Name- Debashish Saha **Roll no- MI22MTECH11001**

1. Developing simple classification algorithms using a simple 3-layer Neural network, two-layer CNN, and a two-layer LSTM on the following databases:
a. EHR – Pancreatic

CODE-

```
#importing libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv('EHR_Data.csv')
#print(df)


#Cleaning and Encoding the Data

# Binary encode the 'Sensitization type' column
df['Sensitization type'] = df['Sensitization type'].map({'Widespread sensitiza
tion': 1, 'Segmental sensitization': 0})

# Print the updated dataframe
print(df)

# replace missing values in 'Gender' column with 'Unknown'
df['Gender'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Gender'] = df['Gender'].str.strip().str.lower()

# one-hot encode the 'Gender' column
onehot = pd.get_dummies(df['Gender'], prefix='Gender')

# concatenate the one-hot encoded 'Gender' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Gender' column
df.drop('Gender', axis=1, inplace=True)

# print the updated dataset
#print(df)


# replace missing values in 'Etiology' column with 'Unknown'
df['Etiology'].fillna('Unknown', inplace=True)
```

```
# remove leading/trailing white space and convert to lowercase
df['Etiology'] = df['Etiology'].str.strip().str.lower()

# one-hot encode the 'Etiology' column
onehot = pd.get_dummies(df['Etiology'], prefix='Etiology')

# concatenate the one-hot encoded 'Etiology' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Etiology' column
df.drop('Etiology', axis=1, inplace=True)

# print the updated dataset
#print(df)


# replace missing values in 'Painless/painfull' column with 'Unknown'
df['Painless/painfull'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Painless/painfull'] = df['Painless/painfull'].str.strip().str.lower()

# one-hot encode the 'Painless/painfull' column
onehot = pd.get_dummies(df['Painless/painfull'], prefix='Painless/painfull')

# concatenate the one-
hot encoded 'Painless/painfull' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Painless/painfull' column
df.drop('Painless/painfull', axis=1, inplace=True)

# print the updated dataset
#print(df)

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['Duration betwn sx and Dx (months)'] = imputer.fit_transform(df[['Duration
betwn sx and Dx (months)']])

# print the updated dataset
#print(df)


# replace missing values in 'Diabetes' column with 'Unknown'
df['Diabetes'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Diabetes'] = df['Diabetes'].str.strip().str.lower()

# one-hot encode the 'Diabetes' column
onehot = pd.get_dummies(df['Diabetes'], prefix='Diabetes')
```

```
# concatenate the one-hot encoded 'Diabetes' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Diabetes' column
df.drop('Diabetes', axis=1, inplace=True)

# print the updated dataset
#print(df)

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['Duration betwn CP Dx and diabetes Dx (months)'] = imputer.fit_transform(df
[['Duration betwn CP Dx and diabetes Dx (months)']])

# print the updated dataset
#print(df)



# replace missing values in 'Overall_Interventions_yes_no' column with 'Unknow
n'
df['Overall_Interventions_yes_no'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Overall_Interventions_yes_no'] = df['Overall_Interventions_yes_no'].str.st
rip().str.lower()

# one-hot encode the 'Overall_Interventions_yes_no' column
onehot = pd.get_dummies(df['Overall_Interventions_yes_no'], prefix='Overall_In
terventions_yes_no')

# concatenate the one-
hot encoded 'Overall_Interventions_yes_no' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Overall_Interventions_yes_no' column
df.drop('Overall_Interventions_yes_no', axis=1, inplace=True)

# print the updated dataset
#print(df)



# replace missing values in 'Pancreatic_Atrophy' column with 'Unknown'
df['Pancreatic_Atrophy'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Pancreatic_Atrophy'] = df['Pancreatic_Atrophy'].str.strip().str.lower()

# one-hot encode the 'Pancreatic_Atrophy' column
onehot = pd.get_dummies(df['Pancreatic_Atrophy'], prefix='Pancreatic_Atrophy')
```

```
# concatenate the one-
hot encoded 'Pancreatic_Atrophy' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Pancreatic_Atrophy' column
df.drop('Pancreatic_Atrophy', axis=1, inplace=True)

# replace missing values in 'Dilatation_of_MPD' column with 'Unknown'
df['Dilatation_of_MPD'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Dilatation_of_MPD'] = df['Dilatation_of_MPD'].str.strip().str.lower()

# one-hot encode the 'Dilatation_of_MPD' column
onehot = pd.get_dummies(df['Dilatation_of_MPD'], prefix='Dilatation_of_MPD')

# concatenate the one-
hot encoded 'Dilatation_of_MPD' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Dilatation_of_MPD' column
df.drop('Dilatation_of_MPD', axis=1, inplace=True)


# replace missing values in 'MPD_stricture' column with 'Unknown'
df['MPD_stricture'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['MPD_stricture'] = df['MPD_stricture'].str.strip().str.lower()

# one-hot encode the 'MPD_stricture' column
onehot = pd.get_dummies(df['MPD_stricture'], prefix='MPD_stricture')

# concatenate the one-
hot encoded 'MPD_stricture' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'MPD_stricture' column
df.drop('MPD_stricture', axis=1, inplace=True)

# replace missing values in 'MPD_calculus' column with 'Unknown'
df['MPD_calculus'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['MPD_calculus'] = df['MPD_calculus'].str.strip().str.lower()

# one-hot encode the 'MPD_calculus' column
onehot = pd.get_dummies(df['MPD_calculus'], prefix='MPD_calculus')

# concatenate the one-
hot encoded 'MPD_calculus' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'MPD_calculus' column
```

```
df.drop('MPD_calculus', axis=1, inplace=True)

# replace missing values in 'Parenchymal_calcification_' column with 'Unknown'
df['Parenchymal_calcification_'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Parenchymal_calcification_'] = df['Parenchymal_calcification_'].str.strip(
).str.lower()

# one-hot encode the 'Parenchymal_calcification_' column
onehot = pd.get_dummies(df['Parenchymal_calcification_'], prefix='Parenchymal_
calcification_')

# concatenate the one-
hot encoded 'Parenchymal_calcification_' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Parenchymal_calcification_' column
df.drop('Parenchymal_calcification_', axis=1, inplace=True)

# replace missing values in 'Radiation_of_pain' column with 'Unknown'
df['Radiation_of_pain'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Radiation_of_pain'] = df['Radiation_of_pain'].str.strip().str.lower()

# one-hot encode the 'Radiation_of_pain' column
onehot = pd.get_dummies(df['Radiation_of_pain'], prefix='Radiation_of_pain')

# concatenate the one-
hot encoded 'Radiation_of_pain' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Radiation_of_pain' column
df.drop('Radiation_of_pain', axis=1, inplace=True)

# replace missing values in 'Development_of_new_areas_of_pain' column with 'Un
known'
df['Development_of_new_areas_of_pain'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Development_of_new_areas_of_pain'] = df['Development_of_new_areas_of_pain'
].str.strip().str.lower()

# one-hot encode the 'Development_of_new_areas_of_pain' column
onehot = pd.get_dummies(df['Development_of_new_areas_of_pain'], prefix='Develo
pment_of_new_areas_of_pain')

# concatenate the one-
hot encoded 'Development_of_new_areas_of_pain' columns with the original datas
et
df = pd.concat([df, onehot], axis=1)

# drop the original 'Development_of_new_areas_of_pain' column
```

```
df.drop('Development_of_new_areas_of_pain', axis=1, inplace=True)


# replace missing values in 'Overall_Pain_Pattern' column with 'Unknown'
df['Overall_Pain_Pattern'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Overall_Pain_Pattern'] = df['Overall_Pain_Pattern'].str.strip().str.lower(
)

# one-hot encode the 'Overall_Pain_Pattern' column
onehot = pd.get_dummies(df['Overall_Pain_Pattern'], prefix='Overall_Pain_Patte
rn')

# concatenate the one-
hot encoded 'Overall_Pain_Pattern' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Overall_Pain_Pattern' column
df.drop('Overall_Pain_Pattern', axis=1, inplace=True)


# replace missing values in 'Pain continuous intermitent' column with 'Unknown
'
df['Pain continuous intermitent'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Pain continuous intermitent'] = df['Pain continuous intermitent'].str.stri
p().str.lower()

# one-hot encode the 'Pain continuous intermitent' column
onehot = pd.get_dummies(df['Pain continuous intermitent'], prefix='Pain contin
uous intermitent')

# concatenate the one-
hot encoded 'Pain continuous intermitent' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Pain continuous intermitent' column
df.drop('Pain continuous intermitent', axis=1, inplace=True)



# print the updated dataset
#print(df)


# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['Rumination'] = imputer.fit_transform(df[['Rumination']])

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')
```

```
# fill in the missing values in the column with mean value
df['Magnification'] = imputer.fit_transform(df[['Magnification']])

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['Helplessness'] = imputer.fit_transform(df[['Helplessness']])

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['IZBICKI_Final_Score'] = imputer.fit_transform(df[['IZBICKI_Final_Score']])

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['HADS_Depression_Score_'] = imputer.fit_transform(df[['HADS_Depression_Scor
e_']])

# replace missing values in 'HADS_depression_category' column with 'Unknown'
df['HADS_depression_category'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['HADS_depression_category'] = df['HADS_depression_category'].str.strip().st
r.lower()

# one-hot encode the 'HADS_depression_category' column
onehot = pd.get_dummies(df['HADS_depression_category'], prefix='HADS_depressio
n_category')

# concatenate the one-
hot encoded 'HADS_depression_category' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'HADS_depression_category' column
df.drop('HADS_depression_category', axis=1, inplace=True)

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['HADS_Anxiety_score'] = imputer.fit_transform(df[['HADS_Anxiety_score']])


# replace missing values in 'HADS_anxiety_category' column with 'Unknown'
df['HADS_anxiety_category'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['HADS_anxiety_category'] = df['HADS_anxiety_category'].str.strip().str.lowe
r()
```

```python
# one-hot encode the 'HADS_anxiety_category' column
onehot = pd.get_dummies(df['HADS_anxiety_category'], prefix='HADS_anxiety_category')

# concatenate the one-hot encoded 'HADS_anxiety_category' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'HADS_anxiety_category' column
df.drop('HADS_anxiety_category', axis=1, inplace=True)


# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['Pain_DETECT_final_score'] = imputer.fit_transform(df[['Pain_DETECT_final_score']])


# replace missing values in 'PainDETECT_typeofpain' column with 'Unknown'
df['PainDETECT_typeofpain'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['PainDETECT_typeofpain'] = df['PainDETECT_typeofpain'].str.strip().str.lower()

# one-hot encode the 'PainDETECT_typeofpain' column
onehot = pd.get_dummies(df['PainDETECT_typeofpain'], prefix='PainDETECT_typeofpain')

# concatenate the one-hot encoded 'PainDETECT_typeofpain' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'PainDETECT_typeofpain' column
df.drop('PainDETECT_typeofpain', axis=1, inplace=True)


# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['BDI_II_score'] = imputer.fit_transform(df[['BDI_II_score']])

# replace missing values in 'BDI_II_Category' column with 'Unknown'
df['BDI_II_Category'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['BDI_II_Category'] = df['BDI_II_Category'].str.strip().str.lower()

# one-hot encode the 'BDI_II_Category' column
onehot = pd.get_dummies(df['BDI_II_Category'], prefix='BDI_II_Category')
```

```
# concatenate the one-
hot encoded 'BDI_II_Category' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'BDI_II_Category' column
df.drop('BDI_II_Category', axis=1, inplace=True)

# create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# fill in the missing values in the column with mean value
df['EORTC_Global_Score'] = imputer.fit_transform(df[['EORTC_Global_Score']])

# fill in the missing values in the column with mean value
df['EORTC_Fatigue'] = imputer.fit_transform(df[['EORTC_Fatigue']])

# fill in the missing values in the column with mean value
df['EORTC_Nausea_Vomiting'] = imputer.fit_transform(df[['EORTC_Nausea_Vomiting
']])

# fill in the missing values in the column with mean value
df['EORTC_Pain'] = imputer.fit_transform(df[['EORTC_Pain']])

# fill in the missing values in the column with mean value
df['EORTC_Dyspnoea'] = imputer.fit_transform(df[['EORTC_Dyspnoea']])

# fill in the missing values in the column with mean value
df['EORTC_Insomnia'] = imputer.fit_transform(df[['EORTC_Insomnia']])

# fill in the missing values in the column with mean value
df['EORTC_Appetite_loss'] = imputer.fit_transform(df[['EORTC_Appetite_loss']])

# fill in the missing values in the column with mean value
df['EORTC_Constipation'] = imputer.fit_transform(df[['EORTC_Constipation']])

# fill in the missing values in the column with mean value
df['EORTC_Diarrhoea'] = imputer.fit_transform(df[['EORTC_Diarrhoea']])

# fill in the missing values in the column with mean value
df['EORTC_Financial_difficulties'] = imputer.fit_transform(df[['EORTC_Financia
l_difficulties']])

# fill in the missing values in the column with mean value
df['EORTC_Physical'] = imputer.fit_transform(df[['EORTC_Physical']])

# fill in the missing values in the column with mean value
df['EORTC_Role'] = imputer.fit_transform(df[['EORTC_Role']])

# fill in the missing values in the column with mean value
df['EORTC_Emotional'] = imputer.fit_transform(df[['EORTC_Emotional']])

# fill in the missing values in the column with mean value
df['EORTC_Cognitive'] = imputer.fit_transform(df[['EORTC_Cognitive']])
```

```
# fill in the missing values in the column with mean value
df['EORTC_Social'] = imputer.fit_transform(df[['EORTC_Social']])

# fill in the missing values in the column with mean value
df['PAN28_pancreatic_pain'] = imputer.fit_transform(df[['PAN28_pancreatic_pain
']])

# fill in the missing values in the column with mean value
df['PAN28_digestive_sym'] = imputer.fit_transform(df[['PAN28_digestive_sym']])

# fill in the missing values in the column with mean value
df['PAN28_altered_bowel'] = imputer.fit_transform(df[['PAN28_altered_bowel']])

# fill in the missing values in the column with mean value
df['PAN28_Hepatic'] = imputer.fit_transform(df[['PAN28_Hepatic']])

# fill in the missing values in the column with mean value
df['PAN28_Bodyimage'] = imputer.fit_transform(df[['PAN28_Bodyimage']])

# fill in the missing values in the column with mean value
df['PAN28_Activity'] = imputer.fit_transform(df[['PAN28_Activity']])

# fill in the missing values in the column with mean value
df['PAN28_Satisfaction_with_healthcare'] = imputer.fit_transform(df[['PAN28_Sa
tisfaction_with_healthcare']])

# fill in the missing values in the column with mean value
df['PAN28_Sexuality'] = imputer.fit_transform(df[['PAN28_Sexuality']])


# replace missing values in 'P-QST_Yes_No' column with 'Unknown'
df['P-QST_Yes_No'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['P-QST_Yes_No'] = df['P-QST_Yes_No'].str.strip().str.lower()

# one-hot encode the 'P-QST_Yes_No' column
onehot = pd.get_dummies(df['P-QST_Yes_No'], prefix='P-QST_Yes_No')

# concatenate the one-hot encoded 'P-
QST_Yes_No' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'P-QST_Yes_No' column
df.drop('P-QST_Yes_No', axis=1, inplace=True)


# replace missing values in 'Sensitization_yes_no' column with 'Unknown'
df['Sensitization_yes_no'].fillna('Unknown', inplace=True)

# remove leading/trailing white space and convert to lowercase
df['Sensitization_yes_no'] = df['Sensitization_yes_no'].str.strip().str.lower(
)

# one-hot encode the 'Sensitization_yes_no' column
```

```
onehot = pd.get_dummies(df['Sensitization_yes_no'], prefix='Sensitization_yes_
no')

# concatenate the one-
hot encoded 'Sensitization_yes_no' columns with the original dataset
df = pd.concat([df, onehot], axis=1)

# drop the original 'Sensitization_yes_no' column
df.drop('Sensitization_yes_no', axis=1, inplace=True)

# print the updated dataset
print(df)
```

