Which of the following Spark config properties represents the number of partitions used in wide transformations like **join()**?

- ○ **spark.shuffle.file.buffer**
- ○ **spark.shuffle.io.maxRetries**
- [✓ **spark.sql.shuffle.partitions**]
- ○ **spark.default.parallelism**
- ○ **spark.shuffle.partitions**

The code block shown below contains an error. The code block is intended to write DataFrame **storesDF** to file path **filePath** as parquet and partition by values in column **division**. Identify the error.

Code block:

```
storesDF.write.repartition("division").parquet(filePath)
```

- ○ **There is no parquet() operation for DataFrameWriter — the save() operation should be used instead.**
- ○ **The mode() operation must be called to specify that this write should not overwrite existing files.**
- [✓ **There is no repartition() operation for DataFrameWriter — the partitionBy() operation should be used instead.**]
- ○ **DataFrame.write is an operation — it should be followed by parentheses to return a DataFrameWriter.**

Which of the following operations can be used to sort the rows of a DataFrame?

- [✓ **sort() and orderBy()**]
- ○ **orderby()**
- ○ **sort() and orderby()**
- ○ **sort()**
- ○ **orderBy()**

Which of the following code blocks returns a DataFrame where rows in DataFrame **storesDF** containing missing values in every column have been dropped?

- ○ `storesDF.dropna()`
- ○ `storesDF.na.drop()`
- [✓ `storesDF.na.drop("all")`]
- ○ `storesDF.na.drop("all", subset = "sqft")`
- ○ `storesDF.nadrop("all")`

Which of the following DataFrame operations is always classified as a narrow transformation?

- [✓ **DataFrame.select()**]
- ○ **DataFrame.distinct()**
- ○ **DataFrame.sort()**
- ○ **DataFrame.repartition()**
- ○ **DataFrame.join()**

Which of the following describes nodes in cluster-mode Spark?

- ○ **Nodes are the most granular level of execution in the Spark execution hierarchy.**
- ○ **There are driver nodes and worker nodes, both of which can scale horizontally.**
- ○ **Nodes are another term for executors, so they are processing engine instances for performing computations.**
- ○ **There is only one node and it hosts both the driver and executors.**
- [✓ **Worker nodes are machines that host the executors responsible for the execution of tasks.**]

Which of the following code blocks will **not always** return the exact number of distinct values in column division?

- ○ `storesDF.select("division").dropDuplicates().count()`
- [✓ `storesDF.agg(approx_count_distinct(col("division")).alias("divisionDistinct"))`]
- ○ `storesDF.select("division").distinct().count()`
- ○ `storesDF.agg(countDistinct(col("division")).alias("divisionDistinct"))`
- ○ `storesDF.agg(approx_count_distinct(col("division"),0).alias("divisionDistinct"))`

The code block shown below should return a new DataFrame that is the result of a cross join between DataFrame **storesDF** and DataFrame **employeesDF**. Choose the response that correctly fills in the numbered blanks within the code block to complete this task.

Code block:

`__1__.__2__(__3__)`

- ○ **1. storesDF**
  2. join
  3. employeesDF, "cross"
- ○ **1. storesDF**
  2. crossJoin
  3. employeesDF, "storeId"
- [✓ **1. storesDF**
  2. crossJoin
  3. employeesDF, "storeId"]
- ○ **1. storesDF**
  2. crossJoin
  3. employeesDF
- ○ **1. storesDF**
  2. join
  3. employeesDF, "storeId", "cross"

Which of the following code blocks returns a new DataFrame from DataFrame **storesDF** where column **numberOfManagers** is the constant integer 1?

- ○ `storesDF.withColumn("numberOfManagers", lit("1"))`
- ○ `storesDF.withColumn("numberOfManagers", 1)`
- ○ `storesDF.withColumn("numberOfManagers", IntegerType(1))`
- [✓ `storesDF.withColumn("numberOfManagers", lit(1))`]
- ○ `storesDF.withColumn("numberOfManagers", col(1))`

Which of the following code blocks returns a new DataFrame from DataFrame **storesDF** where column **numberOfManagers** is the constant integer 1?

- ○ `storesDF.withColumn("numberOfManagers", lit("1"))`
- ○ `storesDF.withColumn("numberOfManagers", 1)`
- ○ `storesDF.withColumn("numberOfManagers", IntegerType(1))`
- [✓ `storesDF.withColumn("numberOfManagers", lit(1))`]
- ○ `storesDF.withColumn("numberOfManagers", col(1))`