Music Genre Predictor Change Log

18/02/25

- For this model I am going to use K Means Clustering (KMC).
- This dataset was obtained from kaggle (https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db/data) however it is data that was directly pulled from Spotify's API.
- This dataset contains roughly 232,000 rows of data.
- I chose to use KMC for this model as it handles larger dataset well and it works particularly well with numerical data.
- My aim for this is to be able to predict what genre a song is in based off of the data I give it.

22/02/25

- I researched into the columns and what all of them mean so that I can get a better understanding of my dataset.
- Ran into an issue with my clustering as it isnt working correctly.
- I added a standard scaler so that the data wouldn't be as affected by outliers and all the features use the same scale.

25/02/05

- I added the elbow method to help try and rectify the issue but this unfortunately didn't have an effect.
- I added PCA to see if reducing the features would help fix the issue but unfortunately this had no effect either.
- I then tried to take a sample of my data to see if that was the issue but this didn't fix it either as the clustering still wasn't working.
- Despite adding these new features and methods to try and get my data to fit to the model I had no luck.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

ds = pd.read_csv('SpotifyFeatures.csv')
print(ds.columns)
# This is just importing the data and printing the first 5 rows of the
data to ensure that the data is imported correctly - AD
```

```
Index(['genre', 'artist_name', 'track_name', 'track_id', 'popularity',
       'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence'],
      dtype='object')
```

Column Breakdown:

- genre: This is the target value, genre that the song belongs too
- artist_name: The name of the artist who performed the track.
- track_name: The title of the track.
- track_id: A unique identifier assigned to the track by Spotify.
- popularity: A numerical value indicating the track's popularity on Spotify.
- acousticness: A measure from 0 to 1 of the likelihood that the track is acoustic.
- danceability: A measure from 0 to 1 of how suitable the track is for dancing.
- duration_ms: The length of the track in milliseconds.
- energy: A measure from 0 to 1 of the track's intensity and activity.
- instrumentalness: A measure from 0 to 1 indicating the likelihood that the track is instrumental.
- key: The key in which the track is composed, represented as an number.
- liveness: A measure from 0 to 1 indicating the presence of a live audience in the recording.
- loudness: The overall loudness of the track in decibels.
- mode: Indicates the modality of the track: 1 for major, 0 for minor.
- speechiness: A measure from 0 to 1 of the presence of spoken words in the track.
- tempo: The speed of the track in beats per minute.
- time_signature: An integer representing the number of beats in each bar.
- valence: A measure from 0 to 1 of the musical positiveness conveyed by the track.

This took me about 20 minutes to fully understand all of these columns and what they mean.

```
from sklearn.preprocessing import LabelEncoder
# Drop the unnecessary columns

ds = ds.drop(columns=["track_id", "track_name", "artist_name",
"time_signature"])

# Create and fit label encoder
encoder = LabelEncoder()
ds["genre_encoded"] = encoder.fit_transform(ds["genre"])
ds["Mode_encoded"] = encoder.fit_transform(ds["mode"])
ds["Key_encoded"] = encoder.fit_transform(ds["key"])

