BUS306A Final Project

Mary Ardoin, Josh Rassin, Allyssa Utecht

Mount to Drive and Load Data

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

import pandas as pd
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
## load data
df = pd.read_csv("/content/drive/My Drive/BUS306A/data/spotify-2023.csv",
                 encoding='latin-1')
```

```
df_copy = df.copy()
df
```

	track_name	artist(s)_name	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_c
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	553	
1	LALA	Myke Towers	1	2023	3	23	1474	
2	vampire	Olivia Rodrigo	1	2023	6	30	1397	
3	Cruel Summer	Taylor Swift	1	2019	8	23	7858	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	3133	
...	...	...	...	...	...	...	...	
948	My Mind & Me	Selena Gomez	1	2022	11	3	953	
949	Bigger Than The Whole Sky	Taylor Swift	1	2022	10	21	1180	
950	A Veces (feat. Feid)	Feid, Paulo Londra	2	2022	11	3	573	
951	En La De Ella	Feid, Sech, Jhayco	3	2022	10	20	1320	
952	Alone	Burna Boy	1	2022	11	4	782	

953 rows × 24 columns

Load Libraries

```
## import function for logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
## import function to split traning and testing datasets
from sklearn.model_selection import train_test_split
## import function to code categorical data
from sklearn.preprocessing import LabelEncoder
## import evaluation metrics
from sklearn import metrics
import matplotlib.pyplot as plt
from math import exp
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential, Model
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import layers
```

Question 2

Describe your data (e.g., how data were collected, the meaning of each column and each row), provide descriptive statistics of the data (e.g., the number of classes in labels and major categorical variables, the mean and standard deviation of major quantitative variables, the distribution of text length, etc.) and reflect on the potential biases and limitations of the data.

```
df.shape
```

```
(953, 24)
```

```
list(df.columns)
```

```
['track_name',
 'artist(s)_name',
 'artist_count',
 'released_year',
 'released_month',
 'released_day',
 'in_spotify_playlists',
 'in_spotify_charts',
 'streams',
 'in_apple_playlists',
 'in_apple_charts',
 'in_deezer_playlists',
 'in_deezer_charts',
 'in_shazam_charts',
 'bpm',
 'key',
 'mode',
 'danceability_%',
 'valence_%',
 'energy_%',
 'acousticness_%',
 'instrumentalness_%',
 'liveness_%',
 'speechiness_%']
```

```
df.head()
```

	track_name	artist(s)_name	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_cha
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	553	
1	LALA	Myke Towers	1	2023	3	23	1474	
2	vampire	Olivia Rodrigo	1	2023	6	30	1397	
3	Cruel Summer	Taylor Swift	1	2019	8	23	7858	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	3133	

5 rows × 24 columns

```
df.describe()
```

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts	in_apple_playlists
count	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000
mean	1.556139	2018.238195	6.033578	13.930745	5200.124869	12.009444	67.812172
std	0.893044	11.116218	3.566435	9.201949	7897.608990	19.575992	86.441493
min	1.000000	1930.000000	1.000000	1.000000	31.000000	0.000000	0.000000
25%	1.000000	2020.000000	3.000000	6.000000	875.000000	0.000000	13.000000
50%	1.000000	2022.000000	6.000000	13.000000	2224.000000	3.000000	34.000000
75%	2.000000	2022.000000	9.000000	22.000000	5542.000000	16.000000	88.000000
max	8.000000	2023.000000	12.000000	31.000000	52898.000000	147.000000	672.000000

```
df.dtypes
```

```
track_name             object
artist(s)_name         object
artist_count            int64
released_year           int64
released_month          int64
released_day            int64
in_spotify_playlists    int64
in_spotify_charts       int64
streams                object
in_apple_playlists      int64
in_apple_charts         int64
in_deezer_playlists    object
in_deezer_charts        int64
in_shazam_charts       object
bpm                     int64
key                    object
mode                   object
danceability_%          int64
valence_%               int64
energy_%                int64
acousticness_%          int64
instrumentalness_%      int64
liveness_%              int64
speechiness_%           int64
dtype: object
```

Question 3

Preprocess your data (e.g., drop missing values, convert variables in the desired format, clean texts, etc.), select the correct X and y variables, and split them into training and testing sets.

