proj2_pyspark_nb_submission

July 2, 2021

0.1 Notebook for Pyspark for Project 2

[1]:
```
spark
```

[1]:
```
<pyspark.sql.session.SparkSession at 0x7ff229574ac8>
```

0.1.1 Imports of libraries for data transformation

[2]:
```
# Imports
import sys
import json
from pyspark.sql import Row
import pprint

p = pprint.PrettyPrinter(indent=1)
```

0.1.2 Consume data from our kafka topic userAct

Read in data to `messages` spark dataframe. spark.read.format("kafka") tells us we are reading in data from kafka. We specify in option our bootstrap servers and the kafka port number specified in our docker-compose file. We also specify that we want to subscribe to the topic userAct. Our startingOffsets and endingOffsets are earliest and latest to specify that we want to read from the beginning to the end of the entire data. Lastly, load() will load this data into the messages spark data frame.

[3]:
```
messages = spark.read.format("kafka").option("kafka.bootstrap.servers", "kafka:
↪29092").option("subscribe","userAct").option("startingOffsets", "earliest").
↪option("endingOffsets", "latest").load()
```

Object `messages` is a dataframe. We cache the messages to speed up access since we will be using them frequently in this process.

[4]:
```
type(messages)
```

[4]:
```
pyspark.sql.dataframe.DataFrame
```

[5]:
```
messages.cache()
```

[5]:
```
DataFrame[key: binary, value: binary, topic: string, partition: int, offset:
bigint, timestamp: timestamp, timestampType: int]
```

Prints the schema of the messages data we just read in. See that there are key value pairs. Values are the item of interest for us.

[6]:
```
messages.printSchema()
```

```
root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

Show the top 20 rows of the messages.

[7]:
```
messages.show()
```

```
+----+--------------------+-------+---------+------+--------------------+-------
------+
| key|               value|  topic|partition|offset|
timestamp|timestampType|
+----+--------------------+-------+---------+------+--------------------+-------
------+
|null|[7B 22 6B 65 65 6…|userAct|        0|     0|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     1|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     2|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     3|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     4|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     5|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     6|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     7|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     8|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|     9|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    10|1969-12-31 23:59:…|
```

```
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    11|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    12|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    13|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    14|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    15|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    16|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    17|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    18|1969-12-31 23:59:…|
0|
|null|[7B 22 6B 65 65 6…|userAct|        0|    19|1969-12-31 23:59:…|
0|
+----+--------------------+-------+---------+------+--------------------+-------
------+
only showing top 20 rows
```

Create new dataframe `messages_as_strings` as messages but only selecting the `value` column casted as a string data type.

[8]:
```
messages_as_strings=messages.selectExpr("CAST(value AS STRING)")
```

[9]:
```
messages_as_strings.show()
```

```
+--------------------+
|               value|
+--------------------+
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
```

```
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
|{"keen_timestamp"…|
+--------------------+
only showing top 20 rows
```

[10]:
```
messages_as_strings.cache()
```

[10]:
```
DataFrame[value: string]
```

[11]:
```
messages_as_strings.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

Using the `.count()` method lets us count the number of rows in this df.

[12]:
```
messages_as_strings.count()
```

[12]:
```
3280
```

Examine Data: Selecting one value row from `messages_as_strings`, select the 'value' column. Take 1 entry, if this was 200 it would take 200 entries from the beginning. `[0]` specifies the index of the entry to extract. The `value` takes out the value of this row, or the dict object.

