'telecom.csv'

In [4]:
```python
import pandas as pd
import numpy as np

pd.set_option('display.max_columns', None)
df = pd.read_csv("data/telecom.csv")

del df["customerID"]
df = df.replace({"TotalCharges": {" ": 0},
                 "Churn": {"Yes": 1, "No": 0}})

y = df["Churn"]
del df["Churn"]

df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], downcast="float")

df = df.replace({"gender": {"Male": 1, "Female": 0},
                 "Partner": {"Yes": 1, "No": 0},
                 "Dependents": {"Yes": 1, "No": 0},
                 "PhoneService": {"Yes": 1, "No": 0},
                 "MultipleLines": {"Yes": 2, "No": 1, "No phone service": 0},
                 "InternetService": {"Fiber optic": 2, "DSL": 1, "No": 0},
                 "OnlineSecurity": {"Yes": 2, "No": 1, "No internet service": 0},
                 "OnlineBackup": {"Yes": 2, "No": 1, "No internet service": 0},
                 "DeviceProtection": {"Yes": 2, "No": 1, "No internet service": 0},
                 "TechSupport": {"Yes": 2, "No": 1, "No internet service": 0},
                 "StreamingTV": {"Yes": 2, "No": 1, "No internet service": 0},
                 "StreamingMovies": {"Yes": 2, "No": 1, "No internet service": 0},
                 "Contract": {"Month-to-month": 2, "Two year": 1, "One year": 0},
                 "PaperlessBilling": {"Yes": 1, "No": 0},
                 "PaymentMethod": {"Electronic check": 3, "Mailed check": 2, "Credit ca
```

In [3]:
```python
df
```

Out[3]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	0	0	1	1	1	1	1	0
1	1	0	0	0	50	1	1	2
2	1	0	1	0	55	1	2	2
3	0	1	1	0	7	1	2	2
4	0	1	0	0	2	1	1	2
...	...	...	...	...	...	...	...	...
5995	1	1	0	0	2	1	1	2
5996	1	0	1	1	72	1	2	2
5997	0	1	0	0	7	1	1	1
5998	1	0	1	0	70	1	2	0
5999	0	1	1	1	66	1	2	2

6000 rows × 19 columns

Split data into 2 parts, take the small one as the final test. You will be using for each model's final evaluation

In [3]:
```python
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split(df, y, test_size=1/6,random_state=
```

Creating a dictionary for kepping the metrics from each model and the necessary functions

In [4]:
```python
models = ["Logistic Regression", "Decision Tree", "SVM", "KNN", "Random Forest", "Ense
coefs = dict()

for model in models:
    coefs[model] = {
        "Accuracy": 0,
        "Precision": 0,
        "Recall | Sensitivity": 0,
        "Specificity": 0,
        "Negative predictive value": 0
    }


def perf_measure(y_actual, y_hat):
    TP, FP, TN, FN = 0, 0, 0, 0

    for i in range(len(y_actual)):
        if y_hat[i] == 1 and y_actual[i] == 1:
            TP += 1
        elif y_hat[i] == 1 and y_actual[i] == 0:
            FP += 1
        elif y_hat[i] == 0 and y_actual[i] == 0:
            TN += 1
        elif y_hat[i] == 0 and y_actual[i] == 1:
            FN += 1

    return TP, FP, TN, FN


def return_metrics(tp, fp, tn, fn):
    metrics = list()

    metrics.append(round(((tp + tn) / (tp + tn + fp + fn)), 2))
    metrics.append(round((tp / (tp + fp)), 2))
    metrics.append(round((tp / (tp + fn)), 2))
    metrics.append(round((tn / (tn + fp)), 2))
    metrics.append(round((tn / (tn + fn)), 2))

    return metrics


def update_metrics(metrics, model):
    coefs[model]["Accuracy"] = metrics[0]
    coefs[model]["Precision"] = metrics[1]
    coefs[model]["Recall | Sensitivity"] = metrics[2]
    coefs[model]["Specificity"] = metrics[3]
    coefs[model]["Negative predictive value"] = metrics[4]


def print_metrics(metrics):
    print("Accuracy:", metrics[0])
    print("Precision:", metrics[1])
    print("Recall | Sensitivity:", metrics[2])
    print("Specificity:", metrics[3])
    print("Negative predictive value:", metrics[4])
```

Logistic regression

In [5]:
```python
import warnings
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

warnings.filterwarnings('ignore')

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

param_grid = {"penalty": ["l1", "l2", "elasticnet", "none"],
              "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
              "multi_class": ["auto", "ovr", "multinomial"]}

grid_search = GridSearchCV(LogisticRegression(random_state=0), param_grid, cv=6, n_jol
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "Logistic Regression")
print_metrics(metrics)

grid_search.best_estimator_
```

```
Accuracy: 0.8
Precision: 0.68
Recall | Sensitivity: 0.56
Specificity: 0.9
Negative predictive value: 0.84
```

