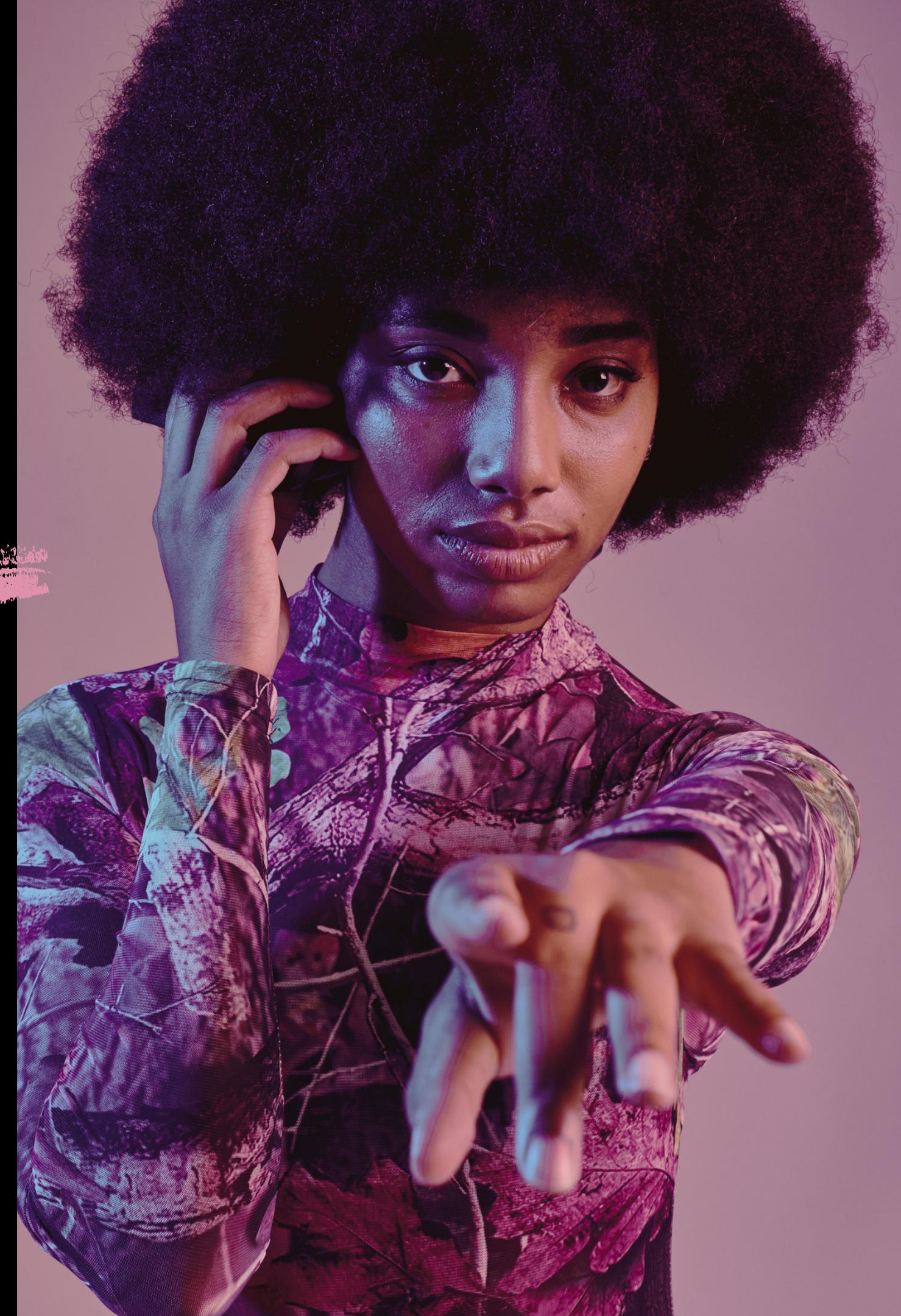


# KNN MUSIC GENRE CLASSIFICATION

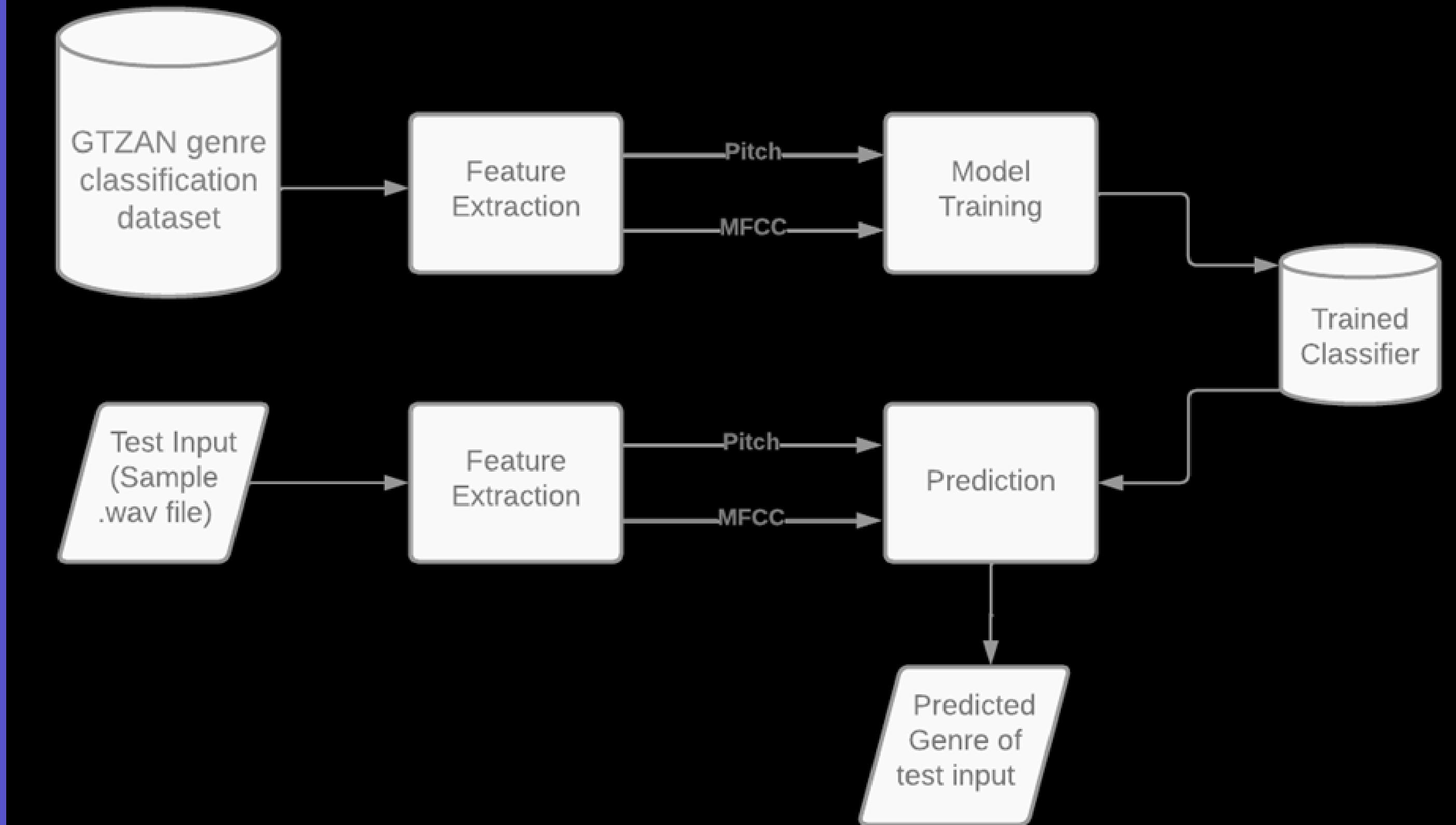
Daniel Cont