```
# Transform the categorical (object) columns into integers
encoder = LabelEncoder()
df2 = df.select_dtypes(include='object').apply(encoder.fit_transform)

# Merge the categorical columns to the integer/float columns
df = pd.concat([df2, df_copy.select_dtypes(exclude='object')], axis = 1)

print(df.dtypes)
```

```
track_name             int64
artist(s)_name         int64
streams                int64
in_deezer_playlists    int64
in_shazam_charts       int64
key                    int64
mode                   int64
artist_count           int64
released_year          int64
released_month         int64
released_day           int64
in_spotify_playlists   int64
in_spotify_charts      int64
in_apple_playlists     int64
in_apple_charts        int64
in_deezer_charts       int64
```

```
bpm                   int64
danceability_%        int64
valence_%             int64
energy_%              int64
acousticness_%        int64
instrumentalness_%    int64
liveness_%            int64
speechiness_%         int64
dtype: object
```

```
print(df.isna().sum())
print(df.shape)
```

```
track_name            0
artist(s)_name        0
streams               0
in_deezer_playlists   0
in_shazam_charts      0
key                   0
mode                  0
artist_count          0
released_year         0
released_month        0
released_day          0
in_spotify_playlists  0
in_spotify_charts     0
in_apple_playlists    0
in_apple_charts       0
in_deezer_charts      0
bpm                   0
danceability_%        0
valence_%             0
energy_%              0
acousticness_%        0
instrumentalness_%    0
liveness_%            0
speechiness_%         0
dtype: int64
(953, 24)
```

```
df = df.dropna()
print(df.shape)
```

```
(953, 24)
```

```
df.head()
```

	track_name	artist(s)_name	streams	in_deezer_playlists	in_shazam_charts	key	mode	artist_count	released_year	released
0	687	326	156	217	185	2	0	2	2023	
1	397	401	125	262	118	3	0	1	2023	
2	936	431	151	337	193	7	0	1	2023	
3	170	558	864	50	149	0	0	1	2019	
4	864	43	499	330	125	0	1	1	2023	

5 rows × 24 columns

```
sorted_df = df.reset_index()
df = df.sort_values(by=['in_spotify_charts'], ascending=[False])
df = df.reset_index(drop=True)
df['top_countdown'] = range(1, len(sorted_df) + 1)
```

```
# Create top 100 indicator
df['top_100'] = df['top_countdown'].apply(lambda x: 1 if x <= 100 else 0)
```

```
df.head()
```

	track_name	artist(s)_name	streams	in_deezer_playlists	in_shazam_charts	key	mode	artist_count	released_year	released
0	687	326	156	217	185	2	0	2	2023	
1	60	223	423	329	198	8	1	1	2022	
2	277	395	121	308	2	11	0	1	2023	
3	936	431	151	337	193	7	0	1	2023	
4	351	25	114	264	174	11	1	1	2013	

5 rows × 26 columns

```
df['top_100'].value_counts()
```

```
top_100
0    853
1    100
Name: count, dtype: int64
```

```
weights = {0:10., 1:90.}
weights = dict(weights)
```

```
cols_to_use = ['artist_count',
 'released_year',
 'released_month',
 'released_day',
 'in_spotify_playlists',
 'bpm',
 'danceability_%',
 'valence_%',
 'energy_%',
 'acousticness_%',
 'instrumentalness_%',
 'liveness_%',
 'speechiness_%']
X = df[cols_to_use].values
y = df[['top_100']].values
```

```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle = True, random_state = 123)
```

Question 4

Randomly sample 500 data points from your original data and show the classes (if your y is quantitative, group them into 2 or 3 categories such as low, medium, and high) of the sample in a 2D space with PCA

