```
from pyspark.sql import SparkSession, functions as F, types as T, Window

spark = SparkSession.builder.appName("OnlineOrdersPipeline").config("spark.sql.shuffle.partitions","8").getOrCreate()

orders_data = [
("O001","Delhi ","Laptop","45000","2024-01-05","Completed"),
("O002","Mumbai","Mobile ","32000","05/01/2024","Completed"),
("O003","Bangalore","Tablet","30000","2024/01/06","Completed"),
("O004","Delhi","Laptop","","2024-01-07","Cancelled"),
("O005","Mumbai","Mobile","invalid","2024-01-08","Completed"),
("O006","Chennai","Tablet",None,"2024-01-08","Completed"),
("O007","Delhi","Laptop","47000","09-01-2024","Completed"),
("O008","Bangalore","Mobile","28000","2024-01-09","Completed"),
("O009","Mumbai","Laptop","55000","2024-01-10","Completed"),
("O009","Mumbai","Laptop","55000","2024-01-10","Completed")
]
```

```
#TASK1
orders_schema = T.StructType([
    T.StructField("order_id",  T.StringType(), True),
    T.StructField("city",      T.StringType(), True),
    T.StructField("product",   T.StringType(), True),
    T.StructField("amount",    T.StringType(), True),
    T.StructField("order_date",T.StringType(), True),
    T.StructField("status",    T.StringType(), True),
])
```

```
#TASK2
orders_raw = spark.createDataFrame(orders_data, orders_schema)
```

```
#TASK3
orders_raw.printSchema()
```

```
root
 |-- order_id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- product: string (nullable = true)
 |-- amount: string (nullable = true)
 |-- order_date: string (nullable = true)
 |-- status: string (nullable = true)
```

PHASE 2

```
#TASK4 &TASK5
trim_norm = lambda c: F.upper(F.trim(F.regexp_replace(F.col(c), r"\s+", " ")))
orders_t4 = (orders_raw
             .withColumn("order_id", trim_norm("order_id"))
             .withColumn("city",     trim_norm("city"))
             .withColumn("product",  trim_norm("product"))
             .withColumn("status",   trim_norm("status")))
```

```
#TASK6
to_amount_int = lambda c: F.when(F.trim(F.col(c)).rlike(r"^\d+$"), F.trim(F.col(c)).cast("int")).otherwise(F.lit(None).cast("int
orders_t6 = orders_t4.withColumn("amount_int", to_amount_int("amount"))
```

```
#TASK7
orders_t7 = orders_t6.withColumn("amount_invalid", F.col("amount_int").isNull())
```

```
#8
orders_dedup_exact = orders_t8.dropDuplicates()
```

```
#df=df.filter(F.col("status")=="Completed")
#9
df = spark.createDataFrame(orders_data, orders_schema)
df_completed = df.filter(F.upper(F.trim(F.col("status"))).like("COMP%"))

df_completed.show(truncate=False)
```

```
+--------+---------+-------+-------+----------+---------+
|order_id|city     |product|amount |order_date|status   |
+--------+---------+-------+-------+----------+---------+
|O001    |Delhi    |Laptop |45000  |2024-01-05|Completed|
|O002    |Mumbai   |Mobile |32000  |05/01/2024|Completed|
|O003    |Bangalore|Tablet |30000  |2024/01/06|Completed|
|O005    |Mumbai   |Mobile |invalid|2024-01-08|Completed|
|O006    |Chennai  |Tablet |NULL   |2024-01-08|Completed|
|O007    |Delhi    |Laptop |47000  |09-01-2024|Completed|
|O008    |Bangalore|Mobile |28000  |2024-01-09|Completed|
|O009    |Mumbai   |Laptop |55000  |2024-01-10|Completed|
|O009    |Mumbai   |Laptop |55000  |2024-01-10|Completed|
+--------+---------+-------+-------+----------+---------+
```

```
#TASK 10,TASK 11,TASK12
from pyspark.sql import functions as F

amount_int = F.when(F.trim(F.col("amount")).rlike(r"^\d+$"), F.trim(F.col("amount")).cast("int"))

clean_view = (
    orders_raw
    .withColumn("amount_int", amount_int)
    .withColumn("status_norm", F.upper(F.trim(F.col("status"))))
    .withColumn("city_norm",   F.upper(F.trim(F.col("city"))))
