```
pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.46.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.6)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (24.1)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (3.1.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->shap) (1.16
```

```
pip install lime
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 4.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from lime) (3.8.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lime) (4.66.6)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from lime) (1.5.2)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.10/dist-packages (from lime) (0.24.0)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (3.4.2)
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (10.4.0)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2.36.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2024
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (24.1)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->lime) (3.5
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283834 sha256=1a5bec0f6b5870d035e26f1b9290f001a11edd4a8a113aef
  Stored in directory: /root/.cache/pip/wheels/fd/a2/af/9ac0a1a85a27f314a06b39e1f492bee1547d52549a4606ed89
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1
```

```
pip install dalex
```

```
Collecting dalex
  Downloading dalex-1.7.1.tar.gz (1.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 11.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dalex) (75.1.0)
Requirement already satisfied: pandas>=1.5.3 in /usr/local/lib/python3.10/dist-packages (from dalex) (2.2.2)
Requirement already satisfied: numpy>=1.23.3 in /usr/local/lib/python3.10/dist-packages (from dalex) (1.26.4)
Requirement already satisfied: scipy>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from dalex) (1.13.1)
Requirement already satisfied: plotly>=5.1.0 in /usr/local/lib/python3.10/dist-packages (from dalex) (5.24.1)
Requirement already satisfied: tqdm>=4.61.2 in /usr/local/lib/python3.10/dist-packages (from dalex) (4.66.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5.3->dalex) (2.8.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5.3->dalex) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.5.3->dalex) (2024.2)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.1.0->dalex) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.1.0->dalex) (24.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.5.3->dale
Building wheels for collected packages: dalex
  Building wheel for dalex (setup.py) ... done
  Created wheel for dalex: filename=dalex-1.7.1-py3-none-any.whl size=1042797 sha256=fbc6797e7bcd415f3e8ca45de1b49d1981aa124c68079e9
  Stored in directory: /root/.cache/pip/wheels/fe/c2/41/63e006b3312a4e17299bed5f83d985dea872368f9ab16bb20f
Successfully built dalex
Installing collected packages: dalex
Successfully installed dalex-1.7.1
```

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, precision_recall_fscore_support, roc_auc_score,roc_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
#from pytorch_tabnet.tab_model import TabNetClassifier
import torch
from imblearn.over_sampling import SMOTE
from sklearn.impute import SimpleImputer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, LSTM, Dense, Dropout,SimpleRNN
from sklearn.preprocessing import StandardScaler
from keras.callbacks import ModelCheckpoint


import shap
import lime
import lime.lime_tabular
import dalex as dx


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install torch torchmetrics pytorch-tabular
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.0+cu121)
Collecting torchmetrics
  Downloading torchmetrics-1.5.2-py3-none-any.whl.metadata (20 kB)
Collecting pytorch-tabular
  Downloading pytorch_tabular-1.1.0-py2.py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.26.4)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (24.1)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.11.8-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: pandas>=1.1.5 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.2.2)
Requirement already satisfied: scikit-learn>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (1.5.2)
Collecting pytorch-lightning<2.2.0,>=2.0.0 (from pytorch-tabular)
  Downloading pytorch_lightning-2.1.4-py3-none-any.whl.metadata (21 kB)
Collecting omegaconf>=2.3.0 (from pytorch-tabular)
  Downloading omegaconf-2.3.0-py3-none-any.whl.metadata (3.9 kB)
Collecting torchmetrics
  Downloading torchmetrics-1.2.1-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: tensorboard!=2.5.0,>2.2.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.17.
Requirement already satisfied: protobuf<4.26.0,>=3.20.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (3.20.3
Collecting pytorch-tabnet==4.1 (from pytorch-tabular)
  Downloading pytorch_tabnet-4.1.0-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: PyYAML<6.1.0,>=5.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (6.0.2)
Requirement already satisfied: matplotlib>3.1 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (3.8.0)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (7.7.1)
Collecting einops<0.8.0,>=0.6.0 (from pytorch-tabular)
  Downloading einops-0.7.0-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: rich>=11.0.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (13.9.3)
Requirement already satisfied: scipy>1.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabnet==4.1->pytorch-tabular) (
Requirement already satisfied: tqdm>=4.36 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabnet==4.1->pytorch-tabular)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetri
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular) (0.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular) (1
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabu
Collecting antlr4-python3-runtime==4.9.* (from omegaconf>=2.3.0->pytorch-tabular)
  Downloading antlr4-python3-runtime-4.9.3.tar.gz (117 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.0/117.0 kB 4.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.5->pytorch-tabular) (202
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.5->pytorch-tabular) (2
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.0.0->pytorch-tabul
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.0.0->pytorch-tab
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->pytorch-tabula
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->pytorch
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorch-t
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorch
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorc
```

```
!pip install --upgrade pytorch-tabular
```

```
Requirement already satisfied: pytorch-tabular in /usr/local/lib/python3.10/dist-packages (1.1.0)
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.5.0+cu121)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (1.26.4)
Requirement already satisfied: pandas>=1.1.5 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.2.2)
Requirement already satisfied: scikit-learn>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (1.5.2)
Requirement already satisfied: pytorch-lightning<2.2.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular)
Requirement already satisfied: omegaconf>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.3.0)
Requirement already satisfied: torchmetrics<1.3.0,>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (1.2
Requirement already satisfied: tensorboard!=2.5.0,>2.2.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (2.17.
Requirement already satisfied: protobuf<4.26.0,>=3.20.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (3.20.3
Requirement already satisfied: pytorch-tabnet==4.1 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (4.1.0)
Requirement already satisfied: PyYAML<6.1.0,>=5.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (6.0.2)
Requirement already satisfied: matplotlib>3.1 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (3.8.0)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (7.7.1)
Requirement already satisfied: einops<0.8.0,>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (0.7.0)
Requirement already satisfied: rich>=11.0.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabular) (13.9.3)
Requirement already satisfied: scipy>1.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabnet==4.1->pytorch-tabular) (
Requirement already satisfied: tqdm>=4.36 in /usr/local/lib/python3.10/dist-packages (from pytorch-tabnet==4.1->pytorch-tabular)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular) (0.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular) (1
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabular)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>3.1->pytorch-tabu
Requirement already satisfied: antlr4-python3-runtime==4.9.* in /usr/local/lib/python3.10/dist-packages (from omegaconf>=2.3.0->p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.5->pytorch-tabular) (202
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.5->pytorch-tabular) (2
Requirement already satisfied: fsspec>=2022.5.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2022.5.0->pytorch-
```

```
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning<2.2.0,
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning<2.2.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.0.0->pytorch-tabul
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich>=11.0.0->pytorch-tab
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->pytorch-tabula
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.3.0->pytorch
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorch-t
Requirement already satisfied: grpcio>=1.48.2 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorch
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorc
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pyt
Requirement already satisfied: six>1.9 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorch-tabula
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard!=2.5.0,>2.2.0->pytorc
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.11.0->pytorch-tabular) (3.16.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.11.0->pytorch-tabular) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.11.0->pytorch-tabular) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.11.0->pytorch-tabular) (1.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.11.0->
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tabular) (5.
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tabul
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tabular) (5.
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tab
Requirement already satisfied: ipython>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tabular) (7.34
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->pytorch-tab
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2022.5.0
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets->pyto
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets->pytorc

Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->noteboo
```

```
import pandas as pd

# Load the dataset
columns = ['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset']
df = pd.read_csv("indian_liver_patient.csv", header=0,names=columns)

# Display basic information about the dataset
print("Shape of the dataset:", df.shape)
print("\nData types of columns:")
print(df.dtypes)
print("\nPreview of the first few rows:")
print(df.head())
print("\nSummary statistics:")
print(df.describe())
```

```
Shape of the dataset: (583, 11)

Data types of columns:
Age                           int64
Gender                       object
Total_Bilirubin             float64
Direct_Bilirubin            float64
Alkaline_Phosphotase          int64
Alamine_Aminotransferase      int64
Aspartate_Aminotransferase    int64
Total_Protiens              float64
Albumin                     float64
Albumin_and_Globulin_Ratio  float64
Dataset                       int64
dtype: object

Preview of the first few rows:
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0   65  Female              0.7               0.1                   187
1   62    Male             10.9               5.5                   699
2   62    Male              7.3               4.1                   490
3   58    Male              1.0               0.4                   182
4   72    Male              3.9               2.0                   195

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Dataset
0      3.3                        0.90        1
1      3.2                        0.74        1
2      3.3                        0.89        1
3      3.4                        1.00        1
4      2.4                        0.40        1

Summary statistics:
              Age  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
count  583.000000       583.000000        583.000000            583.000000
```

```
mean    44.746141         3.298799          1.486106           290.576329
std     16.189833         6.209522          2.808498           242.937989
min      4.000000         0.400000          0.100000            63.000000
25%     33.000000         0.800000          0.200000           175.500000
50%     45.000000         1.000000          0.300000           208.000000
75%     58.000000         2.600000          1.300000           298.000000
max     90.000000        75.000000         19.700000          2110.000000

       Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
count                583.000000                  583.000000      583.000000
mean                  80.713551                  109.910806        6.483190
std                  182.620356                  288.918529        1.085451
min                   10.000000                   10.000000        2.700000
25%                   23.000000                   25.000000        5.800000
50%                   35.000000                   42.000000        6.600000
75%                   60.500000                   87.000000        7.200000
max                 2000.000000                 4929.000000        9.600000
```

```
# Preprocessing
# Encoding categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])  # Encode 'Gender'

# Separating features and target
X = df.drop('Dataset', axis=1)  # Features
y = df['Dataset']  # Target



imputer = SimpleImputer(strategy='mean') # You can choose other strategies like 'median' or 'most_frequent'

# Fit the imputer on your training data and transform both training and testing data
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

smote = SMOTE(random_state=42)
X, y= smote.fit_resample(X, y)

# Scaling numerical features
scaler = StandardScaler()
X[['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
   'Alamine_Aminotransferase', 'Aspartate_Aminotransferase', 'Total_Protiens',
   'Albumin', 'Albumin_and_Globulin_Ratio']] = scaler.fit_transform(X[['Age', 'Total_Bilirubin', 'Direct_Bilirubin',
   'Alkaline_Phosphotase', 'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
   'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio']])

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


X_test.isna().sum()
```

	0
Age	0
Gender	0
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	0
Alamine_Aminotransferase	0
Aspartate_Aminotransferase	0
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	0

dtype: int64

```
print(y.value_counts())
```

```
Dataset
1    416
2    416
Name: count, dtype: int64
```

```
# prompt: change each value in y_combined to 0 if it is 1 and 1 if it is 2

y = np.where(y == 1, 0, np.where(y == 2, 1, y))
```

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = np.array(X).reshape(X.shape[0], X.shape[1], 1)  # Shape (samples, features, channels)


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

# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)


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


X_train
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransfera
477	0.697983	1.000000	-0.385832	-0.396968	-0.205877	-0.270066	-0.2285
346	0.136345	1.000000	-0.385832	-0.396968	-0.697523	-0.334696	-0.3144
462	-0.362888	1.000000	-0.366998	-0.396968	-0.720935	-0.282992	-0.2857
671	1.188481	0.427996	-0.375059	-0.414698	-0.375790	-0.284828	-0.2571
302	0.385962	0.000000	-0.178657	-0.231271	0.777414	-0.244215	-0.2285
...	...	...	...	...	...	...	
71	1.946066	0.000000	-0.348164	-0.396968	-0.355712	-0.295918	-0.2407
106	-0.487696	1.000000	0.499369	0.472941	-0.557053	-0.218363	0.0169
270	-0.425292	1.000000	-0.366998	-0.396968	-0.135642	0.195263	-0.1384
435	-1.673375	0.000000	-0.404666	-0.438392	-0.271430	-0.244215	-0.2735
102	-1.673375	1.000000	-0.329330	-0.396968	-0.187148	-0.192511	-0.1753

665 rows × 10 columns

```
y_train_categorical.shape
```

```
(665, 2)
```

```
# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(SimpleRNN(100, return_sequences=True))
model.add(SimpleRNN(75, return_sequences=True))
model.add(SimpleRNN(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_rnn.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
```

```
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6105 - loss: 0.6735
Epoch 1: val_accuracy improved from -inf to 0.61078, saving model to best_model_rnn.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.6234 - loss: 0.6568 - val_accuracy: 0.6108 - val_loss: 0.6378
Epoch 2/200
18/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6930 - loss: 0.5987
Epoch 2: val_accuracy improved from 0.61078 to 0.73054, saving model to best_model_rnn.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6915 - loss: 0.5992 - val_accuracy: 0.7305 - val_loss: 0.5351
Epoch 3/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7126 - loss: 0.5399
Epoch 3: val_accuracy did not improve from 0.73054
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7116 - loss: 0.5431 - val_accuracy: 0.7305 - val_loss: 0.5267
Epoch 4/200
15/21 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7087 - loss: 0.5551
Epoch 4: val_accuracy did not improve from 0.73054
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7127 - loss: 0.5518 - val_accuracy: 0.7186 - val_loss: 0.5241
Epoch 5/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6939 - loss: 0.5360
Epoch 5: val_accuracy improved from 0.73054 to 0.73653, saving model to best_model_rnn.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6958 - loss: 0.5361 - val_accuracy: 0.7365 - val_loss: 0.5187
Epoch 6/200
16/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7135 - loss: 0.5321
Epoch 6: val_accuracy did not improve from 0.73653
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7112 - loss: 0.5336 - val_accuracy: 0.7305 - val_loss: 0.5183
Epoch 7/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7019 - loss: 0.5793
Epoch 7: val_accuracy did not improve from 0.73653
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7052 - loss: 0.5732 - val_accuracy: 0.7365 - val_loss: 0.5093
Epoch 8/200
19/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7184 - loss: 0.5448
Epoch 8: val_accuracy improved from 0.73653 to 0.74850, saving model to best_model_rnn.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7207 - loss: 0.5430 - val_accuracy: 0.7485 - val_loss: 0.5068
Epoch 9/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7025 - loss: 0.5465
Epoch 9: val_accuracy improved from 0.74850 to 0.75449, saving model to best_model_rnn.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7035 - loss: 0.5457 - val_accuracy: 0.7545 - val_loss: 0.5020
Epoch 10/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7660 - loss: 0.4950
Epoch 10: val_accuracy did not improve from 0.75449
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7579 - loss: 0.5019 - val_accuracy: 0.7365 - val_loss: 0.5292
Epoch 11/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7183 - loss: 0.5277
Epoch 11: val_accuracy did not improve from 0.75449
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7236 - loss: 0.5261 - val_accuracy: 0.7425 - val_loss: 0.5029
Epoch 12/200
16/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7163 - loss: 0.5513
Epoch 12: val_accuracy did not improve from 0.75449
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7135 - loss: 0.5474 - val_accuracy: 0.7246 - val_loss: 0.5269
Epoch 13/200
15/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7158 - loss: 0.5489
Epoch 13: val_accuracy did not improve from 0.75449
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7160 - loss: 0.5441 - val_accuracy: 0.7425 - val_loss: 0.5106
Epoch 14/200
16/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7358 - loss: 0.5050
Epoch 14: val_accuracy did not improve from 0.75449
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7344 - loss: 0.5077 - val_accuracy: 0.7485 - val_loss: 0.5032
Epoch 15/200
17/21 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7227 - loss: 0.5219
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_rnn.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')
```

```
# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8458 - loss: 0.4748
Test Accuracy: 0.8623
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 114ms/step
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        75
           1       0.89      0.86      0.87        92

    accuracy                           0.86       167
   macro avg       0.86      0.86      0.86       167
weighted avg       0.86      0.86      0.86       167
```

```
# Load the best model
best_model = load_model('best_model_rnn.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/rnn_model_ind.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
best_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 8, 32)	128
max_pooling1d (MaxPooling1D)	(None, 4, 32)	0
dropout (Dropout)	(None, 4, 32)	0
simple_rnn (SimpleRNN)	(None, 4, 100)	13,300
simple_rnn_1 (SimpleRNN)	(None, 4, 75)	13,200
simple_rnn_2 (SimpleRNN)	(None, 50)	6,300
dense (Dense)	(None, 2)	102

Total params: 99,092 (387.08 KB)
Trainable params: 33,030 (129.02 KB)
Non-trainable params: 0 (0.00 B)

```
# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(75, return_sequences=True))
model.add(LSTM(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
```

```
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
19/21 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.5836 - loss: 0.6777
Epoch 1: val_accuracy improved from -inf to 0.58084, saving model to best_model.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 79ms/step - accuracy: 0.5881 - loss: 0.6739 - val_accuracy: 0.5808 - val_loss: 0.6162
Epoch 2/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6407 - loss: 0.5960
Epoch 2: val_accuracy improved from 0.58084 to 0.62275, saving model to best_model.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6418 - loss: 0.5956 - val_accuracy: 0.6228 - val_loss: 0.5916
Epoch 3/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6810 - loss: 0.5976
Epoch 3: val_accuracy improved from 0.62275 to 0.71856, saving model to best_model.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6817 - loss: 0.5966 - val_accuracy: 0.7186 - val_loss: 0.5658
Epoch 4/200
19/21 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7006 - loss: 0.5766
Epoch 4: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6993 - loss: 0.5762 - val_accuracy: 0.7186 - val_loss: 0.5595
Epoch 5/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6871 - loss: 0.5627
Epoch 5: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.6876 - loss: 0.5634 - val_accuracy: 0.6886 - val_loss: 0.5719
Epoch 6/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7198 - loss: 0.5483
Epoch 6: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7189 - loss: 0.5493 - val_accuracy: 0.7006 - val_loss: 0.5628
Epoch 7/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6836 - loss: 0.5716
Epoch 7: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6834 - loss: 0.5717 - val_accuracy: 0.7066 - val_loss: 0.5688
Epoch 8/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6781 - loss: 0.5767
Epoch 8: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6791 - loss: 0.5756 - val_accuracy: 0.7066 - val_loss: 0.5535
Epoch 9/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6772 - loss: 0.5943
Epoch 9: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6776 - loss: 0.5933 - val_accuracy: 0.7186 - val_loss: 0.5616
Epoch 10/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6792 - loss: 0.5644
Epoch 10: val_accuracy did not improve from 0.71856
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6796 - loss: 0.5643 - val_accuracy: 0.7186 - val_loss: 0.5560
Epoch 11/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6944 - loss: 0.5507
Epoch 11: val_accuracy improved from 0.71856 to 0.72455, saving model to best_model.keras
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6950 - loss: 0.5516 - val_accuracy: 0.7246 - val_loss: 0.5552
Epoch 12/200
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6886 - loss: 0.5562
Epoch 12: val_accuracy did not improve from 0.72455
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6889 - loss: 0.5564 - val_accuracy: 0.7126 - val_loss: 0.5559
Epoch 13/200
19/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6905 - loss: 0.5527
Epoch 13: val_accuracy did not improve from 0.72455
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6915 - loss: 0.5526 - val_accuracy: 0.7066 - val_loss: 0.5519
Epoch 14/200
20/21 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7046 - loss: 0.5509
Epoch 14: val_accuracy did not improve from 0.72455
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7045 - loss: 0.5507 - val_accuracy: 0.7066 - val_loss: 0.5609
Epoch 15/200
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
```

```
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7669 - loss: 0.5946
Test Accuracy: 0.8024
WARNING:tensorflow:5 out of the last 13 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distrib
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step
              precision    recall  f1-score   support

           0       0.80      0.75      0.77        75
           1       0.80      0.85      0.83        92

    accuracy                           0.80       167
   macro avg       0.80      0.80      0.80       167
weighted avg       0.80      0.80      0.80       167
```

```
# Load the best model
best_model = load_model('best_model.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/lstm_model_ind.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
X = X.values
y=y.values
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-121-d364a71ec970> in <cell line: 2>()
      1 X = X.values
----> 2 y=y.values

AttributeError: 'numpy.ndarray' object has no attribute 'values'
```

```
import numpy as np

def augment_data(X, y, num_augmentations=5):
    X_augmented = []
    y_augmented = []

    for i in range(len(X)):
        # Original sample
        X_augmented.append(X[i])
        y_augmented.append(y[i])

        for _ in range(num_augmentations):
            # Create new augmented samples
            sample = X[i]

            # Add noise
            noise = np.random.normal(0, 0.01, sample.shape)
            augmented_sample = sample + noise

            # Time shifting
            if np.random.rand() > 0.5:  # Randomly decide to shift
                shift = np.random.randint(-5, 5)  # Shift by a random amount
                augmented_sample = np.roll(augmented_sample, shift)

            # Append augmented sample and corresponding label
            X_augmented.append(augmented_sample)
            y_augmented.append(y[i])

    return np.array(X_augmented), np.array(y_augmented)

# Generate augmented data
X_augmented, y_augmented = augment_data(X, y)

# Optionally, you can concatenate the original and augmented data
X_combined = np.concatenate((X, X_augmented), axis=0)
y_combined = np.concatenate((y, y_augmented), axis=0)
```

```
X_combined.shape
```

```
(5824, 10)
```

```
print("Unique classes in y_train:", np.unique(y_combined))
```

```
Unique classes in y_train: [0 1]
```

```
print("Unique classes in y_train:", np.unique(y_combined))
```

```
Unique classes in y_train: [0 1]
```

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = X_combined.reshape(X_combined.shape[0], X_combined.shape[1], 1)  # Shape (samples, features, channels)


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_combined, test_size=0.2, random_state=42)

# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)
```

```
y_train[2]
```

```
0
```

```
print(y_train_categorical.shape)
```

```
(4659, 2)
```

```
# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(SimpleRNN(100, return_sequences=True))
model.add(SimpleRNN(75, return_sequences=True))
model.add(SimpleRNN(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_rnn2.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
```

