Use markdown to label each (sub)question neatly.

This notebook serves as your report. All your answers should be presented within it.

You can submit multiple notebooks (e.g. 1 notebook per part / question).

Before submission, remember to tidy up the notebook and retain only relevant parts.

Part A Question 1

In [ ]:
```
import time
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import shap
shap.initjs()


import IPython.display as ipd

from scipy.io import wavfile as wav

from sklearn import preprocessing
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix


import tensorflow.keras as keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import *
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping,ModelCheckpoint,
```

```
c:\Users\JoeTe\AppData\Local\Programs\Python\Python310\lib\site-packages\tqdm\auto.py:2
2: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://
ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [ ]:
```
SEED = 42

os.environ['TF_CUDNN_DETERMINISTIC'] = '1'

import random
random.seed(SEED)

import numpy as np
```

```
np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
```

Read Data

In [ ]:
```
df = pd.read_csv('./full.csv')
df.head()
```

Out[ ]:

	filename	tempo	total_beats	average_beats	chroma_stft_mean	chroma
0	app_3001_4001_phnd_neg_0000.wav	184.570312	623	69.222222	0.515281	
1	app_3001_4001_phnd_neg_0001.wav	151.999081	521	74.428571	0.487201	
2	app_3001_4001_phnd_neg_0002.wav	112.347147	1614	146.727273	0.444244	
3	app_3001_4001_phnd_neg_0003.wav	107.666016	2060	158.461538	0.454156	
4	app_3001_4001_phnd_neg_0004.wav	75.999540	66	33.000000	0.478780	

5 rows × 78 columns

In [ ]:
```
df['label'] = df['filename'].str.split('_').str[-2]
```

In [ ]:
```
df['label'].value_counts()
```

Out[ ]:
```
pos    92826
neg    89428
Name: label, dtype: int64
```

Split and scale dataset

In [ ]:
```
columns_to_drop = ['label','filename']

def split_dataset(df, columns_to_drop, test_size, random_state):
  label_encoder = preprocessing.LabelEncoder()

  df['label'] = label_encoder.fit_transform(df['label'])

  df_train, df_test = train_test_split(df, test_size=test_size, random_state=random_sta

  df_train2 = df_train.drop(columns_to_drop,axis=1)
  y_train2 = df_train['label'].to_numpy()

  df_test2 = df_test.drop(columns_to_drop,axis=1)
  y_test2 = df_test['label'].to_numpy()

  return df_train2, y_train2, df_test2, y_test2

def preprocess_dataset(df_train, df_test):

  standard_scaler = preprocessing.StandardScaler()
  df_train_scaled = standard_scaler.fit_transform(df_train)
```

```
  df_test_scaled = standard_scaler.transform(df_test)

  return df_train_scaled, df_test_scaled

X_train, y_train, X_test, y_test = split_dataset(df, columns_to_drop, test_size=0.3, ra

X_train_scaled, X_test_scaled = preprocess_dataset(X_train, X_test)
```

In [ ]:
```
features = []
for i in X_train.columns[0:]:
    features.append(i)
```

Question 1A

In [ ]:
```
num_neurons = 128
learning_rate = 0.001
batch_size = 256
no_epochs = 100
```

In [ ]:
```
model = Sequential([Dense(num_neurons, activation='relu'),
                    Dropout(0.2), Dense(num_neurons, activation ='relu'),
                    Dropout(0.2), Dense(num_neurons, activation='relu'),
                    Dropout(0.2), Dense(1, activation='sigmoid')])
```

Callback to monitor val accuracy. I am not sure if val_loss should be used instead when I am using the validation accuracy for justification on the plots and optimal batch size & number of neurons.

In [ ]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
model.compile(loss= 'binary_crossentropy', optimizer = "adam", metrics= ['accuracy']) #
```

In [ ]:
```
history = {}
history['model'] = model.fit(X_train_scaled, y_train,
                             epochs = no_epochs, verbose = 1,
                             batch_size = batch_size, validation_data = (X_test_scaled, y
```

