Herzkrankheitsvorhersage

April 22, 2024

Inhaltsverzeichnis

1 Motivation

Herzkrankheiten stellen ein bedeutendes und weit verbreitetes globales Gesundheitsproblem dar, das Millionen von Menschen auf der ganzen Welt betrifft. Ihre schwerwiegenden Folgen machen sie zur Haupttodesursache nicht nur in entwickelten Nationen, sondern auch in zahlreichen Entwicklungsländern. Mit seinem Spektrum an Symptomen und Komplikationen, einschließlich Faktoren wie Brustschmerzen und anderen in unserer Datenbank dokumentierten Faktoren, wird die schnelle und effektive Behandlung von Herzkrankheiten von größter Bedeutung. Die zentralen Ziele, die wir uns gesetzt haben, beinhalten die frühzeitige Vorhersage, Diagnose und anschließende Behandlung von Herzkrankheiten. Mit diesem Bestreben zielen wir darauf ab, Gesundheitsfachleuten und -systemen die Fähigkeit zu geben, das Vorhandensein von Herzkrankheiten bei Patienten vorauszusehen. Dieses Projekt nimmt einen besonderen Stellenwert im Bereich der Data Science ein und genießt innerhalb der Kaggle-Gemeinschaft einen ausgezeichneten Status als renommiertes Projekt.

2 Bibliotheken

Die folgenden Zellen werden verwendet, um externe Python-Bibliotheken oder Module einzubinden, die wir in diesem Notebook verwenden müssen. Durch die Nutzung von Python-Paketen, wie den folgenden aufgelisteten Paketen, können Analysten effizient Datenübersichten durchführen und mit einer umfassenden Datenanalyse starten, um wertvolles Wissen aus komplexen Datensätzen zu extrahieren

2.1 Verarbeitung

PCA (z.B. Datenreduktion: Auswahl, Aggregation oder Zusammenfassung der Daten zur Reduzierung ihrer Größe oder Komplexität) und Vorverarbeitung (z.B. Datennormalisierung: Skalierung oder Standardisierung der Daten, um sie über verschiedene Merkmale oder Proben hinweg vergleichbar oder kompatibel zu machen) sind beide mit der Datenvorbereitungsphase der Datenwissenschaft verbunden, die den Prozess der Umwandlung von Rohdaten in ein Format darstellt, das für die Analyse und Modellierung verwendet werden kann.

pandas: Dies ist ein Python-Paket, das schnelle und einfach zu verwendende Datenstrukturen und Analysetools bietet, wie z.B. DataFrame und Series.

scikit-learn: Dies ist ein weiteres Python-Paket, das verschiedene Tools für die Datenanalyse (z.B. explorative Datenanalysen (EDA)) aber auch für das maschinelle Lernen (siehe den nächsten Abschnitt) bereitstellt, einschließlich PCA und anderen Vorverarbeitungsmethoden.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np
from collections import Counter
import random
import re
```

2.2 Modellierung

Die Modellierungsphase beinhaltet das Erstellen, Trainieren und Bewerten eines maschinellen Lernmodells mit den Daten, die in den vorherigen Phasen gesammelt, annotiert und aufbereitet wurden. Es gibt viele verschiedene Arten von Klassifikationsmodellen, wie logistische Regression, Entscheidungsbäume, Support-Vektor-Maschinen, ***neuronale Netzwerke***, usw.

scikit-learn: ist ein Python-Paket, das verschiedene Werkzeuge für maschinelles Lernen bietet, wie Klassifikation, Regression, Clustering, Merkmalsauswahl, Modellbewertung und Optimierung von Modell-Hyperparametern. Scikit-learn ist auch auf der Basis von *NumPy* und *Scipy* aufgebaut (die tatsächlich sowohl für die Verarbeitung als auch für die Modellierung in der Datenwissenschaft verwendet werden) und folgt einer konsistenten und einfachen Schnittstelle für das Erstellen und Verwenden von maschinellen Lernmodellen.

Keras st ein beliebtes Python-Paket, das eine High-Level-API für das Erstellen und Verwenden von Deep-Learning-Modellen bietet. Deep Learning ist ein Zweig des maschinellen Lernens, der mehrere Schichten künstlicher neuronaler Netzwerke verwendet, um aus komplexen und hochdimensionalen Daten zu lernen. Keras kann mit verschiedenen Backends arbeiten, wie *TensorFlow*

oder Theano, die Frameworks sind, die Low-Level-Operationen für das Bauen und Ausführen von Deep-Learning-Modellen anbieten. Keras kann bei der Datenrepräsentation, dem Datenlernen und der Datengenerierung helfen.

```
# from scipy import interp # This can be replaced newly by 'interp' from numpy.
from tqdm import tqdm
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.utils import to_categorical
```

2.3 Visualisierung

Die Datenvisualisierung ist ein wichtiger Aspekt der Datenanalyse, da sie dazu beitragen kann, Muster, Trends, Ausreißer und Beziehungen zu enthüllen, die aus numerischen oder textuellen Daten allein nicht offensichtlich erscheinen könnten. Es gibt viele Tools und Bibliotheken in Python für die Erstellung verschiedener Arten von Diagrammen, wie zum Beispiel **matplotlib** , **seaborn** (Es handelt sich um eine auf matplotlib basierende Visualisierungsbibliothek), und **plotly** (express).

```
# Import all necessary packages
from pathlib import Path
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sbn
import seaborn.objects as so

sbn.set_style('whitegrid')
sbn.set(font_scale=1.0, font='sans-serif')
#import seaborn_image as isns
%matplotlib inline
plt.rcParams["figure.figsize"] = [8, 6]
```

```
font = {'family': 'serif',
        'color': 'darkblue',
        'weight': 'normal',
        'size': 18,
        }
kwargs = {'edgecolor': "black",  # for edge color
          'linewidth': 2,  # line width of spot
          }
from IPython.display import Markdown,display
from termcolor import colored
#import ast
```

3 Datenübersicht

Eine gründliche Übersicht über die Daten ist der Grundstein jedes erfolgreichen Datenanalyseprojekts. Sie ermöglicht es Analysten, datenbezogene Herausforderungen zu identifizieren, die Charakteristiken des Datensatzes zu verstehen und wichtige Muster und Erkenntnisse aufzudecken.

```
df_heart_disease = pd.read_csv('../da_proj/datasets/heart_disease.csv', sep=',',
                               encoding='utf-8-sig')
print(colored('The Head of Dataset:', attrs=['bold'], color='green'))
display(df_heart_disease.head(3))

print(colored('The Tail of Dataset:', attrs=['bold'], color='red'))
display( df_heart_disease.tail(3))

print(colored('The Dataset (short) Overview:', attrs=['bold'], color='blue'))
display(df_heart_disease.info())
df_test = df_heart_disease.head()
```

The Head of Dataset:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1

The Tail of Dataset:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
The Dataset (short) Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB

None
```

3.1 Datenattribute

Die oben genannten Header sind die Merkmale oder Attribute eines Datensatzes, die häufig für die Vorhersage von Herzkrankheiten verwendet werden. Dieser Datensatz wird als Herzkrankheits UCI Datensatz bezeichnet und wie oben gezeigt enthält er ***303*** Patientenakten mit jeweils 14 Merkmalen. Die Merkmale sind:

- **Age:** das Alter des Patienten in Jahren
- **Sex:** das Geschlecht des Patienten (1 = männlich, 0 = weiblich)
- **Cp:** die Art der Brustschmerzen des Patienten (0 = typische Angina, 1 = atypische Angina pectoris, 2 = nicht-anginöser Schmerz, 3 = asymptomatisch)
- **Trestbps:** der Ruheblutdruck des Patienten in mm Hg
- **Chol:** der Serumcholesterinspiegel des Patienten in mg/dl
- **Fbs:** der nüchterne Blutzuckerspiegel des Patienten (> 120 mg/dl, 1 = wahr, 0 = falsch)
- **Restecg:** die Ruhe-EKG-Ergebnisse des Patienten (0 = normal, 1 = ST-T-Wellen-Anomalie vorhanden, 2 = wahrscheinliche oder definitive linksventrikuläre Hypertrophie)
- **Thalach:** die maximale Herzfrequenz, die der Patient erreicht hat

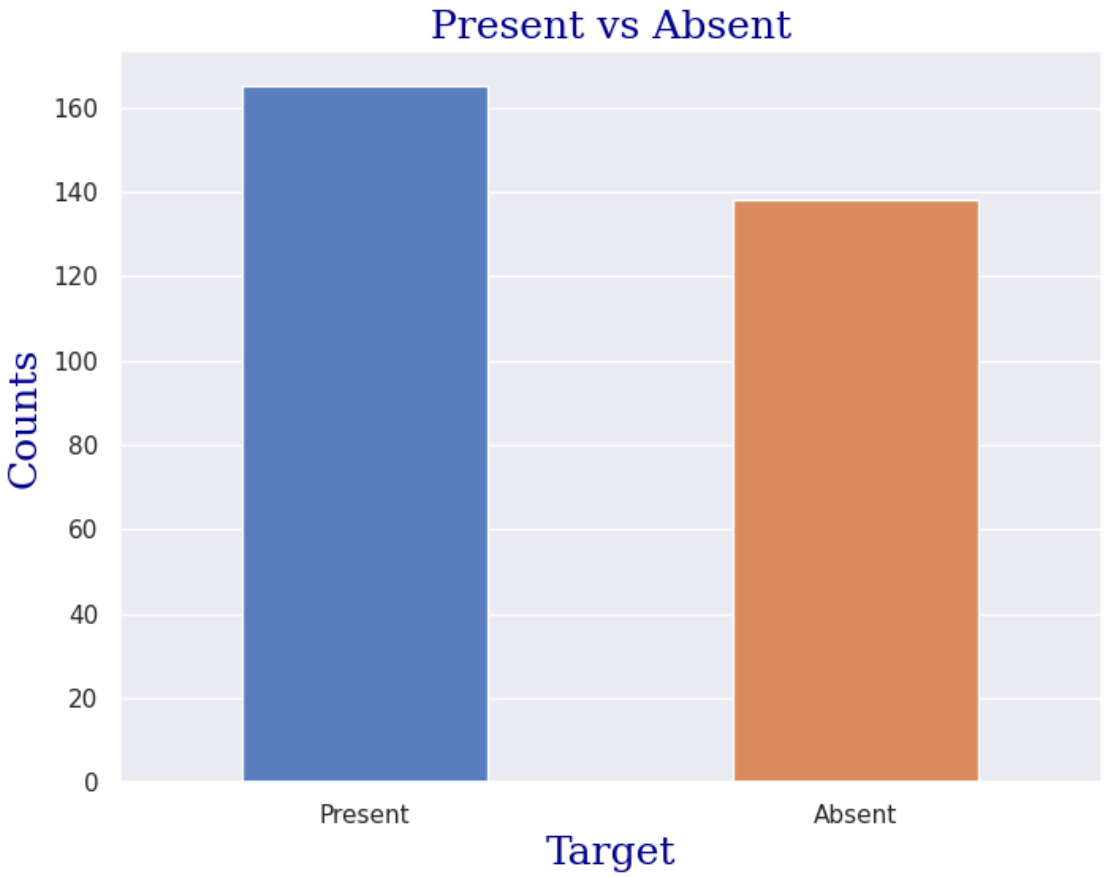
- **Exang:** die durch Bewegung induzierte Angina des Patienten (1 = ja, 0 = nein)
- **Oldpeak:** die ST-Senkung, die durch Bewegung im Vergleich zur Ruhe induziert wird
- **Slope:** die Steigung des ST-Segments im Höhepunkt der Bewegung (1 = ansteigend, 2 = flach, 3 = abfallend)
- **Ca:** die Anzahl der durch Fluoroskopie gefärbten Hauptgefäße (0-3)
- **Thal:** der Thalassämie-Status des Patienten (3 = normal, 6 = fester Defekt, 7 = reversibler Defekt)
- **Target:** das Vorhandensein von Herzkrankheiten beim Patienten (1 = ja, 0 = nein)

3.2 Anwesenheit und Abwesenheit

Dieser Abschnitt bezieht sich auf den Vergleich der beiden möglichen Ergebnisse, wie kurz im letzten Abschnitt erwähnt. Die Zielvariable ist ein entscheidender Schwerpunkt der Analyse, um das Vorhandensein oder Fehlen von Herzkrankheiten bei den untersuchten Patienten zu verstehen.

```
"""
Comparison between patients with heart disease and without that according their␣
 ↪age
present:  with heart disease
absent:  without heart disease
"""
# Firstly, we condense the dataframe so it counts the number of times each
# target appears
present_absent = df_heart_disease['target'].value_counts().rename_axis
('target').to_frame('counts').reset_index()
present_absent['target'] = present_absent['target'].replace({1:'Present', 0:
 ↪'Absent'})
# display(present_absent)

# Plot a bar chart of the counts
fg1 = sbn.barplot(data= present_absent, x='target', hue='target',
                  y='counts', palette='muted', width=0.5)
fg1.set_title('Present vs Absent', fontdict=font)
fg1.set_xlabel('Target', fontdict=font)
fg1.set_ylabel('Counts', fontdict=font)
```

```
Text(0, 0.5, 'Counts')
```

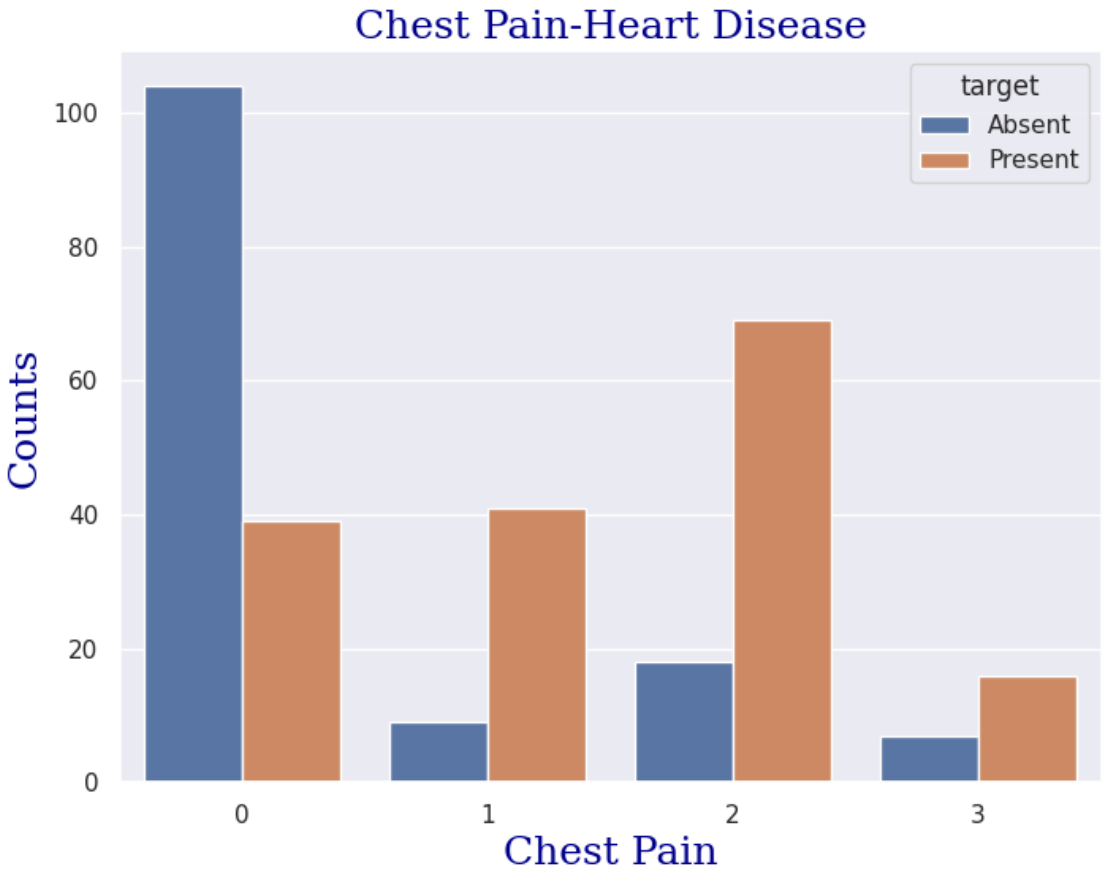


Als Nächstes wäre es nützlich für uns, Einblick zu gewinnen, ob der Brustschmerz (en. chest pain (cp)) des Patienten eine wichtige Rolle beim Vorhandensein oder Fehlen von Herzkrankheiten spielt.

```
present_absent_cp = df_heart_disease[['target', 'cp']].value_counts().to_frame(
    'counts').reset_index()
present_absent_cp['target'] = present_absent_cp['target'
                                    ].replace({1:'Present', 0:'Absent'})
display(present_absent_cp)
#present_absent_age.plot()
fg2 = sbn.barplot(data=present_absent_cp, x='cp', y='counts',hue='target')
fg2.set_title('Chest Pain-Heart Disease', fontdict=font)
fg2.set_xlabel('Chest Pain', fontdict=font)
fg2.set_ylabel('Counts', fontdict=font)
```

	target	cp	counts
0	Absent	0	104
1	Present	2	69
2	Present	1	41
3	Present	0	39
4	Absent	2	18
5	Present	3	16
6	Absent	1	9
7	Absent	3	7

```
Text(0, 0.5, 'Counts')
```



Nur die Mehrheit der Patienten mit dem ersten Typ von Brustschmerzen, nämlich *typische Angina* hat keine Herzkrankheiten.

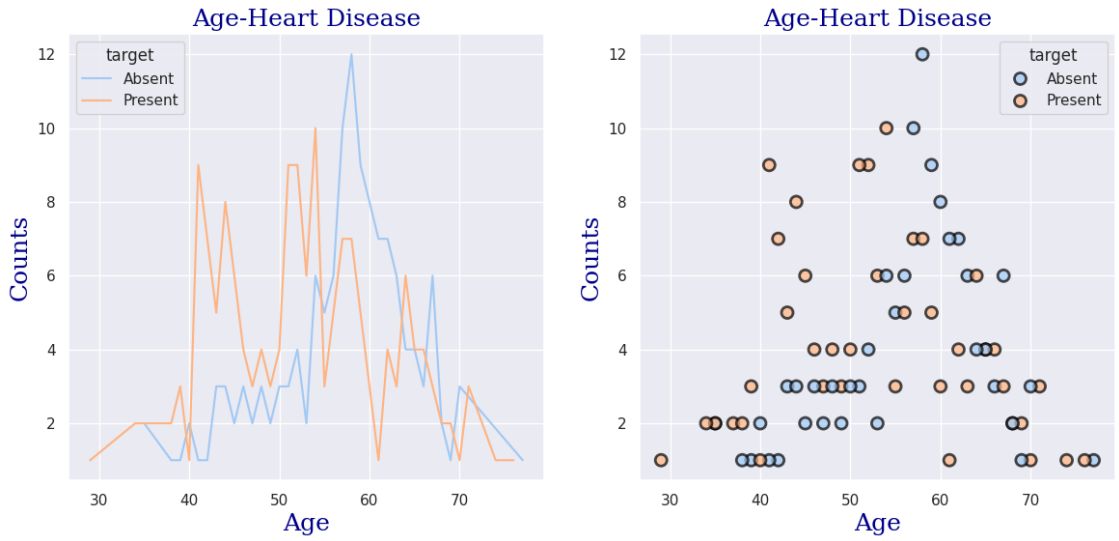
```
present_absent_age = df_heart_disease[['target', 'age']].value_counts().
  ↪to_frame(
    'counts').reset_index()
present_absent_age['target'] = present_absent_age['target'].replace(
    {1:'Present', 0:'Absent'})
display(present_absent_age)
#present_absent_age.plot()
ax1= plt.subplots(nrows=1, ncols=2, figsize=(14,6))[1]
fg3 = sbn.lineplot(data=present_absent_age, x='age', y='counts', hue='target',
                   ax=ax1[0], palette='pastel')
sbn.move_legend(ax1[0], 'upper left')
ax1[0].set_title('Age-Heart Disease', fontdict=font)
ax1[0].set_xlabel('Age', fontdict=font)
ax1[0].set_ylabel('Counts', fontdict=font)

fg4= sbn.scatterplot(data=present_absent_age, x='age', y='counts', ax=ax1[1],
```

```
                    hue= 'target', palette='pastel',
 s=75, alpha=0.7,**kwargs)
ax1[1].set_title('Age-Heart Disease', fontdict=font)
ax1[1].set_xlabel('Age', fontdict=font)
ax1[1].set_ylabel('Counts', fontdict=font)
```

	target	age	counts
0	Absent	58	12
1	Absent	57	10
2	Present	54	10
3	Absent	59	9
4	Present	52	9
...	...	...	...
70	Absent	69	1
71	Absent	77	1
72	Present	29	1
73	Absent	38	1
74	Present	76	1

75 rows × 3 columns

```
Text(0, 0.5, 'Counts')
```



4 Explorative Datenanalyse

Der letzte Abschnitt hat gezeigt, dass das Betrachten eines einzelnen Merkmals (in diesem Fall das Alter der Patienten) allein keine klaren oder abschließenden Informationen liefert, um das Vorhandensein oder Fehlen von Herzkrankheiten bei den Patienten genau vorherzusagen.

Der Grund für diese Einschränkung liegt darin, dass das Vorhandensein oder Fehlen von Herzkrankheiten von mehreren Faktoren beeinflusst wird (Multivariate Analyse (MVA)), und ein einzelnes Merkmal, wie das Alter, möglicherweise nicht die volle Komplexität der Beziehung erfassen kann. Um diese Einschränkung zu überwinden und die Vorhersagegenauigkeit zu verbessern, ist es notwendig, mit weiteren Schritten fortzufahren, wie wir in den beiden nächsten Abschnitten erklären können.

4.1 Beziehungsattribute

Um Verhältnisse zwischen Merkmalen zu verstehen, werden wir in den nächsten Schritten die Merkmale hinsichtlich ihrer Korrelationen visualisieren.

Zuerst teilen wir den Datensatz in die Merkmalsmatrix `X` und das Zielarray als entsprechende Klassenlabels `y`.

Danach wird die Merkmalsmatrix mit StandardScaler (aus scikit-learn) standardisiert, der die Merkmale auf einen Nullmittelwert und eine Einheitsvarianz skaliert.

```
X = df_heart_disease.iloc[:,0:13].values
y = df_heart_disease.iloc[:,13].values

X_sts = StandardScaler().fit_transform(X)
```

Datennormalisierung ist eine Vorverarbeitungstechnik, die die Daten auf einen standardisierten Bereich skaliert. Durch die Normalisierung der Daten stellen wir sicher, dass alle Merkmale die

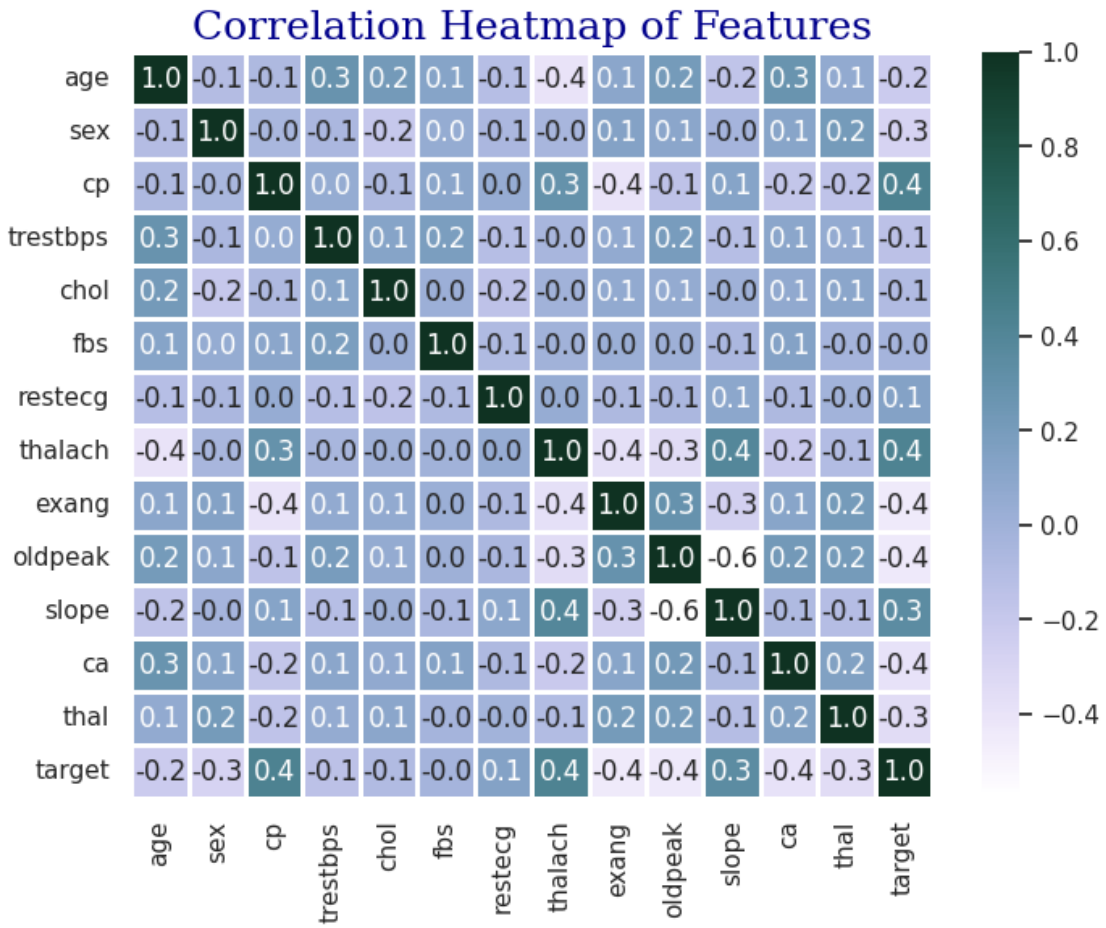
gleiche Skala haben, was besonders nützlich ist, wenn man es mit Merkmalen zu tun hat, die unterschiedliche Einheiten oder Bereiche haben. Dies kann die Leistung bestimmter Machine-Learning-Algorithmen verbessern, die empfindlich auf die Skala der Merkmale reagieren.

```
df_norm = pd.DataFrame(X_sts, index=df_heart_disease.index,␣
 ↪columns=df_heart_disease.columns[0:13])

# Insert a non-feature target column to the dataframe
df_norm['target'] = df_heart_disease['target']
df_norm.head(10)

X = df_norm.iloc[:,0:13].values
y = df_norm.iloc[:,13].values
```

Wir berechnen die Korrelationsmatrix für ein normalisiertes DataFrame und erstellen dann ein Heatmap, um die Korrelationen zwischen den Spalten visuell darzustellen. Die Heatmap zeigt die Korrelationswerte zwischen verschiedenen Merkmalen (Spalten) des DataFrames, wobei die x-Achse und die y-Achse mit den Spaltennamen des DataFrames beschriftet sind.

```
corr = df_norm.corr()
fg5 = sbn.heatmap(data=corr,xticklabels=corr.columns, yticklabels=corr.columns,␣
 ↪linewidths=.75,
 annot=True, fmt='.1f', cmap=sbn.cubehelix_palette(as_cmap=True, start=2,␣
 ↪light=1))
fg5.set_title('Correlation Heatmap of Features', fontdict=font)
```

```
Text(0.5, 1.0, 'Correlation Heatmap of Features')
```



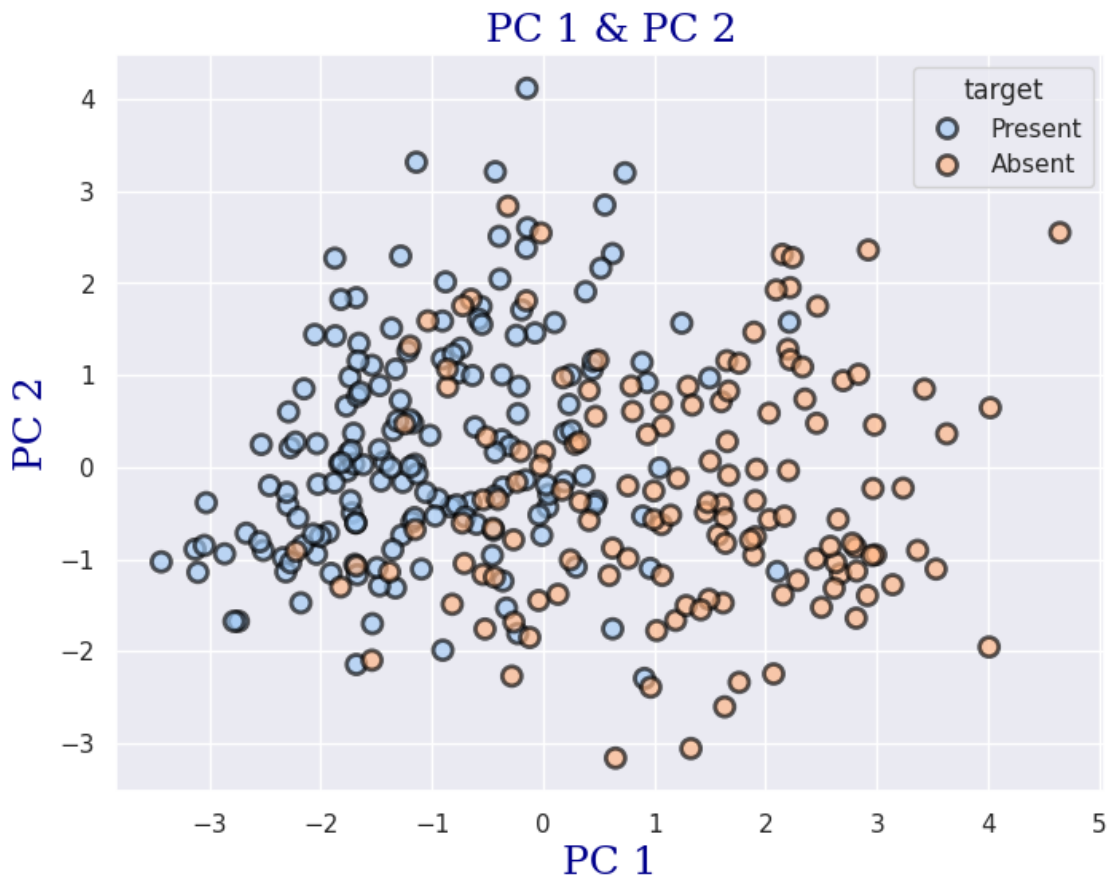
4.2 Principal Component Analysis

PCA (auf Deutsch Hauptkomponentenanalyse) wird als Vorverarbeitungsschritt eingesetzt, um die Daten zu vereinfachen und deren Exploration und Visualisierung zu erleichtern. Es hilft dabei, starke Muster und Beziehungen in den Daten zu erkennen, die zum Verständnis der zugrunde liegenden Struktur beitragen und Informationen für nachfolgende Analysen oder Modellierungsaufgaben liefern können.

```
pca = PCA(n_components=2)
pcs = pca.fit_transform(X)
df_pc = pd.DataFrame(data = pcs, columns = ['PC 1', 'PC 2'])
#dff:dtat frame final
dff = pd.concat([df_pc, df_heart_disease[['target']]], axis = 1)
dff['target'] = dff['target'].replace({1:'Present', 0:'Absent'})
x_tar = dff[df_heart_disease['target'] == 'Present']
y_tar = dff[dff['target'] == 'Absent']
fg6 = sbn.scatterplot(data=dff, x='PC 1', y='PC 2', hue='target',␣
 ↪palette='pastel',
```

```
        s=75, alpha=0.65,**kwargs)
fg6.set_title('PC 1 & PC 2', fontdict=font)
fg6.set_xlabel('PC 1', fontdict=font)
fg6.set_ylabel('PC 2', fontdict=font)
```

```
Text(0, 0.5, 'PC 2')
```



5 Datengetriebenes Modell

Datengetriebenes Modell besteht aus vielen Rechenmethoden, die in zwei Hauptkategorien nämlich Clustering und Klassifikation unterteilt werden können.

