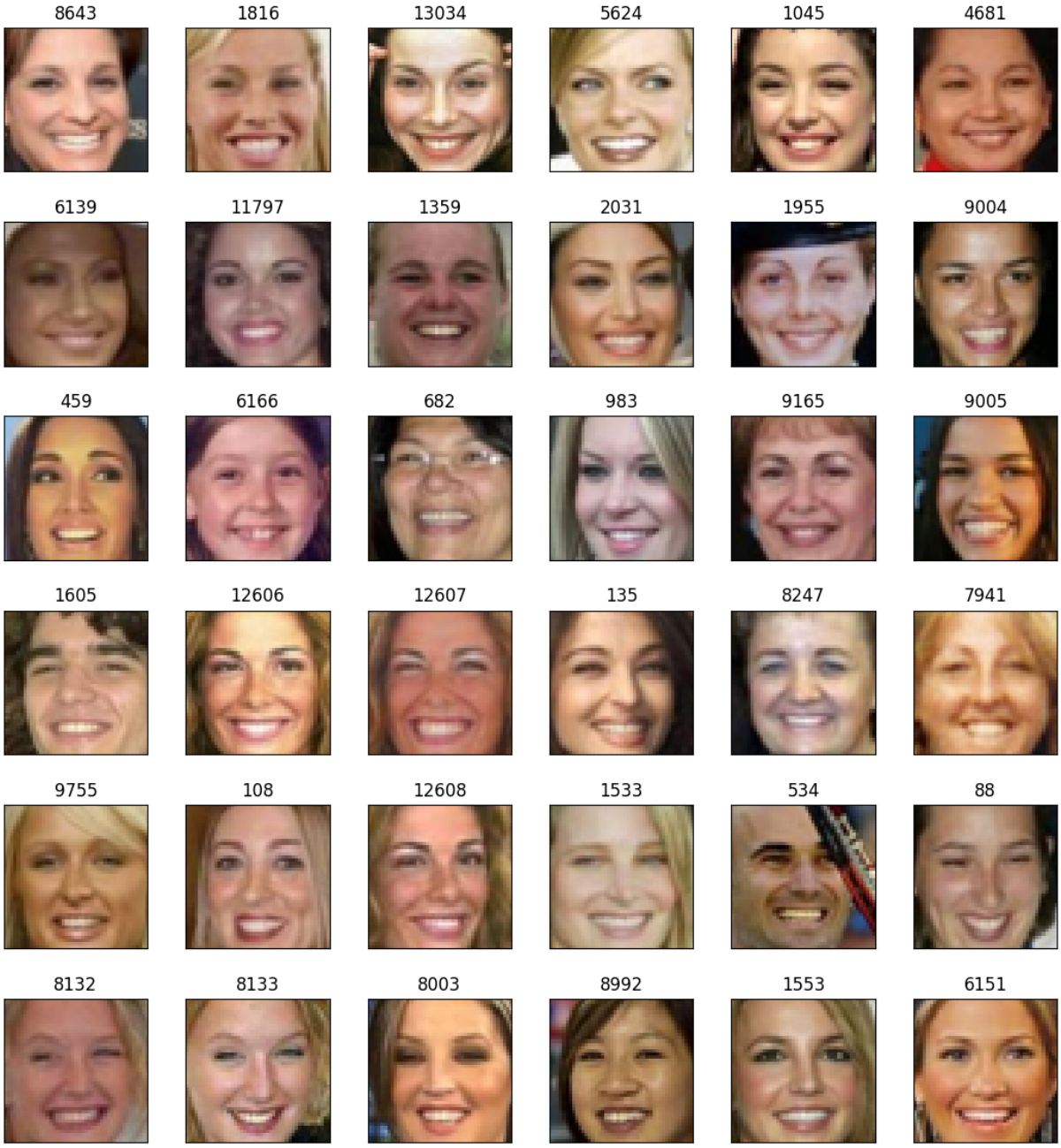
In [29]:
```
def plot_gallery(images, h, w, n_row=3, n_col=6, with_title=False, titles=[]):
    plt.figure(figsize=(1.75 * n_col, 2 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        try:
            plt.imshow(images[i].reshape((h, w, 3)), cmap=plt.cm.gray, vmin=-1, vma
            if with_title:
                plt.title(titles[i])
            plt.xticks(())
            plt.yticks(())
        except:
            pass
```

In [30]:
```
IMAGE_H = data.shape[1]
IMAGE_W = data.shape[2]
N_CHANNELS = 3
```

In [42]:
```
smile_ids = attrs['Smiling'].sort_values(ascending=False).head(36).index.values
smile_data = data[smile_ids]

no_smile_ids = attrs['Smiling'].sort_values(ascending=True).head(36).index.values
no_smile_data = data[no_smile_ids]

eyeglasses_ids = attrs['Eyeglasses'].sort_values(ascending=False).head(36).index.va
eyeglasses_data = data[eyeglasses_ids]

sunglasses_ids = attrs['Sunglasses'].sort_values(ascending=False).head(36).index.va
sunglasses_data = data[sunglasses_ids]

mustache_ids = attrs['Mustache'].sort_values(ascending=False).head(36).index.values
mustache_data = data[mustache_ids]

male_ids = attrs['Male'].sort_values(ascending=False).head(36).index.values
male_data = data[male_ids]

female_ids = attrs['Male'].sort_values(ascending=True).head(36).index.values
female_data = data[female_ids]

eyeclosed_ids = attrs['Eyes Open'].sort_values(ascending=True).head(36).index.value
```

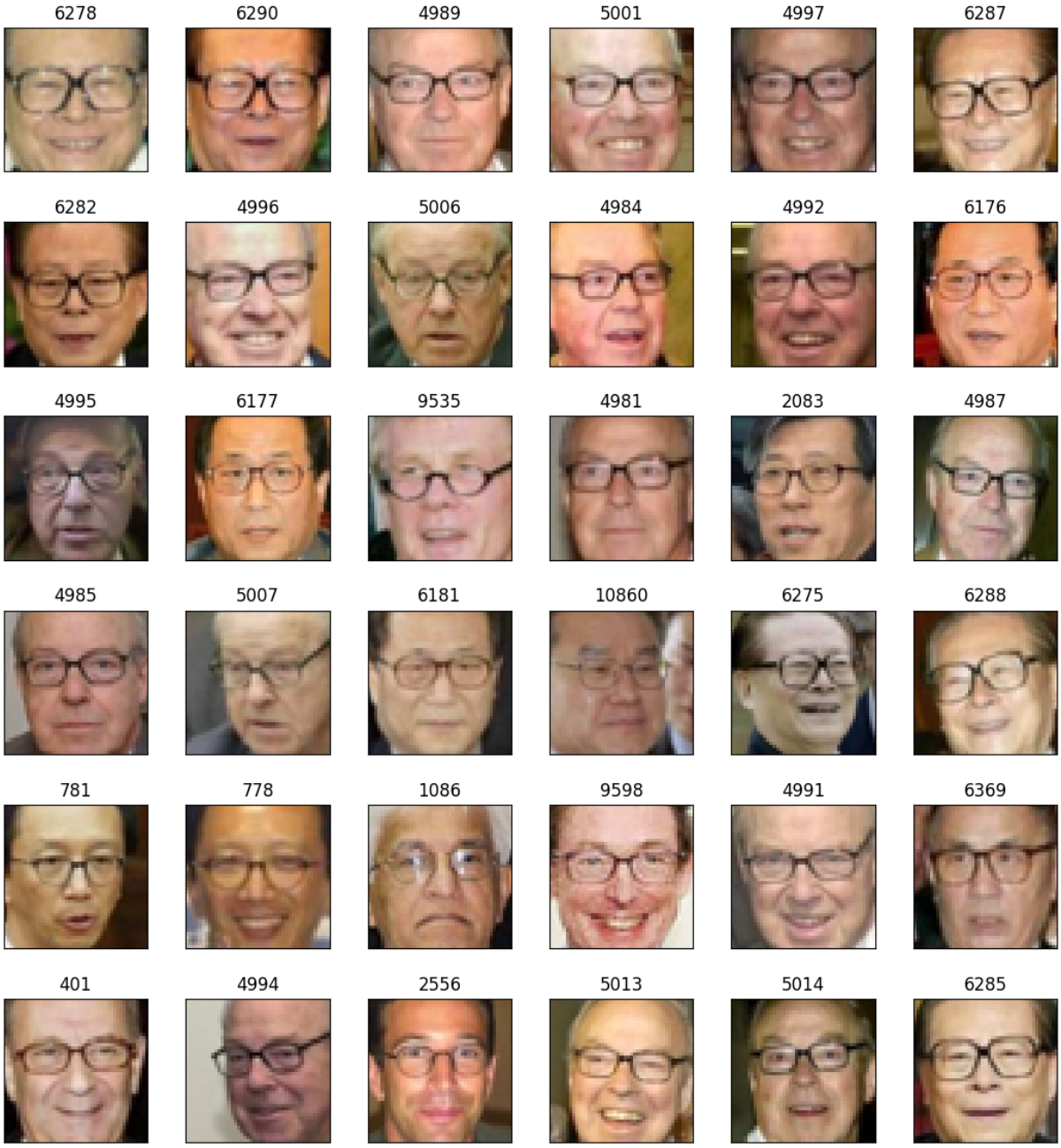
```
eyeclosed_data = data[eyeclosed_ids]

mouthopen_ids = attrs['Mouth Wide Open'].sort_values(ascending=False).head(36).inde
mouthopen_data = data[mouthopen_ids]

makeup_ids = attrs['Heavy Makeup'].sort_values(ascending=False).head(36).index.valu
makeup_data = data[makeup_ids]
```

In [32]:
```
plot_gallery(smile_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, title
```

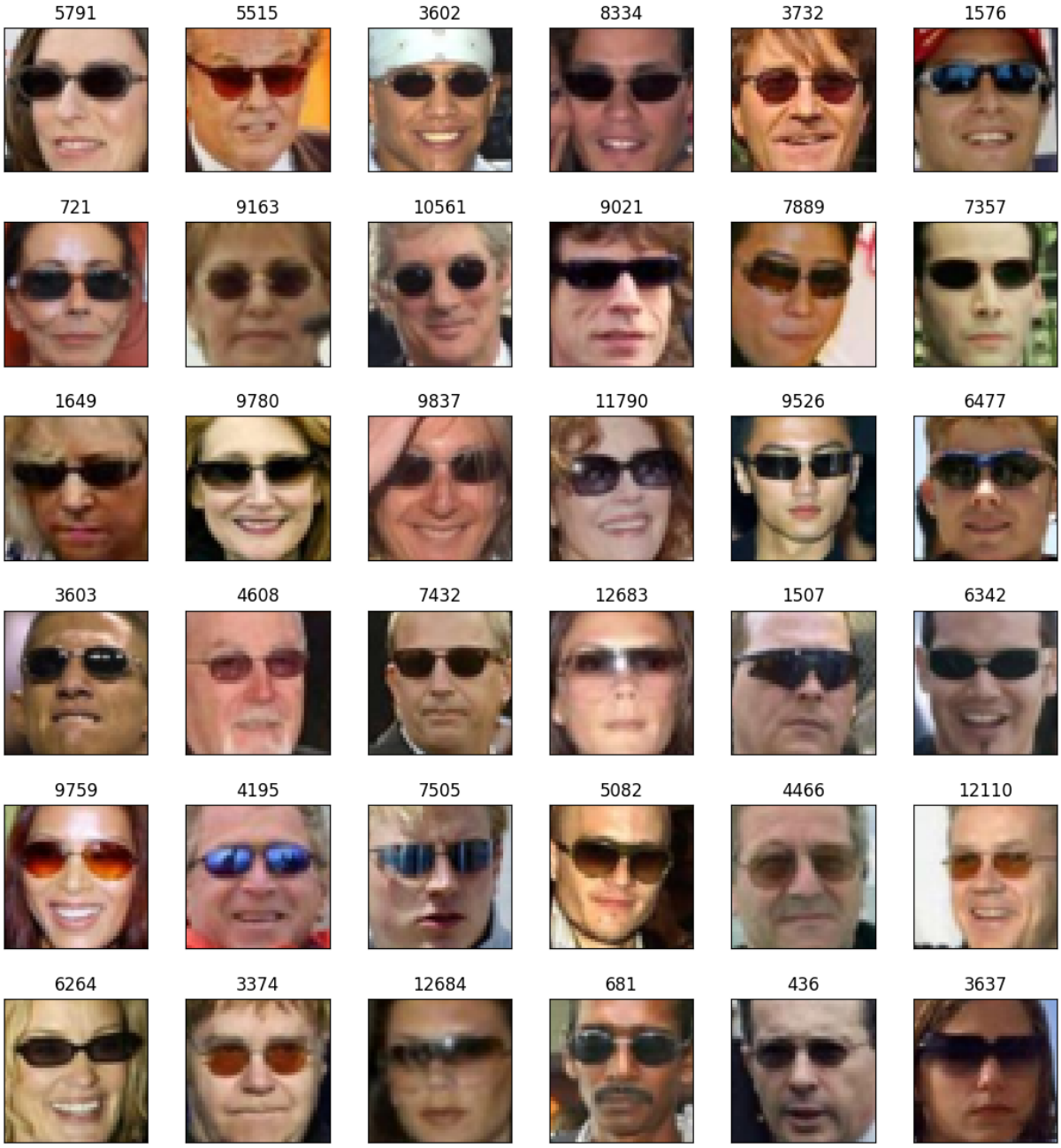


In [33]:
```
plot_gallery(no_smile_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, ti
```

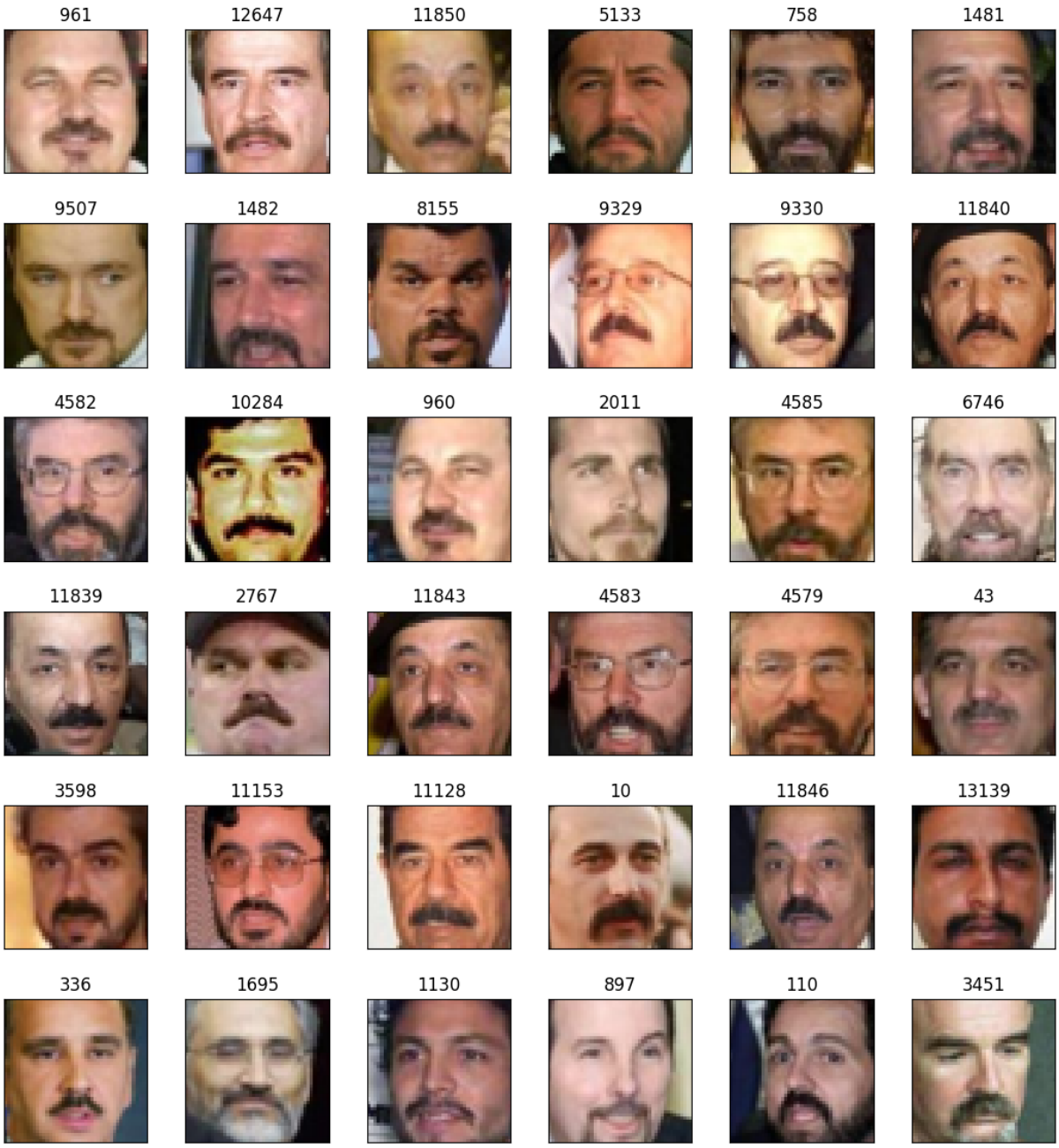
In [34]:
```
plot_gallery(eyeglasses_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,
```