[13]:
```
messages_as_strings.select('value').take(1)[0].value
```

[13]:
```
'{"keen_timestamp":"1516717442.735266","max_attempts":"1.0","started_at":"2018-0
1-23T14:23:19.082Z","base_exam_id":"37f0a30a-7464-11e6-aa92-a8667f27e5dc","user_
exam_id":"6d4089e4-bde5-4a22-b65f-18bce9ab79c8","sequences":{"questions":[{"user
_incomplete":true,"user_correct":false,"options":[{"checked":true,"at":"2018-01-
23T14:23:24.670Z","id":"49c574b4-5c82-4ffd-9bd1-c3358faf850d","submitted":1,"cor
rect":true},{"checked":true,"at":"2018-01-23T14:23:25.914Z","id":"f2528210-35c3-
4320-acf3-9056567ea19f","submitted":1,"correct":true},{"checked":false,"correct"
:true,"id":"d1bf026f-554f-4543-bdd2-54dcf105b826"}],"user_submitted":true,"id":"
7a2ed6d3-f492-49b3-b8aa-d080a8aad986","user_result":"missed_some"},{"user_incomp
lete":false,"user_correct":false,"options":[{"checked":true,"at":"2018-01-23T14:
23:30.116Z","id":"a35d0e80-8c49-415d-b8cb-c21a02627e2b","submitted":1},{"checked
":false,"correct":true,"id":"bccd6e2e-2cef-4c72-8bfa-317db0ac48bb"},{"checked":t
rue,"at":"2018-01-23T14:23:41.791Z","id":"7e0b639a-2ef8-4604-b7eb-5018bd81a91b",
"submitted":1,"correct":true}],"user_submitted":true,"id":"bbed4358-999d-4462-95
96-bad5173a6ecb","user_result":"incorrect"},{"user_incomplete":false,"user_corre
ct":true,"options":[{"checked":false,"at":"2018-01-23T14:23:52.510Z","id":"a9333
```

679-de9d-41ff-bb3d-b239d6b95732"},{"checked":false,"id":"85795acc-b4b1-4510-bd6e
-41648a3553c9"},{"checked":true,"at":"2018-01-23T14:23:54.223Z","id":"c185ecdb-4
8fb-4edb-ae4e-0204ac7a0909","submitted":1,"correct":true},{"checked":true,"at":"
2018-01-23T14:23:53.862Z","id":"77a66c83-d001-45cd-9a5a-6bba8eb7389e","submitted
":1,"correct":true}],"user_submitted":true,"id":"e6ad8644-96b1-4617-b37b-a263dde
d202c","user_result":"correct"},{"user_incomplete":false,"user_correct":true,"op
tions":[{"checked":false,"id":"59b9fc4b-f239-4850-b1f9-912d1fd3ca13"},{"checked"
:false,"id":"2c29e8e8-d4a8-406e-9cdf-de28ec5890fe"},{"checked":false,"id":"62fee
e6e-9b76-4123-bd9e-c0b35126b1f1"},{"checked":true,"at":"2018-01-23T14:24:00.807Z
","id":"7f13df9c-fcbe-4424-914f-2206f106765c","submitted":1,"correct":true}],"us
er_submitted":true,"id":"95194331-ac43-454e-83de-ea8913067055","user_result":"co
rrect"}],"attempt":1,"id":"5b28a462-7a3b-42e0-b508-09f3906d1703","counts":{"inco
mplete":1,"submitted":4,"incorrect":1,"all_correct":false,"correct":2,"total":4,
"unanswered":0}},"keen_created_at":"1516717442.735266","certification":"false","
keen_id":"5a6745820eb8ab00016be1f1","exam_name":"Normal Forms and All That Jazz
Master Class"}'

Uses `json.loads` to pass the above output into a json object. We use `json.dumps` to nicely print the json so we can see the fields, even those nested inside.

