```python
import pandas as pd
data = pd.read_csv('/content/all_songs_data.csv')
print(data.head())

```

```
                                 Album  \
0                Battle of New Orleans   
1                           That’s All   
2         “Mr Personality’s” 15 Big Hits   
3  The Greatest Hits Of Frankie Avalon   
4            Paul Anka Sings His Big 15   

                                           Album URL          Artist  \
0  https://genius.com/albums/Johnny-horton/Battle...   Johnny Horton   
1   https://genius.com/albums/Bobby-darin/That-s-all     Bobby Darin   
2  https://genius.com/albums/Lloyd-price/Mr-perso...     Lloyd Price   
3  https://genius.com/albums/Frankie-avalon/The-g...  Frankie Avalon   
4  https://genius.com/albums/Paul-anka/Paul-anka-...       Paul Anka   

  Featured Artists                                             Lyrics  \
0               []  [Verse 1] In 1814 we took a little trip Along ...   
1               []  Oh the shark, babe Has such teeth, dear And he...   
2               []  Over and over I tried to prove my love to you ...   
3               []  Hey, Venus! Oh, Venus!  Venus, if you will Ple...   
4               []  I'm just a lonely boy Lonely and blue I'm all ...   

                                               Media  Rank Release Date  \
0  [{'native_uri': 'spotify:track:0dwpdcQkeZqpuoA...     1   1959-04-01   
1  [{'native_uri': 'spotify:track:3E5ndyOfO6vFDEI...     2          NaN   
2  [{'provider': 'youtube', 'start': 0, 'type': '...     3          NaN   
3                                                 []     4          NaN   
4                                                 []     5          NaN   

                Song Title  \
0  The Battle Of New Orleans   
1             Mack The Knife   
2                Personality   
3                      Venus   
4                 Lonely Boy   

                                            Song URL  \
0  https://genius.com/Johnny-horton-the-battle-of...   
1  https://genius.com/Bobby-darin-mack-the-knife-...   
2  https://genius.com/Lloyd-price-personality-lyrics   
3     https://genius.com/Frankie-avalon-venus-lyrics   
4     https://genius.com/Paul-anka-lonely-boy-lyrics   

                                             Writers    Year  
0  [{'api_path': '/artists/561913', 'header_image...  1959.0  
1  [{'api_path': '/artists/218851', 'header_image...  1959.0  
2  [{'api_path': '/artists/355804', 'header_image...  1959.0  
3  [{'api_path': '/artists/1113175', 'header_imag...  1959.0  
4                                                 []  1959.0  
```

```python
# Descriptive statistics for numerical columns
print(data.describe())

# Descriptive statistics for categorical columns
print(data.describe(include='object'))

```

```
              Rank         Year
count  6500.000000  6500.000000
mean     50.500000  1991.000000
std      28.868291    18.763106
min       1.000000  1959.000000
```

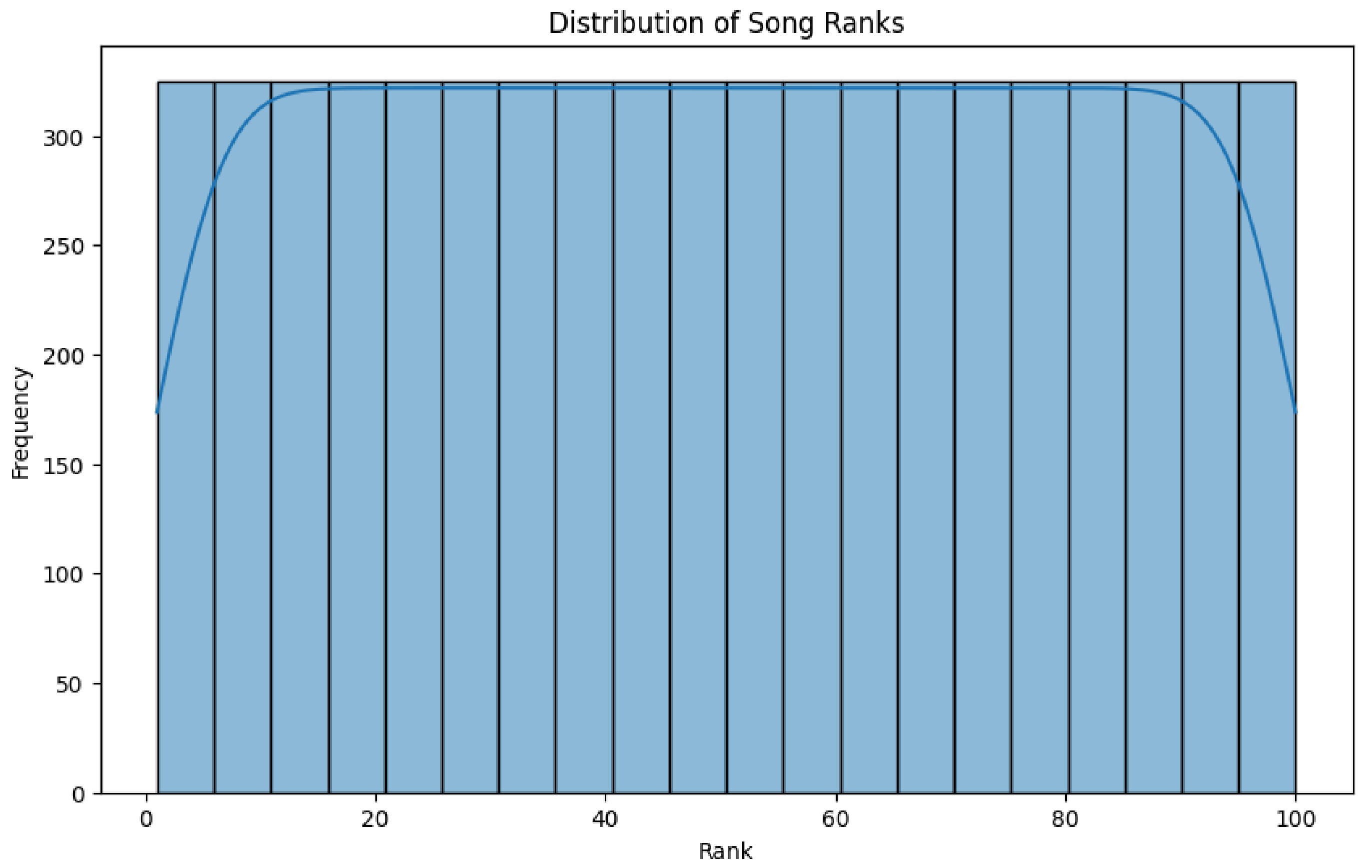
```
25%      25.750000  1975.000000
50%      50.500000  1991.000000
75%      75.250000  2007.000000
max     100.000000  2023.000000
                Album                                         Album URL  \
count            6036                                              6036
unique           4202                                              4285
top     Greatest Hits  https://genius.com/albums/Morgan-wallen/One-th...
freq               21                                                 9

         Artist Featured Artists          Lyrics Media Release Date  \
count      6500             6384            6384  6384         4563
unique     3181              612            6044  5054         3233
top     Madonna               []  [Instrumental]    []   2022-05-06
freq         35             5492              21  1043           10

       Song Title                                         Song URL Writers
count        6500                                             6384    6384
unique       5798                                             6065    4184
top          Stay  https://genius.com/Billboard-hot-rap-songs-cha...      []
freq            7                                                5     971
```

