You can follow the steps shown by Sumit Sir in the videos. Just note that the VACUUM and OPTIMIZE commands are not supported in the free edition.

Change Data Feed:
Can follow the same steps as shown by sir.

```
%sql
use catalog deltalake_catalog;
```

Python Code:
spark.sql("USE CATALOG deltalake_catalog")

```
%sql
DROP TABLE IF EXISTS demo;


CREATE TABLE demo (
  id INT,
  name STRING,
  amount DOUBLE
);
```

Python Code:
spark.sql("DROP TABLE IF EXISTS demo")

```
spark.sql("""
   CREATE TABLE demo (
      id INT,
      name STRING,
      amount DOUBLE
   )
""")
```

```
%sql
INSERT INTO demo VALUES (1, 'alice', 100.0), (2, 'bob', 200.0), (3, 'carol',
300.0);
SELECT * FROM demo;
```

Python Code:

```
spark.sql("""
   INSERT INTO demo VALUES
   (1, 'alice', 100.0),
   (2, 'bob', 200.0),
   (3, 'carol', 300.0)
""")

spark.sql("SELECT * FROM demo").show()
```

```
%sql
describe history demo;
```

Python Code:
spark.sql("DESCRIBE HISTORY demo").show(truncate=False)

```
%sql
ALTER TABLE demo SET TBLPROPERTIES (delta.enableChangeDataFeed = true);
```

Python Code:
```
spark.sql("""
   ALTER TABLE demo
   SET TBLPROPERTIES (delta.enableChangeDataFeed = true)
""")
```

```
%sql
describe history demo;
```

Python Code:
spark.sql("DESCRIBE HISTORY demo").show(truncate=False)

```
%sql
-- Update
UPDATE demo SET amount = amount * 1.1 WHERE id = 2;
-- Insert additional
INSERT INTO demo VALUES (4, 'dan', 400.0);
-- Delete
DELETE FROM demo WHERE id = 1;
```

Python Code:
spark.sql("UPDATE demo SET amount = amount * 1.1 WHERE id = 2")

spark.sql("INSERT INTO demo VALUES (4, 'dan', 400.0)")

spark.sql("DELETE FROM demo WHERE id = 1")

```
%sql
SELECT * FROM table_changes('demo', 2)
```

Python Code:
spark.sql("SELECT * FROM table_changes('demo', 2)").show(truncate=False)

```
%sql
SELECT * FROM table_changes('demo', 2, 4)
```

Python Code:

spark.sql("SELECT * FROM table_changes('demo', 2, 4)").show(truncate=False)

```
%sql
DELETE FROM demo WHERE id = 2;
```

Python Code:

spark.sql("DELETE FROM demo WHERE id = 2")

```
%sql
SELECT * FROM table_changes('demo', 2)
```

Python Code:

spark.sql("SELECT * FROM table_changes('demo', 2)").show(truncate=False)

```
%sql
DROP TABLE IF EXISTS demo_deleted_data_batch;


CREATE TABLE IF NOT EXISTS demo_deleted_data_batch (
  id INT,
  name STRING,
  amount DOUBLE,
  _commit_type STRING,
  _commit_version LONG,
  _commit_timestamp TIMESTAMP
)
USING delta;
```

Python Code:

```
spark.sql("DROP TABLE IF EXISTS demo_deleted_data_batch")

spark.sql("""
   CREATE TABLE IF NOT EXISTS demo_deleted_data_batch (
      id INT,
      name STRING,
      amount DOUBLE,
      _commit_type STRING,
      _commit_version LONG,
      _commit_timestamp TIMESTAMP
   )
   USING delta
""")
```

```
%sql
INSERT INTO demo_deleted_data_batch
SELECT id, name, amount, _change_type, _commit_version, _commit_timestamp
FROM table_changes('demo', 2)
```

```
where _change_type = 'delete';
```

Python Code:

```
spark.sql("""
   INSERT INTO demo_deleted_data_batch
   SELECT id, name, amount, _change_type, _commit_version, _commit_timestamp
   FROM table_changes('demo', 2)
   WHERE _change_type = 'delete'
""")
```

```
%sql
select * from demo_deleted_data_batch;
```

Python Code:

```
spark.sql("SELECT * FROM demo_deleted_data_batch").show(truncate=False)
```

```
from pyspark.sql.functions import col
(spark.readStream.format("delta")
    .option("readChangeFeed", "true")
    .option("startingVersion", 2)
    .table("demo")
    .filter(col("_change_type") == "delete")
    .select("id", "name")
    .writeStream
    .outputMode("append")
    .option("checkpointLocation",
"/Volumes/deltalake_catalog/default/delta_volume1/checkpoint/")
    .trigger(availableNow=True)
    .table("demo_deleted_data_streaming"))
```

Above code is already in python

```
%sql
select * from demo_deleted_data_streaming;
```

Python Code:

```
spark.sql("SELECT * FROM demo_deleted_data_streaming").show(truncate=False)
```

```
%sql
DELETE FROM demo WHERE id = 3;
```

Python Code:

```
spark.sql("DELETE FROM demo WHERE id = 3")
```

```
from pyspark.sql.functions import col


(spark.readStream.format("delta")
```

```
    .option("readChangeFeed", "true")
    .option("startingVersion", 2)
    .table("demo")
    .filter(col("_change_type") == "delete")
    .select("id", "name")
    .writeStream
    .outputMode("append")
    .option("checkpointLocation",
"/Volumes/deltalake_catalog/default/delta_volume1/checkpoint1/")
    .trigger(processingTime="2 seconds")
    .table("demo_deleted_data_streaming_auto"))
```

Above code is already in python
(NOTE: processing time is not supported in databricks free edition)

```
%sql
select * from demo_deleted_data_streaming_auto;
```

Python Code:
spark.sql("SELECT * FROM demo_deleted_data_streaming_auto").show(truncate=False)

```
%sql
INSERT INTO demo VALUES
  (6, 'Frank', 600.0),
  (7, 'Grace', 700.0),
  (8, 'Heidi', 800.0),
  (9, 'Ivan', 900.0),
  (10, 'Judy', 1000.0);
```

Python Code:
spark.sql("""
  INSERT INTO demo VALUES
    (6, 'Frank', 600.0),
    (7, 'Grace', 700.0),
    (8, 'Heidi', 800.0),
    (9, 'Ivan', 900.0),
    (10, 'Judy', 1000.0)
""")

```
%sql
DELETE FROM demo WHERE id = 7;
```

Python Code:
spark.sql("DELETE FROM demo WHERE id = 7")

Shallow Clone:

```
%sql
use catalog deltalake_catalog;
```

Python Code:

```
spark.sql("USE CATALOG deltalake_catalog")
```

```
%sql
DROP TABLE IF EXISTS orders_managed;

CREATE OR REPLACE TABLE orders_managed (
order_id BIGINT,
sku STRING,
product_name STRING,
product_category STRING,
qty INT,
unit_price DECIMAL(10,2)
);

ALTER TABLE orders_managed ADD CONSTRAINT valid_qty CHECK (qty > 0);
```

Python Code:

```
spark.sql("DROP TABLE IF EXISTS orders_managed")

spark.sql("""
  CREATE OR REPLACE TABLE orders_managed (
    order_id BIGINT,
    sku STRING,
    product_name STRING,
    product_category STRING,
    qty INT,
    unit_price DECIMAL(10,2)
  )
""")

spark.sql("""
  ALTER TABLE orders_managed
  ADD CONSTRAINT valid_qty CHECK (qty > 0)
""")
```

```
%sql
INSERT INTO orders_managed VALUES
```

```
  (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
  (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
  (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
  (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
  (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99);
```

Python Code:

```
spark.sql("""
   INSERT INTO orders_managed VALUES
     (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
     (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
     (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
     (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
     (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99)
""")
```

```
%sql
INSERT INTO orders_managed VALUES
  (6, 'SKU-6001', 'Bluetooth Headphones', 'Electronics', 1, 2599.00),
  (7, 'SKU-7001', 'Running Shoes', 'Fitness', 2, 3499.00),
  (8, 'SKU-8001', 'Ballpoint Pen Pack', 'Stationery', 10, 99.00)
```

Python Code:

```
spark.sql("""
   INSERT INTO orders_managed VALUES
     (6, 'SKU-6001', 'Bluetooth Headphones', 'Electronics', 1, 2599.00),
     (7, 'SKU-7001', 'Running Shoes', 'Fitness', 2, 3499.00),
     (8, 'SKU-8001', 'Ballpoint Pen Pack', 'Stationery', 10, 99.00)
""")
```

```
%sql
DELETE FROM orders_managed
WHERE order_id = 4;
```

Python Code:

```
spark.sql("DELETE FROM orders_managed WHERE order_id = 4")
```

```
%sql
UPDATE orders_managed
SET qty = 3,
unit_price = 3299.00
WHERE order_id = 7;
```

