In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
WARNING:tensorflow:From c:\Users\SA RAVI\anaconda3\envs\aimlsem1\lib\site-package
s\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is de
precated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:
```
file_path = "D:/2nd sem/Deep-learning/ICU_filtered.csv"  # Specify the file path
dataset = pd.read_csv(file_path)
```

In [ ]:
```
# Filter data to include only samples where "In-hospital_death" is equal to 1 fo
train_dataset = dataset[dataset["In-hospital_death"] == 1]

# Preprocessing steps for training data (replace this with your preprocessing co
# Impute missing values
imputer = SimpleImputer(strategy='mean')  # Use mean imputation for missing valu
X_train_imputed = imputer.fit_transform(train_dataset.drop(columns=["In-hospital
y_train = train_dataset["In-hospital_death"]

# Normalize the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)

# Preprocessing steps for test data (replace this with your preprocessing code)
X_test_imputed = imputer.transform(dataset.drop(columns=["In-hospital_death"]))
y_test = dataset["In-hospital_death"]

# Normalize the features
X_test_scaled = scaler.transform(X_test_imputed)
```

In [ ]:
```
## Parameters for the autoencoder
batch_size = 256
max_epochs = 50
learning_rate = 1e-03
latent_dim = 128
hidden_dim = 256
original_dim = X_train_scaled.shape[1]
```

In [ ]:
```
training_dataset = tf.data.Dataset.from_tensor_slices((X_train_scaled, y_train))
test_dataset = tf.data.Dataset.from_tensor_slices((X_test_scaled, y_test)).batch
```

In [ ]:
```
## Encoder
class Encoder(tf.keras.layers.Layer):
  # Define input independent model information
  def __init__(self, hidden_dim, latent_dim):
    super(Encoder, self).__init__()
```

```
    self.encoder_layer1 = tf.keras.layers.Dense(units = hidden_dim, activation =
    self.encoder_layer2 = tf.keras.layers.Dense(units = latent_dim, activation =

  ## Method for forward propagation
  def call(self, input_features):
    a = self.encoder_layer1(input_features)
    a = self.encoder_layer2(a)
    return a
```

In [ ]:
```
## Decoder
class Decoder(tf.keras.layers.Layer):
  def __init__(self, latent_dim, hidden_dim, original_dim):
    super(Decoder, self).__init__()
    self.decoder_layer1 = tf.keras.layers.Dense(units = hidden_dim, activation =
    self.decoder_layer2 = tf.keras.layers.Dense(units = original_dim, activation

  def call(self, encoded_features):
    a = self.decoder_layer1(encoded_features)
    a = self.decoder_layer2(a)
    return a
```

In [ ]:
```
## Autoencoder
class Autoencoder(tf.keras.Model):
  def __init__(self, latent_dim, hidden_dim, original_dim):
    super(Autoencoder, self).__init__()
    self.loss = []
    self.encoder = Encoder(hidden_dim = hidden_dim, latent_dim = latent_dim)
    self.decoder = Decoder(latent_dim = latent_dim, hidden_dim = hidden_dim, ori

  def call(self, input_features):
    encoded_features = self.encoder(input_features)
    reconstructed_features = self.decoder(encoded_features)
    return reconstructed_features
```

In [ ]:
```
## Build model
autoencoder = Autoencoder(latent_dim = latent_dim,
                          hidden_dim = hidden_dim,
                          original_dim = original_dim)
```

```
WARNING:tensorflow:From c:\Users\SA RAVI\anaconda3\envs\aimlsem1\lib\site-package
s\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please u
se tf.compat.v1.get_default_graph instead.
```

In [ ]:
```
## Optimizer
opt = tf.keras.optimizers.Adam(learning_rate = learning_rate)
```

