

# Actividad 4. Spark ML

Pablo González Sánchez. N° exp. 21938788  
Grandes volúmenes de datos. Universidad Europea

## 1. Análisis de los datos

### 1.1. Preparación de los CSV

Los CSV proporcionados vienen con un formato distinto al que estamos acostumbrados, por eso hay que convertirlos a un dataframe especificando algunas características:

```
CSVanime = spark.read.option("quote", "\"").option("escape", "\\").csv('../dataset_valoraciones_anime/anime.csv', inferSchema=True, header=True)
```

Para asegurarme de que trabaja todo correctamente, he hecho una serie de comprobaciones:

```
print("Dos primeros nombres que contienen [,]:")
CSVanime.select("Name", "ID").filter(col("Name").like("%,%")).show(2, truncate=False)
print("Dos primeros nombres que contienen [']:")
CSVanime.select("Name", "ID").filter(col("Name").like("%'")).show(2, truncate=False)
print("Dos primeros nombres que contienen [\"]:")
CSVanime.select("Name", "ID").filter(col("Name").like("%\"")).show(2, truncate=False)
print("Dos primeros nombres con caracteres japoneses:")
CSVanime.select("Japanese name", "ID").show(2, truncate=False)
```

```
Dos primeros nombres que contienen [,]:
+-----+-----+
|Name                                     |ID |
+-----+-----+
|Ima, Soko ni Iru Boku                   |160|
|Chiisana Obake: Acchi, Kocchi, Socchi  |310|
+-----+-----+
```

```
Dos primeros nombres con caracteres japoneses:
+-----+-----+
|Japanese name                          |ID |
+-----+-----+
|カウボーイビバップ                     |1  |
|カウボーイビバップ 天国の扉           |5  |
+-----+-----+
```

```
Dos primeros nombres que contienen [']:
+-----+-----+
|Name                                     |ID |
+-----+-----+
|Mahou Shoujo Lyrical Nanoha A's        |77 |
|Mobile Suit Gundam: The 08th MS Team - Miller's Report|83 |
+-----+-----+
```

```
Dos primeros nombres que contienen [\"]:
+-----+-----+
|Name                                     |ID |
+-----+-----+
|Love Hina: Motoko no Sentaku, Koi ka Ken... "Naku na"|963 |
|Lupin III: Episode 0 "First Contact"      |1418|
+-----+-----+
```

## 1.2. Ítems mejor y peor valorados

Para ello, he agrupado los ratings por ítem, extrayendo la valoración media y la cantidad de ítems en el grupo. Luego, los uní al dataframe de animes para sacar su nombre.

Para darle más sentido a esta clasificación, me quedé solo con aquellos ítems con al menos 50 ratings. El resultado son los primeros ítems ordenados por valoración media:

```
minValoraciones = 50
cantidadTop = 3
mejorMedia = CSVrating.groupBy("anime_id").agg(F.mean("rating"), F.count("rating"))
mejorMedia = mejorMedia.withColumnRenamed("avg(rating)", "average")
mejorMedia = mejorMedia.withColumnRenamed("count(rating)", "count")
juntar = CSVanime.select("ID", "Name")
mejorMedia = mejorMedia.join(juntar, CSVrating.anime_id==juntar.ID)
mejorMedia = mejorMedia.where(col("count") > minValoraciones)
print("Top", cantidadTop, "items con mayor nota media (con al menos", minValoraciones, "valoraciones):")
mejorMedia.sort("average", ascending=False).withColumnRenamed("count", "Nº ratings").drop("ID", "anime_id").show(cantidadTop, truncate=False)
print("Top", cantidadTop, "items con peor nota media (con al menos", minValoraciones, "valoraciones):")
mejorMedia.sort("average", ascending=True).withColumnRenamed("count", "Nº ratings").drop("ID", "anime_id").show(cantidadTop, truncate=False)
```

