labassignment7

August 2, 2024

1 Lab Assignment 7: Database Queries

1.1 DS 6001: Practice and Application of Data Science

1.1.1 Instructions

Please answer the following questions as completely as possible using text, code, and the results of code as needed. Format your answers in a Jupyter notebook. To receive full credit, make sure you address every part of the problem, and make sure your document is formatted in a clean and professional way.

1.1.2 Problem 0

Import the following libraries, load the `.env` file where you store your passwords (see the notebook for module 4 for details), and turn off the error tracebacks to make errors easier to read:

[ ]:
```
import numpy as np
import pandas as pd
import sys
import os
import requests
import psycopg2
import pymongo
import json
from bson.json_util import dumps, loads
from sqlalchemy import create_engine
import dotenv
from pymongo import MongoClient


# change to the directory where your .env file is
os.chdir(r"C:\Users\qaism\OneDrive - University of␣
↪Virginia\Documents\GitHub\MSDS\ds6001databases")

dotenv.load_dotenv() # register the .env file where passwords are stored
sys.tracebacklimit = 0 # turn off the error tracebacks
```

1.1.3 Problem 1

For this problem, we will be building a PostgreSQL database that contains the collected works of Shakespeare.

The data were collected by Catherine Devlin from the repository at https://opensourceshakespeare.org/. The database will have four tables, one representing works by Shakespeare, one for characters that appear in Shakespeare's plays, one for chapters (this is, scenes within acts), and one for paragraphs (that is, lines of dialogue). The data to populate these four tables are here:

```
[ ]: works = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localdata/
      ↪Works.csv")
     characters = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/
      ↪localdata/Characters.csv")
     chapters = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localdata/
      ↪Chapters.csv")
     paragraphs = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/
      ↪localdata/Paragraphs.csv")
```

In PostgreSQL, it is best practice to convert all column names to lower-case, as case sensitive column names will require extraneous double-quotes in any query. We first convert the column names in all four dataframe to lowercase:

```
[ ]: works.columns = works.columns.str.lower()
     characters.columns = characters.columns.str.lower()
     chapters.columns = chapters.columns.str.lower()
     paragraphs.columns = paragraphs.columns.str.lower()
```

You will build a database and populate it with these data. The ER diagram for the database is:

There's no codebook, unfortunately, but the values in the columns are mostly self-explanatory:

```
[ ]: works.head()
```

```
[ ]:        workid                     title                     longtitle  \
     0     12night             Twelfth Night  Twelfth Night, Or What You Will
     1    allswell  All's Well That Ends Well       All's Well That Ends Well
     2  antonycleo        Antony and Cleopatra         Antony and Cleopatra
     3  asyoulikeit              As You Like It               As You Like It
     4 comedyerrors           Comedy of Errors         The Comedy of Errors

        date genretype  notes     source  totalwords  totalparagraphs
     0  1599         c    NaN       Moby       19837             1031
     1  1602         c    NaN       Moby       22997             1025
     2  1606         t    NaN       Moby       24905             1344
     3  1599         c    NaN  Gutenberg       21690              872
     4  1589         c    NaN       Moby       14692              661
```

```
[ ]: characters.head()
```

[ ]:
```
              charid           charname              abbrev        works  \
0  1apparition-mac   First Apparition    First Apparition      macbeth
1         1citizen      First Citizen       First Citizen  romeojuliet
2     1conspirator  First Conspirator   First Conspirator   coriolanus
3   1gentleman-oth    First Gentleman     First Gentleman      othello
4            1goth         First Goth          First Goth        titus

  description  speechcount
0         NaN          1.0
1         NaN          3.0
2         NaN          3.0
3         NaN          1.0
4         NaN          4.0
```

[ ]:
```
chapters.head()
```

[ ]:
```
    workid  chapterid  section  chapter            description
0  12night    18704.0      1.0      1.0  DUKE ORSINO's palace.
1  12night    18705.0      1.0      2.0         The sea-coast.
2  12night    18706.0      1.0      3.0         OLIVIA'S house.
3  12night    18707.0      1.0      4.0  DUKE ORSINO's palace.
4  12night    18708.0      1.0      5.0         OLIVIA'S house.
```

