
AN ANIME
PRESENTATION
Coffee

PRESENTED BY

ALI ATAOLLAHI

PARISA MOHAMMADI

FARNAZ SEDAGHATI

INTRODUCTION

WHAT IS ANIME?

Anime is a dynamic Japanese animation style known for its stunning visuals and imaginative themes. It spans TV series, films, and web series, captivating all ages with whimsical to deeply emotional stories.

IMPORTANCE

Anime, captivating mainly the youth with over 60% under 30, influences pop culture (70%), fosters creativity (80%), and unites global communities (75%), making it an essential and enriching phenomenon.

SOURCE

MYANIMELIST.NET

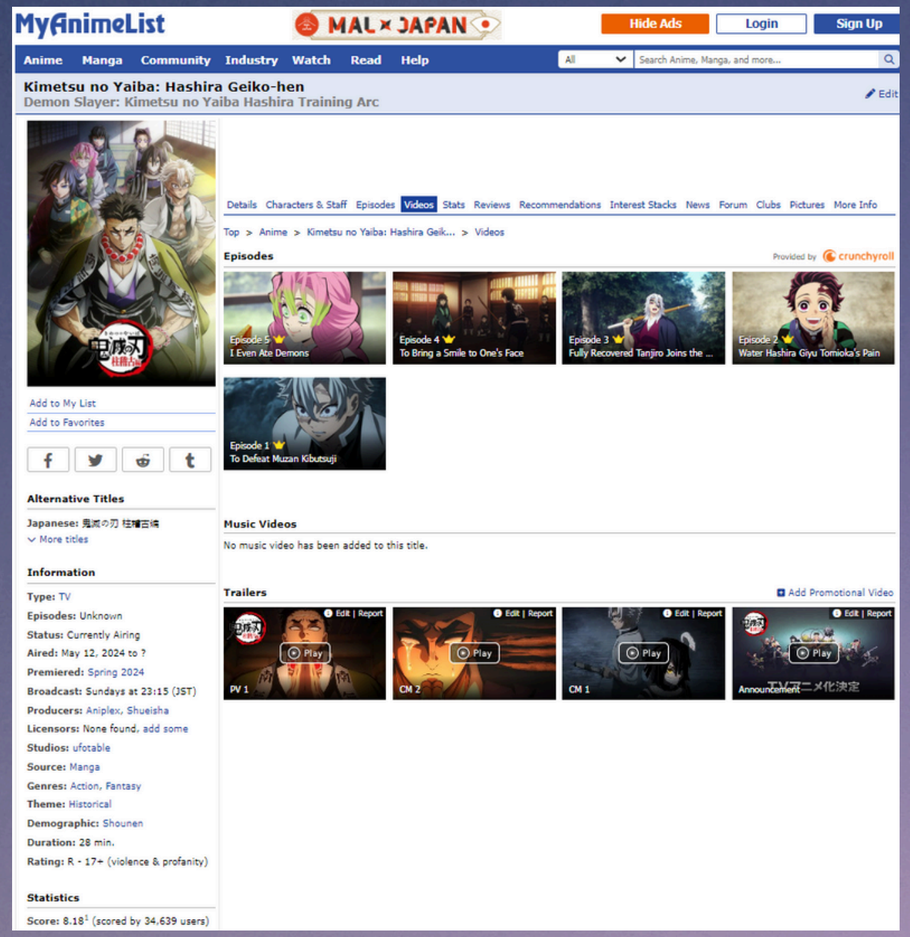
MyAnimeList (MAL) is a premier platform for anime and manga fans, offering a comprehensive database, personalized recommendations, and vibrant community discussions since 2004.

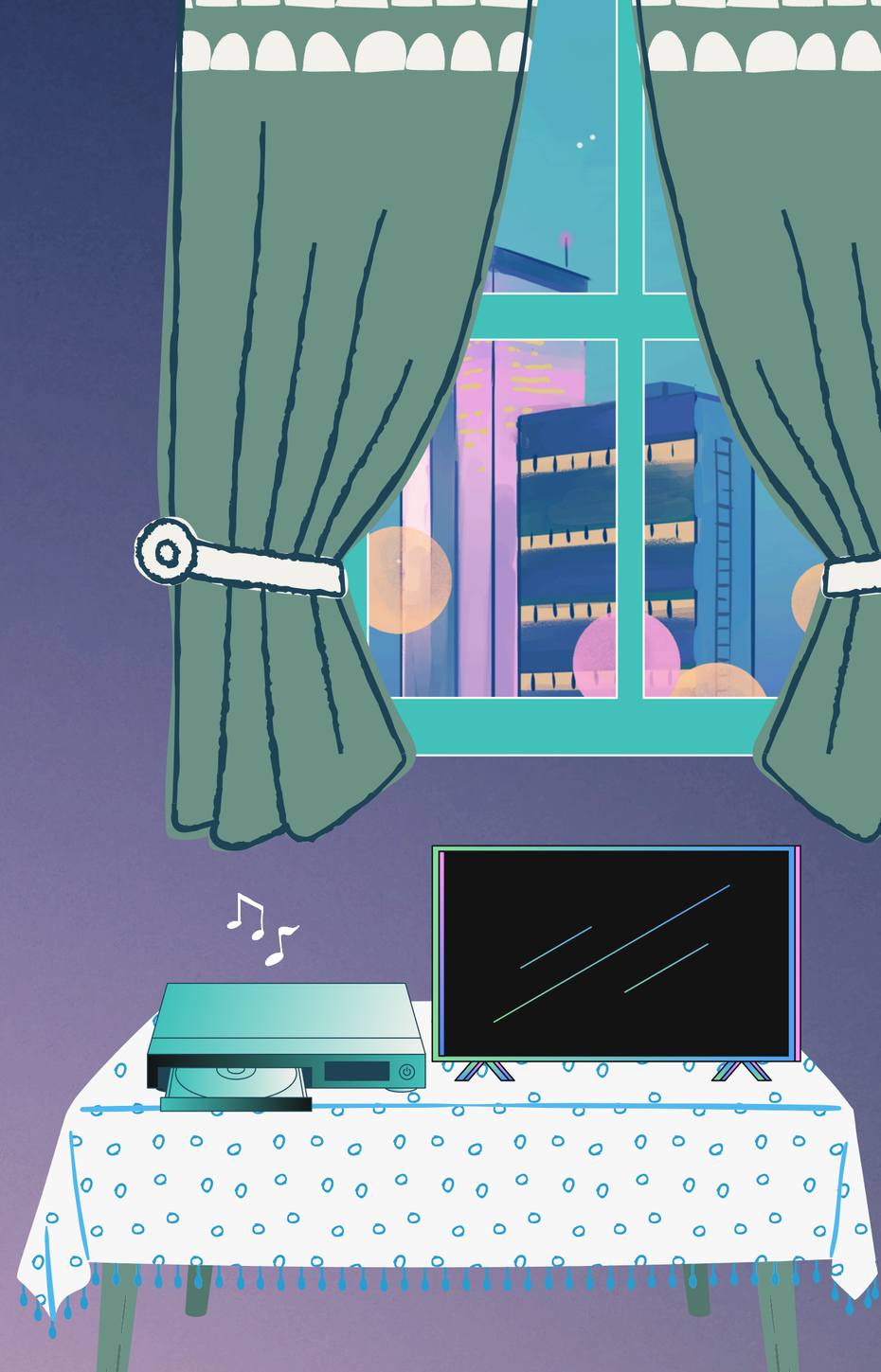
GENRES

WHAT ARE THE GENRES OF ANIME?

Anime genres include Adventure (One Piece), Drama (Your Lie in April), Fantasy (Attack on Titan), Action (Naruto), Comedy (One Punch Man), Romance (Fruits Basket), Supernatural (Tokyo Ghoul), Slice of Life (Clannad), and ...

HOW DID WE SCRAP?



