Colab Link - [Link for Colab]

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

Importing the require libraries:

+ Code + Text

[1]
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
```

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

Choose Files spotify_pre...1226519.csv
- **spotify_preprocessed_-401226519.csv**(text/csv) - 1656073 bytes, last modified: 10/28/2023 - 100% done

```
Saving spotify_preprocessed_-401226519.csv to spotify_preprocessed_-401226519 (1).csv
```

[4]
```python
import io
data = pd.read_csv(io.BytesIO(uploaded['spotify_preprocessed_-401226519 (1).csv']))
#data = pd.read_csv('spotify_preprocessed_-401226519.csv')
```

[5]
```python
data
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	chorus_hit	sections	target
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	0.093023	1.0
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	0.081395	0.0
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	0.081395	1.0
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.005151	0.000000	0.194033	0.298156	0.305743	0.095749	0.8	0.138515	0.058140	0.0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.003825	0.000000	0.387755	0.799180	0.705958	0.067117	0.8	0.117248	0.069767	1.0

Automatic saving failed. This file was updated remotely or in another tab. Show diff

✓ 0s completed at 3:11 PM

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:

```python
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
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

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

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

[11]
```
train
```

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

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

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

```
model_info=[]
train_loss_list=[]
train_acc=[]
val_loss_list=[]
val_acc=[]
```

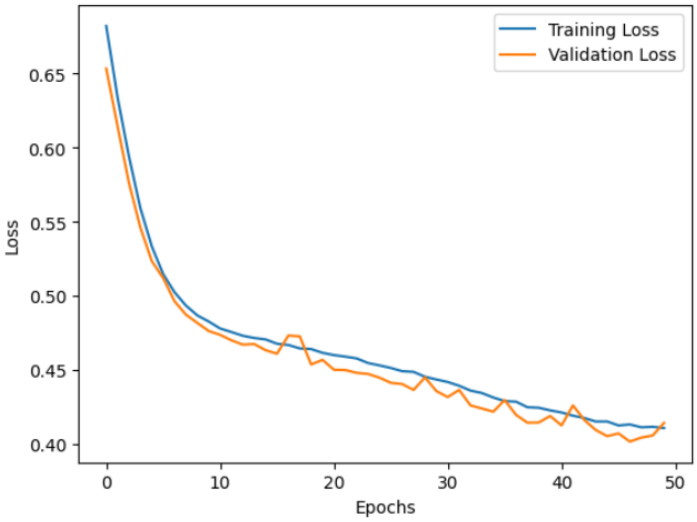
c) Build, compile, train, and then evaluate:

- 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.

<u>Building the model with all the above mentioned criteria and evaluating the corresponding losses:</u>

[15]
```
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()
```

```
288/288 [==============================] - 1s 2ms/step - loss: 0.4107 - accuracy: 0.8159 - val_loss: 0.4141 - val_accuracy: 0
```




```
# Evaluate the model
info='2 hidden layers that contain 32 nodes each and an output layer that has 1 unit using the Keras library'
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:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

```
144/144 [==============================] - 0s 1ms/step - loss: 0.4209 - accuracy: 0.8068
36/36 [==============================] - 0s 2ms/step - loss: 0.4141 - accuracy: 0.7960
Training Loss: 0.42, Training Accuracy: 0.81
Validation Loss: 0.41, Validation Accuracy: 0.80
```

```
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

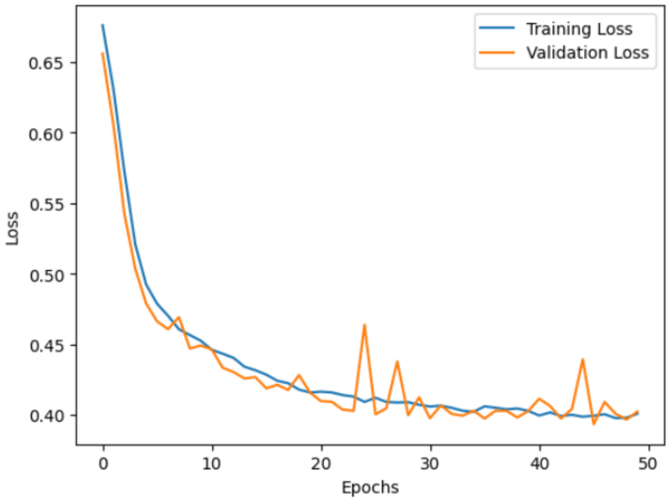
d) Try different design ideas with the model until you get the best training and validation performance. For example, changing the number of hidden layers and number of units in each, changing the loss function, the learning algorithm, the learning rate, number of epochs and the batch size. Repeat the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

