Ministry of Higher Education and Scientific Research
University of Abdelhafid Boussouf - Mila
Institute of Mathematics and Computer Science
Department of Computer Science
Master 2 I2A – Big Data
2025/2026

Directed Works TD 6 – Data Cleaning, Joins, Window Functions & Intro to MLlib (Solution)

Exercise 1: Data Cleaning & Schema Reasoning

You are given a DataFrame sales with columns:
Product, Category, Price, Quantity, Region
(loaded from CSV → all strings)

Question 1: Infer the schema after cleaning

```
sales2 = sales.withColumn("Price", col("Price").cast("double")) \
              .withColumn("Quantity", col("Quantity").cast("int")) \
              .withColumn("Region", upper(trim(col("Region"))))
```

Write the resulting schema:

```
Column        Type
Product       string
Category      string
Price         double
Quantity      int
Region        string
```

- Price cast → double
- Quantity cast → int
- Region stays string (transformed but type unchanged)

Question 2: Missing data count (complete the code)

```
from pyspark.sql.functions import sum, col

missing = sales.select(
    sum(col("Price").isNull().cast("int")).alias("MissingPrice"),
    sum(col("Quantity").isNull().cast("int")).alias("MissingQuantity"),
)
missing.show()
```

.show() is the action used to display the computed counts.

Question 3: Mean vs median imputation (short reasoning)
Median is more robust to extreme values.
A highly skewed distribution has outliers that pull the mean strongly, but not the median.
So median gives a more realistic central value for imputation.

Question 4: Write a filter expression:

```
df.filter((col("Price") > 0) & (col("Price") <= 10000))
```

Exercise 2: Joins

You have:
• sales_clean
• products(*Product, ProductID, Supplier*)

Question 1:
We keep all rows from sales_clean
✓ Missing products → nulls filled from right side

Question 2: Complete the join code

```
result = sales_clean.join(products, on="Product", how="left")
```

Question 3: Interpretation
Returns products that have NO matching entry in sales_clean.
Left-anti = rows in left that do not match.

Question 4: Broadcast? → YES
Because:
products = tiny (200 rows)
sales_clean = huge (3M rows)
broadcasting small table avoids shuffle → faster join

Exercise 3: Window Functions

Given DataFrame df(*Product, Category, Region, Revenue*).

Question 1: Window specification

```
w = Window.partitionBy("Category").orderBy(col("Revenue").desc())
```

Question 2: Add a rank column

```
ranked = df.withColumn("Rank", rank(col("Revenue").desc()).over(w))
```

Question 3: Running sum window

```
w2 = Window.partitionBy("Region") \
           .orderBy("Product") \
           .rowsBetween(Window.unboundedPreceding, Window.currentRow)
```

Question 4: Explanation
Window functions add new columns but do not remove or group rows.
They compute values over partitions, not between them.

Exercise 4 – MLlib Introduction

Dataset:
Age (int), Salary (double), Purchased (0/1)

Question 1: VectorAssembler inputs

```
assembler = VectorAssembler(
    inputCols=["Age", "Salary"],
    outputCol="features"
)
```

Question 2: Write the 3 MLlib steps (in order)

1. Assemble the features → assembler.transform()
2. Create the model → LogisticRegression()
3. Fit the model → .fit()

Question 3: Probability vs prediction

- probability = the model's confidence for class 0 or 1
- prediction = the final class (0 or 1), chosen by threshold

Question 4: Two advantages of DataFrame-based MLlib

1. Catalyst optimizer → faster execution
2. Integrated with DataFrames → simpler, fewer lines of code
3. Pipeline support → easier workflow

(Any 2 correct)

Exercise 5 – True / False

a) TRUE — window partitioning requires shuffle
b) TRUE — show() triggers computation

c) TRUE — broadcast avoids shuffle in joins
d) FALSE — vectors can be dense or sparse
e) TRUE − SQL → Catalyst → physical plan