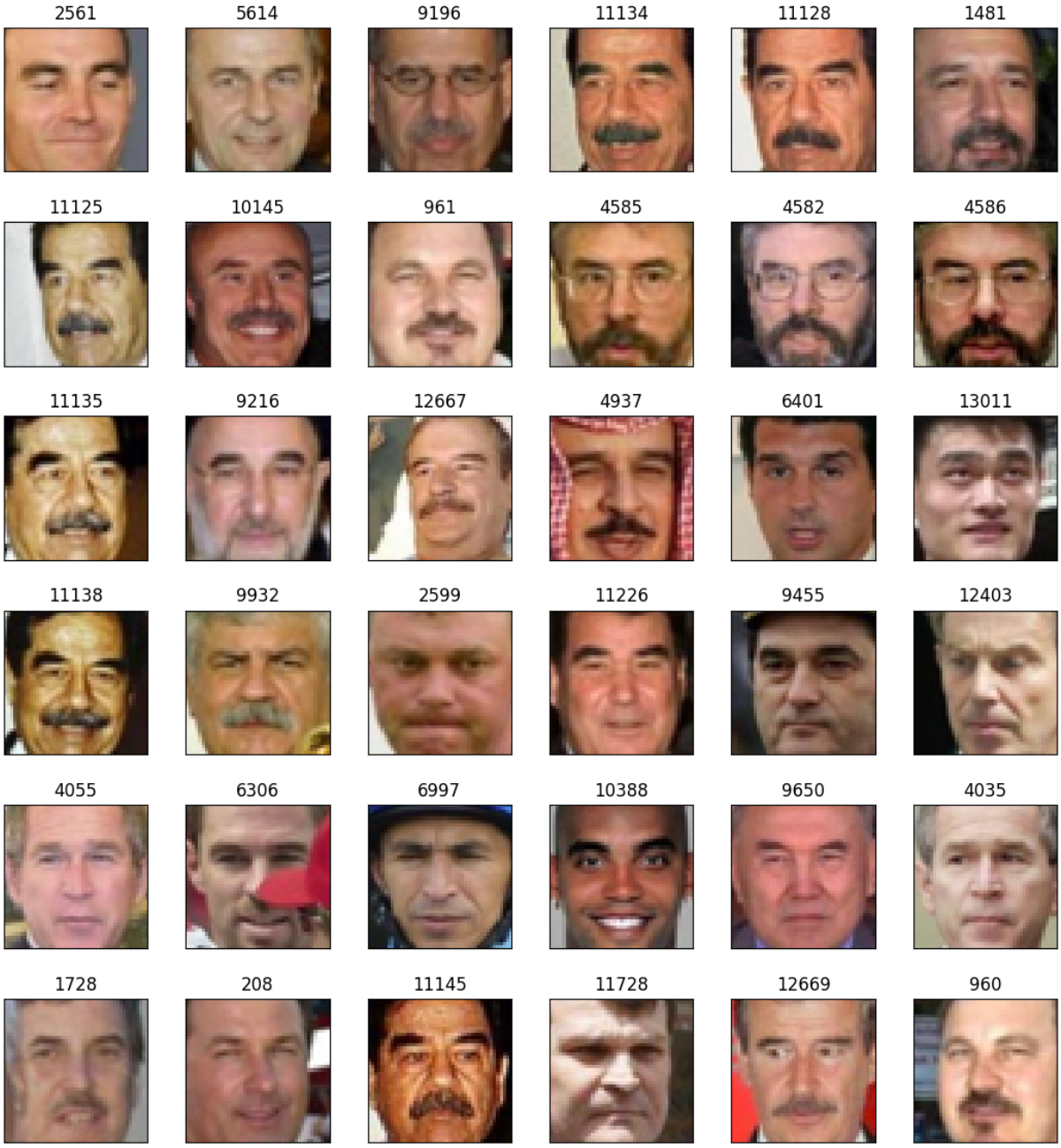
In [35]:
```
plot_gallery(sunglasses_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,
```



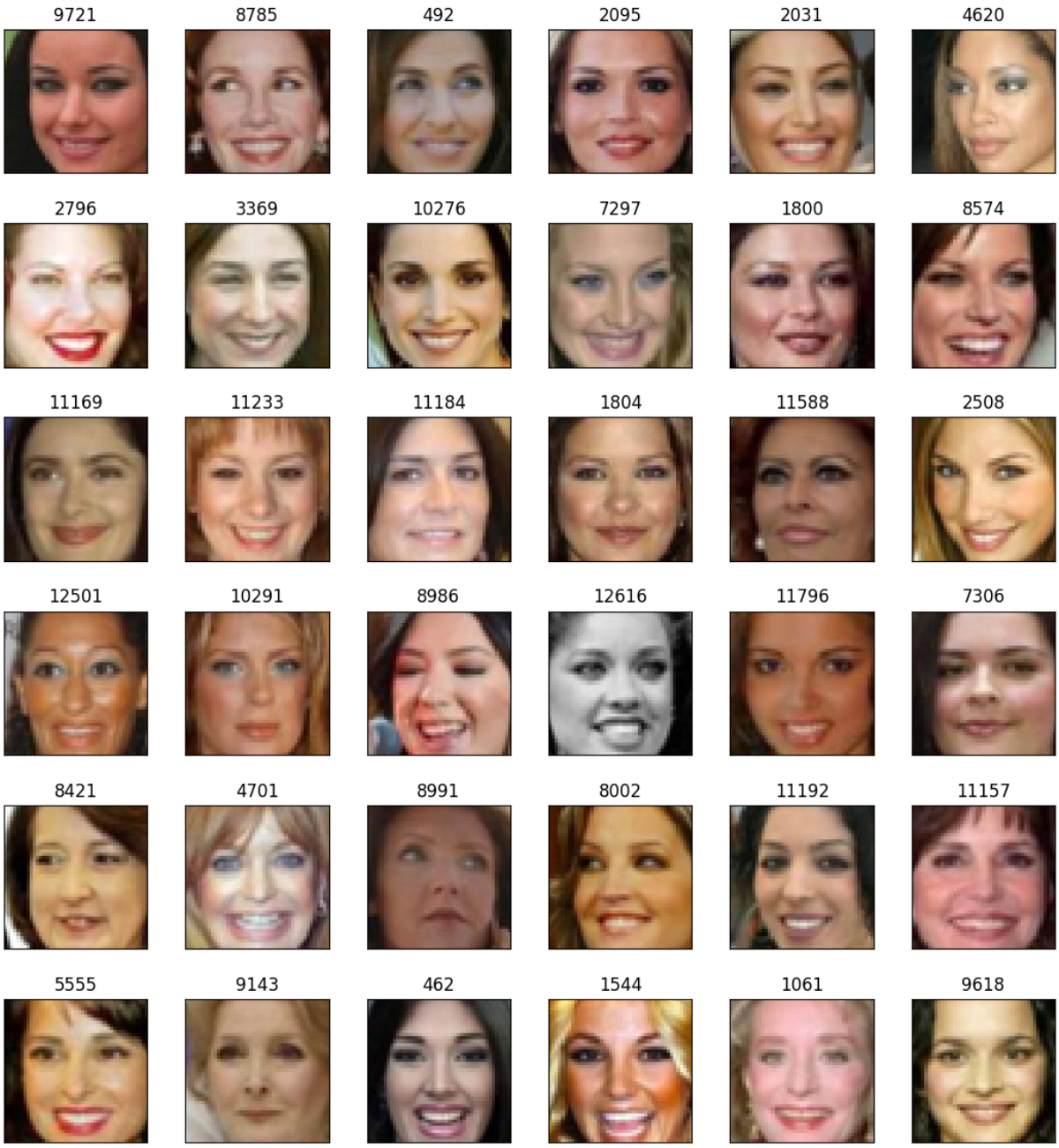
In [36]:
```
plot_gallery(mustache_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, ti
```



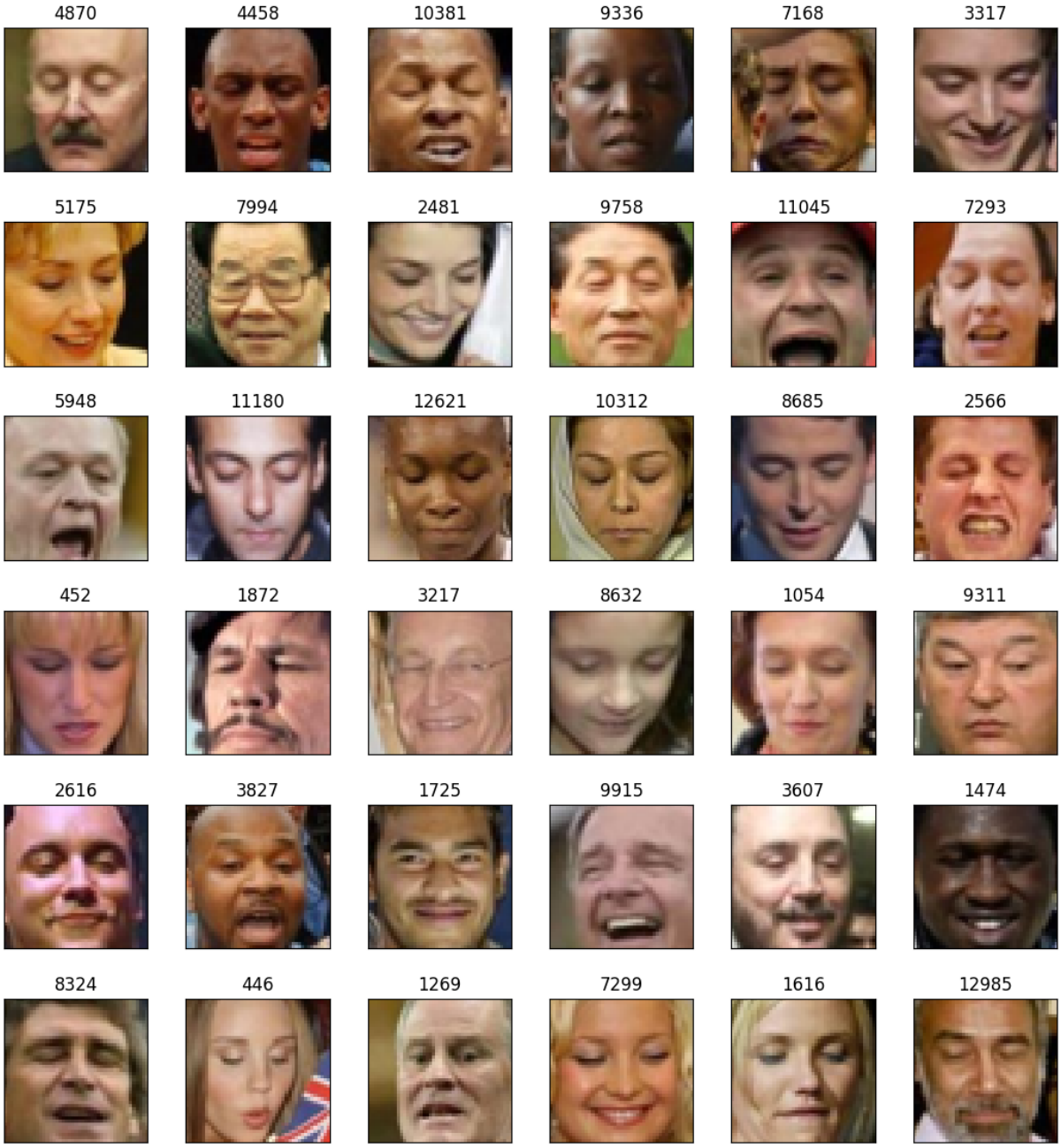
In [37]:
```
plot_gallery(male_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, titles
```



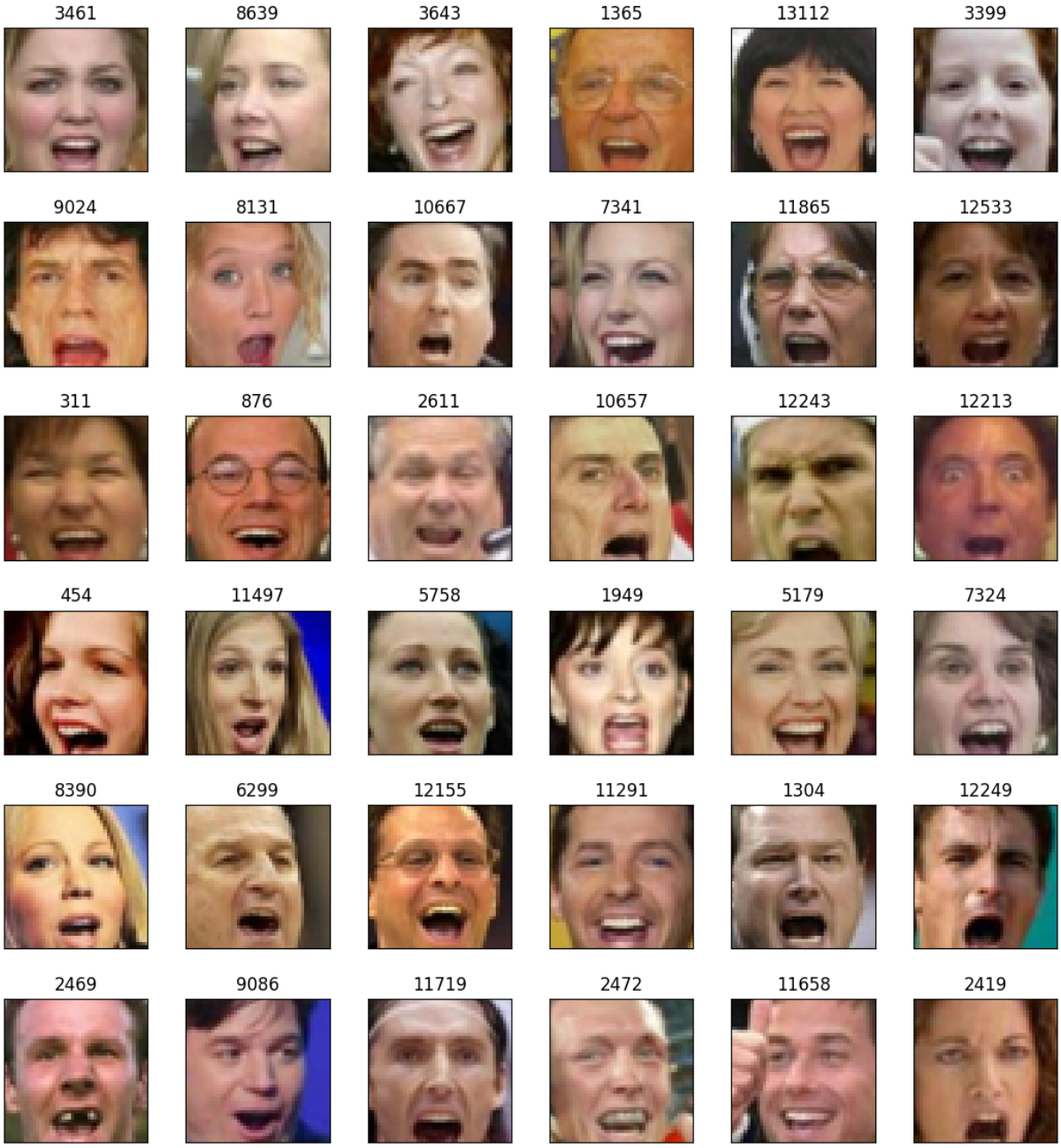
In [38]:
```
plot_gallery(female_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, titl
```



