Lab Assignment 7: Database Queries

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/dmitrymikhaylov/Documents/learn/uva/spring2022/DS6001/s
urfing_data_pipeline/M6")

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.

... «(Life is) a tale told by an idiot, full of sound and fury, signifying nothing» ...
TOO LONG!
twitter
CHAPPATTE

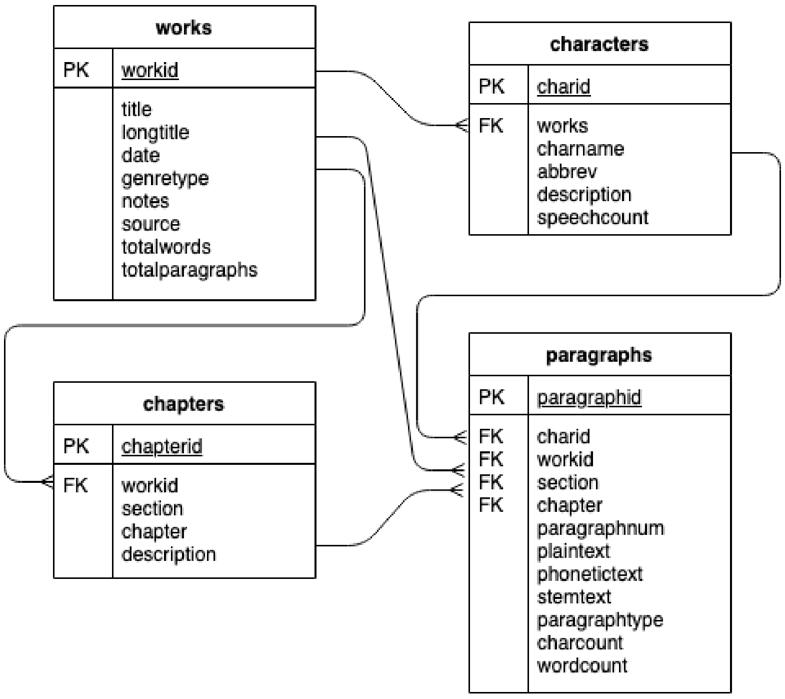
The data were collected by [Catherine Devlin (https://github.com/catherinedevlin/opensourceshakespeare)](https://github.com/catherinedevlin/opensourceshakespeare) from the repository at [https://opensourceshakespeare.org/ (https://opensourceshakespeare.org/)](https://opensourceshakespeare.org/). The 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/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 (https://stackoverflow.com/questions/20878932/are-postgresql-column-names-case-sensitive)](https://stackoverflow.com/questions/20878932/are-postgresql-column-names-case-sensitive) 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	totalparagra
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	paragrapht
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]:
```
dotenv.load_dotenv()
pgpassword = os.getenv("pgpassword")
```

In [9]:
```
dbserver = psycopg2.connect(
    user='dmitrymikhaylov',
    password=pgpassword,
    host="localhost"
)
dbserver.autocommit = True
```

In [10]:
```
cursor = dbserver.cursor()
```

In [11]:
```
try:
    cursor.execute("CREATE DATABASE psql_database")
except:
    cursor.execute("DROP DATABASE psql_database")
    cursor.execute("CREATE DATABASE psql_database")
```

In [12]:
```
psql_database = psycopg2.connect(
    user='dmitrymikhaylov',
    password=pgpassword,
    host="localhost",
    database="psql_database"
)
```

In [13]:
```
# Creating an “engine” to interface with the selected database
#from sqlalchemy import create_engine
engine = create_engine("postgresql+psycopg2://{user}:{pw}@localhost/{db}
"
                       .format(user="dmitrymikhaylov", pw=pgpassword, db
="psql_database"))
```

```
In [14]: works.to_sql('works', con = engine, index=False, chunksize=1000, if_exis
         ts = 'replace')
         characters.to_sql('characters', con = engine, index=False, chunksize=100
         0, if_exists = 'replace')
         chapters.to_sql('chapters', con = engine, index=False, chunksize=1000, i
         f_exists = 'replace')
         paragraphs.to_sql('paragraphs', con = engine, index=False, chunksize=100
         0, if_exists = 'replace')
```

```
Out[14]: 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 [110]: query_b = '''
          SELECT works.title, works.date AS year, works.totalwords, CASE
              WHEN works.date < 1600 THEN 'early'
              WHEN works.date BETWEEN 1600 AND 1607 THEN 'middle'
              WHEN works.date > 1607 THEN 'late'
              ELSE NULL
              END AS era
          FROM works
          ORDER BY totalwords DESC
          OFFSET 7
          LIMIT 5
          '''
          pd.read_sql_query(query_b, con=engine)
```

Out[110]:

	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 [16]:
```
query_c = '''
SELECT works.genretype as genre_type, AVG(works.totalwords) AS avg_words
FROM works
GROUP by genre_type
'''
pd.read_sql_query(query_c, con=engine)
```

