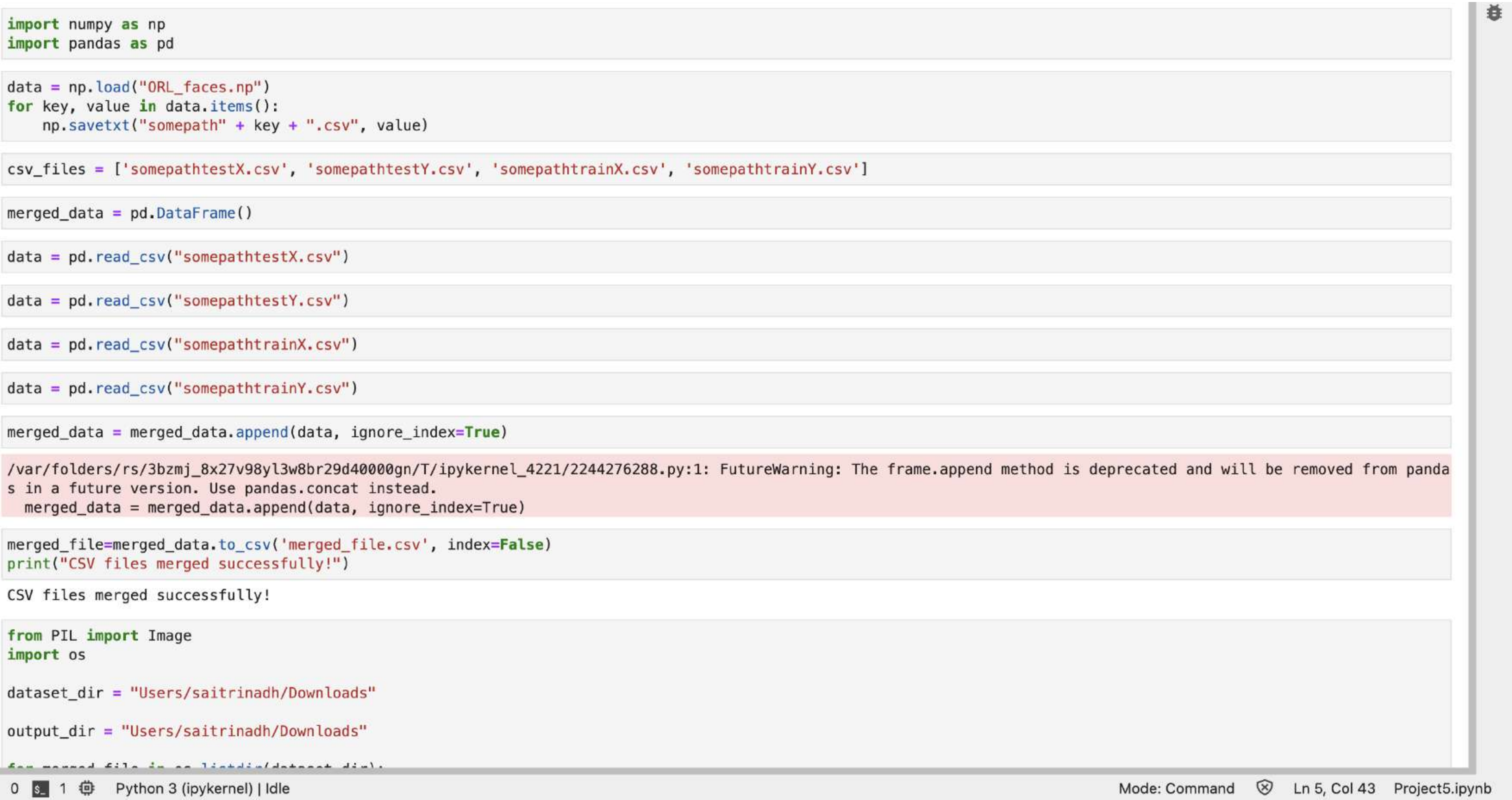


```
import numpy as np
import pandas as pd
```

```
data = np.load("ORL_faces.np")
for key, value in data.items():
    np.savetxt("somepath" + key + ".csv", value)
```

```
csv_files = ['somepathtestX.csv', 'somepathtestY.csv', 'somepathtrainX.csv', 'somepathtrainY.csv']
```

```
merged_data = pd.DataFrame()
```

```
data = pd.read_csv("somepathtestX.csv")
```

```
data = pd.read_csv("somepathtestY.csv")
```

```
data = pd.read_csv("somepathtrainX.csv")
```

```
data = pd.read_csv("somepathtrainY.csv")
```