# Retrieve the mapping
genre_mapping = dict(zip(encoder.classes_,
encoder.transform(encoder.classes_)))

# Display mapping
print(genre_mapping)
```

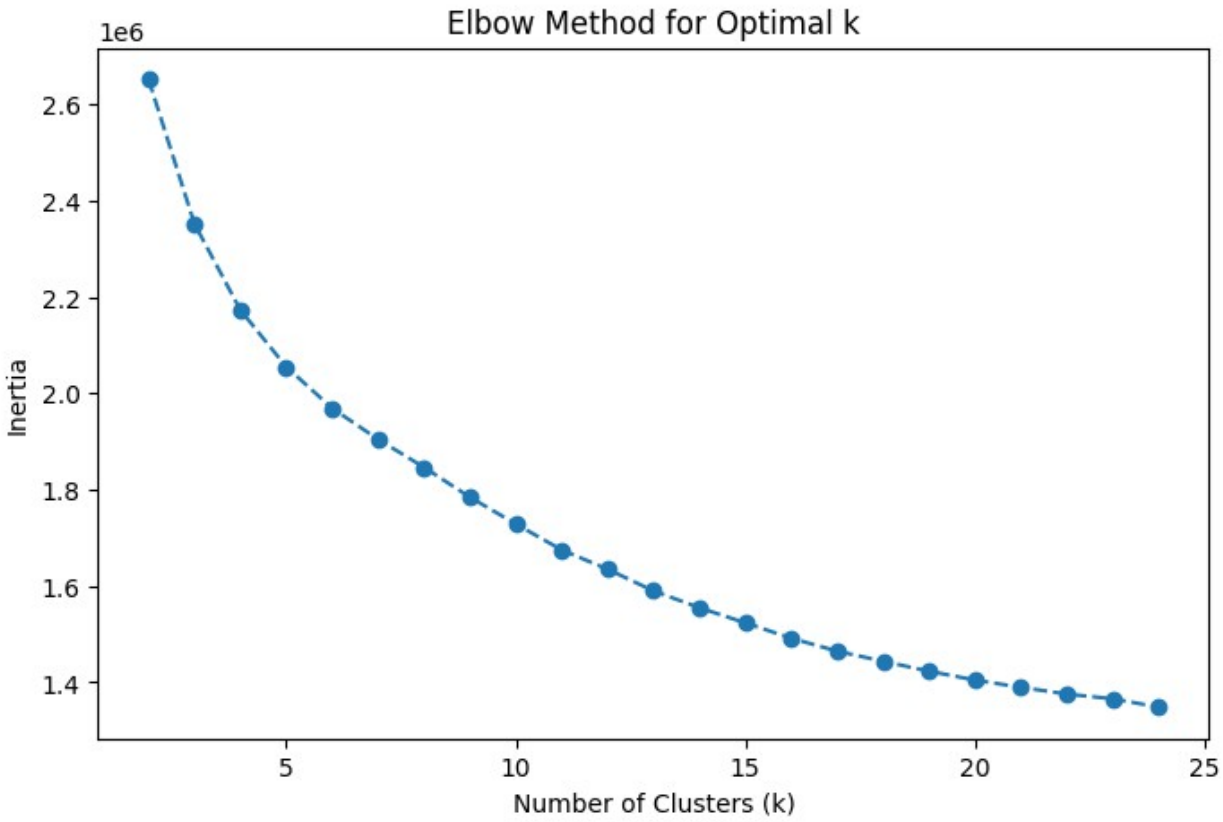
```
{'A': np.int64(0), 'A#': np.int64(1), 'B': np.int64(2), 'C':
np.int64(3), 'C#': np.int64(4), 'D': np.int64(5), 'D#': np.int64(6),
'E': np.int64(7), 'F': np.int64(8), 'F#': np.int64(9), 'G':
np.int64(10), 'G#': np.int64(11)}
```