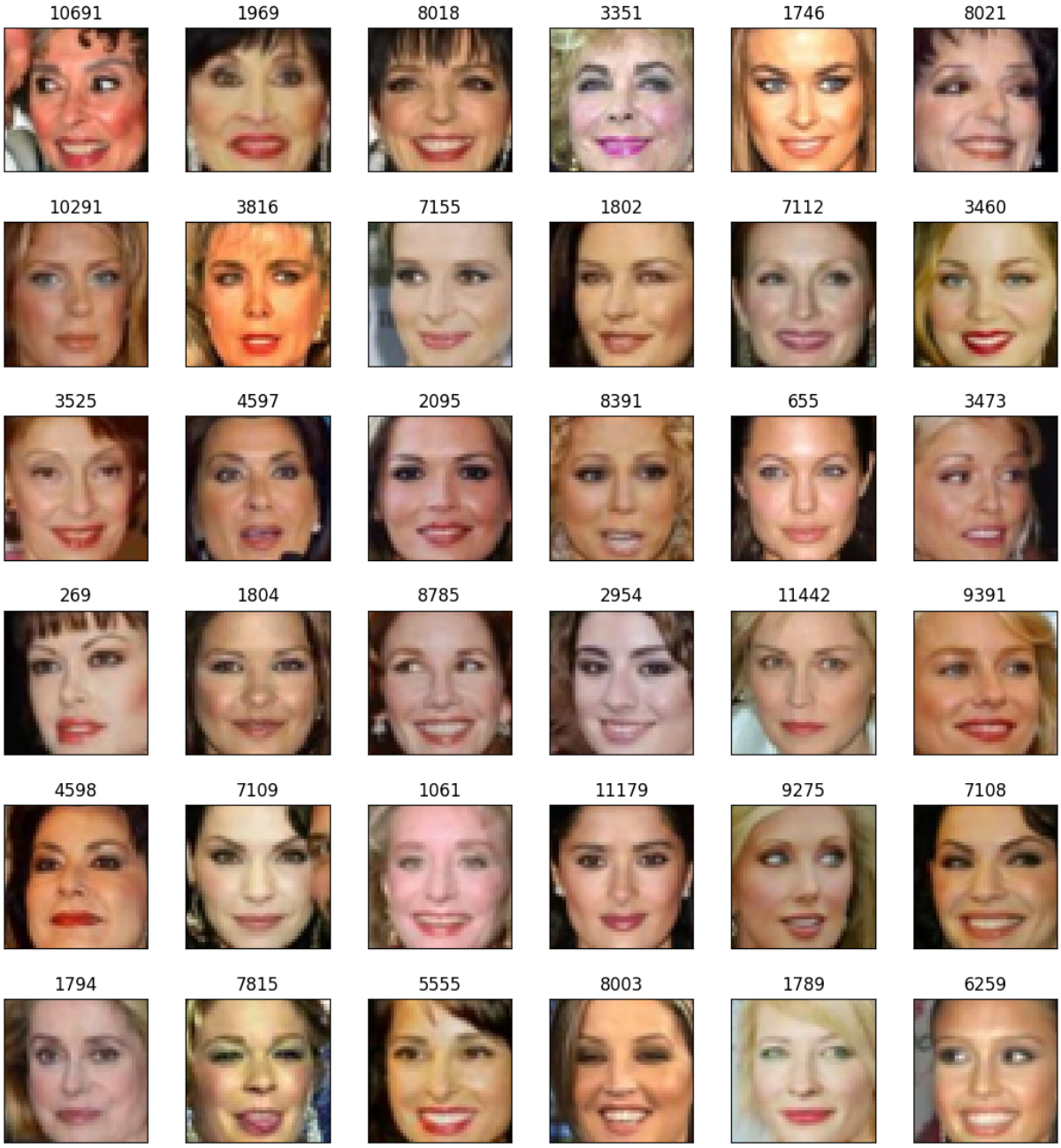
In [39]:
```
plot_gallery(eyeclosed_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, t
```



In [40]:
```
plot_gallery(mouthopen_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, t
```



In [41]:
```
plot_gallery(makeup_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True, titl
```



Constructing the encoder

Recall that the encoder part of the VAE architecture maps a data point to a variational mean and (log) variance. The mean is a point in the latent space.

In [7]:
```
LATENT_SPACE_SIZE = 100
```

The "reparameterization trick" draws samples from the variational distribution that are parameterized by the variational mean and variance, so that the parameters of the encoder network can be trained.

In [8]:
```
def sample_latent_features(distribution):
    distribution_mean, distribution_variance = distribution
    batch_size = tensorflow.shape(distribution_variance)[0]
```

```
    random = tensorflow.keras.backend.random_normal(shape=(batch_size, tensorflow.s
    return distribution_mean + tensorflow.exp(0.5 * distribution_variance) * random
```

In [9]:

```
input_data = tensorflow.keras.layers.Input(shape=(45, 45, 3))

encoder = tensorflow.keras.layers.Conv2D(64, (5,5), activation='relu')(input_data)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(64, (3,3), activation='relu')(encoder)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(32, (3,3), activation='relu')(encoder)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Flatten()(encoder)

distribution_mean = tensorflow.keras.layers.Dense(LATENT_SPACE_SIZE, name='variatio
distribution_variance = tensorflow.keras.layers.Dense(LATENT_SPACE_SIZE, name='vari
latent_encoding = tensorflow.keras.layers.Lambda(sample_latent_features)([distribut

encoder_model = tensorflow.keras.Model(input_data, latent_encoding)
encoder_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_1 (InputLayer)           [(None, 45, 45, 3)]  0           []

 conv2d (Conv2D)                (None, 41, 41, 64)   4864        ['input_1[0][0]']

 max_pooling2d (MaxPooling2D)   (None, 20, 20, 64)   0           ['conv2d[0][0]']

 conv2d_1 (Conv2D)              (None, 18, 18, 64)   36928       ['max_pooling2d[0]
[0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 9, 9, 64)    0           ['conv2d_1[0][0]']

 conv2d_2 (Conv2D)              (None, 7, 7, 32)     18464       ['max_pooling2d_1
[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 3, 3, 32)    0           ['conv2d_2[0][0]']

 flatten (Flatten)              (None, 288)          0           ['max_pooling2d_2
[0][0]']

 variational_mean (Dense)       (None, 100)          28900       ['flatten[0][0]']

 variational_log_variance (Dens  (None, 100)         28900       ['flatten[0][0]']
 e)

 lambda (Lambda)                (None, 100)          0           ['variational_mean
[0][0]',
                                                                  'variational_log_v
ariance[0][0]'
                                                                  ]

==========================================================================================
Total params: 118,056
Trainable params: 118,056
Non-trainable params: 0
__________________________________________________________________________________________
```

Construct the decoder

The decoder network in the VAE architecture maps a latent vector to an image. This is done with a series of transposed convolutional layers, since it must map from low to high dimensions.

In [10]:
```
decoder_input = tensorflow.keras.layers.Input(shape=(LATENT_SPACE_SIZE,))
```

```
decoder = tensorflow.keras.layers.Reshape((1, 1, 100))(decoder_input)
decoder = tensorflow.keras.layers.Conv2DTranspose(64, (3,3), activation='relu')(dec
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(32, (3,3), activation='relu')(dec
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(16, (5,5), activation='relu')(dec
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder_output = tensorflow.keras.layers.Conv2DTranspose(3, (6,6), activation='relu

decoder_model = tensorflow.keras.Model(decoder_input, decoder_output)
decoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 100)]             0

 reshape (Reshape)           (None, 1, 1, 100)         0

 conv2d_transpose (Conv2DTra  (None, 3, 3, 64)         57664
 nspose)

 up_sampling2d (UpSampling2D  (None, 6, 6, 64)         0
 )

 conv2d_transpose_1 (Conv2DT  (None, 8, 8, 32)         18464
 ranspose)

 up_sampling2d_1 (UpSampling  (None, 16, 16, 32)       0
 2D)

 conv2d_transpose_2 (Conv2DT  (None, 20, 20, 16)       12816
 ranspose)

 up_sampling2d_2 (UpSampling  (None, 40, 40, 16)       0
 2D)

 conv2d_transpose_3 (Conv2DT  (None, 45, 45, 3)        1731
 ranspose)

=================================================================
Total params: 90,675
Trainable params: 90,675
Non-trainable params: 0
_________________________________________________________________
```

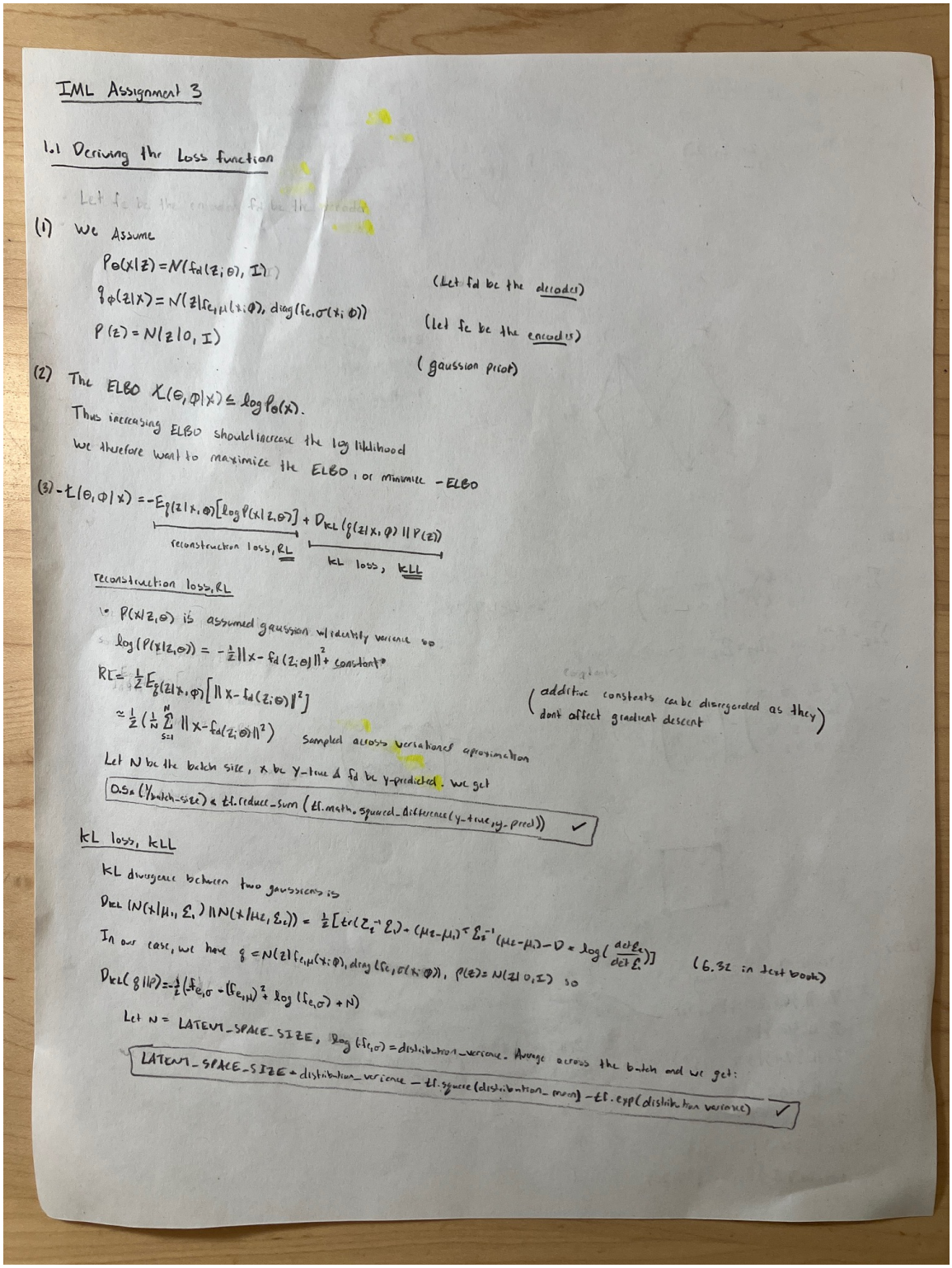
In [11]:
```
encoded = encoder_model(input_data)
decoded = decoder_model(encoded)
autoencoder = tensorflow.keras.models.Model(input_data, decoded)
```

In [23]:
```
def get_loss(distribution_mean, distribution_variance, factor, batch_size):

    def get_reconstruction_loss(y_true, y_pred, factor, batch_size):
        reconstruction_loss = tensorflow.math.squared_difference(y_true, y_pred)
        reconstruction_loss_batch = tensorflow.reduce_sum(reconstruction_loss)/batc
        return 0.5*reconstruction_loss_batch*factor

    def get_kl_loss(distribution_mean, distribution_variance, batch_size):
        kl_loss = LATENT_SPACE_SIZE + distribution_variance - tensorflow.square(dis
        kl_loss_batch = tensorflow.reduce_sum(kl_loss)/batch_size
        return kl_loss_batch*(-0.5)

    def total_loss(y_true, y_pred):
        reconstruction_loss_batch = get_reconstruction_loss(y_true, y_pred, factor,
        kl_loss_batch = get_kl_loss(distribution_mean, distribution_variance, batch
        return reconstruction_loss_batch + kl_loss_batch

    return total_loss
```

1.1 Deriving the loss function (5 points)

Derive the loss function defined in the cell above from the probability model perspective. You can ignore the scalar `factor` in your derivation. Show your work using either LaTeX or a picture of your written solution.

Hint: Think about how how the total loss is related to the ELBO.



The following three cells train the model. You can just run them. It may take a while to run on your laptop.

In [24]:
```
X_train, X_val = train_test_split(data, test_size=0.2, random_state=365)
```

In [25]:
```
batch_size = 64
autoencoder.compile(loss=get_loss(distribution_mean, distribution_variance, factor
                                  batch_size = batch_size), optimizer='adam')
```

```
autoencoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 45, 45, 3)]       0
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 45, 45, 3)]       0

 model (Functional)          (None, 100)               118056

 model_1 (Functional)        (None, 45, 45, 3)         90675

=================================================================
Total params: 208,731
Trainable params: 208,731
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
# autoencoder.fit(X_train, X_train, epochs=50, batch_size=64, validation_data=(X_va
```

In [27]:
```
# Save your trained model (for later use)

# autoencoder.save("My_Trained_VAE")
# encoder_model.save("My_Trained_encoder")
# decoder_model.save("My_Trained_decoder")
```

If you experience issues while training the models above or if the process is taking too long, you may also use our pre-trained versions of the VAE, encoder, and decoder, which are available on Canvas.

No points will be taken off if you decide to use the pre-trained models, but we do recommend trying out the training process above. In case you encounter any issues when loading in the models, please post your questions on Ed discussion.