[ ]:
```
paragraphs.head()
```

[ ]:
```
    workid paragraphid paragraphnum  charid  \
0  12night      630863            3     xxx
1  12night      630864            4  ORSINO
2  12night      630865           19   CURIO
3  12night      630866           20  ORSINO
4  12night      630867           21   CURIO

                                          plaintext  \
0  [Enter DUKE ORSINO, CURIO, and other Lords; Mu…
1  If music be the food of love, play on;\n[p]Giv…
2                       Will you go hunt, my lord?\n
3                                     What, Curio?\n
4                                       The hart.\n

                                       phonetictext  \
0        ENTR TK ORSN KR ANT OOR LRTS MSXNS ATNTNK
1  IF MSK B 0 FT OF LF PL ON JF M EKSSS OF IT OT …
2                                   WL Y K HNT M LRT
3                                              HT KR
4                                              0 HRT

                                          stemtext paragraphtype  section  \
```

```
0  enter duke orsino curio and other lord musicia…            b       1.0
1  if music be the food of love plai on give me e…            b       1.0
2                        will you go hunt my lord               b       1.0
3                                      what curio               b       1.0
4                                        the hart               b       1.0

   chapter  charcount  wordcount
0      1.0       65.0        9.0
1      1.0      646.0      114.0
2      1.0       27.0        6.0
3      1.0       13.0        2.0
4      1.0       10.0        2.0
```

Part a Connect to your local PostgreSQL server (take steps to hide your password!), create a new database for the Shakespeare data, use `create_engine()` from `sqlalchemy` to connect to the database, and create the works, characters, chapters, and paragraphs tables populated with the data from the four dataframes shown above. [2 points]

[ ]:
```python
db_name = 'shakespeare_db'
db_user = os.getenv('POSTGRES_USER')
db_password = os.getenv('POSTGRES_PASSWORD')
db_host = os.getenv('POSTGRES_HOST')
db_port = os.getenv('POSTGRES_PORT')

conn = psycopg2.connect(
    dbname='postgres',
    user=db_user,
    password=db_password,
    host=db_host,
    port=db_port
)
conn.autocommit = True
cursor = conn.cursor()

cursor.execute(f"CREATE DATABASE {db_name}")
cursor.close()
conn.close()
```

```
---------------------------------------------------------------------------
DuplicateDatabase                         Traceback (most recent call last)
Cell In[9], line 17
     14 conn.autocommit = True
     15 cursor = conn.cursor()
---> 17 cursor.execute(f"CREATE DATABASE {db_name}")
     18 cursor.close()
     19 conn.close()
```

```
DuplicateDatabase: database "shakespeare_db" already exists
```

[ ]:
```
engine = create_engine(f'postgresql://{db_user}:{db_password}@{db_host}:
↪{db_port}/{db_name}')

works.to_sql('works', engine, if_exists='replace', index=False)
characters.to_sql('characters', engine, if_exists='replace', index=False)
chapters.to_sql('chapters', engine, if_exists='replace', index=False)
paragraphs.to_sql('paragraphs', engine, if_exists='replace', index=False)
```

[ ]:
```
475
```

Part b Write a query to display `title`, `date`, and `totalwords` from the `works` table. Rename `date` to `year`, and sort the output by `totalwords` in descending order. Also create a new column called `era` which is equal to "early" for works created before 1600, "middle" for works created between 1600 and 1607, and "late" for works created after 1607. Finally, display only the 7th through 11th rows of the output data. [1 point]

[ ]:
```
query = """
SELECT
    title,
    date AS year,
    totalwords,
    CASE
        WHEN date < 1600 THEN 'early'
        WHEN date BETWEEN 1600 AND 1607 THEN 'middle'
        ELSE 'late'
    END AS era
FROM works
ORDER BY totalwords DESC
OFFSET 6 LIMIT 5;
"""
result = pd.read_sql(query, engine)
result
```