Out[5]:
```
LogisticRegression(random_state=0, solver='newton-cg')
```

Decision tree

In [6]:
```python
from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

param_grid = {"max_depth": range(3, 7),
              "min_samples_split": range(2, 7),
              "min_samples_leaf": range(1, 4),
              "max_features": range(10, 15)}

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=6, n_jobs=2)
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "Decision Tree")
print_metrics(metrics)
```

```
Accuracy: 0.79
Precision: 0.71
Recall | Sensitivity: 0.43
Specificity: 0.93
Negative predictive value: 0.81
```

SVM

In [7]:
```python
from sklearn import svm

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

param_grid = {"kernel": ["sigmoid"]}

grid_search = GridSearchCV(svm.SVC(), param_grid, cv=6, n_jobs=2)
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "SVM")
print_metrics(metrics)
```

```
Accuracy: 0.59
Precision: 0.23
Recall | Sensitivity: 0.21
Specificity: 0.73
Negative predictive value: 0.7
```

KNN

In [8]:
```python
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

param_grid = {"n_neighbors": range(3, 6),
              "weights": ["uniform", "distance"],
              "algorithm": ["auto", "ball_tree", "kd_tree", "brute"]}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=6, n_jobs=2)
grid_search.fit(X_train, y_train)

y_pred = grid_search.best_estimator_.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "KNN")
print_metrics(metrics)

grid_search.best_estimator_
```

```
Accuracy: 0.77
Precision: 0.68
Recall | Sensitivity: 0.32
Specificity: 0.94
Negative predictive value: 0.78
```

Out[8]:
```
KNeighborsClassifier(n_neighbors=4)
```

Random Forest

In [9]:
```python
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

param_grid = {"n_estimators": range(2, 20)}

grid_search = GridSearchCV(RandomForestClassifier(random_state=1), param_grid, cv=6, r
grid_search.fit(X_train, y_train)

y_pred = grid_search.best_estimator_.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "Random Forest")
print_metrics(metrics)
```

```
Accuracy: 0.81
Precision: 0.73
Recall | Sensitivity: 0.5
Specificity: 0.93
Negative predictive value: 0.83
```

Ensemble Learning

In [10]:
```python
import warnings
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingClassifier

warnings.filterwarnings('ignore')

X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.2,ra

estimators = [
    ('rf', RandomForestClassifier(n_estimators=16, random_state=1)),
    ('lr', LogisticRegression(random_state=0, solver='newton-cg'))
]

clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression(random_state=0, solver=
)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

tp, fp, tn, fn = perf_measure(y_test.tolist(), y_pred.tolist())
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "Ensemble Learning")
print_metrics(metrics)
```

```
Accuracy: 0.81
Precision: 0.71
Recall | Sensitivity: 0.54
Specificity: 0.91
Negative predictive value: 0.84
```

