Group 03 – Project 4 Write Up

This analysis was conducted by Carlos Ruiz, Daniel Purrier, Steven Madden and Amar Patil. In it we analyze Anime Recommendation Database from the Kaggle Dataset, to provide a content based recommendation and visualization. Our goal with this dataset was to build an Anime recommender that can take into account viewer preferences of type and genre, returning new anime properties the user would enjoy.

We chose this dataset based on file size, number of rows, and perceived amount of cleaning necessary. We estimated we might only lose around 3.5% of rows when cleaning was done. We attempted to answer

.

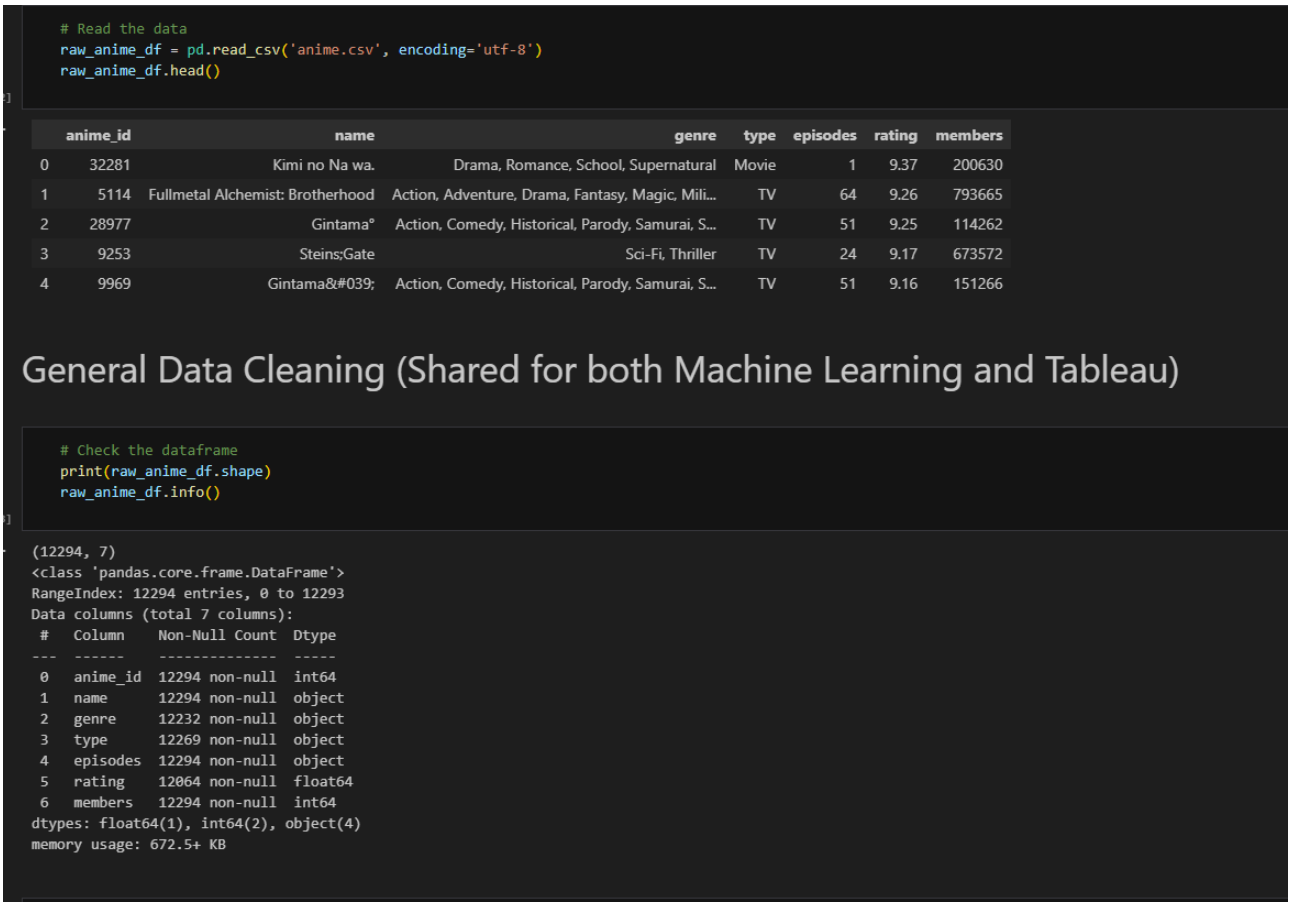
1. Could we build the basic recommender?
2. Could we refine it based on genre and type preferences?
3. Could we then publish it as a usable web app with interactive data visualizations?

