Free Questions for Databricks-Certified-Associate-Developer-for-Apache-Spark-3.0 by certscare

Shared by Greene on 12-12-2023

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

The code block shown below should return a new 2-column DataFrame that shows one attribute from column attributes per row next to the associated itemName, for all suppliers in column supplier

whose name includes Sports. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Sample of DataFrame itemsDf:

1. +------+----------------------------------+---------------------------+------------------+

2. |itemId|itemName |attributes |supplier |

3. +------+----------------------------------+---------------------------+------------------+

4. |1 |Thick Coat for Walking in the Snow|[blue, winter, cozy] |Sports Company Inc.|

5. |2 |Elegant Outdoors Summer Dress |[red, summer, fresh, cooling]|YetiX |

6. |3 |Outdoors Backpack |[green, summer, travel] |Sports Company Inc.|

7. +------+----------------------------------+---------------------------+------------------+

Code block:

itemsDf.__1__(__2__).select(__3__, __4__)

Options:

A- 1. filter
2. col('supplier').isin('Sports')
3. 'itemName'
4. explode(col('attributes'))

B- 1. where
2. col('supplier').contains('Sports')
3. 'itemName'
4. 'attributes'

C- 1. where
2. col(supplier).contains('Sports')
3. explode(attributes)
4. itemName

D- 1. where
2. 'Sports'.isin(col('Supplier'))
3. 'itemName'
4. array_explode('attributes')

E- 1. filter
2. col('supplier').contains('Sports')
3. 'itemName'
4. explode('attributes')

Answer:

E

Explanation:

Output of correct code block:

+---------------------------------+------+

|itemName |col |

+---------------------------------+------+

|Thick Coat for Walking in the Snow|blue |

|Thick Coat for Walking in the Snow|winter|

|Thick Coat for Walking in the Snow|cozy |

|Outdoors Backpack |green |

|Outdoors Backpack |summer|

|Outdoors Backpack |travel|

+---------------------------------+------+

The key to solving this Question: is knowing about Spark's explode operator. Using this operator, you can extract values from arrays into single rows. The following guidance steps through

the

answers systematically from the first to the last gap. Note that there are many ways to solving the gap questions and filtering out wrong answers, you do not always have to start filtering out from the

first gap, but can also exclude some answers based on obvious problems you see with them.

The answers to the first gap present you with two options: filter and where. These two are actually synonyms in PySpark, so using either of those is fine. The answer options to this gap therefore do

not help us in selecting the right answer.

The second gap is more interesting. One answer option includes 'Sports'.isin(col('Supplier')). This construct does not work, since Python's string does not have an isin method. Another option

contains col(supplier). Here, Python will try to interpret supplier as a variable. We have not set this variable, so this is not a viable answer. Then, you are left with answers options that include col

('supplier').contains('Sports') and col('supplier').isin('Sports'). The Question: states that we are looking for suppliers whose name includes Sports, so we have to go for the contains operator

here.

We would use the isin operator if we wanted to filter out for supplier names that match any entries in a list of supplier names.

Finally, we are left with two answers that fill the third gap both with 'itemName' and the fourth gap either with explode('attributes') or 'attributes'. While both are correct Spark syntax, only explode

('attributes') will help us achieve our goal. Specifically, the Question: asks for one attribute from column attributes per row - this is what the explode() operator does.

One answer option also includes array_explode() which is not a valid operator in PySpark.

More info: pyspark.sql.functions.explode --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 39 (Databricks import instructions)

Question 2

Question Type: **MultipleChoice**

The code block shown below should add a column itemNameBetweenSeparators to DataFrame itemsDf. The column should contain arrays of maximum 4 strings. The arrays should be composed of

the values in column itemsDf which are separated at - or whitespace characters. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Sample of DataFrame itemsDf:

1. +------+---------------------------------+------------------+

2. |itemId|itemName |supplier |

3. +------+---------------------------------+------------------+

4. |1 |Thick Coat for Walking in the Snow|Sports Company Inc.|

5. |2 |Elegant Outdoors Summer Dress |YetiX |

6. |3 |Outdoors Backpack |Sports Company Inc.|

7. +------+---------------------------------+------------------+

Code block:

itemsDf.__1__(__2__, __3__(__4__, "[\s\-]", __5__))

Options:

A- 1. withColumn
2. 'itemNameBetweenSeparators'
3. split
4. 'itemName'
5. 4
(Correct)

B- 1. withColumnRenamed
2. 'itemNameBetweenSeparators'
3. split
4. 'itemName'
5. 4

C- 1. withColumnRenamed
2. 'itemName'
3. split
4. 'itemNameBetweenSeparators'
5. 4

D- 1. withColumn
2. 'itemNameBetweenSeparators'
3. split
4. 'itemName'
5. 5

E- 1. withColumn
2. itemNameBetweenSeparators
3. str_split
4. 'itemName'
5. 5

Answer:

A

Explanation:

This Question: deals with the parameters of Spark's split operator for strings.