```
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6505 - loss: 0.6408
Epoch 1: val_accuracy improved from -inf to 0.66867, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.6505 - loss: 0.6405 - val_accuracy: 0.6687 - val_loss: 0.5932
Epoch 2/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6701 - loss: 0.5944
Epoch 2: val_accuracy improved from 0.66867 to 0.68584, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.6703 - loss: 0.5941 - val_accuracy: 0.6858 - val_loss: 0.5760
Epoch 3/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6793 - loss: 0.5948
Epoch 3: val_accuracy improved from 0.68584 to 0.70129, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.6796 - loss: 0.5945 - val_accuracy: 0.7013 - val_loss: 0.5541
Epoch 4/200
141/146 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7109 - loss: 0.5492
Epoch 4: val_accuracy improved from 0.70129 to 0.72103, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7101 - loss: 0.5497 - val_accuracy: 0.7210 - val_loss: 0.5395
Epoch 5/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6960 - loss: 0.5621
Epoch 5: val_accuracy did not improve from 0.72103
146/146 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.6961 - loss: 0.5621 - val_accuracy: 0.7202 - val_loss: 0.5322
Epoch 6/200
142/146 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7012 - loss: 0.5574
Epoch 6: val_accuracy improved from 0.72103 to 0.73820, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.7015 - loss: 0.5570 - val_accuracy: 0.7382 - val_loss: 0.5191
Epoch 7/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7277 - loss: 0.5240
Epoch 7: val_accuracy improved from 0.73820 to 0.75193, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.7277 - loss: 0.5240 - val_accuracy: 0.7519 - val_loss: 0.5105
Epoch 8/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7418 - loss: 0.5183
Epoch 8: val_accuracy did not improve from 0.75193
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7418 - loss: 0.5183 - val_accuracy: 0.7356 - val_loss: 0.5122
Epoch 9/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7376 - loss: 0.5003
Epoch 9: val_accuracy improved from 0.75193 to 0.76996, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.7377 - loss: 0.5003 - val_accuracy: 0.7700 - val_loss: 0.4807
Epoch 10/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7298 - loss: 0.5172
Epoch 10: val_accuracy did not improve from 0.76996
146/146 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.7298 - loss: 0.5172 - val_accuracy: 0.7459 - val_loss: 0.5084
Epoch 11/200
144/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7554 - loss: 0.4835
Epoch 11: val_accuracy did not improve from 0.76996
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7555 - loss: 0.4834 - val_accuracy: 0.7657 - val_loss: 0.4850
Epoch 12/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7536 - loss: 0.4829
Epoch 12: val_accuracy improved from 0.76996 to 0.77253, saving model to best_model_rnn2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.7537 - loss: 0.4828 - val_accuracy: 0.7725 - val_loss: 0.4547
Epoch 13/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7719 - loss: 0.4510
Epoch 13: val_accuracy did not improve from 0.77253
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.7719 - loss: 0.4511 - val_accuracy: 0.7717 - val_loss: 0.4635
Epoch 14/200
145/146 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7725 - loss: 0.4502
Epoch 14: val_accuracy did not improve from 0.77253
146/146 ━━━━━━━━━━━━━━━━━━━━ 6s 27ms/step - accuracy: 0.7724 - loss: 0.4504 - val_accuracy: 0.7717 - val_loss: 0.4587
Epoch 15/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7809 - loss: 0.4485
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_rnn2.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9148 - loss: 0.3501
Test Accuracy: 0.9236
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       580
           1       0.91      0.94      0.92       585

    accuracy                           0.92      1165
   macro avg       0.92      0.92      0.92      1165
weighted avg       0.92      0.92      0.92      1165
```

```
# Load the best model
best_model = load_model('best_model_rnn2.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/rnn_model2_ind.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(75, return_sequences=True))
model.add(LSTM(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_lstm2.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6077 - loss: 0.6434
Epoch 1: val_accuracy improved from -inf to 0.66094, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.6079 - loss: 0.6433 - val_accuracy: 0.6609 - val_loss: 0.6122
Epoch 2/200
145/146 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6653 - loss: 0.6088
Epoch 2: val_accuracy improved from 0.66094 to 0.67983, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6653 - loss: 0.6087 - val_accuracy: 0.6798 - val_loss: 0.5954
Epoch 3/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6625 - loss: 0.5982
Epoch 3: val_accuracy improved from 0.67983 to 0.68670, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.6626 - loss: 0.5982 - val_accuracy: 0.6867 - val_loss: 0.5782
Epoch 4/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6918 - loss: 0.5823
Epoch 4: val_accuracy improved from 0.68670 to 0.70129, saving model to best_model_lstm2.keras
```

```
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.6918 - loss: 0.5823 - val_accuracy: 0.7013 - val_loss: 0.5639
Epoch 5/200
142/146 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6922 - loss: 0.5760
Epoch 5: val_accuracy improved from 0.70129 to 0.70472, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.6921 - loss: 0.5759 - val_accuracy: 0.7047 - val_loss: 0.5538
Epoch 6/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6880 - loss: 0.5696
Epoch 6: val_accuracy improved from 0.70472 to 0.70987, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6881 - loss: 0.5695 - val_accuracy: 0.7099 - val_loss: 0.5473
Epoch 7/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7167 - loss: 0.5444
Epoch 7: val_accuracy improved from 0.70987 to 0.71502, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7166 - loss: 0.5445 - val_accuracy: 0.7150 - val_loss: 0.5420
Epoch 8/200
145/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6998 - loss: 0.5567
Epoch 8: val_accuracy improved from 0.71502 to 0.71931, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6998 - loss: 0.5567 - val_accuracy: 0.7193 - val_loss: 0.5338
Epoch 9/200
144/146 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7061 - loss: 0.5503
Epoch 9: val_accuracy did not improve from 0.71931
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.7061 - loss: 0.5503 - val_accuracy: 0.7193 - val_loss: 0.5321
Epoch 10/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7192 - loss: 0.5468
Epoch 10: val_accuracy improved from 0.71931 to 0.72876, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.7192 - loss: 0.5468 - val_accuracy: 0.7288 - val_loss: 0.5172
Epoch 11/200
142/146 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7208 - loss: 0.5321
Epoch 11: val_accuracy did not improve from 0.72876
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.7208 - loss: 0.5322 - val_accuracy: 0.7279 - val_loss: 0.5215
Epoch 12/200
143/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7255 - loss: 0.5229
Epoch 12: val_accuracy did not improve from 0.72876
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.7253 - loss: 0.5231 - val_accuracy: 0.7262 - val_loss: 0.5182
Epoch 13/200
145/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7237 - loss: 0.5316
Epoch 13: val_accuracy did not improve from 0.72876
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7237 - loss: 0.5316 - val_accuracy: 0.7236 - val_loss: 0.5272
Epoch 14/200
144/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7270 - loss: 0.5196
Epoch 14: val_accuracy improved from 0.72876 to 0.74592, saving model to best_model_lstm2.keras
146/146 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7270 - loss: 0.5196 - val_accuracy: 0.7459 - val_loss: 0.5077
Epoch 15/200
146/146 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7353 - loss: 0.5102
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_lstm2.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9204 - loss: 0.4608
Test Accuracy: 0.9279
37/37 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step
              precision    recall  f1-score   support

           0       0.94      0.91      0.93       580
           1       0.91      0.95      0.93       585

    accuracy                           0.93      1165
   macro avg       0.93      0.93      0.93      1165
weighted avg       0.93      0.93      0.93      1165
```

```
# Load the best model
best_model = load_model('best_model_lstm2.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/lstm_model2_ind.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
# Convert X_train and X_test to NumPy arrays
```

```
X_train_np = X_train.values
X_test_np = X_test.values


X_train, X_test, y_train, y_test = train_test_split(X_combined, y_combined, test_size=0.2, random_state=42)


tabnet_model = TabNetClassifier(optimizer_fn=torch.optim.Adam,
                                optimizer_params=dict(lr=0.0001)) # Pass learning rate to optimizer_params
# Fit the model using NumPy arrays
tabnet_model.fit(X_train, y_train,
                 eval_set=[(X_test, y_test)],
                 eval_name=['test'],
                 eval_metric=['accuracy'],
                 max_epochs=1000,
                 patience=100)
```

```
epoch 0  | loss: 0.77791 | test_accuracy: 0.48584 |  0:00:00s
epoch 1  | loss: 0.77928 | test_accuracy: 0.47811 |  0:00:00s
epoch 2  | loss: 0.77442 | test_accuracy: 0.48069 |  0:00:01s
epoch 3  | loss: 0.77558 | test_accuracy: 0.48755 |  0:00:01s
epoch 4  | loss: 0.7711  | test_accuracy: 0.49871 |  0:00:02s
epoch 5  | loss: 0.76733 | test_accuracy: 0.50215 |  0:00:02s
epoch 6  | loss: 0.76494 | test_accuracy: 0.49614 |  0:00:03s
epoch 7  | loss: 0.75717 | test_accuracy: 0.50043 |  0:00:03s
epoch 8  | loss: 0.76293 | test_accuracy: 0.50472 |  0:00:04s
epoch 9  | loss: 0.76324 | test_accuracy: 0.49871 |  0:00:04s
epoch 10 | loss: 0.75795 | test_accuracy: 0.503   |  0:00:04s
epoch 11 | loss: 0.75533 | test_accuracy: 0.51159 |  0:00:04s
epoch 12 | loss: 0.7499  | test_accuracy: 0.50901 |  0:00:05s
epoch 13 | loss: 0.76667 | test_accuracy: 0.50901 |  0:00:05s
epoch 14 | loss: 0.76091 | test_accuracy: 0.50558 |  0:00:05s
epoch 15 | loss: 0.75131 | test_accuracy: 0.5073  |  0:00:05s
epoch 16 | loss: 0.75801 | test_accuracy: 0.5073  |  0:00:06s
epoch 17 | loss: 0.75305 | test_accuracy: 0.50901 |  0:00:06s
epoch 18 | loss: 0.74771 | test_accuracy: 0.50815 |  0:00:06s
epoch 19 | loss: 0.74527 | test_accuracy: 0.50215 |  0:00:06s
epoch 20 | loss: 0.74801 | test_accuracy: 0.50815 |  0:00:07s
epoch 21 | loss: 0.73476 | test_accuracy: 0.51073 |  0:00:07s
epoch 22 | loss: 0.73832 | test_accuracy: 0.51502 |  0:00:07s
epoch 23 | loss: 0.73898 | test_accuracy: 0.51159 |  0:00:07s
epoch 24 | loss: 0.73058 | test_accuracy: 0.51845 |  0:00:08s
epoch 25 | loss: 0.72669 | test_accuracy: 0.52532 |  0:00:08s
epoch 26 | loss: 0.73334 | test_accuracy: 0.52704 |  0:00:08s
epoch 27 | loss: 0.72657 | test_accuracy: 0.52532 |  0:00:08s
epoch 28 | loss: 0.73403 | test_accuracy: 0.52704 |  0:00:09s
epoch 29 | loss: 0.73376 | test_accuracy: 0.52704 |  0:00:09s
epoch 30 | loss: 0.72394 | test_accuracy: 0.5279  |  0:00:09s
epoch 31 | loss: 0.7291  | test_accuracy: 0.52361 |  0:00:10s
epoch 32 | loss: 0.71664 | test_accuracy: 0.52275 |  0:00:10s
epoch 33 | loss: 0.73696 | test_accuracy: 0.52532 |  0:00:10s
epoch 34 | loss: 0.71474 | test_accuracy: 0.52961 |  0:00:11s
epoch 35 | loss: 0.73008 | test_accuracy: 0.53391 |  0:00:11s
epoch 36 | loss: 0.73435 | test_accuracy: 0.52704 |  0:00:12s
epoch 37 | loss: 0.72294 | test_accuracy: 0.53476 |  0:00:12s
epoch 38 | loss: 0.72928 | test_accuracy: 0.54077 |  0:00:12s
epoch 39 | loss: 0.73148 | test_accuracy: 0.53391 |  0:00:13s
epoch 40 | loss: 0.71995 | test_accuracy: 0.54249 |  0:00:13s
epoch 41 | loss: 0.71614 | test_accuracy: 0.54506 |  0:00:14s
epoch 42 | loss: 0.72324 | test_accuracy: 0.54335 |  0:00:14s
epoch 43 | loss: 0.72377 | test_accuracy: 0.54592 |  0:00:14s
epoch 44 | loss: 0.71623 | test_accuracy: 0.54163 |  0:00:14s
epoch 45 | loss: 0.71981 | test_accuracy: 0.54764 |  0:00:15s
epoch 46 | loss: 0.71661 | test_accuracy: 0.5382  |  0:00:15s
epoch 47 | loss: 0.71111 | test_accuracy: 0.55279 |  0:00:15s
epoch 48 | loss: 0.70537 | test_accuracy: 0.54335 |  0:00:16s
epoch 49 | loss: 0.70725 | test_accuracy: 0.5588  |  0:00:16s
epoch 50 | loss: 0.70706 | test_accuracy: 0.55021 |  0:00:16s
epoch 51 | loss: 0.71022 | test_accuracy: 0.55107 |  0:00:16s
epoch 52 | loss: 0.70785 | test_accuracy: 0.55536 |  0:00:17s
epoch 53 | loss: 0.71267 | test_accuracy: 0.55107 |  0:00:17s
epoch 54 | loss: 0.70904 | test_accuracy: 0.55622 |  0:00:17s
epoch 55 | loss: 0.71557 | test_accuracy: 0.55193 |  0:00:17s
epoch 56 | loss: 0.70962 | test_accuracy: 0.56567 |  0:00:18s
epoch 57 | loss: 0.70848 | test_accuracy: 0.5691  |  0:00:18s
```

```
# Make predictions
y_pred = tabnet_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6961
                  precision    recall  f1-score   support
```

```
           1       0.74      0.61      0.67       580
           2       0.67      0.78      0.72       585

    accuracy                           0.70      1165
   macro avg       0.70      0.70      0.69      1165
weighted avg       0.70      0.70      0.69      1165
```

```
y_train.dtypes
```

```
dtype('int64')
```

```
# Load the best model
tabnet_model.save_model('/content/drive/My Drive/Honors Deep Learning/tabnet_model_ind.h5')  # Saves the model as an HDF5 file
```

```
Successfully saved model at /content/drive/My Drive/Honors Deep Learning/tabnet_model_ind.h5.zip
'/content/drive/My Drive/Honors Deep Learning/tabnet_model_ind.h5.zip'
```

```
# Define the model components
def create_mlp(hidden_units, dropout_rate, activation, normalization_layer, name=None):
    mlp_layers = []
    for units in hidden_units:
        mlp_layers.append(normalization_layer())
        mlp_layers.append(layers.Dense(units, activation=activation))
        mlp_layers.append(layers.Dropout(dropout_rate))

    return keras.Sequential(mlp_layers, name=name)

def create_tabtransformer_classifier(
    num_transformer_blocks,
    num_heads,
    mlp_hidden_units_factors,
    dropout_rate,
):
    # Create model inputs.
    inputs = layers.Input(shape=(X_train.shape[1],))

    # Create a dense layer to prepare input for transformer.
    x = layers.Dense(64)(inputs)  # Transform the input dimension
    x = layers.LayerNormalization()(x)

    # Create multiple layers of the Transformer block.
    for block_idx in range(num_transformer_blocks):
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads,
            key_dim=64,
            dropout=dropout_rate,
            attention_axes=1
        )(x, x)

        # Skip connection 1.
        x = layers.Add()([attention_output, x])
        # Layer normalization 1.
        x = layers.LayerNormalization()(x)

        # Feedforward.
        feedforward_output = create_mlp(
            hidden_units=[64],  # Keeping it consistent with the input dimension
            dropout_rate=dropout_rate,
            activation=keras.activations.gelu,
            normalization_layer=layers.LayerNormalization,
            name=f"feedforward_{block_idx}",
        )(x)

        # Skip connection 2.
        x = layers.Add()([feedforward_output, x])
        # Layer normalization 2.
        x = layers.LayerNormalization()(x)

    # Final MLP
    mlp_hidden_units = [int(factor * x.shape[-1]) for factor in mlp_hidden_units_factors]
    x = create_mlp(
        hidden_units=mlp_hidden_units,
        dropout_rate=dropout_rate,
        activation=keras.activations.selu,
        normalization_layer=layers.BatchNormalization,
        name="MLP",
    )(x)

    # Add a sigmoid as a binary classifier.
```

```
    outputs = layers.Dense(units=1, activation="sigmoid")(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model

# Create the Tab Transformer classifier
model = create_tabtransformer_classifier(
    num_transformer_blocks=2,
    num_heads=4,
    mlp_hidden_units_factors=[0.5, 0.5],
    dropout_rate=0.1,
)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Model Loss: {loss}, Model Accuracy: {accuracy}")
```

```
Epoch 1/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.3089 - loss: -0.2350 - val_accuracy: 0.3959 - val_loss: -6.6492
Epoch 2/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.4498 - loss: -8.3750 - val_accuracy: 0.4936 - val_loss: -36.7935
Epoch 3/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.5082 - loss: -39.1240 - val_accuracy: 0.4946 - val_loss: -76.3413
Epoch 4/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.5060 - loss: -107.9595 - val_accuracy: 0.4946 - val_loss: -103.3192
Epoch 5/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5047 - loss: -211.5944 - val_accuracy: 0.4946 - val_loss: -382.0914
Epoch 6/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5179 - loss: -401.2961 - val_accuracy: 0.4946 - val_loss: -614.7556
Epoch 7/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5080 - loss: -693.4138 - val_accuracy: 0.4946 - val_loss: -856.4550
Epoch 8/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5053 - loss: -1041.4093 - val_accuracy: 0.4946 - val_loss: -1447.0804
Epoch 9/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5073 - loss: -1533.5580 - val_accuracy: 0.4946 - val_loss: -1885.8180
Epoch 10/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4910 - loss: -2169.2092 - val_accuracy: 0.4946 - val_loss: -2617.8909
Epoch 11/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.4980 - loss: -2886.2290 - val_accuracy: 0.4946 - val_loss: -3945.6594
Epoch 12/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.4979 - loss: -3691.1912 - val_accuracy: 0.4946 - val_loss: -4512.4570
Epoch 13/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.5083 - loss: -4522.6372 - val_accuracy: 0.4946 - val_loss: -5617.9097
Epoch 14/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.5063 - loss: -5646.0630 - val_accuracy: 0.4946 - val_loss: -6432.0903
Epoch 15/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.4854 - loss: -7186.7969 - val_accuracy: 0.4946 - val_loss: -8654.0234
Epoch 16/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4984 - loss: -8366.2305 - val_accuracy: 0.4946 - val_loss: -10876.6182
Epoch 17/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5014 - loss: -9968.0508 - val_accuracy: 0.4946 - val_loss: -12914.7871
Epoch 18/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5039 - loss: -11481.2109 - val_accuracy: 0.4946 - val_loss: -10900.1230
Epoch 19/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4930 - loss: -13427.7412 - val_accuracy: 0.4946 - val_loss: -15397.5039
Epoch 20/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5039 - loss: -15574.1973 - val_accuracy: 0.4946 - val_loss: -18339.2461
Epoch 21/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5038 - loss: -17802.7910 - val_accuracy: 0.4946 - val_loss: -21498.6211
Epoch 22/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4978 - loss: -20384.3848 - val_accuracy: 0.4946 - val_loss: -23736.0391
Epoch 23/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.4980 - loss: -23109.3281 - val_accuracy: 0.4946 - val_loss: -25029.0840
Epoch 24/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.4941 - loss: -26279.7051 - val_accuracy: 0.4946 - val_loss: -28181.3301
Epoch 25/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.5089 - loss: -28526.5703 - val_accuracy: 0.4946 - val_loss: -34184.6445
Epoch 26/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5147 - loss: -31441.5117 - val_accuracy: 0.4946 - val_loss: -39420.6953
Epoch 27/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5143 - loss: -35474.4297 - val_accuracy: 0.4946 - val_loss: -42576.3594
Epoch 28/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5001 - loss: -39088.5859 - val_accuracy: 0.4946 - val_loss: -43430.5117
Epoch 29/50
```

```
from tensorflow.keras.models import load_model

# Load the model
loaded_model = load_model('/content/drive/My Drive/Honors Deep Learning/rnn_model_ind.h5')

# Verify the loaded model's architecture
```

```
loaded_model.summary()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
conv1d_8 (Conv1D)	(None, 8, 32)	128
max_pooling1d_8 (MaxPooling1D)	(None, 4, 32)	0
dropout_89 (Dropout)	(None, 4, 32)	0
simple_rnn (SimpleRNN)	(None, 4, 100)	13,300
simple_rnn_1 (SimpleRNN)	(None, 4, 75)	13,200
simple_rnn_2 (SimpleRNN)	(None, 50)	6,300
dense_76 (Dense)	(None, 3)	153

Total params: 33,083 (129.23 KB)
Trainable params: 33,081 (129.22 KB)

```
from tensorflow.keras.models import load_model

# Load the model
rnn_ind_model = load_model('/content/drive/My Drive/Honors Deep Learning/rnn_model_ind.h5')

# Verify the loaded model's architecture
rnn_ind_model.summary()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 8, 32)	128
max_pooling1d (MaxPooling1D)	(None, 4, 32)	0
dropout (Dropout)	(None, 4, 32)	0
simple_rnn (SimpleRNN)	(None, 4, 100)	13,300
simple_rnn_1 (SimpleRNN)	(None, 4, 75)	13,200
simple_rnn_2 (SimpleRNN)	(None, 50)	6,300
dense (Dense)	(None, 2)	102

Total params: 33,032 (129.04 KB)
Trainable params: 33,030 (129.02 KB)

```
# Prepare a subset of the data for explanation (e.g., first 100 samples)
X_explain = X_train[:100]
print(X_explain)
```

```
          Age    Gender  Total_Bilirubin  Direct_Bilirubin  \
477  0.697983  1.000000        -0.385832         -0.396968
346  0.136345  1.000000        -0.385832         -0.396968
462 -0.362888  1.000000        -0.366998         -0.396968
671  1.188481  0.427996        -0.375059         -0.414698
302  0.385962  0.000000        -0.178657         -0.231271
..        ...       ...              ...               ...
361  1.322025  0.000000        -0.366998         -0.396968
56  -0.674909  1.000000        -0.348164         -0.396968
434 -0.862121  0.000000        -0.366998         -0.396968
294 -1.361355  1.000000        -0.046819         -0.065574
809 -0.550100  0.344277        -0.335814         -0.396968

