Check In #1:

Goal: Predicting the genre of a track

Our team has chosen the Hugging Face Spotify Tracks dataset.

The key features we have decided to study are **popularity, duration, danceability, energy, key, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo**, track genre(for training) because these would be most useful for genre prediction.

In [1]:
```
%pip install datasets
%pip install plotly
%pip install matplotlib
%pip install seaborn
%pip install datascience
%pip install scikit-learn
```

```
Requirement already satisfied: datasets in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (3.0.2)
Requirement already satisfied: filelock in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (3.16.1)
Requirement already satisfied: numpy>=1.17 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (2.0.2)
Requirement already satisfied: pyarrow>=15.0.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (17.0.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (2.2.3)
Requirement already satisfied: requests>=2.32.2 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (4.66.5)
Requirement already satisfied: xxhash in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from fsspec[http]<=2024.9.0,>=2023.1.0->datasets) (2024.9.0)
Requirement already satisfied: aiohttp in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (3.10.10)
Requirement already satisfied: huggingface-hub>=0.23.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (0.26.1)
Requirement already satisfied: packaging in c:\users\windows\appdata\roaming\python\python39\site-packages (from datasets) (24.1)
Requirement already satisfied: pyyaml>=5.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: yarl<2.0,>=1.12.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (1.16.0)
Requirement already satisfied: async-timeout<5.0,>=4.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\windows\appdata\roaming\python\python39\site-packages (from huggingface-hub>=0.23.0->datasets) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests>=2.32.2->datasets) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests>=2.32.2->datasets) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests>=2.32.2->datasets) (2024.8.30)
Requirement already satisfied: colorama in c:\users\windows\appdata\roaming\python\python39\site-packages (from tqdm>=4.66.3->datasets) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\windows\appdata\roaming\python\python39\site-packages (from pandas->datasets) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\windows\appdata\roaming\python\python39\site-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.16.0)
Requirement already satisfied: propcache>=0.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from yarl<2.0,>=1.12.0->aiohttp->datasets) (0.2.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: plotly in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from plotly) (9.0.0)
Requirement already satisfied: packaging in c:\users\windows\appdata\roaming\python\python39\site-packages (from plotly) (24.1)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.23 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (6.4.5)
Requirement already satisfied: zipp>=3.1.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from importlib-resources>=3.2.0->matplotlib) (3.20.2)
Requirement already satisfied: six>=1.5 in c:\users\windows\appdata\roaming\python\python39\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: seaborn in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from seaborn) (2.0.2)
Requirement already satisfied: pandas>=1.2 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (6.4.5)
Requirement already satisfied: pytz>=2020.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: zipp>=3.1.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from importlib-resources>=3.2.0->matplotlib!=3.6.1,>=3.4->seaborn) (3.20.2)
Requirement already satisfied: six>=1.5 in c:\users\windows\appdata\roaming\python\python39\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: datascience in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (0.17.6)
Requirement already satisfied: folium>=0.9.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (0.18.0)
Requirement already satisfied: setuptools in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (58.1.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (3.9.2)
Requirement already satisfied: pandas in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (2.2.3)
Requirement already satisfied: scipy in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (1.13.1)
Requirement already satisfied: numpy in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (2.0.2)
Requirement already satisfied: ipython in c:\users\windows\appdata\roaming\python\python39\site-packages (from datascience) (8.18.1)
Requirement already satisfied: plotly in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (5.24.1)
Requirement already satisfied: branca in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from datascience) (0.8.0)
Requirement already satisfied: jinja2>=2.9 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from folium>=0.9.1->datascience) (3.1.4)
Requirement already satisfied: requests in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from folium>=0.9.1->datascience) (2.32.3)
Requirement already satisfied: xyzservices in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from folium>=0.9.1->datascience) (2024.9.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (1.3.0)
```

```
Requirement already satisfied: cycler>=0.10 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib>=3.0.0->datascience) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\windows\appdata\roaming\python\python39\site-packages (from matplotlib>=3.0.0->datascience) (2.9.0.post0)
Requirement already satisfied: importlib-resources>=3.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from matplotlib>=3.0.0->datascience) (6.4.5)
Requirement already satisfied: decorator in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (0.19.1)
Requirement already satisfied: matplotlib-inline in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (0.1.7)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (3.0.48)
Requirement already satisfied: pygments>=2.4.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (2.18.0)
Requirement already satisfied: stack-data in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (0.6.3)
Requirement already satisfied: traitlets>=5 in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (5.14.3)
Requirement already satisfied: typing-extensions in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (4.12.2)
Requirement already satisfied: exceptiongroup in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (1.2.2)
Requirement already satisfied: colorama in c:\users\windows\appdata\roaming\python\python39\site-packages (from ipython->datascience) (0.4.6)
Requirement already satisfied: pytz>=2020.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas->datascience) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from pandas->datascience) (2024.2)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from plotly->datascience) (9.0.0)
Requirement already satisfied: zipp>=3.1.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from importlib-resources>=3.2.0->matplotlib>=3.0.0->datascience) (3.20.2)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\windows\appdata\roaming\python\python39\site-packages (from jedi>=0.16->ipython->datascience) (0.8.4)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from jinja2>=2.9->folium>=0.9.1->datascience) (3.0.2)
Requirement already satisfied: wcwidth in c:\users\windows\appdata\roaming\python\python39\site-packages (from prompt-toolkit<3.1.0,>=3.0.41->ipython->datascience) (0.2.13)
Requirement already satisfied: six>=1.5 in c:\users\windows\appdata\roaming\python\python39\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->datascience) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests->folium>=0.9.1->datascience) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests->folium>=0.9.1->datascience) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests->folium>=0.9.1->datascience) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from requests->folium>=0.9.1->datascience) (2024.8.30)
Requirement already satisfied: executing>=1.2.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from stack-data->ipython->datascience) (2.1.0)
Requirement already satisfied: asttokens>=2.1.0 in c:\users\windows\appdata\roaming\python\python39\site-packages (from stack-data->ipython->datascience) (2.4.1)
Requirement already satisfied: pure-eval in c:\users\windows\appdata\roaming\python\python39\site-packages (from stack-data->ipython->datascience) (0.2.3)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: scikit-learn in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\windows\appdata\local\programs\python\python39\lib\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```python
import pandas as pd

df = pd.read_csv("hf://datasets/maharshipandya/spotify-tracks-dataset/dataset.csv")
```

```
c:\Users\WINDOWS\AppData\Local\Programs\Python\Python39\lib\site-packages\tqdm\auto.py:21: 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 [3]:

```python
from datascience import *
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
```

In [4]:

```python
df.columns
```

Out[4]:

```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

In [5]:

```python
df.sample(10)
```

Out[5]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	...	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
66552	66552	3Xh4kaQAumTW993ETAdACB	Ray Remesch	EFlashApps Nursery Rhymes, Vol. 3	English Spanish Color Song	26	67010	False	0.716	0.3490	...	-8.007	1	0.0330	0.690000	0.000000	0.1770	0.645	121.973	4	kids
16534	16534	7HnAXewQTDWHRZ2FNrVlZY	Michael Harrison	Revelation	Revelation: Music in Pure Intonation: Night Vi...	45	187960	False	0.329	0.0130	...	-33.238	0	0.0486	0.994000	0.919000	0.0671	0.285	71.608	4	classical
76735	76735	3fOkhpPPYDD5sydEM1a2uV	Georges Bizet;Elisabeth Vidal;Michel Plasson;W...	Bizet : Carmen	Carmen, Act III, No.20 Trio: Mêlons! Coupons! ...	21	185760	False	0.468	0.0984	...	-19.091	1	0.0618	0.982000	0.000002	0.1820	0.156	120.414	4	opera
36145	36145	55HsZ3r9G0G58AYWZFY5VI	YL	RAP MUSCULATION RENTRÉE 2022	Niya	0	180120	True	0.753	0.7590	...	-7.848	0	0.0669	0.157000	0.000000	0.1020	0.514	105.015	3	french
5013	5013	0cSkn2l67csUljEy0EEBPn	LiSA	炎	炎	67	275000	False	0.477	0.6850	...	-4.554	1	0.0325	0.105000	0.000000	0.2770	0.308	152.040	4	anime
67266	67266	4pJUzyPGGBdV2yKlqYLNTN	Jhayco;Anuel AA	Frescura y Perreo	Ley Seca	0	263666	False	0.759	0.8430	...	-3.718	1	0.0913	0.127000	0.000000	0.1140	0.560	105.016	4	latin
6801	6801	049SQ7EJTypWBZFhqBDuhC	I AM	Life Through Torment	Face of Death	18	125000	True	0.236	0.9620	...	-5.760	1	0.1280	0.000004	0.048600	0.5070	0.420	95.121	4	black-metal
96098	96098	6Q4no26EaF80ttkI9HEWME	Casuarina	Roda de Samba do Casuarina e Convidados	Canto de Ossanha	47	264533	False	0.435	0.7080	...	-8.022	0	0.0375	0.649000	0.002100	0.7410	0.729	95.750	4	samba
83338	83338	0F9fZwEu2cWSgqoRWzAtAS	ARTY;Griff Clawson	Live For	Live For	63	171433	False	0.552	0.9100	...	-3.226	1	0.0385	0.224000	0.000014	0.1340	0.369	126.007	4	progressive-house
18853	18853	0Bd1WeTf27JqYQLQJPWKRo	Tom Papa	Calm, Cool, and Collected	Uncle Tom	20	76493	False	0.479	0.7970	...	-9.380	0	0.8530	0.801000	0.000000	0.8440	0.819	94.891	1	comedy

10 rows × 21 columns

The one missing row might be because the name and artists are in Korean since it is a k-pop genre song.

In [6]:

```
missing_prop = df.isna().sum() / len(df.index)
missing_prop.sort_values()
missing_rows = df[df.isna().any(axis=1)]

print("Number of missing data rows:", len(missing_rows))
print("Missing data rows:")
print(missing_rows)
```

```
Number of missing data rows: 1
Missing data rows:
       Unnamed: 0                track_id artists album_name track_name  \
65900       65900  1kR4gIb7nGxHPI3D2ifs59     NaN        NaN        NaN

       popularity  duration_ms  explicit  danceability  energy  ...  loudness  \
65900           0            0     False         0.501   0.583  ...     -9.46

       mode  speechiness  acousticness  instrumentalness  liveness  valence  \
65900     0       0.0605          0.69           0.00396    0.0747    0.734

         tempo  time_signature  track_genre
65900  138.391               4        k-pop

[1 rows x 21 columns]
```

In [7]:

```
# List of number of tracks (rows) in each unique track_genre
for genre in df['track_genre'].unique():
  print(f"{genre}: {len(df[df['track_genre'] == genre])}")
```

acoustic: 1000
afrobeat: 1000
alt-rock: 1000
alternative: 1000
ambient: 1000
anime: 1000
black-metal: 1000
bluegrass: 1000
blues: 1000
brazil: 1000
breakbeat: 1000
british: 1000
cantopop: 1000
chicago-house: 1000
children: 1000
chill: 1000
classical: 1000
club: 1000
comedy: 1000
country: 1000
dance: 1000
dancehall: 1000
death-metal: 1000
deep-house: 1000
detroit-techno: 1000
disco: 1000
disney: 1000
drum-and-bass: 1000
dub: 1000
dubstep: 1000
edm: 1000
electro: 1000
electronic: 1000
emo: 1000
folk: 1000
forro: 1000
french: 1000
funk: 1000
garage: 1000
german: 1000
gospel: 1000
goth: 1000
grindcore: 1000
groove: 1000
grunge: 1000
guitar: 1000
happy: 1000
hard-rock: 1000
hardcore: 1000
hardstyle: 1000
heavy-metal: 1000
hip-hop: 1000
honky-tonk: 1000
house: 1000
idm: 1000
indian: 1000
indie-pop: 1000
indie: 1000
industrial: 1000
iranian: 1000
j-dance: 1000
j-idol: 1000
j-pop: 1000
j-rock: 1000
jazz: 1000
k-pop: 1000
kids: 1000
latin: 1000
latino: 1000
malay: 1000
mandopop: 1000
metal: 1000
metalcore: 1000
minimal-techno: 1000
mpb: 1000
new-age: 1000
opera: 1000
pagode: 1000
party: 1000
piano: 1000
pop-film: 1000
pop: 1000
power-pop: 1000
progressive-house: 1000

```
psych-rock: 1000
punk-rock: 1000
punk: 1000
r-n-b: 1000
reggae: 1000
reggaeton: 1000
rock-n-roll: 1000
rock: 1000
rockabilly: 1000
romance: 1000
sad: 1000
salsa: 1000
samba: 1000
sertanejo: 1000
show-tunes: 1000
singer-songwriter: 1000
ska: 1000
sleep: 1000
songwriter: 1000
soul: 1000
spanish: 1000
study: 1000
swedish: 1000
synth-pop: 1000
tango: 1000
techno: 1000
trance: 1000
trip-hop: 1000
turkish: 1000
world-music: 1000
```

Data Cleaning:

There is only 1 row missing some values, so we are removing this row.