Der Schwerpunkt liegt hier darauf, datengetriebene Ansätze zu nutzen, um ein Klassifikationsmodell zu entwickeln, das die Klassenzuordnungen neuer Daten basierend auf ihren Merkmalen genau vorhersagt und die Klassifikation validiert. Der Prozess der datengetriebenen Klassifikation umfasst typischerweise viele Schritte, wie sie in den nächsten Abschnitten aufgeführt sind.

Datenvorverarbeitung:

Dieser Schritt umfasst im Wesentlichen das Bereinigen, Transformieren und Vorbereiten des Daten-

satzes für das Modellieren. Dies beinhaltet die Behandlung fehlender Werte, die Kodierung kategorischer Variablen und die Skalierung numerischer Merkmale, wenn erforderlich.

Hinweis: Dies wurde bereits im letzten Abschnitt durchgeführt.

5.1 Merkmalsauswahl

Die Auswahl relevanter Merkmale oder das Extrahieren wichtiger Informationen aus den Daten, die als Eingaben für das Klassifikationsmodellverwendet werden sollen. Dieser Schritt zielt darauf ab, die Dimensionalität zu reduzieren und sich auf die informativsten Merkmale zu konzentrieren.

Im Folgenden wählen wir die Klasse Sequential Feature Selektor (`SFS`) mit der Einstellung, die im `sfs_config` Wörterbuch definiert ist. Dieses Wörterbuch enthält spezifische Konfigurationen zur Anpassung des Verhaltens des SFS: - `forward`: gibt an, ob der Sequential Feature Selektor einen Vorwärts-Merkmalsauswahlprozess durchführen wird. Bei der Vorwärts-Merkmalsauswahl beginnt der Prozess mit einem leeren Satz von Merkmalen und fügt iterativ jeweils ein Merkmal hinzu, das die Leistung des Modells am meisten verbessert. Dies wird fortgesetzt, bis ein Stoppkriterium erfüllt ist.

- floating: kombiniert die Vorwärts-Merkmalsauswahl und die Rückwärtsauswahl (es ist das Gegenteil der Vorwärts-Merkmalsauswahl, beginnt mit allen Merkmalen und entfernt iterativ jeweils ein Merkmal, das den geringsten Einfluss auf die Leistung des Modells hat), was grundsätzlich teurer ist als die individuelle Vorwärts- oder Rückwärtsauswahl, aber den Vorteil bietet, eine größere Bandbreite an Merkmalskombinationen zu erkunden, was potenziell zu besseren Merkmalsuntergruppen für eine verbesserte Modellleistung führen kann.
- `scoring`: Dies legt den Scoring-Parameter auf 'Genauigkeiten' fest, was bedeutet, dass die Genauigkeitsmetrik zur Bewertung der Qualität der ausgewählten Merkmalsuntergruppen verwendet wird.
- `cv`: Dies legt den cv-Parameter auf 5 fest, was bedeutet, dass eine 5-fache Kreuzvalidierung während des Merkmalsauswahlprozesses verwendet wird. Dies bedeutet, dass der Datensatz in 5 Teilmengen (Folds) aufgeteilt wird und der Merkmalsauswahlalgorithmus 5 Mal ausgeführt wird. In jedem Durchlauf wird eine der Teilmengen als Testset verwendet, und die anderen vier Teilmengen werden als Trainingsset verwendet. Der Zweck der Kreuzvalidierung besteht darin, eine robustere Schätzung der Modellleistung zu liefern, indem es auf verschiedenen Teilmengen der Daten ausgewertet wird.

Die Verwendung einer Pipeline, die SFS integriert, bietet eine strukturierte und kontrollierte Umgebung für die Merkmalsauswahl und stellt sicher, dass sie auf konsistente und zuverlässige Weise in verschiedenen Phasen des Modellierungsprozesses durchgeführt wird. Sie unterstützt auch geeignete Bewertungstechniken wie Kreuzvalidierung und erleichtert die Experimentierung und Anpassung.

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# To avoid the repetition of som settings of the SFS
sfs_config = {
    'forward': True,
    'floating': False,
    'scoring': 'accuracies',
    'cv': 5
```

```
}
```

5.2 Train-Test Aufteilung

Aufteilung des Datensatzes in Trainings- und Testuntergruppen. Der Trainingssatz wird verwendet, um das Klassifikationsmodell zu trainieren, während der Testsatz verwendet wird, um seine Leistung auf unbekannten Daten zu bewerten. Die untenstehende Aufteilung trennt die Merkmalsmatrix **X** (enthält Eingabemerkmale) von der Zielmatrix ***y*** (enthält Klassenlabels). Die Daten werden in Trainings- und Testsätze aufgeteilt, wobei 70% der Daten für das Training und 30% für das Testen verwendet werden, wie kurz oben erwähnt. Das Festlegen des Parameters *random_state* stellt sicher, dass die Trainings- und Testsätze jedes Mal, wenn Sie den Code ausführen, gleich sind, was für die Reproduzierbarkeit nützlich sein kann. Dieser Prozess ist ein entscheidender Schritt bei der Datenvorbereitung für den Aufbau und die Bewertung von Vorhersagemodellen bei Aufgaben des maschinellen Lernens in späteren Schritten.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,␣
 ↪random_state=0)
```

5.3 Modellauswahl und -bewertung

In diesem Abschnitt konzentrieren wir uns auf die entscheidenden Schritte der Modellauswahl und -bewertung für Klassifizierungsaufgaben.

Durch die Einbeziehung sowohl der Modellauswahl als auch der Bewertung können wir fundierte Entscheidungen darüber treffen, welcher Klassifikationsalgorithmus am besten zu unserem spezifischen Problem und Datensatz passt. Der Abschnitt bietet ein umfassendes Verständnis dieser kritischen Schritte beim Aufbau effektiver Klassifikationsmodelle. Um dies zu erreichen, implementieren wir in diesem Abschnitt einen objektorientierten Programmieransatz (OOP), indem wir die Schlüsselfunktionen in eine **ModelEvaluator** Klasse kapseln. Durch die Nutzung dieser ModelEvaluator Klasse vereinfachen wir den Modellauswahlprozess, bewerten die Modellleistung und visualisieren die ROC-Kurven und Wahrheitsmatrix. Der OOP-Ansatz verbessert die Modularität, Wiederverwendbarkeit und Lesbarkeit des Codes und erleichtert so den systematischen Vergleich und die Bewertung verschiedener Klassifikationsalgorithmen.

Hinweis: Bei K-Nearest Neighbors (**KNN**) sollte die Anzahl der für die Klassifikation berücksichtigten Nachbarn idealerweise eine ungerade Zahl sein. Dies ist besonders wichtig in Szenarien der binären Klassifikation, wo eine ungerade Zahl Unentschiedenheiten verhindert, wenn die Nachbarn abstimmen. Wenn es bei dem Abstimmungsprozess ein Unentschieden gibt (z.B. eine gleiche Anzahl von nächsten Nachbarn aus jeder Klasse), stellt die Verwendung einer ungeraden Anzahl von Nachbarn (siehe die Methode *k_opt()* unten) sicher, dass eine Klasse mehr Stimmen hat als die andere, was zu einer klaren Mehrheitsentscheidung führt.

```
def aligned_df(direction, df):
    return df.style.set_properties(**{'text-align': direction}).
 ↪set_table_styles(
            [dict(selector='th', props=[('text-align', direction)])])
```

```
from sklearn.model_selection import train_test_split
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from tqdm import tqdm
from sklearn import model_selection
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


class ModelEvaluator:
    results = {'acc_test': {}, 'acc_train': {}, }
    #-----------------------------------------------------------------------
    # Constructor
    #-----------------------------------------------------------------------
    def __init__(self, model=LinearDiscriminantAnalysis(), df=df_heart_disease,
                 n=13, cvn=5, mif=1, maf=5, head='target'):

        # Check the number of the features of dataframe greater
        #  than cross validation number.
        if df.shape[1] < n:
            raise ValueError('Number of features of dataframe is less than'+
             'the target index number')

        # Check the maximum number of the feature range of a SFS greater
        #  than its minimum number.
        elif maf < mif:
            raise ValueError('Maximum number of feature range of a SFS is'+
                             'less than the minimum number')

        # Check the maximum number of the feature range less than or equal
        # cross validation number.
        elif maf > cvn:
            raise ValueError('Maximum number of feature range of a SFS is'+
                            'greater than the the cross-validation number')
        else:
            X_ = df.iloc[:,0:n].values
            y_ = df.iloc[:,n].values
            df_norm = pd.DataFrame(X_sts, index=df.index, columns=df.columns[0:
 ↪13])
            df_norm[head] = df_heart_disease[head]
            df_norm.head(10)

            self.X = df_norm.iloc[:,0:13].values
```

```
            self.y = df_norm.iloc[:,13].values
            self.X_train, self.X_test, self.y_train, self.y_test =␣
↪train_test_split(
                                self.X, self.y, test_size=0.3, random_state=0)
            self.sfs = SFS(estimator=model, k_features=(mif, maf), forward=True,
            floating=False, scoring='accuracy', cv=cvn).fit(self.X, self.y)
            self.df = df
            self.model = model

    #-------------------------------------------------------------------------
    # This function aims to find the optimal number of neighbors for a
    # K-Nearest Neighbors classifier using cross-validation. It takes
    # training data X_train and corresponding labels y_train as inputs.
    # It iterates over a range of potential neighbor counts, constructs
    # KNN models for each count, evaluates them using cross-validation,
    # and stores the mean accuracy scores. Finally, it determines the neighbor
    # count with the highest accuracy and prints the result. The optimal number
    # of neighbors is returned as the output of the function.
    #-------------------------------------------------------------------------
    def k_opt(self, X_train = None, y_train=None):
        if X_train is None:
            X_train = self.X_train
        if y_train is None:
            y_train = self.y_train
        Ns = [N for N in list(range(1,50)) if N % 2 == 1]
        cv_scores = []

        for k in Ns:
            knn = KNeighborsClassifier(n_neighbors = k + 1, weights='uniform',
                                       p=2, metric='euclidean')
            kf = model_selection.KFold(n_splits=10, shuffle=True,␣
↪random_state=123)
            scores = model_selection.cross_val_score(knn, X_train, y_train,
                                                     cv=kf, scoring='accuracy')
            cv_scores.append(scores.mean()*100)
            #print("k=%d %0.2f (+/- %0.2f)" % (k_value, scores.mean()*100,␣
↪scores.std()*100))

        #opt_k = Ns[cv_scores.index(max(cv_scores))]
        opt_k = Ns[np.argmax(cv_scores)]
        #print ()
        print(( 'The optimal number of neighbors is %d with avg_acc of KNN'+
                '%0.1f%%.' % (opt_k, cv_scores[Ns.index(opt_k)])))
        return opt_k

    #-------------------------------------------------------------------------
```

```
    # This This method encapsulates the process of building, training,␣
↪evaluating,
    # and plotting the results of a neural network model for a given dataset.
    # The flexibility in specifying different parameters allows you to␣
↪experiment
    # with different network architectures and training settings. The boolean␣
↪input
    #'_tqdm' determines whether the model is trained using Keras' built-in␣
↪progress
    #  bar (when _tqdm is set to False) or using a loop over the epoch number␣
↪with
    #  a tqdm progress bar (when _tqdm is set to True).
    #-------------------------------------------------------------------------
    def nnw(self, hu1=64, hu2=32, lr=0.001, num_epoch=1000, _tqdm=True,
            X_train=None, y_train=None, X_test=None, y_test=None):
        if X_train is None:
            X_train = self.X_train
        if y_train is None:
            y_train = self.y_train
        if X_test is None:
            X_test = self.X_test
        if y_test is None:
            y_test = self.y_test

        # Build model
        model = Sequential()
        model.add(Dense(hu1, input_dim=13, activation='relu'))
        model.add(Dense(hu2, activation='relu'))
        model.add(Dense(2, activation='softmax'))
        # Choose optimizer and loss function
        optimizer = SGD(learning_rate=lr, momentum=0.7)
        model.compile(loss='categorical_crossentropy', optimizer=optimizer,
                      metrics=['accuracy'])

        # Convert labels to one-hot encoding
        yTrain_oneHot = to_categorical(y_train, num_classes=2)
        if _tqdm:
            # Train the model with tqdm progress bar
            progress_bar = tqdm(range(num_epoch), unit="epoch")
            loss_per_epoch = []  # Store loss values for each epoch
            for epoch in progress_bar:
                history = model.fit(X_train, yTrain_oneHot, epochs=1, verbose=0)
                 # Append the loss for this epoch
                loss_per_epoch.append(history.history['loss'][0])
        else:
            # Train the model with built-in progress bar
```

```
            history = model.fit(X_train, yTrain_oneHot, epochs=num_epoch,␣
↪verbose=1)
            loss_per_epoch = history.history['loss']
        # Plot the training loss
        epochs = np.arange(1, num_epoch + 1)
        fg10 = sbn.lineplot(x= epochs, y=loss_per_epoch,
                            label='Training', legend=True)
        fg10.set_ylabel('Average Loss', fontdict=font)
        fg10.set_xlabel('Epochs', fontdict=font)
        fg10.set_title('Heart Disease', fontdict=font)
        fg10.legend()

        # Evaluate the model on test data
        yTest_oneHot = to_categorical(y_test, num_classes=2)
        accuracies_test = model.evaluate(X_test, yTest_oneHot, verbose=0)[1]

        # Evaluate the model on training data
        accuracies_train = model.evaluate(X_train, yTrain_oneHot, verbose=0)[1]

        print('Accuracies of the network on test data: {:.2f}%'.
              format(accuracies_test * 100))
        print('Accuracies of the network on training data: {:.2f}%'.
              format(accuracies_train * 100))


    #-----------------------------------------------------------------
    # The function valuates the model's accuracy on both the training
    # and test sets, and it allows for easy tracking and comparison of
    # different models by storing their accuracy scores in dictionaries.
    #-----------------------------------------------------------------
    def accuracies(self, model= None, display=True,
     X_train=None,X_test=None, y_train=None, y_test=None):

        if model is None:
            model = self.model
        if X_train is None:
            X_train = self.X_train
        if X_test is None:
            X_test = self.X_test
        if y_train is None:
            y_train = self.y_train
        if y_test is None:
            y_test = self.y_test

        model.fit(self.X_train, self.y_train)
        y_pred = model.predict(X_test)
        # Update results dictionaries
```

```
        acc_train = round(model.score(self.X_train, y_train) * 100, 2)
        acc_val = round(model.score(self.X_test, y_test) * 100, 2)
        if 'Pipeline' in str(model):
            method_name = str(model.steps[-1][1])[0:str(
                model.steps[-1][1]).find('(')]
            method_name += '_pip'
        else:
            method_name = str(model)[0:str(model).find('(')]

        self.results['acc_test'][method_name] = acc_val
        self.results['acc_train'][method_name] = acc_train

        method_data = {method_name:{'y_pred': [y_pred],
                                    'acc_val':acc_val, 'acc_train': acc_train}}

        if display:
            print('acc_train: %.2f\n'
                'acc_val: %.2f' %(acc_train, acc_val))
        else:
            return method_data

    #-----------------------------------------------------------------------
    # This function encapsulates the steps needed to create and train a
    # pipeline,making it convenient work with models that involve feature
    # selection.
    #-----------------------------------------------------------------------
    def getPipe(self):
        pipe = make_pipeline(
            self.sfs,
            StandardScaler(),
            self.model
        )
        pipe = pipe.fit(self.X, self.y)
        return pipe

    #-----------------------------------------------------------------------
    # The following function serves as a utility to print and retrieve
    # the names and indices of the selected features obtained from the
    # Sequential Feature Selector object.
    #-----------------------------------------------------------------------
    def display_features(self, sfs=None):
        if sfs is None:
            sfs = self.sfs
        #if display == 0:
            # Assuming sfs is an instance of the SequentialFeatureSelector class
        feature_names_dict = {sfs.k_feature_idx_[i]: self.df.iloc[:,
                                                    sfs.k_feature_idx_[i]].name
```

```
                        for i in range(len(sfs.k_feature_idx_))}
        print('Selected features: ',  feature_names_dict)
        print('\n')
        return tuple(np.array(sfs.k_feature_idx_[0:]))

    #-------------------------------------------------------------------------
    # This function provides a quick and easy way to assess the performance of
    # a classification model and is useful for evaluating its effectiveness
    # in distinguishing between different classes.
    #-------------------------------------------------------------------------
    def class_report(self):
        report = classification_report(y_true=self.y_test,
        y_pred=self.getPipe().predict(self.X_test), output_dict=True)
        df_report = pd.DataFrame(report).transpose()
        return aligned_df(df= df_report, direction='left')

    #-----------------------------------------------------------------------
    # The val_diags function provides a comprehensive analysis of the
    # classification model's performance using cross-validated ROC curves,
    #  which helps assess the model's ability to distinguish between
    # classes and compare its performance across different folds.
    # Additionally it plots histogram of the mean TPR and TNR values.
    #-----------------------------------------------------------------------
    def val_diags(self, model=None, X=None, y=None, cvn=5):
        cv = StratifiedKFold(n_splits=cvn)
        # List to store TPR, TNR, and AUC values
        tprs = []
        tnrs = []
        aucs = []

        ax2 = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))[1]
        _fprs = np.linspace(start=0, stop=1, num=300)

        if model is None:
            model = self.getPipe()
        if X is None:
            X = self.X
        if y is None:
            y = self.y
        attrs = self.display_features()

        for i, (train, test) in enumerate(cv.split(X=X[:, attrs], y=y)):
            predicts = model.fit(X[train], y[train].ravel()).
↪predict_proba(X[test])
            fpr, tpr = roc_curve(y[test].ravel(), predicts[:, 1])[0:2]

            tprs.append(np.interp(_fprs, fpr, tpr))
```

```
            #fprs.append(fpr)

            # Calculate TNR based on TPR and FPR
            tnr = 1 - fpr
            tnrs.append(np.interp(_fprs, fpr, tnr))  # Interpolate TNR values

            tprs[-1][0] = 0.0

            roc_auc = auc(fpr, tpr)
            aucs.append(roc_auc)
            fg7 = sbn.lineplot(data=pd.DataFrame({'fpr': fpr, 'tpr': tpr}),
                               x='fpr', y='tpr', alpha=0.3, legend='full',
                         label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc),
                         err_style='band', ax=ax2[0])

        fg7 = sbn.lineplot(x=[0, 1], y=[0, 1], label='Diagonal', legend='full',
                         lw=1.5, alpha=0.7, ax=ax2[0])

        mean_tpr = np.mean(tprs, axis=0)
        mean_tpr[-1] = 1.0
        mean_auc = auc(_fprs, mean_tpr)
        std_auc = np.std(aucs)
        fg7 = sbn.lineplot(x=_fprs, y=mean_tpr,
                         label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)'
                % (mean_auc, std_auc), lw=2.2, alpha=.7, err_style='band',␣
↪ax=ax2[0])

        # Calculate and plot Mean TNR (Specificity)
        mean_tnr = np.mean(tnrs, axis=0)
        mean_tnr[-1] = 1.0  # Set the last TNR value to 1.0 for proper plotting
        #fg7 = sbn.lineplot(x=_fprs, y=mean_tnr, label='Mean TNR (Specificity)',
        #  lw=2.2, alpha=.7, err_style='band', ax=ax2[0])

        fg7.set_title('ROC', fontdict=font)
        fg7.set_xlabel('FPR', fontdict=font)
        fg7.set_ylabel('TRR', fontdict=font)

        fg81= sbn.histplot(mean_tpr, label='TPR',ax=ax2[1], kde=True,␣
↪legend=True,
                         cbar=True, thresh=0, stat='frequency')
        fg81.legend()
        fg82= sbn.histplot(mean_tnr, label='TNR',ax=ax2[1],␣
↪kde=True,legend=True,
                         cbar=True, thresh=0, stat='frequency')
        fg82.legend()
        fg82.set_ylabel('Frequency',fontdict= font)
```

```
        #fg81= sbn.lineplot(x=_fprs,y=mean_tnr, label='TP',ax=ax2[1], ␣
↪legend=True)



  ␣
↪#--------------------------------------------------------------------------------
    # The cfm(abbreviation of Confusion Matrix) function provides an
    # easy-to-interpret visualization of the classification model's performance
    # in terms of true positive,true negative, false positive, and false␣
↪negative
    # predictions, allowing for quick assessment of the model's effectiveness␣
↪in
    # distinguishing between classes.
  ␣
↪#--------------------------------------------------------------------------------
    def cfm(self, model=None, X_test=None, y_test=None):
        if model is None:
            model = self.getPipe()

        if X_test is None:
            X_test = self.X_test

        if y_test is None:
            y_test = self.y_test

        y_pred = model.predict(X_test)
        cm = metrics.confusion_matrix(y_pred= y_pred, y_true= y_test,␣
↪labels=[1, 0])
        fg9 = sbn.heatmap(cm, xticklabels=["1", "0"], yticklabels=["1", "0"],
                          annot=True, fmt='.2f',
          cmap=sbn.cubehelix_palette(as_cmap=True, start=2, light=.75, dark=.25,
                                     gamma=.5,rot=-.2), linewidths=.75)
        #model_name = str(model.steps[-1][1])
        model_name = str(self.model)
        model_name = model_name[0:str(model_name).find('(')]
        fg9.set_title(model_name, fontdict=font)
        fg9.set_xlabel('Predicted label', fontdict=font)
        fg9.set_ylabel('True label', fontdict=font)
```

5.3.1 Modellbewertung

In diesem Abschnitt bewerten wir die Leistung mehrerer Klassifikationsalgorithmen, die häufig für Datenklassifikationsaufgaben verwendet werden, wie zum Beispiel *LinearDiscriminant Analysis (LDA)*, *Decision Trees*, *Random Forest*, *Support Vector Classification (SVC)*, *Logistic Regression*, *Gradient Boosting*, *K-Nearest Neighbors*,und *Neural Netzwerke*. Die Bewertung erfolgt auf der Grundlage der folgenden Metriken:

- **Präzision:** Sie repräsentiert den Anteil der wahren positiven Vorhersagen (en. true positive predictions(`TPP`)) (korrekt vorhergesagte positive Instanzen) an allen positiven Vorhersagen (sowohl wahre Positiven als auch falsche Positiven). Sie misst die Genauigkeit der positiven Vorhersagen.
- **Recall:** Sie ist auch bekannt als Sensitivität oder wahre positive Rate. Sie repräsentiert den Anteil der wahren positiven Vorhersagen an allen tatsächlichen positiven Instanzen im Datensatz. Sie misst die Fähigkeit des Modells, positive Instanzen zu identifizieren.
- **F1-score:** Sie ist das harmonische Mittel aus Präzision und Recall. Sie bietet ein ausgewogenes Maß für Präzision und Recall, insbesondere wenn es ein Ungleichgewicht zwischen positiven und negativen Instanzen gibt.
- **Support:** Sie gibt die Anzahl der Instanzen in jeder Klasse im Testset an und zeigt die Verteilung der Daten über die Klassen.
- **accuracies:** Sie repräsentiert die Gesamtrichtigkeit der Vorhersagen des Modells. Sie ist der Anteil der korrekt vorhergesagten Instanzen (sowohl wahre Positiven als auch wahre Negativen) an der Gesamtzahl der Instanzen.

Die Bewertungsergebnisse helfen uns später, die Leistung verschiedener Modelle zu vergleichen und das für unser spezifisches Klassifikationsproblem am besten geeignete Modell auszuwählen.

Lineare Diskriminanzanalyse (LDA)

Es handelt sich um eine Technik zur Dimensionsreduktion und Klassifikation, die häufig im Bereich des maschinellen Lernens und der Statistik verwendet wird. Ihr Hauptziel ist es, eine lineare Kombination von Merkmalen zu finden, die verschiedene Klassen in einem Datensatz am besten trennt. Sie wird oft für überwachte Klassifikationsaufgaben verwendet, bei denen die Klassen der Daten bekannt sind.