     Alkaline_Phosphotase  Alamine_Aminotransferase  \
477             -0.205877                 -0.270066
346             -0.697523                 -0.334696
462             -0.720935                 -0.282992
671             -0.375790                 -0.284828
302              0.777414                 -0.244215
..                    ...                       ...
361             -0.383806                 -0.276529
56              -0.308889                 -0.257141
434             -0.327618                 -0.218363
294              0.356003                 -0.263604
```

```
809            -0.333451               -0.175560

     Aspartate_Aminotransferase  Total_Protiens   Albumin  \
477                   -0.228511       -1.360250 -1.055899
346                   -0.314438        0.401435  0.266730
462                   -0.285796       -0.087922 -0.923636
671                   -0.257137        0.052232 -0.168153
302                   -0.228511        0.303564  0.398993
..                          ...             ...       ...
361                   -0.244878        0.303564 -0.394585
56                    -0.265337        1.478020  1.060307
434                   -0.212144        0.988663  0.531256
294                   -0.273521        1.771634  1.721622
809                   -0.220462        0.691836  1.667402

     Albumin_and_Globulin_Ratio
477                   -0.310496
346                   -0.242492
462                   -1.126550
671                   -0.340424
302                    0.097531
..                          ...
361                   -0.922537
56                     0.097531
434                   -0.174487
294                    0.437554
809                    2.232402

[100 rows x 10 columns]
```

```
# prompt: give feature names of the above in list

feature_names = list(X.columns)
print(feature_names)
```

```
['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase', 'Alamine_Aminotransferase', 'Aspartate_Aminotransfe
```

```
shap.initjs()
```

```
# Initialize the SHAP explainer
explainer = shap.KernelExplainer(rnn_ind_model.predict, X_explain)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 199ms/step
```

```
print(rnn_ind_model.predict(X_explain))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
[[9.10695851e-01 8.93041566e-02]
 [6.89601541e-01 3.10398430e-01]
 [3.07641894e-01 6.92358136e-01]
 [4.97642197e-02 9.50235665e-01]
 [9.55212057e-01 4.47879098e-02]
 [2.34993082e-02 9.76500750e-01]
 [9.98077035e-01 1.92300265e-03]
 [6.75105393e-01 3.24894637e-01]
 [2.96358541e-02 9.70364213e-01]
 [9.58525121e-01 4.14747596e-02]
 [9.91846919e-02 9.00815248e-01]
 [9.96225893e-01 3.77410254e-03]
 [9.93148386e-01 6.85164193e-03]
 [2.90561840e-02 9.70943809e-01]
 [2.38333102e-02 9.76166546e-01]
 [9.19923902e-01 8.00761059e-02]
 [2.67758053e-02 9.73224223e-01]
 [9.57462728e-01 4.25372012e-02]
 [9.99942124e-01 5.78231047e-05]
 [9.94562328e-01 5.43751242e-03]
 [9.93988693e-01 6.01137523e-03]
 [3.89437191e-02 9.61056292e-01]
 [3.07913274e-02 9.69208658e-01]
 [5.71760178e-01 4.28239822e-01]
 [4.05669538e-03 9.95943248e-01]
 [2.18463004e-01 7.81536996e-01]
 [9.36131835e-01 6.38680607e-02]
 [9.98572826e-01 1.42711715e-03]
 [6.06084943e-01 3.93914968e-01]
 [9.04412806e-01 9.55871791e-02]
 [5.09881973e-01 4.90118027e-01]
 [9.44731951e-01 5.52680120e-02]
 [5.21010645e-02 9.47898924e-01]
 [4.36321795e-01 5.63678026e-01]
 [4.34749462e-02 9.56525028e-01]
```

```
[6.32348135e-02 9.36765194e-01]
[5.41981161e-01 4.58018899e-01]
[3.27101462e-02 9.67289805e-01]
[9.82744694e-01 1.72552578e-02]
[9.95574057e-01 4.42585768e-03]
[1.04873493e-01 8.95126402e-01]
[9.98802185e-01 1.19792321e-03]
[4.89007324e-01 5.10992587e-01]
[9.72927153e-01 2.70728059e-02]
[3.84033591e-01 6.15966439e-01]
[9.02216375e-01 9.77835208e-02]
[8.39439780e-03 9.91605639e-01]
[9.95400846e-01 4.59918752e-03]
[2.51976222e-01 7.48023748e-01]
[4.07198548e-01 5.92801452e-01]
[9.35146287e-02 9.06485319e-01]
[1.32271394e-01 8.67728710e-01]
[2.75583863e-01 7.24416196e-01]
[9.89911914e-01 1.00880340e-02]
[3.18945348e-02 9.68105495e-01]
[1.74585097e-02 9.82541502e-01]
[3.13672610e-02 9.68632877e-01]
```

```
# Calculate SHAP values
shap_values = explainer.shap_values(X_explain)
```

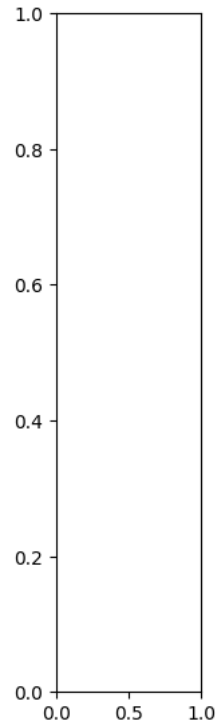
```
100%                                                  100/100 [28:26<00:00, 17.25s/it]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 13s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 13s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
```

```
shap_values.shape
```

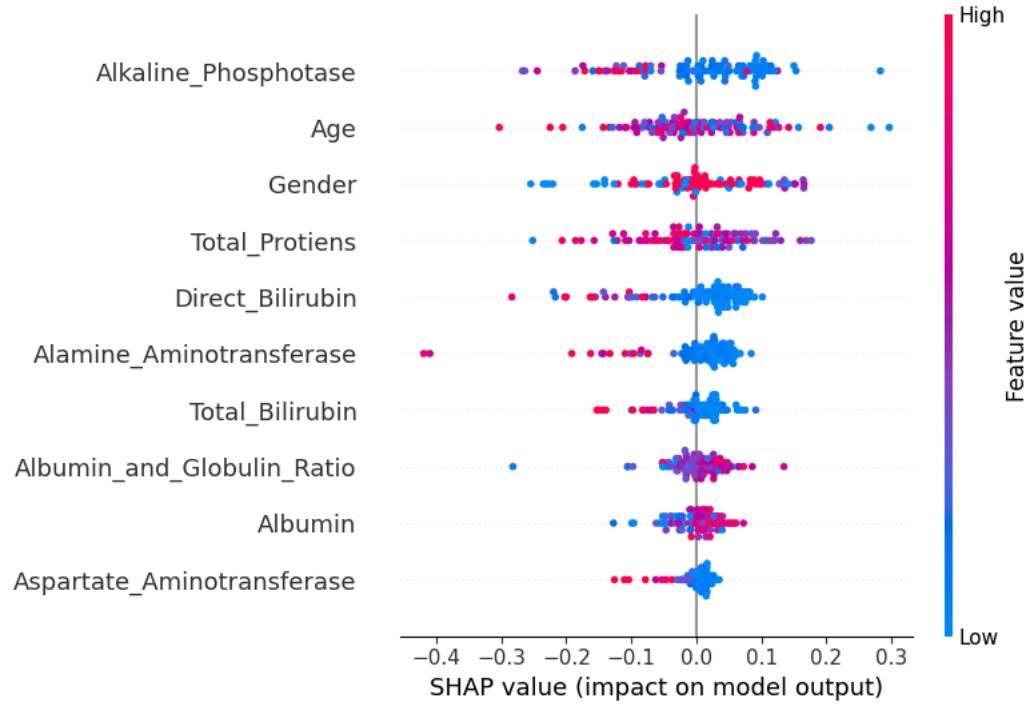
```
(100, 10, 2)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

```
# Plot the SHAP values
shap.summary_plot(shap_values, X_explain, feature_names=feature_names)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-153-86abd3b7893d> in <cell line: 2>()
      1 # Plot the SHAP values
----> 2 shap.summary_plot(shap_values, X_explain, feature_names=feature_names)

/usr/local/lib/python3.10/dist-packages/shap/plots/_beeswarm.py in summary_legacy(shap_values, features, feature_names,
max_display, plot_type, color, axis_color, title, alpha, show, sort, color_bar, plot_size, layered_violin_max_num_bins,
class_names, class_inds, color_bar_label, cmap, show_values_in_legend, use_log_scale)
    593         summary_legacy(
    594             proj_shap_values, features[:, sort_inds] if features is not None else None,
--> 595             feature_names=feature_names[sort_inds],
    596             sort=False, show=False, color_bar=False,
    597             plot_size=None,

TypeError: only integer scalar arrays can be converted to a scalar index
```

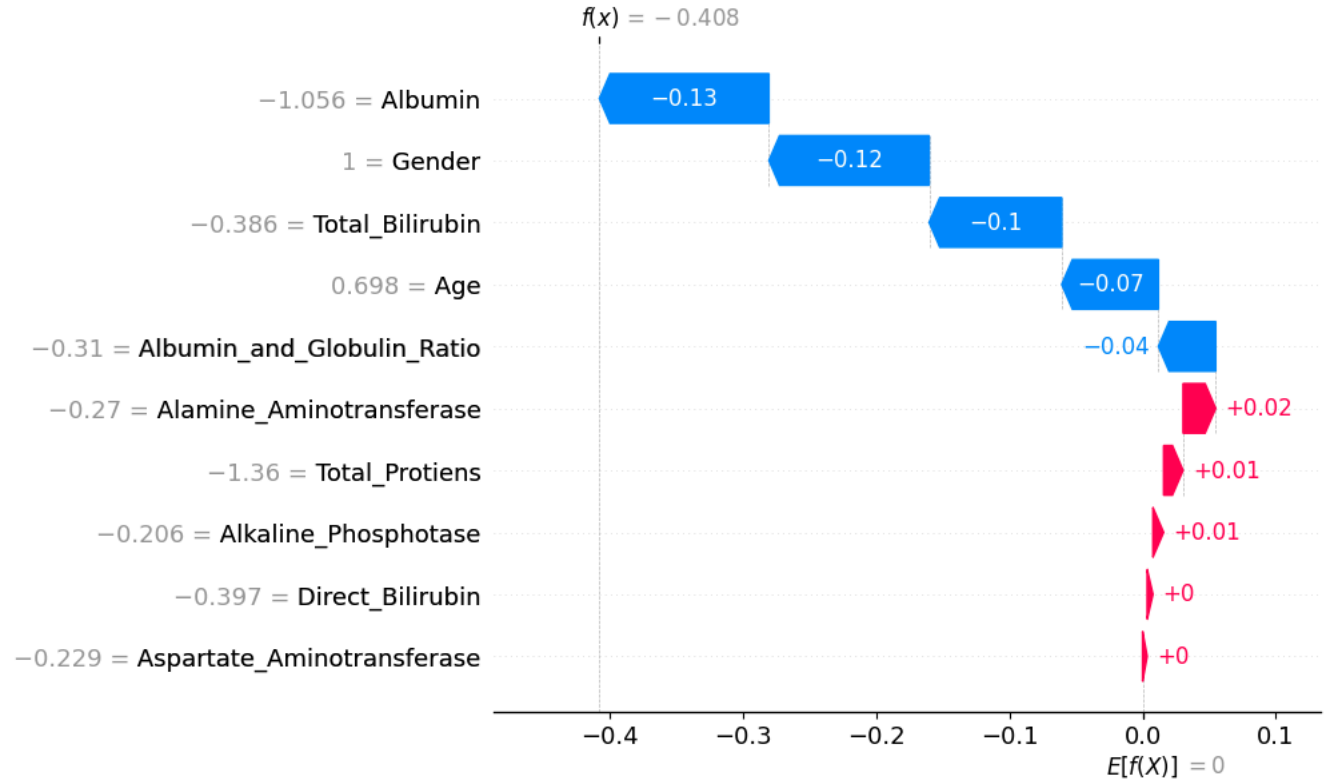



```
# Select SHAP values for the first output (class 0)
shap_values_class_0 = shap_values[:, :, 1]

# Plot the summary for the first output
shap.summary_plot(shap_values_class_0, X_explain, feature_names=feature_names)
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5

plt.figure(figsize=(40, 10))
# Loop through each instance
for i in range(num_instances):
    # Create an Explanation object
    expl = shap.Explanation(values=shap_values[i,:,1],
                            data=X_explain.iloc[i],
                            feature_names=feature_names)

    # Set base_values attribute
    expl.base_values = np.mean(shap_values)
    # Plot the waterfall plot
    shap.plots.waterfall(expl)

plt.tight_layout()
plt.show()
```



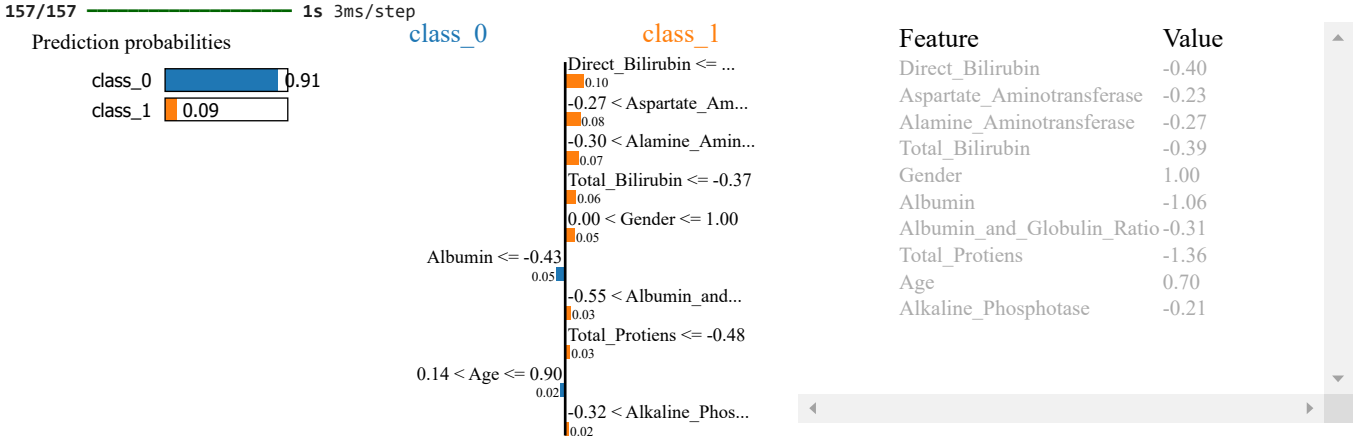

```
# Importing the module for LimeTabularExplainer
from lime import lime_tabular

# Instantiating the explainer object by passing in the training set,
# and the extracted features
# Convert the NumPy array X_train to a DataFrame
X_train_df = pd.DataFrame(X_explain, columns=feature_names)

X_train_df.reset_index(drop=True, inplace=True)

# Make sure that the feature names (col) match the columns of your DataFrame
col = X_train_df.columns.tolist()

# Instantiate the explainer object by passing in the DataFrame
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_df.values,
                                                   feature_names=feature_names,
                                                   class_names=['class_0', 'class_1'],
                                                   mode='classification')

print(X_explain.iloc[0])
```

```
Age                          0.697983
Gender                       1.000000
Total_Bilirubin             -0.385832
Direct_Bilirubin            -0.396968
Alkaline_Phosphotase        -0.205877
Alamine_Aminotransferase    -0.270066
Aspartate_Aminotransferase  -0.228511
Total_Protiens              -1.360250
```

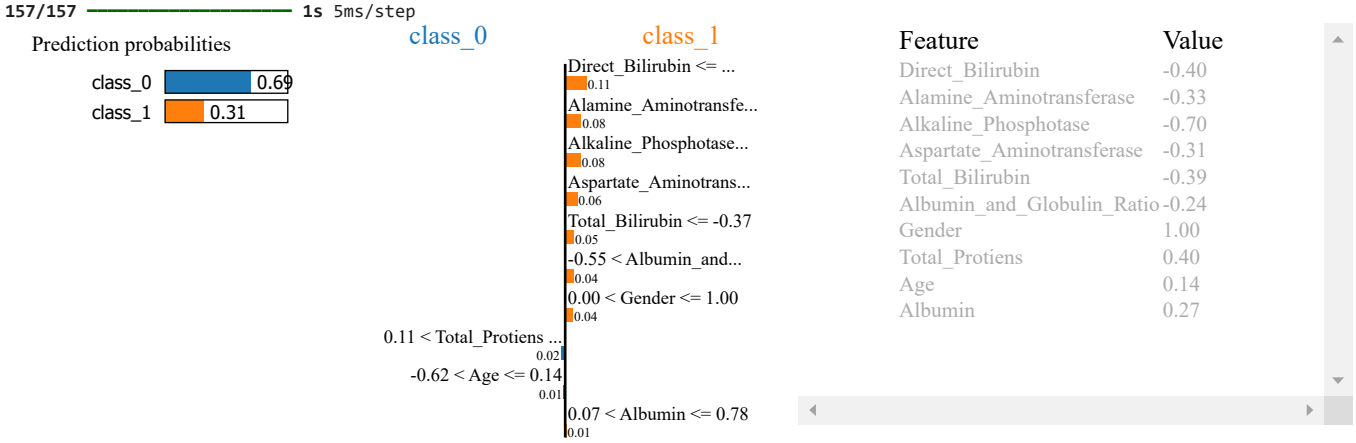
```
Albumin                      -1.055899
Albumin_and_Globulin_Ratio   -0.310496
Name: 477, dtype: float64
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5


# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[i]

    # Explain the prediction for the instance
    explanation = explainer.explain_instance(instance, rnn_ind_model.predict)

    # Plot the explanation in the corresponding subplot
    explanation.show_in_notebook()
```

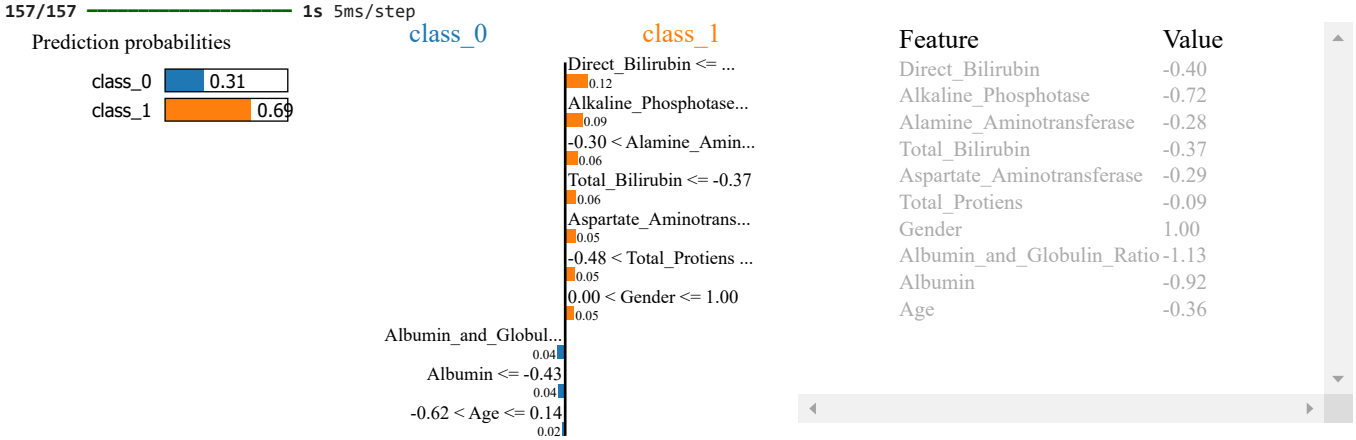
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step



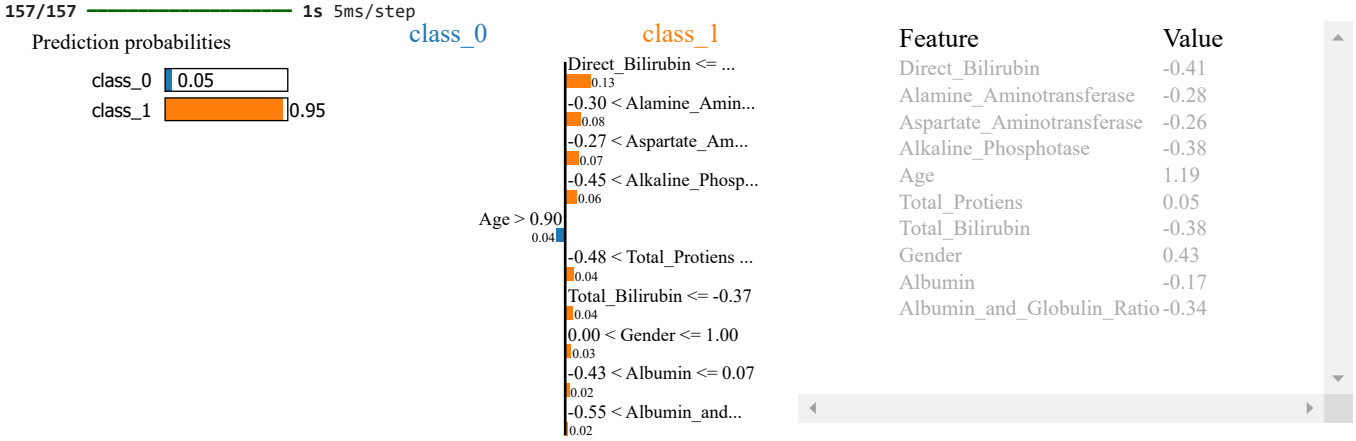
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step