Output-

```
     Sensitization type  Duration betwn sx and Dx (months)  \
0                     0                          88.000000
1                     0                           0.000000
2                     0                           0.000000
3                     0                           9.000000
4                     0                          10.000000
..                  ...                                ...
369                   1                          29.004587
370                   1                          29.004587
371                   1                          29.004587
372                   1                          29.004587
373                   1                          29.004587

     Duration betwn CP Dx and diabetes Dx (months)  Rumination  Magnification
\
0                                        16.000000         0.0       0.000000
1                                         0.000000         2.0       0.000000
2                                        39.902439         1.0       0.000000
3                                         0.000000         3.0       1.000000
4                                        39.902439         0.0       0.000000
..                                             ...         ...            ...
369                                       6.000000         0.0       3.000000
370                                       6.000000        11.0       3.867209
371                                       6.000000        13.0       0.000000
372                                      46.000000         8.0       6.000000
373                                       5.000000        13.0       5.000000

     Helplessness  IZBICKI_Final_Score  HADS_Depression_Score_  \
0             0.0                 60.0                     0.0
1             0.0                 35.0                     0.0
2             0.0                 35.0                     0.0
3             0.0                 47.5                     2.0
4             0.0                 31.5                     0.0
..            ...                  ...                     ...
369           8.0                 50.0                     5.0
370          10.0                 80.0                     9.0
371           0.0                 80.0                     9.0
372           1.0                 80.0                     9.0
373          11.0                 80.0                     9.0

     HADS_Anxiety_score  Pain_DETECT_final_score  ...  \
0                   0.0                     13.0  ...
1                   0.0                      0.0  ...
2                   0.0                      4.0  ...
3                   6.0                      5.0  ...
4                   0.0                      2.0  ...
..                  ...                      ...  ...
```

```
369              13.0                         9.0  ...
370              13.0                         7.0  ...
371              13.0                         7.0  ...
372              13.0                         7.0  ...
373              13.0                         7.0  ...

     BDI_II_Category_mild depression  BDI_II_Category_mild mood disturbance  \
0                                  0                                      0
1                                  0                                      0
2                                  0                                      0
3                                  0                                      0
4                                  0                                      0
..                               ...                                    ...
369                                0                                      0
370                                1                                      0
371                                1                                      0
372                                1                                      0
373                                1                                      0

     BDI_II_Category_moderate depression  BDI_II_Category_normal  \
0                                      0                       1
1                                      0                       1
2                                      0                       1
3                                      0                       1
4                                      0                       1
..                                   ...                     ...
369                                    0                       0
370                                    0                       0
371                                    0                       0
372                                    0                       0
373                                    0                       0

     BDI_II_Category_severe depression  BDI_II_Category_unknown  \
0                                    0                        0
1                                    0                        0
2                                    0                        0
3                                    0                        0
4                                    0                        0
..                                 ...                      ...
369                                  0                        0
370                                  0                        0
371                                  0                        0
372                                  0                        0
373                                  0                        0

     P-QST_Yes_No_no  P-QST_Yes_No_unknown  P-QST_Yes_No_yes  \
0                  0                     1                 0
1                  0                     1                 0
2                  0                     1                 0
3                  0                     1                 0
4                  0                     1                 0
..               ...                   ...               ...
369                0                     0                 1
370                0                     0                 1
371                0                     0                 1
372                0                     0                 1
373                0                     0                 1

     Sensitization_yes_no_yes
0                           1
1                           1
2                           1
3                           1
4                           1
..                        ...
```

```
369                            1
370                            1
371                            1
372                            1
373                            1

[374 rows x 114 columns]
```

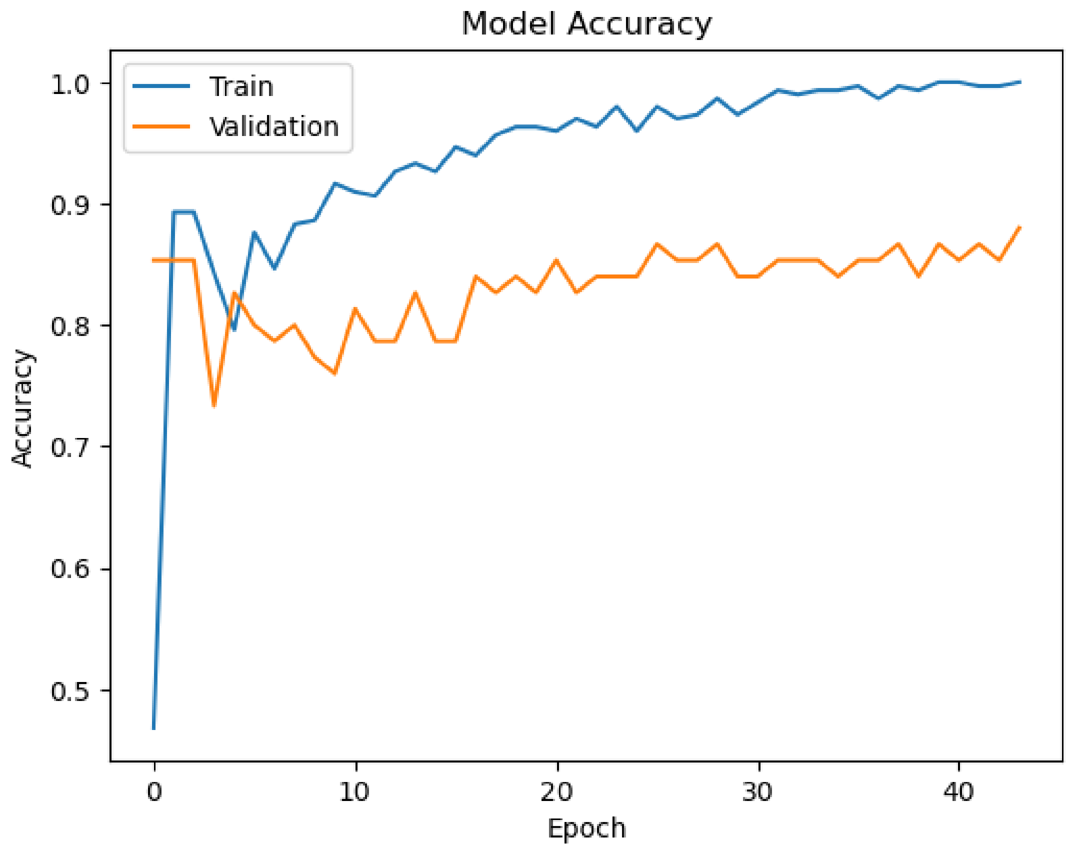
3-layer Neural network-

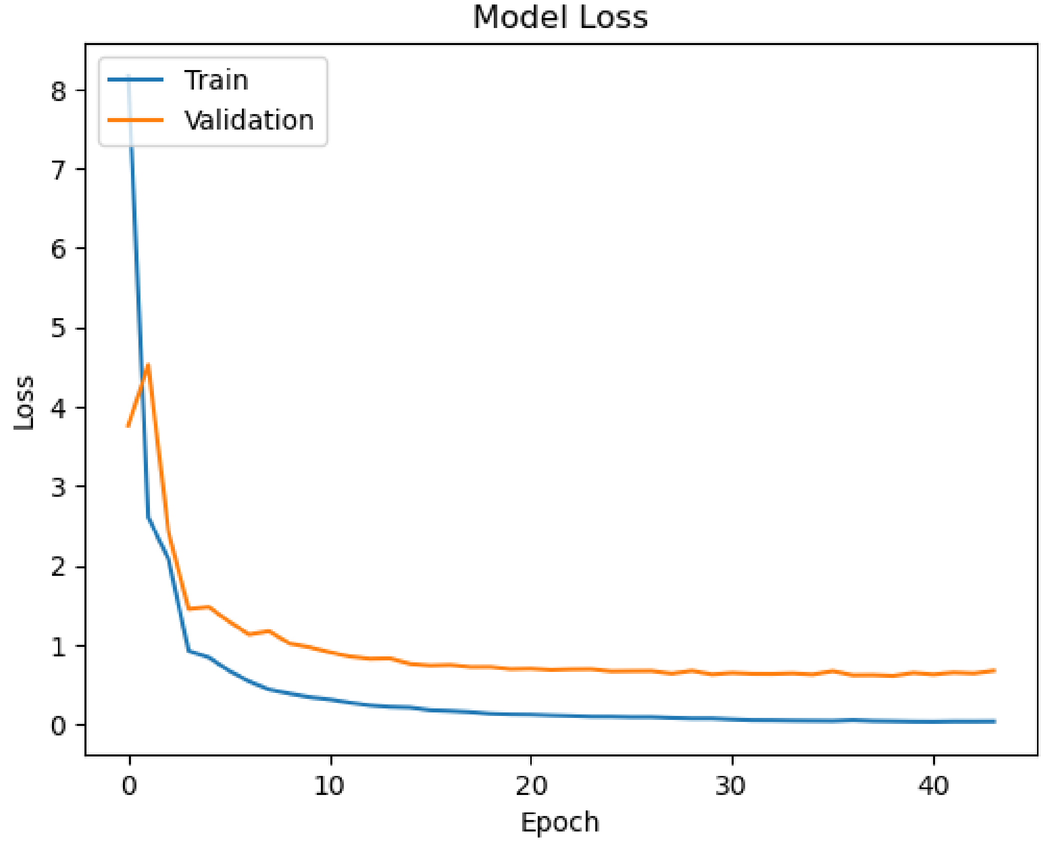
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# 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, rando
m_state=42)

# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)


# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_da
ta=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

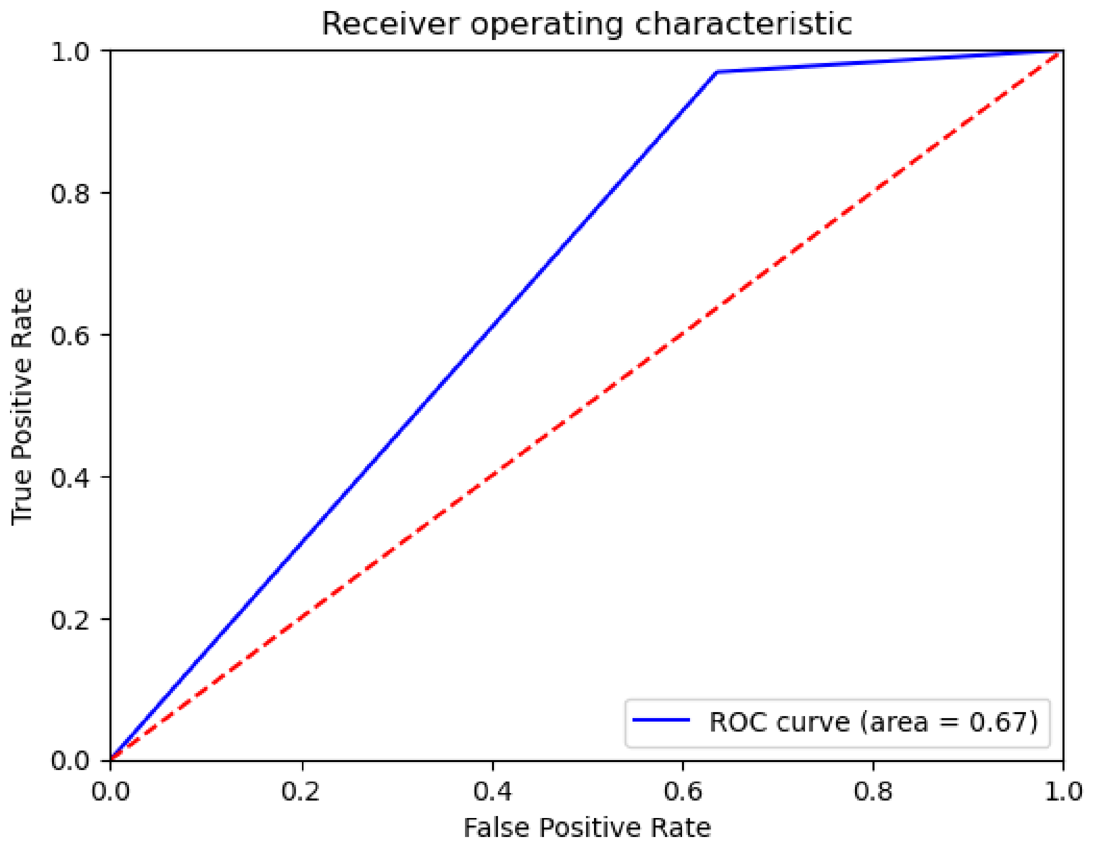
Output-

```
Epoch 44/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0331 - accuracy:
1.0000 - val_loss: 0.6717 - val_accuracy: 0.8800
3/3 [==============================] - 0s 4ms/step - loss: 0.6717 - accuracy:
0.8800
Test loss: 0.6716741919517517
Test accuracy: 0.8799999952316284
```






```
3/3 [==============================] - 0s 3ms/step
Confusion matrix:
 [[ 4  7]
 [ 2 62]]
Precision: 0.8985507246376812
Recall: 0.96875
F1-score: 0.9323308270676692
ROC AUC: 0.6661931818181819
```



Two-layer CNN -

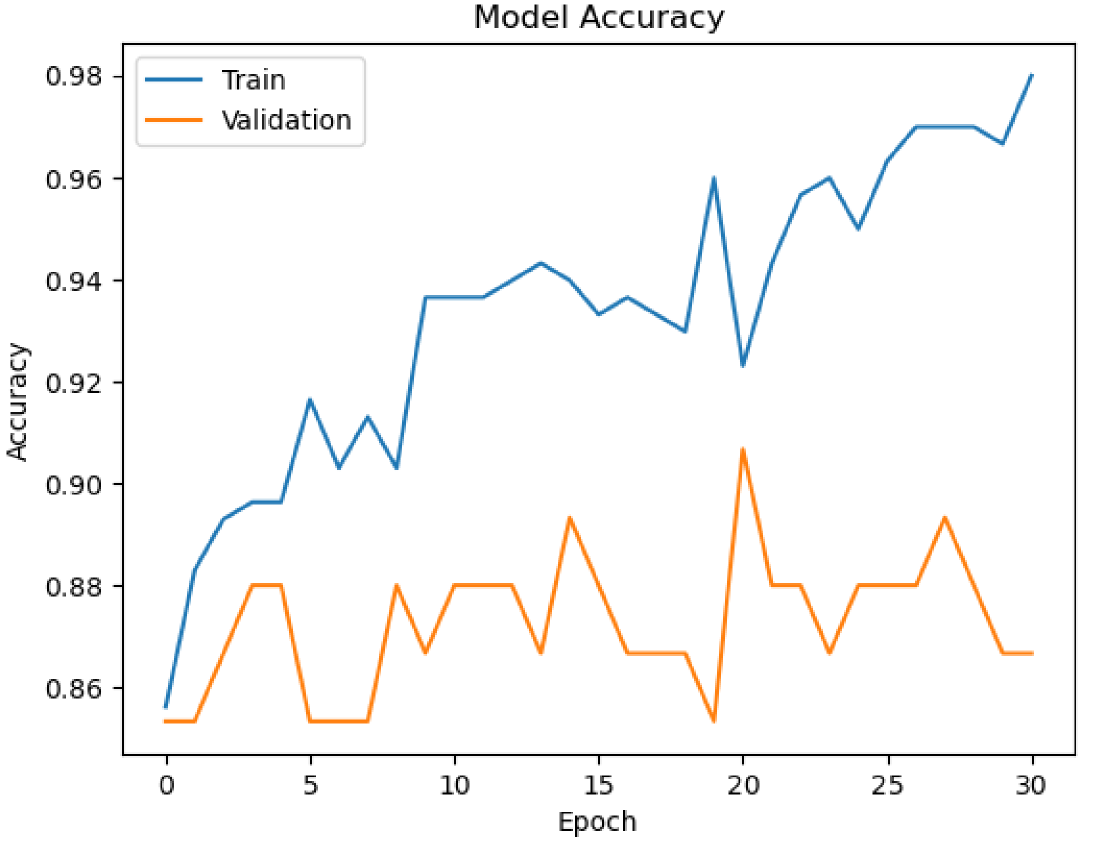
```
#CNN
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten, MaxPooling1D
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# 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, rando
m_state=42)

#Reshape input data for CNN
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

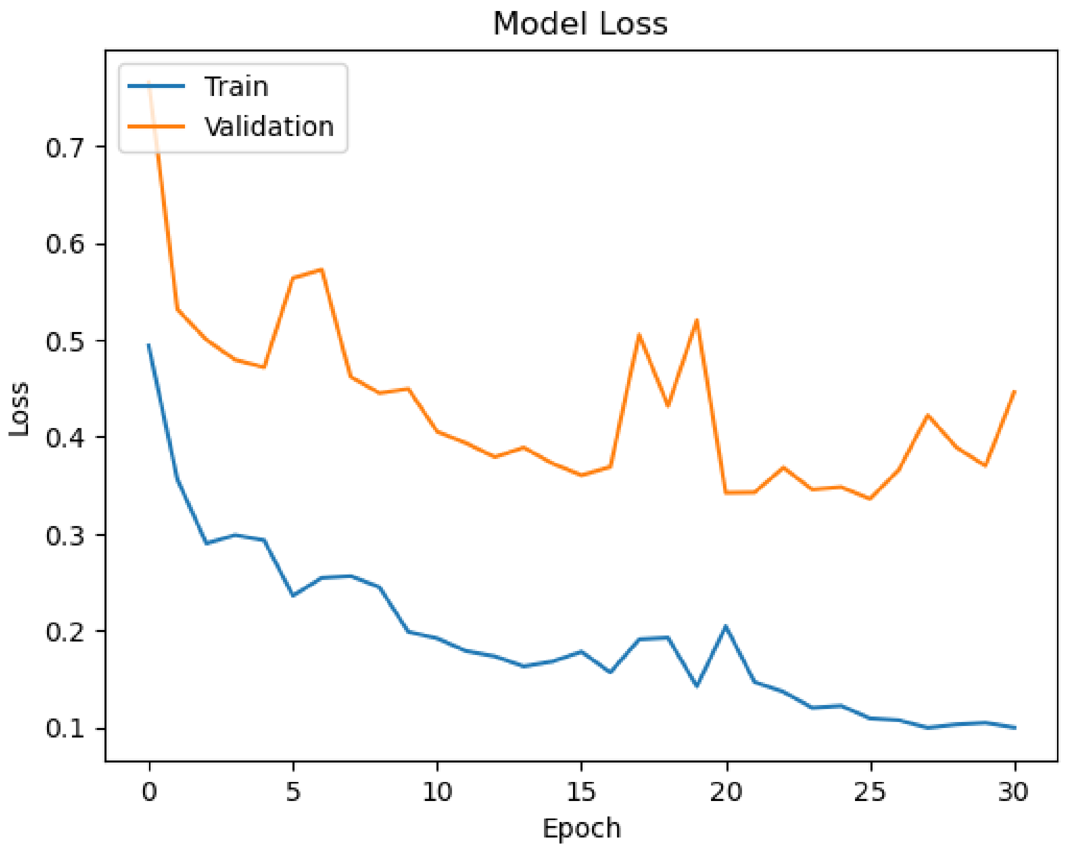
```
# Define the model architecture
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=X_t
rain.shape[1:]))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)


# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_da
ta=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