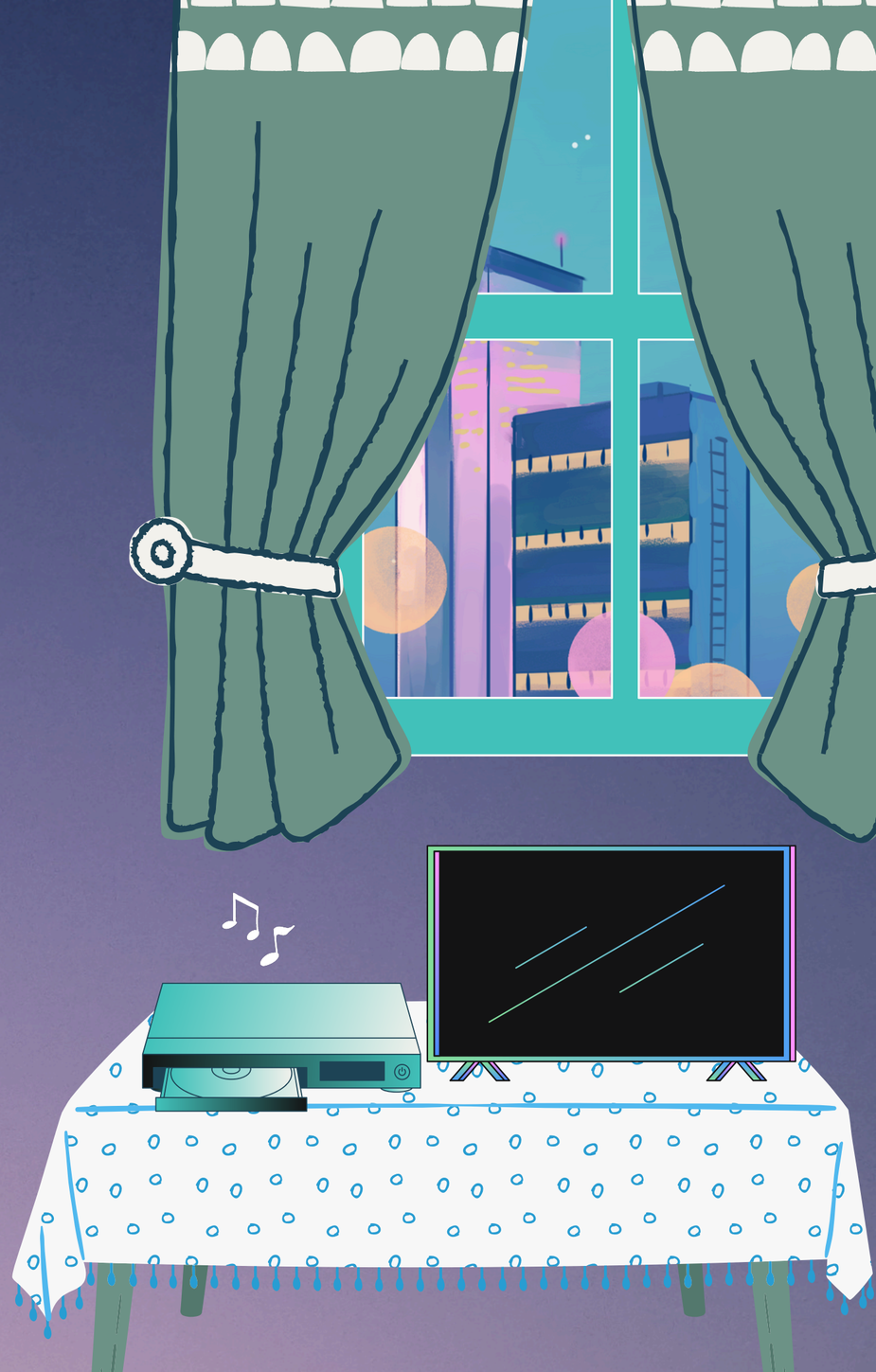


HOW DID WE SCRAP?

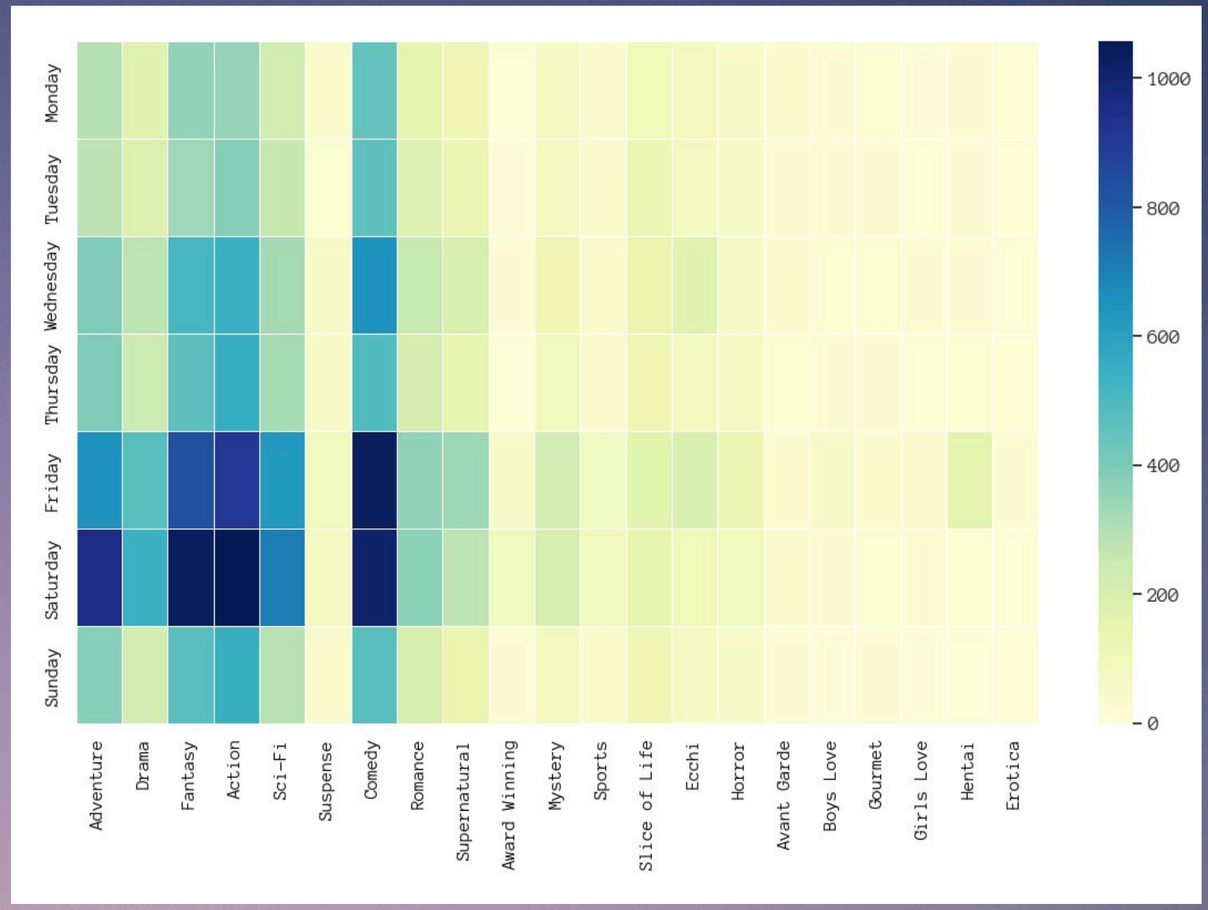
anime_info_final.csv

```
Unnamed: 0,title,Type,Episodes,Status,Aired,Premier
0,Sousou_no_Frieren,['TV'],28,Finished Airing,"Sep
1,Fullmetal_Alchemist__Brotherhood,['TV'],64,Finish
2,Steins_Gate,['TV'],24,Finished Airing,"Apr 6, 20
3,Gintama°,['TV'],51,Finished Airing,"Apr 8, 2015
4,Shingeki_no_Kyojin_Season_3_Part_2,['TV'],10,Fin
5,Gintama__The_Final,['Movie'],1,Finished Airing,"
6,Hunter_x_Hunter_2011,['TV'],148,Finished Airing,
7,Gintama,['TV'],51,Finished Airing,"Apr 4, 2011 t
8,Bleach__Sennen_Kessen-hen,['TV'],13,Finished Air
9,Ginga_Eiyuu_Densetsu,['OVA'],110,Finished Airing
10,Gintama__Enchousen,['TV'],13,Finished Airing,"O
11,Kaguya-sama_wa_Kokurasetai__Ultra_Romantic,['TV
12,Fruits_Basket__The_Final,['TV'],13,Finished Air
13,Gintama,['TV'],12,Finished Airing,"Jan 9, 2017
14,Gintama,['TV'],201,Finished Airing,"Apr 4, 2006
15,Clannad__After_Story,['TV'],24,Finished Airing,
16,Koe_no_Katachi,['Movie'],1,Finished Airing,"Sep
17,Kusuriya_no_Hitorigoto,['TV'],24,Finished Airing
18,3-gatsu_no_Lion_2nd_Season,['TV'],22,Finished A
19,Code_Geass__Hangyaku_no_Lelouch_R2,['TV'],25,Fi
20,Gintama_Movie_2__Kanketsu-hen_-_Yorozuya_yo_Eie
21,Shingeki_no_Kyojin__The_Final_Season_-_Kanketsu
22,Boku_no_Kokoro_no_Yabai_Yatsu_Season_2,['TV'],1
23,Gintama__Shirogane_no_Tamashii-hen_-_Kouhan-sen
24,Monster,['TV'],74,Finished Airing,"Apr 7, 2004
25,Owarimonogatari_2nd_Season,['TV Special'],7,Fin
26,Violet_Evergarden_Movie,['Movie'],1,Finished Ai
27,Jujutsu_Kaisen_2nd_Season,['TV'],23,Finished Ai
28,Kimi_no_Na_wa,['Movie'],1,Finished Airing,"Aug
29,Kingdom_3rd_Season,['TV'],26,Finished Airing,"Ap
30,Vinland_Saga_Season_2,['TV'],24,Finished Airing
```