To solve this question, you first need to understand the difference between DataFrame.withColumn() and DataFrame.withColumnRenamed(). The correct option here is DataFrame.withColumn()

since, according to the question, we want to add a column and not rename an existing column. This leaves you with only 3 answers to consider.

The second gap should be filled with the name of the new column to be added to the DataFrame. One of the remaining answers states the column name as itemNameBetweenSeparators, while the

other two state it as 'itemNameBetweenSeparators'. The correct option here is 'itemNameBetweenSeparators', since the other option would let Python try to interpret itemNameBetweenSeparators

as the name of a variable, which we have not defined. This leaves you with 2 answers to consider.

The decision boils down to how to fill gap 5. Either with 4 or with 5. The Question: asks for arrays of maximum four strings. The code in gap 5 relates to the limit parameter of Spark's split

operator

(see documentation linked below). The documentation states that 'the resulting array's length will not be more than limit', meaning that we should pick the answer option with 4 as the code in the

fifth gap here.

On a side note: One answer option includes a function str_split. This function does not exist in pySpark.

More info: pyspark.sql.functions.split --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 38 (Databricks import instructions)

Question 3

Question Type: **MultipleChoice**

Which of the following code blocks creates a new DataFrame with 3 columns, productId, highest, and lowest, that shows the biggest and smallest values of column value per value in column

productId from DataFrame transactionsDf?

Sample of DataFrame transactionsDf:

1. +-------------+---------+-----+-------+---------+----+

2. |transactionId|predError|value|storeId|productId| f|

3. +-------------+---------+-----+-------+---------+----+

4. | 1| 3| 4| 25| 1|null|

5. | 2| 6| 7| 2| 2|null|

6. | 3| 3| null| 25| 3|null|

7. | 4| null| null| 3| 2|null|

8. | 5| null| null| null| 2|null|

9. | 6| 3| 2| 25| 2|null|

10. +----------+---------+-----+-------+---------+----+

Options:

A- transactionsDf.max('value').min('value')

B- transactionsDf.agg(max('value').alias('highest'), min('value').alias('lowest'))

C- transactionsDf.groupby(col(productId)).agg(max(col(value)).alias('highest'), min(col(value)).alias('lowest'))

D- transactionsDf.groupby('productId').agg(max('value').alias('highest'), min('value').alias('lowest'))

E- transactionsDf.groupby('productId').agg({'highest': max('value'), 'lowest': min('value')})

Answer:

D

Explanation:

transactionsDf.groupby('productId').agg(max('value').alias('highest'), min('value').alias('lowest'))

Correct. groupby and aggregate is a common pattern to investigate aggregated values of groups.

transactionsDf.groupby('productId').agg({'highest': max('value'), 'lowest': min('value')})

Wrong. While DataFrame.agg() accepts dictionaries, the syntax of the dictionary in this code block is wrong. If you use a dictionary, the syntax should be like {'value': 'max'}, so using the column

name as the key and the aggregating function as value.

transactionsDf.agg(max('value').alias('highest'), min('value').alias('lowest'))

Incorrect. While this is valid Spark syntax, it does not achieve what the Question: asks for. The Question: specifically asks for values to be aggregated per value in column productId -

this column is

not considered here. Instead, the max() and min() values are calculated as if the entire DataFrame was a group.

transactionsDf.max('value').min('value')

Wrong. There is no DataFrame.max() method in Spark, so this command will fail.

transactionsDf.groupby(col(productId)).agg(max(col(value)).alias('highest'), min(col(value)).alias('lowest'))

No. While this may work if the column names are expressed as strings, this will not work as is. Python will interpret the column names as variables and, as a result, pySpark will not understand

which columns you want to aggregate.