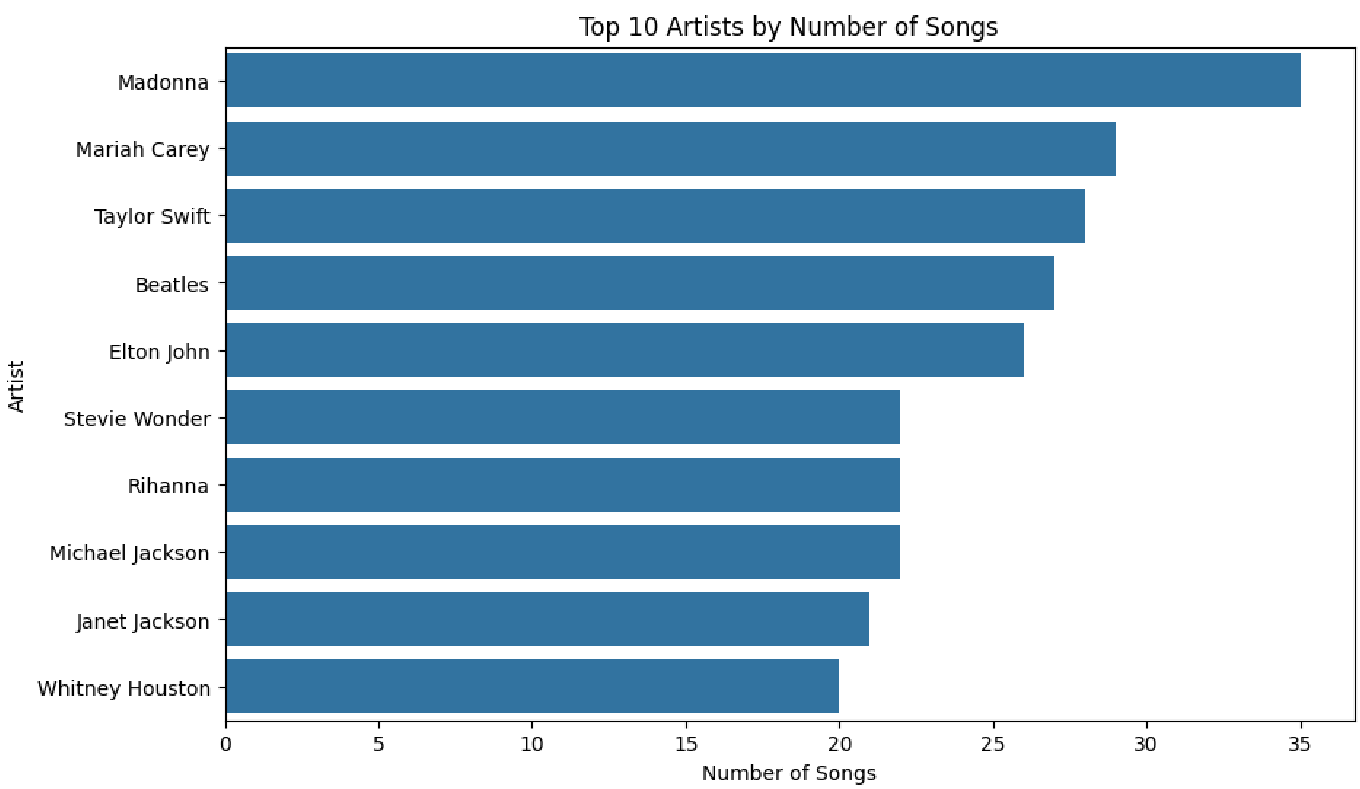
```python
rank_stats = data['Rank'].describe()
rank_frequency = data['Rank'].value_counts().sort_index()
top_artists = data['Artist'].value_counts().head(10)

print("Descriptive Statistics for Song Ranks:")
print(rank_stats)
print("\nFrequency of Songs by Rank:")
print(rank_frequency)
print("\nTop 10 Artists by Number of Songs:")
print(top_artists)
```

```
Descriptive Statistics for Song Ranks:
count    6500.000000
mean       50.500000
std        28.868291
min         1.000000
25%        25.750000
50%        50.500000
75%        75.250000
max       100.000000
Name: Rank, dtype: float64

Frequency of Songs by Rank:
Rank
1      65
2      65
3      65
4      65
5      65
       ..
96     65
97     65
98     65
99     65
100    65
Name: count, Length: 100, dtype: int64

Top 10 Artists by Number of Songs:
Artist
Madonna          35
Mariah Carey     29
Taylor Swift     28
Beatles          27
Elton John       26
```

```
Stevie Wonder      22
Rihanna            22
Michael Jackson    22
Janet Jackson      21
Whitney Houston    20
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(data['Rank'], bins=20, kde=True)
plt.title('Distribution of Song Ranks')
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.show()

top_artists = data['Artist'].value_counts().head(10)
plt.figure(figsize=(10, 6))
sns.barplot(x=top_artists.values, y=top_artists.index)
plt.title('Top 10 Artists by Number of Songs')
plt.xlabel('Number of Songs')
plt.ylabel('Artist')
plt.show()

```


Distribution of Song Ranks
Frequency
Rank
0
50
100
150
200
250
300
0
20
40
60
80
100