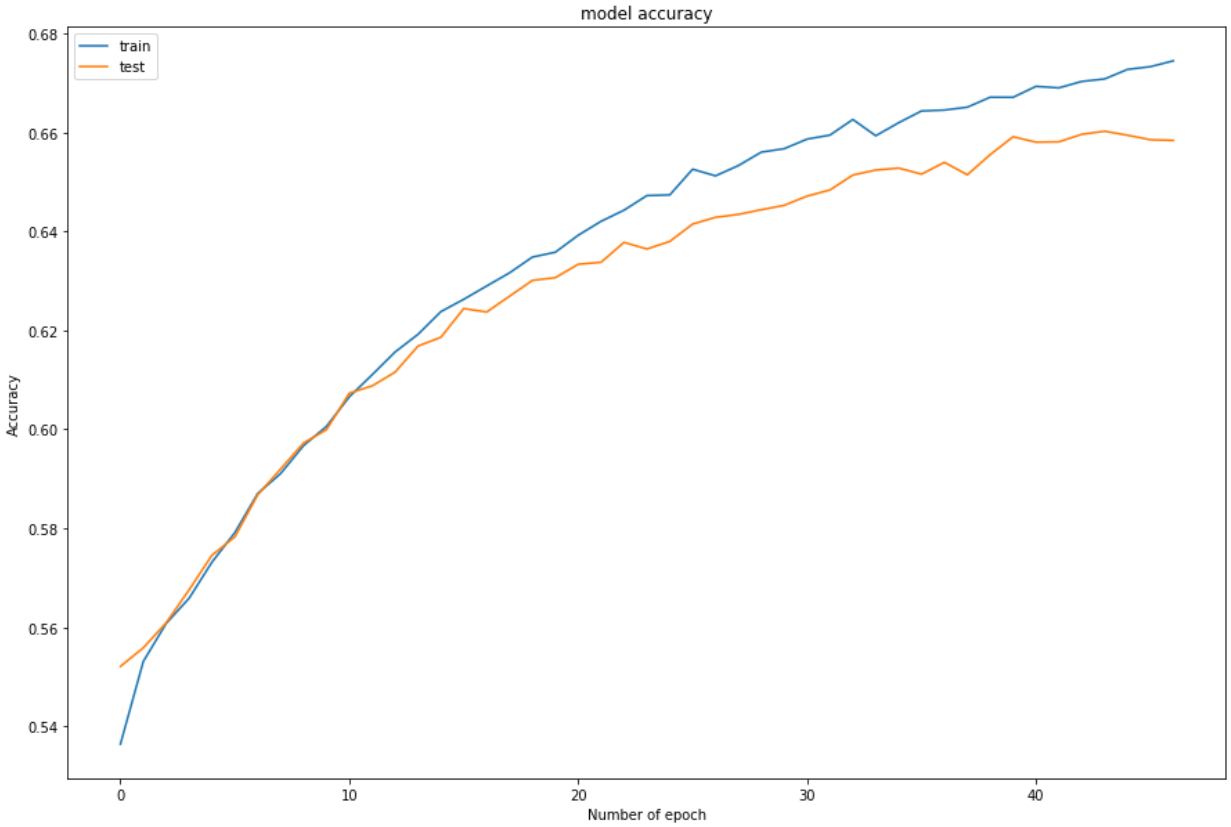
```
Epoch 1/100
499/499 [==============================] - 4s 4ms/step - loss: 0.6896 - accuracy: 0.536
4 - val_loss: 0.6847 - val_accuracy: 0.5521
Epoch 2/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6834 - accuracy: 0.553
1 - val_loss: 0.6817 - val_accuracy: 0.5558
Epoch 3/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6800 - accuracy: 0.560
8 - val_loss: 0.6800 - val_accuracy: 0.5609
Epoch 4/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6772 - accuracy: 0.565
8 - val_loss: 0.6765 - val_accuracy: 0.5676
Epoch 5/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6740 - accuracy: 0.573
1 - val_loss: 0.6734 - val_accuracy: 0.5745
Epoch 6/100
499/499 [==============================] - 2s 5ms/step - loss: 0.6704 - accuracy: 0.579
1 - val_loss: 0.6707 - val_accuracy: 0.5783
Epoch 7/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6661 - accuracy: 0.587
0 - val_loss: 0.6676 - val_accuracy: 0.5868
Epoch 8/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6632 - accuracy: 0.591
0 - val_loss: 0.6629 - val_accuracy: 0.5920
Epoch 9/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6589 - accuracy: 0.596
7 - val_loss: 0.6601 - val_accuracy: 0.5972
Epoch 10/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6558 - accuracy: 0.600
6 - val_loss: 0.6582 - val_accuracy: 0.5999
Epoch 11/100
499/499 [==============================] - 2s 5ms/step - loss: 0.6522 - accuracy: 0.606
6 - val_loss: 0.6529 - val_accuracy: 0.6073
Epoch 12/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6483 - accuracy: 0.611
0 - val_loss: 0.6502 - val_accuracy: 0.6088
Epoch 13/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6453 - accuracy: 0.615
6 - val_loss: 0.6484 - val_accuracy: 0.6115
Epoch 14/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6416 - accuracy: 0.619
2 - val_loss: 0.6449 - val_accuracy: 0.6168
Epoch 15/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6390 - accuracy: 0.623
8 - val_loss: 0.6422 - val_accuracy: 0.6186
Epoch 16/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6363 - accuracy: 0.626
3 - val_loss: 0.6401 - val_accuracy: 0.6244
Epoch 17/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6337 - accuracy: 0.628
9 - val_loss: 0.6383 - val_accuracy: 0.6237
Epoch 18/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6308 - accuracy: 0.631
6 - val_loss: 0.6359 - val_accuracy: 0.6269
Epoch 19/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6283 - accuracy: 0.634
8 - val_loss: 0.6332 - val_accuracy: 0.6301
Epoch 20/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6269 - accuracy: 0.635
8 - val_loss: 0.6345 - val_accuracy: 0.6306
```

```
Epoch 21/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6239 - accuracy: 0.639
2 - val_loss: 0.6305 - val_accuracy: 0.6333
Epoch 22/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6227 - accuracy: 0.642
0 - val_loss: 0.6297 - val_accuracy: 0.6337
Epoch 23/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6192 - accuracy: 0.644
3 - val_loss: 0.6275 - val_accuracy: 0.6378
