

A Python Code

A .1 Creation of one single dataset from the tsv imdb file

```
1
2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None #100000
6 title_basics = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
    title_basics.tsv",sep='\t',nrows=numberofrows,header=0)
7 title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
    title_principals.tsv",nrows=numberofrows,sep='\t',header=0)
8
9 keep_col = ["tconst","titleType","primaryTitle","originalTitle","startYear",
    "runtimeMinutes","genres"]
10 title_basics = title_basics[keep_col]
11 title_basics = title_basics[title_basics["titleType"].str.contains("movie")
    == True]
12
13 print(title_basics.head(3))
14
15 merged1=pd.merge(title_basics,title_principals,how='inner',on='tconst')
16 del title_basics,title_principals
17 print(merged1)
18
19 name_basics=pd.read_csv("/content/drive/MyDrive/Dataset/Original/name_basics
    .tsv",sep='\t',nrows=numberofrows,header=0)
20
21 merged2=pd.merge(merged1,name_basics,how='inner',on='nconst')
22 del merged1
23 merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear","
    knownForTitles","primaryProfession"])
24 del name_basics
25 print(merged2)
26
27 category=merged2.groupby('tconst')['category'].apply(list).reset_index(name=
    'category')
28 job=merged2.groupby('tconst')['job'].apply(list).reset_index(name='job')
29 characters=merged2.groupby('tconst')['characters'].apply(list).reset_index(
    name='characters')
30 primaryName=merged2.groupby('tconst')['primaryName'].apply(list).reset_index
    (name='primaryName')
31 result=merged2.drop_duplicates(subset=['tconst'])
32 result=result.drop(['category'],axis=1).drop(['job'],axis=1).drop(['
    characters'],axis=1).drop(['primaryName'],axis=1)
33 result=result.merge(category,on='tconst').merge(job,on='tconst').merge(
    characters,on='tconst').merge(primaryName,on='tconst')
34 print(result)
35
36 result.to_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",index=
    False)
```

A .2 Creation of one single dataset from the csv kaggle file

```
1
2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
```

```

5  numberofrows=None #100000
6  movies = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_movies.
    csv",nrows=numberofrows,header=0)
7  reviews = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
    rotten_reviews.csv",nrows=numberofrows,header=0)
8  to_keep = ["rotten_tomatoes_link", "movie_title", "production_company", "
    critics_consensus",
9             "tomatometer_status", "tomatometer_rating", "tomatometer_count",
10            "audience_status", "audience_rating", "audience_count",
11            "tomatometer_top_critics_count", "tomatometer_fresh_critics_count
    ",
12            "tomatometer_rotten_critics_count"]
13
14  movies = movies[to_keep]
15  to_drop = ["publisher_name"]
16  reviews=reviews.drop(columns=to_drop)
17
18  merged=pd.merge(movies, reviews, how='inner', on="rotten_tomatoes_link")
19  print(merged)
20
21  categories = {}
22  arr = ["critic_name", "top_critic", "review_type", "review_score", "
    review_date", "review_content"]
23  for x in arr:
24      categories[x]=merged.groupby('rotten_tomatoes_link')[x].apply(list).
        reset_index(name=x)
25
26  result=merged.drop_duplicates(subset=['rotten_tomatoes_link'])
27  for x in arr:
28      result=result.drop([x], axis=1)
29  for x in arr:
30      result=result.merge(categories[x], on='rotten_tomatoes_link')
31
32  print(result)
33
34  result.to_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv", index=
    False)

```

A .3 Merging of the file generated in the previous script

```

1
2  import pandas as pd
3  from google.colab import drive
4  drive.mount('/content/drive')
5  numberofrows=None
6  imdb = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",
    nrows=numberofrows,header=0)
7  rotten = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",
    nrows=numberofrows,header=0)
8
9  merged = {}
10 choose = ['primaryTitle', 'originalTitle']
11 rowHeadDataset = 20
12 for x in choose:
13     merged[x]=pd.merge(imdb, rotten, how='inner', left_on=x, right_on='
        movie_title')
14     print(x)
15     merged[x]=merged[x].drop_duplicates(subset=[x])
16     merged[x]=merged[x].drop(columns=['titleType', 'tomatometer_count', '
        tomatometer_top_critics_count'])
17     merged[x]=merged[x].rename(columns={'startYear': 'year'})