Out[16]:

	genre_type	avg_words
0	c	20212.071429
1	h	24236.000000
2	t	23817.363636
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]

```
In [25]: query_d = '''
         SELECT charid, plaintext, LENGTH(plaintext) as speach_length
         FROM paragraphs
         WHERE charid LIKE 'hamlet'
         ORDER BY speach_length DESC
         LIMIT 1
         '''
         pd.read_sql_query(query_d, con=engine).plaintext
         pd.set_option('display.max_colwidth', -1)
         print(pd.read_sql_query(query_d, con=engine).plaintext)
```

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

<ipython-input-25-6d4990bc919d>:9: FutureWarning: Passing a negative in
teger is deprecated in version 1.0 and will not be supported in future
version. Instead, use None to not limit the column width.
  pd.set_option('display.max_colwidth', -1)
```

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 [18]:
```
query_e = '''
SELECT w.title, c.section AS act, c.chapter AS scene, c.description
FROM chapters as c
INNER JOIN works as w
    ON c.workid = w.workid
WHERE c.description LIKE '%%palace%%'
'''
pd.read_sql_query(query_e, con=engine)
```

Out[18]:

	title	act	scene	description
0	Twelfth Night	1.0	1.0	DUKE ORSINO's palace.
1	Twelfth Night	1.0	4.0	DUKE ORSINO's palace.
2	Twelfth Night	2.0	4.0	DUKE ORSINO's palace.
3	All's Well That Ends Well	1.0	1.0	Rousillon. The COUNT's palace.
4	All's Well That Ends Well	1.0	2.0	Paris. The KING's palace.
...	...	...	...	...
114	The Winter's Tale	2.0	1.0	A room in LEONTES' palace.
115	The Winter's Tale	2.0	3.0	A room in LEONTES' palace.
116	The Winter's Tale	4.0	2.0	Bohemia. The palace of POLIXENES.
117	The Winter's Tale	5.0	1.0	A room in LEONTES' palace.
118	The Winter's Tale	5.0	2.0	Before LEONTES' palace.

119 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 [19]: query_f = '''
         SELECT c.charname, w.title, c.description, c.speechcount, AVG(p.wordcoun
         t) as average_speech
         FROM paragraphs as p
         INNER JOIN characters as c
             ON p.charid = c.charid
         INNER JOIN works as w
             ON c.works = w.workid
         WHERE c.speechcount >= 20
         GROUP BY c.charname, c.description, w.title, c.speechcount
         ORDER BY average_speech DESC
         '''
         pd.read_sql_query(query_f, con=engine)
```

Out[19]:

	charname	title	description	speechcount	average_speech
0	King Richard II	Richard II	king of England	98.0	61.765306
1	Queen Katharine	Henry VIII	wife to King Henry, afterwards divorced	50.0	59.360000
2	Constance	King John	mother to Arthur	36.0	59.222222
3	Duke of Buckingham	Henry VIII	None	26.0	57.307692
4	Oberon	Midsummer Night's Dream	king of the fairies	29.0	55.655172
...	...	...	...	...	...
337	First Murderer	Macbeth	None	21.0	8.666667
338	Curtis	Taming of the Shrew	None	20.0	8.550000
339	Lucius	Julius Caesar	servant to Brutus	24.0	8.541667
340	Alice	Henry V	a lady attending on Princess Katherine	22.0	7.454545
341	(stage directions)	As You Like It	None	126.0	4.309517

342 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 [20]: query_g = '''
SELECT w.title, w.genretype, w.date, c.description
FROM chapters as c
INNER JOIN works as w
    ON c.workid = w.workid
WHERE (c.description NOT LIKE '%%palace%%') AND (c.description NOT LIKE
 '%%castle%%')
'''
pd.read_sql_query(query_g, con=engine)
```

Out[20]:

	title	genretype	date	description
0	Twelfth Night	c	1599	The sea-coast.
1	Twelfth Night	c	1599	OLIVIA'S house.
2	Twelfth Night	c	1599	OLIVIA'S house.
3	Twelfth Night	c	1599	The sea-coast.
4	Twelfth Night	c	1599	A street.
...	...	...	...	...
792	The Winter's Tale	c	1610	Bohemia. A desert country near the sea.
793	The Winter's Tale	c	1610	Chorus as Time speaks.
794	The Winter's Tale	c	1610	A road near the Shepherd's cottage.
795	The Winter's Tale	c	1610	The Shepherd's cottage.
796	The Winter's Tale	c	1610	A chapel in PAULINA'S house.

797 rows × 4 columns

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 [15]:
```python
const = requests.get("https://github.com/jkropko/DS-6001/raw/master/loca
ldata/const.json")
const_json = json.loads(const.text)
pd.DataFrame.from_records(const_json)
```

Out[15]:

	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 (https://datarepository.wolframcloud.com/resources/World-Constitutions). 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) (https://www.v-dem.net/en/) 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 [16]: myclient = pymongo.MongoClient("mongodb://localhost/")
         db = myclient["db"]
         const_collection = db["const"]
         const_collection

Out[16]: Collection(Database(MongoClient(host=['localhost:27017'], document_clas
         s=dict, tz_aware=False, connect=True), 'db'), 'const')

In [17]: const_collection.delete_many({})

Out[17]: <pymongo.results.DeleteResult at 0x7f8b9ec29080>

In [18]: constitution_data = const_collection.insert_many(const_json)

In [19]: len(const_json)

Out[19]: 145

In [20]: const_collection.count_documents({})

Out[20]: 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 [55]:
```
# GREATER THAN
myquery1 = {'adopted': {'$gte': 1990}}
text1 = dumps(const_collection.find(myquery1))
rec1 = loads(text1)
pd.DataFrame.from_records(rec1)[['country', 'adopted', 'democracy']]
```

Out[55]:

	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
...	...	...	...
67	Uzbekistan	1992	0.195932
68	Viet Nam	1992	0.251461
69	Yemen	1991	0.125708
70	Zambia	1991	0.405497
71	Zimbabwe	2013	0.315359

72 rows × 3 columns

In [56]:
```
# AND
myquery2 = {'adopted': {'$gte': 1990}, 'democracy': {'$lte': 0.5}}
text2 = dumps(const_collection.find(myquery2))
rec2 = loads(text2)
pd.DataFrame.from_records(rec2)[['country', 'adopted', 'democracy']]
```

Out[56]:

	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

	country	adopted	democracy
34	Yemen	1991	0.125708
35	Zambia	1991	0.405497
36	Zimbabwe	2013	0.315359

