Indiana University, Bloomington

Anime CRP

Classifier, Recommender, Predictor

Authors

Eanumula Eswar Sai Yashwanth

Ram Kiran Devireddy

Gowri Shankar Badugu

ABSTRACT

Models which are automated algorithms like **classifiers** and **recommenders** are used frequently nowadays. The use of data classifiers and recommenders which are used to analyze large data sets, to classify the data in it and to recommend something out of it are applied in many fields. One of those applications is **Anime CRP** (Anime Classifier, Recommender and Predictor) which uses the anime data set and classifies the data in it and recommends **anime** to user. This will be useful in entertainment field. Firstly, the Classifier classifies the anime based on a given category and suggests whether a given anime is recommended to the user or not. Secondly, using the above classifier along with clustering, the dataset can be grouped as Beginner, Pro, and Advanced based on number of episodes and genre to enable **categorial recommendation**. Thus, the Recommender recommends multiple groups of anime to watch for a user based on his/her preferences like **genre**, number of **episodes**, **popularity**, and **ranking**. Here, the recommender is designed based on a data mining classification algorithm called **K-Nearest Neighbors (KNN)** and on the other hand, the classifier is designed with a data mining clustering algorithm called **K-Means Clustering**. Finally, the future developments of this project would be building an anime **predictor** (predicting the anime a given user will be watching next) with the help of **association rules**.

Keywords: classifiers, recommenders, Anime CRP, anime, categorial recommendation, genre, episodes, popularity, ranking, K-Nearest Neighbors (KNN), K-Means Clustering, predictor and Association rules.

INTRODUCTION

One of the major applications of data mining algorithms like KNN and K-Means clustering are to build the automated algorithms like classifiers and recommenders which work on past data.

1.1 Classifiers:

Classifiers are used in many fields like health & safety, military, business, entertainment fields etc. The growth of Classifiers usage in entertainment field is increased as number of anime watchers are increased nowadays so that the OTT's like Netflix, Hulu and Amazon are using the classifiers to classify the anime based on genre, number of episodes, ranking and popularity.

1.2 Recommenders:

Usage of Recommenders in many sectors has also been increased which the algorithms recommend something based on past data. In entertainment fields, the platforms like Netflix, Hulu and Amazon uses these recommenders where it recommends the user to watch anime based on the history of anime the user watched. The recommendations are made based on the anime genres, length of episodes, number of episodes and ranking of anime he/she watched.

1.3 Predictor:

Predictors are often used in entertainment field these days by famous streaming services like Netflix. Streaming platforms these days are using the watch history of their customers to predict the most likely shows/movies they'll be watching to include in their platform. Such predictor can be built using association rule mining. Therefore, we can also build a predictor for anime using the anime watch history of different users.

1.4 Dataset:

Data on user preferences for about 16,214 anime, collected from 47k individuals, are included in this data set. This data set also includes a collection of the user ratings of about 130k for each anime that they have added to their completed list. Thus, though this dataset, we'll get the information related to anime (genre, rating, type, popularity etc.) and info related to user's anime watch history (anime watch history of the users).

- According to Kaggle, this dataset was pulled from myanimelist.net API.
- **anime.csv**
  1. u_id: a unique identification number to differentiate each anime. More like anime_id.
     Sample values: 1, 5, 6, 9 …...,40960
  2. title: contains complete name of the anime. More like name.
     Sample values: One Piece, Naruto, Fullmetal Alchemist etc
  3. synopsis: shows the summary of that particular anime.
  4. genre: gives us the information about which genre the anime belongs to.
     Sample values: ['Sci-Fi', 'Shounen', 'Adventure', 'Mystery', 'Drama', 'Fantasy'] etc
  5. aired: range of date showing the starting and ending dates of anime.
     Sample values: Oct 4, 2015, to Mar 27, 2016
  6. episodes: tells us the number of episodes the anime contains.
     Sample values: 1, 64, 1015 etc
  7. members: shows the community of the anime
  8. popularity: an evaluation metric of the anime showing the popularity of the anime among the anime community.
  9. ranked: rank of the anime among other anime.

DISCUSSION

K-Nearest Neighbors (KNN):

Basic idea behind KNN is If it walks like a duck, quacks like a duck, then it's probably a duck. The value of k is the number of nearest neighbors to retrieve. It is a method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote). Based on those K neighbors' label's, it labels the unknown record. The method requires set of labeled records, Proximity metric to compute distance/similarity between a pair of records like Euclidean distance, and K value.

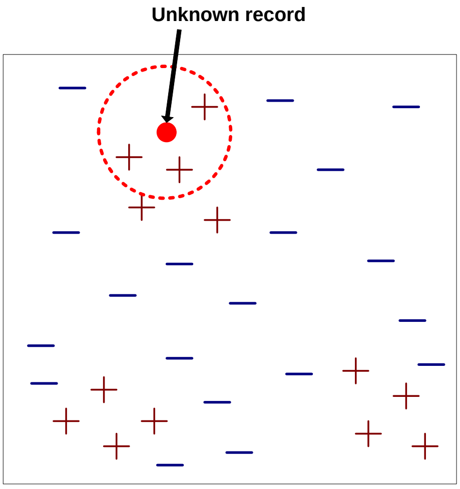


Fig 1.3.1 K-Nearest Neighbors

Determine the class label by taking the majority vote of class labels among the k-nearest neighbors. Weight the vote according to distance weight factor, $w = 1/d^2$.

1.3.1 Challenges in KNN:

1. Data preprocessing is often required. Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes

   Example:

   - height of a person may vary from 1.5m to 1.8m.
   - weight of a person may vary from 90lb to 300lbs.
   - income of a person may vary from $10K to $1M.

2. Choosing the value of k:
   - If k is too small, sensitive to noise points.
   - If k is too large, neighborhood may include points from other classes

3. Handling Missing Values:
   - The respective rows can be deleted.
   - Or else, respective null values can be replaced with the mean value of the respective column.

4. Handling Irrelevant and Redundant Attributes:
   - Irrelevant attributes add noise to the proximity measure.

- Redundant attributes bias the proximity measure towards certain attributes.

1.4. Clustering:

Given a set of objects, place them in groups such that the objects in a group are similar (or related) to one another and different from (or unrelated to) the objects in other groups.

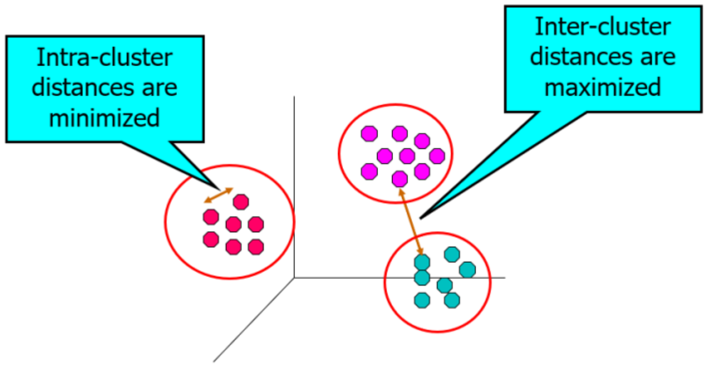
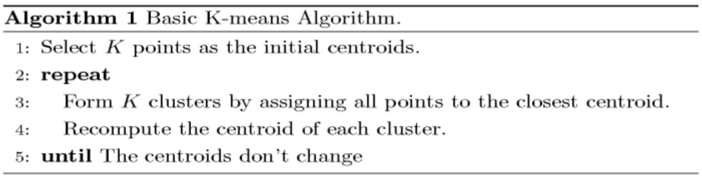

Intra-cluster distances are minimized
Inter-cluster distances are maximized

Fig 1.4.1 Clustering

1.4.1 K-Means Clustering:

A cluster refers to a collection of data points aggregated together because of certain similarities.

Define a target number k, which refers to the number of centroids need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.