Python Code:

```
spark.sql("""
   UPDATE orders_managed
   SET qty = 3,
      unit_price = 3299.00
```

```
  WHERE order_id = 7
""")
```

```
%sql
DESCRIBE HISTORY orders_managed
```

Python Code:

```
spark.sql("DESCRIBE HISTORY orders_managed").show(truncate=False)
```

```
%sql
INSERT INTO orders_managed VALUES
  (9, 'SKU-9001', 'Stainless Steel Bottle', 'Kitchen', 2, 499.00),
  (10, 'SKU-1002', 'Smart Watch', 'Electronics', 1, 7499.00)
```

Python Code:

```
spark.sql("""
  INSERT INTO orders_managed VALUES
    (9, 'SKU-9001', 'Stainless Steel Bottle', 'Kitchen', 2, 499.00),
    (10, 'SKU-1002', 'Smart Watch', 'Electronics', 1, 7499.00)
""")
```

```
%sql
describe detail orders_managed;
```

Python Code:

```
spark.sql("DESCRIBE DETAIL orders_managed").show(truncate=False)
```

```
%sql
select * from orders_managed;
```

Python Code:

```
spark.sql("SELECT * FROM orders_managed").show(truncate=False)
```

```
%sql
SELECT DISTINCT _metadata.file_path FROM orders_managed;
```

Python Code:

```
spark.sql("SELECT DISTINCT _metadata.file_path FROM orders_managed").show(truncate=False)
```

```
%sql
DROP TABLE IF EXISTS orders_managed_sclone;


CREATE OR REPLACE TABLE orders_managed_sclone SHALLOW CLONE orders_managed;
DESCRIBE DETAIL orders_managed_sclone;
```

Python Code:

```
spark.sql("DROP TABLE IF EXISTS orders_managed_sclone")

spark.sql("""
  CREATE OR REPLACE TABLE orders_managed_sclone
```

```
  SHALLOW CLONE orders_managed
""")
spark.sql("DESCRIBE DETAIL orders_managed_sclone").show(truncate=False)
```

```
%sql
select * from orders_managed_sclone order by order_id
```

Python Code:

```
spark.sql("SELECT * FROM orders_managed_sclone ORDER BY order_id").show(truncate=False)
```

```
%sql
select * from orders_managed order by order_id
```

Python Code:

```
spark.sql("SELECT * FROM orders_managed ORDER BY order_id").show(truncate=False)
```

```
%sql
SELECT DISTINCT _metadata.file_path FROM orders_managed_sclone;
```

Python Code:

```
spark.sql("SELECT DISTINCT _metadata.file_path FROM
orders_managed_sclone").show(truncate=False)
```

```
%sql
INSERT INTO orders_managed VALUES
  (11, 'SKU-1101', 'Resistance Bands Set', 'Fitness', 1, 899.00),
  (12, 'SKU-1201', 'Sticky Notes Pack', 'Stationery', 6, 59.00)
```

Python Code:

```
spark.sql("""
  INSERT INTO orders_managed VALUES
    (11, 'SKU-1101', 'Resistance Bands Set', 'Fitness', 1, 899.00),
    (12, 'SKU-1201', 'Sticky Notes Pack', 'Stationery', 6, 59.00)
""")
```

```
%sql
describe history orders_managed
```

Python Code:

```
spark.sql("DESCRIBE HISTORY orders_managed").show(truncate=False)
```

```
%sql
describe history orders_managed_sclone
```

Python Code:

```
spark.sql("DESCRIBE HISTORY orders_managed_sclone").show(truncate=False)
```

```
%sql
INSERT INTO orders_managed_sclone VALUES
  (13, 'SKU-1301', 'Non-stick Frying Pan', 'Kitchen', 1, 1299.00)
```

Python Code:

```
spark.sql("""
   INSERT INTO orders_managed_sclone VALUES
    (13, 'SKU-1301', 'Non-stick Frying Pan', 'Kitchen', 1, 1299.00)
""")
```

```
%sql
DELETE FROM orders_managed_sclone
WHERE order_id = 3;
```

Python Code:

```
spark.sql("""
   DELETE FROM orders_managed_sclone
   WHERE order_id = 3
""")
```

```
%sql
describe history orders_managed_sclone
```

Python Code:

```
spark.sql("DESCRIBE HISTORY orders_managed_sclone").show(truncate=False)
```

```
%sql
DELETE FROM orders_managed;
```

Python Code:

```
spark.sql("DELETE FROM orders_managed")
```

```
%sql
VACUUM orders_managed RETAIN 0 HOURS DRY RUN;
```

(NOTE: vacuum is not supported in Databricks free edition)

Python code:

```
spark.sql("VACUUM orders_managed RETAIN 0 HOURS DRY RUN")
```

```
%sql
ALTER TABLE orders_managed
SET TBLPROPERTIES(
  'delta.deletedFileRetentionDuration' = 'interval 0 hours'
)
```

Python Code:

```
spark.sql("""
   ALTER TABLE orders_managed
   SET TBLPROPERTIES (
    'delta.deletedFileRetentionDuration' = 'interval 0 hours'
   )
""")
```

```
%sql
```

```
VACUUM orders_managed DRY RUN;
```

Python Code;
spark.sql("VACUUM orders_managed DRY RUN")

```
%sql
VACUUM orders_managed;
```

Python Code:
spark.sql("VACUUM orders_managed")

```
%sql
select * from orders_managed_sclone order by order_id;
```

Python Code:
spark.sql("SELECT * FROM orders_managed_sclone ORDER BY order_id").show(truncate=False)

```
%sql
select * from orders_managed@v6
```

Python Code:
spark.sql("SELECT * FROM orders_managed VERSION AS OF 6").show(truncate=False)

```
%sql
select distinct _metadata.file_path from orders_managed@v9
```

Python Code:
spark.sql("""
  SELECT DISTINCT _metadata.file_path
  FROM orders_managed VERSION AS OF 9
""").show(truncate=False)

```
%sql
DELETE FROM orders_managed_sclone;
```

Python Code:
spark.sql("DELETE FROM orders_managed_sclone")

```
%sql
ALTER TABLE orders_managed_sclone
SET TBLPROPERTIES(
  'delta.deletedFileRetentionDuration' = 'interval 0 hours'
)
```

Python Code:
spark.sql("""
  ALTER TABLE orders_managed_sclone
  SET TBLPROPERTIES (
   'delta.deletedFileRetentionDuration' = 'interval 0 hours'
)

""")

```
%sql
VACUUM orders_managed_sclone DRY RUN;
```

Python Code:
spark.sql("VACUUM orders_managed_sclone DRY RUN")

```
%sql
restore table orders_managed version as of 8;
```

Python Code:
spark.sql("RESTORE TABLE orders_managed TO VERSION AS OF 8")

```
%sql
drop table orders_managed;
```

Python Code:
spark.sql("DROP TABLE orders_managed")

Deep Cloning:
Most of the codes are similar to shallow clone

DeltaLake Uniform:

```
%sql
DROP TABLE IF EXISTS orders_managed;
CREATE OR REPLACE TABLE orders_managed (
order_id BIGINT,
sku STRING,
product_name STRING,
product_category STRING,
qty INT,
unit_price DECIMAL(10,2)
) TBLPROPERTIES ( 'delta.columnMapping.mode' = 'name',
'delta.enableIcebergCompatV2' = 'true',
'delta.universalFormat.enabledFormats' = 'iceberg'
)
```

Python Code:
spark.sql("DROP TABLE IF EXISTS orders_managed")

spark.sql("""
  CREATE OR REPLACE TABLE orders_managed (
    order_id BIGINT,
    sku STRING,
    product_name STRING,

```
    product_category STRING,
    qty INT,
    unit_price DECIMAL(10,2)
  )
  TBLPROPERTIES (
    'delta.columnMapping.mode' = 'name',
    'delta.enableIcebergCompatV2' = 'true',
    'delta.universalFormat.enabledFormats' = 'iceberg'
  )
""")
```

```
%sql
INSERT INTO orders_managed VALUES
  (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
  (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
  (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
  (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
  (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99);
```

Python Code:

```
spark.sql("""
  INSERT INTO orders_managed VALUES
    (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
    (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
    (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
    (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
    (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99)
""")
```

```
%sql
describe formatted orders_managed
```

Python Code:

```
spark.sql("DESCRIBE FORMATTED orders_managed").show(truncate=False)
```

```
%sql
show tblproperties orders_managed;
```

Python Code:

```
spark.sql("SHOW TBLPROPERTIES orders_managed").show(truncate=False)
```

