Colab Link - [link]

a) Import the data file (spotify_preprocessed.csv) to your code. The data is preprocessed and ready to use.

Importing the require libraries:

```
[1]  1 import numpy as np
     2 import pandas as pd
     3 import matplotlib.pyplot as plt
     4 from tensorflow import keras
     5 from tensorflow.keras import layers

[2]  1 from google.colab import files
     2 uploaded = files.upload()

     1 import io
     2 data = pd.read_csv(io.BytesIO(uploaded['spotify_preprocessed_-401226519.csv']))
     3 #data = pd.read_csv('spotify_preprocessed_-401226519.csv')

[4]  1 data
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	chorus_hit	s
0	0.738790	0.626533	0.090909	0.899432	0.0	0.070809	0.020080	0.000000	0.068476	0.723361	0.400098	0.093080	0.8	0.193225	
1	0.418807	0.247058	0.454545	0.687954	0.0	0.012962	0.874498	0.818090	0.080700	0.256148	0.676658	0.086266	0.6	0.155665	
2	0.530910	0.415269	0.818182	0.862211	0.0	0.031601	0.161647	0.000000	0.094582	0.280738	0.773251	0.103036	0.8	0.210605	

b) Shuffle the data then split it into training (90% of the data) and test set (10% of the data). Split the training set further into training and validation sets with 80% and 20% percentages respectively.

Shuffling the data:

```
     1 data.sample(frac=1).reset_index(drop=True) # shuffled the data
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	chorus_hit	s
0	0.559208	0.890863	0.000000	0.864942	1.0	0.016283	0.008735	0.048543	0.076039	0.381148	0.574577	0.119522	0.8	0.204268	
1	0.710492	0.907885	0.636364	0.873651	0.0	0.082271	0.017269	0.000084	0.424013	0.861680	0.498945	0.140222	0.8	0.116151	
2	0.632129	0.560450	0.000000	0.846342	1.0	0.013819	0.204819	0.197990	0.076764	0.533811	0.236685	0.041156	0.8	0.242942	
3	0.254462	0.931915	0.181818	0.929859	1.0	0.042528	0.000092	0.624121	0.735833	0.187500	0.687905	0.055331	0.8	0.175462	
4	0.154332	0.988986	0.727273	0.891433	1.0	0.091591	0.000003	0.003879	0.129804	0.356557	0.635984	0.093541	0.8	0.155059	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6393	0.608185	0.802753	0.909091	0.932568	0.0	0.018747	0.007480	0.000027	0.601160	0.908811	0.335899	0.112763	0.8	0.261876	
6394	0.323030	0.983980	0.181818	0.923859	0.0	0.068452	0.000091	0.000036	0.329742	0.497951	0.895489	0.123042	0.8	0.094283	
6395	0.770353	0.882853	0.727273	0.900163	1.0	0.040386	0.194779	0.000000	0.310059	0.394467	0.353259	0.113633	0.8	0.219791	
6396	0.548324	0.588485	0.636364	0.826065	0.0	0.011784	0.311245	0.000000	0.132912	0.560451	0.584617	0.101349	0.8	0.123327	
6397	0.646278	0.432290	0.545455	0.799338	0.0	0.208356	0.444779	0.000000	0.119445	0.091906	0.335958	0.112959	0.8	0.379682	

6398 rows × 16 columns

+ Code + Text

```
     1 from sklearn.model_selection import train_test_split
     2 train, test = train_test_split(data,test_size = 0.1,random_state=42)

[10] 1 train,val_data = train_test_split(train,test_size = 0.2,random_state=42)

[11] 1 train

[12] 1 X_train = train.drop('target',axis=1)
     2 y_train = train['target']

[13] 1 X_val = val_data.drop('target',axis=1)
     2 y_val = val_data['target']

[14] 1 X_test = test.drop('target',axis=1)
     2 y_test = test['target']
```

c) Build, compile, train, and then evaluate:

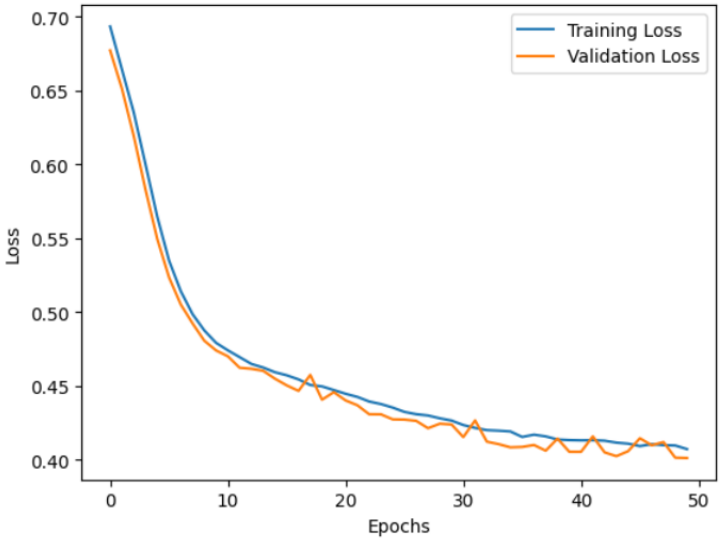
- Build a neural network with 2 hidden layers that contain 32 nodes each and an output layer that has 1 unit using the Keras library.

- Compile the network. Select binary cross-entropy (binary_crossentropy) as the loss function. Use stochastic gradient descent learning (SGD, learning rate of 0.01).
- Train the network for 50 epochs and a batch size of 16.
- Plot the training loss and validation loss (i.e., the learning curve) for all the epochs. 2 e. Use the evaluate() Keras function to find the training and validation loss and accuracy.

Building the model with all the above mentioned criteria and evaluating the corresponding losses:

```
import tensorflow as tf
from tensorflow import keras

# Build the neural network 2 hidden layers that contain 32 nodes each and an output layer that has 1 unit using the Keras library.
model = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the network
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 50/50
288/288 [==============================] - 1s 3ms/step - loss: 0.4072 - accuracy: 0.8163 - val_loss: 0.4012 - val_accu
```




```
# Evaluate the model
train_loss, train_accuracy = model.evaluate(X_train, y_train)
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

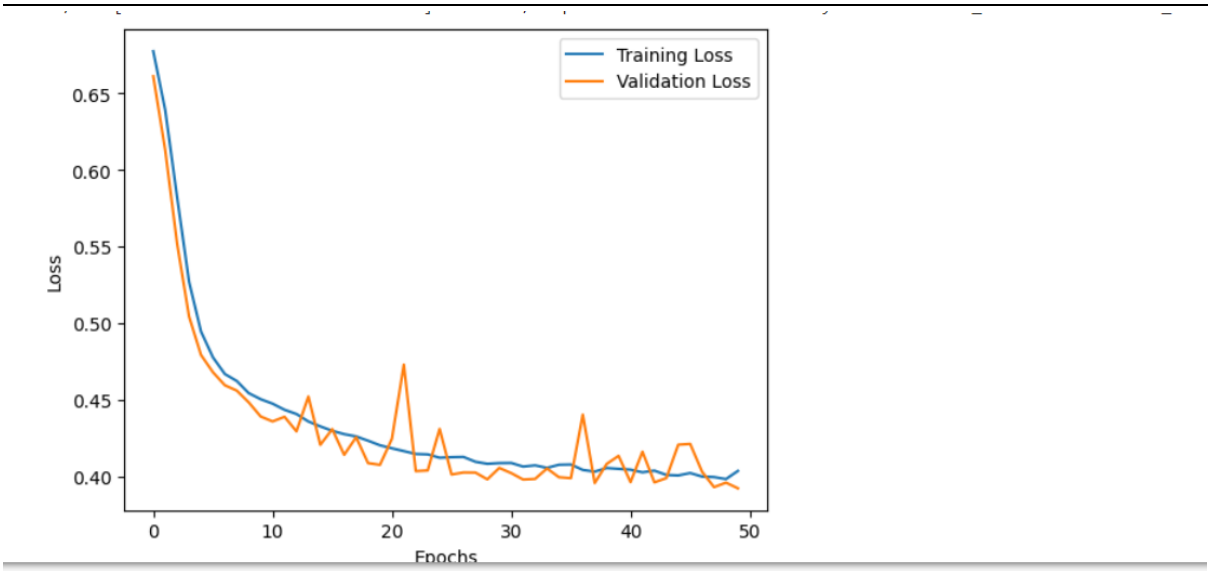
```
144/144 [==============================] - 1s 3ms/step - loss: 0.4028 - accuracy: 0.8224
36/36 [==============================] - 0s 3ms/step - loss: 0.4012 - accuracy: 0.8134
Training Loss: 0.4028, Training Accuracy: 0.8224
Validation Loss: 0.4012, Validation Accuracy: 0.8134
```

d) Try different design ideas with the model until you get the best training and validation performance. For example, changing the number of hidden layers and number of units in each, changing the loss function, the learning algorithm, the learning rate, number of epochs and the batch size. Repeat the scores in a table.

Trying for different combinations as mentioned above:

For 3 layers, 64,32,32, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16