Data Cleaning:

To proceed with answering these questions, we first needed to look at the dataset we are working with and modify it to better fit the questions that we are posing. Below is our raw data set, it has 8 columns and provides a good working amount of data. Reviewing the columns and their titles, we considered which columns would have the most impact in answering our questions, which columns need to be manipulated, and which can be dropped all together.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231 entries, 0 to 230
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Serial Number    231 non-null    int64
 1   Country          231 non-null    object
 2   Total Cases      231 non-null    object
 3   Total Deaths     225 non-null    object
 4   Total Recovered  211 non-null    object
 5   Active Cases     213 non-null    object
 6   Total Test       213 non-null    object
 7   Population       229 non-null    object
dtypes: int64(1), object(7)
memory usage: 14.6+ KB
```

We started by reading in our dataset looking at its shape and searching for nulls. Since the impact of dropping null values is less than 1% in genre and type columns, we can drop all rows with null values. We lost another 2.5% when we found entries in the episode column containing non-numeric values.



```
# Read the data
raw_anime_df = pd.read_csv('anime.csv', encoding='utf-8')
raw_anime_df.head()
```

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

General Data Cleaning (Shared for both Machine Learning and Tableau)

```
# Check the dataframe
print(raw_anime_df.shape)
raw_anime_df.info()
```

```
(12294, 7)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12294 entries, 0 to 12293
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   anime_id  12294 non-null  int64
 1   name      12294 non-null  object
 2   genre     12232 non-null  object
 3   type      12269 non-null  object
 4   episodes  12294 non-null  object
 5   rating    12064 non-null  float64
 6   members   12294 non-null  int64
dtypes: float64(1), int64(2), object(4)
memory usage: 672.5+ KB
```

Look at null values

```
[4]
# Look at null values in the dataframe
raw_anime_df.isnull().sum()
```

```
anime_id      0
name          0
genre        62
type         25
episodes      0
rating      230
members       0
dtype: int64
```

```
[5]
# Convert null values to percentages to understand the impact of dropping them
(raw_anime_df.isnull().sum() / 12294 * 100).apply(lambda x: f'{x:.3f}%')
```

```
anime_id    0.000%
name        0.000%
genre       0.504%
type        0.203%
episodes    0.000%
rating      1.871%
members     0.000%
dtype: object
```

We found some strange symbols in some of the names, so we removed them. Then it was a matter of taking all the unique entries in the genre and type columns and turning them into their own Boolean columns, and then dropping the genre and type columns.

```
# Replace hearts and stars with a space
anime_df.loc[anime_df['name'].str.contains('♥'), 'name'] = anime_df['name'].str.replace('♥', ' ')
anime_df.loc[anime_df['name'].str.contains('★'), 'name'] = anime_df['name'].str.replace('★', ' ')
anime_df.loc[anime_df['name'].str.contains(r'[^\x00-\x7F]', na=False)]
```

	name	episodes	rating	Action	Adventure	Cars	Comedy	Dementia	Demons	Drama	Ecchi	Fantasy	Game	Harem	Hentai	Historical	Horror	Josei	Kids	Magic	MartialArts	Mecha	Military
0	Kimi no Na wa.	1	9.37	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	Fullmetal Alchemist: Brotherhood	64	9.26	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1
2	Gintama	51	9.25	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
3	Steins Gate	24	9.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	Gintama'	51	9.16	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Once done, we dropped just a few more columns, when we realized several genre columns were of an 18+ plus variety, so we decided to drop them.

	anime_id	name	episodes	rating	members	Action	Adventure	Cars	Comedy	Dementia	Demons	Drama	Fantasy	Game	Historical	Horror	Josei	Kids	Magic	MartialArts	Mecha	Military
0	32281	Kimi no Na wa.	1	9.37	200630	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	5114	Fullmetal Alchemist: Brotherhood	64	9.26	793665	1	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1
2	28977	Gintama	51	9.25	114262	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
3	9253	Steins Gate	24	9.17	673572	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	9969	Gintama'	51	9.16	151266	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0

With that done, we were ready to start work

Question 1: Could we build a basic recommender?

Utilizing jupyter notebook, we imported pandas, numpy, pickle, and scikit learn as sklearn (for processing)

```
# Import the required modules
import pandas as pd
pd.set_option('display.max_columns', None)
import numpy as np

# Pre-Processing
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
import pickle

# Models
from sklearn.neighbors import NearestNeighbors

# suppress warnings
import warnings
warnings.filterwarnings('ignore')
```

.

Following an in class example we then built the recommender, first by programming the model_fit: then the recommender itself.