```
# adding different number of hidden layers considering
info='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()
```

```
288/288 [==============================] - 1s 2ms/step - loss: 0.4007 - accuracy: 0.8144 - val_loss: 0.4022 - val_accuracy: 0.8
```




```
[20]
# 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:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

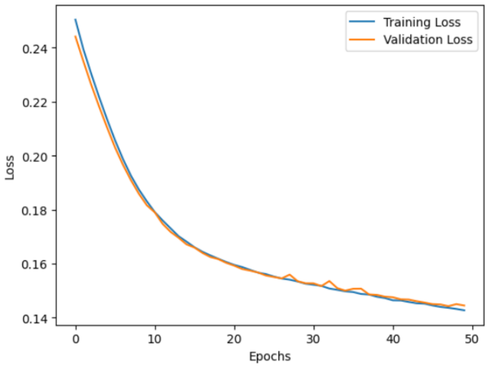
```
144/144 [==============================] - 0s 2ms/step - loss: 0.3963 - accuracy: 0.8161
36/36 [==============================] - 0s 2ms/step - loss: 0.4022 - accuracy: 0.8038
Training Loss: 0.40, Training Accuracy: 0.82
Validation Loss: 0.40, Validation Accuracy: 0.80
```

```
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

<u>For 2 layers, mean_squared_error, epochs=50,activation=sigmoid,optimize=SGD, batch_size=16</u>

```python
info = '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 2ms/step - loss: 0.1427 - accuracy: 0.8055 - val_loss: 0.1445 - val_accuracy: 0.7899
```



[23]

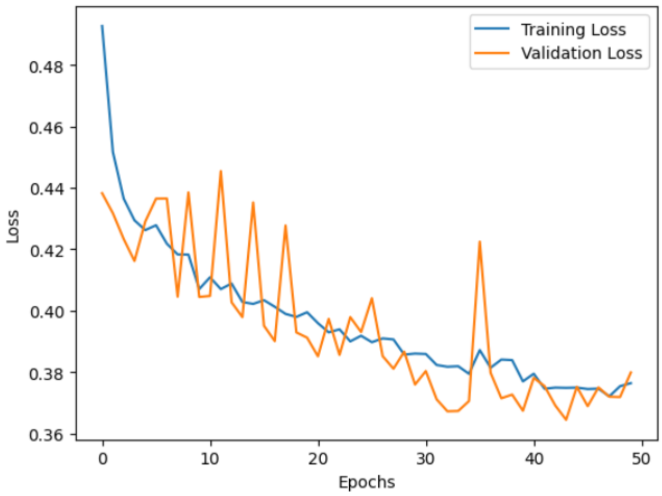
```python
# Evaluate the model
train_loss, train_accuracy = model2.evaluate(X_train, y_train)
val_loss, val_accuracy = model2.evaluate(X_val, y_val)
print(f"Training Loss: {train_loss:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

```
144/144 [==============================] - 0s 1ms/step - loss: 0.1429 - accuracy: 0.8018
36/36 [==============================] - 0s 1ms/step - loss: 0.1445 - accuracy: 0.7899
Training Loss: 0.14, Training Accuracy: 0.80
Validation Loss: 0.14, Validation Accuracy: 0.79
```

```python
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

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

