In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from tqdm.notebook import tqdm
warnings.filterwarnings("ignore")
%matplotlib inline

import tensorflow as tf
from keras.preprocessing.image import load_img
from keras.models import Sequential, Model
from keras.layers import (Dense,Conv2D,Dropout,Flatten,)
```

In [ ]:

```
#lables, age, gender, ethnicity
image_paths = []
age_labels = []
gender_labels =[]

for filename in tqdm(os.listdir(BASE_DIR)):
    image_path = os.path.join(BASE_DIR, filename)
    temp = filename.split('_')
    age = int(temp[0])
    gender = int(temp[1])
    image_paths.append(image_path)
    age_labels.append(age)
    gender_labels.append(gender)
```

In [ ]:

```
#convert to dataframe
df = pd.DataFrame()
df['image'], df['age'],df['gender'] = image_paths, age_labels, gender_labels
df.head()
```

In [ ]:

```
# map labels for gender

gender_dict = (0:'Male', 1:'Female')
```

In [ ]:

```
# display picture size
from PIL import Image
img = Image.open(df['image'][0])
plt.axis('off')
plt.imshow(img);
```

In [ ]:

```
#plot the differences in age from the dataset
sns.distplot(df['age'])
```

In [ ]:

```
# display the number of males to females from the dataset

sns.countplot(df['gender'])
```

In [ ]:

```
# display grid of images
plt.figure(figsize=(20,20))
files = df.iloc[0:25]
```

Loading [MathJax]/extensions/Safe.js

```
for index, file, age, gender in files.itertuples():
    plt.subplot(5,5, index+1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title(f'Age:(age) Gender : (gender_dict[gender])')
    plt.axis('off')
```

In [ ]:

```
### Feature Extraction
```

In [ ]:

```
def extract_features(images):
    for image in tqdm(images):
        img = load_img(image, grayscale=True)
        img = img.resize((128,128), Image.ANTIALIAS)
        img = np.array(img)
        features.append(img)

        features = np.array(features)
        # ignore this step if using RGB

        features = features.reshape(len(features). 128,128, 1)
        return features
```

In [ ]:

```
X = extract_features(df['image'])
```

In [ ]:

```
X.shape
```

In [ ]:

```
# normalize the images
X = X/255.0
```

In [ ]:

```
y_gender = np.array(df['gender'])
y_age = np.array(df['age'])
```

In [ ]:

```
input_shape = (128,128, 1)
```

In [ ]:

```
### Model Creation
```

In [ ]:

```
inputs = Input((input_shape))

# conv layers
conv_1 = Conv2D(32, kernel_size=(3,3), activation='relu') (inputs)
maxp_1 = MaxPooling2D(pool_size=(2,2)) (conv_1)

conv_2 = Conv2D(64, kernel_size=(3,3), activation='relu') (maxp_1)
maxp_2 = MaxPooling2D(pool_size=(2,2)) (conv_2)

conv_3 = Conv2D(128, kernel_size=(3,3), activation='relu') (maxp_2)
maxp_3 = MaxPooling2D(pool_size=(2,2)) (conv_3)

conv_4 = Conv2D(256, kernel_size=(3,3), activation='relu') (maxp_3)
maxp_4 = MaxPooling2D(pool_size=(2,2)) (conv_4)
```

```
conv_5 = Conv2D(512, kernel_size=(3,3), activation='relu') (maxp_4)
maxp_5 = MaxPooling2D(pool_size=(2,2)) (conv_5)

#flattern the layers

flattern = Flattern() (maxp_5)
```

In [ ]:

```
# connect layers fully
dense_1 = Dense(256, activation='relu') (flattern)
dense_2 = Dense(256, activation='relu') (flattern)

dropout_1 = Dropout(0,3) (dense_1)
dropout_2 = Dropout(0,3) (dense_2)

output_1 = Dense(1, activation= 'sigmoid', name='gender_out') (dropout_1)
output_2 = Dense(1, activation= 'relu', name='age_out') (dropout_2)

#compile the model

model = model(inputs=[inputs], outputs=[output_1,output_2])

model.compile(loss=['binary_crossentropy', 'mae'], optimizer='adam', metrics=['accuracy'])
```

In [ ]:

```
#get the summary of the model
model.summary()
```

In [ ]:

```
#plot the model
from tensorflow.keras.utils import plot_model
plot_model(model)
```

In [ ]:

```
### Train the model
```

In [ ]:

```
result = model.fit(x=X, y=[y_gender, y_age], batch_size=64, epochs=15, validation_split=0
```

In [ ]:

```
### Display the results
```

In [ ]:

```
#plot diagram for gender

accu = result.result['gender_out_accuracy']
val_acc = result.result['val_gender_out_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='validation Accuracy')
plt.title('Accuracy Graph')
plt.legend()
plt.figure()

loss = result.result['gender_out_loss']
val_loss = result.result['val_gender_out_loss']


plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='validation Loss')
plt.title('Loss Graph')
```

Loading [MathJax]/extensions/Safe.js

```
plt.legend()
plt.show()
```

In [ ]:

```
#plot diagram for age
loss = result.result['age_out_loss']
val_loss = result.result['val_age_out_loss']
epochs = range(len(acc))


plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='validation Loss')
plt.title('Loss Graph')
plt.legend()
plt.show()
```

In [ ]:

```
### prediction
```

In [ ]:

```
image_index = 230
print('original Gender:', gender_dict[y_gender[image_index]], 'original Age', y_age[image_

#predict using model
pred = model.predict(X[image_index].reshape(1,128,128,1))
#predict gender
pred_gender = gender_dict[round(pred[0][0][0])]
#predict age
pred_age = round(pred[1][0][0])

print('Predicted Gender:', pred_gender, 'Predicted Age:', pred_age)
plt.imshow(X[image_index].reshape(128, 128), cmap='gray');
plt.axis('off')
```