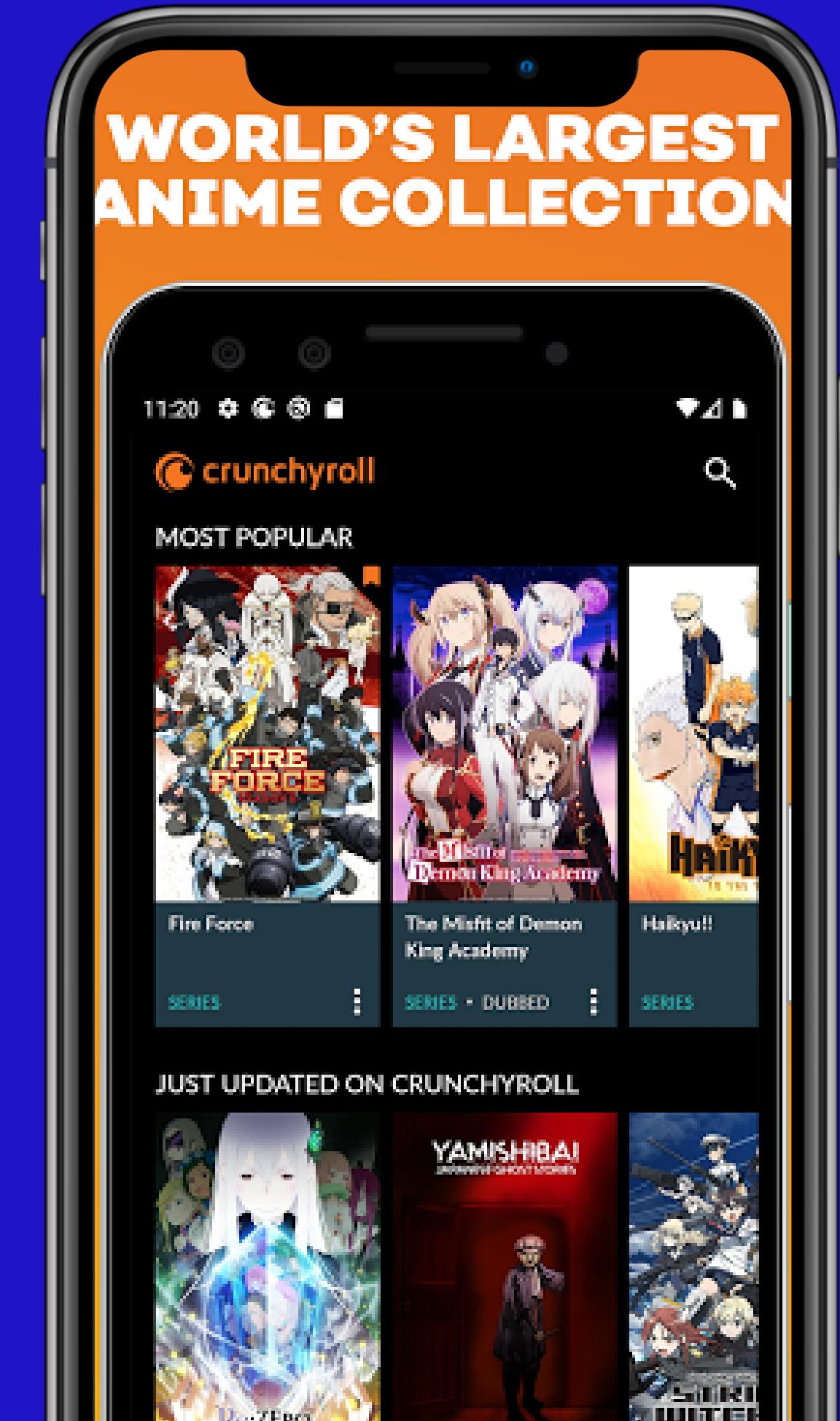


# Sistema de recomendacion de anime

- [Alexander Sanchez Sanchez](#)
- [Juan David Cruz Garcia](#)
- [Juan Sebastian Perez Camacho](#)
- [Kennet Santiago Sanchez Roldan](#)





**Recapitulando...**

**Buscamos crear un sistema de recomendación de anime basado en los animes que un usuario ha calificado previamente. Nuestro interés en esta herramienta nace de:**



- Aplicar los conceptos aprendidos.
- Atender a la oportunidad de negocio que nace del crecimiento de la industria

**El objetivo de esta entrega es entrenar los modelos tanto de clustering como de clasificación.**

- **Debemos permitir que a través de los datos obtenidos, se creen clústeres para los animes**
- **Se deben clasificar a los usuarios en una de estas categorías con base a la calificación dada a animes**





# Carga de datos y depuración

# CARGA DE LOS DATAFRAMES

```
1 #Anime.csv
2 path = Path(os.getcwd())
3 path = str(path.parent.absolute())
4 path = path+"/datos/anime.csv"
5 dfAnime = pd.read_csv(path,na_values='?')
6 dfAnime.info()
```

```
<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
```

```
1 #rating.csv
2 path2 = Path(os.getcwd())
3 path2 = str(path2.parent.absolute())
4 path2 = path2+"/datos/rating.csv"
5 dfRating = pd.read_csv(path2,na_values='?')
6 dfRating.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7813737 entries, 0 to 7813736
Data columns (total 3 columns):
 #   Column      Dtype  
---  --          ---  
 0   user_id    int64  
 1   anime_id    int64  
 2   rating      int64  
dtypes: int64(3)
memory usage: 178.8 MB
```

# AJUSTE DE LOS TIPOS DE DATOS

Definimos tipos para cada columna del dataframe que tenga como tipo "object"

```
1 #anime.csv
2 dfAnime['name'] = dfAnime['name'].astype("string")
3 dfAnime['genre'] = dfAnime['genre'].astype("string")
4 dfAnime['type'] = dfAnime['type'].astype("string")
5 dfAnime['episodes']=pd.to_numeric(dfAnime.episodes, errors='coerce').dropna().astype(int)
6 dfAnime.info()
```

```
<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   string 
 2   genre       12232 non-null   string 
 3   type        12269 non-null   string 
 4   episodes    11954 non-null   float64
 5   rating      12064 non-null   float64
 6   members     12294 non-null   int64  
dtypes: float64(2), int64(2), string(3)
memory usage: 672.5 KB
```

# ELIMINACION DE DATOS NULOS O INNECESARIOS

La cantidad de datos nulos es:

	anime_id	name	genre	type	episodes	rating	members
0	0	0	62	25	340	230	0

La cantidad de datos nulos es

	user_id	anime_id	rating
0	0	0	0

```
1 dfAnime=dfAnime.dropna()
2 dfRating = dfRating[dfRating.rating != -1]
```

# ELIMINACION DE DATOS DUPLICADOS

```
1 # Anime.csv  
2 duplicados = dfAnime[dfAnime.duplicated()].shape[0]  
3 print("Numero de datos duplicados ",duplicados)
```

Numero de datos duplicados 0

```
1 # Rating.csv  
2 duplicados = dfRating[dfRating.duplicated()].shape[0]  
3 print("Numero de datos duplicados ",duplicados)
```

Numero de datos duplicados 1

```
1 # Rating.csv  
2 dfRating.drop_duplicates(keep='first',inplace=True)  
3 duplicados = dfRating[dfRating.duplicated()].shape[0]  
4 print("Numero de datos duplicados ",duplicados)
```

