AYDIN ADNAN MENDERES UNIVERSITY

ENGINEERING FACULTY

COMPUTER SCIENCE ENGINEERING DEPARTMANT

# Recommendation System With Spark

CSE424 BIG DATA ANALYSIS, SPRING 2024/2025

Student's Name SURNAME:

211805018 - Köksal Kerem TANIL

211805021 - Rıza KARAKAYA

211805014 – Mustafa Cihan AYINDI

211805015 – Yusuf TURAN

Lecturer:

Asst. Prof. Dr. Hüseyin ABACI

# ***Goodreads Book Recommendation System using PySpark and ALS***

**To Do List:**

- Environment Setup & Configuration
- Dataset Acquisition & Initial Preparation
- Data Preprocessing for Recommendation Model
- Exploratory Data Analysis & Visualization
- ALS Model Training & Hyperparameter Tuning:
- Model Evaluation & Selection
- Report Preparation

## 1. Environment Setup & Configuration

The successful execution of this PySpark-based Goodreads book recommendation system relies on a properly configured development environment. This section outlines the key software components installed and configured.

- **1.1. Core System and Python Environment:**
  - **Java Development Kit (JDK):** Apache Spark requires a compatible JDK (e.g., JDK 8 or 11) as it runs on the Java Virtual Machine (JVM). This was installed, and the JAVA_HOME environment variable was set accordingly.
    - **Download JDK11**
  - **Python Installation:** A suitable version of Python (e.g., 3.8+) was installed to serve as the programming language for PySpark and data manipulation tasks.
    - **Download Python**
  - **Jupyter Notebook/JupyterLab:** The project was developed within Jupyter Notebook, an interactive computing environment, installed via pip (pip install notebook).
    - **Install Jupyter Notebook**
- **1.2. Apache Spark and PySpark Setup:**
  - **Apache Spark Installation:** The Apache Spark distributed processing engine (e.g., Spark 3.x.x) was downloaded and extracted. The SPARK_HOME environment variable was configured to point to this installation directory, and Spark's bin directory was added to the system PATH.
    - **Download Apache Spark**
  - **Hadoop Binaries:** For local mode on Windows, winutils.exe and associated Hadoop binaries were made available, and HADOOP_HOME was configured to prevent common file system errors.
    - **Download Apache Hadoop**
  - **PySpark Library Installation:** The PySpark Python API was installed using pip (pip install pyspark).
    - **Install PySpark**
- **1.3. Essential Python Libraries:**
  - Key Python libraries for data analysis (pandas, numpy), machine learning (via PySpark's MLlib, implicitly used with pyspark.ml), visualization (matplotlib, seaborn), and system utilities (psutil, socket, platform, os, json, datetime) were imported directly within the Jupyter Notebook in the initial setup cell.

In [1]:
```python
# Hücre 1: Gerekli Kütüphaneleri Yükleme ve Temel Bilgiler (Güncellenmiş)

# Temel Python ve Sistem Kütüphaneleri
import socket
import psutil
import platform
import os
import shutil # Checkpoint dizinini yönetmek için (opsiyonel)
import json # JSON işlemleri için
import datetime # Zaman damgaları ve süreler için

# PySpark Kütüphaneleri
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr, lit, udf, monotonically_increasing_id, broadcast
from pyspark.ml.recommendation import ALS, ALSModel # ALSModel'i yükleme için ekledik
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import StringIndexer, IndexToString
from pyspark.ml.linalg import Vectors, VectorUDT, DenseVector # VectorUDT nadiren doğrudan gerekir
from pyspark.sql.types import FloatType, ArrayType, DoubleType, IntegerType

# Veri Analizi ve Görselleştirme Kütüphaneleri
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Görselleştirmelerin notebook içinde görünmesi için
%matplotlib inline
```

- **1.4. SparkSession Initialization:**
  - A SparkSession was programmatically created and configured within the notebook. This involved:
    - Setting an application name ("GoodreadsRecommendationSystem_Project").
    - Allocating resources (driver/executor memory, cores).
    - Configuring the KryoSerializer for efficiency.
    - Defining and setting a CHECKPOINT_DIRECTORY for Spark's checkpointing mechanism, which is beneficial for iterative algorithms like ALS.

In [3]:
```python
# Hücre 3: Spark Session Başlatma (Düzeltilmiş Kısım)

# ... (SparkSession.builder kısmı aynı kalacak) ...
spark = SparkSession.builder \
    .appName("GoodreadsRecommendationSystem_Project") \
    .config("spark.executor.memory", "8g") \
    .config("spark.driver.memory", "8g") \
    .config("spark.executor.cores", "4") \
    .config("spark.sql.shuffle.partitions", "200") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("spark.kryoserializer.buffer.max", "1024m") \
    .getOrCreate()

spark.sparkContext.setCheckpointDir(CHECKPOINT_DIRECTORY)

# Düzeltilmiş print satırı:
checkpoint_dir_status = spark.sparkContext.getCheckpointDir()
if checkpoint_dir_status:
    print(f"Spark Session Başlatıldı. Checkpoint dizini: {checkpoint_dir_status}")
else:
    print(f"Spark Session Başlatıldı. Checkpoint dizini: Ayarlanmadı") # veya None durumu için başka bir mesaj

print(f"Spark Version: {spark.version}")
spark
```

```
25/05/29 16:59:06 WARN Utils: Your hostname, Hatice-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; using 172.20.10.5 instead (on interface en0)
25/05/29 16:59:06 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/29 16:59:07 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark Session Başlatıldı. Checkpoint dizini: file:/Users/haticekandemir/python_spark/kerem/checkpoint_dir_als_project_v2/174f7a5e-cffb-4366-ac72-7ea14c960e3d
Spark Version: 3.5.3
```

- **1.5. Project Configuration and Paths:**
  Before data loading, key project paths and constants were defined to ensure a structured workflow and reproducibility. This included:
  - YOUR_STUDENT_ID_LAST_4_DIGITS: Used as a seed for random operations.
  - CHECKPOINT_DIRECTORY: Specified path for Spark's checkpointing.
  - BASE_PROJECT_PATH: The root directory for storing all project outputs like trained models, results, and logs.

- Derived paths for RESULTS_JSON_PATH (model progress), BEST_MODEL_INFO_PATH (best model metadata), ERRORS_LOG_PATH (error logs), and BEST_MODEL_SAVE_PATH (saved best ALS model).
- A helper function log_error_message was also defined for standardized error logging.

```
# --- Değişken Tanımlamaları ---
YOUR_STUDENT_ID_LAST_4_DIGITS = 5018 # BU DEĞERİ DEĞİŞTİRİN!
CHECKPOINT_DIRECTORY = "checkpoint_dir_als_project_v2" # Spark checkpoint'leri için

# Proje Dosya Yolları
BASE_PROJECT_PATH = "project/goodreads_models" # Ana proje dizininiz
RESULTS_JSON_PATH = os.path.join(BASE_PROJECT_PATH, "progress.json") # İlerleme kayıtları
BEST_MODEL_INFO_PATH = os.path.join(BASE_PROJECT_PATH, "best_model_info.json") # En iyi modelin meta verileri
ERRORS_LOG_PATH = os.path.join(BASE_PROJECT_PATH, "errors.log") # Hata logları
BEST_MODEL_SAVE_PATH = os.path.join(BASE_PROJECT_PATH, "best_model") # Kaydedilecek en iyi Spark modeli yolu

# Gerekirse ana proje dizinini oluştur (veya var olanı kullan)
try:
    os.makedirs(BASE_PROJECT_PATH, exist_ok=True)
    print(f"Ana proje dizini '{BASE_PROJECT_PATH}' mevcut veya oluşturuldu.")
except OSError as error:
    print(f"'{BASE_PROJECT_PATH}' dizini oluşturulurken hata: {error}")
    # Bu durumda script'in devam etmemesi daha iyi olabilir, kullanıcıya dizini kontrol etmesini söyleyin.
    # raise # veya exit()

print("\n--- Temel Değişkenler ve Yollar ---")
print(f"Öğrenci ID (Seed): {YOUR_STUDENT_ID_LAST_4_DIGITS}")
print(f"Spark Checkpoint Dizini: {CHECKPOINT_DIRECTORY}")
print(f"İlerleme (Progress) JSON Dosyası: {RESULTS_JSON_PATH}")
print(f"En İyi Model Bilgi JSON Dosyası: {BEST_MODEL_INFO_PATH}")
print(f"Hata Log Dosyası: {ERRORS_LOG_PATH}")
print(f"En İyi Model Kayıt Yolu (Spark Modeli): {BEST_MODEL_SAVE_PATH}")

# --- Hata Loglama Fonksiyonu ---
def log_error_message(message, log_path=ERRORS_LOG_PATH):
    """Belirtilen yola zaman damgasıyla hata mesajı yazar."""
    try:
        with open(log_path, "a", encoding="utf-8") as f: # encoding eklendi
            timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            f.write(f"[{timestamp}] ERROR: {message}\n")
    except Exception as e:
        print(f"!!! Hata log dosyasına ({log_path}) yazılırken bir sorun oluştu: {e} !!!")
        print(f"    Orijinal hata mesajı: {message}")

print("\nKütüphaneler yüklendi ve temel değişkenler/fonksiyonlar ayarlandı.")
```

```
Ana proje dizini 'project/goodreads_models' mevcut veya oluşturuldu.

--- Temel Değişkenler ve Yollar ---
Öğrenci ID (Seed): 5018
Spark Checkpoint Dizini: checkpoint_dir_als_project_v2
İlerleme (Progress) JSON Dosyası: project/goodreads_models/progress.json
En İyi Model Bilgi JSON Dosyası: project/goodreads_models/best_model_info.json
Hata Log Dosyası: project/goodreads_models/errors.log
En İyi Model Kayıt Yolu (Spark Modeli): project/goodreads_models/best_model

Kütüphaneler yüklendi ve temel değişkenler/fonksiyonlar ayarlandı.
```

- **1.6. PC Information:**

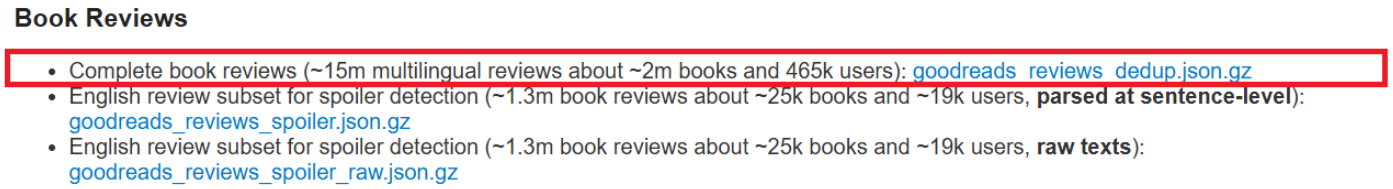
In [2]:
```
# Bilgisayar Bilgileri
print("--- Bilgisayar Bilgileri ---")
print(f"Hostname: {socket.gethostname()}")
try:
    hostname = socket.gethostname()
    local_ip = socket.gethostbyname(hostname)
    print(f"Yerel IP Adresi (Tahmini): {local_ip}")
except socket.gaierror:
    print("Yerel IP adresi alınamadı. Ağ bağlantınızı kontrol edin.")

print(f"İşletim Sistemi: {platform.system()} {platform.release()}")
print(f"İşlemci: {platform.processor()}")

# RAM Bilgileri
vmem = psutil.virtual_memory()
total_ram_gb = vmem.total / (1024**3)
available_ram_gb = vmem.available / (1024**3)
print(f"Toplam RAM: {total_ram_gb:.2f} GB")
print(f"Kullanılabilir RAM: {available_ram_gb:.2f} GB")

cpu_cores = psutil.cpu_count(logical=True)
print(f"CPU Çekirdek Sayısı (Mantıksal): {cpu_cores}")
print("--------------------------")
```

```
--- Bilgisayar Bilgileri ---
Hostname: Hatice-MacBook-Pro.local
Yerel IP Adresi (Tahmini): 127.0.0.1
İşletim Sistemi: Darwin 23.2.0
İşlemci: arm
Toplam RAM: 16.00 GB
Kullanılabilir RAM: 10.56 GB
CPU Çekirdek Sayısı (Mantıksal): 8
--------------------------
```

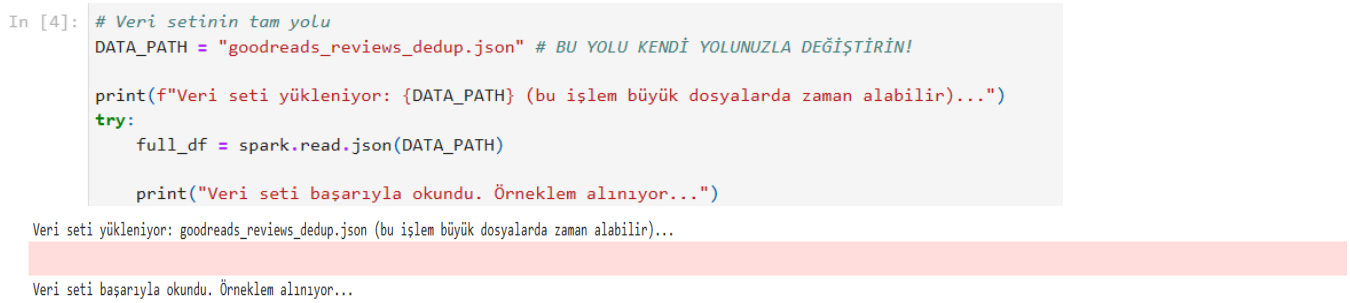
## 2. Dataset Acquisition and Preparation

This phase involved obtaining the Goodreads book review dataset and performing the initial steps to prepare it for the recommendation model. The quality and structure of the input data are critical for building an effective recommendation system.

- **2.1. Dataset Overview (Referenced from UCSD Book Graph):**
  - **Dataset Source:** The project utilized the "Goodreads Book Reviews Dataset" (goodreads_reviews_dedup.json), sourced from the UCSD Book Graph dataset collection, publicly available at: **Goodreads Book Reviews Dataset.**
  - **Size:** The complete reviews dataset contains approximately 15 million records.
  - **Languages:** The review texts are multilingual.
  - **Scope (Approximate):** The broader collection covers ~2 million unique books and ~465,000 unique users.
  - **Dataset Content and Purpose:** This dataset contains a large collection of book reviews from Goodreads. Each entry includes user IDs, book IDs, and ratings (typically on a 1-5 scale, though raw data may include 0s which are handled in preprocessing). The primary purpose of using this dataset is to leverage the explicit feedback (ratings) provided by users to build a collaborative filtering-based book recommendation system. The goal is to understand user preferences and item characteristics from these interactions to predict how a user might rate a book they have not yet encountered, and subsequently recommend books they are likely to find engaging.
  - **Format:** The data was in JSON format.




- **2.2. Data Loading and Sampling:**
  - The full JSON dataset was loaded into a Spark DataFrame (full_df).



```
In [4]: # Veri setinin tam yolu
DATA_PATH = "goodreads_reviews_dedup.json" # BU YOLU KENDİ YOLUNUZLA DEĞİŞTİRİN!

print(f"Veri seti yükleniyor: {DATA_PATH} (bu işlem büyük dosyalarda zaman alabilir)...")
try:
    full_df = spark.read.json(DATA_PATH)

    print("Veri seti başarıyla okundu. Örneklem alınıyor...")
```

```
Veri seti yükleniyor: goodreads_reviews_dedup.json (bu işlem büyük dosyalarda zaman alabilir)...

Veri seti başarıyla okundu. Örneklem alınıyor...
```

  - A 10% random sample (sampled_df) was created for development efficiency, using a reproducible seed.

```
sample_fraction = 0.1 # %0.1 deneyelim.
sampled_df = full_df.sample(withReplacement=False, fraction=sample_fraction, seed=YOUR_STUDENT_ID_LAST_4_DIGITS)

sampled_df.cache() # Örneklenmiş veriyi bellekte tut

print("\nÖrneklenmiş Veri Şeması:")
sampled_df.printSchema()
```

```
Örneklenmiş Veri Şeması:
root
 |-- book_id: string (nullable = true)
 |-- date_added: string (nullable = true)
 |-- date_updated: string (nullable = true)
 |-- n_comments: long (nullable = true)
 |-- n_votes: long (nullable = true)
 |-- rating: long (nullable = true)
 |-- read_at: string (nullable = true)
 |-- review_id: string (nullable = true)
 |-- review_text: string (nullable = true)
 |-- started_at: string (nullable = true)
 |-- user_id: string (nullable = true)
```

- sampled_df count and sampled_df show output.

```
sampled_df_count = sampled_df.count()
print(f"\nÖrneklenmiş Veri Satır Sayısı ({sample_fraction*100}%): {sampled_df_count}")

if sampled_df_count == 0:
    print("UYARI: Örneklem sonucu boş DataFrame! Sample fraction çok küçük olabilir veya veri setinde sorun olabilir.")
else:
    print("\nÖrneklenmiş Veriden İlk 5 Satır:")
    sampled_df.show(5, truncate=False)
```

```
Örneklenmiş Veri Satır Sayısı (10.0%): 1573038
```

```
[Stage 1:=============================================================> (121 + 4) / 125]
Örneklenmiş Veri Satır Sayısı (10.0%): 1573038

Örneklenmiş Veriden İlk 5 Satır:
+--------+--------------------------+--------------------------+----------+-------+------+--------------------------+------------------------+------------------------+
|book_id |date_added                |date_updated              |n_comments|n_votes|rating|read_at                   |review_id               |review_text
|started_at                |user_id                 |
+--------+--------------------------+--------------------------+----------+-------+------+--------------------------+------------------------+------------------------+
|9460786 |Tue Feb 28 17:52:08 -0800 2017|Fri May 26 12:04:23 -0700 2017|3         |13     |4     |Fri May 26 15:17:45 -0700 2017|fc24c814e34ec6da3aa5293b7ec70dd9|Giving a high rating because I heard the organizer of the Long Now Foundation speak and it was very inspi
ring. The Interval Cafe in San Francisco is also awesome and has The Long Library in it. \n The part of the story that I liked most was the power of long thinking. How Oxford College has some gorgeous oak tree beams in their dining room, and they were crumbling, and
so they wondered how to replace them. They created a search, and happened to ask the Oxford groundskeeper if any of the oak trees on campus would work, and he said yes, they should use the grove that was planted 500 years ago for that very purpose. So they did, and t
hen planted another grove for 500 years from now. That's the power of long term thinking - how many of us are able to think that far ahead? That story is what inspired the clock, which is getting closer to completion and sounds pretty cool. \n The book was published
in 1999 so I found a bunch of the ideas dated and found myself skimming through them, so only partially read this.
|Fri May 05 00:00:00 -0700 2017|8842281e1d1347389f2ab93d60773d4d|
|16981   |Mon Dec 05 10:46:44 -0800 2016|Wed Mar 22 11:37:04 -0700 2017|0         |1      |3     |                          |a5d2c3628987712d0e05c4f90798eb67|Recommended by Don Katz. Avail for free in December: http://www.audible.com/mt/ellison2?so...
|                          |8842281e1d1347389f2ab93d60773d4d|
|19398490|Sun Jan 03 21:20:46 -0800 2016|Tue Sep 20 23:30:15 -0700 2016|5         |35     |4     |Tue Sep 13 11:51:51 -0700 2016|ea4a220b10e6b5c796dae0e3b970aff1|A beautiful story. It is rare to encounter a book that does such a good job painting the scenes in your m
ind - you really felt like you were there and got to know the characters and the people they came across. I generally love WWII books and movies, but wasn't sure if I'd like one featuring a blind girl and a young Nazi radio operator - but he brings a lot of life to t
hem. \n That said, the novel drifts around a lot, and the plotline jumping forward and backward in time drove me nuts. \n I thought a lot about if there is a theme to the book. One was the diamond and if it was really cursed (based on the Hope Diamond perhaps?). Anot
her is that 20K Leagues is an awesome book. But I think the book was a lot about fear and the unknown, and how people deal with it. Marie-Laure was blind and couldn't tell what was happening around her - yet she was the bravest one. Werner was afraid of ending up as
a miner, which drove him to join the army, and learn about radios. And Uncle Etienne was afraid dying from a sniper he couldn't see, so he didn't leave his house and he created a radio transmitter. So the moral of the story is... don't be paralyzed by your fears. \n
```

- **2.3. Initial Data Transformation for ALS:**
  - Essential columns (user_id, book_id, rating) were selected.
  - The rating column was cast to float, creating als_df.

```
if 'sampled_df' in locals() and sampled_df.count() > 0 :
    print("--- Veri Dönüşümü ve Temel Hazırlık (ALS için) ---")

    als_df = sampled_df.select(
        col("user_id").alias("original_user_id"), # Orijinal ID'yi sakla
        col("book_id").alias("original_book_id"), # Orijinal ID'yi sakla
        col("rating").cast("float")
    )

    print("\nSeçilen ve dönüştürülen sütunlarla DataFrame (als_df):")
    als_df.show(5, truncate=False)
    als_df.printSchema()
```

```
--- Veri Dönüşümü ve Temel Hazırlık (ALS için) ---

Seçilen ve dönüştürülen sütunlarla DataFrame (als_df):
+--------------------------------+----------------+------+
|original_user_id                |original_book_id|rating|
+--------------------------------+----------------+------+
|8842281e1d1347389f2ab93d60773d4d|9460786         |4.0   |
|8842281e1d1347389f2ab93d60773d4d|16981           |3.0   |
|8842281e1d1347389f2ab93d60773d4d|19398490        |4.0   |
|8842281e1d1347389f2ab93d60773d4d|18662473        |5.0   |
|8842281e1d1347389f2ab93d60773d4d|23158207        |3.0   |
+--------------------------------+----------------+------+
only showing top 5 rows
```

## 3. Data Preprocessing for Recommendation Model:

After initial data loading and sampling, further preprocessing steps were applied to prepare the data specifically for the Alternating Least Squares (ALS) recommendation algorithm. This involved cleaning the data, splitting it into training and testing sets, and converting identifiers into a numerical format suitable for the model.

- **3.1. Data Cleaning and Final Selection:**
    - Objective: To ensure the data used for ALS (df_for_als) contains only valid and relevant records.
    - Process:
        - The als_df (created in step 2.3) was further processed by dropping rows containing any null (missing) values in the original_user_id, original_book_id, or rating columns. This resulted in the als_df_cleaned DataFrame.

```
print(f"\nİşlem öncesi satır sayısı (als_df): {als_df.count()}")
als_df_cleaned = als_df.na.drop(subset=["original_user_id", "original_book_id", "rating"])
print(f"Eksik değerler temizlendikten sonra satır sayısı (als_df_cleaned): {als_df_cleaned.count()}")
```

```
İşlem öncesi satır sayısı (als_df): 1573038
Eksik değerler temizlendikten sonra satır sayısı (als_df_cleaned): 1573038
```

        - The distribution of rating values in als_df_cleaned was examined using groupBy("rating").count() to understand the frequency of each rating score.

```
print("\nRating değerlerinin dağılımı (als_df_cleaned):")
als_df_cleaned.groupBy("rating").count().orderBy("rating").show()
```

```
Rating değerlerinin dağılımı (als_df_cleaned):
+------+------+
|rating| count|
+------+------+
|   0.0| 55090|
|   1.0| 44977|
|   2.0|110695|
|   3.0|310694|
|   4.0|525416|
|   5.0|526166|
+------+------+
```

- The cleaned DataFrame was then assigned to df_for_als and cached for efficient subsequent use.

```
# Sonraki adımlarda kullanmak üzere temizlenmiş DataFrame'i saklayalım
df_for_als = als_df_cleaned.cache() # Temizlenmiş veriyi de cache'leyelim
print(f"df_for_als {df_for_als.count()} satır ile hazırlandı ve cache'lendi.")
```

```
df_for_als 1573038 satır ile hazırlandı ve cache'lendi.
```

- **3.2. RDD API Demonstration for Data Understanding:**
  - Objective: To illustrate an alternative way of aggregating data using Spark's RDD API, reinforcing understanding of rating distributions.
  - Process: df_for_als was converted to an RDD, and map and reduceByKey operations were used to count occurrences of each rating.

```
rating_counts_rdd = df_for_als.rdd.map(lambda row: (int(row.rating), 1)) \
                                  .reduceByKey(lambda a, b: a + b) \
                                  .sortBy(lambda x: x[0]) # Rating'e göre sırala
```

```
RDD ile hesaplanan rating sayıları (rating, count):
[Stage 32:=========================================================>(123 + 2) / 125]
(0, 55090)
(1, 44977)
(2, 110695)
(3, 310694)
(4, 525416)
(5, 526166)
```

- **3.3. Train-Test Split:**
  - Objective: To divide the dataset into separate sets for training the model and evaluating its performance on unseen data.
  - Process:
    - df_for_als was randomly split into training_df (70% of the data) and test_df (30% of the data) using randomSplit().
    - A seed (5018) was used for reproducibility.

```
(training_df, test_df) = df_for_als.randomSplit([0.7, 0.3], seed=YOUR_STUDENT_ID_LAST_4_DIGITS)
```

```
--- Veri Setini Eğitim ve Test Olarak Ayırma, Indexleme ve Checkpoint ---
Toplam satır sayısı (df_for_als): 1573038
Eğitim seti satır sayısı: 1101291
Test seti satır sayısı: 471747
```

- **3.4. Numerical ID Indexing:**
  - Objective: To convert the string-based original_user_id and original_book_id columns into numerical representations (user_id_indexed, item_id_indexed), as ALS requires numerical inputs.
  - Process:

- Two StringIndexer instances were created: one for user IDs and one for item (book) IDs. The handleInvalid='skip' strategy was used to ignore any new IDs in the test set not seen during training.
- The user_indexer was fitted on the training_df to create user_indexer_model. This model was then used to transform both training_df and test_df.
- Similarly, the item_indexer was fitted on the user-indexed training_df to create item_indexer_model. This model then transformed both the user-indexed training and test sets.
- This two-step indexing ensures that the mapping from string ID to numerical ID is consistent across both datasets and is based solely on the training data.

```
# StringIndexer'ları hazırlama ve uygulama
user_indexer = StringIndexer(inputCol="original_user_id", outputCol="user_id_indexed", handleInvalid='skip')
item_indexer = StringIndexer(inputCol="original_book_id", outputCol="item_id_indexed", handleInvalid='skip')

user_indexer_model = user_indexer.fit(training_df)
training_indexed_df = user_indexer_model.transform(training_df)
test_indexed_df = user_indexer_model.transform(test_df) # Test setini de aynı user indexer ile dönüştür

item_indexer_model = item_indexer.fit(training_indexed_df) # Item indexer'ı eğitim seti üzerinden fit et
training_final_df = item_indexer_model.transform(training_indexed_df)
test_final_df = item_indexer_model.transform(test_indexed_df) # Test setini de aynı item indexer ile dönüştür

print("\nString Indexer sonrası eğitim verisinden örnek:")
training_final_df.select("original_user_id", "user_id_indexed", "original_book_id", "item_id_indexed", "rating").show(3, truncate=False)
```

```
String Indexer sonrası eğitim verisinden örnek:
25/05/29 17:00:09 WARN DAGScheduler: Broadcasting large task binary with size 27.6 MiB
+--------------------------------+---------------+----------------+---------------+------+
|original_user_id                |user_id_indexed|original_book_id|item_id_indexed|rating|
+--------------------------------+---------------+----------------+---------------+------+
|005f35d2daeb6f48bdb13fbd8c6a2522|17448.0        |17727860        |211143.0       |3.0   |
|005f35d2daeb6f48bdb13fbd8c6a2522|17448.0        |18295750        |4458.0         |5.0   |
|005f35d2daeb6f48bdb13fbd8c6a2522|17448.0        |18655646        |227065.0       |3.0   |
+--------------------------------+---------------+----------------+---------------+------+
only showing top 3 rows
```

- **3.5. Checkpointing and Caching:**
  - Objective: To optimize performance and provide fault tolerance for the subsequent iterative ALS training process.
  - Process:
    - The final indexed DataFrames, training_final_df and test_final_df, were checkpointed using the .checkpoint() method. An action (.count()) was called to trigger the materialization of the checkpoint.

```
# DataFrame Checkpoint'lerini Uygula
print("\nDataFrame'ler checkpoint ediliyor...")
training_final_df = training_final_df.checkpoint()
training_final_df.count() # Eylem
print("training_final_df başarıyla checkpoint edildi.")

test_final_df = test_final_df.checkpoint()
test_final_df.count() # Eylem
print("test_final_df başarıyla checkpoint edildi.")
```

```
DataFrame'ler checkpoint ediliyor...
25/05/29 17:00:10 WARN DAGScheduler: Broadcasting lar

training_final_df başarıyla checkpoint edildi.
25/05/29 17:00:20 WARN DAGScheduler: Broadcasting lar

test_final_df başarıyla checkpoint edildi.
```

- The checkpointed DataFrames were then cached using .cache() to keep them in memory for faster access during model training.

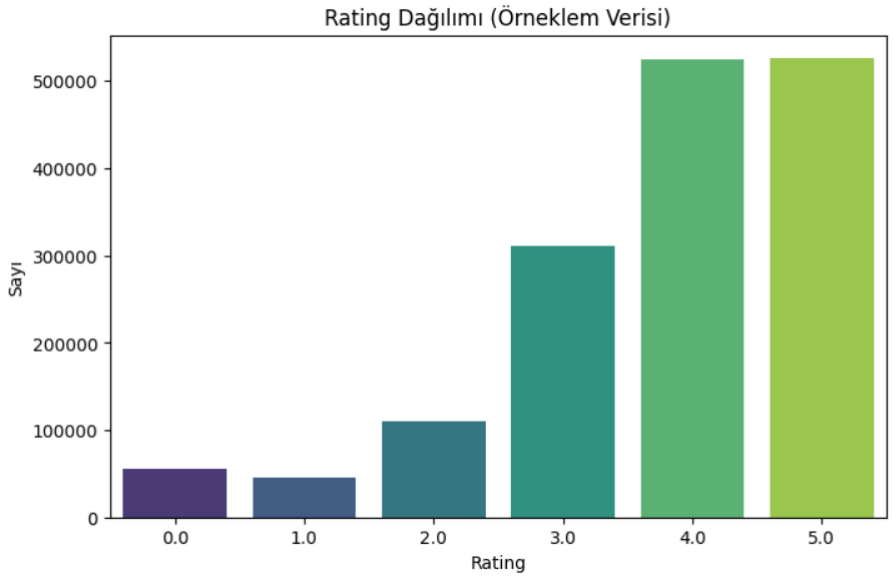
```
# Cache'leme
training_final_df.cache()
test_final_df.cache()
print("İndekslenmiş ve Checkpoint'lenmiş DataFrame'ler cache'lendi.")
```

```
İndekslenmiş ve Checkpoint'lenmiş DataFrame'ler cache'lendi.
```

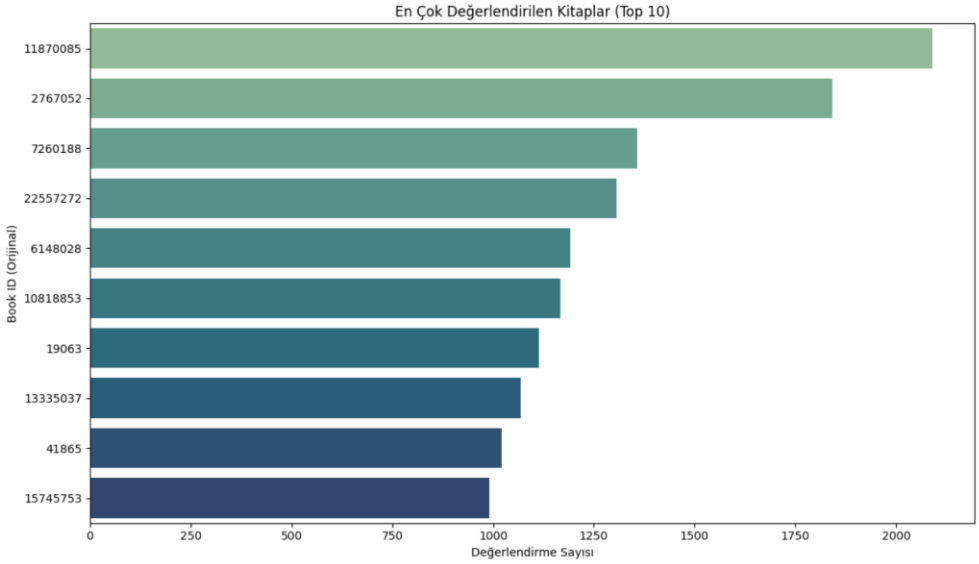
## 4. Data Visualization:

Visualizations were employed at different stages of the project to gain insights into the dataset's characteristics and to understand model performance across various hyperparameter configurations. Pandas, Matplotlib, and Seaborn libraries were used for creating these plots, with Spark DataFrames being converted to Pandas DataFrames where necessary for compatibility.

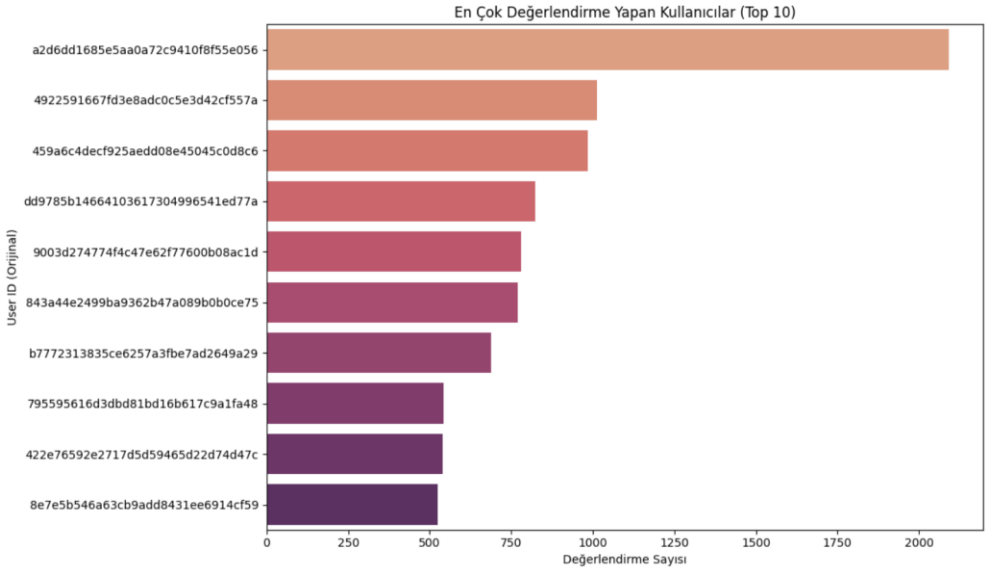
- **4.1. Exploratory Data Visualizations:**
  - Objective: To understand the fundamental properties of the prepared dataset (df_for_als) before model training.
  - Process & Visualizations:
    - Rating Distribution: A count plot was generated to show the frequency of each rating value (0.0 to 5.0) in the df_for_als dataset. This helps in understanding the overall sentiment and potential biases in ratings.




    - Top 10 Most Rated Books: The 10 books with the highest number of ratings were identified and visualized using a horizontal bar plot. This highlights popular items in the dataset.

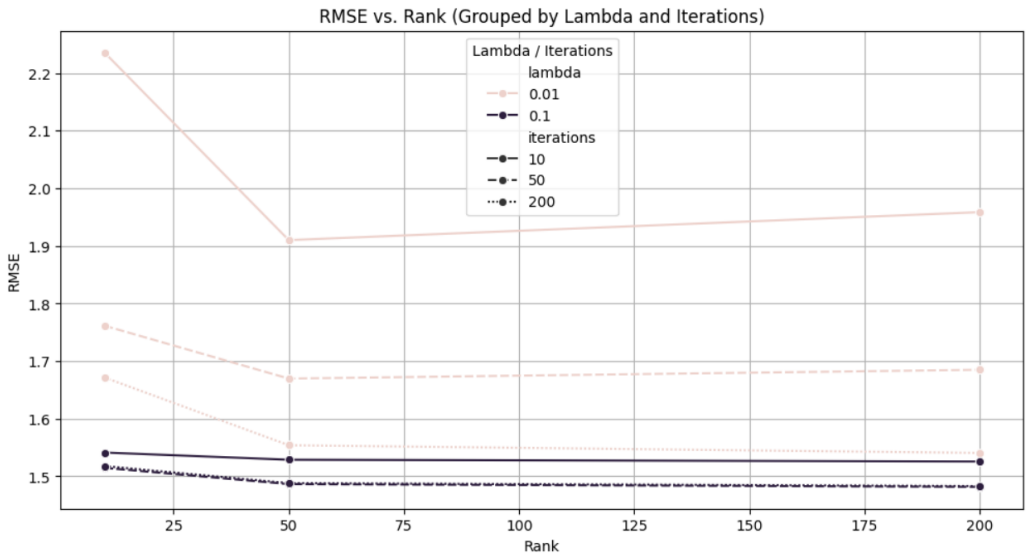



- Top 10 Most Active Users: The 10 users who provided the most ratings were identified and visualized, also with a horizontal bar plot. This helps in understanding user engagement levels.

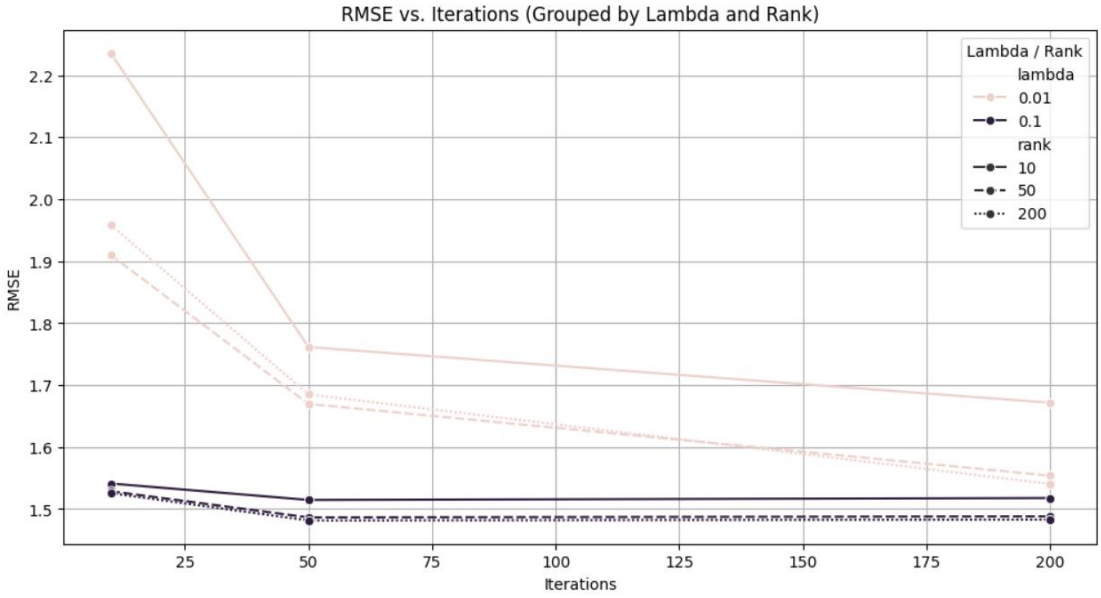



- Outcome: These initial visualizations provided a clear understanding of the rating landscape, item popularity, and user activity within the sampled dataset.

- **4.2. Model Performance Visualization (Hyperparameter Tuning):**
  - Objective: To analyze how different ALS hyperparameter settings (rank, iterations, lambda) impacted the model's performance, specifically the Root Mean Square Error (RMSE). This was based on the results collected during the hyperparameter tuning loop (Cell 9) and stored in results_df_display.
  - Process & Visualizations:
    - RMSE vs. Rank: A line plot was generated showing the relationship between the rank hyperparameter and the resulting RMSE. Lines were differentiated by lambda (color) and iterations (style) to observe their combined influence.




- RMSE vs. Iterations: A similar line plot was created to show the relationship between the number of iterations and RMSE, with lines differentiated by lambda (color) and rank (style).




  - Outcome: These plots helped in visually identifying trends, potential optimal ranges for hyperparameters, and understanding the sensitivity of the ALS model's performance to changes in these parameters. For instance, it allowed for observation of whether increasing rank or iterations consistently improved RMSE, and how this was affected by the regularization parameter lambda.

## 5. Model Training (Alternating Least Squares - ALS)

This phase focused on training the Alternating Least Squares (ALS) collaborative filtering model using various hyperparameter configurations to identify the optimal setup for providing book recommendations. The training process was designed to be iterative and to leverage Spark's distributed computing capabilities.

- **5.1. Hyperparameter Tuning Setup:**
    - Objective: To define a grid of hyperparameters for the ALS algorithm and to manage the training progress, allowing for resumption and avoiding redundant computations.
    - Process:
        - Progress Loading: The system first checked for an existing progress file (RESULTS_JSON_PATH). If found, previously completed hyperparameter combinations and their results (including RMSE and MSE) were loaded. This allowed the tuning process to resume from where it left off or to skip already evaluated configurations.
        - Hyperparameter Grid Definition: A set of values for key ALS hyperparameters was defined:
            - ranks =: The number of latent factors to compute.
            - iterations =: The number of iterations for the ALS algorithm to run.
            - lambdas = [0.01, 0.1]: The regularization parameter to prevent overfitting.
        - Best Model Initialization: Variables to store the best model object (best_model_obj_als), its RMSE (best_rmse_val), and its parameters (best_model_params_info) were initialized, potentially using values from loaded progress.

- **5.2. Iterative Model Training and Evaluation Loop:**
    - Objective: To systematically train and evaluate ALS models for each combination of hyperparameters in the defined grid.
    - Process:
        - A nested loop iterated through all combinations of rank, iterations, and lambda.
        - Skipping Processed Combinations: Before training, each combination was checked against the processed_params_set. If already evaluated, the current iteration was skipped with a notification.
        - ALS Model Instantiation: For each new combination, an ALS model instance was created with:
            - userCol="user_id_indexed", itemCol="item_id_indexed", ratingCol="rating".
            - coldStartStrategy="drop": To handle users/items in the test set not present in the training set by dropping these predictions during evaluation.
            - seed=YOUR_STUDENT_ID_LAST_4_DIGITS: For reproducibility of the ALS algorithm's internal random initializations.
            - nonnegative=True: To ensure that the latent factors are non-negative.
        - Model Fitting: The ALS model was trained (fitted) on the training_final_df . The start and end times were recorded to calculate the training duration.
        - Prediction & Evaluation: The trained fitted_model was used to make predictions on the test_final_df. These predictions were then evaluated using RegressionEvaluator to compute MSE and RMSE.
        - Result Logging: The parameters, computed MSE and RMSE, training duration, and a timestamp for the current model run were stored in a dictionary and appended to results_list.
        - Progress Saving: After each successful model training and evaluation, the updated results_list was saved to RESULTS_JSON_PATH to ensure that progress was not lost in case of interruption.

- **5.3. Best Model Selection and Saving:**
  - Objective: To identify the hyperparameter combination yielding the lowest RMSE and to save this best model for future use.
  - Process:
    - Within the training loop, if a newly trained model's RMSE (rmse_val) was lower than the current best_rmse_val:
      - best_rmse_val was updated.
      - The current fitted_model was stored as best_model_obj_als.
      - The parameters and metrics of this new best model were stored in best_model_params_info.
      - The best_model_obj_als (the Spark model object) was saved to the predefined BEST_MODEL_SAVE_PATH using best_model_obj_als.write().overwrite().save(). This allows the model to be loaded later without retraining.
      - The best_model_params_info dictionary was saved to BEST_MODEL_INFO_PATH as a JSON file, providing a human-readable record of the best model's configuration and performance.

- **5.4. Quantitative Performance Summary:**
  - Objective: To get a comparative overview of all trained model configurations and their performance metrics (MSE and RMSE).
  - Process:
    - All results collected in results_list (from current and previous runs) were consolidated into a Pandas DataFrame (results_df_display).
    - This DataFrame was sorted by rmse (ascending) and then by model_no to clearly identify the best performing models.
    - The complete table of results, including model number, rank, iterations, lambda, MSE, RMSE, training duration, and timestamp, was printed.

In [9]:

```
import json # JSON işlemleri için

if 'training_final_df' in locals() and 'test_final_df' in locals() and training_final_df.count() > 0:
    print("\n--- ALS Model Eğitimi ve Değerlendirme Döngüsü ---")

    results_list = []
    processed_params_set = set()

    if os.path.exists(RESULTS_JSON_PATH):
        print(f"Mevcut ilerleme dosyası bulundu: {RESULTS_JSON_PATH}. Yükleniyor...")
        try:
            with open(RESULTS_JSON_PATH, 'r') as f:
                results_list = json.load(f) # JSON'dan liste olarak oku
            for res in results_list:
                # processed_params_set için tipleri doğru al
                param_tuple_load = (
                    int(res.get('rank', 0)),
                    int(res.get('iterations', 0)),
                    float(res.get('lambda', 0.0))
                )
                processed_params_set.add(param_tuple_load)
            print(f"{len(results_list)} adet önceki model sonucu (ilerleme) yüklendi.")
        except json.JSONDecodeError:
            print(f"{RESULTS_JSON_PATH} geçerli bir JSON değil veya boş. Yeni ilerleme ile devam edilecek.")
            results_list = []
        except Exception as e:
            print(f"İlerleme yüklenirken hata: {e}. Yeni ilerleme ile devam edilecek.")
            log_error_message(f"İlerleme yüklenirken hata: {e}")
            results_list = []

    # Parametreler (proje gereksinimlerine göre güncelledim)
    ranks = [10, 50, 200]
    iterations = [10, 50, 200]
    lambdas = [0.01, 0.1]

    best_model_obj_als = None
    best_rmse_val = float('inf')
    best_model_params_info = {} # En iyi modelin bilgilerini saklamak için

    if results_list: # Önceki RMSE'yi kontrol et
        valid_rmses = [r['rmse'] for r in results_list if r.get('rmse') is not None and pd.notna(r['rmse'])]
        if valid_rmses:
            best_rmse_val = min(valid_rmses)
            # En iyi RMSE'ye sahip olan modelin bilgilerini de bul
            for res in results_list:
                if res.get('rmse') == best_rmse_val:
                    best_model_params_info = res.copy() # Kopyasını al
                    break
            print(f"Önceki sonuçlardan bulunan en iyi RMSE: {best_rmse_val:.4f}")
            if best_model_params_info:
                 print(f"Bu RMSE'ye ait parametreler: {best_model_params_info}")


    evaluator_mse = RegressionEvaluator(metricName="mse", labelCol="rating", predictionCol="prediction")
    evaluator_rmse = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
```

```
model_count_total = len(ranks) * len(iterations) * len(lambdas)
current_model_idx = 0

if results_list: # Model no için
    valid_model_nos = [int(r['model_no']) for r in results_list if r.get('model_no') is not None and isinstance(r.get('model_no'), (int, float, str)) and str(r.get('model_no')).isdigit()]
    last_model_no = max(valid_model_nos) if valid_model_nos else 0
else:
    last_model_no = 0

for rank_val in ranks:
    for iter_val in iterations:
        for lambda_val in lambdas:
            param_tuple_check = (int(rank_val), int(iter_val), float(lambda_val))

            if param_tuple_check in processed_params_set:
                print(f"\n--- Parametreler: Rank={rank_val}, Iter={iter_val}, Lambda={lambda_val} DAHA ÖNCE İŞLENMİŞ. Atlanıyor...")
                continue

            current_model_idx +=1
            effective_model_no = last_model_no + current_model_idx

            print(f"\n--- Model {effective_model_no} (Yeni Denenen: {current_model_idx}/{model_count_total - len(processed_params_set)}) ---")
            print(f"Parametreler: Rank={rank_val}, Iterations={iter_val}, Lambda={lambda_val}")

            als_model_loop = ALS(
                maxIter=iter_val, regParam=lambda_val, rank=rank_val,
                userCol="user_id_indexed", itemCol="item_id_indexed", ratingCol="rating",
                coldStartStrategy="drop", seed=YOUR_STUDENT_ID_LAST_4_DIGITS, nonnegative=True
            )

            try:
                start_time = datetime.datetime.now()
                print(f"Model eğitimi başladı: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")
                fitted_model = als_model_loop.fit(training_final_df)
                end_time = datetime.datetime.now()
                training_duration = (end_time - start_time).total_seconds()
                print(f"Model eğitimi tamamlandı. Süre: {training_duration:.2f} saniye.")

                print("Test verisi üzerinde tahminler yapılıyor...")
                predictions_loop = fitted_model.transform(test_final_df)
                predictions_cleaned_loop = predictions_loop.na.drop(subset=["prediction"])

                mse_val, rmse_val = float('nan'), float('nan')

                if predictions_cleaned_loop.count() == 0:
                    print(f"UYARI: Model için geçerli tahmin üretilemedi.")
                else:
                    mse_val = evaluator_mse.evaluate(predictions_cleaned_loop)
                    rmse_val = evaluator_rmse.evaluate(predictions_cleaned_loop)
                    print(f"Model Sonuçları - MSE: {mse_val:.4f}, RMSE: {rmse_val:.4f}")

                current_result_dict = {
                    "model_no": effective_model_no, "rank": rank_val, "iterations": iter_val,
                    "lambda": lambda_val, "mse": mse_val, "rmse": rmse_val,
                    "training_duration_sec": training_duration,
                    "timestamp": end_time.strftime('%Y-%m-%d %H:%M:%S')
                }
```

```
                    }
                    results_list.append(current_result_dict)
                    processed_params_set.add(param_tuple_check)

                    if not pd.isna(rmse_val) and rmse_val < best_rmse_val:
                        best_rmse_val = rmse_val
                        best_model_obj_als = fitted_model
                        best_model_params_info = current_result_dict.copy() # En iyi modelin bilgilerini güncelle
                        print(f"YENİ EN İYİ MODEL! RMSE: {best_rmse_val:.4f} (Model No: {effective_model_no})")

                        try:
                            best_model_obj_als.write().overwrite().save(BEST_MODEL_SAVE_PATH)
                            print(f"Yeni en iyi model '{BEST_MODEL_SAVE_PATH}' yoluna kaydedildi.")
                            # En iyi modelin bilgilerini best_model_info.json'a yaz
                            with open(BEST_MODEL_INFO_PATH, 'w') as f_info:
                                json.dump(best_model_params_info, f_info, indent=4)
                            print(f"En iyi model bilgileri '{BEST_MODEL_INFO_PATH}' dosyasına kaydedildi.")
                        except Exception as save_e:
                            error_msg = f"En iyi model kaydedilirken/bilgi yazılırken sorun: {save_e}"
                            print(f"HATA: {error_msg}")
                            log_error_message(error_msg)

                    # Her başarılı modelden sonra ilerlemeyi (progress.json) kaydet
                    try:
                        with open(RESULTS_JSON_PATH, 'w') as f_prog:
                            json.dump(results_list, f_prog, indent=4)
                        print(f"Güncel ilerleme '{RESULTS_JSON_PATH}' dosyasına kaydedildi.")
                    except Exception as prog_save_e:
                        error_msg_prog = f"İlerleme dosyası kaydedilirken hata: {prog_save_e}"
                        print(f"HATA: {error_msg_prog}")
                        log_error_message(error_msg_prog)


                except Exception as e:
                    error_msg_train = f"Model (R:{rank_val},I:{iter_val},L:{lambda_val}) eğitilirken/değerlendirilirken: {e}"
                    print(f"HATA: {error_msg_train}")
                    log_error_message(error_msg_train)
                    # İsteğe bağlı: Hatalı denemeyi de results_list'e ekleyip progress.json'a yazabilirsiniz
                    # results_list.append({
                    #     "model_no": effective_model_no, "rank": rank_val, "iterations": iter_val,
                    #     "lambda": lambda_val, "error": str(e),
                    #     "timestamp": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                    # })
                    # with open(RESULTS_JSON_PATH, 'w') as f_prog_err:
                    #    json.dump(results_list, f_prog_err, indent=4)

    print("\n--- Tüm Modellerin Sonuçları (İlerleme Dosyasından) ---")
    if results_list:
        # Pandas DataFrame'e çevirip göstermek için
        results_df_display = pd.DataFrame(results_list)
        results_df_display = results_df_display.sort_values(
            by=['rmse', 'model_no'],
            ascending=[True, True],
            na_position='last' # NaN RMSE'leri sona at
        )
        print(results_df_display.to_string())
```

```
    print("\n--- Tüm Modellerin Sonuçları (İlerleme Dosyasından) ---")
    if results_list:
        # Pandas DataFrame'e çevirip göstermek için
        results_df_display = pd.DataFrame(results_list)
        results_df_display = results_df_display.sort_values(
            by=['rmse', 'model_no'],
            ascending=[True, True],
            na_position='last' # NaN RMSE'leri sona at
        )
        print(results_df_display.to_string())

        if best_model_params_info: # En son güncellenen best_model_params_info'yu kullan
            print(f"\n--- En İyi Model (Metriklere Göre Saklanan Bilgiler) ---")
            print(f"Parametreler: Rank={best_model_params_info.get('rank')}, Iterations={best_model_params_info.get('iterations')}, Lambda={best_model_params_info.get('lambda')}")
            print(f"En İyi RMSE: {best_model_params_info.get('rmse'):.4f} (Model No: {best_model_params_info.get('model_no')})")
        elif os.path.exists(BEST_MODEL_INFO_PATH): # Eğer best_model_params_info boşsa ama dosya varsa oradan oku
             try:
                with open(BEST_MODEL_INFO_PATH, 'r') as f_info_read:
                    best_model_params_info = json.load(f_info_read)
                print(f"\n--- En İyi Model (Kayıtlı Bilgi Dosyasından Yüklendi) ---")
                print(f"Parametreler: Rank={best_model_params_info.get('rank')}, Iterations={best_model_params_info.get('iterations')}, Lambda={best_model_params_info.get('lambda')}")
                print(f"En İyi RMSE: {best_model_params_info.get('rmse'):.4f} (Model No: {best_model_params_info.get('model_no')})")
             except Exception as read_info_e:
                print(f"Kayıtlı en iyi model bilgisi okunurken hata: {read_info_e}")
                log_error_message(f"Kayıtlı en iyi model bilgisi ({BEST_MODEL_INFO_PATH}) okunurken hata: {read_info_e}")


    if best_model_obj_als:
        print(f"\n'best_model_obj_als' en iyi modeli içeriyor. RMSE: {best_rmse_val:.4f}")
    elif os.path.exists(BEST_MODEL_SAVE_PATH) and os.path.isdir(BEST_MODEL_SAVE_PATH):
        print(f"\n'best_model_obj_als' None, ancak '{BEST_MODEL_SAVE_PATH}' yolunda kayıtlı bir model bulundu.")
    else:
        print("\nHiçbir başarılı model eğitilemedi, 'best_model_obj_als' None ve kayıtlı model bulunamadı.")
else:
    print("UYARI: 'training_final_df' veya 'test_final_df' bulunamadı veya boş. Model eğitimi atlanıyor.")
```

```
--- ALS Model Eğitimi ve Değerlendirme Döngüsü ---
Mevcut ilerleme dosyası bulundu: project/goodreads_models/progress.json. Yükleniyor...
18 adet önceki model sonucu (ilerleme) yüklendi.
Önceki sonuçlardan bulunan en iyi RMSE: 1.4816
Bu RMSE'ye ait parametreler: {'model_no': 16, 'rank': 200, 'iterations': 50, 'lambda': 0.1, 'mse': 2.1952746325194052, 'rmse': 1.4816459200900212, 'training_duration_sec': 2013.558835, 'timestamp': '2025-05-29 02:39:02'}

--- Parametreler: Rank=10, Iter=10, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=10, Iter=10, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=10, Iter=50, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=10, Iter=50, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=10, Iter=200, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=10, Iter=200, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=10, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=10, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=50, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=50, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=200, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=50, Iter=200, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=10, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=10, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=50, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=50, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=200, Lambda=0.01 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...

--- Parametreler: Rank=200, Iter=200, Lambda=0.1 DAHA ÖNCE İŞLENMİŞ. Atlanıyor...
```

```
--- Tüm Modellerin Sonuçları (İlerleme Dosyasından) ---
```

| | model_no | rank | iterations | lambda | mse | rmse | training_duration_sec | timestamp |
|---|---|---|---|---|---|---|---|---|
| 15 | 16 | 200 | 50 | 0.10 | 2.195275 | 1.481646 | 2013.558835 | 2025-05-29 02:39:02 |
| 17 | 18 | 200 | 200 | 0.10 | 2.199403 | 1.483038 | 8144.169123 | 2025-05-29 08:24:09 |
| 9 | 10 | 50 | 50 | 0.10 | 2.209808 | 1.486542 | 191.197721 | 2025-05-28 21:02:25 |
| 11 | 12 | 50 | 200 | 0.10 | 2.214733 | 1.488198 | 727.366289 | 2025-05-28 21:30:08 |
| 3 | 4 | 10 | 50 | 0.10 | 2.294386 | 1.514723 | 74.897171 | 2025-05-28 20:41:45 |
| 5 | 6 | 10 | 200 | 0.10 | 2.303708 | 1.517797 | 259.448414 | 2025-05-28 20:51:20 |
| 13 | 14 | 200 | 10 | 0.10 | 2.327752 | 1.525697 | 501.431408 | 2025-05-28 22:01:53 |
| 7 | 8 | 50 | 10 | 0.10 | 2.337243 | 1.528804 | 50.808603 | 2025-05-28 20:54:06 |
| 16 | 17 | 200 | 200 | 0.01 | 2.373466 | 1.540606 | 12461.073737 | 2025-05-29 06:07:38 |
| 1 | 2 | 10 | 10 | 0.10 | 2.375499 | 1.541266 | 24.594411 | 2025-05-28 20:38:30 |
| 10 | 11 | 50 | 200 | 0.01 | 2.414762 | 1.553950 | 885.246199 | 2025-05-28 21:17:36 |
| 8 | 9 | 50 | 50 | 0.01 | 2.787811 | 1.669674 | 260.488919 | 2025-05-28 20:58:50 |
| 4 | 5 | 10 | 200 | 0.01 | 2.794285 | 1.671612 | 273.542115 | 2025-05-28 20:46:40 |
| 14 | 15 | 200 | 50 | 0.01 | 2.839103 | 1.684964 | 4288.742056 | 2025-05-29 02:04:43 |
| 2 | 3 | 10 | 50 | 0.01 | 3.102760 | 1.761465 | 77.927497 | 2025-05-28 20:40:09 |
| 6 | 7 | 50 | 10 | 0.01 | 3.648276 | 1.910046 | 72.411992 | 2025-05-28 20:52:53 |
| 12 | 13 | 200 | 10 | 0.01 | 3.835922 | 1.958551 | 1324.861174 | 2025-05-28 21:52:39 |
| 0 | 1 | 10 | 10 | 0.01 | 4.998173 | 2.235659 | 28.902374 | 2025-05-28 20:37:38 |

```
--- En İyi Model (Metriklere Göre Saklanan Bilgiler) ---
Parametreler: Rank=200, Iterations=50, Lambda=0.1
En İyi RMSE: 1.4816 (Model No: 16)

'best_model_obj_als' None, ancak 'project/goodreads_models/best_model' yolunda kayıtlı bir model bulundu.
```

## 6. Model Evaluation & Analysis

Once the hyperparameter tuning loop for the ALS models was completed (or previous progress was loaded), this phase focused on evaluating the performance of the trained models, selecting the best one, and conducting further analysis to understand its recommendation capabilities.

- **6.1 Loading and Verifying the Best Model:**
  - Objective: To ensure the best identified ALS model object (best_model_obj_als) and its parameters (best_model_params_info) were correctly loaded into the current session for further analysis, especially if the notebook was run in segments or restarted.
  - Process:
    - The code checked if best_model_obj_als was already populated.
    - If not, it attempted to load the saved Spark ALSModel from BEST_MODEL_SAVE_PATH.
    - Similarly, if best_model_params_info was not available, it attempted to load it from BEST_MODEL_INFO_PATH.

In [11]:
```python
from pyspark.ml.recommendation import ALSModel

# Bu hücre, 'best_model_obj_als' None ise veya mevcut değilse modeli yüklemeye çalışır.
# 'best_model_params_info' da global veya bir önceki hücreden aktarılmış olmalı.

model_loaded_successfully = False
if 'best_model_obj_als' not in globals() or best_model_obj_als is None: # globals() ile kontrol daha güvenli
    if os.path.exists(BEST_MODEL_SAVE_PATH) and os.path.isdir(BEST_MODEL_SAVE_PATH):
        print(f"'{BEST_MODEL_SAVE_PATH}' yolundan en iyi model yükleniyor...")
        try:
            best_model_obj_als = ALSModel.load(BEST_MODEL_SAVE_PATH)
            print("En iyi Spark modeli başarıyla yüklendi.")
            model_loaded_successfully = True

            # En iyi modelin bilgilerini de yükle (eğer best_model_params_info boşsa)
            if ('best_model_params_info' not in globals() or not best_model_params_info) and os.path.exists(BEST_MODEL_INFO_PATH):
                try:
                    with open(BEST_MODEL_INFO_PATH, 'r') as f_info_read_load:
                        best_model_params_info = json.load(f_info_read_load) # global best_model_params_info'yu güncelle
                    print(f"En iyi model bilgileri '{BEST_MODEL_INFO_PATH}' dosyasından yüklendi.")
                except Exception as read_info_e_load:
                    error_msg_load_info = f"Model yüklenirken bilgi dosyası ({BEST_MODEL_INFO_PATH}) okunurken hata: {read_info_e_load}"
                    print(f"HATA: {error_msg_load_info}")
                    log_error_message(error_msg_load_info) # Hata log dosyasına yaz
        except Exception as load_e:
            error_msg_load_model = f"Spark modeli yüklenirken sorun oluştu: {load_e}"
            print(f"HATA: {error_msg_load_model}")
            log_error_message(error_msg_load_model) # Hata log dosyasına yaz
            best_model_obj_als = None
    else:
        print(f"Kayıtlı model '{BEST_MODEL_SAVE_PATH}' yolunda bulunamadı. 'best_model_obj_als' hala None.")
elif best_model_obj_als:
    print("'best_model_obj_als' zaten mevcut bir Spark modeli içeriyor. Yükleme yapılmadı.")
    model_loaded_successfully = True # Zaten yüklü veya mevcut

if model_loaded_successfully and best_model_params_info:
    print("\nKullanıma Hazır En İyi Model Bilgileri:")
    print(f"  Rank: {best_model_params_info.get('rank')}, Iter: {best_model_params_info.get('iterations')}, Lambda: {best_model_params_info.get('lambda')}")
    print(f"  RMSE: {best_model_params_info.get('rmse'):.4f}, Model No: {best_model_params_info.get('model_no')}")
elif model_loaded_successfully:
    print("Spark modeli mevcut/yüklendi ancak detaylı parametre bilgisi (best_model_params_info) bulunamadı.")
else:
    print("Kullanılacak Spark modeli bulunamadı/yüklenemedi.")
```

```
'project/goodreads_models/best_model' yolundan en iyi model yükleniyor...
En iyi Spark modeli başarıyla yüklendi.

Kullanıma Hazır En İyi Model Bilgileri:
  Rank: 200, Iter: 50, Lambda: 0.1
  RMSE: 1.4816, Model No: 16
```

- **6.2. Qualitative Analysis: Prediction vs. Actual Ratings**
  - Objective: To manually inspect and qualitatively assess the predictions made by the best model on a sample of the test data.
  - Process:
    - The best model (best_model_obj_als) was used to generate predictions on the test_final_df.

```python
predictions_best_model = best_model_obj_als.transform(test_final_df)
```

    - IndexToString was utilized to convert the indexed user_id_indexed and item_id_indexed back to their original string representations (original_user_id_retrieved, original_book_id_retrieved).

```python
# User ID dönüştürücü (StringIndexerModel'den labels alarak)
user_converter_compare = IndexToString(
    inputCol=best_model_obj_als.getUserCol(), # Modelin userCol adını kullan
    outputCol="original_user_id_retrieved",
    labels=user_indexer_model.labels
)
# Item ID dönüştürücü
item_converter_compare = IndexToString(
    inputCol=best_model_obj_als.getItemCol(), # Modelin itemCol adını kullan
    outputCol="original_book_id_retrieved",
    labels=item_indexer_model.labels
)

# Dönüşümleri uygula
predictions_with_ids = user_converter_compare.transform(predictions_best_model_cleaned)
predictions_with_ids = item_converter_compare.transform(predictions_with_ids)
```

- A sample of predictions, including the original user/book IDs, the actual rating, and the model's prediction, was displayed.

```
predictions_with_ids.select(
    "original_user_id_retrieved",
    "original_book_id_retrieved",
    "rating",  # Gerçek rating
    "prediction" # Tahmin edilen rating
).show(20, truncate=False) # İlk 20 örneği göster
```

- **6.3. Item-to-Item Recommendation using Cosine Similarity:**
  - Objective: To demonstrate how item embeddings (factor vectors) from the ALS model can be used to find items similar to a given item.
  - Process:
    - A target item was selected (in the notebook, the first item from itemFactors was used, which corresponded to original book ID "2767052").
    - The factor vector of this target item was retrieved.
    - Cosine similarity was calculated between the target item's factor vector and the factor vectors of all other items using a Spark UDF.
    - The top 5 books most similar to the target book were identified and displayed.

- **6.4. User-Item Recommendation (Predicting Users for a Specific Item):**
  - Objective: To demonstrate how user and item embeddings can be used to predict which users are most likely to prefer a specific item.
  - Process:
    - Using the same target item (e.g., book ID "2767052") from the previous step.
    - User factor vectors were retrieved from the model.
    - The dot product between each user's factor vector and the target item's factor vector was calculated (via a UDF) to estimate a rating (prediction score).
    - Users were ranked based on these predicted scores, and the top 10 users most likely to prefer the target book were displayed with their original IDs and prediction scores.

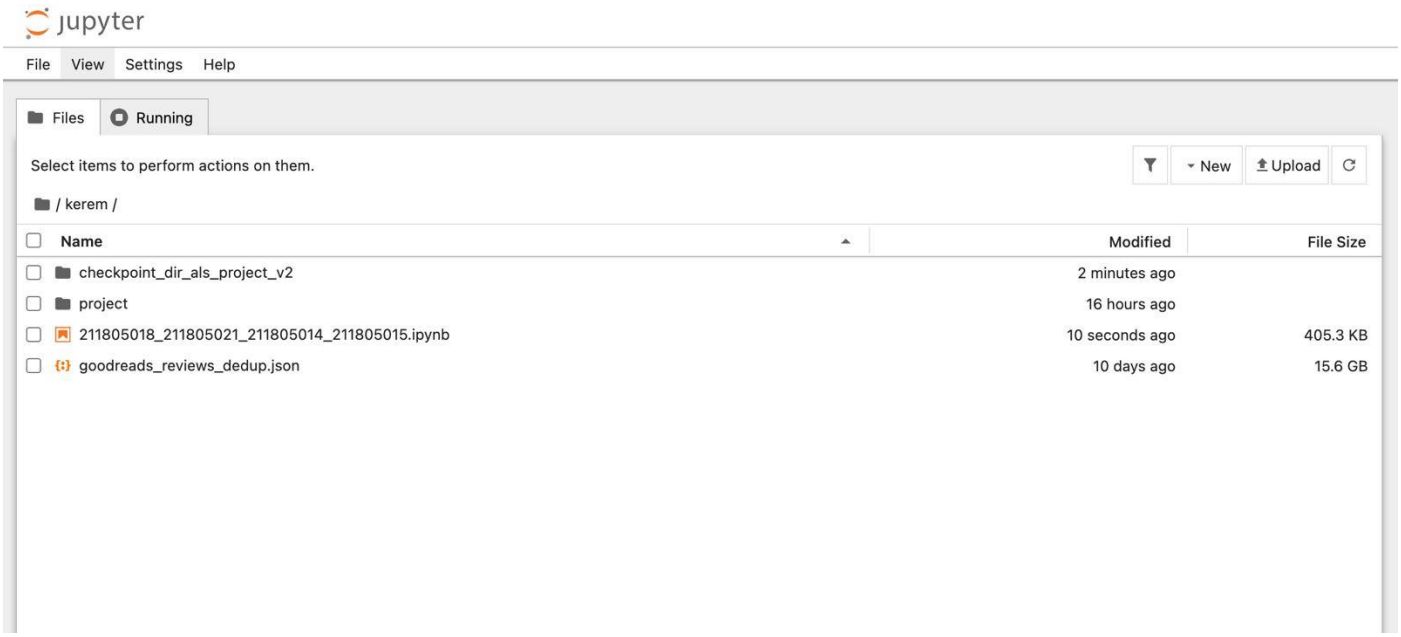
In [13]:
```
# Hücre 11 (veya uygun bir sonraki hücre): Kosinüs Benzerliği ve X Ürününü Sevecek Kullanıcılar (Sonuç Odaklı)

import numpy as np
from pyspark.sql.functions import col, lit, desc, explode
from pyspark.sql.types import FloatType, IntegerType, DoubleType
from pyspark.ml.linalg import DenseVector
from pyspark.ml.feature import IndexToString
from pyspark.ml.recommendation import ALSModel

# Kosinüs benzerliği için helper fonksiyon (hücre başında tanımlı olmalı)
def cosine_similarity_np(vec1_np, vec2_np):
    """İki NumPy vektörü arasındaki kosinüs benzerliğini hesaplar."""
    if not isinstance(vec1_np, np.ndarray) or not isinstance(vec2_np, np.ndarray):
        return 0.0
    dot_product = np.dot(vec1_np, vec2_np)
    norm_vec1 = np.linalg.norm(vec1_np)
    norm_vec2 = np.linalg.norm(vec2_np)
    if norm_vec1 == 0 or norm_vec2 == 0:
        return 0.0
    return dot_product / (norm_vec1 * norm_vec2)

# Gerekli ana değişkenlerin varlığını kontrol et
required_vars_exist_initial = all(var_name in globals() and globals()[var_name] is not None
                                  for var_name in ['BEST_MODEL_SAVE_PATH', 'item_indexer_model', 'user_indexer_model', 'spark'])

best_model_obj_als = None

if required_vars_exist_initial:
    print(f"'{BEST_MODEL_SAVE_PATH}' yolundan en iyi model yükleniyor...")
    try:
        loaded_model_for_cell_11 = ALSModel.load(BEST_MODEL_SAVE_PATH)
        best_model_obj_als = loaded_model_for_cell_11
        print("Model başarıyla yüklendi ve best_model_obj_als güncellendi.")
    except Exception as e_load_model:
        print(f"HATA: Model yüklenirken sorun oluştu: {e_load_model}")
        if 'log_error_message' in globals(): log_error_message(f"Model yükleme hatası (Hücre 11): {e_load_model}")

if best_model_obj_als is not None:
    print("\n--- Kosinüs Benzerliği ve Ürün X İçin Kullanıcı Önerisi ---")

    model_user_col = best_model_obj_als.getUserCol()
    model_item_col = best_model_obj_als.getItemCol()
    print(f"Modelin kullandığı User Column: {model_user_col}")
    print(f"Modelin kullandığı Item Column: {model_item_col}")

    item_converter = IndexToString(inputCol=model_item_col, outputCol="original_book_id_retrieved", labels=item_indexer_model.labels)
    user_converter = IndexToString(inputCol=model_user_col, outputCol="original_user_id_retrieved", labels=user_indexer_model.labels)

    target_item_id_indexed_for_cosine = None
    target_item_original_id_for_cosine = "Bilinmeyen_Kitap_X_CosSim"

    # --- 1. Benzer Kitapların Bulunması ---
    item_factors_for_cosine_df = None
    try:
        print("\n--- 1. Benzer Kitapların Bulunması ---") # Temiz başlık
        item_factors_df_temp = best_model_obj_als.itemFactors
        if item_factors_df_temp is None or item_factors_df_temp.count() == 0:
            print("UYARI: best_model_obj_als.itemFactors boş veya None.")
            raise ValueError("Item faktörleri alınamadı veya boş.")

        item_factors_for_cosine_df = item_factors_df_temp.select(
            col("id").alias(model_item_col),
            col("features").alias("item_features_cosine")
        ).persist()

        if item_factors_for_cosine_df.count() > 0:
            target_item_row_cosine = item_factors_for_cosine_df.first()

            if target_item_row_cosine is None:
                print(f"UYARI: Hedef item (ilk item) faktörleri alınamadı.")
                raise ValueError("Hedef item faktörü alınamadı.")

            target_item_id_indexed_for_cosine = target_item_row_cosine[model_item_col]
            target_item_features_raw = target_item_row_cosine["item_features_cosine"]

            target_item_original_id_df_cosine = item_converter.transform(
                spark.createDataFrame([(target_item_id_indexed_for_cosine,)], [model_item_col])
            )
            target_item_original_id_for_cosine = target_item_original_id_df_cosine.first()["original_book_id_retrieved"]
            print(f"Hedef Kitap (Benzerlik için): Indexlenmiş ID {target_item_id_indexed_for_cosine} (Orijinal ID: {target_item_original_id_for_cosine})")

            if isinstance(target_item_features_raw, DenseVector):
                target_item_vector_np_cosine = np.array(target_item_features_raw.toArray())
            elif isinstance(target_item_features_raw, list):
                target_item_vector_np_cosine = np.array(target_item_features_raw)
            elif hasattr(target_item_features_raw, 'toArray'):
                target_item_vector_np_cosine = np.array(target_item_features_raw.toArray())
            else:
                err_msg_type = f"Hedef item için beklenmedik özellik vektörü tipi: {type(target_item_features_raw)}"
                print(f"HATA: {err_msg_type}")
                if 'log_error_message' in globals(): log_error_message(err_msg_type)
                raise TypeError(err_msg_type)

            def udf_cosine_similarity_internal(other_vector_features_raw):
                if other_vector_features_raw is None: return 0.0
                if isinstance(other_vector_features_raw, DenseVector):
                    other_vector_np = np.array(other_vector_features_raw.toArray())
                elif isinstance(other_vector_features_raw, list):
                    other_vector_np = np.array(other_vector_features_raw)
                elif hasattr(other_vector_features_raw, 'toArray'):
                    other_vector_np = np.array(other_vector_features_raw.toArray())
                else:
                    return 0.0
                return float(cosine_similarity_np(target_item_vector_np_cosine, other_vector_np))

            cosine_sim_udf_for_spark = udf(udf_cosine_similarity_internal, FloatType())

            similarities_df_all_items = item_factors_for_cosine_df.withColumn(
                "similarity", cosine_sim_udf_for_spark(col("item_features_cosine"))
            )

            similarities_df_filtered_items = similarities_df_all_items.filter(
                col(model_item_col) != target_item_id_indexed_for_cosine
```

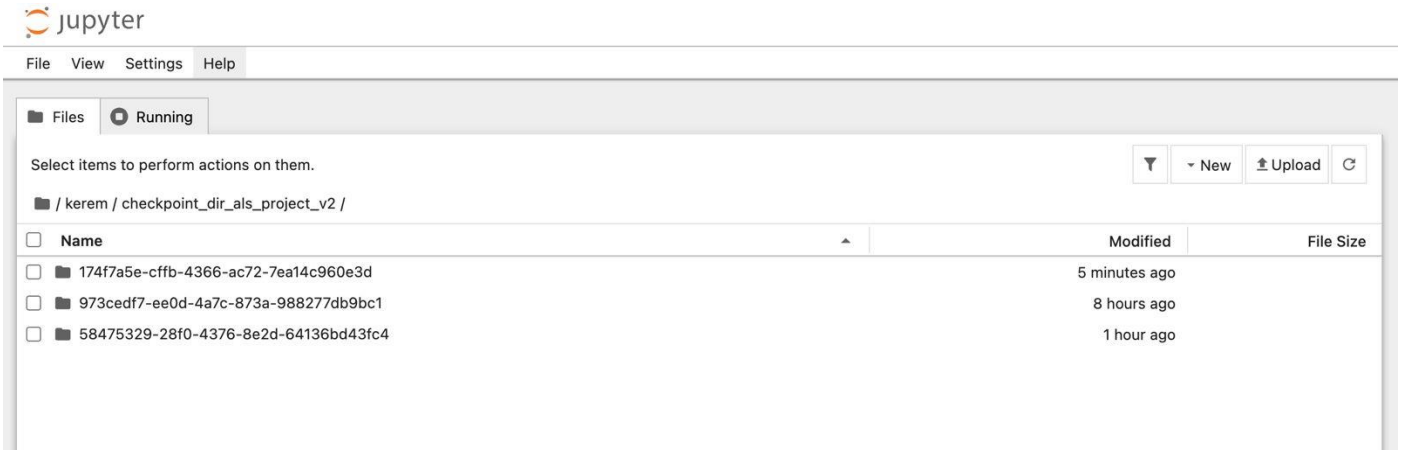
```
            )

            print(f"\n'{target_item_original_id_for_cosine}' adlı kitaba en benzer 5 kitap (Indexlenmiş ID ve Benzerlik):")
            top_5_similar_items_cosine = similarities_df_filtered_items.orderBy(
                col("similarity").desc_nulls_last()
            ).select(
                col(model_item_col).alias("id"),
                "similarity"
            ).limit(5)
            top_5_similar_items_cosine.show(truncate=False)
        else:
            print("UYARI: Item faktörleri kosinüs benzerliği için boş.")
            target_item_id_indexed_for_cosine = None

    except Exception as e_cos_sim_block:
        err_msg_cos = f"Benzer kitaplar hesaplanırken hata: {e_cos_sim_block}"
        print(f"HATA: {err_msg_cos}")
        if 'log_error_message' in globals(): log_error_message(err_msg_cos)
        target_item_id_indexed_for_cosine = None
    finally:
        if item_factors_for_cosine_df is not None:
            item_factors_for_cosine_df.unpersist()

    # --- 2. '{target_item_original_id_for_cosine}' Kitabını En Çok Sevecek Kullanıcılar ---
    if target_item_id_indexed_for_cosine is not None:
        user_factors_df = None
        target_item_factor_row_for_rec = None

        try:
            print(f"\n--- 2. '{target_item_original_id_for_cosine}' Kitabını En Çok Sevecek Kullanıcılar (Manuel Yöntem) ---")

            user_col_name = model_user_col
            item_col_name = model_item_col

            user_factors_df = best_model_obj_als.userFactors.select(
                col("id").alias(user_col_name),
                col("features").alias("user_features")
            ).persist()

            item_factors_all_for_rec_df = best_model_obj_als.itemFactors.select(
                 col("id").alias(item_col_name),
                 col("features").alias("item_features")
            )

            target_item_factor_row_for_rec = item_factors_all_for_rec_df.filter(col(item_col_name) == target_item_id_indexed_for_cosine).first()

            if user_factors_df.count() == 0:
                print("UYARI: User faktörleri boş.")
                raise ValueError("User faktörleri boş.")
            if not target_item_factor_row_for_rec:
                print(f"UYARI: Hedef item ID {target_item_id_indexed_for_cosine} için faktör bulunamadı (kullanıcı önerisi kısmı).")
                raise ValueError("Hedef item faktörü bulunamadı.")

            target_item_features_vector_for_rec_np = np.array(target_item_factor_row_for_rec["item_features"])

            def dot_product_udf_rec(user_f_list_rec):
                if user_f_list_rec is None:
                    return None
                user_f_np_rec = np.array(user_f_list_rec)
                return float(np.dot(user_f_np_rec, target_item_features_vector_for_rec_np))

            dot_product_spark_udf_rec = udf(dot_product_udf_rec, DoubleType())

            predictions_df_rec = user_factors_df.withColumn(
                "prediction", dot_product_spark_udf_rec(col("user_features"))
            )

            predictions_cleaned_df_rec = predictions_df_rec.na.drop(subset=["prediction"])

            print(f"'{target_item_original_id_for_cosine}' için en yüksek tahmini alan 10 kullanıcı (Orijinal ID, Indexlenmiş ID ve Tahmin):")

            top_10_users_indexed_rec = predictions_cleaned_df_rec.orderBy(desc("prediction")) \
                .select(col(user_col_name).alias("id_for_conversion"), col("prediction")).limit(10)

            top_10_users_with_original_id_rec = user_converter.transform(
                top_10_users_indexed_rec.withColumnRenamed("id_for_conversion", user_col_name)
            ).select("original_user_id_retrieved", col(user_col_name).alias("indexed_id"), "prediction")

            top_10_users_with_original_id_rec.show(truncate=False)

        except Exception as e_user_rec_block:
            print(f"HATA: Kullanıcı önerileri hesaplanırken hata: {e_user_rec_block}")
            if 'log_error_message' in globals(): log_error_message(f"Kullanıcı önerileri hatası: {e_user_rec_block}")
        finally:
            if user_factors_df is not None:
                user_factors_df.unpersist()

    elif target_item_id_indexed_for_cosine is None:
        print("UYARI: Hedef item ID'si belirlenemediği için kullanıcı önerileri yapılamıyor.")

else:
    print("UYARI: Model yüklenemediği için Kosinüs Benzerliği ve Kullanıcı Önerisi adımları atlanıyor.")
```

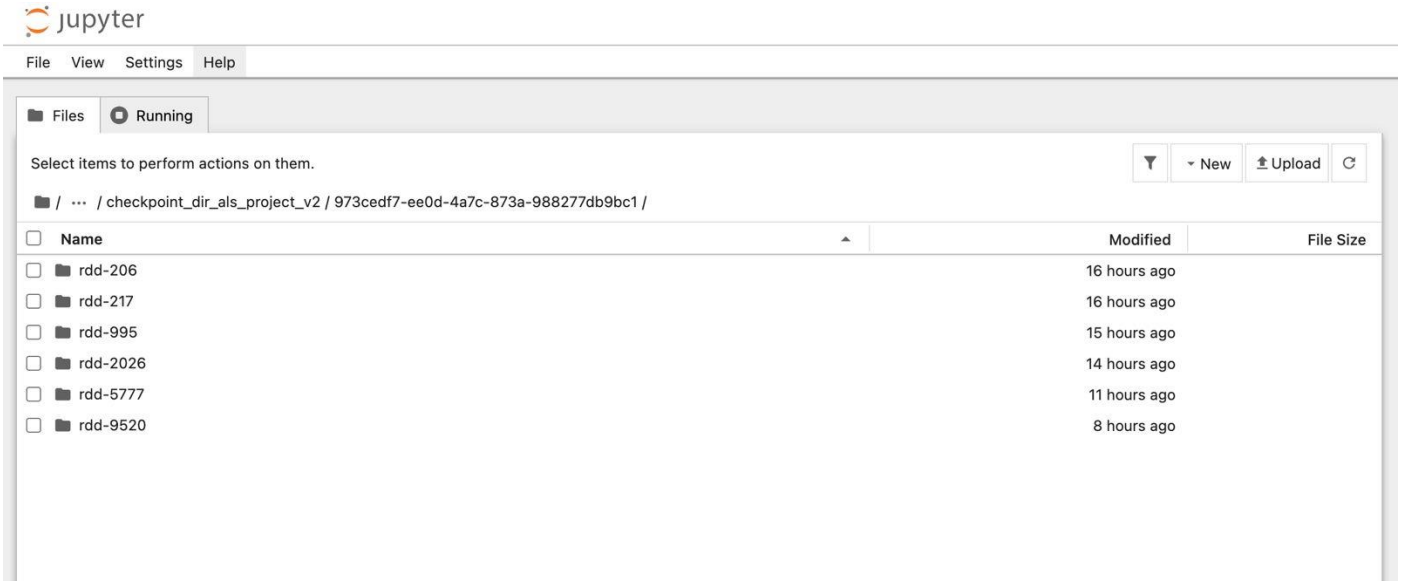
```
'project/goodreads_models/best_model' yolundan en iyi model yükleniyor...
Model başarıyla yüklendi ve best_model_obj_als güncellendi.

--- Kosinüs Benzerliği ve Ürün X İçin Kullanıcı Önerisi ---
Modelin kullandığı User Column: user_id_indexed
Modelin kullandığı Item Column: item_id_indexed

--- 1. Benzer Kitapların Bulunması ---
25/05/29 17:00:54 WARN DAGScheduler: Broadcasting large task binary with size 4.3 MiB
25/05/29 17:00:54 WARN DAGScheduler: Broadcasting large task binary with size 4.3 MiB
25/05/29 17:00:55 WARN DAGScheduler: Broadcasting large task binary with size 4.3 MiB
Hedef Kitap (Benzerlik için): Indexlenmiş ID 1 (Orijinal ID: 2767052)

'2767052' adlı kitaba en benzer 5 kitap (Indexlenmiş ID ve Benzerlik):

+------+----------+
|id    |similarity|
+------+----------+
|135002|0.99999547|
|141951|0.99999505|
|413649|0.99999464|
|178308|0.99999464|
|420567|0.99999446|
+------+----------+


--- 2. '2767052' Kitabını En Çok Sevecek Kullanıcılar (Manuel Yöntem) ---
'2767052' için en yüksek tahmini alan 10 kullanıcı (Orijinal ID, Indexlenmiş ID ve Tahmin):
[Stage 103:=============================================>           (8 + 2) / 10]
+--------------------------------+----------+-----------------+
|original_user_id_retrieved      |indexed_id|prediction       |
+--------------------------------+----------+-----------------+
|974393c92e52f1b573833221b8b06cc2|177543    |6.46620428506955 |
|76d354d3d049909862c35519b8e6e4fe|166417    |6.444799054846947|
|e77b9f7626aa66fc06e89009eac7dfad|205176    |6.374721096459405|
|3e0b015e6edcf9f5d552cf2b667df981|146857    |6.374721096459405|
|99f532d1004f9b747b17a716d77e515b|178506    |6.362013075515409|
|f5501aa6cd80880b998ebbc5fcfbc769|209969    |6.215206429876238|
|fa204d95b87630df6c58a80a44f7f4f3|211684    |6.153350596655537|
|076b8f5c5fdc1cbe872894a5c2ea38ce|127860    |6.127850535607053|
|cdf2096ea5bc866bf29cf85570b66275|196364    |6.098920717838127|
|4d1c73f5884168b21fab8e1545d5ab63|152147    |6.048563437301397|
+--------------------------------+----------+-----------------+
```

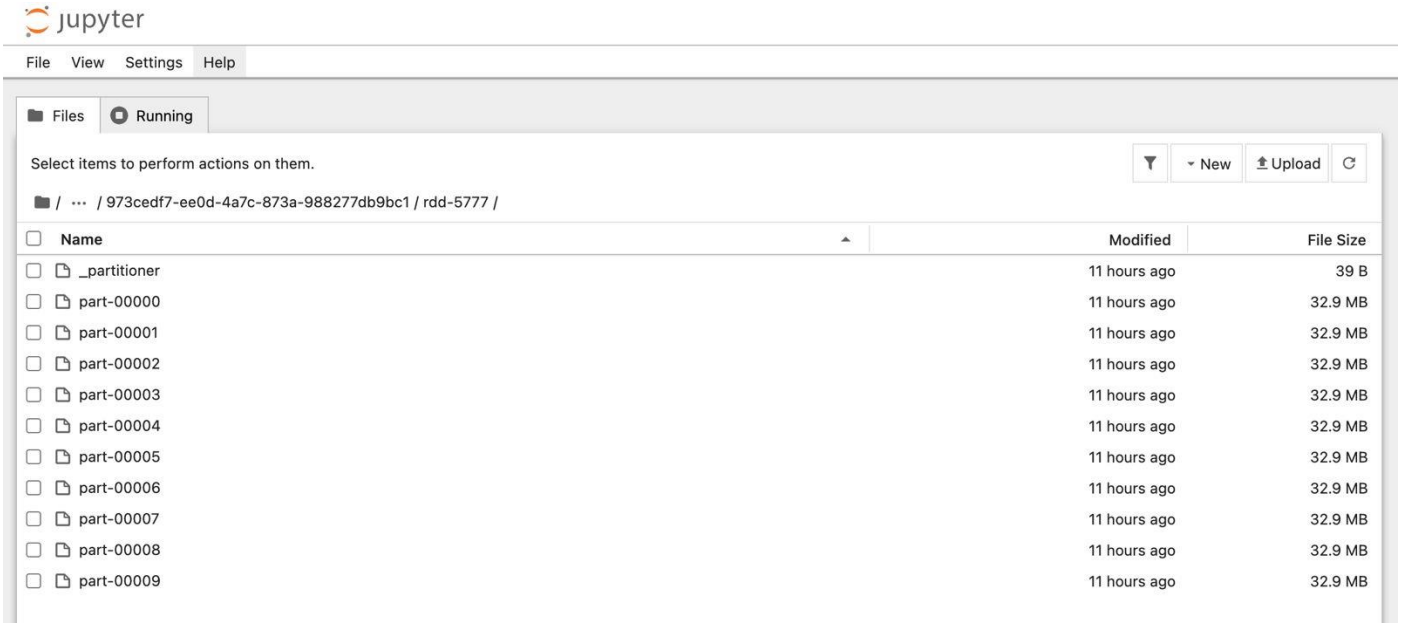
- Outcome: This comprehensive evaluation provided both quantitative (RMSE/MSE across different configurations) and qualitative (sample predictions, similarity-based recommendations) insights into the ALS model's performance and capabilities on the Goodreads dataset.

## *Δ File Directory of Checkpoints*

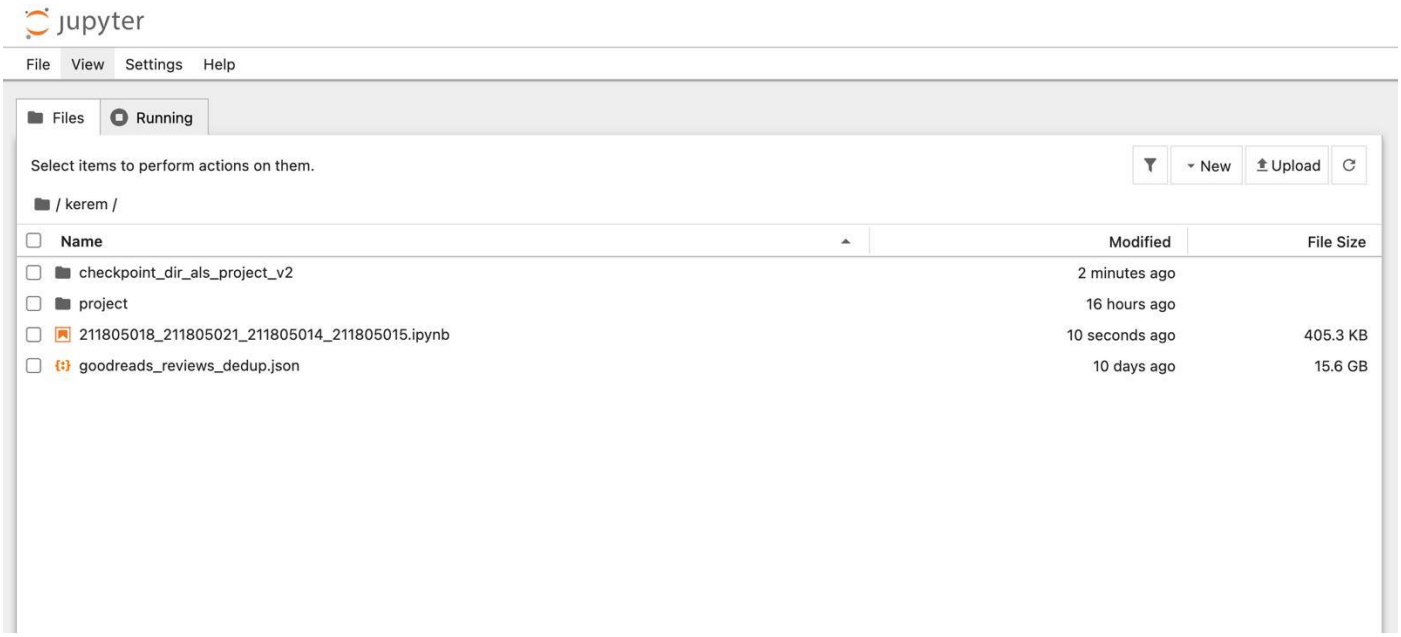


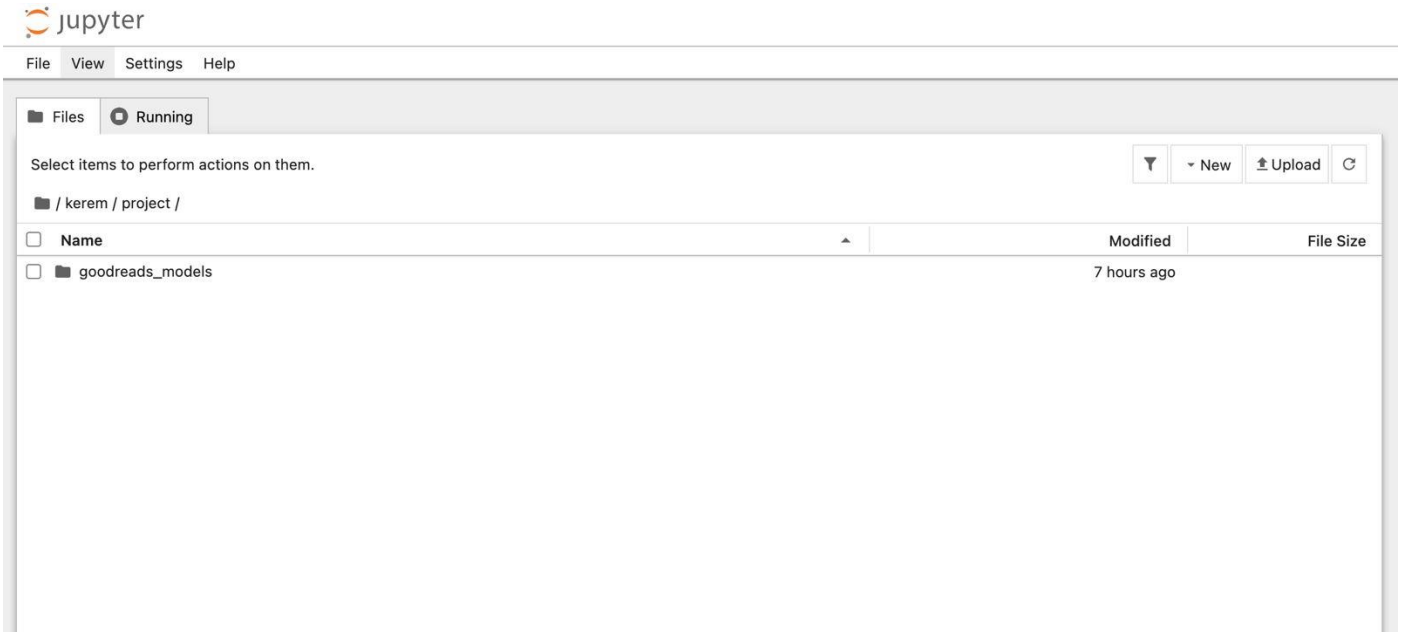



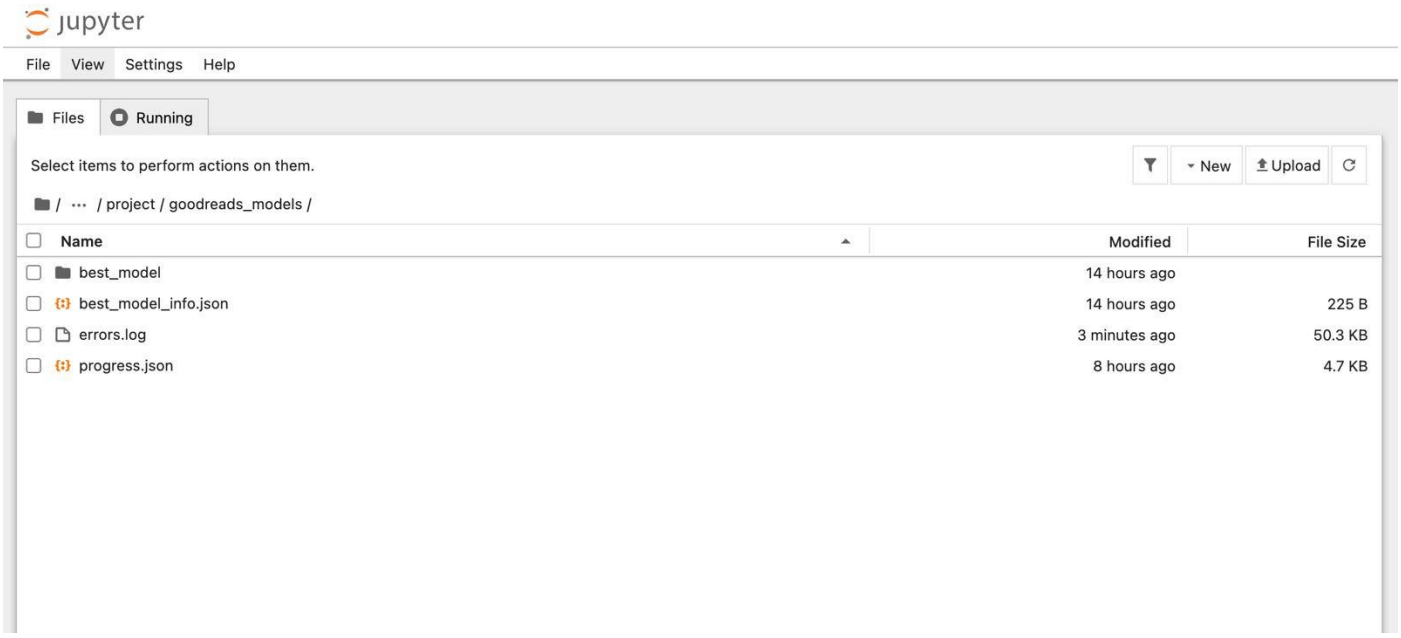


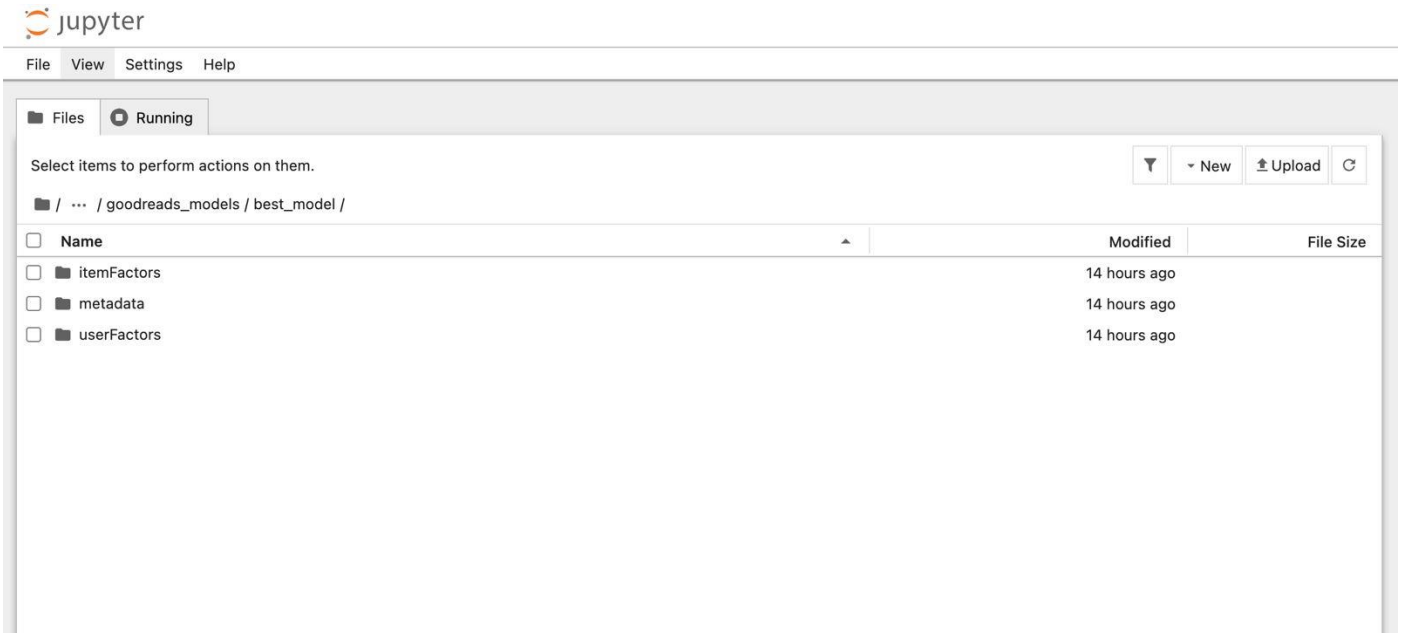



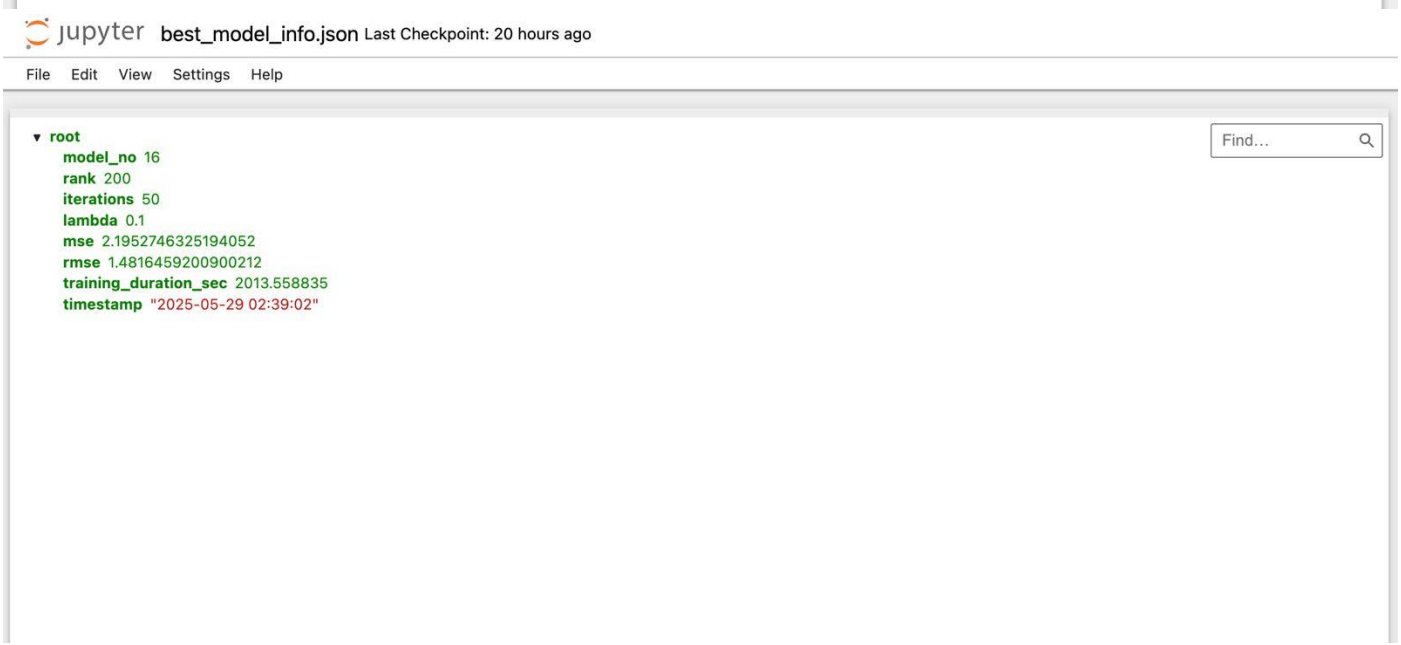

## ***Δ File Directory of Models and Best Model***

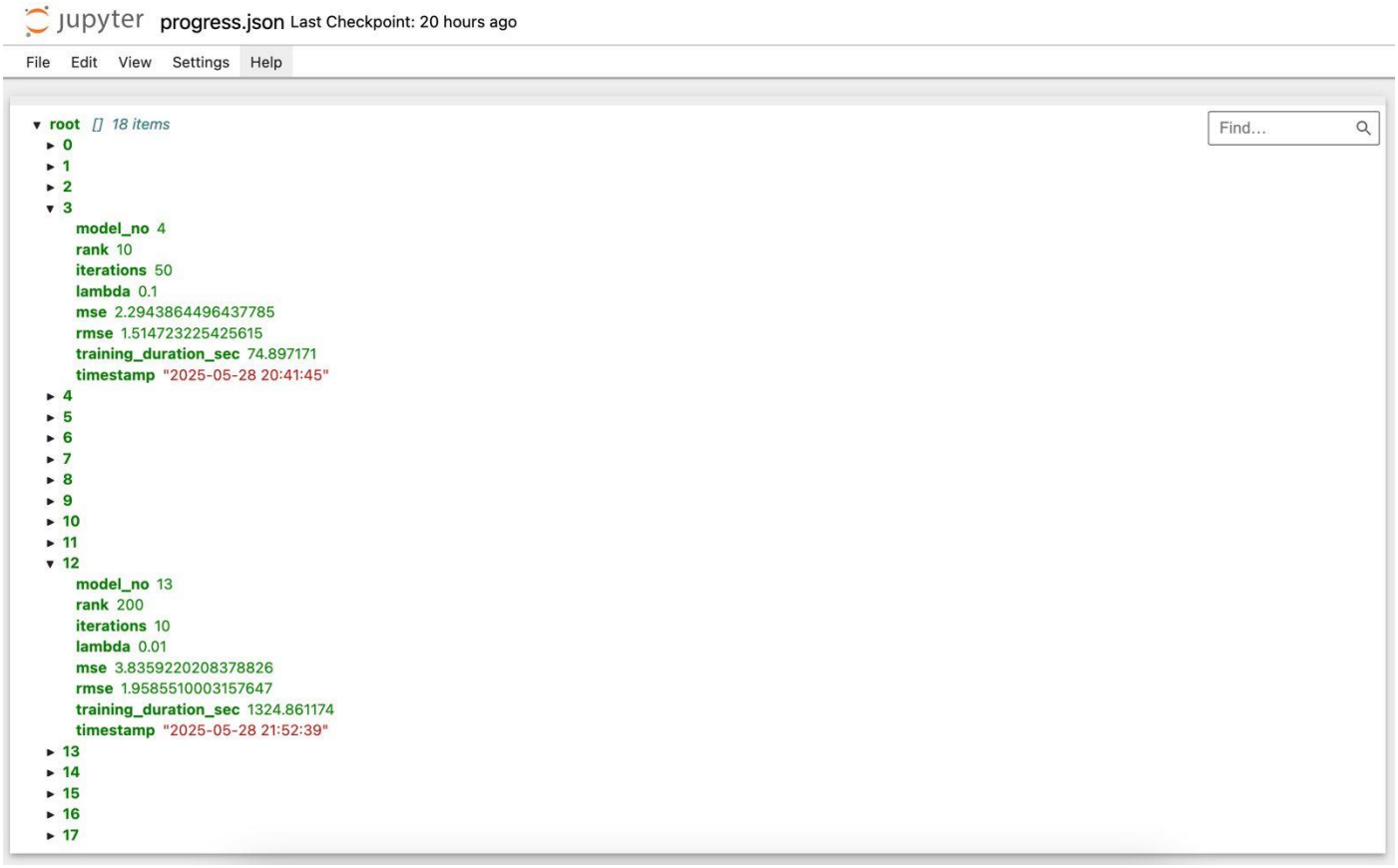

jupyter
File View Settings Help
Files Running
Select items to perform actions on them.
New Upload
/ kerem /
Name Modified File Size
checkpoint_dir_als_project_v2 2 minutes ago
project 16 hours ago
211805018_211805021_211805014_211805015.ipynb 10 seconds ago 405.3 KB
goodreads_reviews_dedup.json 10 days ago 15.6 GB


jupyter
File View Settings Help
Files Running
Select items to perform actions on them.
New Upload
/ kerem / project /
Name Modified File Size
goodreads_models 7 hours ago


jupyter
File View Settings Help
Files Running
Select items to perform actions on them.
New Upload
/ ··· / project / goodreads_models /
Name Modified File Size
best_model 14 hours ago
best_model_info.json 14 hours ago 225 B
errors.log 3 minutes ago 50.3 KB
progress.json 8 hours ago 4.7 KB


jupyter
File View Settings Help
Files Running
Select items to perform actions on them.
New Upload
/ ··· / goodreads_models / best_model /
Name Modified File Size
itemFactors 14 hours ago
metadata 14 hours ago
userFactors 14 hours ago


jupyter best_model_info.json Last Checkpoint: 20 hours ago
File Edit View Settings Help
Find...
root
model_no 16
rank 200
iterations 50
lambda 0.1
mse 2.1952746325194052
rmse 1.4816459200900212
training_duration_sec 2013.558835
timestamp "2025-05-29 02:39:02"




**WORK SHARING POLICY:**

| Team Member | Duty | Percent |
|---|---|---|
| 211805021 - Rıza KARAKAYA | Project Setup & Data Foundation | 25% |
| 211805014 – Mustafa Cihan AYINDI | Data Preprocessing & Feature Engineering for Modeling | 25% |
| 211805018 - Köksal Kerem TANIL | Model Training, Optimization & Core Evaluation | 25% |
| 211805015 – Yusuf TURAN | Advanced Analysis, Visualization & Documentation | 25% |

**REFERENCES:**


- *Dataset:* https://cseweb.ucsd.edu/~jmcauley/datasets/goodreads.html
- *Apache Spark Official Documentation:* https://spark.apache.org/docs/latest/
- *PySpark API Documentation:* https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html
- *ALS Algorithms:* https://medium.com/aimonks/alternating-least-squares-a-cornerstone-in-modern-data-analysis-and-recommendation-systems-ceb7310313f9
- *Recommender System using ALS in Pyspark:* https://medium.com/@brunoborges_38708/recommender-system-using-als-in-pyspark-10329e1d1ee1
- *Cosine Similarity:* https://www.geeksforgeeks.org/cosine-similarity/
- *Cosine Similarity & Python App:* https://medium.com/machine-learning-t%C3%BCrkiye/cosine-similarity-python-uygulamas%C4%B1-926ff395c47e
- *Recommender Systems:* https://medium.com/data-science/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748

- *Recommender System using Pyspark:* https://www.geeksforgeeks.org/recommender-system-using-pyspark-python/
- *Matplotlib Documentation:* https://matplotlib.org/stable/contents.html
- *Seaborn Documentation:* https://seaborn.pydata.org/api.html
- *Pandas Documentation*: https://pandas.pydata.org/pandas-docs/stable/