For this project, we will create an anime recommendation system. A user will input his/hers genre of choice and the model will output the top 10 anime of that genre, along with their descriptions.

The Data used for this project consist of about 17.562 anime and the preference from 325.772 different users.

Datasets Description:

The Anime_df has 16214 rows and 5 columns:

- MAL_IDName: the anime id (Primary key)
- Name: the title of the anime
- Score: the average rating (however we will drop that column)
- Genres: the genres of each anime entry
- Synopsis: a brief description of the anime.

The Ratings_df has 57633278 rows and 3 columns:

- user_id: Id of the user who left a rating
- anime_id: anime id (foreigh key)
- rating: rating from 1 to 10

The Big Data problem surrounding this project is Volume, because of the large number of rows of the Ratings_df. Moreover, we will perform analysis on batch data.

Setting Up Pyspark

```
# innstall java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# install spark (change the version number if needed)
```

```
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz

# unzip the spark file to the current folder
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

# set your spark folder to your system path environment.
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"


# install findspark using pip
!pip install -q findspark


! pip install pyspark
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.3.1.tar.gz (281.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 2.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.5
  Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 KB 13.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.3.1-py2.py3-none-any.whl size=281845512 sha256=a2fb0778e2b37994376b88
  Stored in directory: /root/.cache/pip/wheels/43/dc/11/ec201cd671da62fa9c5cc77078235e40722170ceba231d7598
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.5 pyspark-3.3.1
```

```
import findspark
findspark.init()
findspark.find()
```

```
'/content/spark-3.0.0-bin-hadoop3.2'
```

```
from pyspark.context import SparkContext, SparkConf
from pyspark.sql import *
from pyspark.sql import SparkSession
from pyspark.sql import Window
from pyspark.sql import Row
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.sql import functions as F
from pyspark.sql.functions import explode
from pyspark.sql.functions import rank
from pyspark.sql.functions import col
from pyspark.sql.functions import avg
from pyspark.sql.functions import split
from pyspark.sql.functions import round
import time
from functools import reduce


spark = SparkSession.builder\
        .master('local[5]')\
        .appName('Anime recommendation system')\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

Importing Data

```
Anime_df = spark.read.csv('/content/drive/MyDrive/Colab Notebooks/anime_with_synopsis.csv', header=True, inferSchema=True)
Ratings_df = spark.read.csv('/content/drive/MyDrive/Colab Notebooks/rating_complete.csv', header=True, inferSchema=True)


def withColumnRenamed(existingName: any, newName: any):
  DataFrame

Anime_df = Anime_df.withColumnRenamed("sypnopsis","synopsis")


Anime_df.show(5)
```

```
+------+--------------------+-----+--------------------+--------------------+
|MAL_ID|                Name|Score|              Genres|            synopsis|
+------+--------------------+-----+--------------------+--------------------+
|     1|        Cowboy Bebop| 8.78|Action, Adventure...|"In the year 2071...|
|     5|Cowboy Bebop: Ten...| 8.39|Action, Drama, My...|other day, anothe...|
|     6|              Trigun| 8.24|Action, Sci-Fi, A...|"Vash the Stamped...|
|     7|  Witch Hunter Robin| 7.27|Action, Mystery, ...|ches are individu...|
|     8|      Bouken Ou Beet| 6.98|Adventure, Fantas...|It is the dark ce...|
+------+--------------------+-----+--------------------+--------------------+
only showing top 5 rows
```

```
Ratings_df.show(5)
```

```
+-------+--------+------+
|user_id|anime_id|rating|
+-------+--------+------+
|      0|     430|     9|
|      0|    1004|     5|
|      0|    3010|     7|
|      0|     570|     7|
|      0|    2762|     9|
+-------+--------+------+
only showing top 5 rows
```

Data Processing

For the project, we will use the Anime_df as well as the Ratings_df. We will drop the 'Score' column for the Anime_df as we will create a model that will aggregate the ratings of the Ratings_df and then output the score.