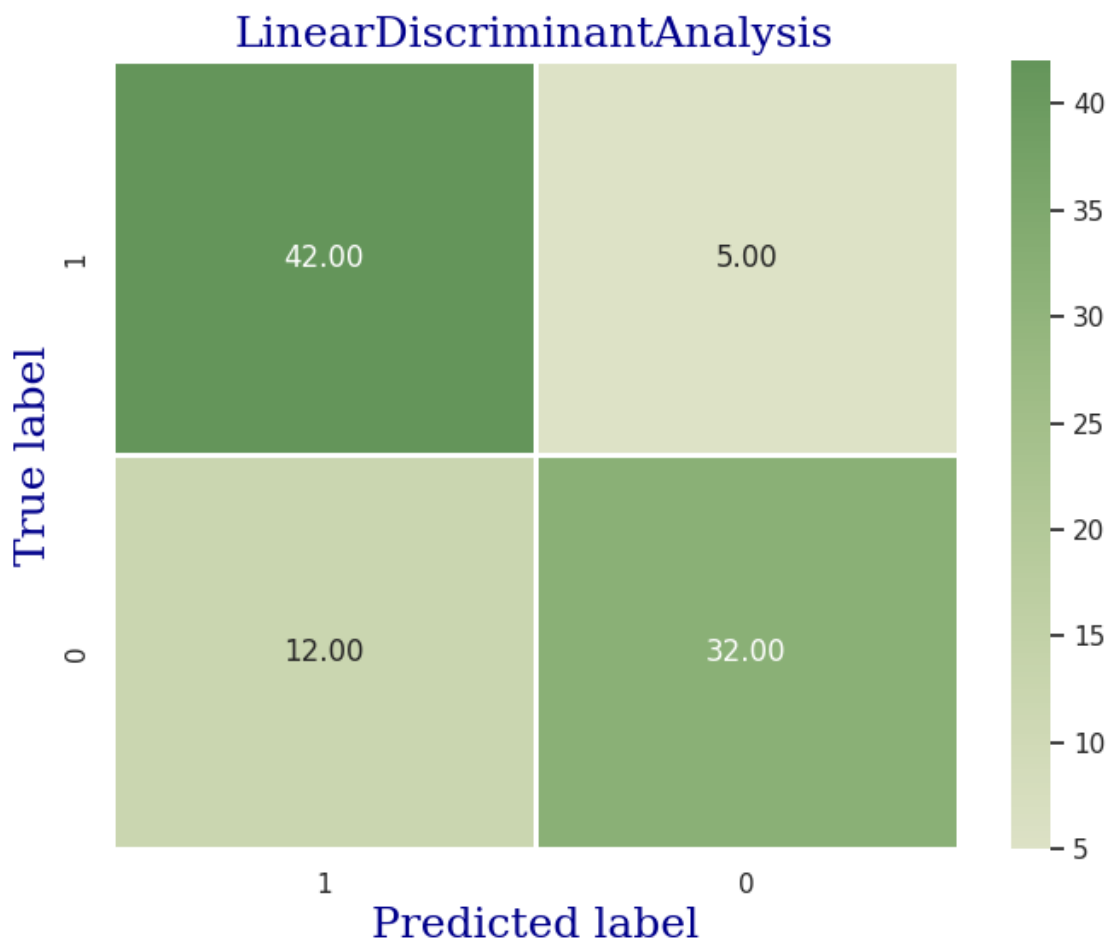
```
evaluator = ModelEvaluator(model=LinearDiscriminantAnalysis())

evaluator.accuracies()
```

```
acc_train: 85.38
acc_val: 80.22
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.864865	0.727273	0.790123	44.000000
1	0.777778	0.893617	0.831683	47.000000
accuracy	0.813187	0.813187	0.813187	0.813187
macro avg	0.821321	0.810445	0.810903	91.000000
weighted avg	0.819886	0.813187	0.811588	91.000000

```
evaluator.cfm()
```

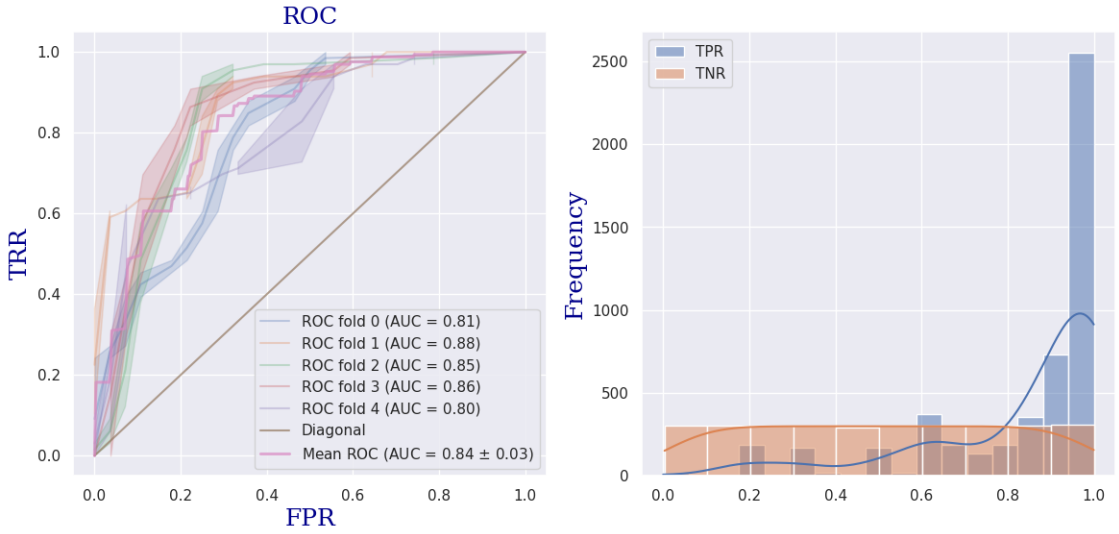


```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 83.49
acc_val: 79.12
```

```
evaluator.val_diags()
```

```
Selected features:  {2: 'cp', 7: 'thalach', 9: 'oldpeak', 11: 'ca', 12: 'thal'}
```



Entscheidungsbäume:

Der Entscheidungsbaum ist ein beliebter und interpretierbarer Algorithmus des maschinellen Lernens, der sowohl für Klassifikations- als auch für Regressionsaufgaben verwendet wird. Er teilt die Daten rekursiv auf der Grundlage der Merkmale auf und erzeugt eine baumähnliche Struktur, bei der jeder interne Knoten eine Entscheidung auf der Grundlage eines Merkmals darstellt und jeder Blattknoten ein Klassenlabel oder eine Regressionsausgabe darstellt.

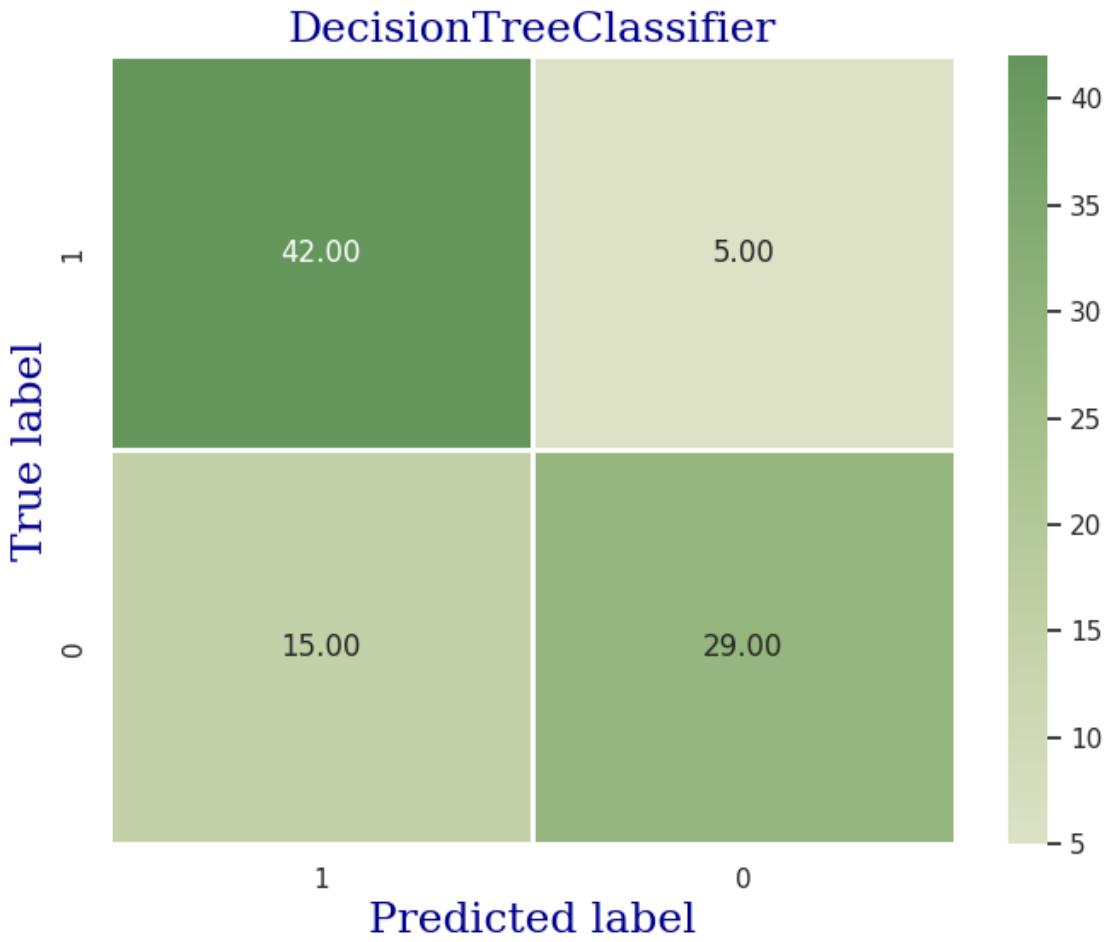
```
evaluator = ModelEvaluator(model=tree.DecisionTreeClassifier())

evaluator.accuracies()
```

```
acc_train: 100.00
acc_val: 74.73
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.852941	0.659091	0.743590	44.000000
1	0.736842	0.893617	0.807692	47.000000
accuracy	0.780220	0.780220	0.780220	0.780220
macro avg	0.794892	0.776354	0.775641	91.000000
weighted avg	0.792978	0.780220	0.776698	91.000000

```
evaluator.cfm()
```

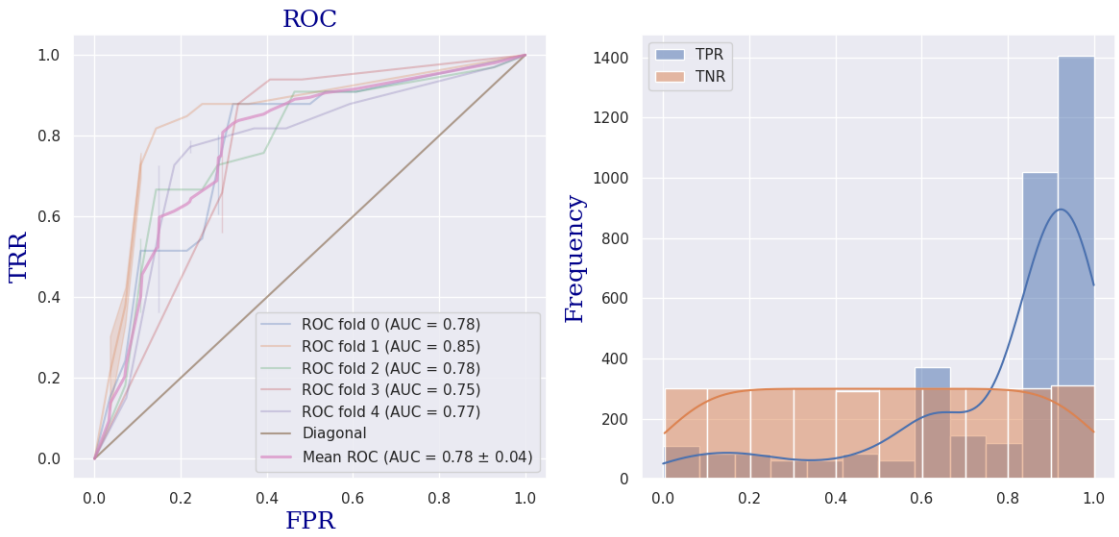


```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 88.68
acc_val: 74.73
```

```
evaluator.val_diags()
```

```
Selected features:  {8: 'exang', 11: 'ca', 12: 'thal'}
```



Random Forests:

Es handelt sich um einen beliebten Algorithmus des maschinellen Lernens, der für Klassifikationsaufgaben verwendet wird. Es handelt sich um eine Methode des Ensemble-Lernens, die während des Trainings mehrere Entscheidungsbäume erstellt und deren Vorhersagen kombiniert, um die endgültige Klassifikationsentscheidung zu treffen.

- **n_estimators:** Die Anzahl der Entscheidungsbäume, die im Random Forest erstellt werden sollen. Eine Erhöhung der Anzahl der Schätzer verbessert in der Regel die Leistung des Modells, erhöht aber auch die Trainingszeit und den Speicherbedarf.
- **random_state:** Der Samen, der vom Zufallszahlengenerator verwendet wird. Er wird verwendet, um die Reproduzierbarkeit der Ergebnisse zu gewährleisten. Wenn random_state=0 im RandomForestClassifier gesetzt wird, bedeutet dies, dass der Samen des Zufallszahlengenerators auf den Wert 0 festgelegt ist. Dies stellt sicher, dass die zufällige Initialisierung des Algorithmus jedes Mal, wenn Sie den Code ausführen, gleich ist, was zu konsistenten Ergebnissen führt. Es entfernt effektiv die Zufälligkeit im Prozess und macht die Ergebnisse reproduzierbar.

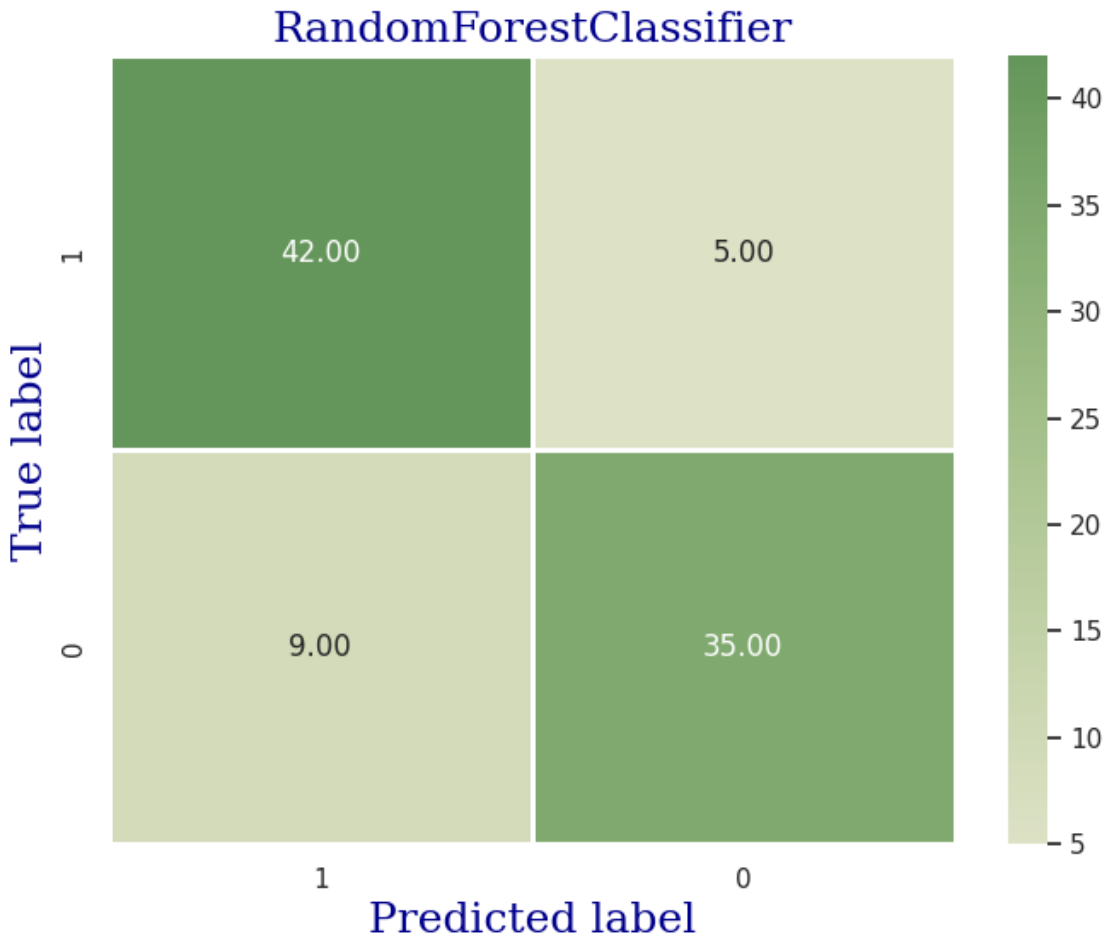
```
evaluator = ModelEvaluator( model=RandomForestClassifier(n_estimators=50,␣
 ↪random_state = 0))
evaluator.accuracies()
```

```
acc_train: 100.00
acc_val: 84.62
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.875000	0.795455	0.833333	44.000000
1	0.823529	0.893617	0.857143	47.000000
accuracy	0.846154	0.846154	0.846154	0.846154
macro avg	0.849265	0.844536	0.845238	91.000000
weighted avg	0.848416	0.846154	0.845631	91.000000

```
evaluator.cfm()
```

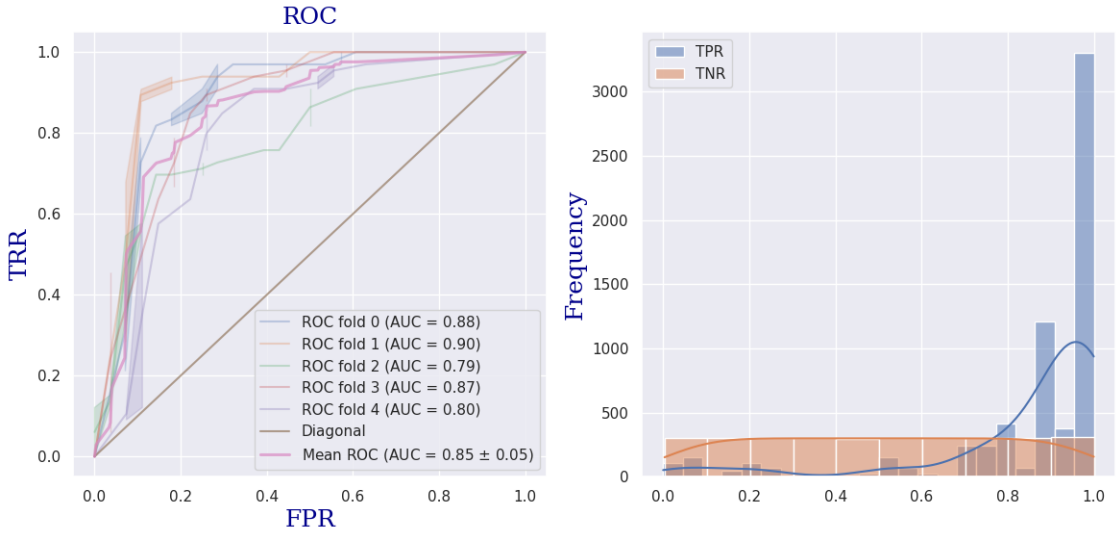



```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 93.40
acc_val: 68.13
```

```
evaluator.val_diags()
```

```
Selected features:  {2: 'cp', 11: 'ca', 12: 'thal'}
```



Support Vector Classification (SVC): SVC bezieht sich speziell auf die Implementierung von Support Vector Machines (SVM) für Klassifikationsaufgaben. Es wird verwendet, wenn man mit gelabelten Daten umgeht und zielt darauf ab, die beste Hyperebene zu finden, die den Abstand zwischen den Klassen maximiert.

- **kernel:** Dieser Parameter gibt an, welche Art von Funktion verwendet wird, um die Eingabemerkmale in einen höherdimensionalen Raum zu transformieren, in dem die Klassen besser getrennt werden können. In diesem Fall bedeutet *`linear`*, dass ein linearer Kernel verwendet wird, was bedeutet, dass das Modell davon ausgeht, dass die Daten durch eine gerade Linie im Merkmalsraum getrennt werden können.
- **probability:** Sie gibt an, ob das SVC-Modell Wahrscheinlichkeitsschätzungen für seine Vorhersagen liefern soll. Wenn sie auf True gesetzt ist, berechnet das Modell die Wahrscheinlichkeitsscores jeder Klassenprognose. Dies kann hilfreich sein für Aufgaben wie die ROC-AUC-Bewertung oder wenn Sie das Vertrauen des Modells in seine Vorhersagen bewerten möchten.

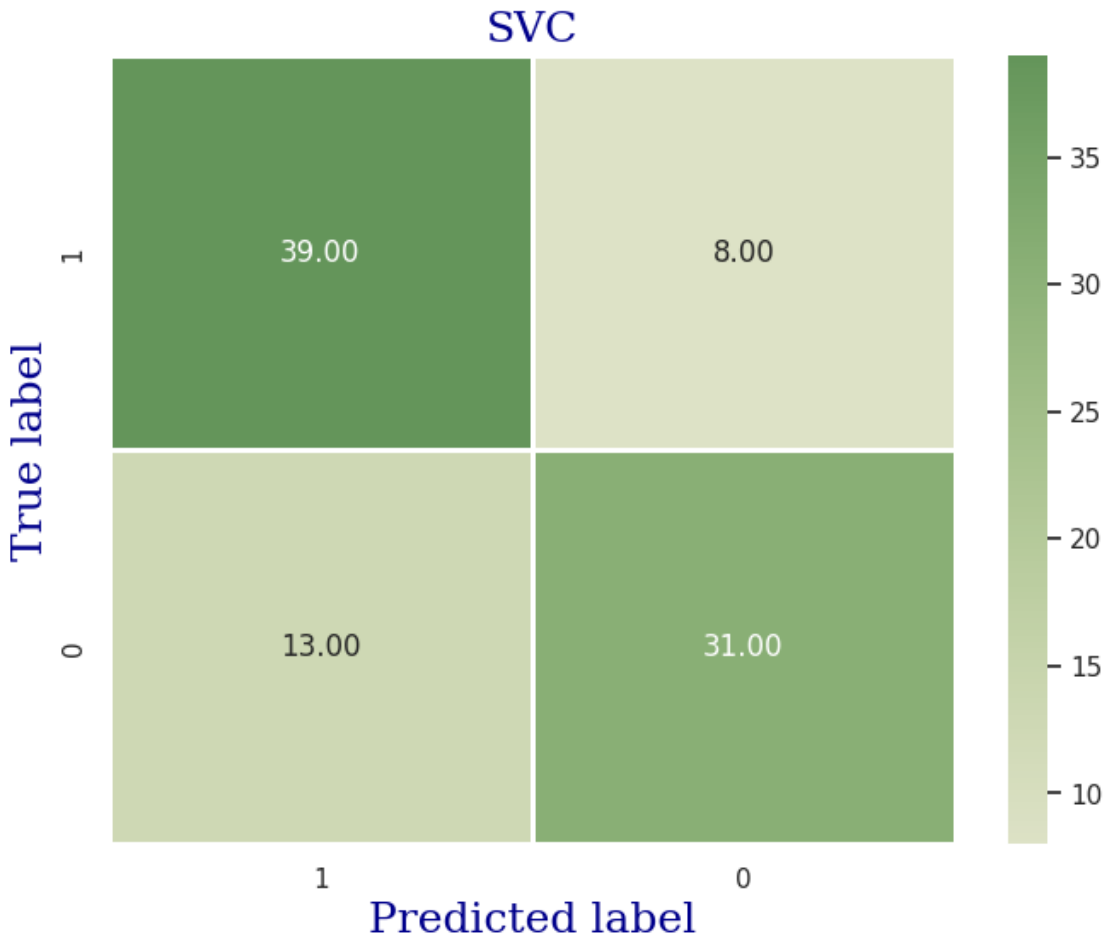
```
evaluator = ModelEvaluator(model=SVC(kernel='linear',probability=True))
evaluator.accuracies()
```

```
acc_train: 86.79
acc_val: 80.22
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.794872	0.704545	0.746988	44.000000
1	0.750000	0.829787	0.787879	47.000000
accuracy	0.769231	0.769231	0.769231	0.769231
macro avg	0.772436	0.767166	0.767433	91.000000
weighted avg	0.771696	0.769231	0.768107	91.000000

```
evaluator.cfm()
```

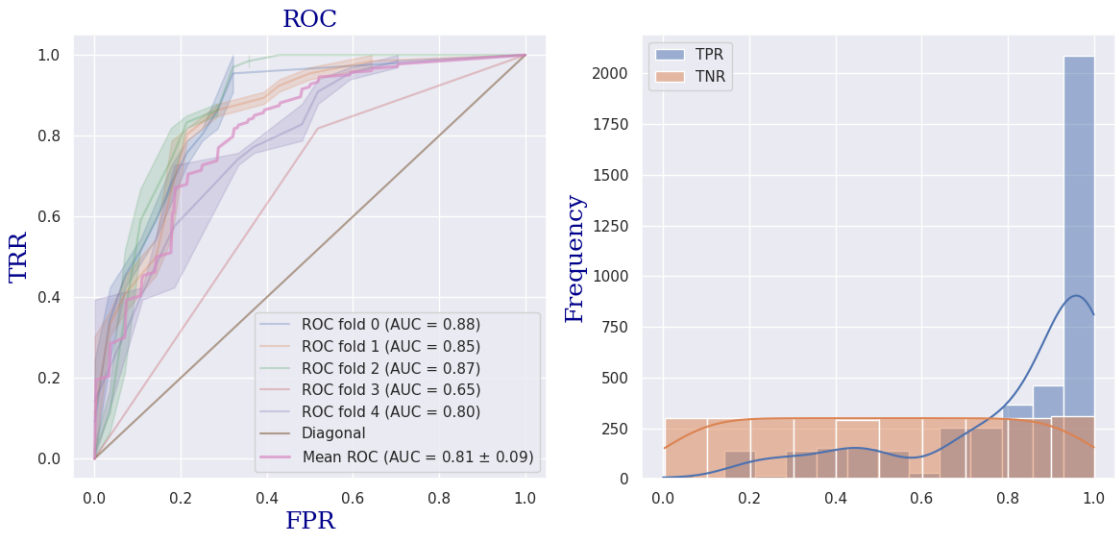


```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 84.43
acc_val: 79.12
```

```
evaluator.val_diags()
```

```
Selected features:  {2: 'cp', 7: 'thalach', 8: 'exang', 11: 'ca'}
```



Logistic Regression:

Es handelt sich um einen Klassifikationsalgorithmus, der für binäre und multiklassen Klassifikationsaufgaben verwendet wird. Das logistische Regressionsmodell ist ein linearer Klassifikator, der die Wahrscheinlichkeit, dass eine Probe zu einer bestimmten Klasse gehört, mit der logistischen Sigmoid-Funktion.