Top 3 items con mayor nota media (con al menos 50 valoraciones):

average	Nº ratings	Name
9.24564778301433	20794	Gintama°
9.237009769219878	134197	Fullmetal Alchemist: Brotherhood
9.224729815638906	9438	Ginga Eiyuu Densetsu

Top 3 items con peor nota media (con al menos 50 valoraciones):

average	Nº ratings	Name
1.749909974792942	5554	Tenkuu Danzai Skelter+Heaven
1.842294767162746	4166	Utsu Musume Sayuri
1.9944827586206897	725	Tsui no Sora

## 1.3. Valoraciones por género

La misma lógica seguí para obtener los géneros con mayores valoraciones, con una peculiaridad: Los géneros venían en una cadena de texto listándolos, así que primero tuve que convertir esa lista a un array, dividido por comas, eliminar los espacios, y luego con la función explode() dividir cada género en una fila independiente:

```
print("Géneros con mejor valoración media:")
CSVgeneros = CSVanime.select("Genres", "ID").filter(CSVanime.Genres != "Unknown")
CSVgeneros = CSVgeneros.join(CSVrating, CSVgeneros.ID==CSVrating.anime_id).drop("ID", "anime_id", "user_id")
CSVgeneros = CSVgeneros.select(split(col("Genres"), ","), "rating").withColumnRenamed("split(Genres, ,, -1)", "Genres")
CSVgeneros = CSVgeneros.select(explode(CSVgeneros.Genres), CSVgeneros.rating).withColumnRenamed("col", "Genero")
CSVgeneros = CSVgeneros.withColumn("Genero", trim(CSVgeneros.Genero))
CSVgeneros.groupBy("Genero").agg(F.mean("rating")).sort("avg(rating)", ascending=False).show(5)
print("Géneros con peor valoración media:")
CSVgeneros.groupBy("Genero").agg(F.mean("rating")).sort("avg(rating)", ascending=True).show(5)
```

Géneros con mejor valoración media:

Genero	avg(rating)
Thriller	8.076588242745846
Samurai	7.935889107317298
Historical	7.863716624096956
Military	7.861498646075896
Police	7.835031637882817

Géneros con peor valoración media:

Genero	avg(rating)
Yaoi	6.0414369519964675
Hentai	6.341275593904351
Yuri	6.568662562954673
Kids	6.9025075230360144
Shoujo Ai	7.065559471779689

## 1.4. Estudios con mejor y peor nota

Del mismo modo, para los estudios mejor y peor valorados:

```
CSVestudios = CSVanime.select("Studios", "Score")
CSVestudios = CSVestudios.filter((CSVestudios.Score != "Unknown") & (CSVestudios.Studios != "Unknown"))
CSVestudios = CSVestudios.groupBy("Studios").agg(F.mean("Score"), F.count("Score"))
CSVestudios = CSVestudios.withColumnRenamed("avg(Score)", "Valoracion_media").withColumnRenamed("count(Score)", "Cantidad")
CSVestudios = CSVestudios.filter(CSVestudios.Cantidad >= 30).drop("Cantidad")
print("Estudios con peor valoración media (con al menos 30 valoraciones):")
CSVestudios.orderBy('Valoracion_media', ascending=True).show(5, truncate=False)
print("Estudios con mejor valoración media (con al menos 30 valoraciones):")
CSVestudios.orderBy('Valoracion_media', ascending=False).show(5, truncate=False)
```

Estudios con peor valoración media (con al menos 30 valoraciones):

Studios	Valoracion_media
DLE	5.843207547169812
Magic Bus	6.241388888888888
Studio 4°C	6.268596491228069
Arms	6.2762162162162145
Seven	6.388412698412699

Estudios con mejor valoración media (con al menos 30 valoraciones):

Studios	Valoracion_media
Wit Studio	7.457999999999999
Bones	7.352520325203253
White Fox	7.350769230769229
Kyoto Animation	7.321727272727272
David Production	7.296969696969697

