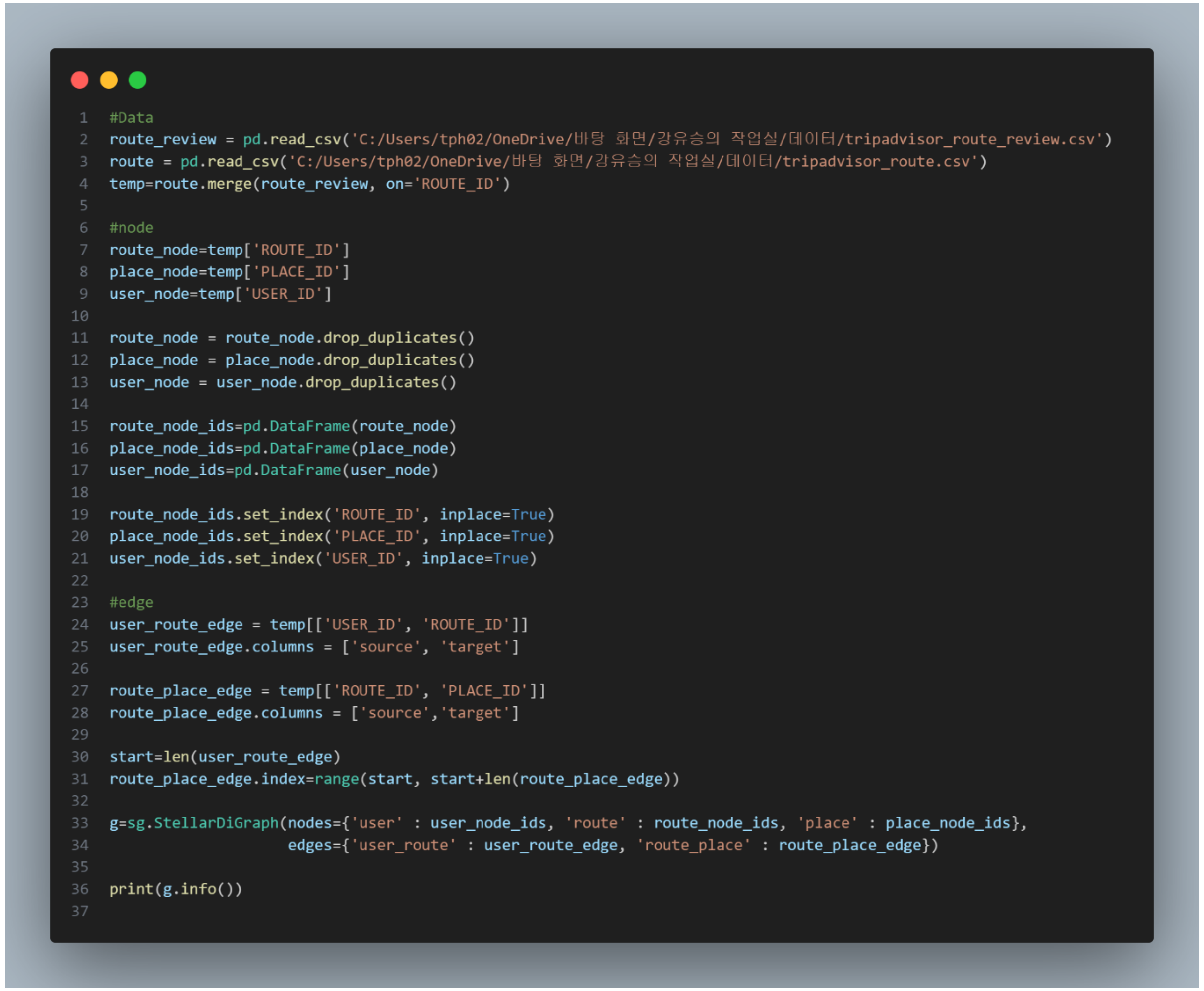
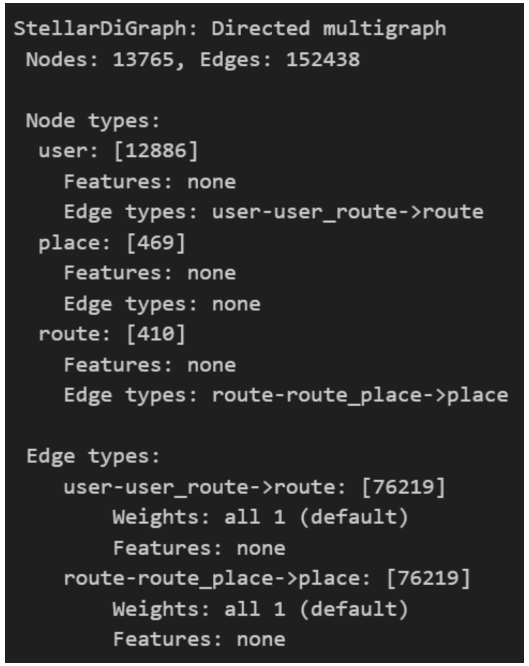
코드 설명