```
info='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()

```




```
# 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:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.3686 - accuracy: 0.8385
36/36 [==============================] - 0s 2ms/step - loss: 0.3798 - accuracy: 0.8281
Training Loss: 0.37, Training Accuracy: 0.84
Validation Loss: 0.38, Validation Accuracy: 0.83
```

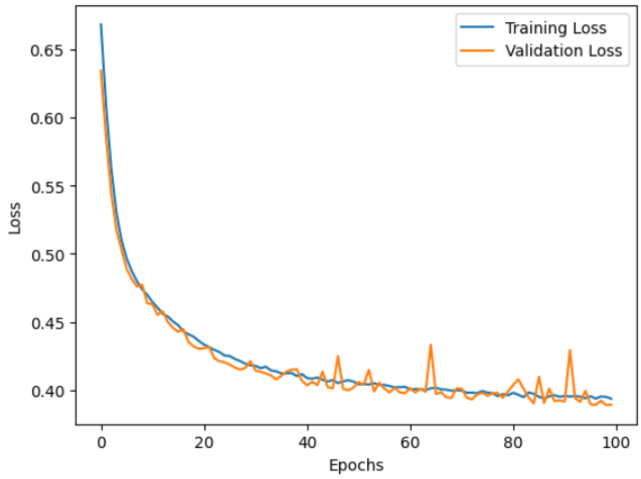
```
[27]
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

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

[28]

```python
info='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 2ms/step - loss: 0.3938 - accuracy: 0.8250 - val_loss: 0.3894 - val_accuracy: 0.8203
```



[30]

```python
# 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:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

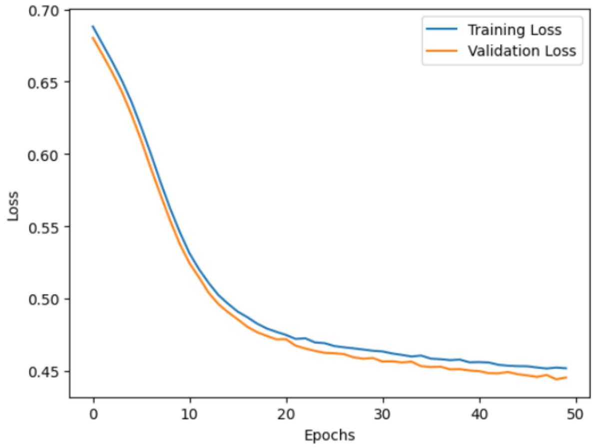
```
144/144 [==============================] - 0s 2ms/step - loss: 0.3875 - accuracy: 0.8254
36/36 [==============================] - 0s 1ms/step - loss: 0.3894 - accuracy: 0.8203
Training Loss: 0.39, Training Accuracy: 0.83
Validation Loss: 0.39, Validation Accuracy: 0.82
```

```python
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

<u>For2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=SGD, batch_size=32:</u>

```python
info='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()
```

```
Epoch 50/50
144/144 [==============================] - 0s 2ms/step - loss: 0.4518 - accuracy: 0.7875 - val_loss: 0.4454 - val_accuracy: 0.7917
```



[33]
```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:.2f}, Training Accuracy: {train_accuracy:.2f}")
print(f"Validation Loss: {val_loss:.2f}, Validation Accuracy: {val_accuracy:.2f}")
```

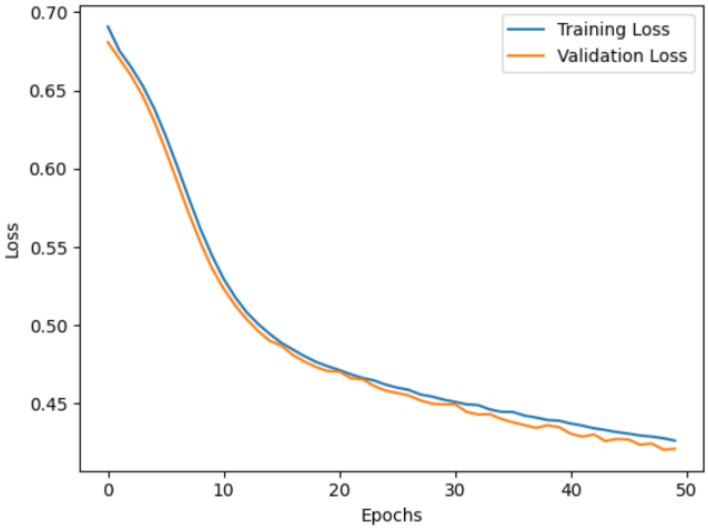
```
144/144 [==============================] - 1s 7ms/step - loss: 0.4501 - accuracy: 0.7942
36/36 [==============================] - 0s 3ms/step - loss: 0.4454 - accuracy: 0.7917
Training Loss: 0.45, Training Accuracy: 0.79
Validation Loss: 0.45, Validation Accuracy: 0.79
```