In [22]:
```
# Load Pre-trained Model

batch_size = 64
autoencoder = tensorflow.keras.models.load_model("/home/jacob/classes/SDS365/assn3/
autoencoder.compile(loss=get_loss(distribution_mean, distribution_variance, factor

encoder_model = tensorflow.keras.models.load_model("/home/jacob/classes/SDS365/assn
encoder_model.compile(loss=get_loss(distribution_mean, distribution_variance, facto

decoder_model = tensorflow.keras.models.load_model("/home/jacob/classes/SDS365/assn
decoder_model = tensorflow.keras.models.load_model("/home/jacob/classes/SDS365/assn
decoder_model.compile(loss=get_loss(distribution_mean, distribution_variance, facto
```

```
WARNING:tensorflow:SavedModel saved prior to TF 2.5 detected when loading Keras mode
l. Please ensure that you are saving the model with model.save() or tf.keras.model
s.save_model(), *NOT* tf.saved_model.save(). To confirm, there should be a file name
d "keras_metadata.pb" in the SavedModel directory.
WARNING:tensorflow:SavedModel saved prior to TF 2.5 detected when loading Keras mode
l. Please ensure that you are saving the model with model.save() or tf.keras.model
s.save_model(), *NOT* tf.saved_model.save(). To confirm, there should be a file name
d "keras_metadata.pb" in the SavedModel directory.
WARNING:tensorflow:SavedModel saved prior to TF 2.5 detected when loading Keras mode
l. Please ensure that you are saving the model with model.save() or tf.keras.model
s.save_model(), *NOT* tf.saved_model.save(). To confirm, there should be a file name
d "keras_metadata.pb" in the SavedModel directory.
WARNING:tensorflow:SavedModel saved prior to TF 2.5 detected when loading Keras mode
l. Please ensure that you are saving the model with model.save() or tf.keras.model
s.save_model(), *NOT* tf.saved_model.save(). To confirm, there should be a file name
d "keras_metadata.pb" in the SavedModel directory.
```

1.2 Reconstructing faces (3 points)

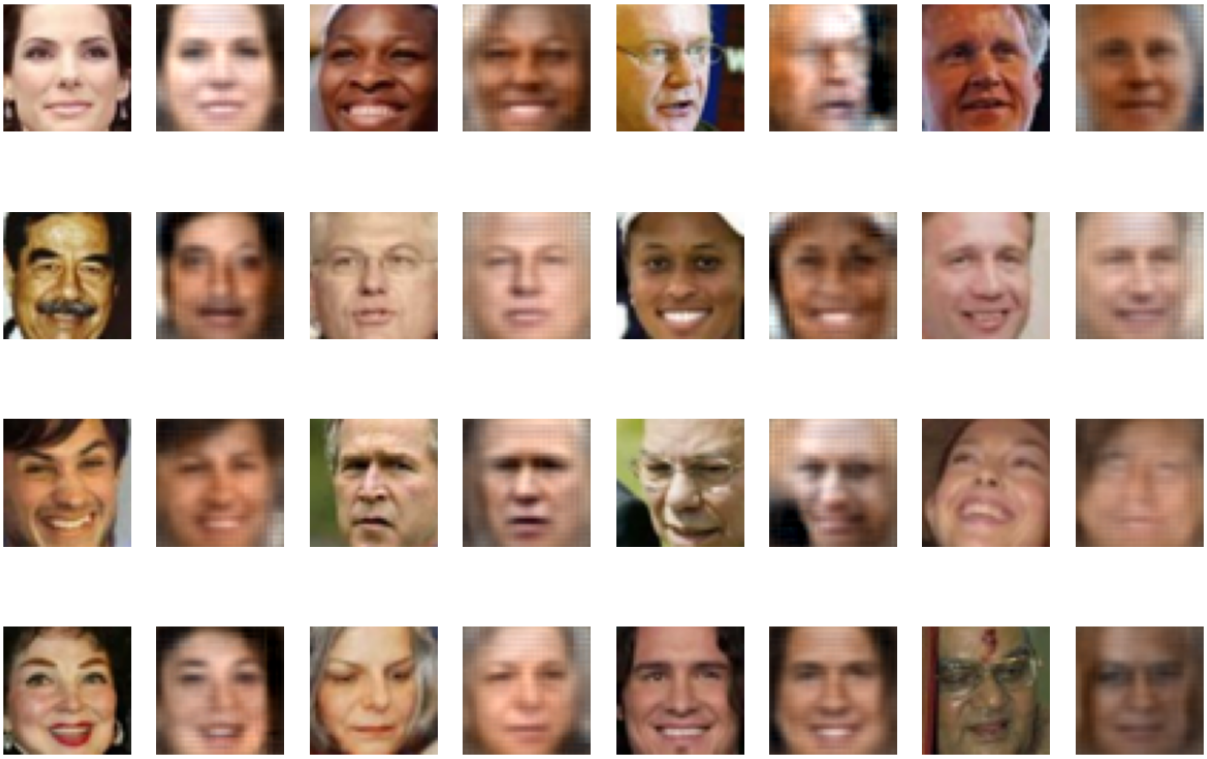
The following cell encodes and reconstructs 16 random faces from the validation set with the trained VAE. Run the cell and comment on the reconstructed faces. (3 points)

- Do the reconstructed faces resemble the original images? How are they similar/different?
- Are there any faces that are reconstructed better or worse than the others? Can you think of why?
- Comment on any other aspects of your findings that are interesting to you.

In [28]:

```
sample_index = random.sample(range(1, len(X_val)), 16)

fig, axs = plt.subplots(4, 8)
fig.set_figheight(10)
fig.set_figwidth(15)

for i in range(4):
    for j in range(4):
        axs[i, 2*j].imshow(X_val[sample_index[4*i+j], :, :, :])
        axs[i, 2*j].axis('off')
        axs[i, 2*j+1].imshow(np.clip(autoencoder.predict(np.array([X_val[sample_ind
        axs[i, 2*j+1].axis('off')
```

```
/home/jacob/lpd/classes/SDS365/IMLvenv/lib/python3.10/site-packages/keras/engine/tra
ining_v1.py:2067: UserWarning: `Model.state_updates` will be removed in a future ver
sion. This property should not be used in TensorFlow 2.0, as `updates` are applied a
utomatically.
  updates=self.state_updates,
```



The reconstructed faces are all recognizably similar to the original, but blurier and with less distinct features. There's a face of an older white man what reconstructed particularly well. I'm interested if that may reflect the disproportionate number of samples of older white men in the dataset. There's also a man with a scar on his forward, which the deconstruction did not capture. This makes sense because the latent space compresses the image, and there's probably not enough images with forhead scars to produce a forhead scar region of the latent space. The reconstructions did a relatively good job of capturing face angle, when the face angle was facing forward or slightly off to the side. It did less good for more extreme head orientations. Again, this makes sense because the reconstruction is going to be better at a situation where there were more training data like it.

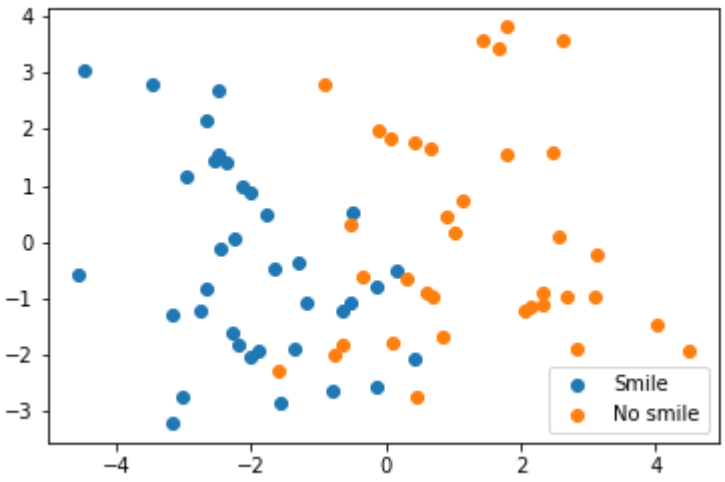
1.3 Visualizing the latent space (10 points)

In `vae_demo` from class, the MNIST digits were generated from a two-dimensional latent space. In the current model, the latent space has more than two dimensions, so to visualize it we need to use a dimensionality reduction technique. (If you are not familiar with PCA, please refer to the material for Week 7 of iML.)

In this problem, you will first implement the function `LatentSpace_2D` . (6 points)

1. Calculate the latent space encodings for two sets of faces that are different in one attribute, e.g. smile vs. no smile.
2. Use PCA to reduce the dimension of the latent space codes to two.
3. Visualize the latent space after dimensionality reduction with a scatter plot. Clearly color-code and label the two different groups.

Here is an example using smile_data and no_smile_data.

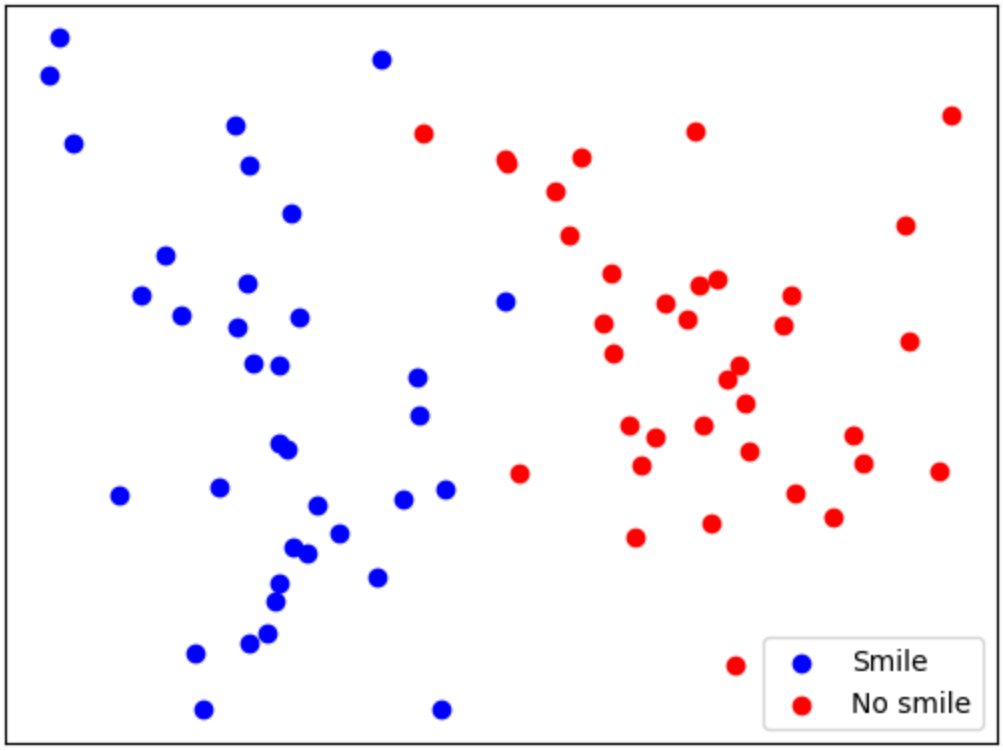


Visualize the latent space for at least three pairs of face groups including smile vs. no smile. Comment on how the scatter plots look.

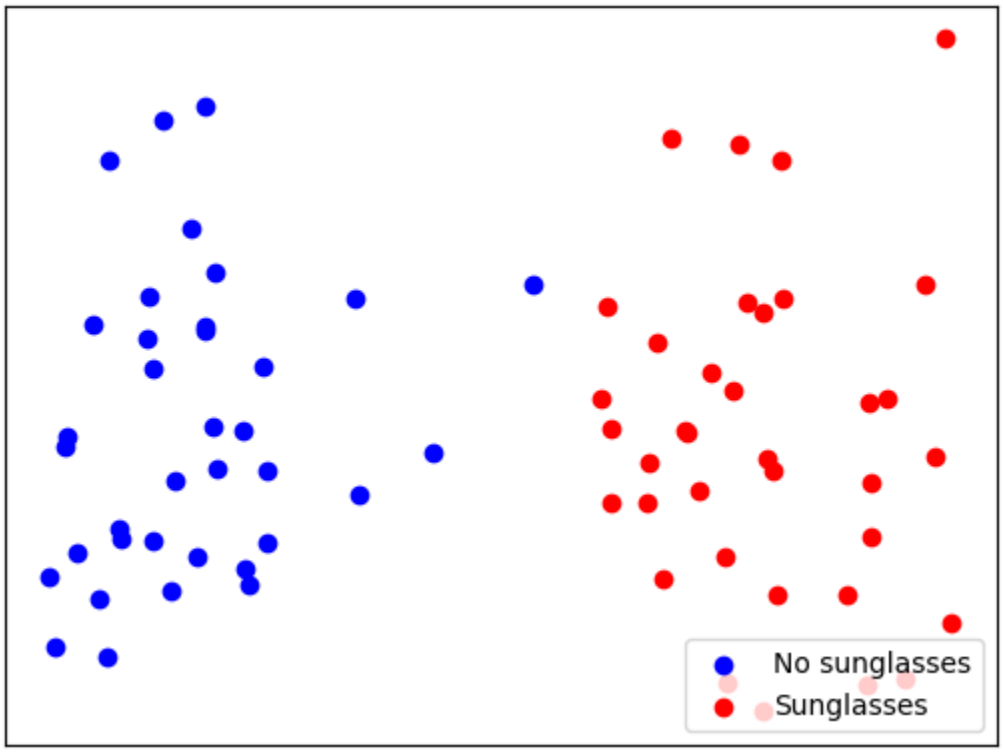
- Are the two groups separable in the two-dimensional latent space? Is this what you expected? Why or why not? (2 points)
- How do the plots for the three different attributes differ from each other? (2 points)

In [130…
```
def LatentSpace_2D(encoder_model, data1, label1, data2, label2):
    # Calculate the latent space encodings for two sets of faces
    data1_encoded = encoder_model.predict(data1)
    data2_encoded = encoder_model.predict(data2)
    data_encoded = np.concatenate((data1_encoded, data2_encoded), axis=0)

    # Use PCA to reduce the dimension of the latent space codes to two.
    pca = PCA(2).fit(data_encoded)
    pcs = pca.fit_transform(data_encoded)

    # Visualize the latent space after dimensionality reduction with a scatter plot
    labels = [label1, label2]
    colors = ['blue', 'red']
    y = [0] * len(data1) + [1] * len(data2)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    for i in np.arange(2):
        mask = i == y
        ax.scatter(pcs[mask, 0], pcs[mask, 1], c=colors[i], label=labels[i])
    ax.legend(loc='lower right')
    ax.set_xticks([])
    ax.set_yticks([])
```

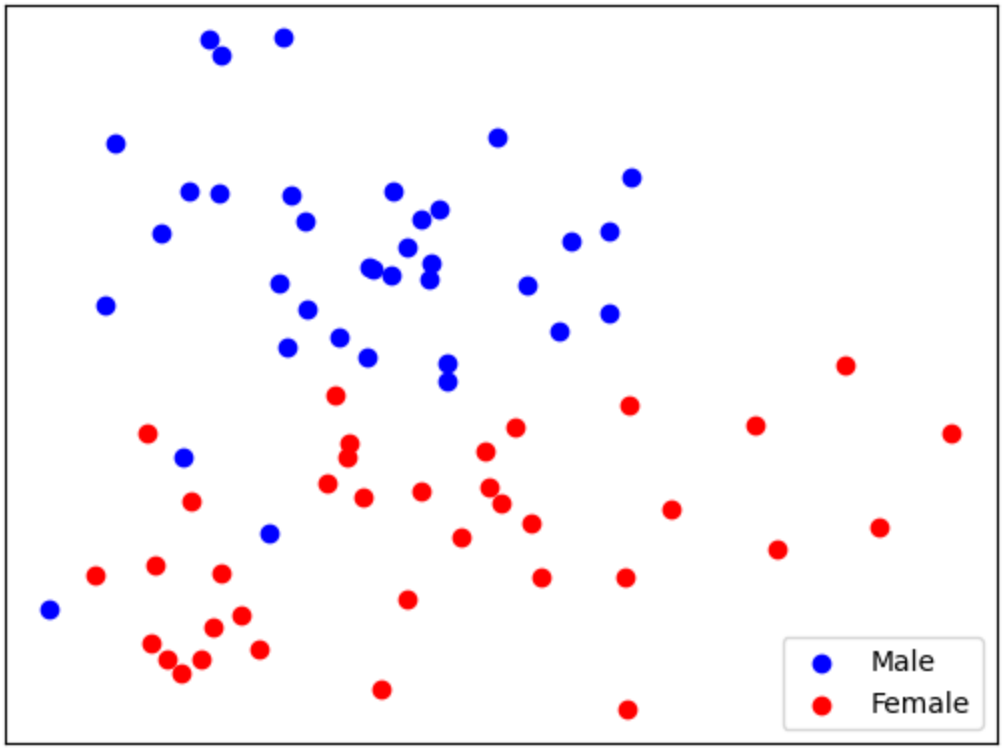
In [131…
```
LatentSpace_2D(encoder_model, smile_data, 'Smile', no_smile_data, 'No smile')
```



In [132…
```
LatentSpace_2D(encoder_model,smile_data,'No sunglasses',sunglasses_data,'Sunglasses
```



In [133…
```
LatentSpace_2D(encoder_model,male_data,'Male',female_data,'Female')
```



All categories show two groups that are relatively seperable in latent space. In terms of spatial seperation between the groups of interest, the sunglasses divide is greatest, the smile divide is second greatest and the gender divide is the least. This makes sense because it aligns with the magnitude of visual difference within each category. Its very visually obvious if someone is wearing sunglasses or not and more than that, its a binary state. There's a spectrum to smiles, so there's some ambiguity. Finally, whene it comes to gender, male and female faces can look pretty similar. Something unique about the gender data is that the area taken up by the female latent points seems much wider than the area taken up by the male latent points. I wonder if this is because there's more variety in feminine faces than masculine ones.