    .withColumn("product_norm",F.upper(F.trim(F.col("product"))))
    .filter(F.col("status_norm").like("COMP%"))
    .filter(F.col("amount_int").isNotNull())
)

print("Total revenue per city:")
clean_view.groupBy("city_norm").agg(F.sum("amount_int").alias("total_revenue")).show()

print("Total revenue per product:")
clean_view.groupBy("product_norm").agg(F.sum("amount_int").alias("total_revenue")).show()

print("Average order value per city:")
clean_view.groupBy("city_norm").agg(F.avg("amount_int").alias("avg_order_value")).show()
```

```
Total revenue per city:
+---------+-------------+
|city_norm|total_revenue|
+---------+-------------+
|    DELHI|        92000|
|BANGALORE|        58000|
|   MUMBAI|       142000|
+---------+-------------+

Total revenue per product:
+------------+-------------+
|product_norm|total_revenue|
+------------+-------------+
|      LAPTOP|       202000|
|      MOBILE|        60000|
|      TABLET|        30000|
+------------+-------------+

Average order value per city:
+---------+------------------+
|city_norm|   avg_order_value|
+---------+------------------+
|    DELHI|           46000.0|
|BANGALORE|           29000.0|
|   MUMBAI|47333.333333333336|
+---------+------------------+
```

```
#TASK 13,TASK 14
from pyspark.sql import functions as F, Window
```

```
df_clean = (df
    .withColumn("amount_int", F.when(F.col("amount").rlike(r"^\d+$"), F.col("amount").cast("int")))
    .filter(F.upper(F.col("status")).like("COMP%"))
    .filter(F.col("amount_int").isNotNull())
)

rev_city = df_clean.groupBy("city").agg(F.sum("amount_int").alias("total_revenue"))

w = Window.orderBy(F.col("total_revenue").desc())
ranked_cities = rev_city.withColumn("rank", F.dense_rank().over(w))

print("Rank cities by total revenue:")
ranked_cities.show()

print("Top-performing city:")
ranked_cities.filter(F.col("rank")==1).show()
```

```
Rank cities by total revenue:
+---------+-------------+----+
|     city|total_revenue|rank|
+---------+-------------+----+
|   Mumbai|       142000|   1|
|Bangalore|        58000|   2|
|    Delhi|        47000|   3|
|   Delhi |        45000|   4|
+---------+-------------+----+

Top-performing city:
+------+-------------+----+
|  city|total_revenue|rank|
+------+-------------+----+
|Mumbai|       142000|   1|
+------+-------------+----+
```

```
#TASK 15
from pyspark.sql import functions as F

df_cached = df_clean.cache()
print("Rows after cache materialization:", df_cached.count())
```

```
Rows after cache materialization: 7
```

```
#TASK16
rev_city = df_cached.groupBy("city").agg(F.sum("amount_int").alias("total_revenue"))
rev_city.show()

rev_product = df_cached.groupBy("product").agg(F.sum("amount_int").alias("total_revenue"))
rev_product.show()
```

```
+---------+-------------+
|     city|total_revenue|
+---------+-------------+
|Bangalore|        58000|
|   Delhi |        45000|
|   Mumbai|       142000|
|    Delhi|        47000|
+---------+-------------+

+-------+-------------+
|product|total_revenue|
+-------+-------------+
| Laptop|       202000|
| Tablet|        30000|
|Mobile |        32000|
| Mobile|        28000|
+-------+-------------+
```

```
#TASK 17

print("\n== Plan: df_cached ==")
df_cached.explain(True)

print("\n== Plan: rev_city ==")
```

```
rev_city.explain(True)

print("\n== Plan: rev_product ==")
rev_product.explain(True)
```

```
      +- Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(amount#66, ^\d+$) THEN
         +- LogicalRDD [order_id#63, city#64, product#65, amount#66, order_date#67, status#68], false

== Optimized Logical Plan ==
Aggregate [city#64], [city#64, sum(amount_int#161) AS total_revenue#631L]