More info: pyspark.sql.DataFrame.agg --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 32 (Databricks import instructions)

Question 4

Question Type: **MultipleChoice**

Which of the following code blocks returns a DataFrame with approximately 1,000 rows from the 10,000-row DataFrame itemsDf, without any duplicates, returning the same rows even if the code

block is run twice?

Options:

A- itemsDf.sampleBy('row', fractions={0: 0.1}, seed=82371)

B- itemsDf.sample(fraction=0.1, seed=87238)

C- itemsDf.sample(fraction=1000, seed=98263)

D- itemsDf.sample(withReplacement=True, fraction=0.1, seed=23536)

E- itemsDf.sample(fraction=0.1)

Answer:

B

Explanation:

itemsDf.sample(fraction=0.1, seed=87238)

Correct. If itemsDf has 10,000 rows, this code block returns about 1,000, since DataFrame.sample() is never guaranteed to return an exact amount of rows. To ensure you are not returning

duplicates, you should leave the withReplacement parameter at False, which is the default. Since the Question: specifies that the same rows should be returned even if the code block is run

twice,

you need to specify a seed. The number passed in the seed does not matter as long as it is an integer.

itemsDf.sample(withReplacement=True, fraction=0.1, seed=23536)

Incorrect. While this code block fulfills almost all requirements, it may return duplicates. This is because withReplacement is set to True.

Here is how to understand what replacement means: Imagine you have a bucket of 10,000 numbered balls and you need to take 1,000 balls at random from the bucket (similar to the problem in the

question). Now, if you would take those balls with replacement, you would take a ball, note its number, and put it back into the bucket, meaning the next time you take a ball from the bucket there

would be a chance you could take the exact same ball again. If you took the balls without replacement, you would leave the ball outside the bucket and not put it back in as you take the next 999

balls.

itemsDf.sample(fraction=1000, seed=98263)

Wrong. The fraction parameter needs to have a value between 0 and 1. In this case, it should be 0.1, since 1,000/10,000 = 0.1.

itemsDf.sampleBy('row', fractions={0: 0.1}, seed=82371)

No, DataFrame.sampleBy() is meant for stratified sampling. This means that based on the values in a column in a DataFrame, you can draw a certain fraction of rows containing those values from

the DataFrame (more details linked below). In the scenario at hand, sampleBy is not the right operator to use because you do not have any information about any column that the sampling should

depend on.

itemsDf.sample(fraction=0.1)

Incorrect. This code block checks all the boxes except that it does not ensure that when you run it a second time, the exact same rows will be returned. In order to achieve this, you would have to

specify a seed.

More info:

- pyspark.sql.DataFrame.sample --- PySpark 3.1.2 documentation

- pyspark.sql.DataFrame.sampleBy --- PySpark 3.1.2 documentation

- Types of Samplings in PySpark 3. The explanations of the sampling... | by Pinar Ersoy | Towards Data Science

Question 5

Question Type: **MultipleChoice**

The code block shown below should return a copy of DataFrame transactionsDf with an added column cos. This column should have the values in column value converted to degrees and having

the cosine of those converted values taken, rounded to two decimals. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Code block:

transactionsDf.__1__(__2__, round(__3__(__4__(__5__)),2))

Options:

A- 1. withColumn
2. col('cos')
3. cos
4. degrees
5. transactionsDf.value

B- 1. withColumnRenamed
2. 'cos'
3. cos
4. degrees
5. 'transactionsDf.value'

C- 1. withColumn
2. 'cos'
3. cos
4. degrees
5. transactionsDf.value

D- 1. withColumn
2. col('cos')
3. cos
4. degrees

5. col('value')

E- 1. withColumn
2. 'cos'
3. degrees
4. cos
5. col('value')

Answer:

C

Explanation:

Correct code block:

transactionsDf.withColumn('cos', round(cos(degrees(transactionsDf.value)),2))

This Question: is especially confusing because col, 'cos' are so similar. Similar-looking answer options can also appear in the exam and, just like in this question, you need to pay attention to

the

details to identify what the correct answer option is.

The first answer option to throw out is the one that starts with withColumnRenamed: The Question: speaks specifically of adding a column. The withColumnRenamed operator only renames

an

existing column, however, so you cannot use it here.

Next, you will have to decide what should be in gap 2, the first argument of transactionsDf.withColumn(). Looking at the documentation (linked below), you can find out that the first argument of

withColumn actually needs to be a string with the name of the column to be added. So, any answer that includes col('cos') as the option for gap 2 can be disregarded.