1.4 Morphing between faces (4 points)

Morph at least 5 pairs of faces with the function `morphBetweenImages` and comment on what you observe.

- Briefly explain how the morphing works. (2 points)
- Do the generated faces look like what you expected? Does any of the pairs work better than the others? If so, what kind of image pairs work better? (2 points)

In [134…

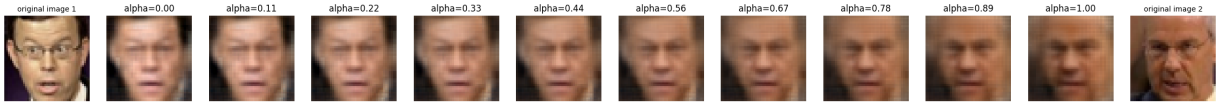
```
# Don't change the function
def morphBetweenImages(img1, img2, num_of_morphs):
    alpha = np.linspace(0,1,num_of_morphs)
    z1 = encoder_model.predict(np.array([img1]))
    z2 = encoder_model.predict(np.array([img2]))
    fig = plt.figure(figsize=(30,5))
```

```
    ax = fig.add_subplot(1, num_of_morphs+2, 1)
    ax.imshow(img1)
    ax.axis('off')
    ax.set_title(loc='center', label='original image 1',fontsize=10)

    for i in range(num_of_morphs):
        z = z1*(1-alpha[i]) + z2*alpha[i]
        new_img = decoder_model.predict(z)

        ax = fig.add_subplot(1, num_of_morphs+2, i+2)
        ax.imshow(np.clip(new_img.squeeze(),0,1))
        ax.axis('off')
        ax.set_title(loc='center', label='alpha={:.2f}'.format(alpha[i]))

    ax = fig.add_subplot(1, num_of_morphs+2, num_of_morphs+2)
    ax.imshow(img2)
    ax.axis('off')
    ax.set_title(loc='center', label='original image 2',fontsize=10)
    return
```

In [135…

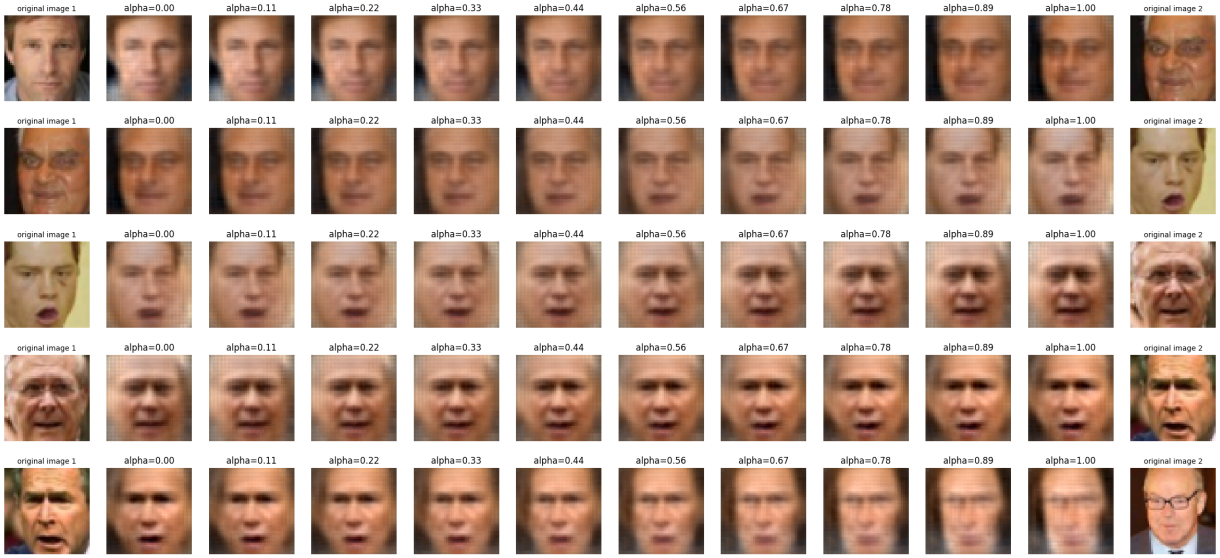
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

```
/home/jacob/lpd/classes/SDS365/IMLvenv/lib/python3.10/site-packages/keras/engine/training_v1.py:2067: UserWarning: `Model.state_updates` will be removed in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  updates=self.state_updates,
```



In [146…

```
morphBetweenImages(data[0000], data[1000],10)
morphBetweenImages(data[1000], data[2000],10)
morphBetweenImages(data[2000], data[3000],10)
morphBetweenImages(data[3000], data[4000],10)
morphBetweenImages(data[4000], data[5000],10)
```



The morphing function calculates a vector from one image to the other in the latent space. It then generates images using the latent values along that vector, starting with latent values close to the first image and moving toward latent values close to the second iamge.

The generated faces aren't exactly what I expected. I'm suprised that they all look like faces, because I didn't fully expect intermediate regions of the latent space to generate valid faces. The fact that they do shows the usefulness of VAE. I do notice though that the intermediate images are blurry. It seems that transforming between similar faces (like the two old men in the 4th transformation) work best. This makes sense, because these images are closest in the latent space so the latent distance between each intermediate frame can be less.

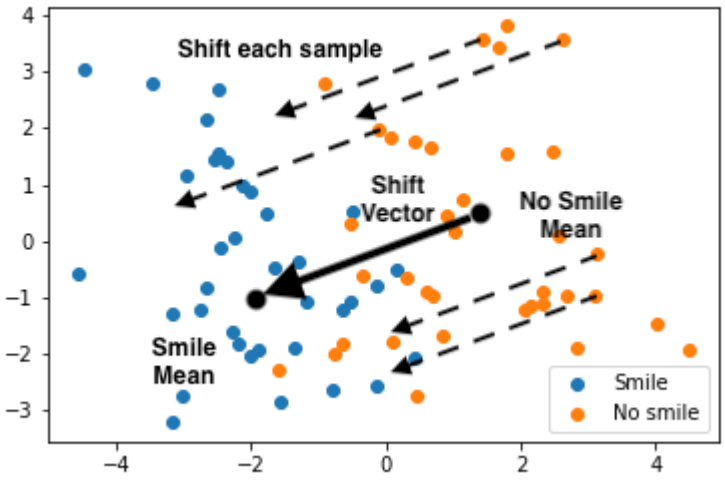
1.5 Attribute shift (10 points)

In 1.3, we've seen that faces with the same attributes form clusters in the latent space. In this problem, you will implement a function `AttributeShift` to change one attribute of the faces.

First implement the function `AttributeShift` . (5 points)

1. Calculate the latent space codes for two sets of faces that are different in one attribute, e.g. smile vs. no smile.
2. Calculate the mean latent space code for each group.
3. Get the attribute shifting vector by taking the difference between the two codes.
4. Perform attribute shift by adding the attribute shifting vector to the latent space code of the faces you want to modify.
5. Generate the image using the new latent space codes.

Here is a diagram demonstrating the shift in the latent space. Please note that the two-dimensional latent space is just for demonstration purpose. You should *not* use PCA in this problem. Instead, use the original latent space.



Perform attribute shift on at three attributes including smile. Comment on the faces with shifted attributes. (5 points)

- Do the generated faces look like what you expected? If not, can you think of some possible reasons?
- Do the faces with new attributes resemble the original faces? If not, can you think of some possible reasons?
- Which of the attribute shift is more successful? What are some possible reasons?
- Comment on any other aspects of your findings that are interesting to you.

In [152…

```
data1 = no_smile_data
data2 = smile_data

# calculate latent space codes
z1 = encoder_model.predict(data1)
z2 = encoder_model.predict(data2)

# calculate the mean of the latent space codes
z1_mean = np.mean(z1, axis=0)
z2_mean = np.mean(z2, axis=0)

# get the attribute shifting vector
# (shift from data1 to data2)
z_shift = z2_mean - z1_mean

# perform attribute shifting
z1_shifted = z1 + z_shift

# generate images from the shifted latent codes
data1_shifted = decoder_model.predict(z1_shifted)
```

In [157…

```
# Don't change this helper function!
def PlotAttributeShift(data2,pic_output):
    sample_index = random.sample(range(1, len(data2)), 16)

    fig, axs = plt.subplots(4, 8)
    fig.set_figheight(10)
    fig.set_figwidth(15)

    for i in range(4):
        for j in range(4):
            axs[i, 2*j].imshow(data2[sample_index[4*i+j], :, :, :])
            axs[i, 2*j].axis('off')
            axs[i, 2*j+1].imshow(np.clip(pic_output[sample_index[4*i+j]],0,1))
            axs[i, 2*j+1].axis('off')
```

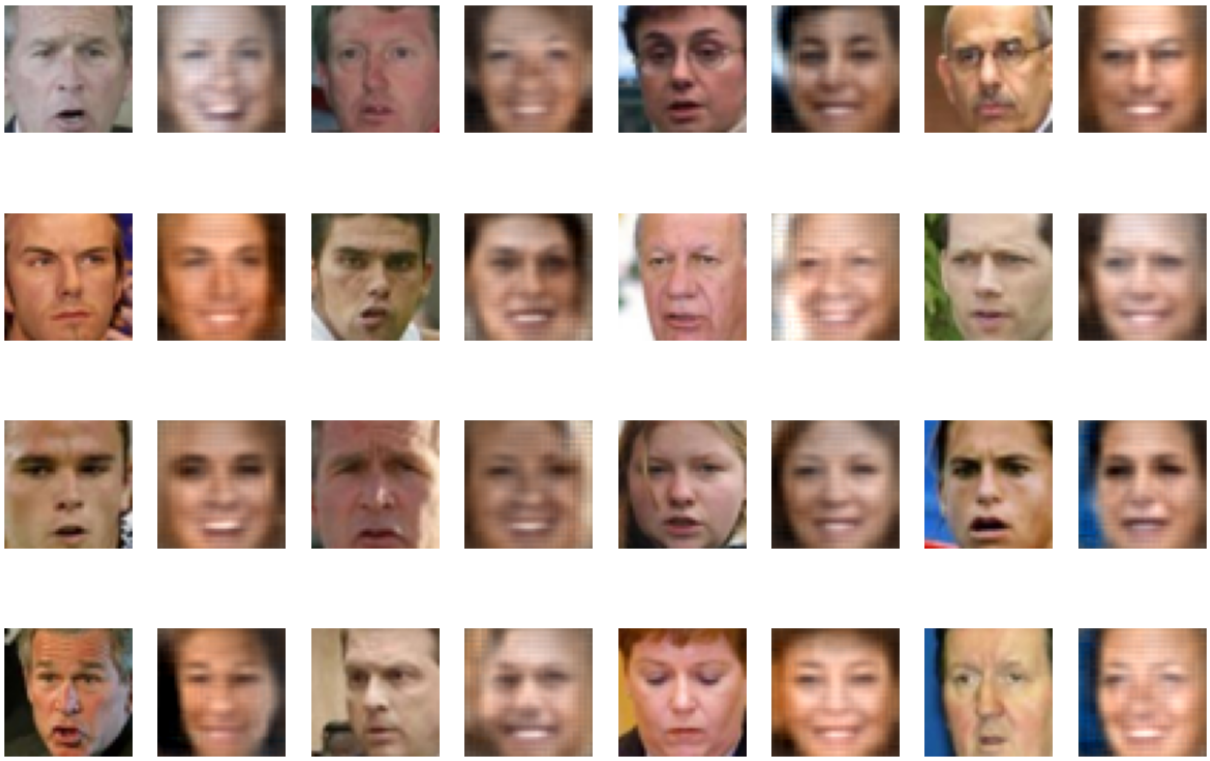
In [ ]:

```
# shift from data2 to data1
def AttributeShift(encoder_model,decoder_model,data1,data2):
    # calculate latent space codes
    z1 = encoder_model.predict(data1)
    z2 = encoder_model.predict(data2)

    # calculate the mean of the latent space codes
```

```
    z1_mean = np.mean(z1, axis=0)
    z2_mean = np.mean(z2, axis=0)

    # get the attribute shifting vector
    z_shift = z1_mean - z2_mean

    # perform attribute shifting
    z2_shifted = z2 + z_shift

    # generate images from the shifted latent codes
    pic_output = decoder_model.predict(z2_shifted)

    return pic_output
```

In [160…

```
pic_output = AttributeShift(encoder_model,decoder_model,smile_data,no_smile_data)
PlotAttributeShift(no_smile_data,pic_output)

pic_output = AttributeShift(encoder_model,decoder_model,sunglasses_data,no_smile_da
PlotAttributeShift(no_smile_data,pic_output)

pic_output = AttributeShift(encoder_model,decoder_model,makeup_data,no_smile_data)
PlotAttributeShift(no_smile_data,pic_output)
```



Across the board, all the generated faces are blurrier than the actual pictures. This makes sense because the generated pictures are from some intermediate area of the latent space where their potentially isn't as much data to refine the model, and the face is a combination of facial features. Also, across the board, the generated face always looked like a face, but often did not closely resemble the source face being shifted. To a certain extent this is realistic. For example, if someone goes from smiling to not smiling its not just their mouth that changes. They, for example, may look up and their cheeks may rise. The bigger changes

make sense because the latent space may not have a single axis for each characteristic, so that moving from one image to another is a shift across multiple axis and thus adjusts multiple different features, besides just the targeted one.

Generating a smile was succesful in that every picture generated actually was of someone smiling. Generating sunglasses was much less successful. The generated images seem to have darkened eyes that look more like black eyes than actual glasses. I'm guessing it struggled so much with glasses, because the difference between darkened eyes and glasses is subtle. Finally, adding makeup did pretty well. It's interesting that adding makeup also made the faces more feminine. This makes sense because there's probably high correlation between wearing makeup and being a women in the image set, so they might occupy similarly regions of the latent space.

1.6 Generating new faces (3 points)

Variational autoencoders can be used to generate new data; this is why they are generative models. We can sample new data points from the distribution in latent space and reconstruct new, fake faces based on them.