Obj

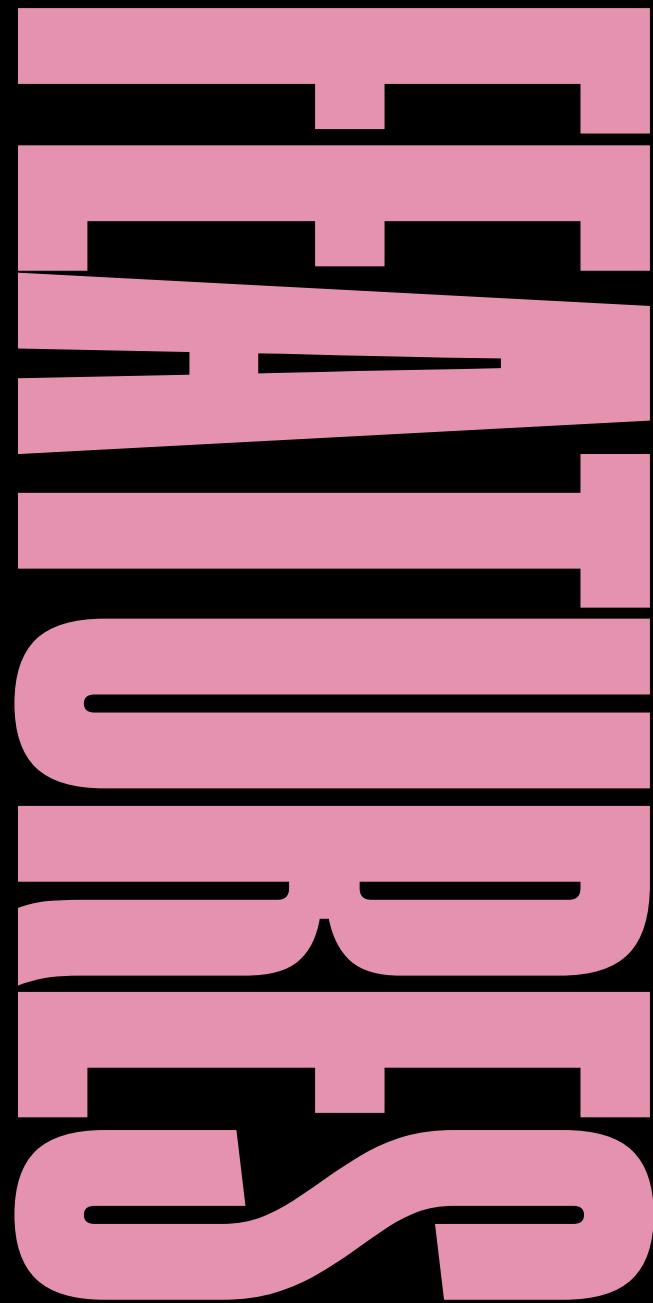


- 01/ Modelo.
- 02/ Extracción de características.
- 03/ Entrenamiento del modelo.
- 04/ Predicciones.
- 05/ Implementación.
- 06/ Resultados y discusión.
- 07/ Conclusiones.
- 08/ Bibliografía



