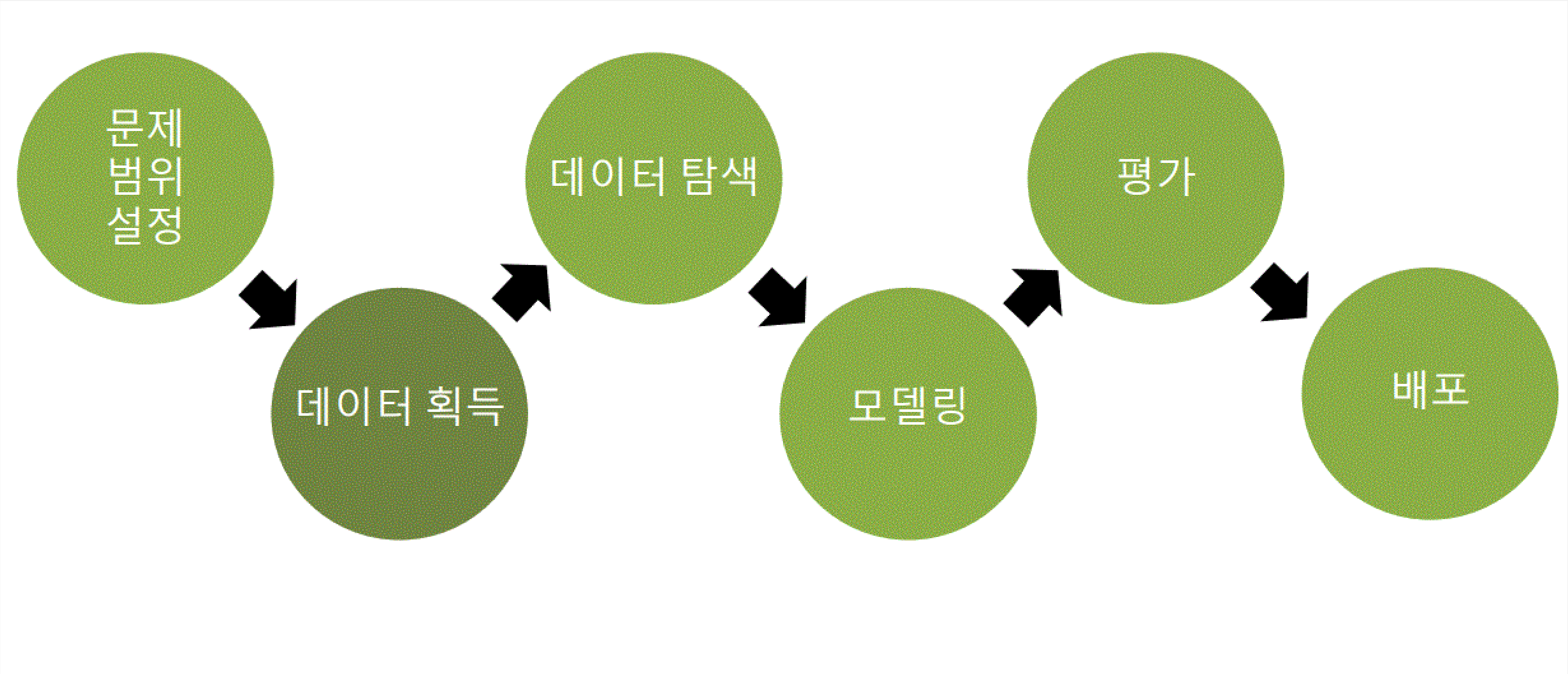
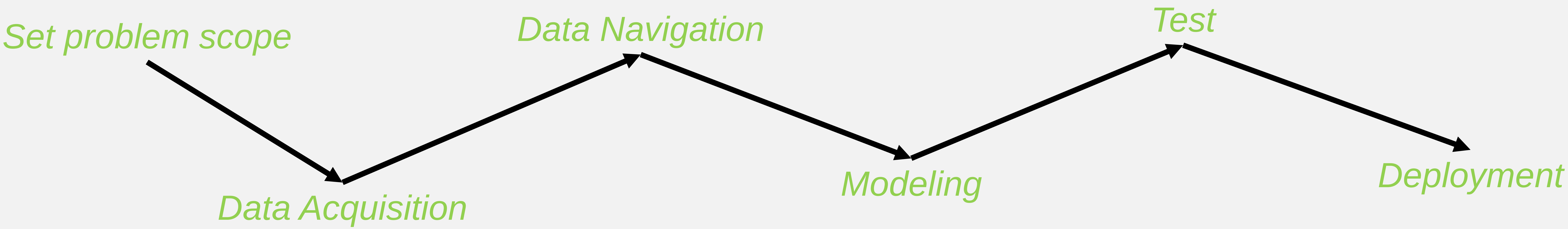
Personalized food warning system
Project Cycle & Code Review

