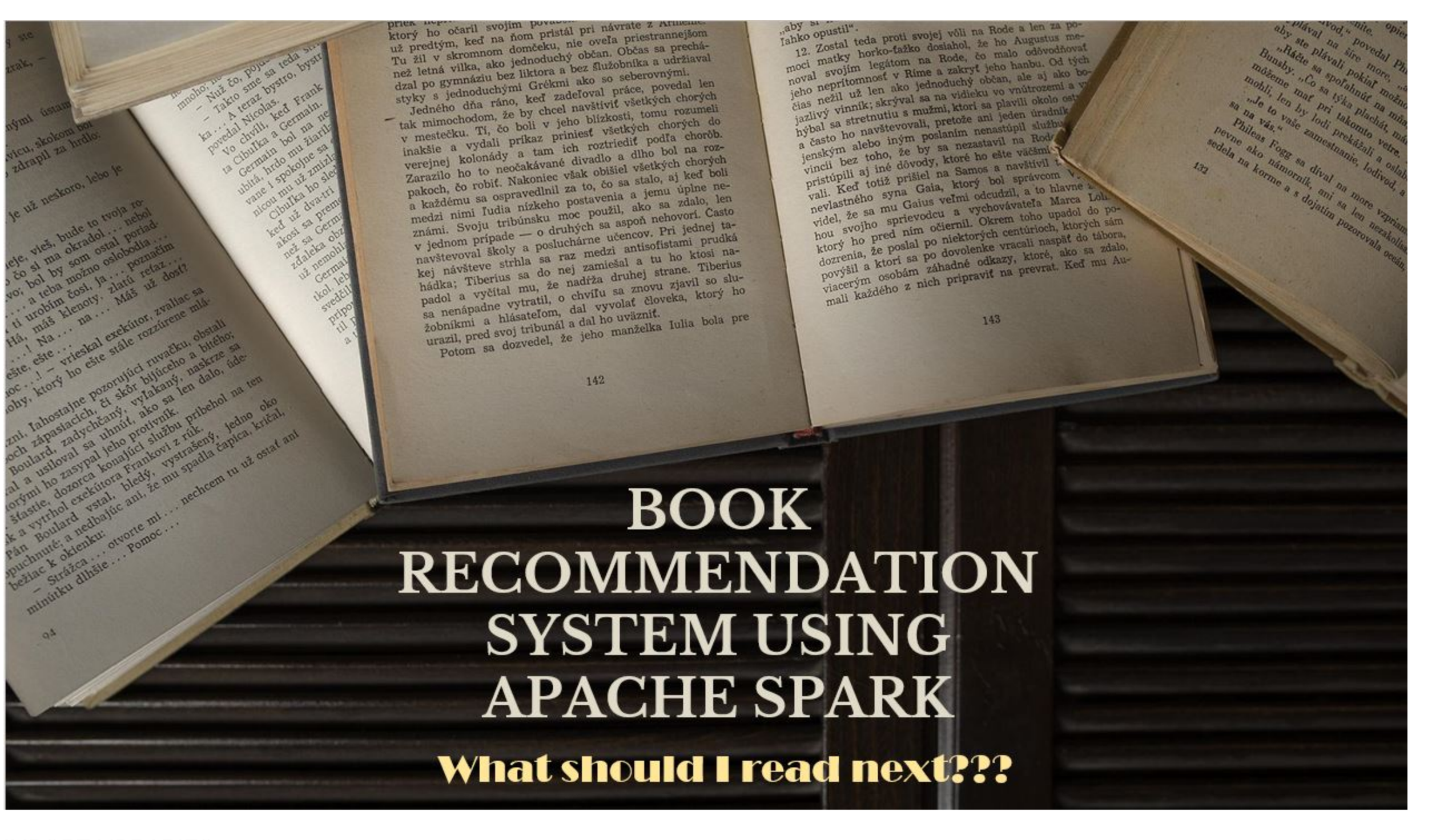


IMPORT LIBRARIES

In [2]:
```
import pyspark as ps
import pandas as pd
from pyspark.sql import SQLContext
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import Row
from pyspark.ml.recommendation import ALS
from pyspark.sql.functions import udf, col, when
import numpy as np
import matplotlib.pyplot as plt
```

In [23]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import datetime
%matplotlib inline
import plotly_express  as px
import warnings
warnings.filterwarnings('ignore')
```

KNN Model

In [28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
```

In [29]:
```
books = pd.read_csv('C:/Users/Anjana/Desktop/bdf/books.csv')
ratings = pd.read_csv('C:/Users/Anjana/Desktop/bdf/ratings.csv')
```

In [30]:
```
# Show all columns in DataFrame
pd.set_option('display.max_columns', None)
```

In [31]:
```
merged_df = pd.merge(books, ratings, how='left', left_on=['id'], right_on=['book_id'])
df = merged_df[['id','original_title', 'user_id', 'rating']]

df = df.rename(columns = {'id':'book_id'})
df.head(200)
```

Out[31]:

	book_id	original_title	user_id	rating
0	1	The Hunger Games	314	5
1	1	The Hunger Games	439	3
2	1	The Hunger Games	588	5
3	1	The Hunger Games	1169	4
4	1	The Hunger Games	1185	4
...	...	...	...	...
195	2	Harry Potter and the Philosopher's Stone	51166	5
196	2	Harry Potter and the Philosopher's Stone	51460	4
197	2	Harry Potter and the Philosopher's Stone	51838	4
198	2	Harry Potter and the Philosopher's Stone	52036	3
199	2	Harry Potter and the Philosopher's Stone	53292	5

200 rows × 4 columns

In [32]:
```
ratings_df = df.pivot_table(index='book_id',columns='user_id',values='rating').fillna(0)

pd.set_option('display.max_columns', 100)
ratings_df.head()
```

Out[32]:

user_id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
book_id																																															
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 53424 columns

In [33]:
```
ratings_matrix = csr_matrix(ratings_df.values)
```

In [34]:
```
model_knn = NearestNeighbors(metric='cosine', algorithm = 'brute')
model_knn.fit(ratings_matrix)
```

Out[34]:
```
NearestNeighbors(algorithm='brute', metric='cosine')
```

In [58]:
```
def get_book_id(book_title):
    target_df = df.loc[df['original_title'] == book_title]
    return target_df['book_id'].iloc[0]

id_TheHungerGames = get_book_id('Catching Fire')
print(id_TheHungerGames)
```

```
6148028
```

