

Python-Machine-Learning-KNN-Recommendation-Engine Author: Fran Panteli 1

Contents

1.1 Notebook Contents

1 Contents

- 1.1 Notebook Contents
- 1.2 Instructions
- 1.3 Load Libraries

2. Prepare the Data

- 2.1 Import Data
- 2.2 Remove Null Values From the Data Frames
- 2.3 Remove Users With Less Than 200 Ratings
- 2.4 Remove Books With Less Than 100 Ratings
- 2.5 Prepare the Dataset for KNN

3. Train the Model Using KNN

- 3.1 Train the Model Using KNN
- 3.2 Create the `get_recommends()` Function

4. Using the `get_recommends()` Function to Make Predictions

1.2 Instructions

In this challenge, you will create a book recommendation algorithm using **K-Nearest Neighbors**.

You will use the [Book-Crossings dataset](#). This dataset contains 1.1 million ratings (scale of 1-10) of 270,000 books by 90,000 users.

After importing and cleaning the data, use `NearestNeighbors` from `sklearn.neighbors` to develop a model that shows books that are similar to a given book. The Nearest Neighbors algorithm measures distance to determine the “closeness” of instances.

Create a function named `get_recommends` that takes a book title (from the dataset) as an argument and returns a list of 5 similar books with their distances from the book argument.

This code:

```
get_recommends("The Queen of the Damned (Vampire Chronicles  
(Paperback) ") )
```

should return:

```
[  
    'The Queen of the Damned (Vampire Chronicles (Paperback))',
```

```
[
  ['Catch 22', 0.793983519077301],
  ['The Witching Hour (Lives of the Mayfair Witches)',
0.7448656558990479],
  ['Interview with the Vampire', 0.7345068454742432],
  ['The Tale of the Body Thief (Vampire Chronicles (Paperback))',
0.5376338362693787],
  ['The Vampire Lestat (Vampire Chronicles, Book II)',
0.5178412199020386]
]
```

Notice that the data returned from `get_recommends()` is a list. The first element in the list is the book title passed in to the function. The second element in the list is a list of five more lists. Each of the five lists contains a recommended book and the distance from the recommended book to the book passed in to the function.

If you graph the dataset (optional), you will notice that most books are not rated frequently. To ensure statistical significance, remove from the dataset users with less than 200 ratings and books with less than 100 ratings.

The first three cells import libraries you may need and the data to use. The final cell is for testing. Write all your code in between those cells.

1.3 Load Libraries

```
# import libraries and modules (you may add additional imports but you
may not have to) import numpy as np import pandas as pd
from scipy.sparse import csr_matrix
# -> We are using the K-Means algorithm to perform clustering - this
is done with the sklearn module
from sklearn.neighbors import NearestNeighbors
# -> To plot the results import
matplotlib.pyplot as plt
```

2. Prepare the Data

2.1 Import Data

```
# get data files
"""
    -> This cell is sending an HTTP request to freeCodeCamp to use the
    book reviews which we
        are going to use to train the model with
    -> We don't have the entire dataset in the project files -> we are
    importing it into the notebook
    -> This only works in the Google Colaboratory environment -> if you
    download a local copy of the
        notebook and try importing the reviews into there it won't work
"""

!wget https://cdn.freecodecamp.org/project-
data/books/bookcrossings.zip

!unzip book-crossings.zip

"""
    -> We have imported in the book ratings in a zip form -> and then
    unzipped the files
    -> So we now have an unclean imported unzipped dataset of book reviews
    -> We need to clean the data before we can train the model on it """

--2024-02-13 14:50:07--
https://cdn.freecodecamp.org/project-data/books/book-crossings.zip
Resolving cdn.freecodecamp.org (cdn.freecodecamp.org)... 104.26.2.33,
172.67.70.149, 104.26.3.33, ...
Connecting to cdn.freecodecamp.org (cdn.freecodecamp.org)|
104.26.2.33|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26085508 (25M) [application/zip]
Saving to: 'book-crossings.zip'
book-crossings.zip 100%[=====>] 24.88M 96.6MB/s in
0.3
2024-02-13 14:50:08 (96.6 MB/s) - 'book-crossings.zip' saved
[26085508/26085508]

Archive:      book-crossings.zip
inflating:    BX-Book-Ratings.csv
inflating:    BX-Books.csv
inflating:    BX-Users.csv
```

```

{"type": "string"}

books_filename    = 'BX-Books.csv'
ratings_filename = 'BX-Book-Ratings.csv'

"""
    Now we are processing those book reviews:
    -> We have one variable which stores the names of the books
    -> And then we have another variable which stores the reviews
    themselves
    -> They are CSV files (which are like Excel spreadsheets)
    -> We have a spreadsheet with the name of the books (books_filename)
    and a spreadsheet with the
        reviews for those books (ratings_filename)
"""

{"type": "string"}

# import csv data into dataframes
df_books = pd.read_csv(
books_filename,      encoding =
"ISO-8859-1",      sep=";",
header=0,
    names=['isbn', 'title', 'author'],
usecols=['isbn', 'title', 'author'],
    dtype={'isbn': 'str', 'title': 'str', 'author': 'str'})

df_ratings = pd.read_csv(
ratings_filename,
encoding = "ISO-8859-1",
sep=";",      header=0,
    names=['user', 'isbn', 'rating'],
usecols=['user', 'isbn', 'rating'],
    dtype={'user': 'int32', 'isbn': 'str', 'rating': 'float32'})

"""
We are still processing the data for the model:
    -> We have imported the names of the books and the reviews for
    those books - the two different      CSV files into variables
    -> Now we are taking those variables and converting them into
    pandas dataframes
    -> We are doing this both for the names of the books and for the
    reviews
    -> The top block of Python is for the names of the books
    -> The bottom block is for their reviews
"""

