Lab Assignment 7: Database Queries - Rachel Holman

DS 6001: Practice and Application of Data Science

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.

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:

In [1]:
```
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

# change to the directory where your .env file is
os.chdir("/Users/rachelholman/Desktop/MSDS/DS6001 - Application of DS/Module 7-

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

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

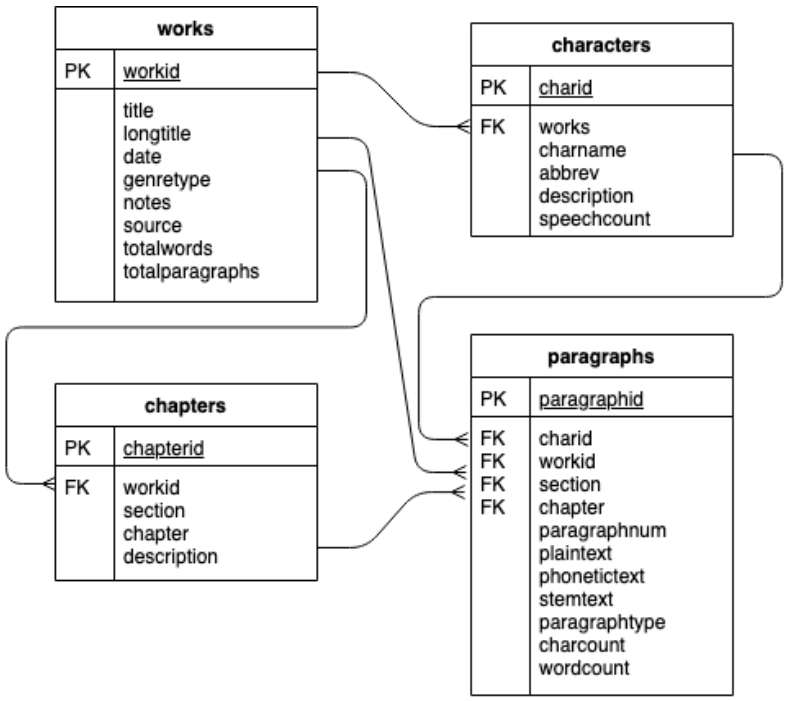
chapters (this is, scenes within acts), and one for paragraphs (that is, lines of dialogue). The data to populate these four tables are here:

In [2]:
```
works = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localdata/Wo
characters = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localda
chapters = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localdata
paragraphs = pd.read_csv("https://github.com/jkropko/DS-6001/raw/master/localda
```

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:

In [3]:
```
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:

In [4]:
```
works.head()
```

Out[4]:

	workid	title	longtitle	date	genretype	notes	source	totalwords	totalparag
0	12night	Twelfth Night	Twelfth Night, Or What You Will	1599	c	NaN	Moby	19837	
1	allswell	All's Well That Ends Well	All's Well That Ends Well	1602	c	NaN	Moby	22997	
2	antonycleo	Antony and Cleopatra	Antony and Cleopatra	1606	t	NaN	Moby	24905	
3	asyoulikeit	As You Like It	As You Like It	1599	c	NaN	Gutenberg	21690	
4	comedyerrors	Comedy of Errors	The Comedy of Errors	1589	c	NaN	Moby	14692	

In [5]:
```
characters.head()
```

Out[5]:

	charid	charname	abbrev	works	description	speechcount
0	1apparition-mac	First Apparition	First Apparition	macbeth	NaN	1.0
1	1citizen	First Citizen	First Citizen	romeojuliet	NaN	3.0
2	1conspirator	First Conspirator	First Conspirator	coriolanus	NaN	3.0
3	1gentleman-oth	First Gentleman	First Gentleman	othello	NaN	1.0
4	1goth	First Goth	First Goth	titus	NaN	4.0

In [6]:
```
chapters.head()
```

Out[6]:

	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.

In [7]:
```
paragraphs.head()
```

Out[7]:

	workid	paragraphid	paragraphnum	charid	plaintext	phonetictext	stemtext	paragrapl
0	12night	630863	3	xxx	[Enter DUKE ORSINO, CURIO, and other Lords; Mu...	ENTR TK ORSN KR ANT O0R LRTS MSXNS ATNTNK	enter duke orsino curio and other lord musicia...	
1	12night	630864	4	ORSINO	If music be the food of love, play on;\n[p]Giv...	IF MSK B 0 FT OF LF PL ON JF M EKSSS OF IT 0T ...	if music be the food of love plai on give me e...	
2	12night	630865	19	CURIO	Will you go hunt, my lord?\n	WL Y K HNT M LRT	will you go hunt my lord	
3	12night	630866	20	ORSINO	What, Curio? \n	HT KR	what curio	
4	12night	630867	21	CURIO	The hart.\n	0 HRT	the hart	