In [59]:
```
def get_title(book_id):
    target_df = df.loc[df['book_id'] == book_id]
    return target_df['original_title'].iloc[0]

print(get_title(6148028))
```

```
Catching Fire
```

In [60]:
```
def get_recomm(book_title, num_neighbors=10, display=False):
    book_ids = []

    query_index = get_book_id(book_title) - 1

    if num_neighbors > 0:
        distances, indices = model_knn.kneighbors(ratings_df.iloc[query_index,:].values.reshape(1, -1), n_neighbors = num_neighbors + 1)
    else:
        distances, indices = model_knn.kneighbors(ratings_df.iloc[query_index,:].values.reshape(1, -1), n_neighbors = 10 + 1)

    for i in range(0, len(distances.flatten())):
        if display is True:
            if i == 0:
                print('Recommendations for ', book_title, '\n')
            else:
                print('{0}\t Book ID: {1}\t  Distance: {2}:\n'.format(i, ratings_df.index[indices.flatten()[i]], distances.flatten()[i]))

        book_ids.append(ratings_df.index[indices.flatten()[i]])

    return book_ids
```

In [39]:
```
for b in recommendations_for_TheHungerGames[1:]:
    print('id:', b, '\t\tBook: ', get_title(b))
```

```
id: 17          Book:  Catching Fire
id: 31          Book:  The Help
id: 2           Book:  Harry Potter and the Philosopher's Stone
id: 20          Book:  Mockingjay
id: 3           Book:  Twilight
id: 93          Book:  The Secret Garden
id: 5           Book:  The Great Gatsby
id: 16          Book:  Män som hatar kvinnor
id: 9           Book:  Angels & Demons
id: 37          Book:  The Lion, the Witch and the Wardrobe
```

In [40]:
```
# Top 10 recommendations for Harry Potter and the Philosopher's Stone

book_ids_for_H = get_recomm('Harry Potter and the Philosopher\'s Stone', num_neighbors=10)
# skip the first item
for b in book_ids_for_H[1:]:
    print(get_title(b))
```

```
To Kill a Mockingbird
Memoirs of a Geisha
Nineteen Eighty-Four
The Great Gatsby
 The Fellowship of the Ring
Lord of the Flies
Harry Potter and the Prisoner of Azkaban
The Hobbit or There and Back Again
Het Achterhuis: Dagboekbrieven 14 juni 1942 - 1 augustus 1944
Jane Eyre
```

In [42]:
```
# Top 15 recommendations for Twilight

book_ids_for_H = get_recomm('Twilight', num_neighbors=15)
for b in book_ids_for_H[1:]:
    print(get_title(b))
```

```
Harry Potter and the Philosopher's Stone
Pride and Prejudice
The Hunger Games
Memoirs of a Geisha
The Kite Runner
The Catcher in the Rye
The Book Thief
O Alquimista
The Giver
Het Achterhuis: Dagboekbrieven 14 juni 1942 - 1 augustus 1944
 The Fellowship of the Ring
Män som hatar kvinnor
The Lovely Bones
Jane Eyre
Fahrenheit 451
```

To show all predicted images

In [3]:
```
from IPython.display import Image
from IPython.display import display

#create a spark session
#create a spark context

conf = ps.SparkConf().setMaster("yarn-client").setAppName("BDF Project")
conf.set("spark.executor.heartbeatInterval","3600s")

spark = ps.sql.SparkSession.builder \
           .master("local[4]")\
           .appName("BDF Project") \
           .getOrCreate()

sc = spark.sparkContext
sqlContext = SQLContext(sc)
```

```
C:\Users\Anjana\anaconda3\lib\site-packages\pyspark\sql\context.py:77: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
```

In [4]:
```
ratings_df = spark.read.csv('C:/Users/Anjana/Desktop/bdf/ratings.csv',header=True, inferSchema=True)
ratings_df.printSchema()
```

```
root
 |-- book_id: integer (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- rating: integer (nullable = true)
```

In [5]:
```
ratings_df.describe()
```

Out[5]:
```
DataFrame[summary: string, book_id: string, user_id: string, rating: string]
```

In [6]:
```
ratings_df.show(10)
```

```
+-------+-------+------+
|book_id|user_id|rating|
+-------+-------+------+
|      1|    314|     5|
|      1|    439|     3|
|      1|    588|     5|
|      1|   1169|     4|
|      1|   1185|     4|
|      1|   2077|     4|
|      1|   2487|     4|
|      1|   2900|     5|
|      1|   3662|     4|
|      1|   3922|     5|
+-------+-------+------+
only showing top 10 rows
```

In [7]:
```
books_df = spark.read.csv('C:/Users/Anjana/Desktop/bdf/books.csv',header=True, inferSchema=True)
books_df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- book_id: integer (nullable = true)
 |-- best_book_id: integer (nullable = true)
 |-- work_id: integer (nullable = true)
 |-- books_count: integer (nullable = true)
 |-- isbn: string (nullable = true)
 |-- isbn13: double (nullable = true)
 |-- authors: string (nullable = true)
 |-- original_publication_year: double (nullable = true)
 |-- original_title: string (nullable = true)
 |-- title: string (nullable = true)
 |-- language_code: string (nullable = true)
 |-- average_rating: string (nullable = true)
 |-- ratings_count: string (nullable = true)
 |-- work_ratings_count: string (nullable = true)
 |-- work_text_reviews_count: string (nullable = true)
 |-- ratings_1: double (nullable = true)
 |-- ratings_2: integer (nullable = true)
 |-- ratings_3: integer (nullable = true)
 |-- ratings_4: integer (nullable = true)
 |-- ratings_5: integer (nullable = true)
 |-- image_url: string (nullable = true)
 |-- small_image_url: string (nullable = true)
```

In [8]:
```
books_df.show(n=1)
```

```
+---+-------+------------+-------+-----------+---------+----------------+--------------+-------------------------+----------------+--------------------+-------------+-------------
-+-------------+------------------+-----------------------+---------+---------+---------+---------+---------+--------------------+--------------------+
| id|book_id|best_book_id|work_id|books_count|     isbn|          isbn13|       authors|original_publication_year|  original_title|               title|language_code|average_ratin
g|ratings_count|work_ratings_count|work_text_reviews_count|ratings_1|ratings_2|ratings_3|ratings_4|ratings_5|           image_url|     small_image_url|
+---+-------+------------+-------+-----------+---------+----------------+--------------+-------------------------+----------------+--------------------+-------------+-------------
-+-------------+------------------+-----------------------+---------+---------+---------+---------+---------+--------------------+--------------------+
|  1|2767052|     2767052|2792775|        272|439023483|9.78043902348E12|Suzanne Collins|                   2008.0|The Hunger Games|The Hunger Games ...|          eng|          4.3
4|      4780653|           4942365|                 155254|  66715.0|   127936|   560092|  1481305|  2706317|https://images.gr...|https://images.gr...|
+---+-------+------------+-------+-----------+---------+----------------+--------------+-------------------------+----------------+--------------------+-------------+-------------
-+-------------+------------------+-----------------------+---------+---------+---------+---------+---------+--------------------+--------------------+
only showing top 1 row
```