Numero de datos duplicados 0



Ajustes necesarios

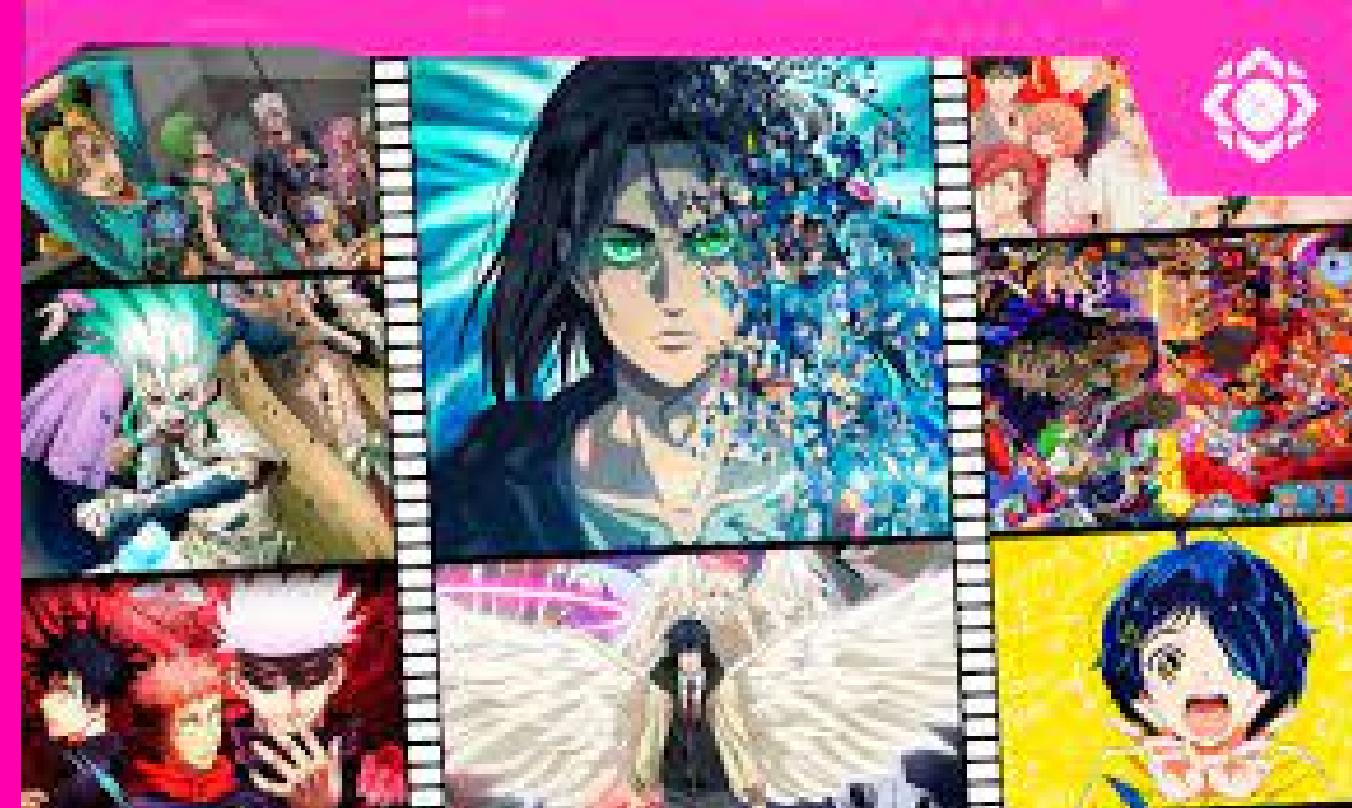


**Ajustes anime.csv**

# AJUSTES A LOS TIPOS DE DATOS

```
1 dfAnime['episodes'] = dfAnime['episodes'].astype(int)
2 dfAnime=dfAnime.replace({'Movie': '0', 'TV': '1','OVA':'2','ONA':'2','Special':'3','Music':'4'})
3 dfAnime['type'] = dfAnime['type'].astype("category")
```

- Convertimos "episodes" en una variable de numeros enteros para poder realizar adecuadamente el conteo
- Convertimos "type" en una variable de tipo categoria para poder realizar analisis sobre este



Toda pelicula tendra solamente un episodio mientras que las series pueden llegar incluso a 100 episodios

Se necesita entonces particionar el dataframe, uno para series y otro para películas.

## SURGE UN PROBLEMA

El dataframe cuenta con todo tipo de animes, desde series a películas, esto genera que la cantidad de episodios se vea condicionada.



# SE TOMA UNA DECISION

Teniendo en cuenta nuestro contexto y para evitarnos entrenar mas modelos de lo necesario, solo usaremos los animes que sean del tipo "series"

```
1 df_series = dfAnime.loc[dfAnime['type'] == '1']
2 df_series = df_series.drop('type', axis=1)
```

# SEPARACION DE CATEGORIAS

**El dataframe tiene todos los generos de un anime en una sola columna, para que esta informacion sea util, debemos clasificarla**

**La estrategia aplicada es convertir la informacion contenida en esta columna en columnas correspondientes a los generos.**

**De esta manera, una fila tendra 1 o 0 en una columna de genero dependiendo de si el anime pertenece o no a esta.**

# ¿QUE GENEROS ESCOGER?

**Los generos que contiene el dataframe son demasiados, es conveniente no usarlos todos.**

```
1 genreCount = df_series[["genre"]]
2 genreCount["genre"] = genreCount["genre"].str.split(", | , | ,")
3 genreCount = genreCount.explode("genre")
4 genreCount["genre"] = genreCount["genre"].str.title()
5
6 print(f'Total unique genres are {len(genreCount["genre"].unique())}')
7 print(f'Occurrences of unique genres :')
8 genreCount["genre"].value_counts().to_frame().T.style.set_properties(**{"background-color": "#2a9d8f", "color": "white", "border": "1px solid black", "font-size": "10pt"}).reset_index()
9
```

Total unique genres are 41  
Occurrences of unique genres :

	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Slice Of Life	School	Kids	Supernatural	Mecha	Magic	Shoujo	Historical	Ecchi
genre	1769	1052	897	752	720	686	676	637	534	532	438	409	372	333	277	267	237

# CREAR LAS COLUMNAS

Se crean las columnas y se elimina la columna 'genre' pues ya esta representada por las columnas recien creadas

```
1 def createNewColumn(colname):
2     df_series[colname] = np.where(df_series.genre.str.contains(colname),1,0)
3     df_series[colname] = df_series[colname].astype(int)

1 def containsGenre(dfInfo,word):
2     if str(df_series["genre"]).find(word):
3         return 1
4     return 0

1 #Ciclo que se encarga de crear 15 columnas de los generos mas populares
2 for i in range(15):
3     createNewColumn(genreCount["genre"].value_counts().index.tolist()[i])
4
```

# DATAFRAME RESULTANTE

Ajustes rating.csv

# VER SI UN ANIME LE GUSTO O NO A UN USUARIO

**Estrategia:** Extraer el promedio de sus calificaciones.

**Si la calificacion dada es mayor o igual a la media, significa que el usuario disfruto del anime, en caso contrario, se considerara una opinion negativa**

Promedio
7,00
7,25
8,50
9,25
7,63

# DATAFRAME RESULTANTE

```
1 dfRating['ratingMean']=dfRating["rating"].mean()  
2 dfRating['Liked']=np.where(dfRating.rating >= dfRating.ratingMean ,1,0)  
3 dfRating.head()
```

	<u>user_id</u>	<u>anime_id</u>	rating	ratingMean	Liked
47	1	8074	10	7.808497	1
81	1	11617	10	7.808497	1
83	1	11757	10	7.808497	1
101	1	15451	10	7.808497	1
153	2	11771	10	7.808497	1

# Búsqueda y eliminación de outliers

# ESTRATEGIA

Usaremos el iqr (rango intercuartil) para solo dejar los datos que se encuentren en este.

First Quartile	Median Second Quartile	Third Quartile	
First Quarter	Second Quarter	Third Quarter	Fourth Quarter
24, 25, 26,	27, 30, 32,	40, 44, 50,	52, 55, 57
$Q_1$	$Q_2$	$Q_3$	
26½	36	51	