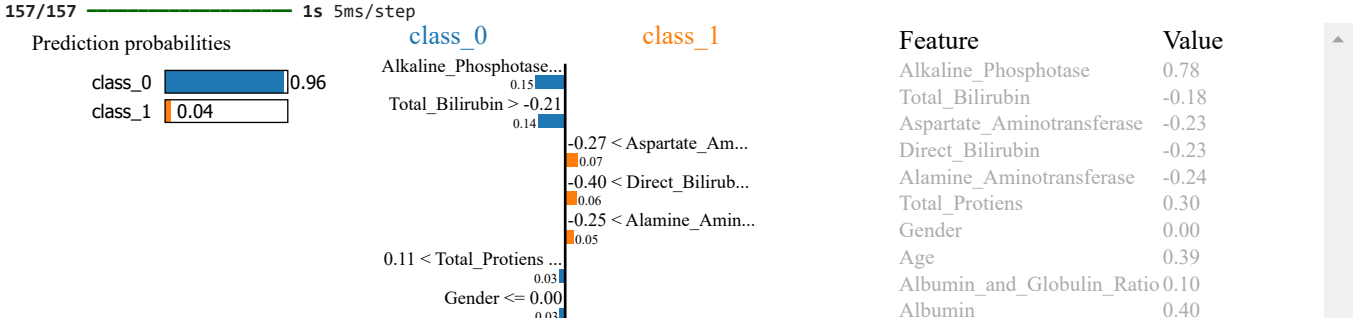
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step

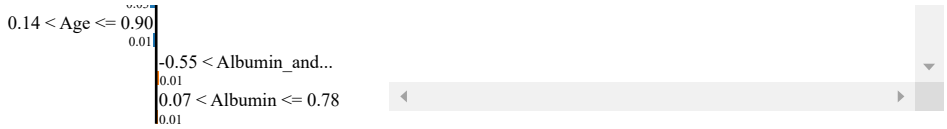


157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step



157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step

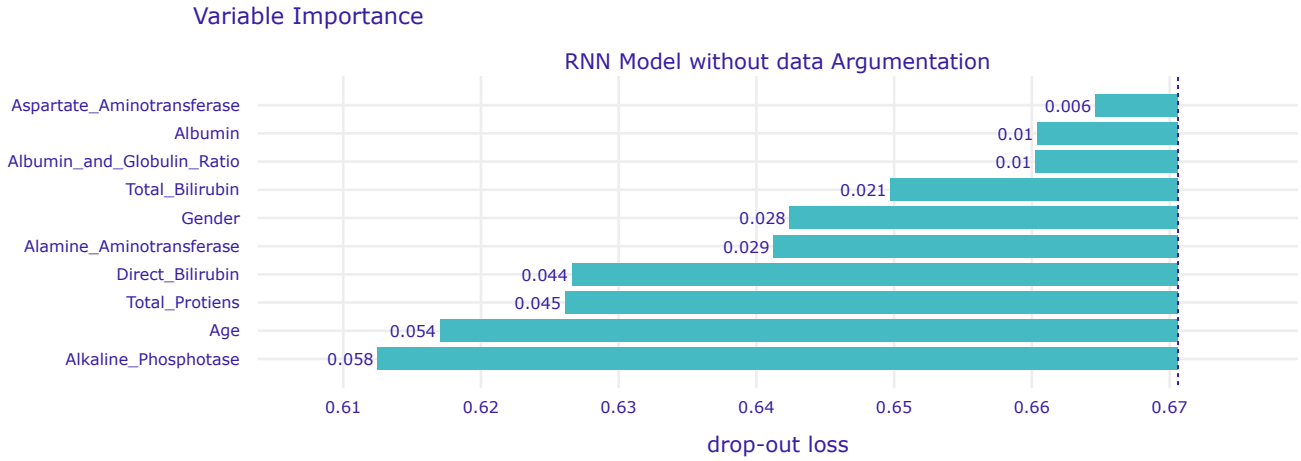




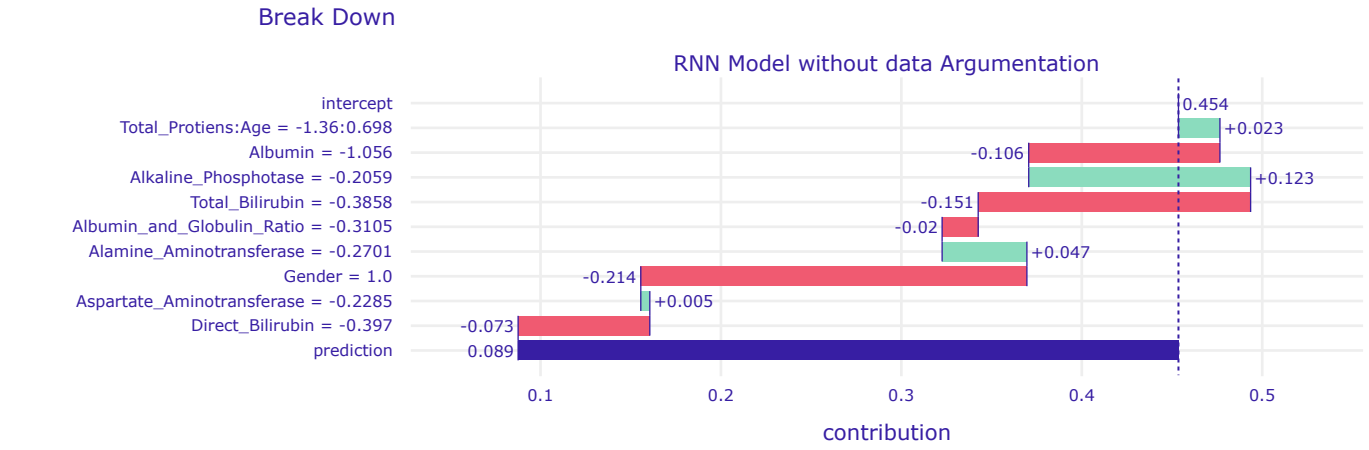

```
explainer = dx.Explainer(rnn_ind_model, X_train, y_train, label="RNN Model without data Argumentation")
```

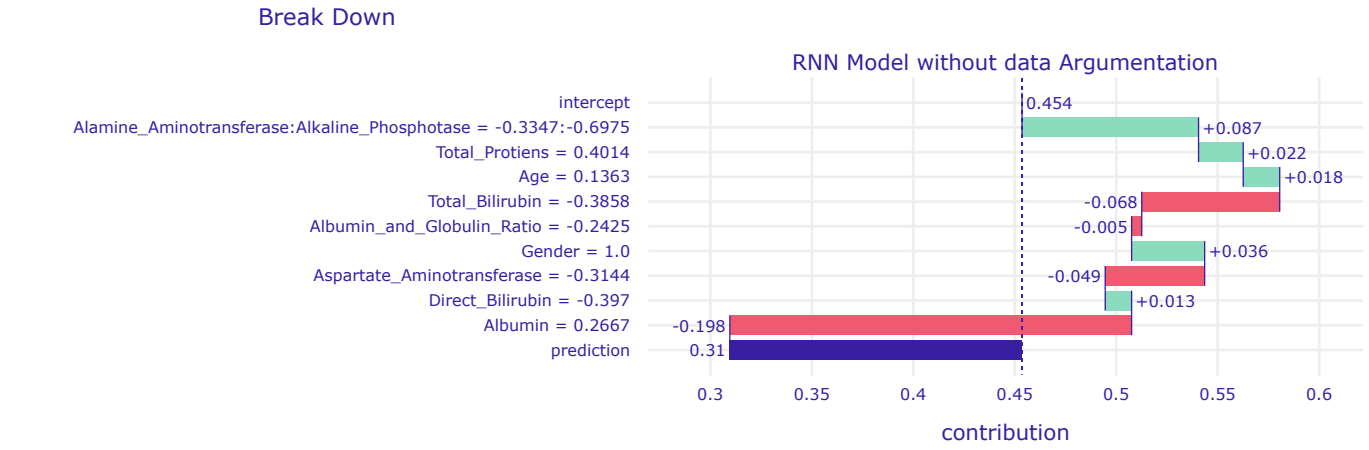
```
Preparation of a new explainer is initiated

  -> data              : 665 rows 10 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 665 values
  -> model_class       : keras.src.models.sequential.Sequential (default)
  -> label             : RNN Model without data Argumentation
  -> predict function  : <function yhat_tf_classification at 0x7a0610c9c430> will be used (default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 1.08e-05, mean = 0.454, max = 1.0
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = 0.242, mean = 1.03, max = 1.82
  -> model_info        : package keras

A new explainer has been created!
```

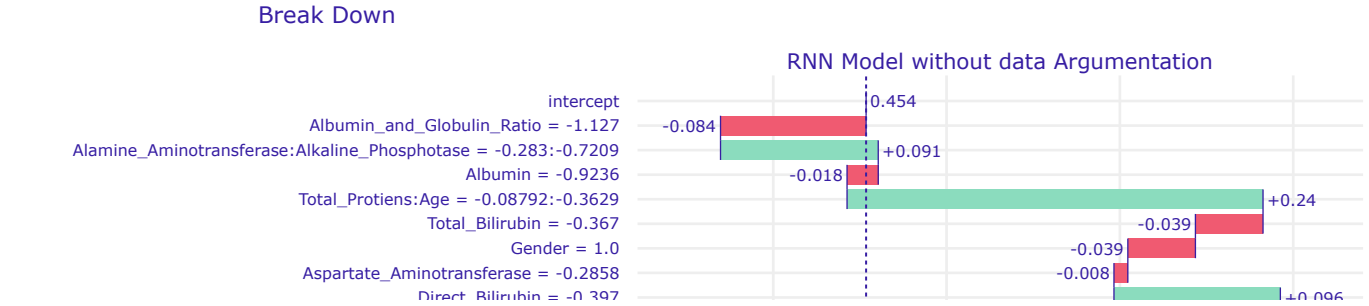
```
fi = explainer.model_parts().plot()
```




```
num_instances = 5


# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[i, :]

    # Explain the prediction for the instance
    pp = explainer.predict_parts(instance).plot()
```








```
from tensorflow.keras.models import load_model

# Load the model
lstm_ind_model = load_model('/content/drive/My Drive/Honors Deep Learning/lstm_model2_ind.h5')

# Verify the loaded model's architecture
lstm_ind_model.summary()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv1d_3 (Conv1D)	(None, 8, 32)	128
max_pooling1d_3 (MaxPooling1D)	(None, 4, 32)	0
dropout_3 (Dropout)	(None, 4, 32)	0
lstm_3 (LSTM)	(None, 4, 100)	53,200
lstm_4 (LSTM)	(None, 4, 75)	52,800
lstm_5 (LSTM)	(None, 50)	25,200
dense_3 (Dense)	(None, 2)	102

```
Total params: 131,432 (513.41 KB)
Trainable params: 131,430 (513.40 KB)
```

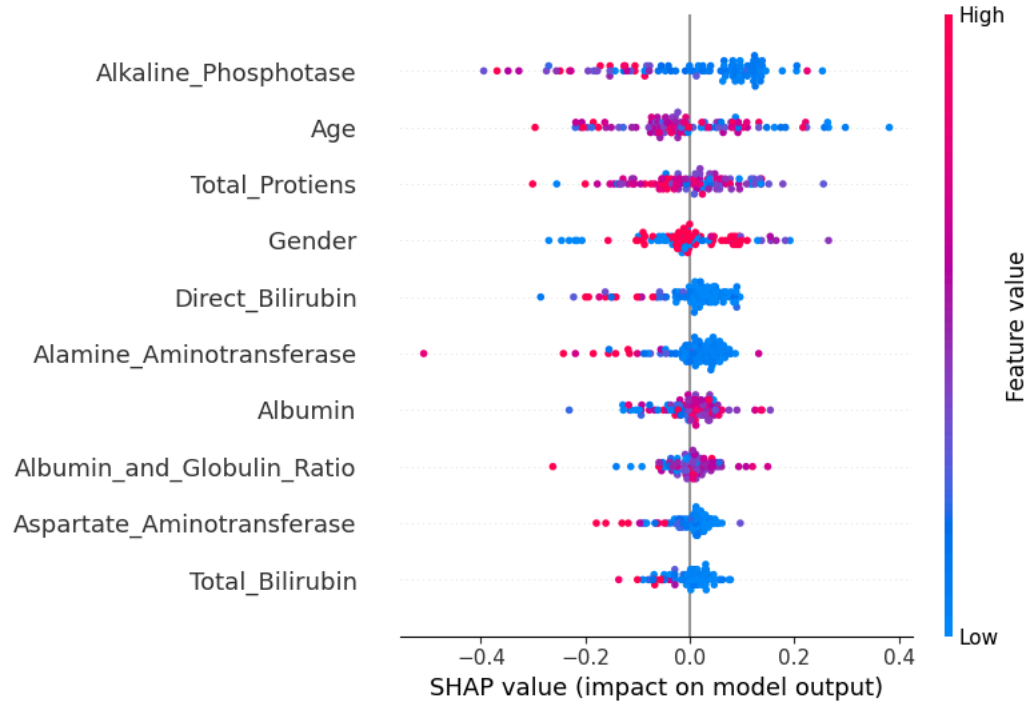
```
# Initialize the SHAP explainer
explainer = shap.KernelExplainer(lstm_ind_model.predict, X_explain)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 552ms/step
```

```
# Calculate SHAP values
shap_values_ind = explainer.shap_values(X_explain)
```

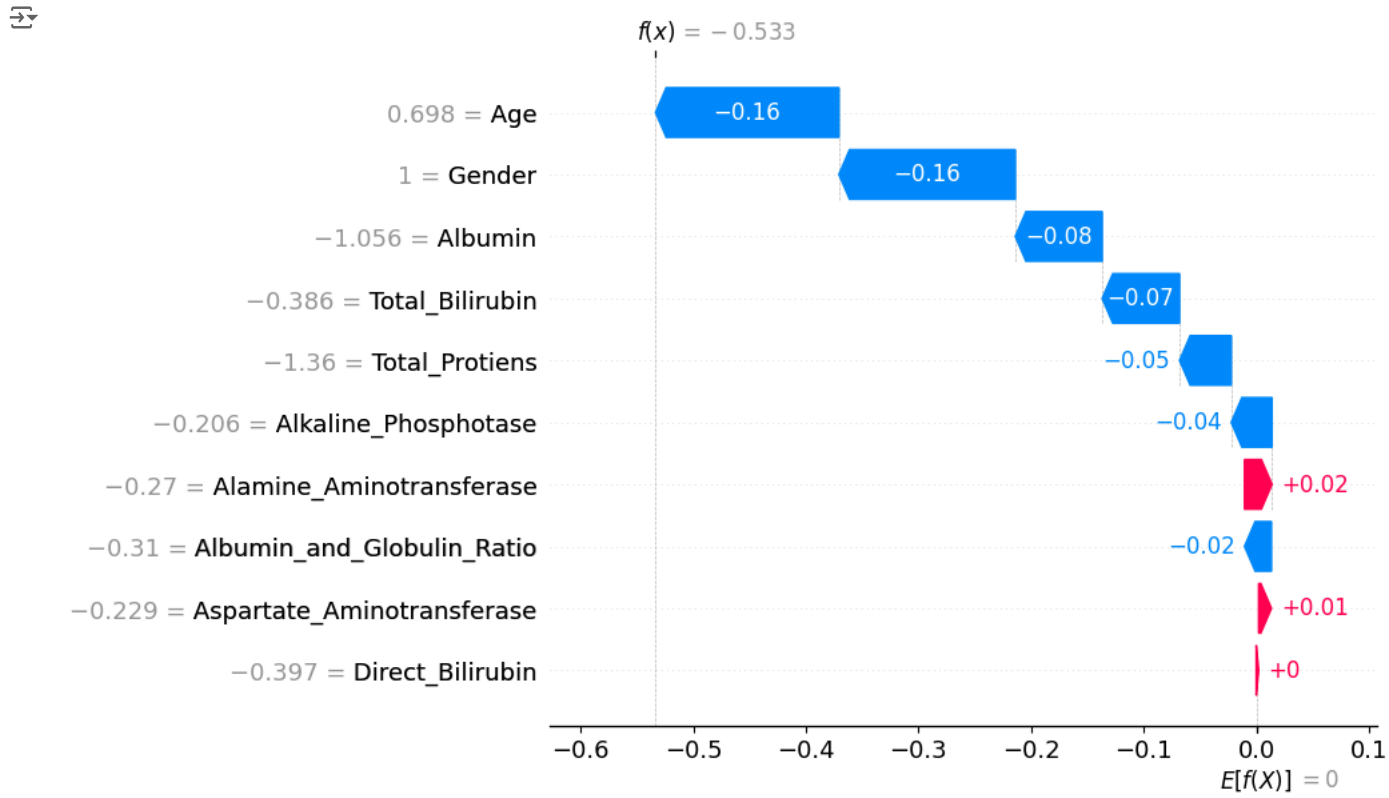
```
100%                                                  100/100 [38:10<00:00, 27.37s/it]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 31s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 32s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 31s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 31s 10ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 29s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 29s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 19s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 20s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 5ms/step
```

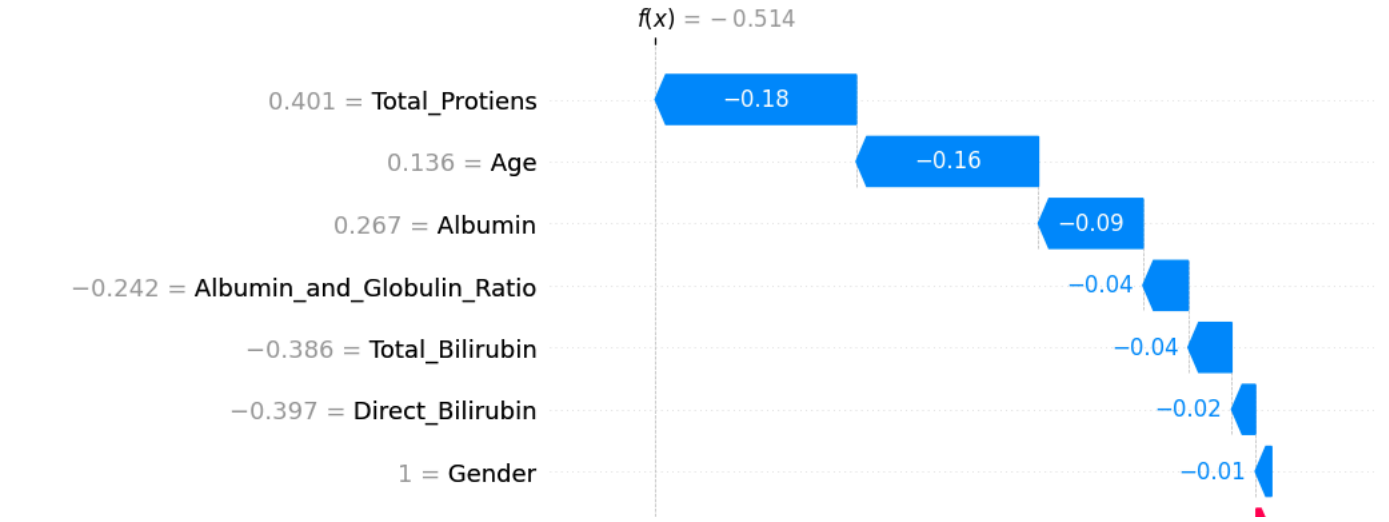
```
# Select SHAP values for the first output (class 0)
shap_values_ind_class_1 = shap_values_ind[:, :, 1]

# Plot the summary for the first output
shap.summary_plot(shap_values_ind_class_1, X_explain, feature_names=feature_names)
```




```
3194/3194 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5

plt.figure(figsize=(40, 10))
# Loop through each instance
for i in range(num_instances):
    # Create an Explanation object
    expl = shap.Explanation(values=shap_values_ind[i,:,1],
                            data=X_explain.iloc[i],
                            feature_names=feature_names)

    # Set base_values attribute
    expl.base_values = np.mean(shap_values_ind)
    # Plot the waterfall plot
    shap.plots.waterfall(expl)

plt.tight_layout()
plt.show()
```





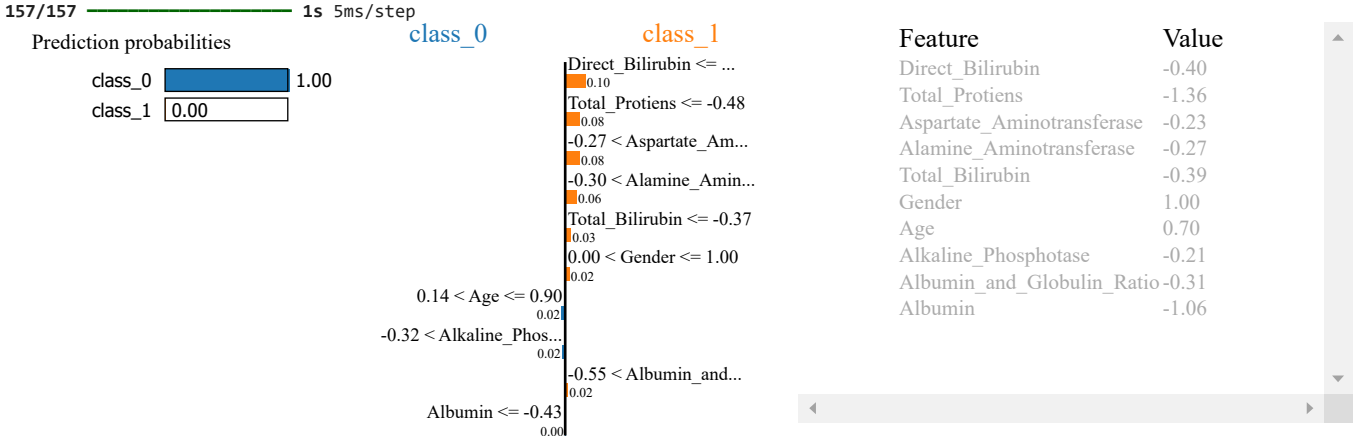

```
# Importing the module for LimeTabularExplainer
from lime import lime_tabular