Neural Networks

In [11]:
```python
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

X = train_X.iloc[:,:].values
y = pd.DataFrame(train_y).iloc[:,:].values

X = StandardScaler().fit_transform(X)
y = OneHotEncoder().fit_transform(y).toarray()

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

model = Sequential()
model.add(Dense(16, input_dim=19, activation = "sigmoid"))
model.add(Dense(12, activation="sigmoid"))
model.add(Dense(2, activation="softmax"))

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

model.fit(X_train, y_train, epochs=50, batch_size=64)

y_pred = np.around(model.predict(X_test))

def compiler(list):
    if list[0] == 1 and list[1] == 0:
        return 1
    elif list[0] == 0 and list[1] == 1:
        return 0
    elif list[0] == list[1]:
        return np.random.randint(0, 2)

y_test = list(map(compiler, y_test))
y_pred = list(map(compiler, y_pred))

tp, fp, tn, fn = perf_measure(y_test, y_pred)
metrics = return_metrics(tp, fp, tn, fn)
update_metrics(metrics, "Neural Networks")
print_metrics(metrics)
```

```
Epoch 1/50
63/63 [==============================] - 1s 2ms/step - loss: 0.5748 - accuracy: 0.7368
Epoch 2/50
63/63 [==============================] - 0s 2ms/step - loss: 0.5279 - accuracy: 0.7409
Epoch 3/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4896 - accuracy: 0.7370
Epoch 4/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4520 - accuracy: 0.7786
Epoch 5/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4402 - accuracy: 0.7874
Epoch 6/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4290 - accuracy: 0.7959
Epoch 7/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4266 - accuracy: 0.7922
Epoch 8/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4150 - accuracy: 0.8070
Epoch 9/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4091 - accuracy: 0.8130
Epoch 10/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4107 - accuracy: 0.8089
Epoch 11/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4357 - accuracy: 0.7999
Epoch 12/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4162 - accuracy: 0.8033
Epoch 13/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4119 - accuracy: 0.8009
Epoch 14/50
63/63 [==============================] - 0s 6ms/step - loss: 0.4210 - accuracy: 0.8059
Epoch 15/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4184 - accuracy: 0.8152
Epoch 16/50
63/63 [==============================] - 0s 5ms/step - loss: 0.4261 - accuracy: 0.8043
Epoch 17/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4220 - accuracy: 0.8030
Epoch 18/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4112 - accuracy: 0.8065
Epoch 19/50
63/63 [==============================] - 0s 5ms/step - loss: 0.4058 - accuracy: 0.8117
Epoch 20/50
63/63 [==============================] - 0s 5ms/step - loss: 0.4001 - accuracy: 0.8168
Epoch 21/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4172 - accuracy: 0.8055
Epoch 22/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4196 - accuracy: 0.8034
Epoch 23/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4152 - accuracy: 0.8112
Epoch 24/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4044 - accuracy: 0.8117
Epoch 25/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4126 - accuracy: 0.8059
Epoch 26/50
63/63 [==============================] - 0s 4ms/step - loss: 0.3935 - accuracy: 0.8197
Epoch 27/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4046 - accuracy: 0.8135
Epoch 28/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4179 - accuracy: 0.8098
Epoch 29/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4225 - accuracy: 0.7963
Epoch 30/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4242 - accuracy: 0.8033
Epoch 31/50
63/63 [==============================] - 0s 4ms/step - loss: 0.3964 - accuracy: 0.8180
Epoch 32/50
63/63 [==============================] - 0s 3ms/step - loss: 0.3951 - accuracy: 0.8251
Epoch 33/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4101 - accuracy: 0.8062
Epoch 34/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4176 - accuracy: 0.8050
Epoch 35/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4067 - accuracy: 0.8142
Epoch 36/50
63/63 [==============================] - 0s 5ms/step - loss: 0.4099 - accuracy: 0.8046
Epoch 37/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4128 - accuracy: 0.8033
Epoch 38/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4204 - accuracy: 0.8037
Epoch 39/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4139 - accuracy: 0.8065
Epoch 40/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4173 - accuracy: 0.8037
Epoch 41/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4068 - accuracy: 0.8092
Epoch 42/50
63/63 [==============================] - 0s 5ms/step - loss: 0.4007 - accuracy: 0.8113
Epoch 43/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4001 - accuracy: 0.8139
Epoch 44/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4103 - accuracy: 0.8067
Epoch 45/50
63/63 [==============================] - 0s 2ms/step - loss: 0.3888 - accuracy: 0.8195
Epoch 46/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4011 - accuracy: 0.8156
Epoch 47/50
63/63 [==============================] - 0s 3ms/step - loss: 0.4013 - accuracy: 0.8159
Epoch 48/50
63/63 [==============================] - 0s 4ms/step - loss: 0.4064 - accuracy: 0.8091
Epoch 49/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4053 - accuracy: 0.8110
Epoch 50/50
63/63 [==============================] - 0s 2ms/step - loss: 0.4084 - accuracy: 0.8103
Accuracy: 0.81
Precision: 0.84
Recall | Sensitivity: 0.91
Specificity: 0.55
Negative predictive value: 0.7
```

The table of metrics

In [12]:
```python
table = pd.DataFrame(coefs).transpose()
table
```

Out[12]:

	Accuracy	Precision	Recall | Sensitivity	Specificity	Negative predictive value
Logistic Regression	0.80	0.68	0.56	0.90	0.84
Decision Tree	0.79	0.71	0.43	0.93	0.81
SVM	0.59	0.23	0.21	0.73	0.70
KNN	0.77	0.68	0.32	0.94	0.78
Random Forest	0.81	0.73	0.50	0.93	0.83
Ensemble Learning	0.81	0.71	0.54	0.91	0.84
Neural Networks	0.81	0.84	0.91	0.55	0.70

As we can see Neural Networks give the best results in 3 most important metrics, so we will use them for our final evaluation.

