### **SQL CODE done in SQLite**

-####################################

--################################################# BOOKS ################################################

--###########################################################################

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

--Retrieving all records from the BOOKS Table:

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

SELECT \*

FROM BOOKS;

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

--Uppercase and Trimming (removing beginning and ending spaces) / DISTINCT to get rid of duplicates across the 5 columns

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

DROP TABLE IF EXISTS BOOK\_CLEANING1;

CREATE TABLE BOOK\_CLEANING1 AS

SELECT DISTINCT TRIM(ISBN) as ISBN, TRIM(UPPER(Book\_Title)) as TITLE, TRIM(UPPER(Book\_Author)) as AUTHOR, Year\_of\_Pub as YEAR, TRIM(UPPER(Publisher)) as PUBLISHER

FROM BOOKS;

SELECT \*

FROM BOOK\_CLEANING1;

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

--AUTHOR-- Handling initials

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

DROP TABLE IF EXISTS BOOK\_CLEANING2;

CREATE TABLE BOOK\_CLEANING2 AS

SELECT ISBN, TITLE, TRIM(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

CASE

WHEN AUTHOR LIKE 'A %' THEN 'A. ' || SUBSTR(AUTHOR, 3) --handles beginning initials (first character)

WHEN AUTHOR LIKE 'B %' THEN 'B. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'C %' THEN 'C. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'D %' THEN 'D. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'E %' THEN 'E. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'F %' THEN 'F. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'G %' THEN 'G. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'H %' THEN 'H. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'I %' THEN 'I. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'J %' THEN 'J. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'K %' THEN 'K. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'L %' THEN 'L. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'M %' THEN 'M. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'N %' THEN 'N. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'O %' THEN 'O. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'P %' THEN 'P. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'Q %' THEN 'Q. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'R %' THEN 'R. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'S %' THEN 'S. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'T %' THEN 'T. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'U %' THEN 'U. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'V %' THEN 'V. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'W %' THEN 'W. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'X %' THEN 'X. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'Y %' THEN 'Y. ' || SUBSTR(AUTHOR, 3)

WHEN AUTHOR LIKE 'Z %' THEN 'Z. ' || SUBSTR(AUTHOR, 3)

ELSE AUTHOR

END,

' A ', ' A. '), ' B ', ' B. '), ' C ', ' C. '), ' D ', ' D. '), ' E ', ' E. '), -- handles initials in the middle

' F ', ' F. '), ' G ', ' G. '), ' H ', ' H. '), ' I ', ' I. '), ' J ', ' J. '),

' K ', ' K. '), ' L ', ' L. '), ' M ', ' M. '), ' N ', ' N. '), ' O ', ' O. '),

' P ', ' P. '), ' Q ', ' Q. '), ' R ', ' R. '), ' S ', ' S. '), ' T ', ' T. '),

' U ', ' U. '), ' V ', ' V. '), ' W ', ' W. '), ' X ', ' X. '), ' Y ', ' Y. '),

' Z ', ' Z. '), '.', '. '), '. ', '. ')

) AS AUTHOR,

YEAR, PUBLISHER

FROM BOOK\_CLEANING1;

SELECT AUTHOR

FROM BOOK\_CLEANING2

ORDER BY AUTHOR;

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

--TITLE --Remove spaces before colon, beginning apostrophe, and periods in TITLE

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

DROP TABLE IF EXISTS BOOK\_CLEANING3;

CREATE TABLE BOOK\_CLEANING3 AS

SELECT ISBN,

CASE

WHEN TITLE LIKE ''' %' THEN '''' || LTRIM(SUBSTR(REPLACE(REPLACE(REPLACE(REPLACE(TITLE, ' :', ':'), ' .', '.'), ' ''', ''''), ' \\', '\\'), 3))

ELSE REPLACE(REPLACE(REPLACE(REPLACE(TITLE, ' :', ':'), ' .', '.'), ' ''', ''''), ' \\', '\\')

END AS TITLE,

AUTHOR, YEAR, PUBLISHER

FROM BOOK\_CLEANING2;

SELECT \*

FROM BOOK\_CLEANING3;

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

--ISBN-- Remove spaces and view suspicious ISBNs

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

SELECT REPLACE(ISBN, ' ', '') AS ISBN

FROM BOOK\_CLEANING3

WHERE LENGTH(REPLACE(ISBN, ' ', '')) NOT IN (10, 13);

--noticed ISBN with 10 digits was appearing, investigated and fixed

SELECT ISBN, LENGTH(ISBN) AS original\_length, LENGTH(TRIM(ISBN)) AS trimmed\_length, LENGTH(REPLACE(TRIM(ISBN), ' ', '')) AS cleaned\_length

FROM BOOK\_CLEANING3

WHERE ISBN LIKE '%0486404242%';

--make table

DROP TABLE IF EXISTS BOOK\_CLEANING4;

CREATE TABLE BOOK\_CLEANING4 AS

SELECT REPLACE(REPLACE(TRIM(ISBN), ' ', ''), CHAR(9), '') AS ISBN, TITLE, AUTHOR, YEAR, PUBLISHER

FROM BOOK\_CLEANING3

WHERE LENGTH(REPLACE(REPLACE(TRIM(ISBN), ' ', ''), CHAR(9), '')) IN (10, 13);

SELECT \*

FROM BOOK\_CLEANING4;

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

--Special Characters-- identify patterns and replace with reasonable substitutes

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

--Identify patterns --

SELECT \*

FROM BOOK\_CLEANING4

WHERE (INSTR(TITLE, 'Ã') > 0 OR

INSTR(AUTHOR, 'Ã') > 0 OR

INSTR(PUBLISHER, 'Ã') > 0 OR

INSTR(TITLE, 'Â') > 0 OR

INSTR(AUTHOR, 'Â') > 0 OR

INSTR(PUBLISHER, 'Â') > 0);

SELECT TITLE, AUTHOR

FROM BOOK\_CLEANING4

WHERE (

INSTR(TITLE, 'Ã?Â') > 0 OR

INSTR(TITLE, 'Ã') > 0 OR

INSTR(TITLE, 'Â') > 0 )

AND TITLE NOT LIKE '%Ã?Â®%'

AND TITLE NOT LIKE '%Ã?Â¢%'

AND TITLE NOT LIKE '%Ã?Â¤%'

AND TITLE NOT LIKE '%Ã?Â¡%'

AND TITLE NOT LIKE '%Ã?Â©%'

AND TITLE NOT LIKE '%Ã?Â«%'

AND TITLE NOT LIKE '%Ã?Â»%'

AND TITLE NOT LIKE '%Ã?Â¼%'

AND TITLE NOT LIKE '%Ã?Â±%'

AND TITLE NOT LIKE '%Ã?Â¹%'

AND TITLE NOT LIKE '%Ã?Â¶%'

AND TITLE NOT LIKE '%Ã?Â¨%'

AND TITLE NOT LIKE '%Ã?Â´%'

AND TITLE NOT LIKE '%Ã?Â§%'

AND TITLE NOT LIKE '%Ã?Âª%'

AND TITLE NOT LIKE '%Ã?Â³%'

AND TITLE NOT LIKE '%Ã?Âº%'

AND TITLE NOT LIKE '%Ã?Â¿%'

AND TITLE NOT LIKE '%Ã?Â¯%'

AND TITLE NOT LIKE '%Ã?Â?%';

SELECT TITLE, AUTHOR

FROM BOOK\_CLEANING4

WHERE INSTR(TITLE, 'Ã') > 0

AND TITLE NOT LIKE '%Ã?Â%'

AND TITLE NOT LIKE '%Ã©%'

AND TITLE NOT LIKE '%Ã§%'

AND TITLE NOT LIKE '%Ã£%'

AND TITLE NOT LIKE '%Ã¤%'

AND TITLE NOT LIKE '%Ã¡%'

AND TITLE NOT LIKE '%Ã±%'

AND TITLE NOT LIKE '%Ã³%'

AND TITLE NOT LIKE '%Ã¨%'

AND TITLE NOT LIKE '%Ã¢%'

AND TITLE NOT LIKE '%Ãª%'

AND TITLE NOT LIKE '%Ã¯%'

AND TITLE NOT LIKE '%Ã«%'

AND TITLE NOT LIKE '%Ã¬%'

AND TITLE NOT LIKE '%Ã®%'

AND TITLE NOT LIKE '%Ãº%'

AND TITLE NOT LIKE '%Ã¶%'

AND TITLE NOT LIKE '%Ã¼%'

AND TITLE NOT LIKE '%Ã¹%'

AND TITLE NOT LIKE '%Ã¥%'

AND TITLE NOT LIKE '%Ã²%'

AND TITLE NOT LIKE '%Ã´%'

AND TITLE NOT LIKE '%Ãµ%'

AND TITLE NOT LIKE '%Ã½%'

AND TITLE NOT LIKE '%Ã»%'

ORDER BY AUTHOR;

SELECT TITLE, AUTHOR

FROM BOOK\_CLEANING4

WHERE INSTR(TITLE, 'Â') > 0

AND TITLE NOT LIKE '%Ã?Â®%'

AND TITLE NOT LIKE '%Ã?Â¢%'

AND TITLE NOT LIKE '%Ã?Â¤%'

AND TITLE NOT LIKE '%Ã?Â¡%'

AND TITLE NOT LIKE '%Ã?Â©%'

AND TITLE NOT LIKE '%Ã?Â«%'

AND TITLE NOT LIKE '%Ã?Â»%'

AND TITLE NOT LIKE '%Ã?Â¼%'

AND TITLE NOT LIKE '%Ã?Â±%'

AND TITLE NOT LIKE '%Ã?Â¹%'

AND TITLE NOT LIKE '%Ã?Â¶%'

AND TITLE NOT LIKE '%Ã?Â¨%'

AND TITLE NOT LIKE '%Ã?Â´%'

AND TITLE NOT LIKE '%Ã?Â§%'

AND TITLE NOT LIKE '%Ã?Âª%'

AND TITLE NOT LIKE '%Ã?Â³%'

AND TITLE NOT LIKE '%Ã?Âº%'

AND TITLE NOT LIKE '%Ã?Â¿%'

AND TITLE NOT LIKE '%Ã?Â¯%'

AND TITLE NOT LIKE '%Ã?Â?%'

AND TITLE NOT LIKE '%Ã?Â%'

AND TITLE NOT LIKE '%Ã?Â%'

AND TITLE NOT LIKE '%Ã©%'

AND TITLE NOT LIKE '%Ã§%'

AND TITLE NOT LIKE '%Ã£%'

AND TITLE NOT LIKE '%Ã¤%'

AND TITLE NOT LIKE '%Ã¡%'

AND TITLE NOT LIKE '%Ã±%'

AND TITLE NOT LIKE '%Ã³%'

AND TITLE NOT LIKE '%Ã¨%'

AND TITLE NOT LIKE '%Ã¢%'

AND TITLE NOT LIKE '%Ãª%'

AND TITLE NOT LIKE '%Ã¯%'

AND TITLE NOT LIKE '%Ã«%'

AND TITLE NOT LIKE '%Ã¬%'

AND TITLE NOT LIKE '%Ã®%'

AND TITLE NOT LIKE '%Ãº%'

AND TITLE NOT LIKE '%Ã¶%'

AND TITLE NOT LIKE '%Ã¼%'

AND TITLE NOT LIKE '%Ã¹%'

AND TITLE NOT LIKE '%Ã¥%'

AND TITLE NOT LIKE '%Ã²%'

AND TITLE NOT LIKE '%Ã´%'

AND TITLE NOT LIKE '%Ãµ%'

AND TITLE NOT LIKE '%Ã½%'

AND TITLE NOT LIKE '%Ã»%'

ORDER BY AUTHOR;

--MAKE TABLES--

DROP TABLE IF EXISTS BOOK\_CLEANING5;

CREATE TABLE BOOK\_CLEANING5 AS

SELECT

ISBN,

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(

TITLE,

'Ã?Â¢', 'A'), 'Ã¢', 'A'), 'Ã£', 'A'), 'Ã¥', 'A'), 'Ã?Â¡', 'A'), 'Ã¡', 'A'), 'Ã?Â¤', 'A'), 'Ã¤', 'A'),

'Ã?Â§', 'C'), 'Ã§', 'C'), 'Ã?Â¨', 'E'), 'Ã¨', 'E'), 'Ã?Â©', 'E'), 'Ã©', 'E'), 'Ã?Âª', 'E'), 'Ãª', 'E'),

'Ã?Â«', 'E'), 'Ã«', 'E'), 'Ã?Â¯', 'I'), 'Ã¯', 'I'), 'Ã¬', 'I'), 'Ã?Â®', 'I'), 'Ã®', 'I'),

'Ã?Â±', 'N'), 'Ã±', 'N'), 'Ã?Â³', 'O'), 'Ã³', 'O'), 'Ã?Â¶', 'O'), 'Ã¶', 'O'), 'Ã²', 'O'),

'Ã?Â´', 'O'), 'Ã´', 'O'), 'Ãµ', 'O'), 'Ã?Âº', 'U'), 'Ãº', 'U'), 'Ã?Â¼', 'U'), 'Ã¼', 'U'),

'Ã?Â¹', 'U'), 'Ã¹', 'U'), 'Ã?Â»', 'U'), 'Ã»', 'U'), 'Ã½', 'Y'), 'Â°', ''''

) AS TITLE,

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(

AUTHOR,

'Ã?Â¢', 'A'), 'Ã¢', 'A'), 'Ã£', 'A'), 'Ã¥', 'A'), 'Ã?Â¡', 'A'), 'Ã¡', 'A'), 'Ã?Â¤', 'A'), 'Ã¤', 'A'),

'Ã?Â§', 'C'), 'Ã§', 'C'), 'Ã?Â¨', 'E'), 'Ã¨', 'E'), 'Ã?Â©', 'E'), 'Ã©', 'E'), 'Ã?Âª', 'E'), 'Ãª', 'E'),

'Ã?Â«', 'E'), 'Ã«', 'E'), 'Ã?Â¯', 'I'), 'Ã¯', 'I'), 'Ã¬', 'I'), 'Ã?Â®', 'I'), 'Ã®', 'I'),

'Ã?Â±', 'N'), 'Ã±', 'N'), 'Ã?Â³', 'O'), 'Ã³', 'O'), 'Ã?Â¶', 'O'), 'Ã¶', 'O'), 'Ã²', 'O'),

'Ã?Â´', 'O'), 'Ã´', 'O'), 'Ãµ', 'O'), 'Ã?Âº', 'U'), 'Ãº', 'U'), 'Ã?Â¼', 'U'), 'Ã¼', 'U'),

'Ã?Â¹', 'U'), 'Ã¹', 'U'), 'Ã?Â»', 'U'), 'Ã»', 'U'), 'Ã½', 'Y'), 'Â°', ''''

) AS AUTHOR,

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(

REPLACE(REPLACE(REPLACE(

PUBLISHER,

'Ã?Â¢', 'A'), 'Ã¢', 'A'), 'Ã£', 'A'), 'Ã¥', 'A'), 'Ã?Â¡', 'A'), 'Ã¡', 'A'), 'Ã?Â¤', 'A'), 'Ã¤', 'A'),

'Ã?Â§', 'C'), 'Ã§', 'C'), 'Ã?Â¨', 'E'), 'Ã¨', 'E'), 'Ã?Â©', 'E'), 'Ã©', 'E'), 'Ã?Âª', 'E'), 'Ãª', 'E'),

'Ã?Â«', 'E'), 'Ã«', 'E'), 'Ã?Â¯', 'I'), 'Ã¯', 'I'), 'Ã¬', 'I'), 'Ã?Â®', 'I'), 'Ã®', 'I'),

'Ã?Â±', 'N'), 'Ã±', 'N'), 'Ã?Â³', 'O'), 'Ã³', 'O'), 'Ã?Â¶', 'O'), 'Ã¶', 'O'), 'Ã²', 'O'),

'Ã?Â´', 'O'), 'Ã´', 'O'), 'Ãµ', 'O'), 'Ã?Âº', 'U'), 'Ãº', 'U'), 'Ã?Â¼', 'U'), 'Ã¼', 'U'),

'Ã?Â¹', 'U'), 'Ã¹', 'U'), 'Ã?Â»', 'U'), 'Ã»', 'U'), 'Ã½', 'Y'), 'Â°', ''''

) AS PUBLISHER,

YEAR

FROM BOOK\_CLEANING4;

SELECT \*

FROM BOOK\_CLEANING5;

--Additional cleaning--

SELECT \*

FROM BOOK\_CLEANING5

WHERE (INSTR(TITLE, '\\') > 0 OR

INSTR(AUTHOR, '\\') > 0 OR

INSTR(PUBLISHER, '\\') > 0 OR

INSTR(TITLE, '\') > 0 OR

INSTR(AUTHOR, '\') > 0 OR

INSTR(PUBLISHER, '\') > 0);

DROP TABLE IF EXISTS BOOK\_CLEANING6;

CREATE TABLE BOOK\_CLEANING6 AS

SELECT

ISBN,

CASE WHEN TITLE LIKE '%\\%\\%' THEN REPLACE(REPLACE(TITLE, '\\', '"'), '\', '"')

WHEN TITLE LIKE '%\%\%' THEN REPLACE(TITLE, '\', '"')

ELSE TITLE

END AS TITLE,

AUTHOR, YEAR, PUBLISHER

FROM BOOK\_CLEANING5;

SELECT \*

FROM BOOK\_CLEANING6;

DROP TABLE IF EXISTS BOOK\_CLEANING7;

CREATE TABLE BOOK\_CLEANING7 AS

SELECT ISBN,

CASE

WHEN TITLE LIKE '\ %' THEN REPLACE(REPLACE(LTRIM(SUBSTR(TITLE, 3)), 'Ã?Â?BER', 'UBER'), 'Ã?BER', 'UBER')

WHEN TITLE LIKE '\%' THEN REPLACE(REPLACE(LTRIM(SUBSTR(TITLE, 2)), 'Ã?Â?BER', 'UBER'), 'Ã?BER', 'UBER')

WHEN TITLE LIKE '" %' THEN REPLACE(REPLACE('"' || LTRIM(SUBSTR(TITLE, 3)), 'Ã?Â?BER', 'UBER'), 'Ã?BER', 'UBER')

WHEN TITLE LIKE 'Â¡%' THEN SUBSTR(TITLE, 2)

WHEN TITLE LIKE 'Â¿%' THEN SUBSTR(TITLE, 2)

ELSE REPLACE(REPLACE(TITLE, 'Ã?Â?BER', 'UBER'), 'Ã?BER', 'UBER')

END AS TITLE,

REPLACE(REPLACE(AUTHOR, 'Ã?Â?BER', 'UBER'), 'Ã?BER', 'UBER') AS AUTHOR,

YEAR, PUBLISHER

FROM BOOK\_CLEANING6;

SELECT \*

FROM BOOK\_CLEANING7

ORDER BY AUTHOR;

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

--YEAR -- fixing 0's In Book Years - Haley

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

SELECT COUNT (\*)

FROM BOOK\_CLEANING7

WHERE YEAR < 1400 OR YEAR > 2025

ORDER BY YEAR DESC;

DROP TABLE IF EXISTS BOOK\_CLEANING8;

CREATE TABLE BOOK\_CLEANING8 AS

SELECT ISBN, TITLE, AUTHOR, PUBLISHER,

CASE

WHEN YEAR < 1400 OR YEAR > 2025 THEN NULL

ELSE YEAR

END AS YEAR

FROM BOOK\_CLEANING7;

SELECT \*

FROM BOOK\_CLEANING8;

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

--AUTHOR & PUBLISHER-- Nulling bad entries

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

DROP TABLE IF EXISTS BOOK\_CLEANING9;

CREATE TABLE BOOK\_CLEANING9 AS

SELECT ISBN, TITLE,

CASE

WHEN AUTHOR IN ('NOT APPLICABLE (NA )', 'NOT AVAILABLE', 'NOT AVAILALBLE', 'N/A', 'NA', 'N', 'NONE') THEN NULL

ELSE AUTHOR

END AS AUTHOR,

CASE

WHEN PUBLISHER IN ('NOT AVAIL', 'NOT APPLICABLE', 'N/A') THEN NULL

ELSE PUBLISHER

END AS PUBLISHER,

YEAR

FROM BOOK\_CLEANING8;

SELECT \*

FROM BOOK\_CLEANING9;

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

--Book Data (Duplicates) - do after all cleaning for table is complete

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

SELECT TITLE, COUNT(\*) AS count

FROM BOOK\_CLEANING9

GROUP BY TITLE

HAVING COUNT(\*) > 1

ORDER BY count DESC;

--Assign Book IDs to each title/author combo

DROP TABLE IF EXISTS BOOK\_ID;

CREATE TABLE BOOK\_ID AS

SELECT TITLE, AUTHOR, ROW\_NUMBER() OVER (ORDER BY AUTHOR, TITLE) AS BOOK\_ID

FROM (SELECT DISTINCT TITLE, AUTHOR

FROM BOOK\_CLEANING9);

-- Want to use the earliest publication of each duplicate title --

DROP TABLE IF EXISTS EARLIEST;

CREATE TABLE EARLIEST AS

SELECT TITLE, AUTHOR, MIN(YEAR) AS EARLIEST\_YEAR

FROM BOOK\_CLEANING9

GROUP BY TITLE, AUTHOR;

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

--JOINING for final book table

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

DROP TABLE IF EXISTS BOOKS\_FINAL\_ONE;

CREATE TABLE BOOKS\_FINAL\_ONE AS

SELECT b.ISBN, b.TITLE, b.AUTHOR, b.PUBLISHER, b.YEAR, id.BOOK\_ID

FROM BOOK\_CLEANING9 b

LEFT JOIN BOOK\_ID id

ON b.TITLE = id.TITLE

AND ((b.AUTHOR = id.AUTHOR) OR (b.AUTHOR IS NULL AND id.AUTHOR IS NULL));

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

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

--#added later# AUTHOR dropping those with fewer than 10 distinct BOOK\_IDs

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

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

DROP TABLE IF EXISTS BOOK\_CLEANING10;

CREATE TABLE BOOK\_CLEANING10 AS

SELECT \*

FROM BOOKS\_FINAL\_ONE

WHERE AUTHOR IN (

SELECT AUTHOR

FROM BOOKS\_FINAL\_ONE

WHERE AUTHOR IS NOT NULL

GROUP BY AUTHOR

HAVING COUNT(DISTINCT BOOK\_ID) > 10

);

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

--#finished# AUTHOR and PUBLISHER filtering

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

DROP TABLE IF EXISTS BOOKS\_FINAL\_TWO;

CREATE TABLE BOOKS\_FINAL\_TWO AS

SELECT bf1.ISBN, bf1.TITLE, bf1.AUTHOR, bf1.PUBLISHER, bf1.YEAR, bf1.BOOK\_ID, e.EARLIEST\_YEAR

FROM BOOK\_CLEANING10 bf1

LEFT JOIN EARLIEST e

ON bf1.TITLE = e.TITLE

AND ((bf1.AUTHOR = e.AUTHOR) OR (bf1.AUTHOR IS NULL AND e.AUTHOR IS NULL));

DROP TABLE IF EXISTS BOOKS\_FINAL;

CREATE TABLE BOOKS\_FINAL AS

SELECT ISBN, BOOK\_ID, TITLE, AUTHOR, PUBLISHER, YEAR,

CASE

WHEN YEAR = EARLIEST\_YEAR THEN 1

ELSE 0

END AS IS\_EARLIEST

FROM BOOKS\_FINAL\_TWO

ORDER BY BOOK\_ID;

SELECT \*

FROM BOOKS\_FINAL;

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

-- Drop other BOOK tables

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

DROP TABLE BOOKS;

DROP TABLE BOOK\_CLEANING1;

DROP TABLE BOOK\_CLEANING2;

DROP TABLE BOOK\_CLEANING3;

DROP TABLE BOOK\_CLEANING4;

DROP TABLE BOOK\_CLEANING5;

DROP TABLE BOOK\_CLEANING6;

DROP TABLE BOOK\_CLEANING7;

DROP TABLE BOOK\_CLEANING8;

DROP TABLE BOOK\_CLEANING9;

DROP TABLE BOOK\_CLEANING10;

DROP TABLE BOOK\_ID;

DROP TABLE EARLIEST;

DROP TABLE BOOKS\_FINAL\_ONE;

DROP TABLE BOOKS\_FINAL\_TWO;

--################################################# RATINGS ###########

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

--Clean RATINGS column

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

-- how many ratings equal 0 (over half)

SELECT COUNT(\*)

FROM RATINGS

WHERE RATINGS == 0

GROUP BY RATINGS;

DROP TABLE IF EXISTS RATINGS\_CLEANED1;

CREATE TABLE RATINGS\_CLEANED1 AS

SELECT

USERID,

ISBN,

RATINGS

FROM RATINGS

WHERE RATINGS > 0;

SELECT \*

FROM RATINGS\_CLEANED1;

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

-- Clean ISBNs

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

DROP TABLE IF EXISTS RATINGS\_FINAL;

CREATE TABLE RATINGS\_FINAL AS

SELECT

UserID as USERID,

REPLACE(REPLACE(REPLACE(REPLACE(TRIM(ISBN), '.', ''), '-', ''), '/', ''), CHAR(9), '') AS ISBN,

Ratings AS RATINGS

FROM RATINGS\_CLEANED1

WHERE LENGTH(REPLACE(REPLACE(REPLACE(REPLACE(TRIM(ISBN), '.', ''), '-', ''), '/', ''), CHAR(9), '')) IN (10, 13);

-- Find how many ISBNs were not added to the new table

SELECT COUNT(\*)

FROM RATINGS\_CLEANED1

WHERE LENGTH(REPLACE(REPLACE(REPLACE(REPLACE(TRIM(ISBN), '.', ''), '-', ''), '/', ''), CHAR(9), '')) NOT IN (10, 13);

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

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

--#added later# Users with 5+ ratings

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

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

DROP TABLE IF EXISTS RATINGS\_CLEANED2;

CREATE TABLE RATINGS\_CLEANED2 AS

SELECT USERID

FROM RATINGS\_CLEANED1

GROUP BY USERID

HAVING COUNT(\*) >= 5;

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

--UserID how many left in the table

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

SELECT \*

FROM RATINGS\_FINAL;

SELECT USERID, COUNT(\*) AS NUM\_RATINGS

FROM RATINGS\_FINAL

GROUP BY USERID

HAVING COUNT(\*) >= 5

ORDER BY NUM\_RATINGS DESC;

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

--Drop other RATINGS tables

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

DROP TABLE RATINGS;

DROP TABLE RATINGS\_CLEANED1;

DROP TABLE RATINGS\_CLEANED2;

--################################################# USERS #################################################

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

--Splitting User Locations

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

--- Adding new columns

ALTER TABLE Users ADD COLUMN City;

ALTER TABLE Users ADD COLUMN State\_Region;

ALTER TABLE Users ADD COLUMN Country;

-- CITY UPDATE

UPDATE Users

SET City = TRIM(substr(Location, 1, instr(Location, '|') - 1))

WHERE Location LIKE '%|%';

-- STATE/REGION UPDATE

UPDATE Users

SET State\_Region = TRIM( substr( Location, instr(Location, '|') + 1, instr(substr(Location, instr(Location, '|') + 1), '|') - 1 ) )

WHERE Location LIKE '%|%|%';

-- COUNTRY UPDATE (everything after the last '|')

UPDATE Users

SET Country = TRIM( substr( Location, instr(Location, '|') + instr(substr(Location, instr(Location, '|') + 1), '|') + 1 ) )

WHERE Location LIKE '%|%|%';

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

-- Removing header row - Haley

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

DELETE FROM Users

WHERE UserID = 'User-ID';

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

--- Removing extra | in countries - Haley

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

UPDATE Users

SET Country = TRIM(REPLACE(Country, '|', ''))

WHERE Country LIKE '|%';

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

-- Changing N/A's to Nulls - Haley

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

UPDATE Users

SET City = NULL

WHERE LOWER(City) IN ('n/a', 'n.a.', 'n.a', 'n\a', 'n/a', 'unknown');

UPDATE Users

SET State\_Region = NULL

WHERE LOWER(State\_Region) IN ('n/a', 'n.a.', 'n.a', 'n\a', 'n/a', 'unknown');

UPDATE Users

SET Country = NULL

WHERE LOWER(Country) IN ('n/a', 'n.a.', 'n.a', 'n\a', 'n/a', 'unknown');

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

-- UPPER EVERYTHING - Haley

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

UPDATE Users

SET City = UPPER(City)

WHERE City IS NOT NULL;

UPDATE Users

SET State\_Region = UPPER(State\_Region)

WHERE State\_Region IS NOT NULL;

UPDATE Users

SET Country = UPPER(Country)

WHERE Country IS NOT NULL;

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

-- Age Updates and Duplicate Users

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

--Change 0 Values in Age to NULL

UPDATE Users SET Age = NULL WHERE Age = 0;

--Remove duplicate users and keep first recorded UserID

DELETE FROM Users WHERE rowid NOT IN ( SELECT MIN(rowid) FROM Users GROUP BY UserID HAVING COUNT(\*) > 1 );

--Change age values set to "NULL" to NULL

UPDATE Users SET Age = NULL

WHERE Age = "NULL";

--Removes users that have an age above 110

DELETE FROM Users WHERE Age > 110;

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

-- More Location Fixes -

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

DROP TABLE IF EXISTS USERS\_CLEANING1;

CREATE TABLE USERS\_CLEANING1 AS

SELECT UserID, Location, Age, Country

FROM USERS;

SELECT \*

FROM USERS\_CLEANING1;

--Clean extra pipes

DROP TABLE IF EXISTS USERS\_CLEANING2;

CREATE TABLE USERS\_CLEANING2 AS

SELECT

UserID,

Location,

Age,

CASE

WHEN Country LIKE '%|%' THEN SUBSTR(Country, INSTR(Country, '|') + 1)

ELSE Country

END AS Country

FROM USERS\_CLEANING1;

SELECT \*

FROM USERS\_CLEANING2;

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

--More Cleaning of Country column

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

UPDATE USERS\_CLEANING2

SET Country = NULL

WHERE TRIM(Country) = '';

UPDATE USERS\_CLEANING2

SET Country = TRIM(Country);

--Repeat below until no more remain to left of a |

UPDATE USERS\_CLEANING2

SET Country = TRIM( substr(Country, instr(Country, '|') + 1) )

WHERE Country LIKE '%|%';

UPDATE USERS\_CLEANING2

SET Country = TRIM( substr(Country, instr(Country, '|') + 1) )

WHERE Country LIKE '%|%';

UPDATE USERS\_CLEANING2

SET Country = TRIM( substr(Country, instr(Country, '|') + 1) )

WHERE Country LIKE '%|%';

UPDATE USERS\_CLEANING2

SET Country = TRIM( substr(Country, instr(Country, '|') + 1) )

WHERE Country LIKE '%|%';

UPDATE USERS\_CLEANING2

SET Country = TRIM( substr(Country, instr(Country, '|') + 1) )

WHERE Country LIKE '%|%';

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

--Combine a few common misspellings

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

UPDATE USERS\_CLEANING2

SET Country = 'USA'

WHERE UPPER(Country) LIKE 'UNITED S%' OR UPPER(Country) IN ('UUSA', 'UNITE STATES');

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

--#added later# COUNTRY keeping only USA

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

DROP TABLE IF EXISTS USERS\_FINAL;

CREATE TABLE USERS\_FINAL AS

SELECT UserID as USERID, Age as AGE, Country as COUNTRY

FROM USERS\_CLEANING2

WHERE Country = 'USA';

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

--#finished# COUNTRY keeping only USA

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

SELECT \*

FROM USERS\_FINAL;

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

--Drop other USERS tables

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

DROP TABLE Users;

DROP TABLE USERS\_CLEANING1;

DROP TABLE USERS\_CLEANING2;

--################################################# COMBINE ################################################

DROP TABLE IF EXISTS REC\_BOOKS;

CREATE TABLE REC\_BOOKS AS

SELECT \*

FROM BOOKS\_FINAL

WHERE ISBN IN (SELECT ISBN FROM RATINGS\_FINAL)

OR IS\_EARLIEST = 1;

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

--Make table for content-based filtering

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

DROP TABLE IF EXISTS PRE\_CONTENT\_JOIN;

CREATE TABLE PRE\_CONTENT\_JOIN AS

SELECT

rb.BOOK\_ID,

rb.ISBN,

rb.TITLE,

rb.AUTHOR,

rb.PUBLISHER,

rb.YEAR,

rb.IS\_EARLIEST,

r.USERID,

r.RATINGS

FROM REC\_BOOKS rb

LEFT JOIN RATINGS\_FINAL r

ON rb.ISBN = r.ISBN;

DROP TABLE IF EXISTS TOP\_BOOK\_IDS;

CREATE TABLE TOP\_BOOK\_IDS AS

SELECT BOOK\_ID, COUNT(\*) AS rating\_count

FROM PRE\_CONTENT\_JOIN

WHERE RATINGS IS NOT NULL

GROUP BY BOOK\_ID

ORDER BY rating\_count DESC

LIMIT 15000;

DROP TABLE IF EXISTS CONTENT\_JOIN;

CREATE TABLE CONTENT\_JOIN AS

SELECT \*

FROM PRE\_CONTENT\_JOIN

WHERE BOOK\_ID IN (SELECT BOOK\_ID FROM TOP\_BOOK\_IDS);

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

--Drop last tables

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

DROP TABLE REC\_BOOKS;

DROP TABLE PRE\_CONTENT\_JOIN;

DROP TABLE TOP\_BOOK\_IDS;

### **Python Code done in Pycharm**

import numpy as np

import pandas as pd

import sqlite3

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.spatial import distance

#Connect to Books database

conn = sqlite3.connect(r"C:\Users\spoon\OneDrive\Documents\Classes\780\_Capstone\780\_Data\Books.db")

cur = conn.cursor()

#Load the Content\_Join table into a pandas DataFrame

content\_df = pd.read\_sql\_query("SELECT \* FROM CONTENT\_JOIN;", conn)

#Check head of data to make sure it read in correctly

print(content\_df.head())

print(content\_df.shape)

#Drop rows where author is NULL

content\_df = content\_df.dropna(subset=['AUTHOR'])

print(content\_df.shape)

##########################

#One-Hot Encoding Authors#

##########################

#One-hot encode the AUTHOR column

author\_ohe = pd.get\_dummies(content\_df['AUTHOR'])

#Group BOOK\_IDs and OHE author columns for similarity

content\_for\_similarity = pd.concat([content\_df[['BOOK\_ID']], author\_ohe], axis=1)

#Group by BOOK\_ID and sum the OHE columns

grouped\_bookid = content\_for\_similarity.groupby('BOOK\_ID', as\_index=False)[author\_ohe.columns].sum()

print(grouped\_bookid.shape)

#########################

#Calc. Cosine Similarity#

#########################

#Drop BOOK\_ID to leave only OHE AUTHOR columns

book\_features = grouped\_bookid.drop(columns=['BOOK\_ID'])

print(book\_features.shape)

#Compute cosine similarity matrix

cos\_sim = cosine\_similarity(book\_features)

print(cos\_sim.shape)

print(cos\_sim)

#Change diagonal values to 0

np.fill\_diagonal(cos\_sim, 0)

#Create a DataFrame with the Cosine similarities

book\_df = pd.DataFrame(cos\_sim, index=grouped\_bookid['BOOK\_ID'], columns=grouped\_bookid['BOOK\_ID'])

print(book\_df.shape)

########################

#Create Recommendations#

########################

#Step 1: Create BOOK\_ID to TITLE mapping

book\_id\_to\_title = content\_df[['BOOK\_ID', 'TITLE']].drop\_duplicates().set\_index('BOOK\_ID')['TITLE'].to\_dict()

# Step 2: Choose Sample BOOK\_ID

sample\_book\_id = grouped\_bookid['BOOK\_ID'].iloc[150] # or manually set a book

# Step 2.5: Check what you picked

print("Sample BOOK\_ID:", sample\_book\_id)

print("Sample Book Title:", book\_id\_to\_title.get(sample\_book\_id, "Unknown Book"))

#Step 3: Get Similarity Scores for Sample Book

similarity\_scores = book\_df[sample\_book\_id]

# Step 4: Drop the same book itself

similarity\_scores = similarity\_scores.drop(sample\_book\_id)

# Step 5: Set a minimum similarity threshold to avoid forcing results

similarity\_scores = similarity\_scores[similarity\_scores > 0.3]

# Step 6: Get Top N Recommended BOOK\_IDs

top\_n = 5

recommended\_book\_ids = similarity\_scores.sort\_values(ascending=False).head(top\_n).index

# Step 7: Pull the earliest versions

# Keep only earliest editions for recommended books

recommended\_books\_earliest = content\_df[

(content\_df['BOOK\_ID'].isin(recommended\_book\_ids)) &

(content\_df['IS\_EARLIEST'] == 1)

][['BOOK\_ID', 'TITLE', 'AUTHOR', 'YEAR']]

# Drop duplicates if BOOK\_ID appears more than once

recommended\_books\_earliest = recommended\_books\_earliest.drop\_duplicates(subset=['BOOK\_ID'])

# Create BOOK\_ID ➔ (TITLE, YEAR) Lookup

book\_lookup = recommended\_books\_earliest.set\_index('BOOK\_ID')[['TITLE', 'AUTHOR', 'YEAR']].to\_dict(orient='index')

# Make final recommendations

final\_recommendations = []

for book\_id in recommended\_book\_ids:

if book\_id in book\_lookup:

entry = book\_lookup[book\_id]

final\_recommendations.append((entry['TITLE'], entry['AUTHOR'], entry['YEAR']))

# Step 8: Print Recommendations

sample\_row = content\_df[content\_df['BOOK\_ID'] == sample\_book\_id].drop\_duplicates(subset=['BOOK\_ID']).iloc[0]

sample\_title = sample\_row['TITLE']

sample\_author = sample\_row['AUTHOR']

if not final\_recommendations:

print(f"No strong similar books found for: {sample\_title} by {sample\_author}")

else:

print(f"\nTop Recommended Books similar to: {sample\_title} by {sample\_author}")

for i, (title, author, year) in enumerate(final\_recommendations, 1):

print(f"\n{i}. {title} ({int(year)}) by {author}")

import pandas as pd

import numpy as np

import sqlite3

from sklearn.metrics.pairwise import cosine\_similarity

# Connect to Books database

conn = sqlite3.connect('/Users/cherechi/Desktop/Books\_Final.db')

cur = conn.cursor()

# Load the table into a pandas DataFrame

ratings\_df = pd.read\_sql\_query("SELECT \* FROM CONTENT\_JOIN;", conn)

# Check the data

print(ratings\_df.head())

print(ratings\_df.shape)

#########################################

# Create User-Item Matrix for Filtering #

#########################################

# Create a user-item matrix with users as rows, books as columns, and ratings as values

user\_item\_matrix = ratings\_df.pivot\_table(index='USERID', columns='BOOK\_ID', values='RATINGS')

# Fill missing values with 0

user\_item\_matrix = user\_item\_matrix.fillna(0)

# Transpose the matrix if you want item-based filtering (books as rows)

item\_user\_matrix = user\_item\_matrix.T

##############################

# Compute Cosine Similarities#

##############################

print(ratings\_df.columns)

print(ratings\_df.head())

item\_user\_matrix = user\_item\_matrix.T

print(item\_user\_matrix.shape)

print(item\_user\_matrix.dtypes)

# Compute item-item cosine similarity

item\_similarity\_matrix = cosine\_similarity(item\_user\_matrix)

print("Item-item similarity matrix shape:", item\_similarity\_matrix.shape)

print(item\_similarity\_matrix[:5, :5])

# Step 1: Create book ID to title mapping

book\_id\_to\_title = ratings\_df[['BOOK\_ID', 'TITLE']].drop\_duplicates().set\_index('BOOK\_ID')['TITLE'].to\_dict()

# Step 2: Choose the user ID

user\_id = 55490

# Step 3: Get user's ratings

user\_ratings = user\_item\_matrix.loc[user\_id]

# Step 4: Get books the user rated highly

highly\_rated\_books = user\_ratings[user\_ratings >= 4].index

# Step 5: Initialize score vector

book\_scores = np.zeros(user\_item\_matrix.shape[1])

# Step 6: Score books based on similarity to liked books

for book\_id in highly\_rated\_books:

book\_idx = user\_item\_matrix.columns.get\_loc(book\_id)

similarity\_vector = item\_similarity\_matrix[book\_idx]

book\_scores += similarity\_vector \* user\_ratings[book\_id]

# Step 7: Remove already rated books

for book\_id in user\_ratings[user\_ratings > 0].index:

idx = user\_item\_matrix.columns.get\_loc(book\_id)

book\_scores[idx] = -1 # Exclude from recommendation

# Step 8: Get top N recommended book IDs

top\_indices = book\_scores.argsort()[::-1][:5]

recommended\_book\_ids = user\_item\_matrix.columns[top\_indices]

# Step 9: Convert to titles

recommended\_titles = [book\_id\_to\_title.get(book\_id, f"Book {book\_id}") for book\_id in recommended\_book\_ids]

# Step 10: Print recommendations

print("Top recommended books for user", user\_id)

for i, title in enumerate(recommended\_titles, 1):

print(f"{i}. {title}")

import pandas as pd

import numpy as np

import sqlite3

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.spatial import distance

#Connect to database

conn = sqlite3.connect(r"C:\Users\Noah Moore\Downloads\Books.db")

#Load data

ratings\_df = pd.read\_sql\_query("SELECT \* FROM CONTENT\_JOIN;", conn)

content\_df = ratings\_df.copy()

#Drop rows where author is NULL

content\_df = content\_df.dropna(subset=['AUTHOR'])

#####################################

# Collaborative Filtering Preparation#

#####################################

#Create user-item matrix with all possible BOOK\_IDs

all\_book\_ids = content\_df['BOOK\_ID'].unique()

user\_item\_matrix = ratings\_df.pivot\_table(index='USERID', columns='BOOK\_ID', values='RATINGS').fillna(0)

user\_item\_matrix = user\_item\_matrix.reindex(columns=all\_book\_ids, fill\_value=0)

item\_user\_matrix = user\_item\_matrix.T

#Compute item-item cosine similarity for collaborative filtering

collab\_similarity\_matrix = cosine\_similarity(item\_user\_matrix)

#####################################

# Content-Based Filtering Preparation#

#####################################

#One-hot encode authors and create content similarity matrix

author\_ohe = pd.get\_dummies(content\_df['AUTHOR'])

content\_for\_similarity = pd.concat([content\_df[['BOOK\_ID']], author\_ohe], axis=1)

grouped\_bookid = content\_for\_similarity.groupby('BOOK\_ID', as\_index=False)[author\_ohe.columns].sum()

book\_features = grouped\_bookid.drop(columns=['BOOK\_ID'])

content\_sim\_matrix = cosine\_similarity(book\_features)

np.fill\_diagonal(content\_sim\_matrix, 0)

#Create aligned similarity DataFrames

collab\_sim\_df = pd.DataFrame(collab\_similarity\_matrix,

index=item\_user\_matrix.index,

columns=item\_user\_matrix.index)

content\_sim\_df = pd.DataFrame(content\_sim\_matrix,

index=grouped\_bookid['BOOK\_ID'],

columns=grouped\_bookid['BOOK\_ID'])

#Align the matrices to common book IDs (now redundant but kept for safety)

common\_books = collab\_sim\_df.index.intersection(content\_sim\_df.index)

collab\_sim\_df = collab\_sim\_df.loc[common\_books, common\_books]

content\_sim\_df = content\_sim\_df.loc[common\_books, common\_books]

#Create hybrid similarity matrix

hybrid\_sim\_matrix = 0.5 \* collab\_sim\_df.values + 0.5 \* content\_sim\_df.values

hybrid\_sim\_df = pd.DataFrame(hybrid\_sim\_matrix,

index=common\_books,

columns=common\_books)

########################

# Hybrid Recommendations#

########################

#Create book ID to title mapping

book\_id\_to\_title = ratings\_df[['BOOK\_ID', 'TITLE']].drop\_duplicates().set\_index('BOOK\_ID')['TITLE'].to\_dict()

#Get user input for user ID (55490 is used for our example)

while True:

try:

user\_id = int(input("Enter user ID for recommendations: "))

if user\_id not in user\_item\_matrix.index:

print(f"User ID {user\_id} not found in the database. Please try another ID.")

continue

break

except ValueError:

print("Please enter a valid integer user ID.")

#Get user's ratings

user\_ratings = user\_item\_matrix.loc[user\_id]

#Get books the user rated highly

top\_read\_books = user\_ratings[user\_ratings > 0].sort\_values(ascending=False).head(5).index

if len(top\_read\_books) == 0:

print(f"\nUser {user\_id} hasn't rated any books. Cannot generate recommendations.")

else:

top\_read\_titles = [book\_id\_to\_title.get(book\_id, f"Book ID {book\_id}") for book\_id in top\_read\_books]

#Initialize score vector

book\_scores = np.zeros(len(common\_books))

#Score books based on hybrid similarity to liked books

for book\_id in top\_read\_books:

if book\_id in hybrid\_sim\_df.index:

book\_idx = hybrid\_sim\_df.index.get\_loc(book\_id)

similarity\_vector = hybrid\_sim\_matrix[book\_idx]

book\_scores += similarity\_vector \* user\_ratings[book\_id]

# Remove already rated books

for book\_id in user\_ratings[user\_ratings > 0].index:

if book\_id in hybrid\_sim\_df.index:

idx = hybrid\_sim\_df.index.get\_loc(book\_id)

book\_scores[idx] = -1

#Get top recommended book IDs

top\_indices = book\_scores.argsort()[::-1]

recommended\_book\_ids = hybrid\_sim\_df.index[top\_indices]

#Get recommendation details

recommended\_books\_info = content\_df[

(content\_df['BOOK\_ID'].isin(recommended\_book\_ids)) &

(content\_df['IS\_EARLIEST'] == 1)

][['BOOK\_ID', 'TITLE', 'AUTHOR', 'YEAR']].drop\_duplicates(subset=['BOOK\_ID'])

#Create a dictionary to map book IDs to their info

book\_info\_dict = {row['BOOK\_ID']: row for \_, row in recommended\_books\_info.iterrows()}

#Limit to 2 books per author

final\_recommendations = []

author\_counts = {}

for book\_id in recommended\_book\_ids:

if len(final\_recommendations) >= 5:

break

if book\_id in book\_info\_dict:

book\_info = book\_info\_dict[book\_id]

author = book\_info['AUTHOR']

#Initialize or increment author count

if author not in author\_counts:

author\_counts[author] = 0

if author\_counts[author] < 2:

final\_recommendations.append(book\_info)

author\_counts[author] += 1

#Print recommendations

print("\nBecause you enjoyed these books:")

for i, title in enumerate(top\_read\_titles, 1):

print(f"{i}. {title}")

print("\nWe think you'll enjoy these recommendations:")

if len(final\_recommendations) == 0:

print("No recommendations available based on current user ratings.")

else:

for i, book\_info in enumerate(final\_recommendations[:5], 1): #Ensure we don't exceed 5 even with author limit

print(f"{i}. {book\_info['TITLE']} ({int(book\_info['YEAR'])}) by {book\_info['AUTHOR']}")

import pandas as pd

import numpy as np

import sqlite3

from sklearn.metrics.pairwise import cosine\_similarity

import random

#Connect to database

conn = sqlite3.connect(r"C:\Users\Noah Moore\Downloads\Books.db")

#Load data

ratings\_df = pd.read\_sql\_query("SELECT \* FROM CONTENT\_JOIN;", conn)

content\_df = ratings\_df.copy()

#Drop rows where author is NULL

content\_df = content\_df.dropna(subset=['AUTHOR'])

#####################################

# Collaborative Filtering Preparation

#####################################

#Create user-item matrix with all possible BOOK\_IDs

all\_book\_ids = content\_df['BOOK\_ID'].unique()

user\_item\_matrix = ratings\_df.pivot\_table(index='USERID', columns='BOOK\_ID', values='RATINGS').fillna(0)

user\_item\_matrix = user\_item\_matrix.reindex(columns=all\_book\_ids, fill\_value=0)

item\_user\_matrix = user\_item\_matrix.T

#Compute item-item cosine similarity for collaborative filtering

collab\_similarity\_matrix = cosine\_similarity(item\_user\_matrix)

#####################################

# Content-Based Filtering Preparation

#####################################

#One-hot encode authors and create content similarity matrix

author\_ohe = pd.get\_dummies(content\_df['AUTHOR'])

content\_for\_similarity = pd.concat([content\_df[['BOOK\_ID']], author\_ohe], axis=1)

grouped\_bookid = content\_for\_similarity.groupby('BOOK\_ID', as\_index=False)[author\_ohe.columns].sum()

book\_features = grouped\_bookid.drop(columns=['BOOK\_ID'])

content\_sim\_matrix = cosine\_similarity(book\_features)

np.fill\_diagonal(content\_sim\_matrix, 0)

#Create aligned similarity DataFrames

collab\_sim\_df = pd.DataFrame(collab\_similarity\_matrix,

index=item\_user\_matrix.index,

columns=item\_user\_matrix.index)

content\_sim\_df = pd.DataFrame(content\_sim\_matrix,

index=grouped\_bookid['BOOK\_ID'],

columns=grouped\_bookid['BOOK\_ID'])

#Align matrices to common book IDs

common\_books = collab\_sim\_df.index.intersection(content\_sim\_df.index)

collab\_sim\_df = collab\_sim\_df.loc[common\_books, common\_books]

content\_sim\_df = content\_sim\_df.loc[common\_books, common\_books]

#Create hybrid similarity matrix

hybrid\_sim\_matrix = 0.5 \* collab\_sim\_df.values + 0.5 \* content\_sim\_df.values

hybrid\_sim\_df = pd.DataFrame(hybrid\_sim\_matrix,

index=common\_books,

columns=common\_books)

#Create book ID to title mapping

book\_id\_to\_title = ratings\_df[['BOOK\_ID', 'TITLE']].drop\_duplicates().set\_index('BOOK\_ID')['TITLE'].to\_dict()

#Select 10 random active users

active\_users = user\_item\_matrix[(user\_item\_matrix > 0).any(axis=1)].index.tolist()

random\_users = random.sample(active\_users, min(10, len(active\_users)))

print("\nHYBRID RECOMMENDATIONS FOR 10 RANDOM USERS")

for user\_id in random\_users:

print(f"\n--- Recommendations for User ID: {int(user\_id)} ---")

#Get user's ratings

user\_ratings = user\_item\_matrix.loc[user\_id]

#Get books the user rated highly

top\_read\_books = user\_ratings[user\_ratings > 0].nlargest(5).index

if len(top\_read\_books) == 0:

print(f"User {user\_id} hasn't rated any books.")

continue

print("\nBecause you enjoyed these books:")

for i, book\_id in enumerate(top\_read\_books, 1):

title = book\_id\_to\_title.get(book\_id, f"Book ID {book\_id}")

print(f"{i}. {title} (Rating: {int(user\_ratings[book\_id])})")

#Initialize score vector

book\_scores = np.zeros(len(common\_books))

#Score books based on hybrid similarity to liked books

for book\_id in top\_read\_books:

if book\_id in hybrid\_sim\_df.index:

book\_idx = hybrid\_sim\_df.index.get\_loc(book\_id)

similarity\_vector = hybrid\_sim\_matrix[book\_idx]

book\_scores += similarity\_vector \* user\_ratings[book\_id] # Weight by rating

#Remove already rated books

for book\_id in user\_ratings[user\_ratings > 0].index:

if book\_id in hybrid\_sim\_df.index:

idx = hybrid\_sim\_df.index.get\_loc(book\_id)

book\_scores[idx] = -1

#Get top recommended book IDs

top\_indices = book\_scores.argsort()[::-1]

recommended\_book\_ids = hybrid\_sim\_df.index[top\_indices]

#Get recommendation details

recommended\_books\_info = content\_df[

(content\_df['BOOK\_ID'].isin(recommended\_book\_ids)) &

(content\_df['IS\_EARLIEST'] == 1)

][['BOOK\_ID', 'TITLE', 'AUTHOR', 'YEAR']].drop\_duplicates(subset=['BOOK\_ID'])

#Limit to 2 books per author (weighting is not helping to diversify results completely)

final\_recommendations = []

author\_counts = {}

for book\_id in recommended\_book\_ids:

if len(final\_recommendations) >= 5:

break

book\_info = recommended\_books\_info[recommended\_books\_info['BOOK\_ID'] == book\_id]

if not book\_info.empty:

author = book\_info.iloc[0]['AUTHOR']

author\_counts[author] = author\_counts.get(author, 0) + 1

if author\_counts[author] <= 2:

final\_recommendations.append(book\_info.iloc[0])

print("\nWe think you'll enjoy these recommendations (max 2 per author):")

if len(final\_recommendations) == 0:

print("No recommendations available.")

else:

for i, row in enumerate(final\_recommendations[:5], 1): #Ensure we don't exceed 5 even with author limit

print(f"{i}. {row['TITLE']} ({int(row['YEAR'])}) by {row['AUTHOR']}")

print("\n" + "-" \* 80)