Algorithm 1 Basic K-means Algorithm.

```
Select $K$ points as the initial centroids.
repeat
    Form $K$ clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change
```

Fig 1.4.1.1 Pseudo Code for K-Means clustering

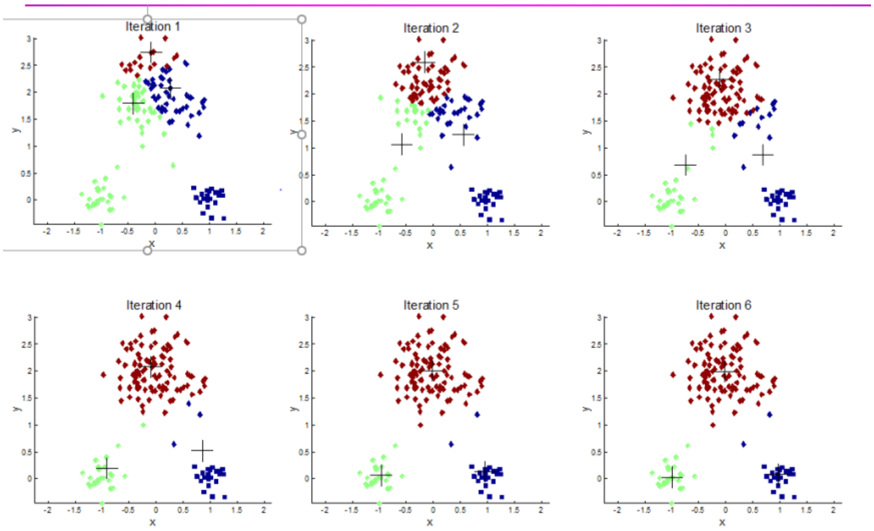


Fig 1.4.1.2 K-Means Clustering

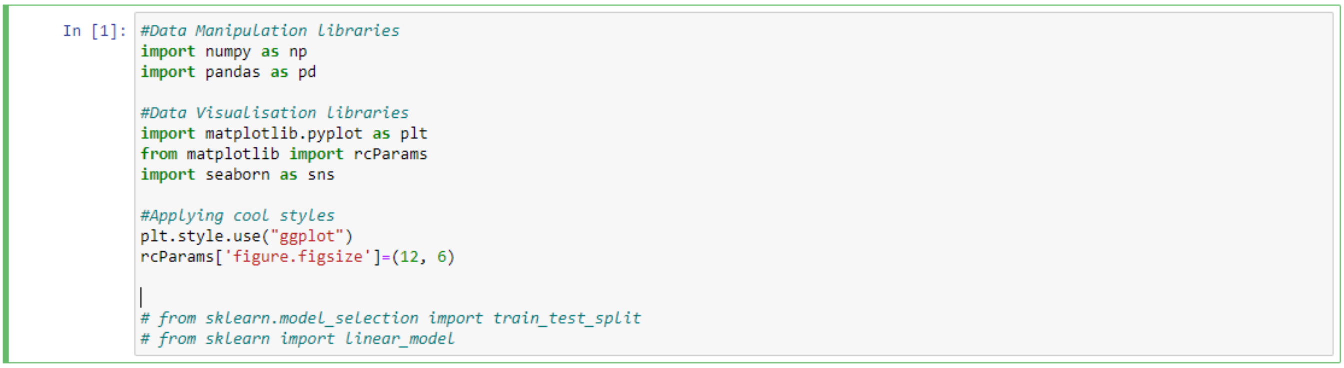
METHODOLOGY

2.1 Classifier:

An algorithm that uses Anime Dataset which is used to design classifier and recommender is proposed to classify the anime data and recommend the anime to the user to watch based on the data present in the dataset. The methods used to classify and recommend are KNN and K-Means clustering with specified K values.

2.1.1 Looking at the Data:

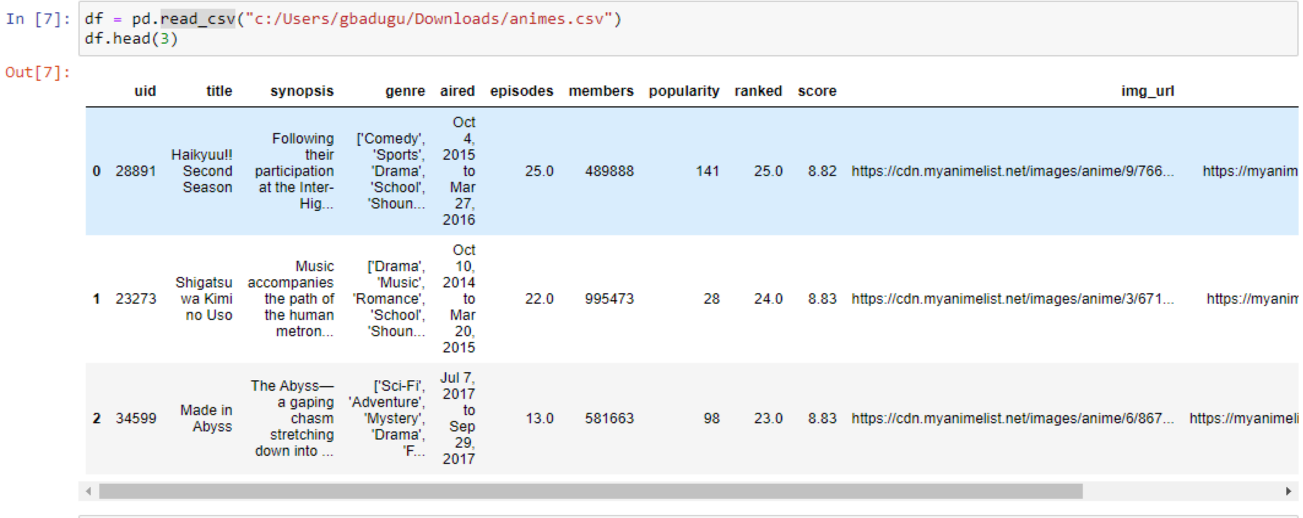
Step 1: Importing the required libraries using command ***"import"***.



```
In [1]: #Data Manipulation libraries
        import numpy as np
        import pandas as pd

        #Data Visualisation libraries
        import matplotlib.pyplot as plt
        from matplotlib import rcParams
        import seaborn as sns

        #Applying cool styles
        plt.style.use("ggplot")
        rcParams['figure.figsize']=(12, 6)

        |
        # from sklearn.model_selection import train_test_split
        # from sklearn import linear_model
```

Fig 2.1.1.1 Importing libraries.

Step 2: Import the data to jupyter notebook using command ***"pd.(read_csv)"***. Printed the first 5 records, just to look at the data loaded using command ***"df.head(5)".***

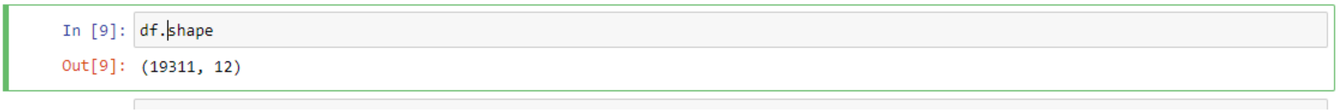


```
In [7]: df = pd.read_csv("c:/Users/gbadugu/Downloads/animes.csv")
        df.head(3)

Out[7]:
```

	uid	title	synopsis	genre	aired	episodes	members	popularity	ranked	score	img_url	
0	28891	Haikyuu!! Second Season	Following their participation at the Inter-Hig...	['Comedy', 'Sports', 'Drama', 'School', 'Shoun...	Oct 4, 2015 to Mar 27, 2016	25.0	489888	141	25.0	8.82	https://cdn.myanimelist.net/images/anime/9/766...	https://myanim
1	23273	Shigatsu wa Kimi no Uso	Music accompanies the path of the human metron...	['Drama', 'Music', 'Romance', 'School', 'Shoun...	Oct 10, 2014 to Mar 20, 2015	22.0	995473	28	24.0	8.83	https://cdn.myanimelist.net/images/anime/3/671...	https://myanim
2	34599	Made in Abyss	The Abyss—a gaping chasm stretching down into ...	['Sci-Fi', 'Adventure', 'Mystery', 'Drama', 'F...	Jul 7, 2017 to Sep 29, 2017	13.0	581663	98	23.0	8.83	https://cdn.myanimelist.net/images/anime/6/867...	https://myanimeli

Fig 2.1.1.2 Importing Dataset

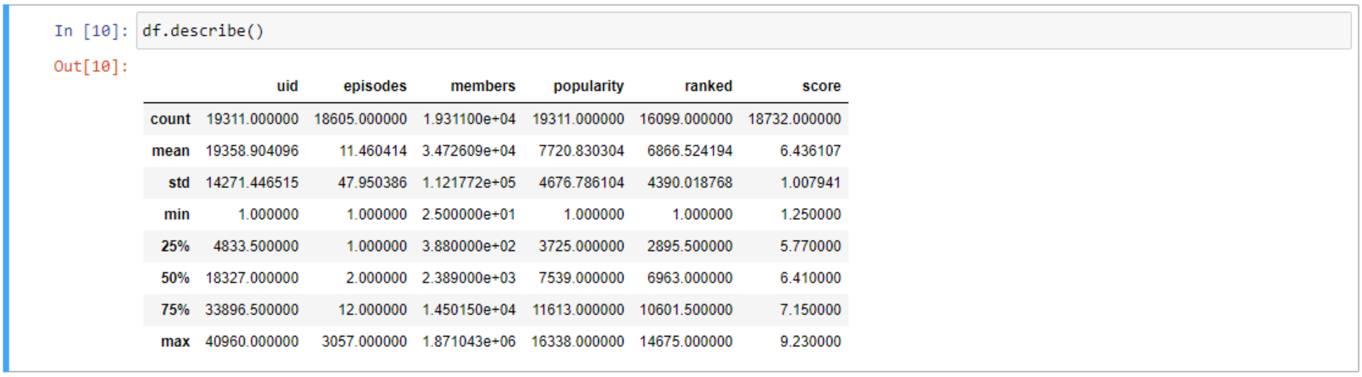
Step 3: Looking at the shape of the dataset i.e., number of columns and rows using command ***"df.shape()".***



```
In [9]: df.shape
Out[9]: (19311, 12)
```

