1_Data_Exploration

April 25, 2025

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
import os
```

[2]:
```python
# --- Configuration ---
# Path to the raw data file
RAW_DATA_PATH = "../data/raw/YearPredictionMSD.txt"
# Directory to save generated plots
PLOT_SAVE_DIR = "../results/plots/"
# Ensure plot directory exists
os.makedirs(PLOT_SAVE_DIR, exist_ok=True)

# --- Optional: Add src to path to reuse functions ---
module_path = os.path.abspath(os.path.join('..', 'src'))
if module_path not in sys.path:
    sys.path.append(module_path)
from data_processing import load_data, create_decade_bins
#If not reusing functions from src, redefine them or necessary parts below
```

[3]:
```python
# --- Data Loading ---
print(f"Loading raw data from: {RAW_DATA_PATH}")
try:
    # Define column names as in data_processing.py
    N_FEATURES = 90
    colnames = ['Year'] + [f'Feature_{i+1}' for i in range(N_FEATURES)]
    df_raw = pd.read_csv(RAW_DATA_PATH, header=None, names=colnames)
    print(f"Data loaded successfully. Shape: {df_raw.shape}")
    print("\nFirst 5 rows of raw data:")
    print(df_raw.head())
    print("\nBasic data info:")
    df_raw.info()
except FileNotFoundError:
    print(f"ERROR: Raw data file not found at {RAW_DATA_PATH}. Please ensure␣
↪it's downloaded.")
    # Exit or handle error appropriately in a real script
```

```
    # For a notebook, we might just stop execution here or raise the error
    raise
except Exception as e:
    print(f"An error occurred during data loading: {e}")
    raise
```

```
Loading raw data from: ../data/raw/YearPredictionMSD.txt
Data loaded successfully. Shape: (515345, 91)

First 5 rows of raw data:
   Year  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
0  2001   49.94357   21.47114   73.07750    8.74861  -17.40628  -13.09905
1  2001   48.73215   18.42930   70.32679   12.94636  -10.32437  -24.83777
2  2001   50.95714   31.85602   55.81851   13.41693   -6.57898  -18.54940
3  2001   48.24750   -1.89837   36.29772    2.58776    0.97170  -26.21683
4  2001   50.97020   42.20998   67.09964    8.46791  -15.85279  -16.81409

   Feature_7  Feature_8  Feature_9  …  Feature_81  Feature_82  Feature_83  \
0  -25.01202  -12.23257    7.83089  …    13.01620   -54.40548    58.99367
1    8.76630   -0.92019   18.76548  …     5.66812   -19.68073    33.04964
2   -3.27872   -2.35035   16.07017  …     3.03800    26.05866   -50.92779
3    5.05097  -10.34124    3.55005  …    34.57337  -171.70734   -16.96705
4  -12.48207   -9.37636   12.63699  …     9.92661   -55.95724    64.92712

   Feature_84  Feature_85  Feature_86  Feature_87  Feature_88  Feature_89  \
0    15.37344     1.11144   -23.08793    68.40795    -1.82223   -27.46348
1    42.87836    -9.90378   -32.22788    70.49388    12.04941    58.43453
2    10.93792    -0.07568    43.20130  -115.00698    -0.05859    39.67068
3   -46.67617   -12.51516    82.58061   -72.08993     9.90558   199.62971
4   -17.72522    -1.49237    -7.50035    51.76631     7.88713    55.66926

   Feature_90
0     2.26327
1    26.92061
2    -0.66345
3    18.85382
4    28.74903

[5 rows x 91 columns]

Basic data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 515345 entries, 0 to 515344
Data columns (total 91 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Year         515345 non-null  int64
 1   Feature_1    515345 non-null  float64
```

```
2   Feature_2   515345 non-null  float64
3   Feature_3   515345 non-null  float64
4   Feature_4   515345 non-null  float64
5   Feature_5   515345 non-null  float64
6   Feature_6   515345 non-null  float64
7   Feature_7   515345 non-null  float64
8   Feature_8   515345 non-null  float64
9   Feature_9   515345 non-null  float64
10  Feature_10  515345 non-null  float64
11  Feature_11  515345 non-null  float64
12  Feature_12  515345 non-null  float64
13  Feature_13  515345 non-null  float64
14  Feature_14  515345 non-null  float64
15  Feature_15  515345 non-null  float64
16  Feature_16  515345 non-null  float64
17  Feature_17  515345 non-null  float64
18  Feature_18  515345 non-null  float64
19  Feature_19  515345 non-null  float64
20  Feature_20  515345 non-null  float64
21  Feature_21  515345 non-null  float64
22  Feature_22  515345 non-null  float64
23  Feature_23  515345 non-null  float64
24  Feature_24  515345 non-null  float64
25  Feature_25  515345 non-null  float64
26  Feature_26  515345 non-null  float64
27  Feature_27  515345 non-null  float64
28  Feature_28  515345 non-null  float64
29  Feature_29  515345 non-null  float64
30  Feature_30  515345 non-null  float64
31  Feature_31  515345 non-null  float64
32  Feature_32  515345 non-null  float64
33  Feature_33  515345 non-null  float64
34  Feature_34  515345 non-null  float64
35  Feature_35  515345 non-null  float64
36  Feature_36  515345 non-null  float64
37  Feature_37  515345 non-null  float64
38  Feature_38  515345 non-null  float64
39  Feature_39  515345 non-null  float64
40  Feature_40  515345 non-null  float64
41  Feature_41  515345 non-null  float64
42  Feature_42  515345 non-null  float64
43  Feature_43  515345 non-null  float64
44  Feature_44  515345 non-null  float64
45  Feature_45  515345 non-null  float64
46  Feature_46  515345 non-null  float64
47  Feature_47  515345 non-null  float64
48  Feature_48  515345 non-null  float64
49  Feature_49  515345 non-null  float64
```