[ ]:

	title	year	totalwords	era
0	King Lear	1605	26119	middle
1	Troilus and Cressida	1601	26089	middle
2	Henry IV, Part II	1597	25692	early
3	Henry VI, Part II	1590	25411	early
4	The Winter's Tale	1610	24914	late

Part c The `genretype` column in the "works" table designates five types of Shakespearean work:

- `t` is a tragedy, such as *Romeo and Juliet* and *Hamlet*
- `c` is a comedy, such as *A Midsummer Night's Dream* and *As You Like It*

- `h` is a history, such as *Henry V* and *Richard III*
- `s` refers to Shakespeare's sonnets
- `p` is a narrative (non-sonnet) poem, such as *Venus and Adonis* and *Passionate Pilgrim*

Write a query that generates a table that reports the average number of words in Shakepeare's works by genre type. Display the genre type and the average wordcount within genre, use appropriate aliases, and sort by the average in descending order. [1 point]

[ ]:
```
query_avg = """
SELECT
    genretype AS genre,
    AVG(totalwords) AS average_wordcount
FROM works
GROUP BY genretype
ORDER BY average_wordcount DESC;
"""

result_c = pd.read_sql(query_avg, engine)
result_c.head()
```

[ ]:

	genre	average_wordcount
0	h	24236.000000
1	t	23817.363636
2	c	20212.071429
3	s	17515.000000
4	p	6181.800000

Part d Use a query to generate a table that contains the text of Hamlet's (the character, not just the play) longest speech, and use the `print()` function to display this text. [1 point]

[ ]:
```
query_d = """
SELECT
    plaintext
FROM paragraphs
WHERE charid = 'hamlet'
ORDER BY wordcount DESC
LIMIT 1;
"""

result_d = pd.read_sql(query_d, engine)
print(result_d['plaintext'].iloc[0])
```

```
Well said, old mole! Canst work i' th' earth so fast?
[p]A worthy pioner! Once more remove, good friends.
```

1.1.4 Part e

Many scenes in Shakespeare's works take place in palaces or castles. Use a query to create a table that lists all of the chapters that take place in a palace. Include the work's title, the section (renamed to "act"), the chapter (renamed to "scene"), and the description of these chapters. The setting of each scene is listed in the `description` column of the "chapters" table. [Hint: be sure to account for case sensitivity] [2 points]

[ ]:
```
query_e = """
SELECT
    w.title,
    c.section AS act,
    c.chapter AS scene,
    c.description AS description
FROM chapters c
JOIN works w ON c.workid = w.workid
WHERE LOWER(c.description) LIKE '%%palace%%';
"""

result_e = pd.read_sql_query(query_e, engine)
result_e
```

[ ]:
```
                         title  act  scene  \
0                Twelfth Night  2.0    4.0
1                Twelfth Night  1.0    4.0
2                Twelfth Night  1.0    1.0
3    All's Well That Ends Well  5.0    3.0
4    All's Well That Ends Well  5.0    2.0
..                         ...  ...    ...
120            The Winter's Tale  5.0    1.0
121            The Winter's Tale  4.0    2.0
122            The Winter's Tale  2.0    3.0
123            The Winter's Tale  2.0    1.0
124            The Winter's Tale  1.0    1.0

                                description
0                     DUKE ORSINO's palace.
1                     DUKE ORSINO's palace.
2                     DUKE ORSINO's palace.
3              Rousillon. The COUNT's palace.
4    Rousillon. Before the COUNT's palace.
..                                      ...
120              A room in LEONTES' palace.
121       Bohemia. The palace of POLIXENES.
122              A room in LEONTES' palace.
123              A room in LEONTES' palace.
124          Antechamber in LEONTES' palace.

[125 rows x 4 columns]
```

1.1.5 Part f

Create a table that lists characters, the plays that the characters appear in, the number of speeches the character gives, and the average length of the speeches that the character gives. Display the character description and the work title, not the ID values. Sort the table by average speech length, and restrict the table to only those characters that give at least 20 speeches. [Hint: you will need to use a subquery.] [2 points]