```

```

18 merged[x]=merged[x].drop(columns=['rotten_tomatoes_link', 'movie_title']+
   j for j in choose if j!=x])
19 print(len(pd.unique(merged[x][x])))
20 print(list(merged[x]))
21 print("=====")
22 merged[x].to_csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv",
   index=False)
23 merged[x]=merged[x].head(rowHeadDataset)
24 merged[x].to_csv(f"/content/drive/MyDrive/Dataset/headDataset{x}.csv",
   index=False)

```

A .4 Collapsing different rows in a single one generating an array for personnel field

```

1
2 import pandas as pd
3 from ast import literal_eval
4 from google.colab import drive
5 drive.mount('/content/drive')
6
7 numberofrows=None
8 df = pd.read_csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.
   csv",nrows=numberofrows,header=0)
9
10 #print([x.split(',') for x in df['genres']])
11 print(df)
12
13 col = ["primaryName","category","job","characters"]
14 col1 = ["critic_name","top_critic","review_type","review_score","review_date",
   "","review_content"]
15
16 df['personnel'] = ""
17 df['review'] = ""
18
19 for row in range(df[col[0]].size):
20     it = df['genres'][row]
21     df['genres'][row] = [',' + x + ',' for x in it.split(',') if it != '\\N']
   else []
22     tmp = []
23     for c in col:
24         tmp.append({c:eval(df[c][row])})
25     res = []
26     for c in range(len(tmp[0][col[0]])):
27         res.append({})
28     for i, j in zip(col, tmp):
29         for idx, x in enumerate(j[i]):
30             #print(i, idx, x)
31             if x != '\\N':
32                 if i == 'characters':
33                     x = eval(x)
34                     res[idx][',' + i + ','] = '"' + str(x).replace("'", "##single-quote
   ##").replace('"', "##double-quote##") + '"'
35 df['personnel'][row] = list(res)
36 #print(res)
37 ###
38 tmp = []
39 for c in col1:
40     to_eval = df[c][row].replace('nan', 'None')
41     arr = eval(to_eval)
42     if c == "review_date":

```

```

43     for i, elem in enumerate(arr):
44         arr[i] = elem + "T00:00:00.000+00:00"
45     tmp.append({c:arr})
46     #print(tmp)
47 res = []
48 for c in range(len(tmp[0][col1[0]])):
49     res.append({})
50 for i, j in zip(col1, tmp):
51     for idx, x in enumerate(j[i]):
52         #print(i, idx, x)
53         if x != '\N':
54             res[idx]["," + i + ","] = "," + str(x).replace("True", "true").
             replace("False", "false").replace("'", "##single-quote##").replace('"', "
             ##double-quote##") + ","
55 df['review'][row] = list(res)
56 #df['review'][row] = eval(str(res))
57 #print(res)
58 #print()
59
60 df=df.drop(columns=col)
61 df=df.drop(columns=col1)
62 df=df.drop(columns=['tconst'])
63
64 print(df["review"][0])
65
66 it = df['personnel'][0]#[4]['review_content']
67 print(type(it))
68 print(it)
69
70 df.to_csv("/content/drive/MyDrive/Dataset/
        movieCollectionEmbeddedReviewPersonnel.csv",index=False)
71 df = df.head(20)
72 df.to_csv("/content/drive/MyDrive/Dataset/
        headmovieCollectionEmbeddedReviewPersonnel.csv",index=False)

