Homework Problem Set F Submission Form

# Overview

| Your Name | Nick Videtti |
| --- | --- |
| Your SU Email | nvidetti@syr.edu |

# Instructions

Put your name and SU email at the top. Answer these questions all from the lab. When asked to include screenshots, please follow the screenshot guidelines from the first homework.

Remember as you complete the homework that it is not only about getting it right/correct. We will discuss the answers in class so it’s important to articulate anything you would like to contribute to the discussion in your answer:

* If you feel the question is vague, include any assumptions you've made.
* If you feel the answer requires interpretation or justification, provide it.
* If you do not know the answer to the question, articulate what you tried and how you are stuck.
* Highlight any doubts or questions you would like me to review.

This how you receive credit for answering questions that might not be correct. In addition, you must complete the reflection portion of the homework assignment for full credit. Since most answers will be similar this is an important part of your individual submission.

Complete Part II of this document first, then go back and complete the Reflection in Part I.

# Part I: Reflection

Use this section to reflect on your learning. To achieve the highest grade on the assignment, you must be as descriptive and personal as possible with your reflection.

1. As you completed this assignment, identify what you learned.

**This assignment taught me a ton about troubleshooting in Drill / Spark SQL and gave me a ton more experience with querying semi-structured JSON data with nested columns. It was also nice to brush up on the MongoDB knowledge that was covered in the asynchronous work.**

1. What barriers or challenges did you encounter while completing this assignment?

**Drill would not let me query MongoDB during my first attempt, but I was able to on my second attempt. Drill SQL seems to be case-sensitive unlike the common SQL syntax, and table references are sometimes still for nested data, even in queries with only one table. Also, it took many hours to figure out how to query the canceled Netflix JSON data using Spark. Exercise 9 also was a vague question, but I did what I could and what I thought may answer the question.**

1. How prepared were you to complete this assignment? What can you do to be better prepared?

**I do not feel like I was very prepared for this assignment because I do not feel that the asynchronous work for this week was enough to prepare for the depth and difficulty of this assignment.**

1. Rate your comfort level with this week’s material. Use the rubric provided.

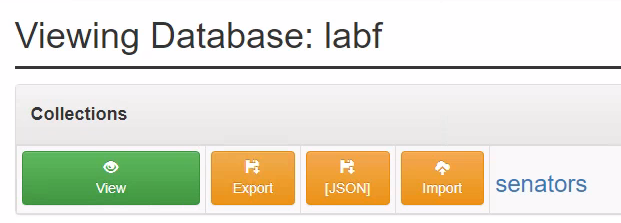
4 ==> I understand this material and can explain it to others.  
3 ==> I understand this material.  
**2 ==> I somewhat understand the material but sometimes need guidance from others.**  
1 ==> I understand very little of this material and need extra help.

# Part II: Questions

**For each question, include a copy of the code required to complete the question along with a screenshot of the code and a screenshot of the output.**

1. Use Spark to load the **/datasets/json-samples/US-Senators.json** into the MongoDB database **labf** under the collection **senators**.







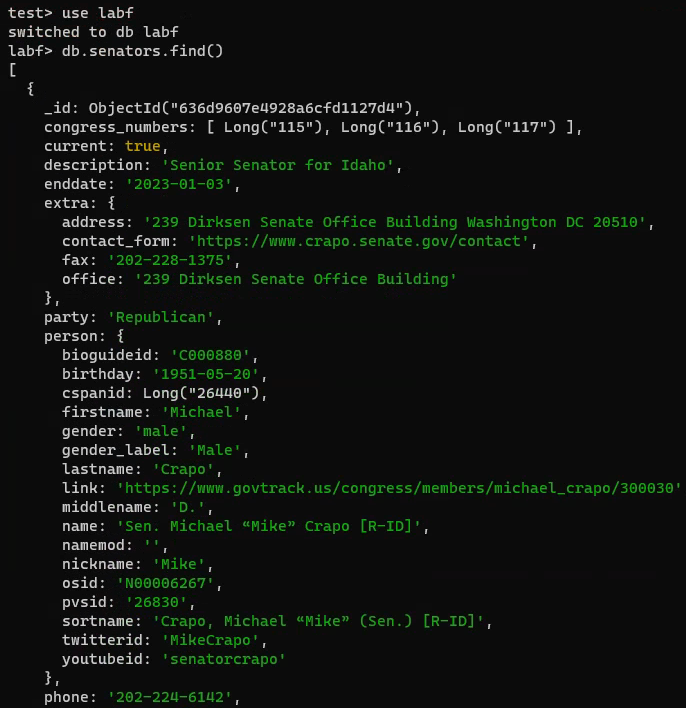
1. From the Mongo client, retrieve the firstname lastname, state, and party for those senators in either the “Republican,” “Democrat,” or “Independent” party.