## 2. Recomendaciones con SparkML

### 2.1. Construcción del dataframe

Para esta tarea hacen falta los tres dataframes, así que lo primero es unirlos. Primero, añadir el de ratings del usuario EP debajo del de ratings general (función union()). El de animes se une después según la columna anime\_id. Luego renombré las columnas y eliminé las filas con datos sin completar

```
CSVml = CSVrating.union(CSVep).withColumnRenamed("rating", "User_rating")
CSVml = CSVml.join(CSVanime, CSVml.anime_id==CSVanime.ID)
CSVml = CSVml.select("user_id", "anime_id", "User_rating", "Genres", "name", "English name", "Type", "Episodes", "Studios", "Source", "Ranked", "Duration", "Popularity", "Score-10", "Score-9", "Score-8", "Score-7", "Score-6", "Score-5", "Score-4", "Score-3", "Score-2", "Score-1")
CSVml = CSVml.withColumnRenamed("English name", "English_name")
CSVml = CSVml.withColumn("anime_id", col("anime_id").cast(IntegerType()))
CSVml = valMedia(CSVml)
CSVml = durationNumber(CSVml)
CSVml = CSVml.where(CSVml.Episodes != "Unknown").where(CSVml.English_name != "null").where(CSVml.English_name != "Unknown")
CSVml = CSVml.na.drop(subset=["DurationN", "Val_media"])
print("El dataframe tiene", CSVml.count(), "filas")
CSVml.printSchema()
CSVml.show(5)
```

```
El dataframe tiene 46357218 filas
root
 |-- user_id: integer (nullable = true)
 |-- anime_id: integer (nullable = true)
 |-- User_rating: double (nullable = true)
 |-- Genres: string (nullable = true)
 |-- name: string (nullable = true)
 |-- English_name: string (nullable = true)
 |-- Type: string (nullable = true)
 |-- Episodes: string (nullable = true)
 |-- Studios: string (nullable = true)
 |-- Source: string (nullable = true)
 |-- Ranked: string (nullable = true)
 |-- Popularity: integer (nullable = true)
 |-- Val_media: double (nullable = true)
 |-- DurationN: double (nullable = true)
```

user_id	anime_id	User_rating	Genres	name	English_name	Type	Episodes	Studios	Source	Ranked	Popularity	Val_media	DurationN
0	430	9.0	Military, Comedy,...	Fullmetal Alchemi...	Fullmetal Alchemi...	Movie	1	Bones	Manga	1361.0	506	7.587968995355985	105.0
0	1004	5.0	Drama, Psychologi...	Kanojo to Kanojo ...	She and Her Cat:T...	OVA	1	Unknown	Original	2226.0	1624	7.350737797956867	4.0
0	3010	7.0	Adventure, Histor...	Kaiketsu Zorro	The Magnificent Z...	TV	52	Ashi Production	Other	2655.0	5104	7.227823867262285	24.0
0	570	7.0	Military, Police,...	Jin-Rou	Jin-Roh:The Wolf ...	Movie	1	Production I.G	Manga	846.0	1181	7.796858984381786	102.0
0	431	8.0	Adventure, Drama,...	Howl no Ugoku Shiro	Howl's Moving Castle	Movie	1	Studio Ghibli	Novel	51.0	98	8.69212647198615	119.0