[ ]:
```
query_f = """
SELECT
    c.charname AS character,
    w.title AS play,
    c.description,
    speech_stats.speech_count,
    speech_stats.avg_speech_length
FROM characters c
JOIN (
    SELECT
        charid,
        workid,
        COUNT(*) AS speech_count,
        AVG(wordcount) AS avg_speech_length
    FROM paragraphs
    GROUP BY charid, workid
    HAVING COUNT(*) >= 20
) AS speech_stats
ON c.charid = speech_stats.charid
JOIN works w
ON speech_stats.workid = w.workid
ORDER BY speech_stats.avg_speech_length DESC;
"""

result_f = pd.read_sql_query(query_f, engine)
print(result_f)
```

```
               character                  play  \
0                   Poet               Sonnets
1               Henry IV        Henry IV, Part I
2                   Poet     Passionate Pilgrim
3               Henry IV       Henry IV, Part II
4         King Richard II            Richard II
..                   …                     …
447   (stage directions)        Romeo and Juliet
448   (stage directions)        The Winter's Tale
449   (stage directions)     Troilus and Cressida
450   (stage directions)           Twelfth Night
451   (stage directions)  Two Gentlemen of Verona

                           description  speech_count  avg_speech_length
```

```
0    the voice of Shakespeare's poetry          154          113.733766
1                      King of England           30           85.900000
2    the voice of Shakespeare's poetry           43           72.651163
3                      King of England           34           71.794118
4                      king of England           98           61.765306
..                                   …            …                 …
447                               None          146            3.342466
448                               None           67            3.238806
449                               None          158            2.936709
450                               None          108            2.935185
451                               None           86            2.500000

[452 rows x 5 columns]
```

1.1.6 Part g

Which Shakepearean works do not contain any scenes in a palace or a castle? Use a query that displays the title, genre type, and publication date of works that do not contain any scenes that take place in a palace or castle. [Hint: use your work in part e as a starting point. You will need a subquery, and you will need to think carefully about the type of join that you need to perform.][2 points]

[ ]:
```
query_g = """
SELECT
    w.title,
    w.genretype,
    w.date AS publication_date
FROM works w
WHERE w.workid NOT IN (
    SELECT DISTINCT c.workid
    FROM chapters c
    WHERE LOWER(c.description) LIKE '%%palace%%'
    OR LOWER(c.description) LIKE '%%castle%%'
);
"""

result_g = pd.read_sql_query(query_g, engine)
print(result_g)
```

```
                     title genretype  publication_date
0               Coriolanus         t              1607
1            Julius Caesar         t              1599
2         Lover's Complaint         p              1609
3      Love's Labour's Lost         c              1594
4       Merchant of Venice         c              1596
5    Merry Wives of Windsor         c              1600
6    Much Ado about Nothing         c              1598
7        Passionate Pilgrim         p              1598
```

```
8   Phoenix and the Turtle         p            1601
9          Rape of Lucrece         p            1594
10        Romeo and Juliet         t            1594
11                 Sonnets         s            1609
12     Taming of the Shrew         c            1593
13                 Tempest         c            1611
14         Timon of Athens         t            1607
15        Venus and Adonis         p            1593
```

1.1.7 Problem 2

The following file contains JSON formatted data of the official English-language translations of every constitution currently in effect in the world:

```
[ ]: const = requests.get("https://github.com/jkropko/DS-6001/raw/master/localdata/
      ↪const.json")
     const_json = json.loads(const.text)
     pd.DataFrame.from_records(const_json)
```

```
[ ]:                                              text             country  \
0    'Afghanistan 2004     Preamble    \nIn the na…        Afghanistan
1    'Albania 1998 (rev. 2012)     Preamble    \nWe…            Albania
2    'Andorra 1993     Preamble    \nThe Andorran P…            Andorra
3    'Angola 2010     Preamble    \nWe, the people …             Angola
4    'Antigua and Barbuda 1981     Preamble    \nWH…  Antigua and Barbuda
..                                                …                    …
140  'Uzbekistan 1992 (rev. 2011)     Preamble    \…         Uzbekistan
141  'Viet Nam 1992 (rev. 2013)     Preamble    \nI…           Viet Nam
142  'Yemen 1991 (rev. 2001)     PART ONE. THE FOUN…              Yemen
143  'Zambia 1991 (rev. 2009)     Preamble    \nWE,…             Zambia
144  'Zimbabwe 2013     Preamble    \nWe the people…           Zimbabwe

     adopted  revised  reinstated  democracy
0       2004      NaN         NaN   0.372201
1       1998   2012.0         NaN   0.535111
2       1993      NaN         NaN        NaN
3       2010      NaN         NaN   0.315043
4       1981      NaN         NaN        NaN
..       …        …           …          …
140     1992   2011.0         NaN   0.195932
141     1992   2013.0         NaN   0.251461
142     1991   2001.0         NaN   0.125708
143     1991   2009.0         NaN   0.405497
144     2013      NaN         NaN   0.315359

[145 rows x 6 columns]
```