```
Anime_df = Anime_df.drop('Score')
```

Then we will explode the genre column, so each row have just one gerne

```
Anime_df.printSchema()
```

```
root
 |-- MAL_ID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Genres: string (nullable = true)
 |-- synopsis: string (nullable = true)
```

Checking for Null values:

```
Anime_df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in Anime_df.columns]).show()
```

```
+------+----+------+--------+
|MAL_ID|Name|Genres|synopsis|
+------+----+------+--------+
|     0|   0|     0|       8|
+------+----+------+--------+
```

```
Ratings_df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in Ratings_df.columns]).show()
```

```
+-------+--------+------+
|user_id|anime_id|rating|
+-------+--------+------+
|      0|       0|     0|
+-------+--------+------+
```

As we can see, Anime_df has 8 null values in the synopsis column, however, we will not drop these rows.

Printing the shapes of the above dataframes:

```
print(f'Anime_df has {Anime_df.count()} rows and {len(Anime_df.columns)} columns')
print(f'Ratings_df has {Ratings_df.count()} rows and {len(Ratings_df.columns)} columns')
```

```
Anime_df has 16214 rows and 4 columns
Ratings_df has 57633278 rows and 3 columns
```

Because the genre column of the Anime_df consists of multiple genres per anime, we will split them, resulting in each row containing only one genre per anime.

```
# Converting each entry of the 'Genres' column to a list of strings, for them to be exploded
Anime_df = Anime_df.withColumn('Genres', split(Anime_df['Genres'], ', '))
Anime_df.show()
```

```
+------+--------------------+--------------------+--------------------+
|MAL_ID|                Name|              Genres|            synopsis|
+------+--------------------+--------------------+--------------------+
|     1|        Cowboy Bebop|[Action, Adventur...|"In the year 2071...|
|     5|Cowboy Bebop: Ten...|[Action, Drama, M...|other day, anothe...|
|     6|              Trigun|[Action, Sci-Fi, ...|"Vash the Stamped...|
|     7|  Witch Hunter Robin|[Action, Mystery,...|ches are individu...|
|     8|      Bouken Ou Beet|[Adventure, Fanta...|It is the dark ce...|
|    15|        Eyeshield 21|[Action, Sports, ...|"Sena is like any...|
|    16|Hachimitsu to Clover|[Comedy, Drama, J...|Yuuta Takemoto, a...|
|    17|Hungry Heart: Wil...|[Slice of Life, C...|Kyosuke Kano has ...|
|    18|Initial D Fourth ...|[Action, Cars, Sp...|"Takumi Fujiwara ...|
|    19|             Monster|[Drama, Horror, M...|Dr. Kenzou Tenma,...|
|    20|              Naruto|[Action, Adventur...|oments prior to N...|
|    21|           One Piece|[Action, Adventur...|"Gol D. Roger was...|
|    22| Tennis no Ouji-sama|[Action, Comedy, ...|The world of tenn...|
|    23|    Ring ni Kakero 1|[Action, Shounen,...|In order to fulfi...|
|    24|       School Rumble|[Comedy, Romance,...|"Just the words "...|
|    25|           Sunabouzu|[Action, Adventur...|"The Great Kanto ...|
|    26|          Texhnolyze|[Action, Sci-Fi, ...|"Texhnolyze takes...|
|    27|       Trinity Blood|[Action, Supernat...|Following Armaged...|
|    28|    Yakitate!! Japan|   [Comedy, Shounen]|hile countries su...|
```

```
|    29|              Zipang|[Action, Military...|ai, an improved K...|
+------+--------------------+--------------------+--------------------+
only showing top 20 rows
```

Explode the 'Genre' column

```
Anime_df.printSchema()
```

```
root
 |-- MAL_ID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Genres: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- synopsis: string (nullable = true)
```

```
Anime_df = Anime_df.select( '*' , explode('Genres').alias('Genres2'))
```

```
Anime_df = Anime_df.drop('Genres')
Anime_df = Anime_df.withColumnRenamed('Genres2', 'Genres')
```

```
Anime_df.show()
```

```
+------+--------------------+--------------------+---------+
|MAL_ID|                Name|            synopsis|   Genres|
+------+--------------------+--------------------+---------+
|     1|        Cowboy Bebop|"In the year 2071...|   Action|
|     1|        Cowboy Bebop|"In the year 2071...|Adventure|
|     1|        Cowboy Bebop|"In the year 2071...|   Comedy|
|     1|        Cowboy Bebop|"In the year 2071...|    Drama|
|     1|        Cowboy Bebop|"In the year 2071...|   Sci-Fi|
|     1|        Cowboy Bebop|"In the year 2071...|    Space|
|     5|Cowboy Bebop: Ten...|other day, anothe...|   Action|
|     5|Cowboy Bebop: Ten...|other day, anothe...|    Drama|
|     5|Cowboy Bebop: Ten...|other day, anothe...|  Mystery|
```