```
def model_fit(anime_length):
    # Step 1: Load the dataset
    df = pd.read_csv("anime_ml.csv")

    # Remove any rows with missing values and reset the index
    df = df.dropna(how="any").reset_index(drop=True)


    # Remove duplicate anime based on track_id and reset index again
    df = df.drop_duplicates(subset=["name"]).reset_index(drop=True)


    # Step 2: Define the columns for metadata and features
    meta_cols = ["anime_id","name"]  # Metadata columns to keep anime info
    feature_cols = ['episodes', 'rating', 'members', 'Action', 'Adventure', 'Cars',
       'Comedy', 'Dementia', 'Demons', 'Drama', 'Fantasy', 'Game',
       'Historical', 'Horror', 'Josei', 'Kids', 'Magic', 'MartialArts',
       'Mecha', 'Military', 'Mystery', 'Parody', 'Police',
       'Psychological', 'Romance', 'Samurai', 'School', 'SciFi', 'Seinen',
       'Shoujo', 'ShoujoAi', 'Shounen', 'ShounenAi', 'SliceofLife', 'Space',
       'Sports', 'SuperPower', 'Supernatural', 'Thriller', 'Vampire', 'Movie',
       'Music', 'ONA', 'OVA', 'Special', 'TV']

    # Step 3: Define preprocessing steps for the data

    # Preprocessing for numeric features: fill missing values with the mean and standardize the values
    numeric_features = []
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),  # Fill missing values with the mean
        ('scaler', StandardScaler())])  # Standardize features

    # Preprocessing for binary features (if any): fill missing values with most frequent and label encode
    binary_features = []
    binary_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent', missing_values=pd.NA)),  # Fill missing values
        ('label', OrdinalEncoder())])  # Convert binary features to ordinal values

    # Preprocessing for categorical features: fill missing values and apply one-hot encoding
    categorical_features = []
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent', missing_values=pd.NA)),  # Fill missing values
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])  # Apply one-hot encoding

    # Combine all preprocessing steps into a single preprocessor
    preprocessor = ColumnTransformer(
        transformers=[
```

```
# Combine all preprocessing steps into a single preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),  # Numeric features preprocessing
        ('binary', binary_transformer, binary_features),  # Binary features preprocessing (if any)
        ('cat', categorical_transformer, categorical_features)])  # Categorical features preprocessing


# Step 5: Prepare the feature matrix for the Nearest Neighbors model
X = df.loc[:, feature_cols]  # Select the feature columns

# Apply preprocessing to the feature matrix
# preprocessor.fit(X)  # Fit the preprocessor to the feature matrix
# X_preprocessed = preprocessor.transform(X)  # Transform the feature matrix

# Step 6: Initialize the Nearest Neighbors model
# # define the number of nearest neighbors to consider and use cosine similarity as the metric
k = anime_length
model1 = NearestNeighbors(n_neighbors=k, metric="cosine")

# Fit the Nearest Neighbors model to the preprocessed data
# model1.fit(X_preprocessed)
model1.fit(X)
# Save the model to a pickle file

with open('anime_length_name.pkl', 'wb') as model_file:
    pickle.dump(model1, model_file)
```

Then the recommender itself:

```
def make_recommendation(anime_model, anime_name):