```python
# adding different number of hidden layers considering
#3 layers, 64,32,32, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16
model1 = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
model1.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model1.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



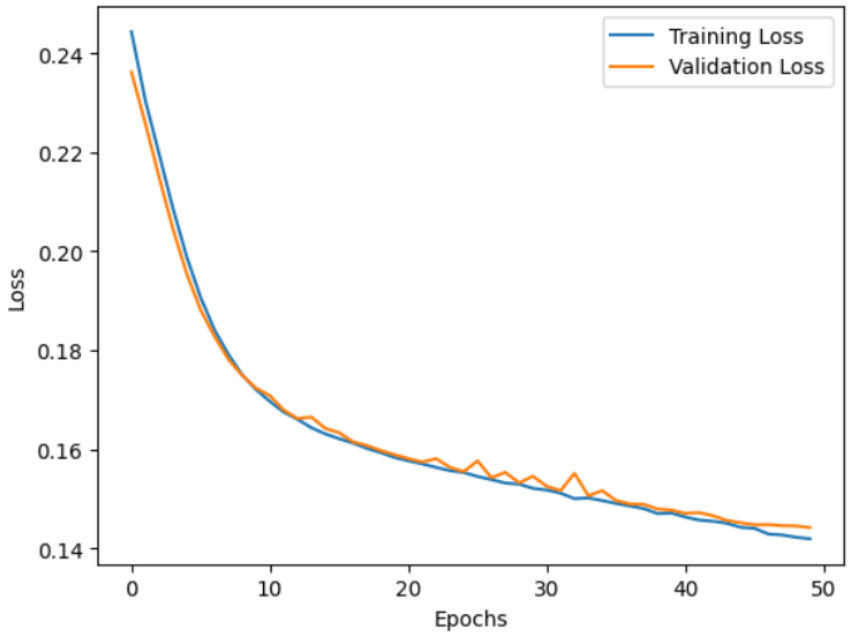

```python
[18]
# Evaluate the above model
train_loss, train_accuracy = model1.evaluate(X_train, y_train)
val_loss, val_accuracy = model1.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 1s 4ms/step - loss: 0.3927 - accuracy: 0.8241
36/36 [==============================] - 0s 3ms/step - loss: 0.3925 - accuracy: 0.8168
Training Loss: 0.3927, Training Accuracy: 0.8241
Validation Loss: 0.3925, Validation Accuracy: 0.8168
```

For 2 layers, mean_squared_error, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16

```python
#creating model with (2 layers, mean_squared_error, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16)
model2 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# modifying loss funtion
model2.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='mean_squared_error', metrics=['accuracy'])

# Train the network
history = model2.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

```

```
288/288 [==============================] - 1s 3ms/step - loss: 0.1419 - accuracy: 0.8037 - val_loss: 0.1
```

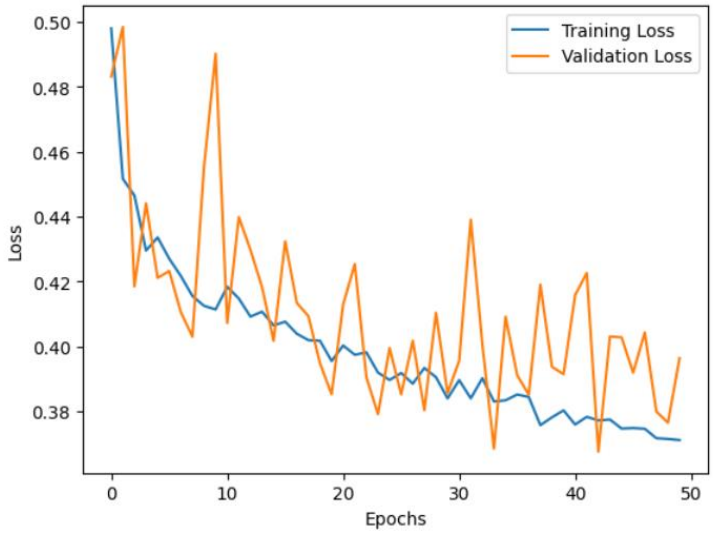



```python
# Evaluate the model
train_loss, train_accuracy = model2.evaluate(X_train, y_train)
val_loss, val_accuracy = model2.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.1427 - accuracy: 0.7992
36/36 [==============================] - 0s 2ms/step - loss: 0.1442 - accuracy: 0.7882
Training Loss: 0.1427, Training Accuracy: 0.7992
Validation Loss: 0.1442, Validation Accuracy: 0.7882
```