```

```
print("df_books:",df_books.head())
print("df_ratings.head:",df_ratings.head())

"""
(Above) this is the format of that dataframe:
-> We have the ISBN of the book
-> And then we have the title
-> For the ratings -> these are numerical values which rate the
books
-> We are not doing NLP which rates the books
-> We are doing K-Means on clusters of those ratings
-> We also have the user who gave that rating -> and each of the
users give different types of ratings

We now have:
-> A pandas dataframe (df_books), which contains the ISBN and name
for each of the books
-> A pandas dataframe (df_ratings), which contains the name of the
user, the rating they left and the
ISBN of the book
-> These were read from the CSV files of the dataset
-> We can cross-reference those datasets -> so if we know the
rating for a book in one dataset, we
can cross-reference its ISBN to find the name of the book from
the other dataset
"""
```

```
title \
0      0195153448      Classical
Mythology
1      0002005018      Clara
Callan  2  0060973129      Decision in
Normandy  3  0374157065  Flu: The Story of the Great Influenza
Pandemic...  4  0393045218      The
Mummies of Urumchi
author  0
Mark P. O. Morford
1      Richard Bruce Wright
2      Carlo D'Este
3      Gina Bari Kolata  4      E. J. W. Barber
df_ratings.head:      user      isbn  rating
0  276725  034545104X      0.0
1  276726  0155061224      5.0
2  276727  0446520802      0.0
3  276729  052165615X      3.0
4  276729  0521795028      6.0
```

```
{"type": "string"}
```

2.2 Remove Null Values From the Data Frames

```
"""
-> We are cleaning the datasets
-> This line calculates the number of null data points in the df_books
data frame
-> isnull is a boolean which checks this condition -> and sum adds all
of them up
-> It creates another data frame which is the same shape as df_books >
and is full of booleans
    -> Then it populates it -> with those booleans indicating if the
value at that point in the data frame is a nan
    -> Then the .sum method adds all of the nulls in those rows
-> We are counting the number of nans in the data frame -> to
clean them
    -> We are doing this using pandas
-> It's returning the number of null values in the isbn, title and
author columns of the df_books dataframe
"""

df_books.isnull().sum()

isbn      0
title     0
author    1
dtype: int64

#Doing the same -> checking the number of null datapoints which are in
the other dataframe
    #-> Remember that we are dealing with two datasets -> the first
contains the names of all of the
        #books and the second contains their reviews
    #-> This cell is checking for the number of null datapoints in the
second dataframe df_ratings.isnull().sum()

user      0
isbn      0
rating    0
dtype: int64

#Now we are getting rid of the rows with na which are in the first
dataframe
    #-> the argument inplace=True meaning we are altering the original
dataframe
df_books.dropna(inplace=True)

#checking the number of nulls in the df_books data frame to see if
they've been dropped and it's worked df_books.isnull().sum()
```

```
isbn      0
title     0
author    0
dtype: int64
```