Fig 2.1.1.3 Shape of the Dataset

Step 4: Described the dataset to know the values of mean, standard deviation, correlations of columns in the dataset using command ***"df.describe()".***



```
In [10]: df.describe()
Out[10]:
```

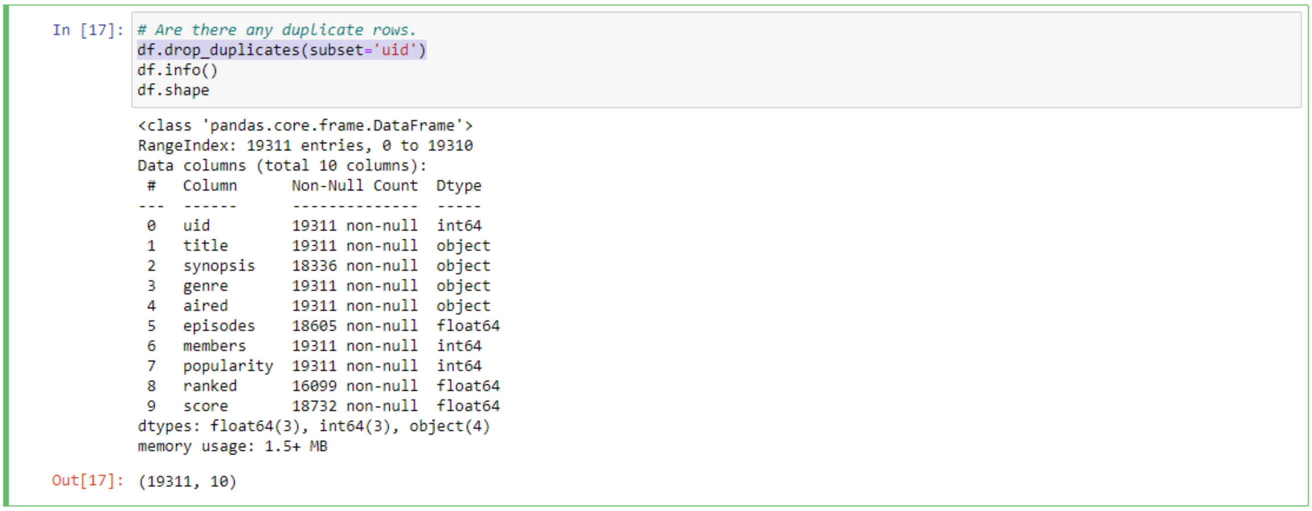
	uid	episodes	members	popularity	ranked	score
count	19311.000000	18605.000000	1.931100e+04	19311.000000	16099.000000	18732.000000
mean	19358.904096	11.460414	3.472609e+04	7720.830304	6866.524194	6.436107
std	14271.446515	47.950386	1.121772e+05	4676.786104	4390.018768	1.007941
min	1.000000	1.000000	2.500000e+01	1.000000	1.000000	1.250000
25%	4833.500000	1.000000	3.880000e+02	3725.000000	2895.500000	5.770000
50%	18327.000000	2.000000	2.389000e+03	7539.000000	6963.000000	6.410000
75%	33896.500000	12.000000	1.450150e+04	11613.000000	10601.500000	7.150000
max	40960.000000	3057.000000	1.871043e+06	16338.000000	14675.000000	9.230000

Fig 2.1.1.4 Describing the dataset.

Basic idea behind the recommender algorithm designed is whenever, user specified his/her preferences like genre, number of episodes, popularity, ranking and name of the anime, the algorithm should return whether the given anime is recommendable or not. Algorithm uses KNN method to label the given anime as recommendable or not using metric Euclidean distance as metric. As mentioned above in section 1.3.1 there are some challenges need to overcome to perform KNN where algorithm uses Exploratory Data Analysis (EDA) as preprocessing the data.

2.1.2 Exploratory Data Analysis:

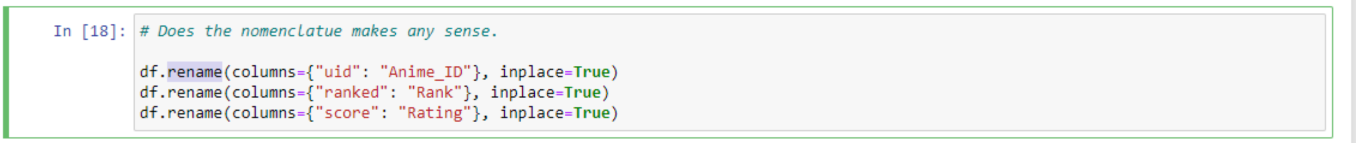
Step 5: Removing the duplicates records from the dataset using uid(anime_id) column using command ***"df.drop_duplicates(subset='uid')"*** **.**



```
In [17]: # Are there any duplicate rows.
         df.drop_duplicates(subset='uid')
         df.info()
         df.shape

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 19311 entries, 0 to 19310
         Data columns (total 10 columns):
          #   Column      Non-Null Count  Dtype  
         ---  ------      --------------  -----  
          0   uid         19311 non-null  int64  
          1   title       19311 non-null  object 
          2   synopsis    18336 non-null  object 
          3   genre       19311 non-null  object 
          4   aired       19311 non-null  object 
          5   episodes    18605 non-null  float64
          6   members     19311 non-null  int64  
          7   popularity  19311 non-null  int64  
          8   ranked      16099 non-null  float64
          9   score       18732 non-null  float64
         dtypes: float64(3), int64(3), object(4)
         memory usage: 1.5+ MB

Out[17]: (19311, 10)
```

Fig 2.1.2.1 Removing duplicate records.

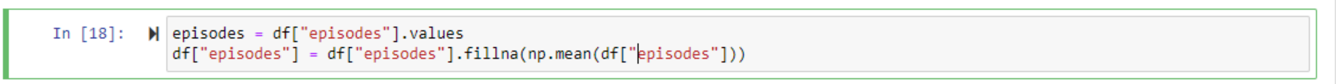
Step 6: Renaming the columns using command ***"df.rename()".***



```
In [18]: # Does the nomenclatue makes any sense.

         df.rename(columns={"uid": "Anime_ID"}, inplace=True)
         df.rename(columns={"ranked": "Rank"}, inplace=True)
         df.rename(columns={"score": "Rating"}, inplace=True)
```

Fig 2.1.2.2 Renaming the column names

Step 7: Replacing the null values with mean value of respective attributes using command "***fillna()***"

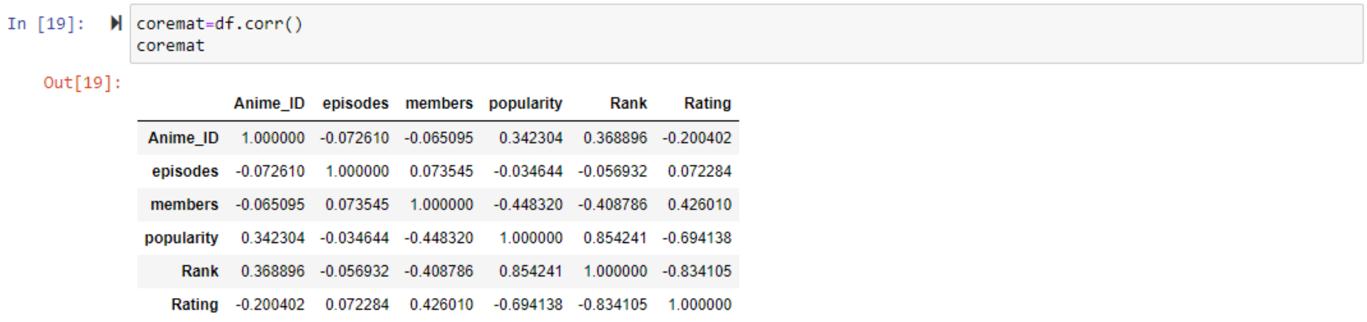


```
In [18]: episodes = df["episodes"].values
         df["episodes"] = df["episodes"].fillna(np.mean(df["episodes"]))
```

Fig 2.1.1.7 Replacing Null Values with mean value

2.1.3 Implementation of KNN:

Step 8: Querying the correlation of each column with each column using command “***corr()***”.