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]

In [8]:
```
import mysql.connector
import psycopg2
from sqlalchemy import create_engine

import dotenv
import os
dotenv.load_dotenv()
postgrespassword = os.getenv('postgrespassword')
```

In [9]:
```
engine = create_engine("postgresql+psycopg2://{user}:{pw}@localhost/{db}"
                       .format(user="rachelholman", pw=postgrespassword, db="sh
```

In [10]:
```
works.to_sql("works", con=engine, index=False, chunksize=1000, if_exists='repla
characters.to_sql("characters", con=engine, index=False, chunksize=1000, if_exi
chapters.to_sql("chapters", con=engine, index=False, chunksize=1000, if_exists=
paragraphs.to_sql("paragraphs", con=engine, index=False, chunksize=1000, if_exi
```

Out[10]:
```
35475
```

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]

In [11]:
```
q1 = '''
SELECT title, date AS year, totalwords, CASE
    WHEN date < 1600 THEN 'early'
    WHEN date BETWEEN 1600 AND 1607 THEN 'middle'
    WHEN date > 1607 THEN 'late'
    END AS era
FROM works
ORDER BY totalwords DESC
LIMIT 5 OFFSET 7
'''
pd.read_sql_query(q1, con=engine)
```

Out[11]:

	title	year	totalwords	era
0	Troilus and Cressida	1601	26089	middle
1	Henry IV, Part II	1597	25692	early
2	Henry VI, Part II	1590	25411	early
3	The Winter's Tale	1610	24914	late
4	Antony and Cleopatra	1606	24905	middle

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]

In [12]:
```
q2 = '''
SELECT CASE WHEN genretype='t' THEN 'Tragedy'
    WHEN genretype='c' THEN 'Comedy'
    WHEN genretype='h' THEN 'History'
    WHEN genretype='s' THEN 'Sonnets'
    WHEN genretype='p' THEN 'Poem'
    END AS Genre_Type, AVG(totalwords) AS Avg_WordCount
FROM works
GROUP BY genretype
ORDER BY AVG(totalwords) DESC
'''
pd.read_sql_query(q2, con=engine)
```

Out[12]:

	genre_type	avg_wordcount
0	History	24236.000000
1	Tragedy	23817.363636
2	Comedy	20212.071429
3	Sonnets	17515.000000
4	Poem	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]

In [13]:
```
q3 = '''
SELECT ch.charname, p.plaintext, p.wordcount
FROM characters ch
INNER JOIN paragraphs p
    ON ch.charid = p.charid
WHERE ch.charname = 'Hamlet'
ORDER BY length(p.plaintext) DESC
LIMIT 1
'''
speech = pd.read_sql_query(q3, con=engine)
print(speech['plaintext'][0])
```

Ay, so, God b' wi' ye! [Exeunt Rosencrantz and Guildens
tern
[p]Now I am alone.
[p]O what a rogue and peasant slave am I!
[p]Is it not monstrous that this player here,
[p]But in a fiction, in a dream of passion,
[p]Could force his soul so to his own conceit
[p]That, from her working, all his visage wann'd,
[p]Tears in his eyes, distraction in's aspect,
[p]A broken voice, and his whole function suiting
[p]With forms to his conceit? And all for nothing!
[p]For Hecuba!
[p]What's Hecuba to him, or he to Hecuba,
[p]That he should weep for her? What would he do,
[p]Had he the motive and the cue for passion
[p]That I have? He would drown the stage with tears
[p]And cleave the general ear with horrid speech;
[p]Make mad the guilty and appal the free,
[p]Confound the ignorant, and amaze indeed
[p]The very faculties of eyes and ears.
[p]Yet I,
[p]A dull and muddy-mettled rascal, peak
[p]Like John-a-dreams, unpregnant of my cause,
[p]And can say nothing! No, not for a king,
[p]Upon whose property and most dear life
[p]A damn'd defeat was made. Am I a coward?
[p]Who calls me villain? breaks my pate across?
[p]Plucks off my beard and blows it in my face?
[p]Tweaks me by th' nose? gives me the lie i' th' throat
[p]As deep as to the lungs? Who does me this, ha?
[p]'Swounds, I should take it! for it cannot be
[p]But I am pigeon-liver'd and lack gall
[p]To make oppression bitter, or ere this
[p]I should have fatted all the region kites
[p]With this slave's offal. Bloody bawdy villain!
[p]Remorseless, treacherous, lecherous, kindless villain!
[p]O, vengeance!
[p]Why, what an ass am I! This is most brave,
[p]That I, the son of a dear father murther'd,
[p]Prompted to my revenge by heaven and hell,
[p]Must (like a whore) unpack my heart with words
[p]And fall a-cursing like a very drab,
[p]A scullion!
[p]Fie upon't! foh! About, my brain! Hum, I have heard
[p]That guilty creatures, sitting at a play,
[p]Have by the very cunning of the scene
[p]Been struck so to the soul that presently
[p]They have proclaim'd their malefactions;
[p]For murther, though it have no tongue, will speak
[p]With most miraculous organ, I'll have these Players
[p]Play something like the murther of my father
[p]Before mine uncle. I'll observe his looks;
[p]I'll tent him to the quick. If he but blench,
[p]I know my course. The spirit that I have seen
[p]May be a devil; and the devil hath power
[p]T' assume a pleasing shape; yea, and perhaps
[p]Out of my weakness and my melancholy,
[p]As he is very potent with such spirits,
[p]Abuses me to damn me. I'll have grounds
[p]More relative than this. The play's the thing

```
[p]Wherein I'll catch the conscience of the King. Exit.
```