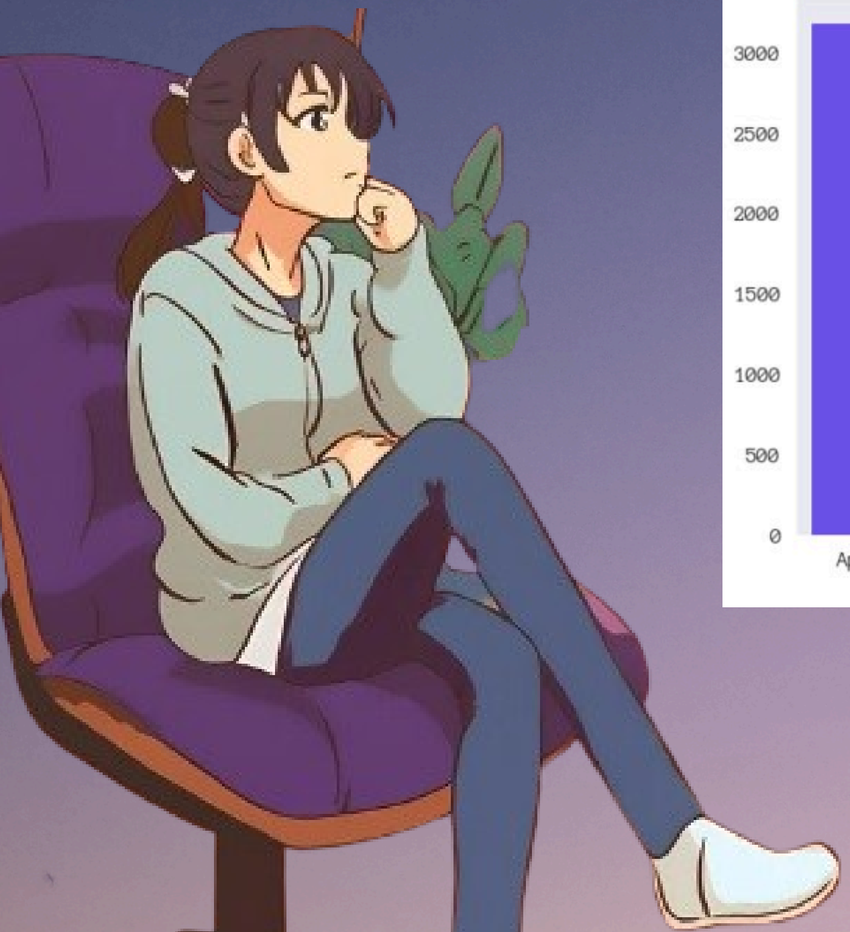
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26440 entries, 0 to 26439
Data columns (total 21 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    26440 non-null  int64
 1   title         26440 non-null  object
 2   Type          26440 non-null  object
 3   Episodes      26440 non-null  object
 4   Status        26440 non-null  object
 5   Aired         26440 non-null  object
 6   Premiered     7874 non-null   object
 7   Producers     26440 non-null  object
 8   Licensors     26440 non-null  object
 9   Studios       26440 non-null  object
 10  Source        24435 non-null  object
 11  Genres        11463 non-null  object
 12  Demographic   9982 non-null   object
 13  Duration      26440 non-null  object
 14  Rating        25923 non-null  object
 15  Score         17197 non-null  float64
