C3389C AY2021 Sem3 Cohort: 1
Individual Coursework Final (CWF) Submission

Personal Details	
Name	**WONG QI YUAN, JEFFREY**
Student No.	**20053371**

Instruction to Students

1. This coursework assignment is to be completed and submitted by each candidate
2. Student must ensure that it is their own work and must be responsible for the safeguarding of the assignment
3. The maximum score achievable for this coursework assignment is 60 marks.
4. Grading Criterion: As specified in CWF Report Template (next section).

1) Develop a prototype recommender system application in Python with the following requirements:

Part (a)

[8 marks] Utilises any open-source dataset with at least 100 items and 100 users. Very large datasets can be trimmed down to a manageable size. Sample datasets for consideration, use discretion in your selection:

- http://cseweb.ucsd.edu/~jmcauley/datasets.html
- http://eigentaste.berkeley.edu/dataset/
- etc.

Report requirements:

- Describe details about the dataset – the curator/ owner, the dataset fields (columns), number of records, time-span of the data collected, etc.
- Describe the reasons for selecting this dataset, e.g., in terms of suitability to use for recommender systems.
- Describe the key fields (columns) in the dataset that are used to generate recommendations.

Book-Crossing Dataset was collected by Cai-Nicolas Ziegler in a 4-week crawl (August/ September 2004) from the Book-Crossing community with kind permission from Ron Hornbaker, CTO of Humankind Systems. The book-crossing dataset contains 278,858 users (anonymized, but with demographic information) providing 1,149,780 ratings (explicit & implicit) about 271,379 books. The Book-Crossing dataset comprises of 3 tables: BX-Users, BX-Books, and BX-Book-Ratings.

For BX-Users, it contains the users where the user ID (‘*User-ID*’) have been anonymized and map to integers. Demographic data is also provided (‘*Location*’, ‘*Age*’) if available, otherwise, these fields might contain NULL-values. Figure 1.1 shows the BX-Users info and Figure 1.2 shows the sample DataFrame of BX-Users.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   User-ID   278858 non-null  int64
 1   Location  278858 non-null  object
 2   Age       168096 non-null  float64
dtypes: float64(1), int64(1), object(1)
```

Figure 1.1: BX-Users Info

users_df - DataFrame

Index	User-ID	Location	Age
0	1	nyc, new york, usa	nan
1	2	stockton, california, usa	18
2	3	moscow, yukon territory, russia	nan
3	4	porto, v.n.gaia, portugal	17
4	5	farnborough, hants, united kingdom	nan
5	6	santa monica, california, usa	61
6	7	washington, dc, usa	nan
7	8	timmins, ontario, canada	nan
8	9	germantown, tennessee, usa	nan
9	10	albacete, wisconsin, spain	26
10	11	melbourne, victoria, australia	14
11	12	fort bragg, california, usa	nan

Format Resize Background color Column min/max Save and Close Close

Figure 1.2: Sample DataFrame of BX-Users

For BX-Books, the books are identified by their respective ISBN. Invalid ISBNs have already been removed from the original dataset by Book-Crossing. Furthermore, some content-based information is given (‘*Book Title*’, ‘*Book-Author*’, ‘*Year-Of-Publication*’, ‘*Publisher*’), obtained from Amazon Web Services. URLs linking to cover images are also given, appearing in three different flavours (‘*Image-URL-S*’, ‘*Image-URL-M*’, ‘*Image-URL-L*’), i.e., small, medium, and large. These URLs point to the Amazon web site. Figure 1.3 shows the BX-Books info and Figure 1.4 shows the sample DataFrame of BX-Books.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 8 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ISBN                 271379 non-null  object
 1   Book-Title           271375 non-null  object
 2   Book-Author          233754 non-null  object
 3   Year-Of-Publication  232410 non-null  object
 4   Publisher            232408 non-null  object
 5   Image-URL-S          228148 non-null  object
 6   Image-URL-M          228148 non-null  object
 7   Image-URL-L          228148 non-null  object
dtypes: object(8)
```

Figure 1.3: BX-Books Info

books_df - DataFrame

Index	ISBN	Book-Title	Book-Author	Of-Public	Publisher
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	374157065	Flu: The Story of the Great Influenza Pandemic…	Gina Bari Kolata	1999	Farrar Straus Giroux
4	393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton & Company
5	399135782	The Kitchen God's Wife	Amy Tan	1991	Putnam Pub Group
6	425176428	What If?: The World's Foremost Military Historians Imagine What Might Have Been	Robert Cowley	2000	Berkley Publishing Group
7	671870432	PLEADING GUILTY	Scott Turow	1993	Audioworks
8	679425608	Under the Black Flag: The Romance and the Reality of Life Among the Pirates	David Cordingly	1996	Random House
9	074322678X	Where You'll Find Me: And Other Stories	Ann Beattie	2002	Scribner
10	771074670	Nights Below Station Street	David Adams Richards	1988	Emblem Editions
11	080652121X	Hitler's Secret Bankers: The Myth of Swiss Neutrality During the Holocaust	Adam Lebor	2000	Citadel Press
12	887841740	The Middle Stories	Sheila Heti	2004	House of Anansi Press
13	1552041778	Jane Doe	R. J. Kaiser	1999	Mira Books
14	1558746218	A Second Chicken Soup for the Woman's Soul (Chicken Soup for the Soul Series)	Jack Canfield	1998	Health Communications

Format Resize Background color Column min/max Save and Close Close

books_df - DataFrame

Index	Publisher	Image-URL-S	Image-URL-M	Image-URL-L
0	xford University Press	http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0195153448.01.MZZZZZZZ.jpg	http://images…
1	arperFlamingo Canada	http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0002005018.01.MZZZZZZZ.jpg	http://images…
2	arperPerennial	http://images.amazon.com/images/P/0060973129.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0060973129.01.MZZZZZZZ.jpg	http://images…
3	arrar Straus Giroux	http://images.amazon.com/images/P/0374157065.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0374157065.01.MZZZZZZZ.jpg	http://images…
4	. W. Norton & Company	http://images.amazon.com/images/P/0393045218.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0393045218.01.MZZZZZZZ.jpg	http://images…
5	utnam Pub Group	http://images.amazon.com/images/P/0399135782.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0399135782.01.MZZZZZZZ.jpg	http://images…
6	erkley Publishing Group	http://images.amazon.com/images/P/0425176428.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0425176428.01.MZZZZZZZ.jpg	http://images…
7	udioworks	http://images.amazon.com/images/P/0671870432.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0671870432.01.MZZZZZZZ.jpg	http://images…
8	andom House	http://images.amazon.com/images/P/0679425608.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0679425608.01.MZZZZZZZ.jpg	http://images…
9	cribner	http://images.amazon.com/images/P/074322678X.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/074322678X.01.MZZZZZZZ.jpg	http://images…
10	mblem Editions	http://images.amazon.com/images/P/0771074670.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0771074670.01.MZZZZZZZ.jpg	http://images…
11	itadel Press	http://images.amazon.com/images/P/080652121X.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/080652121X.01.MZZZZZZZ.jpg	http://images…
12	ouse of Anansi Press	http://images.amazon.com/images/P/0887841740.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/0887841740.01.MZZZZZZZ.jpg	http://images…
13	ira Books	http://images.amazon.com/images/P/1552041778.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/1552041778.01.MZZZZZZZ.jpg	http://images…
14	ealth Communications	http://images.amazon.com/images/P/1558746218.01.THUMBZZZ.jpg	http://images.amazon.com/images/P/1558746218.01.MZZZZZZZ.jpg	http://images…

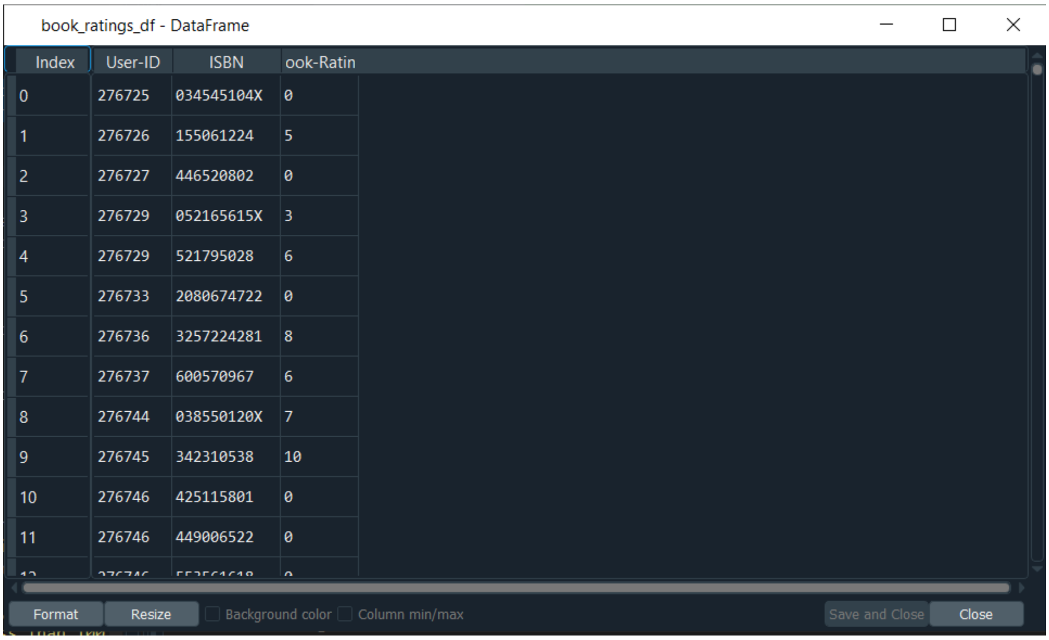
Format Resize Background color Column min/max Save and Close Close