This leaves you with two possible answers. The real difference between these two answers is where the cos and degree methods are, either in gaps 3 and 4, or vice-versa. From the QUESTION

NO: you

can find out that the new column should have 'the values in column value converted to degrees and having the cosine of those converted values taken'. This prescribes you a clear order of

operations: First, you convert values from column value to degrees and then you take the cosine of those values. So, the inner parenthesis (gap 4) should contain the degree method and then,

logically, gap 3 holds the cos method. This leaves you with just one possible correct answer.

More info: pyspark.sql.DataFrame.withColumn --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 49 (Databricks import instructions)

Question 6

Question Type: MultipleChoice

Which of the following code blocks applies the Python function to_limit on column predError in table transactionsDf, returning a DataFrame with columns transactionId and result?

Options:

A- 1. spark.udf.register('LIMIT_FCN', to_limit)
2. spark.sql('SELECT transactionId, LIMIT_FCN(predError) AS result FROM transactionsDf')
(Correct)

B- 1. spark.udf.register('LIMIT_FCN', to_limit)
2. spark.sql('SELECT transactionId, LIMIT_FCN(predError) FROM transactionsDf AS result')

C- 1. spark.udf.register('LIMIT_FCN', to_limit)
2. spark.sql('SELECT transactionId, to_limit(predError) AS result FROM transactionsDf')
spark.sql('SELECT transactionId, udf(to_limit(predError)) AS result FROM transactionsDf')

D- 1. spark.udf.register(to_limit, 'LIMIT_FCN')
2. spark.sql('SELECT transactionId, LIMIT_FCN(predError) AS result FROM transactionsDf')

Answer:

A

Explanation:

spark.udf.register('LIMIT_FCN', to_limit)

spark.sql('SELECT transactionId, LIMIT_FCN(predError) AS result FROM transactionsDf')

Correct! First, you have to register to_limit as UDF to use it in a sql statement. Then, you can use it under the LIMIT_FCN name, correctly naming the resulting column result.

spark.udf.register(to_limit, 'LIMIT_FCN')

spark.sql('SELECT transactionId, LIMIT_FCN(predError) AS result FROM transactionsDf')

No. In this answer, the arguments to spark.udf.register are flipped.

spark.udf.register('LIMIT_FCN', to_limit)

spark.sql('SELECT transactionId, to_limit(predError) AS result FROM transactionsDf')

Wrong, this answer does not use the registered LIMIT_FCN in the sql statement, but tries to access the to_limit method directly. This will fail, since Spark cannot access it.

spark.sql('SELECT transactionId, udf(to_limit(predError)) AS result FROM transactionsDf')

Incorrect, there is no udf method in Spark's SQL.

spark.udf.register('LIMIT_FCN', to_limit)

spark.sql('SELECT transactionId, LIMIT_FCN(predError) FROM transactionsDf AS result')

False. In this answer, the column that results from applying the UDF is not correctly renamed to result.

Static notebook | Dynamic notebook: See test 3, Question: 52 (Databricks import instructions)

Question 7

Question Type: **MultipleChoice**

The code block shown below should return a DataFrame with only columns from DataFrame transactionsDf for which there is a corresponding transactionId in DataFrame itemsDf. DataFrame

itemsDf is very small and much smaller than DataFrame transactionsDf. The query should be executed in an optimized way. Choose the answer that correctly fills the blanks in the code block to

accomplish this.

__1__.__2__(__3__, __4__, __5__)

Options:

A- 1. transactionsDf
2. join
3. broadcast(itemsDf)
4. transactionsDf.transactionId==itemsDf.transactionId
5. 'outer'

B- 1. transactionsDf
2. join
3. itemsDf
4. transactionsDf.transactionId==itemsDf.transactionId
5. 'anti'

C- 1. transactionsDf
2. join
3. broadcast(itemsDf)
4. 'transactionId'
5. 'left_semi'

D- 1. itemsDf
2. broadcast
3. transactionsDf
4. 'transactionId'
5. 'left_semi'

E- 1. itemsDf
2. join

3. broadcast(transactionsDf)
4. 'transactionId'
5. 'left_semi'

Answer:

C

Explanation:

Correct code block:

transactionsDf.join(broadcast(itemsDf), 'transactionId', 'left_semi')

This Question: is extremely difficult and exceeds the difficulty of questions in the exam by far.