MathBits.com

# ELIMINACION DE OUTLIERS

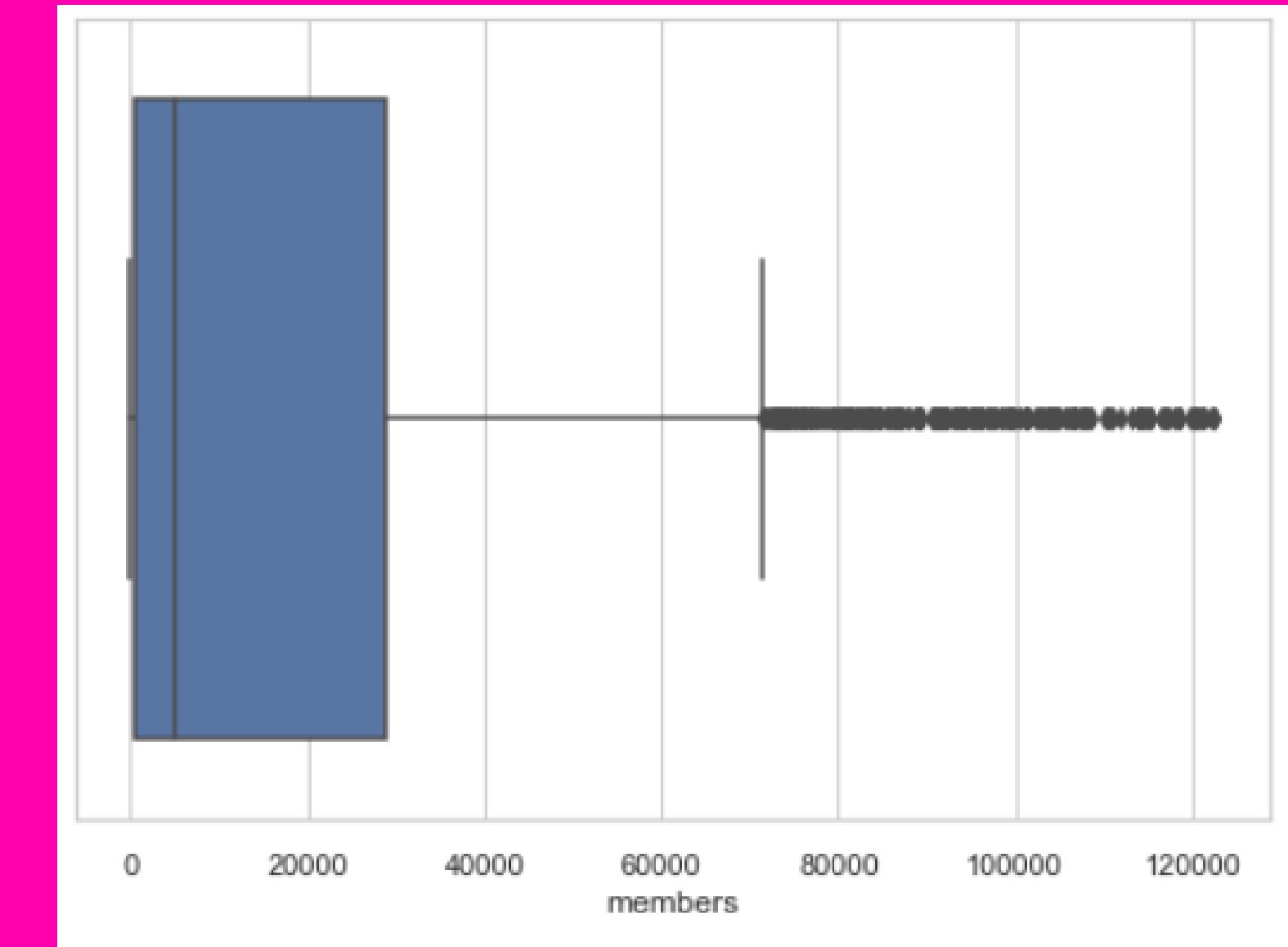
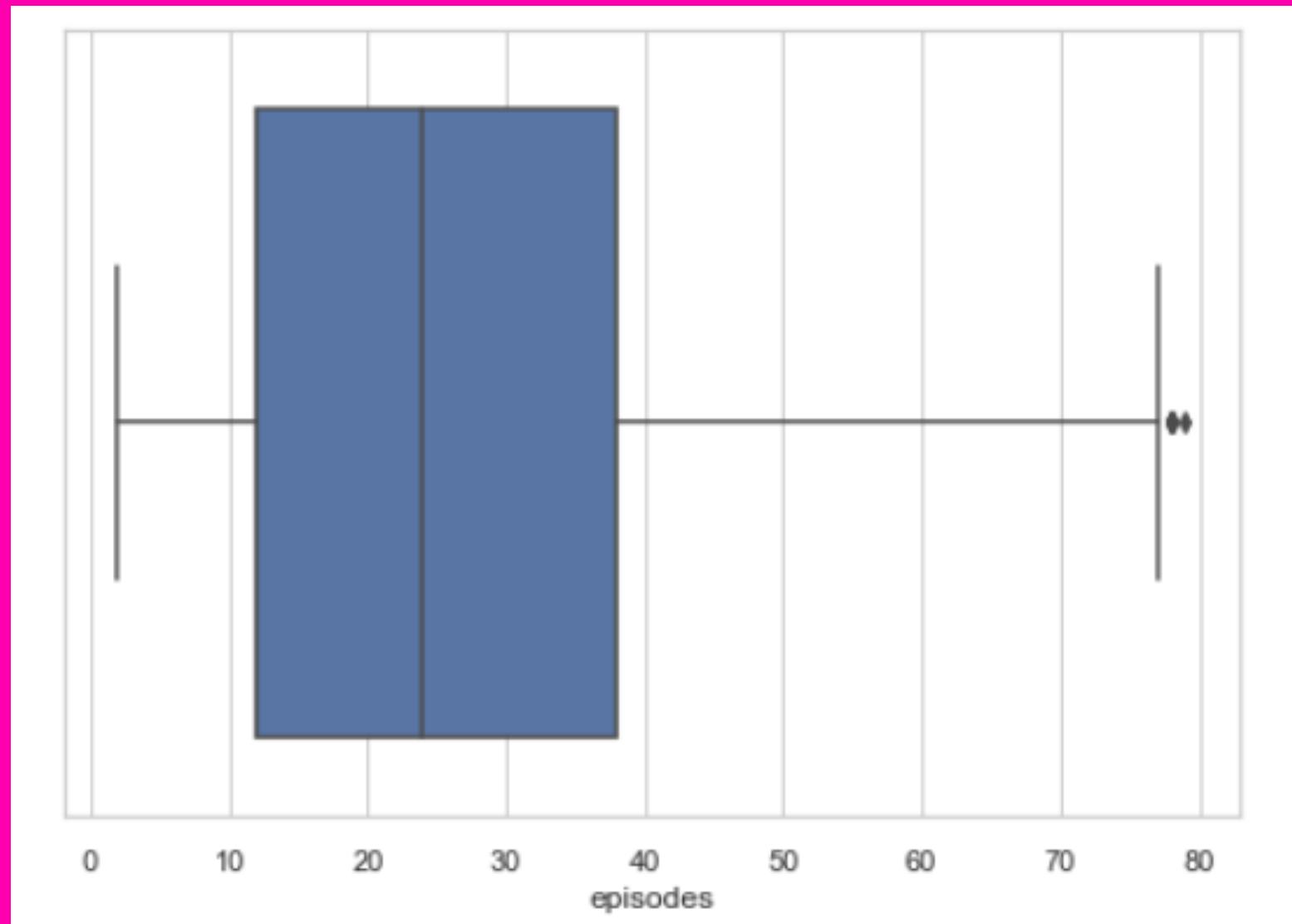
Teniendo en cuenta la naturaleza de las variables, solo es logico eliminar outliers para las variables 'members' y 'episodes'

```
1 q_low = df_series["episodes"].quantile(0.25)
2 q_hi = df_series["episodes"].quantile(0.75)
3 iqr = q_hi - q_low
4
5 lower = q_low - (1.5*iqr)
6 high = q_hi + (1.5*iqr)
7
8 df_series = df_series[(df_series["episodes"] < high) & (df_series["episodes"] > lower)]
```

```
1 q_low = df_series["members"].quantile(0.25)
2 q_hi = df_series["members"].quantile(0.75)
3 iqr = q_hi - q_low
4
5 lower = q_low - (1.5*iqr)
6 high = q_hi + (1.5*iqr)
7
8 df_series = df_series[(df_series["members"] < high) & (df_series["members"] > lower)]
```