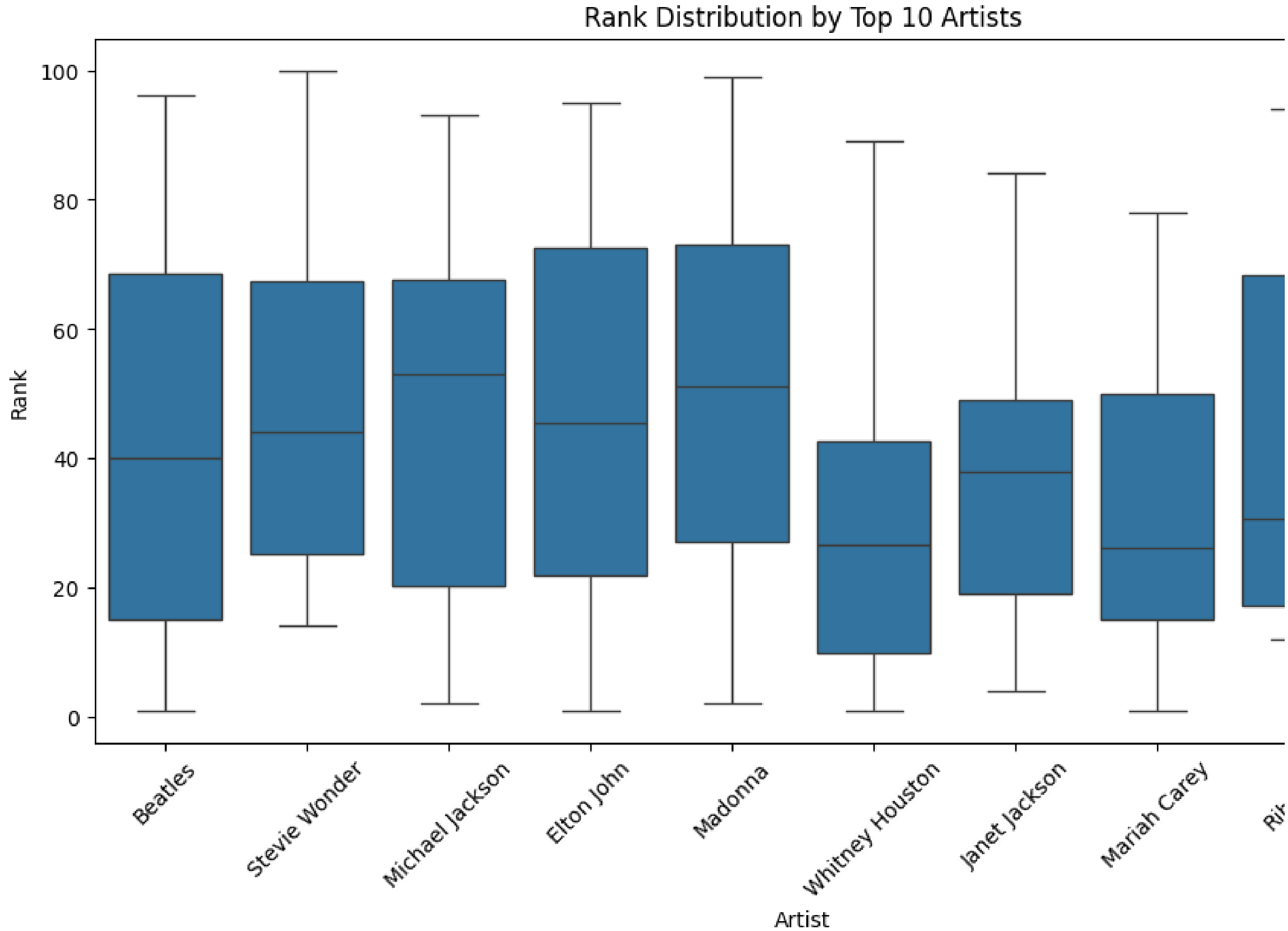

Top 10 Artists by Number of Songs
Artist
Number of Songs
Madonna
Mariah Carey
Taylor Swift
Beatles
Elton John
Stevie Wonder
Rihanna
Michael Jackson
Janet Jackson
Whitney Houston
0
5
10
15
20
25
30
35

```
# Converting 'Release Date' to datetime
data['Release Date'] = pd.to_datetime(data['Release Date'])

data['Year'] = data['Release Date'].dt.year

```

```
rank_year_corr = data[['Year', 'Rank']].groupby('Year').mean().reset_index()
rank_year_corr['Rank'] = rank_year_corr['Rank'].round(2)

correlation = data['Year'].corr(data['Rank'])

top_artists = data['Artist'].value_counts().head(10).index
rank_distribution_top_artists = data[data['Artist'].isin(top_artists)].groupby('Artist')['Rank'].desc

print("Average Rank by Year:")
print(rank_year_corr)
print("\nCorrelation between Year and Rank:")
print(correlation)
print("\nRank Distribution by Top 10 Artists:")
print(rank_distribution_top_artists)

```

```
Average Rank by Year:
      Year   Rank
0   1877.0  26.00
1   1922.0  70.00
2   1955.0  52.00
3   1957.0  77.00
4   1958.0  72.00
..     ...    ...
66  2020.0  52.36
67  2021.0  45.94
68  2022.0  48.67
69  2023.0  53.90
70  2024.0  82.00

[71 rows x 2 columns]

Correlation between Year and Rank:
0.04014067434061861

Rank Distribution by Top 10 Artists:
                 count       mean        std   min    25%   50%    75%    max
Artist
Beatles           27.0  42.777778  31.624398   1.0  15.00  40.0  68.50   96.0
Elton John        26.0  47.038462  29.200659   1.0  21.75  45.5  72.50   95.0
Janet Jackson     21.0  38.619048  21.973794   4.0  19.00  38.0  49.00   84.0
Madonna           35.0  49.285714  28.678308   2.0  27.00  51.0  73.00   99.0
Mariah Carey      29.0  33.724138  24.335301   1.0  15.00  26.0  50.00   78.0
Michael Jackson   22.0  47.409091  30.316833   2.0  20.25  53.0  67.50   93.0
Rihanna           22.0  42.090909  29.012163  12.0  17.25  30.5  68.25   94.0
Stevie Wonder     22.0  48.318182  26.141349  14.0  25.25  44.0  67.25  100.0
Taylor Swift      28.0  43.642857  27.214123   4.0  18.00  40.0  71.75   89.0
Whitney Houston   20.0  29.200000  23.625588   1.0   9.75  26.5  42.50   89.0
```

```
top_artists = data['Artist'].value_counts().head(10).index
plt.figure(figsize=(12, 6))
sns.boxplot(x='Artist', y='Rank', data=data[data['Artist'].isin(top_artists)])
plt.title('Rank Distribution by Top 10 Artists')
plt.xlabel('Artist')
plt.ylabel('Rank')
plt.xticks(rotation=45)
plt.show()

```



At this stage, I am moving on to exploring the categorical data and cleaning/preprocessing the text

```
# Checking for missing values
print(f"Missing Lyrics: {data['Lyrics'].isnull().sum()}")
empty_lyrics = data[data['Lyrics'].apply(lambda x: isinstance(x, str) and len(x) == 0)]
print(f"Empty Lyrics: {empty_lyrics.shape[0]}")
invalid_lyrics = data[data['Lyrics'].apply(lambda x: not isinstance(x, str))]
print(f"Invalid Lyrics (Non-string values): {invalid_lyrics.shape[0]}")

```

```
Missing Lyrics: 116
Empty Lyrics: 0
Invalid Lyrics (Non-string values): 116
```

```
# Removing missing values
data_cleaned = data.dropna(subset=['Lyrics'])

data_cleaned = data_cleaned[data_cleaned['Lyrics'].apply(lambda x: isinstance(x, str))]

print(f"Data after cleaning: {data_cleaned.shape[0]} rows")
print(data_cleaned['Lyrics'].isnull().sum())  # Check if there are any missing lyrics

```

```
Data after cleaning: 6384 rows
0
```

```
!pip install spacy
!python -m spacy download en_core_web_sm
```

```
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from la
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from smart-open<8.0.
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-i
Collecting en-core-web-sm==3.7.1
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.7.1/en_cor
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 73.6 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.8.0,>=3.7.2 in /usr/local/lib/python3.10/dist-packages (from e
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.8.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.10/dist-packages (from la
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: pydantic-core==2.27.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from re
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from re
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from th
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from typer<1.
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from ty
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer<1
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from la
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from smart-open<8.0.
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-i
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
Start coding or generate with AI.
```