The text of the constitutions are available from the Wolfram Data Repository. I also included scores that represent the level of democractic quality in each country as of 2016. These scores are compiled by the Varieties of Democracy (V-Dem) project. Higher scores indicate greater levels of democratic openness and competition.

Part a Connect to your local MongoDB server and create a new collection for the constitution data. Use `.delete_many({})` to remove any existing data from this collection, and insert the data in `const_json` into this collection. [2 points]

[ ]:
```
mongo_host = os.getenv('MONGO_HOST')
mongo_port = int(os.getenv('MONGO_PORT'))
mongo_user = os.getenv('MONGO_INITDB_ROOT_USERNAME')
mongo_password = os.getenv('MONGO_INITDB_ROOT_PASSWORD')

try:
    client = MongoClient(mongo_host, mongo_port, username=mongo_user,␣
 ↪password=mongo_password, authSource='admin', serverSelectionTimeoutMS=5000)
    client.server_info()

    db = client['constitution_db']
    collection = db['constitutions']

    collection.delete_many({})
    collection.insert_many(const_json)

    print("Data inserted successfully into MongoDB collection.")

except pymongo.errors.ServerSelectionTimeoutError as err:
    print(f"Error: Could not connect to MongoDB server: {err}")
except pymongo.errors.OperationFailure as err:
    print(f"Error: Authentication failed: {err}")
```

```
Data inserted successfully into MongoDB collection.
```

Part b Use MongoDB queries and the `dumps()` and `loads()` functions from the `bson` package to produce dataframes with the following restrictions:

- The country, adoption year, and democracy features (and not `_id`, text, revised, or reinstated) for countries with constitutions that were written after 1990
- The country, adoption year, and democracy features (and not `_id`, text, revised, or reinstated) for countries with constitutions that were written after 1990 AND have a democracy score of less than 0.5
- The country, adoption year, and democracy features (and not `_id`, text, revised, or reinstated) for countries with constitutions that were written after 1990 OR have a democracy score of less than 0.5

[1 point]

[ ]:
```
query1 = {"adopted": {"$gt": 1990}}
projection = {"_id": 0, "country": 1, "adopted": 1, "democracy": 1}
results1 = collection.find(query1, projection)
```

```
df1 = pd.DataFrame(loads(dumps(results1)))
df1
```

[ ]:

```
        country  adopted  democracy
0   Afghanistan     2004   0.372201
1       Albania     1998   0.535111
2       Andorra     1993        NaN
3        Angola     2010   0.315043
4       Armenia     1995   0.393278
..          …       …         …
66   Uzbekistan     1992   0.195932
67     Viet Nam     1992   0.251461
68        Yemen     1991   0.125708
69       Zambia     1991   0.405497
70     Zimbabwe     2013   0.315359

[71 rows x 3 columns]
```

[ ]:

```
query2 = {"$and": [{"adopted": {"$gt": 1990}}, {"democracy": {"$lt": 0.5}}]}
results2 = collection.find(query2, projection)
df2 = pd.DataFrame(loads(dumps(results2)))
df2
```