In [9]:
```
books_df.describe()
```

Out[9]:
```
DataFrame[summary: string, id: string, book_id: string, best_book_id: string, work_id: string, books_count: string, isbn: string, isbn13: string, authors: string, original_publicat
ion_year: string, original_title: string, title: string, language_code: string, average_rating: string, ratings_count: string, work_ratings_count: string, work_text_reviews_count:
string, ratings_1: string, ratings_2: string, ratings_3: string, ratings_4: string, ratings_5: string, image_url: string, small_image_url: string]
```

In [17]:
```
ratings_df = ratings_df.withColumn("user_id",
                                   ratings_df["user_id"]
                                   .cast('int'))
ratings_df.printSchema()
```

```
root
 |-- book_id: float (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- rating: float (nullable = true)
```

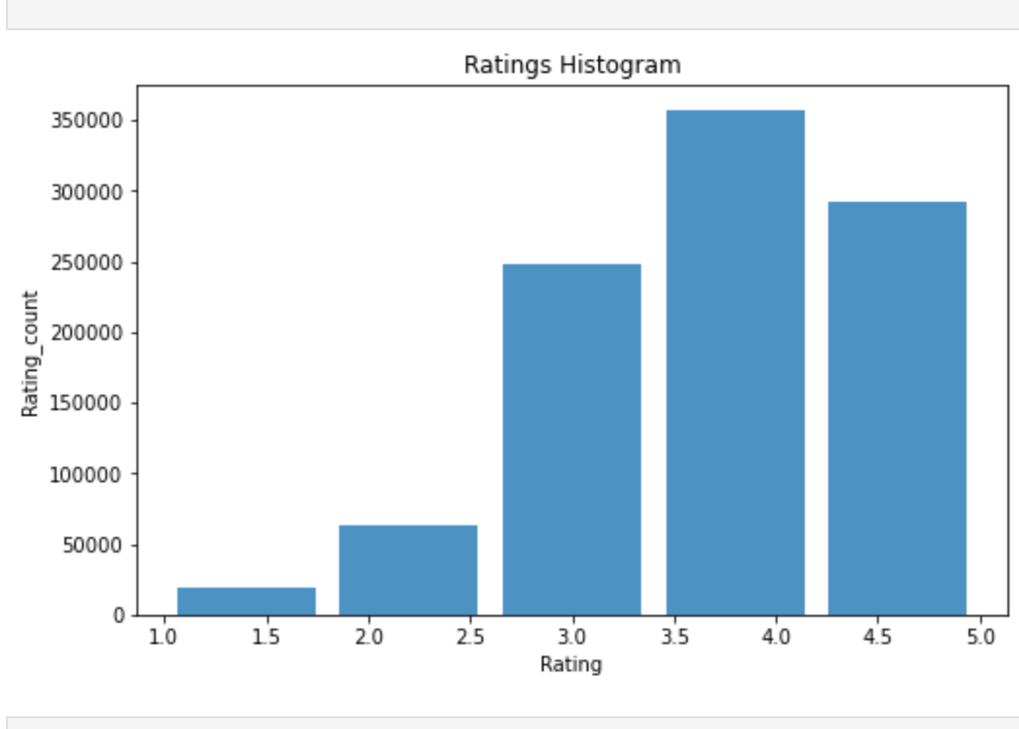
In [18]:
```
ratings_df = ratings_df.withColumn("rating",
                                   ratings_df["rating"]
                                   .cast('int'))
ratings_df.printSchema()
```

```
root
 |-- book_id: float (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- rating: integer (nullable = true)
```

In [19]:
```
ratings_df = ratings_df.withColumn("book_id",
                                   ratings_df["book_id"]
                                   .cast('int'))
ratings_df.printSchema()
```

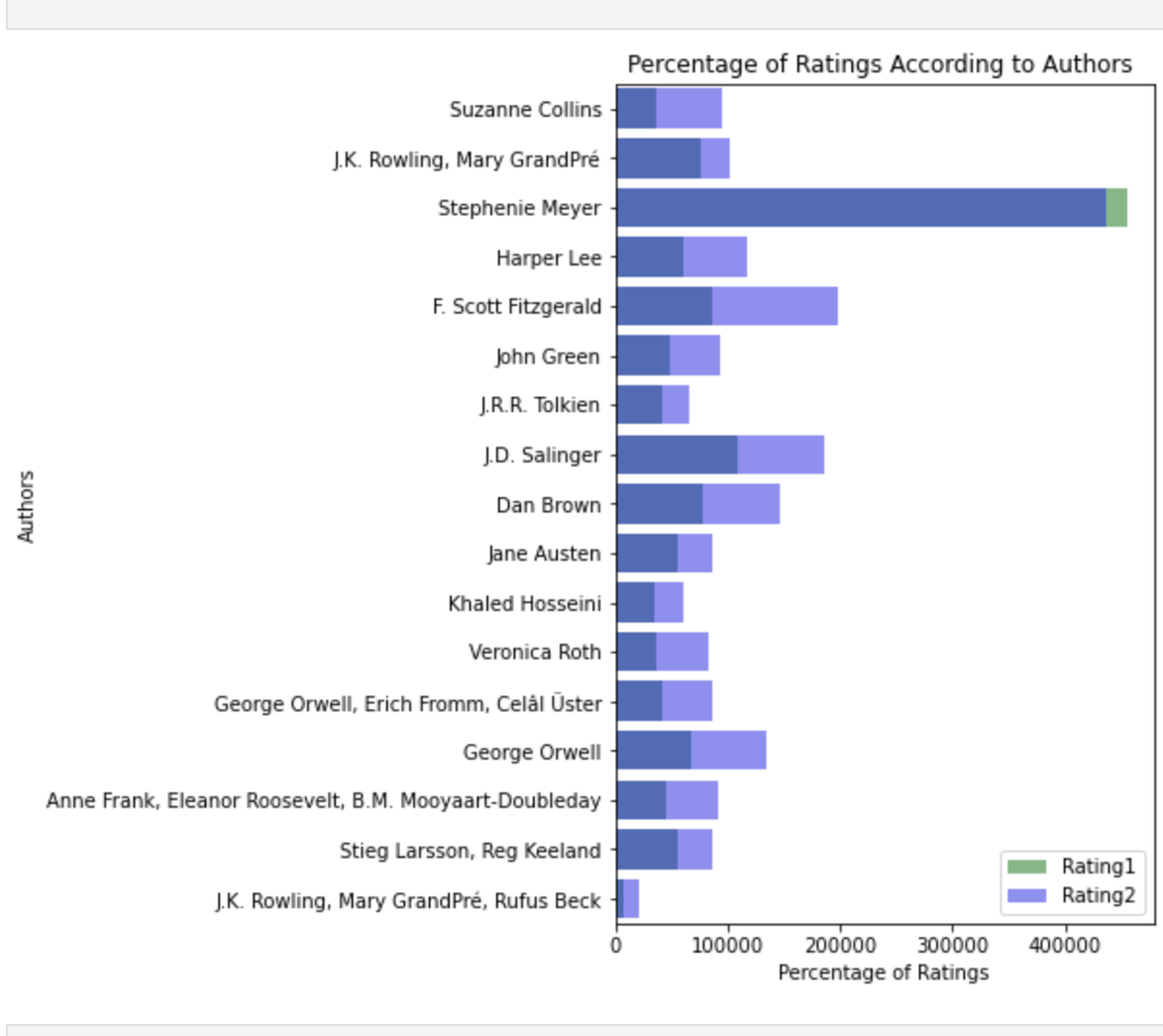
```
root
 |-- book_id: integer (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- rating: integer (nullable = true)
```

In [43]:
```
plt.figure(figsize=(8,5))
# An "interface" to matplotlib.axes.Axes.hist() method
n, bins, patches = plt.hist(x=ratings['rating'],bins=5,alpha=0.8,rwidth=0.85)
plt.xlabel('Rating')
plt.ylabel('Rating_count')
plt.title('Ratings Histogram')
plt.show()
```



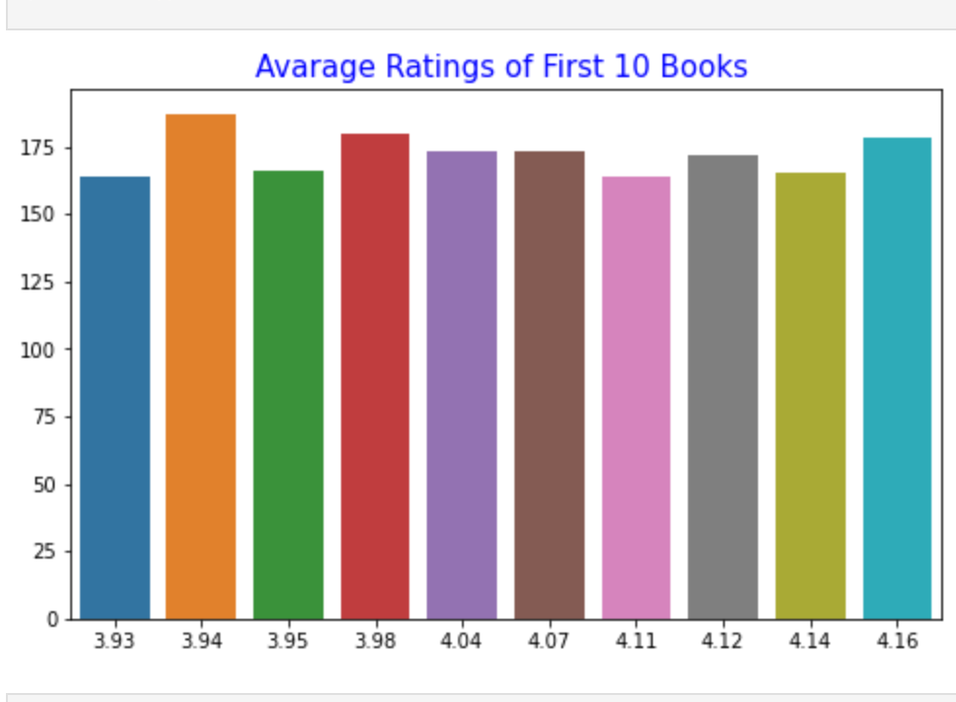
In [44]:
```
#change some features' name.
data1= books.head(20)
data1.rename(columns={'ratings_1':'R1', 'ratings_2':'R2','ratings_3':'R3','ratings_4':'R4','ratings_5':'R5'}, inplace=True)
```

In [45]:
```
# Percentage of Ratings According to Authors
author_list= list(data1['authors'].unique())

ratings1= []
ratings2= []
for i in author_list:

    x = data1[data1['authors']==i]
    ratings1.append(sum(x.R1)/len(x))
    ratings2.append(sum(x.R2)/len(x))

f,ax = plt.subplots(figsize = (5,8))
sns.barplot(x=ratings1,y=author_list,color='green',alpha = 0.5,label='Rating1' )
sns.barplot(x=ratings2,y=author_list,color='blue',alpha = 0.5,label='Rating2' )
ax.legend(loc='lower right',frameon = True)
ax.set(xlabel='Percentage of Ratings', ylabel='Authors',title = "Percentage of Ratings According to Authors ")
plt.show()
```



In [47]:
```
# import plotly.plotly as py
from plotly.offline import init_notebook_mode, iplot, plot
import plotly as py
init_notebook_mode(connected=True) #offline modela ilgili
import plotly.graph_objs as go


from wordcloud import WordCloud
# matplotlib
import matplotlib.pyplot as plt
```

In [49]:
```
data3=books.head(6)
sta = books.average_rating.value_counts().index[:10]
plt.figure(figsize = (8,5))
sns.barplot(x=sta,y =books.average_rating.value_counts().values[:10])
plt.title('Avarage Ratings of First 10 Books ',color = 'blue',fontsize=15)
plt.show()
```