데이터 로드 및 Network 생성



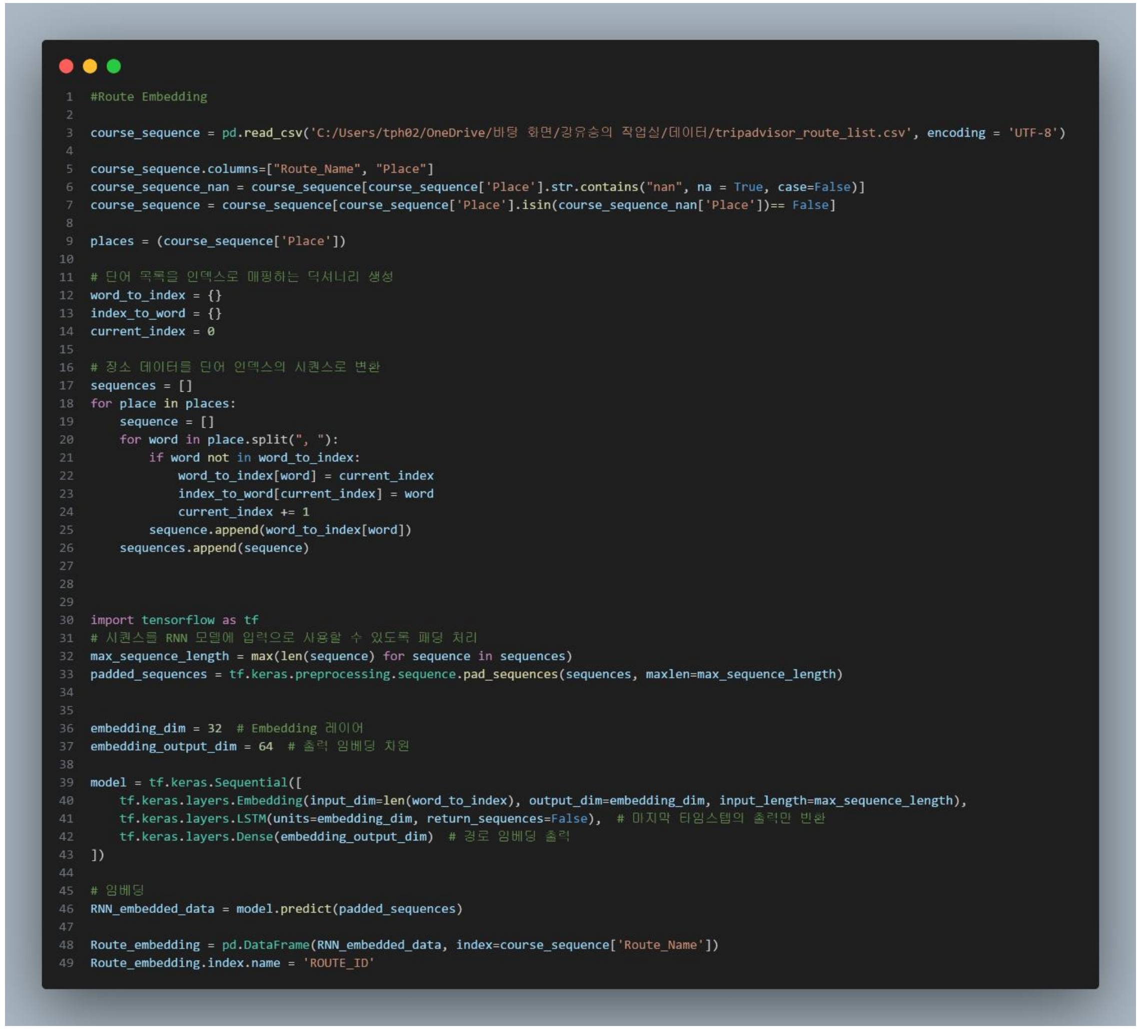
```python
#Data
route_review = pd.read_csv('C:/Users/tph02/OneDrive/바탕 화면/강유승의 작업실/데이터/tripadvisor_route_review.csv')
route = pd.read_csv('C:/Users/tph02/OneDrive/바탕 화면/강유승의 작업실/데이터/tripadvisor_route.csv')
temp=route.merge(route_review, on='ROUTE_ID')

#node
route_node=temp['ROUTE_ID']
place_node=temp['PLACE_ID']
user_node=temp['USER_ID']

route_node = route_node.drop_duplicates()
place_node = place_node.drop_duplicates()
user_node = user_node.drop_duplicates()

route_node_ids=pd.DataFrame(route_node)
place_node_ids=pd.DataFrame(place_node)
user_node_ids=pd.DataFrame(user_node)

route_node_ids.set_index('ROUTE_ID', inplace=True)
place_node_ids.set_index('PLACE_ID', inplace=True)
user_node_ids.set_index('USER_ID', inplace=True)

#edge
user_route_edge = temp[['USER_ID', 'ROUTE_ID']]
user_route_edge.columns = ['source', 'target']

route_place_edge = temp[['ROUTE_ID', 'PLACE_ID']]
route_place_edge.columns = ['source','target']

start=len(user_route_edge)
route_place_edge.index=range(start, start+len(route_place_edge))

g=sg.StellarDiGraph(nodes={'user' : user_node_ids, 'route' : route_node_ids, 'place' : place_node_ids},
                    edges={'user_route' : user_route_edge, 'route_place' : route_place_edge})

print(g.info())

```



```
StellarDiGraph: Directed multigraph
 Nodes: 13765, Edges: 152438

 Node types:
  user: [12886]
    Features: none
    Edge types: user-user_route->route
  place: [469]
    Features: none
    Edge types: none
  route: [410]
    Features: none
    Edge types: route-route_place->place

 Edge types:
    user-user_route->route: [76219]
        Weights: all 1 (default)
        Features: none
    route-route_place->place: [76219]
        Weights: all 1 (default)
        Features: none
```