Manual Optimize:

```
%sql
use catalog deltalake_catalog;
```

Python Code:

```
spark.sql("USE CATALOG deltalake_catalog")
```

```
%sql
```

```
DROP TABLE IF EXISTS orders_managed;


CREATE OR REPLACE TABLE orders_managed (
  order_id BIGINT,
  sku STRING,
  product_name STRING,
  product_category STRING,
  qty INT,
  unit_price DECIMAL(10,2)
)
USING DELTA
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
)
```

Python Code:

```
spark.sql("""
DROP TABLE IF EXISTS orders_managed
""")

spark.sql("""
CREATE OR REPLACE TABLE orders_managed (
 order_id BIGINT,
 sku STRING,
 product_name STRING,
 product_category STRING,
 qty INT,
 unit_price DECIMAL(10,2)
)
USING DELTA
TBLPROPERTIES (
 delta.autoOptimize.optimizeWrite = false,
 delta.autoOptimize.autoCompact = false
)
""")
```

```
%sql
describe detail orders_managed;
```

Python Code:

```
spark.sql("DESCRIBE DETAIL orders_managed").show(truncate=False)
```

```
from pyspark.sql import Row
from pyspark.sql.types import DecimalType
```

```
from pyspark.sql.functions import col
import time
N = 100
for i in range(N):
    row = Row(order_id=int(i+1),
              sku=f"SKU-{(i%10)+1}",
              product_name=f"Product-{(i%50)+1}",
              product_category=f"Category-{(i%5)+1}",
              qty=(i % 10) + 1,
              unit_price=round(10.0 + (i%7)*1.5, 2))
    df = spark.createDataFrame([row])
    df = df.withColumn("qty", col("qty").cast("int")).withColumn("unit_price",
col("unit_price").cast(DecimalType(10,2)))
    df.write.format("delta").mode("append").saveAsTable("orders_managed")
    # optional tiny sleep to mimic real small-batch writes
    # time.sleep(0.02)
print(f"Inserted {N} tiny writes.")
```

Above code is already in python

```
%sql
select * from orders_managed;
```

Python Code:
spark.sql("SELECT * FROM orders_managed").show()

Alternatively, dataframe way:
df_orders = spark.sql("SELECT * FROM orders_managed")

df_orders.show()

```
%sql
select * from orders_managed where order_id = 56;
```

Python Code:
spark.sql("SELECT * FROM orders_managed WHERE order_id = 56").show()

```
query = "select AVG(unit_price) as avg_price from orders_managed"
res = spark.sql(query).collect()
print(res)
```

Above code is already in python

```
%sql
DESCRIBE HISTORY orders_managed;
```

Python Code:
spark.sql("DESCRIBE HISTORY orders_managed").show(truncate=False)

```
%sql
OPTIMIZE orders_managed;
```

Python Code:
spark.sql("OPTIMIZE orders_managed")

```
query = "select AVG(unit_price) as avg_price from orders_managed"
res = spark.sql(query).collect()
print(res)
```

Above code is already in python

Auto Optimize:

```
Code remains the same
```

Demo Partioning:

```
%sql
use catalog deltalake_catalog;
```

Python Code:
spark.sql("USE CATALOG deltalake_catalog")

```
%sql
DROP TABLE IF EXISTS orders_managed;

CREATE OR REPLACE TABLE orders_managed (
  order_id BIGINT,
  sku STRING,
  product_name STRING,
  product_category STRING,
  qty INT,
  unit_price DECIMAL(10,2),
  country STRING
)
USING DELTA;
```

Python Code:
Drop the table if it already exists
spark.sql("""
DROP TABLE IF EXISTS orders_managed
""")

Create or replace the table
spark.sql("""

```
CREATE OR REPLACE TABLE orders_managed (
  order_id BIGINT,
  sku STRING,
  product_name STRING,
  product_category STRING,
  qty INT,
  unit_price DECIMAL(10,2),
  country STRING
)
USING DELTA
""")
```

```
%sql
INSERT INTO orders_managed VALUES
(1, 'SKU-001', 'Widget A', 'Widgets', 2, 199.99, 'IN'),
(2, 'SKU-002', 'Widget B', 'Widgets', 1, 149.99, 'IN'),
(3, 'SKU-003', 'Widget C', 'Widgets', 3, 99.50,  'IN'),
(4, 'SKU-004', 'Widget D', 'Widgets', 5, 75.00,  'IN'),
(5, 'SKU-005', 'Widget E', 'Widgets', 4, 120.00, 'IN'),
(6, 'SKU-006', 'Gadget A', 'Gadgets', 1, 220.00, 'IN'),
(7, 'SKU-007', 'Gadget B', 'Gadgets', 2, 199.00, 'IN'),
(8, 'SKU-008', 'Gadget C', 'Gadgets', 6, 85.00,  'IN'),
(9, 'SKU-009', 'Gadget D', 'Gadgets', 2, 65.50,  'IN'),
(10, 'SKU-010', 'Accessory A', 'Accessories', 10, 5.50, 'IN'),

(11, 'SKU-011', 'Widget F', 'Widgets', 2, 155.00, 'US'),
(12, 'SKU-012', 'Widget G', 'Widgets', 1, 135.00, 'US'),
(13, 'SKU-013', 'Widget H', 'Widgets', 7, 89.99,  'US'),
(14, 'SKU-014', 'Widget I', 'Widgets', 4, 180.00, 'US'),
(15, 'SKU-015', 'Widget J', 'Widgets', 3, 160.00, 'US'),
(16, 'SKU-016', 'Gadget E', 'Gadgets', 2, 99.00, 'US'),
(17, 'SKU-017', 'Gadget F', 'Gadgets', 5, 115.00, 'US'),
(18, 'SKU-018', 'Accessory B', 'Accessories', 12, 8.00, 'US'),
(19, 'SKU-019', 'Accessory C', 'Accessories', 6, 6.50, 'US'),
(20, 'SKU-020', 'Accessory D', 'Accessories', 8, 12.00, 'US'),

(21, 'SKU-021', 'Widget K', 'Widgets', 2, 175.00, 'US'),
(22, 'SKU-022', 'Gadget G', 'Gadgets', 3, 210.00, 'US'),
(23, 'SKU-023', 'Accessory E', 'Accessories', 9, 14.00, 'US'),
(24, 'SKU-024', 'Accessory F', 'Accessories', 11, 9.99, 'US'),
(25, 'SKU-025', 'Widget L', 'Widgets', 4, 140.00, 'US'),
```

```
(26, 'SKU-026', 'Widget M', 'Widgets', 2, 120.00, 'UK'),
(27, 'SKU-027', 'Gadget H', 'Gadgets', 1, 99.00, 'UK'),
(28, 'SKU-028', 'Accessory G', 'Accessories', 5, 6.00, 'UK'),

(29, 'SKU-029', 'Widget N', 'Widgets', 1, 135.00, 'AU'),
(30, 'SKU-030', 'Gadget I', 'Gadgets', 3, 145.00, 'AU'),
(31, 'SKU-031', 'Accessory H', 'Accessories', 2, 12.00, 'AU'),

(32, 'SKU-032', 'Widget O', 'Widgets', 1, 175.00, 'CA'),
(33, 'SKU-033', 'Gadget J', 'Gadgets', 2, 155.00, 'CA'),
(34, 'SKU-034', 'Accessory I', 'Accessories', 4, 11.00, 'CA'),

(35, 'SKU-035', 'Widget P', 'Widgets', 1, 185.00, 'DE'),
(36, 'SKU-036', 'Gadget K', 'Gadgets', 2, 165.00, 'DE'),

(37, 'SKU-037', 'Accessory J', 'Accessories', 6, 7.50, 'IN'),
(38, 'SKU-038', 'Accessory K', 'Accessories', 3, 6.80, 'IN'),
(39, 'SKU-039', 'Accessory L', 'Accessories', 4, 15.00, 'IN'),
(40, 'SKU-040', 'Widget Q', 'Widgets', 2, 190.00, 'IN'),
(41, 'SKU-041', 'Widget R', 'Widgets', 3, 110.00, 'IN'),
(42, 'SKU-042', 'Gadget L', 'Gadgets', 5, 130.00, 'IN'),
(43, 'SKU-043', 'Gadget M', 'Gadgets', 1, 200.00, 'IN'),
(44, 'SKU-044', 'Accessory M', 'Accessories', 9, 10.00, 'IN'),
(45, 'SKU-045', 'Accessory N', 'Accessories', 6, 8.00, 'IN'),

(46, 'SKU-046', 'Widget S', 'Widgets', 1, 175.00, 'US'),
(47, 'SKU-047', 'Gadget N', 'Gadgets', 3, 145.00, 'US'),
(48, 'SKU-048', 'Accessory O', 'Accessories', 2, 12.00, 'US'),
(49, 'SKU-049', 'Accessory P', 'Accessories', 4, 11.00, 'US'),
(50, 'SKU-050', 'Widget T', 'Widgets', 2, 160.00, 'US');
```