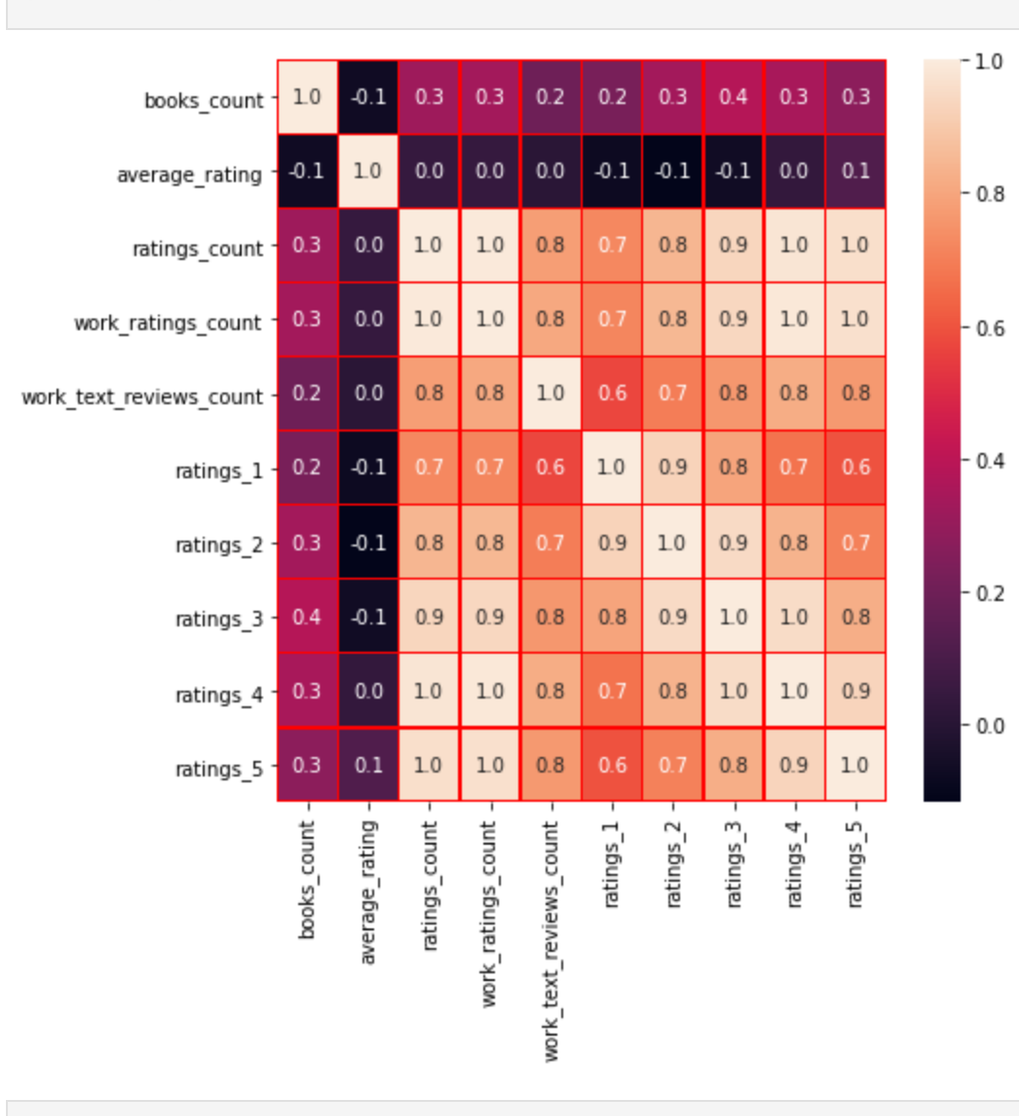
In [53]:
```
books2= books.drop(columns=['work_id','id','book_id','best_book_id','best_book_id','isbn13','original_publication_year',"image_url","small_image_url"])
```

In [54]:
```
books2.corr()
```

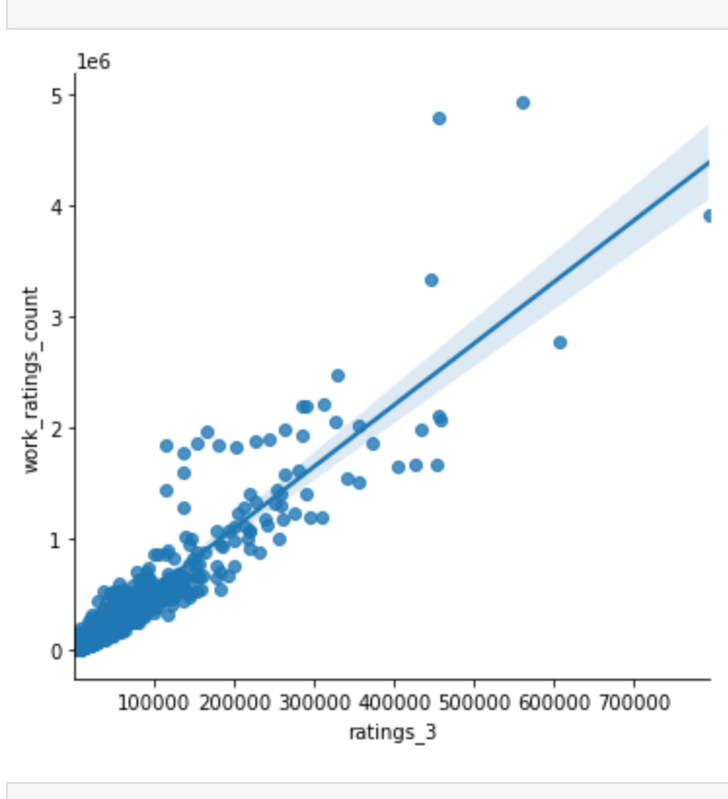
Out[54]:

	books_count	average_rating	ratings_count	work_ratings_count	work_text_reviews_count	ratings_1	ratings_2	ratings_3	ratings_4	ratings_5
books_count	1.000000	-0.069888	0.324235	0.333664	0.198698	0.225763	0.334923	0.383699	0.349564	0.279559
average_rating	-0.069888	1.000000	0.044990	0.045042	0.007481	-0.077997	-0.115875	-0.065237	0.036108	0.115412
ratings_count	0.324235	0.044990	1.000000	0.995068	0.779635	0.723144	0.845949	0.935193	0.978869	0.964046
work_ratings_count	0.333664	0.045042	0.995068	1.000000	0.807009	0.718718	0.848581	0.941182	0.987764	0.966587
work_text_reviews_count	0.198698	0.007481	0.779635	0.807009	1.000000	0.572007	0.696880	0.762214	0.817826	0.764940
ratings_1	0.225763	-0.077997	0.723144	0.718718	0.572007	1.000000	0.926140	0.795364	0.672986	0.597231
ratings_2	0.334923	-0.115875	0.845949	0.848581	0.696880	0.926140	1.000000	0.949596	0.838298	0.705747
ratings_3	0.383699	-0.065237	0.935193	0.941182	0.762214	0.795364	0.949596	1.000000	0.952998	0.825550
ratings_4	0.349564	0.036108	0.978869	0.987764	0.817826	0.672986	0.838298	0.952998	1.000000	0.933785
ratings_5	0.279559	0.115412	0.964046	0.966587	0.764940	0.597231	0.705747	0.825550	0.933785	1.000000