```
 50  Feature_50  515345 non-null  float64
 51  Feature_51  515345 non-null  float64
 52  Feature_52  515345 non-null  float64
 53  Feature_53  515345 non-null  float64
 54  Feature_54  515345 non-null  float64
 55  Feature_55  515345 non-null  float64
 56  Feature_56  515345 non-null  float64
 57  Feature_57  515345 non-null  float64
 58  Feature_58  515345 non-null  float64
 59  Feature_59  515345 non-null  float64
 60  Feature_60  515345 non-null  float64
 61  Feature_61  515345 non-null  float64
 62  Feature_62  515345 non-null  float64
 63  Feature_63  515345 non-null  float64
 64  Feature_64  515345 non-null  float64
 65  Feature_65  515345 non-null  float64
 66  Feature_66  515345 non-null  float64
 67  Feature_67  515345 non-null  float64
 68  Feature_68  515345 non-null  float64
 69  Feature_69  515345 non-null  float64
 70  Feature_70  515345 non-null  float64
 71  Feature_71  515345 non-null  float64
 72  Feature_72  515345 non-null  float64
 73  Feature_73  515345 non-null  float64
 74  Feature_74  515345 non-null  float64
 75  Feature_75  515345 non-null  float64
 76  Feature_76  515345 non-null  float64
 77  Feature_77  515345 non-null  float64
 78  Feature_78  515345 non-null  float64
 79  Feature_79  515345 non-null  float64
 80  Feature_80  515345 non-null  float64
 81  Feature_81  515345 non-null  float64
 82  Feature_82  515345 non-null  float64
 83  Feature_83  515345 non-null  float64
 84  Feature_84  515345 non-null  float64
 85  Feature_85  515345 non-null  float64
 86  Feature_86  515345 non-null  float64
 87  Feature_87  515345 non-null  float64
 88  Feature_88  515345 non-null  float64
 89  Feature_89  515345 non-null  float64
 90  Feature_90  515345 non-null  float64
dtypes: float64(90), int64(1)
memory usage: 357.8 MB
```

[4]:
```
# --- Decade Binning (Reproduce logic from data_processing.py) ---
print("\nCreating decade bins for analysis...")
min_year = 1920 # Start decade reference
```