Epoch 24/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6179 - accuracy: 0.647
2 - val_loss: 0.6260 - val_accuracy: 0.6364
Epoch 25/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6165 - accuracy: 0.647
4 - val_loss: 0.6245 - val_accuracy: 0.6379
Epoch 26/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6139 - accuracy: 0.652
5 - val_loss: 0.6226 - val_accuracy: 0.6415
Epoch 27/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6126 - accuracy: 0.651
2 - val_loss: 0.6220 - val_accuracy: 0.6428
Epoch 28/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6109 - accuracy: 0.653
3 - val_loss: 0.6201 - val_accuracy: 0.6434
Epoch 29/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6076 - accuracy: 0.656
0 - val_loss: 0.6185 - val_accuracy: 0.6444
Epoch 30/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6076 - accuracy: 0.656
7 - val_loss: 0.6180 - val_accuracy: 0.6452
Epoch 31/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6070 - accuracy: 0.658
6 - val_loss: 0.6176 - val_accuracy: 0.6471
Epoch 32/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6047 - accuracy: 0.659
4 - val_loss: 0.6168 - val_accuracy: 0.6484
Epoch 33/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6028 - accuracy: 0.662
6 - val_loss: 0.6138 - val_accuracy: 0.6514
Epoch 34/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6036 - accuracy: 0.659
3 - val_loss: 0.6143 - val_accuracy: 0.6524
Epoch 35/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6024 - accuracy: 0.661
9 - val_loss: 0.6124 - val_accuracy: 0.6528
Epoch 36/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6000 - accuracy: 0.664
3 - val_loss: 0.6122 - val_accuracy: 0.6516
Epoch 37/100
499/499 [==============================] - 2s 4ms/step - loss: 0.6000 - accuracy: 0.664
5 - val_loss: 0.6104 - val_accuracy: 0.6539
Epoch 38/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5983 - accuracy: 0.665
1 - val_loss: 0.6112 - val_accuracy: 0.6514
Epoch 39/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5971 - accuracy: 0.667
1 - val_loss: 0.6103 - val_accuracy: 0.6555
Epoch 40/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5975 - accuracy: 0.667
1 - val_loss: 0.6075 - val_accuracy: 0.6591
```

```
Epoch 41/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5953 - accuracy: 0.669
3 - val_loss: 0.6071 - val_accuracy: 0.6580
Epoch 42/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5944 - accuracy: 0.669
0 - val_loss: 0.6073 - val_accuracy: 0.6581
Epoch 43/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5937 - accuracy: 0.670
3 - val_loss: 0.6051 - val_accuracy: 0.6596
Epoch 44/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5932 - accuracy: 0.670
8 - val_loss: 0.6062 - val_accuracy: 0.6602
Epoch 45/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5910 - accuracy: 0.672
7 - val_loss: 0.6041 - val_accuracy: 0.6594
Epoch 46/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5895 - accuracy: 0.673
3 - val_loss: 0.6050 - val_accuracy: 0.6585
Epoch 47/100
499/499 [==============================] - 2s 4ms/step - loss: 0.5900 - accuracy: 0.674
4 - val_loss: 0.6054 - val_accuracy: 0.6584
```

