```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
spark = SparkSession.builder \
.appName('online shopping platform') \
.getOrCreate()
```

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

PHASE 1 — DATA INGESTION & SCHEMA

```
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")
]
columns = ["order_id", "city ","product","price","order_date","status"]
```

1. Define an explicit schema

```
orders_schema = StructType([
    StructField("order_id", StringType(), True),
    StructField("city", StringType(), True),
    StructField("product", StringType(), True),
    StructField("price", StringType(), True),
    StructField("order_date", StringType(), True),
    StructField("status", StringType(), True)
])
```

2. Create a DataFrame using the schema

```
orders_df = spark.createDataFrame(data=orders_data, schema=orders_schema)
```

Print schema and validate data types

```
orders_df.printSchema()
orders_df.show(truncate=False)
```

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

+--------+---------+-------+-------+----------+---------+
|order_id|city     |product|price  |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|
|O004    |Delhi    |Laptop |       |2024-01-07|Cancelled|
|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|
+--------+---------+-------+-------+----------+---------+
```

PHASE 2 — DATA CLEANING

4. Trim all string columns

```
orders_df = orders_df.withColumn("city", lower(trim(col("city")))) \
                     .withColumn("product", lower(trim(col("product"))))

orders_df.show(truncate=False)
orders_df.printSchema()
```

```
+--------+---------+-------+-------+----------+---------+
|order_id|city     |product|price  |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|
|O004    |delhi    |laptop |       |2024-01-07|Cancelled|
|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|
+--------+---------+-------+-------+----------+---------+

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

5. Standardize city and product values

```
print("Distinct City values after standardization:")
orders_df.select("city").distinct().show(truncate=False)

print("Distinct Product values after standardization:")
orders_df.select("product").distinct().show(truncate=False)
```

```
Distinct City values after standardization:
+---------+
|city     |
+---------+
|delhi    |
|bangalore|
|mumbai   |
|chennai  |
+---------+

Distinct Product values after standardization:
+-------+
|product|
+-------+
|mobile |
|tablet |
|laptop |
+-------+
```

6. Convert amount to IntegerType

```
orders_df = orders_df.withColumn("price", col("price").cast(IntegerType()))
orders_df.printSchema()
```

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

7. Handle invalid and null amount values

```
orders_df = orders_df.withColumn(
    "price",
    when(col("price").isNull() | (col("price") == "") | (col("price") == "in
    .otherwise(col("price"))
    .cast(IntegerType())
)
```

```
orders_df.printSchema()
orders_df.show(truncate=False)
```

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

+--------+---------+-------+-----+----------+---------+
|order_id|city     |product|price|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|
|O004    |delhi    |laptop |0    |2024-01-07|Cancelled|
|O005    |mumbai   |mobile |0    |2024-01-08|Completed|
|O006    |chennai  |tablet |0    |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|
+--------+---------+-------+-----+----------+---------+
```

8. Remove duplicate orders

```
orders_df = orders_df.dropDuplicates()
orders_df.printSchema()
orders_df.show(truncate=False)
```

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

+--------+---------+-------+-----+----------+---------+
|order_id|city     |product|price|order_date|status   |
+--------+---------+-------+-----+----------+---------+
|O002    |mumbai   |mobile |32000|05/01/2024|Completed|
|O005    |mumbai   |mobile |0    |2024-01-08|Completed|
|O001    |delhi    |laptop |45000|2024-01-05|Completed|
|O003    |bangalore|tablet |30000|2024/01/06|Completed|
|O008    |bangalore|mobile |28000|2024-01-09|Completed|
|O007    |delhi    |laptop |47000|09-01-2024|Completed|
|O009    |mumbai   |laptop |55000|2024-01-10|Completed|
|O006    |chennai  |tablet |0    |2024-01-08|Completed|
+--------+---------+-------+-----+----------+---------+
```

9. Keep only Completed orders

```
orders_df = orders_df.filter(col("status") == "Completed")
orders_df.printSchema()
orders_df.show(truncate=False)
```

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

+--------+---------+-------+-----+----------+---------+
|order_id|city     |product|price|order_date|status   |
+--------+---------+-------+-----+----------+---------+
|O002    |mumbai   |mobile |32000|05/01/2024|Completed|
|O005    |mumbai   |mobile |0    |2024-01-08|Completed|
|O001    |delhi    |laptop |45000|2024-01-05|Completed|
|O003    |bangalore|tablet |30000|2024/01/06|Completed|
|O008    |bangalore|mobile |28000|2024-01-09|Completed|
|O007    |delhi    |laptop |47000|09-01-2024|Completed|
|O009    |mumbai   |laptop |55000|2024-01-10|Completed|
|O006    |chennai  |tablet |0    |2024-01-08|Completed|
+--------+---------+-------+-----+----------+---------+
```

PHASE 3 — BASIC ANALYTICS

10. Total revenue per city

```
city_revenue_df = orders_df.groupBy("city").agg(sum("price").alias("total_re
city_revenue_df.show()
```

```
+---------+-------------+
|     city|total_revenue|
+---------+-------------+
|  chennai|            0|
|    delhi|        92000|
|bangalore|        58000|
|   mumbai|        87000|
+---------+-------------+
```

11. Total revenue per product

```
product_revenue_df = orders_df.groupBy("product").agg(sum("price").alias("to
product_revenue_df.show()
```