 16  Ranked        20647 non-null  object
 17  Popularity    26440 non-null  object
 18  Members       26440 non-null  object
 19  Favorites     26440 non-null  object
 20  staff         26440 non-null  object
dtypes: float64(1), int64(1), object(19)
memory usage: 4.2+ MB
```

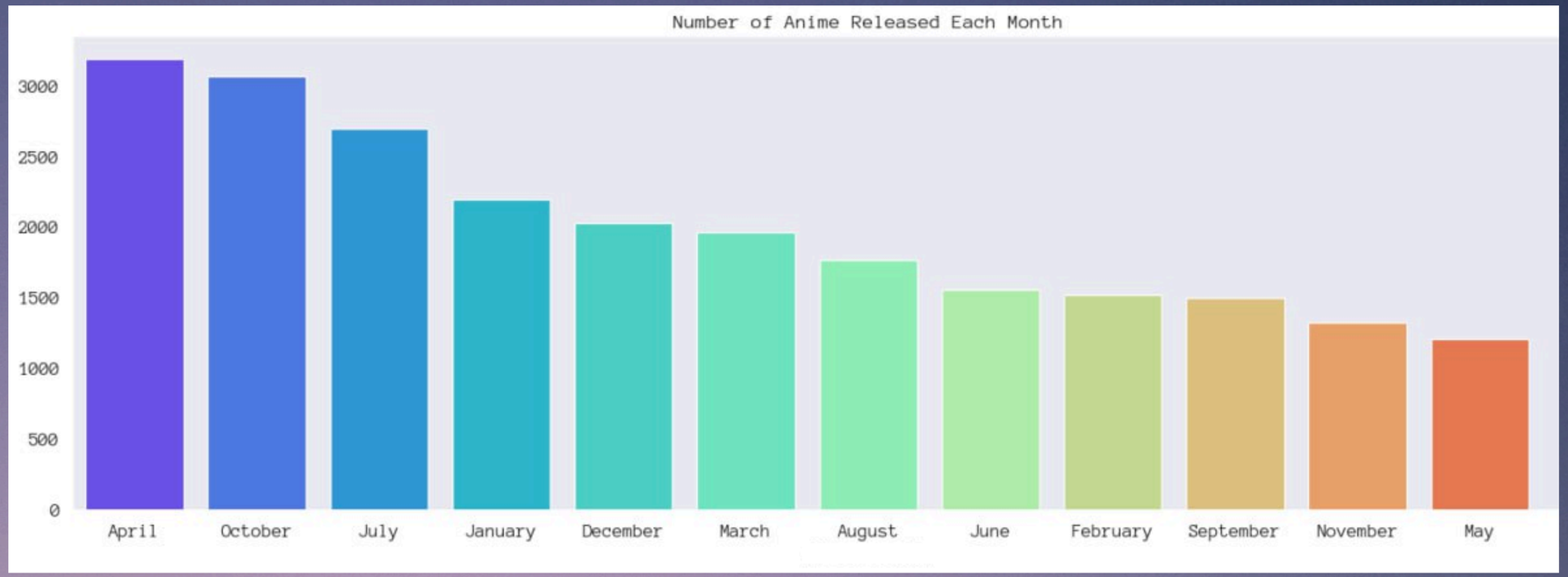


AIRED DAY

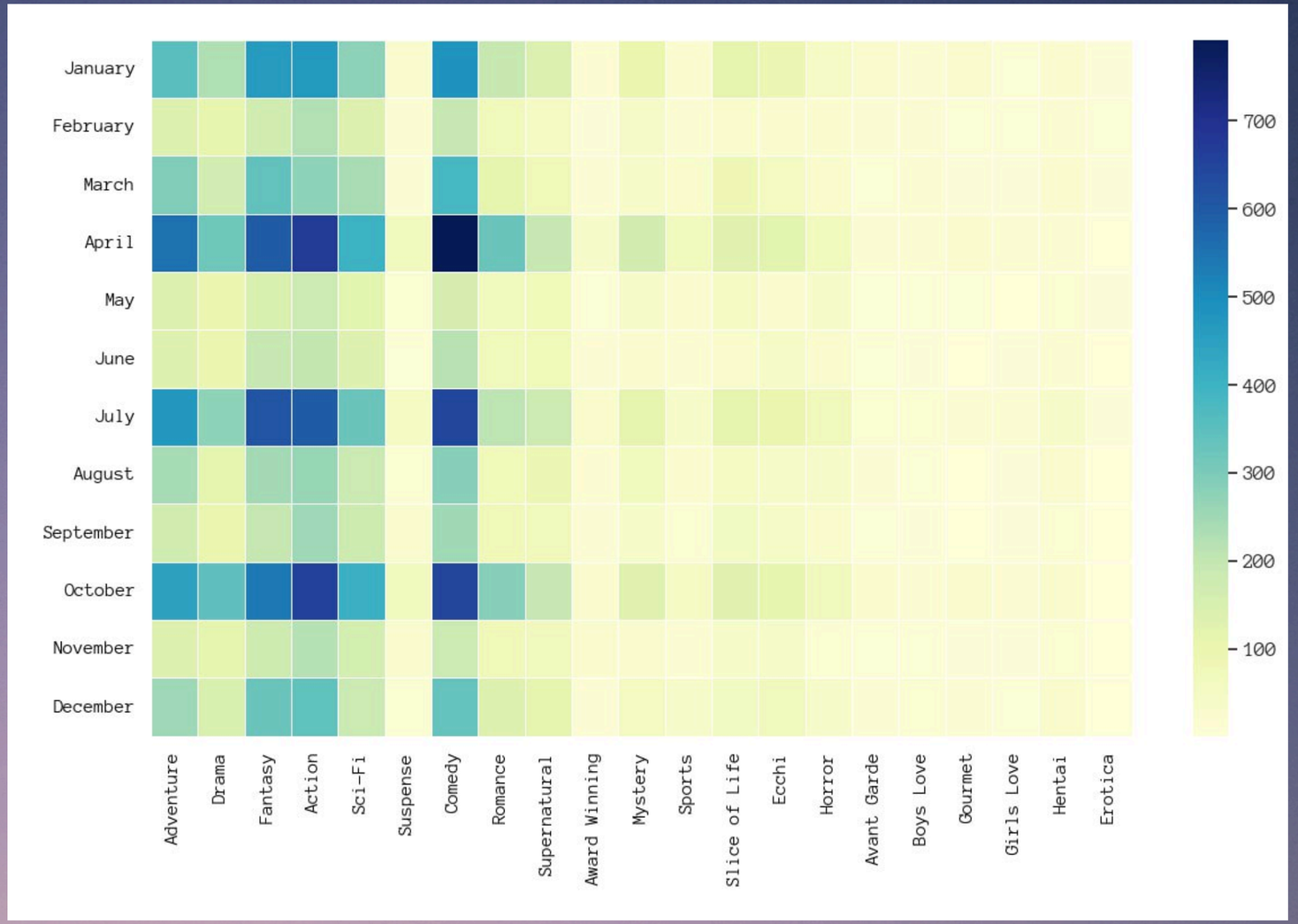

Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Adventure
Drama
Fantasy
Action
Sci-Fi
Suspense
Comedy
Romance
Supernatural
Award Winning
Mystery
Sports
Slice of Life
Ecchi
Horror
Avant Garde
Boys Love
Gourmet
Girls Love
Hentai
Erotica
1000
800
600
400
200
0

AIRED MONTH



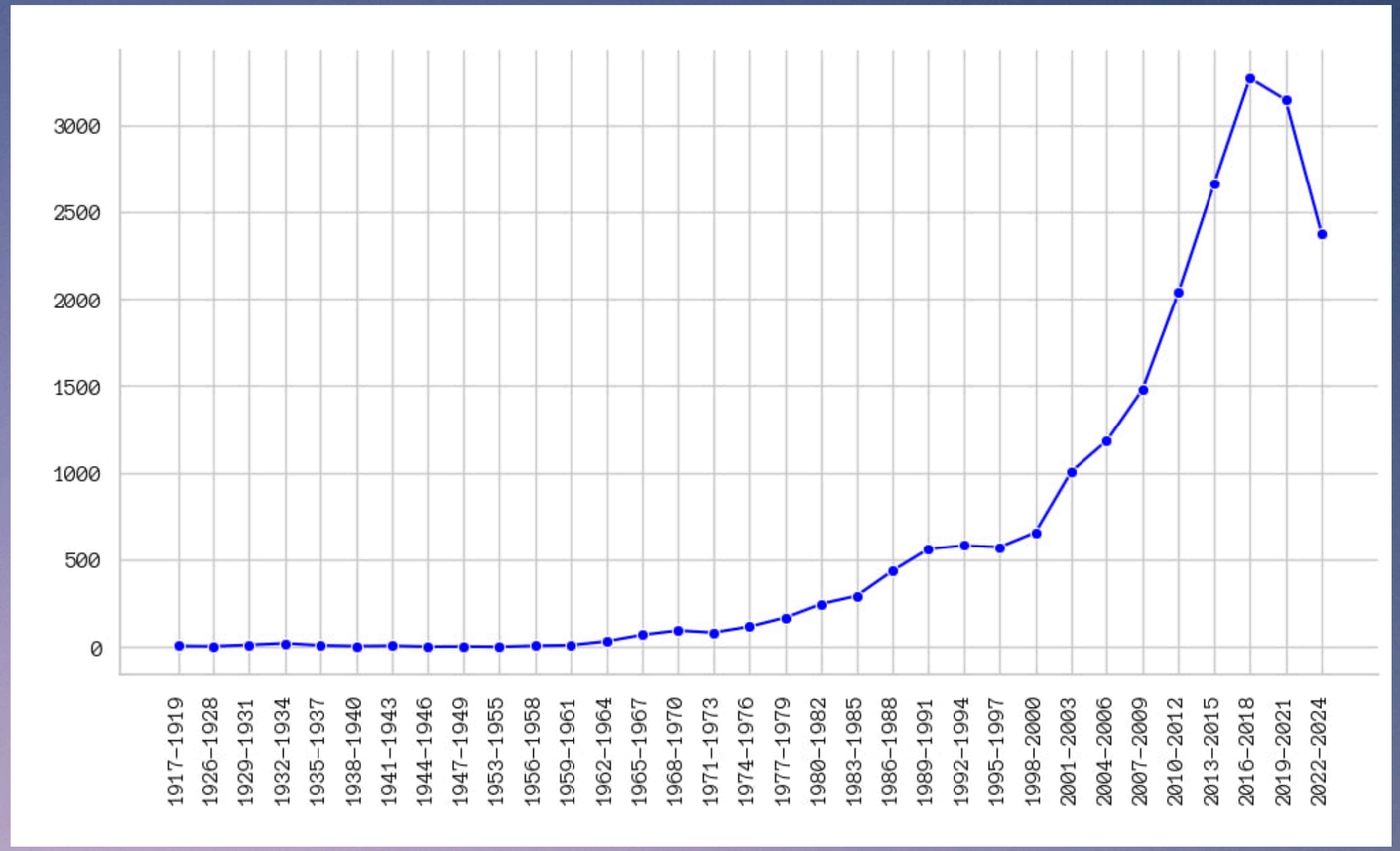

Number of Anime Released Each Month
3000
2500
2000
1500
1000
500
0
April
October
July
January
December
March
August
June
February