Figure 1.4: Sample DataFrame of BX-Books

For BX-Book-Ratings, the table contains the book rating information. Ratings ('*Book-Rating*') are either explicit, expressed on a scale from 1 to 10 (higher values denoting higher appreciation), or implicit, expressed by 0. Figure 1.5 shows the BX-Book-Ratings info and Figure 1.6 shows the sample DataFrame of BX-Book-Ratings.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   User-ID      1048575 non-null  int64
 1   ISBN         1048574 non-null  object
 2   Book-Rating  1048570 non-null  float64
dtypes: float64(1), int64(1), object(1)
```

Figure 1.5: BX-Book-Ratings Info



book_ratings_df - DataFrame

Index	User-ID	ISBN	ook-Ratin
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0
3	276729	052165615X	3
4	276729	521795028	6
5	276733	2080674722	0
6	276736	3257224281	8
7	276737	600570967	6
8	276744	038550120X	7
9	276745	342310538	10
10	276746	425115801	0
11	276746	449006522	0

Figure 1.6: Sample DataFrame for BX-Book-Ratings

In addition, Book-Crossings Dataset consists on the basis of different knowledge sources that are particularly important in the implementation of the recommendation systems via Matrix Factorization in Keras. The knowledge sources include the user's features such as age and location, item features such as IBSN, Book ID, Book Title, Book Author, Year of Publication, etc, and user-item preferences data such as book-ratings. This user-item preference data creates a user profile that plays a crucial role in the recommendation systems. Hence, the datasets described above are necessary and important for the latter parts of the recommendation systems.

Basically, for BX-Book-Ratings, the most imperative columns are 'User-ID', 'ISBN', and 'Book-Ratings' as they are essentially used for train-test-splits, building and training the neural network based and subsequently used it for predictive purpose. As for the BX-Book-Ratings, the most important columns are 'ISBN', 'Book-Title', 'Book-Author', 'Year-Of-Publication', and 'Publisher' as these will be used to generate the books recommendation information. As for the BX-Users, most of the columns are not so practically useful in providing the book recommendation information as the table only consists of the 'User-ID', 'Demographic Location', and 'Age'.

Part (b)

[10 marks] The recommender system should be neural-network-based, i.e., based on Keras, Tensorflow, or other neural network models.

Report requirements

- Describe the steps required to implement a neural-network based recommender system, corresponding to the lines in your code. E.g.
  - Lines 5-7 to read the data from the csv file
  - Lines 10-11 to spilt data into training and testing
  - Lines 12-14 to define the number of layers in the neural network
  - Lines 16-18 to train the model
  - etc.

NOTE: The python file for neural-network implementation is *neural-network.py*

Define Functions	
Lines 15-38	Define function for embedding path for both user and book item and consists of the following items within the function itself: • To create the user embedding path. • To create the book embedding path.
Lines 42-67	Define function for neural network A and consists of the following items within the function itself: • To concatenate the user and book vectors. • To add the fully-connected dense layers with activation function and dropout. • To group the layers into an object with training and inference features such that create the model from inputs and outputs. • To compile the model with optimizer, loss, and metrics. • To return the model.
Lines 71-92	Define function for neural network B and consists of the following items within the function itself: • To concatenate the user and book vectors. • To add the fully-connected dense layers with dropout and activation function. • To group the layers into an object with training and inference features such that create the model from inputs and outputs. • To compile the model with optimizer, loss, and metrics.

	• To return the model.
Lines 96-117	Define function for neural network C and consists of the following items within the function itself: • To concatenate the user and book vectors. • To add the fully-connected dense layers with dropout and activation function. • To group the layers into an object with training and inference features such that create the model from inputs and outputs. • To compile the model with optimizer, loss, and metrics. • To return the model.
Lines 121-131	Define function for loss curve to display the plot for training and validation loss for the respective neural network.
Lines 136-146	Define function for MAE curve to display the plot for training and validation MAE for the respective neural network.
Lines 150-160	Define function for RMSE curve to display the plot for training and validation RMSE for the respective neural network.
Read the Data File	
Lines 165-166	To read the pre-processed data from the CSV file and store it into a new DataFrame called “preprocessed_df”.
Pre-processing	
Lines 169-170	To extract the relevant columns (i.e., user_id, book_id, and book_rating) from the “preprocessed_df” DataFrame and store it into a new DataFrame called “userItemRatings_df”. This newly assigned DataFrame is for deep training purpose.
Lines 172-176	To normalize the ‘book_rating’ column between 0 and 1.
Lines 179-180	To split the data into 70% training and 30% testing sets with random state of 42.
Embedding Path	
Lines 185-186	To count the number of unique users in the “userItemRatings_df” DataFrame and store it into a new variable called ‘num_unique_users’.

Lines 187-188	To count the number of unique books in the “userItemRatings_df” DataFrame and store it into a new variable called ‘num_unique_books’.
Lines 191-192	To define the latent factor.
Lines 194-195	Call the function to create the embedding path for both user and book features.
Model Building for Respective Neural Networks	
Lines 201-202	To create an early stopping object to minimize the chance of overfitting during the model training process.
Lines 204-205	Call the function to build the neural network for recommender system A and store it into a new variable called “model_A”.
Lines 207-208	Display the neural network summary for “model_A”.
Lines 210-217	Train and fit the “model_A” using the training set based on the following parameters: epochs = 30, batch-size = 512, validation-split = 0.2, callbacks = [early_stopping], verbose = 1.
Lines 221-222	Call the function to build the neural network for recommender system B and store it into a new variable called “model_B”.
Lines 224-225	Display the neural network summary for “model_B”.
Lines 227-234	Train and fit the “model_B” using the training set based on the following parameters: epochs = 30, batch-size = 512, validation-split = 0.2, callbacks = [early_stopping], verbose = 1.
Lines 238-239	Call the function to build the neural network for recommender system C and store it into a new variable called “model_C”.
Lines 241-242	Display the neural network summary for “model_C”.
Lines 244-251	Train and fit the “model_C” using the training set based on the following parameters: epochs = 30, batch-size = 512, validation-split = 0.2, callbacks = [early_stopping], verbose = 1.

Model Evaluation for Respective Neural Networks	
Lines 257-258	Call the function to plot the loss curve for recommender system A.
Lines 260-261	Call the function to plot the MAE curve for recommender system A.
Lines 263-264	Call the function to plot the RMSE curve for recommender system A.
Lines 266-270	To evaluate the performance of recommender system A based on loss, MAE and MSE using the testing set.
Lines 274-275	Call the function to plot the loss curve for recommender system B.
Lines 277-278	Call the function to plot the MAE curve for recommender system B.
Lines 280-281	Call the function to plot the RMSE curve for recommender system B.
Lines 283-287	To evaluate the performance of recommender system B based on loss, MAE and MSE using the testing set.
Lines 291-292	Call the function to plot the loss curve for recommender system C.
Lines 294-295	Call the function to plot the MAE curve for recommender system C.
Lines 297-298	Call the function to plot the RMSE curve for recommender system C.
Lines 300-304	To evaluate the performance of recommender system C based on loss, MAE and MSE using the testing set.
Optimize the Latent Factor for the Best Recommender System (Model B)	
Lines 309-310	Initialize the RMSE score for grid-search process.
Lines 311-312	Initialize the MAE score for grid-search process.
Lines 314-349	Using a simple grid-search process via for-loops over a list of latent factor elements to find the best latent factor for Model B based on the RMSE and MAE scores.

Lines 352-353	Display the best RMSE score after the grid-search process.
Lines 354-355	Display the best MAE score after the grid-search process.
Lines 356-357	Display the best latent factor after the grid-search process.
Lines 359-360	Store the best latent factor into Pandas Series.
Rebuild the Recommender System (Model B) with Best Latent Factor	
Lines 365-366	Define the best latent factor based on the grid-search result.
Lines 368-369	Call the function to create the embedding path for both user and book features.
Lines 371-372	Call the function to build the neural network for recommender system B and store it into a new variable called “remodel_B”.
Lines 374-375	Display the neural network summary for “remodel_B”.
Lines 377-384	Train and fit the “remodel_B” using the training set based on the following parameters: epochs = 30, batch-size = 512, validation-split = 0.2, callbacks = [early_stopping], verbose = 1.
Evaluate the Recommender System (Model B) with Best Latent Factor	
Lines 390-391	Call the function to plot the loss curve for recommender system B with the best latent factor.
Lines 393-394	Call the function to plot the MAE curve for recommender system B with the best latent factor.
Lines 396-397	Call the function to plot the RMSE curve for recommender system B with the best latent factor.
Lines 399-402	To evaluate the performance of recommender system B with the best latent factor based on loss, MAE and MSE using the testing set.
Save the Final Model (Model B)	
Line 406	Save the final model B into the .h5 file.

Part (c)

[10 marks] You should highlight at least one pre-processing step that has to be applied to the open-source dataset to make it suitable for your project. Some examples include: trimming size of dataset, handling null values, extracting specific values from a field (e.g., year from a data field), etc.

Report requirements
You should elaborate on:

- the dataset before the pre-processing step
- the Python code applied to the dataset, which should be bug-free, well-structured, and well-documented (with code comments).
- the dataset after the pre-processing step, describing the changes.

Deliverables: all three bullet points above

NOTE: The python file for pre-processing step is *preprocessing.py*

Pre-processing For Books DataFrame:

1) Changing of the column names

   a. Prior to the pre-processing, most of the column names come with **capital letters that are used as the first letter of every word**, and some of the header columns are **using hyphens to connect two words into a single word.** Figure 1.7 shows the column names for Books DataFrame.

books_df - DataFrame

Index	ISBN	Book-Title	Book-Author	Year-Of-Publication	
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford Ur
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFla
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPer
3	374157065	Flu: The Story of the Great Influenza Pandemic…	Gina Bari Kolata	1999	Farrar St
4	393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Nor

Figure 1.7: Column Names for Books DataFrame (Before)

b. In the Python code,
   i. Line 52: using the lower () function to convert the respective column name into lower-case format.
   ii. Line 54: using the replace () function to replace the hyphens (-) with underscore (_).

```
49  # before changing the column names in the books dataframe
50  print(books_df.head())
51  # convert all the column name into lower cases
52  books_df.columns = books_df.columns.str.lower()
53  # convert all the column name with hyphen "-" into underscore "_" by using
54  books_df.columns = books_df.columns.str.replace('-', '_')
55  # after changing the column names in the books dataframe
56  print(books_df.head())
```

Figure 1.8: Python Code

c. After pre-processing, most of the column names are converted into a lower-case format and replace those words with hyphens with an underscore.

books_df - DataFrame

Index	isbn	book_title	book_author	year_of_publication	
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford Unive
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamin
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPerenn
3	374157065	Flu: The Story of the Great Influenza Pandemic…	Gina Bari Kolata	1999	Farrar Strau
4	393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton
5	399135782	The Kitchen God's Wife	Amy Tan	1991	Putnam Pub G

Figure 1.9: Column Names for Books DataFrame (After)

2) Drop the irrelevant columns

   a. Prior to the pre-processing, there are three irrelevant image-URL link columns that aren't significant for recommendation systems. As such, these columns are to be remove from the books DataFrame. Figure 1.10 shows the Books DataFrame with image-URL link columns.

books_df - DataFrame

Index	book_author	of_public	publisher	image_url_s	image_url_m	image_url_l
0	. O. Morford	2002	Oxford Univer…	http://images…	http://images…	http://images…
1	ı Bruce Wright	2001	HarperFlamingo Canada	http://images…	http://images…	http://images…
2	)'Este	1991	HarperPerenni…	http://images…	http://images…	http://images…
3	ıri Kolata	1999	Farrar Straus Giroux	http://images…	http://images…	http://images…
4	[illegible]	[illegible]	W. W. Norton	[illegible]	[illegible]	[illegible]

Figure 1.10: Books DataFrame with image-URL link columns (Before)

   b. In the Python code,
      i. Line 63: Using the Drop () function to drop down the irrelevant image-URL link columns.

```
60  # before drop down the irrelevant url columns
61  print(books_df.head())
62  # drop the irrelevant url columns
63  books_df.drop(columns = ['image_url_s', 'image_url_m', 'image_url_l'], inplace = True)
64  # after drop down the irrelevant url columns
65  print(books_df.head())
```

Figure 1.11: Python Code

   c. After pre-processing, all the irrelevant image-URL link columns are removed by using the Drop (). Figure 1.12 shows the Books DataFrame with image-URL link removed.

books_df - DataFrame

Index	isbn	book_title	book_author	of_public	publisher
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	374157065	Flu: The Story of the Great Influenza Pandemic…	Gina Bari Kolata	1999	Farrar Straus Giroux

Figure 1.12: Books DataFrame with image-URL link removed (After)

3) Changing the Data Type for **Year of Publication** column

a. Prior to the pre-processing, the data type for **'year of publication'** column is in 'object' format. Figure 1.13 shows the Data Columns Info for Books DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   isbn                 271379 non-null  object
 1   book_title           271375 non-null  object
 2   book_author          233754 non-null  object
 3   year_of_publication  232410 non-null  object
 4   publisher            232408 non-null  object
dtypes: object(5)
```

Figure 1.13: Data Columns Info for Books DataFrame (Before)

b. In the Python code,

i. Line 72: using the numeric () function to convert the object data type into integer data type.

```
69  # before changing the data type for year of publication column
70  print(books_df.info())
71  # convert the year of publication column into float instead of object type
72  books_df['year_of_publication'] = pd.to_numeric(books_df['year_of_publication'], errors = 'coerce')
73  # after changing the data type for year of publication column
74  print(books_df.info())
```

Figure 1.14: Python Code

c. After pre-processing, the data type for '**year of publication'** column has been changed to 'integer' format. Figure 1.15 shows the Data Columns Info for Books DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   isbn                 271379 non-null  object
 1   book_title           271375 non-null  object
 2   book_author          233754 non-null  object
 3   year_of_publication  232396 non-null  float64
 4   publisher            232408 non-null  object
dtypes: float64(1), object(4)
```

Figure 1.15: Data Columns Info for Books DataFrame (After)

4) Replace the Zeros, Empty Strings, and NaN values in the **Year of Publication** columns with mean values.

   a. Prior to pre-processing, there are about 0 empty strings, 4465 zero values, and NaN values in the 'year of publication' column in the Books DataFrame. Figure 1.16 shows the results of the empty strings, zero values, and NaN values before replacing with mean values.

```
Number of empty strings in the Year of Publication column: 0
Number of zero values in the Year of Publication column: 4465
Number of NaN values in the Year of Publication column: 38983
```

Figure 1.16: The results of the empty strings, zero values, and NaN values for Year of Publication column in the Books DataFrame (Before)

   b. In the Python Code,
      i. Line 91: using replace () function to replace all the year of publication with zero values with NaN values.
      ii. Line 93: using the fillna () function to replace the NaN values with mean value of the years.

```
90 # replace all the years with zero values with NaN with replace () function
91 books_df['year_of_publication'].replace(0, np.nan, inplace = True)
92 # replace all the years with NaN values with mean value using fillna() function
93 books_df['year_of_publication'].fillna(books_df['year_of_publication'].mean(), inplace = True)
```

Figure 1.17: Python Code

   c. After pre-processing, there are about 0 empty strings, no zero values, and zero NaN values in the 'year of publication' column in the Books DataFrame. Figure 1.17 shows the results after replacing the empty strings, zero values, and NaN values with mean values.

```
Number of empty strings in the Year of Publication column: 0
Number of zero values in the Year of Publication column: 0
Number of NaN values in the Year of Publication column: 0
```

Figure 1.18: The results of the empty strings, zero values, and NaN values for Year of Publication column in the Books DataFrame (After)

5) Remove the old books that belongs before 1900s as it tends to skew the model and seems to be irrelevant in this context. Some of the future books after 2021 appeared to be errors such as Alice in Wonderland should be in 1950s instead, etc

future_books - DataFrame

Index	isbn	book_title	book_author	year_of_publication	publisher
37488	671746103	MY TEACHER FRIED MY BRAINS (RACK SIZE) (MY TEACHER BOOKS)	Coville	2030	Aladdin
55679	671791990	MY TEACHER FLUNKED THE PLANET (RACK SIZE) (MY TEACHER BOOKS)	Bruce Coville	2030	Aladdin
78171	870449842	Crossing America	National Geographic Society	2030	National Geographic
80267	140301690	Alice's Adventures in Wonderland and Through the Looking Glass (Puffin Books)	Lewis Carroll	2050	Puffin Books
97830	140201092	Outline of European Architecture (Pelican S.)	Nikolaus Pevsner	2050	Penguin USA
116058	394701658	Three Plays of Eugene Oneill	Eugene O'Neill	2038	Vintage Books USA
118299	3442436893	Das groÃ?Â?e BÃ?Â¶se- MÃ?Â¤dchen- Lesebuch.	Kathy Lette	2026	Goldmann
228187	671266500	FOREST PEOPLE (Touchstone Books (Hardcover))	Colin M. Turnbull	2030	Simon & Schuster
240184	684718022	In Our Time: Stories (Scribner Classic)	Ernest Hemingway	2030	Collier Books
246858	380000059	CLOUT	D. GIBBONS	2024	Avon
255426	068471809X	To Have and Have Not	Ernest Hemingway	2037	Simon & Schuster
260992	671740989	FOOTBALL SUPER TEAMS : FOOTBALL SUPER TEAMS	Bill Gutman	2030	Simon & Schuster Children's Publishing

Figure 1.19: Future Books (Beyond Year 2021)

old_books - DataFrame

Index	isbn	book_title	book_author	of_public	publisher
227544	9643112136	Dalan-i bihisht (Dastan-i Irani)	Nazi Safavi	1378	Intisharat-i Quqnus
253767	964442011X	Tasht-i khun	IsmaÂºil Fasih	1376	Nashr-i Alburz

Figure 1.20: Old Books (Before 1900s)

a. Prior to the pre-processing, there are about 271,379 books in the Books DataFrame.

b. In the Python code,
    i. Line 129: to remove the old books
    ii. Line 130: to remove the future books

```
127 # old books tends to skew the model and seems to be irrelevant in this context,
128 print("Length of books before removal: {}".format(len(books_df)))
129 books_df = books_df.loc[~(books_df.isbn.isin(old_books.isbn))]
130 books_df = books_df.loc[~(books_df.isbn.isin(future_books.isbn))]
131 print("Length of books after removal: {}".format(len(books_df)))
132 print()
```

Figure 1.21: A Python Code

c. After pre-processing, there are about 271,365 books in the Books DataFrame after removal of the old books and future books.

6) Replace ampersand (&) in the Publisher Data Column with (&) in the Books DataFrame

   a. Prior to pre-processing, the ampersand formatting (&) appeared in the Publisher Column data. Figure 1.22 shows the ampersand formatting appeared in the ‘Publisher’ column data.