Project Cycle





Set problem scope

Data Acquisition

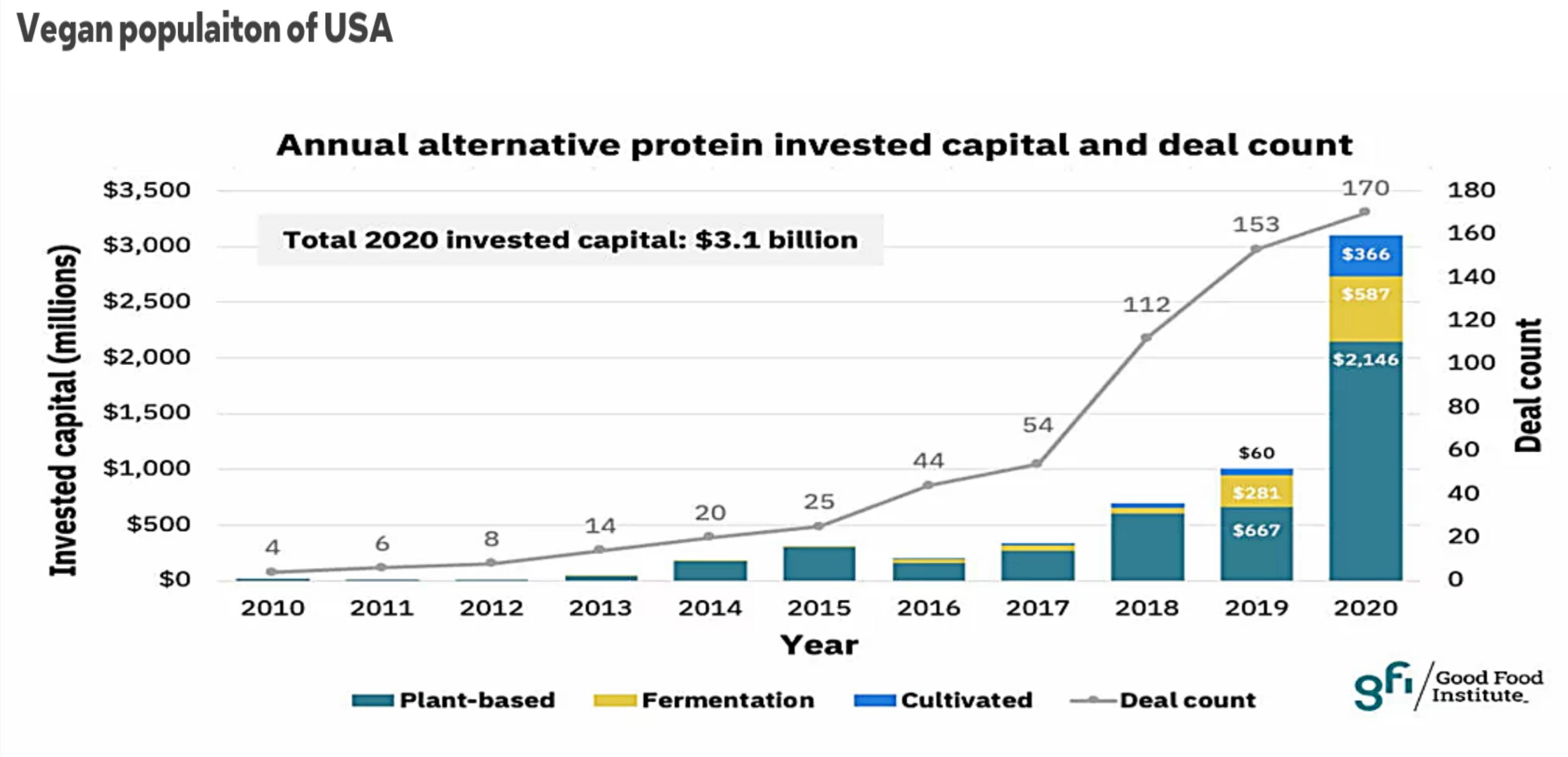
Data Navigation

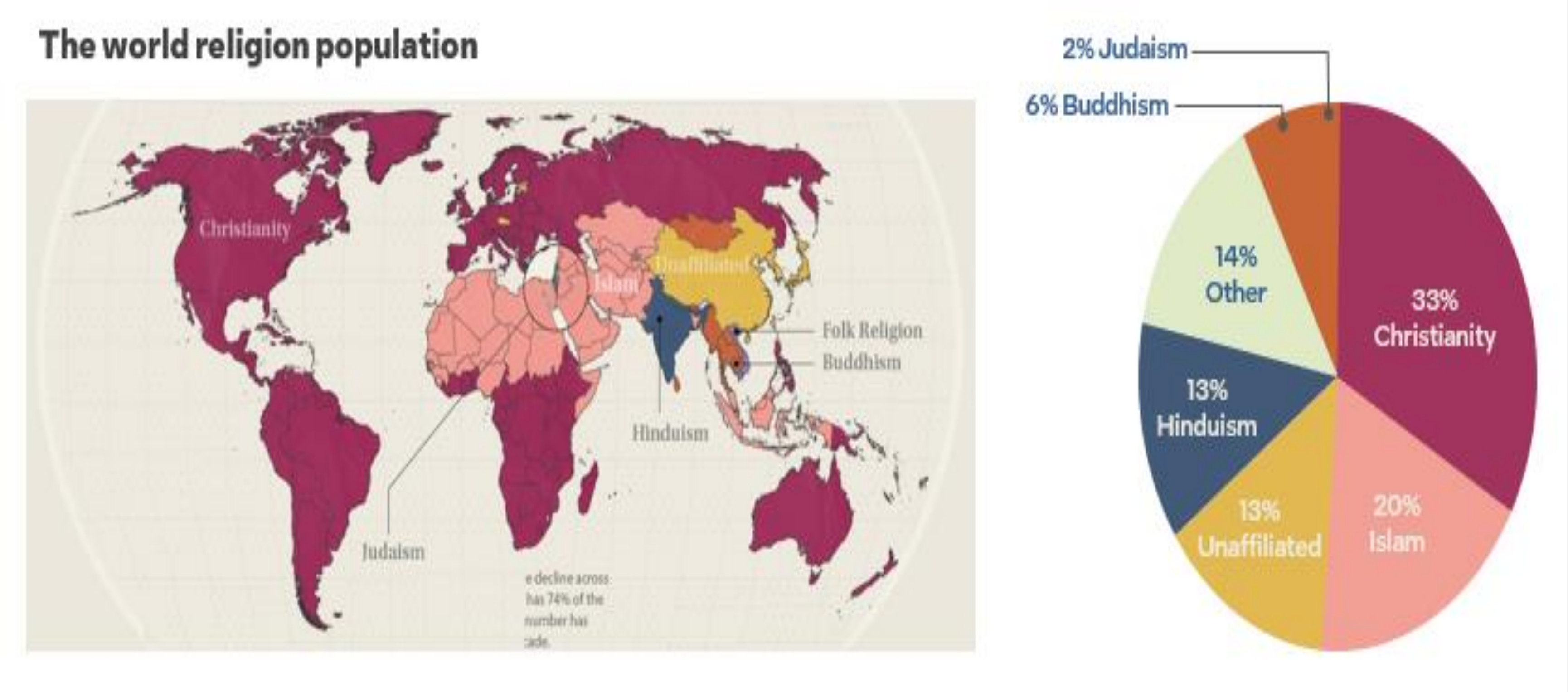
Modeling

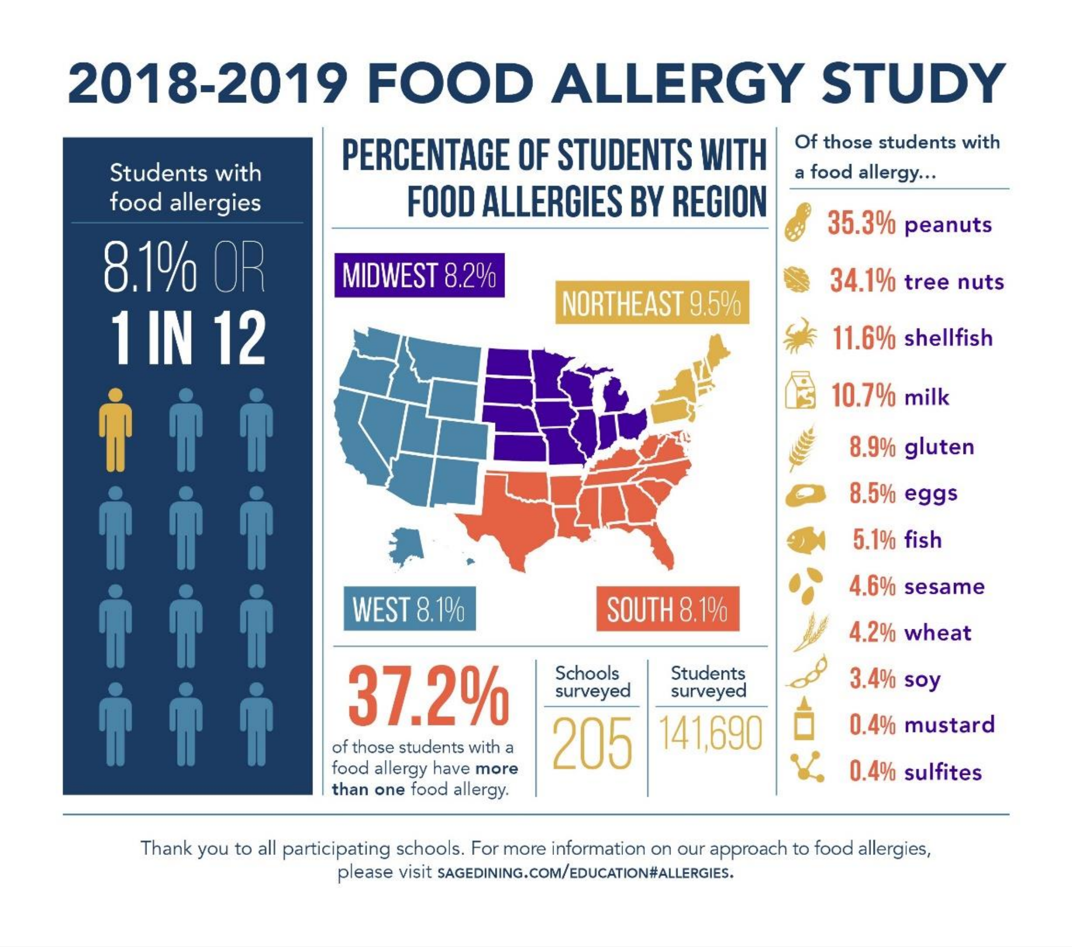
Test

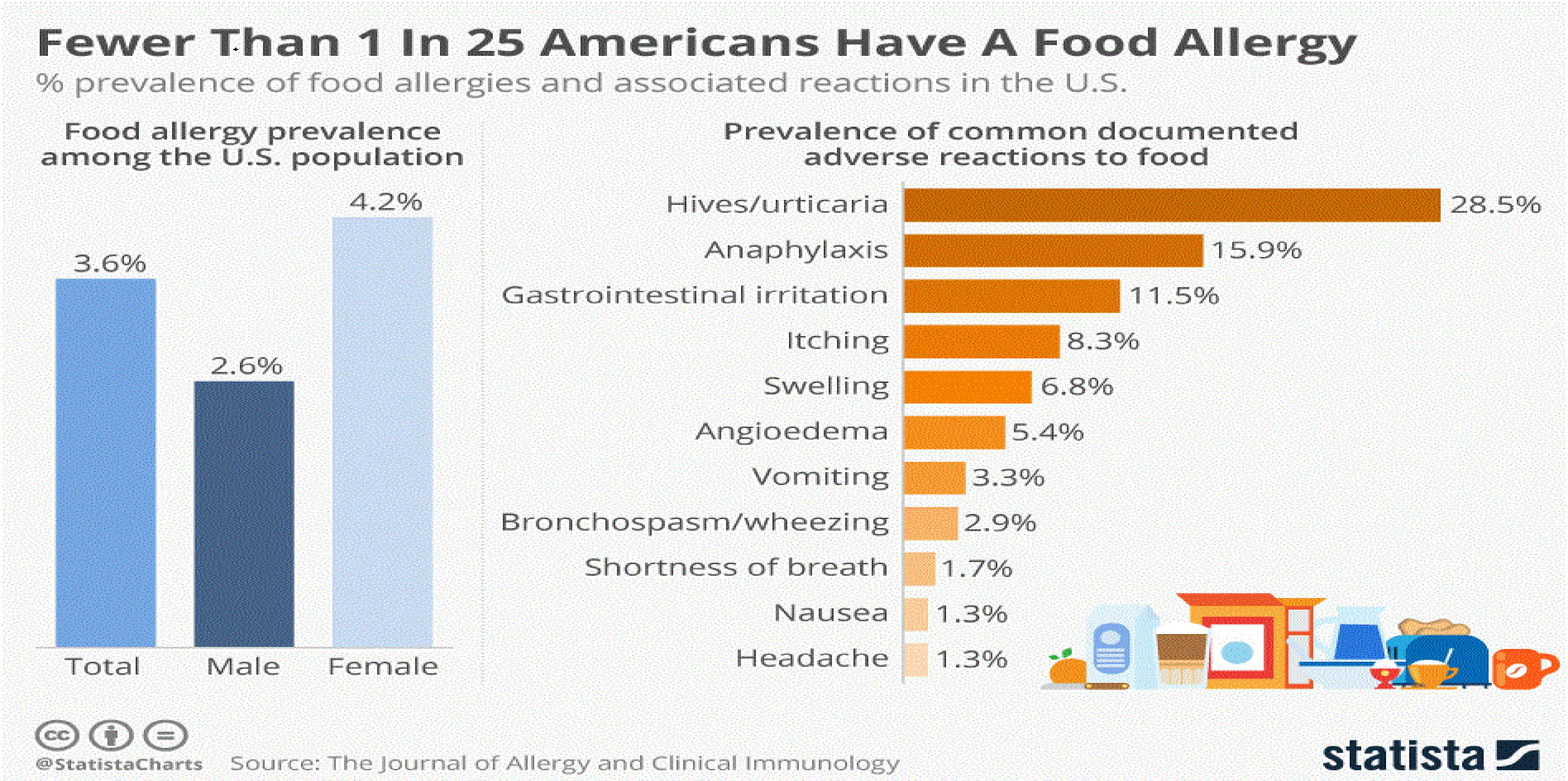
Deployment

1. Set problem scope









1. Set problem scope(The differentiation of technology)

1. It is not to analyze the nutritional content of simple food.

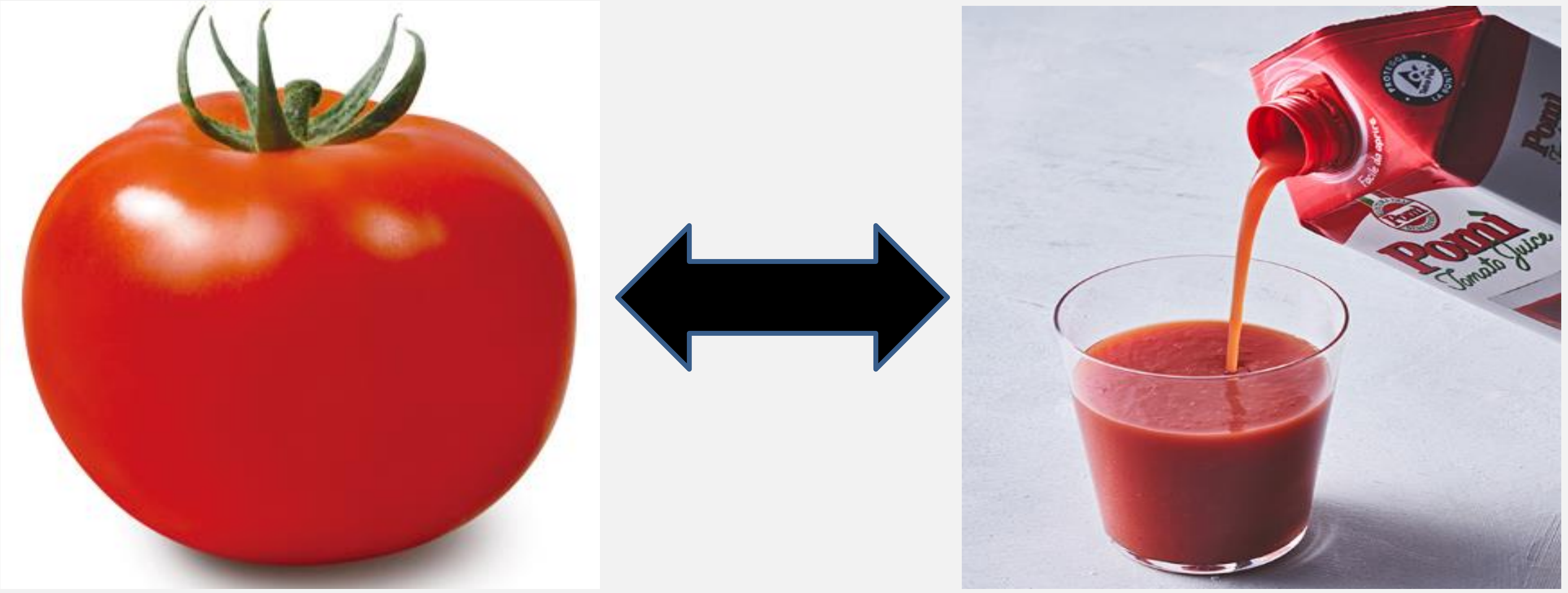