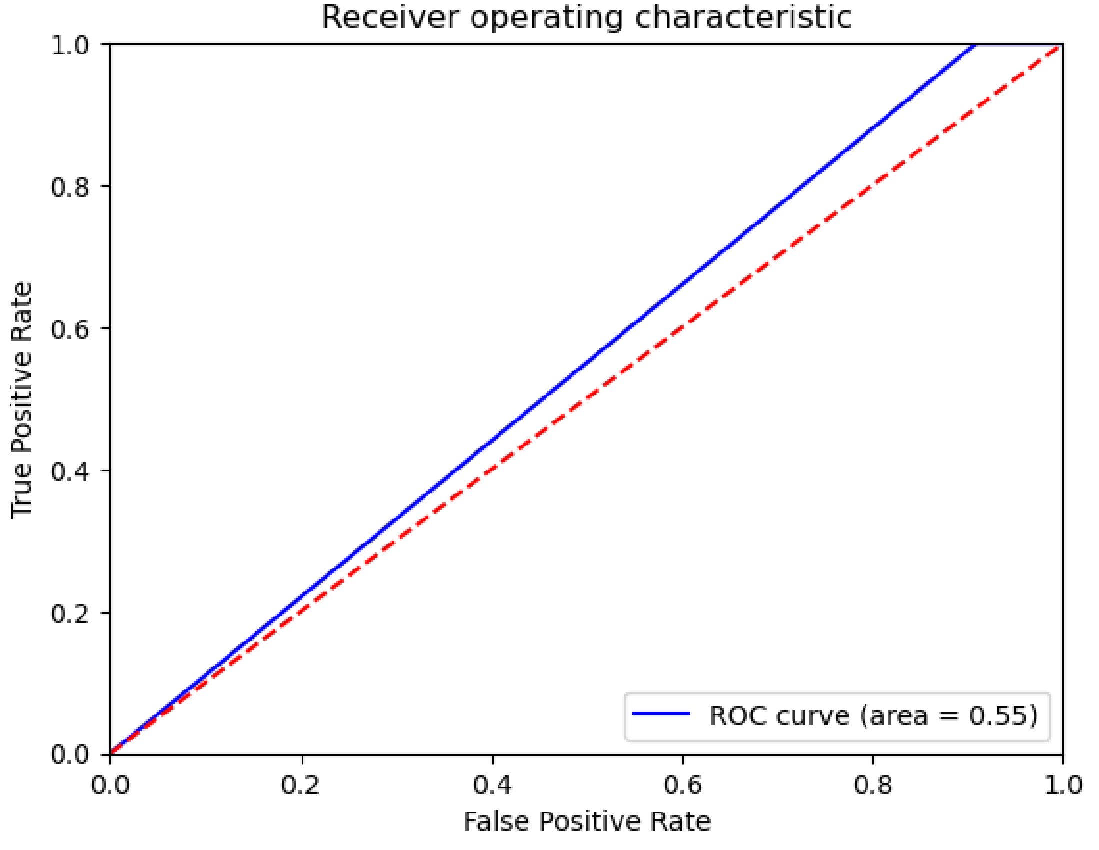
Output-

```
Epoch 31/100
10/10 [==============================] - 0s 18ms/step - loss: 0.0999 - accuracy:
0.9799 - val_loss: 0.4460 - val_accuracy: 0.8667
3/3 [==============================] - 0s 6ms/step - loss: 0.4460 - accuracy:
0.8667
Test loss: 0.44604364037513733
Test accuracy: 0.8666666746139526
```






```
3/3 [==============================] - 0s 5ms/step
Confusion matrix:
 [[ 1 10]
 [ 0 64]]
Precision: 0.8648648648648649
Recall: 1.0
F1-score: 0.927536231884058
ROC AUC: 0.5454545454545454
```



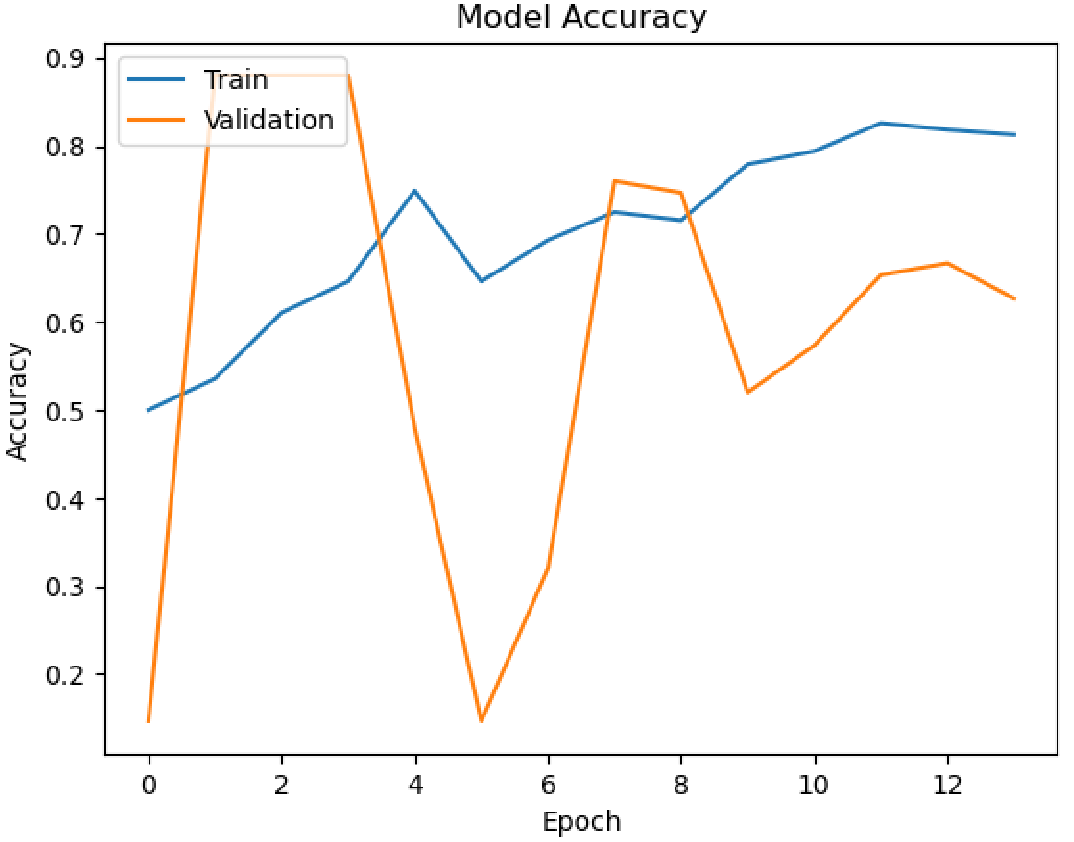
A two-layer LSTM -

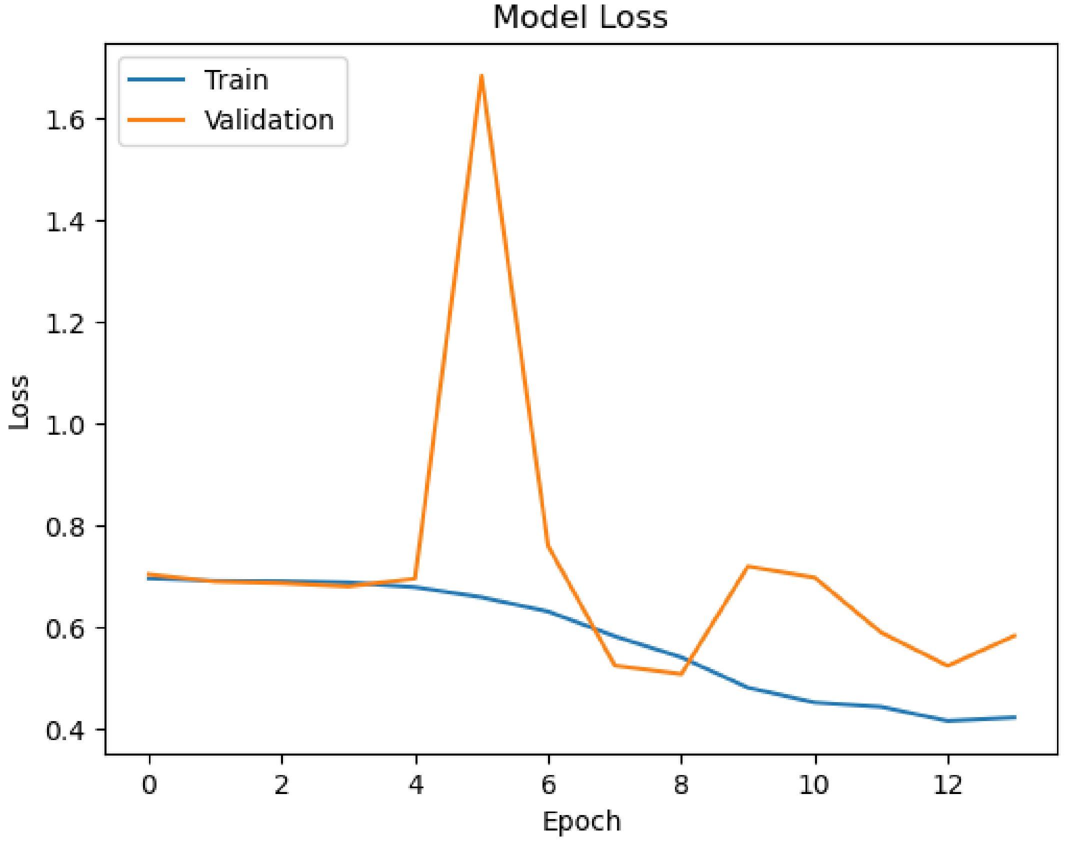
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# 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, rando
m_state=42)

# Resample the training data to address class imbalance
X_train_resampled, y_train_resampled = resample(X_train[y_train==0], y_train[y
_train==0],
                                                replace=True, n_samples=X_trai
n[y_train==1].shape[0],
                                                random_state=42)
X_train_resampled = np.concatenate([X_train_resampled, X_train[y_train==1]], a
xis=0)
y_train_resampled = np.concatenate([y_train_resampled, y_train[y_train==1]], a
xis=0)

# Define the model architecture
model = Sequential()
model.add(LSTM(units=64, return_sequences=True, input_shape=(X_train.shape[1],
 1)))
model.add(LSTM(units=32))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Train the model
history = model.fit(X_train_resampled.reshape(-
1, X_train.shape[1], 1), y_train_resampled, epochs=100, batch_size=32, validat
ion_data=(X_test.reshape(-
1, X_train.shape[1], 1), y_test), callbacks=[early_stopping])
```

```python
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test.reshape(-
1, X_train.shape[1], 1), y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test.reshape(-1, X_train.shape[1], 1))
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

Output-

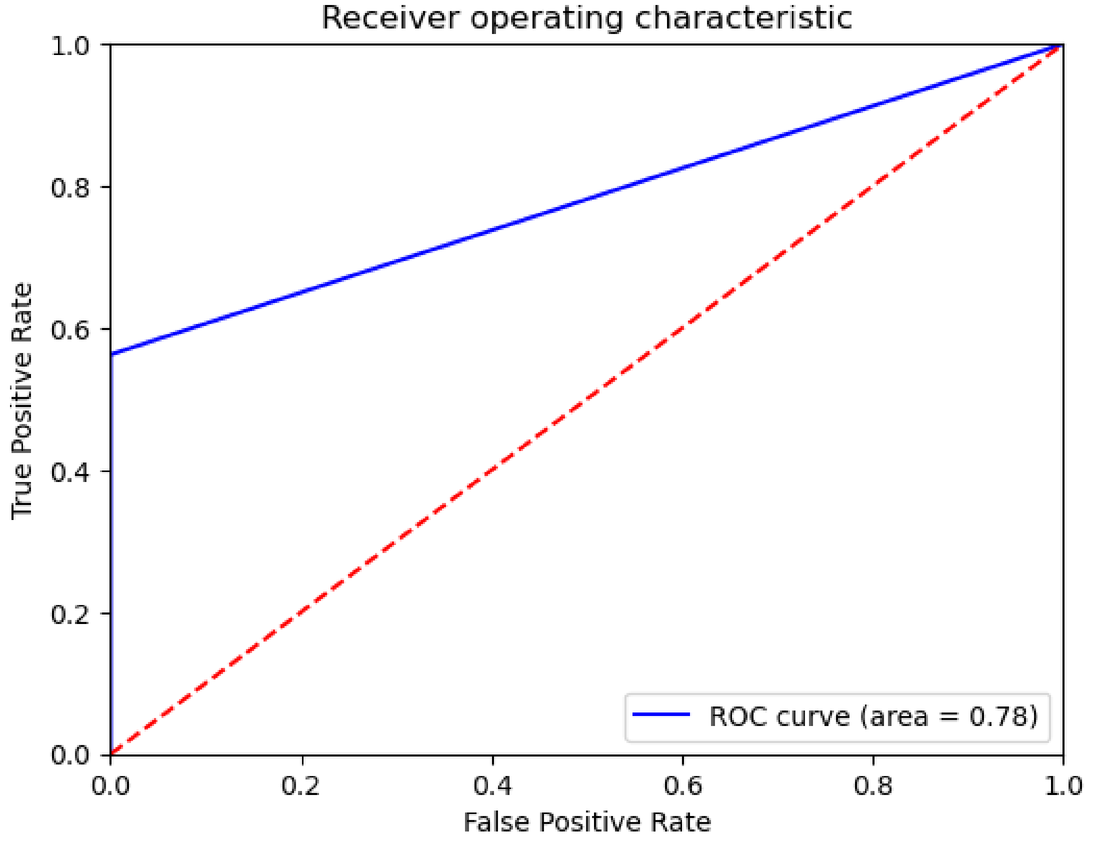
```
Epoch 14/100
17/17 [==============================] - 2s 103ms/step - loss: 0.4225 - accuracy:
0.8127 - val_loss: 0.5832 - val_accuracy: 0.6267
3/3 [==============================] - 0s 26ms/step - loss: 0.5832 - accuracy:
0.6267
Test loss: 0.5832009315490723
Test accuracy: 0.6266666650772095
```






```
3/3 [==============================] - 1s 18ms/step
```

```
Confusion matrix:
 [[11  0]
 [28 36]]
Precision: 1.0
Recall: 0.5625
F1-score: 0.72
ROC AUC: 0.78125
```



