```
# Title: Healthcare Patient Recovery patient

# Objective:
    # Hospital wants to predict will recover after treatment
    # it will help the hospital in several ways such as,
        # Prioritize high risk patient
        # Optimise ICU resources
        # Improve treatment planning

# Targated Varible: Recoverd

# This is Binary class prediction hence, we are going to use Neural Network
                                                                                Loading...
# Dataset :
    # Records - 15000
    # Features - 15
    # DataType - Numerical
    # Complexity - Non-Linear
```

```
# Importing Libraries in needs

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

```
df = pd.read_csv('/content/drive/MyDrive/Reinforcement Learning/Deep Learning/Healthcare Patient Recovery/Healthcare Patient Re
df.head(10)
```

	age	bmi	blood_pressure	oxygen_level	heart_rate	sugar_level	cholesterol	days_hospitalized	previous_conditions	sm
0	69	22.277116	132.209294	91.213847	75.825529	128.241068	231.792260	15	4	
1	32	30.279420	124.766318	94.290118	83.285925	95.673726	174.602623	21	1	
2	78	27.379115	138.648126	94.847213	66.696213	85.198259	220.482972	6	3	
3	38	18.154561	136.346900	90.990286	58.604568	147.488850	205.092398	8	1	
4	41	29.315485	134.886038	95.296480	83.228853	110.394896	172.265873	1	1	
5	20	24.545868	132.861579	96.484114	84.523055	78.531544	184.996640	21	1	
6	39	34.667504	129.990652	97.509037	83.434125	124.322359	267.069997	21	3	
7	70	25.024891	109.040301	90.506089	83.611564	134.725392	178.890823	10	0	
8	19	24.740540	135.839181	98.039703	75.894949	107.001019	225.208817	8	0	
9	47	34.665344	140.979344	88.770377	76.425310	69.705362	205.533862	4	3	

Next steps: New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  15000 non-null  int64
 1   bmi                  15000 non-null  float64
 2   blood_pressure       15000 non-null  float64
 3   oxygen_level         15000 non-null  float64
 4   heart_rate           15000 non-null  float64
 5   sugar_level          15000 non-null  float64
 6   cholesterol          15000 non-null  float64
 7   days_hospitalized    15000 non-null  int64
```

```
 8   previous_conditions   15000 non-null  int64
 9   smoker                15000 non-null  int64
 10  alcohol_use           15000 non-null  int64
 11  treatment_intensity   15000 non-null  int64
 12  medication_adherence  15000 non-null  int64
 13  followup_visits       15000 non-null  int64
 14  stress_level          15000 non-null  int64
 15  recovered             15000 non-null  int64
dtypes: float64(6), int64(10)
memory usage: 1.8 MB
```

```
df.isnull().sum()
```

	0
age	0
bmi	0
blood_pressure	0
oxygen_level	0
heart_rate	0
sugar_level	0
cholesterol	0
days_hospitalized	0
previous_conditions	0
smoker	0
alcohol_use	0
treatment_intensity	0
medication_adherence	0
followup_visits	0
stress_level	0
recovered	0

dtype: int64

Loading...