# ELIMINACION DE OUTLIERS

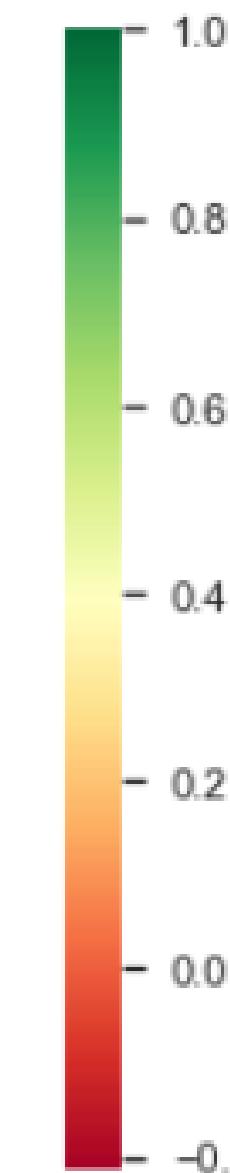
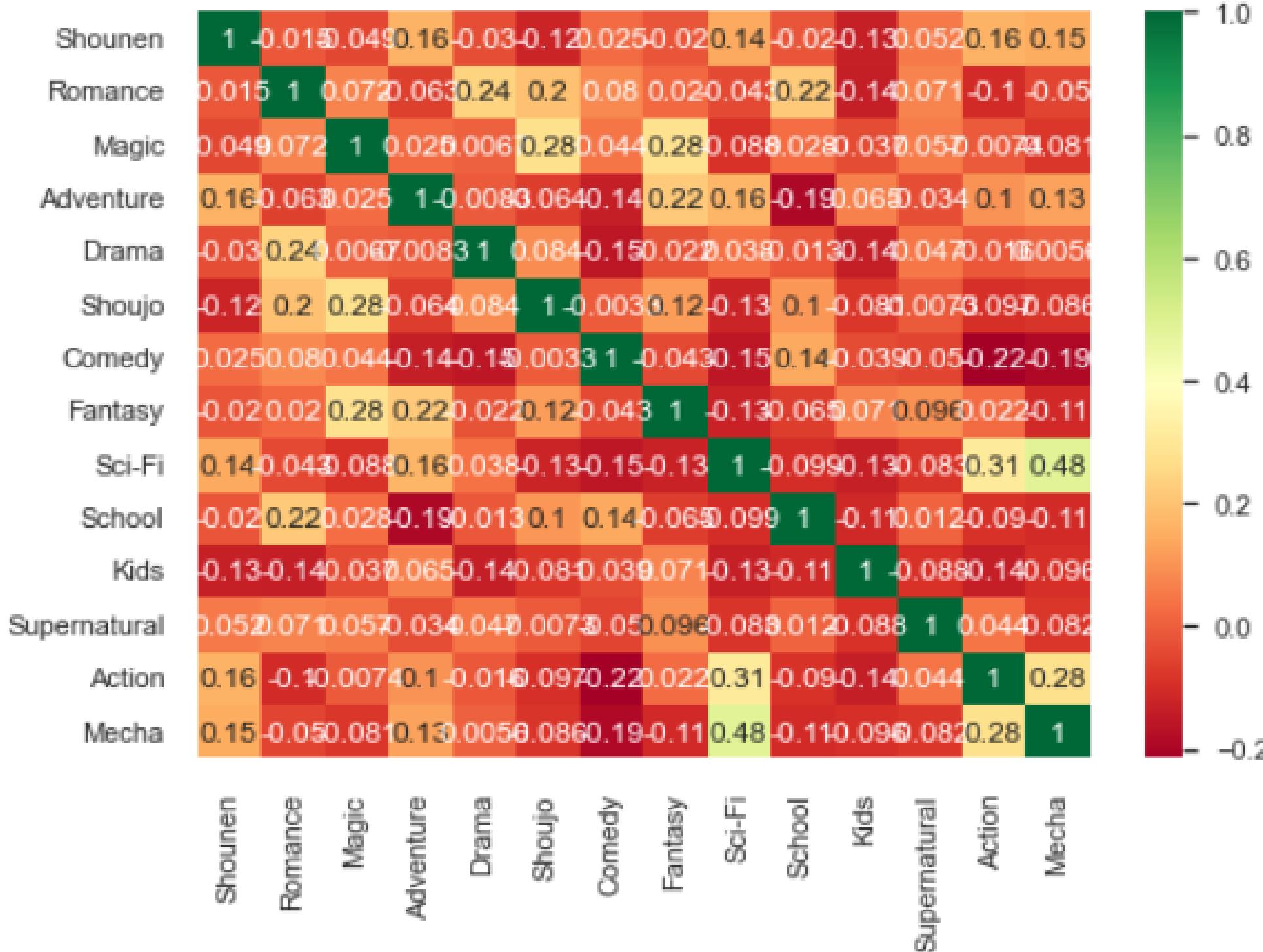
Si bien segun el diagrama de cajas y bigotes, quedan algunos cuartiles, consideramos apropiado no realizar de nuevo la estrategia pues consideramos que lo obtenido es suficiente para trabajar.



# Construcción del modelo de clustering

# ANALISIS DE LAS VARIABLES

Es necesario revisar las correlaciones entre los géneros pues son estos las variables más relevantes



- **Shoujo y Magic: 0.28**
- **Shoujo y fantasy: 0.28**
- **Mecha y Sci-fi: 0.48**
- **School y Romance: 0.22**

# ELIMINACION DE SHOUJO



**Shoujo no es realmente un genero como tal sino que engloba a todos los generos que mayormente se relacionan con mujeres jóvenes.**



**Siendo así, este "genero" puede ser representado por otras variables como magic y fantasy.**

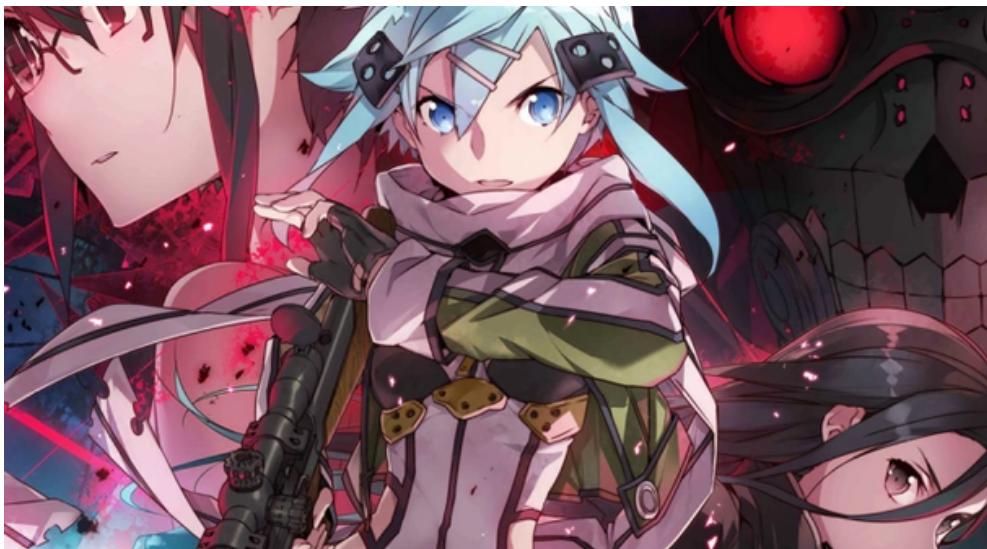


**Y se debe tener en cuenta que generos como Fantasy pueden incluso no tener a las mujeres como objetivo principal**

# ELIMINACION DE MECHA



**Sin importar que el anime sea de un tono humoristico o por el contrario sea de un tono oscuro. Cualquier anime de tipo mecha se identifica por la presencia de robots gigantes o humanoides.**



**Sin embargo, el género sci-fi aborda en general a los conceptos imaginativos y futuristas como la ciencia y la tecnología avanzadas**

# ELIMINACION DE SCHOOL



School es un género bastante influyente, nos indica que al anime tendrá en casi todo momento, lugar en una escuela. Sin embargo, el género en el dataframe no cuenta con la suficiente cantidad de datos para considerarlo una buena variable.



Por el lado positivo, muchos animes del tipo romance son tambien de tipo school, asi que de esta manera podemos cubrir esta falta de informacion.

# NORMALIZACION

Despues de escoger las variables, queremos que todos los datos esten normalizados.