```
|     5|Cowboy Bebop: Ten...|other day, anothe...|   Sci-Fi|
|     5|Cowboy Bebop: Ten...|other day, anothe...|    Space|
|     6|              Trigun|"Vash the Stamped...|   Action|
|     6|              Trigun|"Vash the Stamped...|   Sci-Fi|
|     6|              Trigun|"Vash the Stamped...|Adventure|
|     6|              Trigun|"Vash the Stamped...|   Comedy|
|     6|              Trigun|"Vash the Stamped...|    Drama|
|     6|              Trigun|"Vash the Stamped...|  Shounen|
|     7|  Witch Hunter Robin|ches are individu...|   Action|
|     7|  Witch Hunter Robin|ches are individu...|  Mystery|
|     7|  Witch Hunter Robin|ches are individu...|   Police|
+------+--------------------+--------------------+---------+
only showing top 20 rows
```

```
Ratings = Ratings_df.select(['anime_id' , 'rating'])
Ratings.show(5)
```

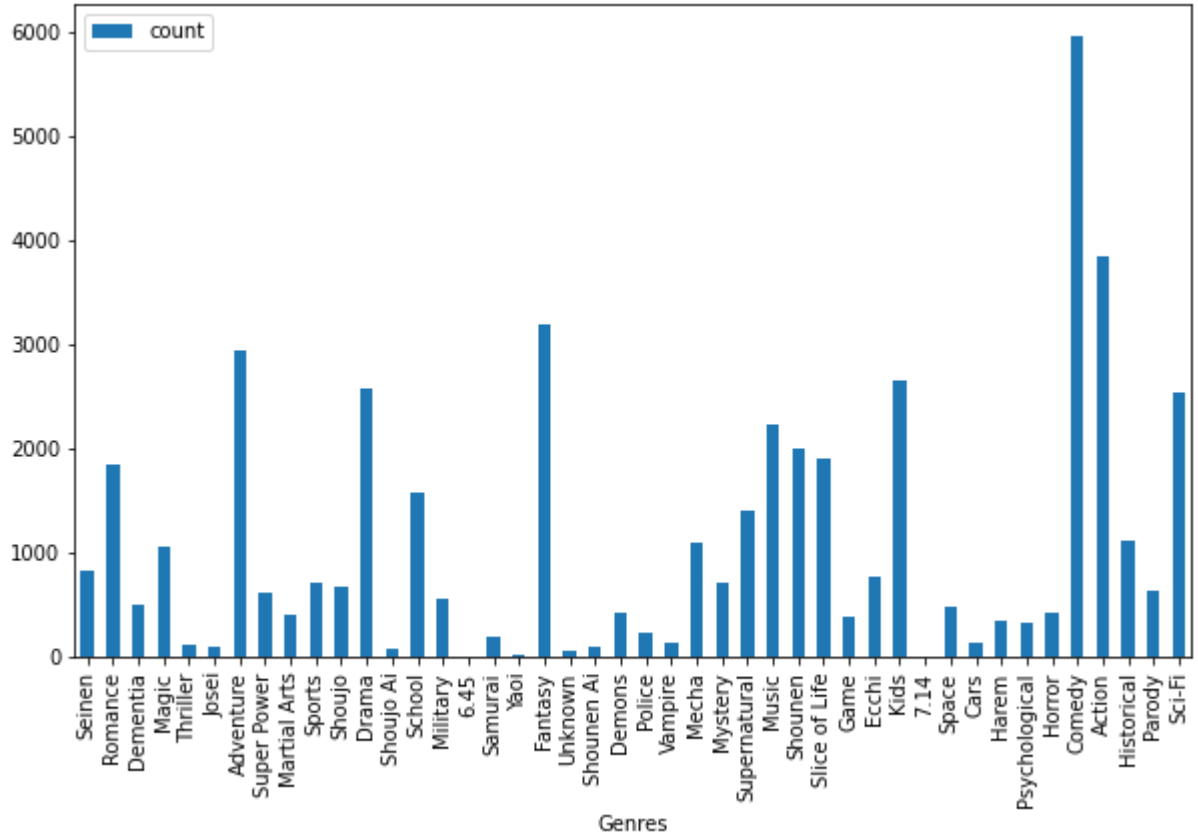
```
+--------+------+
|anime_id|rating|
+--------+------+
|     430|     9|
|    1004|     5|
|    3010|     7|
|     570|     7|
|    2762|     9|
+--------+------+
only showing top 5 rows
```

Visualization of the Genre column

```
# Group column by distinct values and count occurrences
df_count = Anime_df.groupBy('Genres').count()