2. Artificial intelligence analyzes the health(Religion, allergies, vegans, etc) information data entered by the user and analyzes the food that the user should avoid.

3. It is a system that gives warning notifications.

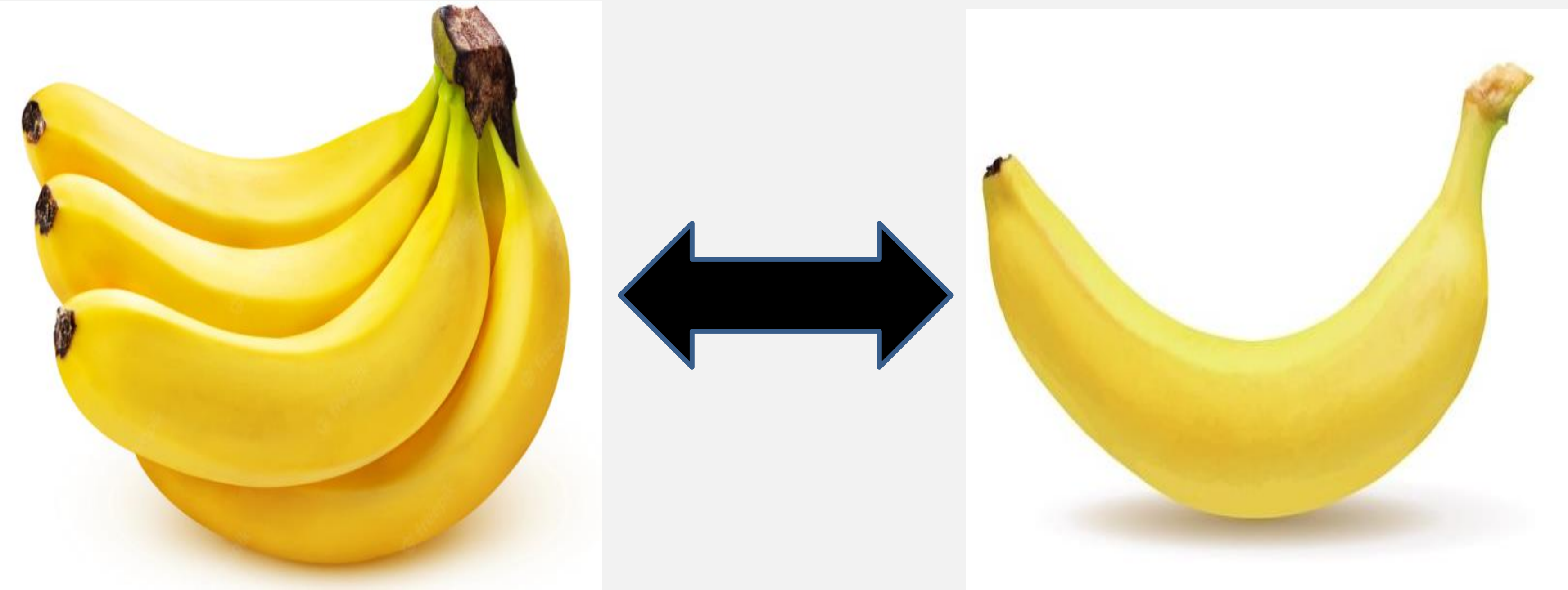
4. It is possible to provide an optimal service that is differentiated for each user.

5. We can respect racial diversity and improve health care.

2. Data Acquisition 3. Data Navigation

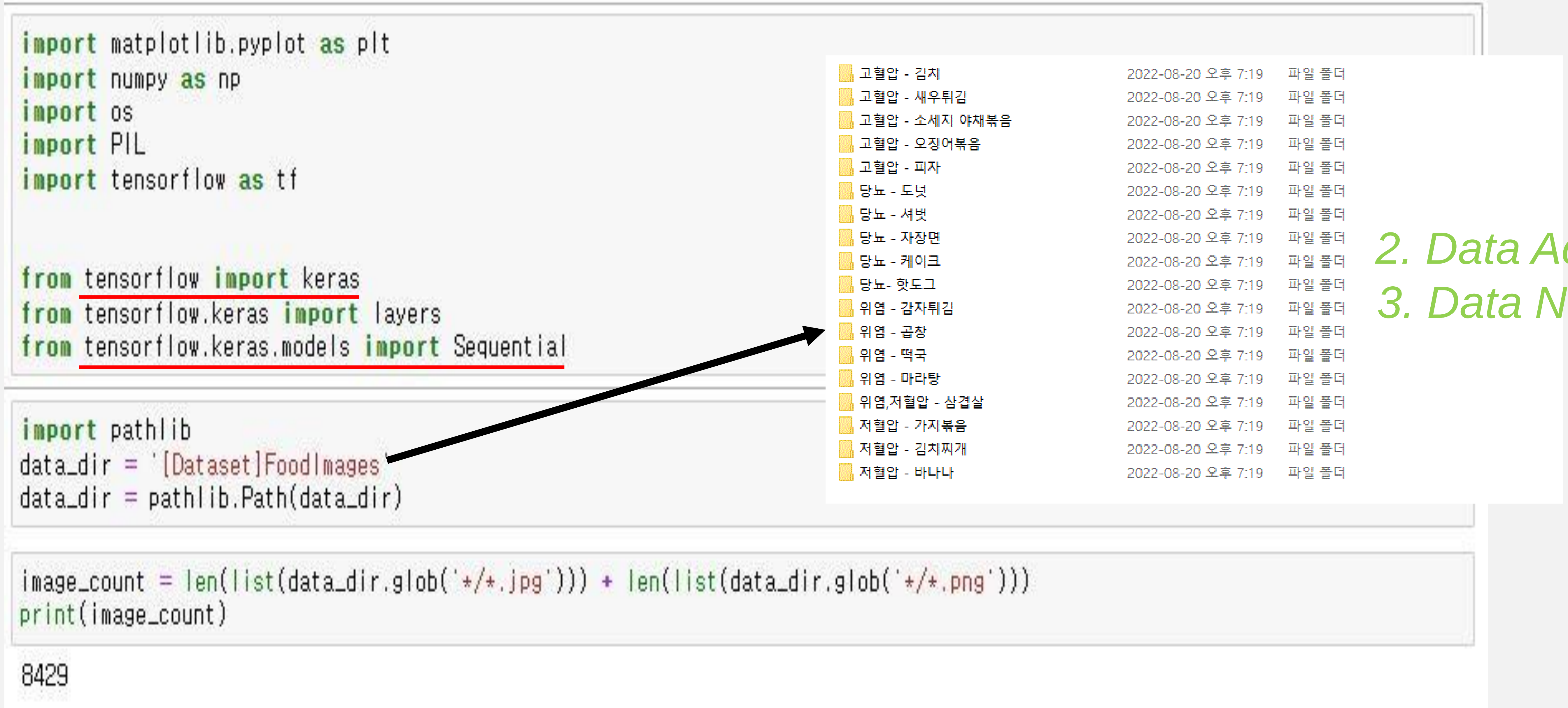


What’s tomato?