De la siguiente manera todos los datos estaran entre 1 y 0

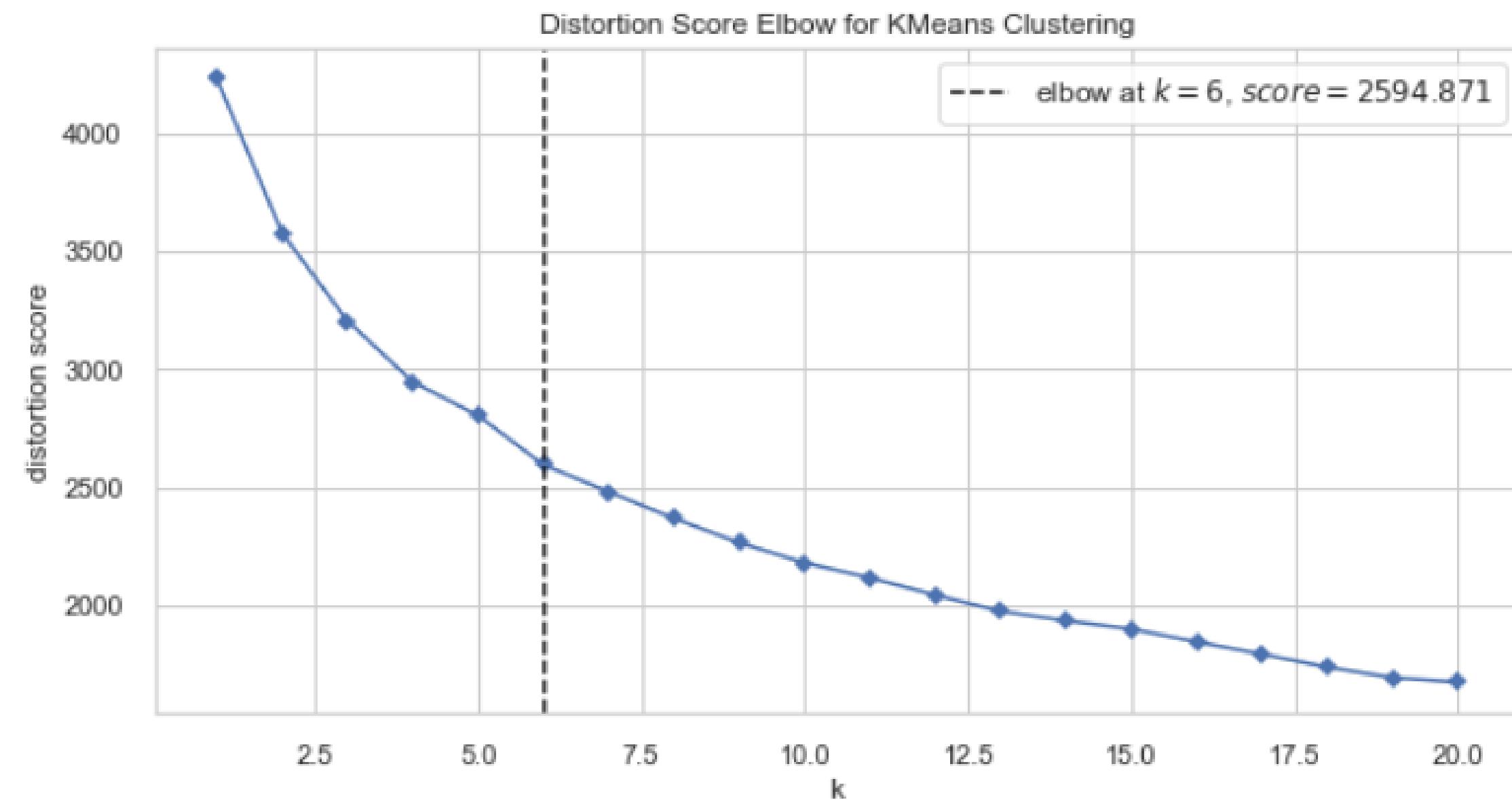
```
1 features = list(set(df_series.columns) - set(["anime_id","name","episodes"]))
2
3 rango_de_salida_de_las_variables_escaladas = (0,1) #Tupla con el siguiente formato: (mínimo deseado, máximo deseado).
4 scaler = MinMaxScaler(feature_range=rango_de_salida_de_las_variables_escaladas) #Instanciamos el objeto para escalar los datos
5
6 df_series_norm = deepcopy(df_series[features]) #Inicializamos este objeto con una copia profunda de las columnas de entrada
7 df_series_norm[features] = scaler.fit_transform(df_series_norm) #Ajustamos y transformamos los datos.
8
```