```
Before replacing the ampersand formatting:
=====================================================
        isbn  ...                   publisher
0  195153448  ...     Oxford University Press
1    2005018  ...      HarperFlamingo Canada
2   60973129  ...            HarperPerennial
3  374157065  ...        Farrar Straus Giroux
4  393045218  ...   W. W. Norton &amp; Company
```

Figure 1.22: Ampersand formatting appeared in the ‘Publisher’ column data (Before)

   b. In Python code,
      i. Line 141: using the replace () to replace the ampersand formatting with (&).

```
136  # replace the ampersand formmating in the Publisher data column with &
137  print("Before replacing the ampersand formatting: ")
138  print("=============================================================================================")
139  print(books_df.head())
140  print()
141  books_df['publisher'] = books_df['publisher'].str.replace('&amp;', '&', regex = False)
142  print("After replacing the ampersand formatting: ")
143  print("=============================================================================================")
144  print(books_df.head())
145  print()
```

Figure 1.23: A Python Code

   c. After pre-processing, the ampersand formatting has been replaced with (&) in the Publisher Data Column. Figure 1.24 shows the ampersand formatting has been replaced with (&) in the ‘Publisher’ column data

```
After replacing the ampersand formatting:
=============================================
        isbn  ...                publisher
0  195153448  ...  Oxford University Press
1    2005018  ...    HarperFlamingo Canada
2   60973129  ...          HarperPerennial
3  374157065  ...     Farrar Straus Giroux
4  393045218  ...   W. W. Norton & Company
```

Figure 1.24: Ampersand formatting replaced with (&) in the ‘Publisher’ column data (After)

7) Remove rows that consists of <u>empty strings, zero values, and NaN values</u> in the **Publisher** Column in the Books DataFrame

   a. Prior to the pre-processing, there are 0 empty strings, no zero values, and 39,034 NaN values found in the Publisher column in the Books DataFrame. Figure 1.25 shows the total number of empty strings, zero values, and NaN values in the ‘Publisher’ column.

```
Number of empty strings in the Publisher column: 0
Number of zero values in the Publisher column: 0
Number of NaN values in the Publisher column: 39034
```

Figure 1.25: Total number of empty strings, zero values, and NaN values in the ‘Publisher’ column (Before)

   b. In Python code,
      i. Line 167: using dropna () function to remove the rows with NaN values that subset with the publisher column.

```
166    # remove the rows that consists of NaN found in the publisher column
167    books_df = books_df.dropna(subset = ['publisher'])
```

Figure 1.26: Python Code

   c. After pre-processing, there are 0 empty strings, no zero values, and 0 NaN values found in the Publisher column in the Books DataFrame. Figure 1.27 shows the total number of empty strings, zero values, and NaN values in the Publisher column.

```
Number of empty strings in the Publisher column: 0
Number of zero values in the Publisher column: 0
Number of NaN values in the Publisher column: 0
```

Figure 1.27: Total number of empty strings, zero values, and NaN values in the ‘Publisher’ column (After)

8) Remove rows that consists of <u>empty strings, zero values, and NaN values</u> in the **Book Author** column in the Books DataFrame

   a. Prior to the pre-processing, there are 0 empty strings, no zero values, and only 1 NaN values found in the Book Author column in the Book DataFrame. Figure 1.28 shows the total number of empty strings, zero values, and NaN values in the Book Author column.

```
Number of empty strings in the Book Author column: 0
Number of zero values in the Book Author column: 0
Number of NaN values in the Book Author column: 1
```

Figure 1.28: Total number of empty strings, zero values, and NaN values in the 'Book Author' column (Before)

   b. In Python code,
      i. Line 196: using dropna () function to remove the rows with NaN values that subset with the book_author column

```
195  # remove the rows that with NaN values that subset with book-author column
196  books_df = books_df.dropna(subset = ['book_author'])
```

Figure 1.29: Python code

   c. After pre-processing, there are about 0 empty strings, no zero values, and 0 NaN values found in the Book Author column in the Books DataFrame. Figure 1.30 shows the total number of empty strings, zero values, and NaN values in the Book Author column.

```
Number of empty strings in the Book Author column: 0
Number of zero values in the Book Author column: 0
Number of NaN values in the Book Author column: 0
```

Figure 1.30: Total number of empty strings, zero values, and NaN values in the Book Author column (After)

Pre-processing For Users DataFrame:

1) Changing of the column names
   a. Prior to the pre-processing, most of the column names come with **capital letters that are used as the first letter of every word**, and some of the header columns are **using hyphens to connect two words into a single word.** Figure 1.31 shows the column names for Books DataFrame.

users_df - DataFrame

Index	User-ID	Location	Age
0	1	nyc, new york, usa	nan
1	2	stockton, california, usa	18
2	3	moscow, yukon territory, russia	nan
3	4	porto, v.n.gaia, portugal	17
4	5	farnborough, hants, united kingdom	nan
5	6	santa monica, california, usa	61
6	7	washington, dc, usa	nan
7	8	timmins, ontario, canada	nan

Figure 1.31: Column names for Users DataFrame (Before)

   b. In Python code,
      i. Line 228: using the lower () function to convert the respective column name into lower-case format.
      ii. Line 230: using the replace () function to replace the hyphens (-) with underscore (_).

```
225    # before changing the column names in the users dataframe
226    print(users_df.head())
227    # convert all the column name into lower cases
228    users_df.columns = users_df.columns.str.lower()
229    # convert all the column name with dash "-" into underscore "_" by using the replace () function
230    users_df.columns = users_df.columns.str.replace('-', '_')
231    # after changing the column names in the users dataframe
232    print(users_df.head())
```

Figure 1.32: Python code

c. After pre-processing, most of the column names are converted into a lower-case format and replace those words with hyphens with an underscore.

users_df - DataFrame

Index	user_id	location	age
0	1	nyc, new york, usa	nan
1	2	stockton, california, usa	18
2	3	moscow, yukon territory, russia	nan
3	4	porto, v.n.gaia, portugal	17
4	5	farnborough, hants, united kingdom	nan
5	6	santa monica, california, usa	61

Figure 1.33: Column Names for Users DataFrame (After)

2) Setting NaN values for Age group before 5 years old and 100 years old

a. Prior to the pre-processing, there are some unrealistic ages such as above 100 years old as well as below 5 years old. It is nearly impossible for a human to live beyond 100 years old and limited knowledge on providing ratings for age below 5 years old. Figure 1.34 shows a list of the unique age found in the Users DataFrame.

```
Before setting null values for age group before 5 years old and above 100 years old
==========================================================================================
[ nan  18.  17.  61.  26.  14.  25.  19.  46.  55.  32.  24.  20.  34.
  23.  51.  31.  21.  44.  30.  57.  43.  37.  41.  54.  42.  50.  39.
  53.  47.  36.  28.  35.  13.  58.  49.  38.  45.  62.  63.  27.  33.
  29.  66.  40.  15.  60.   0.  79.  22.  16.  65.  59.  48.  72.  56.
  67.   1.  80.  52.  69.  71.  73.  78.   9.  64. 103. 104.  12.  74.
  75. 231.   3.  76.  83.  68. 119.  11.  77.   2.  70.  93.   8.   7.
   4.  81. 114. 230. 239.  10.   5. 148. 151.   6. 101. 201.  96.  84.
  82.  90. 123. 244. 133.  91. 128.  94.  85. 141. 110.  97. 219.  86.
 124.  92. 175. 172. 209. 212. 237.  87. 162. 100. 156. 136.  95.  89.
 106.  99. 108. 210.  88. 199. 147. 168. 132. 159. 186. 152. 102. 116.
 200. 115. 226. 137. 207. 229. 138. 109. 105. 228. 183. 204.  98. 223.
 113. 208. 107. 157. 111. 146. 118. 220. 143. 140. 189. 127.]
```

Figure 1.34: A list of unique age found in the Users DataFrame (Before)

b. In Python code,

i. Line 253: Using Boolean Indexing to set the age group below 5 years old and above 100 years old as NaN values.

```
246  # set null values for age group before 5 years old and above 100 years old
247  print("Before setting null values for age group before 5 years old and above 100 years old")
248  print("==========================================================================================")
249  print(users_df.age.unique())
250  print()
251  print("After setting null values for age group before 5 years old and above 100 years old")
252  print("==========================================================================================")
253  users_df.loc[(users_df['age'] < 5) | (users_df['age'] > 100)] = np.nan
254  print(users_df.age.unique())
255  print()
```

Figure 1.35: Python code

c. After pre-processing, the age group below 5 years and above 100 years old are not found in the list of the unique age as being converted into NaN values. Figure 1.36 shows a list of the unique age found in the Users DataFrame.

```
After setting null values for age group before 5 years old and above 100 years old
==========================================================================================
[ nan  18.  17.  61.  26.  14.  25.  19.  46.  55.  32.  24.  20.  34.
  23.  51.  31.  21.  44.  30.  57.  43.  37.  41.  54.  42.  50.  39.
  53.  47.  36.  28.  35.  13.  58.  49.  38.  45.  62.  63.  27.  33.
  29.  66.  40.  15.  60.  79.  22.  16.  65.  59.  48.  72.  56.  67.
  80.  52.  69.  71.  73.  78.   9.  64.  12.  74.  75.  76.  83.  68.
  11.  77.  70.  93.   8.   7.  81.  10.   5.   6.  96.  84.  82.  90.
  91.  94.  85.  97.  86.  92.  87. 100.  95.  89.  99.  88.  98.]
```

Figure 1.36: A list of the unique age found in the Users DataFrame (After)

3) Remove rows that consists of empty strings, zero values, and NaN values in the **Age** column in the Users DataFrame

a. Prior to the pre-processing, there are about 0 empty strings, no zero values, and 112,010 NaN values found in the Age column in the Users DataFrame. Figure 1.37 shows the total number of empty strings, zero values, and NaN values found in the Age column in the Users DataFrame.

```
Number of empty strings in the User Age column: 0
Number of zero values in the User Age column: 0
Number of NaN values in the User Age column: 112010
```

Figure 1.37: Total number of empty strings, zero values, and NaN values in the Age column (Before)

b. In Python code,

   i. Line 282: using dropna () function to remove the rows with NaN values that subset with the age column

```
270 # count the number of empty strings, zero values, and NaN values before removing the NaN
271 empty_values_age = users_df[users_df['age'] == ''']['age'].count()
272 zero_values_age = users_df[users_df['age'] == 0]['age'].count()
273 null_values_age = users_df['age'].isnull().sum()
274
275 # display the total number of empty strings, zero values, and NaN values before removing the NaN
276 print("Number of empty strings in the User Age column: {}".format(empty_values_age))
277 print("Number of zero values in the User Age column: {}".format(zero_values_age))
278 print("Number of NaN values in the User Age column: {}".format(null_values_age))
279 print()
280
281 # remove the rows with NaN that subset with the age columns
282 users_df = users_df.dropna(subset = ['age'])
283
284 # count the number of empty strings, zero values, and NaN values after removing the NaN
285 empty_values_age = users_df[users_df['age'] == ''']['age'].count()
286 zero_values_age = users_df[users_df['age'] == 0]['age'].count()
287 null_values_age = users_df['age'].isnull().sum()
288
289 # display the total number of empty strings, zero values, and NaN values after removing the NaN
290 print("Number of empty strings in the User Age column: {}".format(empty_values_age))
291 print("Number of zero values in the User Age column: {}".format(zero_values_age))
292 print("Number of NaN values in the User Age column: {}".format(null_values_age))
293 print()
```

Figure 1.38: Python code

c. After pre-processing, there are about 0 empty strings, no zero values, and 0 NaN values found in the Age column in the Users DataFrame. Figure 1.39 shows the total number of empty strings, zero values, and NaN values in the Age column in the Users DataFrame.

```
Number of empty strings in the User Age column: 0
Number of zero values in the User Age column: 0
Number of NaN values in the User Age column: 0
```

Figure 1.39: Total number of empty strings, zero values, and NaN values in the Age column (After)

4) Expand the Location column into City, State and Country columns in the Users DataFrame

a. Prior to pre-processing, all of the city, state and country info are stored in the location column. Figure 1.40 shows the city, state, and country stored under the Location column in the Users DataFrame.

users_df - DataFrame

Index	user_id	location	age
1	2	stockton, california, usa	18
3	4	porto, v.n.gaia, portugal	17
5	6	santa monica, california, usa	61
9	10	albacete, wisconsin, spain	26
10	11	melbourne, victoria, australia	14
12	13	barcelona, barcelona, spain	26

Figure 1.40: City, State, and Country are stored under the Location column (Before)

b. In Python code,

i. Line 304: Using split () function to split the string found in the Location column based on the separator specified in the split function, that is, the commas, and then forming a new DataFrame.

ii. Line 305: Create the column names for the new DataFrame specified in Line 304.

iii. Line 306: Join the new DataFrame specified in Line 304 with the existing Users DataFrame.

```
298  # before expanding the location column into city, state, and country columns
299  print("Before expanding the location column into city, state, and country columns ")
300  print("===========================================================================================")
301  print(users_df.head())
302  print()
303
304  users_df_location = users_df['location'].str.split(',', 2, expand = True)
305  users_df_location.columns = ['city', 'state', 'country']
306  users_df = users_df.join(users_df_location)
307
308  print("After expanding the location column into city, state, and country columns ")
309  print("===========================================================================================")
310  print(users_df.head())
311  print()
```

Figure 1.41: Python Code

c. After pre-processing, new columns for City, State and Country respectively are formed in the Users DataFrame.

users_df - DataFrame

Index	user_id	location	age	city	state	country
1	2	stockton, california, usa	18	stockton	california	usa
3	4	porto, v.n.gaia, portugal	17	porto	v.n.gaia	portugal
5	6	santa monica, california, usa	61	santa monica	california	usa
9	10	albacete, wisconsin, spain	26	albacete	wisconsin	spain
10	11	melbourne, victoria, australia	14	melbourne	victoria	australia

Figure 1.42: City, State and Country are formed as a new column in the Users DataFrame (After)

5) Remove rows that consists of empty strings, zero values, and NaN values in the **city** column in the Users DataFrame

a. Prior to pre-processing, there are about 48 empty strings, no zero values, and 0 NaN values found in the City column in the Users DataFrame. Figure 1.43 shows the total number of empty strings, zero values, and NaN values found in the City column in the Users DataFrame.