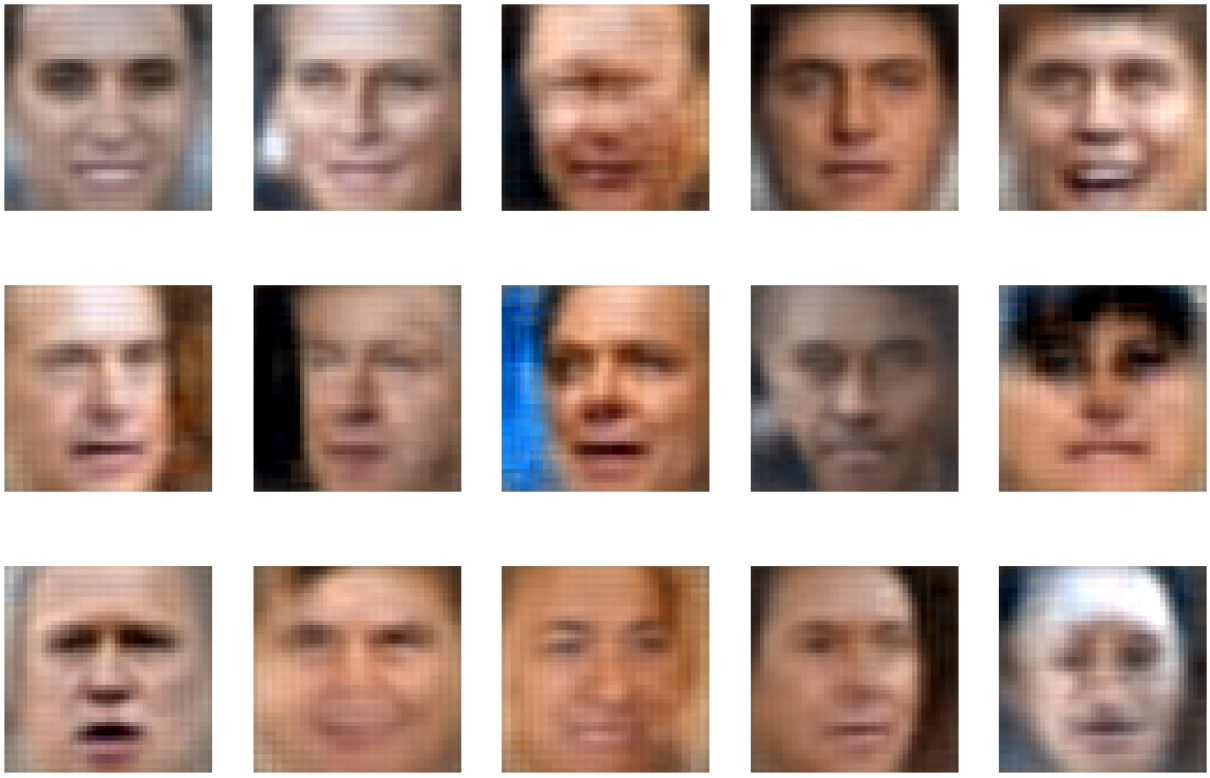
To draw a sample close to an existing sample in the latent space, we can add a scaled random sample from the normal distribution to the latent space code of an existing sample. The scalar, which we call the `noise_level` is a parameter that we can tune.

Run `GenerateFaces` with three different values of `noise_level` and comment on the generated faces. (3 points)

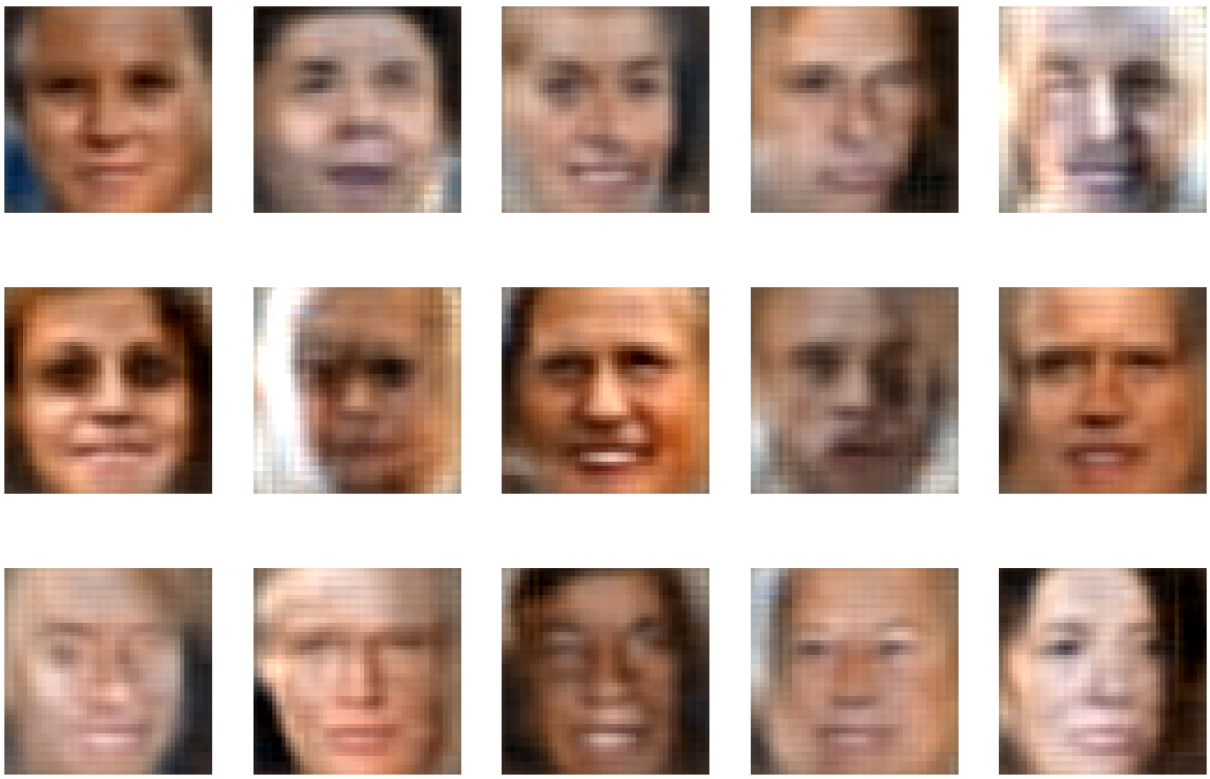
- Do the generated images look like faces?
- What happens when the new samples diverge more from the existing samples? What is a possible reason?

In [164…
```
def GenerateFaces(data, LATENT_SPACE_SIZE, noise_level):
    num_of_images = 15
    sample_index = random.sample(range(1, len(data)), num_of_images)
    latent_space = noise_level*np.random.normal(size=(num_of_images,LATENT_SPACE_SI
    generated_image = decoder_model.predict(latent_space)
    fig = plt.figure(figsize=(num_of_images,10))
    for i in range(num_of_images):
        ax = fig.add_subplot(3, 5, i+1)
        ax.imshow(np.clip(generated_image[i, :,:,:],0,1))
        ax.axis('off')
```

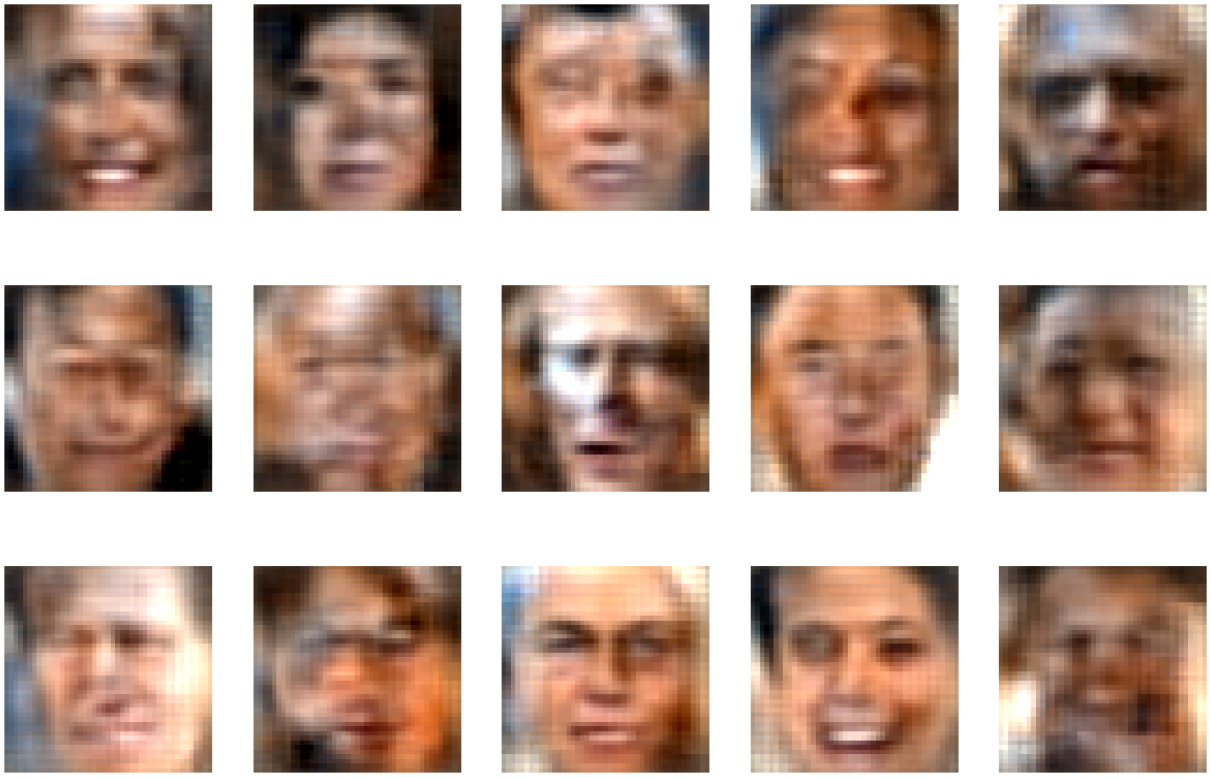
In [165…
```
GenerateFaces(data,LATENT_SPACE_SIZE,0.1)
```



In [166…
```
GenerateFaces(data,LATENT_SPACE_SIZE,0.5)
```



In [167…
```
GenerateFaces(data,LATENT_SPACE_SIZE,1)
```



All of the generated faces look like faces--they have distinguishable eyes, nose, and mouths. They appear increasingly deformed and varied as the noise parameter increases. With noise level 0.1 only 2 appear deformed, at 0.5 noise 4 appear deformed and at 1 noise they almost all do. The reason VAE are useful is that the latent space has a structure so that closeness in the latent space means similarity in the generated image. Therefore, it makes sense that as new samples diverge more and more from existing samples the generated images look less and less like faces.

Optional: Changing the loss function (2 points extra credit)

The variable `factor` is a parameter of the loss function. In this optional problem, you will play with it and think about its effect on the performance of VAE.

Retrain the model with a smaller factor. Repeat 1.2, 1.6 (using `noise_level` = 1) and latent space visualization. Comment on how the reconstruction results, generated new faces and latent space distributions change. (2 points)

- Which model reconstructs the faces better?
- Do the generated faces look different?
- How do the latent variable distributions differ?
- Do the differences make sense? Can you explain what you observed?

In [ ]:
```
# Your code here
```

[Your markdown here]

Problem 2: Are you Schur? (10 points)

The graphical lasso is based on conditional independence properties of Gaussian distributions. This problem asks you to reason about the graphs underlying a multivariate Gaussian.

Let $X = (Y, Z) \in \mathbb{R}^6 \sim N(0, \Sigma)$ be a random Gaussian vector where $Y = (Y_1, Y_2) \in \mathbb{R}^2$ and $Z = (Z_1, Z_2, Z_3, Z_4) \in \mathbb{R}^4$, with $\Sigma^{-1} = \Omega = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$ where

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ -1 & \frac{1}{2} & -\frac{1}{3} & \frac{1}{4} \end{pmatrix} \qquad C = \begin{pmatrix} 2 & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 2 & 0 & 0 \\ 0 & 0 & 2 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 2 \end{pmatrix}.$$

Problem 2.1

Draw the undirected graph of X, arranging the vertices in a hexagon. Explain your answer.

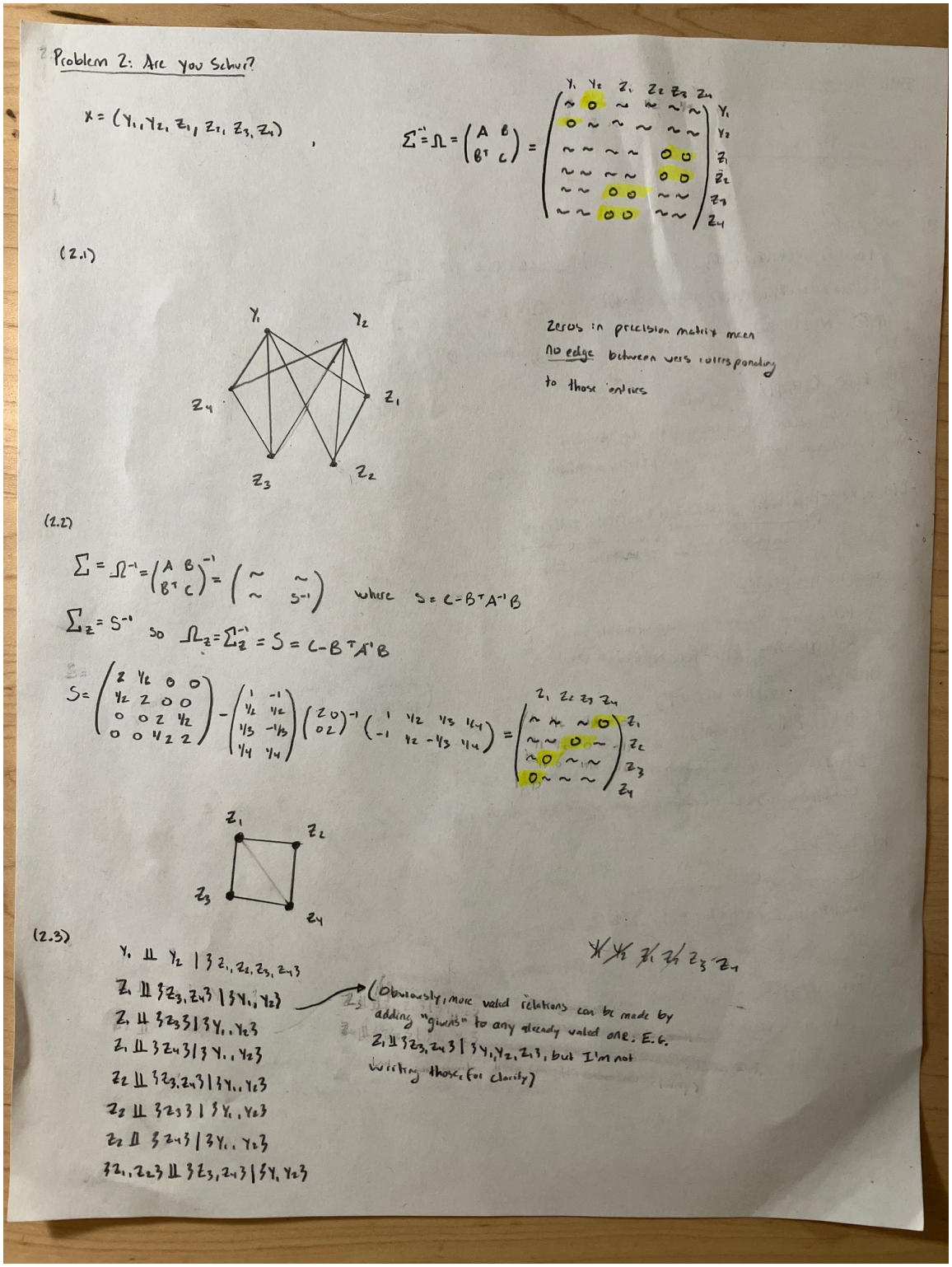
Problem 2.2

Draw the undirected graph of Z, arranging the vertices in a square. Explain your answer.
Hint: The Schur complement $S = C - B^T A^{-1} B$ has the property that

$$\begin{pmatrix} A & B \\ B^T & C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1} B S^{-1} B^T A^{-1} & -A^{-1} B S^{-1} \\ -S^{-1} B^T A^{-1} & S^{-1} \end{pmatrix}$$

Problem 2.3

Write down the set of all conditional independence relations on the undirected graph of X.