# HIPERPARAMETROS DEL MODELO

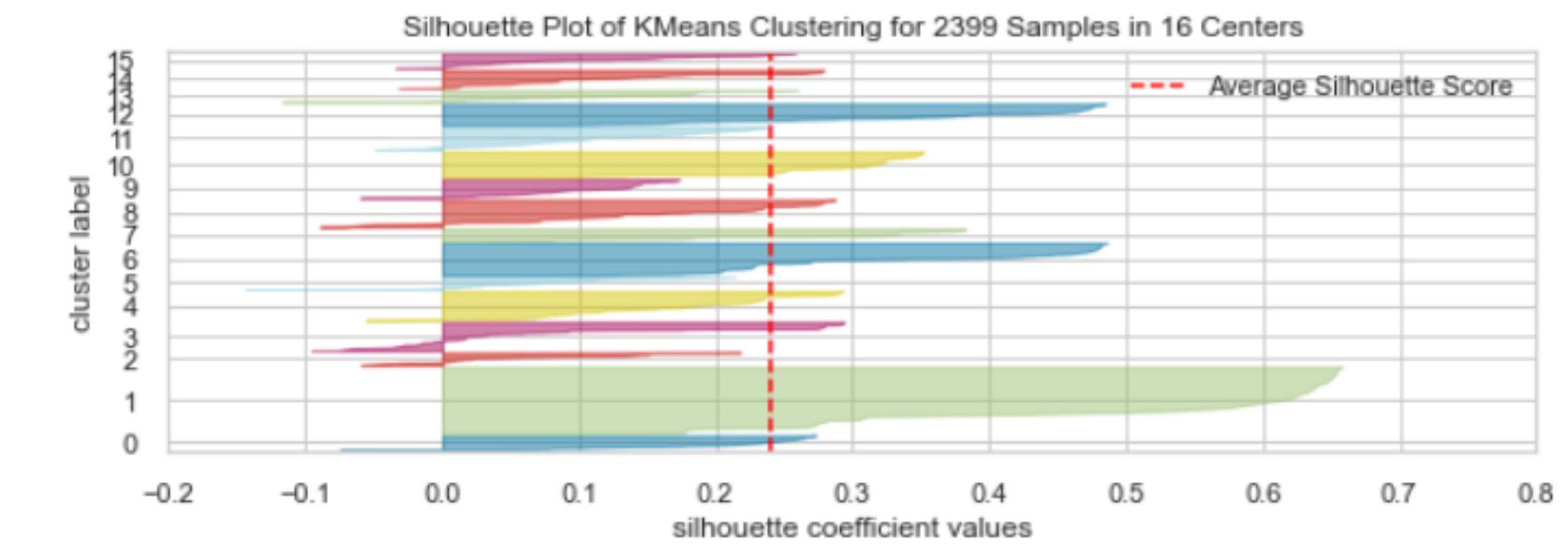
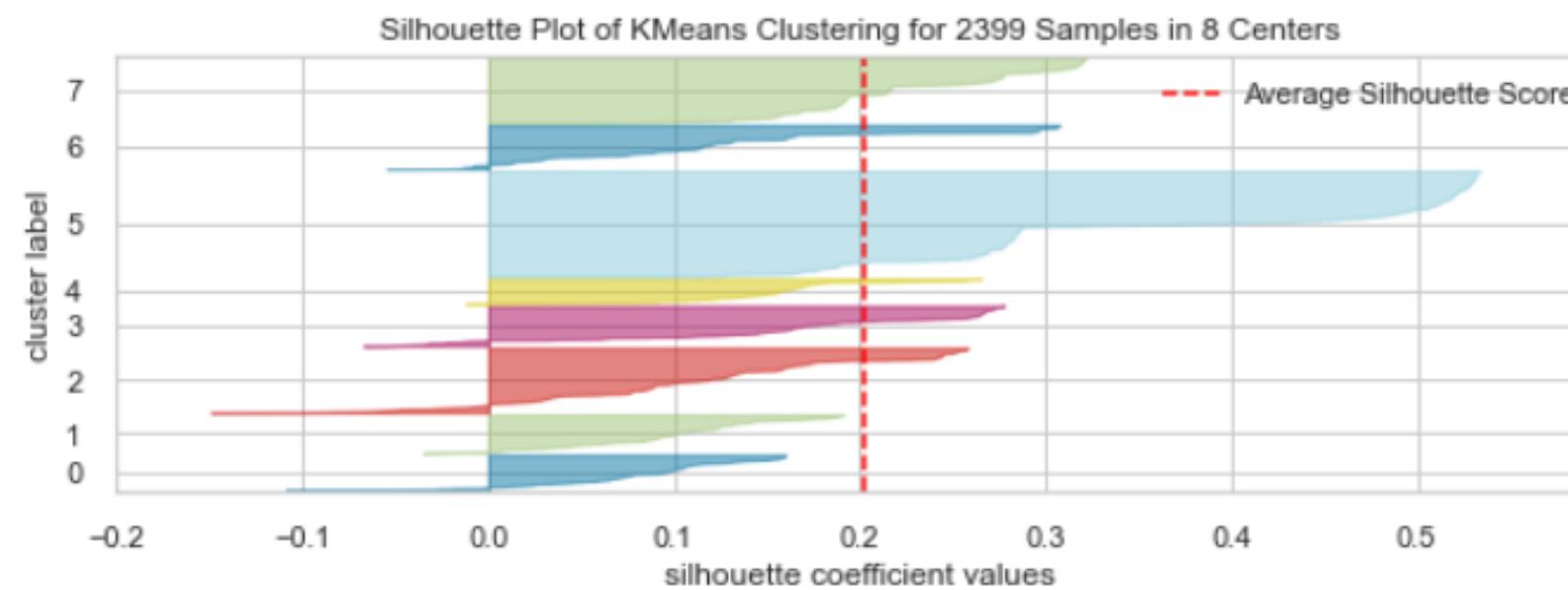
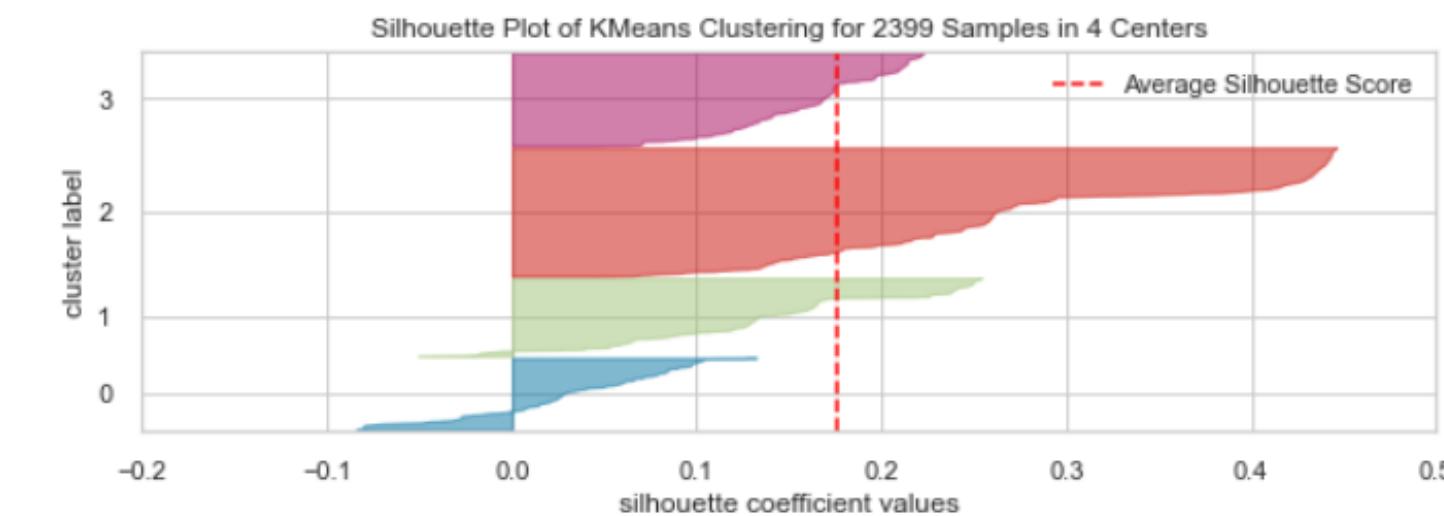
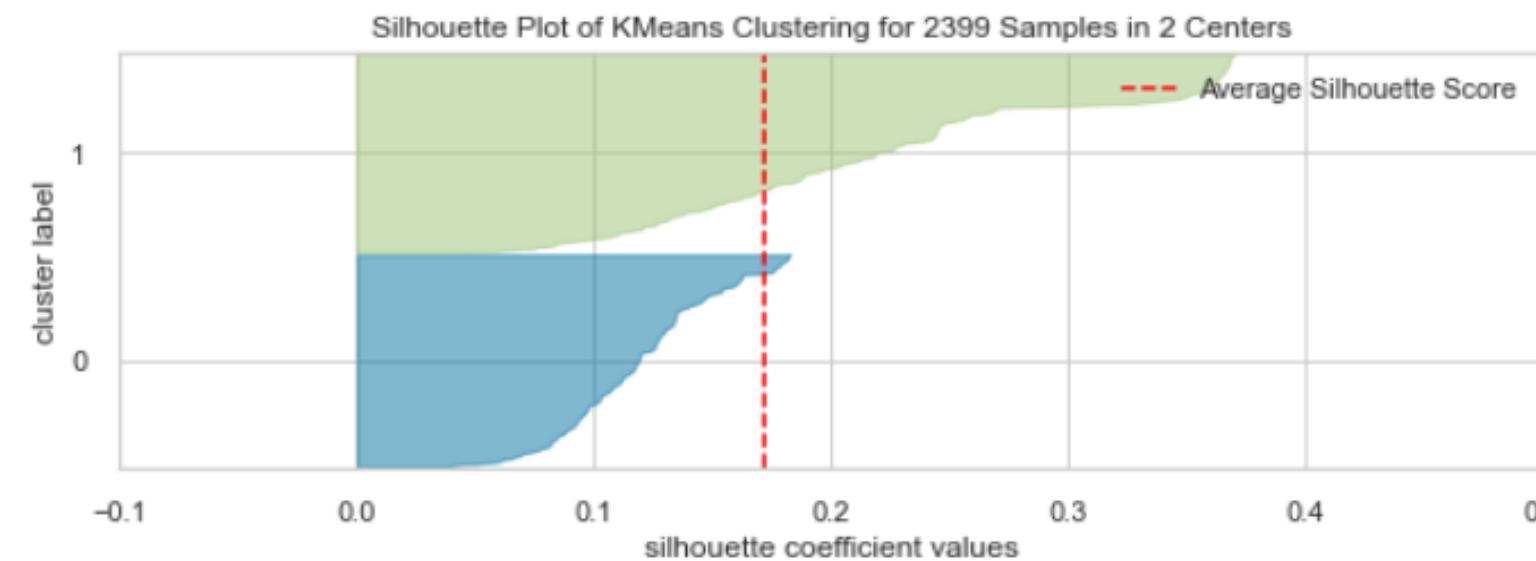
**Usamos kmeans++ pues es una versión mejorada del kmeans estandar que inicializa los centroides de manera inteligente.**

```
: 1 #-----  
2 #-----  
3 # HYPERPARÁMETROS DEL MODELO  
4 #-----  
5 #-----  
6  
7 kmin          = 1      #Límite inferior para explorar el número de grupos.  
8 kmax          = 20     #Límite superior para explorar el número de grupos.  
9 init          ='k-means++' #Se define el método de inicialización. Otra opción válida es 'random'.  
10 n_init        = 10    #Número de inicializaciones aleatorias. Al final scikit learn escoge aquel con la menor iner-  
11                 #(i.e.: suma de cuadrados de distancias de cada punto a su centroide respectivo dentro de cierto rango)  
12                 #https://scikit-learn.org/stable/modules/clustering.html  
13 max_iter      = 300   #Número MÁXIMO de iteraciones para una sola ejecución.  
14 random_seed   = 76    #Semilla aleatoria. Permite obtener los mismos resultados en cada ejecución.  
15
```