Python Code:

```
spark.sql("""
INSERT INTO orders_managed VALUES
(1, 'SKU-001', 'Widget A', 'Widgets', 2, 199.99, 'IN'),
(2, 'SKU-002', 'Widget B', 'Widgets', 1, 149.99, 'IN'),
(3, 'SKU-003', 'Widget C', 'Widgets', 3, 99.50,  'IN'),
(4, 'SKU-004', 'Widget D', 'Widgets', 5, 75.00,  'IN'),
(5, 'SKU-005', 'Widget E', 'Widgets', 4, 120.00, 'IN'),
(6, 'SKU-006', 'Gadget A', 'Gadgets', 1, 220.00, 'IN'),
```

(7, 'SKU-007', 'Gadget B', 'Gadgets', 2, 199.00, 'IN'),
(8, 'SKU-008', 'Gadget C', 'Gadgets', 6, 85.00, 'IN'),
(9, 'SKU-009', 'Gadget D', 'Gadgets', 2, 65.50, 'IN'),
(10, 'SKU-010', 'Accessory A', 'Accessories', 10, 5.50, 'IN'),

(11, 'SKU-011', 'Widget F', 'Widgets', 2, 155.00, 'US'),
(12, 'SKU-012', 'Widget G', 'Widgets', 1, 135.00, 'US'),
(13, 'SKU-013', 'Widget H', 'Widgets', 7, 89.99, 'US'),
(14, 'SKU-014', 'Widget I', 'Widgets', 4, 180.00, 'US'),
(15, 'SKU-015', 'Widget J', 'Widgets', 3, 160.00, 'US'),
(16, 'SKU-016', 'Gadget E', 'Gadgets', 2, 99.00, 'US'),
(17, 'SKU-017', 'Gadget F', 'Gadgets', 5, 115.00, 'US'),
(18, 'SKU-018', 'Accessory B', 'Accessories', 12, 8.00, 'US'),
(19, 'SKU-019', 'Accessory C', 'Accessories', 6, 6.50, 'US'),
(20, 'SKU-020', 'Accessory D', 'Accessories', 8, 12.00, 'US'),

(21, 'SKU-021', 'Widget K', 'Widgets', 2, 175.00, 'US'),
(22, 'SKU-022', 'Gadget G', 'Gadgets', 3, 210.00, 'US'),
(23, 'SKU-023', 'Accessory E', 'Accessories', 9, 14.00, 'US'),
(24, 'SKU-024', 'Accessory F', 'Accessories', 11, 9.99, 'US'),
(25, 'SKU-025', 'Widget L', 'Widgets', 4, 140.00, 'US'),

(26, 'SKU-026', 'Widget M', 'Widgets', 2, 120.00, 'UK'),
(27, 'SKU-027', 'Gadget H', 'Gadgets', 1, 99.00, 'UK'),
(28, 'SKU-028', 'Accessory G', 'Accessories', 5, 6.00, 'UK'),

(29, 'SKU-029', 'Widget N', 'Widgets', 1, 135.00, 'AU'),
(30, 'SKU-030', 'Gadget I', 'Gadgets', 3, 145.00, 'AU'),
(31, 'SKU-031', 'Accessory H', 'Accessories', 2, 12.00, 'AU'),

(32, 'SKU-032', 'Widget O', 'Widgets', 1, 175.00, 'CA'),
(33, 'SKU-033', 'Gadget J', 'Gadgets', 2, 155.00, 'CA'),
(34, 'SKU-034', 'Accessory I', 'Accessories', 4, 11.00, 'CA'),

(35, 'SKU-035', 'Widget P', 'Widgets', 1, 185.00, 'DE'),
(36, 'SKU-036', 'Gadget K', 'Gadgets', 2, 165.00, 'DE'),

(37, 'SKU-037', 'Accessory J', 'Accessories', 6, 7.50, 'IN'),
(38, 'SKU-038', 'Accessory K', 'Accessories', 3, 6.80, 'IN'),
(39, 'SKU-039', 'Accessory L', 'Accessories', 4, 15.00, 'IN'),
(40, 'SKU-040', 'Widget Q', 'Widgets', 2, 190.00, 'IN'),
(41, 'SKU-041', 'Widget R', 'Widgets', 3, 110.00, 'IN'),
(42, 'SKU-042', 'Gadget L', 'Gadgets', 5, 130.00, 'IN'),
(43, 'SKU-043', 'Gadget M', 'Gadgets', 1, 200.00, 'IN'),
(44, 'SKU-044', 'Accessory M', 'Accessories', 9, 10.00, 'IN'),
(45, 'SKU-045', 'Accessory N', 'Accessories', 6, 8.00, 'IN'),

(46, 'SKU-046', 'Widget S', 'Widgets', 1, 175.00, 'US'),
(47, 'SKU-047', 'Gadget N', 'Gadgets', 3, 145.00, 'US'),

(48, 'SKU-048', 'Accessory O', 'Accessories', 2, 12.00, 'US'),
(49, 'SKU-049', 'Accessory P', 'Accessories', 4, 11.00, 'US'),
(50, 'SKU-050', 'Widget T', 'Widgets', 2, 160.00, 'US')
""")

```
%sql
DROP TABLE if exists orders_managed_partitioned;
CREATE TABLE IF NOT EXISTS orders_managed_partitioned (
  order_id BIGINT,
  sku STRING,
  product_name STRING,
  product_category STRING,
  qty INT,
  unit_price DECIMAL(10,2),
  country STRING
)
USING DELTA
PARTITIONED BY (country);
```

Python Code:

```
spark.sql("""
DROP TABLE IF EXISTS orders_managed_partitioned
""")

spark.sql("""
CREATE TABLE IF NOT EXISTS orders_managed_partitioned (
  order_id BIGINT,
  sku STRING,
  product_name STRING,
  product_category STRING,
  qty INT,
  unit_price DECIMAL(10,2),
  country STRING
)
USING DELTA
PARTITIONED BY (country)
""")
```

```
%sql
insert into orders_managed_partitioned
select * from orders_managed;
```

Python Code:

```
spark.sql("""
INSERT INTO orders_managed_partitioned
SELECT * FROM orders_managed
```

""")

```
%sql
INSERT INTO orders_managed_partitioned VALUES
(51, 'SKU-051', 'Widget U', 'Widgets', 2, 145.00, 'IN'),
(52, 'SKU-052', 'Widget V', 'Widgets', 1, 180.00, 'IN'),
(53, 'SKU-053', 'Widget W', 'Widgets', 3, 135.00, 'IN'),
(54, 'SKU-054', 'Gadget O', 'Gadgets', 4, 210.00, 'IN'),
(55, 'SKU-055', 'Gadget P', 'Gadgets', 2, 175.00, 'IN'),
(56, 'SKU-056', 'Accessory Q', 'Accessories', 6, 10.50, 'IN'),
(57, 'SKU-057', 'Accessory R', 'Accessories', 3, 8.25, 'IN'),
(58, 'SKU-058', 'Accessory S', 'Accessories', 5, 11.00, 'IN'),
(59, 'SKU-059', 'Widget X', 'Widgets', 4, 160.00, 'IN'),
(60, 'SKU-060', 'Widget Y', 'Widgets', 2, 190.00, 'IN'),

(61, 'SKU-061', 'Gadget Q', 'Gadgets', 1, 125.00, 'US'),
(62, 'SKU-062', 'Gadget R', 'Gadgets', 2, 135.00, 'US'),
(63, 'SKU-063', 'Gadget S', 'Gadgets', 3, 145.00, 'US'),
(64, 'SKU-064', 'Widget Z', 'Widgets', 5, 165.00, 'US'),
(65, 'SKU-065', 'Widget AA', 'Widgets', 4, 175.00, 'US'),
(66, 'SKU-066', 'Accessory T', 'Accessories', 7, 15.00, 'US'),
(67, 'SKU-067', 'Accessory U', 'Accessories', 6, 10.00, 'US'),
(68, 'SKU-068', 'Accessory V', 'Accessories', 8, 9.50,  'US'),
(69, 'SKU-069', 'Accessory W', 'Accessories', 9, 13.00, 'US'),
(70, 'SKU-070', 'Accessory X', 'Accessories', 11, 7.25, 'US'),

(71, 'SKU-071', 'Widget AB', 'Widgets', 2, 140.00, 'US'),
(72, 'SKU-072', 'Gadget T', 'Gadgets', 3, 200.00, 'US'),
(73, 'SKU-073', 'Accessory Y', 'Accessories', 4, 12.00, 'US'),
(74, 'SKU-074', 'Accessory Z', 'Accessories', 5, 13.50, 'US'),
(75, 'SKU-075', 'Widget AC', 'Widgets', 1, 155.00, 'US'),

(76, 'SKU-076', 'Widget AD', 'Widgets', 1, 175.00, 'UK'),
(77, 'SKU-077', 'Gadget U', 'Gadgets', 2, 165.00, 'UK'),
(78, 'SKU-078', 'Accessory AA', 'Accessories', 3, 8.50, 'UK'),

(79, 'SKU-079', 'Widget AE', 'Widgets', 1, 180.00, 'AU'),
(80, 'SKU-080', 'Gadget V', 'Gadgets', 2, 140.00, 'AU'),
```

```
(81, 'SKU-081', 'Accessory AB', 'Accessories', 4, 6.75, 'AU'),

(82, 'SKU-082', 'Widget AF', 'Widgets', 1, 195.00, 'CA'),
(83, 'SKU-083', 'Gadget W', 'Gadgets', 2, 150.00, 'CA'),
(84, 'SKU-084', 'Accessory AC', 'Accessories', 3, 10.00, 'CA'),

(85, 'SKU-085', 'Widget AG', 'Widgets', 1, 205.00, 'DE'),
(86, 'SKU-086', 'Gadget X', 'Gadgets', 2, 160.00, 'DE'),

(87, 'SKU-087', 'Accessory AD', 'Accessories', 6, 9.50, 'IN'),
(88, 'SKU-088', 'Accessory AE', 'Accessories', 3, 7.25, 'IN'),
(89, 'SKU-089', 'Accessory AF', 'Accessories', 5, 12.25, 'IN'),
(90, 'SKU-090', 'Widget AH', 'Widgets', 2, 175.00, 'IN'),
(91, 'SKU-091', 'Widget AI', 'Widgets', 4, 185.00, 'IN'),
(92, 'SKU-092', 'Gadget Y', 'Gadgets', 1, 220.00, 'IN'),
(93, 'SKU-093', 'Gadget Z', 'Gadgets', 2, 200.00, 'IN'),
(94, 'SKU-094', 'Accessory AG', 'Accessories', 7, 14.00, 'IN'),
(95, 'SKU-095', 'Accessory AH', 'Accessories', 8, 15.00, 'IN'),

(96, 'SKU-096', 'Widget AJ', 'Widgets', 1, 165.00, 'US'),
(97, 'SKU-097', 'Gadget AA', 'Gadgets', 3, 145.00, 'US'),
(98, 'SKU-098', 'Accessory AI', 'Accessories', 2, 11.00, 'US'),
(99, 'SKU-099', 'Accessory AJ', 'Accessories', 4, 13.00, 'US'),
(100, 'SKU-100', 'Widget AK', 'Widgets', 2, 170.00, 'US');
```