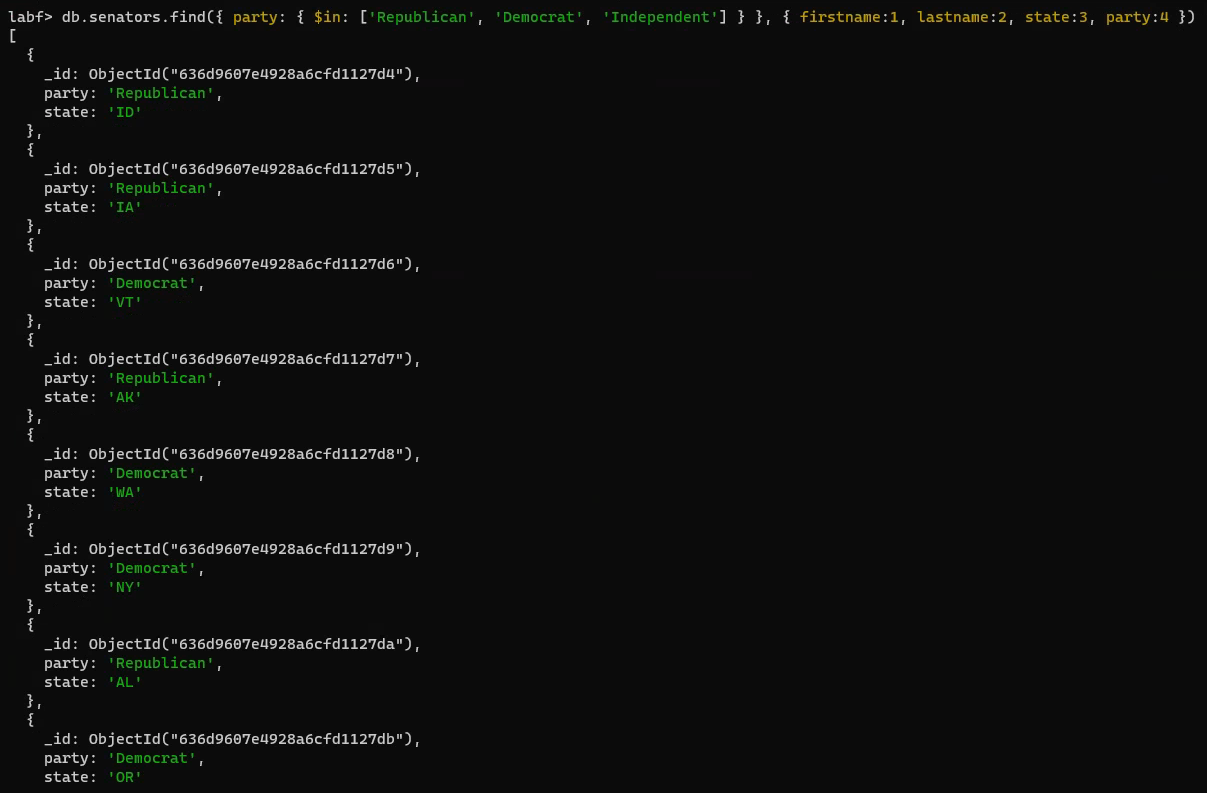
**test> use labf**

**switched to db labf**

**labf> db.senators.find()**

**labf> db.senators.find({ party: { $in: ['Republican', 'Democrat', 'Independent'] } }, { firstname:1, lastname:2, state:3, party:4 })**





1. From the Mongo client, write an in MQL index to improve the query performance of Question 2. Run the getIndexes() function on the collection to prove you created the index. Then use explain to prove the index is being used.