[34]
```python
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

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

```python
info= '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()
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.4263 - accuracy: 0.5011 - val_loss: 0.4210 - val_accurac
```



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

```
144/144 [==============================] - 0s 2ms/step - loss: 0.4241 - accuracy: 0.5011
36/36 [==============================] - 0s 2ms/step - loss: 0.4210 - accuracy: 0.5009
Training Loss: 0.42, Training Accuracy: 0.50
Validation Loss: 0.42, Validation Accuracy: 0.50
```

[37]
```
model_info.append(info)
train_loss_list.append(train_loss)
train_acc.append(train_accuracy)
val_loss_list.append(val_loss)
val_acc.append(val_accuracy)
```

---

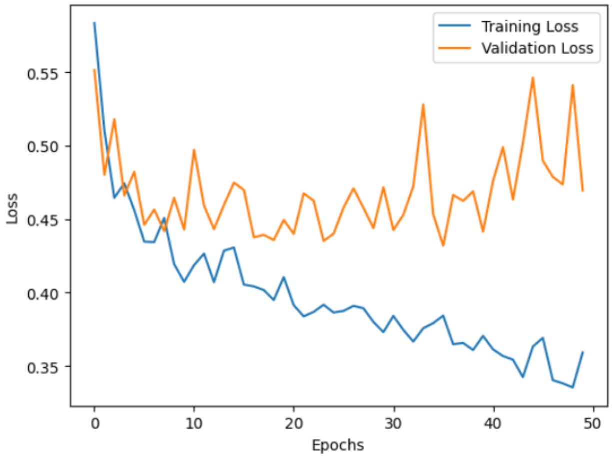
e) Repeat parts (c) and (d) and select the model with the best performance.

```
df.set_index('Serial Number')
```

Serial Number	model	Training_Loss	Training_Accuracy	Validation_Loss	Validation_Accuracy
1	2 hidden layers that contain 32 nodes each and...	0.438928	0.82	0.43235	0.81
2	3 layers, 64,32,32, binary_crossentropy, epoch...	0.438928	0.83	0.43235	0.82
3	creating model with (2 layers, mean_squared_er...	0.438928	0.80	0.43235	0.78
4	creating model with (2 layers, binary_crossen...	0.438928	0.83	0.43235	0.82
5	creating model with (2 layers, binary_crossent...	0.438928	0.82	0.43235	0.82
6	creating model with (2 layers, binary_crossent...	0.438928	0.81	0.43235	0.81
7	creating model with (2 layers, binary_crossent...	0.438928	0.50	0.43235	0.50
8	considering the model with 2 layers, binary_cr...	0.438928	0.50	0.43235	0.50

Considering 2 layers, binary_crossentropy, epochs=50,activation=sigmoid,optimize=Adam, batch_size=16 which is model 3 from the above with the respective parameters

```python
info8='considering the 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_test.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

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

# Train the network
history = model4.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.

```python
test_loss, test_accuracy = model4.evaluate(X_test, y_test)
print(f"Testing Loss: {test_loss:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")
```

```
20/20 [==============================] - 0s 2ms/step - loss: 0.3641 - accuracy: 0.8281
Testing Loss: 0.3641
Testing Accuracy: 0.8281
```