What’s banana?

2. Data Acquisition
3. Data Navigation



```python
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf


from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```python
import pathlib
data_dir = '[Dataset]FoodImages
data_dir = pathlib.Path(data_dir)
```

```python
image_count = len(list(data_dir.glob('*/*.jpg'))) + len(list(data_dir.glob('*/*.png')))
print(image_count)
```

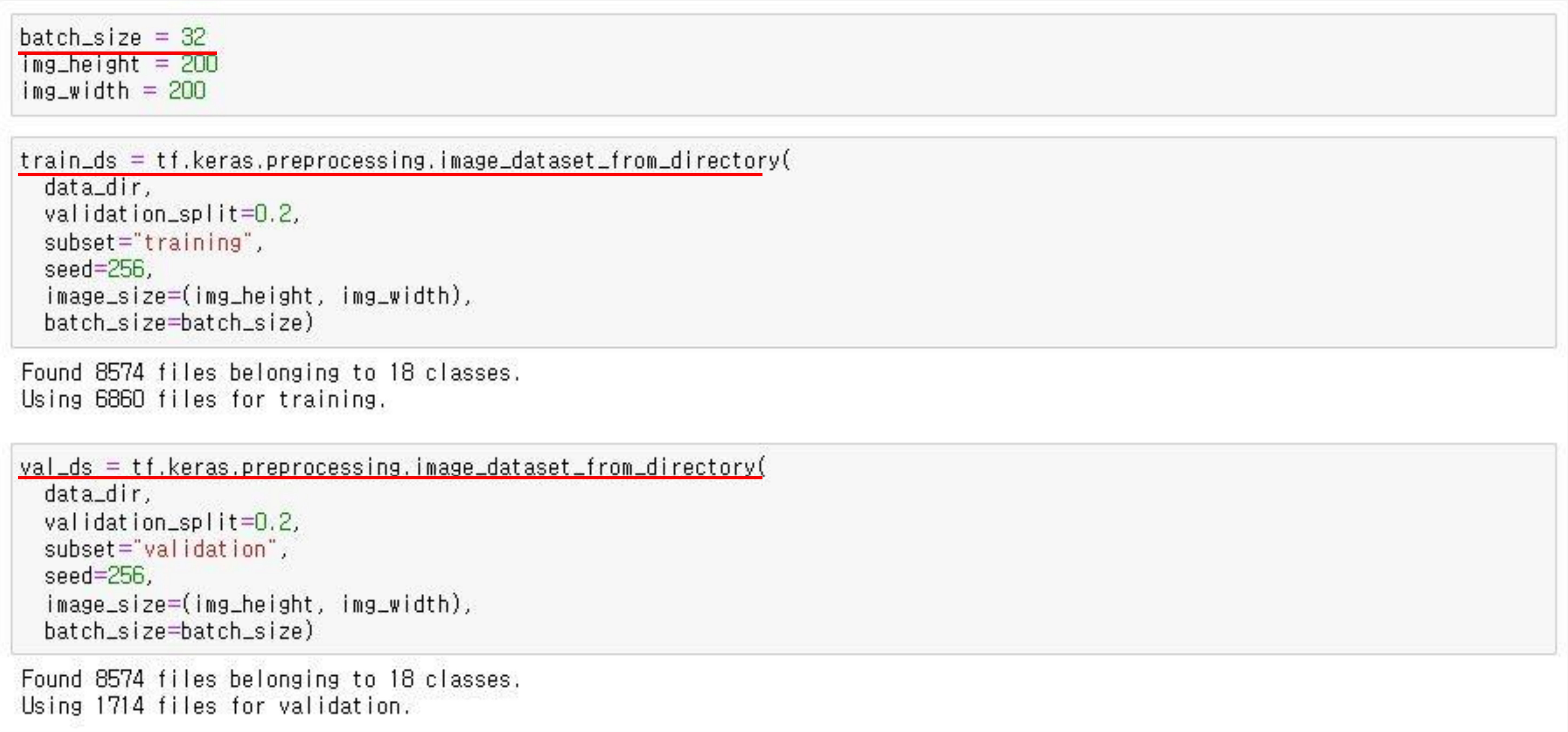
```
8429
```

고혈압 - 김치	2022-08-20 오후 7:19	파일 폴더
고혈압 - 새우튀김	2022-08-20 오후 7:19	파일 폴더
고혈압 - 소세지 야채볶음	2022-08-20 오후 7:19	파일 폴더
고혈압 - 오징어볶음	2022-08-20 오후 7:19	파일 폴더
고혈압 - 피자	2022-08-20 오후 7:19	파일 폴더
당뇨 - 도넛	2022-08-20 오후 7:19	파일 폴더
당뇨 - 셔벗	2022-08-20 오후 7:19	파일 폴더
당뇨 - 자장면	2022-08-20 오후 7:19	파일 폴더
당뇨 - 케이크	2022-08-20 오후 7:19	파일 폴더
당뇨- 핫도그	2022-08-20 오후 7:19	파일 폴더
위염 - 감자튀김	2022-08-20 오후 7:19	파일 폴더
위염 - 곱창	2022-08-20 오후 7:19	파일 폴더
위염 - 떡국	2022-08-20 오후 7:19	파일 폴더
위염 - 마라탕	2022-08-20 오후 7:19	파일 폴더
위염,저혈압 - 삼겹살	2022-08-20 오후 7:19	파일 폴더
저혈압 - 가지볶음	2022-08-20 오후 7:19	파일 폴더
저혈압 - 김치찌개	2022-08-20 오후 7:19	파일 폴더
저혈압 - 바나나	2022-08-20 오후 7:19	파일 폴더

- Tensorflow keras(framework) : Using the Keras of TensorFlow for basic image classification.

- Sequential : Decided to use the sequential model of Keras to learn.

Labeling and purifying collected data



```
batch_size = 32
img_height = 200
img_width = 200
```