```
# Independent (X) and Dependent Variable

x=df.drop('recovered', axis=1)
y=df['recovered']
```

```
# Spliting Dataset into Train(80%) and Test Data(20%)

x_train,x_test, y_train, y_test= train_test_split(x,y, test_size=0.20, random_state=42)
print('Data Splited Succesfully')
```

```
Data Splited Succesfully
```

```
# Scaling the Dataset

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test= scaler.transform(x_test)

print('Data Scaled')
```

```
Data Scaled
```

```
# BUilding a Neural Network

model = Sequential()
model.add(Dense(128, activation='relu', input_shape= (x_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Model Compiling

model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Model Evaluation

model_hist= model.fit(x_train, y_train, epochs=25, batch_size=30, validation_split=0.20)
```

```
Epoch 1/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9555 - loss: 0.1544 - val_accuracy: 0.9663 - val_loss: 0.1133
Epoch 2/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9648 - loss: 0.1143 - val_accuracy: 0.9663 - val_loss: 0.1270
Epoch 3/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9644 - loss: 0.1163 - val_accuracy: 0.9663 - val_loss: 0.1118
Epoch 4/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9609 - loss: 0.1110 - val_accuracy: 0.9663 - val_loss: 0.1104
Epoch 5/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9645 - loss: 0.1029 - val_accuracy: 0.9663 - val_loss: 0.1146
Epoch 6/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9669 - loss: 0.0993 - val_accuracy: 0.9663 - val_loss: 0.1181
Epoch 7/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9654 - loss: 0.1039 - val_accuracy: 0.9663 - val_loss: 0.1143
Epoch 8/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9632 - loss: 0.1071 - val_accuracy: 0.9663 - val_loss: 0.1158
Epoch 9/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9656 - loss: 0.0988 - val_accuracy: 0.9663 - val_loss: 0.1262
Epoch 10/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9649 - loss: 0.1027 - val_accuracy: 0.9663 - val_loss: 0.1181
Epoch 11/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9643 - loss: 0.1058 - val_accuracy: 0.9663 - val_loss: 0.1154
Epoch 12/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9651 - loss: 0.0978 - val_accuracy: 0.9663 - val_loss: 0.1256
Epoch 13/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9663 - loss: 0.1070 - val_accuracy: 0.9663 - val_loss: 0.1268
Epoch 14/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9629 - loss: 0.1019 - val_accuracy: 0.9663 - val_loss: 0.1268
Epoch 15/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9653 - loss: 0.0933 - val_accuracy: 0.9663 - val_loss: 0.1241
Epoch 16/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9633 - loss: 0.0926 - val_accuracy: 0.9663 - val_loss: 0.1146
Epoch 17/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9669 - loss: 0.0837 - val_accuracy: 0.9663 - val_loss: 0.1338
Epoch 18/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9644 - loss: 0.0842 - val_accuracy: 0.9663 - val_loss: 0.1348
Epoch 19/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9620 - loss: 0.0833 - val_accuracy: 0.9663 - val_loss: 0.1310
Epoch 20/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9669 - loss: 0.0792 - val_accuracy: 0.9663 - val_loss: 0.1443