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]

In [14]:
```
q4 = '''
SELECT w.title, c.section AS act, c.chapter AS scene, c.description
FROM works w
INNER JOIN chapters c
    ON w.workid = c.workid
WHERE lower(c.description) LIKE lower('%%palace%%')
'''
pd.read_sql_query(q4, con=engine)
```

Out[14]:

	title	act	scene	description
0	Twelfth Night	2.0	4.0	DUKE ORSINO's palace.
1	Twelfth Night	1.0	4.0	DUKE ORSINO's palace.
2	Twelfth Night	1.0	1.0	DUKE ORSINO's palace.
3	All's Well That Ends Well	5.0	3.0	Rousillon. The COUNT's palace.
4	All's Well That Ends Well	5.0	2.0	Rousillon. Before the COUNT's palace.
...	...	...	...	...
120	The Winter's Tale	5.0	1.0	A room in LEONTES' palace.
121	The Winter's Tale	4.0	2.0	Bohemia. The palace of POLIXENES.
122	The Winter's Tale	2.0	3.0	A room in LEONTES' palace.
123	The Winter's Tale	2.0	1.0	A room in LEONTES' palace.
124	The Winter's Tale	1.0	1.0	Antechamber in LEONTES' palace.

125 rows × 4 columns

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]

In [15]:
```
q5 = '''
SELECT a.charname, a.description, w.title, a.speechcount, AVG(a.wordcount) AS a
```

```
FROM (
    SELECT ch.charname, ch.description, ch.speechcount, ch.works, p.wordcount
    FROM characters ch
    INNER JOIN paragraphs p
        ON ch.charid = p.charid
) AS a
LEFT JOIN works w
    ON a.works = w.workid
WHERE a.speechcount >=20
GROUP BY a.charname, a.speechcount, a.description, w.title
ORDER BY avg_wordcount DESC
'''
pd.read_sql_query(q5, con=engine)
```

Out[15]:

	charname	description	title	speechcount	avg_wordcount
0	Poet	the voice of Shakespeare's poetry	None	733.0	66.062756
1	King Richard II	king of England	Richard II	98.0	61.765306
2	Queen Katharine	wife to King Henry, afterwards divorced	Henry VIII	50.0	59.360000
3	Constance	mother to Arthur	King John	36.0	59.222222
4	Duke of Buckingham	None	Henry VIII	26.0	57.307692
...	...	...	...	...	...
380	First Murderer	None	Macbeth	21.0	8.666667
381	Curtis	None	Taming of the Shrew	20.0	8.550000
382	Lucius	servant to Brutus	Julius Caesar	24.0	8.541667
383	Alice	a lady attending on Princess Katherine	Henry V	22.0	7.454545
384	(stage directions)	None	As You Like It	126.0	4.309517

385 rows × 5 columns

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]

In [16]:

```
q6 = '''
SELECT w.title, CASE WHEN genretype='t' THEN 'Tragedy'
    WHEN genretype='c' THEN 'Comedy'
    WHEN genretype='h' THEN 'History'
    WHEN genretype='s' THEN 'Sonnets'
    WHEN genretype='p' THEN 'Poem'
```

```
    END AS Genre_Type, w.date
FROM works w
WHERE w.title NOT IN
    (SELECT w.title
    FROM works w
    INNER JOIN chapters c
        ON w.workid = c.workid
    WHERE lower(c.description) LIKE lower('%%palace%%')
        OR lower(c.description) LIKE lower('%%castle%%'))
'''
pd.read_sql_query(q6, con=engine)
```