También pasé el dataframe por dos funciones que me construí, una para convertir la columna de duración a números (era cadena de texto sin ningún tipo de estándar) y otra para convertir las 10 columnas de valoraciones a una valoración media:

```
def valMedia(df):
    df = df.withColumn("Val_media", col("Score-10")*10+col("Score-9")*9+col("Score-8")*8+col("Score-7")*7+col("Score-6")*6+col("Score-5")*5+col("Score-4")*4+col("Score-3")*3+col("Score-2")*2+col("Score-1"))
    df = df.drop("Score-1", "Score-2", "Score-3", "Score-4", "Score-5", "Score-6", "Score-7", "Score-8", "Score-9", "Score-10")
    return df

def durationNumber(df):
    df = df.withColumn("Duration", regexp_replace("Duration", " per ep.", ""))
    df = df.withColumn("Duration", regexp_replace("Duration", "\.", ""))
    df = df.withColumn("Duration", regexp_replace("Duration", " hr", "h"))
    df = df.withColumn("Duration", regexp_replace("Duration", " min", "m"))
    df = df.withColumn("Duration", reverse(split(col("Duration"), " ")))
    df = df.withColumn("mins", regexp_replace(col("Duration")[0], "m", ""))
    df = df.withColumn("hours", regexp_replace(col("Duration")[1], "h", ""))
    df = df.fillna({"hours": 0})
    df = df.withColumn("DurationN", col("hours")*60+col("mins"))
    df = df.drop("Duration", "mins", "hours")
    return df
```

NOTA: Los puntos 2.2, 2.3 y 2.4 se realizan dos veces, una para series y otra para películas. En este documento se ilustrará el proceso realizado para las series, pero el proceso para las películas es idéntico, salvo detalles explicados en los siguientes apartados.

## 2.2. Preparación de los datos

Del dataframe anterior, seleccioné sólo las filas cuya columna de tipo era "ONA" (siempre que el número de episodios fuese 2 o mayor) ó "TV". En el caso de las películas, simplemente seleccioné las que tuviesen como tipo "Movie". Luego, con la función StringIndexer pude asignarle un número a cada elemento de una columna de texto. Esto me permitió utilizar las columnas de géneros, tipo (en el caso de las series), estudios y source para el entrenamiento, ya que el algoritmo ALS sólo permite columnas numéricas.

```
EPseries = CSVml.where(((CSVml.Type == "ONA" & (CSVml.Episodes > 1)) | (CSVml.Type == "TV")).drop("Japanese_name"))
print("El dataframe tiene", EPseries.count(), "filas")
EPseries = EPseries.withColumn("Episodes", col("Episodes").cast(IntegerType()))
EPseries = EPseries.withColumn("Ranked", col("Ranked").cast(IntegerType()))
indexer = StringIndexer(inputCol="Genres", outputCol="GenreInt")
EPseries = indexer.fit(EPseries).transform(EPseries)
indexer = StringIndexer(inputCol="Type", outputCol="TypeN")
EPseries = indexer.fit(EPseries).transform(EPseries)
indexer = StringIndexer(inputCol="Studios", outputCol="StudiosN")
EPseries = indexer.fit(EPseries).transform(EPseries)
indexer = StringIndexer(inputCol="Source", outputCol="SourceN")
EPseries = indexer.fit(EPseries).transform(EPseries)
EPseries.show(5)
```

El dataframe tiene 35896854 filas

user_id	anime_id	User_rating	name	English_name	Type	Episodes	Ranked	Popularity	Val_media	Duration	GenresN	TypeN	StudiosN	SourceN			
0	3010	7.0	Adventure, Histor...	Kaiketsu Zorro	The Magnificent Z...	TV	52	Ashi Production	Other	2655	5104	7.227823867262285	24.0	1336.0	0.0	302.0	8.0
0	1571	10.0	Mystery, Comedy, ...	Ghost Hunt	Ghost Hunt	TV	25	J.C.Staff	Light novel	811	766	7.7579663241750785	25.0	479.0	0.0	0.0	1.0
0	121	9.0	Action, Adventure...	Fullmetal Alchemist	Fullmetal Alchemist	TV	51	Bones	Manga	337	52	8.15076332151554	24.0	73.0	0.0	3.0	0.0
0	356	9.0	Action, Supernatu...	Fate/stay night	Fate/stay night	TV	24	Studio Deen	Visual novel	2152	119	7.328928488661634	24.0	115.0	0.0	4.0	3.0
0	1250	7.0	Adventure, Comedy...	Eremerstar Gerad	Elemental Gelade	TV	26	Xebec	Manga	2325	1587	7.252946728949343	24.0	723.0	0.0	17.0	0.0

```
EPseriesN = EPseries.drop("Genres", "Type", "Studios", "Source")
EPseriesN.printSchema()
EPseriesN.show(5)
```