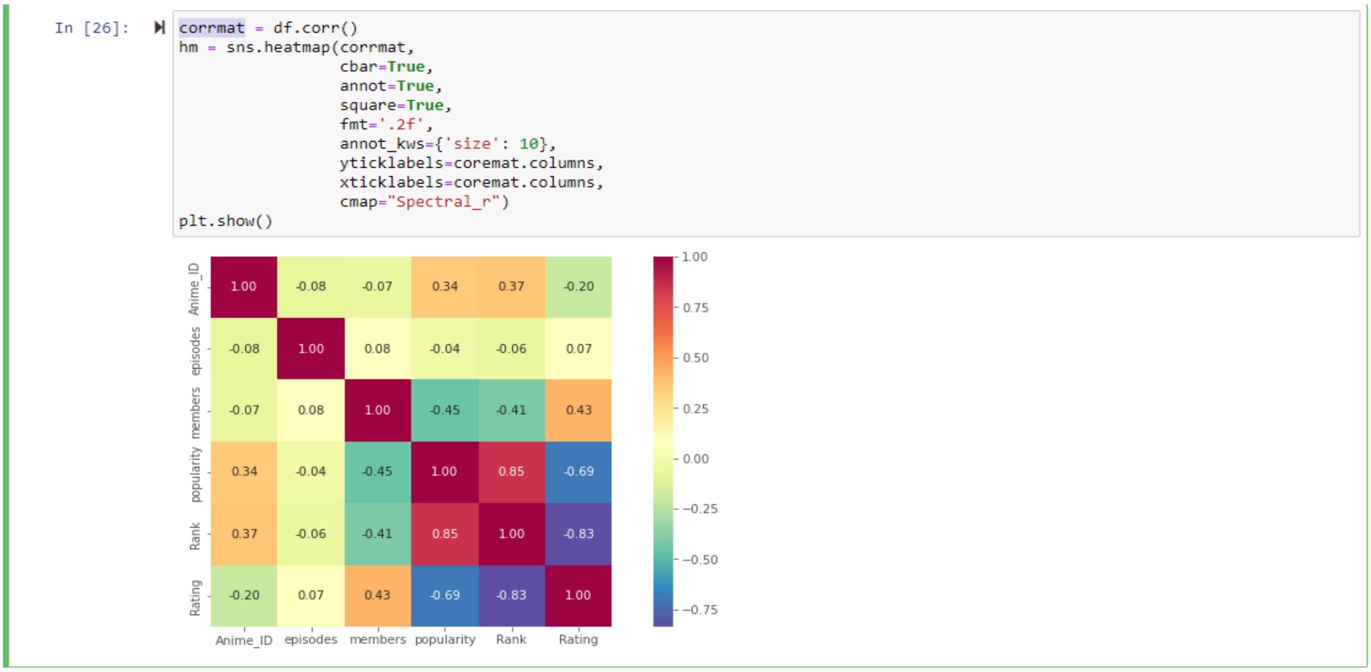
```
In [19]:  coremat=df.corr()
          coremat
```

Out[19]:

	Anime_ID	episodes	members	popularity	Rank	Rating
Anime_ID	1.000000	-0.072610	-0.065095	0.342304	0.368896	-0.200402
episodes	-0.072610	1.000000	0.073545	-0.034644	-0.056932	0.072284
members	-0.065095	0.073545	1.000000	-0.448320	-0.408786	0.426010
popularity	0.342304	-0.034644	-0.448320	1.000000	0.854241	-0.694138
Rank	0.368896	-0.056932	-0.408786	0.854241	1.000000	-0.834105
Rating	-0.200402	0.072284	0.426010	-0.694138	-0.834105	1.000000

Fig 2.1.3.1 Correlation between columns.

Step 9: Printing the corrmap using command “***heatmap()***”.

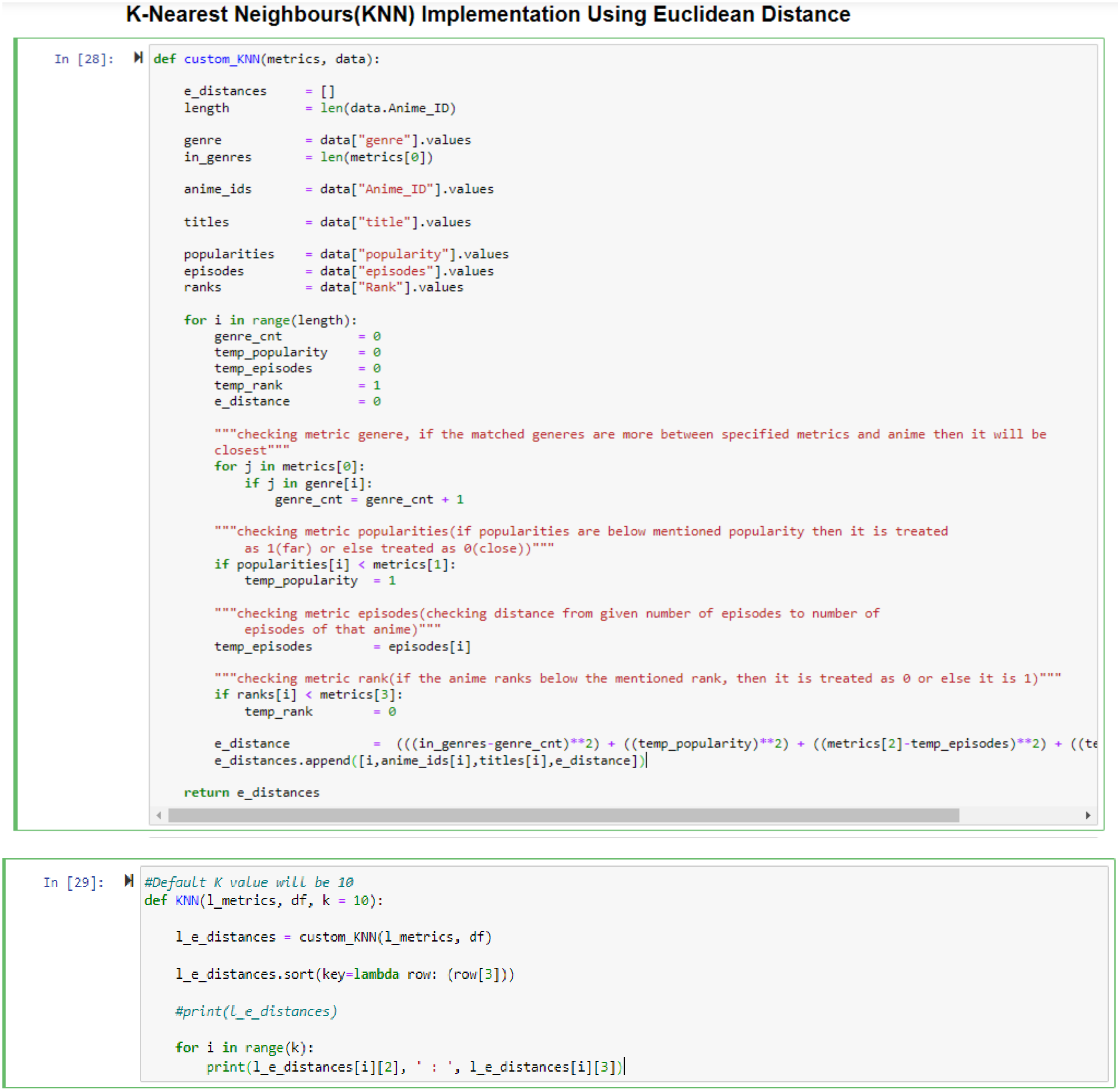


```
In [26]:  corrmat = df.corr()
          hm = sns.heatmap(corrmat,
                           cbar=True,
                           annot=True,
                           square=True,
                           fmt='.2f',
                           annot_kws={'size': 10},
                           yticklabels=coremat.columns,
                           xticklabels=coremat.columns,
                           cmap="Spectral_r")
          plt.show()
```

Fig 2.1.3.2 Heatmap of the data set

Step 10: Implementation of KNN. User also specifies how many anime i.e., K value such that k number of anime will be recommended. User inputs the preferences of his/her through list of metrics like genre, popularity, number of episodes, and rank of the anime. Based on these preferences, the recommender algorithm does the following things,

- Calculates how many genres of that anime matched with the user genre preferences and calculates the distance from that anime to unlabeled record.
- Considers the popularity of that anime as 0 if the given popularity is more than respective anime popularity otherwise it is considered as 1.
- Considers the rank of that anime as 1 if the given rank is more than respective anime’s rank otherwise it considers are 0.
- Episodes will be considered as it is.
- Then, it calculates the distance between all the anime’s and to that unlabeled record.
  l_euclidian_distance = ((given preferences count – matched preferences count) **2) **0.5 + ((given_popularity – (1or 0))**2)**0.5 + ((given_rank– (1or 0))**2)**0.5 + ((given_episodes – anime’s episodes)**2)**0.5