In [55]:
```
f,ax = plt.subplots(figsize=(7,7))
sns.heatmap(books2.corr(), annot=True, linewidths=0.6,linecolor="red", fmt= '.1f',ax=ax)
plt.show()
```



In [57]:
```
sns.lmplot(x="ratings_3", y="work_ratings_count", data=books2)
plt.show()
```



In [ ]:
```
import pyspark.sql.functions as F

df = df.withColumn("AddCol",F.when(F.col("Pclass").like("3"),"three").otherwise("notthree"))
```

In [1]:
```
#import pyspark.sql.functions as F
#cols_to_cast = books_df["average_rating"]
#books_df["average_rating"]=books_df["average_rating"]([F.cast('double')])
```

In [ ]:
```
%sql
select original_publication_year, count(*) as count from books where
original_publication_year > 1950 group by original_publication_year
```

In [ ]:
```
authors_with_most_books = pd.DataFrame(books_df.authors.value_counts()[0:10]).reset_index()
authors_with_most_books.columns = ['author', 'number_of_books']
```

In [ ]:
```
print(books_df['average_rating'])
```

In [ ]:
```
books_df = books_df.withColumn("average_rating",
                               books_df["average_rating"]
                               .cast('float'))
books_df.printSchema()
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.title('Distribution of Average Ratings')
books_df['average_rating'].hist()
display()
```

In [ ]:
```
 pip install plotly==5.7.0
```

In [ ]:
```
pip install chart_studio
```

In [ ]:
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
from chart_studio import plotly
import plotly.plotly as py
import plotly.graph_objs as go
import pandas as pd
import requests
requests.packages.urllib3.disable_warnings()
```

MODEL

In [ ]:
```
training_df, validation_df = ratings_df.randomSplit([.8, .2])
```

In [ ]:
```
iterations = 10
regularization_parameter = 0.1 #lambda
rank = 4 #rank
errors = []
err = 0
```

In [ ]:
```
als = ALS(maxIter=iterations, regParam=regularization_parameter, rank=4, userCol='user_id',itemCol='book_id',ratingCol='rating')
model=als.fit(training_df)
predictions=model.transform(validation_df)
new_predictions=predictions.filter(col('prediction')!= np.nan)
evaluator=RegressionEvaluator(metricName='rmse',labelCol="rating",predictionCol="prediction")
rmse=evaluator.evaluate(new_predictions)
print("Root Mean Square Error Value = "+ str(rmse))
```

In [ ]:
```
als = ALS(maxIter=iterations, regParam=regularization_parameter, rank=5, userCol='user_id',itemCol='book_id',ratingCol='rating')
model=als.fit(training_df)
predictions=model.transform(validation_df)
new_predictions=predictions.filter(col('prediction')!= np.nan)
evaluator=RegressionEvaluator(metricName='rmse',labelCol="rating",predictionCol="prediction")
rmse=evaluator.evaluate(new_predictions)
print("Root Mean Square Error Value = "+ str(rmse))
```

In [ ]:
```
'''for rnk in range(4,7):
    als= ALS(maxIter=iterations, regParam=regularization_parameter,rank=rank,userCol='user_id',itemCol='book_id',ratingCol='rating')
    model=als.fit(training_df)
    predictions=model.transform(validation_df)
    new_predictions=predictions.filter(col('prediction') != np.nan)
    evaluator=RegressionEvaluator(metricName='rmse',labelCol="rating",predictionCol="prediction")
    rmse=evaluator.evaluate(new_predictions)
    print("Root Mean Square Error Value = "+ str(rmse))'''
```

In [ ]:
```
'''ls=ALS(maxIter=iterations, regParam=regularization_parameter,rank=rank,userCol='user_id',itemCol='book_id',ratingCol='rating')
paramGrid=ParamGridBuilder() \
    .addGrid(als.regParam, [0.1, 0.01, 0.18])\
    .addGrid(als.rank, range(4, 10)) \
    .build()
evaluator=RegressionEvaluator(metricName='rmse',labelCol="rating",predictionCol="prediction")
crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)
cvModel= crossval.fit(training_df)'''
```

In [ ]:
```
predictions= model.transform(validation_df)
predictions.show(n = 10)
```

In [ ]:
```
predictions.join(books_df,"book_id").select("user_id","title","prediction").show(5)
```

In [ ]:
```
for_one_user=predictions.filter(col("user_id")==35982).join(books_df,"book_id").select("user_id","title","image_url","prediction")
for_one_user.count()
```

In [ ]:
```
for_one_user.show()
```

In [ ]:
```
 for book in for_one_user.take(10):
        print(book.title)
        display(Image(url=book.image_url))
```

In [ ]:
```
#generate top 5 books recommendations for each user
userRecomments = model.recommendForAllUsers(5)
#generate top 5 books recommendations related to each book
bookRecomments = model.recommendForAllItems(5)
```

In [ ]:
```
userRecomments.select("user_id","recommendations.book_id").show(10,False)
```

In [ ]:
```
userRecomments.printSchema()
```

In [ ]:
```
bookRecomments.select("book_id","recommendations.user_id").show(10, False)
```

In [ ]:
```
# Generate top 10 Book recommendation for a specified set of users
users = ratings_df.select("user_id").distinct().limit(3);
users.show()
```

In [ ]:
```
userSubsetRecs = model.recommendForUserSubset(users, 10)
```

In [ ]:
```
userSubsetRecs.show()
```

In [ ]:
```
userSubsetRecs.select("user_id","recommendations.book_id").show(10, False)
```

In [ ]:
```
# Generate top 10 user recommendations for a specified set of books
books = ratings_df.select("book_id").distinct().limit(3)
books.show()
```

In [ ]:
```
bookSubSetRecs = model.recommendForItemSubset(movies, 10)


bookSubSetRecs.select("book_id","recommendations.user_id").show(10, False)
```

```
import os
import math
from zipfile import ZipFile
from urllib.request import urlretrie
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.lay
from sklearn.model_selecti
```