2.3 Remove Users With Less Than 200 Ratings

```
"""
What this section of code (2.3) is doing:
    -> We are cleaning the data before training the model on it
    -> The last cells removed the null values from the data
    -> We have books and their reviews
    -> And then we have the users and the number of reviews which they
leave
    -> We want to get rid of users who've written less than 200 ratings
        -> because they could create inconsistencies in the data
    -> We also want to remove books with less than 100 ratings
    -> We are first removing users who have left less than 200 ratings
-> There are two data frames we have -> the first is for the books and
the second is for the users
    -> We are working on the data frame for the users -> this is
df_ratings
        -> and removing users with < 200 ratings

First printing out the shape of the data frame:
    -> We are taking the data frame which contains the users -> and
the aim is to remove users with less than 200 ratings
    -> We first return the shape of this data set
    -> The data frame we are working with is called df_ratings
"""

df_ratings.shape
# df_ratings.head
(1149780, 3)
"""
    -> We want to count the number of reviews which each user has left
-> Then to clean the entire data frame from all of the reviews left by
these users
    -> We can partition the dataset
    -> We have 1,149,779 users total
    -> ratings is like a 1D pandas dataframe which contains the user
and the amount of reviews which they left:
        -> We are taking the values in the column with the key
called 'user'
        -> And then calculating the amount of time that user appears
```



```

-> In other words, the amount of reviews which those users
left
-> The aim is to remove the users which leave less than
200 reviews
-> So we are taking the data frame with their users and
counting the amount of reviews which each user left
-> It is taking the `user` collumn of the dataframe and
creating a series
-> And in that series we have the frequency of
times that user appears in the data frame
-> This is how many reviews they've left
-> The format of what this returns:
-> This doesn't return a numpy array, it returns a pandas
series
-> The index of each element in the series is the ID of
the user, and the value is the number of reviews
which they left
-> We sort the values which are left:
-> Now we have a series which contains the number of
reviews each user left, and the index of that user
-> The aim is to get rid of the users who left less than
200 reviews
-> We need the indices of those users -> then we can
remove them from the dataset
-> We take the series with the number of reviews which
each user left, and sort them by the number of
reviews which the users left
-> We are doing this in descending order -> the
users who left the most reviews are at the top
-> We want to shave off the users whose indices are
below 200 in that list
"""

ratings = df_ratings['user'].value_counts()
ratings.sort_values(ascending=False).head()
11676      13602
198711      7550
153662      6109
98391       5891
35859       5850
Name: user, dtype: int64
"""

-> ratings is the pandas series from the previous cell, which contains
the index which represents the user and the amount of reviews
which they left
-> This entire series is in descending order (biggest numbers
at the top)

```


-> We want to remove the reviewers who left less than 200 reviews
 -> What we are doing in this cell is printing out the number of reviews that is
 -> In other words -> the 104378th user from the bottom of that series has left <200 reviews
 -> Meaning that we want to take the indices of these reviewers and remove all of their ratings from our data set
 -> The syntax of this line of code is saying
 -> Take the series which is stored in ratings
 -> Now return the number of users which have left < 200 ratings
 -> The values that series stores are the index of the users and the number of reviews which they've left
 """

```
len(ratings[ratings < 200])
104378
"""
```

Breaking this line down:

-> df_ratings['user']
 -> df_ratings is the pandas data frame which stores the ratings which the user left
 -> There was one data frame for the reviews which the users left, and another data frame for the names of the books and their ibsns
 -> This is taking the data set which contains the indices of the users and the reviews which they left
 -> We are taking the ['user'] collumn from this -> which is the column in the data frame which contains the list of all of the users
 -> .isin
 -> This is a boolean statement
 -> We are asking if these indices are in the list of users who left the reviews
 -> (ratings[ratings < 200].index)
 -> Ratings is the pandas series which contains the indices of the reviewers
 -> We are asking it for the indices of the reviewers who left less than 200 reviews in that series
 -> Combining this with .isin and we are comparing the indices of the users who left less than 200 reviews with the amount of times they appear in the original data frame total
 -> Then we are summing them using the .sum() method


```

    -> Combine everything and we are counting the number of reviews
left by all of the reviews who individually left less than
200 reviews
    -> These are the number of reviews that we are removing
from the data frame which contains the users and
their reviews (df_ratings)
"""

df_ratings['user'].isin(ratings[ratings < 200].index).sum()
622224
"""

    -> We are now defining another pandas data frame df_ratings_rm
-> df_ratings is the data frame which includes the index of the
users who left book reviews and the review they left
    -> df_ratings_rm is that review but without the indices of the
users who left less than 200 reviews
    -> The line in the middle of this block of code is the same as the
one in the previous cell -> just without the 'sum' method
being used
    -> This line is taking the column in the df_ratings data
frame which contains the indices of the users
    -> And then it's asking if the elements in that data frame
are in the one-dimensional pandas series
        called ratings, which contains the index of the user
and the number of reviews they left
    -> We are asking it for the indices of the users who left
less than 200 reviews -> and then removing
        their entries from the data frame which contains all
of the reviews (df_ratings)
    -> Then returning the shape of the cleaned data frame
    -> The data frames we have are
        -> df_ratings <- The pandas data frame with the indices of
the users and the review which they left
        -> ratings <- The pandas series which contains the number
of reviews the users left and their indices
        -> df_ratings_rm <- The pandas data frame which contains
the indices of the users and the reviews
            which they left - only including the users who left
200 or more reviews and excluding the rest
of them
        -> This is the cleaned data frame for the number of
reviews left per user
"""

df_ratings_rm = df_ratings[
    ~df_ratings['user'].isin(ratings[ratings < 200].index)
]