September
November
May

AIRED MONTH

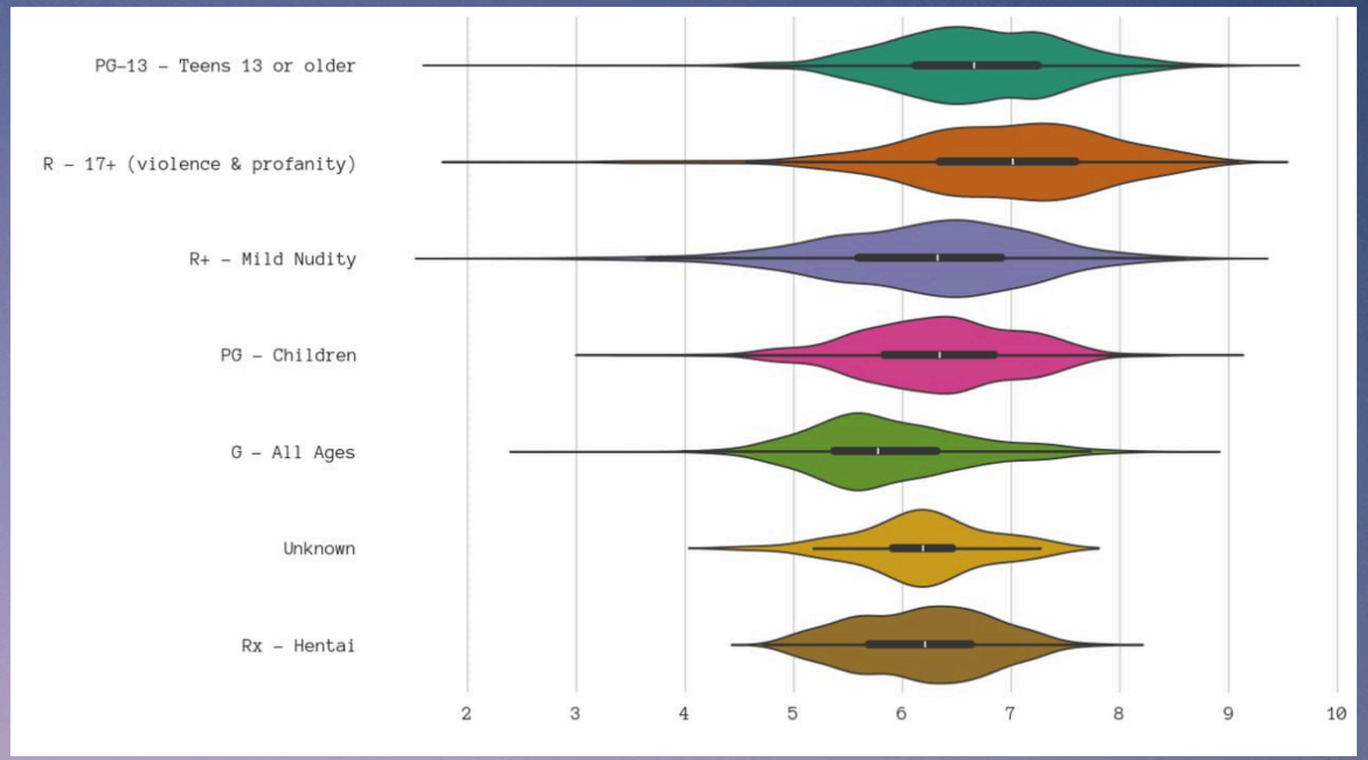

January
February
March
April
May
June
July
August
September
October
November
December
Adventure
Drama
Fantasy
Action
Sci-Fi
Suspense
Comedy
Romance
Supernatural
Award Winning
Mystery
Sports
Slice of Life
Ecchi
Horror
Avant Garde
Boys Love
Gourmet
Girls Love
Hentai
Erotica
700
600
500
400
300
200
100

AIRED SEASON

AIRED YEAR


3000
2500
2000
1500
1000
500
0
1917–1919
1926–1928
1929–1931
1932–1934
1935–1937
1938–1940
1941–1943
1944–1946
1947–1949
1953–1955
1956–1958
1959–1961
1962–1964
1965–1967
1968–1970
1971–1973
1974–1976
1977–1979
1980–1982
1983–1985
1986–1988
1989–1991
1992–1994
1995–1997
1998–2000
2001–2003
2004–2006
2007–2009
2010–2012
2013–2015
2016–2018
2019–2021
2022–2024

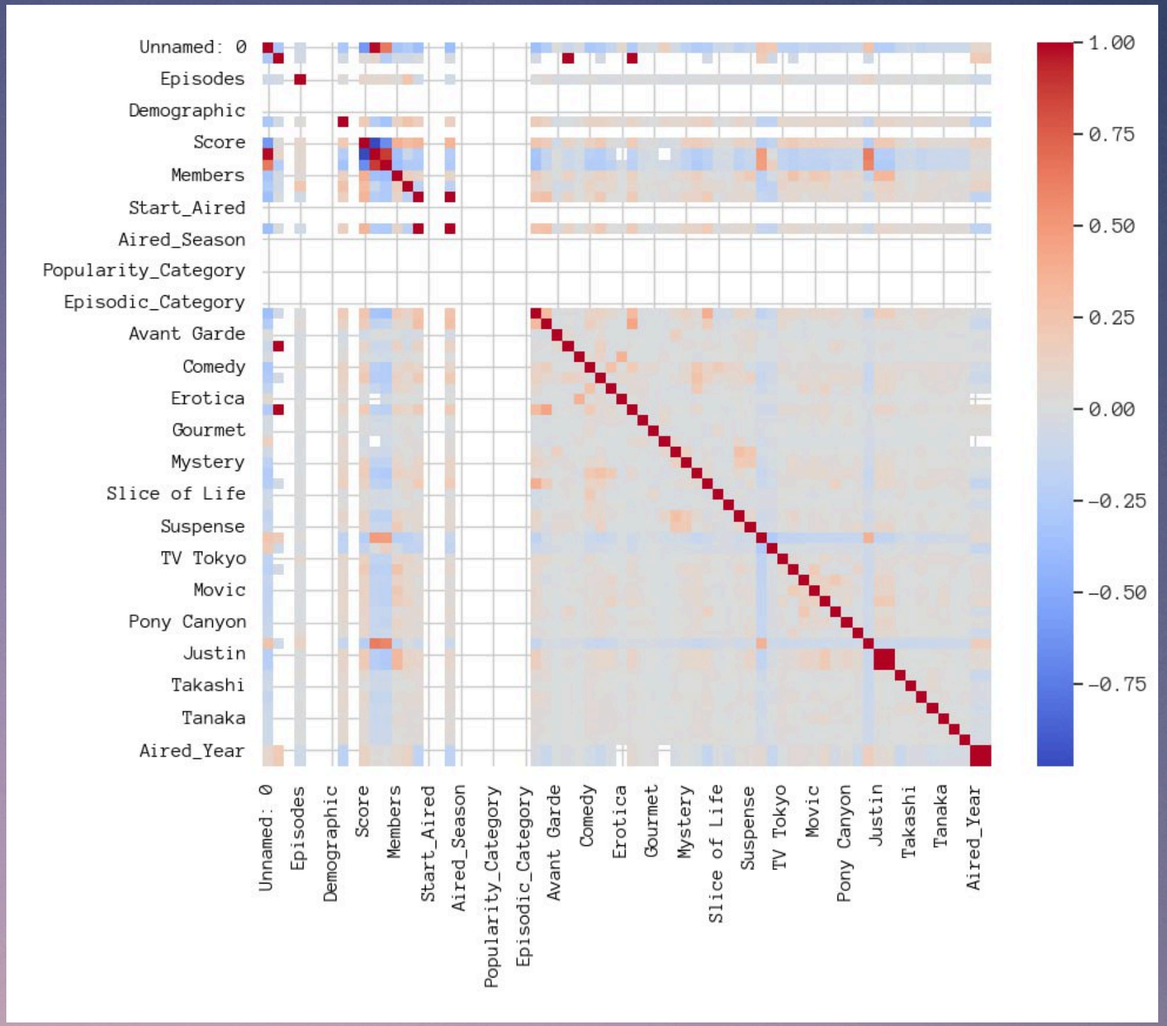
RATING AND SCORE



USING API