# Convert DataFrame to Pandas
df_pd = df_count.toPandas()
```

```
# Create bar plot
df_pd.plot(kind='bar', x='Genres', y='count' , figsize=(10,6))
plt.show()
```



As we can observe, the Genre column is unbalanced as the counts differ. Comedy appears to be the most dominant genre of the dataframe contrastly with '7.14' and '6.45' to be the least popular genres.

First Model Design

For the first phase, we will process the 'Ratings' Dataframe, to compute the average rating of each anime. To achieve this, we will first turn the dataframe into an RDD. Then, with a Map function, we will create key-value pairs, where keys will be the anime ids and values their corresponding ratings. Afterward, the key-value pairs will be grouped by key creating key-lists of value pairs that then be input into a reduce function. Lastly, the reduce function will take the key-lists of values and output the average rating of each key.

```
def average_ratings(df):

    start_time = time.time()

    # Turning the dataframe into an RDD
    RDD = df.rdd

# Map function:
    # Performing the map function in parallel on each chunk, in which it will create key-value pairs, key: anime_id, value: rating:
    Map = RDD.map(lambda x : (x[0], x[1]) )

#Group-By key function:
    # Using the groupByKey() function to group the key-value pairs by key, creating key-lists of values pairs:
    GroupBy = Map.groupByKey()

#Reduce function:
    # Computing the average rating for each key:
    Reduce = GroupBy.mapValues(lambda x : sum(x) / len(x))

    end_time = time.time()
    print('Map-Reduce time: ', end_time - start_time)

    # Converting the RDD back to a DataFrame
    Avrg_df = Reduce.toDF(['anime_id', 'score'])


    end_time = time.time()
    print('Processing time: ', end_time - start_time)

    return Avrg_df
```

```
Average_Ratings = average_ratings(Ratings)
Average_Ratings.show()
```

```
Map-Reduce time:  0.15545392036437988
Processing time:  282.65165519714355
+--------+------------------+
|anime_id|             score|
+--------+------------------+
|    3010| 7.172523961661342|
|     448| 7.083866465045946|
|   36456| 8.239965483879402|
|   32935| 8.859160636758322|
|    9919|7.6643402810525325|
|    1575| 8.768003729304906|
|   38731| 8.159376708583926|
|   35847| 7.182853504986913|
|   37779| 8.566153479646228|
|   18753|7.4640287769784175|
|    4053|  8.08744394618834|
|   16870| 7.640721115780498|
|    8246| 7.317592254293022|
|   14749| 7.143337558527563|
|     154| 7.710264577780667|
|   37450| 8.264643210684385|
|   37086| 7.235680913652881|
|     189| 7.303943499936086|
|   29575|7.4541772151898735|
|   35581| 6.993574297188755|
+--------+------------------+
only showing top 20 rows
```

Second Approach to the First Model:

Another approach to compute the average rating of each anime in the Ratings Dataframe, would be to first group by the anime_id column. This is a similar procedure to the Map and GroupByKey functions. The output is key-lists of values pairs, where the keys are the anime ids paired

with lists of their corresponding ratings. Then, we will apply a reduction function by computing the average rating of each anime

```
def average_ratings(df):

    # We will also compute the processing time
    start_time = time.time()

    # Grouping by the anime_id
    df_grouped = df.groupBy('anime_id')
    # we will aggregate the results computing the average rating of each transaction
    df_avg = df_grouped.agg(avg('rating').alias('Score'))

    end_time = time.time()
    print('Processing time: ', end_time - start_time)


    return df_avg


Average_Ratings = average_ratings(Ratings)
Average_Ratings.show()
```

```
 Processing time:  0.03953242301940918
 +--------+-----------------+
 |anime_id|            Score|
 +--------+-----------------+
 |    1829|7.035614035087719|
 |    5300|8.622606525227313|
 |    8086|7.698882018100659|
 |   22097|7.786092786849789|
 |   30654|8.531057608714937|
 |   36538|8.389559360089812|
 |     496|6.663366336633663|
 |   38422|7.987347170884276|
 |    6336| 8.23187528378992|
 |    1088| 7.97648330855521|
 |     463|7.287537257824143|
 |    2142|6.377174486030575|
 |    9465| 7.95027027027027|
 |    3918|7.183235867446394|
```

```
|   25517|7.883919202518363|
|   30903|6.673501577287066|
|   11033|7.133634602969658|
|   17389|8.448859847979731|
|    2122|6.754385964912281|
|   40515|6.264705882352941|
+--------+-----------------+
only showing top 20 rows
```

Next step is to join the two tables, Average_Ratings and Anime_df, on anime_id.