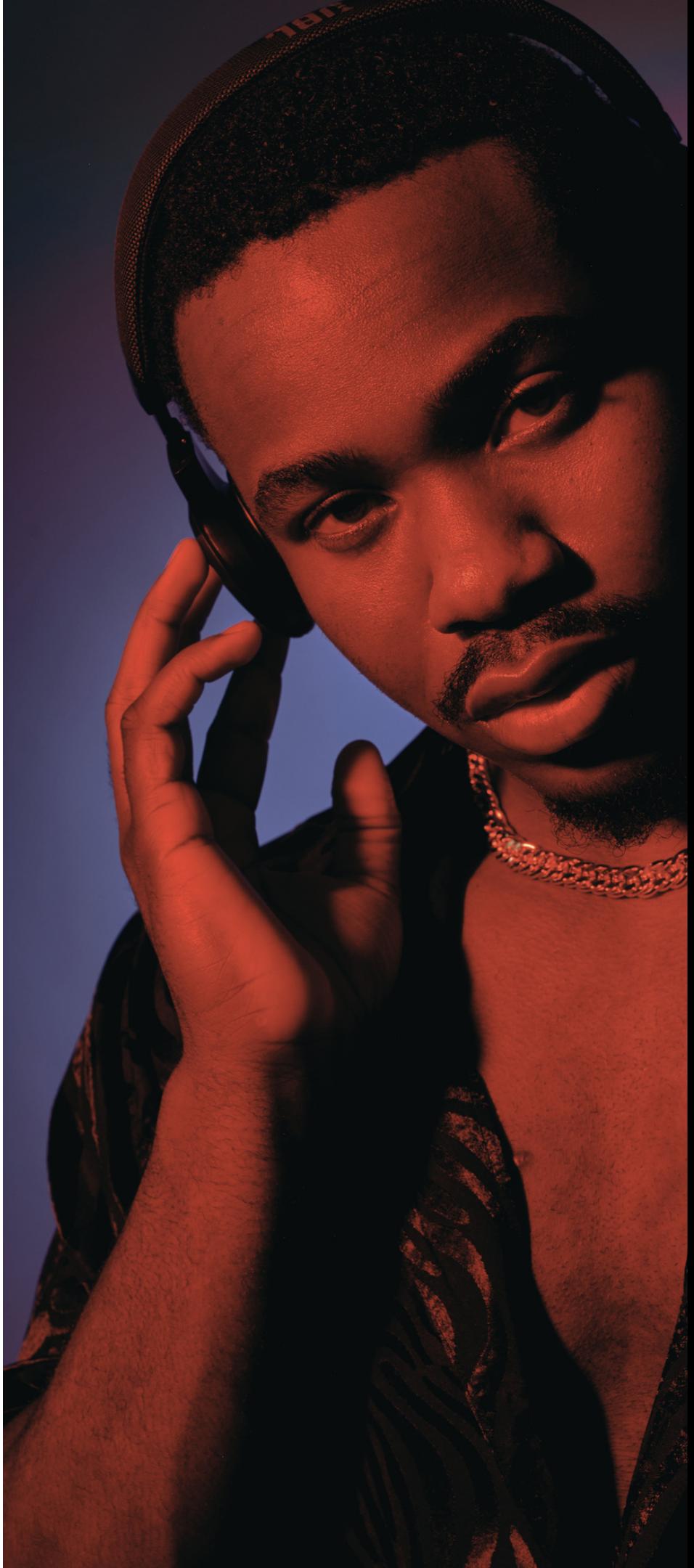
1. Las señales de audio se dividen en cuadros más pequeños.
2. Se identifican las diferentes frecuencias presentes en cada cuadro.
3. Se separan las frecuencias lingüísticas del ruido.
4. Se toma la transformada discreta de coseno (DCT) de estas frecuencias para descartar el ruido, manteniendo solo una secuencia específica de frecuencias que tienen una alta probabilidad de información útil.

## Extracción de características



# ENTRENAMIENTO DEL MODELO

El algoritmo K-vecinos (KNN): Se trata de uno de los algoritmos de aprendizaje automático más sencillos basados en la técnica de aprendizaje supervisado. Se asume la similitud entre el nuevo caso/datos y los casos disponibles y se clasifica el nuevo caso en la categoría más similar a las categorías disponibles.