A first indication of what is asked from you here is the remark that 'the query should be executed in an optimized way'. You also have qualitative information about the size of itemsDf and

transactionsDf. Given that itemsDf is 'very small' and that the execution should be optimized, you should consider instructing Spark to perform a broadcast join, broadcasting the 'very small'

DataFrame itemsDf to all executors. You can explicitly suggest this to Spark via wrapping itemsDf into a broadcast() operator. One answer option does not include this operator, so you can disregard

it. Another answer option wraps the broadcast() operator around transactionsDf - the bigger of the two DataFrames. This answer option does not make sense in the optimization context and can

likewise be disregarded.

When thinking about the broadcast() operator, you may also remember that it is a method of pyspark.sql.functions. One answer option, however, resolves to itemsDf.broadcast([...]). The DataFrame

class has no broadcast() method, so this answer option can be eliminated as well.

All two remaining answer options resolve to transactionsDf.join([...]) in the first 2 gaps, so you will have to figure out the details of the join now. You can pick between an outer and a left semi join. An

outer join would include columns from both DataFrames, where a left semi join only includes columns from the 'left' table, here transactionsDf, just as asked for by the question. So, the correct

answer is the one that uses the left_semi join.

Question 8

Question Type: MultipleChoice

The code block shown below should return a new 2-column DataFrame that shows one attribute from column attributes per row next to the associated itemName, for all suppliers in column supplier

whose name includes Sports. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Sample of DataFrame itemsDf:

1. +------+---------------------------------+---------------------------+------------------+

2. |itemId|itemName |attributes |supplier |

3. +------+---------------------------------+---------------------------+------------------+

4. |1 |Thick Coat for Walking in the Snow|[blue, winter, cozy] |Sports Company Inc.|

5. |2 |Elegant Outdoors Summer Dress |[red, summer, fresh, cooling]|YetiX |

6. |3 |Outdoors Backpack |[green, summer, travel] |Sports Company Inc.|

7. +------+---------------------------------+---------------------------+------------------+

Code block:

itemsDf.__1__(__2__).select(__3__, __4__)

Options:

A- 1. filter
2. col('supplier').isin('Sports')
3. 'itemName'

4. explode(col('attributes'))

B- 1. where
2. col('supplier').contains('Sports')
3. 'itemName'
4. 'attributes'

C- 1. where
2. col(supplier).contains('Sports')
3. explode(attributes)
4. itemName

D- 1. where
2. 'Sports'.isin(col('Supplier'))
3. 'itemName'
4. array_explode('attributes')

E- 1. filter
2. col('supplier').contains('Sports')
3. 'itemName'
4. explode('attributes')

Answer:

E

Explanation:

Output of correct code block:

+---------------------------------+------+

|itemName |col |

+---------------------------------+------+

|Thick Coat for Walking in the Snow|blue |

|Thick Coat for Walking in the Snow|winter|

|Thick Coat for Walking in the Snow|cozy |

|Outdoors Backpack |green |

|Outdoors Backpack |summer|

|Outdoors Backpack |travel|

+---------------------------------+------+

The key to solving this Question: is knowing about Spark's explode operator. Using this operator, you can extract values from arrays into single rows. The following guidance steps through

the

answers systematically from the first to the last gap. Note that there are many ways to solving the gap questions and filtering out wrong answers, you do not always have to start filtering out from the

first gap, but can also exclude some answers based on obvious problems you see with them.

The answers to the first gap present you with two options: filter and where. These two are actually synonyms in PySpark, so using either of those is fine. The answer options to this gap therefore do

not help us in selecting the right answer.

The second gap is more interesting. One answer option includes 'Sports'.isin(col('Supplier')). This construct does not work, since Python's string does not have an isin method. Another option

contains col(supplier). Here, Python will try to interpret supplier as a variable. We have not set this variable, so this is not a viable answer. Then, you are left with answers options that include col

('supplier').contains('Sports') and col('supplier').isin('Sports'). The Question: states that we are looking for suppliers whose name includes Sports, so we have to go for the contains operator

here.

We would use the isin operator if we wanted to filter out for supplier names that match any entries in a list of supplier names.