# Instantiating the explainer object by passing in the training set,
# and the extracted features
# Convert the NumPy array X_train to a DataFrame
X_train_df = pd.DataFrame(X_explain, columns=feature_names)

X_train_df.reset_index(drop=True, inplace=True)

# Make sure that the feature names (col) match the columns of your DataFrame
col = X_train_df.columns.tolist()

# Instantiate the explainer object by passing in the DataFrame
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_df.values,
                                                   feature_names=feature_names,
                                                   class_names=['class_0', 'class_1'],
                                                   mode='classification')

print(X_explain.iloc[0])
```

```
Age                          0.697983
Gender                       1.000000
Total_Bilirubin             -0.385832
Direct_Bilirubin            -0.396968
Alkaline_Phosphotase        -0.205877
Alamine_Aminotransferase    -0.270066
Aspartate_Aminotransferase  -0.228511
Total_Protiens              -1.360250
Albumin                     -1.055899
```

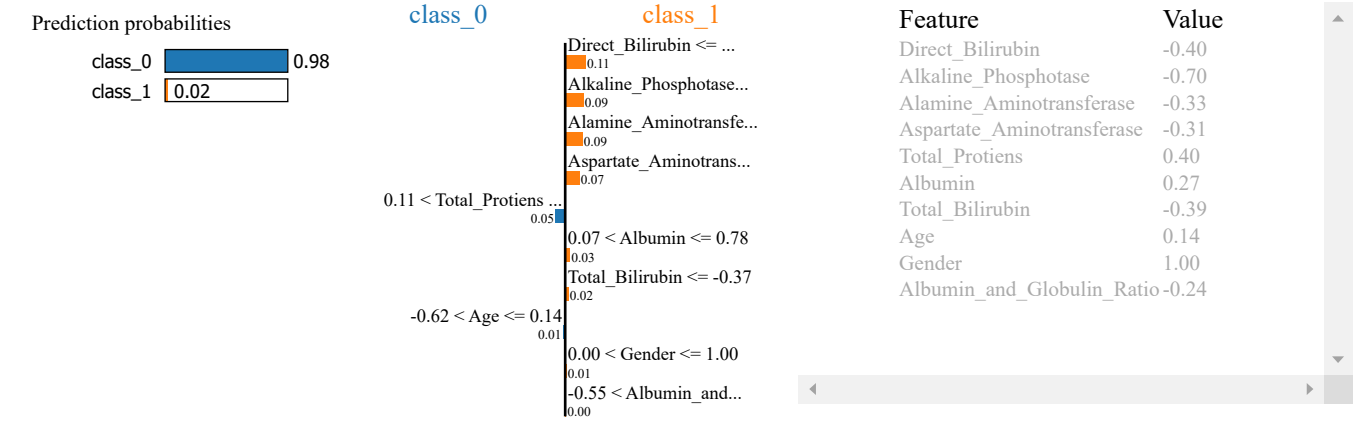
```
Albumin_and_Globulin_Ratio   -0.310496
Name: 477, dtype: float64
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5


# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[i]

    # Explain the prediction for the instance
    explanation = explainer.explain_instance(instance, lstm_ind_model.predict)

    # Plot the explanation in the corresponding subplot
    explanation.show_in_notebook()
```

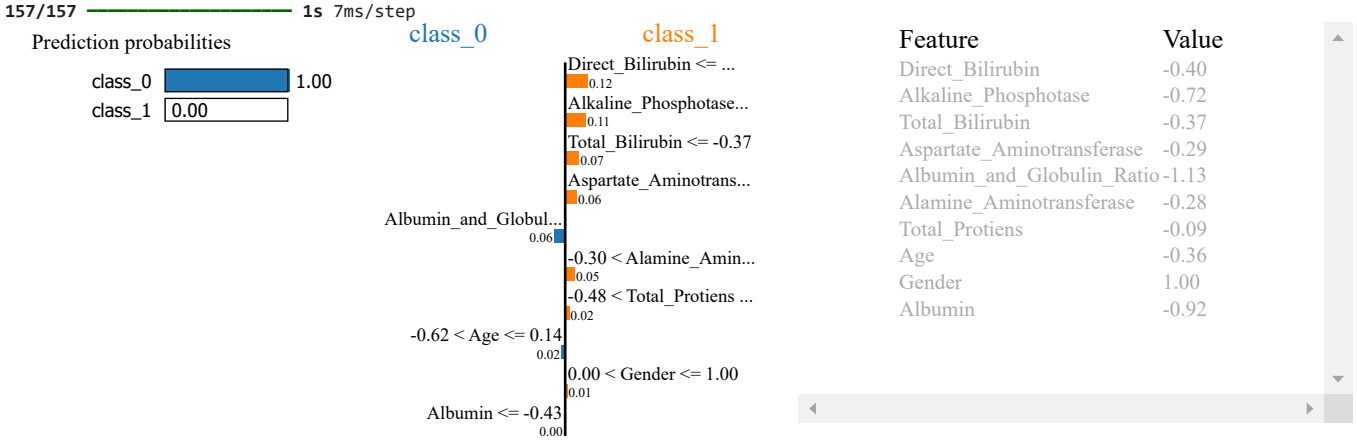
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step



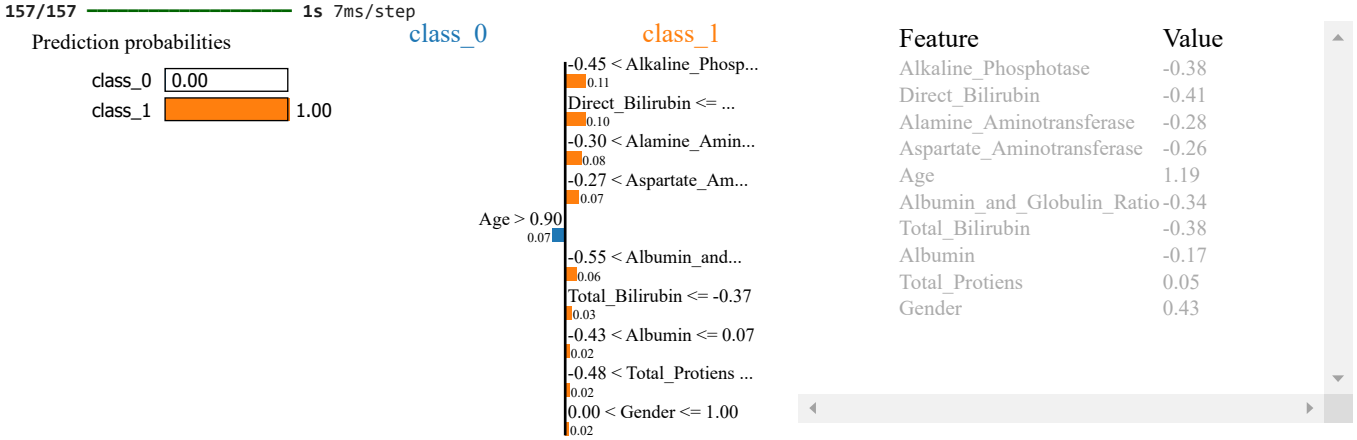
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step



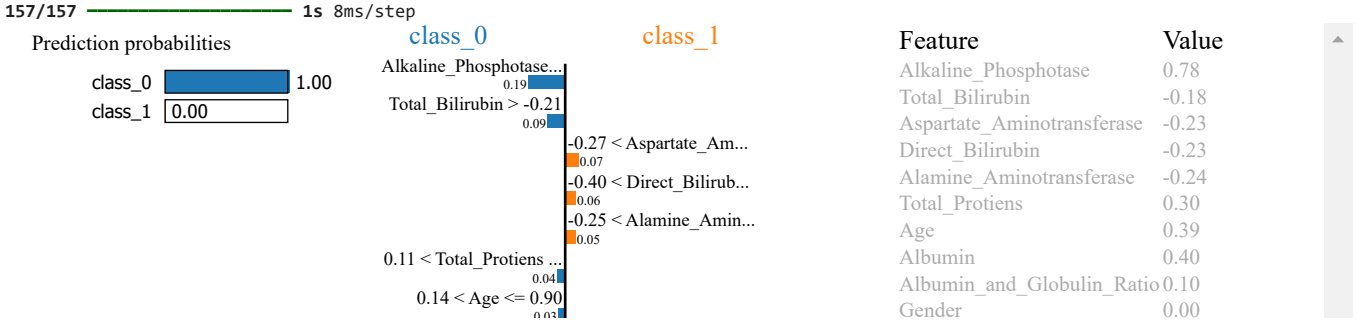
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step

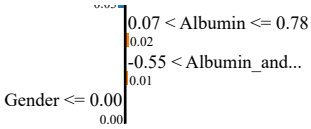


157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step



157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step

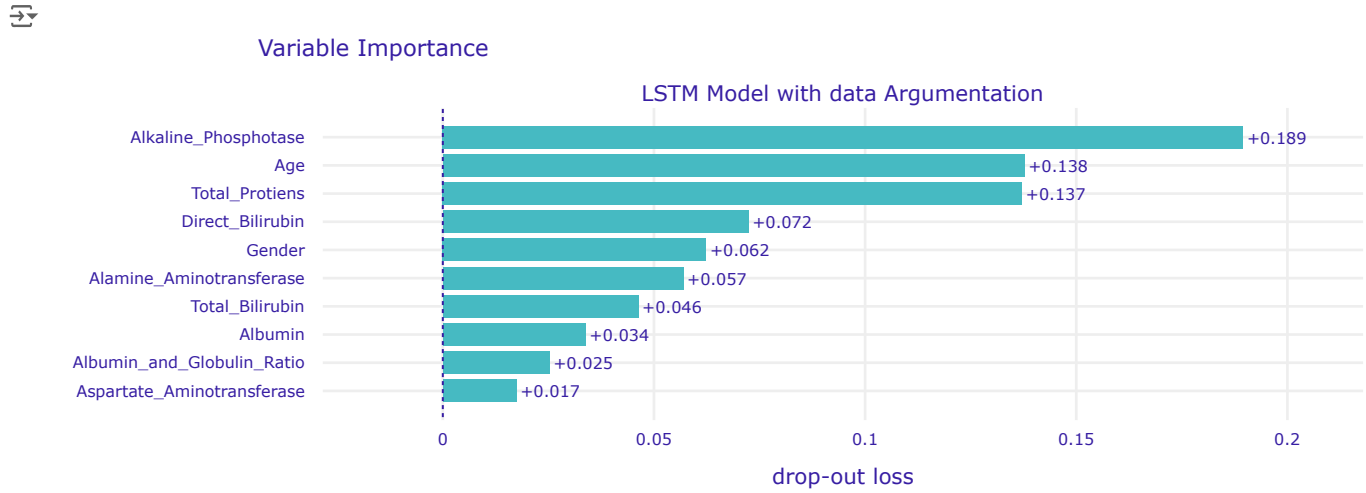




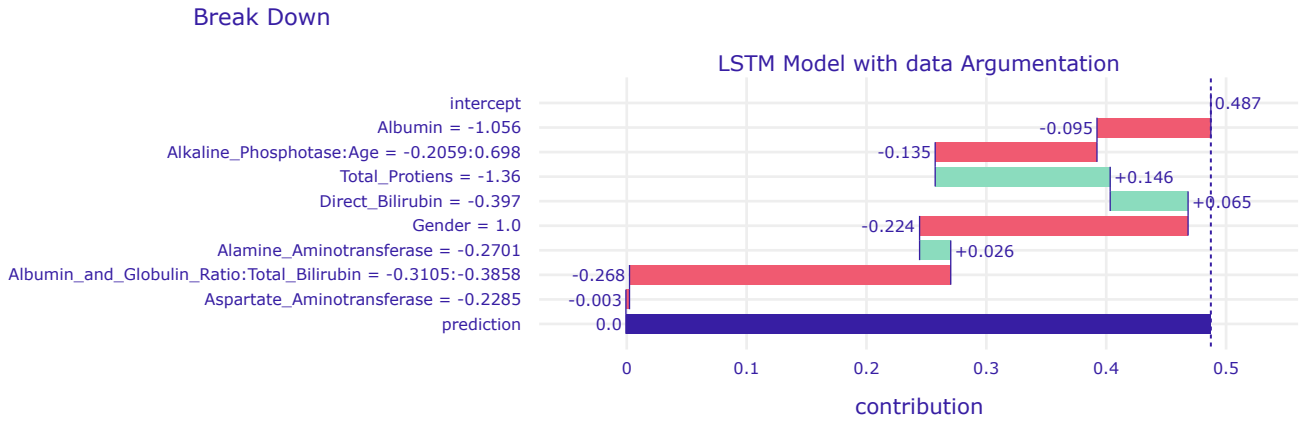

```
explainer = dx.Explainer(lstm_ind_model, X_train, y_train, label="LSTM Model with data Argumentation")
```

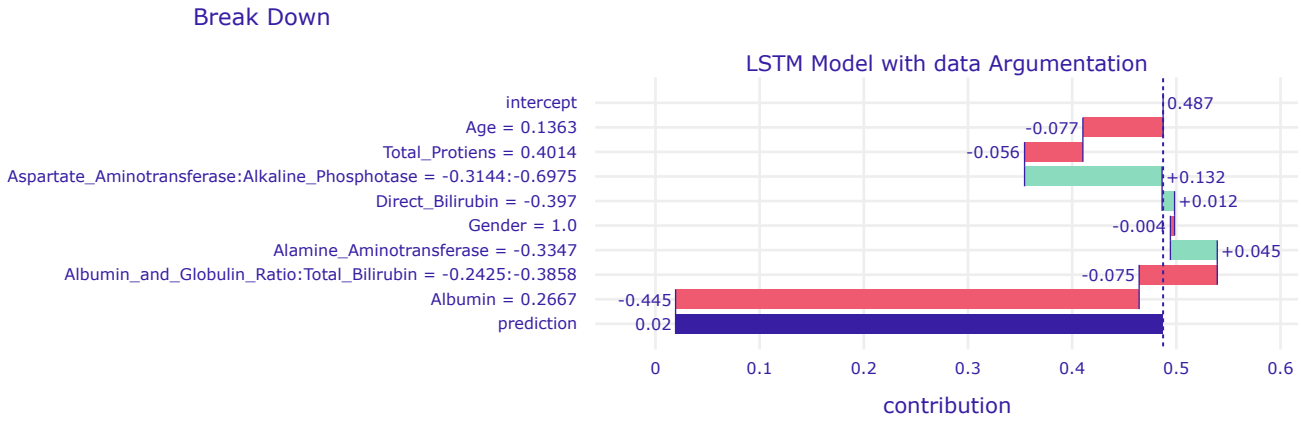
```
Preparation of a new explainer is initiated

  -> data              : 665 rows 10 cols
  -> target variable   : 665 values
  -> model_class       : keras.src.models.sequential.Sequential (default)
  -> label             : LSTM Model with data Argumentation
  -> predict function  : <function yhat_tf_classification at 0x78cd67e29c60> will be used (default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 4.42e-09, mean = 0.487, max = 1.0
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.284, mean = -7.71e-05, max = 0.482
  -> model_info        : package keras

A new explainer has been created!
```

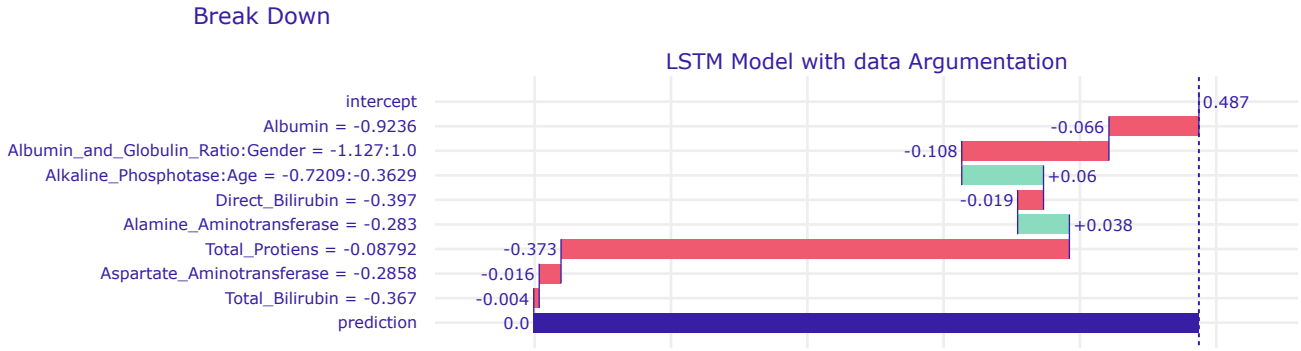
```
fi = explainer.model_parts().plot()
```




```
num_instances = 5


# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[i, :]

    # Explain the prediction for the instance
    pp = explainer.predict_parts(instance).plot()
```







Cirrhosis

```
df_data = pd.read_csv("/cirrhosis.csv")
```

```
def diagnose_dataframe(df, verbose=True):
    """
    Performs various initial inspections on a pandas dataframe and provides details.

    Args:
      df: The pandas dataframe to diagnose.
      verbose: Whether to print details for each inspection.

    Returns:
      A dictionary containing the results of the inspections.
    """

    results = {}

    # Basic information
    results["shape"] = df.shape
    results["columns"] = list(df.columns)
    results["dtypes"] = df.dtypes.to_dict()

    # Check for disparities between dtypes and actual values
    dtypes_disparities = {}
    for col in df.columns:
```

```
            unique_values = df[col].unique()
            dtype = df[col].dtype
            unique_types = set(type(value).__name__ for value in unique_values)
            if len(unique_types) > 1:
                dtypes_disparities[col] = {"dtype": dtype, "unique_types": list(unique_types)}

        results["dtypes_disparities"] = dtypes_disparities

        # Unique values and counts
        for col in df.columns:
            results[f"unique_values_{col}"] = df[col].unique()

        if verbose:
            for key, value in results.items():
                print(f"\n{key}:\n{value}")

        return results


    diagnose_dataframe(df_data)
```

```
shape:
(418, 20)

columns:
['ID', 'N_Days', 'Status', 'Drug', 'Age', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders', 'Edema', 'Bilirubin', 'Cholesterol', 'Albu

dtypes:
{'ID': dtype('int64'), 'N_Days': dtype('int64'), 'Status': dtype('O'), 'Drug': dtype('O'), 'Age': dtype('int64'), 'Sex': dtype('O

dtypes_disparities:
{'Drug': {'dtype': dtype('O'), 'unique_types': ['str', 'float']}, 'Ascites': {'dtype': dtype('O'), 'unique_types': ['str', 'float

unique_values_ID:
[  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342
 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378
 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396
 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414
 415 416 417 418]

unique_values_N_Days:
[ 400 4500 1012 1925 1504 2503 1832 2466 2400   51 3762  304 3577 1217
 3584 3672  769  131 4232 1356 3445  673  264 4079 4127 1444   77  549
 4509  321 3839 4523 3170 3933 2847 3611  223 3244 2297 4467 1350 4453
 4556 3428 4025 2256 2576 4427  708 2598 3853 2386 1000 1434 1360 1847
 3282 4459 2224 4365 4256 3090  859 1487 3992 4191 2769 4039 1170 3458
 4196 4184 4190 1827 1191   71  326 1690 3707  890 2540 3574 4050 4032
 3358 1657  198 2452 1741 2689  460  388 3913  750  130 3850  611 3823
 3820  552 3581 3099  110 3086 3092 3222 3388 2583 2504 2105 2350  980
 3395 3422 3336 1083 2288  515 2033  191 3297  971 3069 2468  824 3255
 1037 3239 1413  850 2944 2796 3149 3150 3098 2990 1297 2106 3059 3050
 2419  786  943 2976 2615 2995 1427  762 2891 2870 1152 2863  140 2666
  853 2835 2475 1536 2772 2797  186 2055 1077 2721 1682 2713 1212 2692
 2574 2301 2657 2644 2624 1492 2609 2580 2573 2563 2556 2555 2241  974
 2527 1576  733 2332 2456  216 2443  797 2449 2330 2363 2365 2357 1592
 2318 2294 2272 2221 2090 2081 2255 2171  904 2216 2195 2176 2178 1786
 1080 2168  790 2170 2157 1235 2050  597  334 1945 2022 1978  999 1967
  348 1979 1165 1951 1932 1776 1882 1908 1874  694 1831  837 1810  930
```

```
df_data.replace({'Status': {'C': 0, 'CL': 1, 'D': 2}}, inplace=True)

cat_features = df_data.select_dtypes('object').columns
num_features =  df_data.select_dtypes(exclude='object').columns
num_features = [feature for feature in num_features if feature != 'Status' and feature != 'Stage']
```

```
def do_log_transform(df, feature):
    df[feature] = np.log1p(df[feature])

encoder = LabelEncoder()

for feature in cat_features:
    df_data[feature] = encoder.fit_transform(df_data[feature])


for feature in num_features:
    do_log_transform(df_data, feature)

df_data.head()
```

	ID	N_Days	Status	Drug	Age	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	Copper
0	0.693147	5.993961	2	0	9.974179	0	1	1	1	2	2.740840	5.568345	1.280934	5.056246
1	1.098612	8.412055	0	0	9.933920	0	0	1	1	0	0.741937	5.713733	1.637053	4.007333
2	1.386294	6.920672	2	0	10.150152	1	0	0	0	1	0.875469	5.176150	1.499623	5.351858
3	1.609438	7.563201	2	0	9.903238	0	0	1	1	1	1.029619	5.501258	1.264127	4.174387
4	1.791759	7.316548	1	1	9.541010	0	0	1	1	0	1.481605	5.634790	1.510722	4.969813

```
def get_features(df, feature_scaling=False):
    # Coagulation
    df['Coagulation'] = df['N_Days'] * (df['Platelets'] / (df['Prothrombin'] + 1e-9))  # Adding small value to avoid division by zero

    # Lipids
    df['Lipids'] = df['N_Days'] * (df['Cholesterol'] / (df['Tryglicerides'] + 1e-9))  # Adding small value to avoid division by zero