2. Data is imbalanced. Use the strategy to overcome the challenge of imbalance (a) undersampling, (b) oversampling

(a) Undersampling applied to simple 3-layer Neural network-

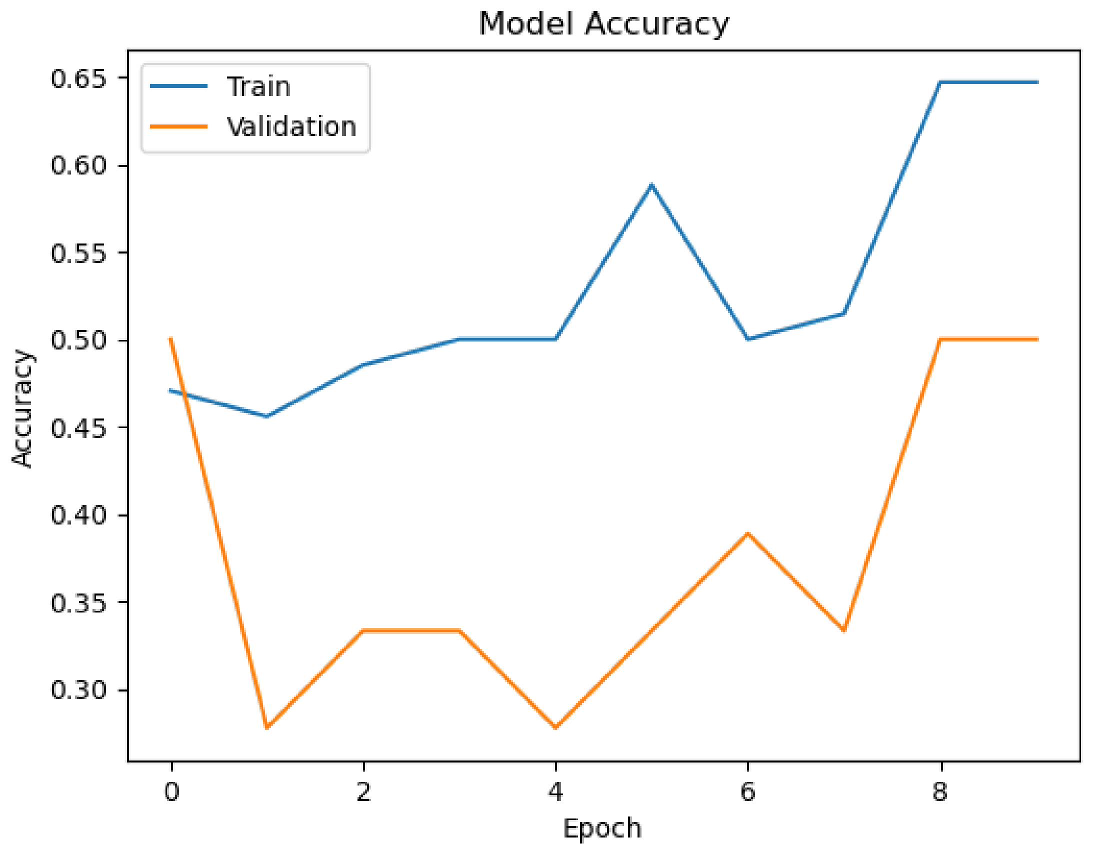
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values
```

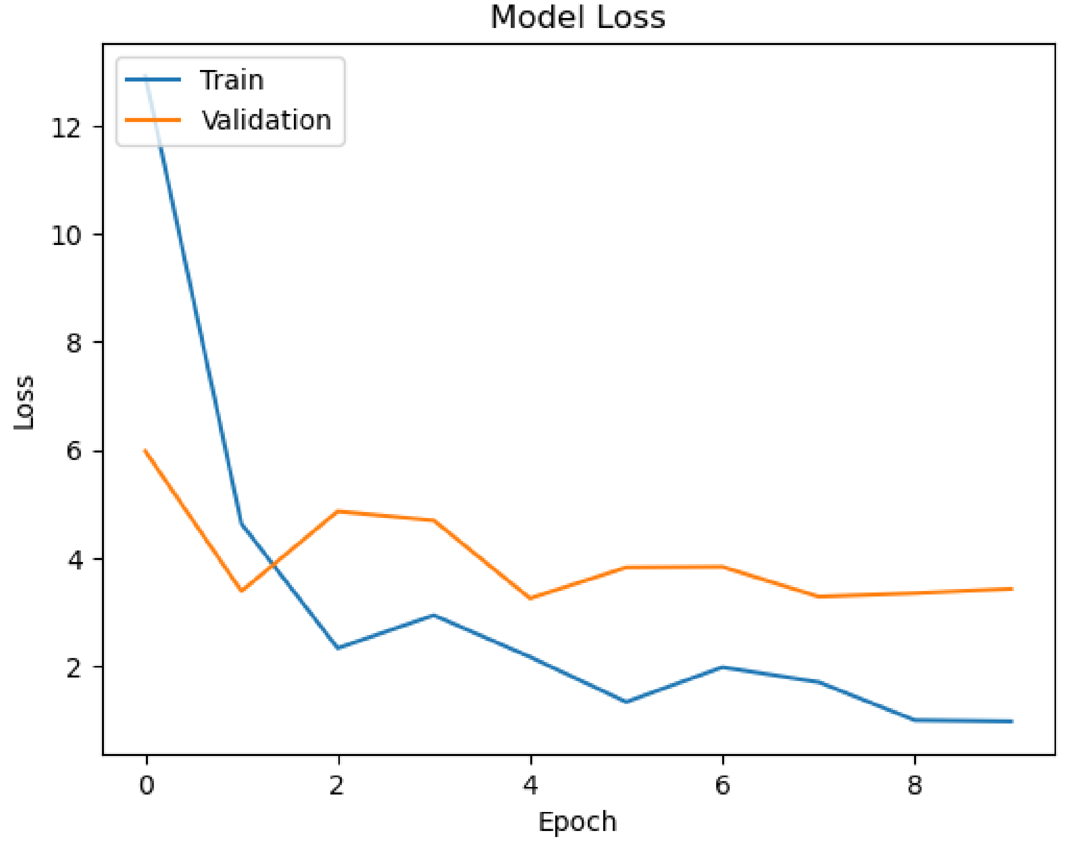
```python
# Undersample the majority class
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)



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



# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)


# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_da
ta=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

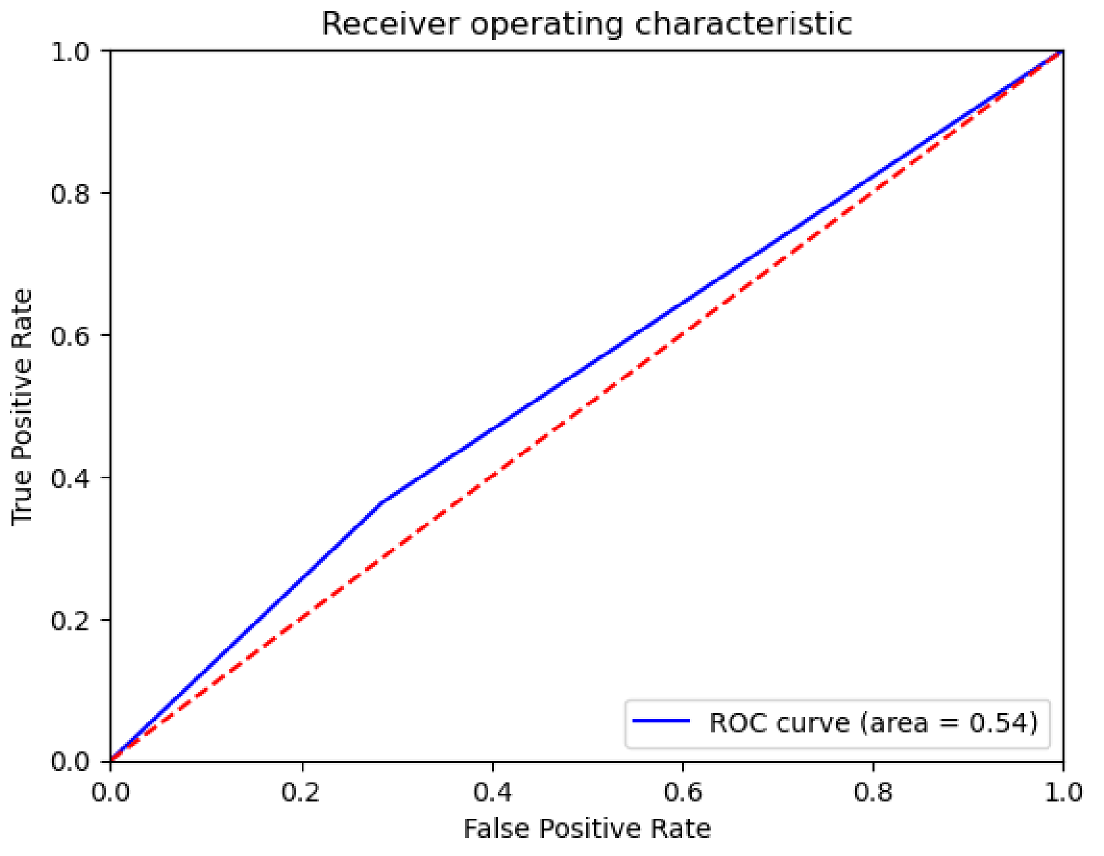
Output-

```
Epoch 10/100
3/3 [==============================] - 0s 31ms/step - loss: 0.9694 - accuracy:
0.6471 - val_loss: 3.4241 - val_accuracy: 0.5000
1/1 [==============================] - 0s 44ms/step - loss: 3.4241 - accuracy:
0.5000
Test loss: 3.4240646362304688
Test accuracy: 0.5
```






```
1/1 [==============================] - 0s 85ms/step
Confusion matrix:
 [[5 2]
 [7 4]]
Precision: 0.6666666666666666
Recall: 0.36363636363636365
F1-score: 0.47058823529411777
ROC AUC: 0.538961038961039
```