```
df_raw['Decade_Start'] = (df_raw['Year'] // 10) * 10
df_raw['Decade_Label'] = ((df_raw['Decade_Start'] - min_year) // 10).astype(int)
df_raw['Decade_Label'] = df_raw['Decade_Label'].clip(lower=0) # Clip years <␣
↪1920
decade_map = {i: f"{min_year + i*10}s" for i in range(10)}
df_raw['Decade_Name'] = df_raw['Decade_Label'].map(decade_map)
print("Decade columns ('Decade_Label', 'Decade_Name') added.")
```

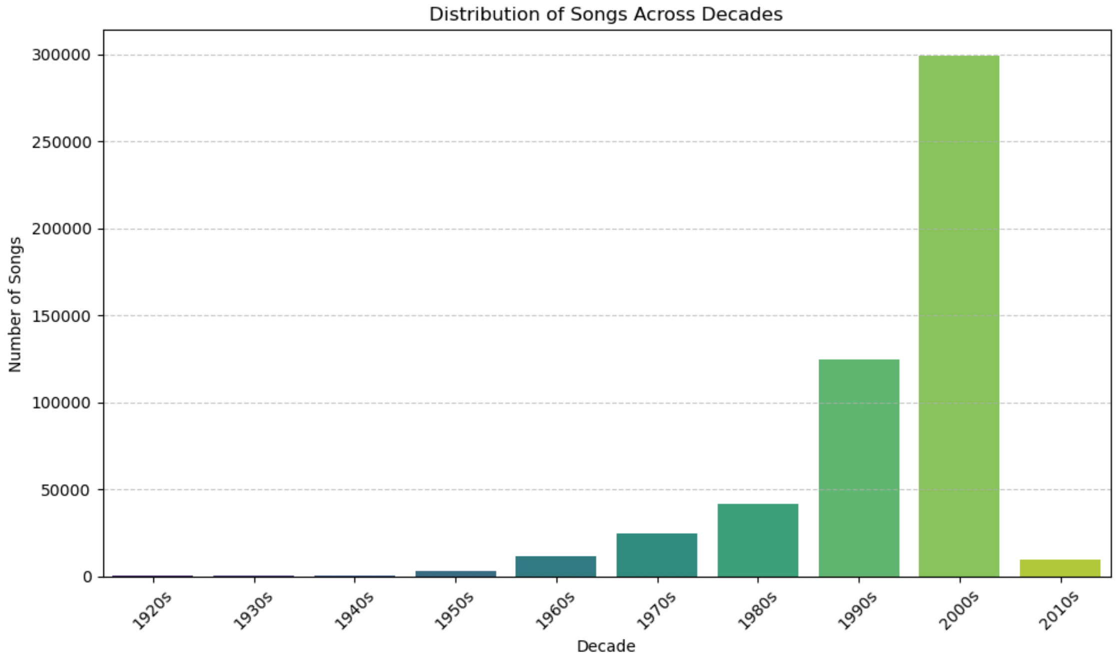
```
Creating decade bins for analysis…
Decade columns ('Decade_Label', 'Decade_Name') added.
```

[5]:

```
# --- 1. Class Balance Analysis ---
print("\n--- 1. Class Balance Analysis ---")
plt.figure(figsize=(10, 6))
sns.countplot(data=df_raw, x='Decade_Name', order=[decade_map[i] for i in␣
↪range(10)], palette='viridis')
plt.title('Distribution of Songs Across Decades')
plt.xlabel('Decade')
plt.ylabel('Number of Songs')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(PLOT_SAVE_DIR, 'eda_decade_distribution.png'))
plt.show()

decade_counts = df_raw['Decade_Name'].value_counts().sort_index()
print("\nSong Counts per Decade:")
print(decade_counts)
print(f"\nObservations: The dataset is heavily imbalanced, with a vast majority␣
↪of songs from the 2000s, followed by the 1990s. Earlier decades have␣
↪significantly fewer samples.")
```

```
--- 1. Class Balance Analysis ---
```

```
/var/folders/yf/9gf2xg3j2q76vf7cw8s4v4tw0000gn/T/ipykernel_10423/3132200938.py:4
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_raw, x='Decade_Name', order=[decade_map[i] for i in
range(10)], palette='viridis')
```



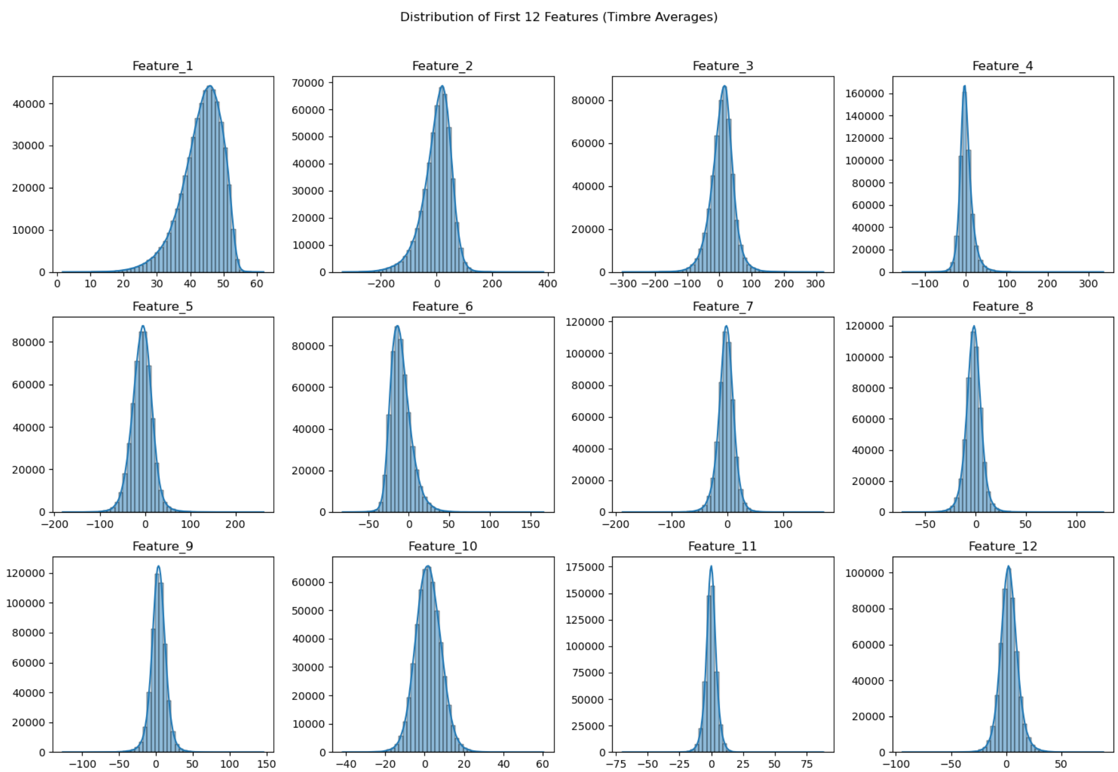