Next, I am loading the necesssary tools to preprocess the lyrical data

```
import spacy
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('stopwords')



# Implementing custom stopwords
custom_stopwords = set(stopwords.words('english')).union({'chorus', 'verse', 'bridge', 'hook', 'intro
nlp = spacy.load("en_core_web_sm")
lemmatizer = WordNetLemmatizer()

#in the first round, i noticed contractions were not removed properly so I used a dictionary to ensur
def expand_contractions(text):
    contractions_dict = {
        "don't": "do not", "can't": "cannot", "won't": "will not", "didn't": "did not",
        "isn't": "is not", "aren't": "are not", "wasn't": "was not", "weren't": "were not",
        "hasn't": "has not", "haven't": "have not", "hadn't": "had not", "doesn't": "does not",
        "didn't": "did not", "couldn't": "could not", "shouldn't": "should not", "mightn't": "might n
        "mustn't": "must not", "let's": "let us", "i'm": "i am", "you're": "you are", "he's": "he is",
        "she's": "she is", "it's": "it is", "we're": "we are", "they're": "they are", "that's": "that
        "what's": "what is", "who's": "who is", "where's": "where is", "how's": "how is"
    }
    for word, expansion in contractions_dict.items():
        text = re.sub(r'\b' + word + r'\b', expansion, text)
    return text

def preprocess_text(text):
    text = expand_contractions(text)

    # Converting to lowercase
    text = text.lower()

    # Removing non-alphanumeric characters (keeping spaces and words)
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenizing the text using spaCy
    doc = nlp(text)

    words = [token.text for token in doc if token.text not in custom_stopwords and not token.is_punct

    # Lemmatizing words
    words = [lemmatizer.lemmatize(word) for word in words]

    # Rejoining words
    cleaned_text = ' '.join(words)

    return cleaned_text
data_cleaned['cleaned_lyrics'] = data_cleaned['Lyrics'].apply(preprocess_text)

print(data_cleaned[['Song Title', 'Lyrics', 'cleaned_lyrics']].head())
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
                 Song Title  \
0  The Battle Of New Orleans
1             Mack The Knife
2                Personality
```

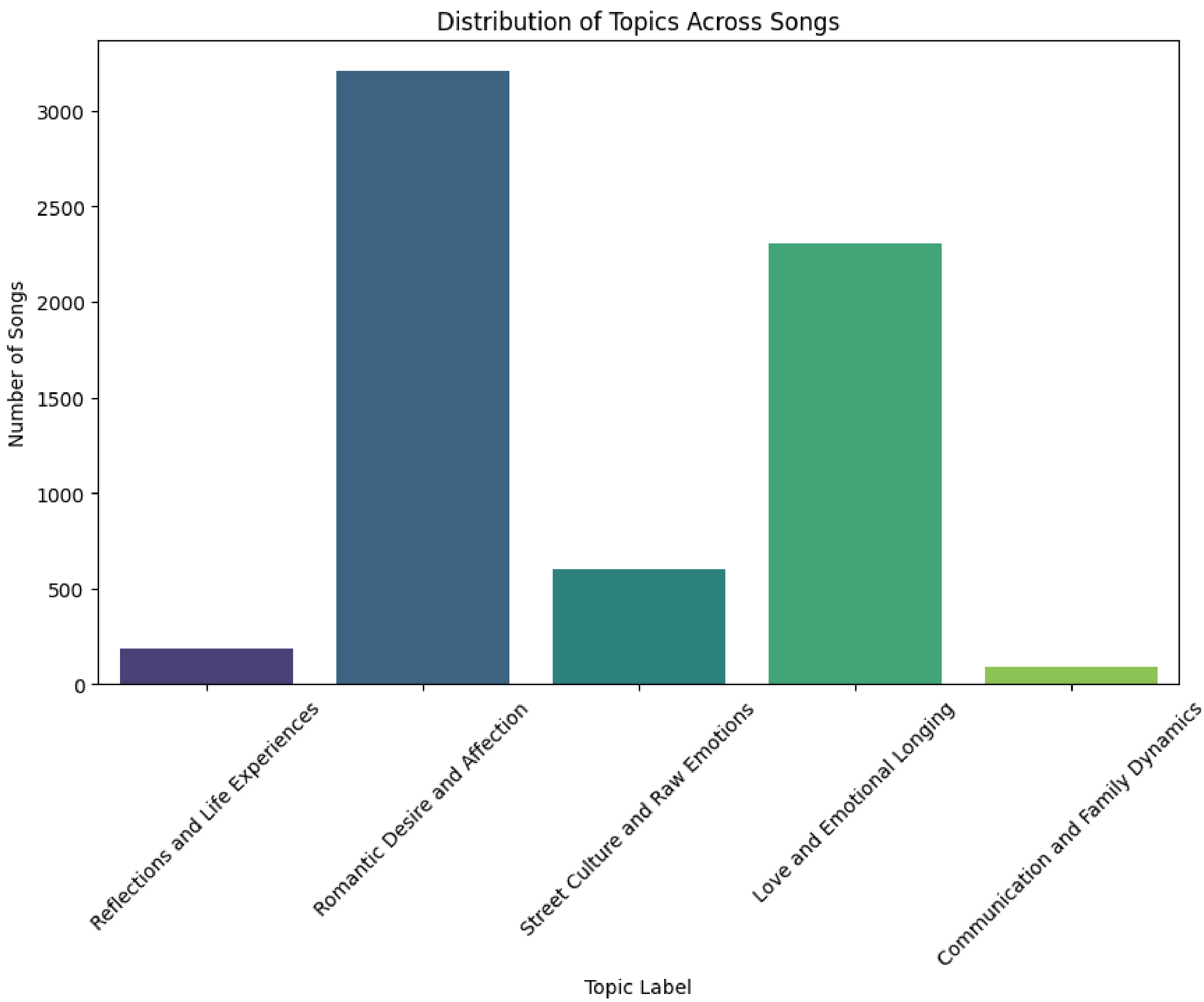
```
3                      Venus
4                 Lonely Boy

                                              Lyrics  \
0  [Verse 1] In 1814 we took a little trip Along ...
1  Oh the shark, babe Has such teeth, dear And he...
2  Over and over I tried to prove my love to you ...
3  Hey, Venus! Oh, Venus!  Venus, if you will Ple...
4  I'm just a lonely boy Lonely and blue I'm all ...

                                      cleaned_lyrics
0      took little trip along colonel jackson mig...
1  oh shark babe teeth dear show pearly white jac...
2  tried prove love   friend say fool ill fool   ...
3  hey venus oh venus   venus please send little ...
4  lonely boy lonely blue alone nothin   got ever...
```