    # Step 1: Load the dataset
    df = pd.read_csv("anime_ml.csv")


    # Load Pickle file
    filename = anime_model
    model1 = pickle.load(open(filename, 'rb'))

    feature_cols = ['episodes', 'rating', 'members', 'Action', 'Adventure', 'Cars',
       'Comedy', 'Dementia', 'Demons', 'Drama', 'Fantasy', 'Game',
       'Historical', 'Horror', 'Josei', 'Kids', 'Magic', 'MartialArts',
       'Mecha', 'Military','Mystery', 'Parody', 'Police',
       'Psychological', 'Romance', 'Samurai', 'School', 'SciFi', 'Seinen',
       'Shoujo', 'ShoujoAi', 'Shounen', 'ShounenAi', 'SliceofLife', 'Space',
       'Sports', 'SuperPower', 'Supernatural', 'Thriller', 'Vampire', 'Movie',
       'Music', 'ONA', 'OVA', 'Special', 'TV']

        # Step 4: Get the target track's ID based on track name and artist
    # Select the most popular track by track_name and track_artist combination
    anime_id = df.loc[(df.name == anime_name)] \
                .sort_values(by="rating", ascending=False).anime_id.values[0]
    # anime_id = 22319

    # Step 7: Extract features of the anime
    anime_features = df.loc[df.anime_id == anime_id, feature_cols]  # Get the feature vector for the target anime
    # anime_features_preprocessed = preprocessor.transform(anime_features)  # Preprocess the target track features
    # print(anime_features)
    # Step 8: Find the nearest neighbors (anime most similar to the target selection)
    # distances, indices = model1.kneighbors(anime_features_preprocessed)  # Get distances and indices of neighbors
    distances, indices = model1.kneighbors(anime_features)  # Get distances and indices of neighbors

    # Step 9: Retrieve the metadata of the recommended tracks
    animes = df.iloc[indices[0]]  # Select anime corresponding to the nearest neighbors
    animes["distance"] = distances[0]  # Add the distance of each neighbor as a new column

    # Step 10: Filter the columns for the final output
    cols = animes.columns # you can explicitly choose to return specific columns here
    animes = animes.loc[:, cols]  # Keep the relevant columns
    animes = animes.sort_values(by="distance")  # Sort the tracks by their distance (most similar first)