I am dropping the track ID, track name and artist name as they arent predictive fields and dont really give me any benefit.

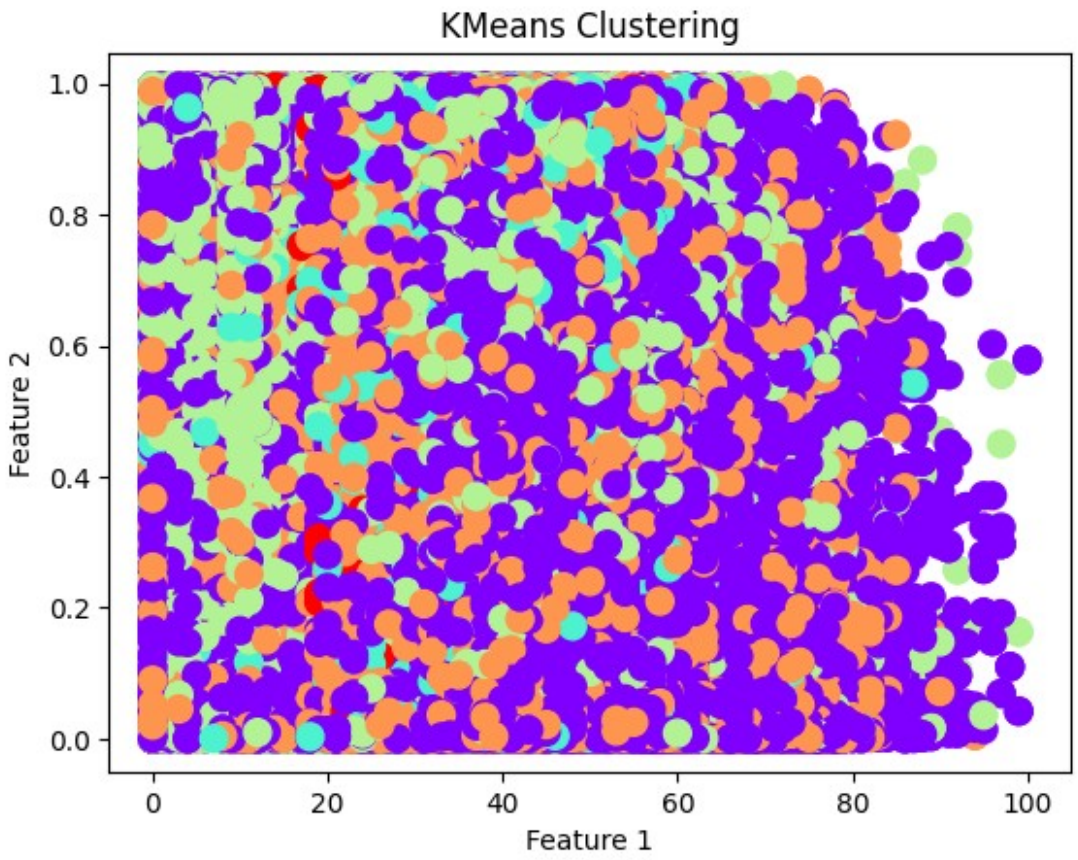
I am then label encoding the genre field as KMC works better with numerical values, as you can see in the output this works by assigning each genre a value that correlates to it.

I learned how to do this by reading through Sk-Learns docs here - (https://scikit-learn.org/0.17/modules/generated/sklearn.preprocessing.LabelEncoder.html)