    # Liver
    df['Liver'] = ((df['Alk_Phos'] / (df['Prothrombin'] + 1e-9)) * (df['Albumin'] / (df['Bilirubin'] + 1e-9)))  # Adding small value to

    # Blood Pressure
    df['BloodPressure'] = (df['Liver'] * df['Albumin'] * df['Prothrombin']) / (df['Platelets'] + 1e-9)  # Adding small value to avoid di

    # Fats
    df['Fats'] = df['Lipids'] / (df['Tryglicerides'] + 1e-9)  # Adding small value to avoid division by zero

    # Foods
    df['Foods'] = df['Liver'] / (df['Cholesterol'] + 1e-9)  # Adding small value to avoid division by zero

    # Age in Years
    df['Age_in_years'] = df['Age']/365.0

    # Drop Age
    df.drop(columns=['Age'],axis=1,inplace = True)

    # Combination Features
    df['combination1'] = ((df['Albumin'] / (df['Bilirubin'] + 1e-9)) * (df['Platelets'] / (df['Alk_Phos'] + 1e-9)) * df['SGOT'] * df['Pr
    df['combination2'] = ((df['Albumin'] / (df['Prothrombin'] + 1e-9)) * (df['SGOT'] / (df['Alk_Phos'] + 1e-9)) * df['Platelets'] * df[
    df['combination3'] = ((df['Cholesterol'] / (df['Copper'] + 1e-9)) * df['Tryglicerides'])  # Adding small value to avoid division by

    return df


get_features(df_data)
```

	ID	N_Days	Status	Drug	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	...	Coagulation	Lipids	Liver	Bl
0	0.693147	5.993961	2	0	0	1	1	1	2	2.740840	...	12.201271	6.476723	1.349316	
1	1.098612	8.412055	0	0	0	0	1	1	0	0.741937	...	18.542441	10.707981	8.019810	
2	1.386294	6.920672	2	0	1	0	0	0	1	0.875469	...	13.555288	8.899206	4.172599	
3	1.609438	7.563201	2	0	0	0	1	1	1	1.029619	...	16.265903	9.179527	4.415122	
4	1.791759	7.316548	1	1	0	0	1	1	0	1.481605	...	14.535320	9.609043	2.680436	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
413	6.028279	6.525030	2	2	0	2	2	2	0	0.788457	...	13.607858	NaN	NaN	
414	6.030685	7.006695	0	2	0	2	2	2	0	0.641854	...	14.561350	NaN	NaN	
415	6.033086	6.962243	0	2	0	2	2	2	0	0.955511	...	14.484925	NaN	NaN	
416	6.035481	6.539586	0	2	0	2	2	2	0	0.587787	...	15.044034	NaN	NaN	
417	6.037871	6.884487	0	2	0	2	2	2	0	0.530628	...	16.462024	NaN	NaN	

418 rows × 29 columns

```
feature_names=['N_Days', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders', 'Edema',
       'Bilirubin', 'Cholesterol', 'Albumin', 'Copper', 'Alk_Phos', 'SGOT',
       'Tryglicerides', 'Platelets', 'Prothrombin', 'Stage', 'Coagulation',
       'Lipids', 'Liver', 'BloodPressure', 'Fats', 'Foods', 'Age_in_years',
       'combination1', 'combination2', 'combination3']


X = df_data.drop(['Status','ID'], axis = 1)
y = df_data['Status']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

X

	N_Days	Drug	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	...	Coagulation	Lipids	Live
0	5.993961	0	0	1	1	1	2	2.740840	5.568345	1.280934	...	12.201271	6.476723	1.34931
1	8.412055	0	0	0	1	1	0	0.741937	5.713733	1.637053	...	18.542441	10.707981	8.01981
2	6.920672	0	1	0	0	0	1	0.875469	5.176150	1.499623	...	13.555288	8.899206	4.17259
3	7.563201	0	0	0	1	1	1	1.029619	5.501258	1.264127	...	16.265903	9.179527	4.41512
4	7.316548	1	0	0	1	1	0	1.481605	5.634790	1.510722	...	14.535320	9.609043	2.68043
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
413	6.525030	2	0	2	2	2	0	0.788457	NaN	1.376244	...	13.607858	NaN	Na
414	7.006695	2	0	2	2	2	0	0.641854	NaN	1.574846	...	14.561350	NaN	Na
415	6.962243	2	0	2	2	2	0	0.955511	NaN	1.486140	...	14.484925	NaN	Na
416	6.539586	2	0	2	2	2	0	0.587787	NaN	1.558145	...	15.044034	NaN	Na
417	6.884487	2	0	2	2	2	0	0.530628	NaN	1.456287	...	16.462024	NaN	Na

418 rows × 27 columns

```
imputer = SimpleImputer(strategy='mean') # You can choose other strategies like 'median' or 'most_frequent'

# Fit the imputer on your training data and transform both training and testing data
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

smote = SMOTE(random_state=42)
X, y= smote.fit_resample(X, y)

# Scaling numerical features
scaler = StandardScaler()
X[['N_Days', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders', 'Edema',
       'Bilirubin', 'Cholesterol', 'Albumin', 'Copper', 'Alk_Phos', 'SGOT',
       'Tryglicerides', 'Platelets', 'Prothrombin', 'Stage', 'Coagulation',
       'Lipids', 'Liver', 'BloodPressure', 'Fats', 'Foods', 'Age_in_years',
       'combination1', 'combination2', 'combination3']] = scaler.fit_transform(X[['N_Days', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'S
       'Bilirubin', 'Cholesterol', 'Albumin', 'Copper', 'Alk_Phos', 'SGOT',
       'Tryglicerides', 'Platelets', 'Prothrombin', 'Stage', 'Coagulation',
       'Lipids', 'Liver', 'BloodPressure', 'Fats', 'Foods', 'Age_in_years',
       'combination1', 'combination2', 'combination3']])
```

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

```
X_train
```

	N_Days	Drug	Sex	Ascites	Hepatomegaly	Spiders	Edema	Bilirubin	Cholesterol	Albumin	...	Coagulation
82	1.414135	-1.149217	-0.368619	-0.657635	0.086585	-0.920385	1.690646	-0.595435	-0.883909	0.154210	...	-0.978826
51	0.733210	-1.149217	3.093598	-0.657635	-1.287284	-0.920385	-0.410625	1.070558	1.827076	0.604407	...	1.016344
218	0.603372	0.166774	-0.368619	-0.657635	-1.287284	-0.920385	-0.410625	-0.411918	-1.338199	1.591144	...	0.237161
557	-0.927872	1.482765	-0.368619	1.720850	1.460454	1.567972	-0.410625	1.034976	-0.051857	-2.928641	...	0.165192
485	-0.092191	-0.083921	-0.368619	-0.657635	-0.175135	-0.920385	-0.410625	0.514057	0.352897	-0.146452	...	0.361729
...	...	...	...	...	...	...	...	...	...	...	...	...
71	1.456030	0.166774	-0.368619	-0.657635	-1.287284	-0.920385	-0.410625	-1.235254	-0.139775	0.245829	...	1.077273
106	1.184431	0.166774	-0.368619	-0.657635	-1.287284	-0.920385	-0.410625	-1.138650	-1.380509	1.306929	...	-0.040478
270	0.193851	0.166774	-0.368619	-0.657635	0.086585	-0.920385	-0.410625	-0.804638	-0.130366	0.154210	...	0.558092
435	0.511425	1.482765	-0.368619	1.720850	1.460454	1.567972	-0.410625	2.415609	-0.051857	-1.154412	...	1.340342
102	-3.216783	0.166774	-0.368619	0.531608	0.086585	0.323793	3.791917	0.033022	-1.742143	0.538113	...	-3.063852

556 rows × 27 columns

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = np.array(X).reshape(X.shape[0], X.shape[1], 1)  # Shape (samples, features, channels)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.2, random_state=42)
# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)
```

```
y_train_categorical.shape
```

```
(556, 3)
```

```
# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(75, return_sequences=True))
model.add(LSTM(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification
```

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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
```

```
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.4273 - loss: 1.0892
Epoch 1: val_accuracy improved from -inf to 0.44286, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - accuracy: 0.4267 - loss: 1.0884 - val_accuracy: 0.4429 - val_loss: 1.0451
Epoch 2/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.4533 - loss: 1.0420
Epoch 2: val_accuracy improved from 0.44286 to 0.48571, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.4546 - loss: 1.0413 - val_accuracy: 0.4857 - val_loss: 1.0053
Epoch 3/200
17/18 ━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.5113 - loss: 0.9859
Epoch 3: val_accuracy improved from 0.48571 to 0.56429, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.5137 - loss: 0.9838 - val_accuracy: 0.5643 - val_loss: 0.9294
Epoch 4/200
17/18 ━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6080 - loss: 0.8778
Epoch 4: val_accuracy improved from 0.56429 to 0.60000, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.6044 - loss: 0.8798 - val_accuracy: 0.6000 - val_loss: 0.9062
Epoch 5/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6132 - loss: 0.8747
Epoch 5: val_accuracy improved from 0.60000 to 0.64286, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.6108 - loss: 0.8758 - val_accuracy: 0.6429 - val_loss: 0.8859
Epoch 6/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6181 - loss: 0.8586
Epoch 6: val_accuracy did not improve from 0.64286
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6184 - loss: 0.8574 - val_accuracy: 0.6286 - val_loss: 0.8812
Epoch 7/200
17/18 ━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.6324 - loss: 0.8282
Epoch 7: val_accuracy did not improve from 0.64286
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.6338 - loss: 0.8272 - val_accuracy: 0.6357 - val_loss: 0.8496
Epoch 8/200
17/18 ━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.6396 - loss: 0.8157
Epoch 8: val_accuracy did not improve from 0.64286
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.6391 - loss: 0.8167 - val_accuracy: 0.6071 - val_loss: 0.9367
Epoch 9/200
17/18 ━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.5720 - loss: 0.8781
Epoch 9: val_accuracy did not improve from 0.64286
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.5752 - loss: 0.8734 - val_accuracy: 0.5786 - val_loss: 0.8422
Epoch 10/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.6182 - loss: 0.8487
Epoch 10: val_accuracy improved from 0.64286 to 0.65000, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.6218 - loss: 0.8420 - val_accuracy: 0.6500 - val_loss: 0.8536
Epoch 11/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6583 - loss: 0.7947
Epoch 11: val_accuracy did not improve from 0.65000
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.6584 - loss: 0.7936 - val_accuracy: 0.6429 - val_loss: 0.8274
Epoch 12/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6262 - loss: 0.7958
Epoch 12: val_accuracy improved from 0.65000 to 0.66429, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.6306 - loss: 0.7924 - val_accuracy: 0.6643 - val_loss: 0.8093
Epoch 13/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6639 - loss: 0.7804
Epoch 13: val_accuracy did not improve from 0.66429
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.6604 - loss: 0.7851 - val_accuracy: 0.6214 - val_loss: 0.8563
Epoch 14/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6041 - loss: 0.8434
Epoch 14: val_accuracy improved from 0.66429 to 0.67143, saving model to best_model.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.6061 - loss: 0.8412 - val_accuracy: 0.6714 - val_loss: 0.8186
Epoch 15/200
16/18 ━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6518 - loss: 0.7549
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model.keras')

# Step 1: Evaluate the model on the test set
```

```
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8478 - loss: 0.7240
Test Accuracy: 0.8429
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 115ms/step
              precision    recall  f1-score   support

           0       0.83      0.80      0.81        54
           1       0.95      0.93      0.94        42
           2       0.77      0.82      0.79        44

    accuracy                           0.84       140
   macro avg       0.85      0.85      0.85       140
weighted avg       0.85      0.84      0.84       140
```

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = np.array(X).reshape(X.shape[0], X.shape[1], 1)  # Shape (samples, features, channels)


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

# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)

# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(SimpleRNN(100, return_sequences=True))
model.add(SimpleRNN(75, return_sequences=True))
model.add(SimpleRNN(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_rnn.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)
```

```
# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.5042 - loss: 1.0041
Epoch 1: val_accuracy improved from -inf to 0.65714, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 49ms/step - accuracy: 0.5170 - loss: 0.9867 - val_accuracy: 0.6571 - val_loss: 0.7908
Epoch 2/200
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6376 - loss: 0.8068
Epoch 2: val_accuracy did not improve from 0.65714
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6486 - loss: 0.7935 - val_accuracy: 0.6571 - val_loss: 0.7712
Epoch 3/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7422 - loss: 0.6227
Epoch 3: val_accuracy improved from 0.65714 to 0.70000, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7414 - loss: 0.6233 - val_accuracy: 0.7000 - val_loss: 0.7649
Epoch 4/200
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7617 - loss: 0.5751
Epoch 4: val_accuracy improved from 0.70000 to 0.76429, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7612 - loss: 0.5757 - val_accuracy: 0.7643 - val_loss: 0.7019
Epoch 5/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7865 - loss: 0.5816
Epoch 5: val_accuracy did not improve from 0.76429
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7878 - loss: 0.5760 - val_accuracy: 0.7357 - val_loss: 0.6742
Epoch 6/200
14/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7810 - loss: 0.5325
Epoch 6: val_accuracy improved from 0.76429 to 0.78571, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7813 - loss: 0.5305 - val_accuracy: 0.7857 - val_loss: 0.6220
Epoch 7/200
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8094 - loss: 0.4585
Epoch 7: val_accuracy did not improve from 0.78571
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8093 - loss: 0.4597 - val_accuracy: 0.7571 - val_loss: 0.5921
Epoch 8/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8547 - loss: 0.3927
Epoch 8: val_accuracy improved from 0.78571 to 0.79286, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8524 - loss: 0.3997 - val_accuracy: 0.7929 - val_loss: 0.5323
Epoch 9/200
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8485 - loss: 0.4352
Epoch 9: val_accuracy did not improve from 0.79286
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8475 - loss: 0.4306 - val_accuracy: 0.7857 - val_loss: 0.5614
Epoch 10/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8449 - loss: 0.3737
Epoch 10: val_accuracy improved from 0.79286 to 0.80714, saving model to best_model_rnn.keras
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8452 - loss: 0.3756 - val_accuracy: 0.8071 - val_loss: 0.5580
Epoch 11/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8684 - loss: 0.3469
Epoch 11: val_accuracy did not improve from 0.80714
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.8647 - loss: 0.3515 - val_accuracy: 0.7929 - val_loss: 0.4947
Epoch 12/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9108 - loss: 0.2934
Epoch 12: val_accuracy did not improve from 0.80714
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9062 - loss: 0.2987 - val_accuracy: 0.7714 - val_loss: 0.5191
Epoch 13/200
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9158 - loss: 0.2664
Epoch 13: val_accuracy did not improve from 0.80714
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9120 - loss: 0.2755 - val_accuracy: 0.7786 - val_loss: 0.5261
Epoch 14/200
16/18 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8747 - loss: 0.3271
Epoch 14: val_accuracy did not improve from 0.80714
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8773 - loss: 0.3229 - val_accuracy: 0.7714 - val_loss: 0.5939
Epoch 15/200
15/18 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9148 - loss: 0.2609
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_rnn.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8769 - loss: 0.5430
Test Accuracy: 0.8571
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 151ms/step
              precision    recall  f1-score   support
```

```
              0       0.84      0.85      0.84        54
              1       0.91      0.95      0.93        42
              2       0.83      0.77      0.80        44

       accuracy                           0.86       140
      macro avg       0.86      0.86      0.86       140
   weighted avg       0.86      0.86      0.86       140
```

```
# Load the best model
best_model = load_model('best_model.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/lstm_model_cirr.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
best_model = load_model('best_model_rnn.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/rnn_model_cirr.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
X = X.values
y=y.values


import numpy as np

def augment_data(X, y, num_augmentations=5):
    X_augmented = []
    y_augmented = []

    for i in range(len(X)):
        # Original sample
        X_augmented.append(X[i])
        y_augmented.append(y[i])

        for _ in range(num_augmentations):
            # Create new augmented samples
            sample = X[i]

            # Add noise
            noise = np.random.normal(0, 0.01, sample.shape)
            augmented_sample = sample + noise

            # Time shifting
            if np.random.rand() > 0.5:  # Randomly decide to shift
                shift = np.random.randint(-5, 5)  # Shift by a random amount
                augmented_sample = np.roll(augmented_sample, shift)

            # Append augmented sample and corresponding label
            X_augmented.append(augmented_sample)
            y_augmented.append(y[i])

    return np.array(X_augmented), np.array(y_augmented)

# Generate augmented data
X_augmented, y_augmented = augment_data(X, y)

# Optionally, you can concatenate the original and augmented data
X_combined = np.concatenate((X, X_augmented), axis=0)
y_combined = np.concatenate((y, y_augmented), axis=0)


X_combined.shape
```

```
(4872, 27)
```

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = X_combined.reshape(X_combined.shape[0], X_combined.shape[1], 1)  # Shape (samples, features, channels)


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_combined, test_size=0.2, random_state=42)

# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)
```

```
# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(SimpleRNN(100, return_sequences=True))
model.add(SimpleRNN(75, return_sequences=True))
model.add(SimpleRNN(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification


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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_rnn2.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
119/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5812 - loss: 0.9114
Epoch 1: val_accuracy improved from -inf to 0.67692, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.5824 - loss: 0.9094 - val_accuracy: 0.6769 - val_loss: 0.7215
Epoch 2/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7041 - loss: 0.7090
Epoch 2: val_accuracy improved from 0.67692 to 0.71692, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7040 - loss: 0.7090 - val_accuracy: 0.7169 - val_loss: 0.6397
Epoch 3/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.7448 - loss: 0.6203
Epoch 3: val_accuracy improved from 0.71692 to 0.77436, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.7449 - loss: 0.6201 - val_accuracy: 0.7744 - val_loss: 0.5855
Epoch 4/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7705 - loss: 0.5520
Epoch 4: val_accuracy did not improve from 0.77436
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7705 - loss: 0.5521 - val_accuracy: 0.7713 - val_loss: 0.5232
Epoch 5/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8016 - loss: 0.4790
Epoch 5: val_accuracy improved from 0.77436 to 0.80410, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.8017 - loss: 0.4788 - val_accuracy: 0.8041 - val_loss: 0.4488
Epoch 6/200
118/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8241 - loss: 0.4252
Epoch 6: val_accuracy improved from 0.80410 to 0.84718, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.8243 - loss: 0.4249 - val_accuracy: 0.8472 - val_loss: 0.4039
Epoch 7/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8583 - loss: 0.3686
Epoch 7: val_accuracy did not improve from 0.84718
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.8583 - loss: 0.3686 - val_accuracy: 0.8308 - val_loss: 0.4182
Epoch 8/200
118/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8779 - loss: 0.3289
Epoch 8: val_accuracy improved from 0.84718 to 0.86769, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.8779 - loss: 0.3290 - val_accuracy: 0.8677 - val_loss: 0.3392
Epoch 9/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8868 - loss: 0.3078
Epoch 9: val_accuracy improved from 0.86769 to 0.87692, saving model to best_model_rnn2.keras
```

```
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.8868 - loss: 0.3077 - val_accuracy: 0.8769 - val_loss: 0.3222
Epoch 10/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9023 - loss: 0.2584
Epoch 10: val_accuracy improved from 0.87692 to 0.88410, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9023 - loss: 0.2585 - val_accuracy: 0.8841 - val_loss: 0.2935
Epoch 11/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9245 - loss: 0.2070
Epoch 11: val_accuracy improved from 0.88410 to 0.89128, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9244 - loss: 0.2070 - val_accuracy: 0.8913 - val_loss: 0.2875
Epoch 12/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9216 - loss: 0.2116
Epoch 12: val_accuracy improved from 0.89128 to 0.91692, saving model to best_model_rnn2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.9215 - loss: 0.2117 - val_accuracy: 0.9169 - val_loss: 0.2514
Epoch 13/200
120/122 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9199 - loss: 0.2139
Epoch 13: val_accuracy did not improve from 0.91692
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.9200 - loss: 0.2136 - val_accuracy: 0.8964 - val_loss: 0.2710
Epoch 14/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9436 - loss: 0.1588
Epoch 14: val_accuracy did not improve from 0.91692
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9436 - loss: 0.1589 - val_accuracy: 0.8954 - val_loss: 0.2750
Epoch 15/200
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_rnn2.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9563 - loss: 0.2245
Test Accuracy: 0.9549
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       330
           1       0.97      0.97      0.97       313
           2       0.95      0.95      0.95       332

    accuracy                           0.95       975
   macro avg       0.96      0.96      0.96       975
weighted avg       0.95      0.95      0.95       975
```

```
# Reshape X for CNN input
# Convert to numpy array and reshape X for CNN input
X_reshaped = X_combined.reshape(X_combined.shape[0], X_combined.shape[1], 1)  # Shape (samples, features, channels)


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_combined, test_size=0.2, random_state=42)

# Convert labels to categorical
from tensorflow.keras.utils import to_categorical
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)

# Step 2: Build the CNN + RNN Model
model = Sequential()

# CNN Layers
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# RNN Layer
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(75, return_sequences=True))
model.add(LSTM(50, return_sequences=False))  # You can use return_sequences=True if stacking more RNN layers

# Output Layer
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))  # Assuming binary classification
```

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

# Define the checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model_lstm2.keras',  # Path to save the model file
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Only save the model if validation accuracy improves
    mode='max',  # We're interested in the maximum accuracy
    verbose=1  # Print messages when saving the model
)

# Train the model with the checkpoint callback
history = model.fit(
    X_train,
    y_train_categorical,
    validation_data=(X_test,y_test_categorical),
    epochs=200,
    callbacks=[checkpoint]  # Add the checkpoint to the callbacks
)

# Step 4: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 5: Predictions (optional)
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# If you want to see a classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_classes))
```