[14]:
```
# print json for first assessment to see nested schema

first_message = json.loads(messages_as_strings.select('value').take(1)[0].value)
print(json.dumps(first_message, indent=4, sort_keys=True))
```


```
{
    "base_exam_id": "37f0a30a-7464-11e6-aa92-a8667f27e5dc",
    "certification": "false",
    "exam_name": "Normal Forms and All That Jazz Master Class",
    "keen_created_at": "1516717442.735266",
    "keen_id": "5a6745820eb8ab00016be1f1",
    "keen_timestamp": "1516717442.735266",
    "max_attempts": "1.0",
    "sequences": {
        "attempt": 1,
        "counts": {
            "all_correct": false,
            "correct": 2,
            "incomplete": 1,
            "incorrect": 1,
            "submitted": 4,
            "total": 4,
            "unanswered": 0
        },
        "id": "5b28a462-7a3b-42e0-b508-09f3906d1703",
        "questions": [
            {
                "id": "7a2ed6d3-f492-49b3-b8aa-d080a8aad986",
```

```
    "options": [
        {
            "at": "2018-01-23T14:23:24.670Z",
            "checked": true,
            "correct": true,
            "id": "49c574b4-5c82-4ffd-9bd1-c3358faf850d",
            "submitted": 1
        },
        {
            "at": "2018-01-23T14:23:25.914Z",
            "checked": true,
            "correct": true,
            "id": "f2528210-35c3-4320-acf3-9056567ea19f",
            "submitted": 1
        },
        {
            "checked": false,
            "correct": true,
            "id": "d1bf026f-554f-4543-bdd2-54dcf105b826"
        }
    ],
    "user_correct": false,
    "user_incomplete": true,
    "user_result": "missed_some",
    "user_submitted": true
},
{
    "id": "bbed4358-999d-4462-9596-bad5173a6ecb",
    "options": [
        {
            "at": "2018-01-23T14:23:30.116Z",
            "checked": true,
            "id": "a35d0e80-8c49-415d-b8cb-c21a02627e2b",
            "submitted": 1
        },
        {
            "checked": false,
            "correct": true,
            "id": "bccd6e2e-2cef-4c72-8bfa-317db0ac48bb"
        },
        {
            "at": "2018-01-23T14:23:41.791Z",
            "checked": true,
            "correct": true,
            "id": "7e0b639a-2ef8-4604-b7eb-5018bd81a91b",
            "submitted": 1
        }
    ],
```

```
    "user_correct": false,
    "user_incomplete": false,
    "user_result": "incorrect",
    "user_submitted": true
},
{
    "id": "e6ad8644-96b1-4617-b37b-a263dded202c",
    "options": [
        {
            "at": "2018-01-23T14:23:52.510Z",
            "checked": false,
            "id": "a9333679-de9d-41ff-bb3d-b239d6b95732"
        },
        {
            "checked": false,
            "id": "85795acc-b4b1-4510-bd6e-41648a3553c9"
        },
        {
            "at": "2018-01-23T14:23:54.223Z",
            "checked": true,
            "correct": true,
            "id": "c185ecdb-48fb-4edb-ae4e-0204ac7a0909",
            "submitted": 1
        },
        {
            "at": "2018-01-23T14:23:53.862Z",
            "checked": true,
            "correct": true,
            "id": "77a66c83-d001-45cd-9a5a-6bba8eb7389e",
            "submitted": 1
        }
    ],
    "user_correct": true,
    "user_incomplete": false,
    "user_result": "correct",
    "user_submitted": true
},
{
    "id": "95194331-ac43-454e-83de-ea8913067055",
    "options": [
        {
            "checked": false,
            "id": "59b9fc4b-f239-4850-b1f9-912d1fd3ca13"
        },
        {
            "checked": false,
            "id": "2c29e8e8-d4a8-406e-9cdf-de28ec5890fe"
        },
```

```
                    {
                        "checked": false,
                        "id": "62feee6e-9b76-4123-bd9e-c0b35126b1f1"
                    },
                    {
                        "at": "2018-01-23T14:24:00.807Z",
                        "checked": true,
                        "correct": true,
                        "id": "7f13df9c-fcbe-4424-914f-2206f106765c",
                        "submitted": 1
                    }
                ],
                "user_correct": true,
                "user_incomplete": false,
                "user_result": "correct",
                "user_submitted": true
            }
        ]
    },
    "started_at": "2018-01-23T14:23:19.082Z",
    "user_exam_id": "6d4089e4-bde5-4a22-b65f-18bce9ab79c8"
}
```


Creating a dataframe/table from our raw data Create a new dataframe `assessments2` by taking `messages_as_strings` df from above, it's rdd map, and mapping each row through a function with the `.map` method. Our function is a lambda function that will take the value, make a json of the value, and pass the arguements of the json dictionary into a Spark DF `Row(...)`. Lastly we convert this whole object after the `map` applied into a Spark DF using `.toDF()`.

[15]:
```
assessments2 = messages_as_strings.rdd.map(lambda x: Row(**json.loads(x.
↪value))).toDF()
```