(b) Oversampling applied to simple 3-layer Neural network-

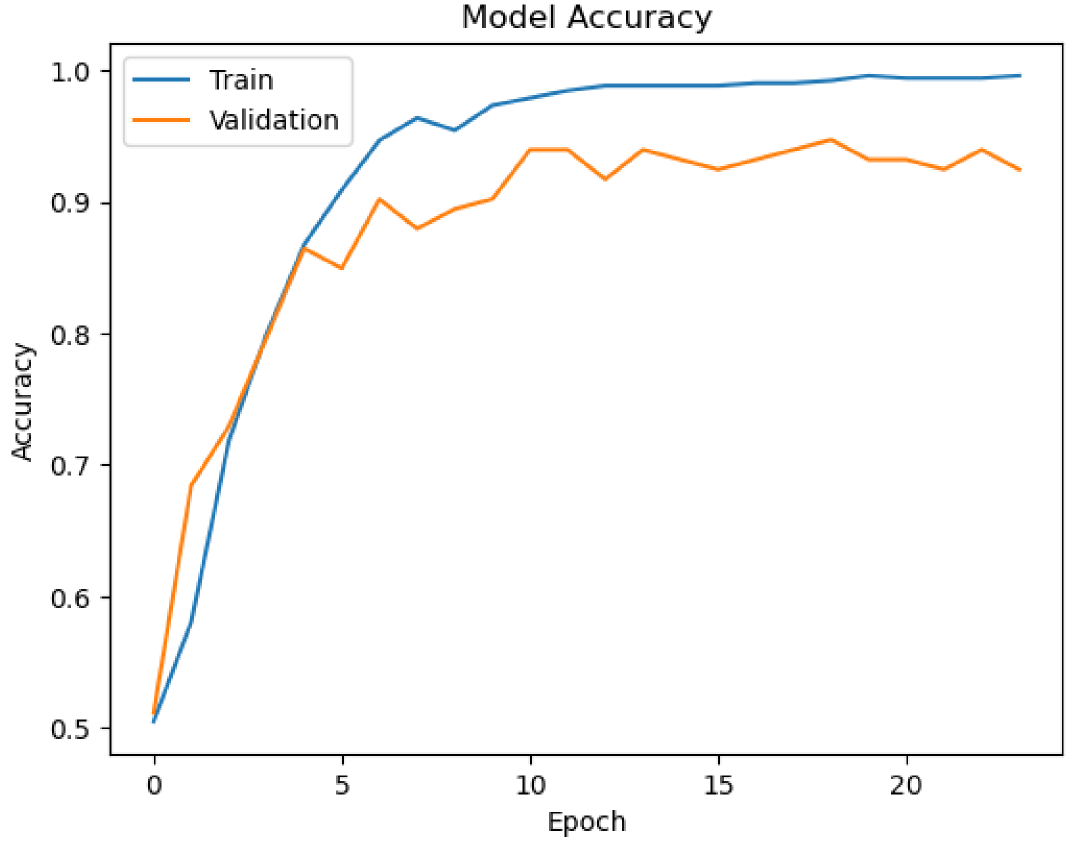
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values



# Oversample the minority class
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)



# Split the dataset into training and testing sets
```

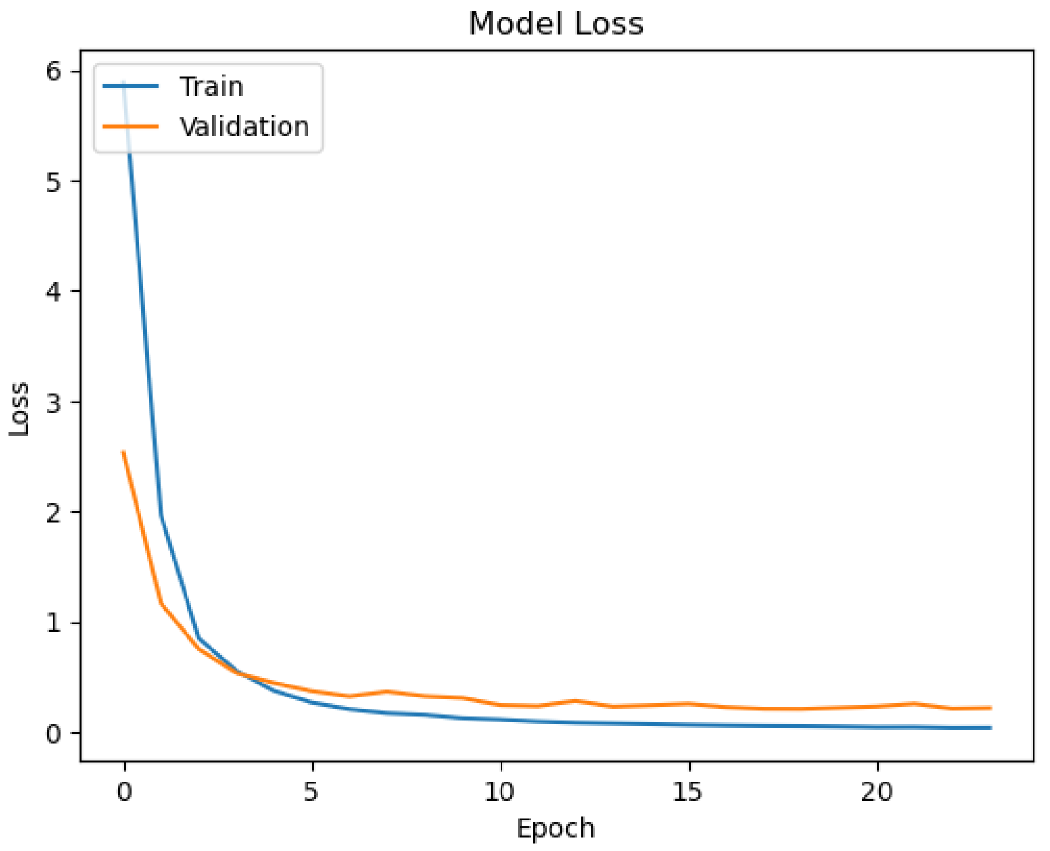
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
test_size=0.2, random_state=42)



# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)


# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_da
ta=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

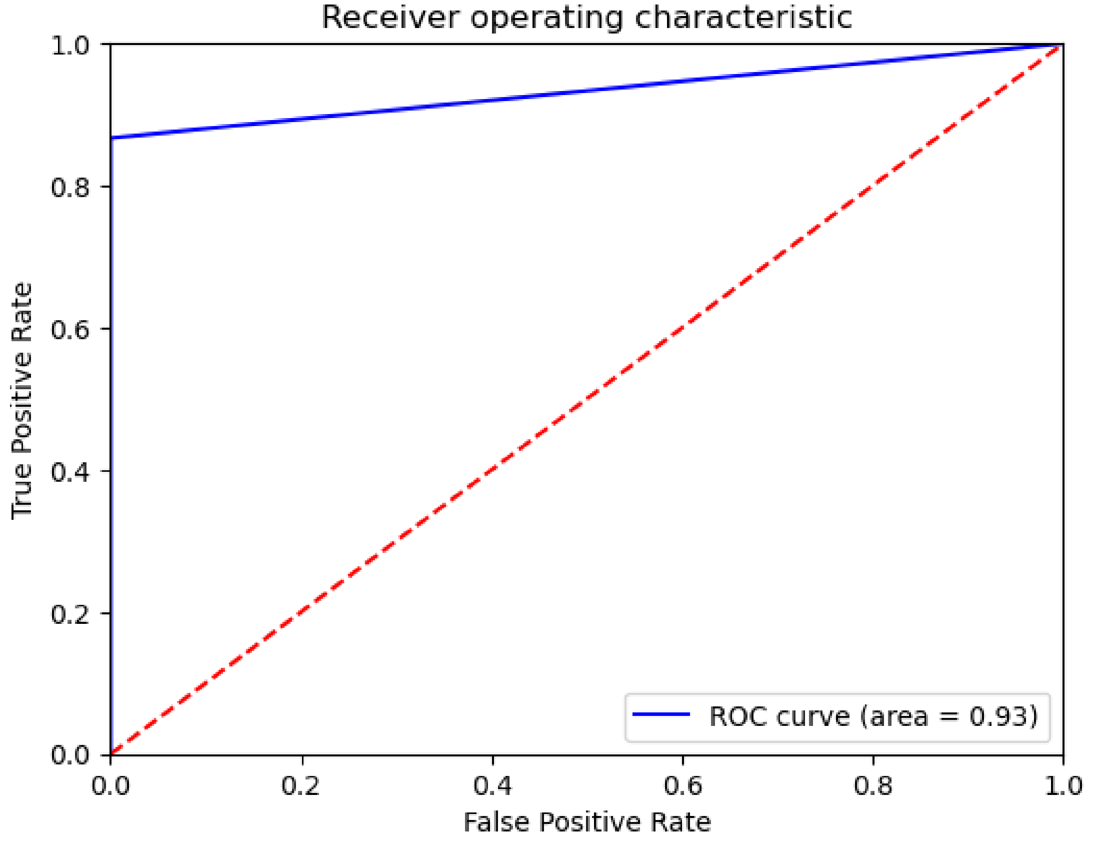
Output-

```
Epoch 24/100
17/17 [==============================] - 0s 7ms/step - loss: 0.0399 - accuracy:
0.9962 - val_loss: 0.2188 - val_accuracy: 0.9248
5/5 [==============================] - 0s 4ms/step - loss: 0.2188 - accuracy:
0.9248
Test loss: 0.21878783404827118
Test accuracy: 0.9248120188713074
```






```
5/5 [==============================] - 0s 4ms/step
Confusion matrix:
 [[58  0]
 [10 65]]
Precision: 1.0
Recall: 0.8666666666666667
F1-score: 0.9285714285714286
ROC AUC: 0.9333333333333333
```



3.Trying (a) k-fold cross-validation, (b) leave one subject out cross-validation and (c) k-stratified cross-validation.

(a) k-fold cross-validation to simple 3-layer Neural network-

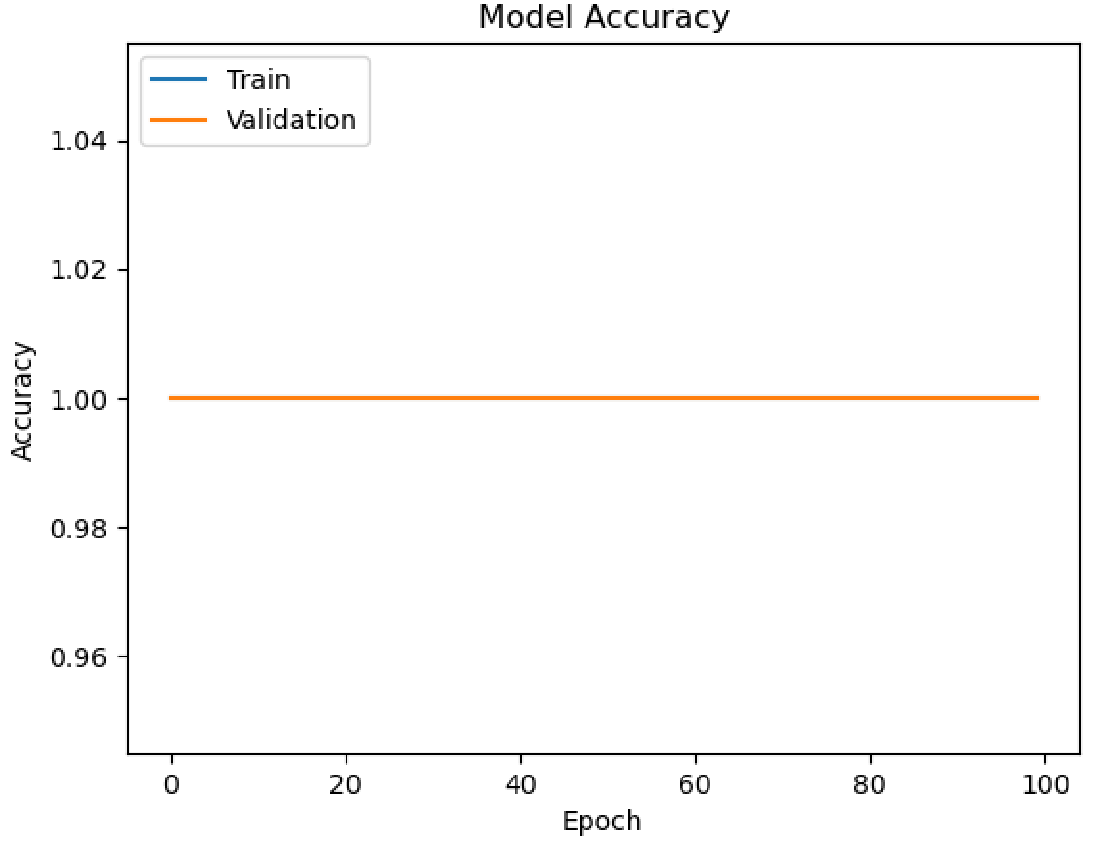
```
#K-Fold CV
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df


# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# Oversample the minority class
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)

# Set up k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_resampled.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Train and evaluate the model using k-fold cross-validation
acc_scores = []
loss_scores = []
for train_idx, test_idx in kf.split(X_resampled, y_resampled):
    X_train, X_test = X_resampled[train_idx], X_resampled[test_idx]
    y_train, y_test = y_resampled[train_idx], y_resampled[test_idx]
```

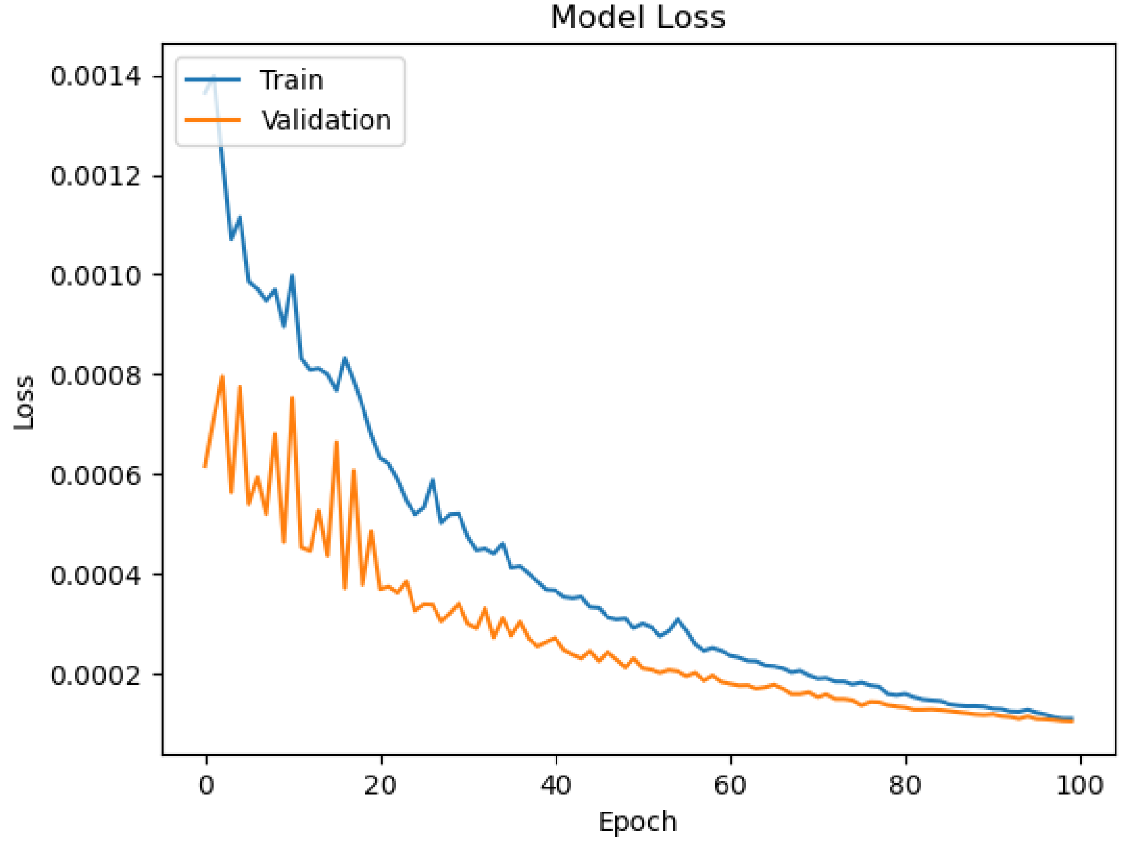
```
    history = model.fit(X_train, y_train, epochs=100, batch_size=32, validatio
n_data=(X_test, y_test), callbacks=[early_stopping], verbose=1)
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    acc_scores.append(accuracy)
    loss_scores.append(loss)

# Compute and print performance metrics
print('Accuracy:', np.mean(acc_scores))
print('Loss:', np.mean(loss_scores))

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
```

```
plt.show()
```

Output-

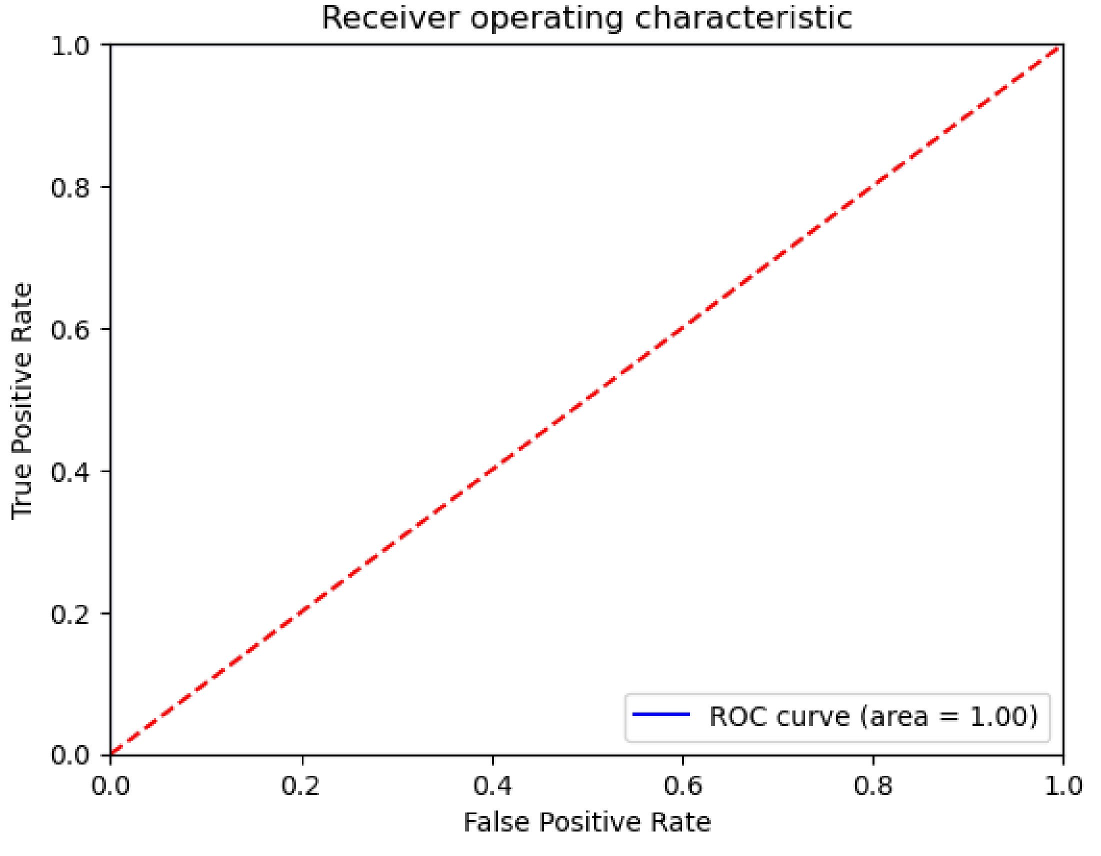
```
Epoch 100/100
17/17 [==============================] - 0s 6ms/step - loss: 1.1074e-04 -
accuracy: 1.0000 - val_loss: 1.0520e-04 - val_accuracy: 1.0000
Accuracy: 0.9909774541854859
Loss: 0.028932575223734602
```






```
5/5 [==============================] - 0s 4ms/step
```

```
Confusion matrix:
 [[72  0]
 [ 0 60]]
Precision: 1.0
Recall: 1.0
F1-score: 1.0
ROC AUC: 1.0
```



(b) leave one subject out cross-validation to simple 3-layer Neural network-

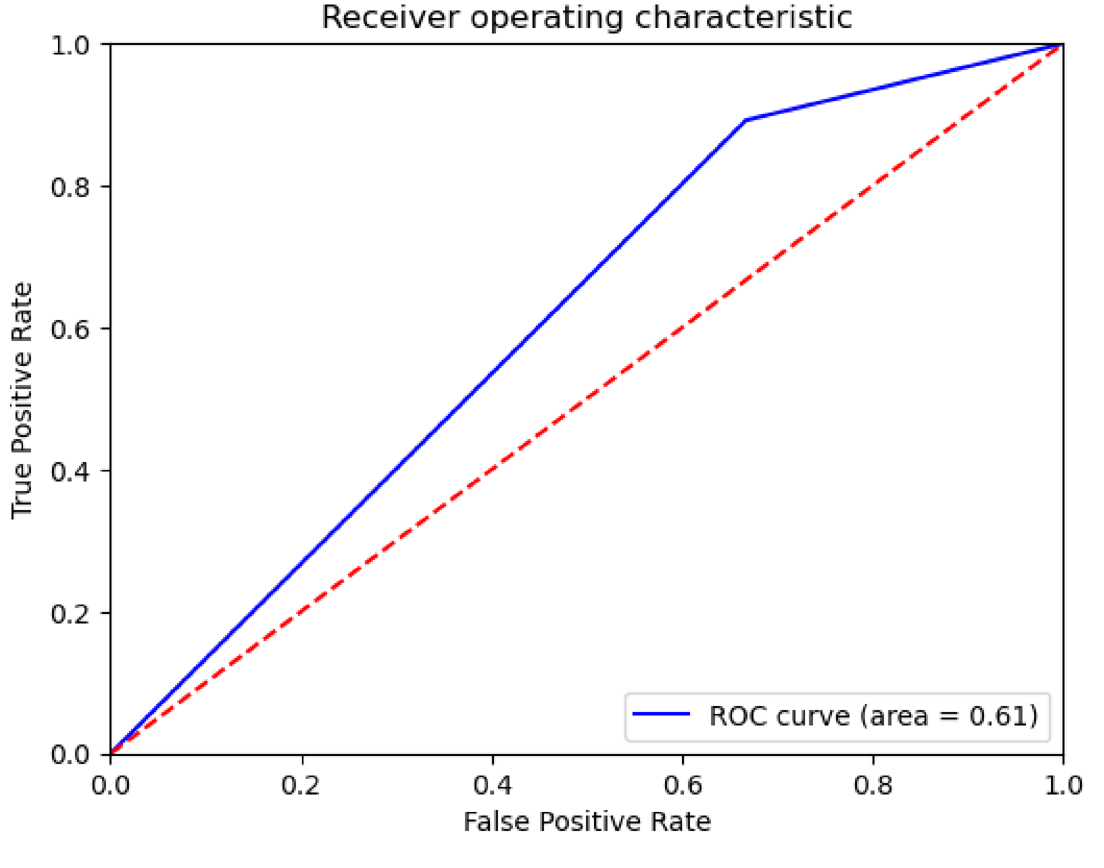
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import LeaveOneGroupOut
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from imblearn.over_sampling import RandomOverSampler
from sklearn.utils import shuffle

# Load the dataset
data = df

# Sort the data by subject id
data = data.sort_values('Sensitization type')

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values
groups = data.iloc[:, 1].values
```

```
# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Set up leave-one-subject-out cross-validation
logo = LeaveOneGroupOut()

# Train and evaluate the model with leave-one-subject-out cross-validation
for train_index, test_index in logo.split(X, y, groups=groups):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Shuffle the training set
    X_train, y_train = shuffle(X_train, y_train, random_state=42)

    # Oversample the minority class in the training set
    ros = RandomOverSampler(random_state=42)
    X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

    # Train the model
    history = model.fit(X_train_resampled, y_train_resampled, epochs=100, batc
h_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])

    # Evaluate the model on the test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print('Test loss:', loss)
    print('Test accuracy:', accuracy)

    # Make predictions on the test set
    y_pred = model.predict(X_test)
    y_pred = (y_pred > 0.5)

    # Compute performance metrics
    from sklearn.metrics import confusion_matrix, precision_score, recall_scor
e, f1_score, roc_curve, roc_auc_score
    cm = confusion_matrix(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    print('Confusion matrix:\n', cm)
    print('Precision:', precision)
```

```
    print('Recall:', recall)
    print('F1-score:', f1)
    print('ROC AUC:', roc_auc)
    plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc='lower right')
    plt.show()
```