```
Epoch 1/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.4610 - loss: 1.0218
Epoch 1: val_accuracy improved from -inf to 0.60410, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 9s 30ms/step - accuracy: 0.4620 - loss: 1.0211 - val_accuracy: 0.6041 - val_loss: 0.8827
Epoch 2/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.6017 - loss: 0.8844
Epoch 2: val_accuracy improved from 0.60410 to 0.63692, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - accuracy: 0.6018 - loss: 0.8842 - val_accuracy: 0.6369 - val_loss: 0.8335
Epoch 3/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.6428 - loss: 0.8206
Epoch 3: val_accuracy improved from 0.63692 to 0.66974, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 5s 39ms/step - accuracy: 0.6429 - loss: 0.8206 - val_accuracy: 0.6697 - val_loss: 0.7776
Epoch 4/200
120/122 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6589 - loss: 0.7847
Epoch 4: val_accuracy improved from 0.66974 to 0.69744, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.6589 - loss: 0.7847 - val_accuracy: 0.6974 - val_loss: 0.7290
Epoch 5/200
120/122 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6715 - loss: 0.7548
Epoch 5: val_accuracy did not improve from 0.69744
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 24ms/step - accuracy: 0.6716 - loss: 0.7548 - val_accuracy: 0.6851 - val_loss: 0.7224
Epoch 6/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.6925 - loss: 0.7201
Epoch 6: val_accuracy did not improve from 0.69744
122/122 ━━━━━━━━━━━━━━━━━━━━ 7s 41ms/step - accuracy: 0.6925 - loss: 0.7202 - val_accuracy: 0.6892 - val_loss: 0.7240
Epoch 7/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6967 - loss: 0.7038
Epoch 7: val_accuracy improved from 0.69744 to 0.73744, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.6967 - loss: 0.7038 - val_accuracy: 0.7374 - val_loss: 0.6349
Epoch 8/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6905 - loss: 0.7013
Epoch 8: val_accuracy improved from 0.73744 to 0.74256, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.6907 - loss: 0.7011 - val_accuracy: 0.7426 - val_loss: 0.6054
Epoch 9/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7274 - loss: 0.6200
Epoch 9: val_accuracy improved from 0.74256 to 0.77333, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 6s 34ms/step - accuracy: 0.7274 - loss: 0.6200 - val_accuracy: 0.7733 - val_loss: 0.5474
Epoch 10/200
120/122 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.7492 - loss: 0.5856
Epoch 10: val_accuracy did not improve from 0.77333
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.7492 - loss: 0.5857 - val_accuracy: 0.7703 - val_loss: 0.5127
Epoch 11/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7621 - loss: 0.5479
Epoch 11: val_accuracy improved from 0.77333 to 0.78667, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 0.7621 - loss: 0.5479 - val_accuracy: 0.7867 - val_loss: 0.5058
Epoch 12/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7843 - loss: 0.5254
Epoch 12: val_accuracy improved from 0.78667 to 0.81744, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.7843 - loss: 0.5254 - val_accuracy: 0.8174 - val_loss: 0.4548
Epoch 13/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.8042 - loss: 0.4657
Epoch 13: val_accuracy improved from 0.81744 to 0.84821, saving model to best_model_lstm2.keras
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.8042 - loss: 0.4658 - val_accuracy: 0.8482 - val_loss: 0.3718
```

```
Epoch 14/200
121/122 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8313 - loss: 0.4066
Epoch 14: val_accuracy did not improve from 0.84821
122/122 ━━━━━━━━━━━━━━━━━━━━ 4s 35ms/step - accuracy: 0.8312 - loss: 0.4070 - val_accuracy: 0.8174 - val_loss: 0.4488
Epoch 15/200
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8365 - loss: 0.4246
```

```
import numpy as np
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report

# Load the best model
best_model = load_model('best_model_lstm2.keras')

# Step 1: Evaluate the model on the test set
loss, accuracy = best_model.evaluate(X_test, y_test_categorical)
print(f'Test Accuracy: {accuracy:.4f}')

# Step 2: Predictions using the best model
predictions = best_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Step 3: Classification report
print(classification_report(y_test, predicted_classes))
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9921 - loss: 0.0218
Test Accuracy: 0.9928
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       330
           1       1.00      0.99      1.00       313
           2       0.99      1.00      0.99       332

    accuracy                           0.99       975
   macro avg       0.99      0.99      0.99       975
weighted avg       0.99      0.99      0.99       975
```

```
best_model = load_model('best_model_rnn2.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/rnn_model2_cirr.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
# Load the best model
best_model = load_model('best_model_lstm2.keras')
best_model.save('/content/drive/My Drive/Honors Deep Learning/lstm_model2_cirr.h5')  # Saves the model as an HDF5 file
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
X_train, X_test, y_train, y_test = train_test_split(X_combined, y_combined, test_size=0.2, random_state=42)
```

```
tabnet_model = TabNetClassifier(optimizer_fn=torch.optim.Adam,
                                optimizer_params=dict(lr=0.0001)) # Pass learning rate to optimizer_params
# Fit the model using NumPy arrays
tabnet_model.fit(X_train, y_train,
                 eval_set=[(X_test, y_test)],
                 eval_name=['test'],
                 eval_metric=['accuracy'],
                 max_epochs=200,
                 patience=100)
```

```
epoch 0  | loss: 1.36007 | test_accuracy: 0.34256 |  0:00:00s
epoch 1  | loss: 1.36525 | test_accuracy: 0.33949 |  0:00:01s
epoch 2  | loss: 1.34169 | test_accuracy: 0.34154 |  0:00:01s
epoch 3  | loss: 1.35852 | test_accuracy: 0.34154 |  0:00:01s
epoch 4  | loss: 1.36022 | test_accuracy: 0.33949 |  0:00:02s
epoch 5  | loss: 1.34867 | test_accuracy: 0.33333 |  0:00:02s
epoch 6  | loss: 1.3424  | test_accuracy: 0.34256 |  0:00:02s
epoch 7  | loss: 1.34211 | test_accuracy: 0.33333 |  0:00:03s
epoch 8  | loss: 1.34285 | test_accuracy: 0.32615 |  0:00:03s
epoch 9  | loss: 1.35653 | test_accuracy: 0.33128 |  0:00:03s
epoch 10 | loss: 1.33498 | test_accuracy: 0.33128 |  0:00:04s
epoch 11 | loss: 1.33878 | test_accuracy: 0.33333 |  0:00:04s
epoch 12 | loss: 1.33551 | test_accuracy: 0.33333 |  0:00:05s
epoch 13 | loss: 1.32144 | test_accuracy: 0.33231 |  0:00:05s
epoch 14 | loss: 1.3254  | test_accuracy: 0.33231 |  0:00:05s
epoch 15 | loss: 1.31459 | test_accuracy: 0.33333 |  0:00:06s
epoch 16 | loss: 1.3174  | test_accuracy: 0.33641 |  0:00:06s
epoch 17 | loss: 1.30956 | test_accuracy: 0.33744 |  0:00:06s
```

```
epoch 18 | loss: 1.31503 | test_accuracy: 0.33949 |  0:00:06s
epoch 19 | loss: 1.31528 | test_accuracy: 0.34051 |  0:00:07s
epoch 20 | loss: 1.3135  | test_accuracy: 0.33231 |  0:00:07s
epoch 21 | loss: 1.32862 | test_accuracy: 0.33333 |  0:00:07s
epoch 22 | loss: 1.30878 | test_accuracy: 0.34051 |  0:00:07s
epoch 23 | loss: 1.29751 | test_accuracy: 0.33231 |  0:00:07s
epoch 24 | loss: 1.30143 | test_accuracy: 0.33436 |  0:00:08s
epoch 25 | loss: 1.30394 | test_accuracy: 0.33436 |  0:00:08s
epoch 26 | loss: 1.30404 | test_accuracy: 0.33641 |  0:00:08s
epoch 27 | loss: 1.3065  | test_accuracy: 0.32615 |  0:00:08s
epoch 28 | loss: 1.28891 | test_accuracy: 0.32615 |  0:00:08s
epoch 29 | loss: 1.2743  | test_accuracy: 0.32923 |  0:00:09s
epoch 30 | loss: 1.31156 | test_accuracy: 0.32718 |  0:00:09s
epoch 31 | loss: 1.27827 | test_accuracy: 0.33128 |  0:00:09s
epoch 32 | loss: 1.27875 | test_accuracy: 0.33436 |  0:00:09s
epoch 33 | loss: 1.29241 | test_accuracy: 0.34462 |  0:00:09s
epoch 34 | loss: 1.28875 | test_accuracy: 0.34667 |  0:00:10s
epoch 35 | loss: 1.28534 | test_accuracy: 0.34051 |  0:00:10s
epoch 36 | loss: 1.25523 | test_accuracy: 0.33026 |  0:00:10s
epoch 37 | loss: 1.29649 | test_accuracy: 0.33744 |  0:00:10s
epoch 38 | loss: 1.26589 | test_accuracy: 0.33026 |  0:00:11s
epoch 39 | loss: 1.25656 | test_accuracy: 0.34154 |  0:00:11s
epoch 40 | loss: 1.28717 | test_accuracy: 0.34051 |  0:00:12s
epoch 41 | loss: 1.27622 | test_accuracy: 0.33641 |  0:00:13s
epoch 42 | loss: 1.26857 | test_accuracy: 0.33538 |  0:00:13s
epoch 43 | loss: 1.25469 | test_accuracy: 0.33641 |  0:00:14s
epoch 44 | loss: 1.27575 | test_accuracy: 0.33846 |  0:00:15s
epoch 45 | loss: 1.28127 | test_accuracy: 0.33538 |  0:00:15s
epoch 46 | loss: 1.2668  | test_accuracy: 0.33128 |  0:00:16s
epoch 47 | loss: 1.23553 | test_accuracy: 0.33641 |  0:00:17s
epoch 48 | loss: 1.25288 | test_accuracy: 0.33744 |  0:00:17s
epoch 49 | loss: 1.27032 | test_accuracy: 0.33231 |  0:00:18s
epoch 50 | loss: 1.24757 | test_accuracy: 0.33231 |  0:00:18s
epoch 51 | loss: 1.25993 | test_accuracy: 0.33436 |  0:00:19s
epoch 52 | loss: 1.26219 | test_accuracy: 0.33949 |  0:00:19s
epoch 53 | loss: 1.23912 | test_accuracy: 0.33538 |  0:00:19s
epoch 54 | loss: 1.24379 | test_accuracy: 0.33128 |  0:00:19s
epoch 55 | loss: 1.25437 | test_accuracy: 0.33333 |  0:00:20s
epoch 56 | loss: 1.25389 | test_accuracy: 0.33744 |  0:00:20s
epoch 57 | loss: 1.23275 | test_accuracy: 0.33949 |  0:00:20s
```

```
# Make predictions
y_pred = tabnet_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.4256
                precision    recall  f1-score   support

             0       0.45      0.50      0.47       330
             1       0.47      0.06      0.10       313
             2       0.40      0.70      0.51       332

      accuracy                           0.43       975
     macro avg       0.44      0.42      0.36       975
  weighted avg       0.44      0.43      0.37       975
```

```
# Load the best model
tabnet_model.save_model('/content/drive/My Drive/Honors Deep Learning/tabnet_model_cirr.h5')  # Saves the model as an HDF5 file
```

```
Successfully saved model at /content/drive/My Drive/Honors Deep Learning/tabnet_model_cirr.h5.zip
'/content/drive/My Drive/Honors Deep Learning/tabnet_model_cirr.h5.zip'
```

```
from tensorflow.keras.models import load_model

# Load the model
rnn_model_cirr = load_model('/content/drive/My Drive/Honors Deep Learning/rnn_model_cirr.h5')

# Verify the loaded model's architecture
rnn_model_cirr.summary()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
conv1d_12 (Conv1D)	(None, 25, 32)	128
max_pooling1d_12 (MaxPooling1D)	(None, 12, 32)	0
dropout_99 (Dropout)	(None, 12, 32)	0
simple_rnn_6 (SimpleRNN)	(None, 12, 100)	13,300
simple_rnn_7 (SimpleRNN)	(None, 12, 75)	13,200
simple_rnn_8 (SimpleRNN)	(None, 50)	6,300
dense_86 (Dense)	(None, 3)	153

Total params: 33,083 (129.23 KB)
Trainable params: 33,081 (129.22 KB)

```
# Prepare a subset of the data for explanation (e.g., first 100 samples)
X_explain = X_train[:100]
print(X_explain)
```

```
        N_Days      Drug       Sex   Ascites  Hepatomegaly   Spiders     Edema  \
82   1.414135 -1.149217 -0.368619 -0.657635      0.086585 -0.920385  1.690646
51   0.733210 -1.149217  3.093598 -0.657635     -1.287284 -0.920385 -0.410625
218  0.603372  0.166774 -0.368619 -0.657635     -1.287284 -0.920385 -0.410625
557 -0.927872  1.482765 -0.368619  1.720850      1.460454  1.567972 -0.410625
485 -0.092191 -0.083921 -0.368619 -0.657635     -0.175135 -0.920385 -0.410625
..        ...       ...       ...       ...           ...       ...       ...
256  0.230231 -1.149217 -0.368619 -0.657635     -1.287284 -0.920385 -0.410625
9   -4.192979  0.166774 -0.368619  0.531608     -1.287284  0.323793  3.791917
22  -2.096513  0.166774 -0.368619  0.531608      0.086585  0.323793  3.791917
302 -0.098553  0.166774 -0.368619 -0.657635      0.086585  0.323793 -0.410625
221 -1.048772  0.166774 -0.368619 -0.657635      0.086585 -0.920385 -0.410625

     Bilirubin  Cholesterol   Albumin  ...  Coagulation    Lipids     Liver  \
82   -0.595435    -0.883909  0.154210  ...    -0.978826  0.869578 -0.106993
51    1.070558     1.827076  0.604407  ...     1.016344  1.818654 -0.895651
218  -0.411918    -1.338199  1.591144  ...     0.237161  0.252863 -0.047840
557   1.034976    -0.051857 -2.928641  ...     0.165192  0.105336  0.193853
485   0.514057     0.352897 -0.146452  ...     0.361729 -0.203206 -0.908867
..         ...          ...       ...  ...          ...       ...       ...
256  -1.235254    -0.542372  1.710602  ...     0.042445 -0.578339  1.827000
9     2.064703    -1.555925 -1.760991  ...    -3.046172 -4.317827 -1.433146
22    2.517172     0.495411 -1.221656  ...    -1.859834 -2.387654 -1.338095
302  -0.804638     0.314403 -0.437545  ...    -0.086244  1.363991  0.353667
221   0.709575     0.314403 -0.125615  ...    -1.143647 -1.157418 -0.965825

     BloodPressure      Fats     Foods  Age_in_years  combination1  \
82        0.551852  0.684781 -0.046005      0.832397     -0.313464
51       -0.936516  1.470780 -1.001215      0.328326     -1.023097
218       0.200318  0.461763  0.068687     -0.496636     -0.014374
557       0.203160  0.094022  0.182752     -0.656400      0.187708
485      -0.995219 -0.321675 -0.936859     -0.204375     -0.782765
..             ...       ...       ...           ...           ...
256       2.322847 -0.747547  1.849121      1.025520      2.783354
9        -1.510602 -3.193821 -1.376148      1.964251     -1.257927
22       -1.364478 -2.128007 -1.349571      0.828321     -1.428706
302       0.283599  2.024739  0.266232      1.223066      0.578598
221      -0.921785 -1.055395 -0.989429     -0.105116     -0.897875

     combination2  combination3
82      -1.130386      0.085347
51       1.519679     -0.645127
218     -0.156686     -0.719678
557     -0.164040      0.131396
485      0.990179     -0.309548
..            ...           ...
256     -1.345232      1.222444
9        2.480748     -1.039311
22       2.117064     -1.364716
302     -0.884503     -1.263904
221      0.700524     -1.009806

[100 rows x 27 columns]
```

```
# prompt: give feature names of the above in list

feature_names = list(X.columns)
print(feature_names)
```

```
['N_Days', 'Drug', 'Sex', 'Ascites', 'Hepatomegaly', 'Spiders', 'Edema', 'Bilirubin', 'Cholesterol', 'Albumin', 'Copper', 'Alk_Phos
```

```
# Initialize the SHAP explainer
explainer = shap.KernelExplainer(rnn_model_cirr.predict, X_explain)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 256ms/step
```

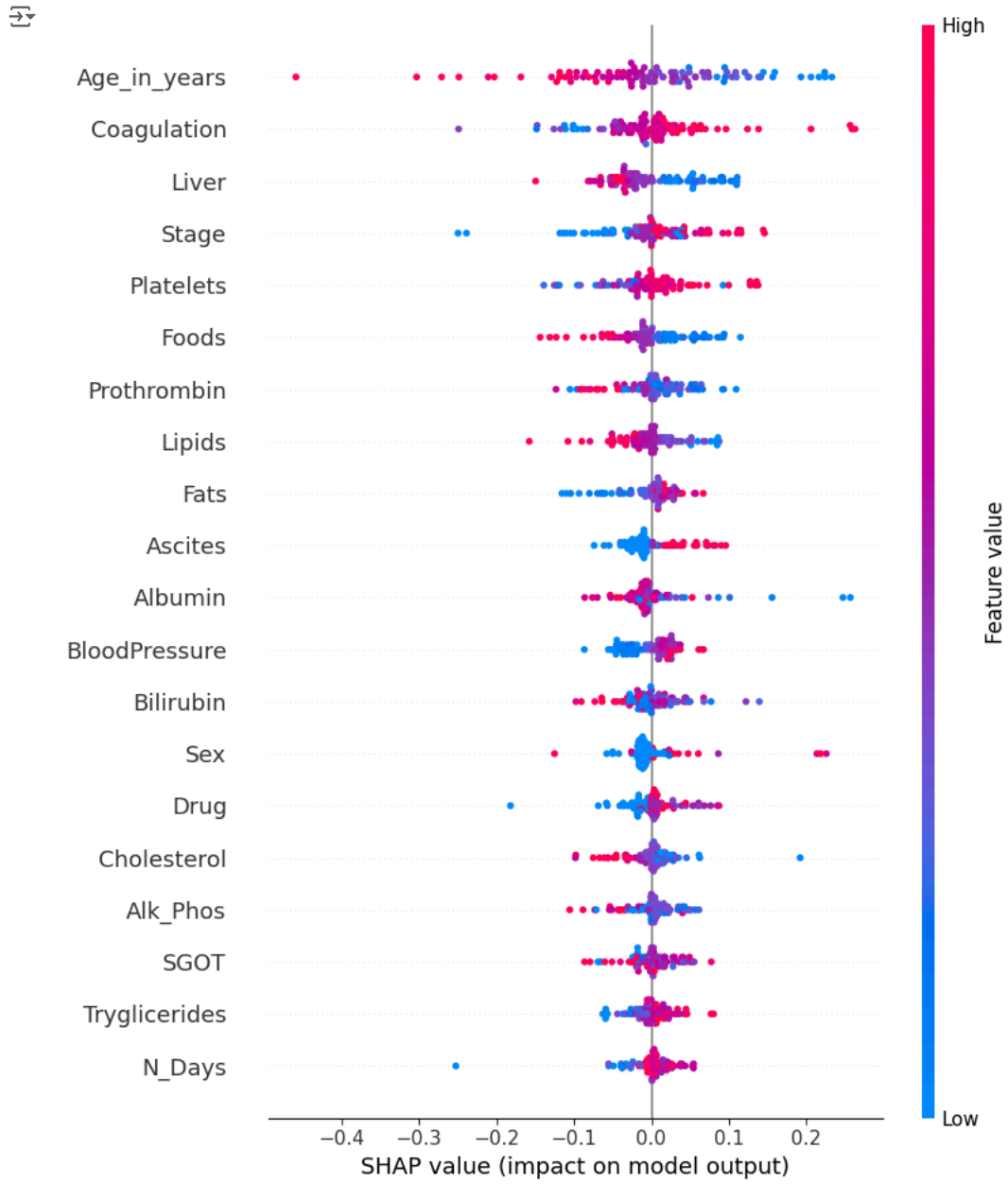
```
print(rnn_model_cirr.predict(X_explain))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
[[9.98932183e-01 3.08764902e-05 1.03700778e-03]
 [9.18514386e-04 9.09476657e-05 9.98990536e-01]
 [9.99522150e-01 4.13960195e-04 6.38690690e-05]
 [4.38405812e-04 9.99021113e-01 5.40478097e-04]
 [3.13017663e-05 9.99924362e-01 4.42699311e-05]
 [5.18806955e-05 9.99275446e-01 6.72715018e-04]
 [3.10079264e-03 9.96882260e-01 1.68650513e-05]
 [9.57125664e-01 1.47776641e-02 2.80965976e-02]
 [9.98057544e-01 1.64367887e-03 2.98774161e-04]
 [6.41015870e-03 9.93427157e-01 1.62756696e-04]
 [9.58777249e-01 8.31886660e-03 3.29038873e-02]
 [4.96416542e-05 9.99804974e-01 1.45438564e-04]
 [6.86107742e-05 5.79217931e-06 9.99925554e-01]
 [2.16116617e-03 3.84789746e-05 9.97800350e-01]
 [1.06108928e-04 9.99328315e-01 5.65479160e-04]
 [7.88130332e-03 3.23085207e-03 9.88887787e-01]
 [6.49798221e-06 8.30895351e-06 9.99985158e-01]
 [9.97169197e-01 2.70219706e-03 1.28596090e-04]
 [2.79329484e-04 9.99686599e-01 3.41245468e-05]
 [9.59286332e-01 3.47200334e-02 5.99364424e-03]
 [8.33262086e-01 9.86997737e-04 1.65750921e-01]
 [5.59353539e-05 5.91801654e-05 9.99884784e-01]
 [3.73231276e-04 2.13167686e-02 9.78309989e-01]
 [6.21850922e-05 6.95815324e-05 9.99868393e-01]
 [1.24019757e-03 1.82353673e-04 9.98577476e-01]
 [3.44551262e-03 4.36947728e-03 9.92185116e-01]
 [9.99900401e-01 9.67410742e-05 2.69343786e-06]
 [3.11030052e-03 9.96855497e-01 3.42297353e-05]
 [2.03411398e-03 9.97862935e-01 1.02900049e-04]
 [9.98342693e-01 1.26151726e-05 1.64477108e-03]
 [2.13061292e-02 5.70897428e-05 9.78636801e-01]
 [9.22853302e-04 9.98975039e-01 1.02022961e-04]
 [9.98900771e-01 8.08023091e-04 2.91289849e-04]
 [2.24126037e-03 1.00293782e-05 9.97748733e-01]
 [1.77361333e-04 3.83954875e-06 9.99818802e-01]
 [1.63187995e-03 9.97815609e-01 5.52548212e-04]
 [2.00322116e-04 9.99723554e-01 7.61490082e-05]
 [8.45673087e-04 9.99137044e-01 1.72703149e-05]
 [2.15955474e-03 9.97827470e-01 1.29096870e-05]
 [1.55722373e-04 1.52821588e-06 9.99842882e-01]
 [4.66099055e-03 9.95331407e-01 7.59872773e-06]
 [7.02547003e-03 9.92768049e-01 2.06502547e-04]
 [9.71167922e-01 2.05458757e-02 8.28616414e-03]
 [2.25602533e-04 9.99698997e-01 7.54104403e-05]
 [9.94046509e-01 1.36426665e-04 5.81705803e-03]
 [1.06208737e-03 9.96999204e-01 1.93871255e-03]
 [9.54182684e-01 4.18963023e-02 3.92114092e-03]
 [1.84517165e-04 1.67402629e-06 9.99813795e-01]
 [1.49172638e-02 1.84592139e-02 9.66623545e-01]
 [3.45313165e-04 9.99453664e-01 2.01016781e-04]
 [4.76718502e-04 9.99258280e-01 2.64984847e-04]
 [2.26232200e-03 9.97724593e-01 1.31392462e-05]
 [6.30839262e-04 9.98864055e-01 5.05062810e-04]
 [9.81004953e-01 1.44250365e-02 4.56999335e-03]
 [3.73584358e-03 1.54931404e-05 9.96248722e-01]
 [9.60926771e-01 5.88067749e-04 3.84851098e-02]
 [9.86704946e-01 1.05644074e-04 1.31894248e-02]
```

```
# Calculate SHAP values
shap_values_cirr = explainer.shap_values(X_explain)
```