Python Code:

```
spark.sql("""
INSERT INTO orders_managed_partitioned VALUES
(51, 'SKU-051', 'Widget U', 'Widgets', 2, 145.00, 'IN'),
(52, 'SKU-052', 'Widget V', 'Widgets', 1, 180.00, 'IN'),
(53, 'SKU-053', 'Widget W', 'Widgets', 3, 135.00, 'IN'),
(54, 'SKU-054', 'Gadget O', 'Gadgets', 4, 210.00, 'IN'),
(55, 'SKU-055', 'Gadget P', 'Gadgets', 2, 175.00, 'IN'),
(56, 'SKU-056', 'Accessory Q', 'Accessories', 6, 10.50, 'IN'),
(57, 'SKU-057', 'Accessory R', 'Accessories', 3, 8.25, 'IN'),
(58, 'SKU-058', 'Accessory S', 'Accessories', 5, 11.00, 'IN'),
(59, 'SKU-059', 'Widget X', 'Widgets', 4, 160.00, 'IN'),
(60, 'SKU-060', 'Widget Y', 'Widgets', 2, 190.00, 'IN'),

(61, 'SKU-061', 'Gadget Q', 'Gadgets', 1, 125.00, 'US'),
(62, 'SKU-062', 'Gadget R', 'Gadgets', 2, 135.00, 'US'),
(63, 'SKU-063', 'Gadget S', 'Gadgets', 3, 145.00, 'US'),
(64, 'SKU-064', 'Widget Z', 'Widgets', 5, 165.00, 'US'),
```

(65, 'SKU-065', 'Widget AA', 'Widgets', 4, 175.00, 'US'),
(66, 'SKU-066', 'Accessory T', 'Accessories', 7, 15.00, 'US'),
(67, 'SKU-067', 'Accessory U', 'Accessories', 6, 10.00, 'US'),
(68, 'SKU-068', 'Accessory V', 'Accessories', 8, 9.50, 'US'),
(69, 'SKU-069', 'Accessory W', 'Accessories', 9, 13.00, 'US'),
(70, 'SKU-070', 'Accessory X', 'Accessories', 11, 7.25, 'US'),

(71, 'SKU-071', 'Widget AB', 'Widgets', 2, 140.00, 'US'),
(72, 'SKU-072', 'Gadget T', 'Gadgets', 3, 200.00, 'US'),
(73, 'SKU-073', 'Accessory Y', 'Accessories', 4, 12.00, 'US'),
(74, 'SKU-074', 'Accessory Z', 'Accessories', 5, 13.50, 'US'),
(75, 'SKU-075', 'Widget AC', 'Widgets', 1, 155.00, 'US'),

(76, 'SKU-076', 'Widget AD', 'Widgets', 1, 175.00, 'UK'),
(77, 'SKU-077', 'Gadget U', 'Gadgets', 2, 165.00, 'UK'),
(78, 'SKU-078', 'Accessory AA', 'Accessories', 3, 8.50, 'UK'),

(79, 'SKU-079', 'Widget AE', 'Widgets', 1, 180.00, 'AU'),
(80, 'SKU-080', 'Gadget V', 'Gadgets', 2, 140.00, 'AU'),
(81, 'SKU-081', 'Accessory AB', 'Accessories', 4, 6.75, 'AU'),

(82, 'SKU-082', 'Widget AF', 'Widgets', 1, 195.00, 'CA'),
(83, 'SKU-083', 'Gadget W', 'Gadgets', 2, 150.00, 'CA'),
(84, 'SKU-084', 'Accessory AC', 'Accessories', 3, 10.00, 'CA'),

(85, 'SKU-085', 'Widget AG', 'Widgets', 1, 205.00, 'DE'),
(86, 'SKU-086', 'Gadget X', 'Gadgets', 2, 160.00, 'DE'),

(87, 'SKU-087', 'Accessory AD', 'Accessories', 6, 9.50, 'IN'),
(88, 'SKU-088', 'Accessory AE', 'Accessories', 3, 7.25, 'IN'),
(89, 'SKU-089', 'Accessory AF', 'Accessories', 5, 12.25, 'IN'),
(90, 'SKU-090', 'Widget AH', 'Widgets', 2, 175.00, 'IN'),
(91, 'SKU-091', 'Widget AI', 'Widgets', 4, 185.00, 'IN'),
(92, 'SKU-092', 'Gadget Y', 'Gadgets', 1, 220.00, 'IN'),
(93, 'SKU-093', 'Gadget Z', 'Gadgets', 2, 200.00, 'IN'),
(94, 'SKU-094', 'Accessory AG', 'Accessories', 7, 14.00, 'IN'),
(95, 'SKU-095', 'Accessory AH', 'Accessories', 8, 15.00, 'IN'),