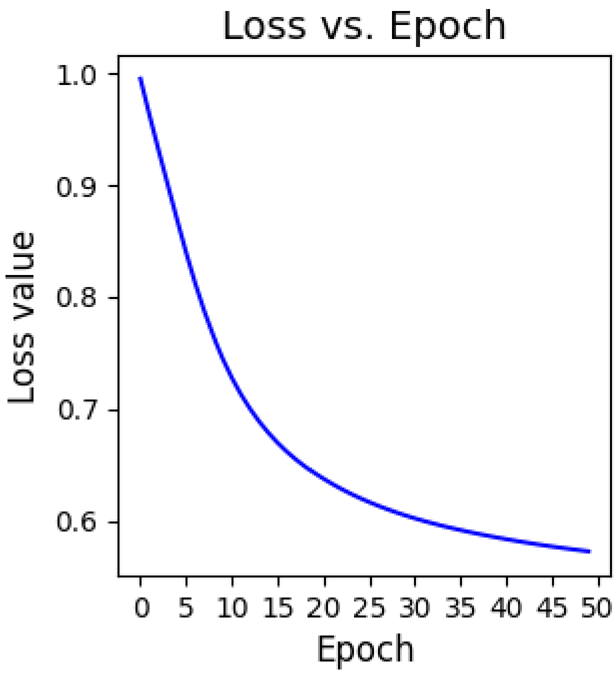
In [ ]:
```
## Custom training - loss
def loss(true, pred):
  return tf.reduce_mean(tf.square(tf.subtract(true, pred)))

## Custom training - compute gradient of loss and update weights
@tf.function
def train(model, loss, opt, original_features, labels):
  with tf.GradientTape() as g:
    pred = tf.cast(model(original_features), tf.float64)
    loss_batch = loss(original_features, pred)
  gradients = g.gradient(loss_batch, model.trainable_variables)
```

```
  opt.apply_gradients(zip(gradients, model.trainable_variables))
  return loss_batch
```

In [ ]:
```
## Train network
# Variable to store training loss per epoch
loss_train_epoch = tf.keras.metrics.Mean()
loss_train_epoch_plot = np.empty(max_epochs)

# Iterate over epochs
for epoch in range(max_epochs):
  for step, (train_batch_features, train_batch_labels) in enumerate(training_dat
    loss_batch = train(autoencoder, loss, opt, train_batch_features, train_batch
    # Append training loss
    loss_train_epoch(loss_batch)
  loss_train_epoch_plot[epoch] = loss_train_epoch.result().numpy()
  print(f'Epoch {epoch+1}, loss = {loss_train_epoch_plot[epoch]}')

# Plot train loss as a function of epoch:
fig, ax = plt.subplots(1, 1, figsize = (4, 4))
fig.tight_layout(pad = 4.0)
ax.plot(loss_train_epoch_plot, 'b')
ax.set_xlabel('Epoch', fontsize = 12)
ax.set_ylabel('Loss value', fontsize = 12)
ax.set_xticks(np.arange(0, max_epochs+1, 5))
ax.set_title('Loss vs. Epoch', fontsize = 14)
```

```
Epoch 1, loss = 0.9945636987686157
Epoch 2, loss = 0.9617888331413269
Epoch 3, loss = 0.9303098320960999
Epoch 4, loss = 0.8995876312255859
Epoch 5, loss = 0.8690134286880493
Epoch 6, loss = 0.8393010497093201
Epoch 7, loss = 0.8116855025291443
Epoch 8, loss = 0.7867234945297241
Epoch 9, loss = 0.7644516229629517
Epoch 10, loss = 0.7448784708976746
Epoch 11, loss = 0.7278242707252502
Epoch 12, loss = 0.7129732370376587
Epoch 13, loss = 0.700039803981781
Epoch 14, loss = 0.6886584758758545
Epoch 15, loss = 0.6786787509918213
Epoch 16, loss = 0.6697908043861389
Epoch 17, loss = 0.6618680357933044
Epoch 18, loss = 0.6548421382904053
Epoch 19, loss = 0.6485129594802856
Epoch 20, loss = 0.6427395343780518
Epoch 21, loss = 0.6374773383140564
Epoch 22, loss = 0.6326277256011963
Epoch 23, loss = 0.6281382441520691
Epoch 24, loss = 0.6239843964576721
Epoch 25, loss = 0.6201500296592712
Epoch 26, loss = 0.6165828108787537
Epoch 27, loss = 0.6132725477218628
Epoch 28, loss = 0.6101773977279663
Epoch 29, loss = 0.6072906255722046
Epoch 30, loss = 0.6046034097671509
Epoch 31, loss = 0.6020655035972595
Epoch 32, loss = 0.5997122526168823
Epoch 33, loss = 0.5975169539451599
Epoch 34, loss = 0.5954146981239319
Epoch 35, loss = 0.5934615731239319
Epoch 36, loss = 0.5916009545326233
Epoch 37, loss = 0.5898301601409912
Epoch 38, loss = 0.5881361365318298
Epoch 39, loss = 0.5865251421928406
Epoch 40, loss = 0.5849697589874268
Epoch 41, loss = 0.5834925770759583
Epoch 42, loss = 0.5820692777633667
Epoch 43, loss = 0.5807075500488281
Epoch 44, loss = 0.5794033408164978
Epoch 45, loss = 0.5781522393226624
Epoch 46, loss = 0.5769534707069397
Epoch 47, loss = 0.5758028030395508
Epoch 48, loss = 0.5746982097625732
Epoch 49, loss = 0.5736390948295593
Epoch 50, loss = 0.5726237297058105
```