[16]:
```
# p.pprint(assessments2.show())
assessments2.show()
```

```
+--------------------+------------+--------------------+------------------+----
---------------+------------------+-----------+--------------------+----------
----------+--------------------+
|        base_exam_id|certification|           exam_name|   keen_created_at|
keen_id|    keen_timestamp|max_attempts|           sequences|
started_at|        user_exam_id|
+--------------------+------------+--------------------+------------------+----
---------------+------------------+-----------+--------------------+----------
----------+--------------------+
|37f0a30a-7464-11e…|       false|Normal Forms and …|
1516717442.735266|5a6745820eb8ab000…| 1516717442.735266|
1.0|Map(questions -> …|2018-01-23T14:23:…|6d4089e4-bde5-4a2…|
|37f0a30a-7464-11e…|       false|Normal Forms and …|
```

```
1516717377.639827|5a674541ab6b0a000…| 1516717377.639827|
1.0|Map(questions -> …|2018-01-23T14:21:…|2fec1534-b41f-441…|
|4beeac16-bb83-4d5…|        false|The Principles of…|
1516738973.653394|5a67999d3ed3e3000…| 1516738973.653394|
1.0|Map(questions -> …|2018-01-23T20:22:…|8edbc8a8-4d26-429…|
|4beeac16-bb83-4d5…|        false|The Principles
of…|1516738921.1137421|5a6799694fc7c7000…|1516738921.1137421|
1.0|Map(questions -> …|2018-01-23T20:21:…|c0ee680e-8892-4e6…|
|6442707e-7488-11e…|        false|Introduction to B…|
1516737000.212122|5a6791e824fccd000…| 1516737000.212122|
1.0|Map(questions -> …|2018-01-23T19:48:…|e4525b79-7904-405…|
|8b4488de-43a5-4ff…|        false|        Learning Git|
1516740790.309757|5a67a0b6852c2a000…| 1516740790.309757|
1.0|Map(questions -> …|2018-01-23T20:51:…|3186dafa-7acf-47e…|
|e1f07fac-5566-4fd…|        false|Git Fundamentals
…|1516746279.3801291|5a67b627cc80e6000…|1516746279.3801291|
1.0|Map(questions -> …|2018-01-23T22:24:…|48d88326-36a3-4cb…|
|7e2e0b53-a7ba-458…|        false|Introduction to P…|
1516743820.305464|5a67ac8cb0a5f4000…| 1516743820.305464|
1.0|Map(questions -> …|2018-01-23T21:43:…|bb152d6b-cada-41e…|
|1a233da8-e6e5-48a…|        false|Intermediate Pyth…|
1516743098.56811|5a67a9ba060087000…|  1516743098.56811|
1.0|Map(questions -> …|2018-01-23T21:31:…|70073d6f-ced5-4d0…|
|7e2e0b53-a7ba-458…|        false|Introduction to P…|
1516743764.813107|5a67ac54411aed000…| 1516743764.813107|
1.0|Map(questions -> …|2018-01-23T21:42:…|9eb6d4d6-fd1f-4f3…|
|4cdf9b5f-fdb7-4a4…|        false|A Practical
Intro…|1516744091.3127241|5a67ad9b2ff312000…|1516744091.3127241|
1.0|Map(questions -> …|2018-01-23T21:45:…|093f1337-7090-457…|
|e1f07fac-5566-4fd…|        false|Git Fundamentals
…|1516746256.5878439|5a67b610baff90000…|1516746256.5878439|
1.0|Map(questions -> …|2018-01-23T22:24:…|0f576abb-958a-4c0…|
|87b4b3f9-3a86-435…|        false|Introduction to M…|
1516743832.99235|5a67ac9837b82b000…|  1516743832.99235|
1.0|Map(questions -> …|2018-01-23T21:40:…|0c18f48c-0018-450…|
|a7a65ec6-77dc-480…|        false|   Python
Epiphanies|1516743332.7596769|5a67aaa4f21cc2000…|1516743332.7596769|
1.0|Map(questions -> …|2018-01-23T21:34:…|b38ac9d8-eef9-495…|
|7e2e0b53-a7ba-458…|        false|Introduction to P…|
1516743750.097306|5a67ac46f7bce8000…| 1516743750.097306|
1.0|Map(questions -> …|2018-01-23T21:41:…|bbc9865f-88ef-42e…|
|e5602ceb-6f0d-11e…|        false|Python Data
Struc…|1516744410.4791961|5a67aedaf34e85000…|1516744410.4791961|
1.0|Map(questions -> …|2018-01-23T21:51:…|8a0266df-02d7-44e…|
|e5602ceb-6f0d-11e…|        false|Python Data
Struc…|1516744446.3999851|5a67aefef5e149000…|1516744446.3999851|
1.0|Map(questions -> …|2018-01-23T21:53:…|95d4edb1-533f-445…|
|f432e2e3-7e3a-4a7…|        false|Working with Algo…|
```

```
1516744255.840405|5a67ae3f0c5f48000…| 1516744255.840405|
1.0|Map(questions -> …|2018-01-23T21:50:…|f9bc1eff-7e54-42a…|
|76a682de-6f0c-11e…|        false|Learning iPython …|
1516744023.652257|5a67ad579d5057000…| 1516744023.652257|
1.0|Map(questions -> …|2018-01-23T21:46:…|dc4b35a7-399a-4bd…|
|a7a65ec6-77dc-480…|        false|   Python
Epiphanies|1516743398.6451161|5a67aae6753fd6000…|1516743398.6451161|
1.0|Map(questions -> …|2018-01-23T21:35:…|d0f8249a-597e-4e1…|
+--------------------+-------------+--------------------+------------------+----
---------------+-----------------+-----------+--------------------+----------
----------+--------------------+
only showing top 20 rows
```