```
Number of empty strings in the City column: 48
Number of zero values in the City column: 0
Number of NaN values in the City column: 0
```

Figure 1.43: Total number of empty strings, zero values, and null values in the city column (Before)

b. In Python code,
   i. Line 328: using the replace () function to replace the empty string with the NaN values
   ii. Line 330: using dropna () function to remove the rows with NaN that subset with the city column

```
316 # count the number of empty string, zero values and null values in the city column before removing NaN
317 empty_values_city = users_df[users_df['city'] == ''][‘city'].count()
318 zero_values_city = users_df[users_df['city'] == 0]['city'].count()
319 null_values_city = users_df['city'].isnull().sum()
320
321 # display the number of empty string, zero values and null values in the city column before removing NaN
322 print("Number of empty strings in the City column: {}".format(empty_values_city))
323 print("Number of zero values in the City column: {}".format(zero_values_city))
324 print("Number of NaN values in the City column: {}".format(null_values_city))
325 print()
326
327 # replace the empty strings with NaN values
328 users_df['city'].replace('', np.nan, inplace = True)
329 # remove the rows consists of NaN that subsets with city column
330 users_df = users_df.dropna(subset = ['city'])
331
332 # count the number of empty string, zero values and null values in the city column after removing NaN
333 empty_values_city = users_df[users_df['city'] == ''][‘city'].count()
334 zero_values_city = users_df[users_df['city'] == 0]['city'].count()
335 null_values_city = users_df['city'].isnull().sum()
336
337 # display the number of empty string, zero values and null values in the city column after removing NaN
338 print("Number of empty strings in the City column: {}".format(empty_values_city))
339 print("Number of zero values in the City column: {}".format(zero_values_city))
340 print("Number of NaN values in the City column: {}".format(null_values_city))
341 print()
```

Figure 1.44: Python Code

c. After pre-processing, there are about 0 empty strings, no zero values, and 0 NaN values found in the city column in the Users DataFrame. Figure 1.45 shows the total number of empty strings, zero values, and NaN values found in the city column in the Users DataFrame.

```
Number of empty strings in the City column: 0
Number of zero values in the City column: 0
Number of NaN values in the City column: 0
```

Figure 1.45: Total number of empty strings, zero values, and NaN values found in the city column (After)

6) Remove rows that consists of empty strings, zero values, and NaN values in the **state** column in the Users DataFrame

   a. Prior to pre-processing, there are about 142 empty strings, no zero values, and 1 NaN values found in the State column in the Users DataFrame. Figure 1.46 shows the total number of empty strings, zero values, and NaN values found in the State column in the Users DataFrame.

```
Number of empty strings in the State column: 142
Number of zero values in the State column: 0
Number of NaN values in the State column: 1
```

Figure 1.46: Total number of empty strings, zero values, and NaN values found in the State column (Before)

   b. In Python code,
      i. Line: 359: using the replace () function to replace the empty string with the NaN values
      ii. Line 361: using dropna () function to remove the rows with NaN that subset with the state column

```
347 # count the number of empty string, zero values, and null values in the state column before removing NaN
348 empty_values_state = users_df[users_df['state'] == ''] ['state'].count()
349 zero_values_state = users_df[users_df['state'] == 0]['state'].count()
350 null_values_state = users_df['state'].isnull().sum()
351
352 # display the total number of empty string, zero values, and null values in the state column before removing NaN
353 print("Number of empty strings in the State column: {}".format(empty_values_state))
354 print("Number of zero values in the State column: {}".format(zero_values_state))
355 print("Number of NaN values in the State column: {}".format(null_values_state))
356 print()
357
358 # replace the empty strings with NaN values
359 users_df['state'].replace('', np.nan, inplace = True)
360 # remove the rows consists of NaN that subsets with state column
361 users_df = users_df.dropna(subset = ['state'])
362
363 # count the number of empty string, zero values, and null values in the state column after removing NaN
364 empty_values_state = users_df[users_df['state'] == ''] ['state'].count()
365 zero_values_state = users_df[users_df['state'] == 0]['state'].count()
366 null_values_state = users_df['state'].isnull().sum()
367
368 # display the total number of empty string, zero values, and null values in the state column after removing NaN
369 print("Number of empty strings in the State column: {}".format(empty_values_state))
370 print("Number of zero values in the State column: {}".format(zero_values_state))
371 print("Number of NaN values in the State column: {}".format(null_values_state))
372 print()
```

Figure 1.47: Python Code

c. After pre-processing, there are about 0 empty strings, no zero values, and 0 NaN values found in the State column in the Users DataFrame. Figure 1.48 shows the total number of empty strings, zero values, and NaN values found in the State column in the Users DataFrame.

```
Number of empty strings in the State column: 0
Number of zero values in the State column: 0
Number of NaN values in the State column: 0
```

Figure 1.48: Total number of empty strings, zero values, and NaN values found in the State column (After)

7) Remove rows that consists of empty strings, zero values, and NaN values in the **country** column in the Users DataFrame

   a. Prior to pre-processing, there are about 1940 empty strings, no zero values, and 1 NaN values found in the Country column in the Users DataFrame. Figure 1.49 shows the total number of empty strings, zero values, and NaN values found in the Country column in the Users DataFrame.

```
Number of empty strings in the Country column: 1940
Number of zero values in the Country column: 0
Number of null values in the Country column: 1
```

Figure 1.49: Total number of empty strings, zero values, and NaN values found in the Country column (Before)

   b. In Python code,
      i. Line: 389: using the replace () function to replace the empty string with the NaN values
      ii. Line 391: using dropna () function to remove the rows with NaN that subset with the city column

```
377 # count the number of empty string, zero values, and null values in the country column before removing NaN
378 empty_values_country = users_df[users_df['country'] == ''']['country'].count()
379 zero_values_country = users_df[users_df['country'] == 0]['country'].count()
380 null_values_country = users_df['country'].isnull().sum()
381
382 # display the total number of empty string, zero values, and null values in the country column before removing NaN
383 print("Number of empty strings in the Country column: {}".format(empty_values_country))
384 print("Number of zero values in the Country column: {}".format(zero_values_country))
385 print("Number of null values in the Country column: {}".format(null_values_country))
386 print()
387
388 # replace the empty strings with NaN values
389 users_df['country'].replace('', np.nan, inplace = True)
390 # remove the rows consists of NaN that subsets with country column
391 users_df = users_df.dropna(subset = ['country'])
392
393 # count the number of empty string, zero values, and null values in the country column after removing NaN
394 empty_values_country = users_df[users_df['country'] == ''']['country'].count()
395 zero_values_country = users_df[users_df['country'] == 0]['country'].count()
396 null_values_country = users_df['country'].isnull().sum()
397
398 # display the total number of empty string, zero values, and null values in the country column after removing NaN
399 print("Number of empty strings in the Country column: {}".format(empty_values_country))
400 print("Number of zero values in the Country column: {}".format(zero_values_country))
401 print("Number of null values in the Country column: {}".format(null_values_country))
402 print()
```

Figure 1.50: Python Code

c. After pre-processing, there are about 0 empty strings, no zero values, and 0 NaN values found in the Country column in the Users DataFrame. Figure 1.51 shows the total number of empty strings, zero values, and NaN values found in the Country column in the Users DataFrame.

```
Number of empty strings in the Country column: 0
Number of zero values in the Country column: 0
Number of null values in the Country column: 0
```

Figure 1.51: Total number of empty strings, zero values, and NaN values found in the Country column (After)

8) Converting the data type for **Age** column from Float to Integer

a. Prior to pre-processing, the data type for Age column in the Users DataFrame is 'Float'. Figure 1.52 shows the data type info for the Users DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 164716 entries, 1 to 278854
Data columns (total 6 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   user_id   164716 non-null  float64
 1   location  164716 non-null  object
 2   age       164716 non-null  float64
 3   city      164716 non-null  object
 4   state     164716 non-null  object
 5   country   164716 non-null  object
```

Figure 1.52: Data Type Info for the Users DataFrame (Before)

b. In Python code,

i. Line 421: using the astype () function to convert the data type for the age column into integer as specified in the function parameter.

```
418  # before changing the data type for age column
419  print(users_df.info())
420  # convert the age column from float into int
421  users_df['age'] = users_df['age'].astype('int')
422  # after changing the data type for age column
423  print(users_df.info())
```

Figure 1.53: Python Code

c. After pre-processing, the data type for Age column has been converted from 'Float' to 'Integer'. Figure 1.54 shows the data type info for the Users DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 164716 entries, 1 to 278854
Data columns (total 6 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   user_id   164716 non-null  float64
 1   location  164716 non-null  object
 2   age       164716 non-null  int32
 3   city      164716 non-null  object
 4   state     164716 non-null  object
 5   country   164716 non-null  object
dtypes: float64(1), int32(1), object(4)
```

Figure 1.54: Data Type Info for the Users DataFrame (After)

Pre-processing For Book-Ratings DataFrame:

1) Changing of the column names

   a. Prior to the pre-processing, most of the column names come with **capital letters that are used as the first letter of every word**, and some of the header columns are **using hyphens to connect two words into a single word.** Figure 1.55 shows the column names for Books-Ratings DataFrame.

book_ratings_df - DataFrame

Index	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0

Figure 1.55: Column names for Book-Ratings DataFrame (Before)

   b. In Python code,
      i. Line 443: using the lower () function to convert the respective column name into lower-case format.
      ii. Line 445: using the replace () function to replace the hyphens (-) with underscore (_).

```
440 # before changing the column names in the users dataframe
441 print(book_ratings_df.head())
442 # convert all the column name into lower cases
443 book_ratings_df.columns = book_ratings_df.columns.str.lower()
444 # convert all the column name with hyphens "-" into underscore "_" by using the replace () function
445 book_ratings_df.columns = book_ratings_df.columns.str.replace('-', '_')
446 # before changing the column names in the users dataframe
447 print(book_ratings_df.head())
```

Figure 1.56: Python Code

   c. After pre-processing, most of the column names are converted into a lower-case format and replace those words with hyphens with an underscore.

book_ratings_df - DataFrame

Index	user_id	isbn	book_rating
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0

Figure 1.57: Column names for Book-Ratings DataFrame (After)

2) Removing the Implicit Rating (i.e., Zero Rating) from the Book-Rating DataFrame
   a. Prior to pre-processing, significantly high number of ratings belongs to zero rating, which indicates an implicit. Figure 1.58 shows the distribution plot of the book ratings before removing zero ratings. Furthermore, the size of the book ratings is 1048575.

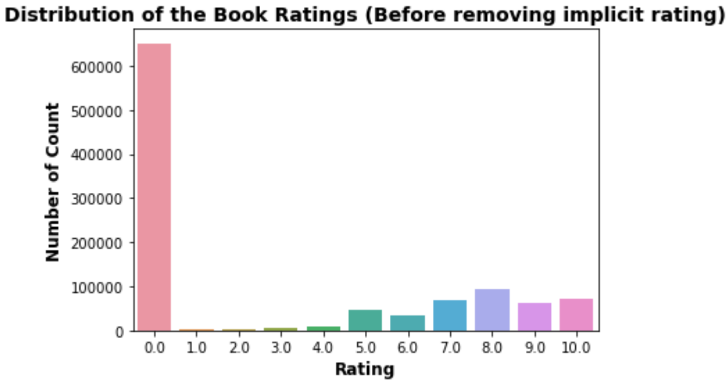


Figure 1.58: Distribution of the Book Ratings (Before)

   b. In Python code,
      i. Line 488: using Logical Boolean Expression to remove the implicit ratings from the Book-Ratings DataFrame.

```
484  # As zero indicates an implicit rating, therefore it will be removes from the data.
485  # As such, it will be focusing more on the explicit ratings instead.
486  print("Number of rows in book ratings before removing zero ratings: {}".format(len(book_ratings_df)))
487  # remove the implicit ratings from the dataframe
488  book_ratings_df = book_ratings_df[book_ratings_df['book_rating'] != 0]
489  print("Number of rows in book ratings after removing zero ratings: {}".format(len(book_ratings_df)))
```

Figure 1.59: Python Code

   c. After pre-processing, the size of the book ratings is 397248. Figure 1.60 shows the distribution plot of the book ratings after removal of the zero ratings from the book-ratings DataFrame.

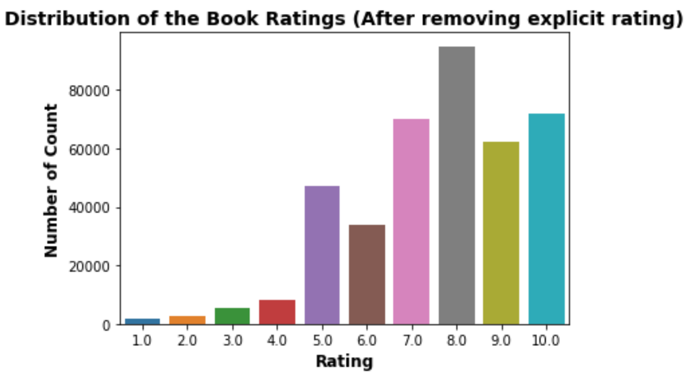


Figure 1.60: Distribution of the Book Ratings (After)

Pre-processing For Combining Users, Books, and Book-Ratings DataFrame:

1) Joining the Pre-processed Books DataFrame with the Pre-processed Book-Ratings DataFrame

   a. Prior to the pre-processing, the number of rows in the pre-processed Books DataFrame is 232330, whereas the number of rows in the pre-processed Book-Ratings is 397248.

```
Number of rows in the Preprocessed Book DataFrame: 232330
Number of rows in the Preprocessed Book Ratings DataFrame: 397248
```

Figure 1.61: Total number of rows exists in the pre-processed book DataFrame and Book-Ratings DataFrame

   b. In Python code,

      i. Line 515: using the join () function to join columns of the other DataFrame based on join key - 'isbn'.

```
509 ##### Joining the Preprocessed Books DataFrame with Preprocessed Book Ratings
510 # print the number of rows in the preprocessed book dataframe
511 print("Number of rows in the Preprocessed Book DataFrame: {}".format(len(book_df_preprocessed)))
512 # print the number of rows in the preprocessed book ratings dataframe
513 print("Number of rows in the Preprocessed Book Ratings DataFrame: {}".format(len(book_ratings_df_preprocessed)))
514 # join the preprocssed book table to the preprocessed book rating table based on the ISBN
515 bk_with_ratings_df = book_ratings_df_preprocessed.join(book_df_preprocessed.set_index('isbn'), on = 'isbn')
516 # print the number of rows in the combination of books and book ratings dataframe
517 print("Number of rows in the New DataFrame (Books + Books-Ratings): {}".format(len(bk_with_ratings_df)))
518 # display the new book with rating dataframe
519 print(bk_with_ratings_df.head())
```

Figure 1.62: Python Code

   c. After pre-processing, both of the pre-processed books DataFrame and pre-processed book-ratings DataFrame merged together to form a new DataFrame called "bk_with_ratings_df". The number of rows in the new DataFrame (books + book-ratings) is 397248.

```
Number of rows in the New DataFrame (Books + Books-Ratings): 397248
```

Figure 1.63: Total number of rows exists new DataFrame (bk_with_ratings_df)

bk_with_ratings_df - DataFrame

Index	user_id	isbn	ook_ratin	book_title	book_author	of_public	publisher
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle
3	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press
4	276729	521795028	6	The Amsterdam Connection : Level 4 (Cambridge English Readers)	Sue Leather	2001	Cambridge University Press
6	276736	3257224281	8	nan	nan	nan	nan
7	276737	600570967	6	nan	nan	nan	nan
8	276744	038550120X	7	A Painted House	JOHN GRISHAM	2001	Doubleday
9	276745	342310538	10	nan	nan	nan	nan
16	276747	60517794	9	Little Altars Everywhere	Rebecca Wells	2003	HarperTorch

Figure 1.64: New DataFrame (bk_with_ratings_df)

2) Remove rows that consists of NaN values in the **book title** column in the new DataFrame (bk_with_ratings_df)

   a. Prior to pre-processing, there are about 82025 rows with NaN values in the book title column in the New DataFrame (bk_with_ratings_df). Figure 1.65 shows the total number of NaN values in the Book Title column in the New DataFrame (bk_with_ratings_df).

```
Before removing all the NaN values, the number of Book Title with NaN values is 82025
```

Figure 1.65: Total number of NaN values in the Book Title column in the New DataFrame (bk_with_ratings_df) (Before)

   b. In Python code,

      i. Line 526: using dropna () function to remove the rows with NaN that subset with the book title column

```
523  # count the number of book title with NaN values before removing NaN
524  print("Before removing all the NaN values, the number of Book Title with NaN values is {}".format(bk_with_ratings_df['book_title']
525  # remove the rows consists of NaN that subset with the book title column
526  bk_with_ratings_df.dropna(subset = ['book_title'], inplace = True)
527  # count the number of book title with NaN values removed
528  print("After removing all the NaN values, the number of Book Title with NaN values is {}".format(bk_with_ratings_df['book_title'].
```

Figure 1.66: Python Code

   c. After pre-processing, there are about 0 rows with NaN values in the book title column in the New DataFrame (bk_with_ratings_df). Figure 1.67 shows the total number of NaN values in the Book Title column in the New DataFrame (bk_with_ratings_df).

```
After removing all the NaN values, the number of Book Title with NaN values is 0
```

Figure 1.67: Total number of NaN values in the Book Title column in the New DataFrame (bk_with_ratings_df) (After)

3) Joining the Pre-processed Users DataFrame with the New DataFrame (bk_with_ratings_df)

   a. Prior to the pre-processing, the number of rows in the pre-processed User DataFrame is 164716, whereas the number of rows in the new DataFrame (bk_with_ratings_df) is 315223.

```
Number of rows in the Preprocessed User DataFrame: 164716
Number of Rows in the New DataFrame (bk_with_ratings_df): 315223
```

Figure 1.68: Total number of rows exists in the pre-processed Users DataFrame and new DataFrame (bk_with_ratings_df)

   b. In Python code,
      i. Line 556: using the join () function to join columns of the other DataFrame based on join key - ‘user_id’.

```
549 ##### Joining the Preprocessed User DataFrame with New DataFrame (bk_with_ratings_df)
550 # print the number of rows in the preprocessed user dataframe
551 print("Number of rows in the Preprocessed User DataFrame: {}".format(len(users_df_preprocessed)))
552 # print the number of rows in the new table (books + book_ratings)
553 print("Number of Rows in the New DataFrame (bk_with_ratings_df): {}".format(len(bk_with_ratings_df)))
554 # print the number of rows in the new table (books + book_ratings + users)
555 bookUserRatings_df = bk_with_ratings_df.join(users_df_preprocessed.set_index('user_id'), on = 'user_id')
556 print("Number of Rows in the New DataFrame (User + bk_with_ratings_df): {}".format(len(bookUserRatings_df)))
```

Figure 1.69: Python Code

   c. After pre-processing, both of the pre-processed books DataFrame and pre-processed book-ratings DataFrame merged together to form a new DataFrame called “bookUserRatings_df”. The number of rows in the new DataFrame (bookUserRatings_df) is 315223.

```
Number of Rows in the New DataFrame (User + bk_with_ratings_df): 315223
```

Figure 1.70: Total number of rows exists new DataFrame (bookUserRatings_df)

bookUserRatings_df - DataFrame

Index	user_id	isbn	ook_ratin	book_title	book_author	of_public	publisher	location	age	city
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle	nan	nan	nan
3	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	rijeka, n/a, croatia	16	rijeka
4	276729	521795028	6	The Amsterdam Connection : Level 4 (Cambridge English Readers)	Sue Leather	2001	Cambridge University Press	rijeka, n/a, croatia	16	rijeka
8	276744	038550120X	7	A Painted House	JOHN GRISHAM	2001	Doubleday	nan	nan	nan
16	276747	60517794	9	Little Altars Everywhere	Rebecca Wells	2003	HarperTorch	iowa city, iowa, usa	25	iowa city
19	276747	671537458	9	Waiting to Exhale	Terry McMillan	1995	Pocket	iowa city, iowa, usa	25	iowa city
20	276747	679776818	8	Birdsong: A Novel of Love and War	Sebastian Faulks	1997	Vintage Books USA	iowa city, iowa, usa	25	iowa city
21	276747	943066433	7	How to Deal With Difficult People	Rick Brinkman	1995	Careertrack Inc.	iowa city, iowa, usa	25	iowa city
23	276747	1885408226	7	The Golden Rule of Schmoozing	Aye Jaye	1998	Listen & Live Audio	iowa city, iowa, usa	25	iowa city

Figure 1.71: New DataFrame (bookUserRatings_df)

4) Remove rows that consists of empty strings, zero values, and NaN values in the **location** column in the new DataFrame (bookUserRatings_df)

   a. Prior to pre-processing, there are about 101079 rows with NaN values in the location column in the New DataFrame (bookUserRatings_df). Figure 1.65 shows the total number of NaN values in the Location column in the New DataFrame (bookUserRatings_df).

```
Number of empty strings in the Location column: 0
Number of zero values in the Location column: 0
Number of null values in the Location column: 101079
```

Figure 1.72: Total number of empty strings, zero values, and NaN values in the Location column in the New DataFrame (bookUserRatings_df) (Before)

   b. In Python code,
      i. Line 574: using dropna () function to remove the rows with NaN that subset with the location column

```
562  # count the number of empty string, zero values, and null values in the location column before removing NaN
563  empty_values_location = bookUserRatings_df[bookUserRatings_df['location'] == ''][['location'].count()
564  zero_values_location = bookUserRatings_df[bookUserRatings_df['location'] == 0][['location'].count()
565  null_values_location = bookUserRatings_df['location'].isnull().sum()
566
567  # display the total number of empty string, zero values, and null values in the location column before removing NaN
568  print("Number of empty strings in the Location column: {}".format(empty_values_location))
569  print("Number of zero values in the Location column: {}".format(zero_values_location))
570  print("Number of null values in the Location column: {}".format(null_values_location))
571  print()
572
573  # remove the rows consists of NaN that subsets with country column
574  bookUserRatings_df = bookUserRatings_df.dropna(subset = ['location'])
575
576  # count the number of empty string, zero values, and null values in the location column after removing NaN
577  empty_values_location = bookUserRatings_df[bookUserRatings_df['location'] == ''][['location'].count()
578  zero_values_location = bookUserRatings_df[bookUserRatings_df['location'] == 0][['location'].count()
579  null_values_location = bookUserRatings_df['location'].isnull().sum()
580
581  # display the total number of empty string, zero values, and null values in the location column after removing NaN
582  print("Number of empty strings in the Location column: {}".format(empty_values_location))
583  print("Number of zero values in the Location column: {}".format(zero_values_location))
584  print("Number of null values in the Location column: {}".format(null_values_location))
585  print()
```

Figure 1.73: Python Code

   c. After pre-processing, there are about 0 rows with NaN values in the location column in the New DataFrame (bookUserRatings_df). Figure 1.67 shows the total number of NaN values in the Location column in the New DataFrame (bookUserRatings_df).

```
Number of empty strings in the Location column: 0
Number of zero values in the Location column: 0
Number of null values in the Location column: 0
```

Figure 1.74: Total number of empty strings, zero values, and NaN values in the Location column in the New DataFrame (bookUserRatings_df) (After)

Part (d)

[10 marks] Recommends a set of (five to ten) items for a user in the dataset. Your program should allow the user to specify which user to generate recommendations for. The recommendations should not be simply random, but based on established recommender system principles, e.g., considering similar users, or similar products.

This means that if the dataset is unchanged, the list of recommended items for the same user should always be the same. Any variations in the list of items recommended should be explained.

Report requirements

- Describe the steps your program uses to generate the recommendations, use the same approach as part b) above to link your explanation to lines of code.

Deliverable: bug-free, well-structured and well-documented (with code comments) Python code.

NOTE: The python file for recommendation system app is *recommender_sys_app.py*

In this project, the book recommendation system basically uses a simple recommender system with matrix factorization using Keras such that both user and book learnt embedding are non-negative values in this case, and it is still workable with negative values to generate the book recommendations. Basically, to recommend the top *n* items to a user is just by taking the embedding vector of the user and do a dot product with all the embedding vectors of the books and then get the top *n* largest values. In this recommendation system app, it will prompt the user for two inputs: preference user ID and the *n* books to be recommended by the system.

Define Functions	
Lines 7-10	Define function for user embedding learnt.
Lines 12-15	Define function for movie embedding learnt.
Lines 17-21	Define function to return the top *n* relevant books ids.
Read the Data File and Load the Keras Model	
Lines 26-27	To read the pre-processed data from the CSV file and store it into a new variable called “preprocessed_df”.
Lines 29-30	Load the Keras Model and store it into a new variable called “model”.

Generate the Book Recommendations	
Lines 33-47	Display the recommender system menu interface.
Lines 49-50	To initialize the variable for while-loop.
Line 53	Begin of the while-loop process by checking the variable initialized in Lines 49-50.
Lines 54-55	Prompt the user to enter his/ her preference user ID and stored the user ID into a new variable called "userID" as string data type.
Lines 57-60	To check whether the input user ID is an integer or not. If it is true, proceed to the next lines. If it is false, it will re-prompt the user to enter his/ her preference user ID and stored the user ID into the same variable called "userID". The whole cycle iterated until the user has entered the user ID as integer.
Lines 62-64	To check whether the input user ID is within the range of 0 to 34116. This is because the "preprocessed_df" consists of 34117 unique users. If it is true, proceed to the next lines. If it is false, it will re-prompt the user to enter his/her preference user ID within that range and stored the user ID into the same variable called "userID". The whole cycle is iterated until the user has entered the user ID within the range of 0 to 34116.
Lines 67-68	Convert the input userID from string data type into integer data type.
Lines 70-71	Prompt the user to enter the number of books to be recommended by the system and stored the number of books into a new variable called "num_of_books" as integer data type.
Lines 73-74	Call the function to generate the embedding matrix for user feature.
Lines 76-77	Call the function to generate the embedding matrix for book feature.
Lines 79-81	Call the function to perform the dot product on user and book embedding matrices and then return the top *n* most relevant books ids.

Lines 83-95	Get the book information (i.e., book id, book title, book author, year of publication, and publisher) from the "preprocessed_df" DataFrame based on the returned top *n* most relevant books ids and store the book information into a new DataFrame called "recommended_books_df".
Lines 97-102	Display the top *n* most relevant books information to the user interface.
Lines 104-105	Prompt the user whether would like to continue to use the recommendation system. If it is true, the whole cycle is iterated again from line 53 to line 105. If it is false, proceed to the next line.
Lines 107-108	Display the goodbye message to the user.

General guidelines to use the Book Recommender System App:

1. The system will show the menu interface and prompt the user to enter his/her preference user ID such that the pre-processing datafile only consists of the user ID instead of the user's name.

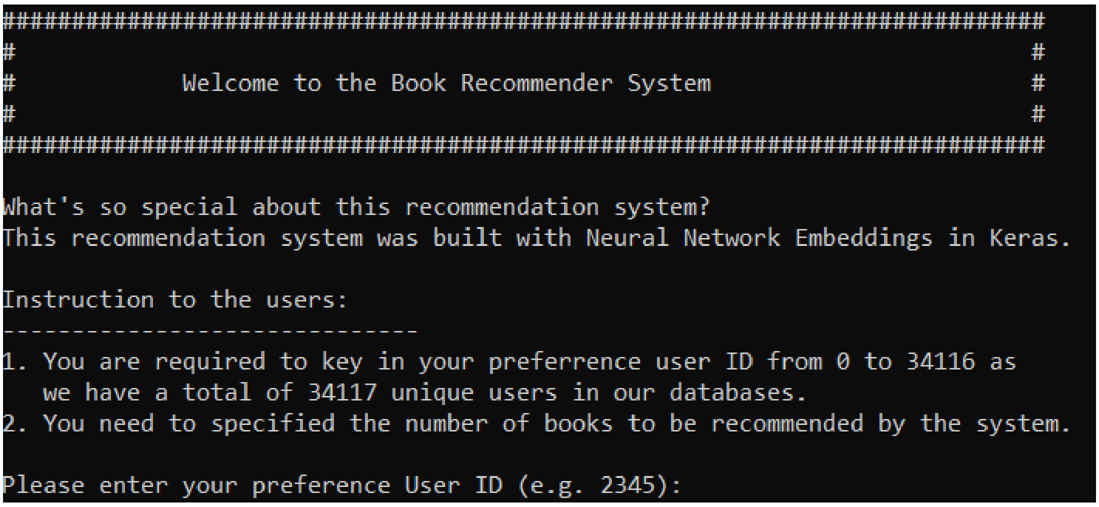


Figure 1.75

2. The user keyed in his/her preference user ID and hit enter.