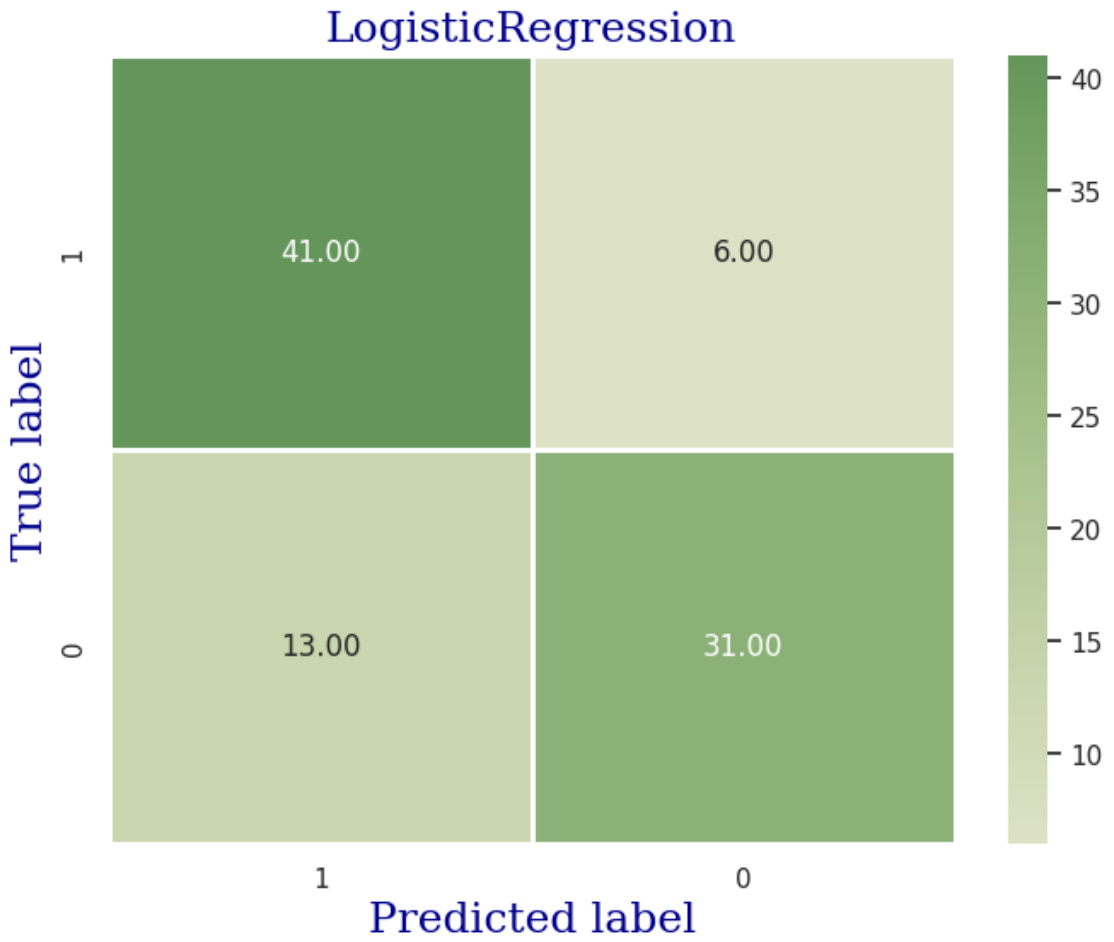
```
evaluator = ModelEvaluator(model= LogisticRegression(max_iter=1000))
evaluator.accuracies()
```

```
acc_train: 86.32
acc_val: 81.32
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.837838	0.704545	0.765432	44.000000
1	0.759259	0.872340	0.811881	47.000000
accuracy	0.791209	0.791209	0.791209	0.791209
macro avg	0.798549	0.788443	0.788657	91.000000
weighted avg	0.797253	0.791209	0.789422	91.000000

```
evaluator.cfm()
```

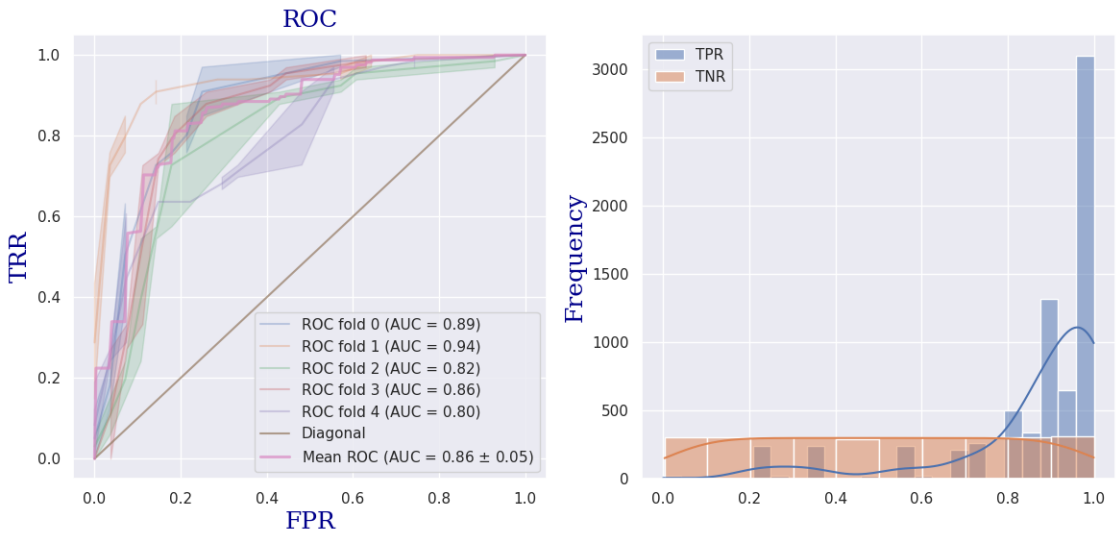


```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 81.60
acc_val: 81.32
```

```
evaluator.val_diags()
```

```
Selected features:  {1: 'sex', 2: 'cp', 8: 'exang', 9: 'oldpeak', 11: 'ca'}
```



Gradient Boosting:

Gradient Boosting ist eine leistungsstarke Technik des maschinellen Lernens, die sowohl für Regressions- als auch für Klassifikationsaufgaben verwendet wird. Sie gehört zur Familie der Ensemble-Lernmethoden, die die Vorhersagen mehrerer schwacher Lernmodelle (in der Regel Entscheidungsbäume) kombinieren, um ein starkes Vorhersagemodell zu erstellen.

- **loss:** Dieser Parameter definiert die Verlustfunktion, die während des Boosting-Prozesses optimiert werden soll. Der 'exponentielle' Verlust eignet sich für Klassifikationsprobleme und ermutigt das Modell, sich stärker auf falsch klassifizierte Proben zu konzentrieren.
- **learning_rate:** Die Lernrate steuert den Beitrag jedes schwachen Lerners (einzelner Entscheidungsbaum) zum Ensemble. Eine niedrigere Lernrate erfordert mehr Iterationen, um eine optimale Leistung zu erreichen, kann aber helfen, Überanpassung zu verhindern.
- **n_estimators:** Dies ist die Anzahl der schwachen Lerner (Entscheidungsbäume), die sequenziell trainiert werden. Eine Erhöhung der Anzahl der Schätzer kann die Leistung des Modells verbessern, erhöht aber auch die Rechenzeit.

- **max_depth:** Dieser Parameter legt die maximale Tiefe der einzelnen Entscheidungsbäume fest. Er steuert die Komplexität der Bäume. Ein tieferer Baum kann komplexere Beziehungen in den Daten erfassen, kann aber auch zu Überanpassung führen.

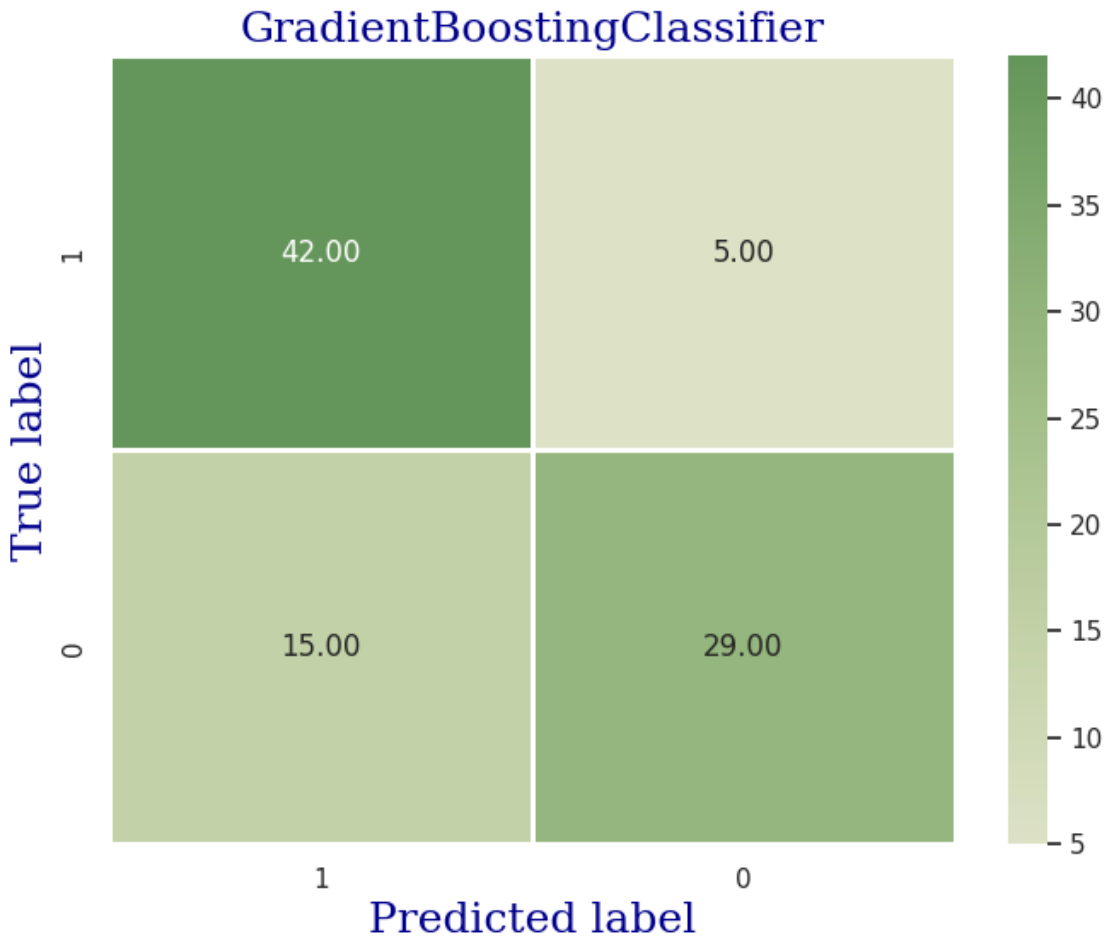
```
evaluator = ModelEvaluator(model= GradientBoostingClassifier(loss='exponential',
                                                             learning_rate=0.05,␣
 ↪n_estimators=200, max_depth=6))
evaluator.accuracies()
```

```
acc_train: 100.00
acc_val: 78.02
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.852941	0.659091	0.743590	44.000000
1	0.736842	0.893617	0.807692	47.000000
accuracy	0.780220	0.780220	0.780220	0.780220
macro avg	0.794892	0.776354	0.775641	91.000000
weighted avg	0.792978	0.780220	0.776698	91.000000

```
evaluator.cfm()
```

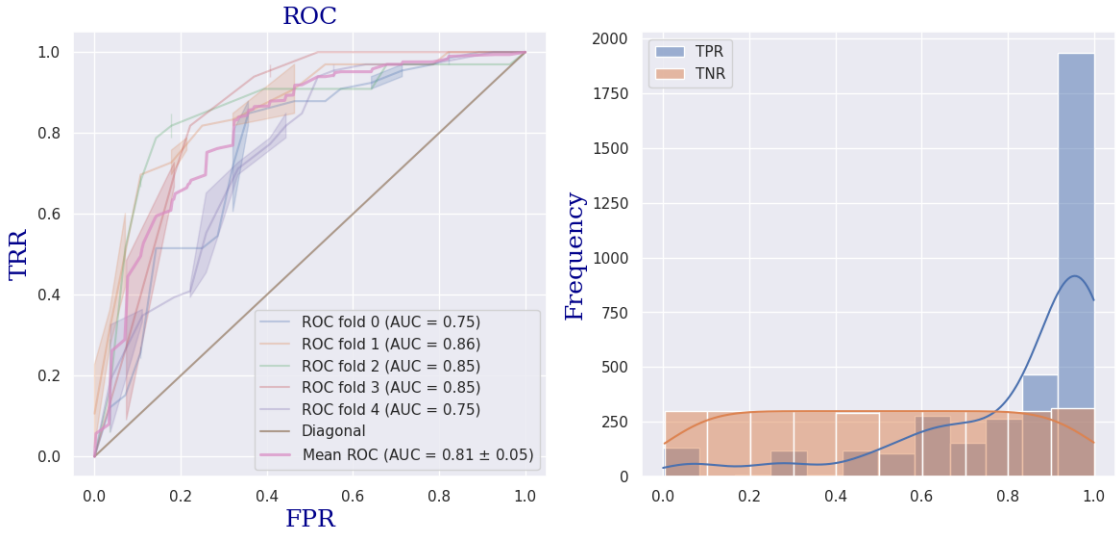



```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 88.68
acc_val: 74.73
```

```
evaluator.val_diags()
```

```
Selected features:  {8: 'exang', 11: 'ca', 12: 'thal'}
```



K-Nearest Neighbors (KNN):

Dies ist eine Klasse aus dem Modul Neighbors Scikit-learn, die den K-Nearest Neighbors-Klassifikator repräsentiert.KNN ist ein einfacher und effektiver Klassifizierungsalgorithmus, der sowohl für binäre als auch für Mehrklassen-Klassifizierungsaufgaben verwendet wird.

- **n_neighbors:** Dies ist ein Parameter des KNeighborsClassifier-Konstruktors. Er gibt die Anzahl der Nachbarn an, die bei der Vorhersage berücksichtigt werden sollen. Jeder Datenpunkt wird durch eine Mehrheitsabstimmung seiner n_neighbors nächsten Nachbarn klassifiziert. Die optimale Anzahl der Nachbarn wird als Ausgabe der Funktion *k_opt()* zurückgegeben.

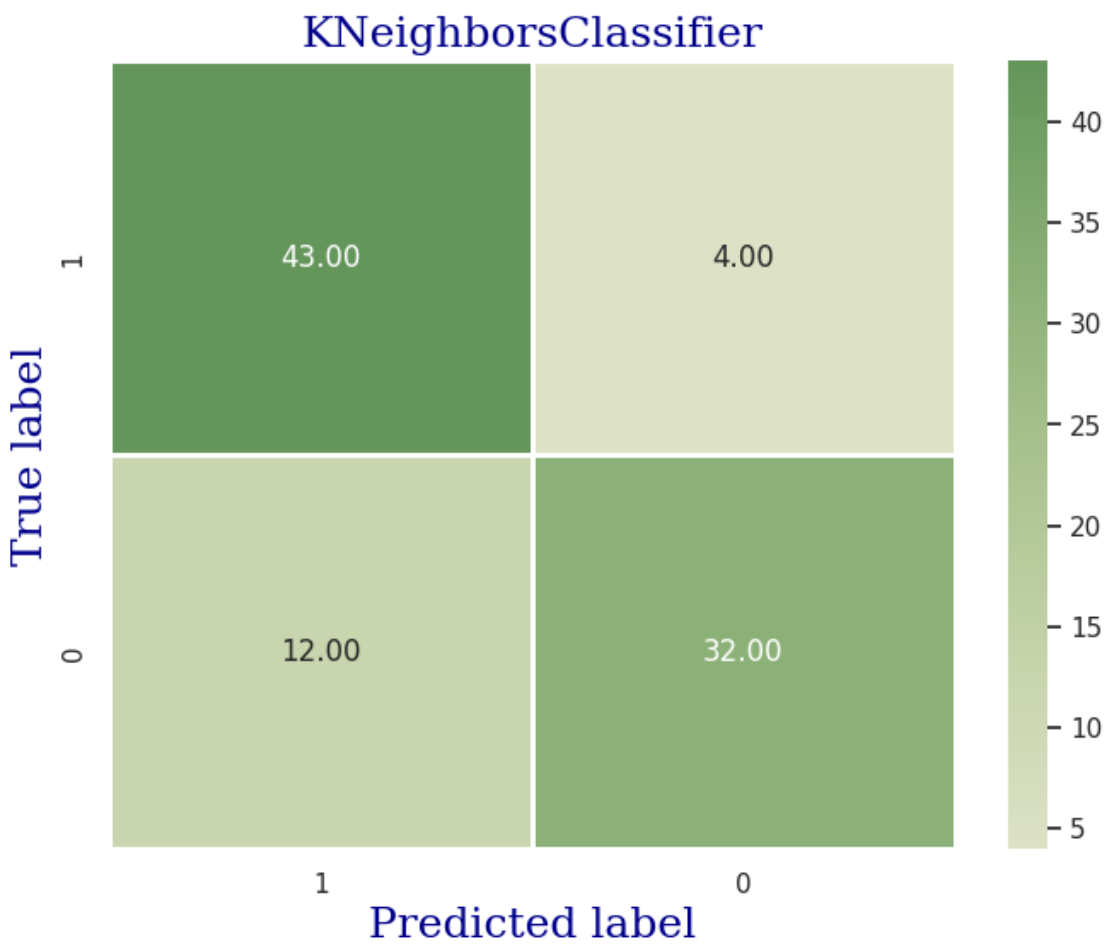
```
evaluator = ModelEvaluator(model=KNeighborsClassifier(n_neighbors=evaluator.
 ↪k_opt()))

evaluator.accuracies()
```

```
The optimal number of neighbors is 31 with avg_acc of KNN 83.6%.
acc_train: 84.91
acc_val: 78.02
```

```
evaluator.class_report()
```

	precision	recall	f1-score	support
0	0.888889	0.727273	0.800000	44.000000
1	0.781818	0.914894	0.843137	47.000000
accuracy	0.824176	0.824176	0.824176	0.824176
macro avg	0.835354	0.821083	0.821569	91.000000
weighted avg	0.833589	0.824176	0.822280	91.000000

```
evaluator.cfm()
```

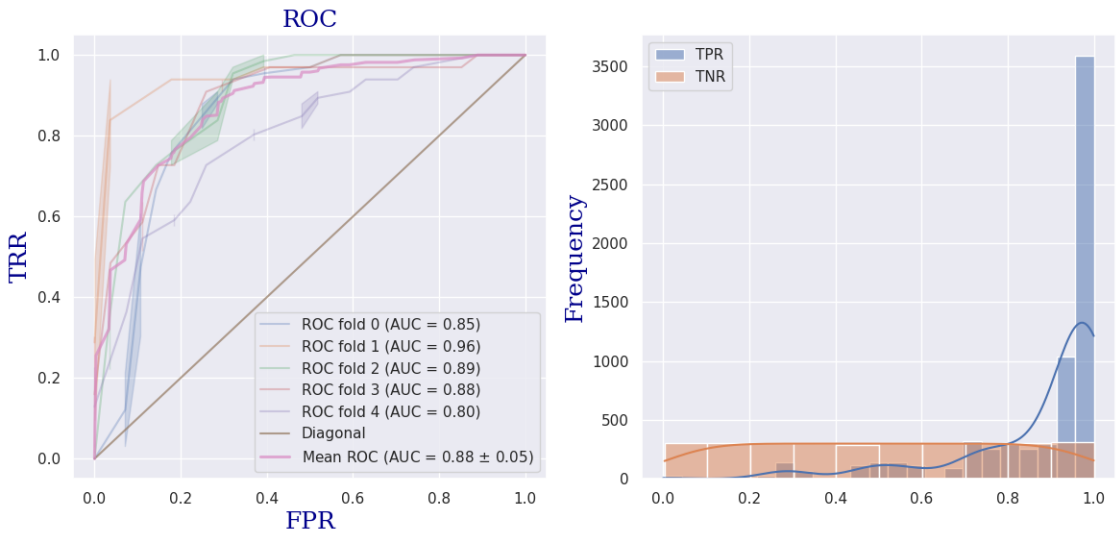


```
evaluator.accuracies(model=evaluator.getPipe())
```

```
acc_train: 83.96
acc_val: 80.22
```

```
evaluator.val_diags()
```

```
Selected features:  {2: 'cp', 7: 'thalach', 11: 'ca', 12: 'thal'}
```



Neuronale Netzwerke:

Neuronale Netzwerke sind fortschrittliche Modelle des maschinellen Lernens, die weit verbreitet für Klassificationsaufgaben. Sie sind inspiriert von den vernetzten Neuronen des menschlichen Gehirns und sind sehr effektiv bei der Lösung komplexer Klassifikationsprobleme. In diesem Kontext lernen sie, Eingabedaten auf verschiedene Klassen abzubilden, was sie zu unschätzbaren Werkzeugen für Aufgaben wie Bilderkennung, Sentimentanalyse, Krankheitsdiagnose und mehr macht. Die Fähigkeit neuronaler Netzwerke, automatisch komplexe Muster und Hierarchien in Daten zu lernen, macht sie zur ersten Wahl bei der Bewältigung komplexer und nichtlinearer Klassifikationsherausforderungen. Um ein effektives neuronales Netzwerk zu erstellen, sollten Sie die folgenden Eingabeparameter berücksichtigen, die die Leistung und Konvergenz des Modells beeinflussen:

- ***Anzahl der versteckten Einheiten:*** Die Fähigkeit eines Modells, komplexe Muster zu lernen, hängt von der Anzahl der versteckten Einheiten in jeder Schicht ab (hier standardmäßig auf hu1= 64 und hu2=32 gesetzt). Mehr versteckte Einheiten können komplexe Beziehungen erfassen, riskieren aber auch Überanpassung(en. Overfitting), wenn sie nicht richtig regularisiert sind. Durch Experimentieren finden Sie das ideale Gleichgewicht von versteckten Einheiten für Ihr spezifisches Problem.

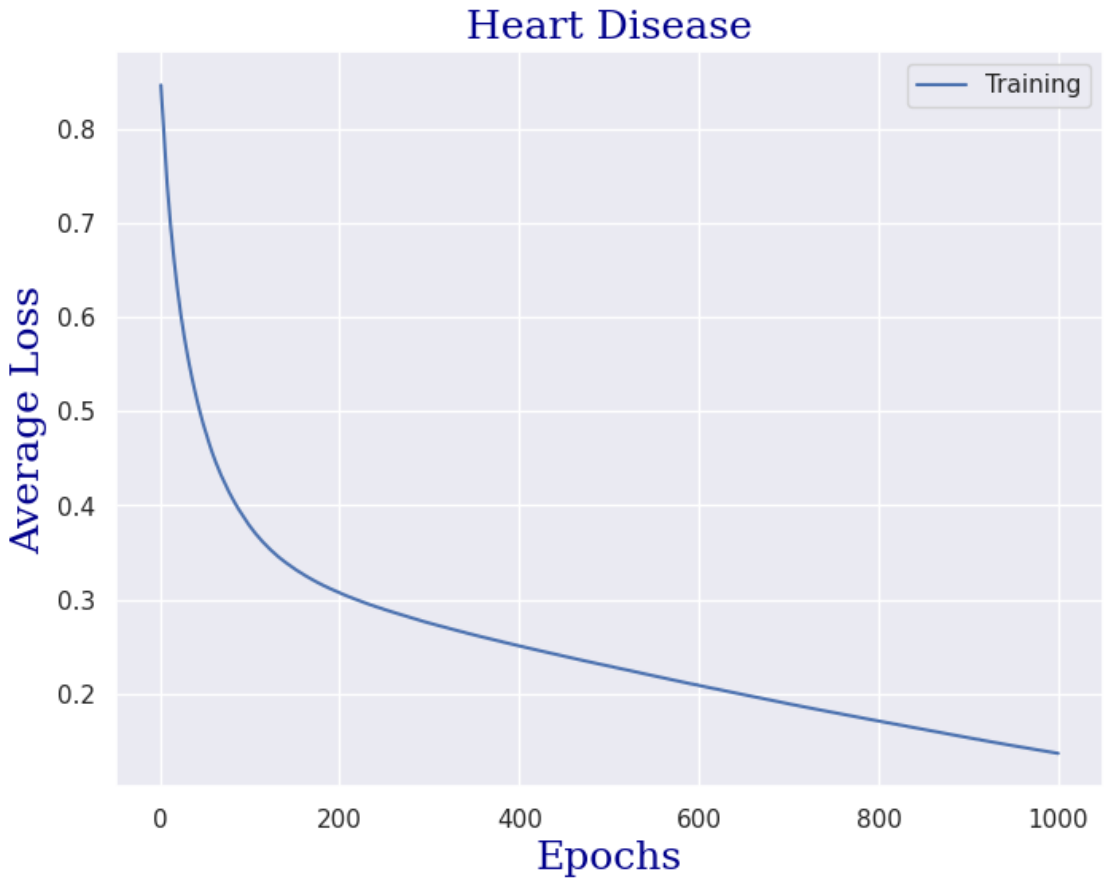
- ***Anzahl der Epochen:*** TDie Epochen bestimmen, wie oft das Modell den Trainingsdatensatz verarbeitet (hier standardmäßig auf 1000 gesetzt). Zu wenige Epochen können zu Unteranpassung führen, während übermäßige Epochen zur Überanpassung führen können. Durch Überwachung der Validierungsleistung und rechtzeitiges Beenden des Trainings kann das optimale Gleichgewicht gefunden werden.
- ***Lernrate:*** Die Lernrate steuert die Schrittgröße bei Parameteraktualisierungen während des Trainings (hier standardmäßig auf 0,001 gesetzt). Eine höhere Rate beschleunigt die Konvergenz, könnte aber zu Überschwingen führen. Umgekehrt kann eine kleinere Rate zu einer langsamen Konvergenz führen. Adaptive Optimierer wie Adam können die manuelle Einstellung der Lernrate erleichtern.