[17]:
```
type(assessments2)
```

[17]:
```
pyspark.sql.dataframe.DataFrame
```

[18]:
```
assessments2.printSchema()
```

```
root
 |-- base_exam_id: string (nullable = true)
 |-- certification: string (nullable = true)
 |-- exam_name: string (nullable = true)
 |-- keen_created_at: string (nullable = true)
 |-- keen_id: string (nullable = true)
 |-- keen_timestamp: string (nullable = true)
 |-- max_attempts: string (nullable = true)
 |-- sequences: map (nullable = true)
 |    |-- key: string
 |    |-- value: array (valueContainsNull = true)
 |    |    |-- element: map (containsNull = true)
 |    |    |    |-- key: string
 |    |    |    |-- value: boolean (valueContainsNull = true)
 |-- started_at: string (nullable = true)
 |-- user_exam_id: string (nullable = true)
```

0.2 Spark SQL: Answering Questions

Sequences is a nested json. Spark SQL can help up unpack data from this data frame. We need Spark SQL to answer business question on the dataframe and get better tables for answering the questions.

First, create a Spark "TempTable" (aka "View"). This is so that we can run Spark SQL queries on our assessments2 table.

[19]:
```
assessments2.registerTempTable('assessments_tbl')
```

[20]:
```
spark.sql("select exam_name, max_attempts as from assessments_tbl order by␣
 ↪max_attempts desc limit 5").show()
```

```
+--------------------+---+
|           exam_name| as|
+--------------------+---+
|Normal Forms and …|1.0|
|Normal Forms and …|1.0|
|The Principles of…|1.0|
|The Principles of…|1.0|
|Introduction to B…|1.0|
+--------------------+---+
```

0.2.1 1) How many assessments are in the dataset?

I count **3280 assessments**. Determined by counting up the rows in the dataset corresponding to assessments taken.

Note that there is a limitation to this since keen_id would have been a unique key for each assessment, but counting unique keen_id's yields less than 3280 assessments. Which is to be believed? We have 38 assessments with not unique keen_id's. Were these assessments given non-unique keen_id's by accident, or is keen_id not a reliable unique primary key for assessments?

[22]:
```
# by row count
spark.sql("select count(*) as total_assessments from assessments_tbl").show()
```

```
+-----------------+
|total_assessments|
+-----------------+
|             3280|
+-----------------+
```

[23]:
```
# by keen id
spark.sql("select count(distinct keen_id) as total_assessments from␣
 ↪assessments_tbl").show()
```

```
+-----------------+
|total_assessments|
+-----------------+
|             3242|
+-----------------+
```

0.2.2 2) How many distinct exams were taken by users?

There were **107 unique exams** taken by users in this dataset. We look at base_exam_id rather than exam_name since some exams could have the same name.

[24]:
```
spark.sql("select count(distinct base_exam_id) as unique_exams from␣
 ↪assessments_tbl").show()
```

```
+------------+
|unique_exams|
+------------+
|         107|
+------------+
```

0.2.3 3) How many people took Learning Git?

We need to create a table for exams and times taken by exam name. On this table, we would run the queries to answer this question. However, there is a problem with how to count the times taken for an exam.