```
In [57]: # OR
myquery3 = {'$or': [{'adopted': {'$gte': 1990}, 'democracy': {'$lte': 0.
5}}]}
text3 = dumps(const_collection.find(myquery3))
rec3 = loads(text3)
pd.DataFrame.from_records(rec3)[['country', 'adopted', 'democracy']]
```

Out[57]:

	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

	country	adopted	democracy
34	Yemen	1991	0.125708
35	Zambia	1991	0.405497
36	Zimbabwe	2013	0.315359

Part c

According to the Varieties of Democracy project, Hungary has become less democratic (https://www.v-dem.net/en/news/democratic-declines-hungary/) 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 [61]: const_collection.update_one({'country': 'Hungary'},
                                     {'$set' : {'democracy': 0.4}})
```

```
Out[61]: <pymongo.results.UpdateResult at 0x7f9edb4a77c0>
```

```
In [65]: myquery4 = {}
         text4 = dumps(const_collection.find(myquery4))
         rec4 = loads(text4)
         df2 = pd.DataFrame.from_records(rec4)[['country', 'adopted', 'democracy'
         ]]
         df2.loc[df2.country=='Hungary']
```

Out[65]:

	country	adopted	democracy
50	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]

```
In [66]: const_collection.create_index([('text', 'text')])
```

```
         ERROR! Session/line number was not unique in database. History logging
         moved to new session 1536
Out[66]: 'text_text'
```

```
In [68]: myquery5 = {'$text': {'$search': 'freedom of speech', '$caseSensitive':
         False}}
         text5 = dumps(const_collection.find(myquery5))
         rec5 = loads(text5)
         pd.DataFrame.from_records(rec5)[['country', 'adopted', 'democracy']]
```

Out[68]:

	country	adopted	democracy
0	Turkmenistan	2008	0.154887
1	Sweden	1974	0.902575
2	Slovenia	1991	0.861380
3	Poland	1997	0.682208
4	Bosnia and Herzegovina	1995	0.338267
...	...	...	...
140	Netherlands	1815	0.859255
141	Denmark	1953	0.883552
142	United States of America	1789	0.849155
143	Australia	1901	0.879540
144	Brunei Darussalam	1959	NaN

145 rows × 3 columns

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 [82]: cursor = const_collection.find(
                         {'$text': {'$search': 'freedom liberty legal justice rights'
         }},
                         {'score': {'$meta': 'textScore'}})
```

```
In [83]: cursor.sort([('score', {'$meta': 'textScore'})])
```

```
Out[83]: <pymongo.cursor.Cursor at 0x7f9edb562d00>
```

```
In [84]: qtext = dumps(cursor)
         qrec = loads(qtext)
         df = pd.DataFrame.from_records(qrec)
```

In [87]:

```
df.head(2)
```

Out[87]:

	_id	text	country	adopted	revised	reinstated	democracy
0	624a1b417b3b0b1909c3b98e	'Serbia 2006 Preamble \nConsidering the...	Serbia	2006	NaN	NaN	0.474443
1	624a1b417b3b0b1909c3b946	'Finland 1999 (rev. 2011) Chapter 1. Funda...	Finland	1999	2011.0	NaN	0.856265

In [88]:

```
df[['country', 'adopted', 'democracy','score']].head(10)
```

Out[88]:

	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

Question 3

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

In [111]:

```
dbserver.close()
```

In [117]:

```
myclient.close()
```