Problem 3: Taking stock (15 points)

A joint distribution of data has a natural graph associated with it. When the distribution is multivariate normal, this graph is encoded in the pattern of zeros and non-zeros in the inverse of the covariance matrix, also known as the "precision matrix."

In class we demonstrated the graphical lasso for estimating the graph on ETF data. In this

problem you will construct two different "portolios" of stocks, and run the graphical lasso to estimate a graph, commenting on your results.

All of the code you might need for this is contained in the demo.

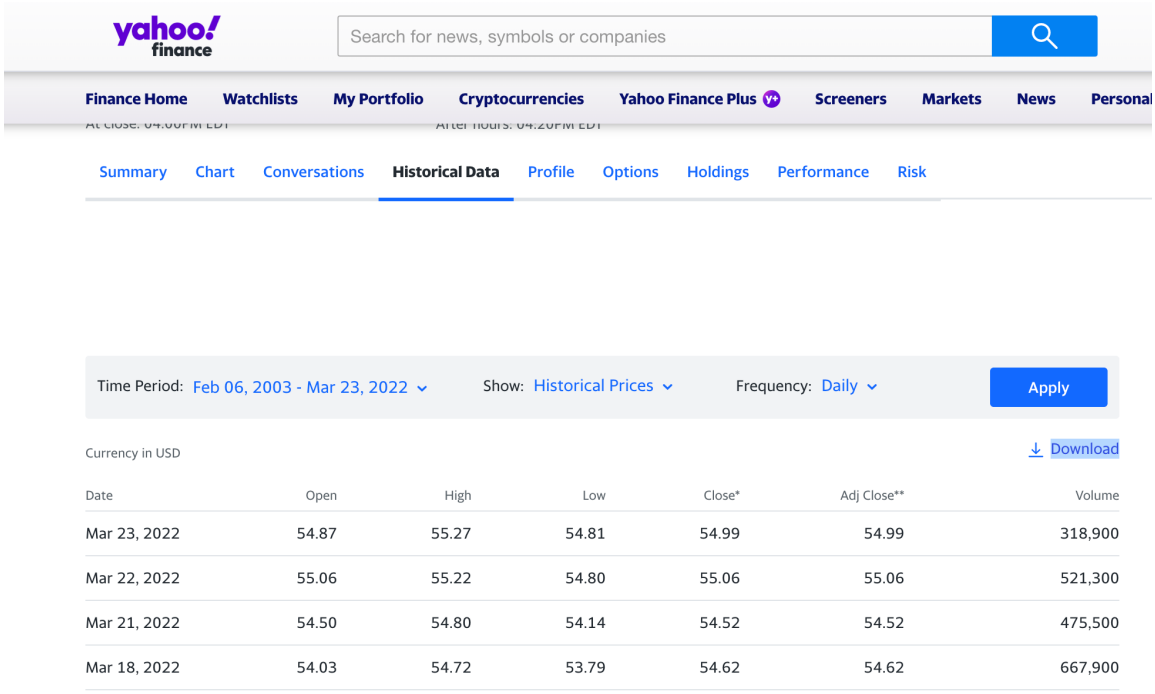
Downloading data

As demonstrated in class, you will run on equities data downloaded from Yahoo finance. Your job is to construct two "portfolios" of stocks, each of which has some kind of organization to it. For example, in one portfolio you might have 5 energy stocks, 5 tech stocks, 5 consumer staples stocks, and 5 ETF stocks. Each portfolio should have at least 20 stocks.

The page https://en.wikipedia.org/wiki/List_of_S%26P_500_companies has GICS sectors, which you may find useful for the glasso problem.

To download the data, follow the procedure outlined below (and discussed briefly in class):

- Search on a ticker symbol, like EZA, using https://finance.yahoo.com/quote/EZA/history?p=EZA
- Select the range of the query, the frequency (daily, weekly, or monthly) and then issue the query. This will give you results like this:



Date	Open	High	Low	Close*	Adj Close**	Volume
Mar 23, 2022	54.87	55.27	54.81	54.99	54.99	318,900
Mar 22, 2022	55.06	55.22	54.80	55.06	55.06	521,300
Mar 21, 2022	54.50	54.80	54.14	54.52	54.52	475,500
Mar 18, 2022	54.03	54.72	53.79	54.62	54.62	667,900

- Next, hover over the Download link, and grab the URL. In this case it gives https://query1.finance.yahoo.com/v7/finance/download/EZA?period1=1044576000&period2=1648080000&interval=1d&events=history&includeAdjuste
- Then, you can use this same URL, but swap in different ticker symbols, to get the corresponding data for range of companies or funds.

In [ ]:
```
# data downloaded from https://www.nasdaq.com/market-activity/stocks/[STOCKSYMBOL]/
# daily stock price data for the past year

communication_stocks = ['GOOGL', 'T', 'CMCSA', 'FOX', 'EA']
```

```
consumer_stocks = ['AMZN', 'ABNB', 'BBY', 'COST', 'CMG']
financial_stocks = ['AXP', 'AFL', 'ALL', 'BLK', 'BAC']
healthcare_stocks = ['DXCM', 'LLY', 'PFE', 'ISRG', 'JNJ']

portfolio1 = communication_stocks + consumer_stocks
portfolio2 = financial_stocks + healthcare_stocks

dir = "/home/jacob/classes/SDS365/assn3/market_data/"
```

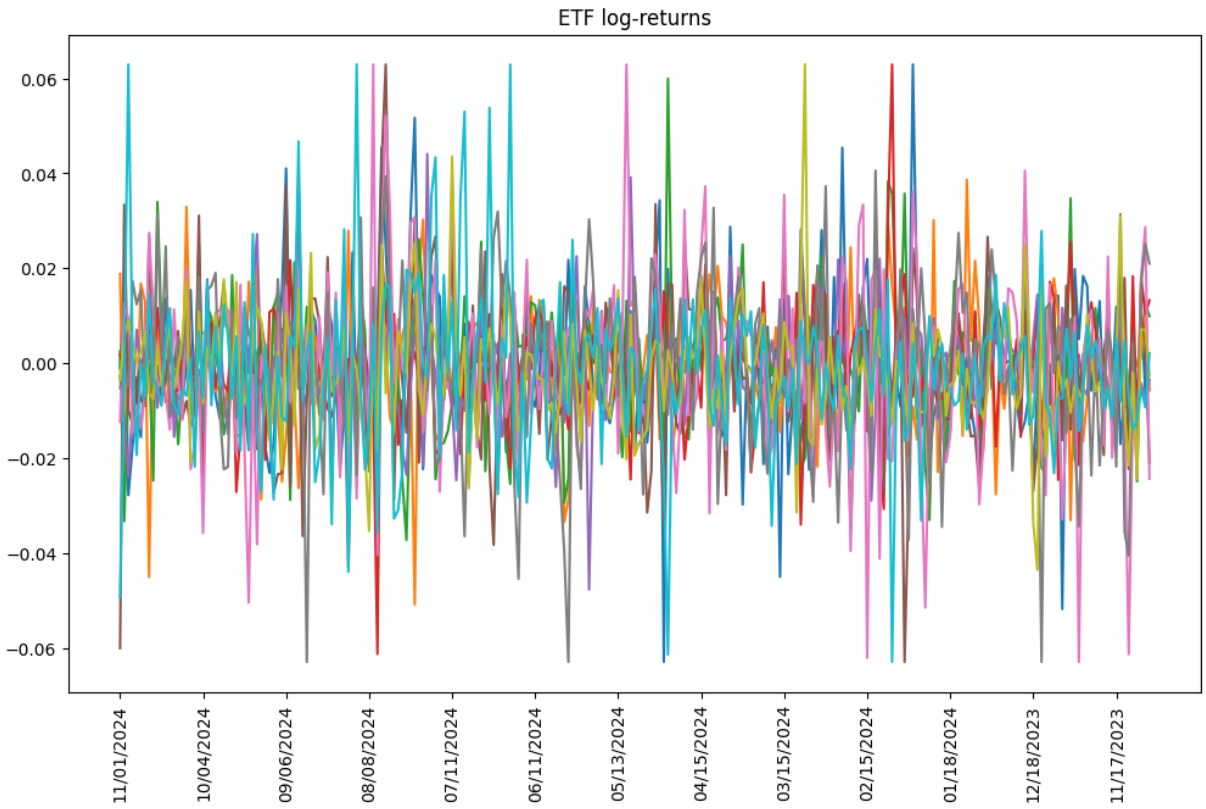
Portfolio 1

In [245…

```
portfolio = portfolio1

# Load Data
path = dir + portfolio[0] + '.csv'

stock_df = pd.read_csv(path)
df = pd.DataFrame()
dates = stock_df['Date']

for stock in portfolio:
    path = dir + stock + '.csv'
    stock_df = pd.read_csv(path, converters={'Close/Last': lambda x: float(x.replac
    df[stock] = stock_df['Close/Last']

df.index = dates
```

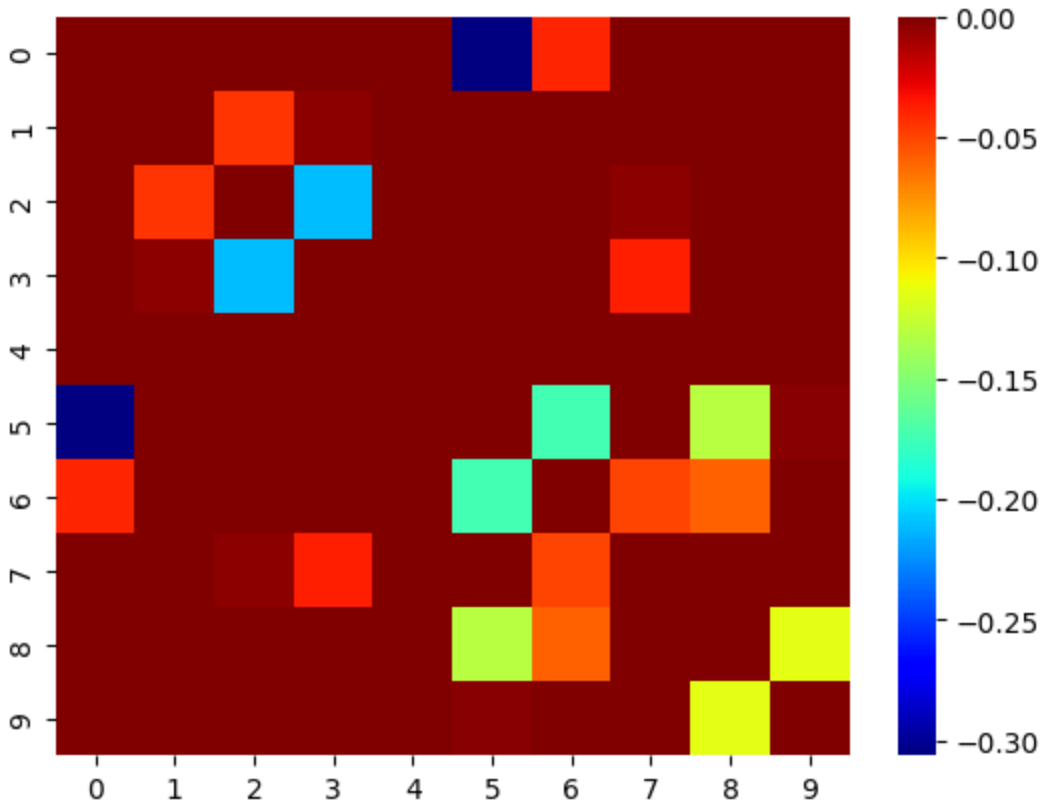
In [247…

```
# normalize the data

# calculate log-return
dflogreturn = np.log1p(df.pct_change()).iloc[1:]

# clip outliers
logreturns = np.array(np.log1p(df.pct_change()).iloc[1:])
clipped_logreturns = np.maximum(logreturns, -4*np.std(logreturns))
clipped_logreturns = np.minimum(clipped_logreturns, 4*np.std(logreturns))

# plot results
plt.figure(figsize=(12, 7))
plt.plot(clipped_logreturns)
plt.xlabel('')
plt.xticks(ticks=np.arange(0, len(dates), 20), labels=np.array(dates)[np.arange(0,
_ = plt.title('ETF log-returns')
```



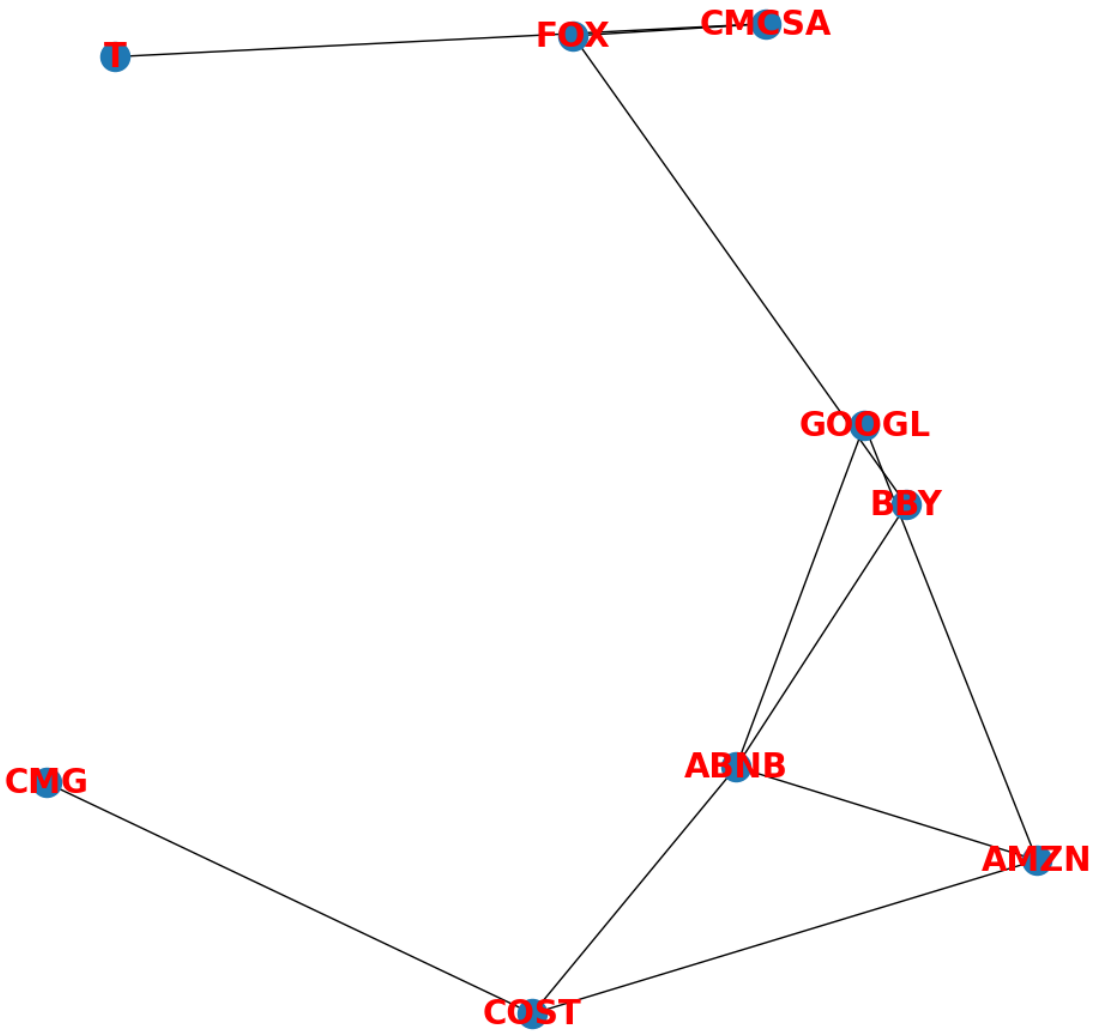
In [261…

```
# Precision matrix
X = np.array(clipped_logreturns)
p = X.shape[1]
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
glasso = GraphicalLasso(alpha=.25).fit(X)
Omegahat = np.around(glasso.precision_, decimals=5)

for j in np.arange(p):
    Omegahat[j,j] = 0
sns.heatmap(Omegahat, cmap='jet')
plt.show()
```