For2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=Adam, batch_size=16:

```python
# creating  model with (2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=Adam, batch_size=16)
model3 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# modifying loss funtion
model3.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model3.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

```



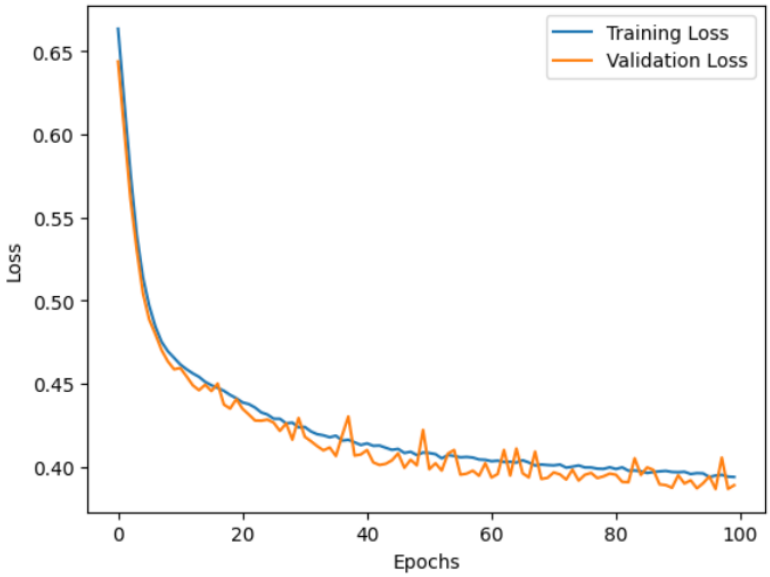

```python
# Evaluate the model
train_loss, train_accuracy = model3.evaluate(X_train, y_train)
val_loss, val_accuracy = model3.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.3705 - accuracy: 0.8326
36/36 [==============================] - 0s 2ms/step - loss: 0.3964 - accuracy: 0.8220
Training Loss: 0.3705, Training Accuracy: 0.8326
Validation Loss: 0.3964, Validation Accuracy: 0.8220
```

For2 layers, binary_crossentropy, epochs=100,activation=sigmoid,optimize=SGD, batch_size=16:

```python
# creating model with (2 layers, binary_crossentropy, epochs=100,activation=sigmoid,optimize=SGD, batch_size=16)
model4 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# modifying loss funtion
model4.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model4.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

```
288/288 [==============================] - 1s 4ms/step - loss: 0.3938 - accuracy: 0.8244 - val_loss: 0.3889 - v
```

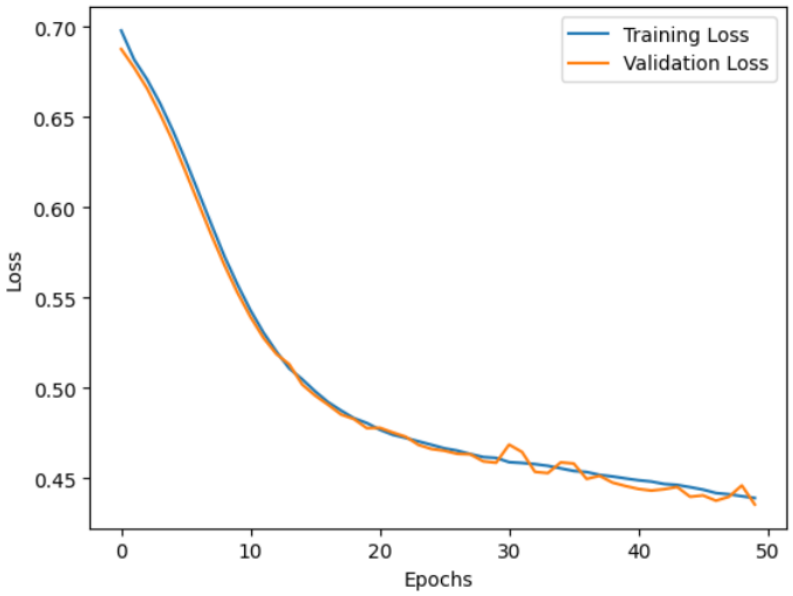



