## Question 1

**Question Type:** **MultipleChoice**

Which of the following code blocks returns the number of unique values in column storeId of DataFrame transactionsDf?

### Options:

**A-** transactionsDf.select('storeId').dropDuplicates().count()

**B-** transactionsDf.select(count('storeId')).dropDuplicates()

**C-** transactionsDf.select(distinct('storeId')).count()

**D-** transactionsDf.dropDuplicates().agg(count('storeId'))

**E-** transactionsDf.distinct().select('storeId').count()

### Answer:

A

### Explanation:

transactionsDf.select('storeId').dropDuplicates().count()

Correct! After dropping all duplicates from column storeId, the remaining rows get counted, representing the number of unique values in the column.

transactionsDf.select(count('storeId')).dropDuplicates()

No. transactionsDf.select(count('storeId')) just returns a single-row DataFrame showing the number of non-null rows. dropDuplicates() does not have any effect in this context.

transactionsDf.dropDuplicates().agg(count('storeId'))

Incorrect. While transactionsDf.dropDuplicates() removes duplicate rows from transactionsDf, it does not do so taking only column storeId into consideration, but eliminates full row duplicates

instead.

transactionsDf.distinct().select('storeId').count()

Wrong. transactionsDf.distinct() identifies unique rows across all columns, but not only unique rows with respect to column storeId. This may leave duplicate values in the column, making the count

not represent the number of unique values in that column.

transactionsDf.select(distinct('storeId')).count()

False. There is no distinct method in pyspark.sql.functions.

## Question 2

**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)

## Question 3

**Question Type:** **MultipleChoice**

The code block shown below should return a DataFrame with all columns of DataFrame transactionsDf, but only maximum 2 rows in which column productId has at least the value 2. Choose the

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

transactionsDf.__1__(__2__).__3__

### Options:

**A-** 1. where
2. 'productId' > 2

3. max(2)

**B-** 1. where
2. transactionsDf[productId] >= 2
3. limit(2)

**C-** 1. filter
2. productId > 2
3. max(2)

**D-** 1. filter
2. col('productId') >= 2
3. limit(2)

**E-** 1. where
2. productId >= 2
3. limit(2)

## Answer:

D

## Explanation:

Correct code block:

transactionsDf.filter(col('productId') >= 2).limit(2)

The filter and where operators in gap 1 are just aliases of one another, so you cannot use them to pick the right answer.

The column definition in gap 2 is more helpful. The DataFrame.filter() method takes an argument of type Column or str. From all possible answers, only the one including col('productId') >= 2 fits

this profile, since it returns a Column type.

The answer option using 'productId' > 2 is invalid, since Spark does not understand that 'productId' refers to column productId. The answer option using transactionsDf[productId] >= 2 is wrong

because you cannot refer to a column using square bracket notation in Spark (if you are coming from Python using Pandas, this is something to watch out for). In all other options, productId is being

referred to as a Python variable, so they are relatively easy to eliminate.

Also note that the Question: asks for the value in column productId being at least 2. This translates to a 'greater or equal' sign (>= 2), but not a 'greater' sign (> 2).

Another thing worth noting is that there is no DataFrame.max() method. If you picked any option including this, you may be confusing it with the pyspark.sql.functions.max method. The correct

method to limit the amount of rows is the DataFrame.limit() method.

More info:

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

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

Static notebook | Dynamic notebook: See test 3, Question: 54 (Databricks import instructions)

## Question 4

**Question Type: MultipleChoice**

Which of the following code blocks returns a single-row DataFrame that only has a column corr which shows the Pearson correlation coefficient between columns predError and value in DataFrame

transactionsDf?

**Options:**

**A-** transactionsDf.select(corr(['predError', 'value']).alias('corr')).first()

**B-** transactionsDf.select(corr(col('predError'), col('value')).alias('corr')).first()

**C-** transactionsDf.select(corr(predError, value).alias('corr'))

**D-** transactionsDf.select(corr(col('predError'), col('value')).alias('corr'))
(Correct)

**E-** transactionsDf.select(corr('predError', 'value'))

## Answer:

D

## Explanation:

In difficulty, this Question: is above what you can expect from the exam. What this Question: wants to teach you, however, is to pay attention to the useful details included in the

documentation.

pyspark.sql.corr is not a very common method, but it deals with Spark's data structure in an interesting way. The command takes two columns over multiple rows and returns a single row - similar to

an aggregation function. When examining the documentation (linked below), you will find this code example:

```
a = range(20)

b = [2 * x for x in range(20)]

df = spark.createDataFrame(zip(a, b), ['a', 'b'])

df.agg(corr('a', 'b').alias('c')).collect()

[Row(c=1.0)]
```

See how corr just returns a single row? Once you understand this, you should be suspicious about answers that include first(), since there is no need to just select a single row. A reason to eliminate those answers is that DataFrame.first() returns an object of type Row, but not DataFrame, as requested in the question.