```
##########################################################################
#                                                                        #
#            Welcome to the Book Recommender System                      #
#                                                                        #
##########################################################################

What's so special about this recommendation system?
This recommendation system was built with Neural Network Embeddings in Keras.

Instruction to the users:
------------------------------
1. You are required to key in your preferrence user ID from 0 to 34116 as
   we have a total of 34117 unique users in our databases.
2. You need to specified the number of books to be recommended by the system.

Please enter your preference User ID (e.g. 2345): 1234
```

Figure 1.76

However, if the user has entered his/her preference user ID as alphabet/ alphanumeric, symbols, etc. The system will then re-prompt the user to entered his/her preference user ID.

```
##########################################################################
#                                                                        #
#            Welcome to the Book Recommender System                      #
#                                                                        #
##########################################################################

What's so special about this recommendation system?
This recommendation system was built with Neural Network Embeddings in Keras.

Instruction to the users:
------------------------------
1. You are required to key in your preferrence user ID from 0 to 34116 as
   we have a total of 34117 unique users in our databases.
2. You need to specified the number of books to be recommended by the system.

Please enter your preference User ID (e.g. 2345): !@#$123abc

Please re-enter your preferrence User ID(e.g. 2345):
```

Figure 1.77

On the other hand, if the user has entered his/her preference user ID that is not within the range of 0 to 34116 (as we have a total of 34117 unique users in the database), the system will re-prompt the user to entered his/her preference user ID within that range.

```
##########################################################################
#                                                                        #
#           Welcome to the Book Recommender System                       #
#                                                                        #
##########################################################################

What's so special about this recommendation system?
This recommendation system was built with Neural Network Embeddings in Keras.

Instruction to the users:
------------------------------
1. You are required to key in your preferrence user ID from 0 to 34116 as
   we have a total of 34117 unique users in our databases.
2. You need to specified the number of books to be recommended by the system.

Please enter your preference User ID (e.g. 2345): !@#$123abc

Please re-enter your preferrence User ID(e.g. 2345): 345678


Please re-enter your preferrence User ID in the range of 0 to 34116:
```

Figure 1.78

3. Given that the user has entered his/her correct preference user ID. The system will then prompt the user to enter the number of books to be recommended by the system.

```
##########################################################################
#                                                                        #
#           Welcome to the Book Recommender System                       #
#                                                                        #
##########################################################################

What's so special about this recommendation system?
This recommendation system was built with Neural Network Embeddings in Keras.

Instruction to the users:
------------------------------
1. You are required to key in your preferrence user ID from 0 to 34116 as
   we have a total of 34117 unique users in our databases.
2. You need to specified the number of books to be recommended by the system.

Please enter your preference User ID (e.g. 2345): 1234

Please enter the number of books to be recommended by the system (e.g. 5):
```

Figure 1.79

4. The user then keyed in the number of books to be recommended by the system and hit enter.

```
################################################################################
#                                                                              #
#             Welcome to the Book Recommender System                           #
#                                                                              #
################################################################################

What's so special about this recommendation system?
This recommendation system was built with Neural Network Embeddings in Keras.

Instruction to the users:
------------------------------
1. You are required to key in your preferrence user ID from 0 to 34116 as
   we have a total of 34117 unique users in our databases.
2. You need to specified the number of books to be recommended by the system.

Please enter your preference User ID (e.g. 2345): 1234

Please enter the number of books to be recommended by the system (e.g. 5): 10
```

Figure 1.80

5. The system will then display the top *n* relevant recommended books based on the User ID specified by the user. Next, the system will then prompt the user whether he/she would like to continue to use the recommendation system.

```
Please enter the number of books to be recommended by the system (e.g. 5): 10

Based on your specified User ID '1234', here are the top 10 relevant recommended books:
---------------------------------------------------------------------------------------

   book_id                                         book_title      book_author  \
0    15095  The Hobbit : The Enchanting Prelude to The Lor...  J.R.R. TOLKIEN
1    22210                                  The Da Vinci Code        Dan Brown
2     7140      Pride and Prejudice (Penguin Popular Classics)     Jane Austen
3    29161                              To Kill a Mockingbird      Harper Lee
4    39136                 Anne Frank: The Diary of a Young Girl     ANNE FRANK
5    26645        Harry Potter and the Goblet of Fire (Book 4)   J. K. Rowling
6    41109      Harry Potter and the Sorcerer's Stone (Book 1)   J. K. Rowling
7    26642  Harry Potter and the Prisoner of Azkaban (Book 3)   J. K. Rowling
8    26641  Harry Potter and the Prisoner of Azkaban (Book 3)   J. K. Rowling
9    92565  Harry Potter and the Sorcerer's Stone (Harry P...   J. K. Rowling

   year_of_publication               publisher
0               1986.0                 Del Rey
1               2003.0               Doubleday
2               1994.0       Penguin Books Ltd
3               1988.0  Little Brown & Company
4               1993.0                  Bantam
5               2000.0              Scholastic
6               1998.0              Scholastic
7               2001.0              Scholastic
8               1999.0              Scholastic
9               1999.0  Arthur A. Levine Books

Would you like to continue to use the system? <Y/N>:
```

Figure 1.81

6. If the user keyed in **YES** as “Y” or “y”, the whole cycle is then iterated from Step 1 to 6.

```
Would you like to continue to use the system? <Y/N>: y

Please enter your preference User ID (e.g. 2345): 2345

Please enter the number of books to be recommended by the system (e.g. 5): 5

Based on your specified User ID '2345', here are the top 5 relevant recommended books:
----------------------------------------------------------------------------------------------

   book_id                                       book_title     book_author  \
0    26642  Harry Potter and the Prisoner of Azkaban (Book 3)   J. K. Rowling
1    41109     Harry Potter and the Sorcerer's Stone (Book 1)   J. K. Rowling
2    92565  Harry Potter and the Sorcerer's Stone (Harry P...   J. K. Rowling
3    15095  The Hobbit : The Enchanting Prelude to The Lor...  J.R.R. TOLKIEN
4    26645       Harry Potter and the Goblet of Fire (Book 4)   J. K. Rowling

   year_of_publication              publisher
0               2001.0             Scholastic
1               1998.0             Scholastic
2               1999.0  Arthur A. Levine Books
3               1986.0                Del Rey
4               2000.0             Scholastic

Would you like to continue to use the system? <Y/N>:
```

Figure 1.82

7. If the user keyed in **NO** as “N” or “n”, the system will stop running and displayed the ending message to the user.

```
Would you like to continue to use the system? <Y/N>: y

Please enter your preference User ID (e.g. 2345): 2345

Please enter the number of books to be recommended by the system (e.g. 5): 5

Based on your specified User ID '2345', here are the top 5 relevant recommended books:
----------------------------------------------------------------------------------------------

   book_id                                       book_title     book_author  \
0    26642  Harry Potter and the Prisoner of Azkaban (Book 3)   J. K. Rowling
1    41109     Harry Potter and the Sorcerer's Stone (Book 1)   J. K. Rowling
2    92565  Harry Potter and the Sorcerer's Stone (Harry P...   J. K. Rowling
3    15095  The Hobbit : The Enchanting Prelude to The Lor...  J.R.R. TOLKIEN
4    26645       Harry Potter and the Goblet of Fire (Book 4)   J. K. Rowling

   year_of_publication              publisher
0               2001.0             Scholastic
1               1998.0             Scholastic
2               1999.0  Arthur A. Levine Books
3               1986.0                Del Rey
4               2000.0             Scholastic

Would you like to continue to use the system? <Y/N>: n

Thank you for using the book recommender system! Goodbye!
```

Figure 1.83

Part (e)

[10 marks] Evaluates/ measures the quality of the recommender system using the available data and splitting it into training and testing sets.

Report requirements

- Clear analysis of the effectiveness of your program to generate good recommendations, using a suitable metric (i.e., method of measurement)
- Describe the conclusions you can draw from the above analysis

Deliverable: bug-free, well-structured and well-documented (with code comments) Python code.

In this project, **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)** are used to estimate the predicting performance of the respective neural-network based models. MAE is used to measure the absolute error between the predicted value and the true value, whereas RMSE is used to evaluate the deviation between the predicted value and the true value. They are the commonly used performance metrics in the field of the recommendation system.

Below is the evaluation analysis for the respective neural network-based models.

Neural Network A:

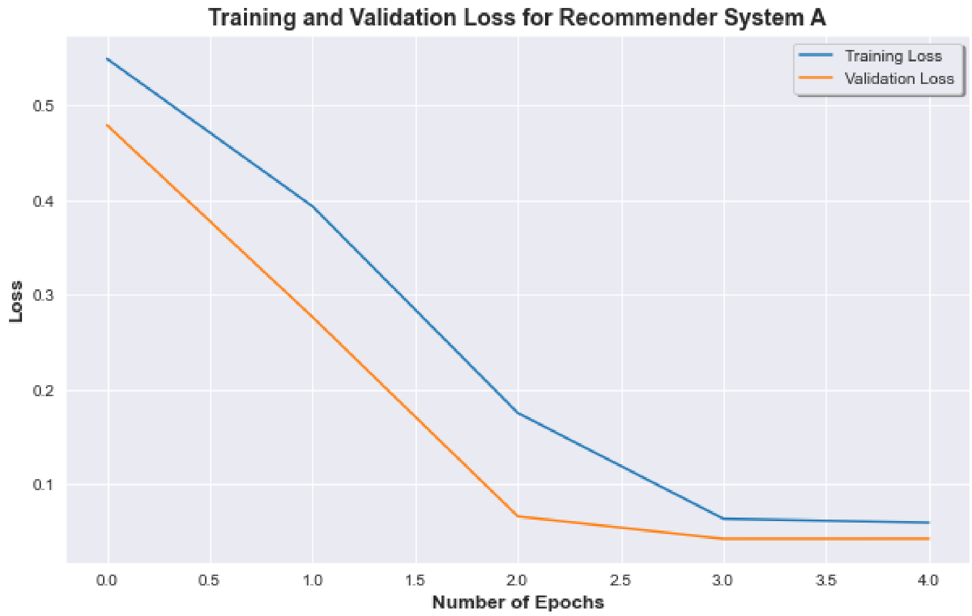


Figure 1.84: Training and Validation Loss for Recommender System A

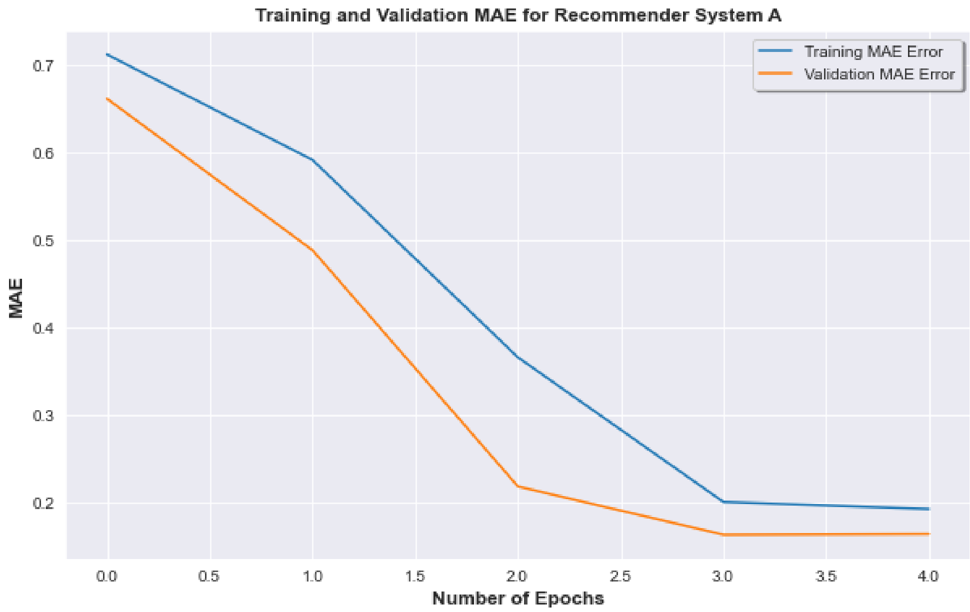


Figure 1.85: Training and Validation MAE for Recommender System A

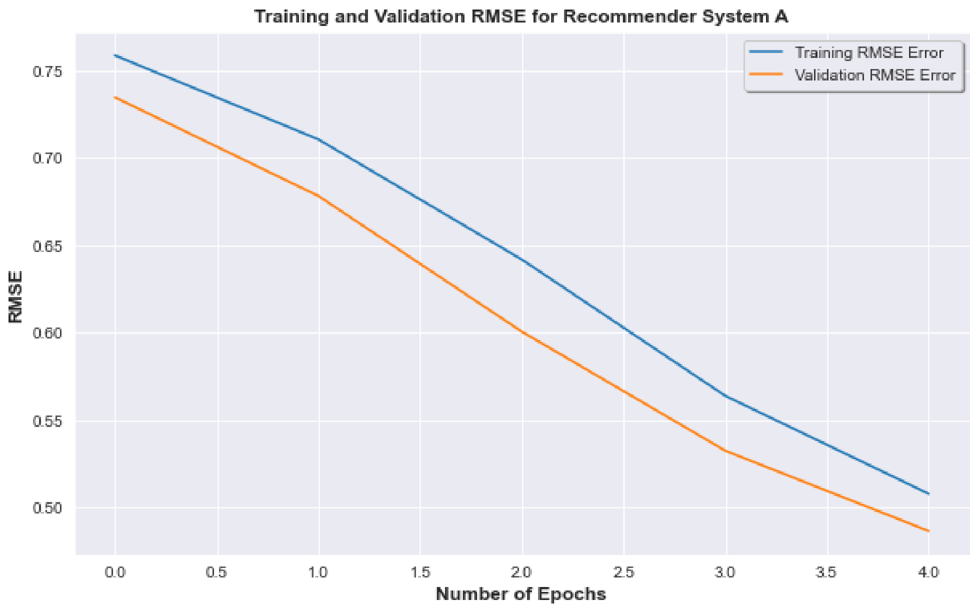


Figure 1.86: Training and Validation RMSE for Recommender System A

Comments:

This neural network (Model A) can be considered moderately good as the model consist some form of complexity in the neural network. From the plots shown above, both training and validation losses decreases to a point of stability. It can be also observed that there is a notable gap in between the training and validation sets, but it can assure that this model does not overfit. Overfitting means that the plot of the validation loss decreases to a point and beings increasing again, whereas the plot of training loss continues to decrease with experience, which does not happen in both cases as shown in the diagram above. The notable gap in between the training loss and validation loss is known as the “generalization gap”, that is, the expected gap in the performance between the training and validation sets. Furthermore, the above explanation is also applied to the RMSE and MAE learning curves.

However, continued training of an optimal fit will likely lead to an overfitting. As such, the early stopping has been included while training the neural network A to avoid the chances of having overfitting. From the plots show above, the training phase for neural network A was halted after the five epochs.

Here are the results for the loss, mean absolute error (MAE), and root mean squared error (RMSE) of neural network A based on the training and validation sets:

```
Epoch 5/30
235/235 [==============================] - 7s 29ms/step - loss: 0.0594 - mae: 0.1931 - rmse: 0.5076 - val_loss: 0.0424 - val_mae: 0.1645 - val_rmse: 0.4863
```

Neural Network A		
	Training Set	**Validation Set**
Loss	0.0594	0.0424
MAE	0.1931	0.1645
RMSE	0.5076	0.4863

The performance of the neural network A was also evaluated using the testing set. This is to ensure that it can generalised well to new and unseen data points and also to make a comparison with the training set to ensure that it is not overfitting and underfitting.

```
126/126 [==============================] - 2s 12ms/step - loss: 0.0424 - mae: 0.1647 - rmse: 0.4741
```

Neural Network A	
	Testing Set
Loss	0.0424
MAE	0.1647
RMSE	0.4741

Since both training and testing sets shows no much variation, but it can be concluded that there is no overfitting and underfitting phenomenon exists in this neural network A.

Neural Network B:

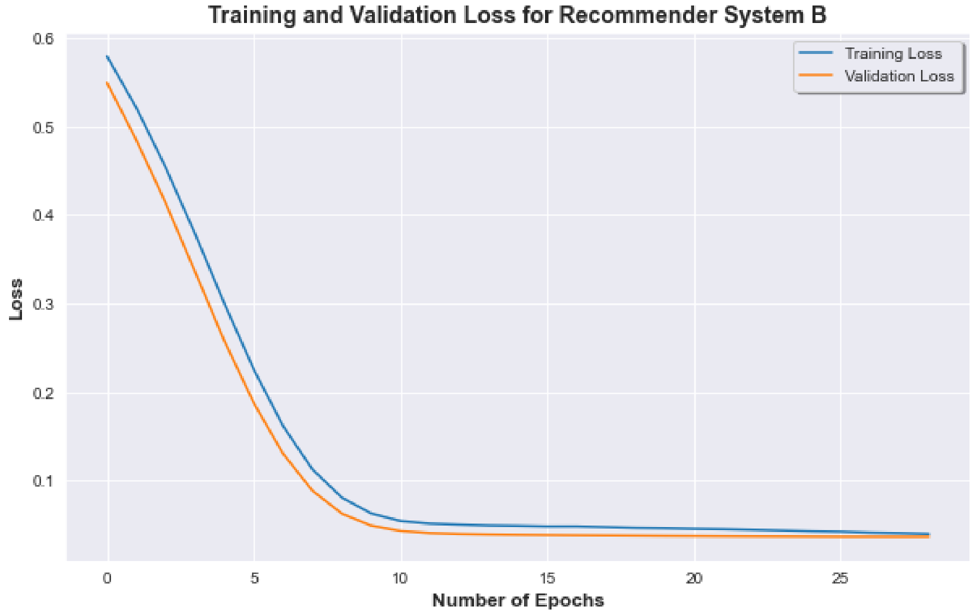


Figure 1.87: Training and Validation Loss for Recommender System B

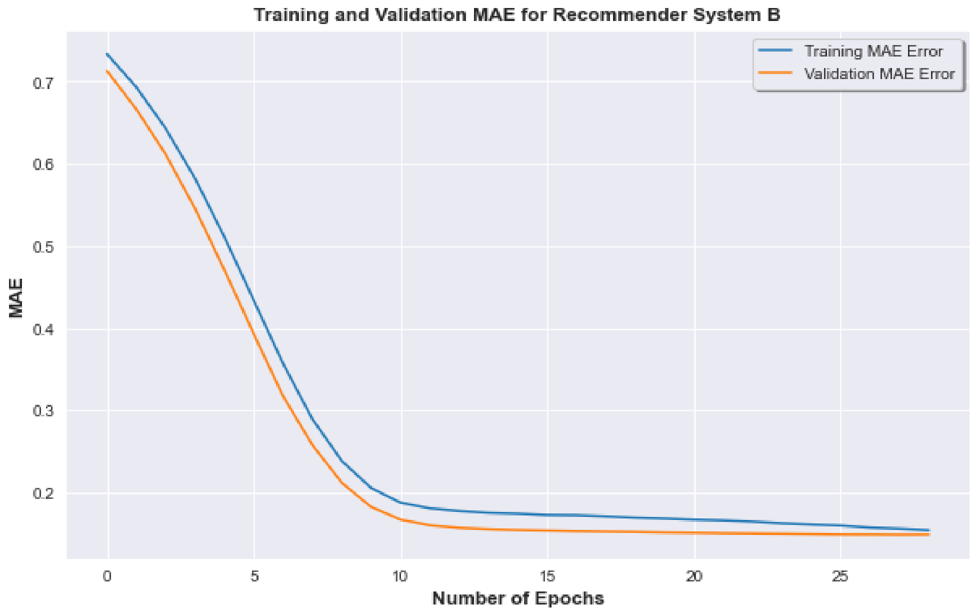


Figure 1.88: Training and Validation MAE for Recommender System B

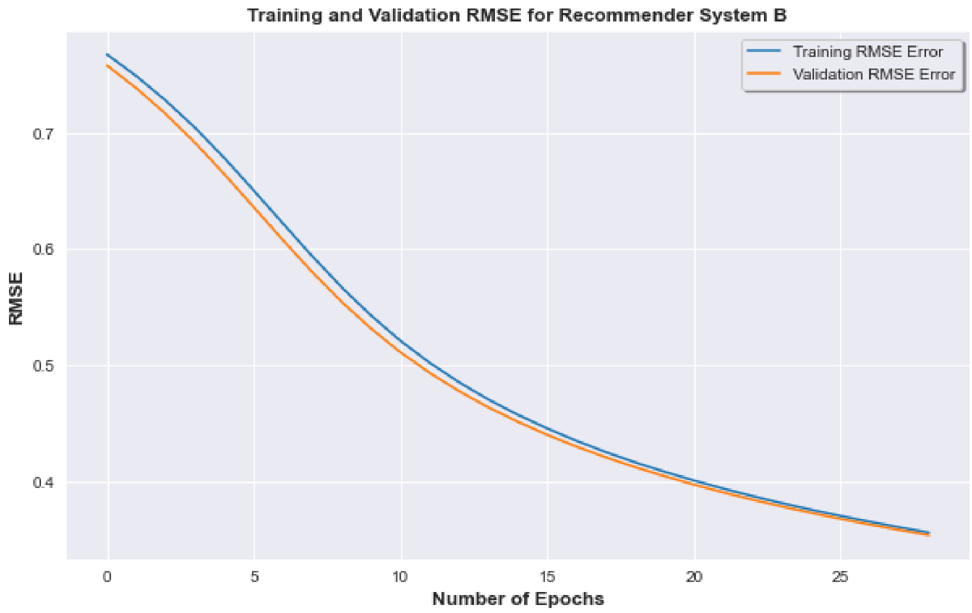


Figure 1.89: Training and Validation RMSE for Recommender System B

Comments:

This neural network (Model B) can be considered as the optimal model. From the plots shown above, both training and validation losses decreases to a point of stability. It can be also observed that there is a notable gap in between the training and validation sets, but it can assure that this model does not overfit. Overfitting means that the plot of the validation loss decreases to a point and beings increasing again, whereas the plot of training loss continues to decrease with experience, which does not happen in both cases as shown in the diagram above. The notable gap in between the training loss and validation loss is known as the “generalization gap”, that is, the expected gap in the performance between the training and validation sets. Furthermore, the above explanation is also applied to the RMSE and MAE learning curves.

However, continued training of an optimal fit will likely lead to an overfitting. As such, the early stopping has been included while training the neural network B to avoid the chances of having overfitting. From the plots show above, the training phase for neural network B was halted after the 29 epochs.

Here are the results for the loss, mean absolute error (MAE), and root mean squared error (RMSE) of neural network B based on the training and validation sets:

```
Epoch 29/30
235/235 [==============================] - 6s 27ms/step - loss: 0.0393 - mae: 0.1543 - rmse: 0.3556 - val_loss: 0.0362 - val_mae: 0.1490 - val_rmse: 0.3534
```

Neural Network B		
	Training Set	**Validation Set**
Loss	0.0393	0.0362
MAE	0.1543	0.1490
RMSE	0.3556	0.3534

The performance of the neural network B was also evaluated using the testing set. This is to ensure that it can generalised well to new and unseen data points and also to make a comparison with the training set to ensure that it is not overfitting and underfitting.

```
126/126 [==============================] - 1s 10ms/step - loss: 0.0364 - mae: 0.1500 - rmse: 0.3521
```

Neural Network B	
	Testing Set
Loss	0.0364
MAE	0.1500
RMSE	0.3521

Since both training and testing sets shows no much difference, therefore it can be concluded that there is no overfitting and underfitting phenomenon exists in this neural network B.

Neural Network C:

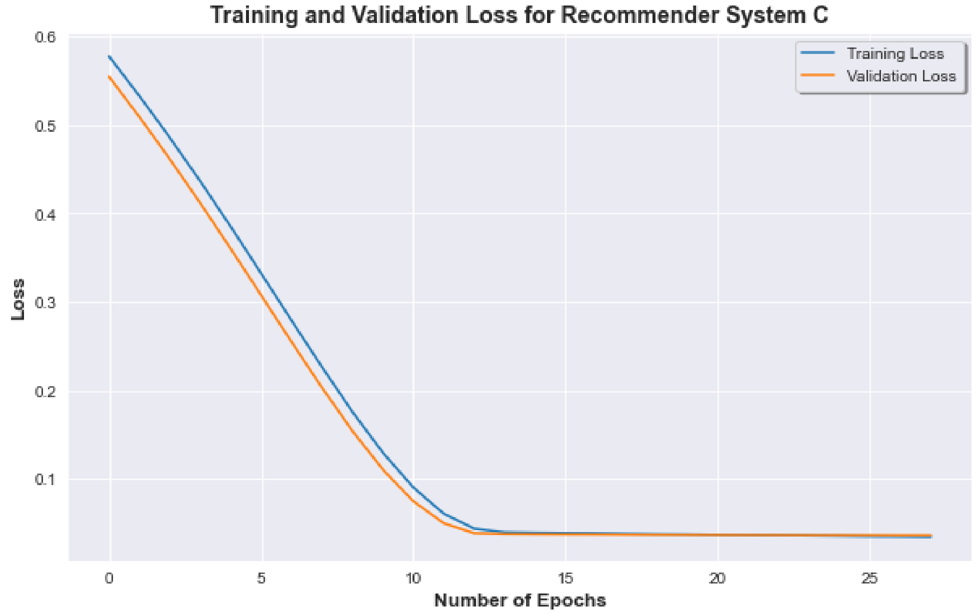


Figure 1.90: Training and Validation Loss for Recommender System C

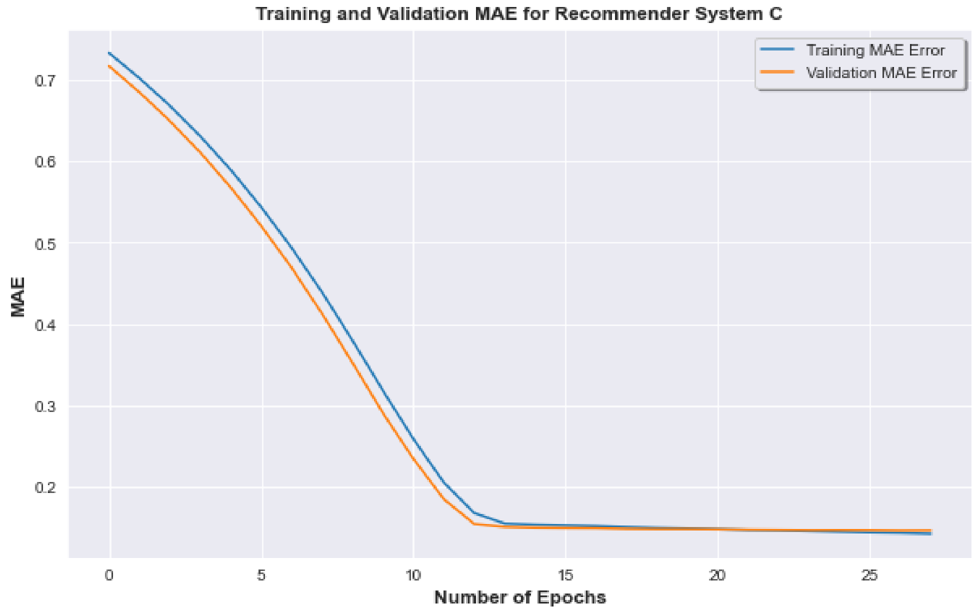


Figure 1.91: Training and Validation MAE for Recommender System C

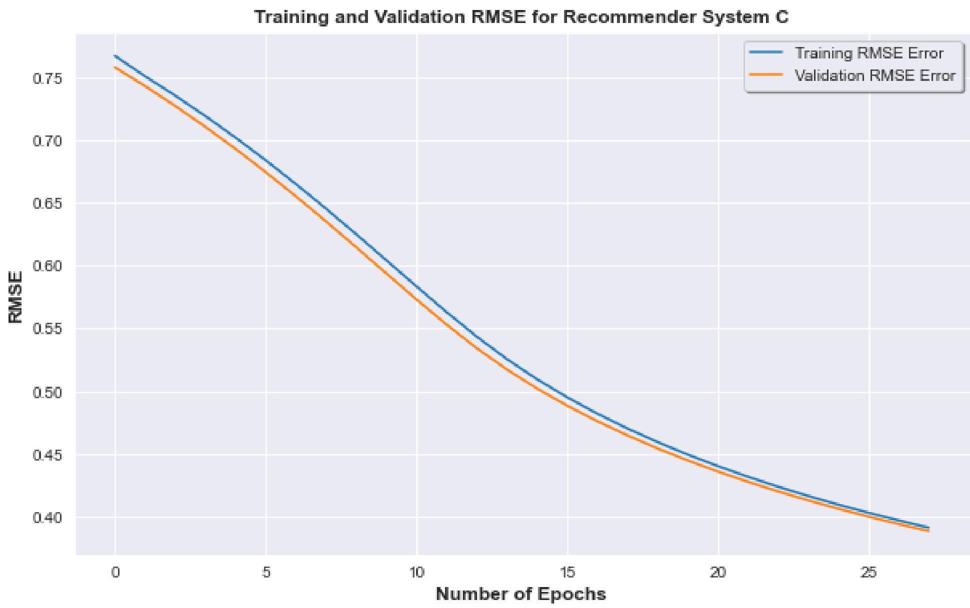


Figure 1.92: Training and Validation RMSE for Recommender System C

Comments:

This neural network (Model C) can also be considered as the optimal model. From the plots shown above, both training and validation losses decreases to a point of stability. It can be also observed that there is a notable gap in between the training and validation sets, but it can assure that this model does not overfit. Overfitting means that the plot of the validation loss decreases to a point and beings increasing again, whereas the plot of training loss continues to decrease with experience, which does not happen in both cases as shown in the diagram above. The notable gap in between the training loss and validation loss is known as the “generalization gap”, that is, the expected gap in the performance between the training and validation sets. Furthermore, the above explanation is also applied to the RMSE and MAE learning curves.

However, continued training of an optimal fit will likely lead to an overfitting. As such, the early stopping has been included during training phase to avoid the overfitting and thus, the model has stopped the training phase after 28 epochs with reference to the validation set.

Here are the results for the loss, mean absolute error (MAE), and root mean squared error (RMSE) of neural network C based on the training and validation sets:

```
Epoch 28/30
235/235 [==============================] - 8s 36ms/step - loss: 0.0348 - mae: 0.1429 - rmse: 0.3913 - val_loss: 0.0363 - val_mae: 0.1470 - val_rmse: 0.3886
```

Neural Network C		
	Training Set	Validation Set
Loss	0.0348	0.0363
MAE	0.1429	0.1470
RMSE	0.3913	0.3886

The performance of the neural network C was also evaluated using the testing set. This is to ensure that it can generalised well to new and unseen data points and also to make a comparison with the training set to ensure that it is not overfitting and underfitting.

```
126/126 [==============================] - 1s 10ms/step - loss: 0.0364 - mae: 0.1478 - rmse: 0.3869
```

Neural Network C	
	Testing Set
Loss	0.0364
MAE	0.1478
RMSE	0.3869

Since both training and testing sets shows no much variation, therefore it can be concluded that there is no overfitting and underfitting phenomenon exists in this neural network C.

NOTE: All the results above may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision.

Conclusion:

Out of the three above models, only the **neural network B will be chosen** as the optimal model because the training, validation, and testing loss, MAE and RMSE are generally better than the other neural network models. Furthermore, neural network B was trained with Adam optimizer which combines the best properties of the AdaGrad and RMSprop algorithm to provide an optimization algorithm that can handle spasm gradients on noisy problems, less time and more efficiently as compared to other optimizers.

Additional Information for Part (e):

Since the neural network B has been chosen as the optimal model for the recommender system, the number of latent factors can be tuned using the simple grid-search process to find the best latent factor based on the MAE and RMSE scores from the testing set. Latent factors are the features in the lower dimension latent space projected from user-item interaction matrix. The idea behind of the matrix factorization is to use the latent factor to represent user preferences or book title in a much lower dimension space.

Similar to the principles component analysis (PCA), the number of latent factors determines the amount of important information that want to store in a lower dimension space. In other words, it shows how effectively represent the characteristics of users and items. As such, the matrix factorization with 1 latent factor represent to a most popular recommender. Increasing the number of latent factors would improve personalization, until the number become too high, at which point the model starts to overfit. Therefore, the early stopping has been included during the grid-search process.

Therefore, the number of latent factors to be tuned was set to {5, 10, 20, 30, 40, 50}.

Here is the outcome of the grid-search results for neural network B:

```
The best RMSE score is 0.356

The best MAE score is 0.150

The best latent factor is {'latent_factor': 10}
```

Figure 1.93: Grid-Search Result of Latent Factor (Neural Network B)

The neural network B has been re-trained using the best latent factor and re-evaluated as shown below:

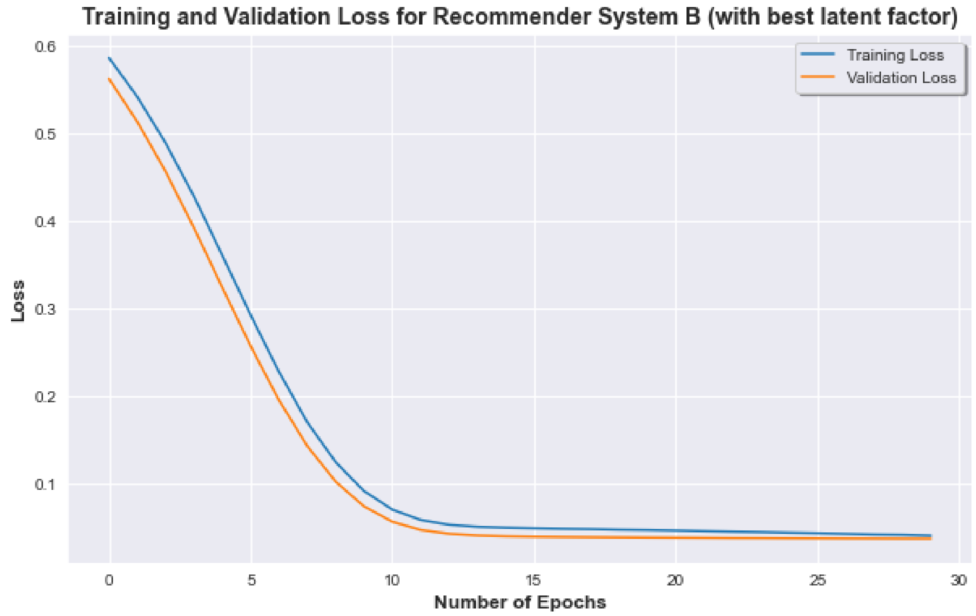


Figure 1.94: Training and Validation Loss for Recommender System B (with best latent factor)

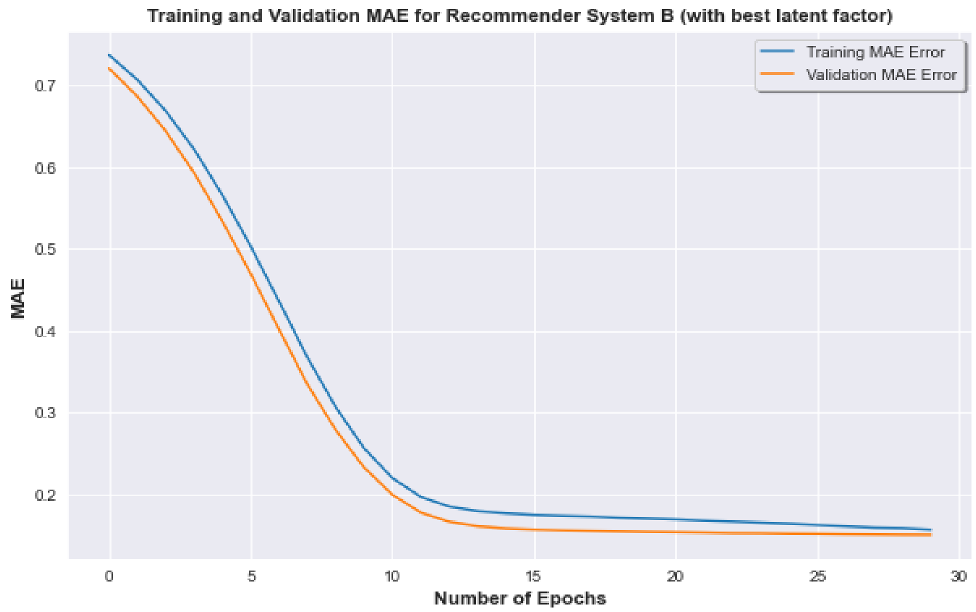


Figure 1.95: Training and Validation MAE for Recommender System B (with best latent factor)

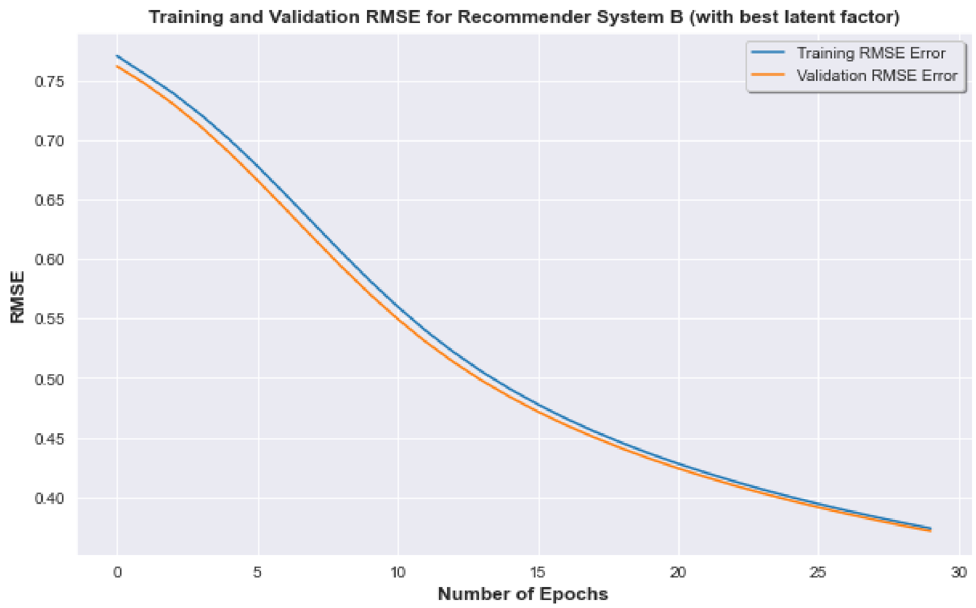


Figure 1.96: Training and Validation RMSE for Recommender System B (with best latent factor)

```
Epoch 30/30
235/235 [==============================] - 7s 30ms/step - loss: 0.0402 - mae: 0.1562 - rmse: 0.3735 - val_loss: 0.0367 - val_mae: 0.1499 - val_rmse: 0.3712
```

Neural Network B (with Best Latent Factor)		
	Training Set	**Validation Set**
Loss	0.0402	0.0367
MAE	0.1526	0.1499
RMSE	0.3735	0.3712

The performance of the neural network B with best latent factor was also evaluated using the testing set. As mentioned earlier before, this is to ensure that it can generalised well to new and unseen data points and also to make a comparison with the training set to ensure that it is not overfitting and underfitting.

```
126/126 [==============================] - 1s 5ms/step - loss: 0.0368 - mae: 0.1507 - rmse: 0.3697
```

Neural Network B (with Best Latent Factor)	
	Testing Set
Loss	0.0368
MAE	0.1507
RMSE	0.3697

All the above results show no sign of overfitting or underfitting and metrics evaluated are generally good. As such, this neural network B with the best latent factor can be used to recommend books to the users.

Reference Links

[1] *Institut For Informatik Freiburg*. Retrieved from: http://www2.informatik.uni-freiburg.de/~cziegler/BX/

[2] Mathew, P. Kuriakose, B. Hegde, V. 2016. *Book Recommendation System through content based and collaborative filtering method*. Retrieved from: https://ieeexplore.ieee.org/document/7684166

[3] Wang, J. Liu, L. 5 Nov 2020. *A multi-attention deep neural network model base on embedding and matrix factorization for recommendation.* Retrieved from: https://www.sciencedirect.com/science/article/pii/S2666307420300097

[4] Batra, N. 18 Dec 2017. *Recommender Systems in Keras*. Retrieved from: https://nipunbatra.github.io/blog/ml/2017/12/18/recommend-keras.html

[5] Lian, K. 17 Nov 2018. *Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization Collaborative Filtering.* Retrieved from: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1

[6] tungnd. *BUILD A SIMPL RECOMMENDER SYSTEM WITH MATRIX FACTORIZATION.* Retrieved from: https://petamind.com/build-a-simple-recommender-system-with-matrix-factorization/