```
model_df = df[cols_to_use + ['top_100']].copy()
```

```
model_df = np.array(model_df, dtype = 'float32')
model_df
```

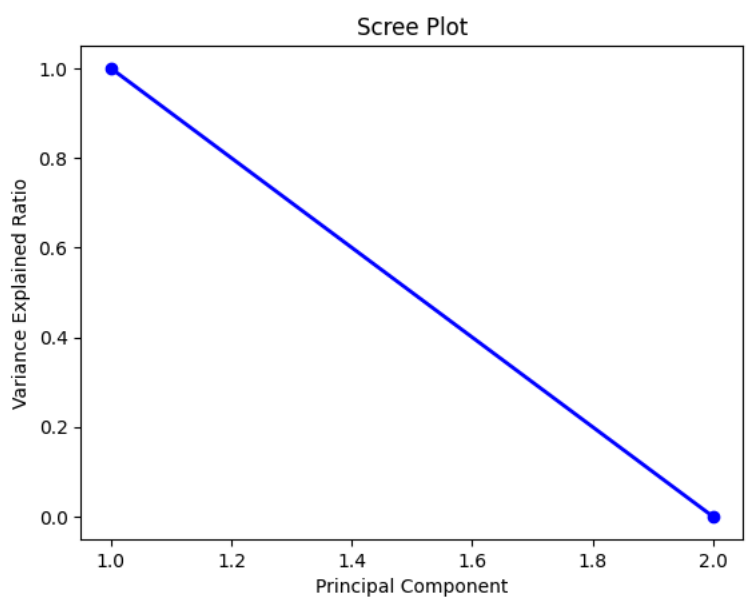
```
array([[2.000e+00, 2.023e+03, 7.000e+00, ..., 8.000e+00, 4.000e+00,
        1.000e+00],
       [1.000e+00, 2.022e+03, 3.000e+00, ..., 3.100e+01, 6.000e+00,
        1.000e+00],
       [1.000e+00, 2.023e+03, 1.000e+00, ..., 3.000e+00, 7.000e+00,
        1.000e+00],
       ...,
       [1.000e+00, 2.022e+03, 2.000e+00, ..., 1.200e+01, 4.000e+00,
        0.000e+00],
       [1.000e+00, 1.952e+03, 1.000e+00, ..., 1.500e+01, 3.000e+00,
        0.000e+00],
       [3.000e+00, 1.958e+03, 1.000e+00, ..., 1.500e+01, 5.000e+00,
        0.000e+00]], dtype=float32)
```

```
# Select X and y, and rescale the value of each pixel to 0-1 range
X_train = model_df[:, -2:]
y_train = model_df[:, -1]
```

```
X_test = model_df[:, -2:]
y_test = model_df[:, -1]


#Initialize the PCA model
pca = PCA(2)

#Transform features into PCA features
X_train_pca = pca.fit_transform(X)
X_train_pca
```

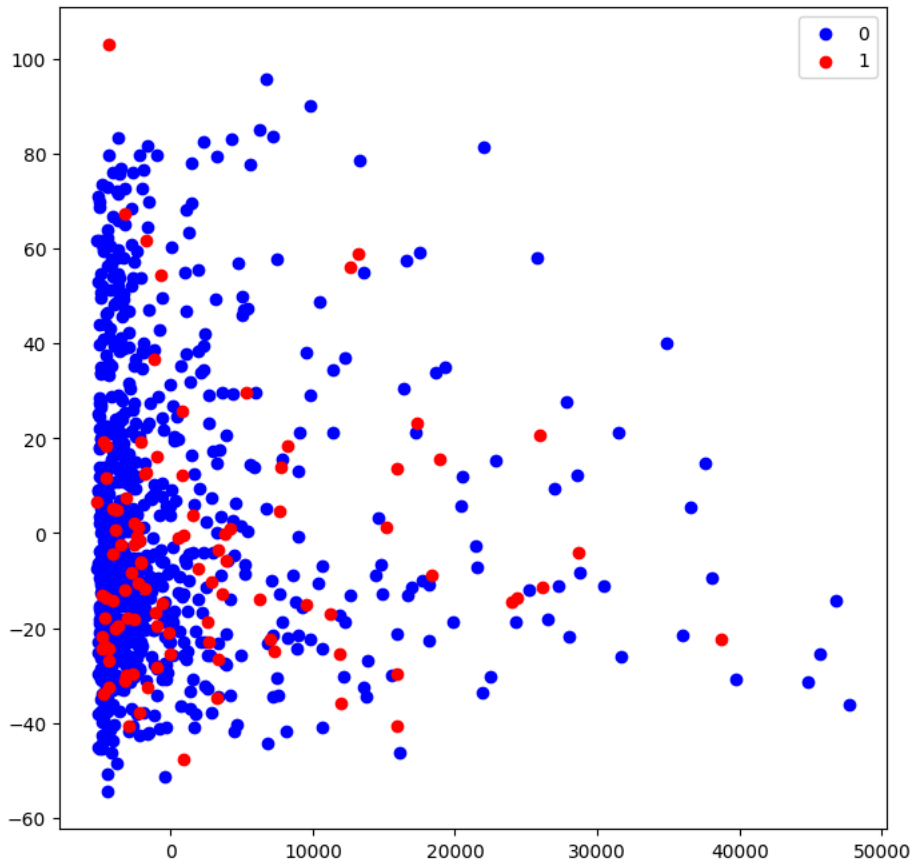
```
array([[-4.64712968e+03, -2.37596486e+01],
       [ 1.83748647e+04, -8.80360573e+00],
       [ 7.01087641e+03, -2.23760929e+01],
       ...,
       [-2.48912085e+03, -2.97701471e+01],
       [ 2.72990068e+03,  2.91008731e+01],
       [ 1.08989571e+03,  4.68338438e+01]])
```