Finally, we are left with two answers that fill the third gap both with 'itemName' and the fourth gap either with explode('attributes') or 'attributes'. While both are correct Spark syntax, only explode

('attributes') will help us achieve our goal. Specifically, the Question: asks for one attribute from column attributes per row - this is what the explode() operator does.

One answer option also includes array_explode() which is not a valid operator in PySpark.

More info: pyspark.sql.functions.explode --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 39 (Databricks import instructions)

Question 9

Question Type: **MultipleChoice**

The code block displayed below contains an error. The code block should merge the rows of DataFrames transactionsDfMonday and transactionsDfTuesday into a new DataFrame, matching

column names and inserting null values where column names do not appear in both DataFrames. Find the error.

Sample of DataFrame transactionsDfMonday:

1. +-------------+---------+-----+-------+---------+----+

2. |transactionId|predError|value|storeId|productId| f|

3. +-------------+---------+-----+-------+---------+----+

4. | 5| null| null| null| 2|null|

5. | 6| 3| 2| 25| 2|null|

6. +-------------+---------+-----+-------+---------+----+

Sample of DataFrame transactionsDfTuesday:

1. +-------+-------------+---------+-----+

2. |storeId|transactionId|productId|value|

3. +-------+-------------+---------+-----+

4. | 25| 1| 1| 4|

5. | 2| 2| 2| 7|

6. | 3| 4| 2| null|

7. | null| 5| 2| null|

8. +-------+-------------+---------+-----+

Code block:

sc.union([transactionsDfMonday, transactionsDfTuesday])

Options:

A- The DataFrames' RDDs need to be passed into the sc.union method instead of the DataFrame variable names.

B- Instead of union, the concat method should be used, making sure to not use its default arguments.

C- Instead of the Spark context, transactionDfMonday should be called with the join method instead of the union method, making sure to use its default arguments.

D- Instead of the Spark context, transactionDfMonday should be called with the union method.

E- Instead of the Spark context, transactionDfMonday should be called with the unionByName method instead of the union method, making sure to not use its default arguments.

Answer:

E

Explanation:

Correct code block:

transactionsDfMonday.unionByName(transactionsDfTuesday, True)

Output of correct code block:

+-------------+---------+-----+-------+---------+----+

|transactionId|predError|value|storeId|productId| f|

+-------------+---------+-----+-------+---------+----+

| 5| null| null| null| 2|null|

| 6| 3| 2| 25| 2|null|

| 1| null| 4| 25| 1|null|

| 2| null| 7| 2| 2|null|

| 4| null| null| 3| 2|null|

| 5| null| null| null| 2|null|

+----------+---------+-----+-------+---------+----+

For solving this question, you should be aware of the difference between the DataFrame.union() and DataFrame.unionByName() methods. The first one matches columns independent of their

names, just by their order. The second one matches columns by their name (which is asked for in the question). It also has a useful optional argument, allowMissingColumns. This allows you to

merge DataFrames that have different columns - just like in this example.

sc stands for SparkContext and is automatically provided when executing code on Databricks. While sc.union() allows you to join RDDs, it is not the right choice for joining DataFrames. A hint away

from sc.union() is given where the Question: talks about joining 'into a new DataFrame'.

concat is a method in pyspark.sql.functions. It is great for consolidating values from different columns, but has no place when trying to join rows of multiple DataFrames.

Finally, the join method is a contender here. However, the default join defined for that method is an inner join which does not get us closer to the goal to match the two DataFrames as instructed,

especially given that with the default arguments we cannot define a join condition.

More info:

- pyspark.sql.DataFrame.unionByName --- PySpark 3.1.2 documentation

- pyspark.SparkContext.union --- PySpark 3.1.2 documentation

- pyspark.sql.functions.concat --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 45 (Databricks import instructions)

Question 10

Question Type: **MultipleChoice**

The code block shown below should show information about the data type that column storeId of DataFrame transactionsDf contains. Choose the answer that correctly fills the blanks in the code

block to accomplish this.

Code block:

transactionsDf.__1__(__2__).__3__

Options:

A- 1. select
2. 'storeId'
3. print_schema()

B- 1. limit
2. 1
3. columns

C- 1. select
2. 'storeId'
3. printSchema()

D- 1. limit
2. 'storeId'
3. printSchema()

E- 1. select
2. storeId
3. dtypes

Answer:

B

Explanation:

Correct code block:

transactionsDf.select('storeId').printSchema()

The difficulty of this Question: is that it is hard to solve with the stepwise first-to-last-gap approach that has worked well for similar questions, since the answer options are so different from

one

another. Instead, you might want to eliminate answers by looking for patterns of frequently wrong answers.