**labf> db.senators.createIndex({party:1})**

**party\_1**

**labf> db.senators.getIndexes()**

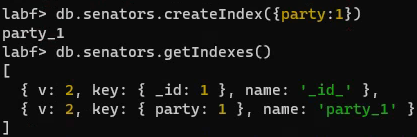
**[**

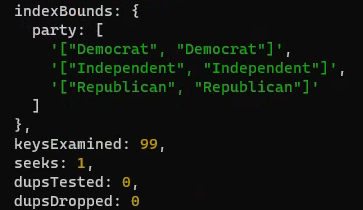
**{ v: 2, key: { \_id: 1 }, name: '\_id\_' },**

**{ v: 2, key: { party: 1 }, name: 'party\_1' }**

**]**

**labf> db.senators.find({ party: { $in: ['Republican', 'Democrat', 'Independent'] } }, { firstname:1, lastname:2, state:3, party:4 }).explain("executionStats")**

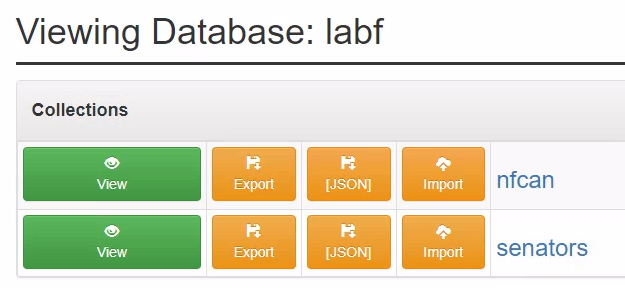




1. Use Spark to load in the **/datasets/netflix-canceled-2021/\*.json** into MongoDB database **labf** under the collection **nfcan**.

**spark.read.option("multiline","true").json("file:///home/jovyan/datasets/netflix-canceled-2021/\*.json")\**

**.write.format("mongo").mode("overwrite").option("database","labf").option("collection","nfcan").save()**  



1. Using Drill, write a query to list the name, premier date, and average rating of cancelled Netflix 2021 shows. Limit the shows to those with an average rating under 7.

**SELECT**

**name SHOW\_NAME,**

**premiered SHOW\_PREIMIERE\_DATE,**

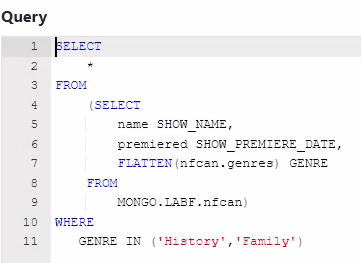
**nfcan.rating.average AVERAGE\_SHOW\_RATING**