```
# scree plot
PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained Ratio')
plt.show()
```




```
# Convert data into a dataframe
pca_df = pd.DataFrame(X_train_pca, columns=['PC1', 'PC2'])
label_df = pd.DataFrame(y_train, columns=['label'])
pca_df = pd.concat([pca_df, label_df], axis = 1)


# Seperate label 0 and 1
pca_df0 = pca_df[pca_df['label']==0]
pca_df1 = pca_df[pca_df['label']==1]


# Plot data in a 2D space
plt.figure(figsize = (8,8))
plt.scatter(pca_df0['PC1'], pca_df0['PC2'], c = 'blue')
plt.scatter(pca_df1['PC1'], pca_df1['PC2'], c = 'red')
plt.legend([0,1])
```

```
<matplotlib.legend.Legend at 0x7f7cab059060>
```



Question 5

Create a simple model such as linear regression, logistic regression, naive Bayes, or decision tree to classify your data (the training and testing data from your original dataset; not the PCA components), and report and interpret results.

```
X = df[cols_to_use].values
y = df[['top_100']].values

# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle = True, random_state = 123)


# train model and balance values
model = DecisionTreeClassifier(class_weight= weights)
model.fit(X_train,y_train.ravel())
```

```
DecisionTreeClassifier(class_weight={0: 10.0, 1: 90.0})
```

```
# get predicted value of training data
y_train_pred = model.predict(X_train)
# get predicted value of testing data
y_test_pred = model.predict(X_test)


## report traning results
print(metrics.classification_report(y_train, y_train_pred, digits = 3))
```

```
              precision    recall  f1-score   support

           0      1.000     1.000     1.000       682
           1      1.000     1.000     1.000        80

    accuracy                          1.000       762
   macro avg      1.000     1.000     1.000       762
```

```
weighted avg      1.000     1.000     1.000       762
```

```
# report testing results
print(metrics.classification_report(y_test, y_test_pred, digits = 3))
```

```
              precision    recall  f1-score   support

           0      0.906     0.953     0.929       171
           1      0.273     0.150     0.194        20

    accuracy                          0.869       191
   macro avg      0.589     0.552     0.561       191
weighted avg      0.839     0.869     0.852       191
```

Question 6

Use the 5-fold cross-validation method with the same model to verify the accuracy and F1 score (R-squared score for regression) you obtained from the above step.

```
k = 10 # number of subsets
kf = KFold(n_splits=k, shuffle=True, random_state=123) # initialize KFold function
model = DecisionTreeClassifier(class_weight = weights) # initialize DecisionTreeClassifier() function
```

```
F1_scores = [] # create an empty list to store the F1 score of each fold
acc_scores = [] # create an empty list to store the accuracy score of each fold

# loop 10 times, create training and testing datasets in each time, fit the model, and produces results
for train_index , test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index] # create training and testing datasets
    y_train, y_test = y[train_index], y[test_index]

    model.fit(X_train,y_train) # fit the model
    y_test_pred = model.predict(X_test) # get predicted y values of the testing dataset

    f1 = metrics.f1_score(y_test, y_test_pred) # calcualte the F1 score of the testing data
    acc = metrics.accuracy_score(y_test, y_test_pred) # calcualte the accuracy score of the testing data
    F1_scores.append(f1)
    acc_scores.append(acc)
```

```
# print average F1 scores and accuracy scores
print(sum(F1_scores)/k)
print(sum(acc_scores)/k)
```

```
0.2952836715713954
0.8730811403508772
```

Question 7

Create a deep learning model, draw a graph to show the structure of the model, and report and interpret the results.