[ ]:

```
                              country  adopted  democracy
0                         Afghanistan     2004   0.372201
1                              Angola     2010   0.315043
2                             Armenia     1995   0.393278
3                             Belarus     1994   0.289968
4              Bosnia and Herzegovina     1995   0.338267
5                            Cambodia     1993   0.313738
6                               Egypt     2014   0.218600
7                   Equatorial Guinea     1991   0.217861
8                             Eritrea     1997   0.075621
9                            Ethiopia     1994   0.254865
10                               Fiji     2013   0.473559
11                             Gambia     1996   0.348132
12                               Iraq     2005   0.455402
13                         Kazakhstan     1995   0.262596
14   Lao People's Democratic Republic     1991   0.094434
15                              Libya     2011   0.294716
16                           Maldives     2008   0.386754
17                         Montenegro     2007   0.455338
18                            Myanmar     2008   0.405772
19                               Oman     1996   0.191211
20                 Russian Federation     1993   0.275516
21                             Rwanda     2003   0.274476
22                       Saudi Arabia     1992   0.024049
```

```
23                          Serbia     2006   0.474443
24                         Somalia     2012   0.177772
25                     South Sudan     2011   0.183267
26                           Sudan     2005   0.311799
27                       Swaziland     2005   0.136008
28            Syrian Arab Republic     2012   0.148212
29                    Turkmenistan     2008   0.154887
30                          Uganda     1995   0.338308
31                         Ukraine     1996   0.361911
32                      Uzbekistan     1992   0.195932
33                        Viet Nam     1992   0.251461
34                           Yemen     1991   0.125708
35                          Zambia     1991   0.405497
36                        Zimbabwe     2013   0.315359
```

[ ]:
```
query3 = {"$or": [{"adopted": {"$gt": 1990}}, {"democracy": {"$lt": 0.5}}]}
results3 = collection.find(query3, projection)
df3 = pd.DataFrame(loads(dumps(results3)))
df3
```

[ ]:
```
        country  adopted  democracy
0   Afghanistan     2004   0.372201
1       Albania     1998   0.535111
2       Andorra     1993        NaN
3        Angola     2010   0.315043
4       Armenia     1995   0.393278
..          …        …          …
78   Uzbekistan     1992   0.195932
79     Viet Nam     1992   0.251461
80        Yemen     1991   0.125708
81       Zambia     1991   0.405497
82     Zimbabwe     2013   0.315359

[83 rows x 3 columns]
```

Part c According to the Varieties of Democracy project, Hungary has become less democratic over the last few years, and can no longer be considered a democracy. Update the record for Hungary to set the democracy score at 0.4. Then query the database to extract the record for Hungary and display the data in a dataframe. [1 point]

You will build a database and populate it with these data. The ER diagram for the database is:

There's no codebook, unfortunately, but the values in the columns are mostly self-explanatory:

[ ]:
```
update_result = collection.update_one(
    {"country": "Hungary"},
    {"$set": {"democracy": 0.4}}
)
```

```python
query_result = collection.find({"country": "Hungary"}, {"_id": 0, "country": 1, "adopted": 1, "democracy": 1})
df_hungary = pd.DataFrame(loads(dumps(query_result)))

df_hungary
```

[ ]:

	country	adopted	democracy
0	Hungary	2011	0.4

Part d Set the `text` field in the database as a text index. Then query the database to find all constitutions that contain the exact phrase "freedom of speech". Display the country name, adoption year, and democracy scores in a dataframe for the constitutions that match this query. [2 points]