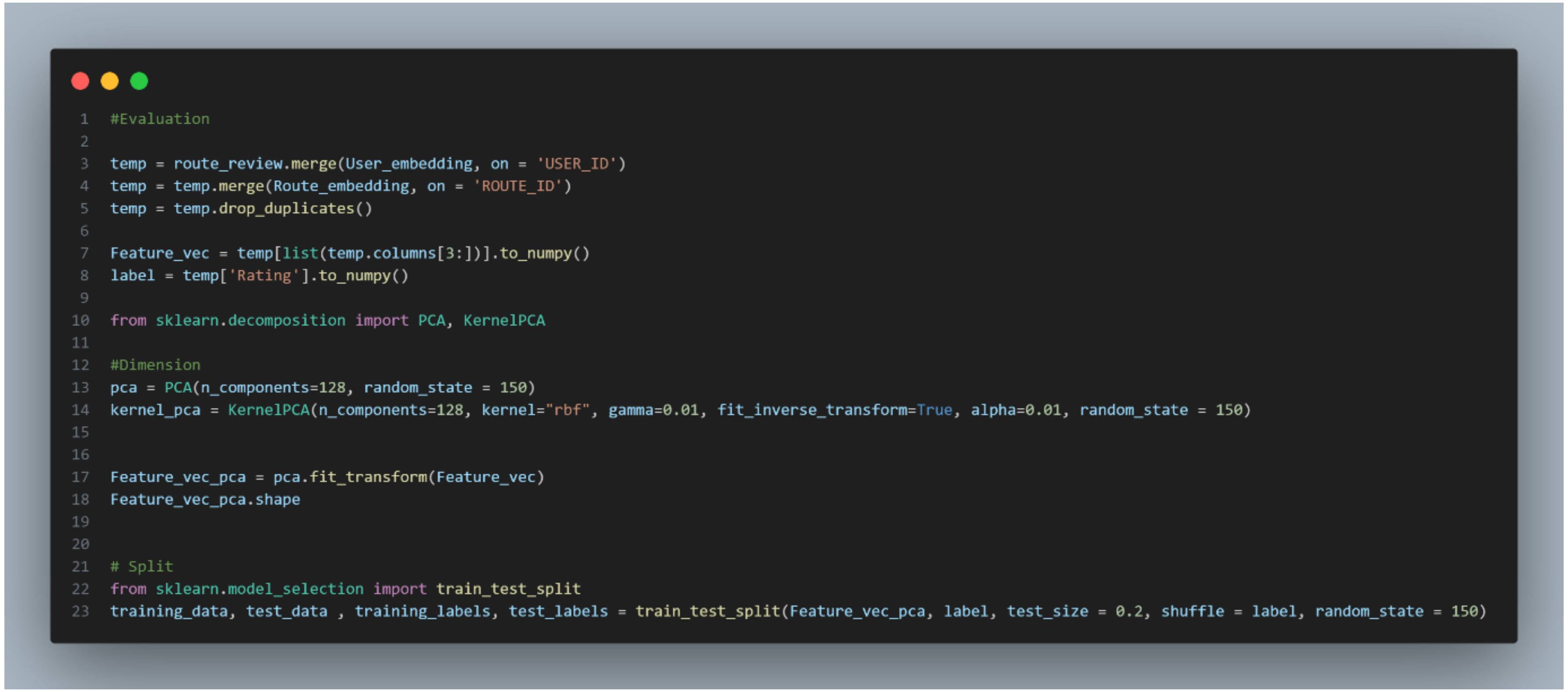
네트워크 생성 결과

시퀀스 처리된 경로 임베딩



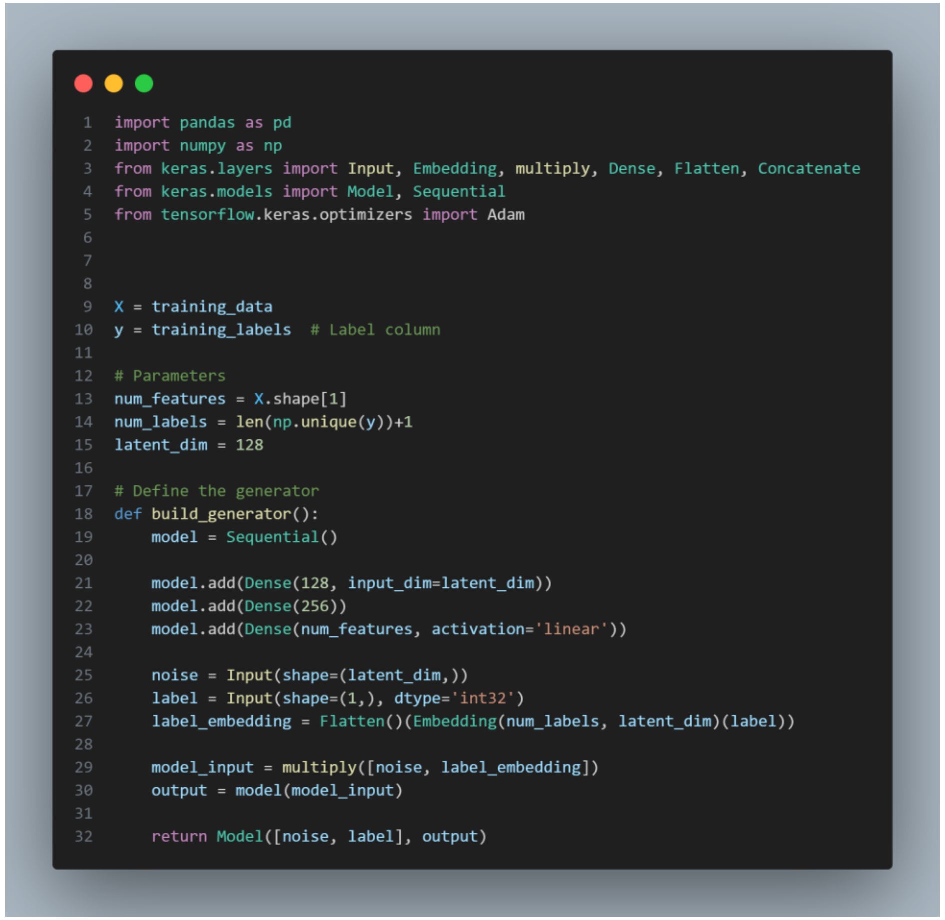
```python
#Route Embedding

course_sequence = pd.read_csv('C:/Users/tph02/OneDrive/바탕 화면/강유승의 작업실/데이터/tripadvisor_route_list.csv', encoding = 'UTF-8')

course_sequence.columns=["Route_Name", "Place"]
course_sequence_nan = course_sequence[course_sequence['Place'].str.contains("nan", na = True, case=False)]
course_sequence = course_sequence[course_sequence['Place'].isin(course_sequence_nan['Place'])== False]

places = (course_sequence['Place'])

# 단어 목록을 인덱스로 매핑하는 딕셔너리 생성
word_to_index = {}
index_to_word = {}
current_index = 0

# 장소 데이터를 단어 인덱스의 시퀀스로 변환
sequences = []
for place in places:
    sequence = []
    for word in place.split(", "):
        if word not in word_to_index:
            word_to_index[word] = current_index
            index_to_word[current_index] = word
            current_index += 1
        sequence.append(word_to_index[word])
    sequences.append(sequence)



import tensorflow as tf
# 시퀀스를 RNN 모델에 입력으로 사용할 수 있도록 패딩 처리
max_sequence_length = max(len(sequence) for sequence in sequences)
padded_sequences = tf.keras.preprocessing.sequence.pad_sequences(sequences, maxlen=max_sequence_length)


embedding_dim = 32  # Embedding 레이어
embedding_output_dim = 64  # 출력 임베딩 차원

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=len(word_to_index), output_dim=embedding_dim, input_length=max_sequence_length),
    tf.keras.layers.LSTM(units=embedding_dim, return_sequences=False),  # 마지막 타임스텝의 출력만 반환
    tf.keras.layers.Dense(embedding_output_dim)  # 경로 임베딩 출력
])

# 임베딩
RNN_embedded_data = model.predict(padded_sequences)

Route_embedding = pd.DataFrame(RNN_embedded_data, index=course_sequence['Route_Name'])
Route_embedding.index.name = 'ROUTE_ID'
```