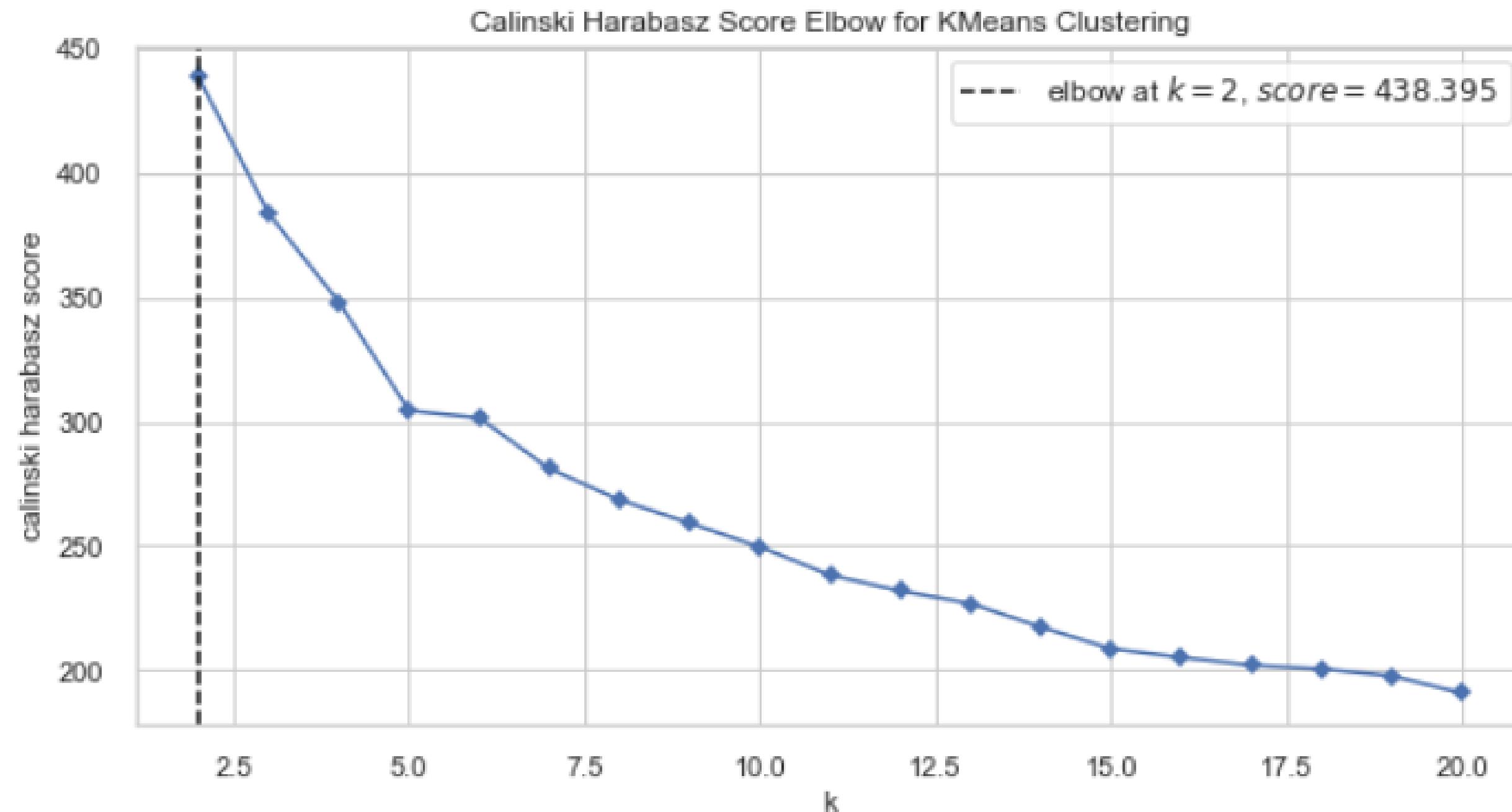
# METODO DEL CODO



# OBSERVACION DE LAS SILUETAS



# CALINSKI HARABASZ



# DECISION FINAL PARA EL K

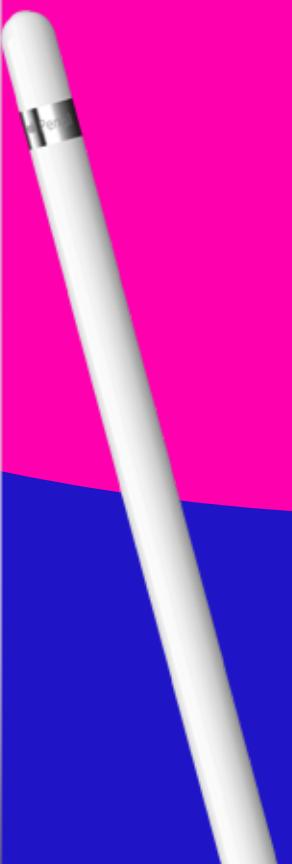
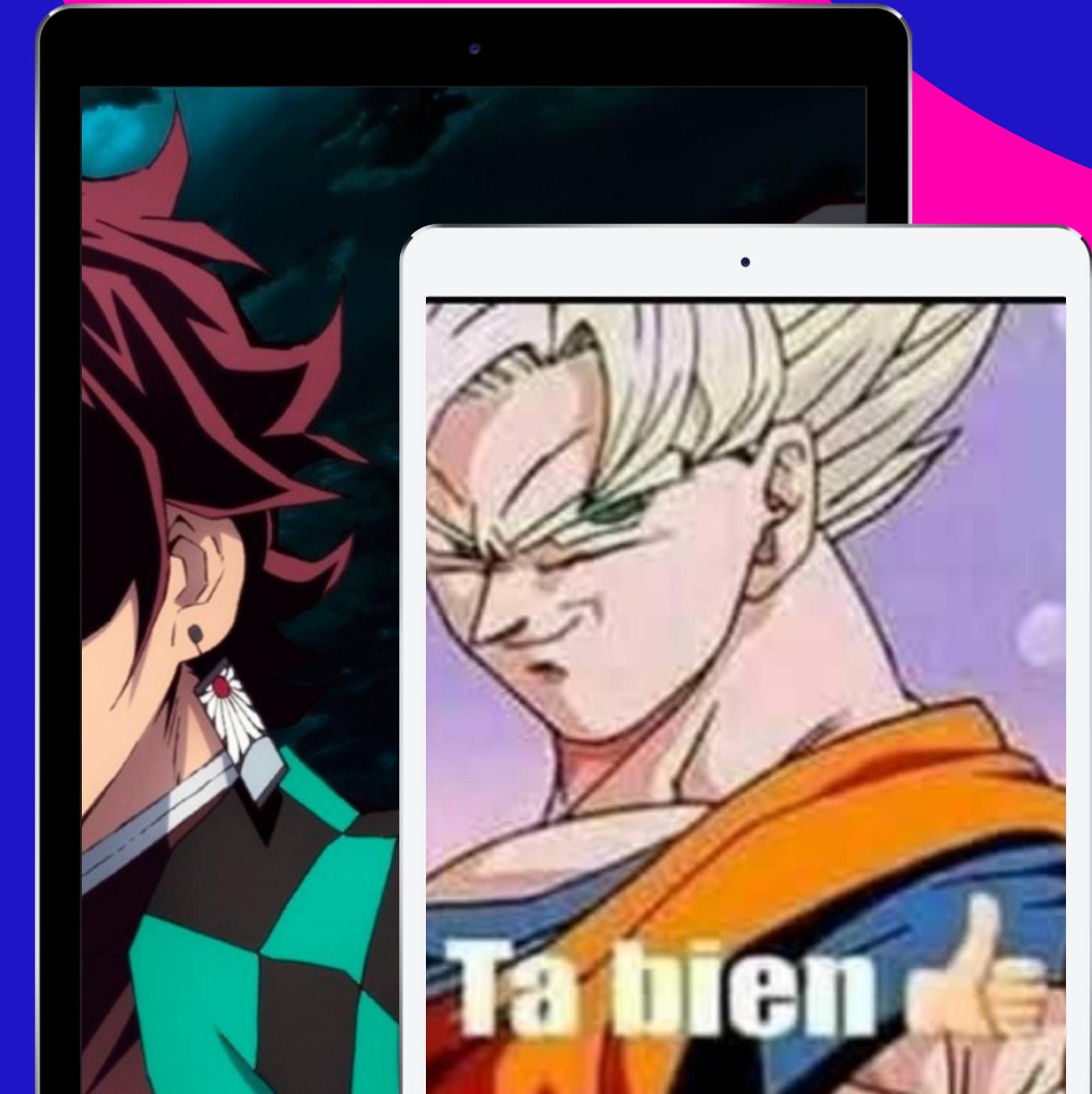
Dado a que los indicadores no llegan a un consenso, es conveniente usar como K el numero de generos que definimos previamente

```
1 #K-means
2 #-----
3 k = 12 #Número de grupos que se escogió después del análisis previo.
4
5 #Ahora se instancia el objeto para utilizar el agrupamiento con k-means.
6 #Para ver todas las opciones del constructor, consulte: https://scikit-learn.org/stable/modules/generated/skLearn.cluster.KMeans.html
7 #Nota: el algoritmo de k-means disponible en scikit-learn funciona únicamente con la distancia euclidiana.
8 #Si requiere aplicar k-means con otras métricas de distancia, puede consultar la librería PyClustering: https://github.com/tirthajyoti/PyClustering
9 kmeans = KMeans(n_clusters = k,           #Se define el número de grupos.
10                  init = 'k-means++', #Se define el método de inicialización. Otra opción es 'random'
11                  n_init = 10,          #Número de inicializaciones aleatorias.
12                  max_iter = 300,       #Número MÁXIMO de iteraciones para una sola ejecución.
13                  random_state = 42)
14
15 #Hagamos el ajuste (i.e.: encontraremos los centroides).
16 kmeans.fit(df_x_train)
17 predict = kmeans.predict(df_series_norm)
18
19 df_series['Classification'] = pd.Series(predict, index=df_series_norm.index)
```