```
Song Counts per Decade:
Decade_Name
1920s       224
1930s       252
1940s       356
1950s      3102
1960s     11739
1970s     24745
1980s     41814
1990s    124713
2000s    299003
2010s      9397
Name: count, dtype: int64
```

Observations: The dataset is heavily imbalanced, with a vast majority of songs from the 2000s, followed by the 1990s. Earlier decades have significantly fewer samples.

[6]:
```
# --- 2. Feature Distribution Analysis ---
print("\n--- 2. Feature Distribution Analysis ---")
# Select a subset of features for detailed analysis (e.g., first 12, often␣
↪timbre averages)
# and maybe a few from the covariance features later on.
features_to_plot = df_raw.columns[1:13] # Features 1 to 12 (Timbre Averages)
print(f"Plotting distributions for features: {list(features_to_plot)}")
```

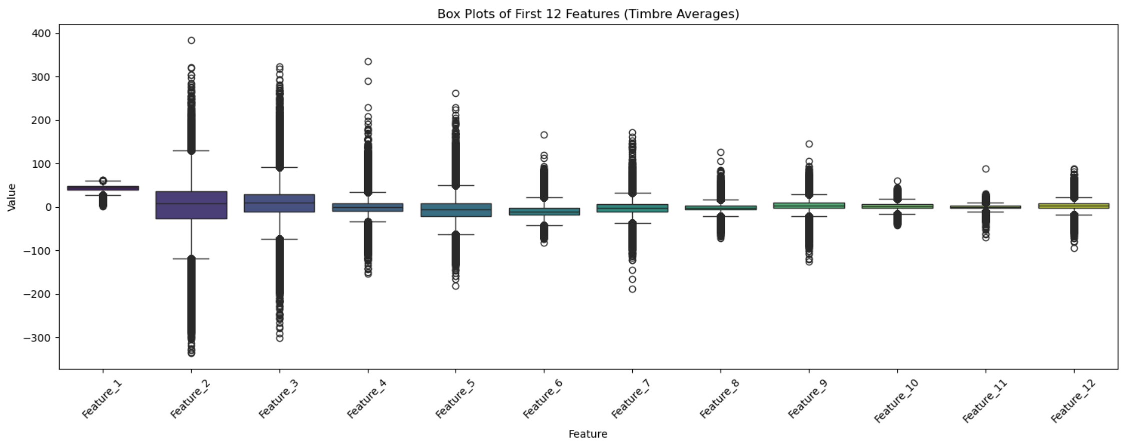
```
--- 2. Feature Distribution Analysis ---
Plotting distributions for features: ['Feature_1', 'Feature_2', 'Feature_3',
'Feature_4', 'Feature_5', 'Feature_6', 'Feature_7', 'Feature_8', 'Feature_9',
'Feature_10', 'Feature_11', 'Feature_12']
```

[7]:
```python
plt.figure(figsize=(15, 10))
for i, col in enumerate(features_to_plot):
    plt.subplot(3, 4, i + 1) # Adjust grid size (3x4) as needed
    sns.histplot(df_raw[col], kde=True, bins=50)
    plt.title(col)
    plt.xlabel('')
    plt.ylabel('')
plt.suptitle('Distribution of First 12 Features (Timbre Averages)', y=1.02)
plt.tight_layout()
plt.savefig(os.path.join(PLOT_SAVE_DIR, 'eda_feature_distributions_hist.png'))
plt.show()
```



[8]:
```python
# Box plots can also show distribution and outliers
plt.figure(figsize=(15, 6))
sns.boxplot(data=df_raw[features_to_plot], palette='viridis')
plt.title('Box Plots of First 12 Features (Timbre Averages)')
plt.xlabel('Feature')
```