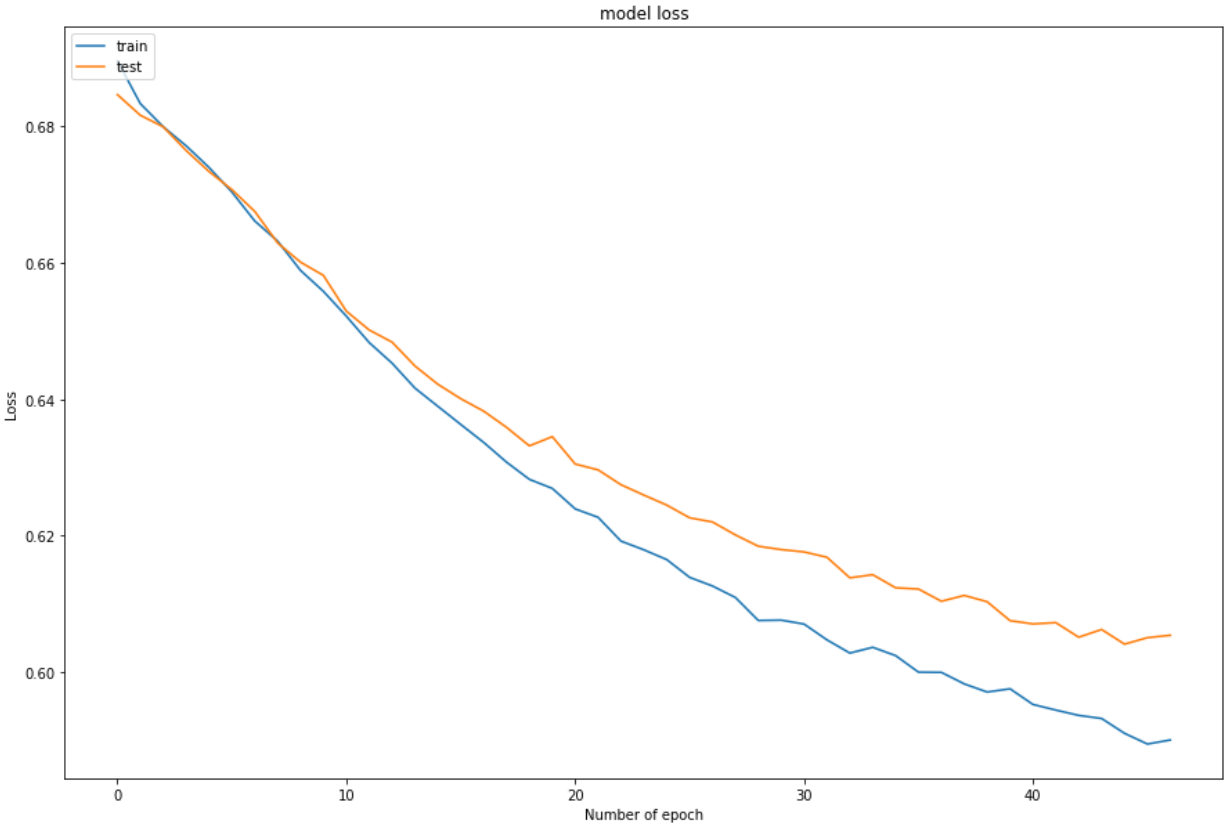
Plot of train and test accuracies training epochs

In [ ]:
```
plt_1 = plt.figure(figsize=(15, 10))
plt.plot(history['model'].history['accuracy'])
plt.plot(history['model'].history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Number of epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Plot of train and test loss training epochs

In [ ]:
```
plt_1 = plt.figure(figsize=(15, 10))
plt.plot(history['model'].history['loss'])
plt.plot(history['model'].history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Number of epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Question 1B

Comments on the line plots

Both training and validation accuracies of the model is increasing at a decreasing rate. Initially as the model trains, both train and test accuracy of the model rises sharply. Soon, its growth would decrease and eventually reach a plateau.

If there was no early-stopping mechanism, the model might overfit which would compromise on the model's accuracy.

Both training and validation loss are decreasing until reaches the minimum. Loss indicates the error or "how bad" the predictions are against the targeted value. We used binary cross entropy as it compares each of the predicated probabilities to the actual class output which can be either 0 or 1. As such, we aim to minimize loss as it indicates robustness of the model.

Why early stopping is used?

Early stopping is required to reduce overfitting without compromising on model accuracy. With the early stopping technique, the model will stop training before it starts to overfit which leads to increase in error and a drop in model's accuracy. With patience of 3, the training would be stopped when validation accuracy do not improve after 3 epoch.