# Inteligencia Artificial



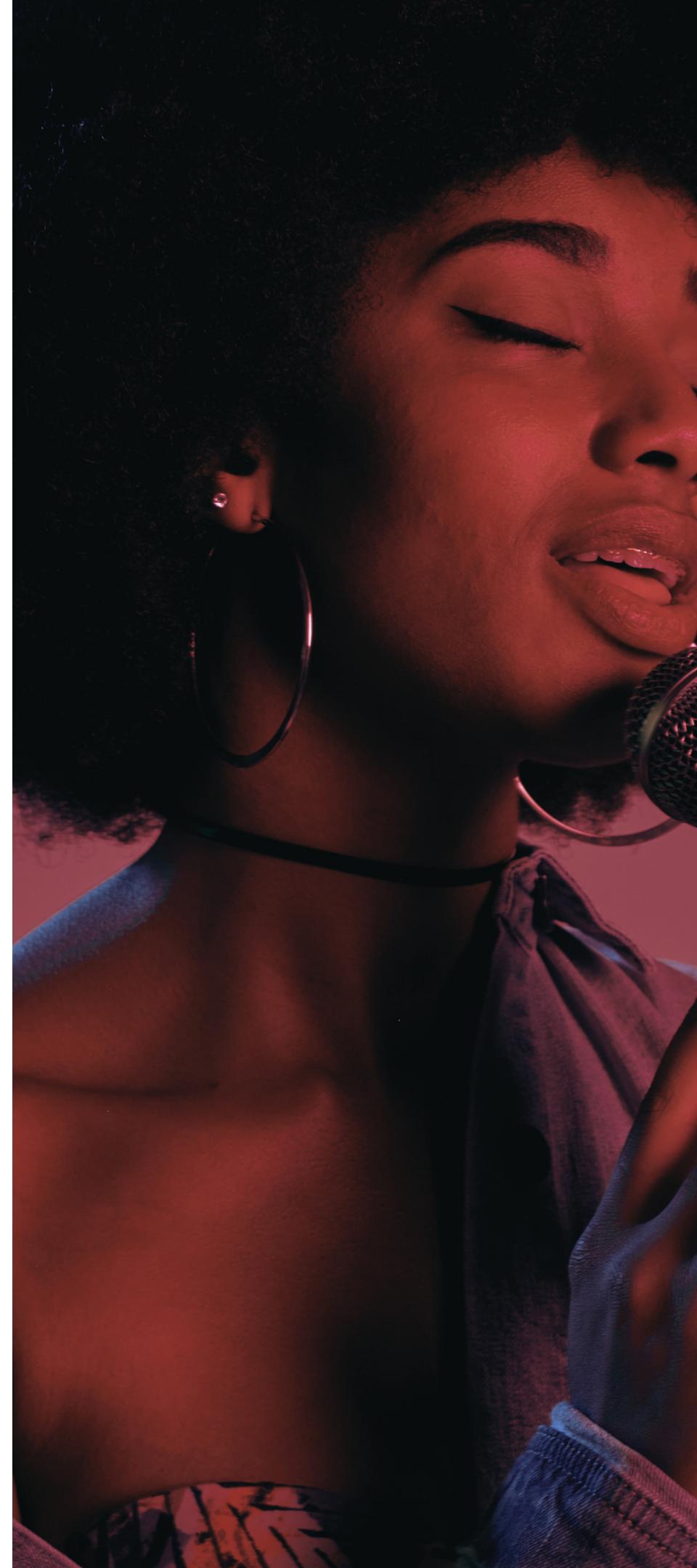
Este paso implica el uso de datos de prueba para probar y predecir el género del archivo de música de muestra en comparación con el modelo entrenado y obtener parámetros de precisión.

## 1. Imports:

```
from python_speech_features import mfcc  
import scipy.io.wavfile as wav  
import numpy as np  
from tempfile import TemporaryFile  
import os  
import pickle  
import random  
import operator  
import math  
import numpy as np
```

## 2. Definición de la función para encontrar vecinos:

```
def getNeighbors(trainingSet, instance, k):  
    distances = []  
    for x in range(len(trainingSet)):  
        dist = distance(trainingSet[x], instance, k )+  
distance(instance, trainingSet[x], k)  
        distances.append((trainingSet[x][2], dist))  
    distances.sort(key=operator.itemgetter(1))  
    neighbors = []  
    for x in range(k):  
        neighbors.append(distances[x][0])  
    return neighbors
```