transactionsDf.select(corr(col('predError'), col('value')).alias('corr'))

Correct! After calculating the Pearson correlation coefficient, the resulting column is correctly renamed to corr.

transactionsDf.select(corr(predError, value).alias('corr'))

No. In this answer, Python will interpret column names predError and value as variable names.

transactionsDf.select(corr(col('predError'), col('value')).alias('corr')).first()

Incorrect. first() returns a row, not a DataFrame (see above and linked documentation below).

transactionsDf.select(corr('predError', 'value'))

Wrong. Whie this statement returns a DataFrame in the desired shape, the column will have the name corr(predError, value) and not corr.

transactionsDf.select(corr(['predError', 'value']).alias('corr')).first()

False. In addition to first() returning a row, this code block also uses the wrong call structure for command corr which takes two arguments (the two columns to correlate).

More info:

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

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

Static notebook | Dynamic notebook: See test 3, Question: 53 (Databricks import instructions)

## Question 5

**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 6

**Question Type: MultipleChoice**

Which of the following code blocks reads in the JSON file stored at filePath, enforcing the schema expressed in JSON format in variable json_schema, shown in the code block below?

Code block:

1. json_schema = """

2. {"type": "struct",

```
"fields": [
{
"name": "itemId",
"type": "integer",
"nullable": true,
"metadata": {}
},
{
"name": "supplier",
"type": "string",
"nullable": true,
"metadata": {}
}
]
}
```

18. """"

## Options:

A- spark.read.json(filePath, schema=json_schema)

B- spark.read.schema(json_schema).json(filePath)

1. schema = StructType.fromJson(json.loads(json_schema))
2. spark.read.json(filePath, schema=schema)

C- spark.read.json(filePath, schema=schema_of_json(json_schema))

D- spark.read.json(filePath, schema=spark.read.json(json_schema))

## Answer:

B

## Explanation:

Spark provides a way to digest JSON-formatted strings as schema. However, it is not trivial to use. Although slightly above exam difficulty, this Question: is beneficial to your exam

preparation, since

it helps you to familiarize yourself with the concept of enforcing schemas on data you are reading in - a topic within the scope of the exam.

The first answer that jumps out here is the one that uses spark.read.schema instead of spark.read.json. Looking at the documentation of spark.read.schema (linked below), we notice that the

operator expects types pyspark.sql.types.StructType or str as its first argument. While variable json_schema is a string, the documentation states that the str should be 'a DDL-formatted string (For

example col0 INT, col1 DOUBLE)'. Variable json_schema does not contain a string in this type of format, so this answer option must be wrong.

With four potentially correct answers to go, we now look at the schema parameter of spark.read.json() (documentation linked below). Here, too, the schema parameter expects an input of type

pyspark.sql.types.StructType or 'a DDL-formatted string (For example col0 INT, col1 DOUBLE)'. We already know that json_schema does not follow this format, so we should focus on how we can

transform json_schema into pyspark.sql.types.StructType. Hereby, we also eliminate the option where schema=json_schema.

The option that includes schema=spark.read.json(json_schema) is also a wrong pick, since spark.read.json returns a DataFrame, and not a pyspark.sql.types.StructType type.

Ruling out the option which includes schema_of_json(json_schema) is rather difficult. The operator's documentation (linked below) states that it '[p]arses a JSON string and infers its schema in DDL

format'. This use case is slightly different from the case at hand: json_schema already is a schema definition, it does not make sense to 'infer' a schema from it. In the documentation you can see an example use case which helps you understand the difference better. Here, you pass string '{a: 1}' to schema_of_json() and the method infers a DDL-format schema STRUCT from it.

In our case, we may end up with the output schema of schema_of_json() describing the schema of the JSON schema, instead of using the schema itself. This is not the right answer option.

Now you may consider looking at the StructType.fromJson() method. It returns a variable of type StructType - exactly the type which the schema parameter of spark.read.json expects.

Although we could have looked at the correct answer option earlier, this explanation is kept as exhaustive as necessary to teach you how to systematically eliminate wrong answer options.

More info:

- pyspark.sql.DataFrameReader.schema --- PySpark 3.1.2 documentation

- pyspark.sql.DataFrameReader.json --- PySpark 3.1.2 documentation

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

Static notebook | Dynamic notebook: See test 3, Question: 51 (Databricks import instructions)

## Question 7

**Question Type: MultipleChoice**

The code block shown below should return the number of columns in the CSV file stored at location filePath. From the CSV file, only lines should be read that do not start with a # character. Choose

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

Code block:

```
__1__(__2__.__3__.csv(filePath, __4__).__5__)
```

## Options:

**A-** 1. size
2. spark
3. read()
4. escape='#'
5. columns

**B-** 1. DataFrame
2. spark
3. read()
4. escape='#'
5. shape[0]

**C-** 1. len
2. pyspark
3. DataFrameReader

4. comment='#'
5. columns

D- 1. size
2. pyspark
3. DataFrameReader
4. comment='#'
5. columns

E- 1. len
2. spark
3. read
4. comment='#'
5. columns

## Answer:

E

## Explanation:

Correct code block:

len(spark.read.csv(filePath, comment='#').columns)

This is a challenging Question: with difficulties in an unusual context: The boundary between DataFrame and the DataFrameReader. It is unlikely that a Question: of this difficulty level

appears in the

exam. However, solving it helps you get more comfortable with the DataFrameReader, a subject you will likely have to deal with in the exam.