```

```
]
df_ratings_rm.shape

(527556, 3)
```

2.4 Remove Books With Less Than 100 Ratings

```
"""
There are two lines of code in this cell:
    -> df_ratings <- this is the pandas data frame which contains
the index of the user,
    the ISBN of the book they reviewed and the review which
they gave it (as a number)
    -> Each book has one ISBN number
    -> Since we want to remove the books from the data frame which
have less than 100 reviews,
    we need to count the number of reviews per book
    -> This is the number of times each ISBN appears in
the df_ratings data frame
    -> The first line in this cell is taking the column of
that data frame which contains
    the ISBN numbers of the books and counting the
number of times which each appears
    -> This is being stored in a variable called ratings -
> it's the equivalent of a frequency
    table for the ISBN of the book and the number of
times it's been reviewed
    -> Which is - a series
    -> It's basically a pandas series which is a histogram
for the ISBN of the book and the
    number of times it's been reviewed
    -> We want to remove the ones which have less
than 100 reviews in that series
    -> The second line of code in this cell is taking that pandas
series and sorting it in ascending order
    -> So now we have a pandas series which contains the
ISBN of the book and the number of times
it was reviewed
    -> That entire thing - sorted in ascending order
"""

ratings = df_ratings['isbn'].value_counts() # we have to use the
original df_ratings to pass the challenge
ratings.sort_values(ascending=False).head()
0971880107    2502
0316666343    1295
0385504209     883
0060928336     732
```



```
0312195516      723
```

```
Name: isbn, dtype: int64
```

```
#This returns the number of books which have less than 100 ratings #This is the number of books which we want to remove from our data set  
len(ratings[ratings < 100])
```

```
339825
```

```
"""
```

```
This line of code is the same as in the previous section, except for the number of book reviews
```

```
-> The previous section took the number of reviews that each customer had left,
```

```
and removed the customers who had left less than 200 reviews
```

```
-> For this we took each customer and indexed them
```

```
-> This is taking the data frame which contains the ISBNs with the books -> and taking
```

```
the ones which have less than 100 reviews
```

```
-> We are calculating the number of books which have less than 100 reviews
```

```
-> The method used to do this is the same as in the previous section, for the number of reviews per customer
```

```
-> But this time for the number of reviews per book
```

```
"""
```

```
df_books['isbn'].isin(ratings[ratings < 100].index).sum()
```

```
269442
```

```
"""
```

```
This code is the same as in the previous section:
```

```
-> The previous section took the number of reviews per reviewer
```

```
-> Then removed the reviews from the reviewers who had left less than 200 reviews
```

```
-> This section uses the same process on its sister data frame
```

```
-> Instead of the number of reviews per reviewer, we are targeting the number of reviews per book (ISBN number)
```

```
-> The code used to do this is the same as in the previous section, except that we are working on a different data frame
```

```
-> The one which contains the number of reviews for each book
```

```
-> Rather than the reviews which the user left
```

```
"""
```

```
df_ratings_rm = df_ratings_rm[
```



```

~df_ratings_rm['isbn'].isin(ratings[ratings < 100].index)
]
df_ratings_rm.shape
(49781, 3)
#To test if the data frame has been properly cleaned

"""
Testing to see if the data cleaning has worked:
    -> This section of the code is removing the books from the
data set who have less than
    100 reviews
    -> We run the risk of removing too many books from the data
set
    -> In this cell we are passing in the names of several books
into a for loop, to return
    booleans to see if those books which we know should be in
the cleaned data set are in
    the cleaned data set
    -> books <- this is the variable which is storing the names of
those books we know should be in the data set
    -> We are then iterating through that array -> and printing
out if those books are still in the data frame or not
    -> We are printing the value of a boolean statement
    -> That entire boolean statement is asking it to print the
number of times that statement is true
    -> And the statement is if the book which is in our list that
we are iterating through, for
    that instance of the book, is in the title column of the
cleaned dataframe of books """

# The list of books in this array should exist in our cleaned data
frame
books = ["Where the Heart Is (Oprah's Book Club (Paperback))",
        "I'll Be Seeing You",
        "The Weight of Water",
        "The Surgeon",
        "I Know This Much Is True"]

for book in books:
print(df_ratings_rm.isbn.isin(df_books[df_books.title ==
book].isbn).sum())

183
75