```
Anime_DF = Anime_df.join(Average_Ratings, Anime_df.MAL_ID == Average_Ratings.anime_id, 'inner')
Anime_DF = Anime_DF.drop('MAL_ID')
Anime_DF.show()
```

```
+--------------------+--------------------+-------------+--------+-----------------+
|                Name|            synopsis|       Genres|anime_id|            Score|
+--------------------+--------------------+-------------+--------+-----------------+
|           Ged Senki|Calamities are pl...|      Fantasy|    1829|7.035614035087719|
|           Ged Senki|Calamities are pl...|        Magic|    1829|7.035614035087719|
|           Ged Senki|Calamities are pl...|    Adventure|    1829|7.035614035087719|
|Zoku Natsume Yuuj...|"s with its prequ...|       Shoujo|    5300|8.622606525227313|
|Zoku Natsume Yuuj...|"s with its prequ...|        Drama|    5300|8.622606525227313|
|Zoku Natsume Yuuj...|"s with its prequ...| Supernatural|    5300|8.622606525227313|
|Zoku Natsume Yuuj...|"s with its prequ...|       Demons|    5300|8.622606525227313|
|Zoku Natsume Yuuj...|"s with its prequ...|Slice of Life|    5300|8.622606525227313|
|Densetsu no Yuush...|""Alpha Stigma""...|      Shounen|    8086|7.698882018100659|
|Densetsu no Yuush...|""Alpha Stigma""...|      Fantasy|    8086|7.698882018100659|
|Densetsu no Yuush...|""Alpha Stigma""...|        Magic|    8086|7.698882018100659|
|Densetsu no Yuush...|""Alpha Stigma""...|    Adventure|    8086|7.698882018100659|
|Densetsu no Yuush...|""Alpha Stigma""...|       Action|    8086|7.698882018100659|
|Magi: Sinbad no B...|"Not so long ago,...|      Shounen|   22097|7.786092786849789|
|Magi: Sinbad no B...|"Not so long ago,...|        Magic|   22097|7.786092786849789|
|Magi: Sinbad no B...|"Not so long ago,...|      Fantasy|   22097|7.786092786849789|
|Magi: Sinbad no B...|"Not so long ago,...|    Adventure|   22097|7.786092786849789|
|Magi: Sinbad no B...|"Not so long ago,...|       Action|   22097|7.786092786849789|
|Ansatsu Kyoushits...|The students retu...|      Shounen|   30654|8.531057608714937|
|Ansatsu Kyoushits...|The students retu...|       School|   30654|8.531057608714937|
```

```
+--------------------+--------------------+-------------+--------+-----------------+
only showing top 20 rows
```

Second Model Design

In the second phase, we will collect the top 10 animes of each genre based on their score and store them in a Dataframe. This will be achieved by first turning the Anime_DF into an RDD and then applying a map function emitting key-value pairs, the key will be the genre and the values will be the name, synopsis, and score. Then, we will group by genre, creating key-lists of values pairs. Lastly, we will sort the key-lists of values pairs by score and save the top 10 animes in a new dataframe.

```
def top_10_by_genre(df):

    start_time = time.time()

    # Converting DataFrame to RDD
    rdd = df.rdd

# Map function:
    # Performing the map function, in which it will creat key-value pairs, key: genre , value: ( Name, Synopsis , Score ):
    Map = rdd.map(lambda x: (x[2], (x[0], x[1], x[4])))

# Group by key:
    # Grouping by key, and creating key-lists of value pairs
    List_of_values = Map.groupByKey().mapValues(lambda x: list(x))

# Sorting phase:

    # We will create an empty list to append the top 10 animes of each genre and then use it to create a dataframe:
    top_10_list = []

    # Creating a for loop, firstly sorting the values of each genre by their score, collecting the top 10 and lastly appending the results in t
    for genre, values in List_of_values.collect():
        top10 = sorted( values, key = lambda x: x[2] , reverse=True )[:10]
```

```
            for row in top10:
                top_10_list.append(Row(genre, row[0] , row[1] , row[2]))

        Top10_DF = spark.createDataFrame(top_10_list)