    # Step 11: Return the recommended tracks as a list of dictionaries
    return animes.to_dict(orient="records")
```

Recommender System:

1. Candidate generation
    a. In this first state, the system starts from huge dataset and generates a smaller subset of data. Example in Youtube reduces large amount of videos down to hundreds or thousands. This is first stage of Recommendation. two common candidate generation approaches:
        i. Content-based Filtering
            1. Uses similarity between items to recommend items similar to what user likes

2. Ex: if User watches dog videos, then the system can recommend cute animals videos to that user.

ii. Collaborative Filtering
1. Recommends items based on the behavior or preferences of similar users.
2. Ex: If used A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A.

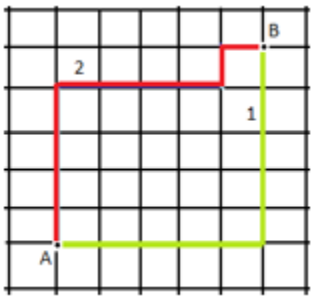
iii. Hybrid Recommender
1. Hybrid recommender systems combine collaborative filtering (analyzing user behavior and preferences) and content-based filtering (focusing on item characteristics) to provide more accurate and diverse recommendations.

2. Scoring
   a. This model scores and ranks the candidates in order to select the set of items (on the order of 10) to display to the user. This is subset of Candidate generation
3. Re-ranking
   a. This is final ranking which takes into account dislikes or likes of newer content. Re-ranking helps to ensure diversity, freshness and fairness.