user_id	anime_id	User_rating	name	English_name	Episodes	Ranked	Popularity	Val_media	Duration	GenresN	TypeN	StudiosN	SourceN
0	3010	7.0	Kaiketsu Zorro	The Magnificent Z...	52	2655	5104	7.227823867262285	24.0	1336.0	0.0	302.0	8.0
0	1571	10.0	Ghost Hunt	Ghost Hunt	25	811	766	7.7579663241750785	25.0	479.0	0.0	0.0	1.0
0	121	9.0	Fullmetal Alchemist	Fullmetal Alchemist	51	337	52	8.15076332151554	24.0	73.0	0.0	3.0	0.0
0	356	9.0	Fate/stay night	Fate/stay night	24	2152	119	7.328928488661634	24.0	115.0	0.0	4.0	3.0
0	1250	7.0	Eremerstar Gerad	Elemental Gelade	26	2325	1587	7.252946728949343	24.0	723.0	0.0	17.0	0.0

## 2.3. Entrenamiento

Con los dataframe listos, lo último fue pasarlos por el algoritmo de recomendaciones de SparkML. Lo primero, lo dividí aleatoriamente en training (80%) y test (20%). Luego utilicé ALS para hacer las transformaciones y generar el modelo y las predicciones. Esto me devolvió las 5 mejores para cada usuario.

```
training, test = EPPelisN.randomSplit([0.8, 0.2])
als = ALS(maxIter=5, regParam=0.01, userCol="user_id", itemCol="anime_id", ratingCol="User_rating", coldStartStrategy="drop")
model = als.fit(training)
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="User_rating", predictionCol="prediction")

userRecs = model.recommendForAllUsers(5)
userRecs.show(5, truncate=False)
```

user_id	recommendations
34	[[{32890, 17.97881}, {40811, 17.349768}, {32269, 16.618448}, {3362, 14.566329}, {16480, 13.598028}]]
53	[[{32890, 21.307026}, {30268, 16.022106}, {16776, 14.56802}, {40811, 13.736708}, {31135, 13.680226}]]
65	[[{33533, 27.701912}, {17469, 27.217438}, {25921, 20.402641}, {13817, 19.524263}, {29135, 17.34979}]]
78	[[{25093, 17.741339}, {30268, 14.929622}, {31135, 11.68317}, {6728, 10.904329}, {7366, 10.706601}]]
85	[[{8598, 7.55478}, {3784, 7.5107765}, {33049, 7.4911385}, {433, 7.3989887}, {1689, 7.351724}]]

Para obtener las recomendaciones para el usuario EP, filtré los resultados por su id

```
recomPelis = userRecs.where(col("user_id") == 666666)
recomPelis = recomPelis.select("user_id", explode(recomPelis.recommendations)).withColumnRenamed("col", "seleccion")
recomPelis = recomPelis.select("user_id", "seleccion.*")
recomPelis.printSchema()
recomPelis.show(truncate=False)

root
|-- user_id: integer (nullable = false)
|-- anime_id: integer (nullable = true)
|-- rating: float (nullable = true)

+-----+-----+-----+
|user_id|anime_id|rating|
+-----+-----+-----+
|666666|16776|21.626589|
|666666|37661|17.088545|
|666666|4621|16.063705|
|666666|23343|16.050005|
|666666|10149|15.463894|
+-----+-----+-----+
```