Latent Vector 모델링



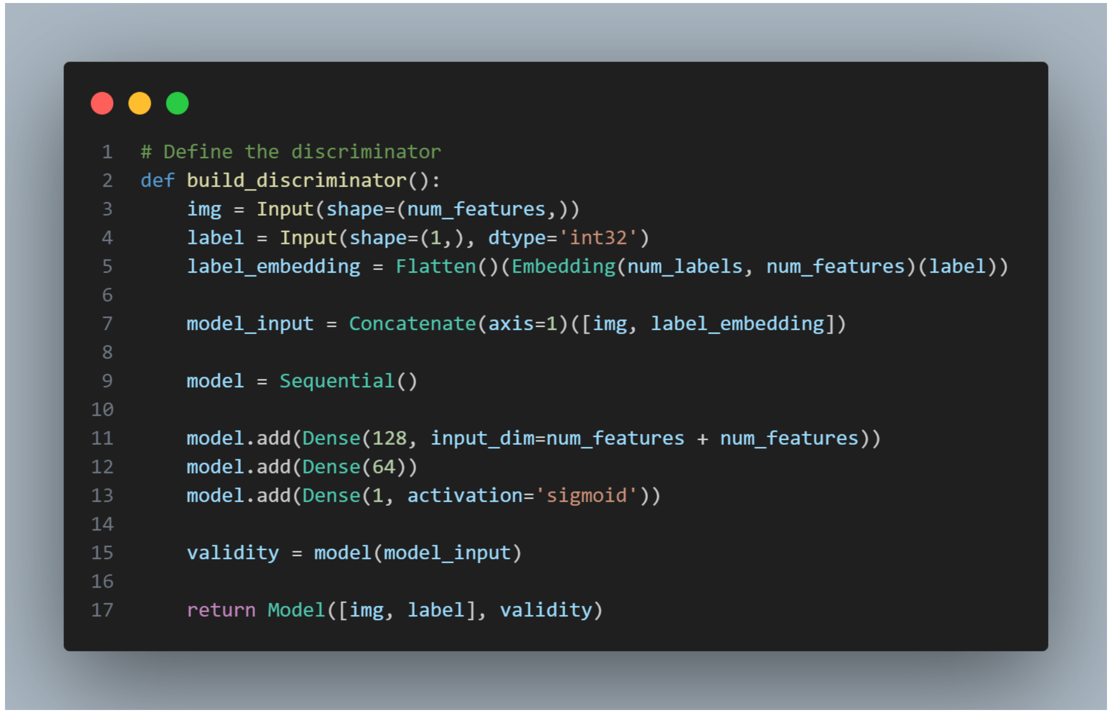
```python
#Evaluation

temp = route_review.merge(User_embedding, on = 'USER_ID')
temp = temp.merge(Route_embedding, on = 'ROUTE_ID')
temp = temp.drop_duplicates()

Feature_vec = temp[list(temp.columns[3:])].to_numpy()
label = temp['Rating'].to_numpy()

from sklearn.decomposition import PCA, KernelPCA

#Dimension
pca = PCA(n_components=128, random_state = 150)
kernel_pca = KernelPCA(n_components=128, kernel="rbf", gamma=0.01, fit_inverse_transform=True, alpha=0.01, random_state = 150)


Feature_vec_pca = pca.fit_transform(Feature_vec)
Feature_vec_pca.shape


# Split
from sklearn.model_selection import train_test_split
training_data, test_data , training_labels, test_labels = train_test_split(Feature_vec_pca, label, test_size = 0.2, shuffle = label, random_state = 150)
```

CGAN



```
import pandas as pd
import numpy as np
from keras.layers import Input, Embedding, multiply, Dense, Flatten, Concatenate
from keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam



X = training_data
y = training_labels  # Label column

# Parameters
num_features = X.shape[1]
num_labels = len(np.unique(y))+1
latent_dim = 128

# Define the generator
def build_generator():
    model = Sequential()

    model.add(Dense(128, input_dim=latent_dim))
    model.add(Dense(256))
    model.add(Dense(num_features, activation='linear'))

    noise = Input(shape=(latent_dim,))
    label = Input(shape=(1,), dtype='int32')
    label_embedding = Flatten()(Embedding(num_labels, latent_dim)(label))

    model_input = multiply([noise, label_embedding])
    output = model(model_input)

    return Model([noise, label], output)
```