K-Nearest Neighbours(KNN) Implementation Using Euclidean Distance

```
In [28]:  def custom_KNN(metrics, data):

              e_distances     = []
              length          = len(data.Anime_ID)

              genre           = data["genre"].values
              in_genres       = len(metrics[0])

              anime_ids       = data["Anime_ID"].values

              titles          = data["title"].values

              popularities    = data["popularity"].values
              episodes        = data["episodes"].values
              ranks           = data["Rank"].values

              for i in range(length):
                  genre_cnt          = 0
                  temp_popularity    = 0
                  temp_episodes      = 0
                  temp_rank          = 1
                  e_distance         = 0

                  """checking metric genere, if the matched generes are more between specified metrics and anime then it will be
                  closest"""
                  for j in metrics[0]:
                      if j in genre[i]:
                          genre_cnt = genre_cnt + 1

                  """checking metric popularities(if popularities are below mentioned popularity then it is treated
                      as 1(far) or else treated as 0(close))"""
                  if popularities[i] < metrics[1]:
                      temp_popularity  = 1

                  """checking metric episodes(checking distance from given number of episodes to number of
                      episodes of that anime)"""
                  temp_episodes         = episodes[i]

                  """checking metric rank(if the anime ranks below the mentioned rank, then it is treated as 0 or else it is 1)"""
                  if ranks[i] < metrics[3]:
                      temp_rank         = 0

                  e_distance            =  (((in_genres-genre_cnt)**2) + ((temp_popularity)**2) + ((metrics[2]-temp_episodes)**2) + ((te
                  e_distances.append([i,anime_ids[i],titles[i],e_distance])

              return e_distances
```

```
In [29]:  #Default K value will be 10
          def KNN(l_metrics, df, k = 10):

              l_e_distances = custom_KNN(l_metrics, df)

              l_e_distances.sort(key=lambda row: (row[3]))

              #print(l_e_distances)

              for i in range(k):
                  print(l_e_distances[i][2], ' : ', l_e_distances[i][3])
```

Fig: 2.1.3.3 KNN Implementation.

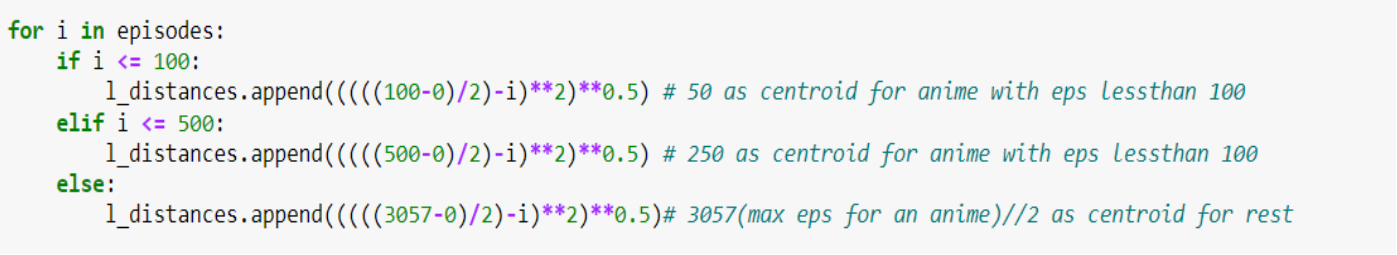
2.2 Categorial Recommender:

2.2.1 Clustering Objective:

Our aim for clustering is to build a categorial recommender. The user should get a recommendation of best or top anime in a certain category. Based on the popularity and rating, can we group the anime into best, worst and avg? Clustering would help us in building a recommender which can answer questions like ‘what are the best / top anime with less than 500 episodes?’. Below, we’ll be trying to build a recommender using k means clustering algorithm.

2.2.1 Implementing K means clustering:

Step 11: Selecting the number of clusters/centroids ‘k’: Firstly, how many clusters do we need? We wanted to cluster the anime based on episodes like, beginner with number of episodes less than or equal to 100, pro anime with number of episodes less than equal to 500 and all the other anime with episodes greater than 500 as advanced. Please make a note that the maximum number of episodes for an anime in our data set is 3057. As we need 3 clusters, we're selecting 3 centroids (k = 3). 50 for beginner anime, 250 for pro anime and 3057/2 for remaining anime.

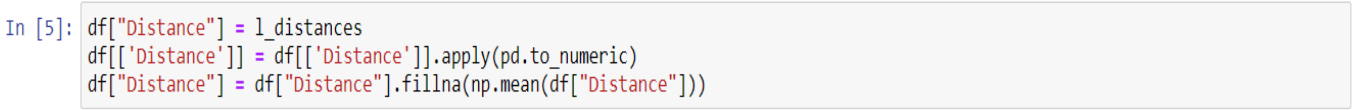


```
for i in episodes:
    if i <= 100:
        l_distances.append(((((100-0)/2)-i)**2)**0.5) # 50 as centroid for anime with eps lessthan 100
    elif i <= 500:
        l_distances.append(((((500-0)/2)-i)**2)**0.5) # 250 as centroid for anime with eps lessthan 100
    else:
        l_distances.append(((((3057-0)/2)-i)**2)**0.5)# 3057(max eps for an anime)//2 as centroid for rest
```

Fig: 2.2.1.1 Euclidean distance calculation for each anime episodes.

Step 12: Euclidean distance: Using the above centroids, we calculated Euclidean distance with every anime episode counts as shown above in the fig 2.2.1.1.

Step 13: As there were few ongoing anime, the episode counts for that anime were not populated, resulting in the null fields in distances. Thus, we replaced the null distances with the mean of the distances.

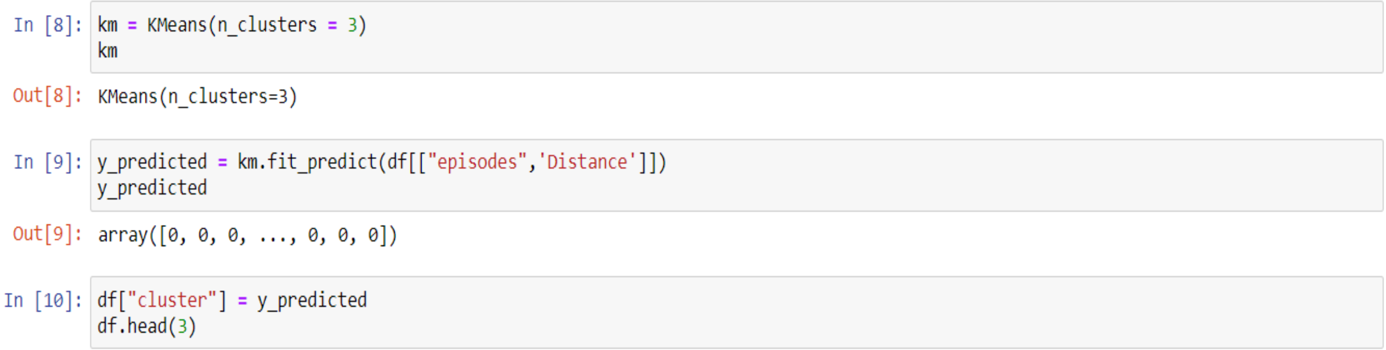


```
In [5]: df["Distance"] = l_distances
        df[['Distance']] = df[['Distance']].apply(pd.to_numeric)
        df["Distance"] = df["Distance"].fillna(np.mean(df["Distance"]))
```

Fig: 2.2.1.2 Replacing nulls with mean of distances

Step 14: Clustering based on above distance: Using the distances we calculated above, we clustered the anime with shortest distance to centroid i.e., Anime episodes count closest to the centroids (50 or 250 or 3057/2).

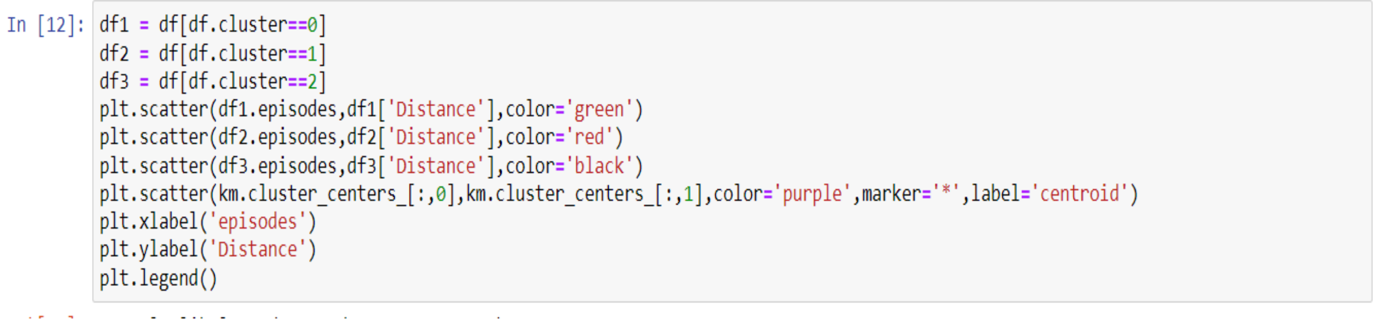
Step 15: Applying the K means clustering: To the above distances we got along with the episode's column, we applied K means clustering.



```
In [8]: km = KMeans(n_clusters = 3)
        km

Out[8]: KMeans(n_clusters=3)

In [9]: y_predicted = km.fit_predict(df[["episodes",'Distance']])
        y_predicted

Out[9]: array([0, 0, 0, ..., 0, 0, 0])

In [10]: df["cluster"] = y_predicted
         df.head(3)
```