In [8]:
```
df = df.dropna()
missing_prop = df.isna().sum() / len(df.index)
missing_prop.sort_values()
missing_rows = df[df.isna().any(axis=1)]

print("Number of missing data rows:", len(missing_rows))
print("Missing data rows:")
print(missing_rows)
```

```
Number of missing data rows: 0
Missing data rows:
Empty DataFrame
Columns: [Unnamed: 0, track_id, artists, album_name, track_name, popularity, duration_ms, explicit, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, track_genre]
Index: []

[0 rows x 21 columns]
```

Exploratory Data Analysis:

Average popularity per Genre

In [9]:
```
avg_popularity_per_genre = df.groupby('track_genre')['popularity'].mean().reset_index()
avg_popularity_per_genre.columns = ["Genre", "Avg Popularity"]
avg_popularity_per_genre = avg_popularity_per_genre.sort_values(by='Avg Popularity', ascending=False)

avg_popularity_per_genre
```

Out[9]:

	Genre	Avg Popularity
81	pop-film	59.283000
65	k-pop	56.952953
15	chill	53.651000
94	sad	52.379000
44	grunge	49.594000
...	...	...
13	chicago-house	12.339000
24	detroit-techno	11.174000
67	latin	8.297000
93	romance	3.245000
59	iranian	2.210000

114 rows × 2 columns

Average Danceability Per Genre Table:

In [10]:

```
avg_dance_per_genre = df.groupby('track_genre')['danceability'].mean().reset_index()
avg_dance_per_genre.columns = ["Genre", "Avg Danceability"]
avg_dance_per_genre_sorted = avg_dance_per_genre.sort_values(by='Avg Danceability', ascending=True)
pd.set_option('display.max_rows', None)
print(avg_dance_per_genre_sorted)
```

	Genre	Avg Danceability
101	sleep	0.167923
42	grindcore	0.271854
6	black-metal	0.296411
59	iranian	0.300686
76	opera	0.313563
75	new-age	0.348455
4	ambient	0.367867
22	death-metal	0.368411
16	classical	0.381923
113	world-music	0.414572
72	metalcore	0.423800
50	heavy-metal	0.428500
93	romance	0.432133
79	piano	0.455098
44	grunge	0.457062
26	disney	0.462874
98	show-tunes	0.463738
71	metal	0.464288
40	gospel	0.473298
82	power-pop	0.473410
41	goth	0.478875
47	hard-rock	0.482250
38	garage	0.484264
85	punk	0.499778
11	british	0.501276
84	psych-rock	0.502554
86	punk-rock	0.507329
63	j-rock	0.509069
64	jazz	0.509975
29	dubstep	0.518087
39	german	0.524338
54	idm	0.527837
58	industrial	0.530416
2	alt-rock	0.534493
7	bluegrass	0.534692
27	drum-and-bass	0.535486
5	anime	0.537451
108	tango	0.537862
49	hardstyle	0.539229
45	guitar	0.540037
90	rock	0.543609
48	hardcore	0.546411
70	mandopop	0.546532
12	cantopop	0.547909
61	j-idol	0.548819
0	acoustic	0.549593
91	rock-n-roll	0.550302
62	j-pop	0.551678
46	happy	0.552847
19	country	0.555294
34	folk	0.558514
3	alternative	0.559927
92	rockabilly	0.561202
99	singer-songwriter	0.562022
102	songwriter	0.562022
9	brazil	0.562948
8	blues	0.568567
52	honky-tonk	0.571223
74	mpb	0.574088
96	samba	0.574897
28	dub	0.575321
18	comedy	0.577195
77	pagode	0.577723
100	ska	0.580676
110	trance	0.583409
43	groove	0.583910
56	indie	0.587272
106	swedish	0.590413
97	sertanejo	0.591647
55	indian	0.592273
81	pop-film	0.597146
33	emo	0.599321
103	soul	0.600112
17	club	0.603354
104	spanish	0.604310
57	indie-pop	0.604370
87	r-n-b	0.614388
112	turkish	0.616077
107	synth-pop	0.620663
83	progressive-house	0.623935
69	malay	0.628617
80	pop	0.630441
111	trip-hop	0.634695

```
10          breakbeat        0.646194
65              k-pop        0.647879
30                edm        0.648220
35              forro        0.649776
36             french        0.651468
31            electro        0.651749
32         electronic        0.652945
15              chill        0.664346
78              party        0.667191
95              salsa        0.668288
53              house        0.668534
1            afrobeat        0.669580
25              disco        0.676692
37               funk        0.678184
60            j-dance        0.680557
109            techno        0.684348
105             study        0.685240
20              dance        0.687856
94                sad        0.692378
23         deep-house        0.710448
14           children        0.716467
67              latin        0.721902
24     detroit-techno        0.722568
73     minimal-techno        0.729395
21          dancehall        0.734169
51            hip-hop        0.736154
88             reggae        0.745331
68             latino        0.757057
89          reggaeton        0.758521
13      chicago-house        0.766176
66               kids        0.778906
```

Average Energy Value Per Genre Table:

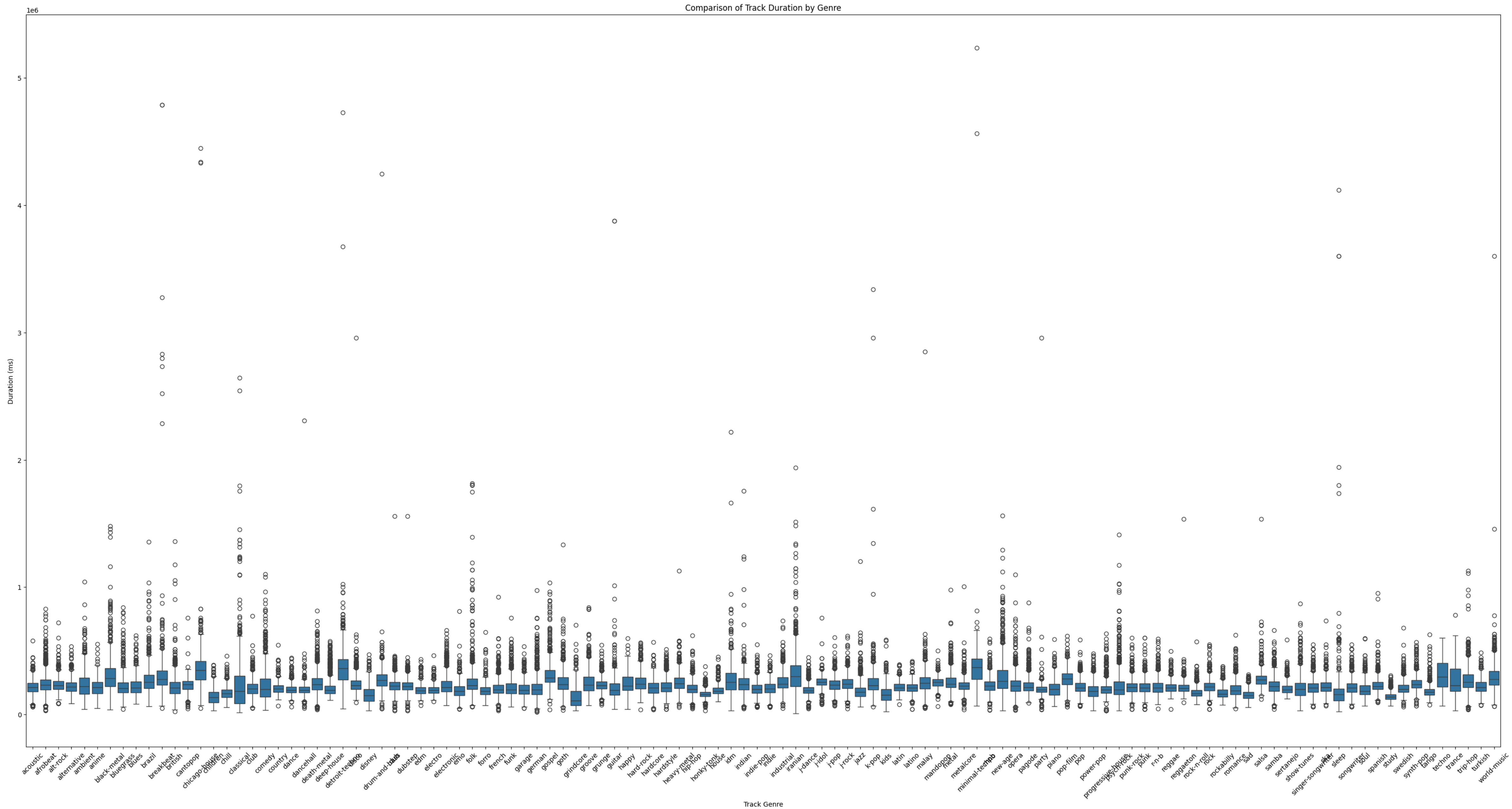
In [11]:
```
avg_energy_per_genre = df.groupby('track_genre')['energy'].mean().reset_index()
avg_energy_per_genre.columns = ['Genre', 'Average Energy']
avg_energy_per_genre_sorted = avg_energy_per_genre.sort_values(by='Average Energy', ascending=True)
pd.set_option('display.max_rows', None)
print(avg_energy_per_genre_sorted)
```

	Genre	Average Energy
16	classical	0.189827
75	new-age	0.214501
4	ambient	0.237162
93	romance	0.294304
26	disney	0.302519
76	opera	0.317054
79	piano	0.320103
45	guitar	0.324999
101	sleep	0.342072
64	jazz	0.352954
52	honky-tonk	0.366957
108	tango	0.372828
98	show-tunes	0.398742
105	study	0.410658
15	chill	0.426723
102	songwriter	0.434188
99	singer-songwriter	0.434188
0	acoustic	0.435368
12	cantopop	0.461696
94	sad	0.462470
14	children	0.494645
70	mandopop	0.498434
11	british	0.507127
91	rock-n-roll	0.526615
7	bluegrass	0.530280
39	german	0.531718
113	world-music	0.532987
103	soul	0.533873
34	folk	0.545807
59	iranian	0.545846
54	idm	0.555399
56	indie	0.556192
57	indie-pop	0.561140
84	psych-rock	0.561503
55	indian	0.567121
40	gospel	0.576256
69	malay	0.578397
74	mpb	0.579787
8	blues	0.581878
36	french	0.594990
19	country	0.596805
81	pop-film	0.604562
80	pop	0.606437
112	turkish	0.609804
66	kids	0.613129
9	brazil	0.620721
106	swedish	0.622003
111	trip-hop	0.622363
37	funk	0.632999
87	r-n-b	0.638130
92	rockabilly	0.659372
31	electro	0.665000
33	emo	0.669967
96	samba	0.672644
5	anime	0.674108
65	k-pop	0.675747
90	rock	0.679071
62	j-pop	0.679604
73	minimal-techno	0.680272
51	hip-hop	0.682530
21	dancehall	0.685262
32	electronic	0.694752
18	comedy	0.699934
1	afrobeat	0.702812
60	j-dance	0.703755
104	spanish	0.707777
20	dance	0.708583
97	sertanejo	0.710391
24	detroit-techno	0.710512
77	pagode	0.712123
107	synth-pop	0.713135
28	dub	0.714817
3	alternative	0.720030
17	club	0.721734
95	salsa	0.724518
88	reggae	0.725791
67	latin	0.727080
68	latino	0.731797
13	chicago-house	0.733215
25	disco	0.737565
89	reggaeton	0.738728
41	goth	0.740970
23	deep-house	0.741855

```
38            garage    0.744350
109           techno    0.746413
2           alt-rock    0.754173
53             house    0.755083
30               edm    0.756196
29           dubstep    0.758969
63            j-rock    0.760684
43            groove    0.763622
35             forro    0.789526
100              ska    0.793134
47         hard-rock    0.795039
82         power-pop    0.801688
44            grunge    0.803290
85              punk    0.809673
86         punk-rock    0.809980
83  progressive-house   0.813359
71             metal    0.840273
48          hardcore    0.842453
110           trance    0.845272
10         breakbeat    0.853275
58        industrial    0.861745
61            j-idol    0.868677
78             party    0.871237
50       heavy-metal    0.874003
6        black-metal    0.874897
27     drum-and-bass    0.876635
49         hardstyle    0.901246
46             happy    0.910971
72         metalcore    0.914485
42         grindcore    0.924201
22       death-metal    0.931470
```

In [12]:

```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Set the plot size
plt.figure(figsize=(40, 20))

# Create a box plot
sns.boxplot(x='track_genre', y='duration_ms', data = df)

# Set labels and title
plt.title('Comparison of Track Duration by Genre')
plt.xlabel('Track Genre')
plt.ylabel('Duration (ms)')

# Rotate x labels for better readability if there are many genres
plt.xticks(rotation=45)

# Show the plot
plt.show()
```



Week 3

In [13]:
```python
from sklearn.model_selection import train_test_split

# note train_test_split shuffles the data set by default
```

```python
# first split the dataset into 70% training and 30% temp
train_data, temp_data = train_test_split(df, test_size=0.3, random_state=42)

# then split the temp dataset into 15% testing and 15% validation
test_data, val_data = train_test_split(temp_data, test_size=0.5, random_state=42)

print(f'Training data shape: {train_data.shape}')
print(f'Testing data shape: {test_data.shape}')
print(f'Validation data shape: {val_data.shape}')

# If need to remake these datasets, uncomment to create new CSV files
# train_data.to_csv('csv/train_data.csv', index=False)
# test_data.to_csv('csv/test_data.csv', index=False)
# val_data.to_csv('csv/val_data.csv', index=False)
```

```
Training data shape: (79799, 21)
Testing data shape: (17100, 21)
Validation data shape: (17100, 21)
```

In [14]:

```python
# for consistency, everyone should use the same datasets by reading the same csv files

train_data = pd.read_csv("./csv/train_data.csv")
test_data = pd.read_csv("./csv/test_data.csv")
val_data = pd.read_csv("./csv/val_data.csv")
```

In [15]:

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_scaled = []
y = []

for d in [train_data, test_data, val_data]:
    non_numeric_cols = d.select_dtypes(include=['object']).columns
    # print(non_numeric_cols)

    X = d.drop(columns=['popularity', 'track_id', 'track_name','album_name', 'artists', 'track_genre'])
    y.append(d['popularity'])

    # for lasso:
    scaler = StandardScaler()
    X_scaled.append(scaler.fit_transform(X))
```

In [16]:

```python
# training data
X_train = X_scaled[0]
y_train = y[0]

# testing data
X_test = X_scaled[1]
y_test = y[1]

# validation data
X_val = X_scaled[2]
y_val = y[2]
```

Regularization

We are using lasso regression (L1 Regularization)

In [17]:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

lasso = Lasso(alpha=0.1) # try changing alpha to make it better!
lasso.fit(X_train, y_train)

y_val_pred = lasso.predict(X_val)
#MSE
val_mse = mean_squared_error(y_val, y_val_pred)
print(f"Validation MSE: {val_mse}")

lasso_coefficients = lasso.coef_
feature_names = X.columns
# feature_names = df.drop(columns=['popularity', 'track_id', 'track_name','album_name', 'artists', 'track_genre']).columns
important_features = [(feature, coef) for feature, coef in zip(feature_names, lasso_coefficients) if coef != 0]
important_features = sorted(important_features, key=lambda x: abs(x[1]), reverse=True)

print("Important Features:")
for feature, coef in important_features:
    print(f"{feature}: {coef}")
```

```
Validation MSE: 482.86028217945733
Important Features:
instrumentalness: -2.414702875009987
valence: -2.2671419290598616
speechiness: -1.418747157755636
danceability: 1.2105724310109753
explicit: 0.8757417797762873
Unnamed: 0: 0.4617996878219175
time_signature: 0.4068828297359902
tempo: 0.30046698670147104
mode: -0.2540433439760859
loudness: 0.2540127477409079
energy: -0.178837834347387
acousticness: -0.09365770338322728
duration_ms: -0.09162713619712619
liveness: 0.022333228072016768
```

Based on these lasso regression results to find the most important predictors of popularity, we have decided to use Danceability as our predictors of popularity so we can apply linear regression.

One question we have: we are a bit confused about why valence is a big negative predictor of popularity. Some of the lasso regression results were did not exactly align with our own assumptions of what they should be.

In [18]:
```
#Linear and LAD Regression Models
from sklego.linear_model import LADRegression
from sklearn.linear_model import LinearRegression

lad_fit = LADRegression()
lad_fit.fit(X_train, y_train)

ls_fit = LinearRegression()
ls_fit.fit(X_train,y_train)
print("Intercept: ", ls_fit.intercept_)
print("Coefficients: ", ls_fit.coef_[0])
```

```
Intercept:  33.2653007519
Coefficients:  0.556606970401
```

In [19]:
```
import plotly.graph_objects as go

# plot both the LS and LAD lines on top of the scatterplot of gr_liv_area against sale_price
fig = px.scatter(train_data, x='danceability', y='popularity')
fig.add_trace(
    go.Scatter(x=train_data['danceability'],
               y=lad_fit.intercept_ + train_data['danceability'] * lad_fit.coef_[0],
               mode='lines',
               name='LAD',
               line={'dash': 'dash',
                     'color': 'red'})
               # L1 model
)
fig.add_trace(
    go.Scatter(x=train_data['danceability'],
               y=ls_fit.intercept_ + train_data['danceability'] * ls_fit.coef_[0],
               mode='lines',
               name='LS',
               line={'dash': 'solid',
                     'color': 'black'})
               # L2 model
)
fig.update_traces(marker=dict(size=2))
```

Not great for visualization, let's try a different form of regression.

In [20]:
```
# Linear Model Evaluation

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Training Data
pred_train_df = pd.DataFrame( # dataframe to compare true value to predicted value
    {'true': y_train,
     'ls_pred': ls_fit.predict(np.array(X_train))})

# compute the rMSE, MAE, MAD, correlation and R2 of true value with predictions
print('LS rMSE:', np.sqrt(mean_squared_error(pred_train_df['true'], pred_train_df['ls_pred'])))
print('LS MAE:', mean_absolute_error(pred_train_df['true'], pred_train_df['ls_pred']))
print('LS MAD:', np.median(np.abs(pred_train_df['true'] - pred_train_df['ls_pred'])))
print('LS correlation:', np.corrcoef(pred_train_df['true'], pred_train_df['ls_pred'])[0, 1])
print('LS R2:', r2_score(pred_train_df['true'], pred_train_df['ls_pred']))

# Validation Data
pred_val_df = pd.DataFrame( # dataframe to compare true value to predicted value
    {'true': y_val,
     'ls_pred': ls_fit.predict(np.array(X_val))})
```

```python
# compute the rMSE, MAE, MAD, correlation and R2 of true value with predictions
print('LS rMSE:', np.sqrt(mean_squared_error(pred_val_df['true'], pred_val_df['ls_pred'])))
print('LS MAE:', mean_absolute_error(pred_val_df['true'], pred_val_df['ls_pred']))
print('LS MAD:', np.median(np.abs(pred_val_df['true'] - pred_val_df['ls_pred'])))
print('LS correlation:', np.corrcoef(pred_val_df['true'], pred_val_df['ls_pred'])[0, 1])
print('LS R2:', r2_score(pred_val_df['true'], pred_val_df['ls_pred']))
```

```
LS rMSE: 22.0393643843
LS MAE: 18.378427291
LS MAD: 15.9652304617
LS correlation: 0.161013356635
LS R2: 0.02592530101480972
LS rMSE: 21.9695910058
LS MAE: 18.3143020336
LS MAD: 15.8762158042
LS correlation: 0.157469512411
LS R2: 0.02470036472170345
```

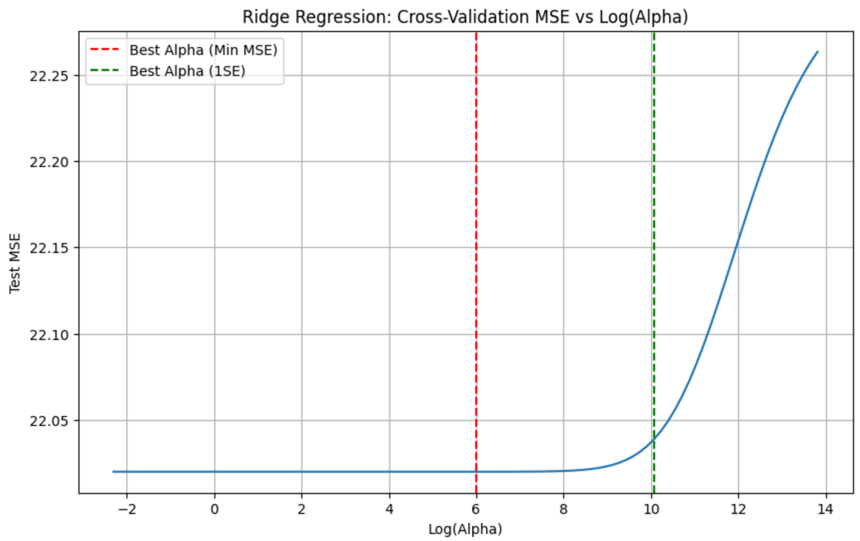
Applying Ridge Regression

In [21]:
```python
# for ridge regression
X = df.drop(columns=['popularity', 'track_id', 'track_name','album_name', 'artists', 'track_genre'])
y = df['popularity']
```

In [22]:
```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold, cross_validate, learning_curve


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled.shape)

alphas = np.logspace(-1, 6, 100)
kf = KFold(n_splits=10, shuffle=True, random_state=42)


ridge_cv_scores = []
for alpha in alphas:
  ridge = Ridge(alpha=alpha)
  ridge_cv = cross_validate(estimator=ridge,
                             X=X_scaled,
                             y=y,
                             cv=kf,
                             scoring='neg_root_mean_squared_error')
  ridge_cv_scores.append({'alpha': alpha,
                           'log_alpha': np.log(alpha),
                           'test_mse': -np.mean(ridge_cv['test_score'])})

ridge_cv_scores_df = pd.DataFrame(ridge_cv_scores)
px.line(ridge_cv_scores_df,
        x='log_alpha',
        y='test_mse',
        title='Ridge')

#find best alpha
ridge_alpha_min = ridge_cv_scores_df.sort_values(by='test_mse').head(1).alpha.values[0]

mse_se_ridge = ridge_cv_scores_df['test_mse'].std() / np.sqrt(10)
mse_min_ridge = ridge_cv_scores_df['test_mse'].min()

ridge_alpha_1se = ridge_cv_scores_df[(ridge_cv_scores_df['test_mse'] <= mse_min_ridge + mse_se_ridge) &
                                     (ridge_cv_scores_df['test_mse'] >= mse_min_ridge - mse_se_ridge)].sort_values(by='alpha', ascending=False).head(1).alpha.values[0]

print('Ridge (min): ', ridge_alpha_min)
print('Ridge (1SE): ', ridge_alpha_1se)

ridge_min_fit = Ridge(alpha=ridge_alpha_min).fit(X=X_scaled, y=y)
ridge_1se_fit = Ridge(alpha=ridge_alpha_1se).fit(X=X_scaled, y=y)
```

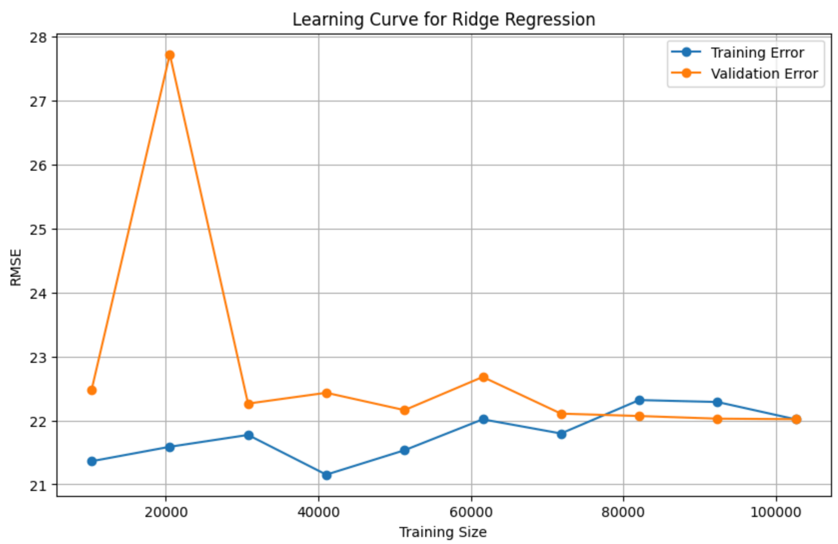
```
(113999, 15)
Ridge (min):  403.70172586
Ridge (1SE):  23644.8941265
```

In [23]:
```python
plt.figure(figsize=(10, 6))
plt.plot(ridge_cv_scores_df['log_alpha'], ridge_cv_scores_df['test_mse'])
plt.title('Ridge Regression: Cross-Validation MSE vs Log(Alpha)')
plt.xlabel('Log(Alpha)')
plt.ylabel('Test MSE')
plt.axvline(x=np.log(ridge_alpha_min), color='r', linestyle='--', label='Best Alpha (Min MSE)')
plt.axvline(x=np.log(ridge_alpha_1se), color='g', linestyle='--', label='Best Alpha (1SE)')
plt.legend()
plt.grid()
plt.show()
```



In [24]:

```
train_sizes, train_scores, test_scores = learning_curve(
    Ridge(alpha=ridge_alpha_min), X_scaled, y, n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 10), cv=kf, scoring='neg_root_mean_squared_error'
)
train_scores_mean = -np.mean(train_scores, axis=1)  # Negate to convert from negative RMSE to positive
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = -np.mean(test_scores, axis=1)    # Negate to convert from negative RMSE to positive
test_scores_std = np.std(test_scores, axis=1)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, label='Training Error', marker='o')
plt.plot(train_sizes, test_scores_mean, label='Validation Error', marker='o')
plt.title('Learning Curve for Ridge Regression')
plt.xlabel('Training Size')
plt.ylabel('RMSE')
plt.legend()
plt.grid()
plt.show()
```



Overfitting or Underfitting?

This graph shows that from 20000 to about 30000 (training size), the training error and validation error gap being really big means the model is overfitting, meaning the model isn't good at generalizing at this smaller size. Then the error gap decreases after 40000 and continues to improve.

WEEK 4

Logistic Regression

We are choosing mode as our response variable for logistic regression.

In [25]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve
```

In [26]:
```
non_numeric_cols = df.select_dtypes(exclude=[np.number]).columns.tolist()
print(non_numeric_cols)
numeric_df = df.select_dtypes(include=[np.number])
correlation_matrix = numeric_df.corr()
mode_correlation = correlation_matrix['mode'].sort_values(ascending=False)
print(mode_correlation)

fig = px.imshow(correlation_matrix, text_auto=True)
fig.update_layout(title='Correlation Matrix')
fig.show()
```

```
['track_id', 'artists', 'album_name', 'track_name', 'explicit', 'track_genre']
mode                1.000000
acousticness        0.095568
valence             0.021964
liveness            0.014004
Unnamed: 0          0.005110
tempo               0.000572
popularity         -0.013948
time_signature     -0.024090
duration_ms        -0.035581
loudness           -0.041768
speechiness        -0.046535
instrumentalness   -0.049961
danceability       -0.069224
energy             -0.078365
key                -0.135911
Name: mode, dtype: float64
```

Although none of the variables show a particularly strong correlation with mode, we will be using **key, acousticness, and energy** as they still were the most correlated. Another option we could have chosen to perform logistic regression on was the "explicit" variable, but there was too large of a class imbalance so we chose mode instead.

In [27]:
```
response = df['mode']
```

In [28]:
```
X = df[['acousticness', 'energy', 'key']]
y = df['mode']
```

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22)
```

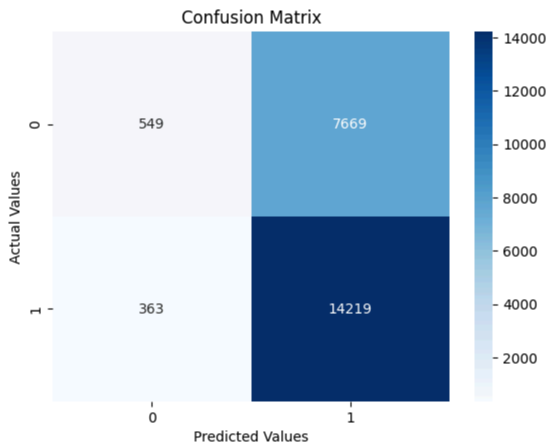
In [30]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[30]:
```
LogisticRegression
LogisticRegression()
```

In [31]:
```
# prediction accuracy and error
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Prediction Accuracy: {accuracy * 100:.2f}%")
print(f"Prediction Error: {(1-accuracy) * 100:.2f}%")
```

```
Prediction Accuracy: 64.77%
Prediction Error: 35.23%
```

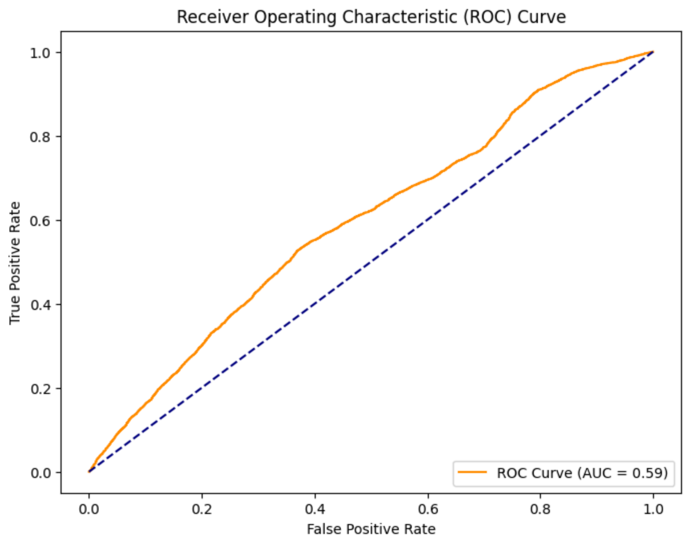
In [32]:
```
# confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.ylabel("Actual Values")
plt.xlabel("Predicted Values")
plt.show()
```



In [33]:
```
# True posiive and negative rate
truePositiveRate = cm[1,1] / (cm[1,1] + cm[1,0])
falsePositiveRate = cm[0,1] / (cm[0,1] + cm[0,0])
print(f"True Positive Rate: {truePositiveRate:.2f}")
print(f"True Negative Rate: {falsePositiveRate:.2f}")
```

```
True Positive Rate: 0.98
True Negative Rate: 0.93
```

In [34]:
```
# ROC Curve
y_prob = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})', color='darkorange')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print('optimal_threshold: ', optimal_threshold)
```




```
optimal_threshold:  0.649974884131
```

In [35]:
```
from sklearn.model_selection import cross_val_score

# -------------------------
# K-Fold Cross-Validation
# -------------------------

# We'll use the previously selected features for this exercise (or all features if you haven't done feature selection yet)
X_cv =  X
y_cv =  y

# Initialize the logistic regression model
model_cv = LogisticRegression()

# Perform 5-Fold Cross-Validation
cv_scores = cross_val_score(model_cv, X_cv, y_cv, cv=5)

# Output the cross-validation scores for each fold
# Modify the printed scores to the acutal cv_scores
print(f"Cross-Validation Scores for each fold: {cv_scores}")
print(f"Mean Cross-Validation Accuracy: {np.mean(cv_scores) * 100:.2f}%")
```

```
Cross-Validation Scores for each fold: [ 0.64342105  0.64320175  0.64969298  0.6452193   0.64331769]
Mean Cross-Validation Accuracy: 64.50%
```

5. To choose a threshold for positive predictions, we examined the ROC curve and which showed how true positive rate (TPR) and false positive rate (FPR) change as the threshold varies, and we used the AUC score as a measure of overall model performance. We selected a threshold of 0.65 from calculating the optimal threshold where where (TPR - FPR) is maximized.

If we want to improve the ROC and AUC, we can convert one of the numerical variables to a categorical one, and perform logistic regression on that.

Week 5

Applying KNN for the energy feature:

In [36]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
```

To split the energy feature into a binary feature, we are specifying the threshold to be the 75th percentile. So the top 25% of tracks will be "high", and everything else is "low".

In [38]:
```
df_copy = df.copy()
threshold = df_copy['energy'].median()

df_copy["energy_level"] = df_copy['energy'].apply(lambda x: "high" if x>=threshold else "low")

#encode as 0 or 1 (1 = high, 0 = low)
le = LabelEncoder()
df_copy['energy_level'] = le.fit_transform(df_copy['energy_level'])
```

```python
# dropping energy_level, and the non numerical features
X = df_copy[['loudness', 'acousticness']]
y = df_copy['energy_level']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [39]:
```python
knn1 = KNeighborsClassifier(n_neighbors=1)
knn15 = KNeighborsClassifier(n_neighbors=15)
knn25 = KNeighborsClassifier(n_neighbors=25)
```

In [40]:
```python
knn1.fit(X_train, y_train)
knn15.fit(X_train, y_train)
knn25.fit(X_train, y_train)
```

Out[40]:
```
KNeighborsClassifier

KNeighborsClassifier(n_neighbors=25)
```

In [41]:
```python
y_pred1 = knn1.predict(X_test)
y_pred15 = knn15.predict(X_test)
y_pred25 = knn25.predict(X_test)
```

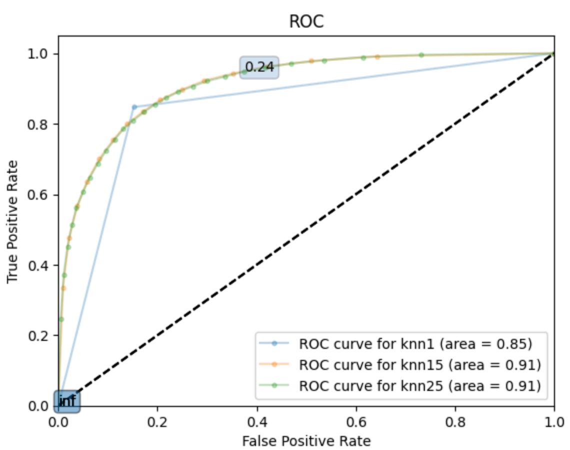
In [42]:
```python
accuracy1 = accuracy_score(y_test, y_pred1)
accuracy15 = accuracy_score(y_test, y_pred15)
accuracy25 = accuracy_score(y_test, y_pred25)

print(accuracy1)
print(accuracy15)
print(accuracy25)
```

```
0.8476608187134503
0.8319005847953216
0.8300292397660819
```

In [43]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
model_cv.fit(X_train, y_train)

# this function may help to manually make confusion table from a different threshold
def t_repredict(model_cv, t, X_train):
    probs = model_cv.predict_proba(X_train)
    p0 = probs[:,0]
    p1 = probs[:,1]
    ypred = (p1 > t)*1
    return ypred

# default threshold
y_pred_train = t_repredict(model_cv, 0.5, X_train)

# Our decided threshold
threshold= 0.65
y_pred_train = t_repredict(model_cv, threshold, X_train)

# Confusion Matrix
conf_matrix_train = confusion_matrix(y_train, y_pred_train)
TN, FP, FN, TP = conf_matrix_train.ravel()  # Extract TN, FP, FN, TP from the matrix

# Prediction Accuracy
accuracy = accuracy_score(y_train, y_pred_train)

# Prediction Error (1 - accuracy)
error = 1 - accuracy

# True Positive Rate (Sensitivity / Recall)
true_positive_rate = TP / (TP + FN)

# True Negative Rate (Specificity)
true_negative_rate = TN / (TN + FP)

# F1 Score
f1 = f1_score(y_train, y_pred_train)

# Display results
print("Confusion Matrix:\n", conf_matrix_train)
print("Prediction Accuracy:", accuracy)
print("Prediction Error:", error)
print("True Positive Rate (Recall/Sensitivity):", true_positive_rate)
print("True Negative Rate (Specificity):", true_negative_rate)
print("F1 Score:", f1)
```

```
Confusion Matrix:
 [[36557  3401]
 [13300 26541]]
Prediction Accuracy: 0.7907116630534218
Prediction Error: 0.20928833694657822
True Positive Rate (Recall/Sensitivity): 0.666173037825
True Negative Rate (Specificity): 0.914885629911
F1 Score: 0.760672370061
```

In [44]:
```python
from sklearn.metrics import roc_curve, auc

# a function to make 'pretty' ROC curves for this model
def make_roc(name, clf, ytest, xtest, ax=None, labe=5, proba=True, skip=0):
    initial=False
    if not ax:
        ax=plt.gca()
        initial=True
    if proba:#for stuff like logistic regression
        fpr, tpr, thresholds=roc_curve(ytest, clf.predict_proba(xtest)[:,1])
    else:#for stuff like SVM
        fpr, tpr, thresholds=roc_curve(ytest, clf.decision_function(xtest))
    roc_auc = auc(fpr, tpr)
    if skip:
        l=fpr.shape[0]
        ax.plot(fpr[0:l:skip], tpr[0:l:skip], '.-', alpha=0.3, label='ROC curve for %s (area = %0.2f)' % (name, roc_auc))
    else:
        ax.plot(fpr, tpr, '.-', alpha=0.3, label='ROC curve for %s (area = %0.2f)' % (name, roc_auc))
    label_kwargs = {}
    label_kwargs['bbox'] = dict(
        boxstyle='round,pad=0.3', alpha=0.2,
    )
    if labe!=None:
        for k in range(0, fpr.shape[0],labe):
            threshold = str(np.round(thresholds[k], 2))
            ax.annotate(threshold, (fpr[k], tpr[k]), **label_kwargs)
    if initial:
        ax.plot([0, 1], [0, 1], 'k--')
        ax.set_xlim([0.0, 1.0])
        ax.set_ylim([0.0, 1.05])
        ax.set_xlabel('False Positive Rate')
        ax.set_ylabel('True Positive Rate')
        ax.set_title('ROC')
    ax.legend(loc="lower right")
    return ax


make_roc("knn1", knn1, y_test, X_test, ax=None, labe=20, proba=True, skip=1);
make_roc("knn15", knn15, y_test, X_test, ax=None, labe=20, proba=True, skip=1);
make_roc("knn25", knn25, y_test, X_test, ax=None, labe=20, proba=True, skip=1);
```



In [45]:
```python
from sklearn.model_selection import cross_val_score

# -------------------------
# K-Fold Cross-Validation
# -------------------------

# We'll use the previously selected features for this exercise (or all features if you haven't done feature selection yet)
```

```python
X_cv =  X
y_cv =  y

# Perform 5-Fold Cross-Validation
cv_scores_knn1 = cross_val_score(knn1, X_cv, y_cv, cv=5)
cv_scores_knn15 = cross_val_score(knn15, X_cv, y_cv, cv=5)
cv_scores_knn25 = cross_val_score(knn25, X_cv, y_cv, cv=5)

# Output the cross-validation scores for each fold
# Modify the printed scores to the acutal cv_scores
print(f"Cross-Validation Scores for each fold for knn1: {cv_scores_knn1}")
print(f"Mean Cross-Validation Accuracy for knn1: {np.mean(cv_scores_knn1) * 100:.2f}%")

print(f"Cross-Validation Scores for each fold for knn15: {cv_scores_knn15}")
print(f"Mean Cross-Validation Accuracy for knn15: {np.mean(cv_scores_knn15) * 100:.2f}%")

print(f"Cross-Validation Scores for each fold for knn25: {cv_scores_knn25}")
print(f"Mean Cross-Validation Accuracyfor knn25: {np.mean(cv_scores_knn25) * 100:.2f}%")
```

```
Cross-Validation Scores for each fold for knn1: [ 0.82263158  0.8120614   0.84184211  0.85837719  0.7959121 ]
Mean Cross-Validation Accuracy for knn1: 82.62%
Cross-Validation Scores for each fold for knn15: [ 0.8247807   0.79622807  0.84065789  0.845       0.79288565]
Mean Cross-Validation Accuracy for knn15: 81.99%
Cross-Validation Scores for each fold for knn25: [ 0.82412281  0.79342105  0.84052632  0.84315789  0.79082416]
Mean Cross-Validation Accuracyfor knn25: 81.84%
```