Epoch 21/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9625 - loss: 0.0795 - val_accuracy: 0.9663 - val_loss: 0.1342
Epoch 22/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9637 - loss: 0.0783 - val_accuracy: 0.9663 - val_loss: 0.1496
Epoch 23/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9665 - loss: 0.0699 - val_accuracy: 0.9663 - val_loss: 0.1646
Epoch 24/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9658 - loss: 0.0709 - val_accuracy: 0.9521 - val_loss: 0.2738
Epoch 25/25
320/320 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9612 - loss: 0.0769 - val_accuracy: 0.9663 - val_loss: 0.1846
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f'\nTest Accuracy',{round(accuracy,2)})
```

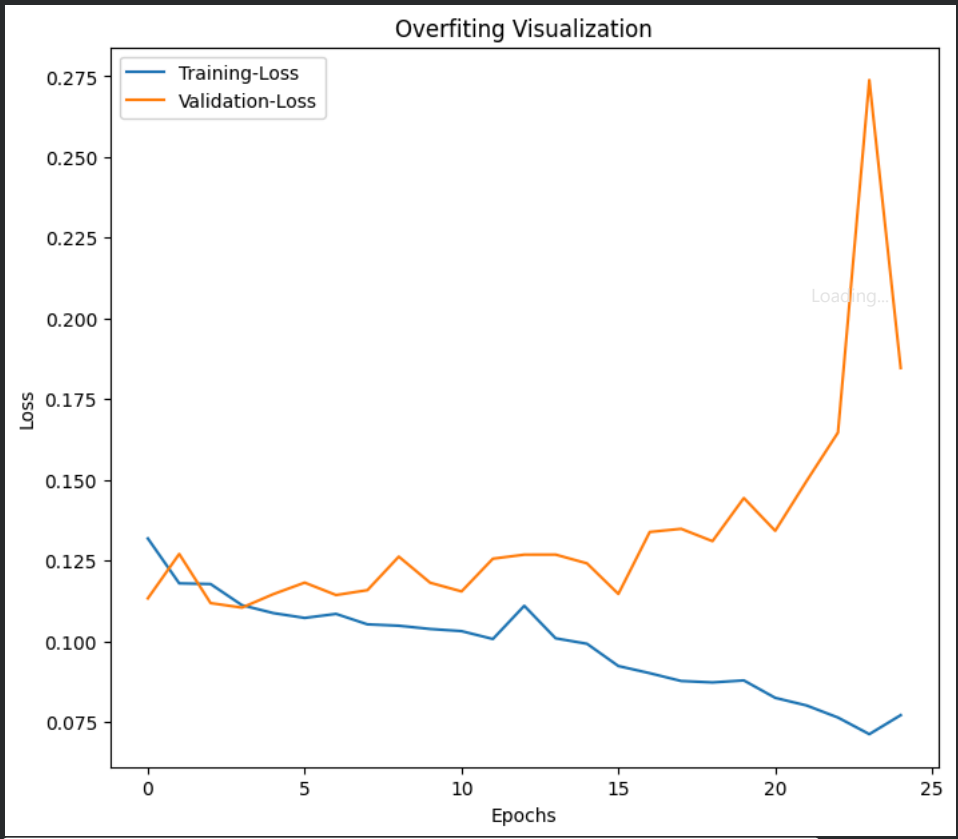
```
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9653 - loss: 0.1555

Test Accuracy {0.96}
```

```
# Plot to see Overfiting

plt.figure(figsize=(8,7))
plt.plot(model_hist.history['loss'], label='Training-Loss')
plt.plot(model_hist.history['val_loss'], label='Validation-Loss')
plt.title('Overfiting Visualization')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()
```




```
# Model is overfiting a lot

# Dropout
# L2 Regularizer
# Early Stoping
# Proper Learning Rate

from tensorflow.keras.layers import Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping

reg_model = Sequential()
reg_model.add(Dense(128, activation='relu',kernel_regularizer=l2(0.001), input_shape= (x_train.shape[1],)))
reg_model.add(Dropout(0.3))

reg_model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
reg_model.add(Dropout(0.3))

reg_model.add(Dense(32, activation='relu',kernel_regularizer=l2(0.001)))
reg_model.add(Dropout(0.3))

reg_model.add(Dense(16, activation='relu',kernel_regularizer=l2(0.001)))
reg_model.add(Dropout(0.3))

reg_model.add(Dense(8, activation='relu',kernel_regularizer=l2(0.001)))
reg_model.add(Dropout(0.3))

reg_model.add(Dense(1, activation='sigmoid'))
```

```
reg_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Early Stopping

early_stop= EarlyStopping(monitor = 'val_loss', patience=5, restore_best_weights = True)
```

```
reg_hist= reg_model.fit(x_train, y_train,
                        epochs=100, batch_size=32,
                        validation_split=0.20,
```

```
                    callbacks=[early_stop])
```

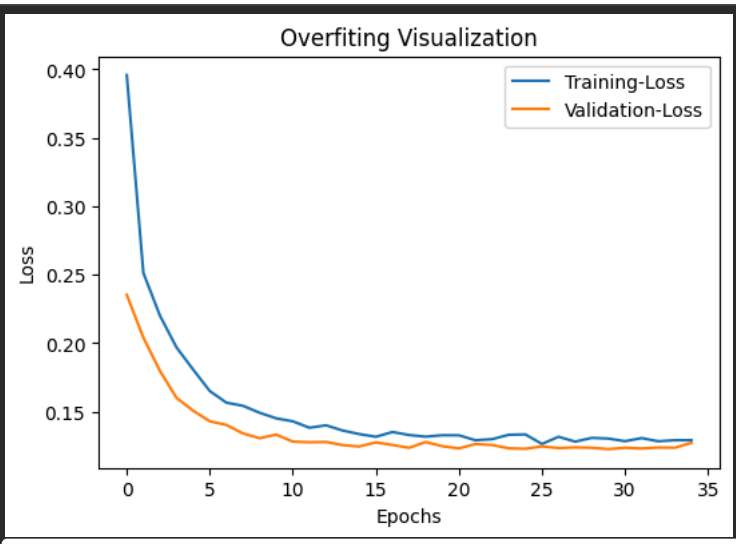
```
Epoch 1/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8711 - loss: 0.5110 - val_accuracy: 0.9663 - val_loss: 0.2353
Epoch 2/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9619 - loss: 0.2641 - val_accuracy: 0.9663 - val_loss: 0.2041
Epoch 3/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9620 - loss: 0.2343 - val_accuracy: 0.9663 - val_loss: 0.1797
Epoch 4/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9654 - loss: 0.1913 - val_accuracy: 0.9663 - val_loss: 0.1601
Epoch 5/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9644 - loss: 0.1795 - val_accuracy: 0.9663 - val_loss: 0.1507
Epoch 6/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9654 - loss: 0.1610 - val_accuracy: 0.9663 - val_loss: 0.1430
Epoch 7/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9639 - loss: 0.1568 - val_accuracy: 0.9663 - val_loss: 0.1403