```python
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(os.path.join(PLOT_SAVE_DIR, 'eda_feature_distributions_box.png'))
plt.show()
```



[9]:
```python
print(f"\nObservations: Examine the plots for skewness, modality (number of peaks), and spread. Many features might appear roughly normally distributed but could have long tails (indicating outliers).")
print("Numerical summary:")
print(df_raw[features_to_plot].describe())
```

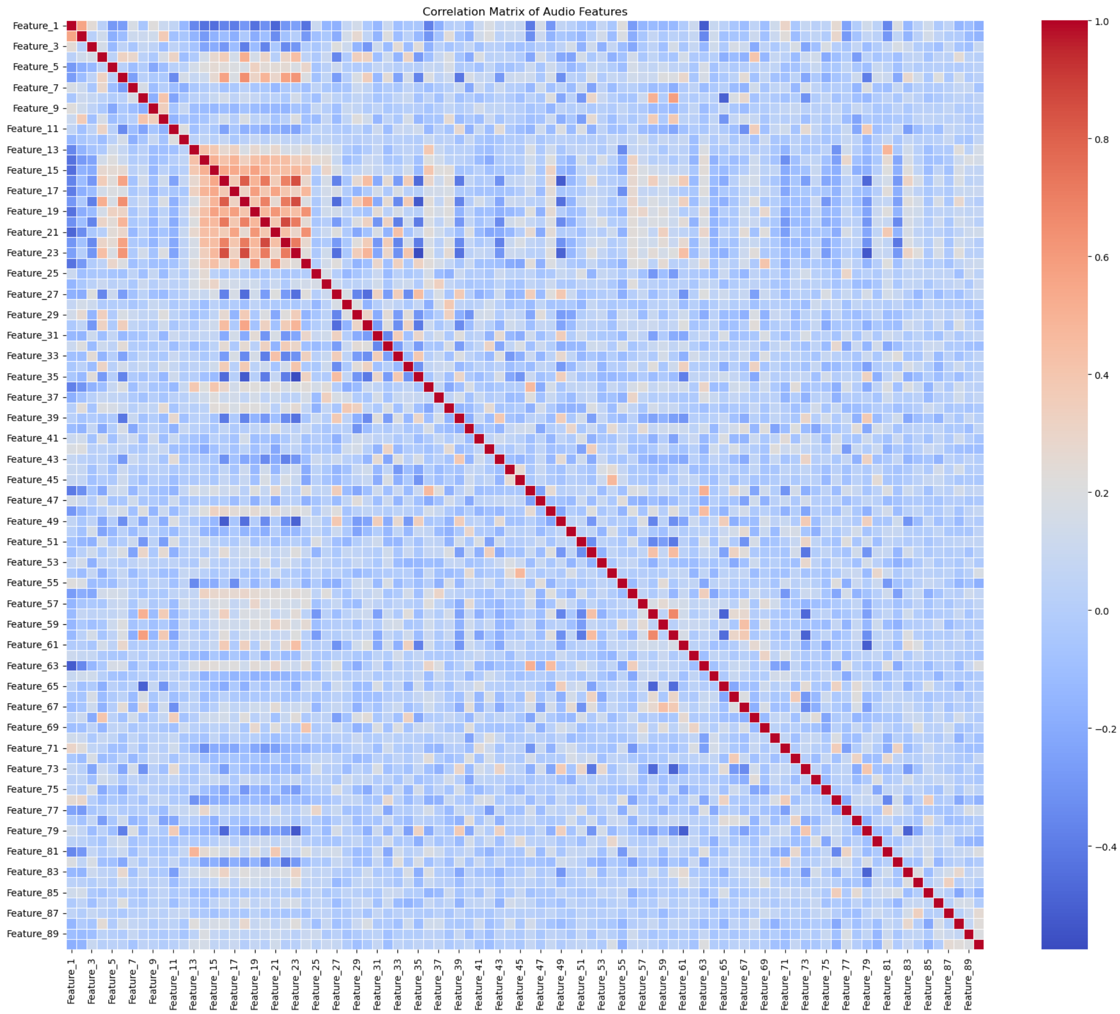
```
Observations: Examine the plots for skewness, modality (number of peaks), and
spread. Many features might appear roughly normally distributed but could have
long tails (indicating outliers).
Numerical summary:
            Feature_1      Feature_2      Feature_3      Feature_4  \
count  515345.000000  515345.000000  515345.000000  515345.000000
mean       43.387126       1.289554       8.658347       1.164124
std         6.067558      51.580351      35.268585      16.322790
min         1.749000    -337.092500    -301.005060    -154.183580
25%        39.954690     -26.059520     -11.462710      -8.487500
50%        44.258500       8.417850      10.476320      -0.652840
75%        47.833890      36.124010      29.764820       8.787540
max        61.970140     384.065730     322.851430     335.771820

            Feature_5      Feature_6      Feature_7      Feature_8  \
count  515345.000000  515345.000000  515345.000000  515345.000000
mean       -6.553601      -9.521975      -2.391089      -1.793236
std        22.860785      12.857751      14.571873       7.963827
min      -181.953370     -81.794290    -188.214000     -72.503850
```