A first pattern that you may recognize by now is that column names are not expressed in quotes. For this reason, the answer that includes storeId should be eliminated.

By now, you may have understood that the DataFrame.limit() is useful for returning a specified amount of rows. It has nothing to do with specific columns. For this reason, the answer that resolves to

limit('storeId') can be eliminated.

Given that we are interested in information about the data type, you should Question: whether the answer that resolves to limit(1).columns provides you with this information. While

DataFrame.columns is a valid call, it will only report back column names, but not column types. So, you can eliminate this option.

The two remaining options either use the printSchema() or print_schema() command. You may remember that DataFrame.printSchema() is the only valid command of the two. The select('storeId')

part just returns the storeId column of transactionsDf - this works here, since we are only interested in that column's type anyways.

More info: pyspark.sql.DataFrame.printSchema --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 57 (Databricks import instructions)

Question 11

Question Type: MultipleChoice

The code block shown below should return an exact copy of DataFrame transactionsDf that does not include rows in which values in column storeId have the value 25. Choose the answer that

correctly fills the blanks in the code block to accomplish this.

Options:

A- transactionsDf.remove(transactionsDf.storeId==25)

B- transactionsDf.where(transactionsDf.storeId!=25)

C- transactionsDf.filter(transactionsDf.storeId==25)

D- transactionsDf.drop(transactionsDf.storeId==25)

E- transactionsDf.select(transactionsDf.storeId!=25)

Answer:

B

Explanation:

transactionsDf.where(transactionsDf.storeId!=25)

Correct. DataFrame.where() is an alias for the DataFrame.filter() method. Using this method, it is straightforward to filter out rows that do not have value 25 in column storeId.

transactionsDf.select(transactionsDf.storeId!=25)

Wrong. The select operator allows you to build DataFrames column-wise, but when using it as shown, it does not filter out rows.

transactionsDf.filter(transactionsDf.storeId==25)

Incorrect. Although the filter expression works for filtering rows, the == in the filtering condition is inappropriate. It should be != instead.

transactionsDf.drop(transactionsDf.storeId==25)

No. DataFrame.drop() is used to remove specific columns, but not rows, from the DataFrame.

transactionsDf.remove(transactionsDf.storeId==25)

False. There is no DataFrame.remove() operator in PySpark.

More info: pyspark.sql.DataFrame.where --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 48 (Databricks import instructions)

Question 12

Question Type: **MultipleChoice**

Which of the following code blocks returns a single-column DataFrame of all entries in Python list throughputRates which contains only float-type values ?

Options:

A- spark.createDataFrame((throughputRates), FloatType)

B- spark.createDataFrame(throughputRates, FloatType)

C- spark.DataFrame(throughputRates, FloatType)

D- spark.createDataFrame(throughputRates)

E- spark.createDataFrame(throughputRates, FloatType())

Answer:

E

Explanation:

spark.createDataFrame(throughputRates, FloatType())

Correct! spark.createDataFrame is the correct operator to use here and the type FloatType() which is passed in for the command's schema argument is correctly instantiated using the parentheses.

Remember that it is essential in PySpark to instantiate types when passing them to SparkSession.createDataFrame. And, in Databricks, spark returns a SparkSession object.

spark.createDataFrame((throughputRates), FloatType)

No. While packing throughputRates in parentheses does not do anything to the execution of this command, not instantiating the FloatType with parentheses as in the previous answer will make this

command fail.

spark.createDataFrame(throughputRates, FloatType)

Incorrect. Given that it does not matter whether you pass throughputRates in parentheses or not, see the explanation of the previous answer for further insights.

spark.DataFrame(throughputRates, FloatType)

Wrong. There is no SparkSession.DataFrame() method in Spark.

spark.createDataFrame(throughputRates)

False. Avoiding the schema argument will have PySpark try to infer the schema. However, as you can see in the documentation (linked below), the inference will only work if you pass in an 'RDD of

either Row, namedtuple, or dict' for data (the first argument to createDataFrame). But since you are passing a Python list, Spark's schema inference will fail.

More info: pyspark.sql.SparkSession.createDataFrame --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 55 (Databricks import instructions)