Fig: 2.2.1.3 Applying K means clustering

Step 15: Evaluating the clustering: We then plotted the distances we got with episode counts and centroids using a scatter plot.



```
In [12]: df1 = df[df.cluster==0]
         df2 = df[df.cluster==1]
         df3 = df[df.cluster==2]
         plt.scatter(df1.episodes,df1['Distance'],color='green')
         plt.scatter(df2.episodes,df2['Distance'],color='red')
         plt.scatter(df3.episodes,df3['Distance'],color='black')
         plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
         plt.xlabel('episodes')
         plt.ylabel('Distance')
         plt.legend()
```

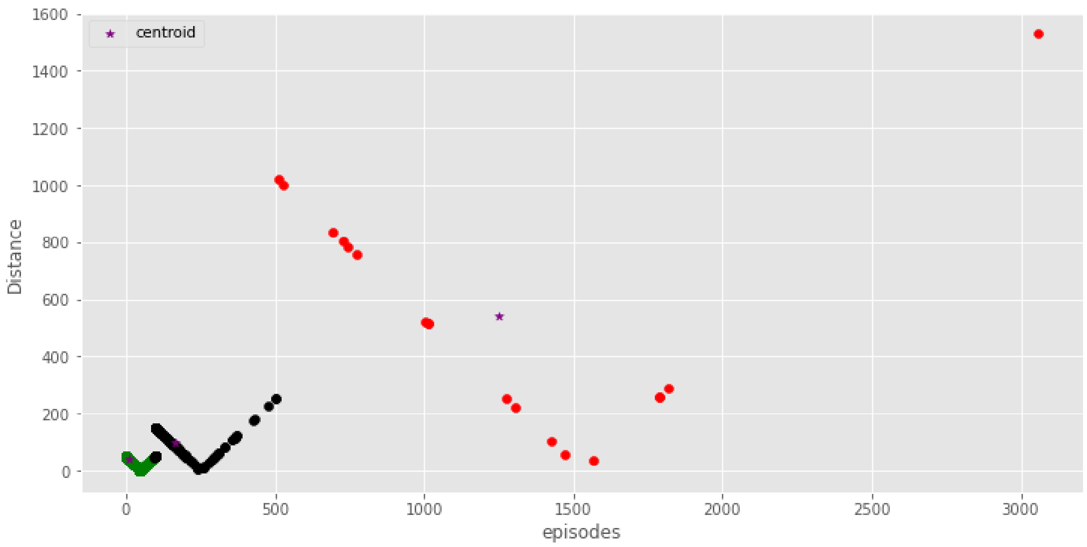
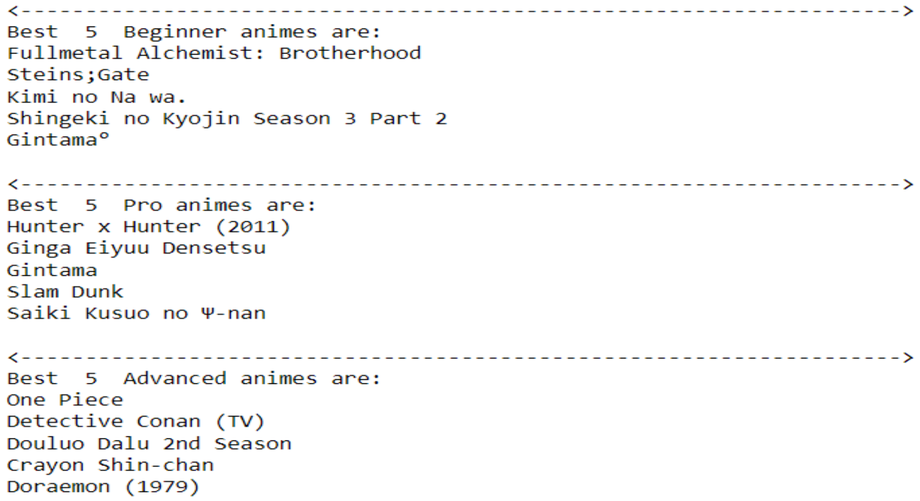


Fig: 2.2.1.4 Plotting the clustering

Now checking the recommendation.

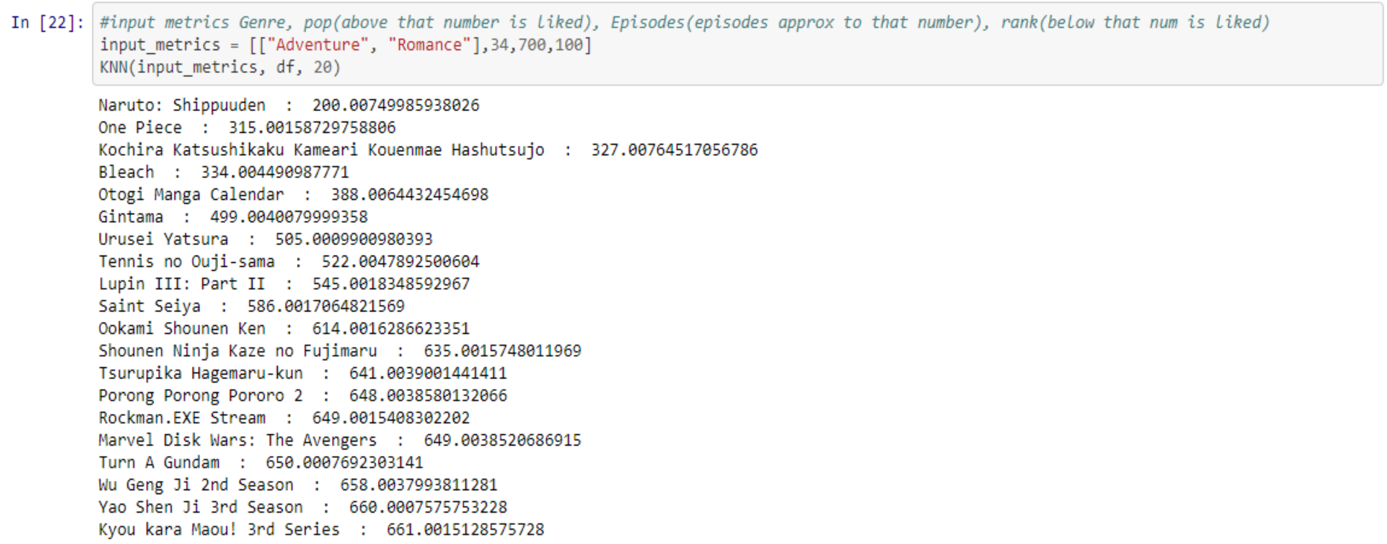


```
<---------------------------------------------------------------------->
Best  5  Beginner animes are:
Fullmetal Alchemist: Brotherhood
Steins;Gate
Kimi no Na wa.
Shingeki no Kyojin Season 3 Part 2
Gintama°

<---------------------------------------------------------------------->
Best  5  Pro animes are:
Hunter x Hunter (2011)
Ginga Eiyuu Densetsu
Gintama
Slam Dunk
Saiki Kusuo no Ψ-nan

<---------------------------------------------------------------------->
Best  5  Advanced animes are:
One Piece
Detective Conan (TV)
Douluo Dalu 2nd Season
Crayon Shin-chan
Doraemon (1979)
```