```

A .5 Generates a hashed password for all the users

```

1 import hashlib
2 #from pprint import pprint as print
3 from pymongo import MongoClient
4
5 def get_database():
6     CONNECTION_STRING = "mongodb://localhost:27017"
7     client = MongoClient(CONNECTION_STRING)
8     return client['rottenMovies']
9
10 if __name__ == "__main__":
11     dbname = get_database()
12     collection = dbname['user']
13     total = collection.count_documents({})
14     for i, user in enumerate(collection.find()):
15         all_reviews = user['last_3_reviews']
16         sorted_list = sorted(all_reviews, key=lambda t: t['review_date'])
17         [-3:]
18
19         hashed = hashlib.md5(user["username"].encode()).hexdigest()
20
21         newvalues = { "$set": { 'password': hashed, 'last_3_reviews':
sorted_list } }
22         filter = { 'username': user['username']}
23         collection.update_one(filter, newvalues)

```

```

23         print(f"{i/total:}%\r", end='')
24     print()

```

A .6 Generates the graph database

```

1  from pymongo import MongoClient
2  from neo4j import GraphDatabase
3  from random import randint, shuffle
4
5  def get_database():
6      CONNECTION_STRING = "mongodb://localhost:27017"
7      client = MongoClient(CONNECTION_STRING)
8      return client['rottenMovies']
9
10 class Neo4jGraph:
11
12     def __init__(self, uri, user, password):
13         self.driver = GraphDatabase.driver(uri, auth=(user, password),
14         database="rottenmoviesgraphdb")
15
16     def close(self):
17         self.driver.close()
18
19     def addUser(self, uid, name, isTop):
20         with self.driver.session() as session:
21             if isTop:
22                 result = session.execute_write(self._addTopCritic, uid, name)
23             else:
24                 result = session.execute_write(self._addUser, uid, name)
25
26     def addMovie(self, mid, title):
27         with self.driver.session() as session:
28             result = session.execute_write(self._addMovie, mid, title)
29
30     def addReview(self, name, mid, freshness, content, date):
31         with self.driver.session() as session:
32             result = session.execute_write(self._addReview, name, mid,
33             freshness, content, date)
34
35     def addFollow(self, uid, cid):
36         with self.driver.session() as session:
37             result = session.execute_write(self._addFollow, uid, cid)
38
39     @staticmethod
40     def _addUser(tx, uid, name):
41         query = "CREATE (n:User{id:\\"" + str(uid) + "\", name:\\"" + name.
42         replace("'", '\\\"') + "\"})"
43         #print(query)
44         result = tx.run(query)
45
46     @staticmethod
47     def _addTopCritic(tx, cid, name):
48         query = "CREATE (m:TopCritic{id:\\"" + str(cid) + "\", name:\\"" + name
49         .replace("'", '\\\"') + "\"})"
50         #print(query)
51         result = tx.run(query)
52
53     @staticmethod
54     def _addMovie(tx, mid, title):