```
import time
from openai import OpenAI

# Function to send prompt to GPT API and get response
def do_prompt(corpus):
    client = OpenAI()
    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a data analyst."},
            {"role": "user", "content": f"""you're going to break the data for a date into smaller pieces. We ha
for specific date:
output: year:<YYYY>, Month:<MM>, day:<DD>
for timeframe:
output: start : year:<YYYY>, Month:<MM>, day:<DD> // end:  year:<YYYY>, Month:<MM>, day:<DD>

if some data is not provided use None for that

process this corpus. i dont want code. get output for each line

Example:
2001 -> year: 2001, Month: None, day: None
Feb 2011 to May 2013 -> start : year: 2011, Month: 2, day: None // end:  year: 2013, Month: 5, day: None
May 2001 -> year: 2001, Month: 5, day: None
Mar 2010 -> year: 2010, Month: 3, day: None
Jan 26, 1990 to Oct 1998 -> start : year: 1990, Month: 1, day: 26 // end:  year: 1998, Month: 10, day: None
2004 -> year: 2004, Month: None, day: None
2008 -> year: 2008, Month: None, day: None
Mar 2011 -> year: 2011, Month: 3, day: None
1986 to 1987 -> start : year: 1986, Month: None, day: None // end:  year: 1987, Month: None, day: None
2003 -> year: 2003, Month: None, day: None

corpus:{corpus}
```

FEATURES CORRELATION



FEATURE ENGINEERING AND SELECTION

```
df
```

0.0s Python

_Day_of_Week	Aired_Month	...	Other_Source	TV	OVA	Movie	ONA	Special	TV Special	Other_Type	year_circle_sin_normalized	year_circle_cos_normalized
0	11	...	0	1	0	0	0	0	0	0	0.066987	0.750000
3	0	...	0	1	0	0	0	0	0	0	0.273005	0.945503
6	0	...	0	1	0	0	0	0	0	0	0.296632	0.956773
6	0	...	0	1	0	0	0	0	0	0	0.296632	0.956773
1	0	...	0	1	0	0	0	0	0	0	0.257595	0.937310
...	...	...	...	...	...	...	...	...	...	...	...	...
2	9	...	0	0	1	0	0	0	0	0	0.058526	0.265264
0	10	...	0	0	1	0	0	0	0	0	0.000000	0.500000
2	0	...	0	0	1	0	0	0	0	0	0.265264	0.941474
0	10	...	0	0	1	0	0	0	0	0	0.000000	0.500000
2	2	...	0	0	1	0	0	0	0	0	0.764960	0.924024

DIMENSIONALITY REDUCTION

```
[9]  transformed_df = pca_(df_s)
     ✓ 0.8s
```

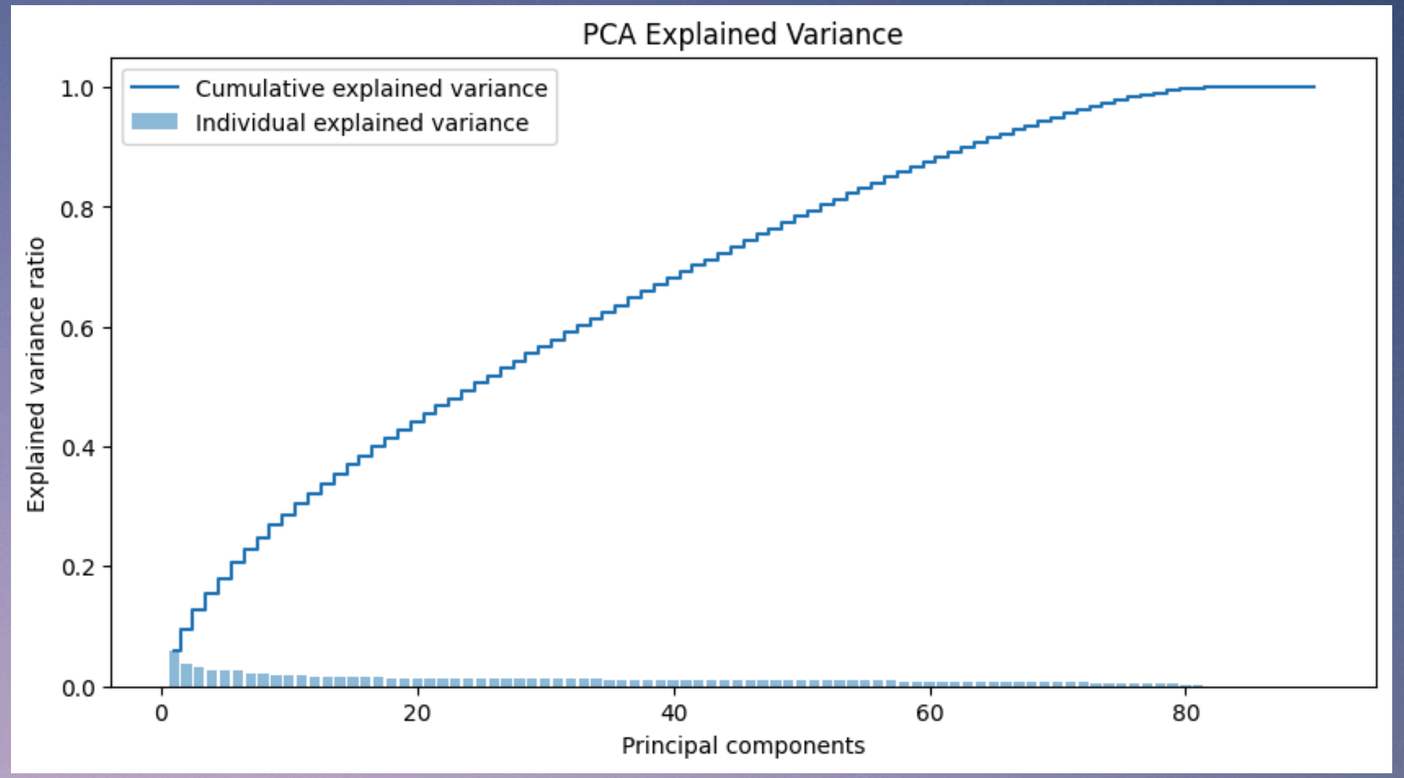
71

	PC1	PC2	PC3	PC4	PC5
Unnamed: 0	-0.302738	0.205031	0.010248	-0.090026	-0.079790
Episodes	0.048736	-0.054710	0.119029	-0.058786	0.024488
Rating	0.161841	0.354290	-0.159220	-0.054612	-0.044270
Members	0.211115	-0.042377	0.145013	0.084159	-0.002313
Duration_minutes	0.165402	-0.066455	-0.352033	0.193962	0.071841
...	...	...	...	...	...
Special	-0.027811	-0.020083	-0.021411	-0.049734	-0.007989
TV Special	0.013770	-0.070465	-0.092518	0.039178	-0.024430
Other_Type	-0.000000	0.000000	-0.000000	-0.000000	0.000000
year_circle_sin_normalized	0.007900	0.001984	-0.064384	0.219574	-0.500085
year_circle_cos_normalized	-0.017016	-0.074641	0.031438	0.014414	-0.193930

	PC6	PC7	PC8	PC9	PC10
Unnamed: 0	-0.123886	-0.013341	0.097808	-0.141856	0.056830
Episodes	0.038471	0.168378	0.036812	0.074901	0.117630
Rating	0.037433	-0.074884	0.104808	-0.081055	0.089092
Members	-0.076998	-0.145813	0.081722	0.022570	-0.039791
Duration_minutes	-0.072063	0.041864	0.048037	-0.050551	-0.187035
...	...	...	...	...	...
Special	0.176382	-0.163245	-0.194599	0.027797	0.046503
TV Special	0.076965	-0.001264	0.012671	-0.036500	-0.046714
Other_Type	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
year_circle_sin_normalized	0.105217	0.012017	-0.060037	0.029412	0.089268

...

DIMENSIONALITY REDUCTION



EVALUATION METRIC