```
from sklearn.preprocessing import StandardScaler
X = ds[['popularity', 'acousticness', 'danceability', 'duration_ms',
'energy',
        'instrumentalness', 'Key_encoded', 'liveness', 'loudness',
'Mode_encoded',
        'speechiness', 'tempo', 'valence', 'genre_encoded']]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Standardize features

from sklearn.cluster import KMeans

inertia = []
k_range = range(2, 25)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()

#  This method was explained to me by ChatGPT, it is a method to
determine the optimal number of clusters for the data. The optimal
point to choose for the cluster is where the slope starts tailing off
so in this example it would be 6/7 - AD
```




```
import numpy as np
import matplotlib.pyplot as plt

# Convert DataFrame to NumPy array
X_np = X.to_numpy() # This line was added by chatGPT to convert the
dataframe to a numpy array as I was having issues figuring out why it
wouldn't plot - AD

est = KMeans(n_clusters=6, random_state=42)
est.fit(X_np)

y_kmeans = est.predict(X_np)

plt.scatter(X_np[:, 0], X_np[:, 1], c=y_kmeans, s=100, cmap='rainbow')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("KMeans Clustering")
plt.show()
```



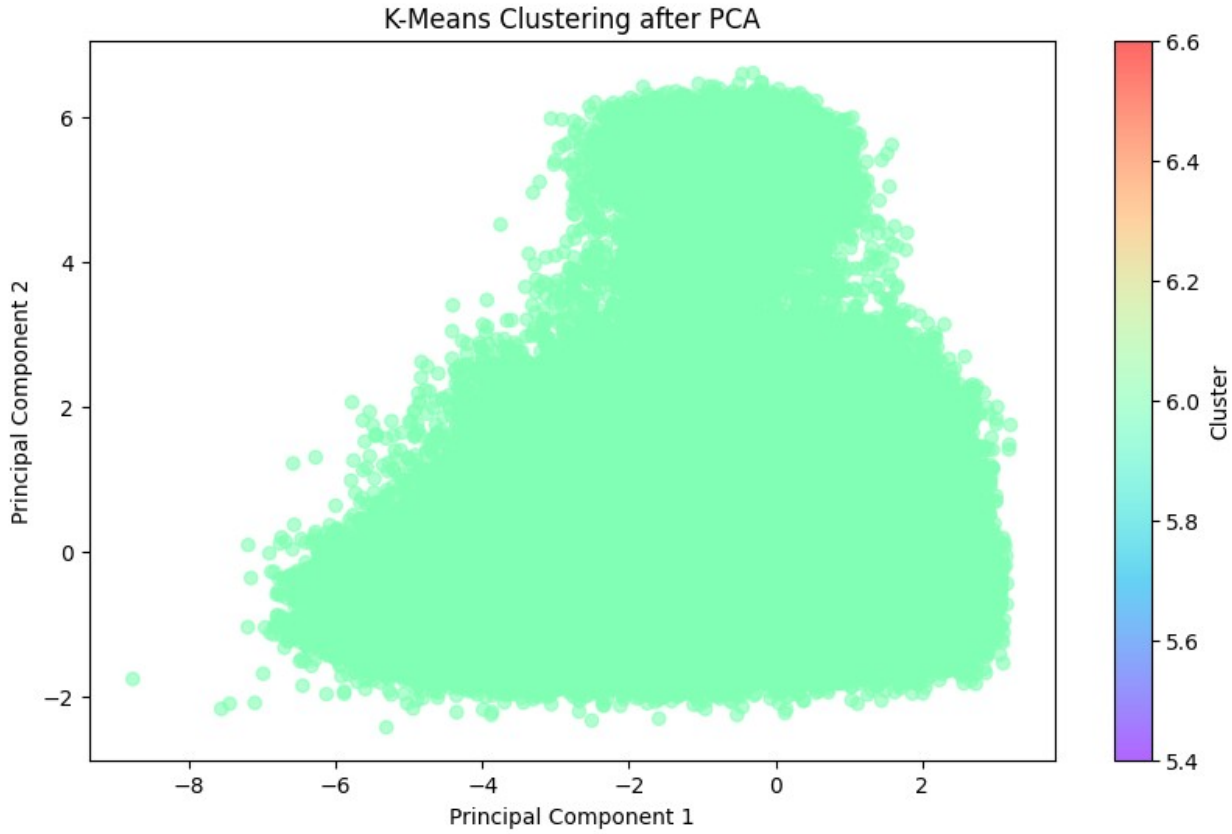
As can be seen in the scatter plot above, the data isn't clustering very well. I believe this to be because there are too many columns in my dataset.

However I do want to research and see if I can find any methods that might help fix the current issue with the data not clustering.

ChatGPT gave me the code block below. Principal Component Analysis (PCA) is a way to simplify data while keeping the most important patterns. It works by finding patterns, creating new features and then using these new features to base its research/learning off of.

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Apply PCA to reduce to 2 components for visualisation
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)  # Use standardized features

# Convert to DataFrame for easy plotting
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3', 'PC4',
'PC5'])
df_pca['Cluster'] = 6  # Add cluster labels

# Scatter plot of PCA components
plt.figure(figsize=(10, 6))
```

```
plt.scatter(df_pca['PC1'], df_pca['PC2'], c=df_pca['Cluster'],
cmap='rainbow', alpha=0.6)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Means Clustering after PCA")
plt.colorbar(label="Cluster")
plt.show()
```



Unfortunately this hasnt worked either, I do believe the issue to be with my dataset as there are so many genres included within the dataset, 26 to be precise.

I am going to try and reduce the dataset to a small sample of the full dataset with 5 or 6 genres to see how the model performs with the smaller amount of data.