```
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras.utils import plot_model
from sklearn.utils import class_weight
```

```
X = df[cols_to_use].values
y = df[['top_100']].values
```

```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle = True, random_state = 123)
```

```
y_train_dummy = to_categorical(y_train)
y_test_dummy = to_categorical(y_test)


model = Sequential() # initialize a sequential model
model.add(Dense(512, activation = 'relu', input_shape = (X_train.shape[1],))) # first layer: 512 neurons
model.add(Dense(256, activation = 'relu'))
model.add(Dense(2, activation = 'sigmoid'))


# view the model structure
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 512)               7168

 dense_7 (Dense)             (None, 256)               131328

 dense_8 (Dense)             (None, 2)                 514

=================================================================
Total params: 139010 (543.01 KB)
Trainable params: 139010 (543.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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


# Checkpoint for the best model
modelpath="/content/drive/My Drive/BUS306A/models/bestmodel"

checkpoint = keras.callbacks.ModelCheckpoint(modelpath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')

# Callbacks for early stop
callbacks = [keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', patience=5),
             keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=1e-4, patience=5),
             checkpoint]


# Train the model
history=model.fit(X_train, y_train_dummy,
                  batch_size=128,
                  epochs=16,
                  validation_split=0.2,
                  verbose = 1,
                  callbacks=callbacks,
                  class_weight = weights)
```

```
Epoch 1/16
5/5 [==============================] - ETA: 0s - loss: 3509.9001 - accuracy: 0.6535
Epoch 1: val_accuracy improved from -inf to 0.29412, saving model to /content/drive/My Drive/BUS306A/models/bestmodel
5/5 [==============================] - 4s 635ms/step - loss: 3509.9001 - accuracy: 0.6535 - val_loss: 366.3411 - val_accurac
Epoch 2/16
1/5 [=====>........................] - ETA: 0s - loss: 2000.4602 - accuracy: 0.2188
Epoch 2: val_accuracy improved from 0.29412 to 0.45752, saving model to /content/drive/My Drive/BUS306A/models/bestmodel
5/5 [==============================] - 1s 178ms/step - loss: 2873.7209 - accuracy: 0.3859 - val_loss: 295.8597 - val_accurac
Epoch 3/16
1/5 [=====>........................] - ETA: 0s - loss: 2687.5354 - accuracy: 0.5000
Epoch 3: val_accuracy improved from 0.45752 to 0.86928, saving model to /content/drive/My Drive/BUS306A/models/bestmodel
5/5 [==============================] - 1s 183ms/step - loss: 1519.2292 - accuracy: 0.6420 - val_loss: 35.5176 - val_accuracy
Epoch 4/16
1/5 [=====>........................] - ETA: 0s - loss: 1082.6780 - accuracy: 0.9141
Epoch 4: val_accuracy did not improve from 0.86928
5/5 [==============================] - 0s 16ms/step - loss: 1492.4741 - accuracy: 0.8571 - val_loss: 102.5285 - val_accuracy
Epoch 5/16
1/5 [=====>........................] - ETA: 0s - loss: 1022.4230 - accuracy: 0.5078
Epoch 5: val_accuracy did not improve from 0.86928
5/5 [==============================] - 0s 18ms/step - loss: 833.3513 - accuracy: 0.4992 - val_loss: 140.3423 - val_accuracy:
Epoch 6/16
1/5 [=====>........................] - ETA: 0s - loss: 963.1713 - accuracy: 0.1953
Epoch 6: val_accuracy did not improve from 0.86928
5/5 [==============================] - 0s 16ms/step - loss: 1108.9703 - accuracy: 0.4384 - val_loss: 89.6145 - val_accuracy:
Epoch 7/16
1/5 [=====>........................] - ETA: 0s - loss: 535.8632 - accuracy: 0.6641
```

```
Epoch 7: val_accuracy did not improve from 0.86928
5/5 [==============================] - 0s 15ms/step - loss: 926.8512 - accuracy: 0.6010 - val_loss: 100.9338 - val_accuracy:
Epoch 8/16
1/5 [=====>.....................] - ETA: 0s - loss: 1021.5978 - accuracy: 0.7266
Epoch 8: val_accuracy did not improve from 0.86928
5/5 [==============================] - 0s 20ms/step - loss: 932.8721 - accuracy: 0.4532 - val_loss: 64.6423 - val_accuracy:
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test_dummy)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
Exception ignored in: <function _xla_gc_callback at 0x7f7cc64d1510>
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/jax/_src/lib/__init__.py", line 98, in _xla_gc_callback
    def _xla_gc_callback(*args):
KeyboardInterrupt:
6/6 [==============================] - 0s 3ms/step - loss: 46.5853 - accuracy: 0.6440
Test Loss: 46.585262298583984, Test Accuracy: 0.6439790725708008
```

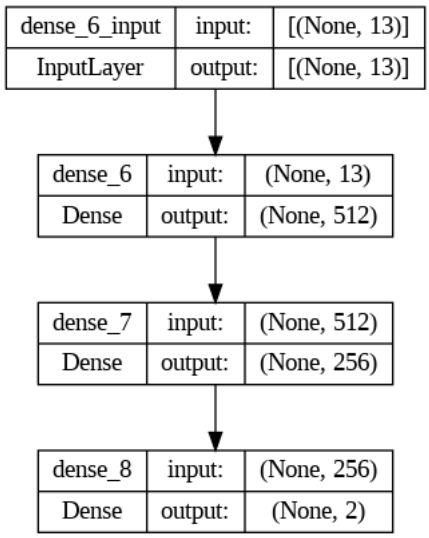
```
print(model.summary())