```python
def evaluate_model(X_test, y_test, model):
    # Evaluate the model on the test data
    results = model.evaluate(X_test, y_test, verbose=0)
    mse, mae, rmse = results[0], results[1], results[2]

    # Calculate R² score
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)

    # Print the evaluation metrics
    print(f"Test MSE: {mse}")
    print(f"Test MAE: {mae}")
    print(f"Test RMSE: {rmse}")
    print(f"Test R² Score: {r2}")
```

[11] ✓ 0.0s

MODEL TRAINING
METHOD ONE: NEURAL NETWORK

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss=MeanSquaredError(),
              metrics=[MeanAbsoluteError(), RootMeanSquaredError()])

model.summary()
```

[13] ✓ 0.1s

```
C:\Users\ali18\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

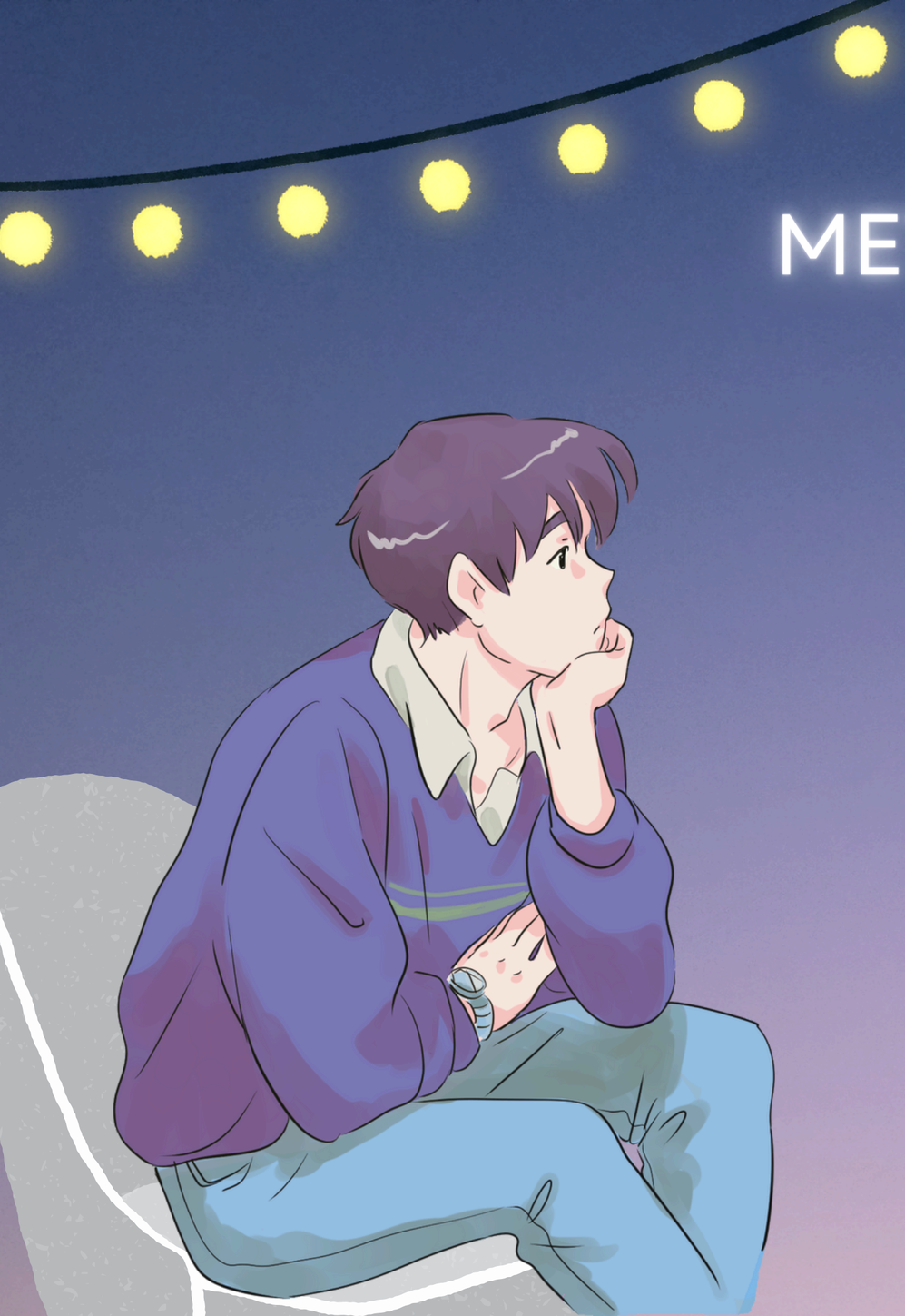
Model: "sequential"

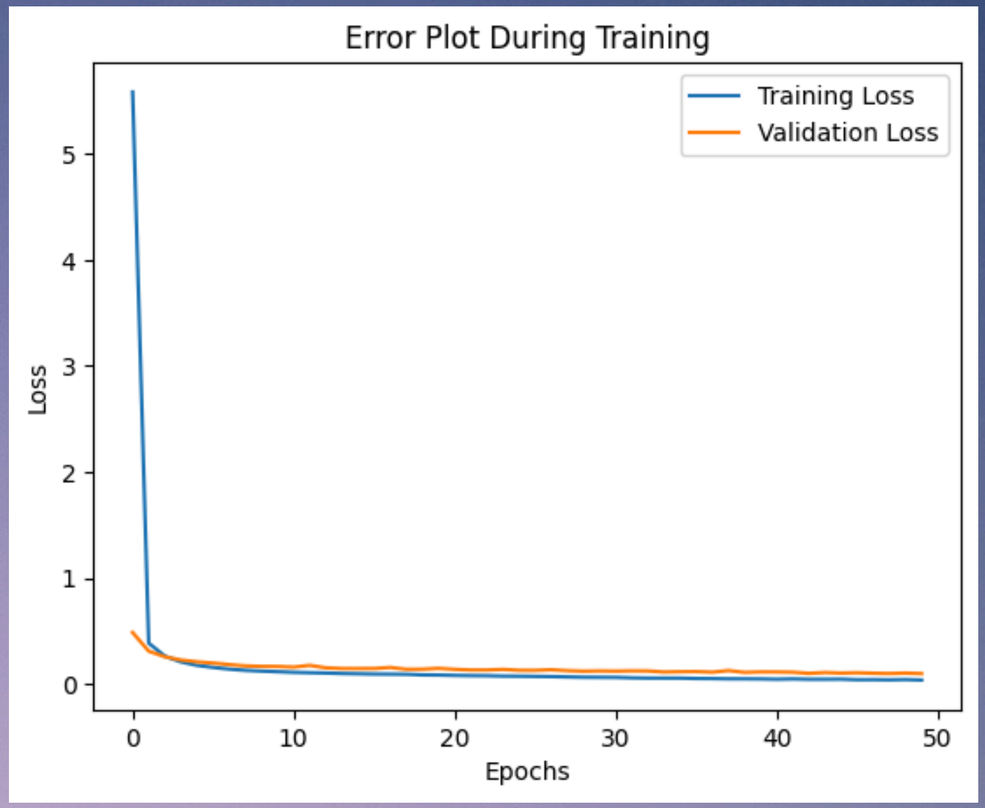
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	5,824
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

MODEL TRAINING
METHOD ONE: NEURAL NETWORK

