```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Lambda
from tensorflow.keras.losses import mse, binary_crossentropy
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt

# Define sampling function for the latent space
def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon

# Define the Variational Autoencoder (VAE)
def build_vae(input_dim, latent_dim):
    # Encoder
    inputs = Input(shape=(input_dim,))
    h = Dense(128, activation='relu')(inputs)
    z_mean = Dense(latent_dim)(h)
    z_log_var = Dense(latent_dim)(h)
    z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])

    encoder = Model(inputs, [z_mean, z_log_var, z], name='encoder')
    encoder.summary()

    # Decoder
    latent_inputs = Input(shape=(latent_dim,))
    h_decoded = Dense(128, activation='relu')(latent_inputs)
    outputs = Dense(input_dim, activation='sigmoid')(h_decoded)

    decoder = Model(latent_inputs, outputs, name='decoder')
    decoder.summary()

    # VAE
    outputs = decoder(encoder(inputs)[2])
    vae = Model(inputs, outputs, name='vae')

    # Loss
    reconstruction_loss = binary_crossentropy(inputs, outputs)
    reconstruction_loss *= input_dim
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
    kl_loss = K.sum(kl_loss, axis=-1)
    kl_loss *= -0.5
    vae_loss = K.mean(reconstruction_loss + kl_loss)
    vae.add_loss(vae_loss)
    vae.compile(optimizer='adam')

    return encoder, decoder, vae

# Train and evaluate the VAE
def train_vae(vae, data, epochs, batch_size):
    history = vae.fit(data, data, epochs=epochs, batch_size=batch_size, validation_split=0.2)
    return history

def generate_samples(decoder, latent_dim, num_samples):
    random_latent_vectors = np.random.normal(size=(num_samples, latent_dim))
    generated_samples = decoder.predict(random_latent_vectors)
    return generated_samples

# Main script
def main():
    # Configuration
    input_dim = 100  # Replace with the actual feature size
    latent_dim = 10
    num_samples = 1000
    epochs = 50
    batch_size = 32

    # Simulate synthetic biological data
    data = np.random.rand(num_samples, input_dim)

    # Build and train VAE
    encoder, decoder, vae = build_vae(input_dim, latent_dim)
```

```
    train_vae(vae, data, epochs, batch_size)

    # Generate synthetic samples
    generated_samples = generate_samples(decoder, latent_dim, 10)

    # Visualize generated samples (example visualization for the first feature)
    plt.figure(figsize=(10, 6))
    plt.hist(generated_samples[:, 0], bins=20, alpha=0.7, label="Generated Feature 1")
    plt.title("Distribution of Generated Feature 1")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100)	0	-
dense (Dense)	(None, 128)	12,928	input_layer[0][0]
dense_1 (Dense)	(None, 10)	1,290	dense[0][0]
dense_2 (Dense)	(None, 10)	1,290	dense[0][0]
lambda (Lambda)	(None, 10)	0	dense_1[0][0], dense_2[0][0]

Total params: 15,508 (60.58 KB)
Trainable params: 15,508 (60.58 KB)
Non-trainable params: 0 (0.00 B)

Model: "decoder"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 10)	0
dense_3 (Dense)	(None, 128)	1,408
dense_4 (Dense)	(None, 100)	12,900

Total params: 14,308 (55.89 KB)
Trainable params: 14,308 (55.89 KB)
Non-trainable params: 0 (0.00 B)

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-391bdef349b2> in <cell line: 91>()
     90
     91 if __name__ == "__main__":
---> 92     main()

 6 frames
/usr/local/lib/python3.10/dist-packages/keras/src/backend/common/keras_tensor.py in __tf_tensor__(self, dtype, name)
    136
    137     def __tf_tensor__(self, dtype=None, name=None):
--> 138         raise ValueError(
    139             "A KerasTensor cannot be used as input to a TensorFlow function. "
    140             "A KerasTensor is a symbolic placeholder for a shape and dtype, "

ValueError: A KerasTensor cannot be used as input to a TensorFlow function. A KerasTensor is a symbolic placeholder for a shape and
dtype, used when constructing Keras Functional models or Keras Functions. You can only use it as input to a Keras layer or a Keras
operation (from the namespaces `keras.layers` and `keras.operations`). You are likely doing something like:

```
x = Input(...)
...
tf_fn(x)  # Invalid.
```

What you should do instead is wrap `tf_fn` in a layer:

```
class MyLayer(Layer):
    def call(self, x):
        return tf_fn(x)

x = MyLayer()(x)
```
```

Next steps: Explain error