(96, 'SKU-096', 'Widget AJ', 'Widgets', 1, 165.00, 'US'),
(97, 'SKU-097', 'Gadget AA', 'Gadgets', 3, 145.00, 'US'),
(98, 'SKU-098', 'Accessory AI', 'Accessories', 2, 11.00, 'US'),
(99, 'SKU-099', 'Accessory AJ', 'Accessories', 4, 13.00, 'US'),
(100, 'SKU-100', 'Widget AK', 'Widgets', 2, 170.00, 'US')
""")

```
%sql
INSERT INTO orders_managed_partitioned VALUES
(101, 'SKU-101', 'Widget AL', 'Widgets', 2, 155.00, 'IN'),
```

```
(102, 'SKU-102', 'Widget AM', 'Widgets', 1, 165.00, 'IN'),
(103, 'SKU-103', 'Widget AN', 'Widgets', 3, 175.00, 'IN'),
(104, 'SKU-104', 'Gadget AB', 'Gadgets', 4, 180.00, 'IN'),
(105, 'SKU-105', 'Gadget AC', 'Gadgets', 2, 195.00, 'IN'),
(106, 'SKU-106', 'Accessory AK', 'Accessories', 5, 10.00, 'IN'),
(107, 'SKU-107', 'Accessory AL', 'Accessories', 3, 11.00, 'IN'),
(108, 'SKU-108', 'Accessory AM', 'Accessories', 4, 12.50, 'IN'),
(109, 'SKU-109', 'Widget AO', 'Widgets', 1, 185.00, 'IN'),
(110, 'SKU-110', 'Widget AP', 'Widgets', 2, 195.00, 'IN'),

(111, 'SKU-111', 'Gadget AD', 'Gadgets', 1, 125.00, 'US'),
(112, 'SKU-112', 'Gadget AE', 'Gadgets', 2, 140.00, 'US'),
(113, 'SKU-113', 'Gadget AF', 'Gadgets', 3, 155.00, 'US'),
(114, 'SKU-114', 'Widget AQ', 'Widgets', 5, 170.00, 'US'),
(115, 'SKU-115', 'Widget AR', 'Widgets', 4, 165.00, 'US'),
(116, 'SKU-116', 'Accessory AN', 'Accessories', 7, 16.00, 'US'),
(117, 'SKU-117', 'Accessory AO', 'Accessories', 6, 9.50, 'US'),
(118, 'SKU-118', 'Accessory AP', 'Accessories', 8, 10.25, 'US'),
(119, 'SKU-119', 'Accessory AQ', 'Accessories', 9, 14.25, 'US'),
(120, 'SKU-120', 'Accessory AR', 'Accessories', 10, 8.00, 'US'),

(121, 'SKU-121', 'Widget AS', 'Widgets', 2, 150.00, 'US'),
(122, 'SKU-122', 'Gadget AG', 'Gadgets', 3, 200.00, 'US'),
(123, 'SKU-123', 'Accessory AS', 'Accessories', 4, 12.75, 'US'),
(124, 'SKU-124', 'Accessory AT', 'Accessories', 5, 13.25, 'US'),
(125, 'SKU-125', 'Widget AT', 'Widgets', 1, 155.00, 'US'),

(126, 'SKU-126', 'Widget AU', 'Widgets', 1, 175.00, 'UK'),
(127, 'SKU-127', 'Gadget AH', 'Gadgets', 2, 165.00, 'UK'),
(128, 'SKU-128', 'Accessory AU', 'Accessories', 3, 9.25, 'UK'),

(129, 'SKU-129', 'Widget AV', 'Widgets', 1, 185.00, 'AU'),
(130, 'SKU-130', 'Gadget AI', 'Gadgets', 2, 145.00, 'AU'),
(131, 'SKU-131', 'Accessory AV', 'Accessories', 4, 6.25, 'AU'),

(132, 'SKU-132', 'Widget AW', 'Widgets', 1, 195.00, 'CA'),
(133, 'SKU-133', 'Gadget AJ', 'Gadgets', 2, 150.00, 'CA'),
(134, 'SKU-134', 'Accessory AW', 'Accessories', 3, 10.50, 'CA'),
```

```
(135, 'SKU-135', 'Widget AX', 'Widgets', 1, 205.00, 'DE'),
(136, 'SKU-136', 'Gadget AK', 'Gadgets', 2, 160.00, 'DE'),

(137, 'SKU-137', 'Accessory AX', 'Accessories', 6, 9.25, 'IN'),
(138, 'SKU-138', 'Accessory AY', 'Accessories', 3, 7.75, 'IN'),
(139, 'SKU-139', 'Accessory AZ', 'Accessories', 5, 12.75, 'IN'),
(140, 'SKU-140', 'Widget AY', 'Widgets', 2, 175.00, 'IN'),
(141, 'SKU-141', 'Widget AZ', 'Widgets', 4, 185.00, 'IN'),
(142, 'SKU-142', 'Gadget AL', 'Gadgets', 1, 220.00, 'IN'),
(143, 'SKU-143', 'Gadget AM', 'Gadgets', 2, 200.00, 'IN'),
(144, 'SKU-144', 'Accessory BA', 'Accessories', 7, 14.50, 'IN'),
(145, 'SKU-145', 'Accessory BB', 'Accessories', 8, 15.50, 'IN'),

(146, 'SKU-146', 'Widget BA', 'Widgets', 1, 165.00, 'US'),
(147, 'SKU-147', 'Gadget AN', 'Gadgets', 3, 145.00, 'US'),
(148, 'SKU-148', 'Accessory BC', 'Accessories', 2, 11.50, 'US'),
(149, 'SKU-149', 'Accessory BD', 'Accessories', 4, 13.75, 'US'),
(150, 'SKU-150', 'Widget BB', 'Widgets', 2, 170.00, 'US');
```

Python Code:

```
# Run SQL inside Python
spark.sql("""
INSERT INTO orders_managed_partitioned VALUES
(101, 'SKU-101', 'Widget AL', 'Widgets', 2, 155.00, 'IN'),
(102, 'SKU-102', 'Widget AM', 'Widgets', 1, 165.00, 'IN'),
(103, 'SKU-103', 'Widget AN', 'Widgets', 3, 175.00, 'IN'),
(104, 'SKU-104', 'Gadget AB', 'Gadgets', 4, 180.00, 'IN'),
(105, 'SKU-105', 'Gadget AC', 'Gadgets', 2, 195.00, 'IN'),
(106, 'SKU-106', 'Accessory AK', 'Accessories', 5, 10.00, 'IN'),
(107, 'SKU-107', 'Accessory AL', 'Accessories', 3, 11.00, 'IN'),
(108, 'SKU-108', 'Accessory AM', 'Accessories', 4, 12.50, 'IN'),
(109, 'SKU-109', 'Widget AO', 'Widgets', 1, 185.00, 'IN'),
(110, 'SKU-110', 'Widget AP', 'Widgets', 2, 195.00, 'IN'),

(111, 'SKU-111', 'Gadget AD', 'Gadgets', 1, 125.00, 'US'),
(112, 'SKU-112', 'Gadget AE', 'Gadgets', 2, 140.00, 'US'),
(113, 'SKU-113', 'Gadget AF', 'Gadgets', 3, 155.00, 'US'),
(114, 'SKU-114', 'Widget AQ', 'Widgets', 5, 170.00, 'US'),
(115, 'SKU-115', 'Widget AR', 'Widgets', 4, 165.00, 'US'),
(116, 'SKU-116', 'Accessory AN', 'Accessories', 7, 16.00, 'US'),
(117, 'SKU-117', 'Accessory AO', 'Accessories', 6, 9.50, 'US'),
(118, 'SKU-118', 'Accessory AP', 'Accessories', 8, 10.25, 'US'),
(119, 'SKU-119', 'Accessory AQ', 'Accessories', 9, 14.25, 'US'),
```

```
(120, 'SKU-120', 'Accessory AR', 'Accessories', 10, 8.00, 'US'),

(121, 'SKU-121', 'Widget AS', 'Widgets', 2, 150.00, 'US'),
(122, 'SKU-122', 'Gadget AG', 'Gadgets', 3, 200.00, 'US'),
(123, 'SKU-123', 'Accessory AS', 'Accessories', 4, 12.75, 'US'),
(124, 'SKU-124', 'Accessory AT', 'Accessories', 5, 13.25, 'US'),
(125, 'SKU-125', 'Widget AT', 'Widgets', 1, 155.00, 'US'),

(126, 'SKU-126', 'Widget AU', 'Widgets', 1, 175.00, 'UK'),
(127, 'SKU-127', 'Gadget AH', 'Gadgets', 2, 165.00, 'UK'),
(128, 'SKU-128', 'Accessory AU', 'Accessories', 3, 9.25, 'UK'),

(129, 'SKU-129', 'Widget AV', 'Widgets', 1, 185.00, 'AU'),
(130, 'SKU-130', 'Gadget AI', 'Gadgets', 2, 145.00, 'AU'),
(131, 'SKU-131', 'Accessory AV', 'Accessories', 4, 6.25, 'AU'),

(132, 'SKU-132', 'Widget AW', 'Widgets', 1, 195.00, 'CA'),
(133, 'SKU-133', 'Gadget AJ', 'Gadgets', 2, 150.00, 'CA'),
(134, 'SKU-134', 'Accessory AW', 'Accessories', 3, 10.50, 'CA'),

(135, 'SKU-135', 'Widget AX', 'Widgets', 1, 205.00, 'DE'),
(136, 'SKU-136', 'Gadget AK', 'Gadgets', 2, 160.00, 'DE'),

(137, 'SKU-137', 'Accessory AX', 'Accessories', 6, 9.25, 'IN'),
(138, 'SKU-138', 'Accessory AY', 'Accessories', 3, 7.75, 'IN'),
(139, 'SKU-139', 'Accessory AZ', 'Accessories', 5, 12.75, 'IN'),
(140, 'SKU-140', 'Widget AY', 'Widgets', 2, 175.00, 'IN'),
(141, 'SKU-141', 'Widget AZ', 'Widgets', 4, 185.00, 'IN'),
(142, 'SKU-142', 'Gadget AL', 'Gadgets', 1, 220.00, 'IN'),
(143, 'SKU-143', 'Gadget AM', 'Gadgets', 2, 200.00, 'IN'),
(144, 'SKU-144', 'Accessory BA', 'Accessories', 7, 14.50, 'IN'),
(145, 'SKU-145', 'Accessory BB', 'Accessories', 8, 15.50, 'IN'),

(146, 'SKU-146', 'Widget BA', 'Widgets', 1, 165.00, 'US'),
(147, 'SKU-147', 'Gadget AN', 'Gadgets', 3, 145.00, 'US'),
(148, 'SKU-148', 'Accessory BC', 'Accessories', 2, 11.50, 'US'),
(149, 'SKU-149', 'Accessory BD', 'Accessories', 4, 13.75, 'US'),
(150, 'SKU-150', 'Widget BB', 'Widgets', 2, 170.00, 'US')
""")
```

```
%sql
describe detail orders_managed_partitioned;
```