Para asociar los ids resultantes, tuve que volver a unir los resultados con el dataframe de animes. Seleccionando las columnas que queríamos mostrar, y ordenando por valoración media como se pedía, obtuve los resultados finales:

```
EPpelisElegidas = EPpelis.where((col("anime_id") == 32890) | (col("anime_id") == 33533) | (col("anime_id") == 21129) | (col("anime_id") == 4621))
EPpelisElegidas = EPpelisElegidas.dropDuplicates(["anime_id"])
EPpelisElegidas = EPpelisElegidas.select("anime_id", "User_rating", "name", "English_name", "Val_media")
EPpelisElegidas = EPpelisElegidas.sort("Val_media", ascending=False)
EPpelisElegidas.show(truncate=False)

+-----+-----+-----+-----+-----+
|anime_id|User_rating|name|English_name|Val_media|
+-----+-----+-----+-----+-----+
|4621|7.0|Arisubyeon-ui Kkumnamu|The Olympic Challenge|6.194444444444445|
|32890|7.0|Tu Xia Zhi Qing Li Chuanshuo|Legend of a Rabbit:The Martial of Fire|5.745762711864407|
|21129|3.0|Youtai Nuhai Zai Shanghai|A Jewish Girl in Shanghai|5.621951219512195|
|37392|2.0|Xi Yang Yang Yu Hui Tai Lang: Zhi Yang Nian Xi Yang Yang|Amazing Pleasant Goat|5.159090909090909|
|33533|5.0|Tokyo Onlypic|Tokyo Onlypic Pictures 2008|5.135135135135135|
+-----+-----+-----+-----+-----+

1 EPseriesElegidas = EPseries.where((col("anime_id") == 3869) | (col("anime_id") == 8786) | (col("anime_id") == 14623) | (col("anime_id") == 19987) | (col("anime_id") == 14623))
2 EPseriesElegidas = EPseriesElegidas.dropDuplicates(["anime_id"])
3 EPseriesElegidas = EPseriesElegidas.select("anime_id", "name", "English_name", "Val_media")
4 EPseriesElegidas = EPseriesElegidas.sort("Val_media", ascending=False)
5 EPseriesElegidas.show(truncate=False)

[Stage 158:=====> (16 + 1) / 17]
+-----+-----+-----+-----+-----+
|anime_id|name|English_name|Val_media|
+-----+-----+-----+-----+-----+
|3869|Sakura Momoko Gekijou: Coji-Coji|Coji-Coji|6.37037037037037|
|8786|Inakappe Taishou|General Inakappe|5.768|
|28145|Johnny Cypher|Johnny Cypher in Dimension Zero|5.625|
|19987|Kaitou Pride|Dr. Zen|5.55|
|14623|Chikyuu SOS Sore Ike Kororin|Do it Kororin Earth SOS|5.375|
+-----+-----+-----+-----+-----+
```

NOTA: Este proceso no pude ejecutarlo localmente en mi ordenador, pues la cantidad de datos y de cálculos necesarios superaban las capacidades de mi portátil, así que este apartado tuve que hacerlo desde un clúster de Google Cloud

### 3. Representación gráfica con API Jikan

Para obtener información detallada de las series y películas resultantes, utilicé la API Jikan. Así, construí un array con los ids de los resultados y los recorrí en bucle, obteniendo la foto de portada, el nombre y la sinopsis, la duración y la cantidad de episodios.

Además, tuve que poner un pequeño delay a cada llamada a la API para que no me bloqueara por DoS:

```
peliculas = [4621, 32890, 21129, 37392, 33533]
series = [3869, 8786, 28145, 19987, 14623]
animés = [peliculas, series]
html_code = ""
print("Recuperando información de la API", end="")
for tipos in animés:
    if(tipos == animés[0]):
        html_code += '<h1 style="text-align:center">Estas son las películas recomendadas para el usuario EP:</h1>'
    else:
        html_code += '<h1 style="text-align:center">Estas son las series recomendadas para el usuario EP:</h1>'
    for anime in tipos:
        api_url = "https://api.jikan.moe/v4/anime/"+str(anime)+"/full"
        response = requests.get(api_url).json()
        nombre = response["data"]["titles"][0]["title"]
        if(response["data"]["synopsis"] == None):
            sinopsis = "No se ha encontrado sinopsis para esta serie"
        else:
            sinopsis = response["data"]["synopsis"]
        imagen = response["data"]["images"]["jpg"]["image_url"]
        if(tipos == series):
            episodiosHTML = "<p><b>Episodios de la serie: </b>" + str(response['data']['episodes']) + "</p>"
        else:
            episodiosHTML = ""
        duracion = response["data"]["duration"]
        generos_ = response["data"]["genres"]
        generos = ""
        for genero in generos_:
            generos += genero["name"] + ", "
        generos = generos[:len(generos)-2]
        if(generos == ""):
            generos = "No se han encontrado géneros para este anime aún"
        html_code += '''
<div style="display:flex;border:1px solid black;padding:10px;width:75%;margin:auto;margin-bottom:10px;">
```