+- Project [city#64, amount_int#161]
   +- InMemoryRelation [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, amount_int#161], StorageLevel(d
         +- *(1) Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(amount#66, ^\
            +- *(1) Filter (isnotnull(status#68) AND (StartsWith(upper(status#68), COMP) AND CASE WHEN RLIKE(amount#66, ^\d+$)
               +- *(1) Scan ExistingRDD[order_id#63,city#64,product#65,amount#66,order_date#67,status#68]

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[city#64], functions=[sum(amount_int#161)], output=[city#64, total_revenue#631L])
   +- Exchange hashpartitioning(city#64, 8), ENSURE_REQUIREMENTS, [plan_id=700]
      +- HashAggregate(keys=[city#64], functions=[partial_sum(amount_int#161)], output=[city#64, sum#748L])
         +- InMemoryTableScan [city#64, amount_int#161]
               +- InMemoryRelation [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, amount_int#161], Sto
                     +- *(1) Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(ar
                        +- *(1) Filter (isnotnull(status#68) AND (StartsWith(upper(status#68), COMP) AND CASE WHEN RLIKE(amount
                           +- *(1) Scan ExistingRDD[order_id#63,city#64,product#65,amount#66,order_date#67,status#68]


== Plan: rev_product ==
== Parsed Logical Plan ==
'Aggregate ['product], ['product, 'sum('amount_int) AS total_revenue#827]
+- Filter isnotnull(amount_int#161)
   +- Filter upper(status#68) LIKE COMP%
      +- Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(amount#66, ^\d+$) THEN
         +- LogicalRDD [order_id#63, city#64, product#65, amount#66, order_date#67, status#68], false

== Analyzed Logical Plan ==
product: string, total_revenue: bigint
Aggregate [product#65], [product#65, sum(amount_int#161) AS total_revenue#827L]
+- Filter isnotnull(amount_int#161)
   +- Filter upper(status#68) LIKE COMP%
      +- Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(amount#66, ^\d+$) THEN
         +- LogicalRDD [order_id#63, city#64, product#65, amount#66, order_date#67, status#68], false

== Optimized Logical Plan ==
Aggregate [product#65], [product#65, sum(amount_int#161) AS total_revenue#827L]
+- Project [product#65, amount_int#161]
   +- InMemoryRelation [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, amount_int#161], StorageLevel(d
         +- *(1) Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(amount#66, ^\
            +- *(1) Filter (isnotnull(status#68) AND (StartsWith(upper(status#68), COMP) AND CASE WHEN RLIKE(amount#66, ^\d+$)
               +- *(1) Scan ExistingRDD[order_id#63,city#64,product#65,amount#66,order_date#67,status#68]

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[product#65], functions=[sum(amount_int#161)], output=[product#65, total_revenue#827L])
   +- Exchange hashpartitioning(product#65, 8), ENSURE_REQUIREMENTS, [plan_id=713]
      +- HashAggregate(keys=[product#65], functions=[partial_sum(amount_int#161)], output=[product#65, sum#944L])
         +- InMemoryTableScan [product#65, amount_int#161]
               +- InMemoryRelation [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, amount_int#161], Sto
                     +- *(1) Project [order_id#63, city#64, product#65, amount#66, order_date#67, status#68, CASE WHEN RLIKE(ar
                        +- *(1) Filter (isnotnull(status#68) AND (StartsWith(upper(status#68), COMP) AND CASE WHEN RLIKE(amount
                           +- *(1) Scan ExistingRDD[order_id#63,city#64,product#65,amount#66,order_date#67,status#68]
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```