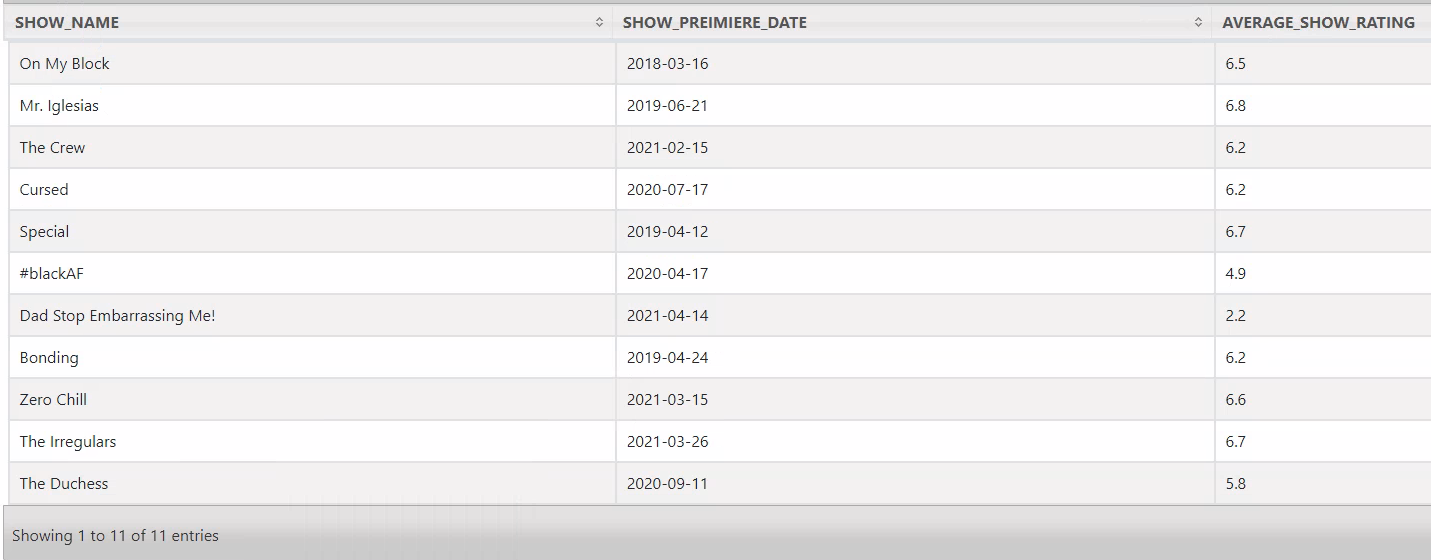
**FROM**

**MONGO.LABF.nfcan**

**WHERE**

**nfcan.rating.average < 7**

****

****

1. Using Drill, write a query to list the name, premier date, and genre for only those shows in the History or Family genres. Make sure to include a column to display the name of the genre.

**SELECT**

**\***

**FROM**

**(SELECT**

**name SHOW\_NAME,**

**premiered SHOW\_PREMIERE\_DATE,**

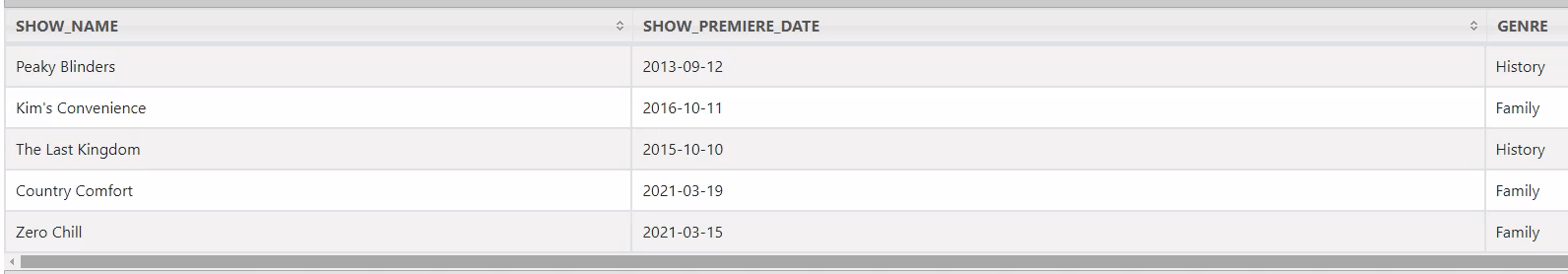
**FLATTEN(nfcan.genres) GENRE**

**FROM**

**MONGO.LABF.nfcan)**

**WHERE**

**GENRE IN ('History','Family')**

****

1. Using Spark or Spark SQL, create a DataFrame or view from the Netflix Cancellations MongoDB data, consisting of show name, season number, episode, number, episode name, airdate, and average rating (for the episode).

**from pyspark.sql.functions import explode**

**nfcan = spark.read.format("mongo").option("database","labf").option("collection","nfcan").load().createOrReplaceTempView('NFCAN')**