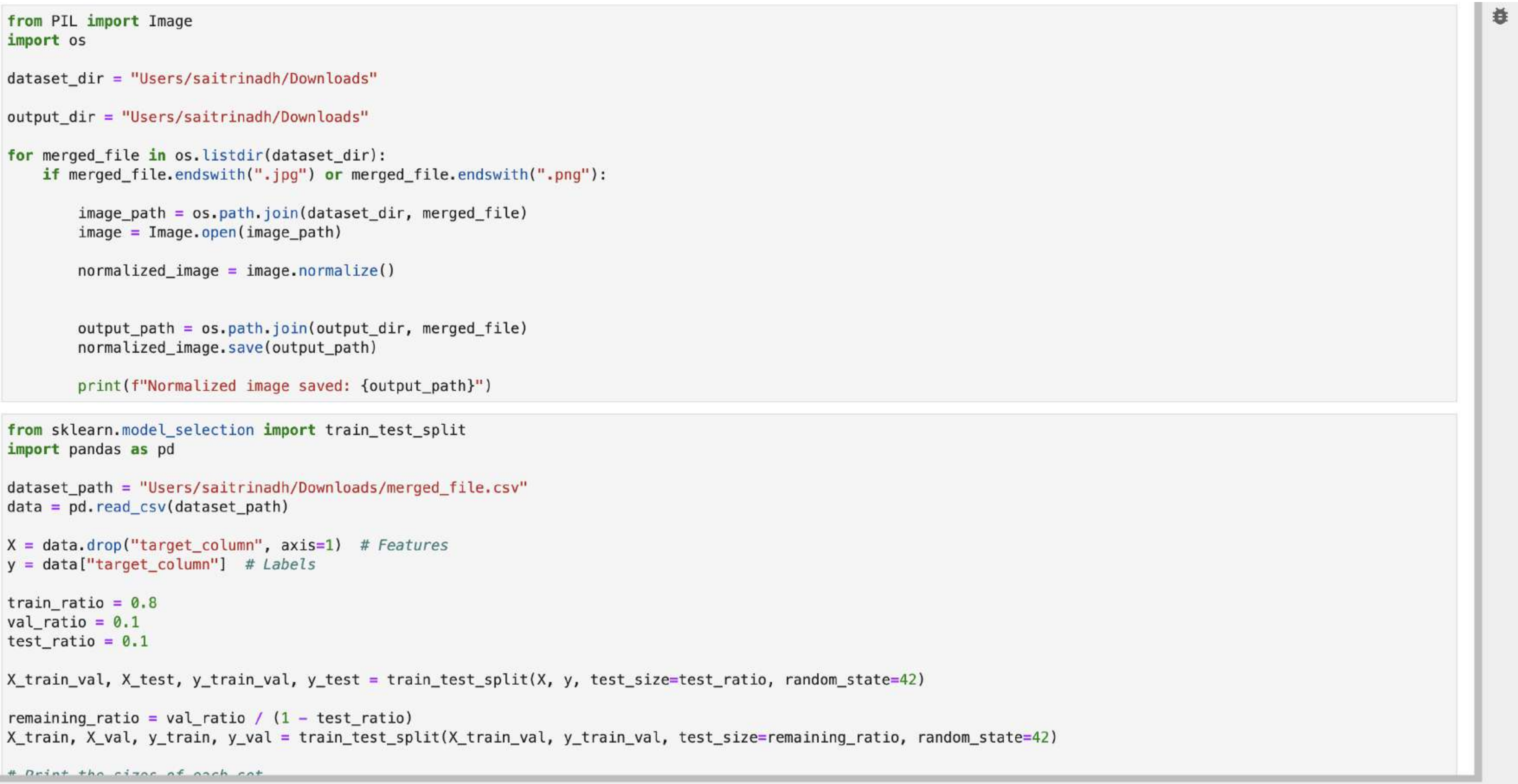
```
merged_data = merged_data.append(data, ignore_index=True)
```

```
/var/folders/rs/3bzmj_8x27v98yl3w8br29d40000gn/T/ipykernel_4221/2244276288.py:1: FutureWarning: The frame.append method is deprecated and will be removed from panda
s in a future version. Use pandas.concat instead.
  merged_data = merged_data.append(data, ignore_index=True)
```

```
merged_file=merged_data.to_csv('merged_file.csv', index=False)
print("CSV files merged successfully!")
```

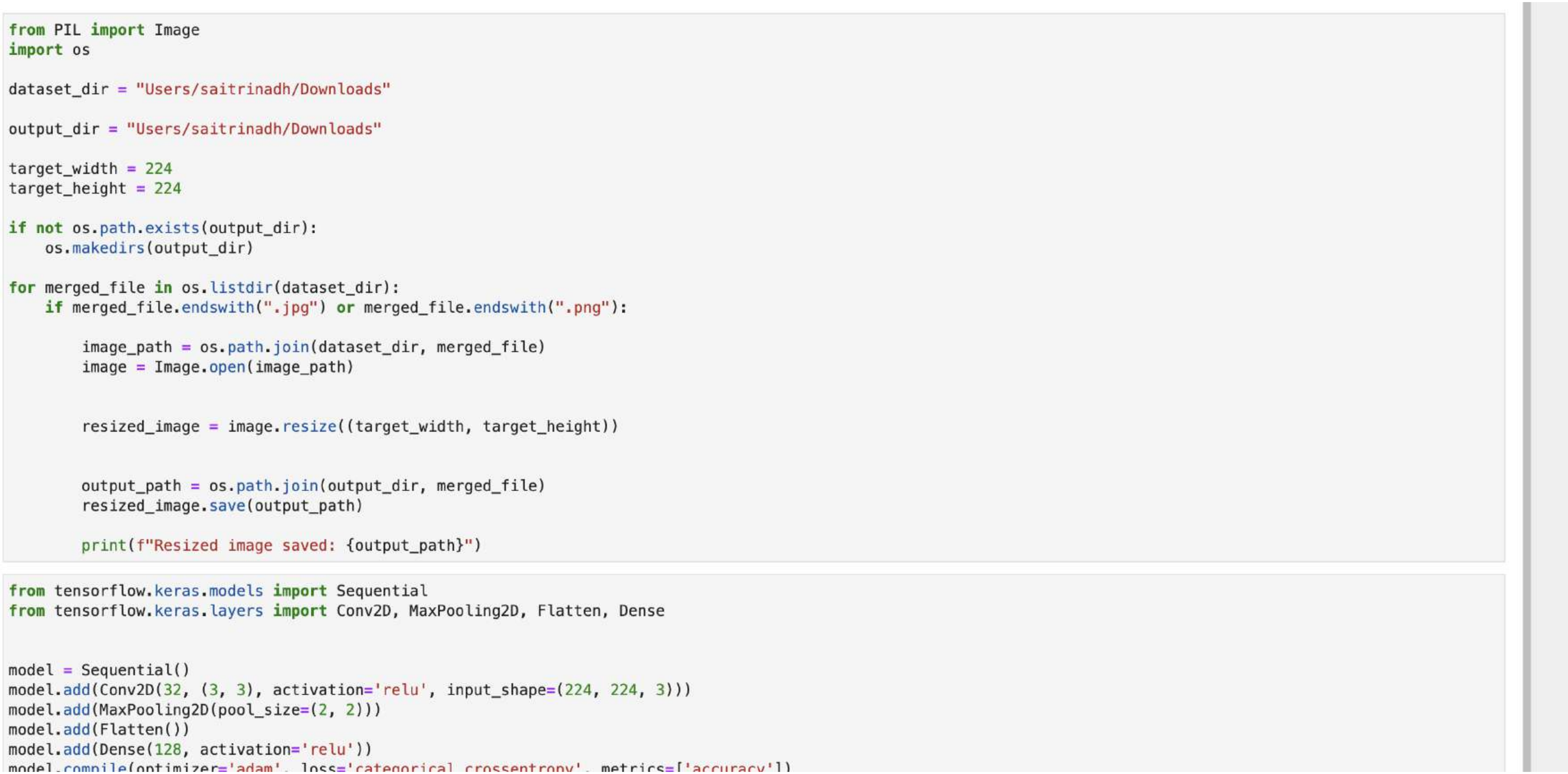
```
CSV files merged successfully!
```

```
from PIL import Image
import os

dataset_dir = "Users/saitrinadh/Downloads"

output_dir = "Users/saitrinadh/Downloads"

for merged_file in os.listdir(dataset_dir):
```



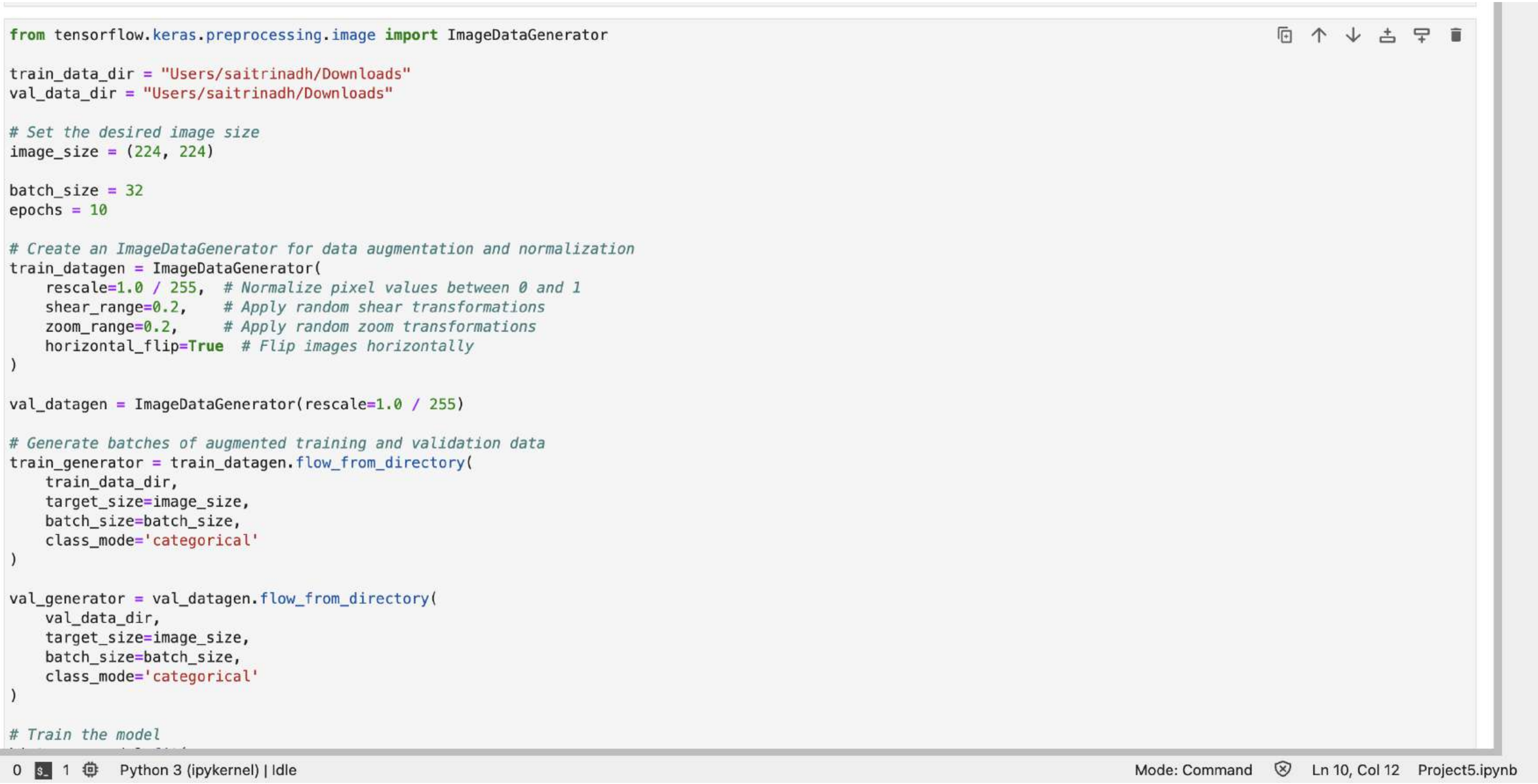
```
from PIL import Image
import os

dataset_dir = "Users/saitrinadh/Downloads"

output_dir = "Users/saitrinadh/Downloads"

for merged_file in os.listdir(dataset_dir):
    if merged_file.endswith(".jpg") or merged_file.endswith(".png"):

        image_path = os.path.join(dataset_dir, merged_file)
        image = Image.open(image_path)

        normalized_image = image.normalize()


        output_path = os.path.join(output_dir, merged_file)
        normalized_image.save(output_path)

        print(f"Normalized image saved: {output_path}")
```

```
from sklearn.model_selection import train_test_split
import pandas as pd

dataset_path = "Users/saitrinadh/Downloads/merged_file.csv"
data = pd.read_csv(dataset_path)

X = data.drop("target_column", axis=1)  # Features
y = data["target_column"]  # Labels

train_ratio = 0.8
val_ratio = 0.1
test_ratio = 0.1

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=test_ratio, random_state=42)

remaining_ratio = val_ratio / (1 - test_ratio)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=remaining_ratio, random_state=42)

# Print the sizes of each set
```

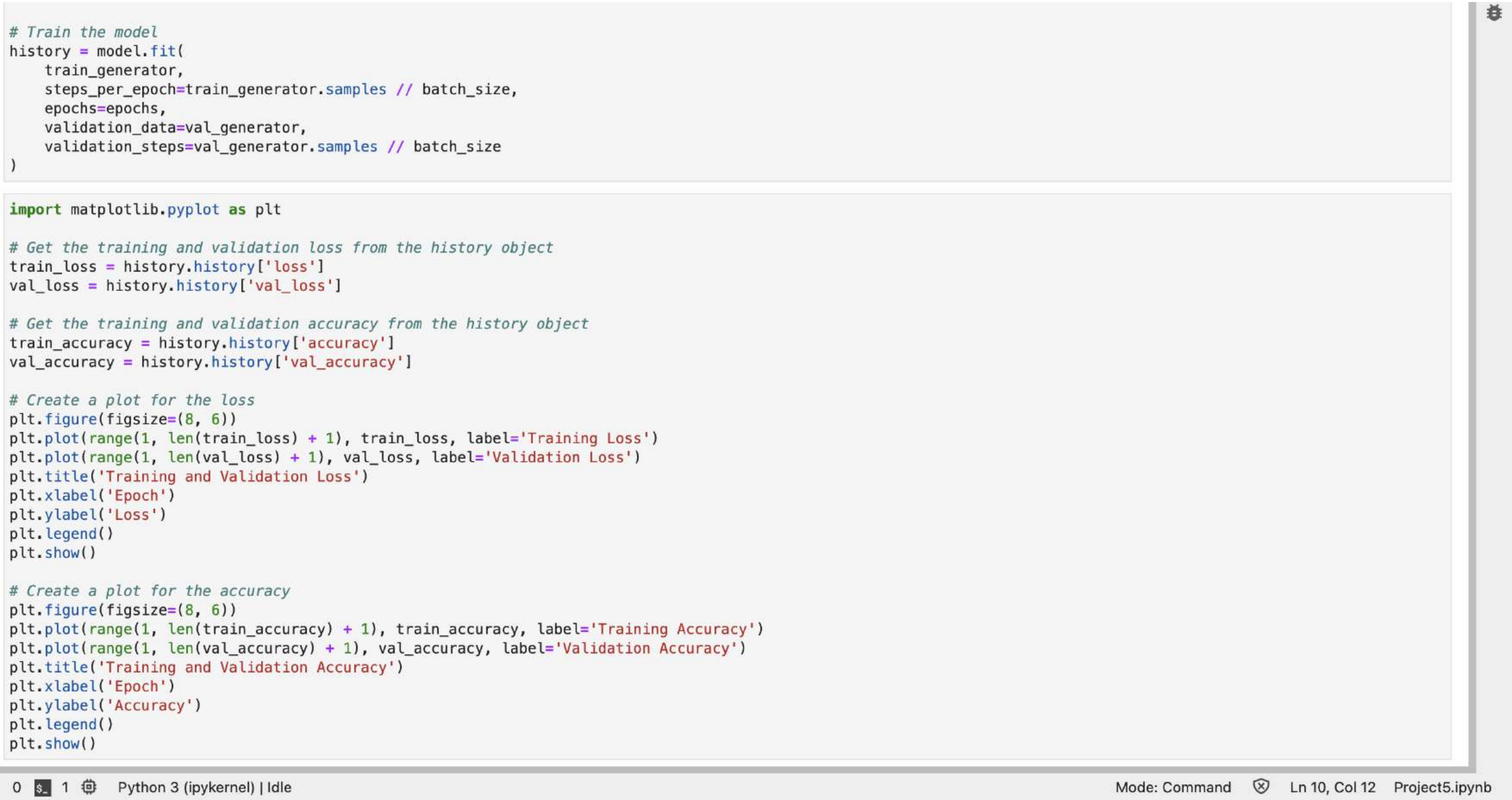


```
from PIL import Image
import os

dataset_dir = "Users/saitrinadh/Downloads"

output_dir = "Users/saitrinadh/Downloads"

target_width = 224
target_height = 224

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for merged_file in os.listdir(dataset_dir):
    if merged_file.endswith(".jpg") or merged_file.endswith(".png"):

        image_path = os.path.join(dataset_dir, merged_file)
        image = Image.open(image_path)


        resized_image = image.resize((target_width, target_height))


        output_path = os.path.join(output_dir, merged_file)
        resized_image.save(output_path)

        print(f"Resized image saved: {output_path}")
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

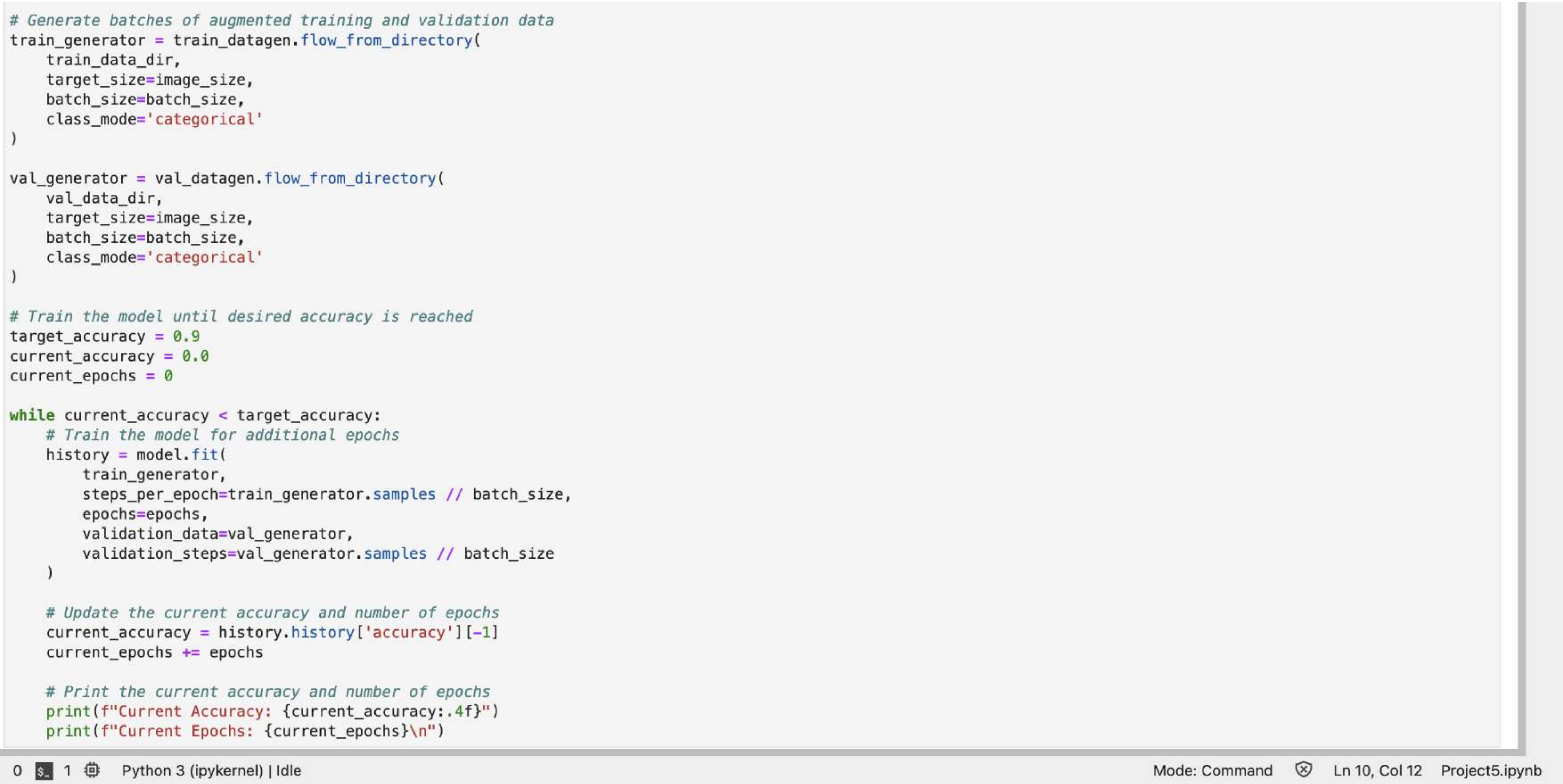