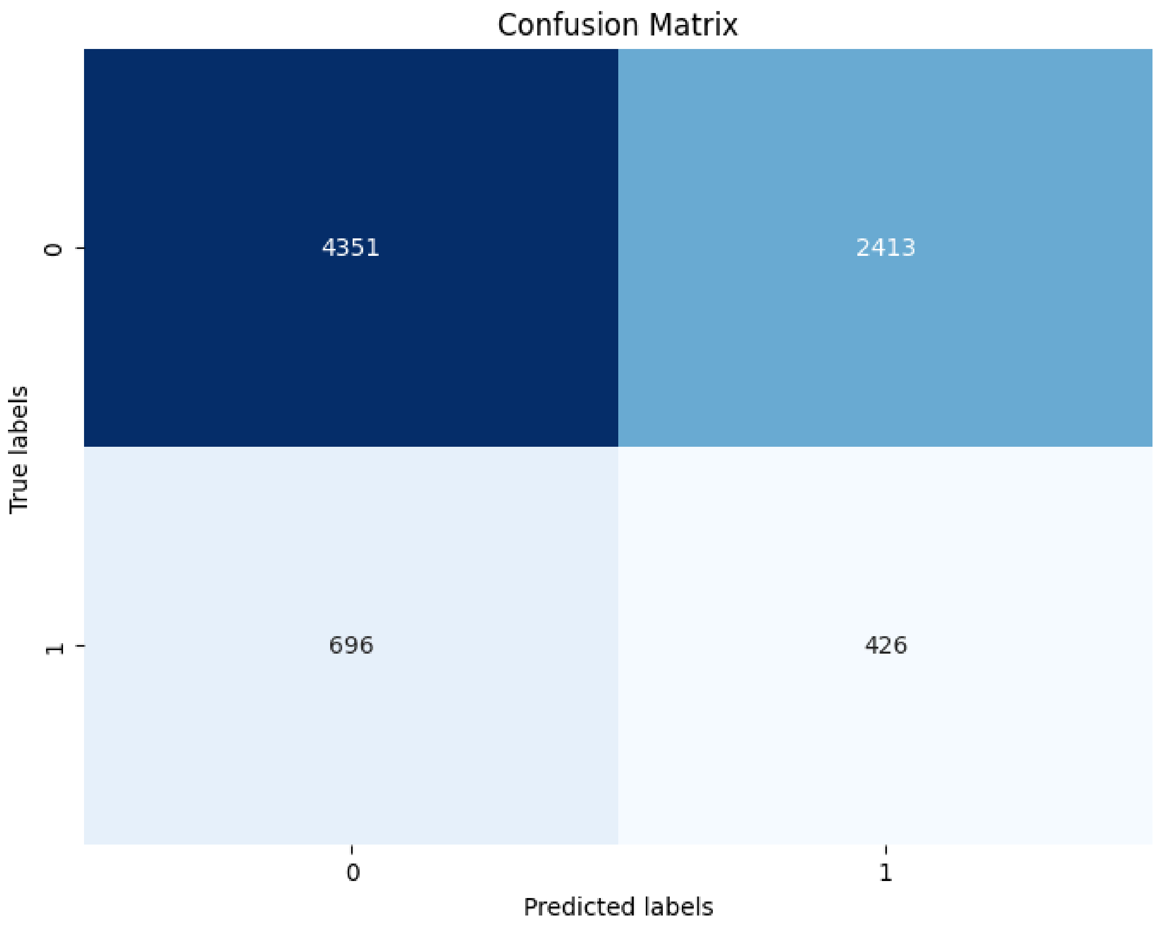
Out[ ]: `Text(0.5, 1.0, 'Loss vs. Epoch')`



In [ ]:

```
# Calculate reconstruction errors on test data
reconstructions = autoencoder.predict(X_test_scaled)
mse = np.mean(np.square(X_test_scaled - reconstructions), axis=1)

# Define a threshold for classification
threshold = 0.5  # You can adjust this threshold as per your requirement

# Classify instances based on reconstruction error
y_pred = np.where(mse > threshold, 1, 0)

# Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix')
    plt.show()

# Flatten true labels
y_test_flat = y_test.values.flatten()

# Plot confusion matrix
plot_confusion_matrix(y_test_flat, y_pred)
```

```
247/247 [==============================] - 1s 2ms/step
```


Confusion Matrix
4351
2413
696
426
0
1
True labels
0
1
Predicted labels