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=256,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 8574 files belonging to 18 classes.
Using 6860 files for training.
```

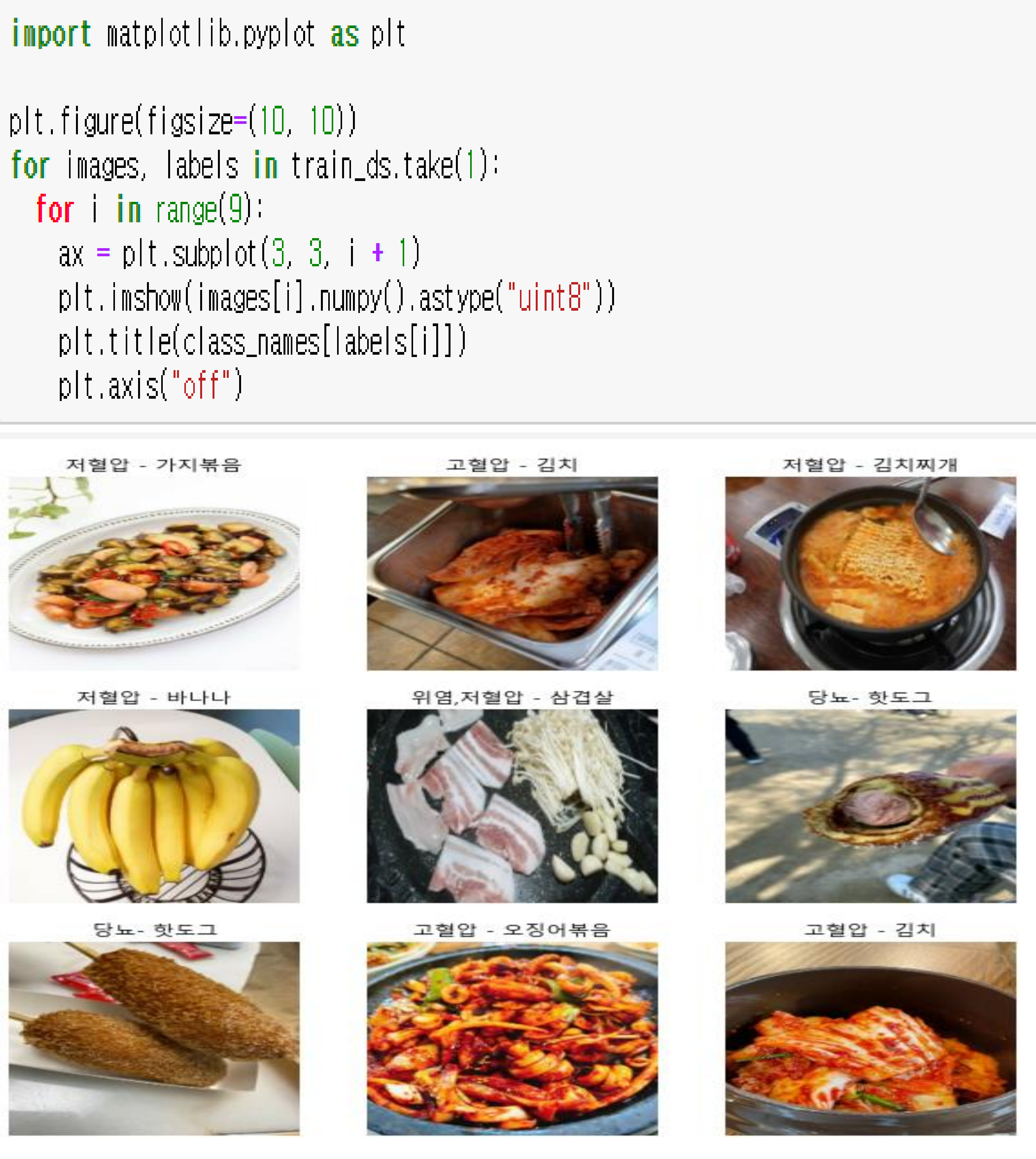
```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=256,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 8574 files belonging to 18 classes.
Using 1714 files for validation.
```

- batch_size : Considering noise and regularization, set a compliant batch size.

- train_ds // val_ds : The process of dividing train data and validation data (image_count)

Recall labeled data using a matplotlib



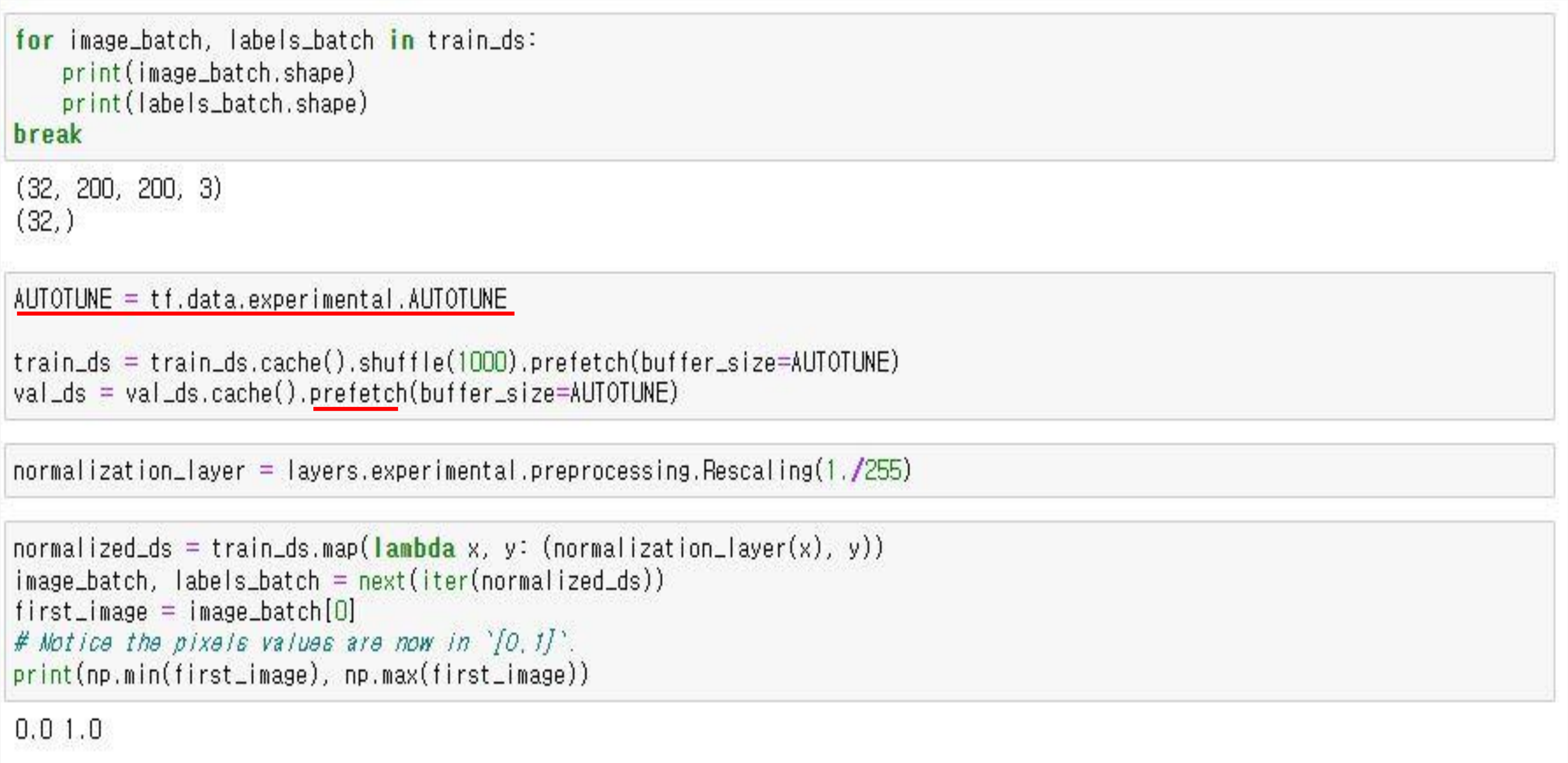
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

- matplotlib : Outputs the data set correctly using the Matlab library.

Explanation : Indicate and label what complications each food causes..

Data preprocessing



```
for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    print(labels_batch.shape)
break
```

```
(32, 200, 200, 3)
(32,)
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

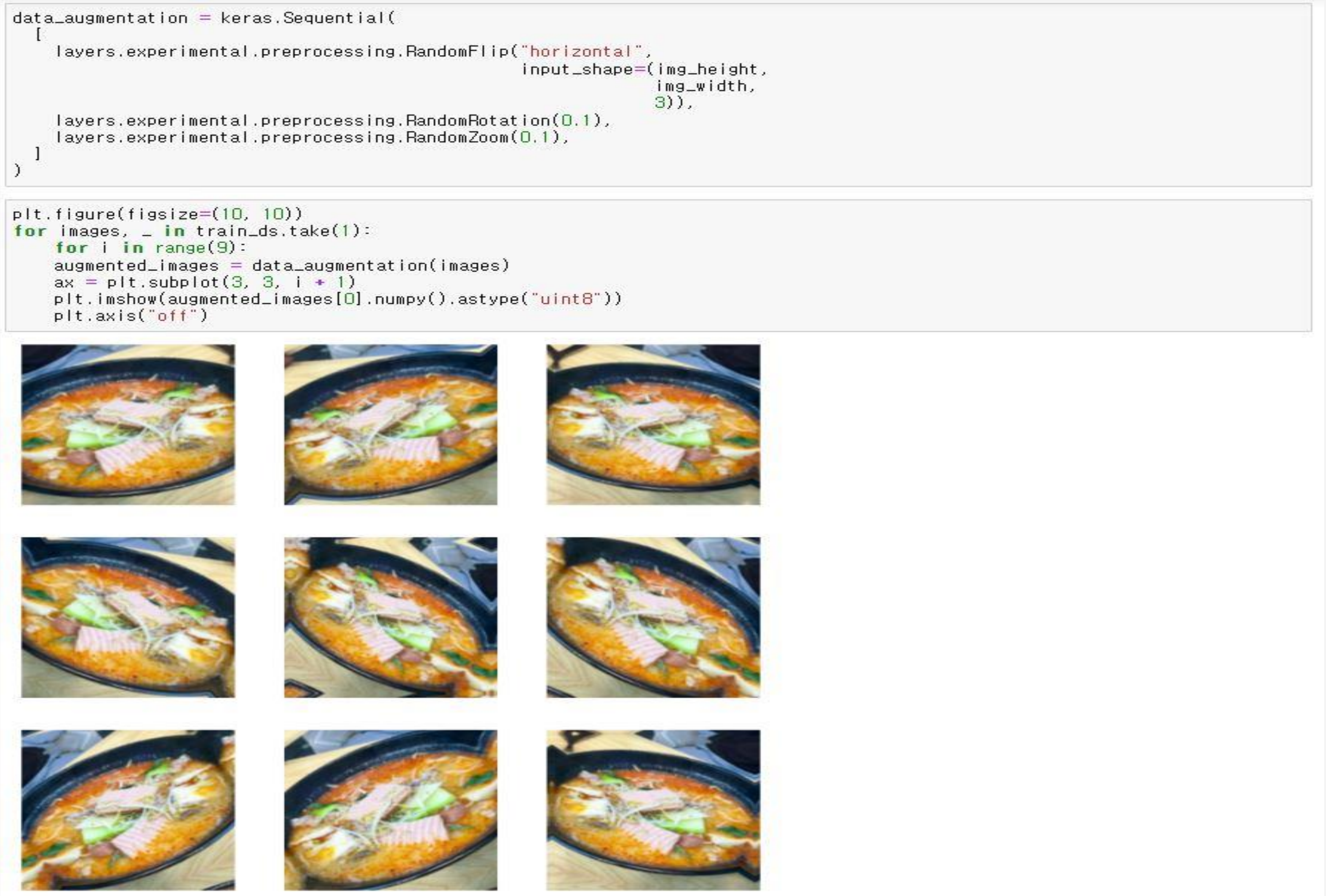
```
0.0 1.0
```

- AUTOTUNE : Using AUTOTUNE to reduce working time by mapping hardware resources in parallel.