```
25%       -20.666450     -18.440990     -10.780600      -6.468420
50%        -6.007770     -11.188390      -2.046670      -1.736450
75%         7.741870      -2.388960       6.508580       2.913450
max       262.068870     166.236890     172.402680     126.741270

           Feature_9     Feature_10     Feature_11     Feature_12
count  515345.000000  515345.000000  515345.000000  515345.000000
mean        3.727876       1.882385      -0.146527       2.546063
std        10.582861       6.530232       4.370848       8.320190
min      -126.479040     -41.631660     -69.680870     -94.041960
25%        -2.293660      -2.444850      -2.652090      -2.550060
50%         3.822310       1.783520      -0.097950       2.313700
75%         9.961820       6.147220       2.435660       7.360330
max       146.297950      60.345350      88.020820      87.913240
```

[10]:

```
# --- 3. Correlation Analysis ---
print("\n--- 3. Correlation Analysis ---")
# Calculate correlation matrix for numerical features (excluding Year and␣
↪derived decade cols)
feature_cols = [col for col in df_raw.columns if col.startswith('Feature_')]
correlation_matrix = df_raw[feature_cols].corr()

plt.figure(figsize=(18, 15))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False, fmt=".1f",␣
↪linewidths=.5) # annot=True is too crowded for 90 features
plt.title('Correlation Matrix of Audio Features')
plt.tight_layout()
plt.savefig(os.path.join(PLOT_SAVE_DIR, 'eda_correlation_heatmap.png'))
plt.show()
```

```
--- 3. Correlation Analysis ---
```




```
[13]: # Find highly correlated pairs (optional)
threshold = 0.8
highly_correlated = correlation_matrix[abs(correlation_matrix) > threshold]
# Stack to get pairs, remove self-correlation, drop duplicates
corr_pairs = highly_correlated.unstack().sort_values(ascending=False).
↪drop_duplicates()
corr_pairs = corr_pairs[corr_pairs != 1.0] # Remove self-correlations

print(f"\nHighly Correlated Feature Pairs (Threshold > {threshold}):")
if not corr_pairs.empty:
    print(corr_pairs)
else:
    print("No feature pairs found with correlation above the threshold.")
```

```
print(f"\nObservations: Look for blocks of high correlation (positive or␣
↪negative) in the heatmap. High correlation might suggest redundancy, but NNs␣
↪can sometimes handle it. The first 12 features (timbre averages) might show␣
↪some correlation among themselves, as might the covariance features.")
```

```
Highly Correlated Feature Pairs (Threshold > 0.8):
Feature_22  Feature_20    0.865684
Feature_18  Feature_23    0.859569
Feature_16  Feature_23    0.846649
            Feature_18    0.809554
Feature_1   Feature_2          NaN
dtype: float64

Observations: Look for blocks of high correlation (positive or negative) in the
heatmap. High correlation might suggest redundancy, but NNs can sometimes handle
it. The first 12 features (timbre averages) might show some correlation among
themselves, as might the covariance features.
```

[14]:
```
# --- 4. Outlier Analysis (using Box Plots from Feature Distribution) ---
print("\n--- 4. Outlier Analysis ---")
print("Refer back to the box plots generated in the 'Feature Distribution␣
↪Analysis' section.")
print("Box plots visually indicate potential outliers as points beyond the␣
↪'whiskers'.")
```

```
--- 4. Outlier Analysis ---
Refer back to the box plots generated in the 'Feature Distribution Analysis'
section.
Box plots visually indicate potential outliers as points beyond the 'whiskers'.
```

[15]:
```
# Example: Calculate IQR bounds for one feature
feature_example = 'Feature_1'
Q1 = df_raw[feature_example].quantile(0.25)
Q3 = df_raw[feature_example].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

[16]:
```
outliers = df_raw[(df_raw[feature_example] < lower_bound) |␣
↪(df_raw[feature_example] > upper_bound)]
print(f"\nExample Outlier Check for '{feature_example}':")
print(f"  IQR: {IQR:.2f}")
print(f"  Lower Bound (Q1 - 1.5*IQR): {lower_bound:.2f}")
print(f"  Upper Bound (Q3 + 1.5*IQR): {upper_bound:.2f}")
print(f"  Number of potential outliers (based on 1.5*IQR rule):␣
↪{len(outliers)}")
```

```
print(f"  Percentage of potential outliers: {len(outliers) / len(df_raw) * 100:.
↪2f}%")
```

```
Example Outlier Check for 'Feature_1':
  IQR: 7.88
  Lower Bound (Q1 - 1.5*IQR): 28.14
  Upper Bound (Q3 + 1.5*IQR): 59.65
  Number of potential outliers (based on 1.5*IQR rule): 10627
  Percentage of potential outliers: 2.06%
```