Output-

```
Epoch 7/100
16/16 [==============================] - 0s 6ms/step - loss: 0.1053 - accuracy:
0.9738 - val_loss: 1.1759 - val_accuracy: 0.8061
4/4 [==============================] - 0s 3ms/step - loss: 1.1759 - accuracy:
0.8061
Test loss: 1.1759247779846191
Test accuracy: 0.8061224222183228
4/4 [==============================] - 0s 2ms/step
Confusion matrix:
 [[ 5 10]
 [ 9 74]]
Precision: 0.8809523809523809
Recall: 0.891566265060241
F1-score: 0.8862275449101796
ROC AUC: 0.6124497991967872
```



(c) k-stratified cross-validation to simple 3-layer Neural network-

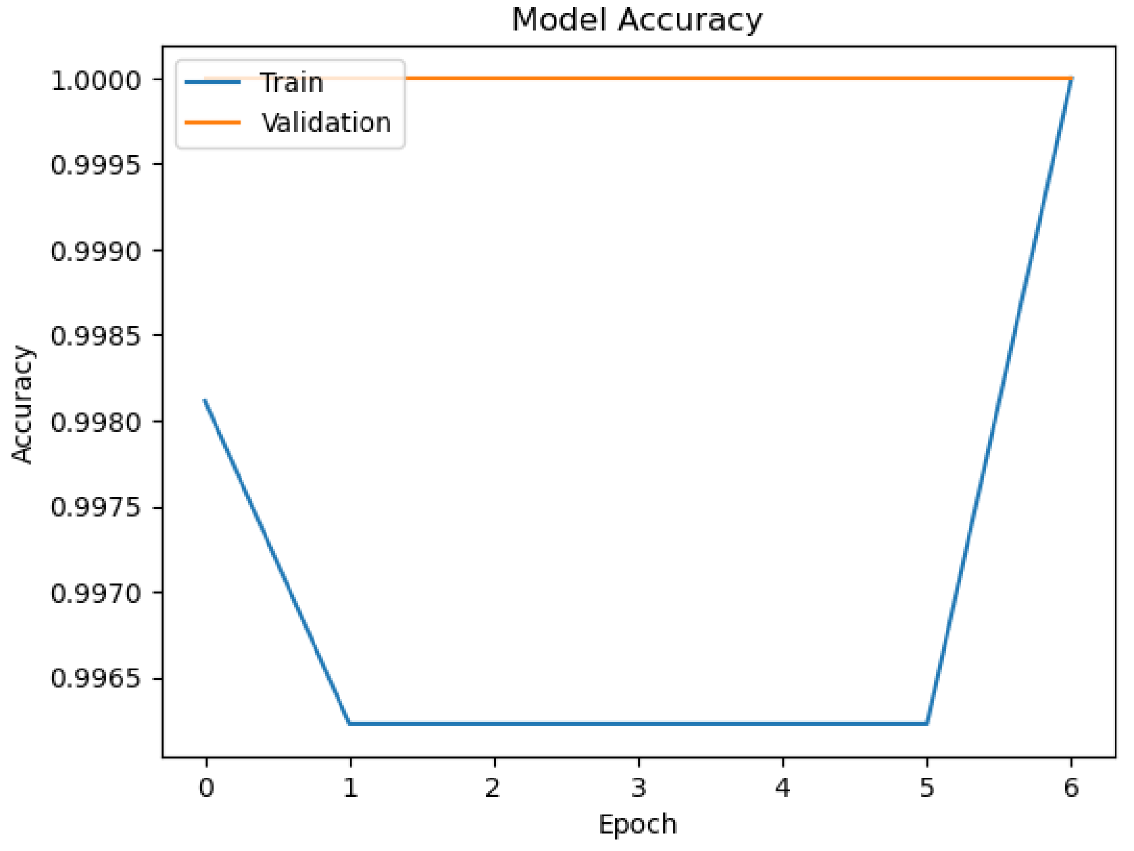
```
#stratified cross-validation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.utils import resample

# Load the dataset
data = df

# Split the dataset into X (independent variables) and y (dependent variable)
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

# Define the number of splits for stratified cross-validation
n_splits = 5

# Define the model architecture
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

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

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Set up stratified cross-validation
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

# Train and evaluate the model with stratified cross-validation
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Oversample the minority class in the training set
    ros = RandomOverSampler(random_state=42)
    X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

    # Train the model
    history = model.fit(X_train_resampled, y_train_resampled, epochs=100, batc
h_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])

    # Evaluate the model on the test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print('Test loss:', loss)
```

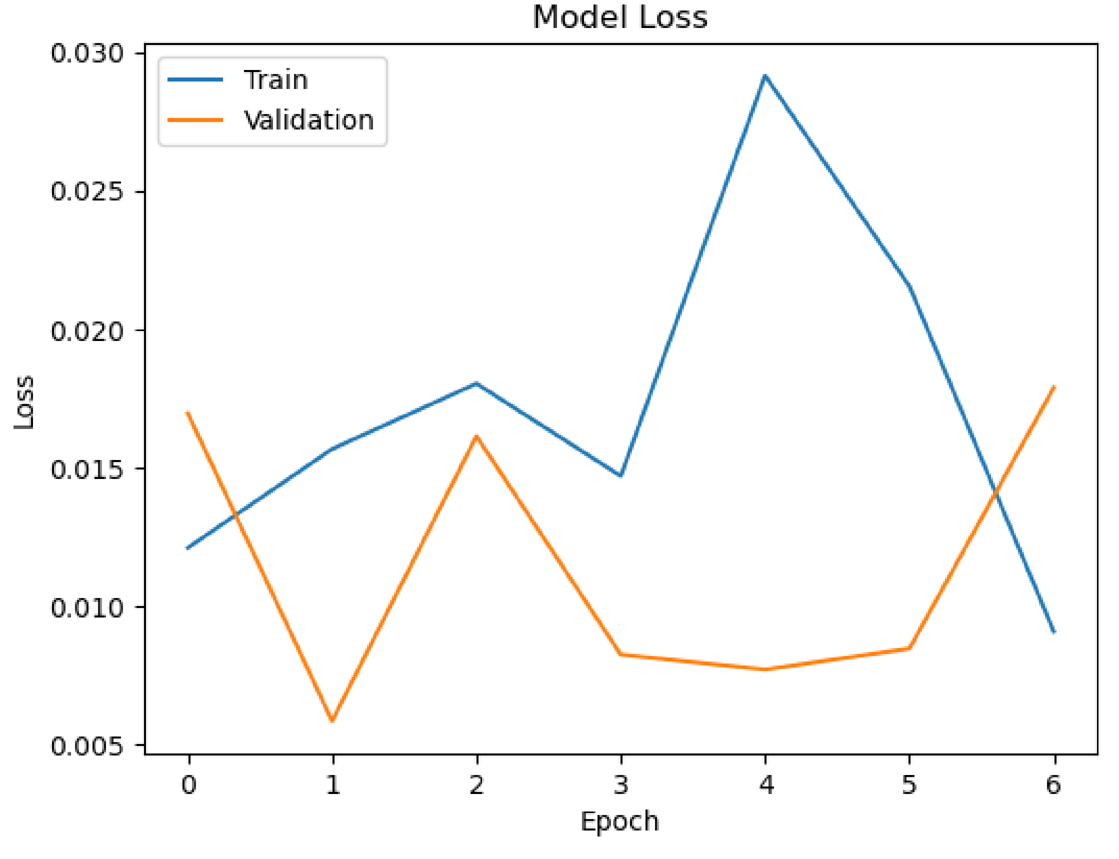
```
    print('Test accuracy:', accuracy)

# Plot accuracy and loss curves
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Make predictions on the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)

# Compute performance metrics
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f
1_score, roc_curve, roc_auc_score
cm = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('ROC AUC:', roc_auc)
plt.plot(fpr, tpr, 'b', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

Output-

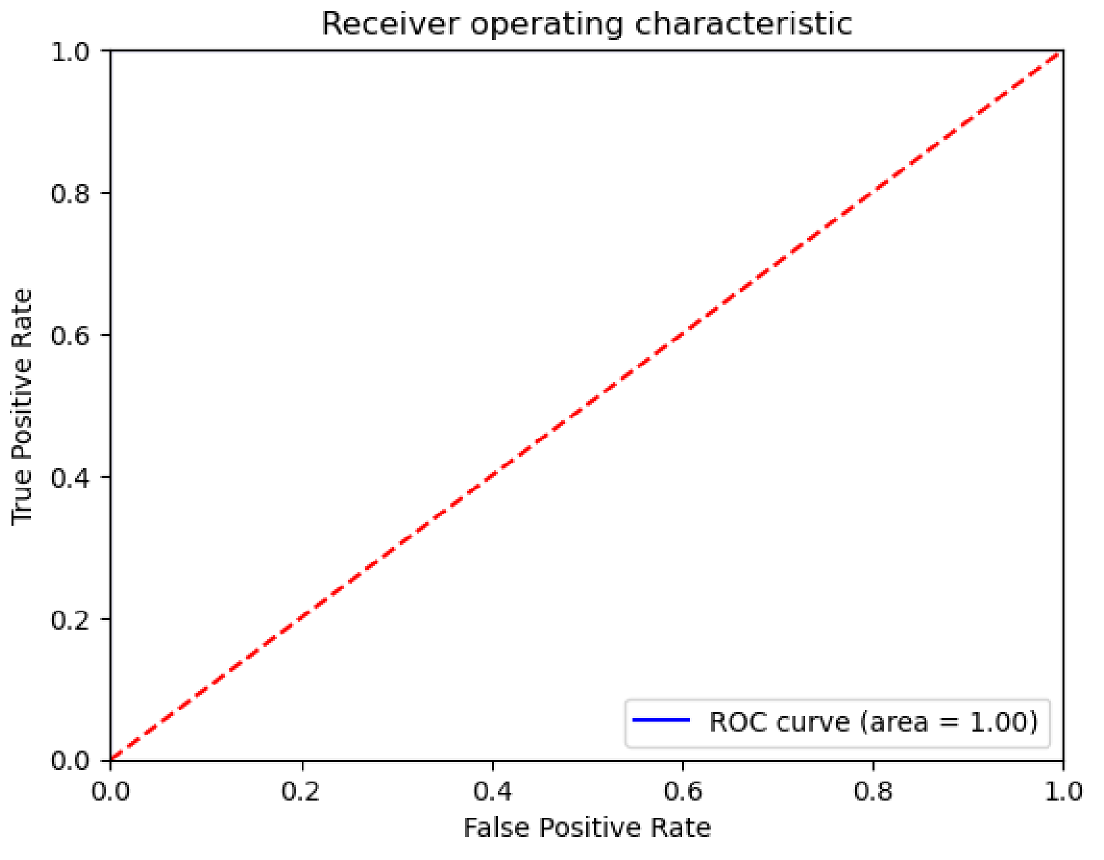
```
Epoch 7/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0091 - accuracy:
1.0000 - val_loss: 0.0179 - val_accuracy: 1.0000
3/3 [==============================] - 0s 5ms/step - loss: 0.0179 - accuracy:
1.0000
Test loss: 0.017888842150568962
```

```
Test accuracy: 1.0
```






```
3/3 [==============================] - 0s 4ms/step
Confusion matrix:
 [[ 8  0]
 [ 0 66]]
Precision: 1.0
Recall: 1.0
F1-score: 1.0
ROC AUC: 1.0
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)