[ ]:

```python
collection.create_index([("text", "text")])

query_freedom_of_speech = collection.find(
    {"$text": {"$search": "\"freedom of speech\""}},
    {"_id": 0, "country": 1, "adopted": 1, "democracy": 1, "score": {"$meta": "textScore"}}
).sort([("score", {"$meta": "textScore"})])

df_freedom_of_speech = pd.DataFrame(loads(dumps(query_freedom_of_speech)))

print(df_freedom_of_speech)
```

```
                     country  adopted  democracy     score
0                    Myanmar     2008   0.405772  1.999446
1                       Fiji     2013   0.473559  1.987625
2                 Bangladesh     1972   0.369978  1.986046
3        Korea (Republic of)     1948   0.757692  1.972244
4                     Gambia     1996   0.348132  1.970934
5                     Jordan     1952   0.270614  1.970513
6                     Mexico     1917   0.672567  1.969448
7                     Cyprus     1960   0.810509  1.969396
8                      Ghana     1992   0.670849  1.939626
9                      India     1949   0.632527  1.937849
10                   Georgia     1995   0.757486  1.878084
11                     China     1982   0.096066  1.877029
12                   Hungary     2011   0.400000  1.876935
13              Sierra Leone     1991   0.564657  1.876923
14              South Africa     1996   0.727070  1.876189
15                  Pakistan     1973   0.430273  1.875471
16               Philippines     1987   0.567393  1.867954
17                 Singapore     1963   0.446464  1.812783
18                  Malaysia     1957   0.345091  1.812688
19                   Eritrea     1997   0.075621  1.755868
20                  Zimbabwe     2013   0.315359  1.753159
```

21	Kenya	2010	0.531911	1.753046
22	Antigua and Barbuda	1981	NaN	1.752573
23	Saint Kitts and Nevis	1983	NaN	1.751588
24	Trinidad and Tobago	1976	0.730927	1.751134
25	Samoa	1962	NaN	1.750810
26	Sri Lanka	1978	0.647035	1.750668
27	Marshall Islands	1979	NaN	1.735062
28	Slovenia	1991	0.861380	1.506431
29	Poland	1997	0.682208	1.505841
30	Croatia	1991	0.710922	1.505056
31	Macedonia (The former Yugoslav Republic of)	1991	0.510983	1.504414
32	Kazakhstan	1995	0.262596	1.503627
33	Finland	1999	0.856265	1.502870
34	Spain	1978	0.834466	1.502784
35	Namibia	1990	0.745421	1.502779
36	Saint Vincent and the Grenadines	1979	NaN	1.502273
37	Dominica	1978	NaN	1.502226
38	Swaziland	2005	0.136008	1.502019
39	Peru	1993	0.730816	1.501971
40	Rwanda	2003	0.274476	1.501947
41	Papua New Guinea	1975	0.488884	1.501847
42	Belize	1981	NaN	1.501801
43	Seychelles	1993	0.589139	1.501721
44	Uganda	1995	0.338308	1.501426
45	Somalia	2012	0.177772	1.501224
46	Liberia	1986	0.629972	1.500673
47	Lao People's Democratic Republic	1991	0.094434	1.500575
48	Bhutan	2008	0.537041	1.497179
49	Korea (Democratic People's Republic of)	1972	0.090438	1.493815
50	Tonga	1875	NaN	1.485681
51	United States of America	1789	0.849155	1.250496

Part e Use a query to search for the terms "freedom", "liberty", "legal", "justice", and "rights". Generate a text score for all of the countries, and display the data for the countries with the top 10 relevancy scores in a dataframe. [2 points]

[ ]:
```
query_terms = collection.find(
    {"$text": {"$search": "freedom liberty legal justice rights"}},
    {"_id": 0, "country": 1, "adopted": 1, "democracy": 1, "score": {"$meta":
 ↪"textScore"}}
).sort([("score", {"$meta": "textScore"})]).limit(10)

df_terms = pd.DataFrame(loads(dumps(query_terms)))

df_terms
```

[ ]:

	country	adopted	democracy	score
0	Serbia	2006	0.474443	5.030999
1	Finland	1999	0.856265	5.029000
2	Estonia	1992	0.909233	5.024473
3	Armenia	1995	0.393278	5.023651
4	Albania	1998	0.535111	5.023087
5	Dominican Republic	2015	0.583654	5.019910
6	Moldova (Republic of)	1994	0.571357	5.017063
7	El Salvador	1983	0.661989	5.016899
8	Georgia	1995	0.757486	5.015282
9	Turkey	1982	0.341745	5.014672

1.1.8 Question 3

Close the connections to the PostgreSQL and MongoDB databases. [1 point]

[ ]:

```
conn.close()
client.close()
```