Para representar esto gráficamente, utilicé una cadena de texto con código HTML, pero con la información mediante variables de python:

```

html_code += '''
<div style="display:flex;border:1px solid black;padding:10px;width:75%;margin:auto;margin-bottom:10px;">
  <div style="padding-right:10px;flex:1;"></div>
  <div style="padding-right:10px;flex:5;">
    <p><b>Nombre: </b>'+nombre+'</p>
    <p><b>Sinopsis:</b></p>
    <p>'+sinopsis+'</p>
    '+episodiosHTML+'
    <p><b>Duración: </b>'+duracion+'</p>
    <p><b>Géneros: </b>'+generos+'</p>
  </div>
</div>
'''
time.sleep(0.5) # Para que la API no me bloquee (por entender que le estoy haciendo DoS)
print(".", end="")
HTML(html_code)

```

Recuperando información de la API.....

## Estas son las películas recomendadas para el usuario EP:



**Nombre:** Arisubyeon-ui Kkumnamu

**Sinopsis:**

This is a story of a 13 year old girl whose only dream is to participate in the Olympic Games: Her Struggles, her victories, her failures and her willpower fascinate all who help her to become a outstanding athlete.

**Duración:** 1 hr 15 min

**Géneros:** Drama, Sports



**Nombre:** Tu Xia Zhi Qing Li Chuanshuo


**Sinopsis:**

Sequel to the movie Legend of a Rabbit It's been couples of years since Tu defeated Slash. More than a hero, Tu is regarded as the protector of the village defeating bandits and upholding justice. However, with no real foundation in Kung Fu, Tu has to start from zero in order to harness the Kung Fu skills which he inherited from the Grandmaster. One day, Tu rescues an old injured warrior who finally turns out to be the evil Zhan. Zhan plans to take over the world of Martial Art by acquiring a precious gem from the real successor Lan of Huo Clan. Upon confrontation, Tu fails to save Lan and the Martial of Fire. The reality hits him and Tu realizes he's just a lucky rabbit who happens to inherit the Grandmaster's Kung Fu. Therefore, Tu restarts to practicing Kung Fu as much encouragement from Penny and Biggie. As time goes by, Tu's able to get on his feet and join his fellow warriors, Penny and Biggie, to stop Zhan's evil plan, while Zhan is getting closer to his goal. Finally, Tu defeats against evil Zhan by holding the True Harmony of the Martial of Fire & Water. (Source: GSC Movies)

**Duración:** 1 hr 35 min

**Géneros:** Action, Adventure, Comedy






**Nombre:** Yutai Nuhai Zai Shanghai

**Sinopsis:**

Set mainly in and around the Shanghai Ghetto in Japanese-occupied Shanghai during the Second World War, the film tells the story of three children. Rina and her younger brother Mishalli are Jewish refugees who escaped Europe but are without their parents. A-Gen is a Chinese boy who meets Rina and helps her and her brother to survive. The children form strong friendships and have adventures as they try and fend off the Japanese army occupying the city, and their allies, the Nazis. In the background, the Second Sino-Japanese War takes place, while the children must face the uncertainty that concerns the fate of Rina and Mishalli's parents in Europe. (Source: AniDB)