Before dealing with the inner parentheses, it is easier to figure out the outer parentheses, gaps 1 and 5. Given the code block, the object in gap 5 would have to be evaluated by the object in gap 1,

returning the number of columns in the read-in CSV. One answer option includes DataFrame in gap 1 and shape[0] in gap 2. Since DataFrame cannot be used to evaluate shape[0], we can discard

this answer option.

Other answer options include size in gap 1. size() is not a built-in Python command, so if we use it, it would have to come from somewhere else. pyspark.sql.functions includes a size() method, but

this method only returns the length of an array or map stored within a column (documentation linked below). So, using a size() method is not an option here. This leaves us with two potentially valid

answers.

We have to pick between gaps 2 and 3 being spark.read or pyspark.DataFrameReader. Looking at the documentation (linked below), the DataFrameReader is actually a child class of pyspark.sql, which means that we cannot import it using pyspark.DataFrameReader. Moreover, spark.read makes sense because on Databricks, spark references current Spark session

(pyspark.sql.SparkSession) and spark.read therefore returns a DataFrameReader (also see documentation below). Finally, there is only one correct answer option remaining.

More info:

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

- pyspark.sql.DataFrameReader.csv --- PySpark 3.1.2 documentation

- pyspark.sql.SparkSession.read --- PySpark 3.1.2 documentation

Static notebook | Dynamic notebook: See test 3, Question: 50 (Databricks import instructions)

## Question 8

**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 9

**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 10

**Question Type: MultipleChoice**

The code block shown below should return a two-column DataFrame with columns transactionId and supplier, with combined information from DataFrames itemsDf and transactionsDf. The code

block should merge rows in which column productId of DataFrame transactionsDf matches the value of column itemId in DataFrame itemsDf, but only where column storeId of DataFrame

transactionsDf does not match column itemId of DataFrame itemsDf. Choose the answer that correctly fills the blanks in the code block to accomplish this.

Code block:

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

## Options:

**A-** 1. join

2. transactionsDf.productId==itemsDf.itemId, how='inner'

3. select

4. 'transactionId', 'supplier'

**B-** 1. select

2. 'transactionId', 'supplier'

3. join

4. [transactionsDf.storeId!=itemsDf.itemId, transactionsDf.productId==itemsDf.itemId]

**C-** 1. join

2. [transactionsDf.productId==itemsDf.itemId, transactionsDf.storeId!=itemsDf.itemId]

3. select

4. 'transactionId', 'supplier'

**D-** 1. filter

2. 'transactionId', 'supplier'

3. join

4. 'transactionsDf.storeId!=itemsDf.itemId, transactionsDf.productId==itemsDf.itemId'

**E-** 1. join

2. transactionsDf.productId==itemsDf.itemId, transactionsDf.storeId!=itemsDf.itemId

3. filter

4. 'transactionId', 'supplier'

## Answer:

C

## Explanation:

This Question: is pretty complex and, in its complexity, is probably above what you would encounter in the exam. However, reading the Question: carefully, you can use your logic skills

to weed out the

wrong answers here.

First, you should examine the join statement which is common to all answers. The first argument of the join() operator (documentation linked below) is the DataFrame to be joined with. Where join is

in gap 3, the first argument of gap 4 should therefore be another DataFrame. For none of the questions where join is in the third gap, this is the case. So you can immediately discard two answers.

For all other answers, join is in gap 1, followed by .(itemsDf, according to the code block. Given how the join() operator is called, there are now three remaining candidates.

Looking further at the join() statement, the second argument (on=) expects 'a string for the join column name, a list of column names, a join expression (Column), or a list of Columns', according to

the documentation. As one answer option includes a list of join expressions (transactionsDf.productId==itemsDf.itemId, transactionsDf.storeId!=itemsDf.itemId) which is unsupported according to the documentation, we can discard that answer, leaving us with two remaining candidates.

Both candidates have valid syntax, but only one of them fulfills the condition in the Question: 'only where column storeId of DataFrame transactionsDf does not match column itemId of

DataFrame

itemsDf'. So, this one remaining answer option has to be the correct one!

As you can see, although sometimes overwhelming at first, even more complex questions can be figured out by rigorously applying the knowledge you can gain from the documentation during the

exam.

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

Static notebook | Dynamic notebook: See test 3, Question: 47 (Databricks import instructions)

20%
DISCOUNT