```python
[24]  # Evaluate the model
      train_loss, train_accuracy = model4.evaluate(X_train, y_train)
      val_loss, val_accuracy = model4.evaluate(X_val, y_val)
      print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
      print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.3887 - accuracy: 0.8281
36/36 [==============================] - 0s 3ms/step - loss: 0.3889 - accuracy: 0.8273
Training Loss: 0.3887, Training Accuracy: 0.8281
Validation Loss: 0.3889, Validation Accuracy: 0.8273
```

For2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=32:

```python
# creating model with (2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=32)
model5 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# modifying loss funtion
model5.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model5.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

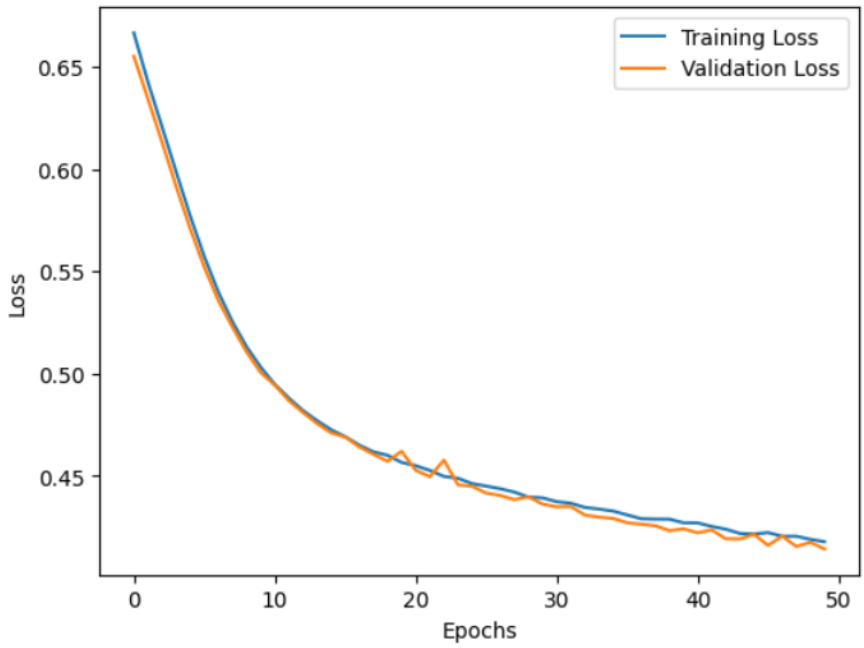



```python
# Evaluate the model
train_loss, train_accuracy = model5.evaluate(X_train, y_train)
val_loss, val_accuracy = model5.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 1s 3ms/step - loss: 0.4371 - accuracy: 0.8048
36/36 [==============================] - 0s 3ms/step - loss: 0.4355 - accuracy: 0.7960
Training Loss: 0.4371, Training Accuracy: 0.8048
Validation Loss: 0.4355, Validation Accuracy: 0.7960
```