In [13]:
```python
X_train = train_X.iloc[:,:].values
y_train = pd.DataFrame(train_y).iloc[:,:].values
X_test = test_X.iloc[:,:].values
y_test = pd.DataFrame(test_y).iloc[:,:].values

X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().fit_transform(X_test)
y_train = OneHotEncoder().fit_transform(y_train).toarray()
y_test = OneHotEncoder().fit_transform(y_test).toarray()

model = Sequential()
model.add(Dense(16, input_dim=19, activation = "sigmoid"))
model.add(Dense(12, activation="sigmoid"))
model.add(Dense(2, activation="softmax"))

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

model.fit(X_train, y_train, epochs=50, batch_size=64)

y_pred = np.around(model.predict(X_test))

def compiler(list):
    if list[0] == 1 and list[1] == 0:
        return 1
    elif list[0] == 0 and list[1] == 1:
        return 0
    elif list[0] == list[1]:
        return np.random.randint(0, 2)

y_test = list(map(compiler, y_test))
y_pred = list(map(compiler, y_pred))

tp, fp, tn, fn = perf_measure(y_test, y_pred)
metrics = return_metrics(tp, fp, tn, fn)
print_metrics(metrics)
```

```
Epoch 1/50
79/79 [==============================] - 1s 2ms/step - loss: 0.5865 - accuracy: 0.7289
Epoch 2/50
79/79 [==============================] - 0s 2ms/step - loss: 0.5389 - accuracy: 0.7275
Epoch 3/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4787 - accuracy: 0.7509
Epoch 4/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4491 - accuracy: 0.7830
Epoch 5/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4340 - accuracy: 0.8032
Epoch 6/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4211 - accuracy: 0.8135
Epoch 7/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4316 - accuracy: 0.7990
Epoch 8/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4070 - accuracy: 0.8148
Epoch 9/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4093 - accuracy: 0.8176
Epoch 10/50
79/79 [==============================] - 0s 4ms/step - loss: 0.4252 - accuracy: 0.8041
Epoch 11/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4244 - accuracy: 0.8087
Epoch 12/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4314 - accuracy: 0.8029
Epoch 13/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4253 - accuracy: 0.8074
Epoch 14/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4277 - accuracy: 0.8023
Epoch 15/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4119 - accuracy: 0.8113
Epoch 16/50
79/79 [==============================] - 0s 4ms/step - loss: 0.4043 - accuracy: 0.8240
Epoch 17/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4033 - accuracy: 0.8153
Epoch 18/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4259 - accuracy: 0.8070
Epoch 19/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4164 - accuracy: 0.8150
Epoch 20/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4275 - accuracy: 0.7990
Epoch 21/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4111 - accuracy: 0.8081
Epoch 22/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4191 - accuracy: 0.8044
Epoch 23/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4143 - accuracy: 0.8117
Epoch 24/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4147 - accuracy: 0.8095
Epoch 25/50
79/79 [==============================] - 1s 6ms/step - loss: 0.4106 - accuracy: 0.8102
Epoch 26/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4162 - accuracy: 0.8104
Epoch 27/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4159 - accuracy: 0.8040
Epoch 28/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4035 - accuracy: 0.8191
Epoch 29/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4107 - accuracy: 0.8101
Epoch 30/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4146 - accuracy: 0.8048
Epoch 31/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4161 - accuracy: 0.7974
Epoch 32/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4092 - accuracy: 0.8138
Epoch 33/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4108 - accuracy: 0.8064
Epoch 34/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4231 - accuracy: 0.8008
Epoch 35/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4146 - accuracy: 0.8045
Epoch 36/50
79/79 [==============================] - 0s 4ms/step - loss: 0.4254 - accuracy: 0.8003
Epoch 37/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4072 - accuracy: 0.8156
Epoch 38/50
79/79 [==============================] - 0s 3ms/step - loss: 0.3980 - accuracy: 0.8096
Epoch 39/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4314 - accuracy: 0.7972
Epoch 40/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4048 - accuracy: 0.8145
Epoch 41/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4111 - accuracy: 0.8121
Epoch 42/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4179 - accuracy: 0.8034
Epoch 43/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4162 - accuracy: 0.8078
Epoch 44/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4187 - accuracy: 0.8054
Epoch 45/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4063 - accuracy: 0.8105
Epoch 46/50
79/79 [==============================] - 0s 3ms/step - loss: 0.4143 - accuracy: 0.8056
Epoch 47/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4202 - accuracy: 0.7993
Epoch 48/50
79/79 [==============================] - 0s 2ms/step - loss: 0.4132 - accuracy: 0.8086
Epoch 49/50
79/79 [==============================] - 0s 3ms/step - loss: 0.3959 - accuracy: 0.8206
Epoch 50/50
79/79 [==============================] - 0s 4ms/step - loss: 0.3987 - accuracy: 0.8212
Accuracy: 0.8
Precision: 0.86
Recall | Sensitivity: 0.88
Specificity: 0.56
Negative predictive value: 0.6
```