The user_exam_id should give a distinct user assessment attempt for an exam, so we should count that for grouped exams. However, this leads to fewer times taken than just counting the number of times an exam showed up for an assessment. We will assume that the number of assessments for an exam gives the number of people who took Learning Git. Therefore we see that **394 people took Learning Git**.

[22]:
```
# based on the number of assessments for this exam
exams_taken_df = spark.sql("select exam_name, count(*) as times_taken from␣
 ↪assessments_tbl group by exam_name order by times_taken desc limit 10")
exams_taken_df.show()
```

```
+--------------------+-----------+
|           exam_name|times_taken|
+--------------------+-----------+
|        Learning Git|        394|
|Introduction to P…|        162|
|Intermediate Pyth…|        158|
|Introduction to J…|        158|
|Learning to Progr…|        128|
|Introduction to M…|        119|
|Software Architec…|        109|
|Beginning C# Prog…|         95|
|    Learning Eclipse|         85|
|Learning Apache M…|         80|
+--------------------+-----------+
```

[26]:
```
# based on the number of distinct user_exam_id for this exam
spark.sql("select exam_name, count(distinct user_exam_id) as times_taken from␣
 ↪assessments_tbl group by exam_name order by times_taken desc limit 10").
 ↪show()
```

```
+--------------------+-----------+
```

```
|           exam_name|times_taken|
+--------------------+-----------+
|        Learning Git|        390|
|Introduction to P…|        162|
|Introduction to J…|        158|
|Intermediate Pyth…|        156|
|Learning to Progr…|        128|
|Introduction to M…|        119|
|Software Architec…|        109|
|    Learning Eclipse|         85|
|Beginning C# Prog…|         83|
|Learning Apache M…|         80|
+--------------------+-----------+
```

Convert this exams_taken_df to a temp table for next queries.

[21]:
```
exams_taken_df = spark.sql("select exam_name, count(*) as times_taken from␣
 ↪assessments_tbl group by exam_name order by times_taken desc")
exams_taken_df.registerTempTable('exams_taken_tbl')
```

0.2.4 4) What is the least common course taken? And the most common?

Least common course taken: **Learning to Visualize Data with D3.js** with 1 assessment. Most common course taken: **Learning Git** with 394 assessments.

[274]:
```
spark.sql("select exam_name, times_taken from exams_taken_tbl order by␣
 ↪times_taken desc limit 1").show(1, False)
```

```
+------------+-----------+
|exam_name   |times_taken|
+------------+-----------+
|Learning Git|394        |
+------------+-----------+
```

[275]:
```
spark.sql("select exam_name, times_taken from exams_taken_tbl order by␣
 ↪times_taken limit 1").show(1, False)
```

```
+-----------------------------------+-----------+
|exam_name                          |times_taken|
+-----------------------------------+-----------+
|Learning to Visualize Data with D3.js|1          |
+-----------------------------------+-----------+
```

1 Save Tables to HDFS

Taking our Spark df, applying method to write it, specifying a parquet file for storage, and entering the path to where we want to store our data in HDFS and the name at the end.

[23]:
```
# Table 1 for Q1), Q2), Q3)
assessments2.write.parquet("/tmp/assessments_tbl")
```

[24]:
```
# Table 2 for Q4)
exams_taken_df.write.parquet("/tmp/exams_taken_tbl")
```

1.1 Extra Work: Making Tables from Nested Columns

1.1.1 Looking at Sequences

Make table that combines exam name and info about score on the exam and number of questions.

1-to-1 between row and extracted data from sequences, so use map instead of flatMap.