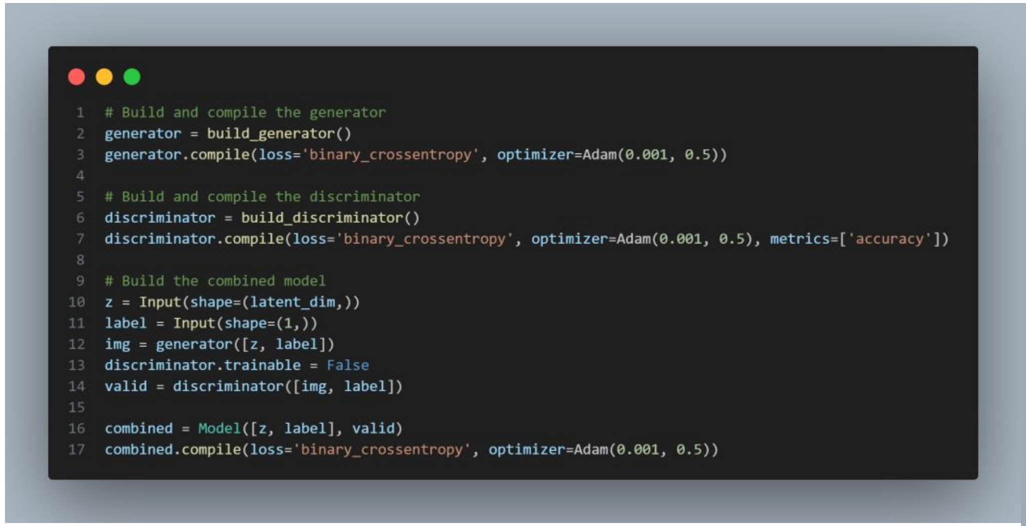
생성자



```
# Define the discriminator
def build_discriminator():
    img = Input(shape=(num_features,))
    label = Input(shape=(1,), dtype='int32')
    label_embedding = Flatten()(Embedding(num_labels, num_features)(label))

    model_input = Concatenate(axis=1)([img, label_embedding])

    model = Sequential()

    model.add(Dense(128, input_dim=num_features + num_features))
    model.add(Dense(64))
    model.add(Dense(1, activation='sigmoid'))

    validity = model(model_input)

    return Model([img, label], validity)
```

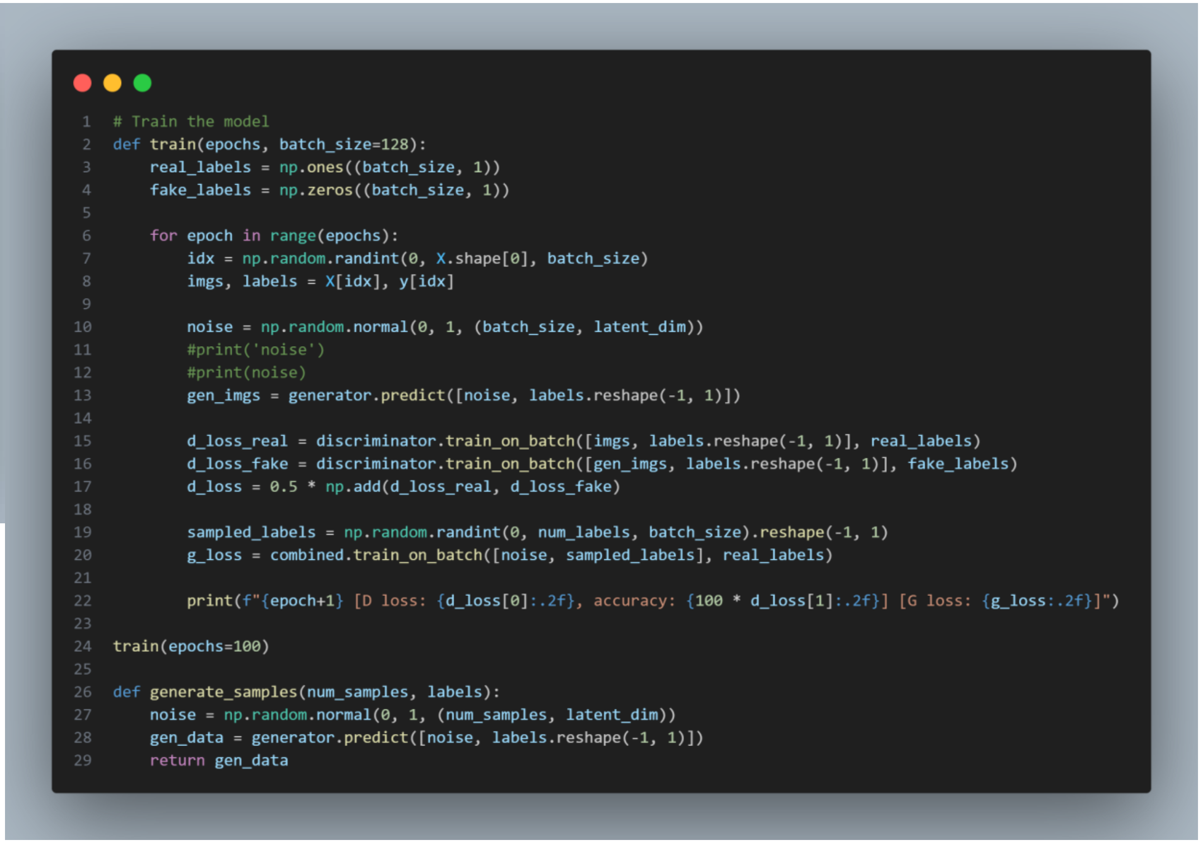
판별자

CGAN



```python
# Build and compile the generator
generator = build_generator()
generator.compile(loss='binary_crossentropy', optimizer=Adam(0.001, 0.5))

# Build and compile the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.001, 0.5), metrics=['accuracy'])

# Build the combined model
z = Input(shape=(latent_dim,))
label = Input(shape=(1,))
img = generator([z, label])
discriminator.trainable = False
valid = discriminator([img, label])

combined = Model([z, label], valid)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.001, 0.5))
```