```

49
57
77

2.5 Prepare the Dataset for KNN

```
"""
    Context:
        -> We have now cleaned the data set
        -> The number of book reviews per user are all above 200 <-
this was the first metric which we targeted
        -> The number of reviews per book is above 100 <- this was the
second metric we targeted (in the block of code above)
        -> The aim is to perform K-Means on these data frames, to hunt
for clusters of similar books
            for the recommendation engine
        -> So now we need to take the cleaned data and put it into a
form which K-Means can be performed on, using scikit
learn

    What this cell does:
        -> We are defining a data frame which K-Means will be
performed on
        -> Now we have the cleaned data, we are creating a pandas data
frame in a variable called df, which
            will be used to perform the algorithm
        -> The data in the cells above was in two data frames
        -> The first was for the books and the second was for the
users
            -> The reviews per book, and the reviews per user
-> The second line in this cell prints out the head of the data
frame
            -> The first line in this cell defines the data frame which we
will perform K-Means on
            -> We are using the cleaned data frame (df_ratings_rm) to
create a pivot table
            -> This is the pivot table which we will perform K-Means on
            -> The arguments of this method are telling it the
architecture of the data frame -> the user column
is the index row of that table
            -> The ISBN column is used as the columns of that table
            -> The values of that table are the ratings
            -> We are replacing the NaN values in the data frame with 0's
-> We then transpose the entire thing to get it into the right format
-> the books are the rows and the users are the columns
            -> The rows are the users, and the columns are the books

```

(ISBNs) -> and their values are the ratings
for that book

-> There are 0's in that pivot table, instead of NaN's

"""

```
df =
df_ratings_rm.pivot_table(index=['user'],columns=['isbn'],values='rating').fillna(0).T df.head()
```

user	254	2276	2766	2977	3363	4017	4385
6242	\						

002542730X	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0							

0060008032	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0							

0060096195	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0							

006016848X	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0							

0060173289	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0							

user	6251	6323	...	274004	274061	274301	274308
274808	\						

isbn	...
------	-----

002542730X	0.0	0.0	...	0.0	0.0	0.0	0.0
0.0							

0060008032	0.0	0.0	...	0.0	0.0	0.0	0.0
0.0							

0060096195	0.0	0.0	...	0.0	0.0	0.0	0.0
0.0							

006016848X	0.0	0.0	...	0.0	0.0	0.0	0.0
0.0							

0060173289	0.0	0.0	...	0.0	0.0	0.0	0.0
0.0							

user	275970	277427	277478	277639	278418
	isbn				

002542730X	0.0	10.0	0.0	0.0	0.0
0060008032	0.0	0.0	0.0	0.0	0.0
0060096195	0.0	0.0	0.0	0.0	0.0
006016848X	0.0	0.0	0.0	0.0	0.0
0060173289	0.0	0.0	0.0	0.0	0.0

[5 rows x 888 columns]


```

"""
-> In the previous cell we defined df -> which was the pandas data
frame we want to perform K-Means on
-> In this cell we are adding the names of the books to this data
frame in string format -> rather than just the ISBNs of those
books
-> df_books.set_index('isbn') in the middle of this line is converting
from the ISBN of the book to the name of the book -> which we are
selecting using the
['title'] key and adding in using the .join method
      in pandas
-> We are taking the data frame which we previously defined to train
the model on and adding the names of the book to that data frame in
string format (rather than just
the ISBNs of those books)
      -> And we are doing this by adding them in from another
data frame, which contains the titles of those books
      rather than just their ISBNs
"""

```

```
df.index = df.join(df_books.set_index('isbn'))['title']
```

```

"""
-> df is now the same as it used to be (it's the data frame which
contains the cleaned data which we      want to perform K-Means
on)
      -> But after the last cell it now contains the names of the
books in string form, rather than      just their ISBNs
      -> The indices in that data frame are the names of those books in
string form
      -> This cell is now sorting these books according to these indices
-> In other words we are putting them in alphabetical, by sorting them
according to their indices
      -> Since their indices are strings representing their names
-> Then we are printing out the head of this data frame to confirm
that this has worked
"""

```

```
df = df.sort_index()
df.head()
```

user	254	2276	2766	2977	3363
4017					
4385	\				
1984	9.0	0.0	0.0	0.0	0.0
0.0					
0.0					
1st to Die: A Novel	0.0	0.0	0.0	0.0	0.0
0.0					