[17]:

```
print(f"\nObservations & Handling Strategy:")
print(" - Many features show points beyond the 1.5*IQR whiskers, suggesting the␣
↪presence of outliers.")
print(" - Strategy Decision: For this project, we used StandardScaler in␣
↪data_processing.py. While StandardScaler is sensitive to outliers, deep␣
↪learning models (especially with techniques like Batch Norm, which we might␣
↪test later) can sometimes be relatively robust.")
print(" - Alternative strategies (not implemented here but considered):")
print("   - Use RobustScaler: Scales using percentiles, less sensitive to␣
↪outliers.")
print("   - Clipping: Cap feature values at certain percentiles (e.g., 1st and␣
↪99th).")
print("   - Transformation: Apply log or Box-Cox transforms if features are␣
↪highly skewed.")
print(" - Chosen Approach: Proceed with StandardScaler, acknowledging the␣
↪presence of outliers. We will monitor model performance and may revisit␣
↪outlier handling if necessary.")
```

```
Observations & Handling Strategy:
 - Many features show points beyond the 1.5*IQR whiskers, suggesting the
presence of outliers.
 - Strategy Decision: For this project, we used StandardScaler in
data_processing.py. While StandardScaler is sensitive to outliers, deep learning
models (especially with techniques like Batch Norm, which we might test later)
can sometimes be relatively robust.
 - Alternative strategies (not implemented here but considered):
   - Use RobustScaler: Scales using percentiles, less sensitive to outliers.
   - Clipping: Cap feature values at certain percentiles (e.g., 1st and 99th).
   - Transformation: Apply log or Box-Cox transforms if features are highly
skewed.
 - Chosen Approach: Proceed with StandardScaler, acknowledging the presence of
outliers. We will monitor model performance and may revisit outlier handling if
necessary.
```

[18]:
```
# --- 5. Categorical Features ---
print("\n--- 5. Categorical Features ---")
# Check data types again after loading
print(df_raw.info())
# Identify non-numeric columns (excluding our derived Decade_Name)
categorical_cols = df_raw.select_dtypes(include=['object', 'category']).columns
print(f"\nPotential categorical columns detected (excluding Decade_Name):␣
↪{list(categorical_cols.drop('Decade_Name', errors='ignore'))}")
print("Observations: As expected for this dataset, all original predictor␣
↪columns (Feature_1 to Feature_90) are numeric (float64). No categorical␣
↪feature embedding strategy is required for the predictors.")
```

```
--- 5. Categorical Features ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 515345 entries, 0 to 515344
Data columns (total 94 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Year          515345 non-null  int64
 1   Feature_1     515345 non-null  float64
 2   Feature_2     515345 non-null  float64
 3   Feature_3     515345 non-null  float64
 4   Feature_4     515345 non-null  float64
 5   Feature_5     515345 non-null  float64
 6   Feature_6     515345 non-null  float64
 7   Feature_7     515345 non-null  float64
 8   Feature_8     515345 non-null  float64
 9   Feature_9     515345 non-null  float64
 10  Feature_10    515345 non-null  float64
 11  Feature_11    515345 non-null  float64
 12  Feature_12    515345 non-null  float64
 13  Feature_13    515345 non-null  float64
 14  Feature_14    515345 non-null  float64
 15  Feature_15    515345 non-null  float64
 16  Feature_16    515345 non-null  float64
 17  Feature_17    515345 non-null  float64
 18  Feature_18    515345 non-null  float64
 19  Feature_19    515345 non-null  float64
 20  Feature_20    515345 non-null  float64
 21  Feature_21    515345 non-null  float64
 22  Feature_22    515345 non-null  float64
 23  Feature_23    515345 non-null  float64
 24  Feature_24    515345 non-null  float64
 25  Feature_25    515345 non-null  float64
 26  Feature_26    515345 non-null  float64
 27  Feature_27    515345 non-null  float64
 28  Feature_28    515345 non-null  float64
```

```
29  Feature_29    515345 non-null  float64
30  Feature_30    515345 non-null  float64
31  Feature_31    515345 non-null  float64
32  Feature_32    515345 non-null  float64
33  Feature_33    515345 non-null  float64
34  Feature_34    515345 non-null  float64
35  Feature_35    515345 non-null  float64
36  Feature_36    515345 non-null  float64
37  Feature_37    515345 non-null  float64
38  Feature_38    515345 non-null  float64
39  Feature_39    515345 non-null  float64
40  Feature_40    515345 non-null  float64
41  Feature_41    515345 non-null  float64
42  Feature_42    515345 non-null  float64
43  Feature_43    515345 non-null  float64
44  Feature_44    515345 non-null  float64
45  Feature_45    515345 non-null  float64
46  Feature_46    515345 non-null  float64
47  Feature_47    515345 non-null  float64
48  Feature_48    515345 non-null  float64
49  Feature_49    515345 non-null  float64
50  Feature_50    515345 non-null  float64
51  Feature_51    515345 non-null  float64
52  Feature_52    515345 non-null  float64
53  Feature_53    515345 non-null  float64
54  Feature_54    515345 non-null  float64
55  Feature_55    515345 non-null  float64
56  Feature_56    515345 non-null  float64
57  Feature_57    515345 non-null  float64
58  Feature_58    515345 non-null  float64
59  Feature_59    515345 non-null  float64
60  Feature_60    515345 non-null  float64
61  Feature_61    515345 non-null  float64
62  Feature_62    515345 non-null  float64
63  Feature_63    515345 non-null  float64
64  Feature_64    515345 non-null  float64
65  Feature_65    515345 non-null  float64
66  Feature_66    515345 non-null  float64
67  Feature_67    515345 non-null  float64
68  Feature_68    515345 non-null  float64
69  Feature_69    515345 non-null  float64
70  Feature_70    515345 non-null  float64
71  Feature_71    515345 non-null  float64
72  Feature_72    515345 non-null  float64
73  Feature_73    515345 non-null  float64
74  Feature_74    515345 non-null  float64
75  Feature_75    515345 non-null  float64
76  Feature_76    515345 non-null  float64
```

```
 77  Feature_77    515345 non-null  float64
 78  Feature_78    515345 non-null  float64
 79  Feature_79    515345 non-null  float64
 80  Feature_80    515345 non-null  float64
 81  Feature_81    515345 non-null  float64
 82  Feature_82    515345 non-null  float64
 83  Feature_83    515345 non-null  float64
 84  Feature_84    515345 non-null  float64
 85  Feature_85    515345 non-null  float64
 86  Feature_86    515345 non-null  float64
 87  Feature_87    515345 non-null  float64
 88  Feature_88    515345 non-null  float64
 89  Feature_89    515345 non-null  float64
 90  Feature_90    515345 non-null  float64
 91  Decade_Start  515345 non-null  int64
 92  Decade_Label  515345 non-null  int64
 93  Decade_Name   515345 non-null  object
dtypes: float64(90), int64(3), object(1)
memory usage: 369.6+ MB
None

Potential categorical columns detected (excluding Decade_Name): []
Observations: As expected for this dataset, all original predictor columns
(Feature_1 to Feature_90) are numeric (float64). No categorical feature
embedding strategy is required for the predictors.
```