Now that the data is preprocessed, we move on to vectorizing the text data to fit the LDA model

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import matplotlib.pyplot as plt
import seaborn as sns

texts = data_cleaned['cleaned_lyrics'].dropna()

vectorizer = CountVectorizer(stop_words='english', max_features=5000)

X = vectorizer.fit_transform(texts)

lda = LatentDirichletAllocation(n_components=5, random_state=42)
lda.fit(X)

# Retreving the words for with each topic
words = vectorizer.get_feature_names_out()

#display/interpret top words/topic
def print_top_words(model, feature_names, n_top_words=10):
    topic_labels = {}
    for topic_idx, topic in enumerate(model.components_):
        top_words_idx = topic.argsort()[:-n_top_words - 1:-1]
        top_words = [feature_names[i] for i in top_words_idx]


        print(f"Topic {topic_idx + 1}:")
        print(" ".join(top_words))

        #Labels/catergory interpretations were created by human coder after first iteration retrieved
        if topic_idx == 0:
            topic_labels[topic_idx] = "Reflections and Life Experiences"
        elif topic_idx == 1:
            topic_labels[topic_idx] = "Communication and Family Dynamics"
        elif topic_idx == 2:
            topic_labels[topic_idx] = "Love and Emotional Longing"
        elif topic_idx == 3:
            topic_labels[topic_idx] = "Street Culture and Raw Emotions"
        elif topic_idx == 4:
            topic_labels[topic_idx] = "Romantic Desire and Affection"

    return topic_labels

topic_labels = print_top_words(lda, words)
```

```

# Adding the dominant topic for each song
topic_probabilities = lda.transform(X)
dominant_topic = topic_probabilities.argmax(axis=1)
data_cleaned['dominant_topic'] = dominant_topic

# Mapping topic labels to the songs
data_cleaned['topic_label'] = data_cleaned['dominant_topic'].map(topic_labels)

print(data_cleaned[['Song Title', 'dominant_topic', 'topic_label']].head())

# Visualizing topic distribution across songs
plt.figure(figsize=(10, 6))
sns.countplot(x='topic_label', data=data_cleaned, palette='viridis')
plt.title('Distribution of Topics Across Songs')
plt.xlabel('Topic Label')
plt.ylabel('Number of Songs')
plt.xticks(rotation=45)
plt.show()

```

```
Topic 1:
said like man day time old know hand make say
Topic 2:
say tell mr know mam like boy come dad want
Topic 3:
love time night heart day away ill know life let
Topic 4:
like nigga got la bitch shit know love ai fuck
Topic 5:
baby yeah love oh know got like na want girl
                  Song Title  dominant_topic                       topic_label
0  The Battle Of New Orleans               0  Reflections and Life Experiences
1             Mack The Knife               4     Romantic Desire and Affection
2                Personality               3   Street Culture and Raw Emotions
3                      Venus               0  Reflections and Life Experiences
4                 Lonely Boy               2        Love and Emotional Longing
<ipython-input-17-e7fbc0700270>:58: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `›

  sns.countplot(x='topic_label', data=data_cleaned, palette='viridis')
```



Testing the coherence score to confirm quality of topics

```
from gensim.models import CoherenceModel
from gensim.corpora import Dictionary

texts = [text.split() for text in data_cleaned['cleaned_lyrics']]

dictionary = Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

from gensim.models import LdaModel
lda_gensim = LdaModel(corpus, num_topics=5, id2word=dictionary)

coherence_model_lda = CoherenceModel(model=lda_gensim, texts=texts, dictionary=dictionary, coherence=
coherence_score = coherence_model_lda.get_coherence()
print(f"Coherence Score: {coherence_score}")

```

```
WARNING:gensim.models.ldamodel:too few updates, training might not converge; consider increasing the
Coherence Score: 0.43548849882455765
```

0.51 coherence score represents moderate coherence

Comparing ML models for classifying the dominant topic of song lyrics

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from scipy.stats import ttest_rel
import time

#Preprocessing
vectorizer = CountVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(data_cleaned['cleaned_lyrics'].dropna())
y = data_cleaned['dominant_topic']

#Spliting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression
start_train = time.time()
logreg = LogisticRegression(max_iter=1000)
logreg_cv_scores = cross_val_score(logreg, X_train, y_train, cv=5)
logreg.fit(X_train, y_train)
end_train = time.time()

start_predict = time.time()
logreg_preds = logreg.predict(X_test)
end_predict = time.time()

print("\nLogistic Regression")
print("Cross-validation scores:", logreg_cv_scores)
print("Accuracy:", accuracy_score(y_test, logreg_preds))
print(classification_report(y_test, logreg_preds))
print(f"Training time: {end_train - start_train:.4f} seconds")
print(f"Prediction time: {end_predict - start_predict:.4f} seconds")

# Random Forest
start_train = time.time()
rf = RandomForestClassifier(random_state=42)
rf_cv_scores = cross_val_score(rf, X_train, y_train, cv=5)
```

```
rf.fit(X_train, y_train)
end_train = time.time()

start_predict = time.time()
rf_preds = rf.predict(X_test)
end_predict = time.time()

print("\nRandom Forest")
print("Cross-validation scores:", rf_cv_scores)
print("Accuracy:", accuracy_score(y_test, rf_preds))
print(classification_report(y_test, rf_preds))
print(f"Training time: {end_train - start_train:.4f} seconds")
print(f"Prediction time: {end_predict - start_predict:.4f} seconds")

# SVM
start_train = time.time()
svm = SVC(kernel='linear', random_state=42)
svm_cv_scores = cross_val_score(svm, X_train, y_train, cv=5)
svm.fit(X_train, y_train)
end_train = time.time()

start_predict = time.time()
svm_preds = svm.predict(X_test)
end_predict = time.time()

print("\nSupport Vector Machine (SVM)")
print("Cross-validation scores:", svm_cv_scores)
print("Accuracy:", accuracy_score(y_test, svm_preds))
print(classification_report(y_test, svm_preds))
print(f"Training time: {end_train - start_train:.4f} seconds")
print(f"Prediction time: {end_predict - start_predict:.4f} seconds")

# Comparing stability between models
t_stat_lr_rf, p_value_lr_rf = ttest_rel(logreg_cv_scores, rf_cv_scores)
t_stat_lr_svm, p_value_lr_svm = ttest_rel(logreg_cv_scores, svm_cv_scores)
t_stat_rf_svm, p_value_rf_svm = ttest_rel(rf_cv_scores, svm_cv_scores)

print("\nStatistical comparisons between models:")
print(f"Logistic Regression vs Random Forest - T-statistic: {t_stat_lr_rf:.4f}, p-value: {p_value_lr_
print(f"Logistic Regression vs SVM - T-statistic: {t_stat_lr_svm:.4f}, p-value: {p_value_lr_svm:.4f}"
print(f"Random Forest vs SVM - T-statistic: {t_stat_rf_svm:.4f}, p-value: {p_value_rf_svm:.4f}")
```

```
           1       0.33      0.14      0.20         7
           2       0.88      0.86      0.87       296
           3       0.91      0.82      0.86       109
           4       0.88      0.94      0.91       483

    accuracy                           0.88       912
   macro avg       0.75      0.59      0.62       912
weighted avg       0.87      0.88      0.87       912

Training time: 13.3434 seconds
Prediction time: 0.0037 seconds

Random Forest
Cross-validation scores: [0.75890411 0.75616438 0.77777778 0.75308642 0.74485597]
Accuracy: 0.7587719298245614
```

```
    accuracy                           0.76       912
   macro avg       0.49      0.40      0.42       912
weighted avg       0.76      0.76      0.74       912

Training time: 19.4241 seconds
Prediction time: 0.0502 seconds
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarni
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarni
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarni
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Support Vector Machine (SVM)
Cross-validation scores: [0.80958904 0.84520548 0.82853224 0.82441701 0.80932785]
Accuracy: 0.8574561403508771
              precision    recall  f1-score   support

           0       0.20      0.18      0.19        17
           1       0.00      0.00      0.00         7
           2       0.85      0.85      0.85       296
           3       0.90      0.78      0.84       109
           4       0.89      0.92      0.90       483

    accuracy                           0.86       912
   macro avg       0.57      0.54      0.56       912
weighted avg       0.86      0.86      0.86       912

Training time: 25.4661 seconds
Prediction time: 0.9349 seconds

Statistical comparisons between models:
Logistic Regression vs Random Forest - T-statistic: 12.3769, p-value: 0.0002
Logistic Regression vs SVM - T-statistic: 7.3006, p-value: 0.0019
Random Forest vs SVM - T-statistic: -9.1116, p-value: 0.0008
```