[25]:
```
def percent_score_from_json_flatMap(row):
    # grab the row for this exam
    exam = json.loads(row.value)

    # check if keys of sequences contains counts
    score_calc = -1 # default value
    num_q_calc = -1 # default value

    # sequences must exist
    if "sequences" in exam.keys():
        # counts must exist
        if "counts" in exam["sequences"]:
            # question data is not weird
            if ("correct" in exam["sequences"]["counts"]) and ("total" in␣
↪exam["sequences"]["counts"]) and (exam["sequences"]["counts"]["total"] != 0):
                score_calc = 100*exam["sequences"]["counts"]["correct"]/
↪exam["sequences"]["counts"]["total"]
                num_q_calc = len(exam["sequences"]["questions"])

    exam_details = {"base_exam_id": exam["base_exam_id"],
         "exam_name": exam["exam_name"],
         "keen_id": exam["keen_id"],
         "score": score_calc,
         "num_questions": num_q_calc,
         "user_exam_id": exam["user_exam_id"]}

    return [Row(**exam_details)]

def percent_score_from_json(row):
    # grab the row for this exam
```

```
    exam = json.loads(row.value)

    # check if keys of sequences contains counts
    score_calc = -1 # default value
    num_q_calc = -1 # default value

    # sequences must exist
    if "sequences" in exam.keys():
        # counts must exist
        if "counts" in exam["sequences"]:
            # question data is not weird
            if ("correct" in exam["sequences"]["counts"]) and ("total" in␣
↪exam["sequences"]["counts"]) and (exam["sequences"]["counts"]["total"] != 0):
                score_calc = 100*exam["sequences"]["counts"]["correct"]/
↪exam["sequences"]["counts"]["total"]
                num_q_calc = len(exam["sequences"]["questions"])

    exam_details = {"base_exam_id": exam["base_exam_id"],
         "exam_name": exam["exam_name"],
         "keen_id": exam["keen_id"],
         "score": score_calc,
         "num_questions": num_q_calc,
         "user_exam_id": exam["user_exam_id"]}

    return Row(**exam_details)
```

[26]:
```
exams_and_scores = messages_as_strings.rdd.map(percent_score_from_json).toDF()
# exams_and_scores = messages_as_strings.rdd.flatMap(percent_score_from_json).
↪toDF()
```

[27]:
```
exams_and_scores.printSchema()
```

```
root
 |-- base_exam_id: string (nullable = true)
 |-- exam_name: string (nullable = true)
 |-- keen_id: string (nullable = true)
 |-- num_questions: long (nullable = true)
 |-- score: double (nullable = true)
 |-- user_exam_id: string (nullable = true)
```

[28]:
```
# exams_and_scores.show(20, False)
exams_and_scores.show()
```

```
+--------------------+--------------------+--------------------+-------------+--
--------------+--------------------+
|        base_exam_id|           exam_name|              keen_id|num_questions|
```

```
score|        user_exam_id|
+--------------------+--------------------+--------------------+-------------+--
--------------+--------------------+
|37f0a30a-7464-11e…|Normal Forms and …|5a6745820eb8ab000…|            4|
50.0|6d4089e4-bde5-4a2…|
|37f0a30a-7464-11e…|Normal Forms and …|5a674541ab6b0a000…|            4|
25.0|2fec1534-b41f-441…|
|4beeac16-bb83-4d5…|The Principles of…|5a67999d3ed3e3000…|            4|
75.0|8edbc8a8-4d26-429…|
|4beeac16-bb83-4d5…|The Principles of…|5a6799694fc7c7000…|            4|
50.0|c0ee680e-8892-4e6…|
|6442707e-7488-11e…|Introduction to B…|5a6791e824fccd000…|            4|
75.0|e4525b79-7904-405…|
|8b4488de-43a5-4ff…|        Learning Git|5a67a0b6852c2a000…|            5|
100.0|3186dafa-7acf-47e…|
|e1f07fac-5566-4fd…|Git Fundamentals …|5a67b627cc80e6000…|            1|
100.0|48d88326-36a3-4cb…|
|7e2e0b53-a7ba-458…|Introduction to P…|5a67ac8cb0a5f4000…|            5|
100.0|bb152d6b-cada-41e…|
|1a233da8-e6e5-48a…|Intermediate Pyth…|5a67a9ba060087000…|            4|
100.0|70073d6f-ced5-4d0…|
|7e2e0b53-a7ba-458…|Introduction to P…|5a67ac54411aed000…|            5|
0.0|9eb6d4d6-fd1f-4f3…|
|4cdf9b5f-fdb7-4a4…|A Practical Intro…|5a67ad9b2ff312000…|            4|
75.0|093f1337-7090-457…|
|e1f07fac-5566-4fd…|Git Fundamentals …|5a67b610baff90000…|            1|
100.0|0f576abb-958a-4c0…|
|87b4b3f9-3a86-435…|Introduction to M…|5a67ac9837b82b000…|
6|66.66666666666667|0c18f48c-0018-450…|
|a7a65ec6-77dc-480…|   Python Epiphanies|5a67aaa4f21cc2000…|
6|66.66666666666667|b38ac9d8-eef9-495…|
|7e2e0b53-a7ba-458…|Introduction to P…|5a67ac46f7bce8000…|            5|
80.0|bbc9865f-88ef-42e…|
|e5602ceb-6f0d-11e…|Python Data Struc…|5a67aedaf34e85000…|            4|
75.0|8a0266df-02d7-44e…|
|e5602ceb-6f0d-11e…|Python Data Struc…|5a67aefef5e149000…|            4|
75.0|95d4edb1-533f-445…|
|f432e2e3-7e3a-4a7…|Working with Algo…|5a67ae3f0c5f48000…|            4|
100.0|f9bc1eff-7e54-42a…|
|76a682de-6f0c-11e…|Learning iPython …|5a67ad579d5057000…|            4|
50.0|dc4b35a7-399a-4bd…|
|a7a65ec6-77dc-480…|   Python Epiphanies|5a67aae6753fd6000…|            6|
100.0|d0f8249a-597e-4e1…|
+--------------------+--------------------+--------------------+-------------+--
--------------+--------------------+
only showing top 20 rows
```