```
dataset = pd.read_csv('/content/rati
train, test = train_test_split(datas
n_users = len(dataset.user_id.unique
n_books = len(dataset.book_id.unique

dataset.head()
```

	book_id	user_id	rating
0	1	314	5
1	1	439	3
2	1	588	5
3	1	1169	4
4	1	1185	4

```
from keras.layers import Input, Embedding, Flatten, Dot, Dense
from keras.models import Model

book_input = Input(shape=[1], name="Book-Input")
book_embedding = Embedding(n_books+1, 5, name="Book-Embedding")(book_input
book_vec = Flatten(name="Flatten-Books")(book_embedding)

user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users+1, 5, name="User-Embedding")(user_input
user_vec = Flatten(name="Flatten-Users")(user_embedding)

prod = Dot(name="Dot-Product", axes=1)([book_vec, user_vec])
model = Model([user_input, book_input], prod)
model.compile('adam', 'mean_squared_error')
```

```
history = model.fit([train.user_id, train.book_id], train.rating, epochs=1(
model.save('regression_model.h5')
```

```
Epoch 1/10
24544/24544 [==============================] - 191s 8ms/step - loss:
Epoch 2/10
24544/24544 [==============================] - 168s 7ms/step - loss:
Epoch 3/10
24544/24544 [==============================] - 165s 7ms/step - loss:
Epoch 4/10
24544/24544 [==============================] - 158s 6ms/step - loss:
Epoch 5/10
24544/24544 [==============================] - 175s 7ms/step - loss:
Epoch 6/10
24544/24544 [==============================] - 178s 7ms/step - loss:
Epoch 7/10
24544/24544 [==============================] - 177s 7ms/step - loss:
Epoch 8/10
24544/24544 [==============================] - 164s 7ms/step - loss:
Epoch 9/10
24544/24544 [==============================] - 167s 7ms/step - loss:
Epoch 10/10
24544/24544 [==============================] - 160s 7ms/step - loss:
```

Predicting using books.csv

```
# Creating dataset for making recommendations for the first user
book_data = np.array(list(set(dataset.book_id)))
user = np.array([1 for i in range(len(book_data))])
predictions = model.predict([user, book_data])
predictions = np.array([a[0] for a in predictions])
recommended_book_ids = (-predictions).argsort()[:5]
print(recommended_book_ids)
print(predictions[recommended_book_ids])
```

```
[5274 9841 8925 8258 8998]
[5.014593  4.9877563 4.947895  4.9065075 4.899353 ]
```

```
books = pd.read_csv('/content/books.csv')
books.head()
```

	id	book_id	best_book_id	work_id	books_count	isbn	isbn
0	1	2767052	2767052	2792775	272	439023483	9.780439e
1	2	3	3	4640799	491	439554934	9.780440e
2	3	41865	41865	3212258	226	316015849	9.780316e
3	4	2657	2657	3275794	487	61120081	9.780061e
4	5	4671	4671	245494	1356	743273567	9.780743e

5 rows × 23 columns

```
print(books[books['id'].isin(recommended_book_ids)])
```

```
        id   book_id  best_book_id   work_id  books_count        isbn
5273  5274     86856         86856    107052           38    60012781
8257  8258  18594594      18594594  26341000           41   345547497
8924  8925  17255186      17255186  23848838           16   988262592
8997  8998    292740        292740   2457130           37  2266071289
9840  9841     15101         15101    876908           55   380815923

            isbn13                              authors  \
5273  9.780060e+12                      Anthony Bourdain
8257  9.780346e+12                       Karin Slaughter
8924  9.780988e+12  Gene Kim, Kevin Behr, George Spafford
```

```
8997  9.782266e+12                    Lorenzo Carcaterra
9840  9.780381e+12                         Joanne Harris

      original_publication_year  \
5273                     2001.0
8257                     2014.0
8924                     2013.0
8997                     1995.0
9840                     1999.0

                                        original_title  ... ratings_
5273  A Cook's Tour: Global Adventures in Extreme Cu...  ...
8257                                          Cop Town  ...
8924                                               NaN  ...
8997                                           Sleepers  ...
9840                                     Blackberry Wine  ...

     work_ratings_count  work_text_reviews_count  ratings_1  ratings_
5273              18151                      885        209        63
8257              13997                     1704        274        78
8924              11237                     1177         73        29
8997              10772                      553         73        30
9840              11107                      699        163        75

      ratings_3  ratings_4  ratings_5  \
5273       4076       7837       5390
8257       3299       5612       4032
8924       1493       4354       5025
8997       1775       4088       4532
9840       3215       4220       2754

                                              image_url  \
5273  https://s.gr-assets.com/assets/nophoto/book/11...
8257  https://images.gr-assets.com/books/1384822680m...
8924  https://images.gr-assets.com/books/1361113128m...
8997  https://images.gr-assets.com/books/1327871596m...
9840  https://s.gr-assets.com/assets/nophoto/book/11...

                                        small_image_url
5273  https://s.gr-assets.com/assets/nophoto/book/50...
8257  https://images.gr-assets.com/books/1384822680s...
8924  https://images.gr-assets.com/books/1361113128s...
8997  https://images.gr-assets.com/books/1327871596s...
9840  https://s.gr-assets.com/assets/nophoto/book/50...

[5 rows x 23 columns]
```

Code snippets ×

Filter code snippets

- Adding form fields
- Camera Capture
- Cross-output communication
- display.Javascript to execute JavaScript from...
- Downloading files or importing data from Go...
- Downloading files to your local file system
- Evaluate a Javascript expression from Pytho...
- Hiding code
- Importing a library that is not in Colaboratory
- Importing data from Google Sheets
- Install [cartopy](http://scitools.org.uk/cartopy...
- Install 7zip reader [libarchive](https://pypi.pyt...
- Install GraphViz & [PyDot](https://pypi.python...
- Javascript to Python communication
- Jupyter Comms
- Jupyter Widgets
- Listing files in Google Drive
- Mounting Google Drive in your VM
- Open files from GCS with gsutil
- Open files from GCS with the Cloud Storage P...
- Open files from GitHub
- Open files from Google Drive
- Open files from your local file system
- Output Handling
- Pandas: display dataframes as interactive tab...
- Pausing output processing

Adding form fields

Insert

Forms example

Forms support multiple types of fields with type checking including sliders, date pickers, input fields, dropdown menus, and dropdown menus that allow input.