To determine the degree of similarity, most recommendation systems rely on one or more of the following:

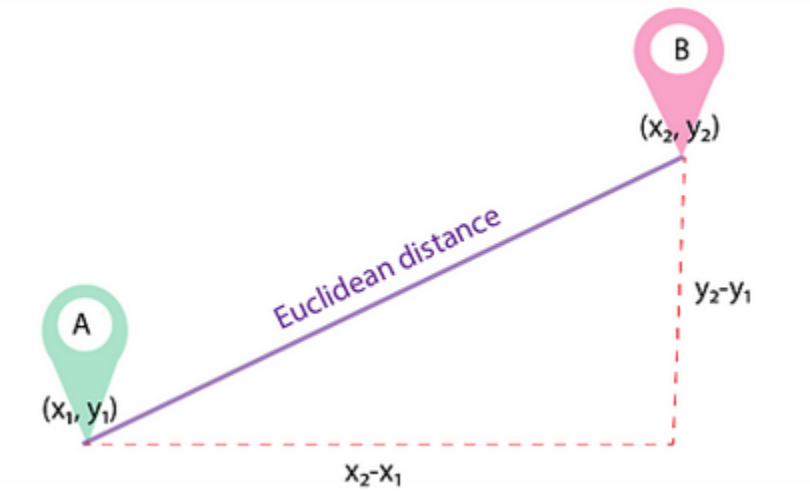
1. Manhattan

Also, know as city block distance, or taxicab geometry wherein the distance is measured between two data points in Grid-like path. As shown below. Mainly used where high dimensionality in the data.



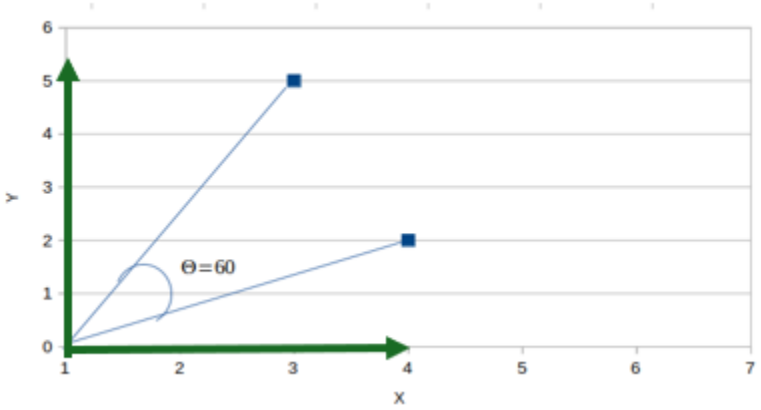
2. Euclidean

This is the shortest distance between 2 data points in the plane. Mainly used for the smaller dimensionality problems.



3. Cosine Distance

Also known as Cosine Similarity is used to find similarities between two data points. Cosine similarity is given by Cos θ, and cosine distance is 1- Cos θ. Mainly used in the Collaborative Filtering based recommender systems to offer future recommendations



Question 2: Could we refine it based on genre and type preferences?

After the base code was working, several attempts were made to have the recommender offer suggestions via genre and type preferences. While the code worked within jupyter notebook, we reached a problem implementing it on the web app side.

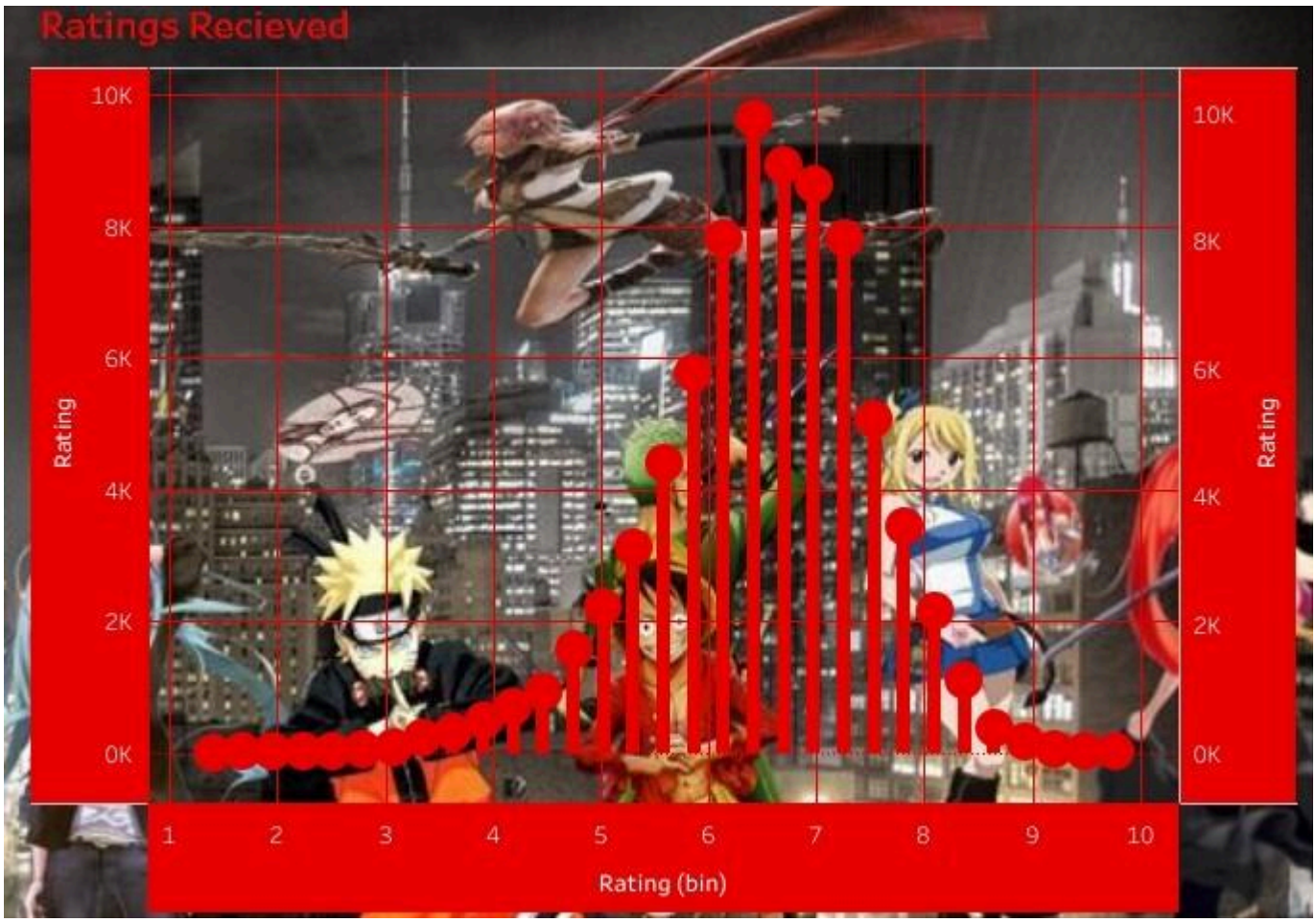
```
# Example usage
anime_length = 10
anime_name = None  # Can be None if no specific anime is needed
genres = ["Comedy"]  # Specify the genres you want to filter by (can be None)
type_filter = None  # Specify the types to filter by (can be None)
rating_min = None  # Minimum rating (can be None if no minimum is needed)
episodes_min = 1  # Minimum episodes (can be None if no minimum is needed)