        # Processing the Dataframe
        Top10_DF = Top10_DF.withColumnRenamed('_1', 'Genre')
        Top10_DF = Top10_DF.withColumnRenamed('_2', 'Name')
        Top10_DF = Top10_DF.withColumnRenamed('_3', 'Synopsis')
        Top10_DF = Top10_DF.withColumn('Score', round(Top10_DF['_4'], 2))
        Top10_DF= Top10_DF.drop('_4')

        end_time = time.time()
        print('Processing time: ', end_time - start_time)

        return Top10_DF


    Top10 = top_10_by_genre(Anime_DF)
    Top10.show()
```

```
Processing time:  90.39276504516602
+-------+--------------------+--------------------+-----+
|  Genre|                Name|            Synopsis|Score|
+-------+--------------------+--------------------+-----+
|Fantasy|Momotarou no Kout...|omotarou themed t...| 10.0|
|Fantasy|Nendjuugyouji Ani...|animation teachin...| 10.0|
|Fantasy|Tatoeba Last Dung...|"long time ago, t...| 10.0|
|Fantasy|Oz no Mahoutsukai...|ducational film a...| 9.33|
|Fantasy|Fullmetal Alchemi...|"""In order for s...| 9.24|
|Fantasy|Hunter x Hunter (...|Hunter x Hunter i...| 9.17|
|Fantasy|Shingeki no Kyoji...|"Seeking to resto...| 9.03|
|Fantasy| Aria the Crepuscolo|The season is fal...|  9.0|
|Fantasy|Re:Zero kara Haji...|Second half of Re...|  9.0|
|Fantasy|            Mushishi|"""Mushi"": the m...| 8.84|
| Shoujo|Natsume Yuujincho...|"Takashi Natsume,...| 8.76|
| Shoujo|Natsume Yuujincho...|Takashi Natsume h...|  8.7|
| Shoujo|Natsume Yuujincho...|"Natsume Yuujinch...| 8.66|
| Shoujo|                Nana|Nana Komatsu is a...| 8.65|
| Shoujo|Zoku Natsume Yuuj...|"s with its prequ...| 8.62|
| Shoujo|Natsume Yuujincho...|Season 5 of Natsu...| 8.61|
```

```
| Shoujo|Fruits Basket 2nd...|ar has passed sin...| 8.56|
| Shoujo|  Versailles no Bara|In 1755, Marie An...| 8.44|
| Shoujo|  Natsume Yuujinchou|"hile most fiftee...| 8.41|
| Shoujo|Natsume Yuujincho...|"Takashi Natsume ...| 8.39|
+-------+--------------------+--------------------+-----+
only showing top 20 rows
```

Now we will create a function, that will asks a user for genres and then return the top animes that belong to those genres.

```
def Anime_Recommendation_System(df):

  print('-------------------------------------------------------------------------------------------------------------------------------------------------------')
  print('Welcome to the Anime Recommendation System!\n')
  print('You will be asked to input your genre of choice and the system will return the top 10 animes that correspond to that genre')
  print('Lets get started!\n')
  print('-------------------------------------------------------------------------------------------------------------------------------------------------------')

  Genres = input( 'What genre you are interested in? \n')

  df.filter(df.Genre == Genres).select('Name', 'Synopsis').show()


Anime_Recommendation_System(Top10)
```

```
---------------------------------------------------------------------------------------------------------------------------------------

Welcome to the Anime Recommendation System!

You will be asked to input your genre of choice and the system will return the top 10 animes that correspond to that
Lets get started!

---------------------------------------------------------------------------------------------------------------------------------------

What genre you are interested in?
Comedy
+--------------------+--------------------+
|                Name|            Synopsis|
+--------------------+--------------------+
```

```
|            Gintama°|Gintoki, Shinpach...|
|Fullmetal Alchemi...|"""In order for s...|
|            Gintama'|fter a one-year h...|
|  Gintama: The Final|  New Gintama movie.|
|             Gintama|The Amanto, alien...|
|Clannad: After Story|Clannad: After St...|
| Gintama': Enchousen|hile Gintoki Saka...|
|Gintama Movie 2: ...|"hen Gintoki appr...|
|Owarimonogatari 2...|Following an enco...|
|            Gintama.|fter joining the ...|
+--------------------+--------------------+
```

Bibliography

Setting up Pyspark on google collab:

https://www.analyticsvidhya.com/blog/2020/11/a-must-read-guide-on-how-to-work-with-pyspark-on-google-colab-for-data-scientists/

https://stackoverflow.com/questions/55240940/error-while-installing-spark-on-google-colab

Datasets:

https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020

Google Cloud Dataproc:

https://holowczak.com/getting-started-with-pyspark-on-google-cloud-platform-dataproc/9/

✓ 9s completed at 3:17 PM