!echo $modelpath
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 512)               7168

 dense_7 (Dense)             (None, 256)               131328

 dense_8 (Dense)             (None, 2)                 514

=================================================================
Total params: 139010 (543.01 KB)
Trainable params: 139010 (543.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
/content/drive/My Drive/BUS306A/models/bestmodel
```

```
for layer in model.layers:
    weights = layer.get_weights()
    for weight in weights:
        print(f"Weight shape: {weight.shape}")

!ls -l $modelpath
```

```
Weight shape: (13, 512)
Weight shape: (512,)
Weight shape: (512, 256)
Weight shape: (256,)
Weight shape: (256, 2)
Weight shape: (2,)
ls: cannot access '/content/drive/My': No such file or directory
ls: cannot access 'Drive/BUS306A/models/bestmodel': No such file or directory
```

```
#Graph structure of model
plot_model(model, show_shapes=True)
```




```
# Load the best model
model.load_weights(modelpath)
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7f7c967e3b80>
```

```
# Make predictions
y_test_pred = model.predict(X_test)
```

```
6/6 [==============================] - 0s 4ms/step
```

```
y_test_pred[0:5]
```

```
array([[1.0000000e+00, 0.0000000e+00],
       [1.0000000e+00, 0.0000000e+00],
       [4.4706266e-02, 0.0000000e+00],
       [3.7814750e-08, 0.0000000e+00],
       [4.5936024e-10, 0.0000000e+00]], dtype=float32)
```

```
# Transform predicted probabilities into labels
y_test_pred_label = y_test_pred.argmax(axis=-1)
```

```
# Accuracy on the testing dataset
metrics.accuracy_score(y_true=y_test, y_pred=y_test_pred_label)
```

```
0.8952879581151832
```

Question 8

Use OpenAI's API to do another task (e.g., few-shot classification, getting text embedding and feeding into a classification model, generating features from texts and feeding into a classification model, etc.)

```
!pip install openai
```

```
Collecting openai
  Downloading openai-1.19.0-py3-none-any.whl (292 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 292.8/292.8 kB 4.2 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 4.3 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from openai) (2.6.4)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.2)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from openai) (4.11.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (3.6)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (1.2
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->openai) (2024.2.2)
Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai)
```

```
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 4.6 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 4.2 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->o
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->op
Installing collected packages: h11, httpcore, httpx, openai
Successfully installed h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 openai-1.19.0
```

```
from openai import OpenAI
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from google.colab import drive

drive.mount('/content/drive')
```

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

```
client = OpenAI(api_key="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")


df_train, df_test = train_test_split(df, test_size=0.2, random_state=123)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-6fff693cf364> in <cell line: 1>()
----> 1 df_train, df_test = train_test_split(df, test_size=0.2, random_state=123)

NameError: name 'df' is not defined
```

```
responses = []

for idx, row in df_test.iloc[0:20,:].iterrows():
    # create features
    features = 'Artist count: '+str(row['artist_count'])+', Released year: '+str(row['released_year'])+', Released month: '+str(
    df_example = df_train.sample(n=4)
    examples = []
    for idx, row in df_example.iterrows():
        example = 'Artist count: '+str(row['artist_count'])+', Released year: '+str(row['released_year'])+', Released month: '+s
        examples.append(example)
    examples ='\n'.join(examples)
    # create prompt
    prompt = f"""
    Learn how select factors influence whether a song is a top 100 song from the following examples:
    {examples}
    What is the probability a song is a top 100 song with the following statistics? Output a number based on your best guess. Th
    {features}
    Top 100:
    """
    print(prompt)
    # feed the prompt into the model and get its response
```