In [43]:

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from tensorflow.keras.callbacks import EarlyStopping
from imblearn.over_sampling import SMOTE
```

In [ ]:

```python
data = pd.read_csv("/content/heart_failure_clinical_records_dataset.csv")
```

In [ ]:

```python
data.head()
```

Out[ ]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodi
0	75.0	0	582	0	20	1	265000.00	1.9	
1	55.0	0	7861	0	38	0	263358.03	1.1	
2	65.0	0	146	0	20	0	162000.00	1.3	
3	50.0	1	111	0	20	0	210000.00	1.9	
4	65.0	1	160	1	20	0	327000.00	2.7	

In [ ]:

```python
EF = data['ejection_fraction'].values
EF
```

Out[ ]:

```
array([20, 38, 20, 20, 20, 40, 15, 60, 65, 35, 38, 25, 30, 38, 30, 50, 38,
       14, 25, 55, 25, 30, 35, 60, 30, 38, 40, 45, 38, 30, 38, 45, 35, 30,
       50, 35, 50, 50, 30, 38, 20, 30, 45, 50, 60, 38, 25, 38, 20, 30, 25,
       20, 62, 50, 38, 30, 35, 40, 20, 20, 25, 40, 35, 35, 80, 20, 15, 25,
       25, 25, 40, 35, 35, 50, 20, 20, 60, 40, 38, 45, 40, 50, 25, 50, 25,
       50, 35, 60, 40, 25, 45, 45, 60, 25, 38, 60, 25, 60, 25, 40, 25, 45,
       25, 30, 50, 30, 45, 35, 38, 35, 60, 35, 25, 60, 40, 40, 60, 60, 60,
       38, 60, 38, 38, 30, 40, 50, 17, 60, 30, 35, 60, 45, 40, 60, 35, 40,
       60, 25, 35, 30, 38, 35, 30, 40, 25, 30, 30, 60, 30, 35, 45, 60, 45,
       35, 35, 25, 35, 25, 50, 45, 40, 35, 40, 35, 30, 38, 60, 20, 40, 35,
       35, 40, 60, 20, 35, 60, 40, 50, 60, 40, 30, 25, 25, 38, 25, 30, 50,
       25, 40, 45, 35, 60, 40, 30, 20, 45, 38, 30, 20, 35, 45, 60, 60, 25,
       40, 45, 40, 38, 40, 35, 17, 62, 50, 30, 35, 35, 50, 70, 35, 35, 20,
       50, 35, 25, 25, 60, 25, 35, 25, 25, 30, 35, 35, 38, 45, 50, 50, 30,
       40, 45, 35, 30, 35, 40, 38, 38, 25, 25, 35, 40, 30, 35, 45, 35, 60,
       30, 38, 38, 25, 50, 40, 40, 25, 60, 38, 35, 20, 38, 38, 35, 30, 40,
```

```
       38, 40, 30, 38, 35, 38, 30, 38, 40, 40, 30, 38, 40, 40, 35, 55, 35,
       38, 55, 35, 38, 35, 38, 38, 60, 38, 45])
```

In [ ]:

```
data['encoded_EF'] = np.where(data['ejection_fraction'] < 50, 0, 1)
data.drop(columns=['ejection_fraction'], inplace=True)