```

```

51     query = "CREATE(o:Movie{id:\\"" + str(mid) + "\"}, title:\\"" + title.
replace("'", '\\"') + "\"})"
52     #print(query)
53     result = tx.run(query)
54
55     @staticmethod
56     def _addReview(tx, name, mid, freshness, content, date): # date in
format YYYY-mm-dd, freshness in [TRUE, FALSE]
57     query = "MATCH(n{name:\\"" + str(name).replace("'", '\\"') + "\"}), (
m:Movie{id:\\"" + str(mid) + "\"}) CREATE (n)-[r:REVIEWED{freshness:\\"" +
freshness + "\", date:date('\\"" + date + "\"'), content:\\"" + content.replace(
'", '\\"') + "\"}]->(m)"
58     #print(query)
59     result = tx.run(query)
60
61     @staticmethod
62     def _addFollow(tx, uid, cid):
63     query = "MATCH(n:User{id:\\"" + str(uid) + "\"}), (m:TopCritic{id:\\""
+ str(cid) + "\"}) CREATE (n)-[r:FOLLOWS]->(m)"
64     #print(query)
65     result = tx.run(query)
66
67 if __name__ == "__main__":
68     # dbs initialization
69     dbname = get_database()
70     graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")
71
72     # user creation
73     collection = dbname['user']
74     total = collection.count_documents({})
75     print(f"user {total} = ")
76     for i, user in enumerate(list(collection.find({}, {"_id":1, "username"
:1, "date_of_birth":1}))) :
77         graphDB.addUser(user['_id'], user['username'], 'date_of_birth' not
in user)
78         if not i%100:
79             print(f" {(i+1)/total:%}\r", end='')
80
81     # movie creation and review linking
82     collection = dbname['movie']
83     total = collection.count_documents({})
84     print(f"\nmovie {total} = ")
85     for i, movie in enumerate(list(collection.find({}, {"_id":1, "
primaryTitle":1, "review":1}))) :
86         graphDB.addMovie(movie['_id'], movie['primaryTitle'])
87         movie['review'] = list({v['critic_name']:v for v in movie['review'
]}.values()) # make unique reviews per critic
88         for rev in movie['review']:
89             graphDB.addReview(rev['critic_name'], movie['_id'], {"Fresh":
TRUE", "Rotten":FALSE"}[rev['review_type']], str(rev['review_content']
[:15], str(rev['review_date'][:10])
90             print(f" {(i+1)/total:%}\r", end='')
91
92     # follow linking
93     collection = dbname['user']
94     uids = [x['_id'] for x in list(collection.find({"date_of_birth":{"
$exists":True}}, {"_id":1}))]
95     cids = [x['_id'] for x in list(collection.find({"date_of_birth":{"
$exists":False}}, {"_id":1}))]
96     total = len(uids)
97     print(f"\nfollow {total} = ")

```

```
98     for i, user in enumerate(uids):
99         shuffle(cids)
100         for j in range(randint(0, 20)):
101             graphDB.addFollow(user, cids[j])
102             print(f"{i/total:%}\r", end='')
103
104 graphDB.close()
```

B Mongosh scripts

B.1 Perform the escape on the string fields

```
1 db.movie.find().forEach(  
2   x => {  
3     print(x.primaryTitle);  
4     x.review = JSON.parse(  
5       x.review.replaceAll('"\'', '')  
6         .replaceAll('\\"', '')  
7         .replaceAll('"false"', 'false')  
8         .replaceAll('"true"', 'true')  
9         .replaceAll('"None"', 'null')  
10        .replaceAll(/\\x\d{2}/g, '')  
11        .replaceAll("##single-quote##", '\')  
12        .replaceAll("##double-quote##", '\\')  
13        .replaceAll("\\x", "x")  
14      );  
15     x.personnel = JSON.parse(  
16       x.personnel.replaceAll('"\'', '')  
17         .replaceAll('\\"', '')  
18         .replaceAll('"None"', 'null')  
19         .replaceAll("##single-quote##", '\')  
20         .replaceAll("##double-quote##", '\\')  
21         .replaceAll('"[\'', '[')  
22         .replaceAll('"[\\', '[')  
23         .replaceAll('\'"]', '"]')  
24         .replaceAll('\\"]', '"]')  
25         .replaceAll('/(\\[[^:]*\\')\\', '\\'([~:]*\\')/g, '$1', '$2')  
26         .replaceAll('/(\\[[^:]*\\')\\', '\\'([~:]*\\')/g, '$1', '$2')  
27         .replaceAll('/(\\[[^:]*\\')\\', '\\'([~:]*\\')/g, '$1', '$2')  
28      );  
29     x.genres = JSON.parse(  
30       x.genres = x.genres.replaceAll('"\'', '')  
31         .replaceAll('\\"', '')  
32         .replaceAll('"None"', 'null')  
33         .replaceAll("##single-quote##", '\')  
34         .replaceAll("##double-quote##", '\\')  
35      );  
36     db.movie.updateOne(  
37       {"_id": x._id},  
38       {$set:  
39         {  
40           "review": x.review,  
41           "personnel": x.personnel,  
42           "genres": x.genres,  
43           "runtimeMinutes": parseInt(x.runtimeMinutes),  
44           "year": parseInt(x.year),  
45           "tomatometer_rating": parseFloat(x.tomatometer_rating),  
46           "audience_rating": parseFloat(x.audience_rating),  
47           "audience_count": parseFloat(x.audience_count),  
48           "tomatometer_fresh_critics_count": parseInt(x.  
49 tomatometer_fresh_critics_count),  
50           "tomatometer_rotten_critics_count": parseInt(x.  
51 tomatometer_rotten_critics_count)  
52         }  
53       }  
54     );  
55   }
```



```

54     }
55 );