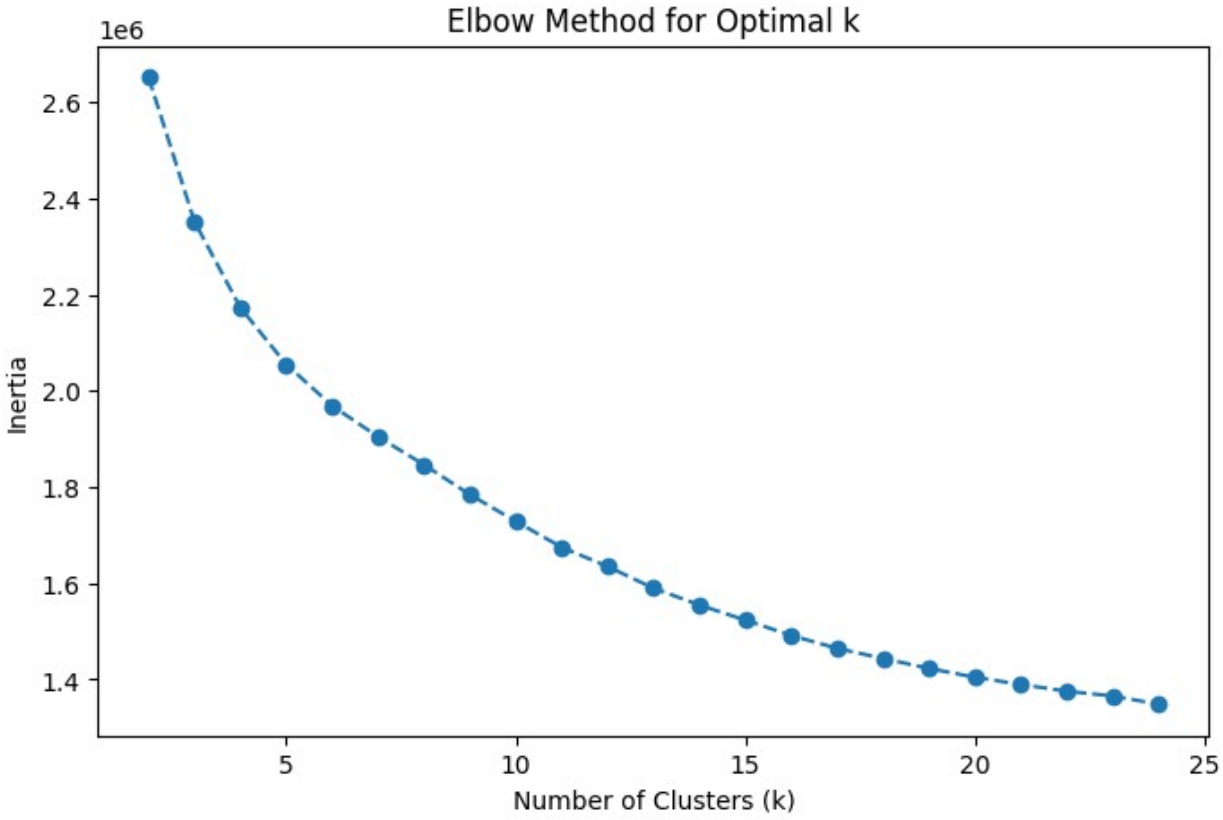
I will be following the exact same process as before but just with the filtered dataset.

```
unique_genres = ds['genre'].unique()
sample_genres = unique_genres[:6]

sample_ds = ds[ds['genre'].isin(sample_genres)]

print(sample_ds.head())
```