- Prefetch : Pipeline used to reduce data processing (task) time on GPU.

Explanation : Set batch size, standardize data, and preprocess data (improve speed)..

Data augmentation techniques



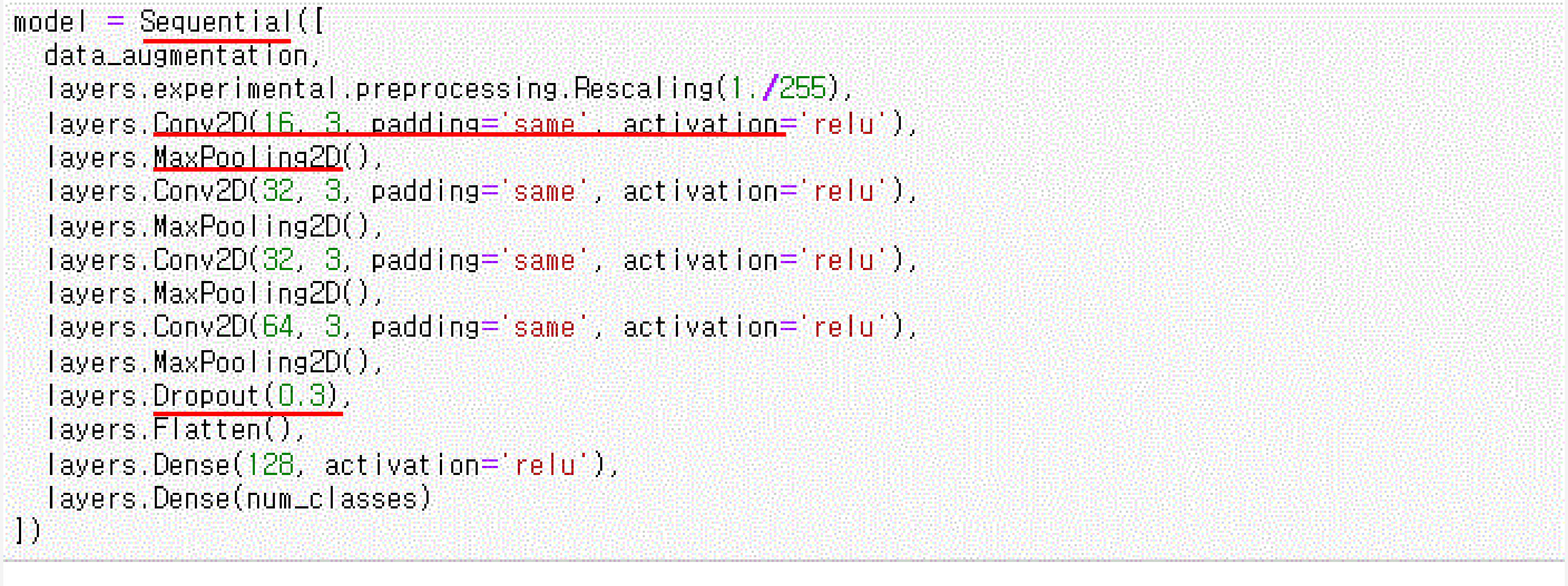
```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

- Explanation

- Determined that the number of data is too small to cause problems with accuracy and loss rate.
- I expanded the learning data and trained it again.
- Extends learning data by adjusting multiple angles of the image

4. Modeling(Model the data.)



```
model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.3),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

- Model Using Sequential Models.

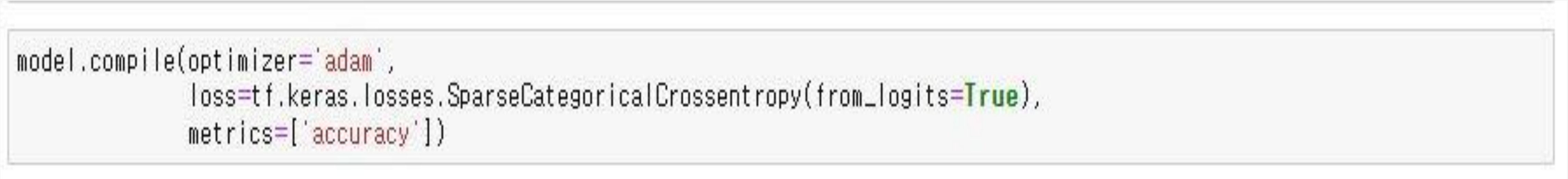
  Reasons for using the Sequential Model: It's a one-step, sequential neural network model, and it's very simple

- Conv2D(activation=‘relu’)
  Set the activation function to relu to prevent gradient varishing problems in the convolution layer

- Dense(activation=‘relu’)
  Set the activation function to relu to reduce the slope problem through backpropagation and produce good performance.

- Explanation
- The sequential model of keras was used as the model. To increase the accuracy of image classification, we created a model alternating the convolutional (Conv2D) layer and Maxpooling2D to reduce the size of the image at each layer, thus reducing the computation and preventing overfitting.

Compile the modeled data



```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

- Why Optimizer Adam?

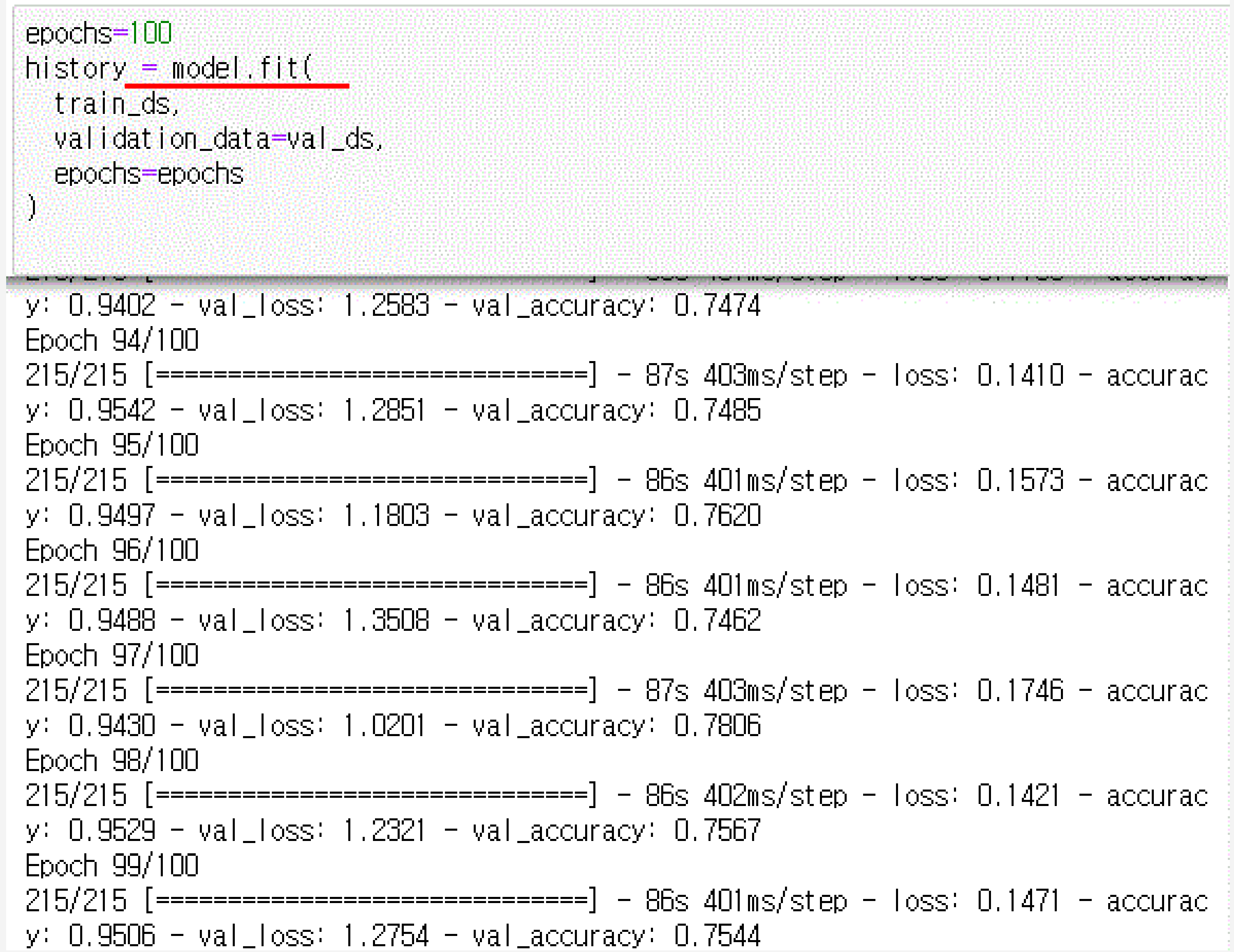
  Adam is considered appropriate because it requires a lot of data to perform efficient operations and reduce memory requirements through a simple implementation

- Explanation

  The model was previously constructed, and before learning the model, the complex function was used to make various settings necessary for the learning process.

  1. The optimizer uses adam to set the optimization method in the learning process in the compilation function.
  2. Set for calculating loss rate in loss facility.
  3. It was set up to monitor learning with metircs.

model learning

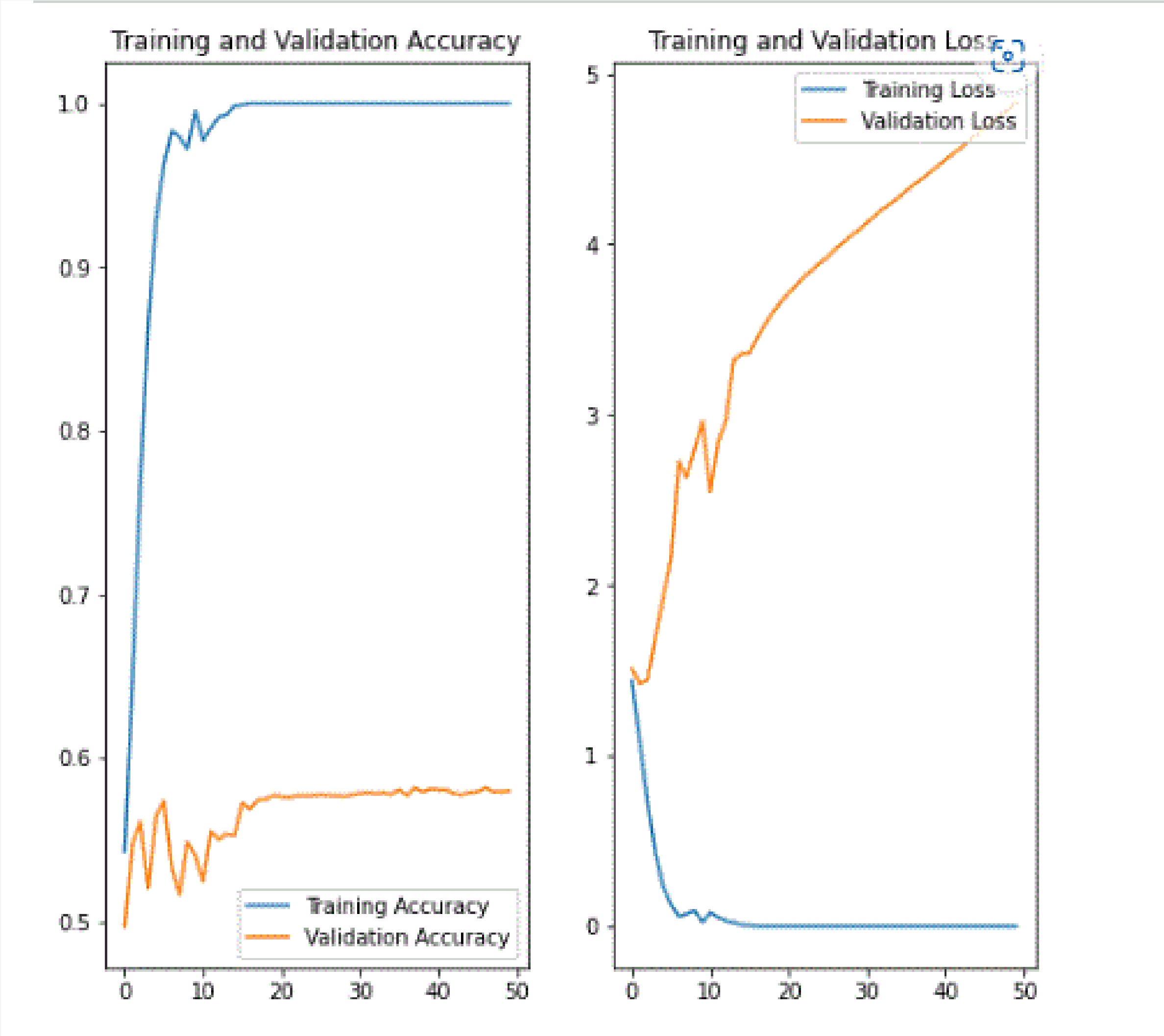


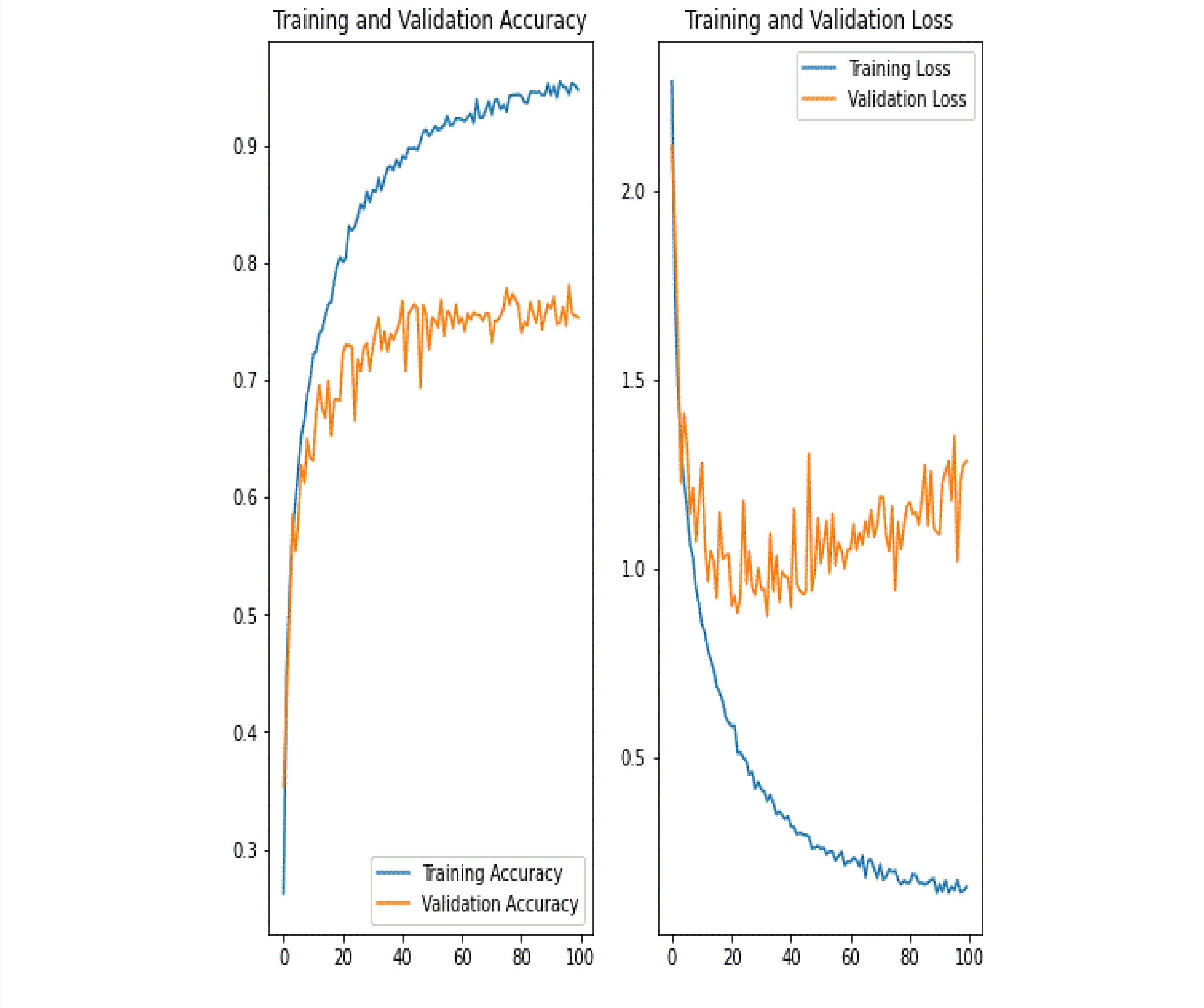
```
epochs=100
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
y: 0.9402 - val_loss: 1.2583 - val_accuracy: 0.7474
Epoch 94/100
215/215 [==============================] - 87s 403ms/step - loss: 0.1410 - accurac
y: 0.9542 - val_loss: 1.2851 - val_accuracy: 0.7485
Epoch 95/100
215/215 [==============================] - 86s 401ms/step - loss: 0.1573 - accurac
y: 0.9497 - val_loss: 1.1803 - val_accuracy: 0.7620
Epoch 96/100
215/215 [==============================] - 86s 401ms/step - loss: 0.1481 - accurac
y: 0.9488 - val_loss: 1.3508 - val_accuracy: 0.7462
Epoch 97/100
215/215 [==============================] - 87s 403ms/step - loss: 0.1746 - accurac
y: 0.9430 - val_loss: 1.0201 - val_accuracy: 0.7806
Epoch 98/100
215/215 [==============================] - 86s 402ms/step - loss: 0.1421 - accurac
y: 0.9529 - val_loss: 1.2321 - val_accuracy: 0.7567
Epoch 99/100
215/215 [==============================] - 86s 401ms/step - loss: 0.1471 - accurac
y: 0.9506 - val_loss: 1.2754 - val_accuracy: 0.7544
```

- Explanation

Use the Fit function to learn the model.
1. The first factor is the training data set (80%)
2. The second factor is val_ds (20%) i.e., data labeled in the train process.
3. The third factor designated the number of studies as 100.





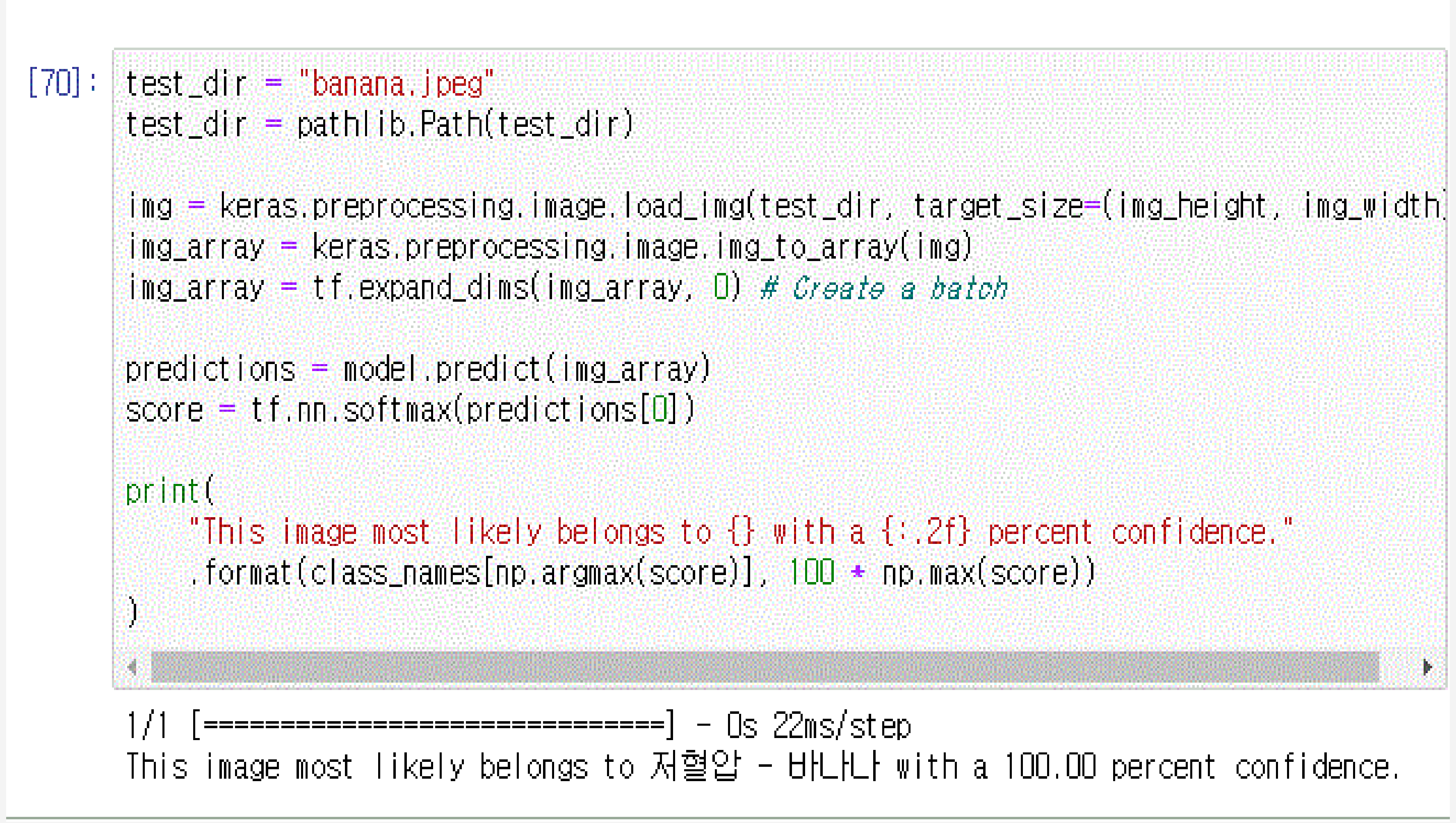
- Explanation

Results of the first training graph
Problem: somewhat low accuracy, somewhat high loss rate

- Explanation

Accuracy is getting higher than before.
The loss rate is also much lower than before.

4. Test



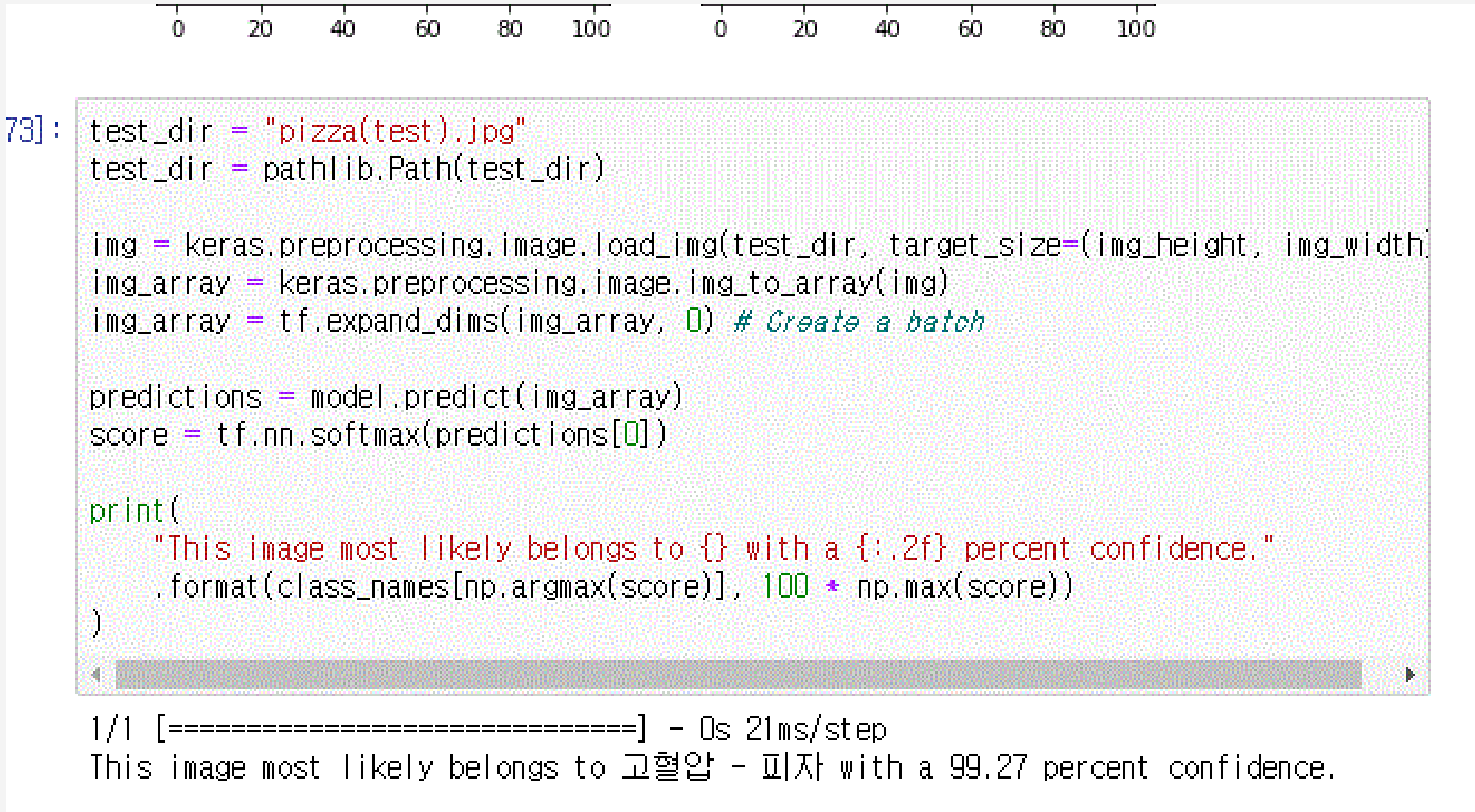
```
[70]: test_dir = "banana.jpeg"
      test_dir = pathlib.Path(test_dir)