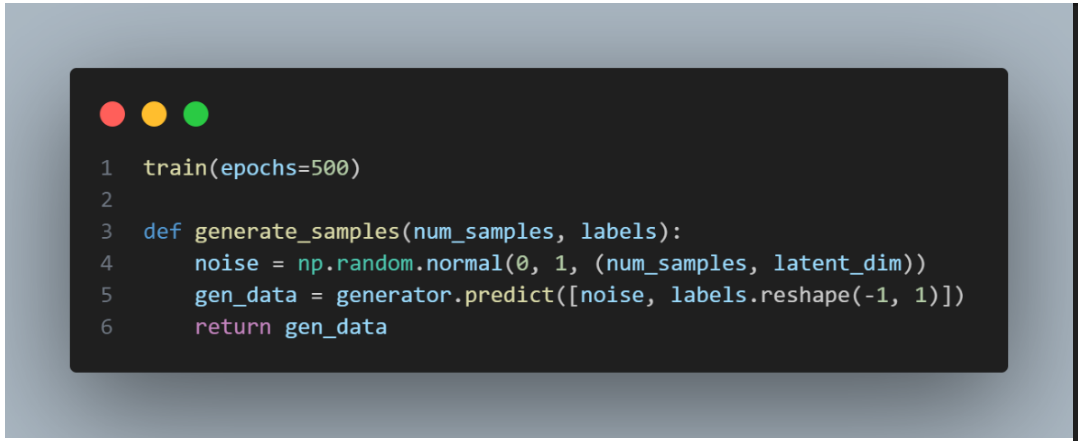
모델 생성



```python
# Train the model
def train(epochs, batch_size=128):
    real_labels = np.ones((batch_size, 1))
    fake_labels = np.zeros((batch_size, 1))

    for epoch in range(epochs):
        idx = np.random.randint(0, X.shape[0], batch_size)
        imgs, labels = X[idx], y[idx]

        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        #print('noise')
        #print(noise)
        gen_imgs = generator.predict([noise, labels.reshape(-1, 1)])

        d_loss_real = discriminator.train_on_batch([imgs, labels.reshape(-1, 1)], real_labels)
        d_loss_fake = discriminator.train_on_batch([gen_imgs, labels.reshape(-1, 1)], fake_labels)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        sampled_labels = np.random.randint(0, num_labels, batch_size).reshape(-1, 1)
        g_loss = combined.train_on_batch([noise, sampled_labels], real_labels)

        print(f"{epoch+1} [D loss: {d_loss[0]:.2f}, accuracy: {100 * d_loss[1]:.2f}] [G loss: {g_loss:.2f}]")

train(epochs=100)

def generate_samples(num_samples, labels):
    noise = np.random.normal(0, 1, (num_samples, latent_dim))
    gen_data = generator.predict([noise, labels.reshape(-1, 1)])
    return gen_data
```