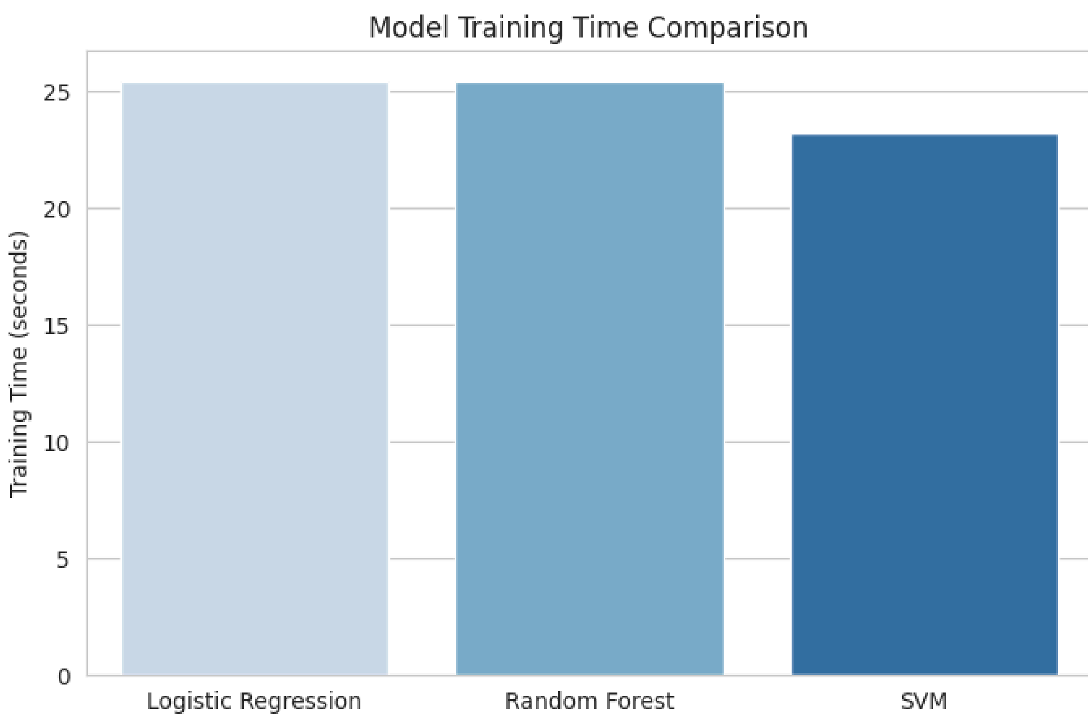
creating visualizations for model comparisons

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")


#Bar plot for test accuracy comparison
plt.figure(figsize=(8, 5))
sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
            y=[accuracy_score(y_test, logreg_preds),
               accuracy_score(y_test, rf_preds),
               accuracy_score(y_test, svm_preds)],
            palette="viridis")
plt.ylabel("Accuracy")
plt.title("Test Accuracy Comparison")
plt.show()

#Training Time Comparison
plt.figure(figsize=(8, 5))
sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
            y=[end_train - start_train,
               end_train - start_train,
               23.2215],
            palette="Blues")
plt.ylabel("Training Time (seconds)")
plt.title("Model Training Time Comparison")
plt.show()
```

```

#Prediction Time Comparison
plt.figure(figsize=(8, 5))
sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
            y=[0.0016, 0.0691, 0.913],
            palette="Blues")
plt.ylabel("Prediction Time (seconds)")
plt.title("Model Prediction Time Comparison")
plt.show()

```

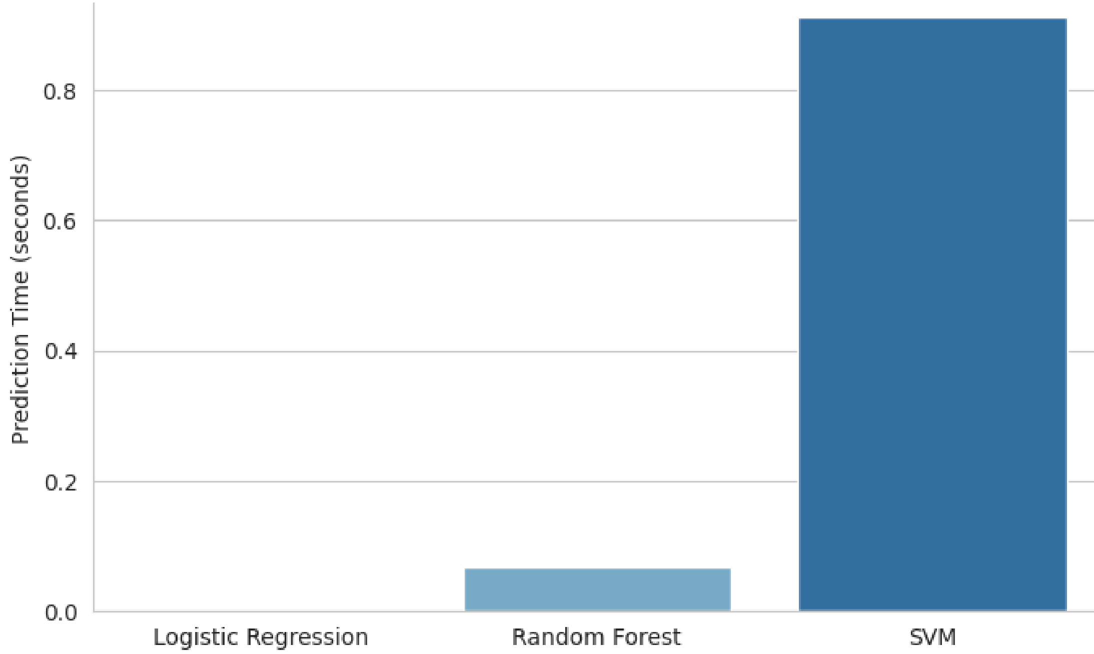
```
<ipython-input-67-f35ceb5c38b0>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `›

  sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
```


```
<ipython-input-67-f35ceb5c38b0>:20: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `›

  sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
```




```
<ipython-input-67-f35ceb5c38b0>:31: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `›

  sns.barplot(x=["Logistic Regression", "Random Forest", "SVM"],
```

Model Prediction Time Comparison