response = make_recommendation(anime_length, anime_name=anime_name, genres=genres, type_filter=type_filter, rating_min=rating_min, episodes_min=episodes_min)

# To test
pd.DataFrame(response)
```

	anime_id	name	episodes	rating	members	Action	Adventure	Cars	Comedy	Dementia	Demons	Drama	Fantasy	Game	Historical	Horror	Josei	Kids	Magic	M
0	16389	Komachi to Dangorou	2	5.38	215	0	1	0	1	0	0	0	0	0	0	0	0	1	0	
1	17163	Peeping Life: The Perfect Evolution Specials	3	5.38	220	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
2	7449	Pacusi no Uta	1	5.38	225	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
3	13087	Youkoso Uchuujin	1	5.38	190	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
4	9303	Noel no Fushigi na Bouken	1	5.38	133	0	1	0	1	0	0	0	1	0	0	0	0	0	0	
5	22511	Kobutori	1	5.38	125	0	0	0	1	0	1	0	0	0	0	0	0	0	0	

It was decided we would try to implement the refined search through the web app itself.

Question 3: Could we publish it as a usable web app with interactive data visualizations?

Utilizing Tableau, we took our clean dataset and started building visualizations that could be interacted with. We wanted these visualizations to reference things in the data such as ratings of animes in the data set, the size of various genres and user feedback.



As well as having quick reference data an interested user might find helpful for their searches.

Then it was a matter of deploying all of it to the web app.

Machine Learning Recommender Website

Our group developed a comprehensive machine learning recommender website designed to provide list of anime recommendations based on a certain anime characters. The website is structured to offer insights into anime name, genre, ratings and episodes. The website has various sections, including a home page, interactive recommender dashboard, Tableau, an "About Us", page that introduces the team, proposal write-up page and a citation page that credits data sources used.

Home Page:

The home page sets the tone for the website by discussing

Dashboard:

The dashboard is dynamic and can be alter as per the user's anime choice. First user selects anime name from the dropdown menu, this will trigger the ML recommender which is content based recommender to populate the list of anime names. Further the user can select additional filters like genre, type (movie, music, and TV etc.), ratings and episode.

Tableau:

Tableau was used to create a visualization story of the anime data. The original data did not require much cleaning, however there were a few explicit episode categories that were removed. Once cleaned multiple charts and visualization were created to explain the data. On the first page shows selections made by anime fans based off of genre. A tree map and a bubble chart were used to display this. A highlight chart can also be found here that showcases the name, rating, episode count, and members of the entire data set. This highlight chart can be searched by typing in the name of any anime a searcher can think of. Once located the highlight chart will display all findings related to the search. On page two is a highlight chart which will only display anime episodes that have a rating of 7 or greater, and can also be searched. The donut chart on this page shows the episode types of the data and finally, a lollipop chart depicts the ratings received for each episode. Multiple functions were used to create this story. The split function was used with the bubble chart as most of the episodes had multiple genre assigned to it in the original data and had to be split from them. The contain function was used to limit results when searching for anime with a rating higher than 7. The last function used, was

needed to address the case sensitive issue while searching the highlight charts. Initially anime could only be located if each word type in began with a capital. After some searching it was discovered the regexp_match function could fix this. Once implemented searches could now be completed without case sensitive issues.

About Us Page:

This page introduces the team members, providing a personal touch to the project. Each member's background is outlined, highlighting their expertise and contributions. This helps build credibility and showcases the diverse skill set within our team.

Citation Page:

The citation page is essential for maintaining academic and professional integrity. It lists all the data sources used in the project, ensuring transparency and giving credit to original authors and datasets. This section shows that the project is well-researched and relies on credible sources .

Technical Aspects:

The website likely utilizes a backend system, possibly built with Flask (as indicated using Python scripts like app.py and sqlHelper.py), to manage data retrieval and processing. The SQLite database and .csv serves as the data storage, housing information on anime, which is then used to generate the visualizations and data tables on the site.

The website is styled using Bootstrap, with the "Minty" theme providing a clean and modern look. The consistent use of this theme across all pages ensures a cohesive user experience. The use of JavaScript libraries like D3.js and Plotly.js for visualizations, along with DataTables for data management, adds a high level of interactivity to the site, allowing users to engage with the data meaningfully.

Conclusion:

The website created by our group is a well-rounded project that effectively combines data analysis, visualization, and user interaction to provide a comprehensive understanding of the pandemic's global impact. The website is not only informative but also accessible, thanks to its user-friendly design and interactive features. The inclusion of various sections, such as the tableau and dashboard, ensures that users can explore the data from multiple perspectives, making the project a valuable tool for anyone interested in high level of anime recommendation.

Bias and Limitations

When considering our dataset there are several biases and limitations that are found within that need to be considered when drawing conclusions. It is limited by the fact that this dataset lacks the most recent anime collection. It was last updated 8 years ago. Additionally, if a year or studio column were present, the team would have added additional layers of features to provide recommendations. And currently, you must no at least one anime for the recommender to work.

Future Work

Some examples of features the team would like to work on in the future are a further refining of the nearest neighbors to yield stronger results. Added search functionality with other columns, like year a project was released or the studio that produced it. Add the ability to show cover art of recommended animes. And further optimize the recommender for people who have no experience with anime

Conclusion

With more than two thirds of the population of the United States enjoying some kind of anime, a recommender may be a useful tool for finding new shows and movies to watch amid the wide variety available. Hopefully a project like this can take some of the guesswork out of finding new entertainment.

Bibliography

- Anime Recommendations Database (kaggle.com)

  Candidate generation overview | Machine Learning | Google for Developers

  Hybrid Recommender Systems: Beginner's Guide (marketsy.ai)
-
- https://select2.org/data-sources/ajax
-
- https://makitweb.com/loading-data-remotely-in-select2-with-ajax/
- worldpopulationreview.com/country-rankings/anime-popularity-by-country
- git.bootcampcontent.com/boot-camp-consortium-east-coast/DATA-PT-EAST-APRIL-041524/-/tree/main/01-Lesson-Plans/23-Project-4-Week-1/3/BOOOTH_RECOMMENDER_EXAMPLE?ref_type=heads
- public.tableau.com/app/profile/sakib.mahmud1560/viz/AnimeAnalyticsthroughStudios/AnimeAnalyticsDashboard