In [262…

```
np.random.seed(42)

precdf = pd.DataFrame(Omegahat, columns=portfolio, index=portfolio)
links = precdf.stack().reset_index()
links.columns = ['var1', 'var2','value']
links=links.loc[(abs(links['value']) > 0.01) &  (links['var1'] != links['var2']) ]

#build the graph using networkx lib
G=nx.from_pandas_edgelist(links,'var1','var2', create_using=nx.Graph())
pos = nx.spring_layout(G, k=1.7/np.sqrt(len(G.nodes())), iterations=20)
plt.figure(3, figsize=(10, 10))
nx.draw(G, pos=pos)
nx.draw_networkx_labels(G, pos=pos, font_color='red', font_size=20, font_weight='bo
plt.show()
```



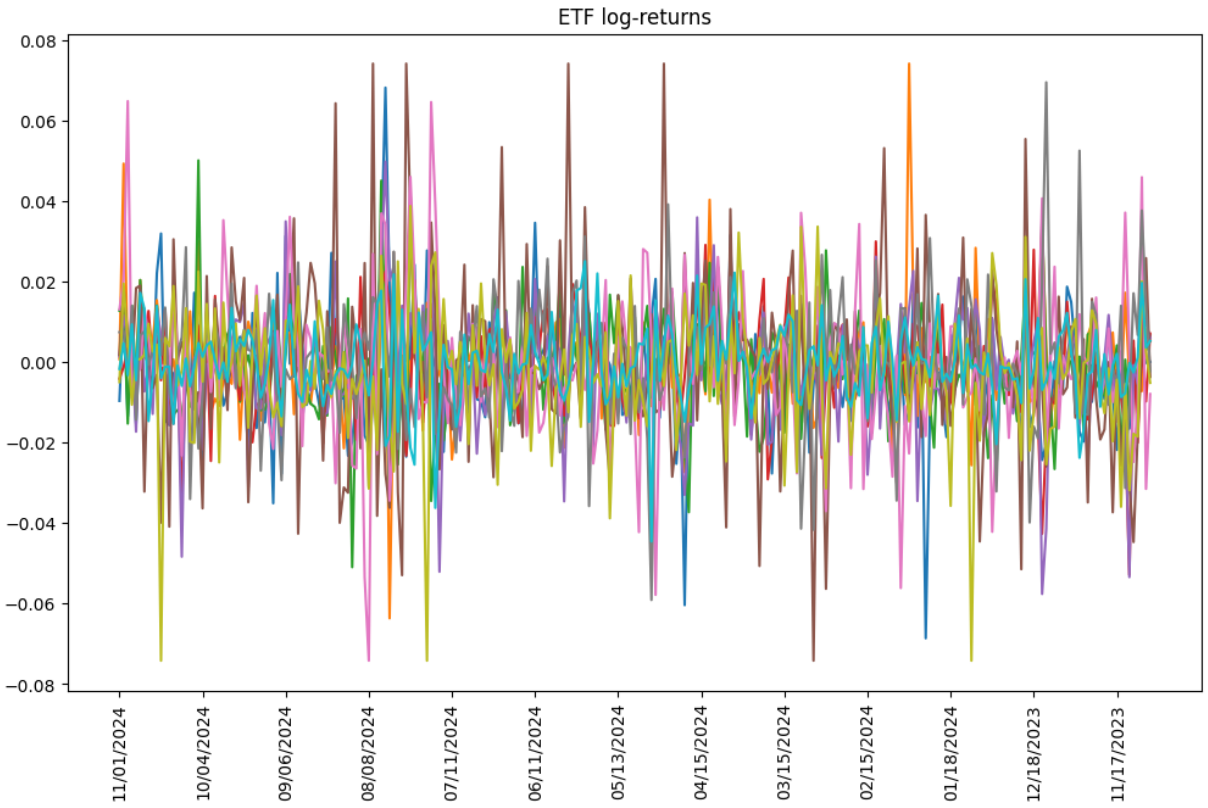
Portfolio 2

In [239…

```
portfolio = portfolio2

# Load Data
path = dir + portfolio[0] + '.csv'

stock_df = pd.read_csv(path)
df = pd.DataFrame()
dates = stock_df['Date']

for stock in portfolio:
    path = dir + stock + '.csv'
    stock_df = pd.read_csv(path, converters={'Close/Last': lambda x: float(x.replac
    df[stock] = stock_df['Close/Last']

df.index = dates
```

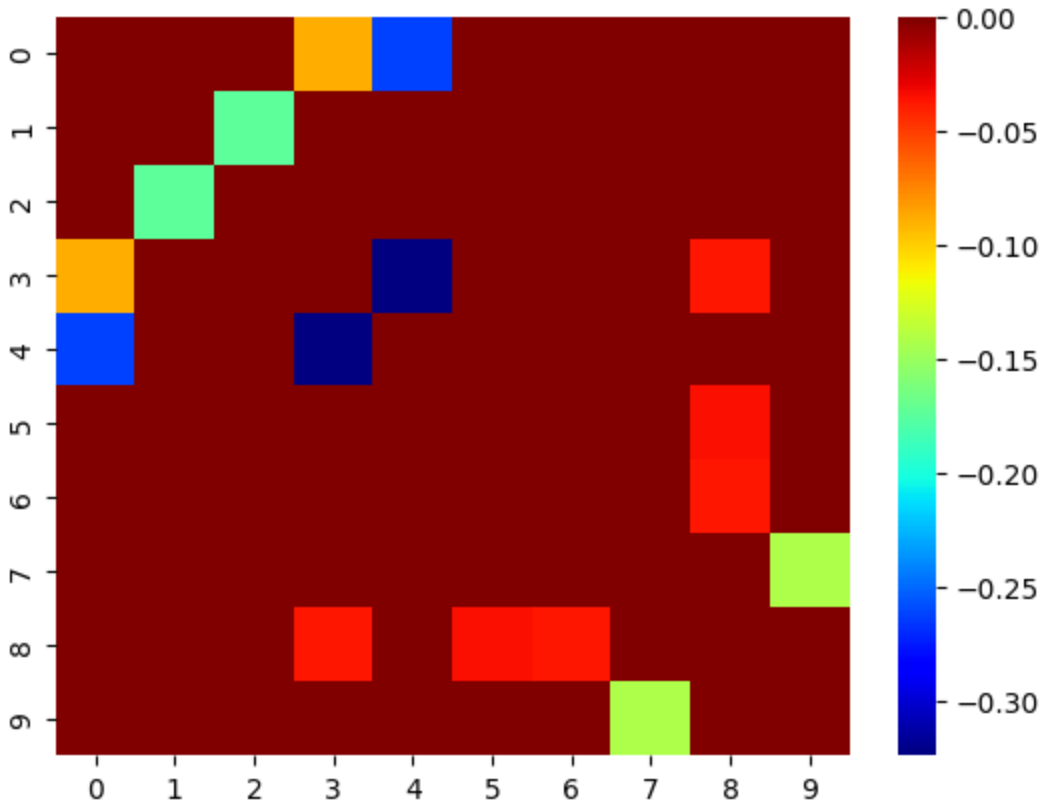
In [240…

```
# normalize the data

# calculate log-return
dflogreturn = np.log1p(df.pct_change()).iloc[1:]

# clip outliers
logreturns = np.array(np.log1p(df.pct_change()).iloc[1:])
clipped_logreturns = np.maximum(logreturns, -4*np.std(logreturns))
clipped_logreturns = np.minimum(clipped_logreturns, 4*np.std(logreturns))

# plot results
plt.figure(figsize=(12, 7))
plt.plot(clipped_logreturns)
plt.xlabel('')
plt.xticks(ticks=np.arange(0, len(dates), 20), labels=np.array(dates)[np.arange(0,
_ = plt.title('ETF log-returns')
```



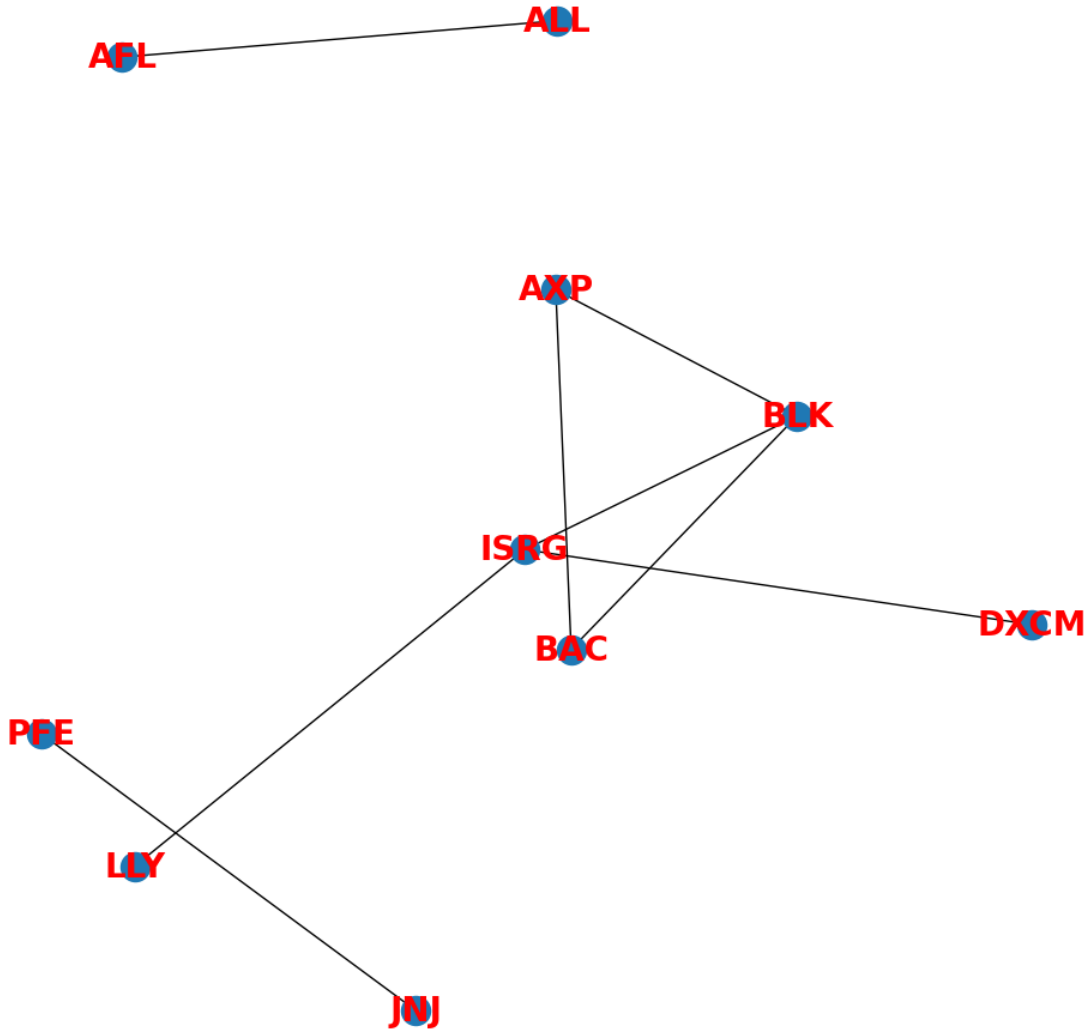
In [241…

```
# Precision matrix
X = np.array(clipped_logreturns)
p = X.shape[1]
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
glasso = GraphicalLasso(alpha=.3).fit(X)
Omegahat = np.around(glasso.precision_, decimals=5)

for j in np.arange(p):
    Omegahat[j,j] = 0
sns.heatmap(Omegahat, cmap='jet')
plt.show()
```



In [242…

```
np.random.seed(42)

precdf = pd.DataFrame(Omegahat, columns=portfolio, index=portfolio)
links = precdf.stack().reset_index()
links.columns = ['var1', 'var2','value']
links=links.loc[(abs(links['value']) > 0.01) &  (links['var1'] != links['var2']) ]

#build the graph using networkx lib
G=nx.from_pandas_edgelist(links,'var1','var2', create_using=nx.Graph())
pos = nx.spring_layout(G, k=1.7/np.sqrt(len(G.nodes())), iterations=20)
plt.figure(3, figsize=(10, 10))
nx.draw(G, pos=pos)
nx.draw_networkx_labels(G, pos=pos, font_color='red', font_size=20, font_weight='bo
plt.show()
```


ALL
AFL
AXP
BLK
ISRG
DXCM
BAC
PFE
LLY
JNJ

Analyzing your portfolios

Your task is to analyze each porfolio using the graphical lasso, and comment on your findings. Here are the types of questions you should address:

- How did you choose the portolio? How did you choose the date range and frequency (daily, weekly, etc.)? Remember, each of the portfolios must contain at least 20 stocks, and be organized in some reasonable way.
- Display the graph obtained with the graphical lasso, using networkx. How did you choose the regularization level? Does the structure of the graph make sense? Is it sensitive to the choice of regularization level? Is this the structure you expected to see when you designed the portfolio? Why or why not?
- What are some of the conditional independence assumptions implied by the graph? Are some parts of the graph more densely connected than others? Why?

I chose portfolio one to have stocks of companies related to communication and consumer discretionary spending. I chose portfolio two to have stocks of companies related to finance and healthcare. I wanted diversified industries. Within each industry, I selected stocks from companies in the S&P500 (big companies are less risky) and selected companies I recognized, so that I know what I'm investing in. I choose a year for the data range, because that offered around 250 datapoints which seemed high enough to identify patterns. I used daily stock numbers because that's the data I most readily had access to.

For both portfolios, the graph was sensitive to the choice of regularizer. I selected the regularization level to be low enough for clusters to appear but not so low that everything is connected, and not so high that nothing is connected at all.

Porfolio 1

- 5 communication stocks: GOOGL (google), T (AT&T), CMCSA (Comcast), FOX (Fox), EA (Electronic Arts)
- 5 consumer discretionary stocks: AMZN (Amazon), ABNB (Airbnb), BBY (Best Buy), COST (Costco), CMG (Chipotle)

Portfolio one shows a main cluster containing ABNB, AMZN, BBY, and GOOGL. It makes sense for Amazon and Google to be together in a main cluster, as they are goliath companies that I would expect to correlate with many different business. I was a bit more suprised though by how central best buy (BBY) was in the graph, because I think of it as a more niche electronics store. One interesting independence relation implied is that T (AT&T) is independent of CMCSA (comcast) given FOX (fox). I would have thought T and CMSCA would be directly connected because of how similar their business are.

Portfolio 2

- 5 financial stocks: AXP (American Express), AFL (Aflac), ALL (Allstate), BLK (BlackRock), BAC (Bank of America)
- 5 healthcare stocks: DXCM (Dexcom), LLY (Eli Lilly), PFE (Pfizer), ISRG (Intuitive Surgical), JNJ (Johnson & Johnson)

Portfolio 2 shows a couple groupings. 1) AFL and ALL, which makes sense as they're both insurance companies. 2) PFE, JNJ which make sense because they're both drug companies. 3) AXP, BLK, BAC which makes sense as they're all banking companies, and finally 4) LLY, ISRG, and DXCM which make sense as they're all medtech companies. Its interesting how the graph showed 4 important categories of stocks, versus the traditional "financial" vs "healthcare" labelling which only gives 2 categories. The banking companies are especially densly connected, suggesting some stronger amount of coupling between them.