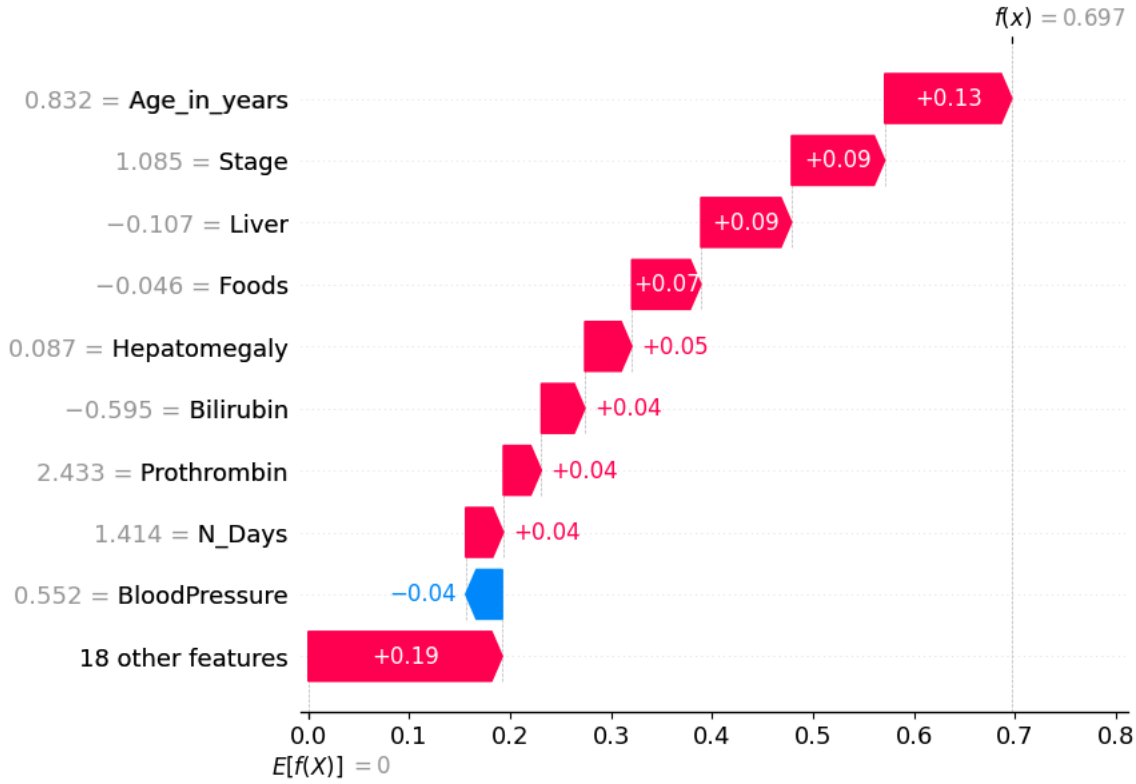
```
100%                                                  100/100 [1:35:28<00:00, 52.86s/it]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 55s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 43s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 43s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 42s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 38s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 58s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 62s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 44s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 60s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 38s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 40s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 37s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 37s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 37s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 39s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 37s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 37s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 41s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```

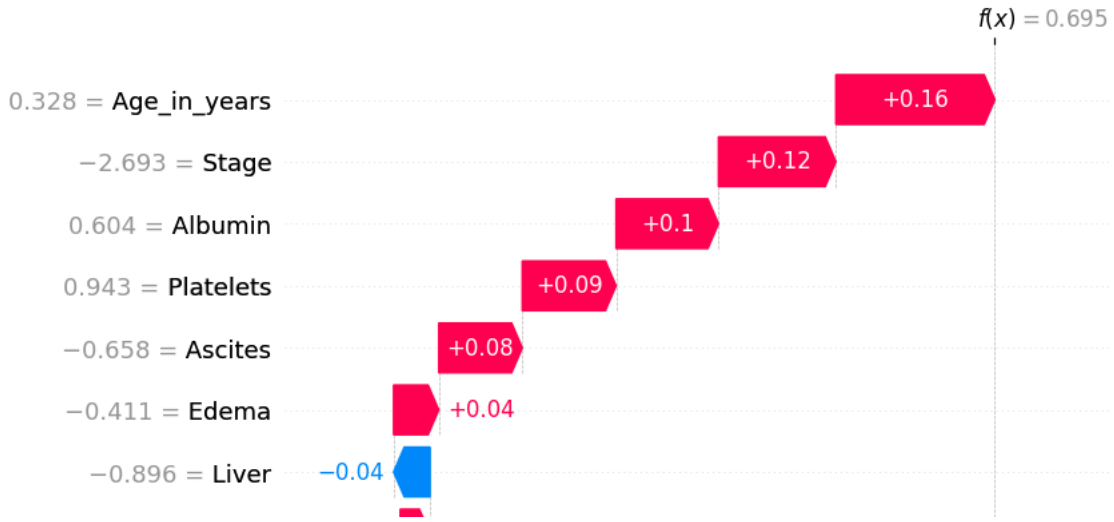
```
# Select SHAP values for the first output (class 0)
shap_values_class_0 = shap_values_cirr[:, :, 1]

# Plot the summary for the first output
shap.summary_plot(shap_values_class_0, X_explain, feature_names=feature_names)
```




```
6569/6569 ━━━━━━━━━━━━━━━━━━━━ 38s 6ms/step
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5

plt.figure(figsize=(40, 10))
# Loop through each instance
for i in range(num_instances):
    # Create an Explanation object
    expl = shap.Explanation(values=shap_values_cirr[5+i,:,1],
                            data=X_explain.iloc[i],
                            feature_names=feature_names)

    # Set base_values attribute
    expl.base_values = np.mean(shap_values_cirr)
    # Plot the waterfall plot
    shap.plots.waterfall(expl)

plt.tight_layout()
plt.show()
```





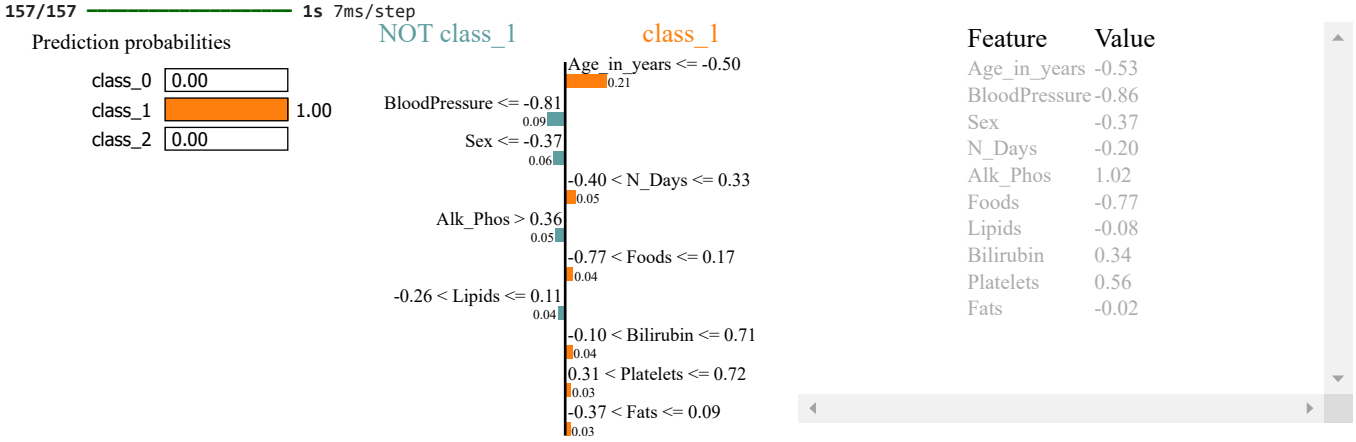

```
# Importing the module for LimeTabularExplainer
from lime import lime_tabular

# Instantiating the explainer object by passing in the training set,
# and the extracted features
# Convert the NumPy array X_train to a DataFrame
X_train_df = pd.DataFrame(X_explain, columns=feature_names)

X_train_df.reset_index(drop=True, inplace=True)

# Make sure that the feature names (col) match the columns of your DataFrame
col = X_train_df.columns.tolist()

# Instantiate the explainer object by passing in the DataFrame
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_df.values,
                                                   feature_names=feature_names,
                                                   class_names=['class_0', 'class_1','class_2'],
                                                   mode='classification')

print(X_explain.iloc[0])
```

```
N_Days           1.414135
Drug            -1.149217
Sex             -0.368619
Ascites         -0.657635
Hepatomegaly     0.086585
Spiders         -0.920385
Edema            1.690646
Bilirubin       -0.595435
Cholesterol     -0.883909
```

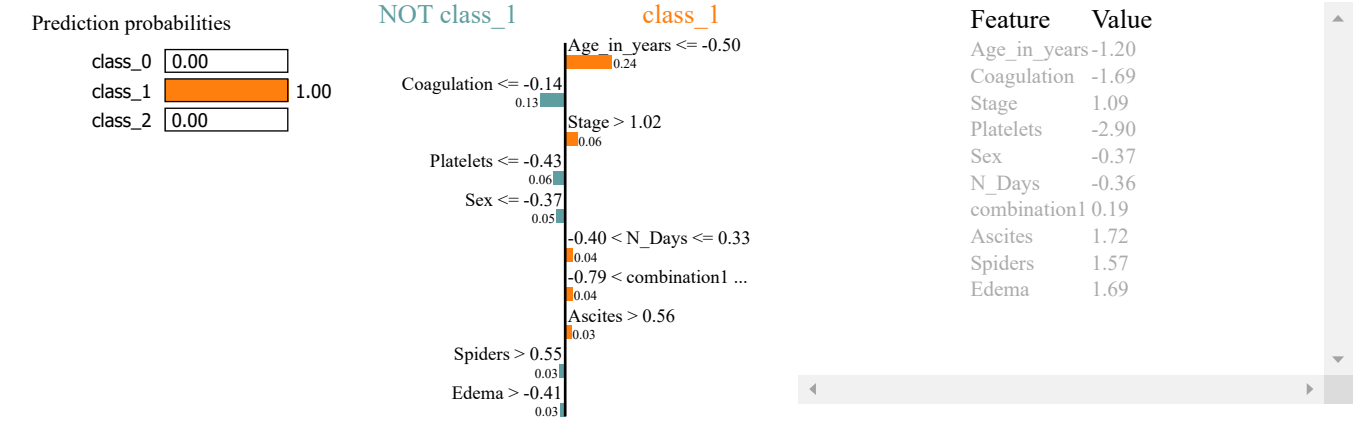
```
Albumin          0.154210
Copper          -0.782862
Alk_Phos        -0.434262
SGOT            -1.417953
Tryglicerides   -0.390242
Platelets       -2.744304
Prothrombin      2.433248
Stage            1.085259
Coagulation     -0.978826
Lipids           0.869578
Liver           -0.106993
BloodPressure    0.551852
Fats             0.684781
Foods           -0.046005
Age_in_years     0.832397
combination1    -0.313464
combination2    -1.130386
combination3     0.085347
Name: 82, dtype: float64
```

```
import matplotlib.pyplot as plt

# Number of instances to explain
num_instances = 5


# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[5+i]

    # Explain the prediction for the instance
    explanation = explainer.explain_instance(instance, rnn_model_cirr.predict)

    # Plot the explanation in the corresponding subplot
    explanation.show_in_notebook()
```

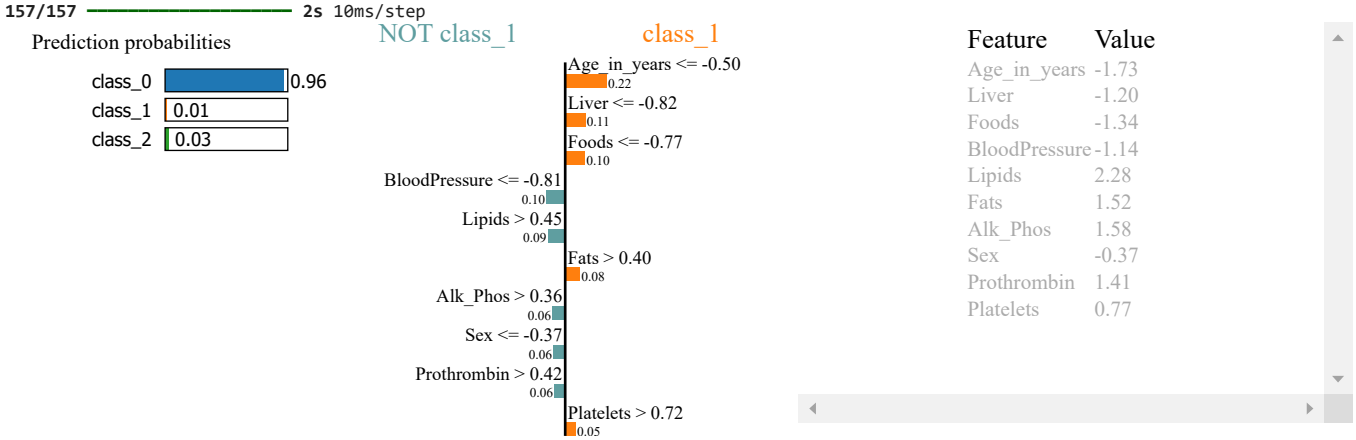
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step



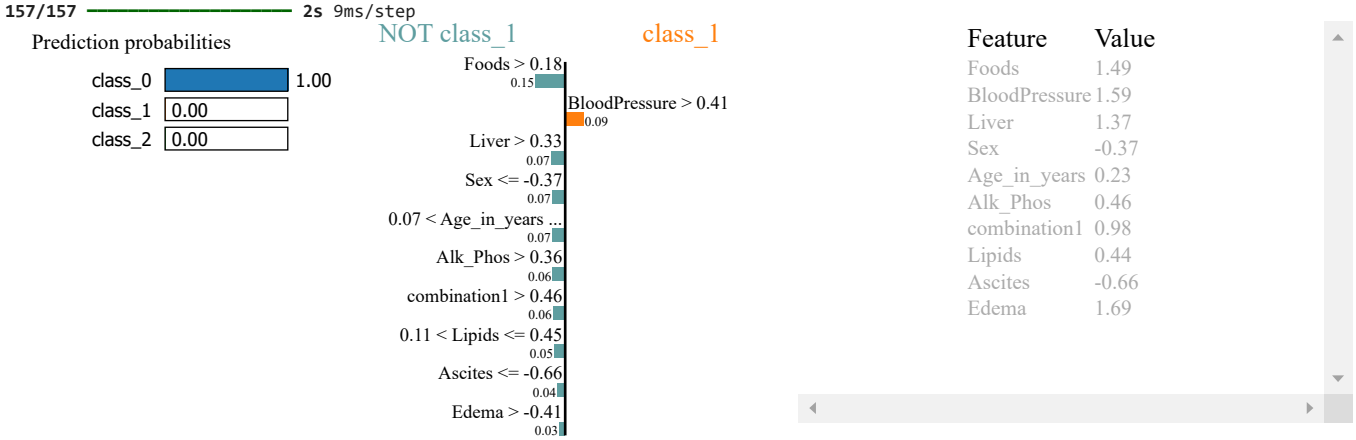
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step



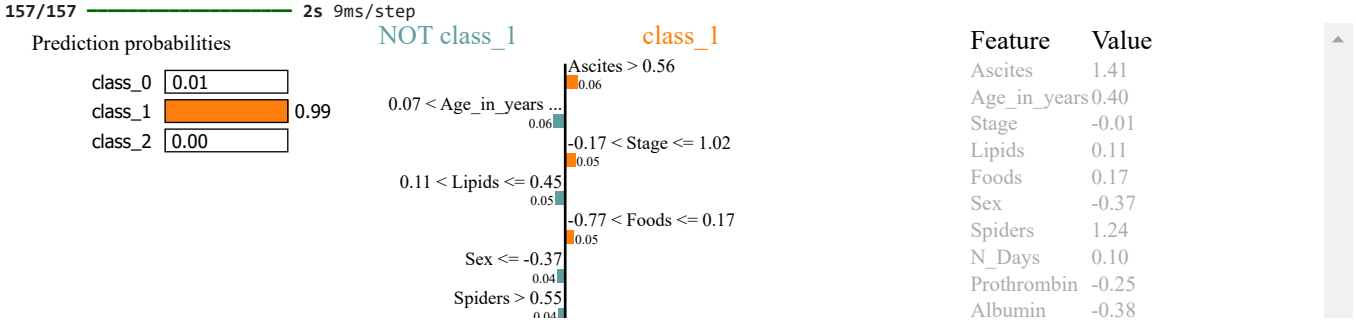
157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step

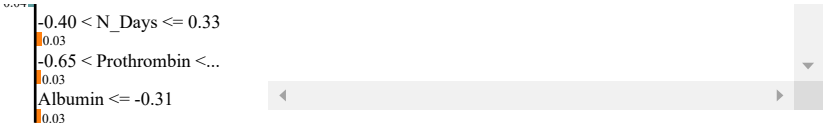


157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step



157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step

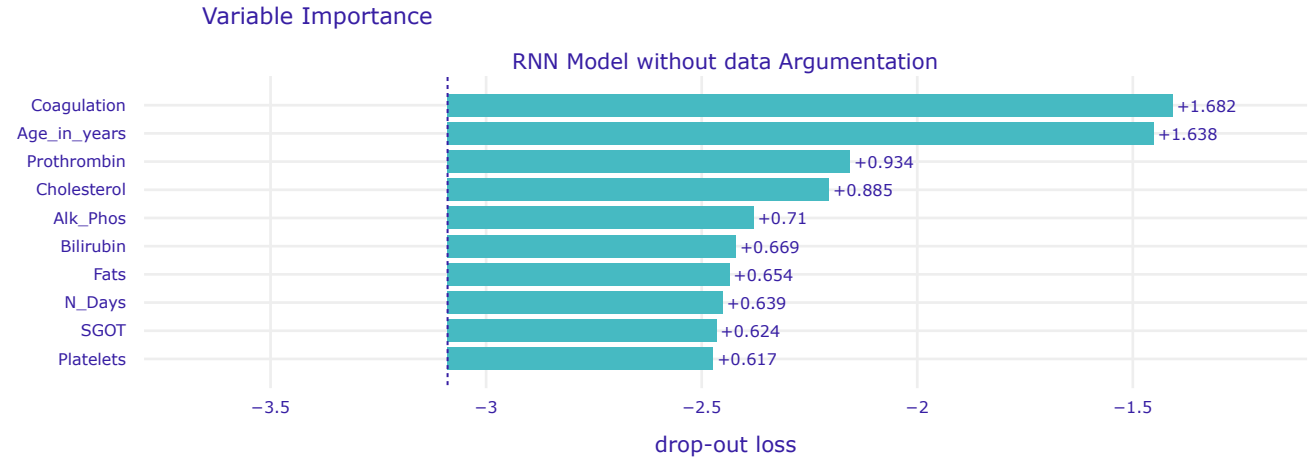





```
explainer = dx.Explainer(rnn_model_cirr, X_train, y_train, label="RNN Model without data Argumentation")
```

```
Preparation of a new explainer is initiated

  -> data              : 556 rows 27 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 556 values
  -> model_class       : keras.src.models.sequential.Sequential (default)
  -> label             : RNN Model without data Argumentation
  -> predict function  : <function yhat_tf_classification at 0x7a0610c9c430> will be used (default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 9.77e-07, mean = 0.345, max = 1.0
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.334, mean = 0.673, max = 2.0
  -> model_info        : package keras

A new explainer has been created!
```

```
fi = explainer.model_parts().plot()
```




```
num_instances = 1

# Loop through each instance
for i in range(num_instances):
    # Choose a specific instance
    instance = X_explain.iloc[9+i, :]

    # Explain the prediction for the instance
    pp = explainer.predict_parts(instance).plot()
```