```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_data_dir = "Users/saitrinadh/Downloads"
val_data_dir = "Users/saitrinadh/Downloads"

# Set the desired image size
image_size = (224, 224)

batch_size = 32
epochs = 10

# Create an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,  # Normalize pixel values between 0 and 1
    shear_range=0.2,    # Apply random shear transformations
    zoom_range=0.2,     # Apply random zoom transformations
    horizontal_flip=True  # Flip images horizontally
)

val_datagen = ImageDataGenerator(rescale=1.0 / 255)

# Generate batches of augmented training and validation data
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

# Train the model
```



```
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size
)
```

```
import matplotlib.pyplot as plt

# Get the training and validation loss from the history object
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the training and validation accuracy from the history object
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a plot for the loss
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Create a plot for the accuracy
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(train_accuracy) + 1), train_accuracy, label='Training Accuracy')
plt.plot(range(1, len(val_accuracy) + 1), val_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
# Generate batches of augmented training and validation data
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

# Train the model until desired accuracy is reached
target_accuracy = 0.9
current_accuracy = 0.0
current_epochs = 0

while current_accuracy < target_accuracy:
    # Train the model for additional epochs
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // batch_size,
        epochs=epochs,
        validation_data=val_generator,
        validation_steps=val_generator.samples // batch_size
    )

    # Update the current accuracy and number of epochs
    current_accuracy = history.history['accuracy'][-1]
    current_epochs += epochs

    # Print the current accuracy and number of epochs
    print(f"Current Accuracy: {current_accuracy:.4f}")
    print(f"Current Epochs: {current_epochs}\n")
```