Python Code:

```
spark.sql("DESCRIBE DETAIL orders_managed_partitioned").show(truncate=False)
```

```
%sql
describe detail orders_managed_partitioned;
```

Python Code:
spark.sql("DESCRIBE DETAIL orders_managed_partitioned").show(truncate=False)

```
%sql
select product_category, sum(qty) as total_qty
from orders_managed_partitioned
where country = 'IN'
group by product_category;
```

Python Code:
spark.sql("""
SELECT product_category, SUM(qty) AS total_qty
FROM orders_managed_partitioned
WHERE country = 'IN'
GROUP BY product_category
""").show()

```
%sql
select count(*) from orders_managed_partitioned;
```

Python Code:
spark.sql("SELECT COUNT(*) FROM orders_managed_partitioned").show()

```
%sql
select sum(qty) as total_qty_over_100
from orders_managed_partitioned
where unit_price > 100.0;
```

Python Code:
spark.sql("""
SELECT SUM(qty) AS total_qty_over_100
FROM orders_managed_partitioned
WHERE unit_price > 100.0
""").show()

```
%sql
optimize orders_managed_partitioned;
```

Python Code:
spark.sql("OPTIMIZE orders_managed_partitioned")

```
%sql
select sum(qty) as total_qty_over_100
from orders_managed_partitioned
where unit_price > 100.0;
```

Python Code:
query = """
SELECT SUM(qty) AS total_qty_over_100

```
FROM orders_managed_partitioned
WHERE unit_price > 100.0
"""
result = spark.sql(query)
result.show()
```

Z-Ordering:

```
%sql
use catalog deltalake_catalog;
```

Python Code:

```
spark.sql("USE CATALOG deltalake_catalog")

%fs
ls /databricks-datasets/nyctaxi/tables/nyctaxi_yellow
```

```
%sql
describe formatted
delta.`dbfs:/databricks-datasets/nyctaxi/tables/nyctaxi_yellow`
```

Python Code:

```
path = "dbfs:/databricks-datasets/nyctaxi/tables/nyctaxi_yellow"

df = spark.sql(f"DESCRIBE FORMATTED delta.`{path}`")
df.show(truncate=False)
```

```
%sql
DROP TABLE IF EXISTS demo_taxi_200files;
CREATE TABLE demo_taxi_200files (
  vendor_id STRING,
  pickup_datetime TIMESTAMP,
  dropoff_datetime TIMESTAMP,
  passenger_count INT,
  trip_distance DOUBLE,
  pickup_longitude DOUBLE,
  pickup_latitude DOUBLE,
  rate_code_id INT,
  store_and_fwd_flag STRING,
  dropoff_longitude DOUBLE,
  dropoff_latitude DOUBLE,
  payment_type STRING,
  fare_amount DOUBLE,
  extra DOUBLE,
  mta_tax DOUBLE,
```

```
  tip_amount DOUBLE,
  tolls_amount DOUBLE,
  total_amount DOUBLE
)
USING DELTA
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
);
```

Python Code:

```
# Drop table if it exists
spark.sql("DROP TABLE IF EXISTS demo_taxi_200files")

# Create table
spark.sql("""
CREATE TABLE demo_taxi_200files (
  vendor_id STRING,
  pickup_datetime TIMESTAMP,
  dropoff_datetime TIMESTAMP,
  passenger_count INT,
  trip_distance DOUBLE,
  pickup_longitude DOUBLE,
  pickup_latitude DOUBLE,
  rate_code_id INT,
  store_and_fwd_flag STRING,
  dropoff_longitude DOUBLE,
  dropoff_latitude DOUBLE,
  payment_type STRING,
  fare_amount DOUBLE,
  extra DOUBLE,
  mta_tax DOUBLE,
  tip_amount DOUBLE,
  tolls_amount DOUBLE,
  total_amount DOUBLE
)
USING DELTA
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
)
""")
```

```
table_location = "dbfs:/databricks-datasets/nyctaxi/tables/nyctaxi_yellow"


# List data files (ignore _delta_log)
```

```
all_files = [f.path for f in dbutils.fs.ls(table_location) if
f.path.endswith(".parquet")]

# Pick first 10 files
files10 = all_files[:10]
print("Using these 10 files:", files10)

df_10 = spark.read.parquet(*files10)

df_10.repartition(200).write.format('delta').mode("overwrite").saveAsTable("dem
o_taxi_200files")
```

Above code is already in python

```
%sql
select * from demo_taxi_200files;
```

Python Code:

```
df = spark.sql("SELECT * FROM demo_taxi_200files")
df.show()
```

```
%sql
select count(*) from demo_taxi_200files;
```

Python Code:

```
df = spark.sql("SELECT COUNT(*) AS total_count FROM demo_taxi_200files")
df.show()
```

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100;
```

Python Code:

```
df = spark.sql("""
   SELECT COUNT(*) AS long_trips
   FROM demo_taxi_200files
   WHERE trip_distance > 100
""")
df.show()
```

```
%sql
select min(trip_distance), max(trip_distance), _metadata.file_name
from demo_taxi_200files
group by _metadata.file_name
order by min(trip_distance);
```

Python Code:

```
query = """
```

```
SELECT
  MIN(trip_distance) AS min_distance,
  MAX(trip_distance) AS max_distance,
  _metadata.file_name
FROM demo_taxi_200files
GROUP BY _metadata.file_name
ORDER BY min_distance
"""
df = spark.sql(query)
df.show(truncate=False)
```

```
%sql
optimize demo_taxi_200files zorder by (trip_distance);
```

Python Code:

```
spark.sql("""
OPTIMIZE demo_taxi_200files
ZORDER BY (trip_distance)
""")
```

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100;
```

Python Code:

```
df = spark.sql("""
SELECT COUNT(*) AS long_trip_count
FROM demo_taxi_200files
WHERE trip_distance > 100
""")
df.show()
```

```
%sql
select min(trip_distance), max(trip_distance), _metadata.file_name
from demo_taxi_200files
group by _metadata.file_name
order by min(trip_distance);
```

Python Code:

```
df = spark.sql("""
SELECT
  MIN(trip_distance) AS min_distance,
  MAX(trip_distance) AS max_distance,
  _metadata.file_name
FROM demo_taxi_200files
GROUP BY _metadata.file_name
ORDER BY min_distance
""")

df.show(truncate=False)
```

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 5 and
trip_distance < 7;
```

Python Code:

```
df = spark.sql("""
SELECT COUNT(*) AS trip_count
FROM demo_taxi_200files
WHERE trip_distance > 5 AND trip_distance < 7
""")
df.show()
```

Liquid Clustering:

```
%sql
use catalog deltalake_catalog;
```

Python Code:

```
spark.sql("USE CATALOG deltalake_catalog")
```

```
%sql
DROP TABLE IF EXISTS demo_taxi_200files;
CREATE TABLE demo_taxi_200files (
  vendor_id STRING,
  pickup_datetime TIMESTAMP,
  dropoff_datetime TIMESTAMP,
  passenger_count INT,
  trip_distance DOUBLE,
  pickup_longitude DOUBLE,
  pickup_latitude DOUBLE,
  rate_code_id INT,
  store_and_fwd_flag STRING,
  dropoff_longitude DOUBLE,
  dropoff_latitude DOUBLE,
  payment_type STRING,
  fare_amount DOUBLE,
  extra DOUBLE,
  mta_tax DOUBLE,
  tip_amount DOUBLE,
  tolls_amount DOUBLE,
  total_amount DOUBLE
)
USING DELTA
```

```
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
);
```

Python Code:

```
from pyspark.sql.types import StructType, StructField, StringType, TimestampType, IntegerType,
DoubleType

spark.sql("DROP TABLE IF EXISTS demo_taxi_200files")

schema = StructType([
   StructField("vendor_id", StringType(), True),
   StructField("pickup_datetime", TimestampType(), True),
   StructField("dropoff_datetime", TimestampType(), True),
   StructField("passenger_count", IntegerType(), True),
   StructField("trip_distance", DoubleType(), True),
   StructField("pickup_longitude", DoubleType(), True),
   StructField("pickup_latitude", DoubleType(), True),
   StructField("rate_code_id", IntegerType(), True),
   StructField("store_and_fwd_flag", StringType(), True),
   StructField("dropoff_longitude", DoubleType(), True),
   StructField("dropoff_latitude", DoubleType(), True),
   StructField("payment_type", StringType(), True),
   StructField("fare_amount", DoubleType(), True),
   StructField("extra", DoubleType(), True),
   StructField("mta_tax", DoubleType(), True),
   StructField("tip_amount", DoubleType(), True),
   StructField("tolls_amount", DoubleType(), True),
   StructField("total_amount", DoubleType(), True)
])

empty_df = spark.createDataFrame([], schema)

empty_df.write.format("delta") \
   .option("path", "/user/hive/warehouse/demo_taxi_200files") \
   .option("delta.autoOptimize.optimizeWrite", "false") \
   .option("delta.autoOptimize.autoCompact", "false") \
   .saveAsTable("demo_taxi_200files")
```

```
# Find the table location from DESCRIBE DETAIL first:
table_location = "dbfs:/databricks-datasets/nyctaxi/tables/nyctaxi_yellow"

# List data files (ignore _delta_log)
all_files = [f.path for f in dbutils.fs.ls(table_location) if
f.path.endswith(".parquet")]
```

```
# Pick first 10 files
files10 = all_files[:10]
print("Using these 10 files:", files10)

# Read them as Parquet
df_10 = spark.read.parquet(*files10)

# Repartition into 200 files and write into your demo Delta table
df_10.repartition(200).write.format("delta").mode("overwrite").saveAsTable("dem
o_taxi_200files")
```

Above code is already in python

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100;
```

Python Code:

```
query = """
SELECT COUNT(*) AS cnt
FROM demo_taxi_200files
WHERE trip_distance > 100
"""
result = spark.sql(query).collect()[0]["cnt"]
print(f"Number of trips with distance > 100: {result}")
```

```
%sql
ALTER TABLE demo_taxi_200files CLUSTER BY (trip_distance);
```

Python Code:

```
spark.sql("""
ALTER TABLE demo_taxi_200files CLUSTER BY (trip_distance)
""")
```

```
%sql
describe history demo_taxi_200files;
```

Python Code:

```
history_df = spark.sql("DESCRIBE HISTORY demo_taxi_200files")
history_df.show(truncate=False)
```

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100;
```

Python Code:

```
df = spark.sql("""
SELECT COUNT(*) AS trip_count
FROM demo_taxi_200files
WHERE trip_distance > 100
""")
```

df.show()

```
%sql
select min(trip_distance), max(trip_distance), _metadata.file_name
from demo_taxi_200files
group by _metadata.file_name
order by min(trip_distance)
```

Python Code:

```
df = spark.sql("""
SELECT
  MIN(trip_distance) AS min_trip_distance,
  MAX(trip_distance) AS max_trip_distance,
  _metadata.file_name
FROM demo_taxi_200files
GROUP BY _metadata.file_name
ORDER BY min_trip_distance
""")

df.show(truncate=False)
```

```
%sql
optimize demo_taxi_200files;
```

Python Code:

```
spark.sql("OPTIMIZE demo_taxi_200files")
```

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100;
```

Python Code:

```
df = spark.sql("""
SELECT COUNT(*) AS trip_count
FROM demo_taxi_200files
WHERE trip_distance > 100
""")
df.show()
```

```
%sql
select min(trip_distance), max(trip_distance), _metadata.file_name
from demo_taxi_200files
group by _metadata.file_name
order by min(trip_distance)
```

Python Code:

```
df = spark.sql("""
SELECT
  MIN(trip_distance) AS min_trip_distance,
  MAX(trip_distance) AS max_trip_distance,
```

```
  _metadata.file_name
FROM demo_taxi_200files
GROUP BY _metadata.file_name
ORDER BY min_trip_distance
""")

df.show(truncate=False)
```

```
%sql
describe detail demo_taxi_200files;
```

Python Code:

```
df = spark.sql("DESCRIBE DETAIL demo_taxi_200files")
df.show(truncate=False)
```

```
%sql
ALTER TABLE demo_taxi_200files CLUSTER BY AUTO;
```

Python Code:

```
spark.sql("ALTER TABLE demo_taxi_200files CLUSTER BY AUTO")
```

```
%sql
ALTER TABLE demo_taxi_200files CLUSTER BY NONE;
```

Python Code:

```
spark.sql("ALTER TABLE demo_taxi_200files CLUSTER BY NONE")
```

Liquid Clustering Demo 2

First three cell's code is as same as liquid clustering

```
%sql
select count(*) from demo_taxi_200files where trip_distance > 100
```

Python Code:

```
df = spark.sql("""
   SELECT COUNT(*) AS count_over_100
   FROM demo_taxi_200files
   WHERE trip_distance > 100
""")
df.show()
```

```
%sql
optimize demo_taxi_200files zorder by (trip_distance)
```

Python Code:

```
spark.sql("""
   OPTIMIZE demo_taxi_200files
   ZORDER BY (trip_distance)
""")
```

```
%sql
```

```
select min(trip_distance), max(trip_distance), _metadata.file_name
from demo_taxi_200files
group by _metadata.file_name
order by min(trip_distance)
```

Python Code:

```
query = """
SELECT
  MIN(trip_distance) AS min_trip_distance,
  MAX(trip_distance) AS max_trip_distance,
  _metadata.file_name
FROM demo_taxi_200files
GROUP BY _metadata.file_name
ORDER BY min_trip_distance
"""

df = spark.sql(query)
df.show(truncate=False)
```

```
%sql
DROP TABLE IF EXISTS demo_taxi_200files;
CREATE TABLE demo_taxi_200files (
  vendor_id STRING,
  pickup_datetime TIMESTAMP,
  dropoff_datetime TIMESTAMP,
  passenger_count INT,
  trip_distance DOUBLE,
  pickup_longitude DOUBLE,
  pickup_latitude DOUBLE,
  rate_code_id INT,
  store_and_fwd_flag STRING,
  dropoff_longitude DOUBLE,
  dropoff_latitude DOUBLE,
  payment_type STRING,
  fare_amount DOUBLE,
  extra DOUBLE,
  mta_tax DOUBLE,
  tip_amount DOUBLE,
  tolls_amount DOUBLE,
  total_amount DOUBLE
)
USING DELTA
```

```
CLUSTER BY (vendor_id, trip_distance)
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
);
```

Python Code:

```
from pyspark.sql.types import StructType, StructField, StringType, TimestampType, IntegerType, DoubleType

spark.sql("DROP TABLE IF EXISTS demo_taxi_200files")

schema = StructType([
    StructField("vendor_id", StringType(), True),
    StructField("pickup_datetime", TimestampType(), True),
    StructField("dropoff_datetime", TimestampType(), True),
    StructField("passenger_count", IntegerType(), True),
    StructField("trip_distance", DoubleType(), True),
    StructField("pickup_longitude", DoubleType(), True),
    StructField("pickup_latitude", DoubleType(), True),
    StructField("rate_code_id", IntegerType(), True),
    StructField("store_and_fwd_flag", StringType(), True),
    StructField("dropoff_longitude", DoubleType(), True),
    StructField("dropoff_latitude", DoubleType(), True),
    StructField("payment_type", StringType(), True),
    StructField("fare_amount", DoubleType(), True),
    StructField("extra", DoubleType(), True),
    StructField("mta_tax", DoubleType(), True),
    StructField("tip_amount", DoubleType(), True),
    StructField("tolls_amount", DoubleType(), True),
    StructField("total_amount", DoubleType(), True)
])

empty_df = spark.createDataFrame([], schema)

empty_df.write.format("delta") \
    .option("delta.autoOptimize.optimizeWrite", "false") \
    .option("delta.autoOptimize.autoCompact", "false") \
    .option("delta.minWriterVersion", "2") \
    .saveAsTable("demo_taxi_200files")

spark.sql("ALTER TABLE demo_taxi_200files CLUSTER BY (vendor_id, trip_distance)")
```

```
# Find the table location from DESCRIBE DETAIL first:
table_location = "dbfs:/databricks-datasets/nyctaxi/tables/nyctaxi_yellow"


# List data files (ignore _delta_log)
```

```
all_files = [f.path for f in dbutils.fs.ls(table_location) if
f.path.endswith(".parquet")]

# Pick first 10 files
files10 = all_files[:10]
print("Using these 10 files:", files10)

# Read them as Parquet
df_10 = spark.read.parquet(*files10)

# Repartition into 200 files and write into your demo Delta table
df_10.repartition(200).write.format("delta").mode("overwrite").saveAsTable("dem
o_taxi_200files")
```

Above code is already in python

```
%sql
describe detail demo_taxi_200files;
```

Python Code:
df = spark.sql("DESCRIBE DETAIL demo_taxi_200files")
df.show(truncate=False)