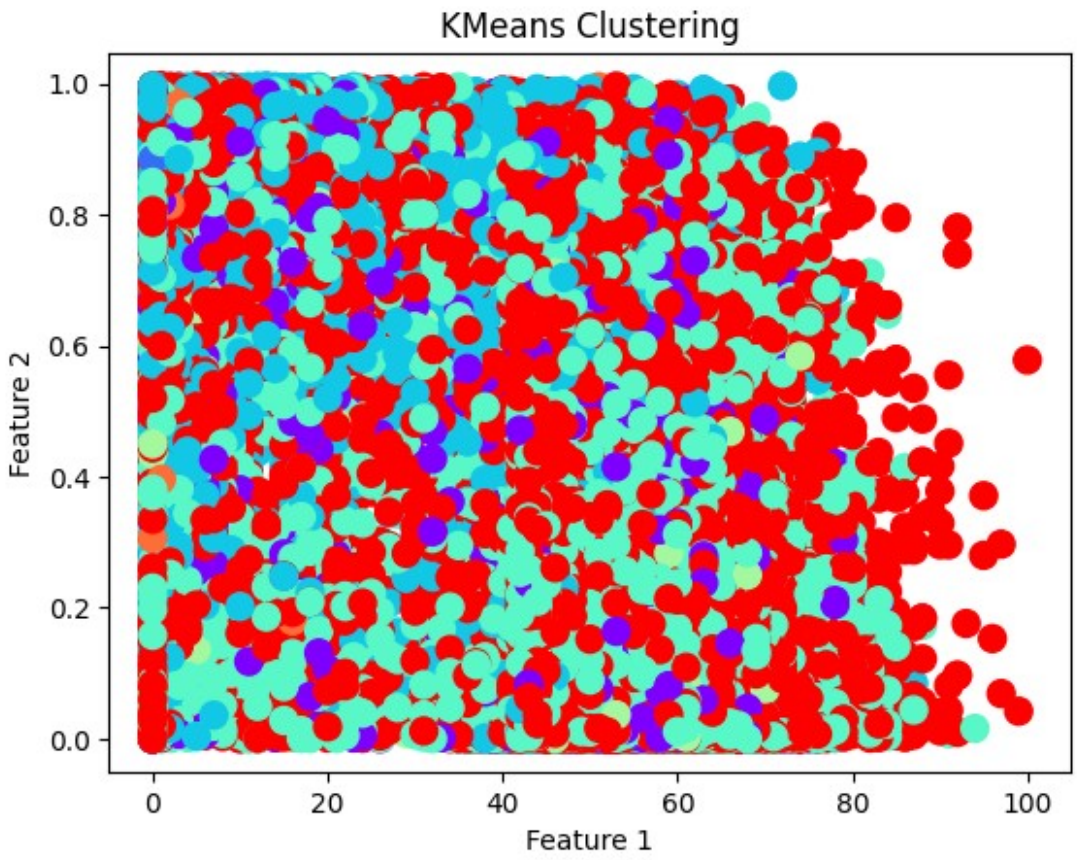
```
   genre  popularity  acousticness  danceability  duration_ms  energy
\
0  Movie           0         0.611         0.389        99373   0.910

1  Movie           1         0.246         0.590       137373   0.737

2  Movie           3         0.952         0.663       170267   0.131

3  Movie           0         0.703         0.240       152427   0.326

4  Movie           4         0.950         0.331        82625   0.225


   instrumentalness key  liveness  loudness   mode  speechiness
tempo  \
0             0.000  C#    0.3460    -1.828  Major       0.0525
166.969
1             0.000  F#    0.1510    -5.559  Minor       0.0868
174.003
2             0.000   C    0.1030   -13.879  Minor       0.0362
99.488
3             0.000  C#    0.0985   -12.178  Major       0.0395
171.758
4             0.123   F    0.2020   -21.150  Major       0.0456
140.576

   valence  genre_encoded  Mode_encoded  Key_encoded
0    0.814             15             0            4
1    0.816             15             1            9
2    0.368             15             1            3
3    0.227             15             0            4
4    0.390             15             0            8
```

```
Sample_X = sample_ds[['popularity', 'acousticness', 'danceability',
'duration_ms', 'energy',
        'instrumentalness', 'Key_encoded', 'liveness', 'loudness',
'Mode_encoded',
        'speechiness', 'tempo', 'valence', 'genre_encoded']]

scaler = StandardScaler()
Sample_X_Scaled = scaler.fit_transform(X)  # Standardize features

inertia = []
k_range = range(2, 25)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(Sample_X_Scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
```

```
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```




```
# Convert DataFrame to NumPy array
Sample_X_np = Sample_X.to_numpy() # This line was added by chatGPT to
convert the dataframe to a numpy array as I was having issues figuring
out why it wouldn't plot - AD

est = KMeans(n_clusters=8, random_state=42)
est.fit(X_np)

y_kmeans = est.predict(Sample_X_np)

plt.scatter(Sample_X_np[:, 0], Sample_X_np[:, 1], c=y_kmeans, s=100,
cmap='rainbow')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("KMeans Clustering")
plt.show()
```



Final Conclusion

- As can be seen above the dataset isn't working well with this clustering model.
- I believe this is due to the high amount of columns and also the high amount of genres that are in the dataset.
- I have tried multiple different methods to try and fix this issue such as PCA, Elbow method to get the optimal amount of clusters, encoding different fields and Scalers
- If I had more time I would like to try getting silhouette scores for a lot of different cluster amounts to see how they are all performing.
- My next step now is to retry this model with a different dataset.