data.head()
```

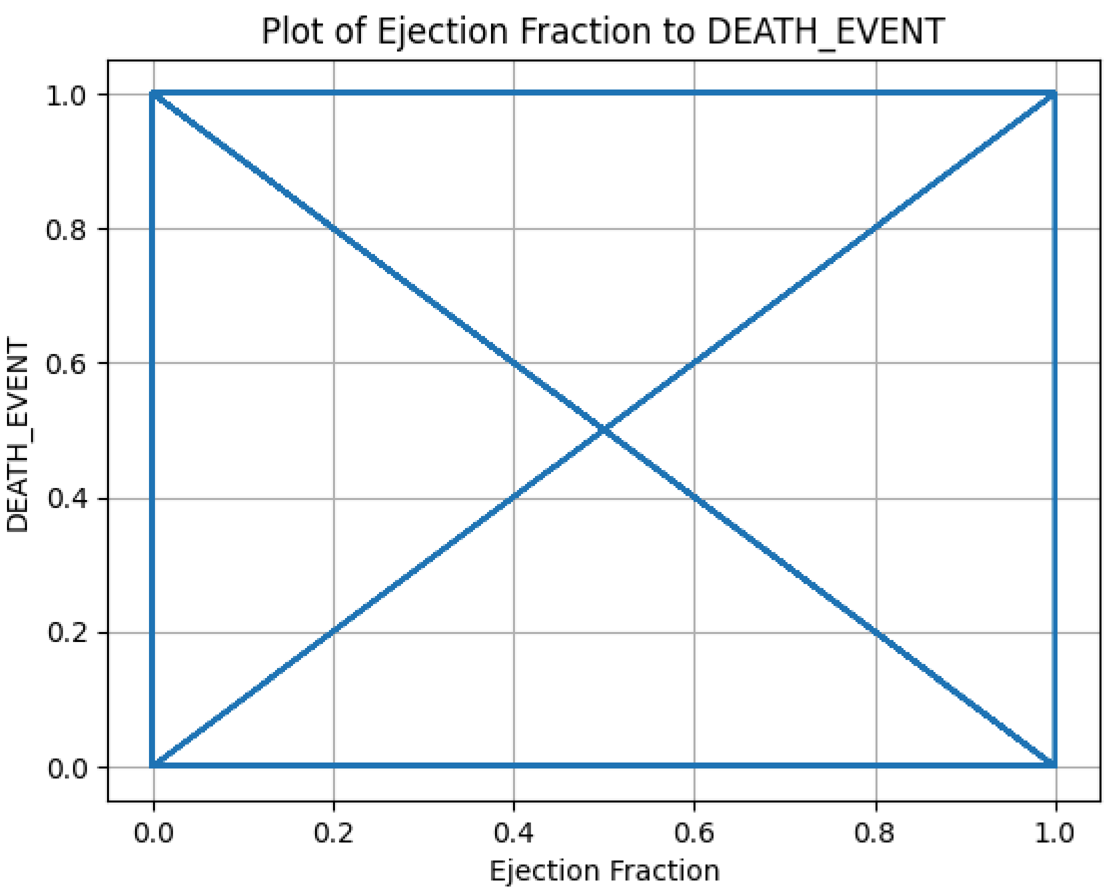
Out[ ]:

	age	anaemia	creatinine_phosphokinase	diabetes	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking
0	75.0	0	582	0	1	265000.00	1.9	130	1	(
1	55.0	0	7861	0	0	263358.03	1.1	136	1	(
2	65.0	0	146	0	0	162000.00	1.3	129	1	:
3	50.0	1	111	0	0	210000.00	1.9	137	1	(
4	65.0	1	160	1	0	327000.00	2.7	116	0	(

In [16]:

```
# Investigation corr. between ejection_fraction and death

# Plotting
plt.plot(data['encoded_EF'], data['DEATH_EVENT'])  # 'o' for scatter plot, you can change it to '-' for line plot
plt.xlabel('Ejection Fraction')
plt.ylabel('DEATH_EVENT')
plt.title('Plot of Ejection Fraction to DEATH_EVENT')
plt.grid(True)
plt.show()
```

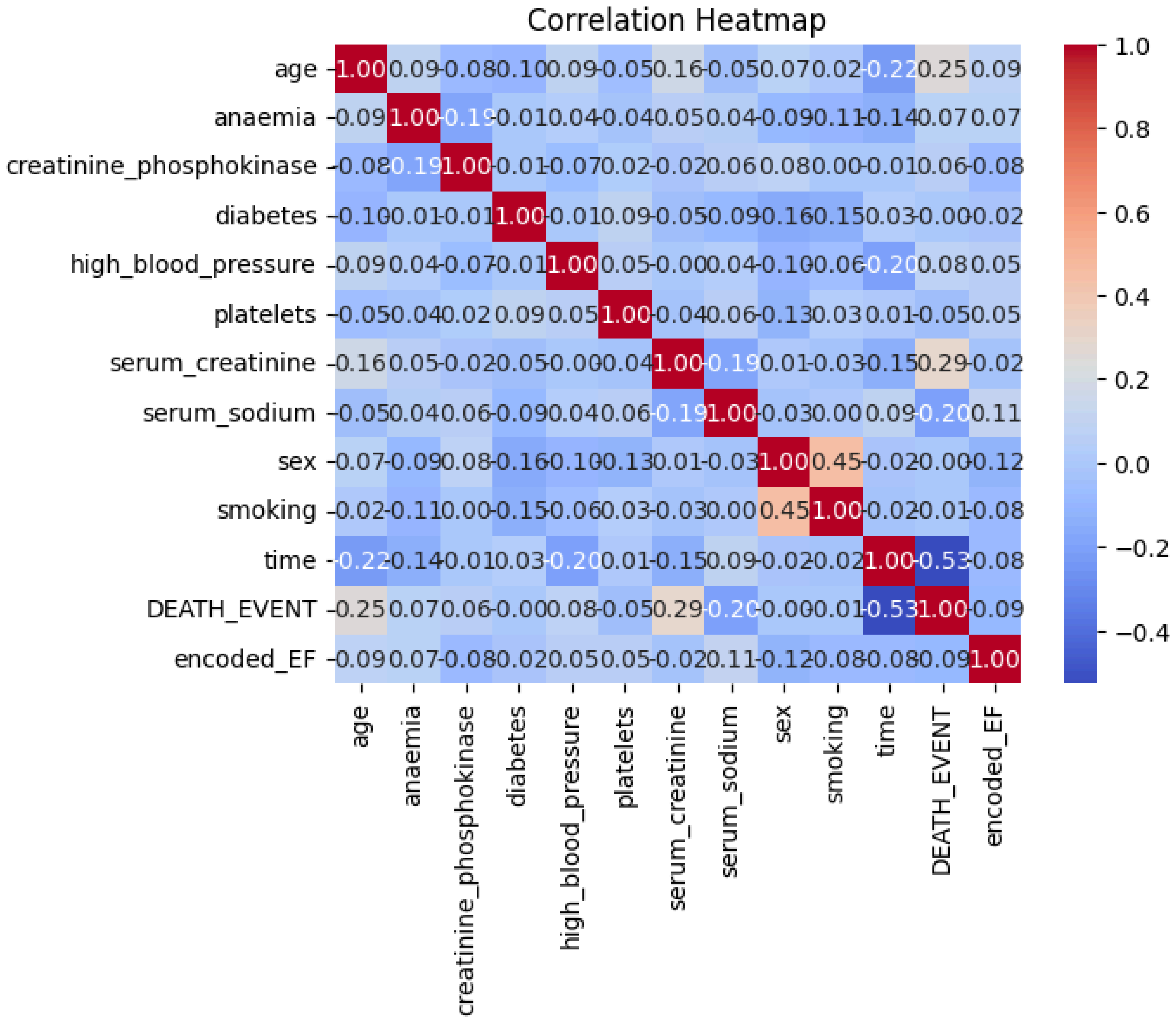


In [17]:

```
import seaborn as sns
```

```python
# Calculate the correlation matrix
corr_matrix = data.corr()

# Plot the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



In [18]:
```python
data.shape
```

Out[18]: (299, 13)

In [19]:
```python
column_names = list(data.columns)

print(column_names)
```

```
['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes', 'high_blood_pressure', 'platelets', 'serum_creatinine',
'serum_sodium', 'sex', 'smoking', 'time', 'DEATH_EVENT', 'encoded_EF']
```