Diese Parameter formen gemeinsam die Architektur des neuronalen Netzwerks und die Trainingsdynamik und leiten es an, effektiv aus den Daten zu lernen und die gewünschte Klassifikationsleistung zu erreichen.

```
evaluator = ModelEvaluator()
```

```
#nnw(self, hu1=64, hu2=32, lr=0.001, num_epoch=1000, _tqdm=True)
evaluator.nnw()
```

```
/home/montaha/Documents/practical/script/python/kaggle/heart_disease/kaggle/da_p
roj/.venv/lib/python3.11/site-packages/keras/src/layers/core/dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-04-17 18:22:11.157304: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:998] successful
NUMA node read from SysFS had negative value (-1), but there must be at least
one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-04-17 18:22:11.157614: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
100%|      | 1000/1000 [00:38<00:00, 25.95epoch/s]

Accuracies of the network on test data: 81.32%
Accuracies of the network on training data: 95.75%
```




```
evaluator.nnw(_tqdm=False)
```

```
Epoch 1/1000

/home/montaha/Documents/practical/script/python/kaggle/heart_disease/kaggle/da_p
roj/.venv/lib/python3.11/site-packages/keras/src/layers/core/dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

7/7             0s 2ms/step -
accuracy: 0.4676 - loss: 0.7475
Epoch 2/1000
7/7             0s 743us/step -
accuracy: 0.4154 - loss: 0.7746
Epoch 3/1000
7/7             0s 989us/step -
accuracy: 0.4233 - loss: 0.7442
Epoch 4/1000
```

```
7/7             0s 749us/step -
accuracy: 0.4438 - loss: 0.7305
Epoch 5/1000
7/7             0s 878us/step -
accuracy: 0.4759 - loss: 0.7211
Epoch 6/1000
7/7             0s 929us/step -
accuracy: 0.5155 - loss: 0.6965
Epoch 7/1000
7/7             0s 661us/step -
accuracy: 0.5381 - loss: 0.6964
Epoch 8/1000
7/7             0s 713us/step -
accuracy: 0.5712 - loss: 0.6773
Epoch 9/1000
7/7             0s 756us/step -
accuracy: 0.5790 - loss: 0.6683
Epoch 10/1000
7/7             0s 736us/step -
accuracy: 0.6009 - loss: 0.6557
Epoch 11/1000
7/7             0s 687us/step -
accuracy: 0.6538 - loss: 0.6473
Epoch 12/1000
7/7             0s 704us/step -
accuracy: 0.6485 - loss: 0.6387
Epoch 13/1000
7/7             0s 730us/step -
accuracy: 0.6512 - loss: 0.6337
Epoch 14/1000
7/7             0s 849us/step -
accuracy: 0.6987 - loss: 0.6266
Epoch 15/1000
7/7             0s 851us/step -
accuracy: 0.6956 - loss: 0.6159
Epoch 16/1000
7/7             0s 720us/step -
accuracy: 0.7057 - loss: 0.6188
Epoch 17/1000
7/7             0s 740us/step -
accuracy: 0.7077 - loss: 0.6155
Epoch 18/1000
7/7             0s 646us/step -
accuracy: 0.6743 - loss: 0.6173
Epoch 19/1000
7/7             0s 729us/step -
accuracy: 0.7459 - loss: 0.5837
Epoch 20/1000
```

```
7/7            0s 718us/step -
accuracy: 0.7334 - loss: 0.5848
Epoch 21/1000
7/7            0s 615us/step -
accuracy: 0.7674 - loss: 0.5726
Epoch 22/1000
7/7            0s 861us/step -
accuracy: 0.7488 - loss: 0.5715
Epoch 23/1000
7/7            0s 883us/step -
accuracy: 0.7323 - loss: 0.5717
Epoch 24/1000
7/7            0s 807us/step -
accuracy: 0.7659 - loss: 0.5566
Epoch 25/1000
7/7            0s 763us/step -
accuracy: 0.7477 - loss: 0.5560
Epoch 26/1000
7/7            0s 781us/step -
accuracy: 0.7759 - loss: 0.5525
Epoch 27/1000
7/7            0s 784us/step -
accuracy: 0.7675 - loss: 0.5531
Epoch 28/1000
7/7            0s 843us/step -
accuracy: 0.7796 - loss: 0.5456
Epoch 29/1000
7/7            0s 729us/step -
accuracy: 0.7565 - loss: 0.5401
Epoch 30/1000
7/7            0s 809us/step -
accuracy: 0.7814 - loss: 0.5234
Epoch 31/1000
7/7            0s 673us/step -
accuracy: 0.7985 - loss: 0.5271
Epoch 32/1000
7/7            0s 787us/step -
accuracy: 0.7725 - loss: 0.5318
Epoch 33/1000
7/7            0s 900us/step -
accuracy: 0.8038 - loss: 0.5227
Epoch 34/1000
7/7            0s 846us/step -
accuracy: 0.7726 - loss: 0.5278
Epoch 35/1000
7/7            0s 923us/step -
accuracy: 0.8016 - loss: 0.5159
Epoch 36/1000
```

```
7/7            0s 704us/step -
accuracy: 0.7711 - loss: 0.5207
Epoch 37/1000
7/7            0s 758us/step -
accuracy: 0.7691 - loss: 0.5046
Epoch 38/1000
7/7            0s 635us/step -
accuracy: 0.8192 - loss: 0.4847
Epoch 39/1000
7/7            0s 738us/step -
accuracy: 0.8113 - loss: 0.4920
Epoch 40/1000
7/7            0s 701us/step -
accuracy: 0.7759 - loss: 0.5065
Epoch 41/1000
7/7            0s 780us/step -
accuracy: 0.7996 - loss: 0.5055
Epoch 42/1000
7/7            0s 763us/step -
accuracy: 0.7930 - loss: 0.5063
Epoch 43/1000
7/7            0s 857us/step -
accuracy: 0.8101 - loss: 0.4810
Epoch 44/1000
7/7            0s 694us/step -
accuracy: 0.8113 - loss: 0.4856
Epoch 45/1000
7/7            0s 696us/step -
accuracy: 0.8041 - loss: 0.4844
Epoch 46/1000
7/7            0s 750us/step -
accuracy: 0.7992 - loss: 0.4843
Epoch 47/1000
7/7            0s 626us/step -
accuracy: 0.8136 - loss: 0.4623
Epoch 48/1000
7/7            0s 865us/step -
accuracy: 0.8239 - loss: 0.4629
Epoch 49/1000
7/7            0s 979us/step -
accuracy: 0.8038 - loss: 0.4824
Epoch 50/1000
7/7            0s 733us/step -
accuracy: 0.8094 - loss: 0.4778
Epoch 51/1000
7/7            0s 748us/step -
accuracy: 0.8368 - loss: 0.4495
Epoch 52/1000
```

```
7/7            0s 689us/step -
accuracy: 0.8129 - loss: 0.4727
Epoch 53/1000
7/7            0s 733us/step -
accuracy: 0.8349 - loss: 0.4353
Epoch 54/1000
7/7            0s 750us/step -
accuracy: 0.7886 - loss: 0.4767
Epoch 55/1000
7/7            0s 710us/step -
accuracy: 0.8169 - loss: 0.4610
Epoch 56/1000
7/7            0s 787us/step -
accuracy: 0.8398 - loss: 0.4370
Epoch 57/1000
7/7            0s 815us/step -
accuracy: 0.8264 - loss: 0.4356
Epoch 58/1000
7/7            0s 734us/step -
accuracy: 0.8228 - loss: 0.4376
Epoch 59/1000
7/7            0s 871us/step -
accuracy: 0.8233 - loss: 0.4424
Epoch 60/1000
7/7            0s 695us/step -
accuracy: 0.8278 - loss: 0.4326
Epoch 61/1000
7/7            0s 766us/step -
accuracy: 0.8329 - loss: 0.4326
Epoch 62/1000
7/7            0s 763us/step -
accuracy: 0.8362 - loss: 0.4326
Epoch 63/1000
7/7            0s 730us/step -
accuracy: 0.8036 - loss: 0.4529
Epoch 64/1000
7/7            0s 805us/step -
accuracy: 0.8232 - loss: 0.4399
Epoch 65/1000
7/7            0s 2ms/step -
accuracy: 0.8116 - loss: 0.4372
Epoch 66/1000
7/7            0s 916us/step -
accuracy: 0.8159 - loss: 0.4374
Epoch 67/1000
7/7            0s 734us/step -
accuracy: 0.8249 - loss: 0.4223
Epoch 68/1000
```

```
7/7            0s 726us/step -
accuracy: 0.8303 - loss: 0.4236
Epoch 69/1000
7/7            0s 841us/step -
accuracy: 0.8421 - loss: 0.4156
Epoch 70/1000
7/7            0s 804us/step -
accuracy: 0.8120 - loss: 0.4282
Epoch 71/1000
7/7            0s 791us/step -
accuracy: 0.8337 - loss: 0.4115
Epoch 72/1000
7/7            0s 961us/step -
accuracy: 0.8344 - loss: 0.4254
Epoch 73/1000
7/7            0s 995us/step -
accuracy: 0.8411 - loss: 0.4170
Epoch 74/1000
7/7            0s 771us/step -
accuracy: 0.8408 - loss: 0.4134
Epoch 75/1000
7/7            0s 841us/step -
accuracy: 0.8255 - loss: 0.4202
Epoch 76/1000
7/7            0s 824us/step -
accuracy: 0.8366 - loss: 0.3944
Epoch 77/1000
7/7            0s 754us/step -
accuracy: 0.8419 - loss: 0.4078
Epoch 78/1000
7/7            0s 640us/step -
accuracy: 0.8506 - loss: 0.4090
Epoch 79/1000
7/7            0s 699us/step -
accuracy: 0.8304 - loss: 0.4200
Epoch 80/1000
7/7            0s 857us/step -
accuracy: 0.8291 - loss: 0.4099
Epoch 81/1000
7/7            0s 818us/step -
accuracy: 0.8399 - loss: 0.4039
Epoch 82/1000
7/7            0s 823us/step -
accuracy: 0.8172 - loss: 0.4115
Epoch 83/1000
7/7            0s 724us/step -
accuracy: 0.8270 - loss: 0.4225
Epoch 84/1000
```

```
7/7            0s 739us/step -
accuracy: 0.8528 - loss: 0.4080
Epoch 85/1000
7/7            0s 783us/step -
accuracy: 0.8470 - loss: 0.3977
Epoch 86/1000
7/7            0s 819us/step -
accuracy: 0.8146 - loss: 0.4295
Epoch 87/1000
7/7            0s 1ms/step -
accuracy: 0.8229 - loss: 0.4222
Epoch 88/1000
7/7            0s 737us/step -
accuracy: 0.8527 - loss: 0.3969
Epoch 89/1000
7/7            0s 707us/step -
accuracy: 0.8729 - loss: 0.3578
Epoch 90/1000
7/7            0s 745us/step -
accuracy: 0.8362 - loss: 0.4045
Epoch 91/1000
7/7            0s 873us/step -
accuracy: 0.8566 - loss: 0.3807
Epoch 92/1000
7/7            0s 775us/step -
accuracy: 0.8794 - loss: 0.3819
Epoch 93/1000
7/7            0s 787us/step -
accuracy: 0.8531 - loss: 0.3893
Epoch 94/1000
7/7            0s 676us/step -
accuracy: 0.8584 - loss: 0.3805
Epoch 95/1000
7/7            0s 637us/step -
accuracy: 0.8373 - loss: 0.3960
Epoch 96/1000
7/7            0s 866us/step -
accuracy: 0.8631 - loss: 0.3568
Epoch 97/1000
7/7            0s 676us/step -
accuracy: 0.8625 - loss: 0.3658
Epoch 98/1000
7/7            0s 877us/step -
accuracy: 0.8464 - loss: 0.3827
Epoch 99/1000
7/7            0s 812us/step -
accuracy: 0.8870 - loss: 0.3364
Epoch 100/1000
```

```
7/7            0s 908us/step -
accuracy: 0.8739 - loss: 0.3614
Epoch 101/1000
7/7            0s 792us/step -
accuracy: 0.8257 - loss: 0.4007
Epoch 102/1000
7/7            0s 721us/step -
accuracy: 0.8552 - loss: 0.3869
Epoch 103/1000
7/7            0s 628us/step -
accuracy: 0.8694 - loss: 0.3599
Epoch 104/1000
7/7            0s 888us/step -
accuracy: 0.8476 - loss: 0.3903
Epoch 105/1000
7/7            0s 750us/step -
accuracy: 0.8555 - loss: 0.3577
Epoch 106/1000
7/7            0s 851us/step -
accuracy: 0.8344 - loss: 0.3771
Epoch 107/1000
7/7            0s 726us/step -
accuracy: 0.8403 - loss: 0.3890
Epoch 108/1000
7/7            0s 771us/step -
accuracy: 0.8970 - loss: 0.3226
Epoch 109/1000
7/7            0s 735us/step -
accuracy: 0.8590 - loss: 0.3635
Epoch 110/1000
7/7            0s 844us/step -
accuracy: 0.8617 - loss: 0.3721
Epoch 111/1000
7/7            0s 742us/step -
accuracy: 0.8813 - loss: 0.3507
Epoch 112/1000
7/7            0s 714us/step -
accuracy: 0.8524 - loss: 0.3725
Epoch 113/1000
7/7            0s 699us/step -
accuracy: 0.8680 - loss: 0.3510
Epoch 114/1000
7/7            0s 669us/step -
accuracy: 0.8613 - loss: 0.3767
Epoch 115/1000
7/7            0s 899us/step -
accuracy: 0.8661 - loss: 0.3641
Epoch 116/1000
```

```
7/7             0s 871us/step -
accuracy: 0.8775 - loss: 0.3434
Epoch 117/1000
7/7             0s 833us/step -
accuracy: 0.8433 - loss: 0.3898
Epoch 118/1000
7/7             0s 684us/step -
accuracy: 0.8764 - loss: 0.3379
Epoch 119/1000
7/7             0s 748us/step -
accuracy: 0.8553 - loss: 0.3667
Epoch 120/1000
7/7             0s 812us/step -
accuracy: 0.8456 - loss: 0.3697
Epoch 121/1000
7/7             0s 728us/step -
accuracy: 0.8872 - loss: 0.3205
Epoch 122/1000
7/7             0s 973us/step -
accuracy: 0.8414 - loss: 0.3996
Epoch 123/1000
7/7             0s 751us/step -
accuracy: 0.8876 - loss: 0.3161
Epoch 124/1000
7/7             0s 946us/step -
accuracy: 0.8834 - loss: 0.3417
Epoch 125/1000
7/7             0s 698us/step -
accuracy: 0.8591 - loss: 0.3515
Epoch 126/1000
7/7             0s 836us/step -
accuracy: 0.8849 - loss: 0.3325
Epoch 127/1000
7/7             0s 805us/step -
accuracy: 0.8343 - loss: 0.3624
Epoch 128/1000
7/7             0s 788us/step -
accuracy: 0.8462 - loss: 0.3532
Epoch 129/1000
7/7             0s 667us/step -
accuracy: 0.8704 - loss: 0.3544
Epoch 130/1000
7/7             0s 650us/step -
accuracy: 0.8502 - loss: 0.3676
Epoch 131/1000
7/7             0s 747us/step -
accuracy: 0.8354 - loss: 0.3456
Epoch 132/1000
```

```
7/7             0s 853us/step -
accuracy: 0.8561 - loss: 0.3476
Epoch 133/1000
7/7             0s 783us/step -
accuracy: 0.8833 - loss: 0.3141
Epoch 134/1000
7/7             0s 762us/step -
accuracy: 0.8591 - loss: 0.3383
Epoch 135/1000
7/7             0s 724us/step -
accuracy: 0.8615 - loss: 0.3336
Epoch 136/1000
7/7             0s 925us/step -
accuracy: 0.8626 - loss: 0.3340
Epoch 137/1000
7/7             0s 710us/step -
accuracy: 0.8497 - loss: 0.3451
Epoch 138/1000
7/7             0s 723us/step -
accuracy: 0.8535 - loss: 0.3580
Epoch 139/1000
7/7             0s 942us/step -
accuracy: 0.8672 - loss: 0.3284
Epoch 140/1000
7/7             0s 886us/step -
accuracy: 0.8564 - loss: 0.3329
Epoch 141/1000
7/7             0s 726us/step -
accuracy: 0.8523 - loss: 0.3574
Epoch 142/1000
7/7             0s 664us/step -
accuracy: 0.8735 - loss: 0.3312
Epoch 143/1000
7/7             0s 753us/step -
accuracy: 0.8562 - loss: 0.3469
Epoch 144/1000
7/7             0s 701us/step -
accuracy: 0.8519 - loss: 0.3624
Epoch 145/1000
7/7             0s 773us/step -
accuracy: 0.8391 - loss: 0.3614
Epoch 146/1000
7/7             0s 828us/step -
accuracy: 0.8719 - loss: 0.3220
Epoch 147/1000
7/7             0s 698us/step -
accuracy: 0.8599 - loss: 0.3371
Epoch 148/1000
```

```
7/7            0s 763us/step -
accuracy: 0.8588 - loss: 0.3447
Epoch 149/1000
7/7            0s 643us/step -
accuracy: 0.8567 - loss: 0.3425
Epoch 150/1000
7/7            0s 793us/step -
accuracy: 0.8565 - loss: 0.3448
Epoch 151/1000
7/7            0s 761us/step -
accuracy: 0.8684 - loss: 0.3278
Epoch 152/1000
7/7            0s 948us/step -
accuracy: 0.8580 - loss: 0.3405
Epoch 153/1000
7/7            0s 669us/step -
accuracy: 0.8505 - loss: 0.3488
Epoch 154/1000
7/7            0s 658us/step -
accuracy: 0.8361 - loss: 0.3494
Epoch 155/1000
7/7            0s 778us/step -
accuracy: 0.9046 - loss: 0.2847
Epoch 156/1000
7/7            0s 753us/step -
accuracy: 0.8693 - loss: 0.3322
Epoch 157/1000
7/7            0s 661us/step -
accuracy: 0.8601 - loss: 0.3501
Epoch 158/1000
7/7            0s 654us/step -
accuracy: 0.8520 - loss: 0.3422
Epoch 159/1000
7/7            0s 651us/step -
accuracy: 0.8893 - loss: 0.3134
Epoch 160/1000
7/7            0s 705us/step -
accuracy: 0.8690 - loss: 0.3318
Epoch 161/1000
7/7            0s 719us/step -
accuracy: 0.8864 - loss: 0.3132
Epoch 162/1000
7/7            0s 945us/step -
accuracy: 0.8999 - loss: 0.3074
Epoch 163/1000
7/7            0s 2ms/step -
accuracy: 0.8500 - loss: 0.3519
Epoch 164/1000
```

```
7/7            0s 807us/step -
accuracy: 0.8654 - loss: 0.3214
Epoch 165/1000
7/7            0s 872us/step -
accuracy: 0.9058 - loss: 0.2814
Epoch 166/1000
7/7            0s 742us/step -
accuracy: 0.8865 - loss: 0.3032
Epoch 167/1000
7/7            0s 790us/step -
accuracy: 0.8898 - loss: 0.2870
Epoch 168/1000
7/7            0s 743us/step -
accuracy: 0.8731 - loss: 0.3322
Epoch 169/1000
7/7            0s 641us/step -
accuracy: 0.8630 - loss: 0.3218
Epoch 170/1000
7/7            0s 761us/step -
accuracy: 0.8713 - loss: 0.3213
Epoch 171/1000
7/7            0s 923us/step -
accuracy: 0.8676 - loss: 0.3080
Epoch 172/1000
7/7            0s 747us/step -
accuracy: 0.8768 - loss: 0.3174
Epoch 173/1000
7/7            0s 910us/step -
accuracy: 0.8677 - loss: 0.3268
Epoch 174/1000
7/7            0s 957us/step -
accuracy: 0.8850 - loss: 0.3027
Epoch 175/1000
7/7            0s 876us/step -
accuracy: 0.8799 - loss: 0.3185
Epoch 176/1000
7/7            0s 672us/step -
accuracy: 0.8689 - loss: 0.3176
Epoch 177/1000
7/7            0s 803us/step -
accuracy: 0.8831 - loss: 0.3156
Epoch 178/1000
7/7            0s 737us/step -
accuracy: 0.8556 - loss: 0.3370
Epoch 179/1000
7/7            0s 858us/step -
accuracy: 0.8584 - loss: 0.3286
Epoch 180/1000
```

```
7/7            0s 867us/step -
accuracy: 0.8656 - loss: 0.3123
Epoch 181/1000
7/7            0s 897us/step -
accuracy: 0.8702 - loss: 0.3293
Epoch 182/1000
7/7            0s 708us/step -
accuracy: 0.8429 - loss: 0.3338
Epoch 183/1000
7/7            0s 723us/step -
accuracy: 0.8906 - loss: 0.2959
Epoch 184/1000
7/7            0s 810us/step -
accuracy: 0.8589 - loss: 0.3417
Epoch 185/1000
7/7            0s 842us/step -
accuracy: 0.8824 - loss: 0.2983
Epoch 186/1000
7/7            0s 781us/step -
accuracy: 0.8751 - loss: 0.3104
Epoch 187/1000
7/7            0s 713us/step -
accuracy: 0.8689 - loss: 0.3017
Epoch 188/1000
7/7            0s 687us/step -
accuracy: 0.8762 - loss: 0.2925
Epoch 189/1000
7/7            0s 614us/step -
accuracy: 0.8632 - loss: 0.3017
Epoch 190/1000
7/7            0s 778us/step -
accuracy: 0.8731 - loss: 0.3122
Epoch 191/1000
7/7            0s 678us/step -
accuracy: 0.8618 - loss: 0.3126
Epoch 192/1000
7/7            0s 820us/step -
accuracy: 0.8514 - loss: 0.3239
Epoch 193/1000
7/7            0s 902us/step -
accuracy: 0.8639 - loss: 0.3027
Epoch 194/1000
7/7            0s 940us/step -
accuracy: 0.8779 - loss: 0.3182
Epoch 195/1000
7/7            0s 751us/step -
accuracy: 0.8831 - loss: 0.3055
Epoch 196/1000
```

```
7/7            0s 795us/step -
accuracy: 0.8885 - loss: 0.2996
Epoch 197/1000
7/7            0s 700us/step -
accuracy: 0.8568 - loss: 0.3189
Epoch 198/1000
7/7            0s 1ms/step -
accuracy: 0.8682 - loss: 0.3212
Epoch 199/1000
7/7            0s 821us/step -
accuracy: 0.8948 - loss: 0.3011
Epoch 200/1000
7/7            0s 782us/step -
accuracy: 0.8865 - loss: 0.2977
Epoch 201/1000
7/7            0s 831us/step -
accuracy: 0.8720 - loss: 0.3384
Epoch 202/1000
7/7            0s 839us/step -
accuracy: 0.8867 - loss: 0.2852
Epoch 203/1000
7/7            0s 662us/step -
accuracy: 0.9042 - loss: 0.2738
Epoch 204/1000
7/7            0s 668us/step -
accuracy: 0.8910 - loss: 0.3032
Epoch 205/1000
7/7            0s 789us/step -
accuracy: 0.8833 - loss: 0.3066
Epoch 206/1000
7/7            0s 750us/step -
accuracy: 0.8791 - loss: 0.3164
Epoch 207/1000
7/7            0s 849us/step -
accuracy: 0.9058 - loss: 0.2676
Epoch 208/1000
7/7            0s 785us/step -
accuracy: 0.9002 - loss: 0.2871
Epoch 209/1000
7/7            0s 848us/step -
accuracy: 0.8714 - loss: 0.2968
Epoch 210/1000
7/7            0s 629us/step -
accuracy: 0.8980 - loss: 0.2917
Epoch 211/1000
7/7            0s 749us/step -
accuracy: 0.8828 - loss: 0.3003
Epoch 212/1000
```

```
7/7             0s 879us/step -
accuracy: 0.8768 - loss: 0.3143
Epoch 213/1000
7/7             0s 955us/step -
accuracy: 0.8868 - loss: 0.3117
Epoch 214/1000
7/7             0s 734us/step -
accuracy: 0.8812 - loss: 0.2846
Epoch 215/1000
7/7             0s 633us/step -
accuracy: 0.8852 - loss: 0.2972
Epoch 216/1000
7/7             0s 714us/step -
accuracy: 0.8685 - loss: 0.3209
Epoch 217/1000
7/7             0s 767us/step -
accuracy: 0.8862 - loss: 0.2827
Epoch 218/1000
7/7             0s 997us/step -
accuracy: 0.8970 - loss: 0.2835
Epoch 219/1000
7/7             0s 820us/step -
accuracy: 0.8612 - loss: 0.3256
Epoch 220/1000
7/7             0s 799us/step -
accuracy: 0.9039 - loss: 0.2826
Epoch 221/1000
7/7             0s 873us/step -
accuracy: 0.8839 - loss: 0.3057
Epoch 222/1000
7/7             0s 782us/step -
accuracy: 0.8799 - loss: 0.3240
Epoch 223/1000
7/7             0s 915us/step -
accuracy: 0.8892 - loss: 0.3027
Epoch 224/1000
7/7             0s 807us/step -
accuracy: 0.8890 - loss: 0.3146
Epoch 225/1000
7/7             0s 856us/step -
accuracy: 0.8829 - loss: 0.2946
Epoch 226/1000
7/7             0s 793us/step -
accuracy: 0.8953 - loss: 0.2840
Epoch 227/1000
7/7             0s 734us/step -
accuracy: 0.8910 - loss: 0.2959
Epoch 228/1000
```

```
7/7            0s 2ms/step -
accuracy: 0.9180 - loss: 0.2667
Epoch 229/1000
7/7            0s 789us/step -
accuracy: 0.9098 - loss: 0.2630
Epoch 230/1000
7/7            0s 819us/step -
accuracy: 0.8957 - loss: 0.2930
Epoch 231/1000
7/7            0s 836us/step -
accuracy: 0.8911 - loss: 0.2904
Epoch 232/1000
7/7            0s 784us/step -
accuracy: 0.8984 - loss: 0.2869
Epoch 233/1000
7/7            0s 662us/step -
accuracy: 0.8831 - loss: 0.2875
Epoch 234/1000
7/7            0s 742us/step -
accuracy: 0.9060 - loss: 0.2789
Epoch 235/1000
7/7            0s 804us/step -
accuracy: 0.8780 - loss: 0.3151
Epoch 236/1000
7/7            0s 873us/step -
accuracy: 0.9090 - loss: 0.2841
Epoch 237/1000
7/7            0s 841us/step -
accuracy: 0.8816 - loss: 0.3000
Epoch 238/1000
7/7            0s 783us/step -
accuracy: 0.8627 - loss: 0.3166
Epoch 239/1000
7/7            0s 744us/step -
accuracy: 0.9008 - loss: 0.2785
Epoch 240/1000
7/7            0s 796us/step -
accuracy: 0.8985 - loss: 0.2969
Epoch 241/1000
7/7            0s 828us/step -
accuracy: 0.9140 - loss: 0.2679
Epoch 242/1000
7/7            0s 736us/step -
accuracy: 0.8854 - loss: 0.2969
Epoch 243/1000
7/7            0s 792us/step -
accuracy: 0.8847 - loss: 0.2959
Epoch 244/1000
```

```
7/7            0s 708us/step -
accuracy: 0.9159 - loss: 0.2558
Epoch 245/1000
7/7            0s 756us/step -
accuracy: 0.8757 - loss: 0.3255
Epoch 246/1000
7/7            0s 691us/step -
accuracy: 0.8860 - loss: 0.2949
Epoch 247/1000
7/7            0s 796us/step -
accuracy: 0.9118 - loss: 0.2728
Epoch 248/1000
7/7            0s 761us/step -
accuracy: 0.8968 - loss: 0.2959
Epoch 249/1000
7/7            0s 710us/step -
accuracy: 0.9069 - loss: 0.2652
Epoch 250/1000
7/7            0s 763us/step -
accuracy: 0.9159 - loss: 0.2552
Epoch 251/1000
7/7            0s 740us/step -
accuracy: 0.8905 - loss: 0.2900
Epoch 252/1000
7/7            0s 739us/step -
accuracy: 0.9180 - loss: 0.2692
Epoch 253/1000
7/7            0s 878us/step -
accuracy: 0.8951 - loss: 0.2806
Epoch 254/1000
7/7            0s 838us/step -
accuracy: 0.8710 - loss: 0.3106
Epoch 255/1000
7/7            0s 794us/step -
accuracy: 0.8828 - loss: 0.3125
Epoch 256/1000
7/7            0s 859us/step -
accuracy: 0.9078 - loss: 0.2636
Epoch 257/1000
7/7            0s 748us/step -
accuracy: 0.9018 - loss: 0.2628
Epoch 258/1000
7/7            0s 692us/step -
accuracy: 0.9220 - loss: 0.2521
Epoch 259/1000
7/7            0s 1ms/step -
accuracy: 0.9205 - loss: 0.2591
Epoch 260/1000
```

```
7/7            0s 780us/step -
accuracy: 0.8855 - loss: 0.3004
Epoch 261/1000
7/7            0s 713us/step -
accuracy: 0.8734 - loss: 0.3049
Epoch 262/1000
7/7            0s 836us/step -
accuracy: 0.9084 - loss: 0.2450
Epoch 263/1000
7/7            0s 1ms/step -
accuracy: 0.8787 - loss: 0.3028
Epoch 264/1000
7/7            0s 968us/step -
accuracy: 0.9146 - loss: 0.2581
Epoch 265/1000
7/7            0s 662us/step -
accuracy: 0.8712 - loss: 0.3050
Epoch 266/1000
7/7            0s 693us/step -
accuracy: 0.9141 - loss: 0.2550
Epoch 267/1000
7/7            0s 723us/step -
accuracy: 0.9029 - loss: 0.2701
Epoch 268/1000
7/7            0s 934us/step -
accuracy: 0.8802 - loss: 0.2971
Epoch 269/1000
7/7            0s 785us/step -
accuracy: 0.8738 - loss: 0.2992
Epoch 270/1000
7/7            0s 764us/step -
accuracy: 0.9138 - loss: 0.2667
Epoch 271/1000
7/7            0s 699us/step -
accuracy: 0.9097 - loss: 0.2717
Epoch 272/1000
7/7            0s 871us/step -
accuracy: 0.8810 - loss: 0.3046
Epoch 273/1000
7/7            0s 715us/step -
accuracy: 0.8974 - loss: 0.2979
Epoch 274/1000
7/7            0s 803us/step -
accuracy: 0.8827 - loss: 0.2904
Epoch 275/1000
7/7            0s 871us/step -
accuracy: 0.8970 - loss: 0.2878
Epoch 276/1000
```

```
7/7            0s 862us/step -
accuracy: 0.8960 - loss: 0.2869
Epoch 277/1000
7/7            0s 829us/step -
accuracy: 0.8860 - loss: 0.2824
Epoch 278/1000
7/7            0s 750us/step -
accuracy: 0.8901 - loss: 0.2895
Epoch 279/1000
7/7            0s 697us/step -
accuracy: 0.8948 - loss: 0.2799
Epoch 280/1000
7/7            0s 693us/step -
accuracy: 0.9182 - loss: 0.2595
Epoch 281/1000
7/7            0s 800us/step -
accuracy: 0.8901 - loss: 0.2865
Epoch 282/1000
7/7            0s 915us/step -
accuracy: 0.9095 - loss: 0.2568
Epoch 283/1000
7/7            0s 732us/step -
accuracy: 0.8848 - loss: 0.2891
Epoch 284/1000
7/7            0s 779us/step -
accuracy: 0.9054 - loss: 0.2781
Epoch 285/1000
7/7            0s 720us/step -
accuracy: 0.9259 - loss: 0.2366
Epoch 286/1000
7/7            0s 712us/step -
accuracy: 0.9073 - loss: 0.2685
Epoch 287/1000
7/7            0s 1ms/step -
accuracy: 0.8989 - loss: 0.2723
Epoch 288/1000
7/7            0s 917us/step -
accuracy: 0.8836 - loss: 0.2849
Epoch 289/1000
7/7            0s 681us/step -
accuracy: 0.8961 - loss: 0.2594
Epoch 290/1000
7/7            0s 752us/step -
accuracy: 0.8981 - loss: 0.2682
Epoch 291/1000
7/7            0s 839us/step -
accuracy: 0.9160 - loss: 0.2534
Epoch 292/1000
```

```
7/7            0s 823us/step -
accuracy: 0.8911 - loss: 0.2733
Epoch 293/1000
7/7            0s 804us/step -
accuracy: 0.8968 - loss: 0.2747
Epoch 294/1000
7/7            0s 1ms/step -
accuracy: 0.8707 - loss: 0.2980
Epoch 295/1000
7/7            0s 866us/step -
accuracy: 0.8872 - loss: 0.2750
Epoch 296/1000
7/7            0s 722us/step -
accuracy: 0.8832 - loss: 0.2920
Epoch 297/1000
7/7            0s 959us/step -
accuracy: 0.9083 - loss: 0.2611
Epoch 298/1000
7/7            0s 741us/step -
accuracy: 0.8786 - loss: 0.3013
Epoch 299/1000
7/7            0s 723us/step -
accuracy: 0.9246 - loss: 0.2425
Epoch 300/1000
7/7            0s 800us/step -
accuracy: 0.8901 - loss: 0.2872
Epoch 301/1000
7/7            0s 937us/step -
accuracy: 0.9123 - loss: 0.2594
Epoch 302/1000
7/7            0s 856us/step -
accuracy: 0.9024 - loss: 0.2693
Epoch 303/1000
7/7            0s 762us/step -
accuracy: 0.8834 - loss: 0.2918
Epoch 304/1000
7/7            0s 709us/step -
accuracy: 0.9074 - loss: 0.2728
Epoch 305/1000
7/7            0s 694us/step -
accuracy: 0.8895 - loss: 0.3077
Epoch 306/1000
7/7            0s 773us/step -
accuracy: 0.9149 - loss: 0.2627
Epoch 307/1000
7/7            0s 801us/step -
accuracy: 0.9045 - loss: 0.2637
Epoch 308/1000
```

```
7/7            0s 797us/step -
accuracy: 0.8995 - loss: 0.2723
Epoch 309/1000
7/7            0s 761us/step -
accuracy: 0.8773 - loss: 0.3029
Epoch 310/1000
7/7            0s 774us/step -
accuracy: 0.8920 - loss: 0.2654
Epoch 311/1000
7/7            0s 743us/step -
accuracy: 0.8726 - loss: 0.3036
Epoch 312/1000
7/7            0s 711us/step -
accuracy: 0.9002 - loss: 0.2673
Epoch 313/1000
7/7            0s 1ms/step -
accuracy: 0.9119 - loss: 0.2763
Epoch 314/1000
7/7            0s 1ms/step -
accuracy: 0.9031 - loss: 0.2674
Epoch 315/1000
7/7            0s 851us/step -
accuracy: 0.9020 - loss: 0.2750
Epoch 316/1000
7/7            0s 729us/step -
accuracy: 0.9047 - loss: 0.2626
Epoch 317/1000
7/7            0s 812us/step -
accuracy: 0.8950 - loss: 0.2746
Epoch 318/1000
7/7            0s 688us/step -
accuracy: 0.9129 - loss: 0.2411
Epoch 319/1000
7/7            0s 700us/step -
accuracy: 0.8739 - loss: 0.2817
Epoch 320/1000
7/7            0s 723us/step -
accuracy: 0.8980 - loss: 0.2588
Epoch 321/1000
7/7            0s 823us/step -
accuracy: 0.8963 - loss: 0.2708
Epoch 322/1000
7/7            0s 853us/step -
accuracy: 0.8835 - loss: 0.2817
Epoch 323/1000
7/7            0s 682us/step -
accuracy: 0.8938 - loss: 0.2873
Epoch 324/1000
```

```
7/7            0s 769us/step -
accuracy: 0.9170 - loss: 0.2493
Epoch 325/1000
7/7            0s 673us/step -
accuracy: 0.8901 - loss: 0.2852
Epoch 326/1000
7/7            0s 833us/step -
accuracy: 0.9269 - loss: 0.2269
Epoch 327/1000
7/7            0s 685us/step -
accuracy: 0.8972 - loss: 0.2594
Epoch 328/1000
7/7            0s 820us/step -
accuracy: 0.8832 - loss: 0.2627
Epoch 329/1000
7/7            0s 826us/step -
accuracy: 0.8900 - loss: 0.2913
Epoch 330/1000
7/7            0s 734us/step -
accuracy: 0.8825 - loss: 0.2875
Epoch 331/1000
7/7            0s 742us/step -
accuracy: 0.8875 - loss: 0.2609
Epoch 332/1000
7/7            0s 637us/step -
accuracy: 0.8958 - loss: 0.2762
Epoch 333/1000
7/7            0s 650us/step -
accuracy: 0.8955 - loss: 0.2730
Epoch 334/1000
7/7            0s 852us/step -
accuracy: 0.9019 - loss: 0.2620
Epoch 335/1000
7/7            0s 775us/step -
accuracy: 0.8991 - loss: 0.2752
Epoch 336/1000
7/7            0s 793us/step -
accuracy: 0.9193 - loss: 0.2359
Epoch 337/1000
7/7            0s 696us/step -
accuracy: 0.8905 - loss: 0.2565
Epoch 338/1000
7/7            0s 1ms/step -
accuracy: 0.9140 - loss: 0.2251
Epoch 339/1000
7/7            0s 724us/step -
accuracy: 0.9107 - loss: 0.2536
Epoch 340/1000
```

```
7/7            0s 783us/step -
accuracy: 0.9114 - loss: 0.2325
Epoch 341/1000
7/7            0s 823us/step -
accuracy: 0.9156 - loss: 0.2455
Epoch 342/1000
7/7            0s 775us/step -
accuracy: 0.9094 - loss: 0.2670
Epoch 343/1000
7/7            0s 740us/step -
accuracy: 0.9174 - loss: 0.2341
Epoch 344/1000
7/7            0s 791us/step -
accuracy: 0.8961 - loss: 0.2793
Epoch 345/1000
7/7            0s 901us/step -
accuracy: 0.9143 - loss: 0.2395
Epoch 346/1000
7/7            0s 1ms/step -
accuracy: 0.9147 - loss: 0.2334
Epoch 347/1000
7/7            0s 764us/step -
accuracy: 0.9067 - loss: 0.2606
Epoch 348/1000
7/7            0s 685us/step -
accuracy: 0.9056 - loss: 0.2647
Epoch 349/1000
7/7            0s 828us/step -
accuracy: 0.9306 - loss: 0.2079
Epoch 350/1000
7/7            0s 807us/step -
accuracy: 0.8668 - loss: 0.2846
Epoch 351/1000
7/7            0s 904us/step -
accuracy: 0.8819 - loss: 0.2740
Epoch 352/1000
7/7            0s 806us/step -
accuracy: 0.8826 - loss: 0.2832
Epoch 353/1000
7/7            0s 732us/step -
accuracy: 0.8771 - loss: 0.2958
Epoch 354/1000
7/7            0s 663us/step -
accuracy: 0.8892 - loss: 0.2586
Epoch 355/1000
7/7            0s 822us/step -
accuracy: 0.8938 - loss: 0.2624
Epoch 356/1000
```

```
7/7            0s 837us/step -
accuracy: 0.8978 - loss: 0.2484
Epoch 357/1000
7/7            0s 735us/step -
accuracy: 0.8994 - loss: 0.2453
Epoch 358/1000
7/7            0s 734us/step -
accuracy: 0.8997 - loss: 0.2710
Epoch 359/1000
7/7            0s 829us/step -
accuracy: 0.8946 - loss: 0.2684
Epoch 360/1000
7/7            0s 1ms/step -
accuracy: 0.9081 - loss: 0.2504
Epoch 361/1000
7/7            0s 1ms/step -
accuracy: 0.9166 - loss: 0.2281
Epoch 362/1000
7/7            0s 820us/step -
accuracy: 0.8915 - loss: 0.2809
Epoch 363/1000
7/7            0s 844us/step -
accuracy: 0.8931 - loss: 0.2530
Epoch 364/1000
7/7            0s 759us/step -
accuracy: 0.9385 - loss: 0.2149
Epoch 365/1000
7/7            0s 715us/step -
accuracy: 0.8954 - loss: 0.2482
Epoch 366/1000
7/7            0s 715us/step -
accuracy: 0.8930 - loss: 0.2746
Epoch 367/1000
7/7            0s 827us/step -
accuracy: 0.9172 - loss: 0.2289
Epoch 368/1000
7/7            0s 855us/step -
accuracy: 0.9131 - loss: 0.2221
Epoch 369/1000
7/7            0s 813us/step -
accuracy: 0.9209 - loss: 0.2374
Epoch 370/1000
7/7            0s 839us/step -
accuracy: 0.9098 - loss: 0.2383
Epoch 371/1000
7/7            0s 709us/step -
accuracy: 0.9323 - loss: 0.2222
Epoch 372/1000
```

```
7/7            0s 701us/step -
accuracy: 0.9145 - loss: 0.2364
Epoch 373/1000
7/7            0s 751us/step -
accuracy: 0.8840 - loss: 0.2883
Epoch 374/1000
7/7            0s 630us/step -
accuracy: 0.9150 - loss: 0.2485
Epoch 375/1000
7/7            0s 709us/step -
accuracy: 0.9075 - loss: 0.2521
Epoch 376/1000
7/7            0s 841us/step -
accuracy: 0.9009 - loss: 0.2620
Epoch 377/1000
7/7            0s 739us/step -
accuracy: 0.8760 - loss: 0.2984
Epoch 378/1000
7/7            0s 710us/step -
accuracy: 0.9369 - loss: 0.2035
Epoch 379/1000
7/7            0s 874us/step -
accuracy: 0.9209 - loss: 0.2250
Epoch 380/1000
7/7            0s 762us/step -
accuracy: 0.9102 - loss: 0.2362
Epoch 381/1000
7/7            0s 757us/step -
accuracy: 0.9163 - loss: 0.2411
Epoch 382/1000
7/7            0s 685us/step -
accuracy: 0.9061 - loss: 0.2499
Epoch 383/1000
7/7            0s 695us/step -
accuracy: 0.9187 - loss: 0.2302
Epoch 384/1000
7/7            0s 853us/step -
accuracy: 0.9105 - loss: 0.2478
Epoch 385/1000
7/7            0s 824us/step -
accuracy: 0.9181 - loss: 0.2225
Epoch 386/1000
7/7            0s 854us/step -
accuracy: 0.8909 - loss: 0.2571
Epoch 387/1000
7/7            0s 734us/step -
accuracy: 0.9003 - loss: 0.2476
Epoch 388/1000
```

```
7/7            0s 881us/step -
accuracy: 0.8633 - loss: 0.2965
Epoch 389/1000
7/7            0s 984us/step -
accuracy: 0.9303 - loss: 0.2188
Epoch 390/1000
7/7            0s 1ms/step -
accuracy: 0.9112 - loss: 0.2288
Epoch 391/1000
7/7            0s 1ms/step -
accuracy: 0.9282 - loss: 0.2364
Epoch 392/1000
7/7            0s 829us/step -
accuracy: 0.8980 - loss: 0.2514
Epoch 393/1000
7/7            0s 808us/step -
accuracy: 0.9230 - loss: 0.2285
Epoch 394/1000
7/7            0s 754us/step -
accuracy: 0.9017 - loss: 0.2624
Epoch 395/1000
7/7            0s 664us/step -
accuracy: 0.9166 - loss: 0.2349
Epoch 396/1000
7/7            0s 676us/step -
accuracy: 0.9124 - loss: 0.2353
Epoch 397/1000
7/7            0s 689us/step -
accuracy: 0.9015 - loss: 0.2426
Epoch 398/1000
7/7            0s 765us/step -
accuracy: 0.9109 - loss: 0.2355
Epoch 399/1000
7/7            0s 753us/step -
accuracy: 0.9146 - loss: 0.2379
Epoch 400/1000
7/7            0s 944us/step -
accuracy: 0.9131 - loss: 0.2320
Epoch 401/1000
7/7            0s 745us/step -
accuracy: 0.8917 - loss: 0.2750
Epoch 402/1000
7/7            0s 707us/step -
accuracy: 0.8961 - loss: 0.2657
Epoch 403/1000
7/7            0s 678us/step -
accuracy: 0.8836 - loss: 0.2786
Epoch 404/1000
```

```
7/7            0s 709us/step -
accuracy: 0.9178 - loss: 0.2153
Epoch 405/1000
7/7            0s 653us/step -
accuracy: 0.9073 - loss: 0.2334
Epoch 406/1000
7/7            0s 800us/step -
accuracy: 0.9107 - loss: 0.2565
Epoch 407/1000
7/7            0s 855us/step -
accuracy: 0.9135 - loss: 0.2241
Epoch 408/1000
7/7            0s 954us/step -
accuracy: 0.9174 - loss: 0.2408
Epoch 409/1000
7/7            0s 1ms/step -
accuracy: 0.8995 - loss: 0.2620
Epoch 410/1000
7/7            0s 1ms/step -
accuracy: 0.8991 - loss: 0.2305
Epoch 411/1000
7/7            0s 773us/step -
accuracy: 0.8993 - loss: 0.2585
Epoch 412/1000
7/7            0s 718us/step -
accuracy: 0.9051 - loss: 0.2515
Epoch 413/1000
7/7            0s 676us/step -
accuracy: 0.9323 - loss: 0.2199
Epoch 414/1000
7/7            0s 884us/step -
accuracy: 0.9160 - loss: 0.2399
Epoch 415/1000
7/7            0s 810us/step -
accuracy: 0.8997 - loss: 0.2701
Epoch 416/1000
7/7            0s 822us/step -
accuracy: 0.9212 - loss: 0.2329
Epoch 417/1000
7/7            0s 749us/step -
accuracy: 0.9109 - loss: 0.2409
Epoch 418/1000
7/7            0s 757us/step -
accuracy: 0.9032 - loss: 0.2462
Epoch 419/1000
7/7            0s 722us/step -
accuracy: 0.9109 - loss: 0.2632
Epoch 420/1000
```

```
7/7            0s 724us/step -
accuracy: 0.8981 - loss: 0.2343
Epoch 421/1000
7/7            0s 911us/step -
accuracy: 0.9070 - loss: 0.2481
Epoch 422/1000
7/7            0s 904us/step -
accuracy: 0.9001 - loss: 0.2612
Epoch 423/1000
7/7            0s 781us/step -
accuracy: 0.9088 - loss: 0.2305
Epoch 424/1000
7/7            0s 792us/step -
accuracy: 0.9315 - loss: 0.2225
Epoch 425/1000
7/7            0s 642us/step -
accuracy: 0.9120 - loss: 0.2187
Epoch 426/1000
7/7            0s 748us/step -
accuracy: 0.9086 - loss: 0.2495
Epoch 427/1000
7/7            0s 662us/step -
accuracy: 0.8990 - loss: 0.2442
Epoch 428/1000
7/7            0s 686us/step -
accuracy: 0.9092 - loss: 0.2448
Epoch 429/1000
7/7            0s 1ms/step -
accuracy: 0.9103 - loss: 0.2448
Epoch 430/1000
7/7            0s 753us/step -
accuracy: 0.9287 - loss: 0.2244
Epoch 431/1000
7/7            0s 724us/step -
accuracy: 0.9328 - loss: 0.2135
Epoch 432/1000
7/7            0s 662us/step -
accuracy: 0.9349 - loss: 0.2033
Epoch 433/1000
7/7            0s 1ms/step -
accuracy: 0.9182 - loss: 0.2310
Epoch 434/1000
7/7            0s 932us/step -
accuracy: 0.9217 - loss: 0.2309
Epoch 435/1000
7/7            0s 751us/step -
accuracy: 0.9031 - loss: 0.2426
Epoch 436/1000
```

```
7/7            0s 765us/step -
accuracy: 0.9082 - loss: 0.2488
Epoch 437/1000
7/7            0s 997us/step -
accuracy: 0.9272 - loss: 0.2148
Epoch 438/1000
7/7            0s 817us/step -
accuracy: 0.9217 - loss: 0.2260
Epoch 439/1000
7/7            0s 731us/step -
accuracy: 0.8879 - loss: 0.2656
Epoch 440/1000
7/7            0s 868us/step -
accuracy: 0.9189 - loss: 0.2090
Epoch 441/1000
7/7            0s 738us/step -
accuracy: 0.9103 - loss: 0.2458
Epoch 442/1000
7/7            0s 712us/step -
accuracy: 0.9336 - loss: 0.2066
Epoch 443/1000
7/7            0s 873us/step -
accuracy: 0.8995 - loss: 0.2349
Epoch 444/1000
7/7            0s 820us/step -
accuracy: 0.9135 - loss: 0.2358
Epoch 445/1000
7/7            0s 724us/step -
accuracy: 0.9070 - loss: 0.2250
Epoch 446/1000
7/7            0s 668us/step -
accuracy: 0.8978 - loss: 0.2599
Epoch 447/1000
7/7            0s 817us/step -
accuracy: 0.9121 - loss: 0.2273
Epoch 448/1000
7/7            0s 843us/step -
accuracy: 0.9312 - loss: 0.2177
Epoch 449/1000
7/7            0s 826us/step -
accuracy: 0.9175 - loss: 0.2188
Epoch 450/1000
7/7            0s 910us/step -
accuracy: 0.8962 - loss: 0.2430
Epoch 451/1000
7/7            0s 728us/step -
accuracy: 0.9017 - loss: 0.2258
Epoch 452/1000
```

```
7/7            0s 711us/step -
accuracy: 0.9223 - loss: 0.2300
Epoch 453/1000
7/7            0s 721us/step -
accuracy: 0.9126 - loss: 0.2188
Epoch 454/1000
7/7            0s 789us/step -
accuracy: 0.8963 - loss: 0.2516
Epoch 455/1000
7/7            0s 789us/step -
accuracy: 0.8928 - loss: 0.2645
Epoch 456/1000
7/7            0s 774us/step -
accuracy: 0.9094 - loss: 0.2310
Epoch 457/1000
7/7            0s 899us/step -
accuracy: 0.8999 - loss: 0.2561
Epoch 458/1000
7/7            0s 712us/step -
accuracy: 0.9126 - loss: 0.2292
Epoch 459/1000
7/7            0s 1ms/step -
accuracy: 0.8771 - loss: 0.2747
Epoch 460/1000
7/7            0s 1ms/step -
accuracy: 0.9075 - loss: 0.2352
Epoch 461/1000
7/7            0s 884us/step -
accuracy: 0.9073 - loss: 0.2461
Epoch 462/1000
7/7            0s 872us/step -
accuracy: 0.8806 - loss: 0.2472
Epoch 463/1000
7/7            0s 733us/step -
accuracy: 0.9143 - loss: 0.2458
Epoch 464/1000
7/7            0s 680us/step -
accuracy: 0.8879 - loss: 0.2432
Epoch 465/1000
7/7            0s 679us/step -
accuracy: 0.9170 - loss: 0.2118
Epoch 466/1000
7/7            0s 727us/step -
accuracy: 0.9168 - loss: 0.2264
Epoch 467/1000
7/7            0s 818us/step -
accuracy: 0.8794 - loss: 0.2698
Epoch 468/1000
```

```
7/7            0s 780us/step -
accuracy: 0.9072 - loss: 0.2370
Epoch 469/1000
7/7            0s 858us/step -
accuracy: 0.9108 - loss: 0.2377
Epoch 470/1000
7/7            0s 746us/step -
accuracy: 0.9251 - loss: 0.2235
Epoch 471/1000
7/7            0s 843us/step -
accuracy: 0.9010 - loss: 0.2457
Epoch 472/1000
7/7            0s 733us/step -
accuracy: 0.9133 - loss: 0.2211
Epoch 473/1000
7/7            0s 656us/step -
accuracy: 0.9031 - loss: 0.2388
Epoch 474/1000
7/7            0s 760us/step -
accuracy: 0.9236 - loss: 0.2257
Epoch 475/1000
7/7            0s 902us/step -
accuracy: 0.8894 - loss: 0.2613
Epoch 476/1000
7/7            0s 825us/step -
accuracy: 0.8973 - loss: 0.2443
Epoch 477/1000
7/7            0s 727us/step -
accuracy: 0.9092 - loss: 0.2434
Epoch 478/1000
7/7            0s 729us/step -
accuracy: 0.9110 - loss: 0.2348
Epoch 479/1000
7/7            0s 717us/step -
accuracy: 0.9053 - loss: 0.2296
Epoch 480/1000
7/7            0s 1ms/step -
accuracy: 0.9086 - loss: 0.2141
Epoch 481/1000
7/7            0s 2ms/step -
accuracy: 0.9059 - loss: 0.2219
Epoch 482/1000
7/7            0s 715us/step -
accuracy: 0.9357 - loss: 0.1945
Epoch 483/1000
7/7            0s 956us/step -
accuracy: 0.9254 - loss: 0.2152
Epoch 484/1000
```

```
7/7             0s 786us/step -
accuracy: 0.8999 - loss: 0.2401
Epoch 485/1000
7/7             0s 796us/step -
accuracy: 0.9307 - loss: 0.2024
Epoch 486/1000
7/7             0s 787us/step -
accuracy: 0.9267 - loss: 0.2137
Epoch 487/1000
7/7             0s 764us/step -
accuracy: 0.9137 - loss: 0.2203
Epoch 488/1000
7/7             0s 936us/step -
accuracy: 0.9008 - loss: 0.2355
Epoch 489/1000
7/7             0s 867us/step -
accuracy: 0.9102 - loss: 0.2342
Epoch 490/1000
7/7             0s 751us/step -
accuracy: 0.9111 - loss: 0.2183
Epoch 491/1000
7/7             0s 811us/step -
accuracy: 0.9191 - loss: 0.2182
Epoch 492/1000
7/7             0s 650us/step -
accuracy: 0.9044 - loss: 0.2247
Epoch 493/1000
7/7             0s 781us/step -
accuracy: 0.8956 - loss: 0.2535
Epoch 494/1000
7/7             0s 681us/step -
accuracy: 0.9120 - loss: 0.2376
Epoch 495/1000
7/7             0s 935us/step -
accuracy: 0.9261 - loss: 0.1985
Epoch 496/1000
7/7             0s 855us/step -
accuracy: 0.9282 - loss: 0.2138
Epoch 497/1000
7/7             0s 823us/step -
accuracy: 0.9084 - loss: 0.2426
Epoch 498/1000
7/7             0s 753us/step -
accuracy: 0.9080 - loss: 0.2146
Epoch 499/1000
7/7             0s 676us/step -
accuracy: 0.8971 - loss: 0.2297
Epoch 500/1000
```

```
7/7             0s 924us/step -
accuracy: 0.9032 - loss: 0.2177
Epoch 501/1000
7/7             0s 826us/step -
accuracy: 0.9092 - loss: 0.2123
Epoch 502/1000
7/7             0s 875us/step -
accuracy: 0.9022 - loss: 0.2364
Epoch 503/1000
7/7             0s 1ms/step -
accuracy: 0.9053 - loss: 0.2154
Epoch 504/1000
7/7             0s 1ms/step -
accuracy: 0.8939 - loss: 0.2311
Epoch 505/1000
7/7             0s 820us/step -
accuracy: 0.9217 - loss: 0.2059
Epoch 506/1000
7/7             0s 898us/step -
accuracy: 0.9134 - loss: 0.2252
Epoch 507/1000
7/7             0s 692us/step -
accuracy: 0.8988 - loss: 0.2180
Epoch 508/1000
7/7             0s 777us/step -
accuracy: 0.8988 - loss: 0.2673
Epoch 509/1000
7/7             0s 666us/step -
accuracy: 0.9191 - loss: 0.2057
Epoch 510/1000
7/7             0s 899us/step -
accuracy: 0.8849 - loss: 0.2428
Epoch 511/1000
7/7             0s 777us/step -
accuracy: 0.8954 - loss: 0.2393
Epoch 512/1000
7/7             0s 1ms/step -
accuracy: 0.9115 - loss: 0.2184
Epoch 513/1000
7/7             0s 902us/step -
accuracy: 0.9317 - loss: 0.2024
Epoch 514/1000
7/7             0s 917us/step -
accuracy: 0.9066 - loss: 0.2211
Epoch 515/1000
7/7             0s 774us/step -
accuracy: 0.9022 - loss: 0.2164
Epoch 516/1000
```

```
7/7             0s 904us/step -
accuracy: 0.9215 - loss: 0.2088
Epoch 517/1000
7/7             0s 1ms/step -
accuracy: 0.9156 - loss: 0.2250
Epoch 518/1000
7/7             0s 834us/step -
accuracy: 0.9080 - loss: 0.2227
Epoch 519/1000
7/7             0s 838us/step -
accuracy: 0.9214 - loss: 0.2271
Epoch 520/1000
7/7             0s 1ms/step -
accuracy: 0.9167 - loss: 0.2112
Epoch 521/1000
7/7             0s 903us/step -
accuracy: 0.9363 - loss: 0.1972
Epoch 522/1000
7/7             0s 884us/step -
accuracy: 0.9275 - loss: 0.2026
Epoch 523/1000
7/7             0s 684us/step -
accuracy: 0.9171 - loss: 0.2249
Epoch 524/1000
7/7             0s 839us/step -
accuracy: 0.9205 - loss: 0.2284
Epoch 525/1000
7/7             0s 795us/step -
accuracy: 0.9333 - loss: 0.1995
Epoch 526/1000
7/7             0s 1ms/step -
accuracy: 0.9052 - loss: 0.2436
Epoch 527/1000
7/7             0s 904us/step -
accuracy: 0.9274 - loss: 0.2119
Epoch 528/1000
7/7             0s 986us/step -
accuracy: 0.9243 - loss: 0.2040
Epoch 529/1000
7/7             0s 674us/step -
accuracy: 0.9318 - loss: 0.2062
Epoch 530/1000
7/7             0s 767us/step -
accuracy: 0.9175 - loss: 0.2270
Epoch 531/1000
7/7             0s 712us/step -
accuracy: 0.9141 - loss: 0.2245
Epoch 532/1000
```

```
7/7             0s 1ms/step -
accuracy: 0.9100 - loss: 0.2481
Epoch 533/1000
7/7             0s 1ms/step -
accuracy: 0.9501 - loss: 0.1902
Epoch 534/1000
7/7             0s 820us/step -
accuracy: 0.9236 - loss: 0.2236
Epoch 535/1000
7/7             0s 775us/step -
accuracy: 0.9438 - loss: 0.1841
Epoch 536/1000
7/7             0s 758us/step -
accuracy: 0.9450 - loss: 0.1950
Epoch 537/1000
7/7             0s 767us/step -
accuracy: 0.8984 - loss: 0.2401
Epoch 538/1000
7/7             0s 924us/step -
accuracy: 0.9042 - loss: 0.2416
Epoch 539/1000
7/7             0s 979us/step -
accuracy: 0.9383 - loss: 0.1904
Epoch 540/1000
7/7             0s 684us/step -
accuracy: 0.9276 - loss: 0.2124
Epoch 541/1000
7/7             0s 766us/step -
accuracy: 0.9242 - loss: 0.2120
Epoch 542/1000
7/7             0s 770us/step -
accuracy: 0.9180 - loss: 0.2188
Epoch 543/1000
7/7             0s 854us/step -
accuracy: 0.9103 - loss: 0.2292
Epoch 544/1000
7/7             0s 731us/step -
accuracy: 0.9242 - loss: 0.1982
Epoch 545/1000
7/7             0s 957us/step -
accuracy: 0.8947 - loss: 0.2421
Epoch 546/1000
7/7             0s 776us/step -
accuracy: 0.9296 - loss: 0.2076
Epoch 547/1000
7/7             0s 915us/step -
accuracy: 0.9248 - loss: 0.2039
Epoch 548/1000
```

```
7/7            0s 864us/step -
accuracy: 0.9318 - loss: 0.2002
Epoch 549/1000
7/7            0s 671us/step -
accuracy: 0.9411 - loss: 0.2062
Epoch 550/1000
7/7            0s 799us/step -
accuracy: 0.9288 - loss: 0.2139
Epoch 551/1000
7/7            0s 835us/step -
accuracy: 0.9142 - loss: 0.2150
Epoch 552/1000
7/7            0s 1ms/step -
accuracy: 0.9204 - loss: 0.2041
Epoch 553/1000
7/7            0s 1ms/step -
accuracy: 0.9322 - loss: 0.2177
Epoch 554/1000
7/7            0s 969us/step -
accuracy: 0.9217 - loss: 0.2149
Epoch 555/1000
7/7            0s 997us/step -
accuracy: 0.8906 - loss: 0.2632
Epoch 556/1000
7/7            0s 739us/step -
accuracy: 0.9082 - loss: 0.2250
Epoch 557/1000
7/7            0s 779us/step -
accuracy: 0.9208 - loss: 0.2303
Epoch 558/1000
7/7            0s 737us/step -
accuracy: 0.9317 - loss: 0.2103
Epoch 559/1000
7/7            0s 843us/step -
accuracy: 0.9134 - loss: 0.2163
Epoch 560/1000
7/7            0s 776us/step -
accuracy: 0.9433 - loss: 0.1852
Epoch 561/1000
7/7            0s 777us/step -
accuracy: 0.9389 - loss: 0.1911
Epoch 562/1000
7/7            0s 672us/step -
accuracy: 0.9137 - loss: 0.2180
Epoch 563/1000
7/7            0s 836us/step -
accuracy: 0.9048 - loss: 0.2234
Epoch 564/1000
```

```
7/7            0s 789us/step -
accuracy: 0.9283 - loss: 0.2115
Epoch 565/1000
7/7            0s 1ms/step -
accuracy: 0.9005 - loss: 0.2497
Epoch 566/1000
7/7            0s 731us/step -
accuracy: 0.9109 - loss: 0.2068
Epoch 567/1000
7/7            0s 682us/step -
accuracy: 0.9236 - loss: 0.1974
Epoch 568/1000
7/7            0s 723us/step -
accuracy: 0.9416 - loss: 0.1896
Epoch 569/1000
7/7            0s 762us/step -
accuracy: 0.9240 - loss: 0.1950
Epoch 570/1000
7/7            0s 776us/step -
accuracy: 0.9226 - loss: 0.2115
Epoch 571/1000
7/7            0s 743us/step -
accuracy: 0.9296 - loss: 0.2104
Epoch 572/1000
7/7            0s 642us/step -
accuracy: 0.8994 - loss: 0.2306
Epoch 573/1000
7/7            0s 720us/step -
accuracy: 0.9324 - loss: 0.2024
Epoch 574/1000
7/7            0s 923us/step -
accuracy: 0.9283 - loss: 0.1942
Epoch 575/1000
7/7            0s 773us/step -
accuracy: 0.9256 - loss: 0.2087
Epoch 576/1000
7/7            0s 710us/step -
accuracy: 0.9339 - loss: 0.1858
Epoch 577/1000
7/7            0s 740us/step -
accuracy: 0.9309 - loss: 0.2103
Epoch 578/1000
7/7            0s 1ms/step -
accuracy: 0.9311 - loss: 0.2012
Epoch 579/1000
7/7            0s 756us/step -
accuracy: 0.9499 - loss: 0.1875
Epoch 580/1000
```

```
7/7            0s 707us/step -
accuracy: 0.9154 - loss: 0.2077
Epoch 581/1000
7/7            0s 643us/step -
accuracy: 0.9189 - loss: 0.2184
Epoch 582/1000
7/7            0s 625us/step -
accuracy: 0.9363 - loss: 0.2015
Epoch 583/1000
7/7            0s 875us/step -
accuracy: 0.9290 - loss: 0.2162
Epoch 584/1000
7/7            0s 807us/step -
accuracy: 0.9384 - loss: 0.1963
Epoch 585/1000
7/7            0s 780us/step -
accuracy: 0.9232 - loss: 0.2235
Epoch 586/1000
7/7            0s 777us/step -
accuracy: 0.9195 - loss: 0.2246
Epoch 587/1000
7/7            0s 677us/step -
accuracy: 0.9188 - loss: 0.2105
Epoch 588/1000
7/7            0s 730us/step -
accuracy: 0.9266 - loss: 0.1913
Epoch 589/1000
7/7            0s 836us/step -
accuracy: 0.9429 - loss: 0.1862
Epoch 590/1000
7/7            0s 924us/step -
accuracy: 0.9017 - loss: 0.2336
Epoch 591/1000
7/7            0s 966us/step -
accuracy: 0.9244 - loss: 0.2032
Epoch 592/1000
7/7            0s 1ms/step -
accuracy: 0.9419 - loss: 0.1770
Epoch 593/1000
7/7            0s 828us/step -
accuracy: 0.9368 - loss: 0.1970
Epoch 594/1000
7/7            0s 922us/step -
accuracy: 0.9269 - loss: 0.1891
Epoch 595/1000
7/7            0s 842us/step -
accuracy: 0.9430 - loss: 0.1781
Epoch 596/1000
```

```
7/7            0s 798us/step -
accuracy: 0.9028 - loss: 0.2316
Epoch 597/1000
7/7            0s 784us/step -
accuracy: 0.9342 - loss: 0.2013
Epoch 598/1000
7/7            0s 805us/step -
accuracy: 0.9307 - loss: 0.1816
Epoch 599/1000
7/7            0s 837us/step -
accuracy: 0.9464 - loss: 0.1830
Epoch 600/1000
7/7            0s 892us/step -
accuracy: 0.9075 - loss: 0.2108
Epoch 601/1000
7/7            0s 725us/step -
accuracy: 0.9358 - loss: 0.1915
Epoch 602/1000
7/7            0s 1ms/step -
accuracy: 0.9263 - loss: 0.1997
Epoch 603/1000
7/7            0s 1ms/step -
accuracy: 0.9391 - loss: 0.1833
Epoch 604/1000
7/7            0s 743us/step -
accuracy: 0.9464 - loss: 0.1851
Epoch 605/1000
7/7            0s 763us/step -
accuracy: 0.9386 - loss: 0.1813
Epoch 606/1000
7/7            0s 791us/step -
accuracy: 0.9332 - loss: 0.1878
Epoch 607/1000
7/7            0s 815us/step -
accuracy: 0.9444 - loss: 0.1881
Epoch 608/1000
7/7            0s 675us/step -
accuracy: 0.9239 - loss: 0.1976
Epoch 609/1000
7/7            0s 817us/step -
accuracy: 0.9332 - loss: 0.2032
Epoch 610/1000
7/7            0s 964us/step -
accuracy: 0.9243 - loss: 0.1881
Epoch 611/1000
7/7            0s 854us/step -
accuracy: 0.9414 - loss: 0.1808
Epoch 612/1000
```

```
7/7            0s 906us/step -
accuracy: 0.9383 - loss: 0.1910
Epoch 613/1000
7/7            0s 769us/step -
accuracy: 0.9035 - loss: 0.2367
Epoch 614/1000
7/7            0s 873us/step -
accuracy: 0.9225 - loss: 0.1902
Epoch 615/1000
7/7            0s 809us/step -
accuracy: 0.9294 - loss: 0.2054
Epoch 616/1000
7/7            0s 782us/step -
accuracy: 0.9208 - loss: 0.1923
Epoch 617/1000
7/7            0s 728us/step -
accuracy: 0.9092 - loss: 0.2269
Epoch 618/1000
7/7            0s 726us/step -
accuracy: 0.9201 - loss: 0.1902
Epoch 619/1000
7/7            0s 926us/step -
accuracy: 0.9306 - loss: 0.1958
Epoch 620/1000
7/7            0s 920us/step -
accuracy: 0.9154 - loss: 0.2178
Epoch 621/1000
7/7            0s 898us/step -
accuracy: 0.9419 - loss: 0.1787
Epoch 622/1000
7/7            0s 883us/step -
accuracy: 0.9465 - loss: 0.1651
Epoch 623/1000
7/7            0s 837us/step -
accuracy: 0.9122 - loss: 0.2324
Epoch 624/1000
7/7            0s 856us/step -
accuracy: 0.9322 - loss: 0.2039
Epoch 625/1000
7/7            0s 745us/step -
accuracy: 0.9365 - loss: 0.1896
Epoch 626/1000
7/7            0s 1ms/step -
accuracy: 0.9423 - loss: 0.1883
Epoch 627/1000
7/7            0s 1ms/step -
accuracy: 0.9149 - loss: 0.1933
Epoch 628/1000
```

```
7/7            0s 790us/step -
accuracy: 0.9156 - loss: 0.2086
Epoch 629/1000
7/7            0s 762us/step -
accuracy: 0.9314 - loss: 0.1820
Epoch 630/1000
7/7            0s 775us/step -
accuracy: 0.9326 - loss: 0.1896
Epoch 631/1000
7/7            0s 695us/step -
accuracy: 0.9230 - loss: 0.1947
Epoch 632/1000
7/7            0s 713us/step -
accuracy: 0.9288 - loss: 0.1997
Epoch 633/1000
7/7            0s 955us/step -
accuracy: 0.9368 - loss: 0.1930
Epoch 634/1000
7/7            0s 786us/step -
accuracy: 0.9451 - loss: 0.1615
Epoch 635/1000
7/7            0s 783us/step -
accuracy: 0.9420 - loss: 0.1749
Epoch 636/1000
7/7            0s 716us/step -
accuracy: 0.9152 - loss: 0.2115
Epoch 637/1000
7/7            0s 692us/step -
accuracy: 0.9515 - loss: 0.1564
Epoch 638/1000
7/7            0s 918us/step -
accuracy: 0.9379 - loss: 0.1867
Epoch 639/1000
7/7            0s 733us/step -
accuracy: 0.9181 - loss: 0.2009
Epoch 640/1000
7/7            0s 861us/step -
accuracy: 0.9303 - loss: 0.1960
Epoch 641/1000
7/7            0s 755us/step -
accuracy: 0.9166 - loss: 0.2059
Epoch 642/1000
7/7            0s 691us/step -
accuracy: 0.9413 - loss: 0.1817
Epoch 643/1000
7/7            0s 695us/step -
accuracy: 0.9400 - loss: 0.1849
Epoch 644/1000
```

```
7/7            0s 919us/step -
accuracy: 0.9290 - loss: 0.2171
Epoch 645/1000
7/7            0s 782us/step -
accuracy: 0.9235 - loss: 0.2280
Epoch 646/1000
7/7            0s 1ms/step -
accuracy: 0.9448 - loss: 0.1751
Epoch 647/1000
7/7            0s 1ms/step -
accuracy: 0.9472 - loss: 0.1856
Epoch 648/1000
7/7            0s 855us/step -
accuracy: 0.9422 - loss: 0.1902
Epoch 649/1000
7/7            0s 675us/step -
accuracy: 0.9308 - loss: 0.2013
Epoch 650/1000
7/7            0s 681us/step -
accuracy: 0.9476 - loss: 0.1758
Epoch 651/1000
7/7            0s 661us/step -
accuracy: 0.9398 - loss: 0.2059
Epoch 652/1000
7/7            0s 826us/step -
accuracy: 0.9396 - loss: 0.1871
Epoch 653/1000
7/7            0s 913us/step -
accuracy: 0.9412 - loss: 0.1942
Epoch 654/1000
7/7            0s 707us/step -
accuracy: 0.9352 - loss: 0.1868
Epoch 655/1000
7/7            0s 688us/step -
accuracy: 0.9437 - loss: 0.1966
Epoch 656/1000
7/7            0s 725us/step -
accuracy: 0.9333 - loss: 0.1958
Epoch 657/1000
7/7            0s 835us/step -
accuracy: 0.9364 - loss: 0.1832
Epoch 658/1000
7/7            0s 744us/step -
accuracy: 0.9347 - loss: 0.1937
Epoch 659/1000
7/7            0s 871us/step -
accuracy: 0.9375 - loss: 0.1908
Epoch 660/1000
```

```
7/7            0s 718us/step -
accuracy: 0.9386 - loss: 0.1904
Epoch 661/1000
7/7            0s 1ms/step -
accuracy: 0.9601 - loss: 0.1604
Epoch 662/1000
7/7            0s 729us/step -
accuracy: 0.9476 - loss: 0.1922
Epoch 663/1000
7/7            0s 808us/step -
accuracy: 0.9323 - loss: 0.1971
Epoch 664/1000
7/7            0s 796us/step -
accuracy: 0.9431 - loss: 0.1849
Epoch 665/1000
7/7            0s 757us/step -
accuracy: 0.9573 - loss: 0.1633
Epoch 666/1000
7/7            0s 803us/step -
accuracy: 0.9438 - loss: 0.1963
Epoch 667/1000
7/7            0s 1ms/step -
accuracy: 0.9569 - loss: 0.1715
Epoch 668/1000
7/7            0s 2ms/step -
accuracy: 0.9439 - loss: 0.1848
Epoch 669/1000
7/7            0s 881us/step -
accuracy: 0.9528 - loss: 0.1703
Epoch 670/1000
7/7            0s 781us/step -
accuracy: 0.9445 - loss: 0.1797
Epoch 671/1000
7/7            0s 1ms/step -
accuracy: 0.9336 - loss: 0.2024
Epoch 672/1000
7/7            0s 934us/step -
accuracy: 0.9421 - loss: 0.2029
Epoch 673/1000
7/7            0s 970us/step -
accuracy: 0.9462 - loss: 0.1851
Epoch 674/1000
7/7            0s 775us/step -
accuracy: 0.9559 - loss: 0.1824
Epoch 675/1000
7/7            0s 705us/step -
accuracy: 0.9571 - loss: 0.1823
Epoch 676/1000
```

```
7/7            0s 1ms/step -
accuracy: 0.9400 - loss: 0.1950
Epoch 677/1000
7/7            0s 744us/step -
accuracy: 0.9655 - loss: 0.1566
Epoch 678/1000
7/7            0s 881us/step -
accuracy: 0.9371 - loss: 0.1856
Epoch 679/1000
7/7            0s 945us/step -
accuracy: 0.9269 - loss: 0.2153
Epoch 680/1000
7/7            0s 809us/step -
accuracy: 0.9424 - loss: 0.1899
Epoch 681/1000
7/7            0s 985us/step -
accuracy: 0.9431 - loss: 0.1829
Epoch 682/1000
7/7            0s 947us/step -
accuracy: 0.9286 - loss: 0.1938
Epoch 683/1000
7/7            0s 1ms/step -
accuracy: 0.9510 - loss: 0.1720
Epoch 684/1000
7/7            0s 801us/step -
accuracy: 0.9383 - loss: 0.1963
Epoch 685/1000
7/7            0s 979us/step -
accuracy: 0.9620 - loss: 0.1690
Epoch 686/1000
7/7            0s 935us/step -
accuracy: 0.9502 - loss: 0.1823
Epoch 687/1000
7/7            0s 822us/step -
accuracy: 0.9400 - loss: 0.1777
Epoch 688/1000
7/7            0s 804us/step -
accuracy: 0.9435 - loss: 0.1828
Epoch 689/1000
7/7            0s 841us/step -
accuracy: 0.9301 - loss: 0.1953
Epoch 690/1000
7/7            0s 1ms/step -
accuracy: 0.9403 - loss: 0.1717
Epoch 691/1000
7/7            0s 726us/step -
accuracy: 0.9467 - loss: 0.1852
Epoch 692/1000
```

```
7/7            0s 940us/step -
accuracy: 0.9352 - loss: 0.1783
Epoch 693/1000
7/7            0s 944us/step -
accuracy: 0.9468 - loss: 0.1792
Epoch 694/1000
7/7            0s 803us/step -
accuracy: 0.9534 - loss: 0.1783
Epoch 695/1000
7/7            0s 1ms/step -
accuracy: 0.9487 - loss: 0.1778
Epoch 696/1000
7/7            0s 819us/step -
accuracy: 0.9454 - loss: 0.1746
Epoch 697/1000
7/7            0s 812us/step -
accuracy: 0.9318 - loss: 0.1775
Epoch 698/1000
7/7            0s 687us/step -
accuracy: 0.9251 - loss: 0.2034
Epoch 699/1000
7/7            0s 1ms/step -
accuracy: 0.9426 - loss: 0.1756
Epoch 700/1000
7/7            0s 885us/step -
accuracy: 0.9430 - loss: 0.1824
Epoch 701/1000
7/7            0s 771us/step -
accuracy: 0.9547 - loss: 0.1774
Epoch 702/1000
7/7            0s 774us/step -
accuracy: 0.9520 - loss: 0.1747
Epoch 703/1000
7/7            0s 790us/step -
accuracy: 0.9495 - loss: 0.1663
Epoch 704/1000
7/7            0s 647us/step -
accuracy: 0.9463 - loss: 0.1758
Epoch 705/1000
7/7            0s 789us/step -
accuracy: 0.9377 - loss: 0.1850
Epoch 706/1000
7/7            0s 871us/step -
accuracy: 0.9522 - loss: 0.1907
Epoch 707/1000
7/7            0s 729us/step -
accuracy: 0.9395 - loss: 0.1890
Epoch 708/1000
```

```
7/7            0s 810us/step -
accuracy: 0.9510 - loss: 0.1673
Epoch 709/1000
7/7            0s 759us/step -
accuracy: 0.9622 - loss: 0.1529
Epoch 710/1000
7/7            0s 819us/step -
accuracy: 0.9512 - loss: 0.1565
Epoch 711/1000
7/7            0s 1ms/step -
accuracy: 0.9336 - loss: 0.1952
Epoch 712/1000
7/7            0s 905us/step -
accuracy: 0.9364 - loss: 0.1864
Epoch 713/1000
7/7            0s 896us/step -
accuracy: 0.9569 - loss: 0.1748
Epoch 714/1000
7/7            0s 994us/step -
accuracy: 0.9589 - loss: 0.1652
Epoch 715/1000
7/7            0s 875us/step -
accuracy: 0.9370 - loss: 0.1795
Epoch 716/1000
7/7            0s 791us/step -
accuracy: 0.9500 - loss: 0.1742
Epoch 717/1000
7/7            0s 1ms/step -
accuracy: 0.9561 - loss: 0.1809
Epoch 718/1000
7/7            0s 709us/step -
accuracy: 0.9379 - loss: 0.1821
Epoch 719/1000
7/7            0s 776us/step -
accuracy: 0.9398 - loss: 0.1898
Epoch 720/1000
7/7            0s 832us/step -
accuracy: 0.9571 - loss: 0.1761
Epoch 721/1000
7/7            0s 759us/step -
accuracy: 0.9366 - loss: 0.1975
Epoch 722/1000
7/7            0s 866us/step -
accuracy: 0.9687 - loss: 0.1637
Epoch 723/1000
7/7            0s 954us/step -
accuracy: 0.9526 - loss: 0.1778
Epoch 724/1000
```

```
7/7            0s 813us/step -
accuracy: 0.9458 - loss: 0.1784
Epoch 725/1000
7/7            0s 943us/step -
accuracy: 0.9471 - loss: 0.1751
Epoch 726/1000
7/7            0s 970us/step -
accuracy: 0.9558 - loss: 0.1574
Epoch 727/1000
7/7            0s 733us/step -
accuracy: 0.9558 - loss: 0.1641
Epoch 728/1000
7/7            0s 868us/step -
accuracy: 0.9467 - loss: 0.1723
Epoch 729/1000
7/7            0s 673us/step -
accuracy: 0.9580 - loss: 0.1561
Epoch 730/1000
7/7            0s 764us/step -
accuracy: 0.9572 - loss: 0.1575
Epoch 731/1000
7/7            0s 793us/step -
accuracy: 0.9546 - loss: 0.1641
Epoch 732/1000
7/7            0s 660us/step -
accuracy: 0.9550 - loss: 0.1735
Epoch 733/1000
7/7            0s 672us/step -
accuracy: 0.9536 - loss: 0.1668
Epoch 734/1000
7/7            0s 744us/step -
accuracy: 0.9448 - loss: 0.1896
Epoch 735/1000
7/7            0s 770us/step -
accuracy: 0.9638 - loss: 0.1525
Epoch 736/1000
7/7            0s 882us/step -
accuracy: 0.9415 - loss: 0.1870
Epoch 737/1000
7/7            0s 758us/step -
accuracy: 0.9437 - loss: 0.1694
Epoch 738/1000
7/7            0s 806us/step -
accuracy: 0.9573 - loss: 0.1484
Epoch 739/1000
7/7            0s 865us/step -
accuracy: 0.9402 - loss: 0.1788
Epoch 740/1000
```

```
7/7            0s 836us/step -
accuracy: 0.9650 - loss: 0.1455
Epoch 741/1000
7/7            0s 1ms/step -
accuracy: 0.9400 - loss: 0.1840
Epoch 742/1000
7/7            0s 1ms/step -
accuracy: 0.9631 - loss: 0.1440
Epoch 743/1000
7/7            0s 647us/step -
accuracy: 0.9357 - loss: 0.1892
Epoch 744/1000
7/7            0s 816us/step -
accuracy: 0.9591 - loss: 0.1565
Epoch 745/1000
7/7            0s 767us/step -
accuracy: 0.9466 - loss: 0.1703
Epoch 746/1000
7/7            0s 885us/step -
accuracy: 0.9422 - loss: 0.1752
Epoch 747/1000
7/7            0s 939us/step -
accuracy: 0.9521 - loss: 0.1574
Epoch 748/1000
7/7            0s 796us/step -
accuracy: 0.9341 - loss: 0.1910
Epoch 749/1000
7/7            0s 690us/step -
accuracy: 0.9692 - loss: 0.1361
Epoch 750/1000
7/7            0s 726us/step -
accuracy: 0.9349 - loss: 0.1727
Epoch 751/1000
7/7            0s 833us/step -
accuracy: 0.9506 - loss: 0.1586
Epoch 752/1000
7/7            0s 853us/step -
accuracy: 0.9538 - loss: 0.1623
Epoch 753/1000
7/7            0s 775us/step -
accuracy: 0.9452 - loss: 0.1806
Epoch 754/1000
7/7            0s 726us/step -
accuracy: 0.9633 - loss: 0.1558
Epoch 755/1000
7/7            0s 833us/step -
accuracy: 0.9543 - loss: 0.1648
Epoch 756/1000
```

```
7/7            0s 846us/step -
accuracy: 0.9583 - loss: 0.1507
Epoch 757/1000
7/7            0s 911us/step -
accuracy: 0.9484 - loss: 0.1830
Epoch 758/1000
7/7            0s 880us/step -
accuracy: 0.9557 - loss: 0.1588
Epoch 759/1000
7/7            0s 818us/step -
accuracy: 0.9409 - loss: 0.1870
Epoch 760/1000
7/7            0s 771us/step -
accuracy: 0.9506 - loss: 0.1548
Epoch 761/1000
7/7            0s 762us/step -
accuracy: 0.9614 - loss: 0.1531
Epoch 762/1000
7/7            0s 794us/step -
accuracy: 0.9546 - loss: 0.1515
Epoch 763/1000
7/7            0s 647us/step -
accuracy: 0.9526 - loss: 0.1525
Epoch 764/1000
7/7            0s 740us/step -
accuracy: 0.9408 - loss: 0.1659
Epoch 765/1000
7/7            0s 668us/step -
accuracy: 0.9586 - loss: 0.1674
Epoch 766/1000
7/7            0s 723us/step -
accuracy: 0.9410 - loss: 0.1825
Epoch 767/1000
7/7            0s 831us/step -
accuracy: 0.9482 - loss: 0.1723
Epoch 768/1000
7/7            0s 864us/step -
accuracy: 0.9359 - loss: 0.1747
Epoch 769/1000
7/7            0s 941us/step -
accuracy: 0.9418 - loss: 0.1850
Epoch 770/1000
7/7            0s 1ms/step -
accuracy: 0.9560 - loss: 0.1620
Epoch 771/1000
7/7            0s 970us/step -
accuracy: 0.9553 - loss: 0.1579
Epoch 772/1000
```

```
7/7             0s 656us/step -
accuracy: 0.9343 - loss: 0.1944
Epoch 773/1000
7/7             0s 799us/step -
accuracy: 0.9515 - loss: 0.1579
Epoch 774/1000
7/7             0s 829us/step -
accuracy: 0.9355 - loss: 0.1778
Epoch 775/1000
7/7             0s 883us/step -
accuracy: 0.9563 - loss: 0.1597
Epoch 776/1000
7/7             0s 715us/step -
accuracy: 0.9534 - loss: 0.1576
Epoch 777/1000
7/7             0s 631us/step -
accuracy: 0.9402 - loss: 0.1669
Epoch 778/1000
7/7             0s 776us/step -
accuracy: 0.9533 - loss: 0.1633
Epoch 779/1000
7/7             0s 713us/step -
accuracy: 0.9249 - loss: 0.2026
Epoch 780/1000
7/7             0s 811us/step -
accuracy: 0.9443 - loss: 0.1610
Epoch 781/1000
7/7             0s 907us/step -
accuracy: 0.9603 - loss: 0.1594
Epoch 782/1000
7/7             0s 793us/step -
accuracy: 0.9527 - loss: 0.1609
Epoch 783/1000
7/7             0s 857us/step -
accuracy: 0.9510 - loss: 0.1689
Epoch 784/1000
7/7             0s 692us/step -
accuracy: 0.9353 - loss: 0.1695
Epoch 785/1000
7/7             0s 872us/step -
accuracy: 0.9375 - loss: 0.1647
Epoch 786/1000
7/7             0s 739us/step -
accuracy: 0.9497 - loss: 0.1729
Epoch 787/1000
7/7             0s 771us/step -
accuracy: 0.9503 - loss: 0.1536
Epoch 788/1000
```

```
7/7             0s 1ms/step -
accuracy: 0.9571 - loss: 0.1596
Epoch 789/1000
7/7             0s 1ms/step -
accuracy: 0.9541 - loss: 0.1526
Epoch 790/1000
7/7             0s 838us/step -
accuracy: 0.9201 - loss: 0.1864
Epoch 791/1000
7/7             0s 885us/step -
accuracy: 0.9334 - loss: 0.1819
Epoch 792/1000
7/7             0s 766us/step -
accuracy: 0.9485 - loss: 0.1558
Epoch 793/1000
7/7             0s 808us/step -
accuracy: 0.9497 - loss: 0.1737
Epoch 794/1000
7/7             0s 789us/step -
accuracy: 0.9241 - loss: 0.1892
Epoch 795/1000
7/7             0s 766us/step -
accuracy: 0.9370 - loss: 0.1868
Epoch 796/1000
7/7             0s 774us/step -
accuracy: 0.9618 - loss: 0.1613
Epoch 797/1000
7/7             0s 770us/step -
accuracy: 0.9537 - loss: 0.1523
Epoch 798/1000
7/7             0s 697us/step -
accuracy: 0.9342 - loss: 0.1606
Epoch 799/1000
7/7             0s 1ms/step -
accuracy: 0.9622 - loss: 0.1468
Epoch 800/1000
7/7             0s 914us/step -
accuracy: 0.9510 - loss: 0.1551
Epoch 801/1000
7/7             0s 714us/step -
accuracy: 0.9525 - loss: 0.1558
Epoch 802/1000
7/7             0s 806us/step -
accuracy: 0.9513 - loss: 0.1401
Epoch 803/1000
7/7             0s 826us/step -
accuracy: 0.9394 - loss: 0.1847
Epoch 804/1000
```

```
7/7            0s 942us/step -
accuracy: 0.9521 - loss: 0.1551
Epoch 805/1000
7/7            0s 862us/step -
accuracy: 0.9342 - loss: 0.1713
Epoch 806/1000
7/7            0s 702us/step -
accuracy: 0.9571 - loss: 0.1437
Epoch 807/1000
7/7            0s 741us/step -
accuracy: 0.9347 - loss: 0.1634
Epoch 808/1000
7/7            0s 1ms/step -
accuracy: 0.9560 - loss: 0.1459
Epoch 809/1000
7/7            0s 811us/step -
accuracy: 0.9564 - loss: 0.1478
Epoch 810/1000
7/7            0s 869us/step -
accuracy: 0.9526 - loss: 0.1437
Epoch 811/1000
7/7            0s 871us/step -
accuracy: 0.9514 - loss: 0.1487
Epoch 812/1000
7/7            0s 848us/step -
accuracy: 0.9658 - loss: 0.1399
Epoch 813/1000
7/7            0s 712us/step -
accuracy: 0.9417 - loss: 0.1582
Epoch 814/1000
7/7            0s 712us/step -
accuracy: 0.9465 - loss: 0.1607
Epoch 815/1000
7/7            0s 928us/step -
accuracy: 0.9529 - loss: 0.1517
Epoch 816/1000
7/7            0s 851us/step -
accuracy: 0.9406 - loss: 0.1622
Epoch 817/1000
7/7            0s 797us/step -
accuracy: 0.9552 - loss: 0.1445
Epoch 818/1000
7/7            0s 665us/step -
accuracy: 0.9625 - loss: 0.1393
Epoch 819/1000
7/7            0s 790us/step -
accuracy: 0.9570 - loss: 0.1476
Epoch 820/1000
```

```
7/7            0s 930us/step -
accuracy: 0.9409 - loss: 0.1697
Epoch 821/1000
7/7            0s 1ms/step -
accuracy: 0.9617 - loss: 0.1538
Epoch 822/1000
7/7            0s 686us/step -
accuracy: 0.9424 - loss: 0.1647
Epoch 823/1000
7/7            0s 821us/step -
accuracy: 0.9521 - loss: 0.1565
Epoch 824/1000
7/7            0s 925us/step -
accuracy: 0.9525 - loss: 0.1492
Epoch 825/1000
7/7            0s 1ms/step -
accuracy: 0.9398 - loss: 0.1730
Epoch 826/1000
7/7            0s 945us/step -
accuracy: 0.9641 - loss: 0.1430
Epoch 827/1000
7/7            0s 941us/step -
accuracy: 0.9429 - loss: 0.1609
Epoch 828/1000
7/7            0s 875us/step -
accuracy: 0.9455 - loss: 0.1639
Epoch 829/1000
7/7            0s 904us/step -
accuracy: 0.9243 - loss: 0.1841
Epoch 830/1000
7/7            0s 1ms/step -
accuracy: 0.9564 - loss: 0.1450
Epoch 831/1000
7/7            0s 842us/step -
accuracy: 0.9657 - loss: 0.1386
Epoch 832/1000
7/7            0s 817us/step -
accuracy: 0.9454 - loss: 0.1591
Epoch 833/1000
7/7            0s 803us/step -
accuracy: 0.9553 - loss: 0.1494
Epoch 834/1000
7/7            0s 776us/step -
accuracy: 0.9273 - loss: 0.1824
Epoch 835/1000
7/7            0s 756us/step -
accuracy: 0.9583 - loss: 0.1422
Epoch 836/1000
```

```
7/7            0s 963us/step -
accuracy: 0.9523 - loss: 0.1696
Epoch 837/1000
7/7            0s 869us/step -
accuracy: 0.9562 - loss: 0.1543
Epoch 838/1000
7/7            0s 744us/step -
accuracy: 0.9538 - loss: 0.1503
Epoch 839/1000
7/7            0s 701us/step -
accuracy: 0.9671 - loss: 0.1436
Epoch 840/1000
7/7            0s 770us/step -
accuracy: 0.9538 - loss: 0.1516
Epoch 841/1000
7/7            0s 953us/step -
accuracy: 0.9583 - loss: 0.1484
Epoch 842/1000
7/7            0s 681us/step -
accuracy: 0.9582 - loss: 0.1393
Epoch 843/1000
7/7            0s 1ms/step -
accuracy: 0.9536 - loss: 0.1427
Epoch 844/1000
7/7            0s 802us/step -
accuracy: 0.9494 - loss: 0.1631
Epoch 845/1000
7/7            0s 882us/step -
accuracy: 0.9498 - loss: 0.1518
Epoch 846/1000
7/7            0s 725us/step -
accuracy: 0.9355 - loss: 0.1673
Epoch 847/1000
7/7            0s 860us/step -
accuracy: 0.9546 - loss: 0.1548
Epoch 848/1000
7/7            0s 915us/step -
accuracy: 0.9457 - loss: 0.1570
Epoch 849/1000
7/7            0s 725us/step -
accuracy: 0.9588 - loss: 0.1593
Epoch 850/1000
7/7            0s 680us/step -
accuracy: 0.9611 - loss: 0.1501
Epoch 851/1000
7/7            0s 683us/step -
accuracy: 0.9551 - loss: 0.1480
Epoch 852/1000
```

```
7/7            0s 652us/step -
accuracy: 0.9556 - loss: 0.1447
Epoch 853/1000
7/7            0s 682us/step -
accuracy: 0.9562 - loss: 0.1489
Epoch 854/1000
7/7            0s 757us/step -
accuracy: 0.9491 - loss: 0.1465
Epoch 855/1000
7/7            0s 790us/step -
accuracy: 0.9553 - loss: 0.1561
Epoch 856/1000
7/7            0s 820us/step -
accuracy: 0.9553 - loss: 0.1539
Epoch 857/1000
7/7            0s 942us/step -
accuracy: 0.9572 - loss: 0.1483
Epoch 858/1000
7/7            0s 831us/step -
accuracy: 0.9351 - loss: 0.1657
Epoch 859/1000
7/7            0s 1ms/step -
accuracy: 0.9556 - loss: 0.1586
Epoch 860/1000
7/7            0s 668us/step -
accuracy: 0.9405 - loss: 0.1664
Epoch 861/1000
7/7            0s 703us/step -
accuracy: 0.9526 - loss: 0.1554
Epoch 862/1000
7/7            0s 712us/step -
accuracy: 0.9562 - loss: 0.1390
Epoch 863/1000
7/7            0s 774us/step -
accuracy: 0.9519 - loss: 0.1584
Epoch 864/1000
7/7            0s 679us/step -
accuracy: 0.9486 - loss: 0.1521
Epoch 865/1000
7/7            0s 705us/step -
accuracy: 0.9372 - loss: 0.1665
Epoch 866/1000
7/7            0s 898us/step -
accuracy: 0.9498 - loss: 0.1483
Epoch 867/1000
7/7            0s 853us/step -
accuracy: 0.9462 - loss: 0.1502
Epoch 868/1000
```

```
7/7            0s 748us/step -
accuracy: 0.9493 - loss: 0.1528
Epoch 869/1000
7/7            0s 895us/step -
accuracy: 0.9471 - loss: 0.1609
Epoch 870/1000
7/7            0s 892us/step -
accuracy: 0.9435 - loss: 0.1572
Epoch 871/1000
7/7            0s 1ms/step -
accuracy: 0.9619 - loss: 0.1407
Epoch 872/1000
7/7            0s 881us/step -
accuracy: 0.9609 - loss: 0.1435
Epoch 873/1000
7/7            0s 922us/step -
accuracy: 0.9593 - loss: 0.1278
Epoch 874/1000
7/7            0s 783us/step -
accuracy: 0.9546 - loss: 0.1437
Epoch 875/1000
7/7            0s 746us/step -
accuracy: 0.9543 - loss: 0.1424
Epoch 876/1000
7/7            0s 726us/step -
accuracy: 0.9539 - loss: 0.1379
Epoch 877/1000
7/7            0s 776us/step -
accuracy: 0.9572 - loss: 0.1380
Epoch 878/1000
7/7            0s 827us/step -
accuracy: 0.9637 - loss: 0.1341
Epoch 879/1000
7/7            0s 954us/step -
accuracy: 0.9603 - loss: 0.1385
Epoch 880/1000
7/7            0s 890us/step -
accuracy: 0.9582 - loss: 0.1359
Epoch 881/1000
7/7            0s 1ms/step -
accuracy: 0.9548 - loss: 0.1582
Epoch 882/1000
7/7            0s 752us/step -
accuracy: 0.9501 - loss: 0.1358
Epoch 883/1000
7/7            0s 732us/step -
accuracy: 0.9286 - loss: 0.1740
Epoch 884/1000
```

```
7/7            0s 737us/step -
accuracy: 0.9335 - loss: 0.1703
Epoch 885/1000
7/7            0s 780us/step -
accuracy: 0.9482 - loss: 0.1568
Epoch 886/1000
7/7            0s 751us/step -
accuracy: 0.9411 - loss: 0.1638
Epoch 887/1000
7/7            0s 719us/step -
accuracy: 0.9521 - loss: 0.1438
Epoch 888/1000
7/7            0s 818us/step -
accuracy: 0.9605 - loss: 0.1386
Epoch 889/1000
7/7            0s 866us/step -
accuracy: 0.9444 - loss: 0.1451
Epoch 890/1000
7/7            0s 879us/step -
accuracy: 0.9497 - loss: 0.1498
Epoch 891/1000
7/7            0s 1ms/step -
accuracy: 0.9473 - loss: 0.1511
Epoch 892/1000
7/7            0s 1ms/step -
accuracy: 0.9499 - loss: 0.1457
Epoch 893/1000
7/7            0s 798us/step -
accuracy: 0.9434 - loss: 0.1519
Epoch 894/1000
7/7            0s 666us/step -
accuracy: 0.9591 - loss: 0.1442
Epoch 895/1000
7/7            0s 728us/step -
accuracy: 0.9516 - loss: 0.1498
Epoch 896/1000
7/7            0s 791us/step -
accuracy: 0.9523 - loss: 0.1279
Epoch 897/1000
7/7            0s 818us/step -
accuracy: 0.9536 - loss: 0.1516
Epoch 898/1000
7/7            0s 805us/step -
accuracy: 0.9613 - loss: 0.1285
Epoch 899/1000
7/7            0s 761us/step -
accuracy: 0.9426 - loss: 0.1576
Epoch 900/1000
```

```
7/7            0s 653us/step -
accuracy: 0.9658 - loss: 0.1305
Epoch 901/1000
7/7            0s 794us/step -
accuracy: 0.9580 - loss: 0.1483
Epoch 902/1000
7/7            0s 850us/step -
accuracy: 0.9533 - loss: 0.1587
Epoch 903/1000
7/7            0s 956us/step -
accuracy: 0.9566 - loss: 0.1379
Epoch 904/1000
7/7            0s 874us/step -
accuracy: 0.9344 - loss: 0.1635
Epoch 905/1000
7/7            0s 2ms/step -
accuracy: 0.9514 - loss: 0.1515
Epoch 906/1000
7/7            0s 958us/step -
accuracy: 0.9534 - loss: 0.1311
Epoch 907/1000
7/7            0s 657us/step -
accuracy: 0.9419 - loss: 0.1471
Epoch 908/1000
7/7            0s 850us/step -
accuracy: 0.9714 - loss: 0.1289
Epoch 909/1000
7/7            0s 805us/step -
accuracy: 0.9540 - loss: 0.1504
Epoch 910/1000
7/7            0s 773us/step -
accuracy: 0.9327 - loss: 0.1723
Epoch 911/1000
7/7            0s 771us/step -
accuracy: 0.9575 - loss: 0.1435
Epoch 912/1000
7/7            0s 812us/step -
accuracy: 0.9382 - loss: 0.1683
Epoch 913/1000
7/7            0s 779us/step -
accuracy: 0.9582 - loss: 0.1412
Epoch 914/1000
7/7            0s 776us/step -
accuracy: 0.9559 - loss: 0.1410
Epoch 915/1000
7/7            0s 853us/step -
accuracy: 0.9608 - loss: 0.1371
Epoch 916/1000
```

```
7/7            0s 712us/step -
accuracy: 0.9616 - loss: 0.1446
Epoch 917/1000
7/7            0s 714us/step -
accuracy: 0.9542 - loss: 0.1434
Epoch 918/1000
7/7            0s 804us/step -
accuracy: 0.9692 - loss: 0.1364
Epoch 919/1000
7/7            0s 857us/step -
accuracy: 0.9572 - loss: 0.1363
Epoch 920/1000
7/7            0s 860us/step -
accuracy: 0.9486 - loss: 0.1483
Epoch 921/1000
7/7            0s 729us/step -
accuracy: 0.9551 - loss: 0.1409
Epoch 922/1000
7/7            0s 812us/step -
accuracy: 0.9581 - loss: 0.1418
Epoch 923/1000
7/7            0s 840us/step -
accuracy: 0.9729 - loss: 0.1316
Epoch 924/1000
7/7            0s 1ms/step -
accuracy: 0.9678 - loss: 0.1313
Epoch 925/1000
7/7            0s 926us/step -
accuracy: 0.9645 - loss: 0.1335
Epoch 926/1000
7/7            0s 1ms/step -
accuracy: 0.9686 - loss: 0.1302
Epoch 927/1000
7/7            0s 968us/step -
accuracy: 0.9754 - loss: 0.1328
Epoch 928/1000
7/7            0s 910us/step -
accuracy: 0.9543 - loss: 0.1407
Epoch 929/1000
7/7            0s 751us/step -
accuracy: 0.9610 - loss: 0.1424
Epoch 930/1000
7/7            0s 809us/step -
accuracy: 0.9537 - loss: 0.1729
Epoch 931/1000
7/7            0s 833us/step -
accuracy: 0.9719 - loss: 0.1279
Epoch 932/1000
```

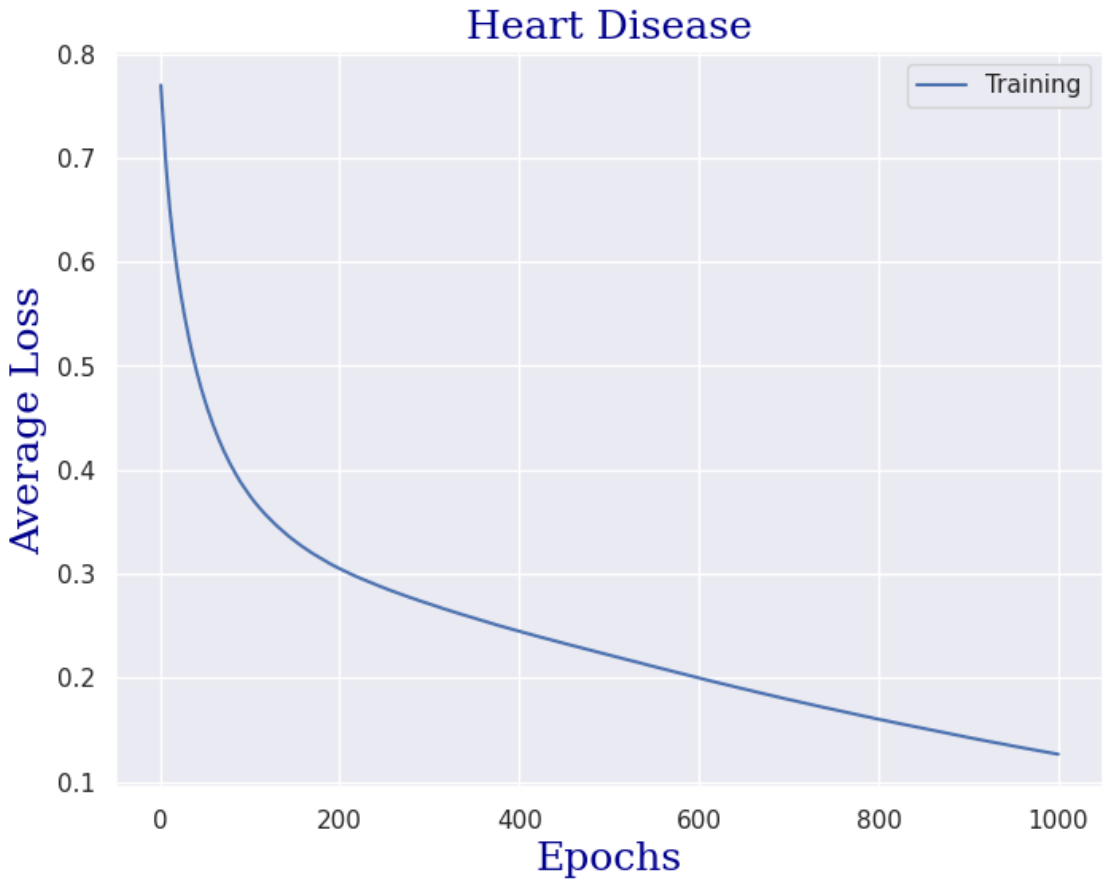
```
7/7            0s 846us/step -
accuracy: 0.9624 - loss: 0.1307
Epoch 933/1000
7/7            0s 751us/step -
accuracy: 0.9611 - loss: 0.1488
Epoch 934/1000
7/7            0s 746us/step -
accuracy: 0.9559 - loss: 0.1459
Epoch 935/1000
7/7            0s 750us/step -
accuracy: 0.9697 - loss: 0.1306
Epoch 936/1000
7/7            0s 666us/step -
accuracy: 0.9716 - loss: 0.1326
Epoch 937/1000
7/7            0s 845us/step -
accuracy: 0.9505 - loss: 0.1382
Epoch 938/1000
7/7            0s 743us/step -
accuracy: 0.9654 - loss: 0.1336
Epoch 939/1000
7/7            0s 756us/step -
accuracy: 0.9681 - loss: 0.1326
Epoch 940/1000
7/7            0s 755us/step -
accuracy: 0.9753 - loss: 0.1216
Epoch 941/1000
7/7            0s 802us/step -
accuracy: 0.9544 - loss: 0.1437
Epoch 942/1000
7/7            0s 825us/step -
accuracy: 0.9600 - loss: 0.1404
Epoch 943/1000
7/7            0s 741us/step -
accuracy: 0.9777 - loss: 0.1266
Epoch 944/1000
7/7            0s 845us/step -
accuracy: 0.9707 - loss: 0.1311
Epoch 945/1000
7/7            0s 919us/step -
accuracy: 0.9600 - loss: 0.1402
Epoch 946/1000
7/7            0s 890us/step -
accuracy: 0.9777 - loss: 0.1227
Epoch 947/1000
7/7            0s 970us/step -
accuracy: 0.9675 - loss: 0.1440
Epoch 948/1000
```

```
7/7             0s 865us/step -
accuracy: 0.9624 - loss: 0.1484
Epoch 949/1000
7/7             0s 689us/step -
accuracy: 0.9740 - loss: 0.1256
Epoch 950/1000
7/7             0s 753us/step -
accuracy: 0.9735 - loss: 0.1332
Epoch 951/1000
7/7             0s 981us/step -
accuracy: 0.9626 - loss: 0.1345
Epoch 952/1000
7/7             0s 933us/step -
accuracy: 0.9683 - loss: 0.1350
Epoch 953/1000
7/7             0s 928us/step -
accuracy: 0.9710 - loss: 0.1303
Epoch 954/1000
7/7             0s 810us/step -
accuracy: 0.9712 - loss: 0.1167
Epoch 955/1000
7/7             0s 657us/step -
accuracy: 0.9655 - loss: 0.1437
Epoch 956/1000
7/7             0s 870us/step -
accuracy: 0.9673 - loss: 0.1406
Epoch 957/1000
7/7             0s 742us/step -
accuracy: 0.9780 - loss: 0.1338
Epoch 958/1000
7/7             0s 886us/step -
accuracy: 0.9836 - loss: 0.1161
Epoch 959/1000
7/7             0s 681us/step -
accuracy: 0.9645 - loss: 0.1362
Epoch 960/1000
7/7             0s 796us/step -
accuracy: 0.9768 - loss: 0.1352
Epoch 961/1000
7/7             0s 869us/step -
accuracy: 0.9808 - loss: 0.1121
Epoch 962/1000
7/7             0s 848us/step -
accuracy: 0.9713 - loss: 0.1241
Epoch 963/1000
7/7             0s 836us/step -
accuracy: 0.9697 - loss: 0.1277
Epoch 964/1000
```

```
7/7            0s 862us/step -
accuracy: 0.9613 - loss: 0.1325
Epoch 965/1000
7/7            0s 750us/step -
accuracy: 0.9761 - loss: 0.1168
Epoch 966/1000
7/7            0s 1ms/step -
accuracy: 0.9752 - loss: 0.1372
Epoch 967/1000
7/7            0s 892us/step -
accuracy: 0.9698 - loss: 0.1338
Epoch 968/1000
7/7            0s 903us/step -
accuracy: 0.9760 - loss: 0.1194
Epoch 969/1000
7/7            0s 796us/step -
accuracy: 0.9753 - loss: 0.1277
Epoch 970/1000
7/7            0s 720us/step -
accuracy: 0.9646 - loss: 0.1455
Epoch 971/1000
7/7            0s 937us/step -
accuracy: 0.9717 - loss: 0.1325
Epoch 972/1000
7/7            0s 1ms/step -
accuracy: 0.9640 - loss: 0.1223
Epoch 973/1000
7/7            0s 1ms/step -
accuracy: 0.9705 - loss: 0.1190
Epoch 974/1000
7/7            0s 1ms/step -
accuracy: 0.9645 - loss: 0.1363
Epoch 975/1000
7/7            0s 849us/step -
accuracy: 0.9670 - loss: 0.1188
Epoch 976/1000
7/7            0s 916us/step -
accuracy: 0.9806 - loss: 0.1224
Epoch 977/1000
7/7            0s 758us/step -
accuracy: 0.9740 - loss: 0.1213
Epoch 978/1000
7/7            0s 750us/step -
accuracy: 0.9692 - loss: 0.1443
Epoch 979/1000
7/7            0s 1ms/step -
accuracy: 0.9732 - loss: 0.1379
Epoch 980/1000
```

```
7/7            0s 813us/step -
accuracy: 0.9765 - loss: 0.1245
Epoch 981/1000
7/7            0s 1ms/step -
accuracy: 0.9651 - loss: 0.1299
Epoch 982/1000
7/7            0s 764us/step -
accuracy: 0.9748 - loss: 0.1411
Epoch 983/1000
7/7            0s 734us/step -
accuracy: 0.9752 - loss: 0.1187
Epoch 984/1000
7/7            0s 855us/step -
accuracy: 0.9735 - loss: 0.1227
Epoch 985/1000
7/7            0s 908us/step -
accuracy: 0.9766 - loss: 0.1080
Epoch 986/1000
7/7            0s 764us/step -
accuracy: 0.9724 - loss: 0.1271
Epoch 987/1000
7/7            0s 802us/step -
accuracy: 0.9823 - loss: 0.1194
Epoch 988/1000
7/7            0s 765us/step -
accuracy: 0.9824 - loss: 0.1272
Epoch 989/1000
7/7            0s 858us/step -
accuracy: 0.9747 - loss: 0.1338
Epoch 990/1000
7/7            0s 918us/step -
accuracy: 0.9802 - loss: 0.1314
Epoch 991/1000
7/7            0s 688us/step -
accuracy: 0.9783 - loss: 0.1340
Epoch 992/1000
7/7            0s 722us/step -
accuracy: 0.9778 - loss: 0.1222
Epoch 993/1000
7/7            0s 810us/step -
accuracy: 0.9849 - loss: 0.1207
Epoch 994/1000
7/7            0s 953us/step -
accuracy: 0.9750 - loss: 0.1195
Epoch 995/1000
7/7            0s 882us/step -
accuracy: 0.9682 - loss: 0.1408
Epoch 996/1000
```

```
7/7             0s 1ms/step -
accuracy: 0.9744 - loss: 0.1331
Epoch 997/1000
7/7             0s 820us/step -
accuracy: 0.9713 - loss: 0.1280
Epoch 998/1000
7/7             0s 977us/step -
accuracy: 0.9841 - loss: 0.1176
Epoch 999/1000
7/7             0s 933us/step -
accuracy: 0.9856 - loss: 0.1114
Epoch 1000/1000
7/7             0s 837us/step -
accuracy: 0.9815 - loss: 0.1202
Accuracies of the network on test data: 84.62%
Accuracies of the network on training data: 97.64%
```



5.3.2 Modellvergleich

Nach dem Training, Testen und Erhalten der jeweiligen Genauigkeiten für die ausgewählten Modelle besteht unser nächster Schritt darin, alle entwickelten Modelle zu vergleichen. Dieser Vergleich ist entscheidend für die Bewertung der Leistung jedes Modells und die fundierte Entscheidungsfindung, welches Modell am besten für unser spezifisches Problem geeignet ist (in unserem Fall die Vorhersage von Herzerkrankungen). Durch eine sorgfältige Analyse der erreichten Genauigkeiten über verschiedene Modelle hinweg, wie im vorherigen Abschnitt hervorgehoben, können wir das Modell identifizieren, das die höchste Klassifikationsleistung zeigt. Dieser informierte Entscheidungsprozess ermöglicht es uns, das optimale Modell für die Bereitstellung auszuwählen oder eine weitere Optimierung in Betracht zu ziehen, falls erforderlich.

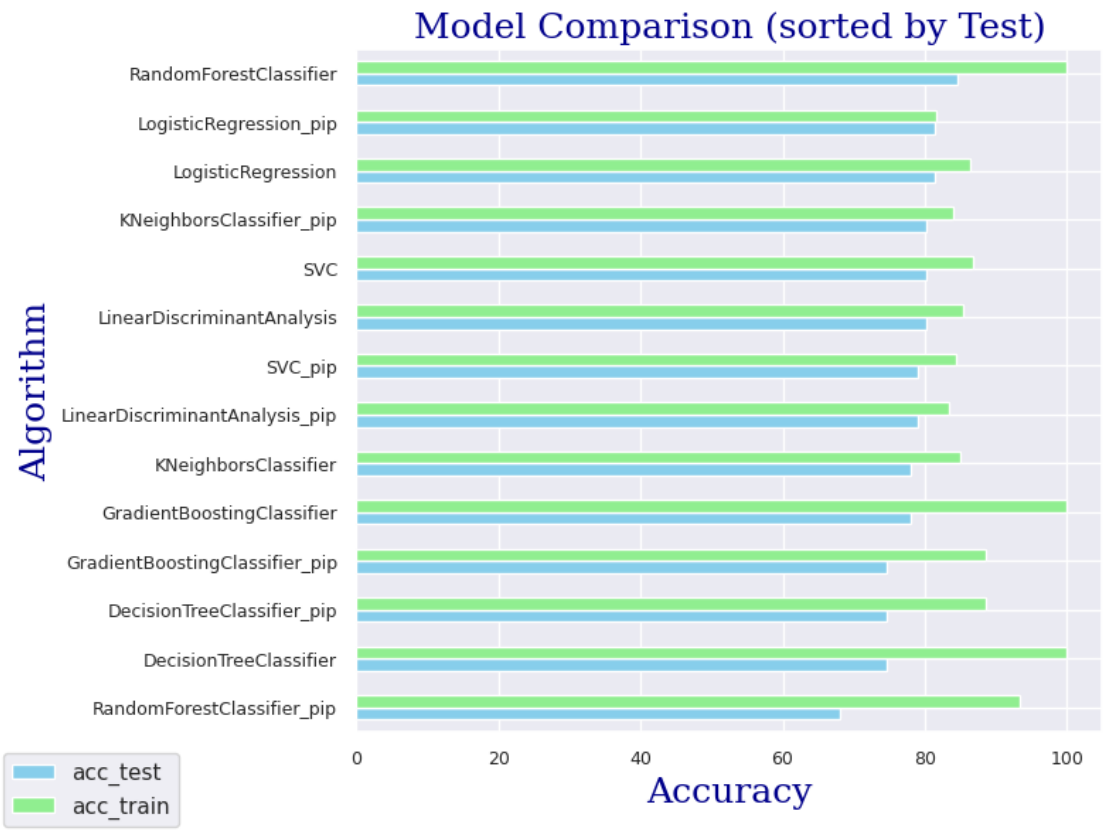
Dieser Abschnitt bietet einen prägnanten Überblick über die Genauigkeitswerte, die von verschiedenen Algorithmen des maschinellen Lernens im letzten Abschnitt zur Vorhersage von Herzerkrankungen erreicht wurden. Der Schwerpunkt liegt hier hauptsächlich auf den Genauigkeitsmetriken für sowohl die Test- als auch die Trainingsdatensätze. Die vorgestellte Tabelle zeigt eine sortierte Liste von Algorithmen zusammen mit ihren entsprechenden Testwerten.

```
results = evaluator.results
df_comparsion = pd.DataFrame.from_dict(results,
orient='index' ).transpose().sort_values('acc_test',ascending= True)
# Sorted  by Test Accuracy
aligned_df(direction='left', df=df_comparsion)
```

	acc_test	acc_train
RandomForestClassifier_pip	68.130000	93.400000
DecisionTreeClassifier	74.730000	100.000000
DecisionTreeClassifier_pip	74.730000	88.680000
GradientBoostingClassifier_pip	74.730000	88.680000
GradientBoostingClassifier	78.020000	100.000000
KNeighborsClassifier	78.020000	84.910000
LinearDiscriminantAnalysis_pip	79.120000	83.490000
SVC_pip	79.120000	84.430000
LinearDiscriminantAnalysis	80.220000	85.380000
SVC	80.220000	86.790000
KNeighborsClassifier_pip	80.220000	83.960000
LogisticRegression	81.320000	86.320000
LogisticRegression_pip	81.320000	81.600000
RandomForestClassifier	84.620000	100.000000

```
fg11 = df_comparsion.plot(kind='barh', color=['skyblue', 'lightgreen'],
                          fontsize=9, legend=False)
plt.figlegend( ncol=1, frameon=True,loc= 'lower left')
#plt.figure(dpi=1000)

fg11.set_xlabel('Accuracy', fontdict=font)
fg11.set_ylabel('Algorithm', fontdict=font)
fg11.set_title('Model Comparison (sorted by Test)',  fontdict=font)
```

```
plt.tight_layout()
```



6 Fazit

In dieser Analyse haben wir uns auf eine umfassende Reise durch die Entwicklung und Bewertung von datengetriebenen Klassifikationsmodellen für die Aufgabe der Herzkrankheitsvorhersage begeben. Beginnend mit der Auswahl relevanter Merkmale und der Erstellung eines ausgewogenen Trainings- und Testdatensatzes, sind wir in das Herz der Modellauswahl und -bewertung eingetaucht.

Der Abschnitt über das datengetriebene Modell beleuchtete die Bedeutung von datengetriebenen Ansätzen bei der Lösung von Klassifikationsproblemen. Durch die Unterteilung von Rechenmethoden in Klassifikationskategorien konzentrierten wir uns auf die Klassifikation - einen Prozess, der darauf abzielt, Klassenlabels für neue Daten auf der Grundlage ihrer Merkmale vorherzusagen. Die Merkmalsauswahl erwies sich als zentraler Schritt, bei dem die Identifizierung entscheidender Merkmale, die genaue Vorhersagen ermöglichen, im Vordergrund stand. Unsere Verwendung des Sequential Feature Selector (SFS) mit einer angepassten Konfiguration ermöglichte es uns, informative Merkmale zu fokussieren und Dimensionalitätsprobleme zu mildern.

Der Schritt der Train-Test-Split war eine grundlegende Vorbereitung, bei der der Datensatz in Trainings- und Testuntergruppen aufgeteilt wurde, um eine robuste Modellbewertung zu er-

möglichen.

Das Herz der Analyse, die Modellauswahl und -bewertung, zeigte unser Engagement für fundierte Entscheidungsfindung. Mit Hilfe der ModelEvaluator-Klasse bewerteten wir systematisch mehrere Klassifikationsalgorithmen auf der Grundlage wesentlicher Metriken wie Präzision, Recall, F1-Score, Support und Genauigkeiten. Dieser Ansatz gewährleistete nicht nur einen rigorosen Vergleich, sondern erleichterte auch die Visualisierung von ROC-Kurven und Verwechslungsmatrizen. Die Nutzung eines objektorientierten Programmierparadigmas verbesserte die Modularität und Wiederverwendbarkeit unseres Codes und untermauerte einen schlanken und umfassenden Bewertungsprozess.

Im Abschnitt Modellvergleich destillierten wir das Wesentliche unserer Bemühungen in eine prägnante, aber aufschlussreiche Tabelle mit Genauigkeitswerten. Diese Tabelle beleuchtete die Leistung verschiedener Algorithmen sowohl auf den Test- als auch auf den Trainingsdatensätzen und lieferte wertvolle Einblicke in ihre Generalisierungsfähigkeiten und potenzielle Überanpassungstendenzen.

Zum Abschluss ist es unerlässlich anzuerkennen, dass, obwohl die Genauigkeit eine wesentliche Metrik ist, der Bewertungsprozess vielschichtig sein sollte. Die Entscheidung für das beste Modell sollte durch Präzision, Recall, F1-Score, Fachwissen und andere kontextbezogene Überlegungen informiert sein.

Letztendlich dient diese Analyse als solide Grundlage für die Bereitstellung eines effektiven Klassifikationsmodells für die Vorhersage von Herzkrankheiten. Sie unterstreicht den iterativen und explorativen Charakter des maschinellen Lernens und ermutigt zur kontinuierlichen Optimierung, Feinabstimmung und Modellverbesserung. Mit diesem Wissen sind wir in der Lage, fundierte Entscheidungen in realen Szenarien zu treffen und zur Weiterentwicklung der Gesundheitsversorgung und datengetriebener Lösungen beizutragen.

Zurück zum Inhaltsverzeichnis