Fig: 2.2.1.5 Results of recommender

RESULTS

3.1 Discussion of classification results:



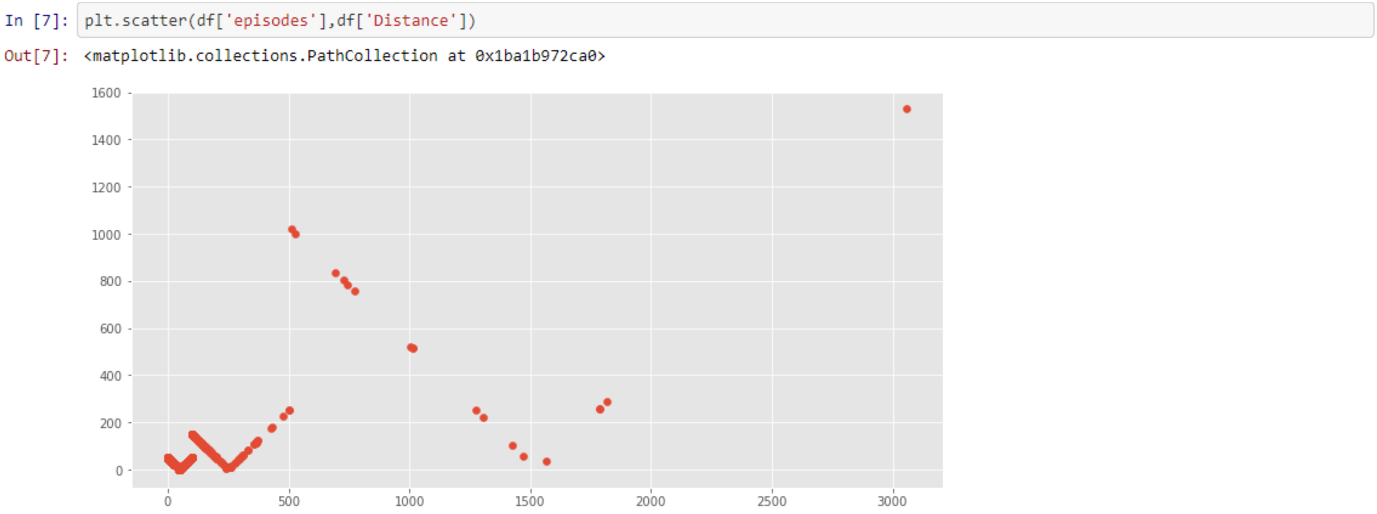
```
In [22]: #input metrics Genre, pop(above that number is liked), Episodes(episodes approx to that number), rank(below that num is liked)
         input_metrics = [["Adventure", "Romance"],34,700,100]
         KNN(input_metrics, df, 20)

         Naruto: Shippuuden  :  200.00749985938026
         One Piece  :  315.00158729758806
         Kochira Katsushikaku Kameari Kouenmae Hashutsujo  :  327.00764517056786
         Bleach  :  334.004490987771
         Otogi Manga Calendar  :  388.0064432454698
         Gintama  :  499.0040079999358
         Urusei Yatsura  :  505.0009900980393
         Tennis no Ouji-sama  :  522.0047892500604
         Lupin III: Part II  :  545.0018348592967
         Saint Seiya  :  586.0017064821569
         Ookami Shounen Ken  :  614.0016286623351
         Shounen Ninja Kaze no Fujimaru  :  635.0015748011969
         Tsurupika Hagemaru-kun  :  641.0039001441411
         Porong Porong Pororo 2  :  648.0038580132066
         Rockman.EXE Stream  :  649.0015408302202
         Marvel Disk Wars: The Avengers  :  649.0038520686915
         Turn A Gundam  :  650.0007692303141
         Wu Geng Ji 2nd Season  :  658.0037993811281
         Yao Shen Ji 3rd Season  :  660.0007575753228
         Kyou kara Maou! 3rd Series  :  661.0015128575728
```

Fig: 3.1.1 Result of our Recommender

- Here based on the input (genre, popularity, episode count, rank) given by the user, we'll get the recommendations.
- Along with this, user can get the top 'k' anime as per his given input metrics.

3.2 Discussion of Clustering results:

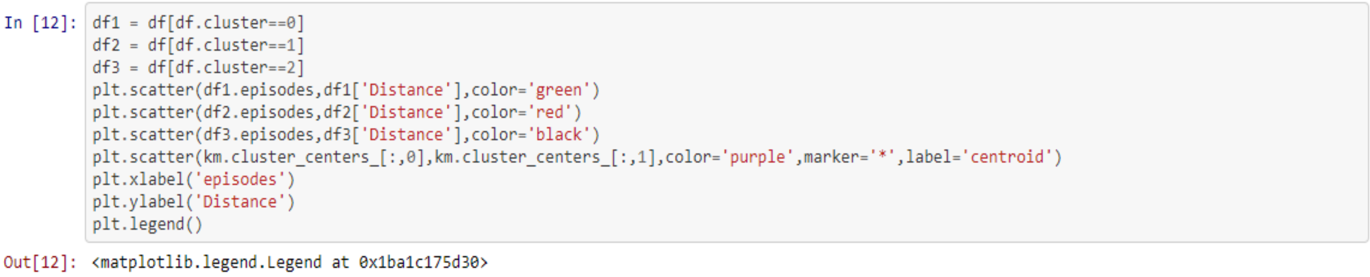
- After applying K means clustering to our data, we used the scatter plot to plot and evaluate the clusters we got. Below is the basic scatter plot of the episodes vs distance we got



```
In [7]: plt.scatter(df['episodes'],df['Distance'])
Out[7]: <matplotlib.collections.PathCollection at 0x1ba1b972ca0>
```

Fig: 3.2.1 Basic scatterplot of episodes vs distance

- Thus, in the above plot we can see the clusters clearly. But let's visualize these clusters with their centroids and with different colors (each cluster with a different color) to check the distribution of anime episodes in each cluster.



```
In [12]: df1 = df[df.cluster==0]
         df2 = df[df.cluster==1]
         df3 = df[df.cluster==2]
         plt.scatter(df1.episodes,df1['Distance'],color='green')
         plt.scatter(df2.episodes,df2['Distance'],color='red')
         plt.scatter(df3.episodes,df3['Distance'],color='black')
         plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
         plt.xlabel('episodes')
         plt.ylabel('Distance')
         plt.legend()

Out[12]: <matplotlib.legend.Legend at 0x1ba1c175d30>
```

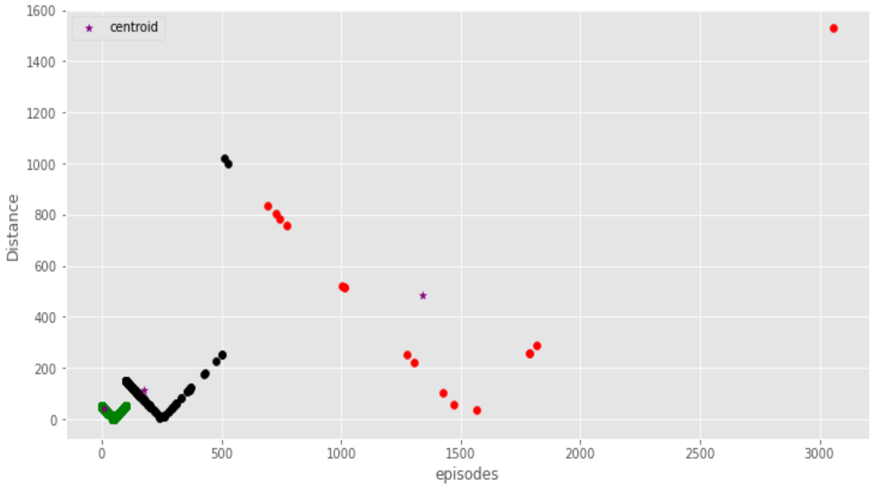
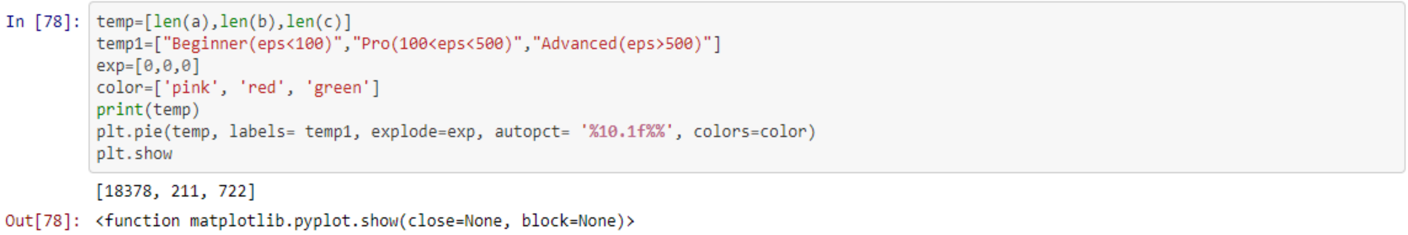


Fig: 3.2.2 scatterplot of episodes vs distance along with centroids

- In the above plot, the purple markers ('*') indicates the centroids and green color scatter shows the distribution of beginner anime (<100 episodes), black shows the pro anime (<500 episodes) and red for advanced anime (> 500 episodes).
- The scatter distribution in the above plot clearly indicates that there are very few advanced anime and the scatter in black and green is denser showing that the anime in beginner and pro clusters are more.
- Let's also verify our above observation through a pie plot.



```
In [78]: temp=[len(a),len(b),len(c)]
         temp1=["Beginner(eps<100)","Pro(100<eps<500)","Advanced(eps>500)"]
         exp=[0,0,0]
         color=['pink', 'red', 'green']
         print(temp)
         plt.pie(temp, labels= temp1, explode=exp, autopct= '%10.1f%%', colors=color)
         plt.show

         [18378, 211, 722]

Out[78]: <function matplotlib.pyplot.show(close=None, block=None)>
```