In [20]:
```python
X = data.drop(['sex', 'time', 'DEATH_EVENT', 'encoded_EF'], axis=1)
```

In [28]:

```
y = data['encoded_EF']
y.describe()
```

Out[28]:

```
count    299.000000
mean       0.200669
std        0.401172
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: encoded_EF, dtype: float64
```

In [35]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Calculate class weights manually
total_samples = len(y_train)
class_counts = np.bincount(y_train)
class_weights = {i: total_samples / (len(class_counts) * class_counts[i]) for i in range(len(class_counts))}
class_weights
```

Out[35]:

```
{0: 0.6223958333333334, 1: 2.5425531914893615}
```

TENSORFLOW

In [38]:

```
# Define custom loss function with class weights
def weighted_binary_crossentropy(y_true, y_pred):
    # Cast y_true to float32 to match the data type of y_pred
    y_true = tf.cast(y_true, tf.float32)

    # Clip predicted values to prevent log(0) and log(1) issues
    y_pred = tf.clip_by_value(y_pred, 1e-7, 1 - 1e-7)

    # Calculate weighted binary crossentropy
    loss = -(class_weights[0] * y_true * tf.math.log(y_pred) + class_weights[1] * (1 - y_true) * tf.math.log(1 - y
    return tf.reduce_mean(loss)


# Define and compile your model with custom loss function
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

In [40]:

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=optimizer, loss=weighted_binary_crossentropy, metrics=['accuracy'])

# Train the model
print("Training...")
history = model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)

# Evaluate the model
print("Evaluating...")
```

```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
Training...
Epoch 1/100
8/8 [==============================] - 1s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 2/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 3/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 4/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 5/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 6/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 7/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 8/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 9/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 10/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 11/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 12/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 13/100
8/8 [==============================] - 0s 2ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 14/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 15/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 16/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 17/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 18/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 19/100
8/8 [==============================] - 0s 4ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 20/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 21/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 22/100
8/8 [==============================] - 0s 4ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 23/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 24/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 25/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 26/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 27/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 28/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 29/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 30/100
```

```
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 31/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 32/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 33/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 34/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 35/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 36/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 37/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 38/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 39/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 40/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 41/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
```



```
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 45/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 46/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 47/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 48/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 49/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 50/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 51/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 52/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 53/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 54/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 55/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 56/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 57/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 58/100
8/8 [==============================] - 0s 2ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 59/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 60/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 61/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 62/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
```

```
Epoch 63/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 64/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 65/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 66/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 67/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 68/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 69/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 70/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 71/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 72/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 73/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 74/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 75/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 76/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 77/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 78/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 79/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 80/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 81/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 82/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 83/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 84/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 85/100
8/8 [==============================] - 0s 4ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 86/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 87/100
8/8 [==============================] - 0s 4ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 88/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 89/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 90/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 91/100
8/8 [==============================] - 0s 4ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 92/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 93/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 94/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
```

```
Epoch 95/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 96/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 97/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 98/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 99/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Epoch 100/100
8/8 [==============================] - 0s 3ms/step - loss: 1.9728 - accuracy: 0.8033
Evaluating...
2/2 [==============================] - 0s 6ms/step - loss: 2.1736 - accuracy: 0.7833
Test Loss: 2.1735646724700928
Test Accuracy: 0.7833333611488342
```

RANDOM FOREST

In [42]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Random Forest Classifier
print("Random Forest Classifier:")
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train_resampled, y_train_resampled)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Accuracy:", rf_accuracy)
print("Classification Report:")
print(classification_report(y_test, rf_predictions))
```

```
Random Forest Classifier:
Accuracy: 0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.91      0.88        47
           1       0.56      0.38      0.45        13

    accuracy                           0.80        60
   macro avg       0.70      0.65      0.67        60
weighted avg       0.78      0.80      0.79        60
```

In [47]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Calculate confusion matrix
cm = confusion_matrix(y_test, rf_predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Random Forest Classifier')
```

```
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```