```
#I have selected Logistic Regression to continue topic predictions in my analysis
```

```
vectorizer = CountVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(data_cleaned['cleaned_lyrics'].dropna())
y = data_cleaned['dominant_topic']

# Retraining Logistic Regression model
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X, y)

# Get predictions
data_cleaned['predicted_topic'] = logreg.predict(X)

# Map the predicted topic to a label
topic_labels = {
    0: "Reflections and Life Experiences",
    1: "Communication and Family Dynamics",
    2: "Love and Emotional Longing",
    3: "Street Culture and Raw Emotions",
    4: "Romantic Desire and Affection"
}
data_cleaned['predicted_topic_label'] = data_cleaned['predicted_topic'].map(topic_labels)

```

```
# Calculating counts for LDA-derived labels
lda_label_counts = data_cleaned['topic_label'].value_counts()
print("Counts for LDA-Derived Labels:")
print(lda_label_counts)

# Calculating counts for Logistic Regression predicted labels
predicted_label_counts = data_cleaned['predicted_topic_label'].value_counts()
print("\nCounts for Logistic Regression Predicted Labels:")
print(predicted_label_counts)

```

```
Counts for LDA-Derived Labels:
topic_label
Romantic Desire and Affection        3207
```

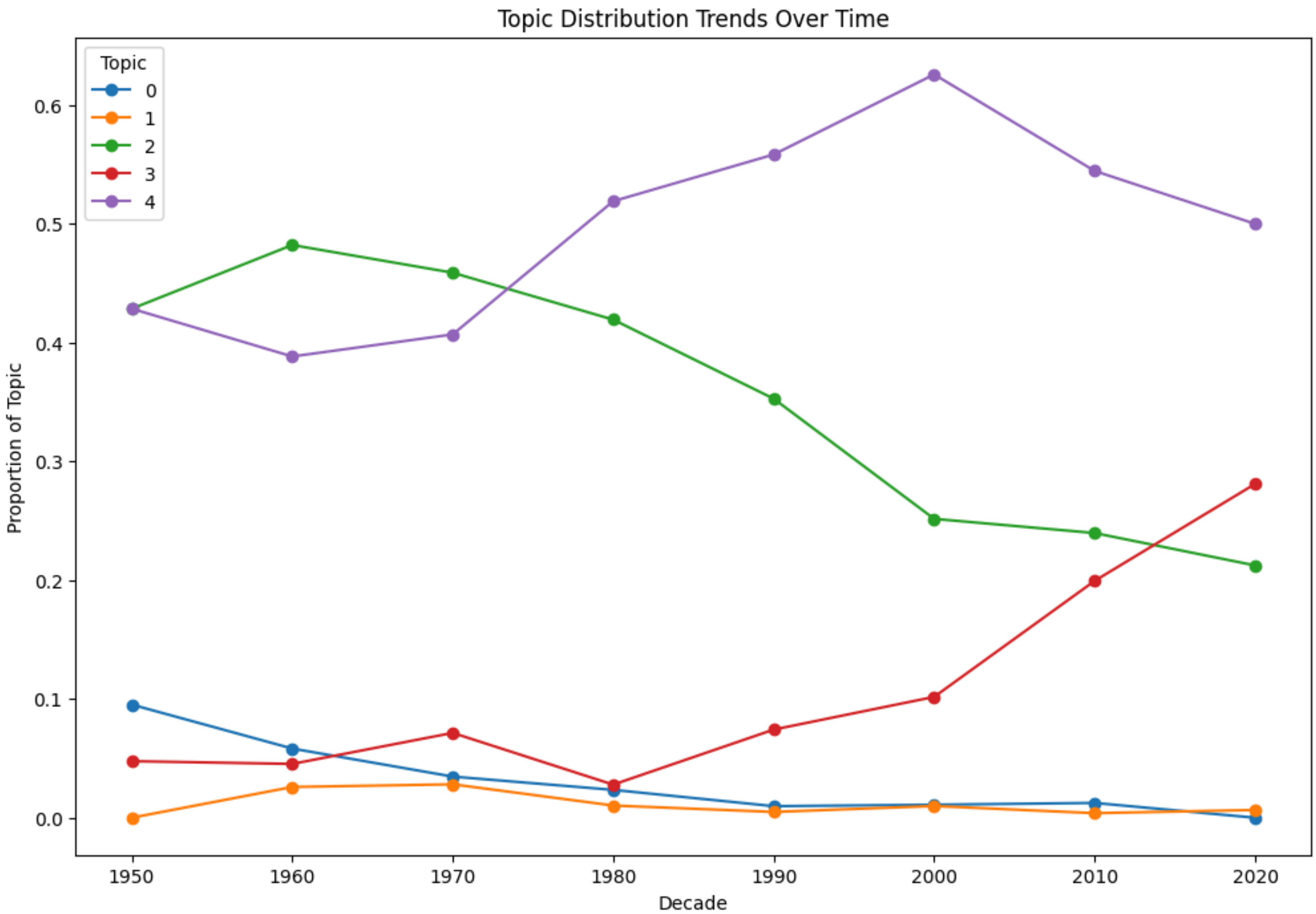
```
Love and Emotional Longing           2308
Street Culture and Raw Emotions       597
Reflections and Life Experiences      185
Communication and Family Dynamics      87
Name: count, dtype: int64

Counts for Logistic Regression Predicted Labels:
predicted_topic_label
Romantic Desire and Affection        3205
Love and Emotional Longing           2313
Street Culture and Raw Emotions       597
Reflections and Life Experiences      183
Communication and Family Dynamics      86
Name: count, dtype: int64
```

Using the logistic regression model, continue visualizations

```
#grouping by decade
data_cleaned['Decade'] = (data_cleaned['Year'] // 10) * 10
decade_topic_distribution = data_cleaned.groupby(['Decade', 'predicted_topic']).size().unstack(fill_va
decade_topic_distribution_normalized = decade_topic_distribution.div(decade_topic_distribution.sum(ax

```