```
+-------+-------------+
|product|total_revenue|
+-------+-------------+
| mobile|        60000|
| tablet|        30000|
| laptop|       147000|
+-------+-------------+
```

12. Average order value per city

```
avg_order_value_df = orders_df.groupBy("city").agg(avg("price").alias("avg_c
avg_order_value_df.show()
```

```
+---------+---------------+
|     city|avg_order_value|
+---------+---------------+
|  chennai|            0.0|
|    delhi|        46000.0|
|bangalore|        29000.0|
|   mumbai|        29000.0|
+---------+---------------+
```

PHASE 4 — WINDOW FUNCTION

13. Rank cities by total revenue

```
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
window_spec = Window.orderBy(col("total_revenue").desc())
ranked_cities_df = city_revenue_df.withColumn("rank", row_number().over(wind
ranked_cities_df.show()
```

```
+---------+-------------+----+
|     city|total_revenue|rank|
+---------+-------------+----+
|    delhi|        92000|   1|
|   mumbai|        87000|   2|
|bangalore|        58000|   3|
|  chennai|            0|   4|
+---------+-------------+----+
```

14. Identify top-performing city

```
top_city_df = ranked_cities_df.orderBy(col("total_revenue").desc()).limit(1)
top_city_df.show()
```

```
+-----+-------------+----+
| city|total_revenue|rank|
+-----+-------------+----+
|delhi|        92000|   1|
+-----+-------------+----+
```

PHASE 5 — PERFORMANCE AWARENESS

15. Cache the cleaned DataFrame

```
orders_df.cache()
orders_df.count()
```

```
8
```

16. Run two aggregations and observe behavior

```
orders_df.groupBy("city").agg(sum("price").alias("total_revenue")).show()
```

```
+---------+-------------+
|     city|total_revenue|
+---------+-------------+
|bangalore|        58000|
|   mumbai|        87000|
|    delhi|        92000|
|  chennai|            0|
+---------+-------------+
```

```
orders_df.groupBy("product").agg(sum("price").alias("total_revenue")).show()
```

```
+-------+-------------+
|product|total_revenue|
+-------+-------------+
| mobile|        60000|
| laptop|       147000|
| tablet|        30000|
+-------+-------------+
```

17. Use explain(True) to inspect the plan

```
orders_df.explain(True)
```

```
== Parsed Logical Plan ==
'Project [unresolvedstarwithcolumns(price, cast('price as int), None)]
+- Filter (status#5 = Completed)
   +- Deduplicate [city#26, order_id#0, price#47, product#27, order_date#4, s
      +- Filter (status#5 = Completed)
         +- Deduplicate [city#26, order_id#0, price#47, product#27, order_dat
            +- Project [order_id#0, city#26, product#27, cast(CASE WHEN ((isn
               +- Project [order_id#0, city#26, lower(trim(product#2, None))
            +- Project [order_id#0, lower(trim(city#1, None)) AS city#2
               +- LogicalRDD [order_id#0, city#1, product#2, price#3, o

== Analyzed Logical Plan ==
order_id: string, city: string, product: string, price: int, order_date: stri
Project [order_id#0, city#26, product#27, cast(price#47 as int) AS price#132,
+- Filter (status#5 = Completed)
   +- Deduplicate [city#26, order_id#0, price#47, product#27, order_date#4, s
      +- Filter (status#5 = Completed)
         +- Deduplicate [city#26, order_id#0, price#47, product#27, order_dat
            +- Project [order_id#0, city#26, product#27, cast(CASE WHEN ((isn
               +- Project [order_id#0, city#26, lower(trim(product#2, None))
                  +- Project [order_id#0, lower(trim(city#1, None)) AS city#2
                     +- LogicalRDD [order_id#0, city#1, product#2, price#3, o

== Optimized Logical Plan ==
Aggregate [city#26, order_id#0, price#47, product#27, order_date#4, status#5]
+- Project [order_id#0, lower(trim(city#1, None)) AS city#26, lower(trim(prod
   +- Filter (isnotnull(status#5) AND (status#5 = Completed))
      +- LogicalRDD [order_id#0, city#1, product#2, price#3, order_date#4, st

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=true
+- == Final Plan ==
   ResultQueryStage 1
   +- *(2) HashAggregate(keys=[city#26, order_id#0, price#47, product#27, ord
      +- AQEShuffleRead coalesced
         +- ShuffleQueryStage 0
            +- Exchange hashpartitioning(city#26, order_id#0, price#47, produ
               +- *(1) HashAggregate(keys=[city#26, order_id#0, price#47, pro
                  +- *(1) Project [order_id#0, lower(trim(city#1, None)) AS c
                     +- *(1) Filter (isnotnull(status#5) AND (status#5 = Comp
                        +- *(1) Scan ExistingRDD[order_id#0,city#1,product#2,
+- == Initial Plan ==
   HashAggregate(keys=[city#26, order_id#0, price#47, product#27, order_date#
   +- Exchange hashpartitioning(city#26, order_id#0, price#47, product#27, or
      +- HashAggregate(keys=[city#26, order_id#0, price#47, product#27, order
         +- Project [order_id#0, lower(trim(city#1, None)) AS city#26, lower(
            +- Filter (isnotnull(status#5) AND (status#5 = Completed))
               +- Scan ExistingRDD[order_id#0,city#1,product#2,price#3,order_
```

Start coding or generate with AI.