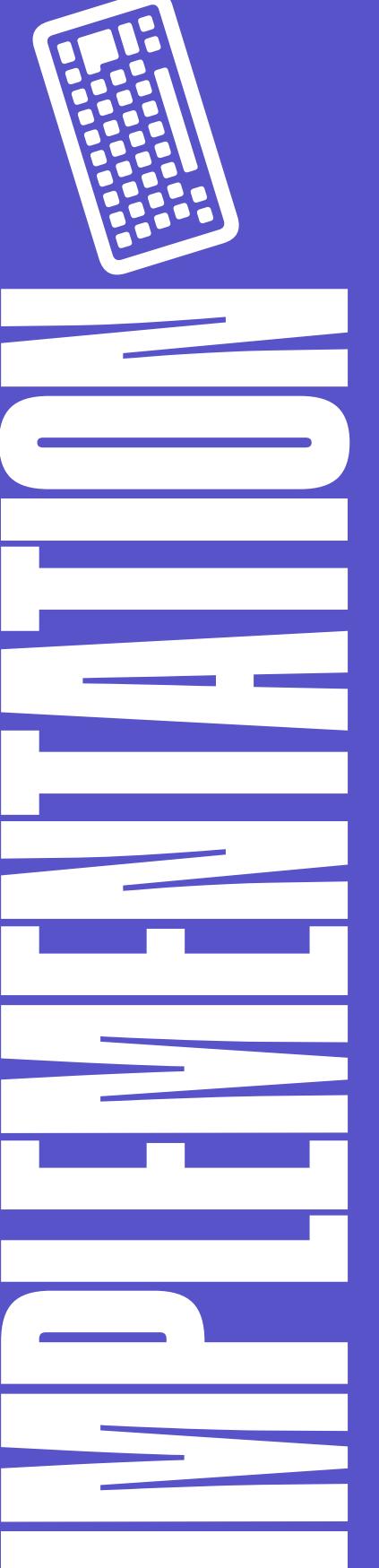
### 3. Identificación de los vecinos más cercanos.

```
def nearestClass(neighbors):  
    classVote = {}  
    for x in range(len(neighbors)):  
        response = neighbors[x]  
        if response in classVote:  
            classVote[response]+=1  
        else:  
            classVote[response]=1  
  
    sorter = sorted(classVote.items(), key =  
operator.itemgetter(1), reverse=True)  
    return sorter[0][0]
```

### 4. Función para la evaluación del modelo y las métricas de rendimiento:

```
def getAccuracy(testSet, predictions):  
    correct = 0  
    for x in range (len(testSet)):  
        if testSet[x][-1]==predictions[x]:  
            correct+=1  
  
    return 1.0*correct/len(testSet)
```





## 5. Implementación de dependencias, es decir, el conjunto de datos de locales.

```
directory = 'genres_original/'  
f= open("my.dat" , 'wb')
```

## 6. Función para la evaluación del modelo y las métricas de rendimiento:

```
for folder in os.listdir(directory):  
    i+=1  
    if i==10 :  
        break  
    for file in os.listdir(directory+folder):  
        (rate,sig) = wav.read(directory+folder+"/"+file)  
        mfcc_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy =  
False) #(first my.dat)  
        # mfcc_feat = mfcc(sig,rate ,winlen=0.020, nfft=1024,  
nfilt=25, appendEnergy = False)  
        covariance = np.cov(np.matrix.transpose(mfcc_feat))  
        mean_matrix = mfcc_feat.mean(0)  
        feature = (mean_matrix , covariance , i)  
        pickle.dump(feature , f)  
  
f.close()
```



## 7. División de entrenamiento y prueba en el conjunto de datos.

```
dataset = []
def loadDataset(filename, split, trset, teset):
    with open('my.dat','rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break
    for x in range(len(dataset)):
        if random.random() < split:
            trset.append(dataset[x])
        else:
            teset.append(dataset[x])
trainingSet = []
testSet = []
loadDataset('my.dat', 0.68, trainingSet, testSet)
```

## 8. Predicciones

```
leng = len(testSet)
predictions = []
for x in range (leng):
    predictions.append(nearestClass(getNeighbors(trainingSet
,testSet[x] , 7)))
accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)
```





```
for folder in os.listdir("/content/drive/MyDrive/Colab_Notebooks/genres_train"):
    results[i]=folder
    i+=1

print(results)

(rate,sig)=wav.read("/content/drive/MyDrive/Colab_Notebooks/genres_test/12470-1-16000.wav")
mfcc_feat=mfcc(sig,rate,winlen=0.020,appendEnergy=False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature=(mean_matrix,covariance,0)

pred=nearestClass(getNeighbors(dataset ,feature , 5))

print("result: "+results[pred])
```

In [15]:

```
leng = len(testSet)
predictions = []
for x in range (leng):
    predictions.append(nearestClass(g

accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)
```

0.6835820895522388

```
PS C:\Users\Cont2\Desktop\ProySIC> & C:/Users/Cont2/Desktop/Python/Python310/python.exe c:/Users/Cont2/Desktop/ProySIC/main.py
0.8378378378378378
PS C:\Users\Cont2\Desktop\ProySIC>
```

# RESULTADOS Y DISCUSIÓN