0.0


```

1st to Die: A Novel      0.0      0.0      0.0      0.0      0.0      0.0
0.0
2nd Chance              0.0      0.0      0.0      0.0      0.0      0.0
0.0
2nd Chance              0.0     10.0      0.0      0.0      0.0      0.0
0.0
user                    6242     6251     6323      ...    274004    274061
274301 \
title                    ...
1984                    0.0      0.0      0.0      ...     0.0      0.0
0.0
1st to Die: A Novel      0.0      0.0      0.0      ...     0.0      0.0
0.0
1st to Die: A Novel      0.0      0.0      0.0      ...     0.0      0.0
0.0
2nd Chance              0.0      0.0      0.0      ...     0.0      0.0
0.0
2nd Chance              0.0      0.0      0.0      ...     0.0      0.0
0.0
user                    274308    274808    275970    277427    277478    277639
278418
title
1984                    0.0      0.0      0.0      0.0      0.0      0.0
0.0
1st to Die: A Novel      0.0      0.0      0.0      0.0      0.0      0.0
0.0
1st to Die: A Novel      0.0      0.0      0.0      0.0      0.0      0.0
0.0
2nd Chance              0.0      0.0      0.0      0.0      0.0      0.0
0.0
2nd Chance              0.0      0.0      0.0      0.0      0.0      0.0
0.0
[5 rows x 888 columns]
#This cell is to return the reviews for a test book in the data frame
df which we are going to use to run K-Means on

"""
-> df is the name of the data frame we are going to perform K-Means on
-> The indices of that data frame are the names of the books in the
data frame, but in string form
-> This data frame has also been cleaned
-> What we are doing in this cell is:
    -> "The Queen of the Damned (Vampire Chronicles
(Paperback))" <- this is the name of a book in the data

```

```

frame in string form
-> The names of the books in string form in the df data
frame are its indices
-> We are asking it to locate the element in the data frame
with that index
-> But since there are multiple elements in the data frame
with that index (that book has been reviewed more than 100 times,
otherwise it wouldn't be in
the data frame df), we are asking it to return the
last 5 reviews for that book
-> We are saying, select all of the reviews for the book
called "The Queen of the Damned (Vampire
Chronicles
(Paperback))" -> now return us the last 5 ->
Setting the indices of the data frame to be the names of the books
allows us to extract their reviews like this -> by targeting the name
of a specific book
-> There are many reviews per book -> so if we dropped the [:5] it
would return all of the reviews for that book -> which are in
number form
-> How we got the names of the books in string form to be the indices
of that data frame was by using the .join method in the previous
cells
"""

df.loc["The Queen of the Damned (Vampire Chronicles (Paperback))"][:5]
user
254      0.0
2276     0.0
3363     0.0
Name: The Queen of the Damned (Vampire Chronicles (Paperback)), dtype:
float32

```

3. Train the Model Using KNN

3.1 Train the Model Using KNN

```

#Training the model using K-Means on the cleaned data frame for the
books from the previous section

"""
-> The aim of this entire section is to train the model using K-
Means on the df pandas data frame

```

```

    -> We have the df data frame from the previous section <- this is
the one we want to perform
        K-Means on
    -> As with the previous projects in the course, we first
initialise the architecture of the machine learning model and
then train it
    -> We are first setting a variable called model which stores the
architecture of the model
    -> We are doing K-Means -> this is taking the different
elements in the data set and storing
        them as vectors in higher dimensional space
    -> Using the 'cosine' metric is the equivalent of doing the
dot product for filtering systems
    -> We are asking the architecture of the model to calculate
the component of one vector (one vector being one book with all its
        reviews from
the different users) in the direction of the
        other vectors in the set
    -> Each of the elements in the set is a vector in higher
dimensional space
    -> This initialises the architecture of the model in the
vector called model -> but it doesn't
        actually populate it
    -> We are then training the model
    -> This is done using the second line in this cell ->
        Fitting the architecture of that model (set using the
variable called model) with the data stored in the cleaned pandas
        data frame called
df from the previous section of the notebook
    -> This creates a trained k-Means model -> on the different book
entries in the data set
    -> model is the variable which contains the clusters of
book entries in the data set
    -> And df is the pandas data frame which stores them ->
without the clustering algorithm having performed K-Means on
        them like the variable

model
"""

model = NearestNeighbors(metric='cosine')
model.fit(df.values)

NearestNeighbors(metric='cosine')