Out[16]:

	title	genre_type	date
0	Coriolanus	Tragedy	1607
1	Julius Caesar	Tragedy	1599
2	Lover's Complaint	Poem	1609
3	Love's Labour's Lost	Comedy	1594
4	Merchant of Venice	Comedy	1596
5	Merry Wives of Windsor	Comedy	1600
6	Much Ado about Nothing	Comedy	1598
7	Passionate Pilgrim	Poem	1598
8	Phoenix and the Turtle	Poem	1601
9	Rape of Lucrece	Poem	1594
10	Romeo and Juliet	Tragedy	1594
11	Sonnets	Sonnets	1609
12	Taming of the Shrew	Comedy	1593
13	Tempest	Comedy	1611
14	Timon of Athens	Tragedy	1607
15	Venus and Adonis	Poem	1593

Problem 2

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

In [17]:

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

Out[17]:

	text	country	adopted	revised	reinstated	democracy
0	'Afghanistan 2004 Preamble \nIn the na...	Afghanistan	2004	NaN	NaN	0.372201
1	'Albania 1998 (rev. 2012) Preamble \nWe...	Albania	1998	2012.0	NaN	0.535111
2	'Andorra 1993 Preamble \nThe Andorran P...	Andorra	1993	NaN	NaN	NaN
3	'Angola 2010 Preamble \nWe, the people ...	Angola	2010	NaN	NaN	0.315043
4	'Antigua and Barbuda 1981 Preamble \nWH...	Antigua and Barbuda	1981	NaN	NaN	NaN
...	...	...	...	...	...	...
140	'Uzbekistan 1992 (rev. 2011) Preamble \...	Uzbekistan	1992	2011.0	NaN	0.195932
141	'Viet Nam 1992 (rev. 2013) Preamble \nI...	Viet Nam	1992	2013.0	NaN	0.251461
142	'Yemen 1991 (rev. 2001) PART ONE. THE FOUN...	Yemen	1991	2001.0	NaN	0.125708
143	'Zambia 1991 (rev. 2009) Preamble \nWE,...	Zambia	1991	2009.0	NaN	0.405497
144	'Zimbabwe 2013 Preamble \nWe the people...	Zimbabwe	2013	NaN	NaN	0.315359

145 rows × 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]

In [18]:
```
myclient = pymongo.MongoClient("mongodb://localhost/")
constdb = myclient["constdb"]
const_collection = constdb["const_collection"]
```

In [19]:
```
const_collection.delete_many({})
const_collection.insert_many(const_json)
const_collection.count_documents({})
```

Out[19]:
```
145
```

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]

In [20]:
```
cursor = const_collection.find({'adopted': {'$gt': 1990}},
                               {'country': 1,
                               'adopted': 1,
                               'democracy': 1,
                               '_id': 0})
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)
```

Out[20]:

	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 × 3 columns

In [21]:
```
cursor = const_collection.find({'adopted': {'$gt': 1990}, 'democracy': {'$lt':
                               {'country': 1,
                               'adopted': 1,
                               'democracy': 1,
                               '_id': 0})
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)
```

Out[21]:

	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

	country	adopted	democracy
35	Zambia	1991	0.405497
36	Zimbabwe	2013	0.315359

In [22]:
```
cursor = const_collection.find({'$or': [{'adopted': {'$gt': 1990}}, {'democracy
                                {'country': 1,
                                'adopted': 1,
                                'democracy': 1,
                                '_id': 0})
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)
```

Out[22]:

	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 × 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]

In [23]:
```
const_collection.update_one({'country': 'Hungary'},
                            {'$set' : {'democracy': 0.4}})
```

Out[23]:
```
<pymongo.results.UpdateResult at 0x108205750>
```

In [24]:
```
cursor = const_collection.find({'country': 'Hungary'})
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)
```

Out[24]:

	_id	text	country	adopted	revised	reinstated	democracy
0	64be73692992d846d44d283d	'Hungary 2011 (rev. 2013) Preamble \nGo...	Hungary	2011	2013.0	None	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]

In [25]:
```
const_collection.create_index([('text', 'text')])
```

Out[25]:
```
'text_text'
```

In [26]:
```
cursor = const_collection.find({'$text': {'$search':'\"freedom of speech\"', '$
                                {'country': 1,
                                 'adopted': 1,
                                 'democracy': 1,
                                 '_id': 0})
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)
```