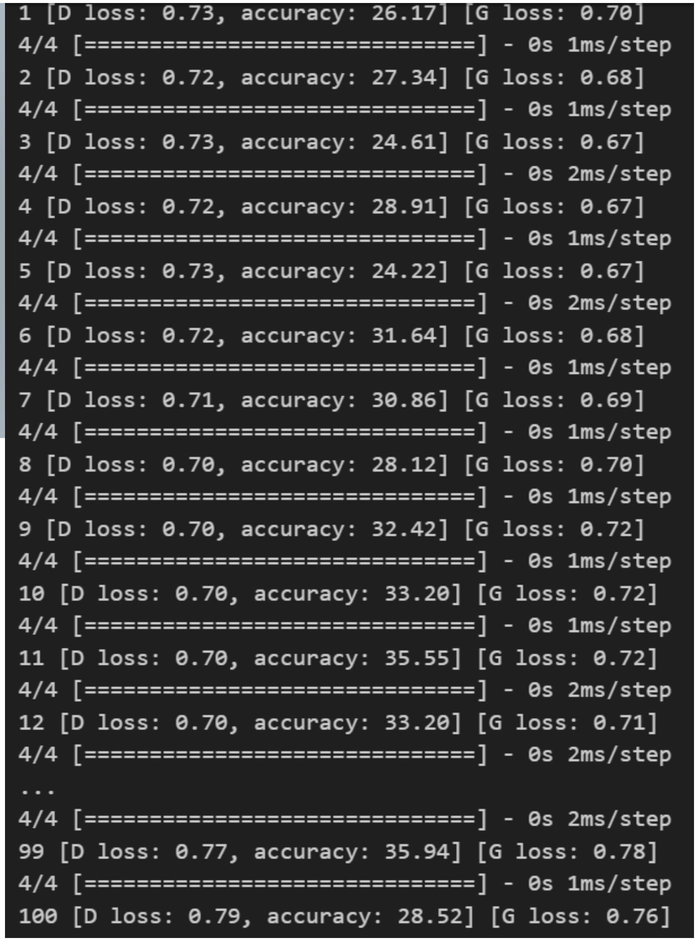
모델 학습

CGAN



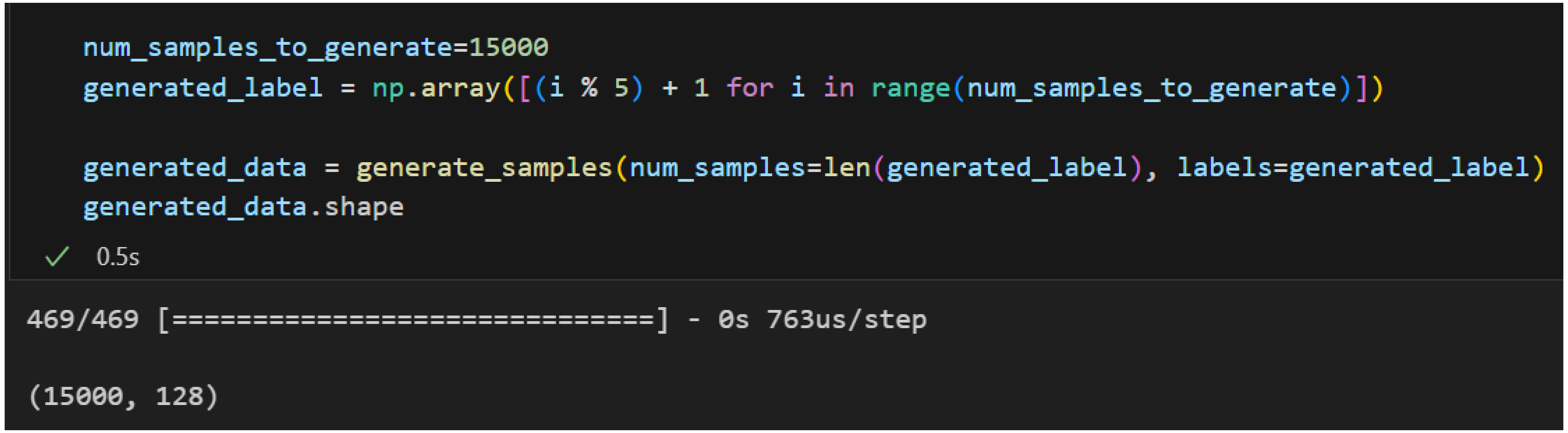
```python
train(epochs=500)

def generate_samples(num_samples, labels):
    noise = np.random.normal(0, 1, (num_samples, latent_dim))
    gen_data = generator.predict([noise, labels.reshape(-1, 1)])
    return gen_data
```

모델 학습 및 학습 과정



```
1 [D loss: 0.73, accuracy: 26.17] [G loss: 0.70]
4/4 [==============================] - 0s 1ms/step
2 [D loss: 0.72, accuracy: 27.34] [G loss: 0.68]
4/4 [==============================] - 0s 1ms/step
3 [D loss: 0.73, accuracy: 24.61] [G loss: 0.67]
4/4 [==============================] - 0s 2ms/step
4 [D loss: 0.72, accuracy: 28.91] [G loss: 0.67]
4/4 [==============================] - 0s 1ms/step
5 [D loss: 0.73, accuracy: 24.22] [G loss: 0.67]
4/4 [==============================] - 0s 2ms/step
6 [D loss: 0.72, accuracy: 31.64] [G loss: 0.68]
4/4 [==============================] - 0s 1ms/step
7 [D loss: 0.71, accuracy: 30.86] [G loss: 0.69]
4/4 [==============================] - 0s 1ms/step
8 [D loss: 0.70, accuracy: 28.12] [G loss: 0.70]
4/4 [==============================] - 0s 1ms/step
9 [D loss: 0.70, accuracy: 32.42] [G loss: 0.72]
4/4 [==============================] - 0s 1ms/step
10 [D loss: 0.70, accuracy: 33.20] [G loss: 0.72]
4/4 [==============================] - 0s 1ms/step
11 [D loss: 0.70, accuracy: 35.55] [G loss: 0.72]
4/4 [==============================] - 0s 2ms/step
12 [D loss: 0.70, accuracy: 33.20] [G loss: 0.71]
4/4 [==============================] - 0s 2ms/step
...
4/4 [==============================] - 0s 2ms/step
99 [D loss: 0.77, accuracy: 35.94] [G loss: 0.78]
4/4 [==============================] - 0s 1ms/step
100 [D loss: 0.79, accuracy: 28.52] [G loss: 0.76]
```

CGAN

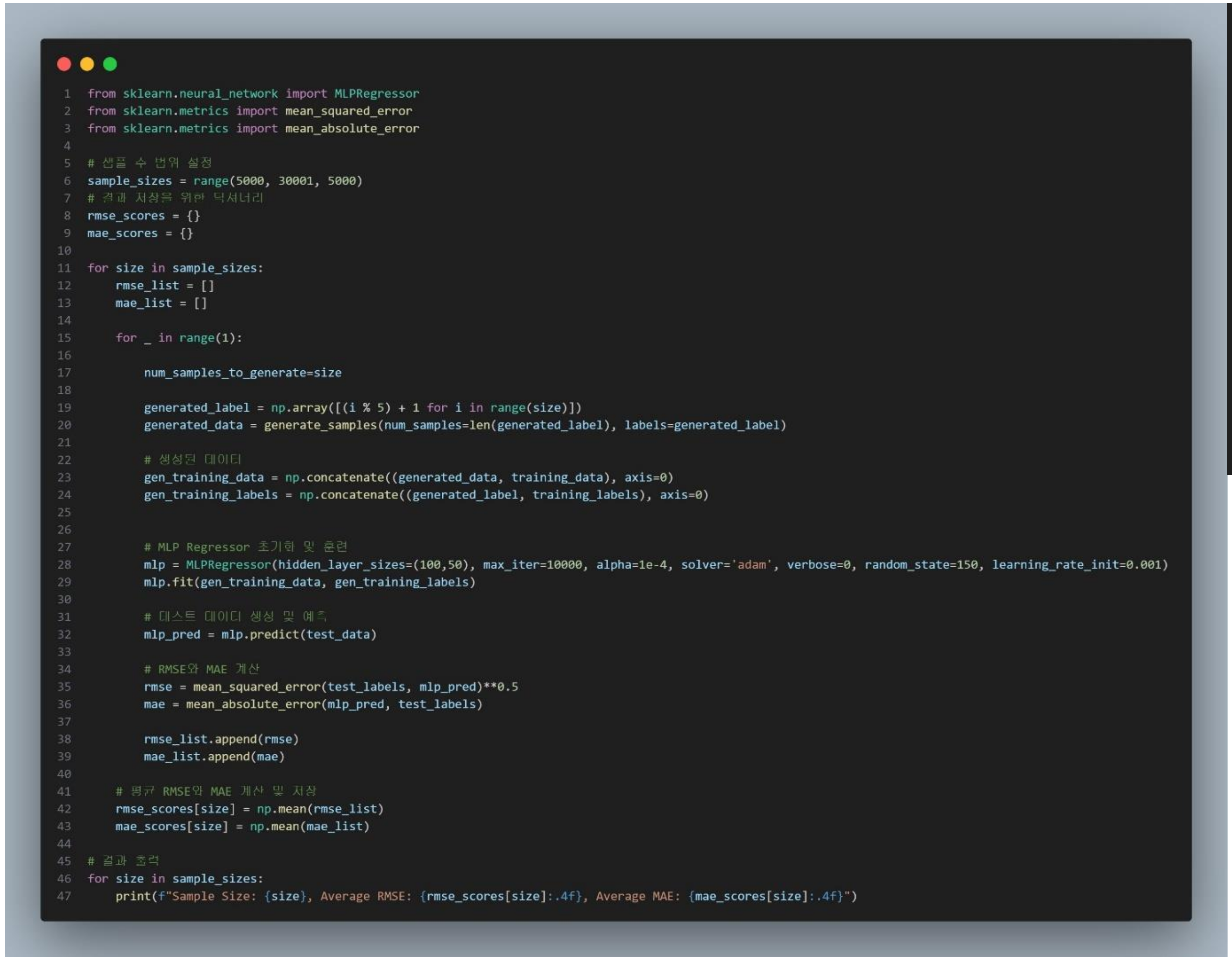


```
num_samples_to_generate=15000
generated_label = np.array([(i % 5) + 1 for i in range(num_samples_to_generate)])

generated_data = generate_samples(num_samples=len(generated_label), labels=generated_label)
generated_data.shape
```