For 2 layers, binary_crossentropy, epochs=50,activation= softmax,optimize=SGD

```python
# creating model with (2 layers, binary_crossentropy, epochs=50,activation= softmax,optimize=SGD)
model6 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='softmax')
])

# modifying loss funtion
model6.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model6.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



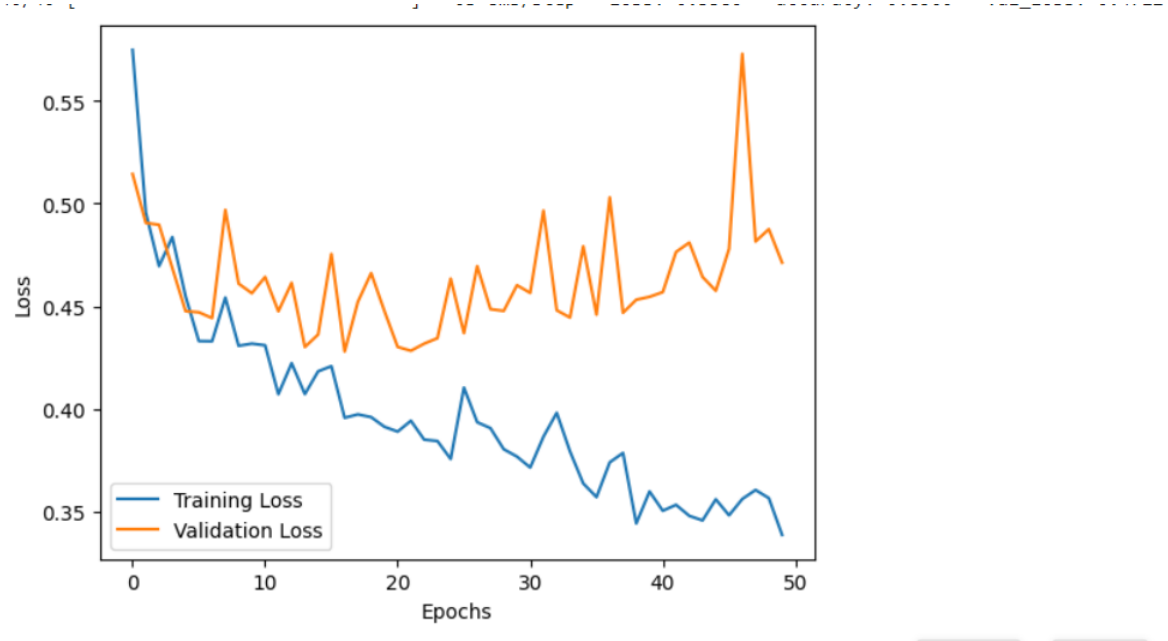

```
[29]  1 # Evaluate the model
      2 train_loss, train_accuracy = model6.evaluate(X_train, y_train)
      3 val_loss, val_accuracy = model6.evaluate(X_val, y_val)
      4 print(f"Training Loss: {train_loss:.4f}, Training Accuracy: {train_accuracy:.4f}")
      5 print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
144/144 [==============================] - 1s 3ms/step - loss: 0.4154 - accuracy: 0.5011
36/36 [==============================] - 0s 5ms/step - loss: 0.4141 - accuracy: 0.5009
Training Loss: 0.4154, Training Accuracy: 0.5011
Validation Loss: 0.4141, Validation Accuracy: 0.5009
```

e) Repeat parts (c) and (d) and select the model with the best performance.

Considering 2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=Adam, batch_size=16

```
# considering the model with best performence and fitting the model.
model7 = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(X_test.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# modifying loss funtion
model7.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the network
history = model7.fit(X_test, y_test, epochs=50, batch_size=16, validation_data=(X_val, y_val))

# Plot the learning curve
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()


```



f) Evaluate the selected model on the test set and report the testing loss and accuracy.

```
[45]  1 # Evaluate the model
      2 train_loss, train_accuracy = model7.evaluate(X_test, y_test)
      3 val_loss, val_accuracy = model4.evaluate(X_val, y_val)
      4 print(f"Test Loss: {train_loss:.4f}, Test Accuracy: {train_accuracy:.4f}")
      5 print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
```

```
20/20 [==============================] - 0s 3ms/step - loss: 0.3255 - accuracy: 0.8500
36/36 [==============================] - 0s 3ms/step - loss: 0.3889 - accuracy: 0.8273
Test Loss: 0.3255, Test Accuracy: 0.8500
Validation Loss: 0.3889, Validation Accuracy: 0.8273
```

Accuracy of above models:

index	models	Training_Loss	Training_Accuracy	Validation_Loss	Validation_Accuracy
0	2 layers,32,32, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD	0.40	0.82	0.40	0.81
1	3 layers, 64,32,32, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16	0.39	0.82	0.39	0.81
2	2 layers, mean_squared_error, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16	0.14	0.79	0.14	0.78
3	2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=Adam, batch_size=16	0.37	0.83	0.39	0.82
4	2 layers, binary_crossentropy, epochs=100,activation=sigmoid,optimize=SGD, batch_size=16	0.38	0.82	0.38	0.82
5	2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=32	0.43	0.80	0.43	0.79
6	2 layers, binary_crossentropy, epochs=50,activation= softmax,optimize=SGD	0.41	0.50	0.41	0.50