```

3.2 Create the get_recommends() Function

```

"""
-> In this section of the notebook, we are defining a function which

```

```

can use the trained model
to make predictions
-> We are printing out the shape of the data frame which K-Means was
performed on (and stored
    in the variable called `model`)
-> df.iloc[0] <- this is the first row of df, we are locating the 0th
row of the pandas data    frame df
-> Then the .shape method is for the shape of it -> the number of rows
and columns which the    user has rated
"""

df.iloc[0].shape
(888,)
"""
    -> df is the name of the data frame which we performed K-Means on
    -> The indices of that data frame are the names of the book whose
reviews it stores, in    string format
    -> We are giving it the name of a test book, in string format
    -> This is the same as one of the indices in the dataset -> and
the values which are stored
        at that point are all of the reviews for that book
    -> In number form
    -> What we are doing in the second line of code in this cell is
telling it to look in the    data frame df
    -> Locate the row which stores the book which has that index ->
those values are all of the    reviews for that book
    -> We are telling it to return the shape of what it finds (this
is the same as the number of    reviews for that book)
"""

title = 'The Queen of the Damned (Vampire Chronicles (Paperback))'
df.loc[title].shape
(888,)
"""
    -> We know
        -> The original cleaned data frame which we performed K-
Means on
            -> The model which has been fitted with this data using the
K-Means algorithm
        -> We want

```



```

    -> To define a function called the get_recommends()
function
    -> We want this to take the input of the name of a book in
string format (which is the same syntax of the indices of the df
        data frame) ->
and then to return a list of its five nearest
        neighbours from K-Means
    -> Along with the book's distance to each of those
neighbours
    -> So, we want to calculate the distance from that book to its
nearest neighbours in the vectorial space
    -> We also want to identify what its nearest neighbours are in
that higher dimensional space -> and output them as part of the
function
    -> We have set the problem up in a way such that the index is the
name of the book in string form -> and now we are calculating the
distance in another
variable which we are calling distance
    -> What this code does
        -> The variable `title` stores the name of that book which
we want to input into our function
        -> The model.kneighbors method
            -> This takes two arguments
                -> The first is the data point whose neighbours
we want to find
                    -> The syntax for this is more complicated
-> [df.loc[title].values]
                    -> This is telling it to look in the pandas
data frame called df, to find the row with the
                        index stored in the variable called title (the
name of the book whose neighbours we are
                        finding), and to extract its values
                    -> We are giving it the ratings for that
book -> which it interprets as a vector in higher
                        dimensional space -> and then we are telling
it the number of neighbours we want it to find
                    -> Since the model is set to use a cosine
metric, it is searching for the books whose reviews
are the most like it
                        -> The second is the number of data points we
want per cluster, including that data point
                            -> The syntax for this is easier than the
first argument which this method takes
                                -> It returns two pieces of data
                                    -> The first is an array of distances -> the
distances in between the data point which we entered
                                        into it (as its first argument) and all other points
in its cluster

```



```

        -> We are storing this argument in a variable
called distance
        -> The second is the indices of its neighbours
in the original data frame which was used to perform
        K-Means
        -> We are storing this in a variable called
indice
        -> We are then printing out both of the values which
these variables store, so that we can see the
syntax of the method's output
"""

distance, indice = model.kneighbors([df.loc[title].values],
n_neighbors=6)

print(distance)
print(indice)
[[0.          0.51784116 0.53763384 0.73450685 0.74486566 0.7939835 ]]
[[612 660 648 272 667 110]]
"""
    -> indice is the variable from the previous cell which contains the
indices of the 6 nearest neighbours from
        our input book in the pandas data frame df    ->
That is a list of all of its 5 nearest neighbours
    -> In this cell, we are extracting the index of the 0th nearest
neighbour in that array
    -> iloc is the method which is extracting the entire row of that
neighbour from the pandas df data frame
    -> That The index is the name of a book in string format
    -> indice[0] is the name of the nearest neighbour to the book which
we input into our function
    -> The indices of the elements in the df data frame are the names of
the books in it
    -> We are telling it to look for all of the entries in the original
df data frame with the index of that nearest        neighbour book
    -> Each of the elements in that data frame with that index are the
reviews for the nearest neighbour to the
        book which we have input into our function
    -> What we are doing is asking it to return the index and values of
those entries in the original data frame
"""

df.iloc[indice[0]].index.values
array(['The Queen of the Damned (Vampire Chronicles (Paperback))',
      'The Vampire Lestat (Vampire Chronicles, Book II)',
      'The Tale of the Body Thief (Vampire Chronicles (Paperback))',