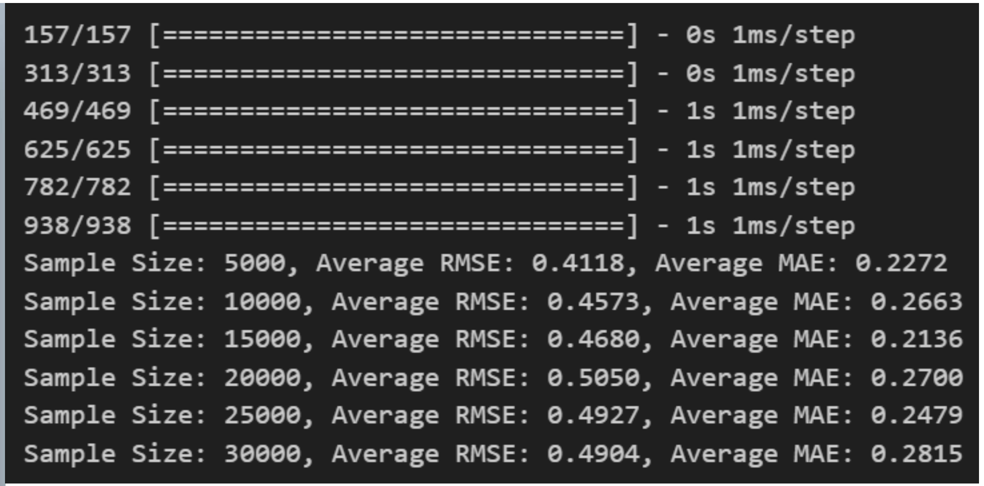
```
469/469 [==============================] - 0s 763us/step

(15000, 128)
```

학습된 CGAN 모델을 사용한 15,000개의 샘플 생성 예시

Predict



```python
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# 샘플 수 범위 설정
sample_sizes = range(5000, 30001, 5000)
# 결과 저장을 위한 딕셔너리
rmse_scores = {}
mae_scores = {}

for size in sample_sizes:
    rmse_list = []
    mae_list = []

    for _ in range(1):

        num_samples_to_generate=size

        generated_label = np.array([(i % 5) + 1 for i in range(size)])
        generated_data = generate_samples(num_samples=len(generated_label), labels=generated_label)

        # 생성된 데이터
        gen_training_data = np.concatenate((generated_data, training_data), axis=0)
        gen_training_labels = np.concatenate((generated_label, training_labels), axis=0)


        # MLP Regressor 초기화 및 훈련
        mlp = MLPRegressor(hidden_layer_sizes=(100,50), max_iter=10000, alpha=1e-4, solver='adam', verbose=0, random_state=150, learning_rate_init=0.001)
        mlp.fit(gen_training_data, gen_training_labels)

        # 테스트 데이터 생성 및 예측
        mlp_pred = mlp.predict(test_data)

        # RMSE와 MAE 계산
        rmse = mean_squared_error(test_labels, mlp_pred)**0.5
        mae = mean_absolute_error(mlp_pred, test_labels)

        rmse_list.append(rmse)
        mae_list.append(mae)

    # 평균 RMSE와 MAE 계산 및 저장
    rmse_scores[size] = np.mean(rmse_list)
    mae_scores[size] = np.mean(mae_list)

# 결과 출력
for size in sample_sizes:
    print(f"Sample Size: {size}, Average RMSE: {rmse_scores[size]:.4f}, Average MAE: {mae_scores[size]:.4f}")
```



```
157/157 [==============================] - 0s 1ms/step
313/313 [==============================] - 0s 1ms/step
469/469 [==============================] - 1s 1ms/step
625/625 [==============================] - 1s 1ms/step
782/782 [==============================] - 1s 1ms/step
938/938 [==============================] - 1s 1ms/step
Sample Size: 5000, Average RMSE: 0.4118, Average MAE: 0.2272
Sample Size: 10000, Average RMSE: 0.4573, Average MAE: 0.2663
Sample Size: 15000, Average RMSE: 0.4680, Average MAE: 0.2136
Sample Size: 20000, Average RMSE: 0.5050, Average MAE: 0.2700
Sample Size: 25000, Average RMSE: 0.4927, Average MAE: 0.2479
Sample Size: 30000, Average RMSE: 0.4904, Average MAE: 0.2815
```

생성된 샘플 사이즈에 따른 예측 정확도

기존 학습 데이터와 생성된 샘플을 결합하여 MLP를 통한 Rating 예측