```
[15]  evaluate_model(X_test_scaled, y_test, model)
      ✓ 0.5s

...   107/107 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
      Test MSE: 0.1031646803021431
      Test MAE: 0.21112336218357086
      Test RMSE: 0.3206464350223541
      Test R² Score: 0.873852871401869
```

MODEL TRAINING
METHOD ONE: NEURAL NETWORK





MODEL TRAINING
METHOD ONE: NEURAL NETWORK WITH PCA

```
X_train, X_test, y_train, y_test = train_test_split(transformed_df, y, test_

model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss=MeanSquaredError(),
              metrics=[MeanAbsoluteError(), RootMeanSquaredError()])

model.summary()
```

[17] ✓ 0.0s

```
C:\Users\ali18\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	4,608
dense_5 (Dense)	(None, 32)	2,080
dense_6 (Dense)	(None, 16)	528

MODEL TRAINING METHOD ONE: NEURAL NETWORK WITH PCA

```
[19]  evaluate_model(X_test, y_test, model)
      ✓ 0.3s

...   107/107 ──────────────────────────────── 0s 1ms/step
      Test MSE: 0.21007582545280457
      Test MAE: 0.3261140286922455
      Test RMSE: 0.4576930105686188
      Test R² Score: 0.7429764393954275
```

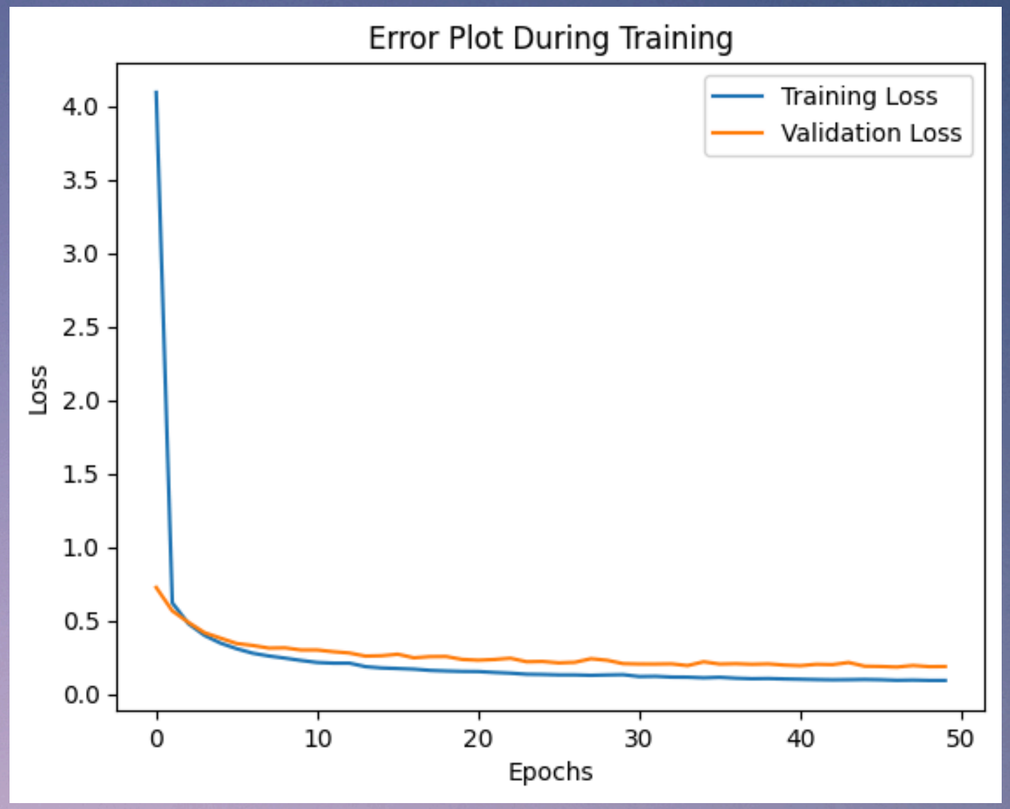
SAVE PCA OUTPUT

```
y = np.array(y)  # replace with your actual target variable data

# Create a DataFrame from the PCA-transformed data
df_transformed = pd.DataFrame(transformed_df)

# Add the target variable to the DataFrame
df_transformed['Score'] = y

# Save the DataFrame to a CSV file
df_transformed.to_csv('transformed_data.csv', index=False)


# Load the PCA-transformed data
df_transformed_loaded = pd.read_csv('transformed_data.csv')

# Display the first few rows of each DataFrame for comparison
print("Original Data:")
print(df.head())

print("\nPCA-Transformed Data:")
print(df_transformed_loaded.head())
```

[22] ✓ 1.7s

```
Original Data:
   Unnamed: 0  Episodes  Rating  Score    Members  Duration_minutes  Episodic
0           0      28.0       2   9.38   708753.0              24.0         1
1           1      64.0       3   9.09  3341090.0              24.0         1
```

MODEL TRAINING METHOD ONE: NEURAL NETWORK WITH PCA



METHODS TWO & THREE

```
# Define features and target variable
X = df.drop(columns=['Score'])  # Features
y = df['Score']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[23] ✓ 0.0s

```
# Define parameter grids
param_grids = {
    "SVM": {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'poly', 'rbf']
    },
    "Decision Tree": {
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 10, 20],
        'min_samples_leaf': [1, 5, 10]
    }
}

# Initialize models
models = {
    "SVM": SVR(),
    "Decision Tree": DecisionTreeRegressor(random_state=42)
}
```

[24] ✓ 0.0s

METHODS TWO & THREE

```python
def grid_search_evaluate(models, param_grids, X_train, y_train, X_test, y_test):
    results = {}

    for name, model in models.items():
        print(f"\nPerforming Grid Search for {name}...")
        grid_search = GridSearchCV(model, param_grids[name], cv=3, scoring='neg_mean_squared_error', n_jobs = -1)
        grid_search.fit(X_train, y_train)

        best_model = grid_search.best_estimator_
        print(f"Best parameters for {name}: {grid_search.best_params_}")

        # Evaluate the best model
        mse, mae, rmse, r2 = evaluate_model(X_test, y_test, best_model)

        results[name] = {
            "best_params": grid_search.best_params_,
            "mse": mse,
            "mae": mae,
            "rmse": rmse,
            "r2": r2
        }

    return results

# Perform Grid Search and evaluate models
results = grid_search_evaluate(models, param_grids, X_train, y_train, X_test, y_test)
print(results)
```

THANK YOU FOR LISTENING!

Don't hesitate to ask any questions!