Out[26]:

	country	adopted	democracy
0	Slovenia	1991	0.861380
1	Poland	1997	0.682208
2	Eritrea	1997	0.075621
3	Croatia	1991	0.710922
4	Macedonia (The former Yugoslav Republic of)	1991	0.510983
5	Kazakhstan	1995	0.262596
6	Zimbabwe	2013	0.315359
7	Kenya	2010	0.531911
8	Fiji	2013	0.473559
9	Finland	1999	0.856265
10	Georgia	1995	0.757486
11	Namibia	1990	0.745421
12	Spain	1978	0.834466
13	Korea (Republic of)	1948	0.757692
14	Antigua and Barbuda	1981	NaN
15	Saint Vincent and the Grenadines	1979	NaN
16	Dominica	1978	NaN
17	Swaziland	2005	0.136008
18	Ghana	1992	0.670849
19	Gambia	1996	0.348132
20	Peru	1993	0.730816
21	Rwanda	2003	0.274476
22	Papua New Guinea	1975	0.488884
23	Sierra Leone	1991	0.564657
24	Belize	1981	NaN
25	Hungary	2011	0.400000
26	Seychelles	1993	0.589139
27	China	1982	0.096066
28	Saint Kitts and Nevis	1983	NaN
29	Uganda	1995	0.338308
30	Bangladesh	1972	0.369978
31	Somalia	2012	0.177772
32	South Africa	1996	0.727070
33	Trinidad and Tobago	1976	0.730927
34	Jordan	1952	0.270614

	country	adopted	democracy
35	Samoa	1962	NaN
36	Sri Lanka	1978	0.647035
37	Liberia	1986	0.629972
38	Mexico	1917	0.672567
39	Lao People's Democratic Republic	1991	0.094434
40	Pakistan	1973	0.430273
41	India	1949	0.632527
42	Myanmar	2008	0.405772
43	Bhutan	2008	0.537041
44	Korea (Democratic People's Republic of)	1972	0.090438
45	Philippines	1987	0.567393
46	Tonga	1875	NaN
47	Marshall Islands	1979	NaN
48	Cyprus	1960	0.810509
49	Singapore	1963	0.446464
50	Malaysia	1957	0.345091
51	United States of America	1789	0.849155

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]

In [27]:
```
cursor = const_collection.find({'$text': {'$search':'freedom liberty legal just
                                {'score': {'$meta': 'textScore'}})
cursor.sort([('score', {'$meta':'textScore'})])
```

Out[27]:
```
<pymongo.cursor.Cursor at 0x13ab1da80>
```

In [28]:
```
qtext = dumps(cursor)
qrec = loads(qtext)
pd.DataFrame.from_records(qrec)[:10]
```

Out[28]:

	_id	text	country	adopted	revised	reinstated	democra
0	64be73692992d846d44d287c	'Serbia 2006 Preamble \nConsidering the...	Serbia	2006	NaN	NaN	0.4744
1	64be73692992d846d44d2834	'Finland 1999 (rev. 2011) Chapter 1. Funda...	Finland	1999	2011.0	NaN	0.8562
2	64be73692992d846d44d2831	'Estonia 1992 (rev. 2011) Preamble \nWi...	Estonia	1992	2011.0	NaN	0.9092
3	64be73692992d846d44d2810	'Armenia 1995 (rev. 2005) Preamble \nTh...	Armenia	1995	2005.0	NaN	0.3932
4	64be73692992d846d44d280c	'Albania 1998 (rev. 2012) Preamble \nWe...	Albania	1998	2012.0	NaN	0.535
5	64be73692992d846d44d282b	'Dominican Republic 2015 Preamble \nWe,...	Dominican Republic	2015	NaN	NaN	0.5836
6	64be73692992d846d44d285f	'Moldova (Republic of) 1994 (rev. 2006) Pr...	Moldova (Republic of)	1994	2006.0	NaN	0.5713
7	64be73692992d846d44d282e	'El Salvador 1983 (rev. 2014) TITLE I ...	El Salvador	1983	2014.0	NaN	0.6619
8	64be73692992d846d44d2837	'Georgia 1995 (rev. 2013) Preamble \nWe...	Georgia	1995	2013.0	NaN	0.7574
9	64be73692992d846d44d2890	'Turkey 1982 (rev. 2011) Preamble \nAff...	Turkey	1982	2011.0	NaN	0.3417

Question 3

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

In [29]:
```
engine.dispose()
myclient.close()
```