**Q7\_Query = spark.sql('''**

**SELECT**

**SHOW\_NAME,**

**SEASON,**

**EPISODE,**

**EPISODE\_NAME,**

**AIR\_DATE,**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**SHOW\_NAME,**

**SEASONPOS,**

**SEASON,**

**EPISODEPOS,**

**EPISODE,**

**EPISODE\_NAMEPOS,**

**EPISODE\_NAME,**

**AIR\_DATEPOS,**

**AIR\_DATE,**

**POSEXPLODE(AVERAGE\_EPISODE\_RATING) (AVERAGE\_EPISODE\_RATINGPOS, AVERAGE\_EPISODE\_RATING)**

**FROM**

**(SELECT**

**SHOW\_NAME,**

**SEASONPOS,**

**SEASON,**

**EPISODEPOS,**

**EPISODE,**

**EPISODE\_NAMEPOS,**

**EPISODE\_NAME,**

**POSEXPLODE(AIR\_DATE) (AIR\_DATEPOS, AIR\_DATE),**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**SHOW\_NAME,**

**SEASONPOS,**

**SEASON,**

**EPISODEPOS,**

**EPISODE,**

**POSEXPLODE(EPISODE\_NAME) (EPISODE\_NAMEPOS, EPISODE\_NAME),**

**AIR\_DATE,**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**SHOW\_NAME,**

**SEASONPOS,**

**SEASON,**

**POSEXPLODE(EPISODE) (EPISODEPOS, EPISODE),**

**EPISODE\_NAME,**

**AIR\_DATE,**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**SHOW\_NAME,**

**POSEXPLODE(SEASON) (SEASONPOS, SEASON),**

**EPISODE,**

**EPISODE\_NAME,**

**AIR\_DATE,**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**NAME SHOW\_NAME,**

**\_EMBEDDED.EPISODES.SEASON SEASON,**

**\_EMBEDDED.EPISODES.NUMBER EPISODE,**

**\_EMBEDDED.EPISODES.NAME EPISODE\_NAME,**

**\_EMBEDDED.EPISODES.AIRDATE AIR\_DATE,**

**\_EMBEDDED.EPISODES.RATING.AVERAGE AVERAGE\_EPISODE\_RATING**

**FROM**

**NFCAN))))))**

**WHERE**

**SEASONPOS = EPISODEPOS**

**AND EPISODEPOS = EPISODE\_NAMEPOS**

**AND EPISODE\_NAMEPOS = AIR\_DATEPOS**

**AND AIR\_DATEPOS = AVERAGE\_EPISODE\_RATINGPOS**

**ORDER BY**

**SHOW\_NAME,**

**SEASON ASC,**

**EPISODE ASC''')**

**Q7\_Query.show()**

****

1. Using the query you wrote in Question 7 (if you want), write a Spark or Spark SQL query to get the lowest rated episodes of each season for the cancelled shows. Display show name, season number, episode number, episode name, and rating for that episode.   
   NOTE: Some shows have more than one episode with the lowest rating.

**Q7\_Query.createOrReplaceTempView('Q7\_QUERY')**

**spark.sql('''**

**SELECT**

**SHOW\_NAME,**

**SEASON,**

**EPISODE,**

**EPISODE\_NAME,**

**AVERAGE\_EPISODE\_RATING**

**FROM**

**(SELECT**

**\*,**

**MIN(AVERAGE\_EPISODE\_RATING) OVER(PARTITION BY SHOW\_NAME,SEASON) MIN\_SEASON\_RATING**

**FROM**

**Q7\_QUERY)**

**WHERE**

**AVERAGE\_EPISODE\_RATING = MIN\_SEASON\_RATING**

**ORDER BY**

**SHOW\_NAME,**

**SEASON,**

**EPISODE''').show()**

****

1. CHALLENGE YOURSELF! Display name of show, a picture of the show, and show summary. Make it interactive so you can select the show and see the details.

**Q9\_Query = spark.sql('''**

**SELECT**

**NAME SHOW\_NAME,**

**URL SHOW\_URL,**

**'',**

**IMAGE.ORIGINAL SHOW\_PICTURE,**

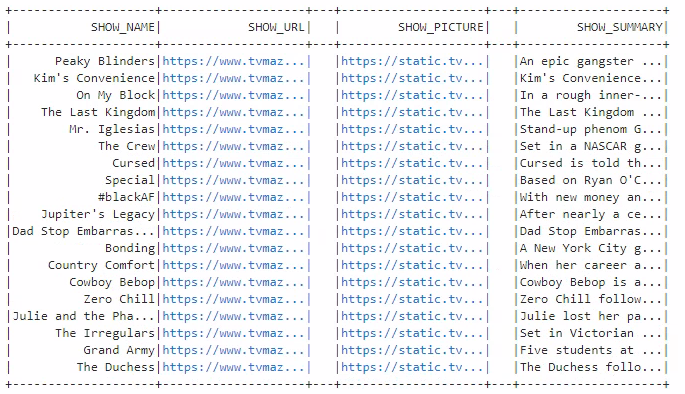
**' ',**

**REPLACE(REPLACE(REPLACE(REPLACE(SUMMARY,'<p>',''),'<b>',''),'</b>',''),'</p>','') SHOW\_SUMMARY**

**FROM**

**NFCAN''')**

**Q9\_Query.show()**

****