```



```

        'Interview with the Vampire',
        'The Witching Hour (Lives of the Mayfair Witches)', 'Catch
22'],
        dtype=object)

"""
    -> Our aim is to define a function which returns the 5 nearest
    neighbours in the data set to the one which
        we input into the function
    -> And for it to return the distance from those neighbours
    to the book
    -> So we are first taking the different parts of that
    function and making them separately
    -> Then in the cell after this we combine them into that
    function
    -> Then in the cell after that we are testing this to see
    if the function works
    -> The Python in this cell is taking the two elements which
    we want this function to return (the distance from the input book
        to its nearest neighbours
    and the title of those neighbours) and putting it
    into a pandas data frame
    -> The first row of this data frame is for the titles of the 0th
    nearest neighbour to the input book
    -> df.iloc[indice[0]].index.values
    -> This is the same as the line of code from the previous
    cell
    -> This returns the values of the reviews from the original
    dataset for the book which is the closest
        neighbour to the input book in our function
    -> The second row of this data frame is for the distance between
    the book which is input into the function and its first nearest
    neighbour
    -> distance is the array which stores the distance between
    our input book and all of its 5 nearest neighbour
        book in higher dimensional space
    -> We are extracting the 0th element from that array ->
    which is the distance between that input book and
    its closes neighbour
    -> This data frame is only for the book which is input into our
    function and for its nearest neighbour
    -> To return the reviews for it and the distance in between
    them
    """

pd.DataFrame({
    'title' : df.iloc[indice[0]].index.values,
    'distance': distance[0]
}) \

```

```
.sort_values(by='distance', ascending=False)
```



```

                    title distance
5                Catch 22 0.793984
4  The Witching Hour (Lives of the Mayfair Witches) 0.744866
3                Interview with the Vampire 0.734507
2  The Tale of the Body Thief (Vampire Chronicles... 0.537634
1  The Vampire Lestat (Vampire Chronicles, Book II) 0.517841
0  The Queen of the Damned (Vampire Chronicles (P... 0.000000
# function to return recommended books - this will be tested
"""
    This cell combines all of the code from this section into the
    get_recommends function (see below)
    -> The argument to the function is the name of the book whose
    nearest neighbours we want
    -> these are the books which we recommend to it using the search
    engine
    -> If the book is not in the data frame, we return an error
    -> Then we find the 6th nearest neighbours using the model
    -> Then we create the data frame from those values and sort it by
    distance
    -> Then we return the values in the syntax that the questions asks
    for """

def get_recommends(title = ""):    try:        book
= df.loc[title]    except KeyError as e:
print('The given book', e, 'does not exist')
return

    distance, indice = model.kneighbors([book.values], n_neighbors=6)

    recommended_books = pd.DataFrame({
        'title' : df.iloc[indice[0]].index.values,
        'distance': distance[0]
    }) \
    .sort_values(by='distance', ascending=False) \
    .head(5).values

    return [title, recommended_books]

#This cell tests the get_recommends function when making a prediction
get_recommends("The Queen of the Damned (Vampire Chronicles
(Paperback))")
['The Queen of the Damned (Vampire Chronicles (Paperback))',
array([[ 'Catch 22', 0.793983519077301],

        ['The Witching Hour (Lives of the Mayfair Witches)',
         0.7448656558990479],

```

```

        ['Interview with the Vampire', 0.7345068454742432],
        ['The Tale of the Body Thief (Vampire Chronicles
(Paperback)) ',
        0.5376338362693787],
        ['The Vampire Lestat (Vampire Chronicles, Book II)',
        0.5178411602973938]], dtype=object)]

```

4. Using the get_recommends() Function to Make Predictions

Use the cell below to test your function. The `test_book_recommendation()` function will inform you if you passed the challenge or need to keep trying.

```

"""
This block of code contains Python tests which test our function to
see if it
    makes correct predictions or not. If the function is working, then
it returns a certain message, if otherwise this does not work.
"""

books = get_recommends("Where the Heart Is (Oprah's Book Club
(Paperback))")
print(books)

def test_book_recommendation():
    test_pass = True
    recommends = get_recommends("Where the Heart Is (Oprah's Book Club
(Paperback))")
    if recommends[0] != "Where the Heart Is (Oprah's Book Club
(Paperback))":
        test_pass = False
    recommended_books = ["I'll Be Seeing You ", 'The Weight of Water',
'The Surgeon', 'I Know This Much Is True']
    recommended_books_dist = [0.8, 0.77, 0.77, 0.77]
    for i in range(2):
        if recommends[1][i][0] not in recommended_books:
            test_pass = False
            print("You haven't passed yet. Keep trying!")
    if abs(recommends[1][i][1] - recommended_books_dist[i]) >= 0.05:
        test_pass = False
        print("You haven't passed yet. Keep trying!")
    else:
        print("You passed the challenge!
🌱🌱🌱🌱🌱")

```

```
test_book_recommendation()
```

```
["Where the Heart Is (Oprah's Book Club (Paperback))", array([[ "I'll  
Be Seeing You", 0.8016210794448853],  
    ['The Weight of Water', 0.7708583474159241],  
    ['The Surgeon', 0.7699410915374756],  
    ['I Know This Much Is True', 0.7677075266838074],  
    ['The Lovely Bones: A Novel', 0.7234864234924316]]],  
dtype=object)]  
You passed the challenge! 🍦🍦🍦🍦🍦
```