```
#@title Example form f
#@markdown Forms suppo

no_type_checking = ''
string_type = 'example
slider_value = 142  #@
number = 102  #@param
date = '2010-11-05'  #
pick_me = "monday"  #@
select_or_input = "app
#@markdown ---
```

View source notebook

Pausing output processing

Saving data to Google Drive

Saving data to Google Sheets

Saving data with gsutil

Saving data with the Cloud Storage Python API

Serving resources

Showing CV2 Images

Tagged Outputs

Third-party Jupyter widgets

Using BigQuery with Cloud API

Using BigQuery with Pandas API

Visualization: Bar Plot in Altair

Visualization: Histogram in Altair

Visualization: Interactive Brushing in Altair

Visualization: Interactive Scatter Plot in Altair

Visualization: Linked Brushing in Altair

Visualization: Linked Scatter-Plot and Histogr...

Visualization: Scatter Plot with Rolling Mean i...

Visualization: Stacked Histogram in Altair

Visualization: Time Series Line Plot in Altair

```
!pip install networkx

# Import libraries
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from community import community_louvain
%matplotlib inline

#Import data set
df = pd.read_csv('../input/book-recommender-dataset/ratings -
Copy.csv',low_memory = False)
df.head()

df = df.iloc[:50000]
df.count()

cleaned_book = df[['book_id','user_id','rating']]
cleaned_book['book_id'] = cleaned_book.book_id.astype(int)

grouped_book = cleaned_book.groupby(['book_id',
'user_id']).sum().reset_index()
grouped_book.rating.loc[grouped_book.rating == 0] = 1

grouped_purchased = grouped_book.query('rating > 0')

#Create a lookup table
item_lookup = df[['book_id', 'title']].drop_duplicates()
item_lookup['book_id'] = item_lookup.book_id.astype(str)

#Count number of products and number of customers in the reduced
dataset
no_books = len(grouped_purchased.book_id.unique())
no_users = len(grouped_purchased.user_id.unique())
print('Number of users in dataset:', no_users)
print('Number of books in dataset:', no_books)

#Turn raw data to pivot ('ratings' matrix)
ratings_new = grouped_purchased.pivot(index = 'user_id',
columns='book_id', values='rating').fillna(0).astype('int')
#Binarize the ratings matrix (indicate only if a customer has
purchased a product or not)
ratings_binary = ratings_new.copy()
ratings_binary[ratings_binary != 0] = 1

#Initialize zeros dataframe for product interactions
books_integer = np.zeros((no_books,no_books))

#Count how many times each product pair has been purchased
print('Counting how many times each pair of products has been
purchased...')
```

```
for i in range(no_books):
    for j in range(no_books):
        if i != j:
            df_ij = ratings_binary.iloc[:,[i,j]] #create a temporary
df with only i and j products as columns
            sum_ij = df_ij.sum(axis=1)
            pairings_ij = len(sum_ij[sum_ij == 2]) #if s1_ij == 2 it
means that both products were purchased by the same customer
            books_integer[i,j] = pairings_ij
            books_integer[j,i] = pairings_ij

#Count how many customers have purchased each item
print('Counting how many times each individual product has been
purchased...')
times_purchased = books_integer.sum(axis = 1)

#Construct final weighted matrix of item interactions
print('Building weighted product matrix...')
books_weighted = np.zeros((no_books,no_books))
for i in range(no_books):
    for j in range(no_books):
        if (times_purchased[i]+times_purchased[j]) !=0: #make sure you
do not divide with zero
            books_weighted[i,j] =
(books_integer[i,j])/(times_purchased[i]+times_purchased[j])

#Get list of item labels (instead of Codes)
nodes_codes = np.array(ratings_binary.columns).astype('str')
item_lookup_dict =
pd.Series(item_lookup.title.values,index=item_lookup.book_id).to_dict(
)
nodes_labels = [item_lookup_dict[code] for code in nodes_codes]

#Create Graph object using the weighted product matrix as adjacency
matrix
G = nx.from_numpy_matrix(books_weighted)
pos=nx.random_layout(G)
labels = {}
for idx, node in enumerate(G.nodes()):
    labels[node] = nodes_labels[idx]

nx.draw_networkx_nodes(G, pos , node_color="skyblue", node_size=30)
nx.draw_networkx_edges(G, pos,  edge_color='k', width= 0.3, alpha=
0.5)
nx.draw_networkx_labels(G, pos, labels, font_size=4)
plt.axis('off')
plt.show() # display

#Export graph to Gephi
H=nx.relabel_nodes(G,labels) #create a new graph with Description
```

```
labels and save to Gephi for visualizations
nx.write_gexf(H, "products.gexf")

#Find communities of nodes (products)
partition = community_louvain.best_partition(G, resolution = 1.5)
values = list(partition.values())

#Check how many communities were created
print('Number of communities:', len(np.unique(values)))

#Create dataframe with product description and community id
book_communities = pd.DataFrame(nodes_labels, columns =
['title_description'])
book_communities['community_id'] = values

book_communities.head()

#Lets take a peek at community 6
book_communities[book_communities['community_id']== 6].head(15)

import sys

#Lets now divide each element in products_weighted dataframe with the
maximum of each row.
#This will normalize values in the row and we can perceive it as the
possibility af a customer also buying
#product in column j after showing interest for the product in row i

#Turn into dataframe
books_weighted_pd = pd.DataFrame(books_weighted, columns =
nodes_labels)
books_weighted_pd.set_index(books_weighted_pd.columns, 'book',
inplace=True)

books_prob = books_weighted_pd.divide(books_weighted_pd.max(axis = 1),
axis = 0)

import pandas as pd

#Now lets select a hypothetical basket of goods (one or more products)
that a customer has already purchased or
#shown an interest for by clicking on an add or something, and then
suggest him relative ones
basket = ['Neuromancer']
#Also select the number of relevant items to suggest
no_of_suggestions = 10

all_of_basket = books_prob[basket]
all_of_basket = all_of_basket.sort_values(by = basket,
ascending=False)
suggestions_to_customer =
```

```
list(all_of_basket.index[:no_of_suggestions])

print("People have also read")
suggestions_list = pd.DataFrame(suggestions_to_customer, columns =
['Some suggestions for you'])
suggestions_list.head(no_of_suggestions + 1)
```