**Duración:** 1 hr 19 min

**Géneros:** No se han encontrado géneros para este anime aún




**Nombre:** Xi Yangyang Yu Hui Tailang: Yang Nian Xi Yangyang

**Sinopsis:**

In Green Green Pastures, there exists a much loved legend about valiant dragon-slayers who lived in the Stone Age. Having grown up listening to stories about their awe-inspiring adventures, Pleasant Goat and Fatty Goat have always wanted to become dragon slayers themselves. While trying to make this dream come true, Fatty Goat inadvertently switches bodies with Big Big Wolf, and the two, along with Pleasant Goat, soon find themselves transported back to the Stone Age. When an evil dragon appears and threatens Stone Age Green Green Pastures, Fatty Goat, Big Big Wolf and Pleasant Goat must rise up to the dragon-slaying challenge. (Source: YesAsia)

**Duración:** 1 hr 26 min

**Géneros:** Adventure, Comedy



**Nombre:** Tokyo Onlypic


**Sinopsis:**

An omnibus film depicting 15 fictional sports events created by different directors. Onlypic stands for only + pictures (video).

**Duración:** 2 hr 47 min

**Géneros:** Comedy, Sports

## Estas son las series recomendadas para el usuario EP:



**Nombre:** Sakura Momoko Gekijou: Coji-Coji

**Sinopsis:**

The only thing that concerns Coji Coji is enjoying life and the wonders of nature. Coji Coji's friends, on the other hand, are a varied bunch of colorful characters whose many idiosyncrasies have a tendency to create turmoil among one another. Coji Coji's carefree ways and tendency to shirk responsibility are sometimes a source of frustration, but his childlike wisdom has a way of resolving all of their differences. (Source: AniDB)

**Episodios de la serie:** 100

**Duración:** 25 min per ep

**Géneros:** Comedy, Fantasy



**Nombre:** Inakappe Taishou

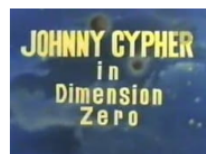
**Sinopsis:**

A comedy about a country boy that travels to the city to study judo. He ends up meeting the daughter of a prominent dojo and her judo skilled cat. Each episode features 2 stories. (Source: AniDB)

**Episodios de la serie:** 104

**Duración:** 24 min per ep

**Géneros:** Comedy, Sports



**Nombre:** Johnny Cypher

**Sinopsis:**

Square-jawed superagent Johnny can travel through inner space, Dimension Zero, and uses his superpowers to combat evil all over the universe. He is helped by the beautiful blonde Zena and tiny alien Rhom from the Black Star. An early Japanese-American co-production by former Disney animator Joe Oriolo for Warner/Seven--a company formed by the merging of Seven Arts production after its merger with Warner Bros. It started airing a year after its release in the US. (Source: The Anime Encyclopedia)

**Episodios de la serie:** 130

**Duración:** 5 min per ep

**Géneros:** Action, Adventure, Sci-Fi



**Nombre:** Kaitou Pride

**Sinopsis:**

Each episode features a part of the story in which a detective tries to hunt down and capture a notorious and narcissistic thief. This series aired daily, Monday to Friday, in 8 minutes segments to complete a single story by the end of the week. (Source: AniDB)

**Episodios de la serie:** 105

**Duración:** 8 min per ep

**Géneros:** Adventure, Comedy



**Nombre:** Chikyuu SOS Sore Ike Kororin

**Sinopsis:**

Jiang came to the old family cherry blossom viewing venue. But the surrounding area was full of garbage. He sits there to eat lunch reluctantly when a strange creature falls from the sky! Kororin is his name. He comes from the Star Oasis and is here to teach the importance of environmental consciousness with his ESP. Jiang, his sister, and Kororin must battle Syndrome, the evil alien plotting the destruction of the Earth. The attempted destruction is usually through some environmental issue such as tearing a hole in the ozone, or changing the weather, or overuse of electricity, etc.

**Episodios de la serie:** 26

**Duración:** 25 min per ep

**Géneros:** Action, Comedy, Sci-Fi