Epoch 8/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9674 - loss: 0.1499 - val_accuracy: 0.9663 - val_loss: 0.1341
Epoch 9/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9614 - loss: 0.1496 - val_accuracy: 0.9663 - val_loss: 0.1306
Epoch 10/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9619 - loss: 0.1517 - val_accuracy: 0.9663 - val_loss: 0.1333
Epoch 11/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9625 - loss: 0.1379 - val_accuracy: 0.9663 - val_loss: 0.1281
Epoch 12/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9633 - loss: 0.1406 - val_accuracy: 0.9663 - val_loss: 0.1277
Epoch 13/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9648 - loss: 0.1373 - val_accuracy: 0.9663 - val_loss: 0.1279
Epoch 14/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9651 - loss: 0.1354 - val_accuracy: 0.9663 - val_loss: 0.1256
Epoch 15/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9641 - loss: 0.1317 - val_accuracy: 0.9663 - val_loss: 0.1246
Epoch 16/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9678 - loss: 0.1232 - val_accuracy: 0.9663 - val_loss: 0.1276
Epoch 17/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9655 - loss: 0.1308 - val_accuracy: 0.9663 - val_loss: 0.1257
Epoch 18/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9664 - loss: 0.1228 - val_accuracy: 0.9663 - val_loss: 0.1237
Epoch 19/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9642 - loss: 0.1333 - val_accuracy: 0.9663 - val_loss: 0.1278
Epoch 20/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9639 - loss: 0.1328 - val_accuracy: 0.9663 - val_loss: 0.1248
Epoch 21/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9623 - loss: 0.1364 - val_accuracy: 0.9663 - val_loss: 0.1232
Epoch 22/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9647 - loss: 0.1294 - val_accuracy: 0.9663 - val_loss: 0.1264
Epoch 23/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9633 - loss: 0.1301 - val_accuracy: 0.9663 - val_loss: 0.1257
Epoch 24/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9632 - loss: 0.1382 - val_accuracy: 0.9663 - val_loss: 0.1233
Epoch 25/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9642 - loss: 0.1314 - val_accuracy: 0.9663 - val_loss: 0.1229
Epoch 26/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9649 - loss: 0.1251 - val_accuracy: 0.9663 - val_loss: 0.1247
Epoch 27/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9635 - loss: 0.1333 - val_accuracy: 0.9663 - val_loss: 0.1234
Epoch 28/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9622 - loss: 0.1277 - val_accuracy: 0.9663 - val_loss: 0.1239
Epoch 29/100
300/300 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9642 - loss: 0.1309 - val_accuracy: 0.9663 - val_loss: 0.1237
```

```
loss, accuracy = reg_model.evaluate(x_test, y_test)
print(f'Test Accuracy:',{round(accuracy,2)})
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9653 - loss: 0.1209
Test Accuracy: {0.96}
```

```
# Plot to see Overfiting

plt.figure(figsize=(6,4))
plt.plot(reg_hist.history['loss'], label='Training-Loss')
plt.plot(reg_hist.history['val_loss'], label='Validation-Loss')
plt.title('Overfiting Visualization')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Overfiting Visualization
Training-Loss
Validation-Loss
Loss
Epochs
0.40
0.35
0.30
0.25
0.20
0.15
0
5
10
15
20
25
30
35

Loading...