[29]:
```
# filter out the nulls for score
exams_and_scores = exams_and_scores.filter("score is not null") # not null
exams_and_scores = exams_and_scores.filter("score != -1") # score not default␣
 ↪val of -1
exams_and_scores = exams_and_scores.filter("num_questions != -1") #␣
 ↪num_questions not default val of -1


# can do it in spark sql, just save out results to spark df

# google to see how to filter out for any columns
```

[30]:
```
# register temp table to run Spark SQL queries on
exams_and_scores.registerTempTable("examsScores_tbl")
```

1.1.2 Total number of exams in our table

[31]:
```
# count number of rows in table
spark.sql("SELECT COUNT(*) as total_num_exams FROM examsScores_tbl").show()
```

```
+---------------+
|total_num_exams|
+---------------+
|           3275|
+---------------+
```

[32]:
```
# test query
spark.sql("select exam_name, count(*) as times_taken from examsScores_tbl group␣
 ↪by exam_name limit 10").show()
```

```
+--------------------+-----------+
|           exam_name|times_taken|
+--------------------+-----------+
|Learning Data Mod…|          9|
|Networking for Pe…|         15|
|Introduction to J…|        158|
|Learning Apache H…|         16|
|Learning Spring P…|          2|
|Learning iPython …|         17|
|Introduction to P…|        162|
|Learning C# Best …|         35|
|Introduction to A…|         14|
|A Practical Intro…|          9|
+--------------------+-----------+
```

1.1.3 Q5) Which exam had the most questions?

The exam **Operating Red Hat Enterprise Linux Servers** has the most questions: 20.

[33]:
```
spark.sql("select distinct(exam_name), num_questions from examsScores_tbl order␣
 ↪by num_questions desc limit 10").show(10, False)
```

```
+----------------------------------------+-------------+
|exam_name                               |num_questions|
+----------------------------------------+-------------+
|Operating Red Hat Enterprise Linux Servers|20           |
|Great Bash                              |10           |
|Learning Linux System Administration    |8            |
|Learning to Program with R              |7            |
|Introduction to Data Science with R     |7            |
|Being a Better Introvert                |7            |
|Understanding the Grails 3 Domain Model |7            |
|What's New in JavaScript                |7            |
|Using Web Components                    |6            |
|Arduino Prototyping Techniques          |6            |
+----------------------------------------+-------------+
```

1.1.4 Q6) What is the average score on Learning Git?

The average score on Learning Git is **67.61%**.

[34]:
```
spark.sql("select exam_name, count(*) as times_taken, round(avg(score),2) as␣
 ↪avg_score from examsScores_tbl group by exam_name having exam_name ==␣
 ↪'Learning Git' order by times_taken desc limit 10").show()
```

```
+------------+-----------+---------+
|   exam_name|times_taken|avg_score|
+------------+-----------+---------+
|Learning Git|        394|    67.61|
+------------+-----------+---------+
```

2 Write to HDFS for table used to answer Q5), Q6)

[35]:
```
# Table 2 for Q5), Q6)
exams_and_scores.write.parquet("/tmp/exams_and_scores_tbl")
```

3 Conclusion of Spark Session. Return to `proj2_writeup.md`, section `Checking saved tables in HDFS`

[ ]: