AYDIN ADNAN MENDERES UNIVERSITY

ENGINEERING FACULTY

COMPUTER SCIENCE ENGINEERING DEPARTMENT

Recommendation System with Spark

CSE424 BIG DATA ANALYSIS

Student’s Name Surname:

201805073 İrem Beyza GÜL

201805066 İlker YAYALAR

201805026 Furkan GÜNDEL

201805035 Burak AYAN

Lecturer:

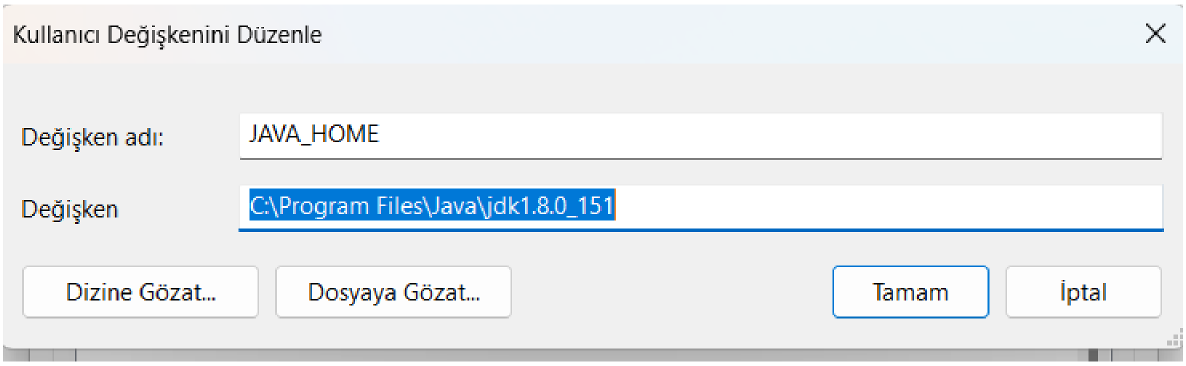
Asst. Prof. Dr.Hüseyin ABACI

Recommendation System with Spark

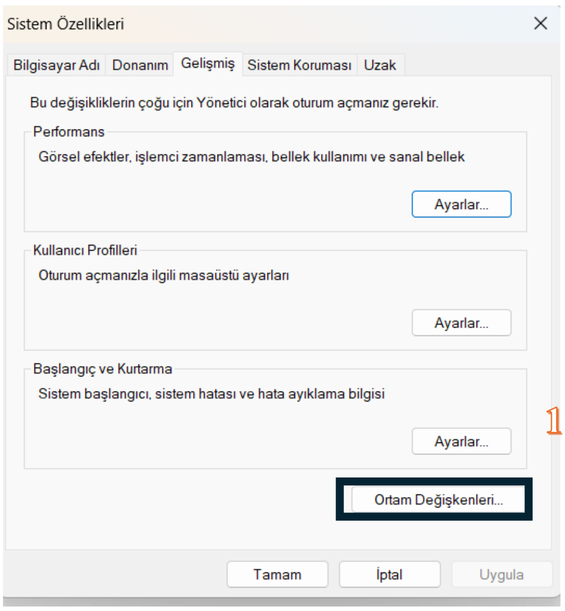
First you need to install spark. What to do for this:

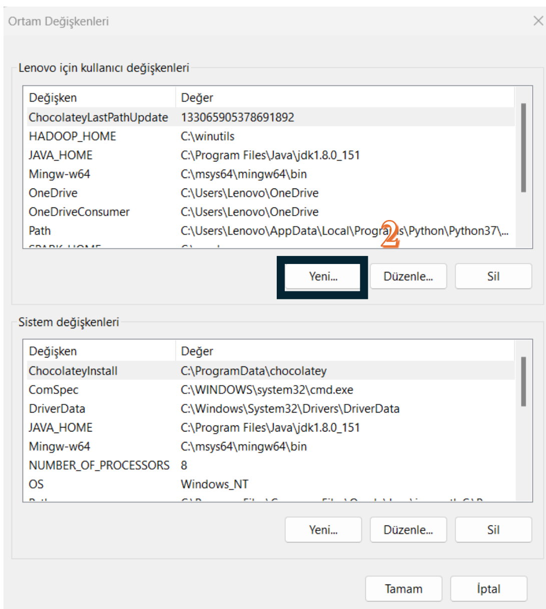
- Install JDK
- Install Pyhton
- Install Spark
- Install winutils.exe
- Install Jupyter Notebook

1- Install Download JDK (Java Development Kit) from https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Choose correct installation file for your computer/OS requirements. After installation, add following new user variable and value to environment variables:
JAVA_HOME C:\Program Files\Java\jdk1.8.0_65



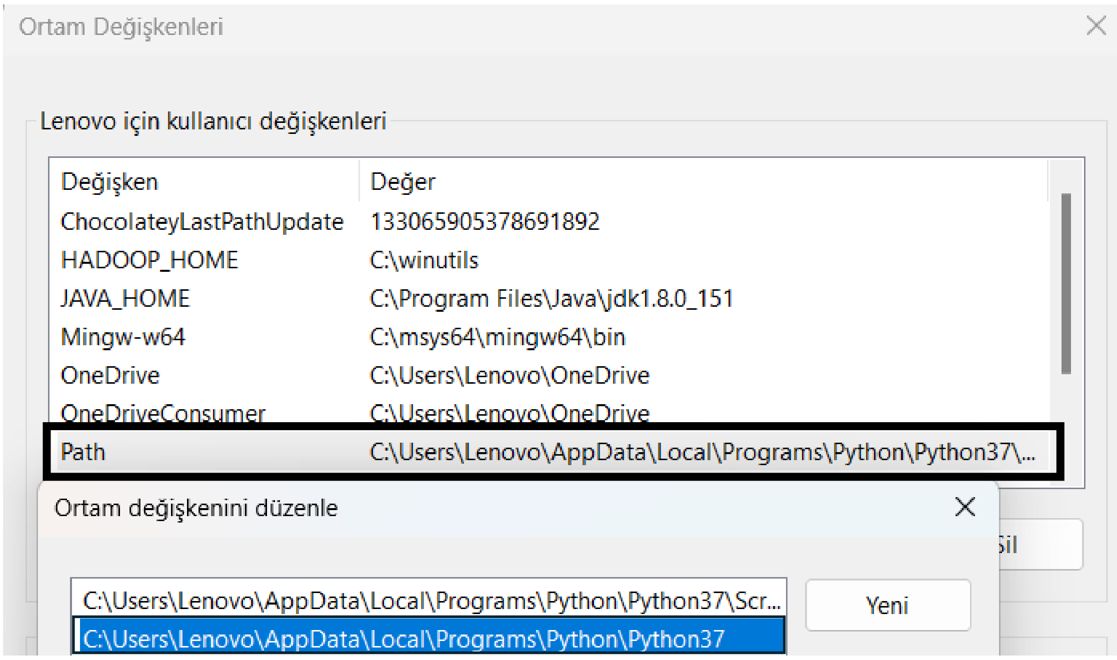
Note: How you can add these to new user variable and value to environment variables:
Search for and open your computer's system and environment variables.





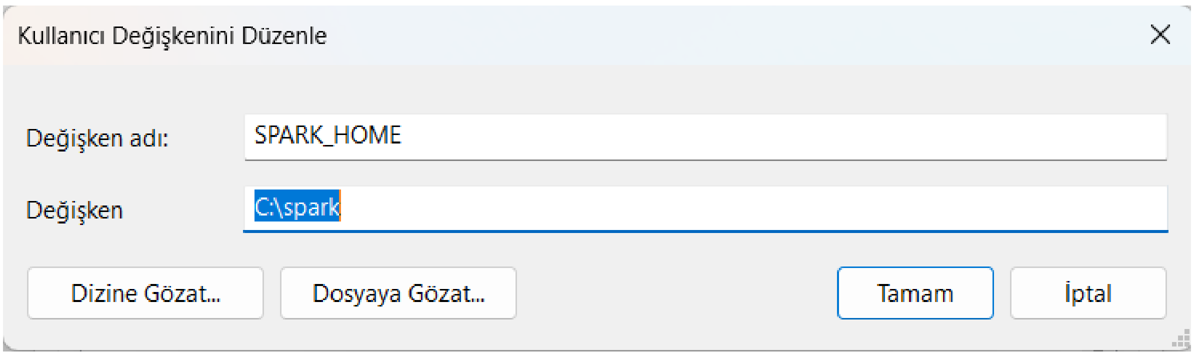
2- Install Python:

Download latest version of Python from https://www.python.org/downloads/. Choose correct installation file for your computer/OS requirements. After installation, add Python directory and Scripts directory into “path” user variable.



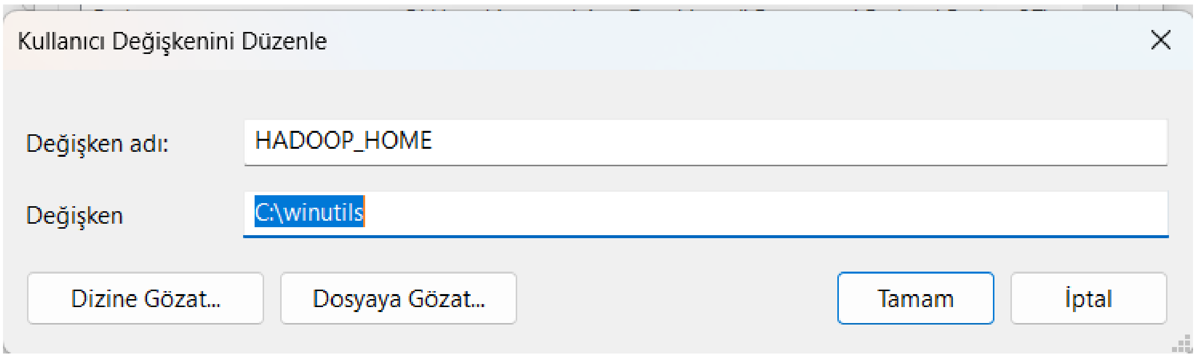
3- Install Spark:

Download latest pre built version of Spark from https://spark.apache.org/downloads.html. Choose correct installation file for your computer/OS requirements. Create a folder in C drive, named as “spark”. Extract the downloaded installation file into “C:\spark”. Add following new user variable and value to environment variables: SPARK_HOME C:\spark



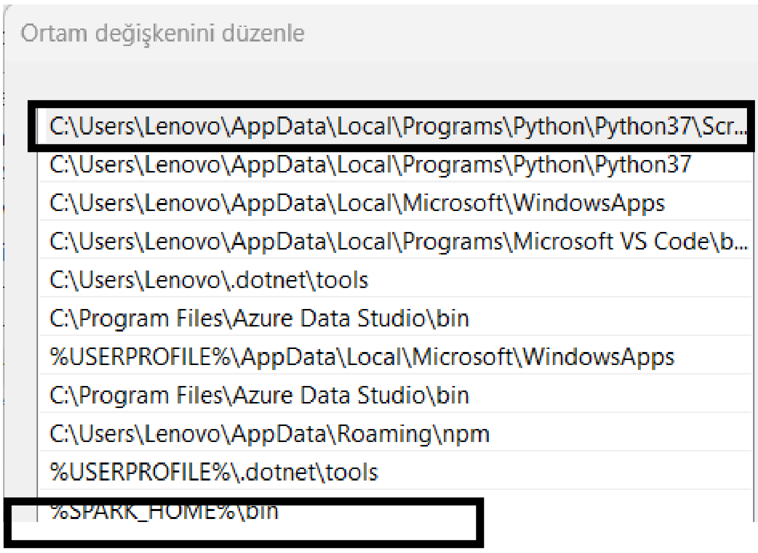
4- Install winutils.exe:

Download winutils.exe from https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin , move it into a C:\winutils\bin folder that you’ve created. Add following new user variable and value to environment variables: HADOOP_HOME C:\winutils



5- Adding Paths:

Click on “Advanced System Settings” and then the “Environment Variables” button. Add the following paths to your PATH user variable:
%SPARK_HOME%\bin
C:\Users\PC\AppData\Local\Programs\Python\Python37\Scripts



6- To install Spark, open cmd, change directory to C:\spark\bin, write pip install pyspark command. After installation change directory to C:\spark, write pyspark, you will see the following screen:

```
C:\spark>pyspark
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
24/05/25 15:44:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Python version 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019 20:34:20)
SparkSession available as 'spark'.
>>> quit()

C:\spark>SUCCESS: The process with PID 15876 (child process of PID 12584) has been terminated.
SUCCESS: The process with PID 12584 (child process of PID 17800) has been terminated.
SUCCESS: The process with PID 17800 (child process of PID 15788) has been terminated.

C:\spark>
```

7- Install Jupyter Notebook:

Change directory to C:\spark\bin in cmd, write pip install Jupyter command. After installation, write jupyter notebook on cmd. Jupyter Notebook UI will open on web browser.

```
C:\spark>cd bin

C:\spark\bin>jupyter notebook

  _   _          _      _
 | | | |_ __  __| |__ _| |_ ___
 | |_| | '_ \/ _` / _` |  _/ -_)
  \___/| .__/\__,_\__,_|\__\___|
       |_|

Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extension
s.

https://jupyter-notebook.readthedocs.io/en/latest/migrate_to_notebook7.html

Please note that updating to Notebook 7 might break some of your extensions.

[I 15:51:39.822 NotebookApp] Serving notebooks from local directory: C:\spark\bin
[I 15:51:39.822 NotebookApp] Jupyter Notebook 6.5.6 is running at:
[I 15:51:39.823 NotebookApp] http://localhost:8888/?token=4e3803059c9f6705888ec9e7c1fa8cd4817c9ffa154921a8
[I 15:51:39.823 NotebookApp]  or http://127.0.0.1:8888/?token=4e3803059c9f6705888ec9e7c1fa8cd4817c9ffa154921a8
[I 15:51:39.823 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:51:39.882 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/Lenovo/AppData/Roaming/jupyter/runtime/nbserver-1016-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=4e3803059c9f6705888ec9e7c1fa8cd4817c9ffa154921a8
     or http://127.0.0.1:8888/?token=4e3803059c9f6705888ec9e7c1fa8cd4817c9ffa154921a8
```

We completed installations. Now we can create or run the projects.

First, let's explain the dataset we used. Our dataset includes Animes and there are 5 different csv files in it, but we used 2 of these csv files (we decided that the other csvs were not necessary for the project). The names of the csvs we use are anime.csv and animelist.csv. The compressed file size of our dataset is 693 MB. The file size we extracted from the compressed file is 2.5 GB. The size of the 2 csv files we used is 1.8 GB, meaning we used 1.8 GB data file in our project. You can see information about the data set we used in the images below:

Kaggle link of our dataset: https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020/data

You can see the column names and the first few lines of our anime_csv file in the image below: (We added the column images one by one because they could not fit on a single screen.)

Type	Episodes	Aired	Premiered	Producers	Licensors	Studios	Source	Duration
TV	26	Apr 3, 1998 to Apr 24, 1999	Spring 1998	Bandai Visual	Funimation, Bandai Entertainment	Sunrise	Original	24 min. per ep.
Movie	1	Sep 1, 2001	Unknown	Sunrise, Bandai Visual	Sony Pictures Entertainment	Bones	Original	1 hr. 55 min.
TV	26	Apr 1, 1998 to Sep 30, 1998	Spring 1998	Victor Entertainment	Funimation, Geneon Entertainment USA	Madhouse	Manga	24 min. per ep.
TV	26	Jul 2, 2002 to Dec 24, 2002	Summer 2002	TV Tokyo, Bandai Visual, Dentsu, Victor Entertainment	Funimation, Bandai Entertainment	Sunrise	Original	25 min. per ep.
TV	52	Sep 30, 2004 to Sep 29, 2005	Fall 2004	TV Tokyo, Dentsu	Unknown	Toei Animation	Manga	23 min. per ep.
TV	145	Apr 6, 2005 to Mar 19, 2008	Spring 2005	TV Tokyo, Nihon Ad Systems, TV Tokyo Music, Shueisha	VIZ Media, Sentai Filmworks	Gallop	Manga	23 min. per ep.
TV	24	Apr 15, 2005 to Sep 27, 2005	Spring 2005	Genco, Fuji TV, Shueisha	VIZ Media, Discotek Media	J.C.Staff	Manga	23 min. per ep.
TV	52	Sep 11, 2002 to Sep 10, 2003	Fall 2002	Unknown	Unknown	Nippon Animation	Manga	23 min. per ep.
TV	24	Apr 17, 2004 to Feb 18, 2006	Spring 2004	OB Planning, Studio Jack	Funimation	A.C.G.T.	Manga	27 min. per ep.
TV	74	Apr 7, 2004 to Sep 28, 2005	Spring 2004	VAP, Shogakukan-Shueisha Productions, Nippon Television Network	VIZ Media	Madhouse	Manga	24 min. per ep.

Rating	Ranked	Popularity	Members	Favorites	Watching	Completed	On-Hold	Dropped	Plan to Watch	Score-10	Score-9	Score-8	Score-7	Score-6	Score-5	Score-4	Score-3	Score-2	Score-1
R - 17+ (violence & profanity)	28.0	39	1251960	61971	105808	718161	71513	26678	329800	2291700	1821260	1316250	623300	206880	89040	31840	13570	7410	15800
R - 17+ (violence & profanity)	159.0	518	273145	1174	4143	208333	1935	770	57964	300430	492010	495050	226320	58050	18770	5770	2210	1090	3790
PG-13 - Teens 13 or older	266.0	201	558913	12944	29113	343492	25465	13925	146918	502290	756510	861420	494320	153760	58380	19650	6640	3160	5330
PG-13 - Teens 13 or older	2481.0	1467	94683	587	4300	46165	5121	5378	33719	21820	48060	101280	116180	57090	29200	10830	3530	1640	1310
PG - Children	3710.0	4369	13224	18	642	7314	766	1108	3394	3120	5290	12420	17130	10680	6340	2650	830	500	270
PG-13 - Teens 13 or older	604.0	1003	148259	2066	13907	78349	14228	11573	30202	92260	149040	228110	167340	62060	26210	7950	3360	1400	1510
PG-13 - Teens 13 or older	468.0	687	214499	4101	11909	81145	11901	11026	98518	118290	163090	200080	130620	55740	31480	13390	4840	2780	3210
PG-13 - Teens 13 or older	1317.0	3612	20470	231	817	13778	828	1168	3879	11230	17770	31020	30750	12860	6020	2180	880	310	320
PG-13 - Teens 13 or older	360.0	1233	117929	979	6082	90967	3053	1356	16471	109480	158200	223790	129120	38740	12360	3690	970	480	2590
R+ - Mild Nudity	30.0	169	614100	29436	64648	214491	47488	23008	264465	773500	606520	434590	220450	88610	43810	20860	8820	5930	11770

In the code, we first printed the desired computer and IP address, then created a sparksession.

```
%config IPCompleter.greedy=True
import findspark
findspark.init()
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
import socket
spark = SparkSession.builder.getOrCreate()
conf = spark.sparkContext.getConf()
computer_info = (socket.gethostname(), socket.gethostbyname(socket.gethostname()), conf.getAll())

print("Bilgisayar Adı:", computer_info[0])
print("IP Adresi:", computer_info[1])
print("Yapılandırma:", computer_info[2])
```

```
Bilgisayar Adı: DESKTOP-3DNAMQC
IP Adresi: 192.168.1.6
Yapılandırma: [('spark.driver.port', '57916'), ('spark.rdd.compress', 'True'), ('spark.serializer.objectStreamReset', '100'),
('spark.master', 'local[*]'), ('spark.executor.id', 'driver'), ('spark.submit.deployMode', 'client'), ('spark.ui.showConsolePro
gress', 'true'), ('spark.app.name', 'pyspark-shell'), ('spark.driver.host', 'DESKTOP-3DNAMQC'), ('spark.app.id', 'local-1716634
528825')]
```

Picture 1

We first read our anime_csv file in the code, then we checked whether there were missing values, null or unknown values in our data, and if there were, we filled them in correctly.

```
spark = SparkSession.builder \
    .appName("Anime Recommendation System") \
    .getOrCreate()

anime_df = spark.read.csv("C:/spark/bin/data/anime.csv", header=True, inferSchema=True)
animelist_df = spark.read.csv("C:/spark/bin/data/animelist.csv", header=True, inferSchema=True)
print("Number of Animes:", anime_df.count())
anime_rdd = anime_df.rdd
anime_fields = anime_rdd.map(lambda row: "|".join([str(item) for item in row]))
print(anime_fields.first())
score_df = anime_df.select("Score")
score_df.show()
```

```
Number of Animes: 17562
1|Cowboy Bebop|8.78|Action, Adventure, Comedy, Drama, Sci-Fi, Space|Cowboy Bebop|カウボーイビバップ|TV|26|Apr 3, 1998 to Apr 24,
1999|Spring 1998|Bandai Visual|Funimation, Bandai Entertainment|Sunrise|Original|24 min. per ep.|R - 17+ (violence & profanity)
|28.0|39|1251960.0|61971|105808|718161|71513|26678|329800|229170.0|182126.0|131625.0|62330.0|20688.0|8904.0|3184.0|1357.0|741.0
|1580.0
+-----+
|Score|
+-----+
| 8.78|
| 8.39|
| 8.24|
| 7.27|
| 6.98|
| 7.95|
| 8.06|
```

Picture 2

```
from pyspark.sql.functions import col, when, mean
from pyspark.sql.types import IntegerType, DoubleType, StringType
def replace_unknown(df):
    for column in df.columns:
        # Get the column data type
        data_type = df.schema[column].dataType

        if isinstance(data_type, (IntegerType, DoubleType)):
            # Replace 'unknown' with None and cast to correct type
            df = df.withColumn(column, when(col(column) == 'Unknown', None).otherwise(col(column).cast(data_type)))
            # Fill missing numeric values with the mean
            mean_value = df.select(mean(col(column))).collect()[0][0]
            df = df.na.fill({column: mean_value})
        elif isinstance(data_type, StringType):
            # Replace 'unknown' with 'Unknown'
            df = df.withColumn(column, when(col(column) == 'Unknown', '0').otherwise(col(column)))

    return df


anime_df = replace_unknown(anime_df)
animelist_df=replace_unknown(animelist_df)
```

```
def check_unknown_values(df):
    for column in df.columns:
        unknown_count = df.filter(col(column) == 'Unknown').count()
        if unknown_count > 0:
            print(f"Column '{column}' has {unknown_count} 'unknown' values.")
        else:
            print(f"Column '{column}' has no 'unknown' values.")


print("Checking 'anime_list_df' for 'unknown' values:")
check_unknown_values(animelist_df)


print("\nChecking 'anime_df' for 'unknown' values:")
check_unknown_values(anime_df)
```

```
Checking 'anime_list_df' for 'unknown' values:
Column 'user_id' has no 'unknown' values.
Column 'anime_id' has no 'unknown' values.
Column 'rating' has no 'unknown' values.
```

Picture 3

```
def check_null_empty_values(df):
    for column in df.columns:
        null_count = df.filter(col(column).isNull()).count()
        empty_count = df.filter(col(column) == '').count()
        total_null_empty_count = null_count + empty_count
        if total_null_empty_count > 0:
            print(f"Column '{column}' has {null_count} null values and {empty_count} empty values.")
        else:
            print(f"Column '{column}' has no null or empty values.")

print("\nChecking for null or empty values in 'anime_list_df':")
check_null_empty_values(animelist_df)


print("\nChecking for null or empty values in 'anime_df':")
check_null_empty_values(anime_df)
```

```
Checking for null or empty values in 'anime_list_df':
Column 'user_id' has no null or empty values.
Column 'anime_id' has no null or empty values.
Column 'rating' has no null or empty values.
Column 'watching_status' has no null or empty values.
Column 'watched_episodes' has no null or empty values.

Checking for null or empty values in 'anime_df':
Column 'MAL_ID' has no null or empty values.
Column 'Name' has no null or empty values.
```

Picture 4

Afterwards, we plotted the graphs of the important columns for us in our Anime_csv file, these columns are:

Aired column: this column contained information in the following format, Apr 3, 1998 to Apr 24, 1999, the start and end date of the anime. However, some anime did not have a start or end date. We filled the places without a start date with Jan 1, 1900, and the places without an end date with Dec 31, 2024. After this, we calculated how many years the anime lasted by taking the start and end year information and graphed it.

```
import re

# 'Aired' sütunundaki her bir değerin istenilen formatta olup olmadığını kontrol eden fonksiyon
def check_aired_date_format(df, column_name):
    # İstenilen tarih formatı için bir düzenli ifade (regular expression)
    date_pattern = r'^\w{3}\s\d{1,2},\s\d{4}\sto\s\w{3}\s\d{1,2},\s\d{4}$'

    # Düzenli ifadeyi kullanarak her bir değeri kontrol et
    invalid_dates = df.filter(~col(column_name).rlike(date_pattern)).count()

    if invalid_dates > 0:
        print(f"Column '{column_name}' contains {invalid_dates} entries with invalid date format.")
    else:
        print(f"All entries in column '{column_name}' have valid date format.")

# 'Aired' sütunundaki tarih formatını kontrol et
check_aired_date_format(anime_df, "Aired")
```

```
Column 'Aired' contains 9825 entries with invalid date format.
```

Picture 5

```python
from pyspark.sql.functions import col, lit, regexp_extract, when, expr, udf
from pyspark.sql.types import StringType
import re

date_pattern = r'(\w{3}\s\d{1,2},\s\d{4})\s?(?:to\s(\w{3}\s\d{1,2},\s\d{4}))?'

# Function to process aired dates
def process_aired_dates(row):
    aired_date = row['Aired']
    match = re.match(date_pattern, aired_date)
    if match:
        start_date = match.group(1)
        end_date = match.group(2)
    else:
        start_date = ''
        end_date = ''
    if start_date == '':
        start_date = 'Jan 1, 1900'
    if end_date == '':
        end_date = 'Dec 31, 2024'
    return (start_date, end_date)

# Define the User Defined Function (UDF) for processing aired dates
process_aired_udf = udf(process_aired_dates, StringType())

# Process the 'Aired' column and add new columns for start and end dates
anime_df = anime_df.withColumn("Start_Date", regexp_extract(col("Aired"), date_pattern, 1)) \
                   .withColumn("End_Date", regexp_extract(col("Aired"), date_pattern, 2))

# Fill missing values
anime_df = anime_df.withColumn("Start_Date", when(col("Start_Date") == "", lit("Jan 1, 1900")).otherwise(col("Start_Date"))) \
                   .withColumn("End_Date", when(col("End_Date") == "", lit("Dec 31, 2024")).otherwise(col("End_Date")))

anime_df = anime_df.withColumn("Start_Year", expr("substring(Start_Date, -4)")) \
                   .withColumn("End_Year", expr("substring(End_Date, -4)")) \
                   .withColumn("Duration", col("End_Year").cast("int") - col("Start_Year").cast("int"))

# Sonuçları göster
anime_df.select("Aired", "Start_Date", "End_Date", "Duration").show(truncate=False)
```

```
+--------------------------+------------+------------+--------+
|Aired                     |Start_Date  |End_Date    |Duration|
+--------------------------+------------+------------+--------+
|Apr 3, 1998 to Apr 24, 1999 |Apr 3, 1998 |Apr 24, 1999|1       |
|Sep 1, 2001               |Sep 1, 2001 |Dec 31, 2024|23      |
|Apr 1, 1998 to Sep 30, 1998 |Apr 1, 1998 |Sep 30, 1998|0       |
|Jul 2, 2002 to Dec 24, 2002 |Jul 2, 2002 |Dec 24, 2002|0       |
|Sep 30, 2004 to Sep 29, 2005|Sep 30, 2004|Sep 29, 2005|1       |
```

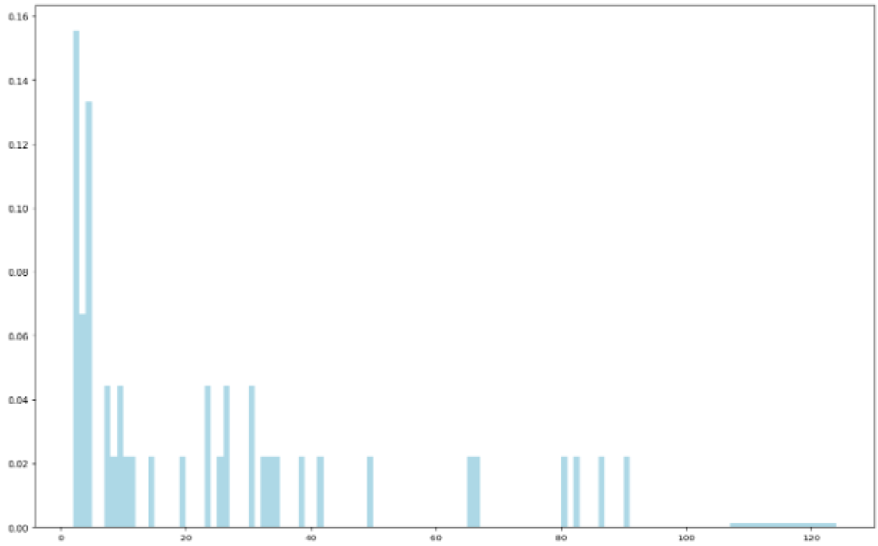
Picture 6

```python
movie_ages = anime_df.select("Start_Year").rdd.map(lambda row: 2024 - int(row["Start_Year"])).countByValue()
values = list(movie_ages.values())
bins = list(movie_ages.keys())
print("Ages:", bins)
print("Frequencies:", values)
```

```
Ages: [26, 23, 22, 20, 19, 25, 21, 29, 27, 28, 36, 31, 24, 45, 35, 33, 39, 38, 30, 32, 34, 46, 51, 18, 37, 40, 42, 47, 41, 44,
48, 56, 43, 124, 17, 53, 57, 49, 62, 59, 55, 50, 60, 16, 52, 54, 58, 61, 79, 15, 12, 91, 81, 14, 93, 92, 90, 64, 66, 13, 65, 7
7, 95, 107, 94, 88, 89, 86, 67, 106, 97, 96, 87, 84, 80, 69, 68, 63, 8, 11, 5, 83, 10, 9, 7, 6, 82, 98, 76, 3, 4, 2]
Frequencies: [199, 319, 314, 337, 360, 213, 326, 173, 181, 189, 145, 178, 197, 66, 173, 176, 111, 128, 182, 193, 184, 49, 26, 4
17, 149, 90, 80, 41, 86, 65, 38, 26, 82, 2099, 432, 30, 25, 32, 9, 19, 30, 33, 14, 429, 23, 34, 23, 10, 1, 507, 672, 8, 4, 511,
4, 2, 9, 2, 3, 615, 4, 2, 4, 11, 4, 1, 7, 2, 3, 7, 2, 2, 1, 3, 1, 1, 1, 4, 874, 685, 724, 1, 822, 772, 904, 872, 2, 1, 1, 168,
602, 1]
```

```python
import matplotlib.pyplot as plt
import numpy as np
bins = np.sort(bins)
print(bins)
plt.hist(values, bins=bins, color='lightblue', density=True)
fig = plt.gcf()
fig.set_size_inches(16, 10)
plt.show()
```

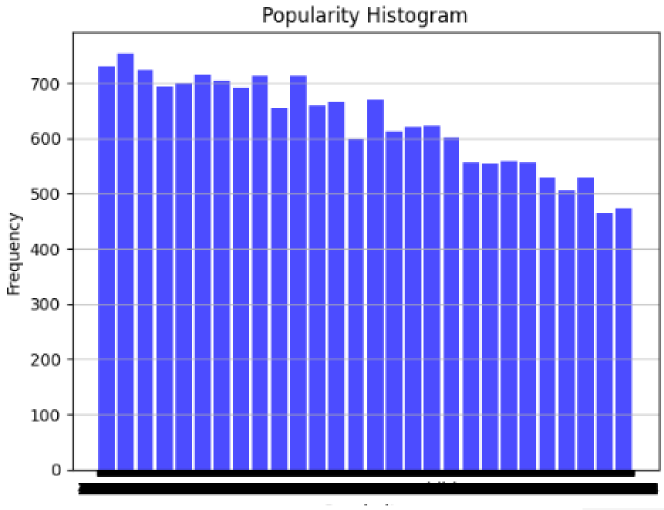
```
[  2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37
  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63  64  65  66  67  68  69  76  77  79  80
  81  82  83  84  86  87  88  89  90  91  92  93  94  95  96  97  98 106
 107 124]
```



Picture 7

```python
popularity_df = anime_df.select("Popularity")
columns = popularity_df.columns
popularity_df = popularity_df.toPandas()
values = popularity_df.iloc[:, 0].tolist()
bins = columns
popularity_df['Popularity'] = popularity_df['Popularity'].replace('Unknown', 0)
print(popularity_df)
values = popularity_df['Popularity'].tolist()
plt.hist(values, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Popularity')
plt.ylabel('Frequency')
plt.title('Popularity Histogram')
plt.show()
```

```
       Popularity
0              39
1             518
2             201
3            1467
4            4369
...           ...
17557       13116
17558       17562
17559       17558
17560       17565
17561       17563

[17562 rows x 1 columns]
```



Picture 8

Since the Score, Popularity, Episodes, Ranked columns contain numeric values, we directly pulled the data from these columns and graphed them. As an example, Popularity is shown in the image:

There were different string values in the Type and Source columns, so we calculated how many anime there were for a value and graphed them accordingly. You can see the code and graph of the Type column below as an example.

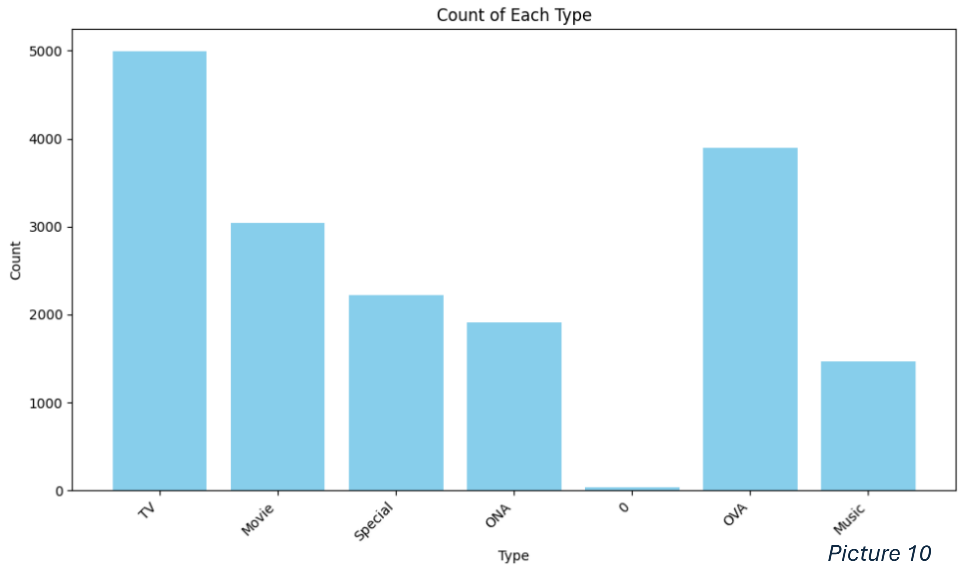
```python
type_rdd = anime_df.select("type").rdd.map(lambda row: (row['type'], 1))
type_counts = type_rdd.reduceByKey(lambda a, b: a + b)
result = type_counts.collect()
filtered_result = [item for item in result if re.match(r'^[a-zA-Z0-9\s]+$', item[0])]
for type, count in filtered_result:
    print(f"{type}: {count}")
```

```
TV: 4994
Movie: 3039
Special: 2218
ONA: 1907
0: 37
OVA: 3893
Music: 1469
```

Picture 9

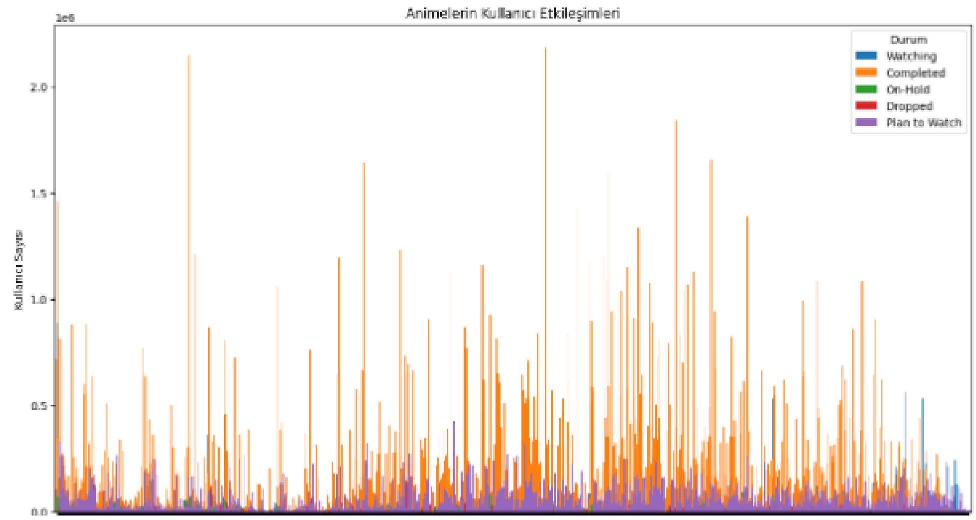
Type column graphic:

```python
types = [item[0] for item in filtered_result]
counts = [item[1] for item in filtered_result]
plt.figure(figsize=(10, 6))
plt.bar(types, counts, color='skyblue')
plt.xlabel('Type')
plt.ylabel('Count')
plt.title('Count of Each Type')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Picture 10

```
watching_list = anime_df.select("Watching").rdd.flatMap(lambda x: x).collect()
completed_list = anime_df.select("Completed").rdd.flatMap(lambda x: x).collect()
on_hold_list = anime_df.select("On-Hold").rdd.flatMap(lambda x: x).collect()
dropped_list = anime_df.select("Dropped").rdd.flatMap(lambda x: x).collect()
plan_to_watch_list = anime_df.select("Plan to Watch").rdd.flatMap(lambda x: x).collect()

# Tek boyutlu listeleri bir sözlük yapısına yerleştir
data = {
    "Watching": watching_list,
    "Completed": completed_list,
    "On-Hold": on_hold_list,
    "Dropped": dropped_list,
    "Plan to Watch": plan_to_watch_list
}

# Pandas DataFrame oluştur
df_prepared = pd.DataFrame(data)

# Grafik ayarları
bar_width = 30  # Çubuk kalınlığı
df_prepared.plot(kind='bar', figsize=(15, 8), width=bar_width)

plt.xlabel('Anime Index')
plt.ylabel('Kullanıcı Sayısı')
plt.title('Animelerin Kullanıcı Etkileşimleri')
plt.xticks(rotation=0)
plt.legend(title='Durum')
plt.show()
```



Picture 11

In the columns Watching, Completed, On-Hold, Dropped, Plan to Watch, which contain the viewing status of anime, information such as how many users watched the anime or planned to watch it is given. We drew a graph according to these categories, anime and number of users.

After our graphic drawings were completed, we removed the columns that would not be useful to us in our file.

You can find our calculation and visualization codes for anime_csv and the rest of our results in our html file.

user_id	anime_id	rating	watching_status	watched_episodes
0	67	9	1	1
0	6702	7	1	4
0	242	10	1	4
0	4898	0	1	1
0	21	10	1	0
0	24	9	1	5
0	2104	0	1	4
0	4722	8	1	4
0	6098	6	1	2
0	3125	9	1	29
0	481	10	1	79
0	68	6	2	23
0	1689	6	2	3
0	2913	6	2	40
0	1250	7	2	26
0	356	9	2	24
0	121	9	2	51
0	430	9	2	1
0	1829	7	2	1
0	1571	10	2	25

Picture 12

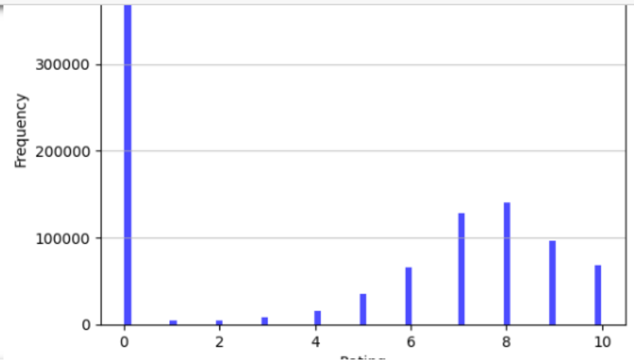
Now we have arrived at our animelist.csv file, which we had read in the beginning. The column names of this dataset are given in the image below.

We made calculations such as the number of users and the number of anime in the Animelist and printed them on the screen.

```
animelist_df = animelist_df.drop("Watching Status")

num_users = animelist_df.select("user_id").distinct().count()
num_anime = animelist_df.select("anime_id").distinct().count()
num_ratings = animelist_df.select("rating").distinct().count()


# Sonuçları yazdır
print('Number of Users: {}, Number of Anime: {}, Number of Ratings: {} '.format(num_users, num_anime, num_ratings))

Number of Users: 325770, Number of Anime: 17562, Number of Ratings: 11
```

Picture 13

```python
anime_list_rdd = animelist_df.rdd
anime_list_fields = anime_list_rdd.map(lambda row: "|".join([str(item) for item in row]))
schema = animelist_df.schema
column_names = [field.name for field in schema]
print(column_names)
print(anime_list_fields.first())
import warnings
warnings.filterwarnings("ignore")
total_rows = animelist_df.count()
print("Veri kümesinde toplam satır sayısı:", total_rows)
animelist_df_first_10000 = animelist_df.limit(1000000)
rating_df = animelist_df_first_10000.select("Rating")

rating_df = rating_df.toPandas()
rating_df['Rating'] = rating_df['Rating'].replace('Unknown', 0)
values = rating_df['Rating'].tolist()
columns = rating_df.columns
plt.hist(values, bins='auto', color='blue', alpha=0.7, rwidth=0.85)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.title('Rating Histogram')
plt.show()
```



Picture 14

The rating column in Animelist is the most important column for us, this column will be used to calculate the rating (in ALS). First, we graphed this rating column, but I should point out that since the dataset is very large, the entire rating column cannot be graphed, so the first 1000000 rows are taken into graphs. The process was done here.

Our data operations on two datasets are completed. Now we will combine our two dataset files with join. However, first, since our anime dataset is named MAL_ID instead of anime_id, we convert MAL_ID to anime_id. Then we combine these two datasets according to anime_id.

```
anime_df = anime_df.withColumnRenamed('MAL_id', 'anime_id')
anime_df.show()
anime_df = anime_df.drop("Start_Date", "End_Date" )
son_data = anime_df.join(animelist_df, on=["anime_id"], how="inner")
son_data.show(15)
54|1160651.0|    71308|    66549|    815938|   35566|   20358|          222240|186797.0|168056.0|157596.0| 93532.0| 38385.0|17571.0
| 9686.0| 4408.0| 3263.0| 3714.0| Oct 4, 1995|Mar 27, 1996|      1995|     1996|          1995|
+--------+--------------------+-----+--------------------+----+--------+--------------------+-----------+-----------+------+
----------+---------+---------+--------+---------+-------+-------+--------------+--------+--------+--------+--------+--------
+-------+-------+-------+-------+-------+------------+------------+----------+--------+------------+
only showing top 20 rows


+--------+--------------------+-----+--------------------+----+--------+--------------------+-----------+-----------+------
+----------+---------+---------+---------+---------+-------+-------+--------------+--------+--------+--------+--------+------
+-------+-------+-------+-------+-------+----------+--------+-------------+-------+------+--------------+---------------+
|anime_id|                Name|Score|              Genres|Type|Episodes|               Aired|  Premiered|     Source|Ranked
|Popularity|  Members|Favorites|Watching|Completed|On-Hold|Dropped|Plan to Watch|Score-10| Score-9| Score-8| Score-7|Score-6
|Score-5|Score-4|Score-3|Score-2|Score-1|Start_Year|End_Year|Premired Year|user_id|rating|watching_status|watched_episodes|
+--------+--------------------+-----+--------------------+----+--------+--------------------+-----------+-----------+------
+----------+---------+---------+---------+---------+-------+-------+--------------+--------+--------+--------+--------+------
+-------+-------+-------+-------+-------+----------+--------+-------------+-------+------+--------------+---------------+
|      67|Basilisk: Kouga N...| 7.58|Action, Adventure...|  TV|      24|Apr 13, 2005 to S...|Spring 2005|      Manga|1325.0
|       940| 156661.0|     1467|    6964|    82153|   6380|   6957|         54207|  6677.0| 11140.0| 19570.0| 17421.0| 7816.0
```

Picture 15

We finalized our dataset, now we made some calculations using this final dataset and then ran our ALS algorithm.

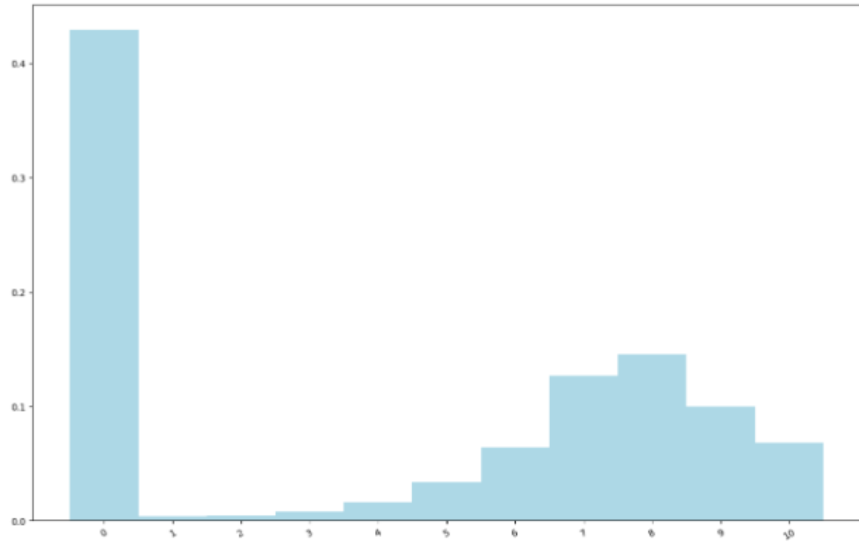
```python
from pyspark.sql.functions import col
rating_fields = son_data.select(col("user_id"), col("anime_id"), col("rating")).rdd.map(lambda row: "|".join([str(item) for ite
ratings = son_data.select("rating").rdd.map(lambda row: int(row['rating']))
stats = ratings.stats()
print(stats)
```

```
(count: 74963444, mean: 4.287202412951999, stdev: 3.9441466964476537, max: 10.0, min: 0.0)
```

Picture 16

```python
count_by_rating = ratings.countByValue()
x_axis = list(count_by_rating.keys())
y_axis = list(count_by_rating.values())
y_axis_normed = np.array(y_axis) / sum(y_axis)
sorted_indices = np.argsort(x_axis)
x_axis_sorted = np.array(x_axis)[sorted_indices]
y_axis_sorted = y_axis_normed[sorted_indices]
```

```python
pos = np.arange(len(x_axis_sorted))
width = 1.0
plt.bar(pos, y_axis_sorted, width, color='lightblue')
plt.xticks(pos, x_axis_sorted, rotation=30)
fig = plt.gcf()
fig.set_size_inches(16, 10)
plt.show()
```



Picture 17

Before running our ALS algorithm, we received an error that there were null and empty parts in the merged data string, so we re-ran the code we wrote to fill these sections correctly for our merged dataset. And our data set was ready for ALS.

We also added a check point before the ALS section and saved it as a textfile. In this way, we tried to free up space in RAM.

CheckPoint

```python
import os

checkpoint_dir = "/path/to/local/checkpoint"

if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
```

```python
import pyspark.sql

dataframe_names = [(name, var) for name, var in globals().items() if isinstance(var, pyspark.sql.DataFrame)]

for name, _ in dataframe_names:
    print(name)
```

```
anime_df
animelist_df
score_df
best_anime
popular_genres
animelist_df_first_10000
son_data
son_data_filled
```

```python
for name, df in dataframe_names:
    df.write.format("parquet").mode("overwrite").save(f"{checkpoint_dir}/{name}")
```

```python
from pyspark.rdd import RDD
rdd_names = [(name, var) for name, var in globals().items() if isinstance(var, RDD)]

for name, _ in rdd_names:
    print(name)
```

```
anime_rdd
anime_fields
type_rdd
type_counts
source_rdd
source_counts
animelist_rdd
animelist_data
animelist_fields
anime_list_rdd
anime_list_fields
rating_fields
ratings
```

```python
import shutil
import os

rdd_names = [(name, var) for name, var in globals().items() if isinstance(var, RDD)]
for name, rdd in rdd_names:
    path = f"{checkpoint_dir}/{name}_checkpoint"
    if os.path.exists(path):
        shutil.rmtree(path)
    rdd.saveAsTextFile(path)
```

Picture 18

ALS PART of the CODE

Before ALS, we converted our dataset to rdd and then split our dataset into 70 by 30 test and training sets.

Later, while running our ALS algorithm, we got RAM error (on all group members), so we reduced our dataset to 1GB as we thought it was important to handle big data and chose to reduce the sorting and iteration specified. However, we still could not run 1 GB of data on our computers due to RAM, so we used the first 10 million data of our data set. We set our rank and iteration values to 10,15,20.

After separating them into test and train, we checked our test and train sets, then we defined rank, iteration and lambda. After these, we ran the codes of our ALS algorithm.

Our ALS algorithm created models for each value, for the models we printed Item vector, Cosine Similarity, Predicted Rating, Top 19 recommendations from 10 users, Top 5 Users for item X, Top 10 Anime for user X.

After these, we printed the Rates and Estimates, Predictions and Test Data information on the screen, as we printed our actual values and predicted values on the screen, and we were able to compare them according to their outputs.

After these procedures, we made MSE and RMSE calculations for each model and completed the process.

Finally, we calculated the similarity and printed it on the screen.

You can find screenshots of the code and outputs of all these operations we performed below and in our code file.

We then graphed and showed the results of different models for RMSE and MSE.

Then, we determined the best model according to these MSE and RMSE values.

🌟 ALS 🌟 ¶

```
rdd = sc.textFile("C:/Users/ilker/Desktop/Anime/animelist.csv")
```

```
header = rdd.first()
```

```
data_rdd = rdd.filter(lambda line: line != header)
```

```
data_rdd.take(10)
```

```
['0,67,9,1,1',
 '0,6702,7,1,4',
 '0,242,10,1,4',
 '0,4898,0,1,1',
 '0,21,10,1,0',
 '0,24,9,1,5',
 '0,2104,0,1,4',
 '0,4722,8,1,4',
 '0,6098,6,1,2',
 '0,3125,9,1,29']
```

```
rating_fields = data_rdd.map(lambda line:line.split(','))
ratings_rdd = rating_fields.map(lambda field : int(field[2]))
```

```
ratings_rdd.stats()
```

```
(count: 109224747, mean: 4.245716641486047, stdev: 3.9128884433444586, max: 10.0, min: 0.0)
```

Picture 19

```
raw_rating = data_rdd.map(lambda x:x.split(',')[:3])
raw_rating.take(5)
```

```
[['0', '67', '9'],
 ['0', '6702', '7'],
 ['0', '242', '10'],
 ['0', '4898', '0'],
 ['0', '21', '10']]
```

```
from pyspark.mllib.recommendation import ALS
from pyspark.mllib.recommendation import Rating

ratingsforAls = raw_rating.map(lambda x: Rating(int(x[0]), int(x[1]), float(x[2])))
ratingsforAls.take(10)
```

```
[Rating(user=0, product=67, rating=9.0),
 Rating(user=0, product=6702, rating=7.0),
 Rating(user=0, product=242, rating=10.0),
 Rating(user=0, product=4898, rating=0.0),
 Rating(user=0, product=21, rating=10.0),
 Rating(user=0, product=24, rating=9.0),
 Rating(user=0, product=2104, rating=0.0),
 Rating(user=0, product=4722, rating=8.0),
 Rating(user=0, product=6098, rating=6.0),
 Rating(user=0, product=3125, rating=9.0)]
```

Picture 20

```
%%time
limited_ratings = ratingsforAls.take(10000000)
```

```
CPU times: total: 8.36 s
Wall time: 35.1 s
```

```
%%time
limited_ratings_rdd = sc.parallelize(limited_ratings)
```

```
CPU times: total: 15.1 s
Wall time: 17.5 s
```

```
from pyspark.mllib.recommendation import ALS
from pyspark.mllib.evaluation import RegressionMetrics
from pyspark.sql import SparkSession
from pyspark.mllib.recommendation import Rating
```

```
train_data, test_data = limited_ratings_rdd.randomSplit([0.7, 0.3], seed=5066)
```

```
print(train_data.take(5))
print(test_data.take(5))
```

```
[Rating(user=0, product=67, rating=9.0), Rating(user=0, product=6702, rating=7.0), Rating(user=0, product=242, rating=10.0), Rating(user=0, product=21, ra
ting=10.0), Rating(user=0, product=2104, rating=0.0)]
[Rating(user=0, product=4898, rating=0.0), Rating(user=0, product=24, rating=9.0), Rating(user=0, product=3125, rating=9.0), Rating(user=0, product=68, ra
ting=6.0), Rating(user=0, product=1250, rating=7.0)]
```

```
from math import sqrt
#Models Load
ranks = [10,15,20]
iterations = [10,15,20]
lambdas = [0.01,0.1]
models = []
evaluations = []
```

Picture 21

```
def anime_for_user(user: int):
    anime_for_user_rdd = limited_ratings_rdd.filter(lambda x: x.user == user)
    anime_for_user_list = anime_for_user_rdd.collect()
    for rating in anime_for_user_list[:10]:
        print(f"User: {rating.user}, Anime: {rating.product}, Rating: {rating.rating}")
    return anime_for_user_list
```

```
def cosineSimilarity(item_id, a, b):
    dot = np.dot(a,b)
    norma = np.linalg.norm(a)
    normb = np.linalg.norm(b)
    cos = dot/(norma * normb)
    return item_id, cos
```

Picture 22

```
%%time
import itertools

param_combinations = itertools.product(ranks, iterations, lambdas)
process_count = 0
for rank, iteration, lambda_ in param_combinations:
    model = ALS.train(train_data, rank=rank, iterations=iteration, lambda_=lambda_,seed=5066)
    process_count += 1
    item_id = 24
    item_vector = model.productFeatures().lookup(item_id)[0]
    print(f"\n Item vector for item_id {item_id}: {item_vector}")

    cosineSimilarity(item_id,item_vector,item_vector)


    predicted_rating = model.predict(0, 24)
    print(f"\n Predicted rating for user 0 and item 24: {predicted_rating}")

    userId = 0
    top_10_recs = model.recommendProducts(userId, 10)
    print(f"\n Top 10 recommendations for user {userId}:")
    for i in top_10_recs:
        print(i)

    top5_item = model.recommendUsers(24, 5)
    print(f"\n Top 5 users for item 24:")
    for i in top5_item:
        print(i)

    anime_for_userx = limited_ratings_rdd.keyBy(lambda x: x.user)
    print(anime_for_userx.take(10))

    animeForUser = anime_for_user(0)
    animeForUser.sort(reverse=True, key=lambda x: x.rating)

    print("\nTop 10 anime for user 0:")
    for rating in animeForUser[:10]:
        print(f" User: {rating.user}, Anime: {rating.product}, Rating: {rating.rating}")

    predictions = model.predictAll(test_data.map(lambda x: (x[0], x[1]))).map(lambda r: ((r[0], r[1]), r[2]))
    rates_and_preds = test_data.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)

    print("Rates and Predictions (Sample):")
    print(rates_and_preds.take(10))

    predictions_list = predictions.collect()
    print("Predictions (Sample):")
    for rating in predictions_list:
        user_id, anime_id = rating[0]
        rating_value = rating[1]
        if user_id == 0:
            print(f" User: {user_id}, Anime: {anime_id}, Rating: {rating_value}")

    print("\nTest Data (Sample):")
    for rating in test_data.take(10):
        print(f" User: {rating.user}, Anime: {rating.product}, Rating: {rating.rating}")

    MSE = rates_and_preds.map(lambda tup: (tup[1][0] - tup[1][1]) ** 2).mean()
    RMSE = sqrt(MSE)
    print("Mean Squared Error (MSE) = ", MSE)
    print("Root Mean Squared Error (RMSE) = ", RMSE)

    evaluations.append(((rank, iteration, lambda_), MSE, RMSE))
    last_evaluation = evaluations[-1]
    print(f"\n Son eleman - Rank: {last_evaluation[0][0]}, Iteration: {last_evaluation[0][1]}, Lambda: {last_evaluation[0][2]}, MSE: {last_evaluation[1]}

    models.append((model, rank, iteration, lambda_))
    print(f"\n {process_count}. process is successful\n")
```

Picture 23

ALS CODES

Output of Code

```
 Item vector for item_id 24: array('d', [-0.45050886273384094, 0.5134439468383789, -0.1632067859172821, 0.8264790177345276, 0.26409047842025757, -1.138
6926174163818, -0.5240490436553955, 0.10144299268722534, 0.02503708004951477, -0.05589338764548302])

 Predicted rating for user 0 and item 24: 2.614560772170855

 Top 10 recommendations for user 0:
Rating(user=0, product=29291, rating=41.774433252583236)
Rating(user=0, product=44086, rating=38.944949962304925)
Rating(user=0, product=29936, rating=38.17166655893764)
Rating(user=0, product=6855, rating=35.80456239945434)
Rating(user=0, product=5002, rating=34.38761101203131)
Rating(user=0, product=33238, rating=33.78274720556394)
Rating(user=0, product=34115, rating=32.36392018157503)
Rating(user=0, product=10562, rating=32.17010733498465)
Rating(user=0, product=39204, rating=32.09720066913147)
Rating(user=0, product=42640, rating=31.421513489581343)
```

Picture 24

```
 Top 5 users for item 24:
Rating(user=26446, product=24, rating=21.14302610901221)
Rating(user=27420, product=24, rating=18.986041043064354)
Rating(user=9269, product=24, rating=18.858588849826855)
Rating(user=18152, product=24, rating=18.848711931737874)
Rating(user=1265, product=24, rating=18.150144736642304)
[(0, Rating(user=0, product=67, rating=9.0)), (0, Rating(user=0, product=6702, rating=7.0)), (0, Rating(user=0, product=242, rating=10.0)), (0, Rating
(user=0, product=4898, rating=0.0)), (0, Rating(user=0, product=21, rating=10.0)), (0, Rating(user=0, product=24, rating=9.0)), (0, Rating(user=0, prod
uct=2104, rating=0.0)), (0, Rating(user=0, product=4722, rating=8.0)), (0, Rating(user=0, product=6098, rating=6.0)), (0, Rating(user=0, product=3125,
rating=9.0))]
User: 0, Anime: 67, Rating: 9.0
User: 0, Anime: 6702, Rating: 7.0
User: 0, Anime: 242, Rating: 10.0
User: 0, Anime: 4898, Rating: 0.0
User: 0, Anime: 21, Rating: 10.0
User: 0, Anime: 24, Rating: 9.0
User: 0, Anime: 2104, Rating: 0.0
```

Picture 25

```
User: 0, Anime: 2104, Rating: 0.0
User: 0, Anime: 4722, Rating: 8.0
User: 0, Anime: 6098, Rating: 6.0
User: 0, Anime: 3125, Rating: 9.0

Top 10 anime for user 0:
 User: 0, Anime: 242, Rating: 10.0
 User: 0, Anime: 21, Rating: 10.0
 User: 0, Anime: 481, Rating: 10.0
 User: 0, Anime: 1571, Rating: 10.0
 User: 0, Anime: 578, Rating: 10.0
 User: 0, Anime: 2236, Rating: 10.0
 User: 0, Anime: 415, Rating: 10.0
 User: 0, Anime: 235, Rating: 10.0
 User: 0, Anime: 67, Rating: 9.0
 User: 0, Anime: 24, Rating: 9.0
```

Picture 26

```
Rates and Predictions (Sample):
[((0, 134), (0.0, 5.462198027097403)), ((2, 38408), (0.0, 1.8541030926109077)), ((2, 40472), (0.0, 0.8116989668051047)), ((2, 2352), (0.0, -0.45728268
0882931)), ((2, 264), (0.0, 2.2881143492575724)), ((2, 4304), (0.0, -0.43843002997115077)), ((2, 39392), (0.0, -1.487392785192653)), ((2, 10920), (0.0,
-0.06598384573142124)), ((3, 199), (9.0, 7.519125863241031)), ((3, 21431), (6.0, 6.933843637599379))]
Predictions (Sample):
 User: 0, Anime: 24, Rating: 2.614560772170855
 User: 0, Anime: 600, Rating: 2.0367736546990547
 User: 0, Anime: 3457, Rating: 2.8635050839315968
 User: 0, Anime: 433, Rating: 3.911826732756996
 User: 0, Anime: 2762, Rating: 8.552963120951768
 User: 0, Anime: 1250, Rating: 2.2710470021639386
 User: 0, Anime: 5114, Rating: 9.993162354090412
 User: 0, Anime: 4898, Rating: 2.844841309157293
 User: 0, Anime: 134, Rating: 5.462198027097403
 User: 0, Anime: 459, Rating: 9.267343815189266
 User: 0, Anime: 1047, Rating: 4.116517745862498
 User: 0, Anime: 245, Rating: 6.609543597715331
 User: 0, Anime: 3125, Rating: 12.535582760011868
```

Picture 27

```
 User: 0, Anime: 3125, Rating: 12.535582760011868
 User: 0, Anime: 2034, Rating: 2.656418565655729
 User: 0, Anime: 1074, Rating: 2.0183478631841036
 User: 0, Anime: 1482, Rating: 5.681864905175012
 User: 0, Anime: 415, Rating: 5.8229626738710385
 User: 0, Anime: 19, Rating: 4.157280783518018
 User: 0, Anime: 1004, Rating: 1.2747094809705273
 User: 0, Anime: 68, Rating: 4.711616807130275
 User: 0, Anime: 164, Rating: 6.841633317805033
 User: 0, Anime: 3010, Rating: 8.226337584678333
 User: 0, Anime: 1535, Rating: 6.270240769822222
 User: 0, Anime: 1571, Rating: 4.290264954918476
 User: 0, Anime: 71, Rating: 5.228001545653825

Test Data (Sample):
 User: 0, Anime: 4898, Rating: 0.0
 User: 0, Anime: 24, Rating: 9.0
 User: 0, Anime: 3125, Rating: 9.0
```

Picture 28

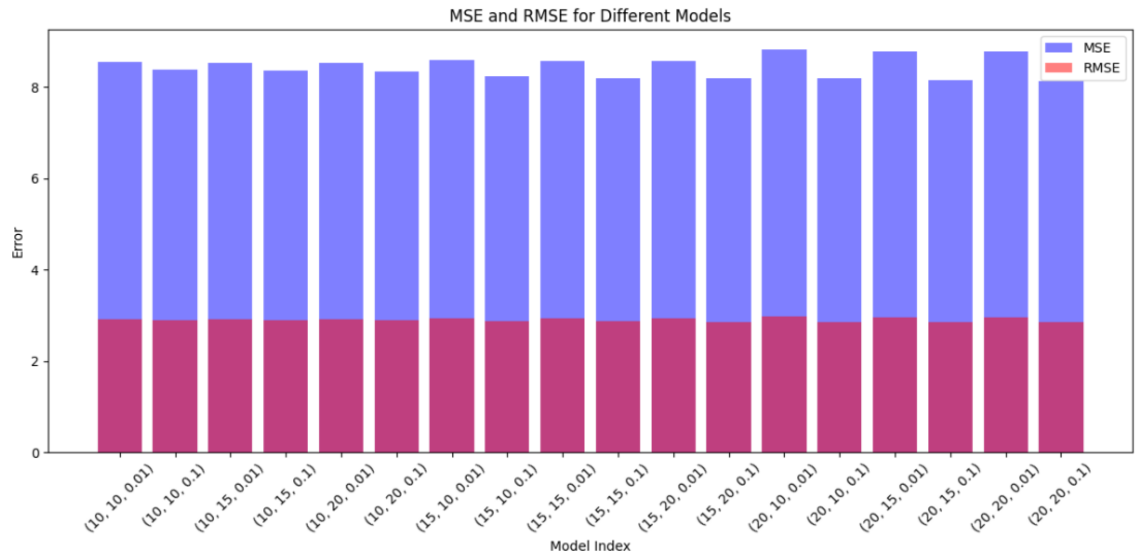
```
User: 0, Anime: 1250, Rating: 7.0
User: 0, Anime: 1571, Rating: 10.0
User: 0, Anime: 2762, Rating: 9.0
User: 0, Anime: 3010, Rating: 7.0
User: 0, Anime: 1004, Rating: 5.0
User: 0, Anime: 433, Rating: 6.0
Mean Squared Error (MSE) =  8.56007581550177
Root Mean Squared Error (RMSE) =  2.925760724239385

Son eleman - Rank: 10, Iteration: 10, Lambda: 0.01, MSE: 8.56007581550177, RMSE: 2.925760724239385

1. process is successful
```

This is the output of a model. We cannot add all the outputs because there are 18 outputs. You can view the rest of the output from the HTML file.

Picture 29

Graph of model results

```
ranks_iterations_lambdas = [(params[0], params[1], params[2]) for params, mse, rmse in evaluations]
mses = [mse for params, mse, rmse in evaluations]
rmses = [rmse for params, mse, rmse in evaluations]

plt.figure(figsize=(12, 6))
plt.bar(range(len(evaluations)), mses, color='b', alpha=0.5, label='MSE')
plt.bar(range(len(evaluations)), rmses, color='r', alpha=0.5, label='RMSE')
plt.xlabel('Model Index')
plt.ylabel('Error')
plt.title('MSE and RMSE for Different Models')
plt.xticks(range(len(evaluations)), ranks_iterations_lambdas, rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```



Picture 30

```
best_model_info = min(evaluations, key=lambda x: x[2])
print("Best Model:", best_model_info)
best_model_params = best_model_info[0]
best_model = ALS.train(train_data, rank=best_model_params[0], iterations=best_model_params[1], lambda_=best_model_params[2])
```

```
Best Model: ((20, 20, 0.1), 8.143839056066907, 2.853741238456442)
```

```
new_item_id = 24
new_item_vector = best_model.productFeatures().lookup(item_id)[0]
new_item_vector
```

```
array('d', [0.21108445525169373, -0.688731849193573, -0.1417933702468872, -1.086441159248352, 0.6083396077156067, -0.9333931803703308, -0.062410924583673
48, -1.0376697778701782, -1.2499043941497803, -0.19543635845184326, 0.6239340305328369, -0.1653013825416565, -0.4606398642063141, 0.3790259063243866, -0.
25339534878730774, 0.0700823962688446, 0.16330190002918243, 0.19273671507835388, 0.13674099743366241, -1.5032151937484741])
```

```
similarities = best_model.productFeatures().map(lambda data: cosineSimilarity(data[0], data[1], new_item_vector))
```

```
similarities.top(10, key=lambda x:x[1])
```

```
[(24, 1.0),
 (846, 0.9069885760931016),
 (189, 0.895339072160908),
 (240, 0.858171311968612З),
 (32789, 0.854749483294183),
 (50, 0.851648428991654З),
 (248, 0.851613796664834б),
 (59, 0.840048131976342),
 (880, 0.834969058253374),
 (71, 0.833311198626763)]
```

🎉 🎉 🎉 🎉 🎉 🎉 🎉

Picture 31