      img = keras.preprocessing.image.load_img(test_dir, target_size=(img_height, img_width
      img_array = keras.preprocessing.image.img_to_array(img)
      img_array = tf.expand_dims(img_array, 0) # Create a batch

      predictions = model.predict(img_array)
      score = tf.nn.softmax(predictions[0])

      print(
          "This image most likely belongs to {} with a {:.2f} percent confidence."
          .format(class_names[np.argmax(score)], 100 * np.max(score))
      )
```

```
1/1 [==============================] - 0s 22ms/step
This image most likely belongs to 저혈압 - 바나나 with a 100.00 percent confidence.
```



```
73]: test_dir = "pizza(test).jpg"
     test_dir = pathlib.Path(test_dir)

     img = keras.preprocessing.image.load_img(test_dir, target_size=(img_height, img_width
     img_array = keras.preprocessing.image.img_to_array(img)
     img_array = tf.expand_dims(img_array, 0) # Create a batch

     predictions = model.predict(img_array)
     score = tf.nn.softmax(predictions[0])

     print(
         "This image most likely belongs to {} with a {:.2f} percent confidence."
         .format(class_names[np.argmax(score)], 100 * np.max(score))
     )
```

```
1/1 [==============================] - 0s 21ms/step
This image most likely belongs to 고혈압 - 피자 with a 99.27 percent confidence.
```

- Explanation

  - When I entered a picture of banana that was not included in the training data into the model, it was 100% accurate because it was banana.

* Currently, a prototype has been created to accurately distinguish food *

Personalized food warning system

THANKS!