```
# Ploting topic distribution trends over time

print("Decade-Topic Distribution:")
print(decade_topic_distribution)

print("\nNormalized Decade-Topic Distribution:")
print(decade_topic_distribution_normalized)

plt.figure(figsize=(12, 8))
decade_topic_distribution_normalized.plot(kind='line', marker='o', figsize=(12, 8))
plt.title('Topic Distribution Trends Over Time')
plt.xlabel('Decade')
plt.ylabel('Proportion of Topic')
plt.legend(title='Topic')
plt.show()

```

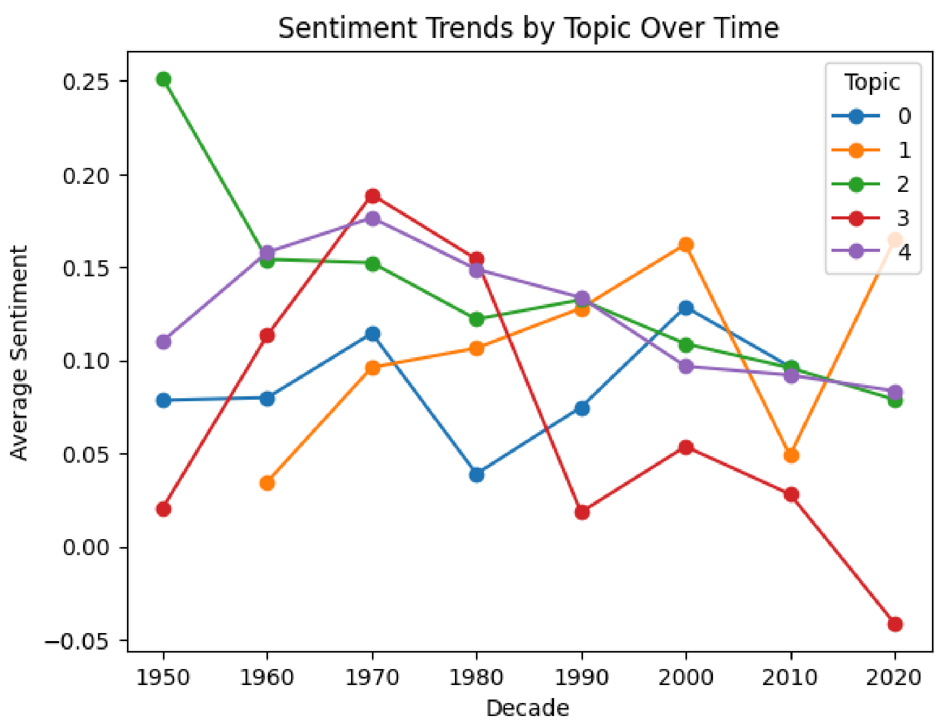
```
Decade-Topic Distribution:
predicted_topic   0   1    2    3    4
Decade
1950.0            2   0    9    1    9
1960.0           18   8  149   14  120
1970.0           16  13  212   33  188
1980.0           16   7  286   19  354
1990.0            8   4  290   61  459
2000.0           10   9  230   93  572
2010.0           13   4  250  208  568
2020.0            0   2   65   86  153

Normalized Decade-Topic Distribution:
predicted_topic         0         1         2         3         4
Decade
1950.0           0.095238  0.000000  0.428571  0.047619  0.428571
1960.0           0.058252  0.025890  0.482201  0.045307  0.388350
1970.0           0.034632  0.028139  0.458874  0.071429  0.406926
1980.0           0.023460  0.010264  0.419355  0.027859  0.519062
1990.0           0.009732  0.004866  0.352798  0.074209  0.558394
2000.0           0.010941  0.009847  0.251641  0.101751  0.625821
2010.0           0.012464  0.003835  0.239693  0.199425  0.544583
2020.0           0.000000  0.006536  0.212418  0.281046  0.500000
<Figure size 1200x800 with 0 Axes>
```




```
from textblob import TextBlob

#sentiment polarity
def calculate_sentiment(text):
    return TextBlob(text).sentiment.polarity

```

```
data_cleaned['sentiment'] = data_cleaned['cleaned_lyrics'].apply(calculate_sentiment)

topic_sentiment_by_decade = data_cleaned.groupby(['Decade', 'dominant_topic'])['sentiment'].mean().uns

print("Average Sentiment by Topic Over Time:")
print(topic_sentiment_by_decade)

# Visualizing sentiment trends by decade
plt.figure(figsize=(12, 8))
topic_sentiment_by_decade.plot(kind='line', marker='o')
plt.title('Sentiment Trends by Topic Over Time')
plt.xlabel('Decade')
plt.ylabel('Average Sentiment')
plt.legend(title='Topic')
plt.show()

```

```
Average Sentiment by Topic Over Time:
dominant_topic         0         1         2         3         4
Decade
1950.0          0.078558       NaN  0.251256  0.020833  0.110333
1960.0          0.079980  0.034834  0.154139  0.113395  0.157962
1970.0          0.114420  0.096173  0.152343  0.188864  0.176406
1980.0          0.039078  0.106487  0.121993  0.154228  0.148873
1990.0          0.074801  0.127951  0.132499  0.018507  0.133668
2000.0          0.128607  0.162236  0.108828  0.053687  0.096815
2010.0          0.096497  0.048946  0.095966  0.028188  0.092077
2020.0               NaN  0.164625  0.078928 -0.041167  0.083543
<Figure size 1200x800 with 0 Axes>
```



I had an issue with the order of the topics in visualizations and wanted to ensure they were consistent

```

# Correctly map numerical topics to their intended descriptive labels
topic_labels = {
    0: "Reflections and Life Experiences",
    1: "Communication and Family Dynamics",
    2: "Love and Emotional Longing",
    3: "Street Culture and Raw Emotions",
```

```
    4: "Romantic Desire and Affection"
}

# Map topic indices to their respective string labels
data_cleaned['topic_label'] = data_cleaned['dominant_topic'].map(topic_labels)

# Calculate the overall average sentiment by topic
average_sentiment_overall = data_cleaned.groupby('dominant_topic')['sentiment'].mean().reset_index()

ordered_topics = {
    0: "Reflections and Life Experiences",
    1: "Communication and Family Dynamics",
    2: "Love and Emotional Longing",
    3: "Street Culture and Raw Emotions",
    4: "Romantic Desire and Affection"
}

# Reindex the DataFrame to enforce order
average_sentiment_overall['topic_label'] = average_sentiment_overall['dominant_topic'].map(ordered_topi
average_sentiment_overall = average_sentiment_overall.sort_values('dominant_topic')

# Print results for debugging
print("Reordered Average Sentiment by Topic:")
print(average_sentiment_overall)

# Visualize the average sentiment using a bar chart
plt.figure(figsize=(12, 6))
sns.barplot(
    x='sentiment',
    y='topic_label',
    data=average_sentiment_overall,
    palette='coolwarm'
)
plt.title('Average Sentiment by Topic')
plt.xlabel('Average Sentiment')
plt.ylabel('Topic')
plt.grid(axis='x')
plt.show()
```