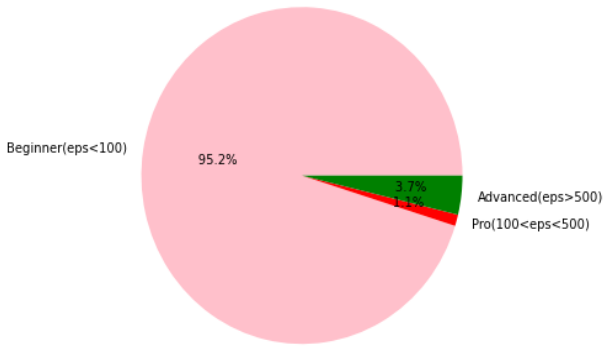
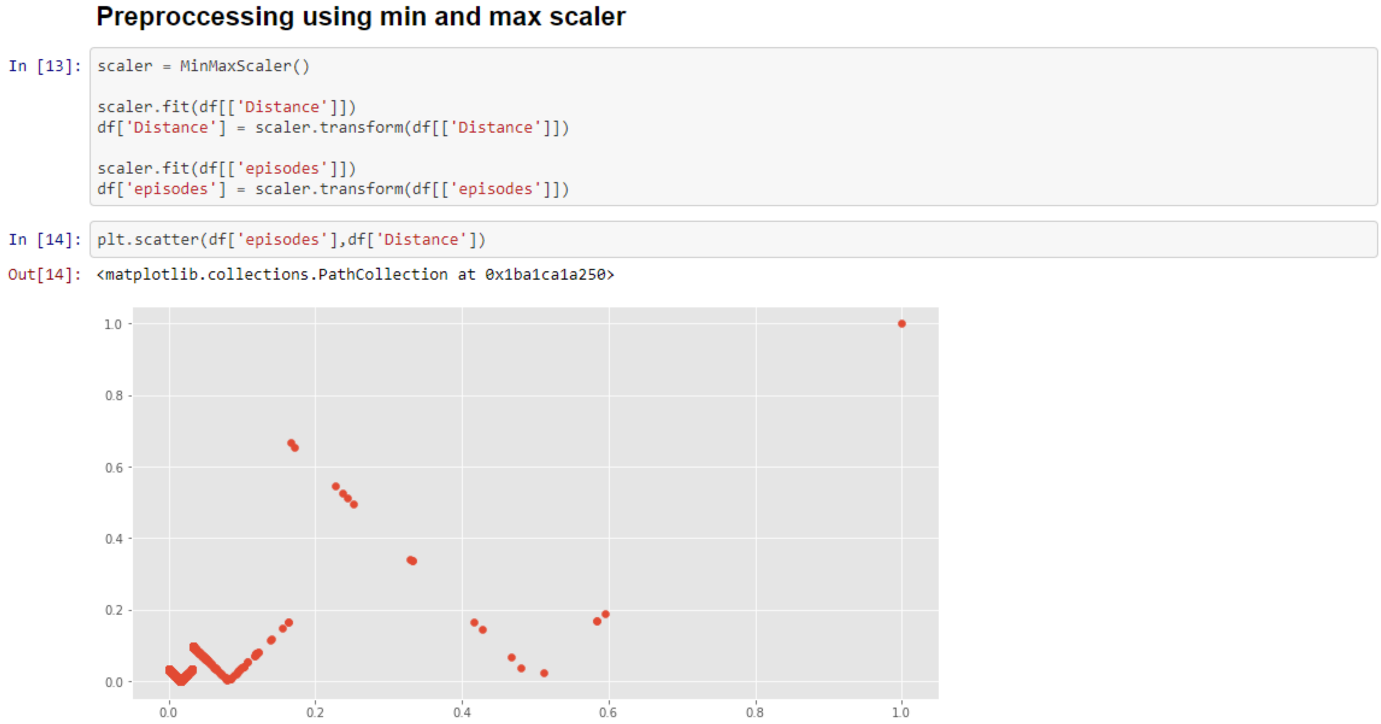


Fig: 3.2.3 pie plot of clusters

- The pie also shows similar results. Most of the anime are in the beginner group (<100 episodes). Rest of the anime which are in the advanced and pro range are very minimal.

- Let’s also scale our scatter plot using min-max scaler

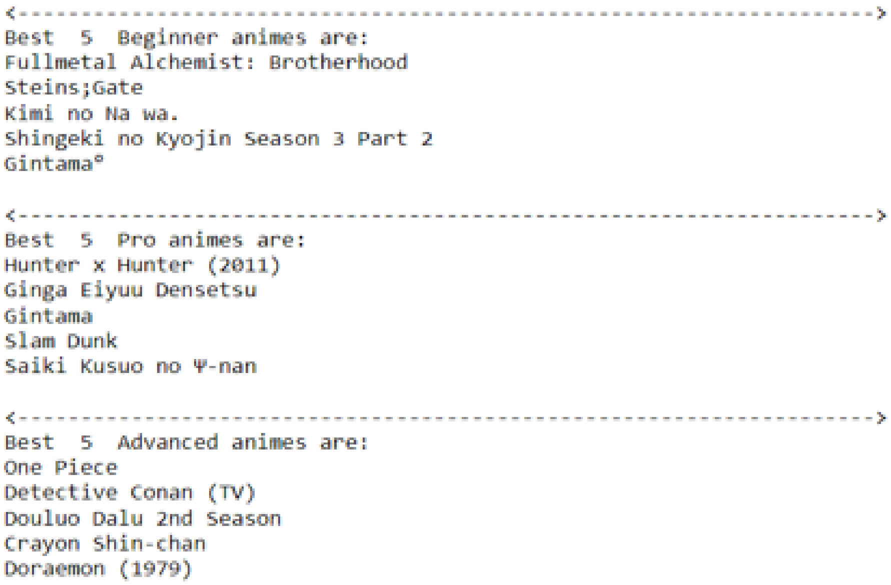


Preproccessing using min and max scaler

```
In [13]: scaler = MinMaxScaler()

         scaler.fit(df[['Distance']])
         df['Distance'] = scaler.transform(df[['Distance']])

         scaler.fit(df[['episodes']])
         df['episodes'] = scaler.transform(df[['episodes']])

In [14]: plt.scatter(df['episodes'],df['Distance'])

Out[14]: <matplotlib.collections.PathCollection at 0x1ba1ca1a250>
```

Fig: 3.2.4 scatterplot after transforming through min-max scaler

- Below is the result of clustering with ‘k’ as 5.



```
<---------------------------------------------------------------------->
Best  5  Beginner animes are:
Fullmetal Alchemist: Brotherhood
Steins;Gate
Kimi no Na wa.
Shingeki no Kyojin Season 3 Part 2
Gintama°

<---------------------------------------------------------------------->
Best  5  Pro animes are:
Hunter x Hunter (2011)
Ginga Eiyuu Densetsu
Gintama
Slam Dunk
Saiki Kusuo no Ψ-nan

<---------------------------------------------------------------------->
Best  5  Advanced animes are:
One Piece
Detective Conan (TV)
Douluo Dalu 2nd Season
Crayon Shin-chan
Doraemon (1979)
```

Fig: 3.2.5 result of clustering

FUTURE SCOPE

3.1 Predictor:

Will we be able to predict the anime a given user is going to watch next? Additionally, we can also build a predictor through the same Anime recommendation dataset. In the Anime recommendation dataset, we have a file named 'rating.csv' which contains the data of user rating to each anime. This dataset can be used as a data which shows us the group of anime watched by each user. By using this user anime watch history if we can find the frequently watched anime sets, we will be able to form association rules with it. Thus, after the association rules were set, if we get a user and his previous watched anime, we might be able to predict the anime he is most likely to watch next.

3.1.2 Example: Lets understand this better with the help of an example.

Let's say that we got a user with anime watch history as {Naruto and Bleach}. Now let's consider the below sets as few of the frequent anime sets, we got from the data set

- {Naruto, Attack on Titan, One Piece}
- {Boku No Hero Academia, Demon Slayer, Bleach}
- {Full Metal Alchemist, Bleach, DBZ}
- {Naruto, Bleach and One Piece} with some x confidence and y support.

Thus, from the above we can say that the user is more likely to watch One Piece next as he watched Naruto and Bleach. Like many other users who watched Naruto followed by Bleach and then One piece, our user might also watch One Piece Next.

Therefore, using the prior knowledge of users watch history, if we can build the association rules and form frequent anime watch sets, we can predict the anime a given user might watch next. This feature will be useful for any anime streaming services or platforms.

REFERENCES

1. https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews
2. https://towardsdatascience.com/exploratory-data-analysis-in-python-a-step-by-step-process-d0dfa6bf94ee
3. https://ieeexplore.ieee.org/abstract/document/9091509
4. https://www.sciencedirect.com/science/article/abs/pii/S0167865517303562
5. https://github.com/codebasics/py/blob/master/ML/13_kmeans/13_kmeans_tutorial.ipynb
6. https://youtu.be/EItlUEPCIzM
7. https://users.cs.duke.edu/~cynthia/docs/RudinLeMa13.pdf