[19]:

```python
# --- Summary of EDA Findings ---
print("\n--- Summary of Key EDA Findings ---")
print("1.  **Target Variable (Decade):** Heavily imbalanced, dominated by 2000s␣
 ↪and 1990s.")
print("2.  **Features:** All 90 predictor features are numeric (float).")
print("3.  **Distributions:** Feature distributions vary. Some are roughly␣
 ↪normal, others might be skewed or have multiple peaks (visual inspection␣
 ↪needed per feature).")
print("4.  **Correlations:** Some correlations exist between features,␣
 ↪particularly noted visually within blocks (e.g., early timbre features,␣
 ↪later covariance features). No extremely high correlations (>0.95) jumped␣
 ↪out immediately in the sample check, but moderate correlations are present.")
print("5.  **Outliers:** Potential outliers detected in many features based on␣
 ↪visual inspection of box plots and IQR checks.")
print("6.  **Missing Values:** No missing values detected by `df.info()`␣
 ↪(consistent with dataset description).")
print("7.  **Preprocessing Decisions (Recap):**")
print("    - Decade binning successfully converted regression to classification.
 ↪")
print("    - Stratified splitting addressed the class imbalance during data␣
 ↪partitioning.")
```

```
print("    - StandardScaler was used for feature scaling, acknowledging outlier␣
  ↪presence.")
print("    - No categorical encoding needed for predictors.")
```

```
--- Summary of Key EDA Findings ---
1.  **Target Variable (Decade):** Heavily imbalanced, dominated by 2000s and
1990s.
2.  **Features:** All 90 predictor features are numeric (float).
3.  **Distributions:** Feature distributions vary. Some are roughly normal,
others might be skewed or have multiple peaks (visual inspection needed per
feature).
4.  **Correlations:** Some correlations exist between features, particularly
noted visually within blocks (e.g., early timbre features, later covariance
features). No extremely high correlations (>0.95) jumped out immediately in the
sample check, but moderate correlations are present.
5.  **Outliers:** Potential outliers detected in many features based on visual
inspection of box plots and IQR checks.
6.  **Missing Values:** No missing values detected by `df.info()` (consistent
with dataset description).
7.  **Preprocessing Decisions (Recap):**
    - Decade binning successfully converted regression to classification.
    - Stratified splitting addressed the class imbalance during data
partitioning.
    - StandardScaler was used for feature scaling, acknowledging outlier
presence.
    - No categorical encoding needed for predictors.
```