# DATAFRAME PARA LA CLASIFICACION

	anime_id	name	episodes	rating_x	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	...	Drama	Romance	Kids	Supernatural	Magic	Classification	user_id	rating_y	ratingMean	Liked
0	28977	Gintama°	51	9.25	114262	1	1	0	1	0	...	0	0	0	0	0	9	43	10	7.808497	1
1	28977	Gintama°	51	9.25	114262	1	1	0	1	0	...	0	0	0	0	0	9	46	10	7.808497	1
2	28977	Gintama°	51	9.25	114262	1	1	0	1	0	...	0	0	0	0	0	9	385	10	7.808497	1
3	28977	Gintama°	51	9.25	114262	1	1	0	1	0	...	0	0	0	0	0	9	392	8	7.808497	1
4	28977	Gintama°	51	9.25	114262	1	1	0	1	0	...	0	0	0	0	0	9	400	9	7.808497	1

# Fusionar dos animes.



Eliminamos los valores 0 en liked del dataframe.

## Probando el filtro de recomendacion

```
##ELiminamos Los valores Liked=0 ya que no son necesarios en Las recomendaciones  
df_depured=df_final[df_final.Liked != 0]
```

Usamos dos animes específicos para ser fusionados.

```
df_depured=df_fusion[df_final.name == "Uchuu Majin Daikengou"]  
df_depured
```

rating_x	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Kids	Supernatural	Magic
6.6	229	0	1	1	1	0	1	0	0	0	0	0

```
df_depured2=df_fusion[df_final.name == "Zenmai Zamurai"]  
df_depured2
```

rating_x	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Kids	Supernatural	Magic
6.44	152	1	0	0	0	0	0	0	0	1	0	0

# FUSIONAMOS LOS ANIMES

```
##Usamos dos animes de prueba  
verify("Uchuu Majin Daikengou", "Zenmai Zamurai")
```

1581854	Uchuu Majin Daikengou	Zenmai Zamurai				name	rating_x	members	Comedy	Action	Adventure	Sci-Fi	Fantasy
1581854	1	1	1	0	1	Shounen	6.52	190.5	1	0	0	0	Romance
1581854	0	0	0	0	0	Drama	0	0	1	0	0	0	Kids
1581854	0	0	0	0	0	Supernatural	0	0	0	0	0	0	Magic

# CONSTRUCCIÓN DEL MODELO

- MAGIC
- SHOUNEN
- COMEDY
- FANTASY
- KIDS
- MEMBERS
- ADVENTURE
- SCI-FI
- ROMANCE
- DRAMA
- ACTION
- SUPERNATURAL

División del conjunto de datos

```
1 input_attr = features
2 target_attr = 'Classification'
3
4 input_attr.remove('rating')
5
6 print(input_attr)
7
8 df_x_knn = deepcopy(df_final[input_attr])
9 df_y_knn = deepcopy(df_final[target_attr])
10
```

Modelo

```
1 k = 12
2
3 knn = neighbors.KNeighborsClassifier(n_neighbors=k)
4 knn.fit(df_x_train_knn, df_y_train_knn)
```

## Precisión del modelo

### Precisión en el conjunto de prueba

```
1 acc_test = metrics.accuracy_score(df_y_test_knn,y_predicted_test_knn)
2 print('Accuracy on test data: %.4f'% acc_test)
✓ 0.1s
```

Accuracy on test data: 0.9992

### Precisión en el conjunto de entrenamiento y validación

```
1 y_predicted_train_knn = knn.predict(df_x_train_knn)
2 y_predicted_val_knn = knn.predict(df_x_val_knn)
3 acc_train = metrics.accuracy_score(df_y_train_knn,y_predicted_train_knn)
4 print('Accuracy on training data: %.4f'% acc_train)
5 acc_val = metrics.accuracy_score(df_y_val_knn,y_predicted_val_knn)
6 print('Accuracy on validation data: %.4f'% acc_val)
```

✓ 3m 45.4s

Accuracy on training data: 0.9994

Accuracy on validation data: 0.9992

# PRECISIÓN DEL MODELO

Usando el protocolo de evaluación train/val/test

Ejemplos

## Ejemplos de predicciones

```
1 predicted_value = knn.predict(df_depured[input_attr])
2
3 print('El valor de clasificación para el dataframe depurado 1 es ',predicted_value)
✓ 0.1s
```

El valor de clasificación para el dataframe depurado 1 es [8]

```
1 predicted_value = knn.predict(df_depured2[input_attr])
2
3 print('El valor de clasificación para el dataframe depurado 2 es ',predicted_value)
✓ 0.7s
```

El valor de clasificación para el dataframe depurado 2 es [7]

# ORDENAMIENTO

De acuerdo al rating y al número de members por categoría

## Los 5 animes más famosos

Por rating

```
1 rating_df_series_by_7 = df_series_by_7.sort_values(by='rating', ascending=False)
2 rating_df_series_by_7.head(5)
```

[114] ✓ 0.1s

Python

...	anime_id	name	episodes	rating	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Kids	Supernatural	Magic	Classification
10718	22219	Warera Salaryman Tou	26	8.67	57	1	0	0	0	0	0	0	0	0	0	0	7
62	32995	Yuri!!! on Ice	12	8.61	103178	1	0	0	0	0	0	0	0	0	0	0	7
108	7655	Major S6	25	8.49	24788	1	0	0	0	0	0	0	1	0	0	0	7
134	31181	Owarimonogatari	12	8.43	107855	1	0	0	0	0	0	0	0	0	1	0	7
197	3091	xxHOLiC Kei	13	8.34	74941	1	0	0	0	0	0	1	0	0	1	0	7

Por miembros

```
1 members_df_series_by_7 = df_series_by_7.sort_values(by='members', ascending=False)
2 members_df_series_by_7.head(5)
```

[115] ✓ 0.1s

Python Python

...	anime_id	name	episodes	rating	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Kids	Supernatural	Magic	Classification
1407	26349	Danna ga Nani wo Itteiru ka Wakaranai Ken	13	7.55	122271	1	0	0	0	0	0	0	0	0	0	0	7
1266	2923	Shugo Chara!	51	7.61	121108	1	0	0	0	0	0	0	0	0	0	1	7
553	10521	Working!!	13	7.98	117328	1	0	0	0	0	0	0	0	0	0	0	7
1263	5277	Sekirei: Pure Engagement	13	7.61	113911	1	1	0	0	0	0	0	0	0	0	0	7
300	28025	Tsukimonogatari	4	8.21	111009	1	0	0	0	0	0	0	0	0	1	0	7

## Por rating y miembros

```
1 members_rating_df_series_by_7 = df_series_by_7.sort_values(by=['rating','members'], ascending=False)
2 members_rating_df_series_by_7.head(5)
```

{116} ✓ 0.1s Python

...	anime_id	name	episodes	rating	members	Comedy	Action	Adventure	Sci-Fi	Fantasy	Shounen	Drama	Romance	Kids	Supernatural	Magic	Classification
	10718	22219 Warera Salaryman Tou	26	8.67	57	1	0	0	0	0	0	0	0	0	0	0	7
	62	32995 Yuri!!! on Ice	12	8.61	103178	1	0	0	0	0	0	0	0	0	0	0	7
	108	7655 Major S6	25	8.49	24788	1	0	0	0	0	0	0	1	0	0	0	7
	134	31181 Owarimonogatari	12	8.43	107855	1	0	0	0	0	0	0	0	0	1	0	7
	197	3091 xxxHOLiC Kei	13	8.34	74941	1	0	0	0	0	0	0	1	0	0	1	0

[+ Code](#) [+ Markdown](#)

**THANK YOU!**