```

B .2 Normalize the date field in the DB

```

1  total = db.movie.find().count();
2  i = 0;
3  db.movie.find().forEach(
4      x => {
5          print(x.primaryTitle);
6          x.review.forEach(rev =>{
7              if(typeof (rev.review_date) === "string" ){
8                  db.movie.updateOne(
9                      {primaryTitle: x.primaryTitle },
10                     { $set: { "review.$[elem].review_date" : new Date(rev.
review_date) } },
11                     { arrayFilters: [ { "elem.critic_name": rev.critic_name
} ] } )
12             )
13         }
14     })
15     print(100*i++/total);
16 });

```

B .3 Create a new collection for the user based on the data present in the movie collection

```

1  total = db.runCommand({ distinct: "movie", key: "review.critic_name", query:
    {"review.critic_name":{"$ne:null}}}).values.length
2  i = 0;
3  db.runCommand(
4  { distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
:{"$ne:null}}}).values.forEach(
5      (x) => {
6          review_arr = []
7          movie_arr = []
8          is_top = false
9          db.movie.aggregate(
10             [
11                 { $project:
12                     {
13                         index: { $indexOfArray: ["$review.critic_name", x]},
14                         primaryTitle: 1
15                     }},
16                 {$match:{index:{$gt:-1}}}
17             ]
18             ).forEach(
19                 y => {
20                     tmp = db.movie.aggregate([
21                         {
22                             $project:
23                             {
24                                 top_critic: {
25                                     $arrayElemAt: ["$review.top_critic", y.index
]
26                             },
27                             primaryTitle: y.primaryTitle,
28                             review_type: {
29                                 $arrayElemAt: ["$review.review_type", y.
index]

```

```

30         },
31         review_score: {
32             $arrayElemAt: ["$review.review_score", y.
index]
33         },
34         review_date: {
35             $arrayElemAt: ["$review.review_date", y.
index]
36         },
37         review_content: {
38             $arrayElemAt: ["$review.review_content", y.
index]
39         }
40     }
41 },
42 {
43     $match: {_id: {$eq: y._id}}
44 }
45 ]).toArray()[0];
46 is_top |= tmp.top_critic;
47 review_arr.push(tmp)
48 //movie_arr.push(tmp._id)
49 movie_arr.push({"movie_id": tmp._id, "primaryTitle": y.
primaryTitle, "review_index": y.index})
50 })
51
52 name_parts = x.split(/\s/)
53 first_name = name_parts.splice(0, 1)[0]
54 last_name = name_parts.join(' ')
55
56 print(100*i++/total, x, is_top)
57 //print(first_name, ':', last_name)
58 //print(review_arr)
59 //print(movie_arr)
60 db.user.insertOne(
61     {
62         "username": x,
63         "password": "",
64         "first_name": first_name,
65         "last_name": last_name,
66         "registration_date": new Date("2000-01-01"),
67         "last_3_reviews": review_arr,
68         "reviews" : movie_arr
69     }
70 );
71 if (!is_top){
72     db.user.updateOne(
73         {"username": x},
74         {$set:
75             {"date_of_birth": new Date("1970-07-20")}}
76     )
77 }
78
79 print("=====")
80 }
81 )

```

C matematica

$$x^2 - 5x + 6 = 0 \quad (1)$$

