DATA INGESTION TASKS

Task 1: Raw Data Ingestion

1. Create Notebook for Ingesting Raw Weather Data:

```
from pyspark.sql.types import StructType, StructField, StringType, DateType, FloatType
weather_schema = StructType([
    StructField("City", StringType(), True),
    StructField("Date", DateType(), True),
    StructField("Temperature", FloatType(), True),
    StructField("Humidity", FloatType(), True)
])
```

2. Read CSV File and Handle Missing File:

```
import os
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("WeatherDataIngestion").getOrCreate()
file_path = "/path/to/weather_data.csv"
if os.path.exists(file_path):
    raw_data = spark.read.csv(file_path, schema=weather_schema, header=True)
else:
    print(f"File {file_path} not found!")
```

3. Save Data to Delta Table:

```
raw_data.write.format("delta").mode("overwrite").save("/path/to/delta/weather_raw")
```

Task 2: Data Cleaning

1. Create Notebook for Cleaning Data:

```
cleaned_data = spark.read.format("delta").load("/path/to/delta/weather_raw")
```

2. Remove Null or Missing Values:

```
cleaned_data = cleaned_data.dropna()
```

3. Save Cleaned Data to a New Delta Table:

```
cleaned_data.write.format("delta").mode("overwrite").save("/path/to/delta/weather_cleaned")
```

Task 3: Data Transformation

1. Create Notebook for Data Transformation:

```
transformed_data = spark.read.format("delta").load("/path/to/delta/weather_cleaned")
```

2. Calculate Average Temperature and Humidity for Each City:

```
from pyspark.sql.functions import avg
avg_data = transformed_data.groupBy("City").agg(
    avg("Temperature").alias("Avg_Temperature"),
    avg("Humidity").alias("Avg_Humidity")
)
```

3. Save Transformed Data to a Delta Table:

```
avg_data.write.format("delta").mode("overwrite").save("/path/to/delta/weather_transformed")
```

Task 4: Pipeline to Execute Notebooks

1. Create a Pipeline:

```
try:
    dbutils.notebook.run("/path/to/ingest_weather_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/clean_weather_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/transform_weather_data", timeout_seconds=3600)
except Exception as e:
    print(f"Pipeline failed with error: {str(e)}")
```

2. Handle Errors and Log Messages:

- Log messages can be generated using Python's `logging` module or Databricks' logging mechanisms.

Bonus Task: Error Handling

1. Advanced Error Handling:

```
import logging
logging.basicConfig(filename='/path/to/logs/pipeline_errors.log', level=logging.ERROR)
try:
except Exception as e:
    logging.error(f"Pipeline error: {str(e)}")
```

Task 1: Raw Data Ingestion

1. CSV Data (Daily Weather Conditions):

City,Date,Temperature,Humidity

New York,2024-01-01,30.5,60

Los Angeles,2024-01-01,25.0,65

Chicago,2024-01-01,-5.0,75

Houston,2024-01-01,20.0,80

Phoenix,2024-01-01,15.0,50

2. Create Notebook for Ingesting Data:

```
from pyspark.sql.types import StructType, StructField, StringType, DateType, FloatType
weather_schema = StructType([
    StructField("City", StringType(), True),
    StructField("Date", DateType(), True),
    StructField("Temperature", FloatType(), True),
    StructField("Humidity", FloatType(), True)
])
```

-Read the CSV file with error handling:

```
import os
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("WeatherDataIngestion").getOrCreate()
file_path = "/path/to/weather_data.csv"
if os.path.exists(file_path):
    raw_data = spark.read.csv(file_path, schema=weather_schema, header=True)
else:
    print(f"File {file_path} not found!")
```

- Save the raw data to Delta:

```
raw_data.write.format("delta").mode("overwrite").save("/path/to/delta/weather_raw")
```

Task 2: Data Cleaning

1. Create Notebook for Cleaning Data:

- Load the raw data from Delta:

```
raw_data = spark.read.format("delta").load("/path/to/delta/weather_raw")
```

- Handle null or incorrect values:

```
cleaned_data = raw_data.dropna()
# Optional: Remove rows with impossible temperature/humidity values
cleaned_data = cleaned_data.filter((cleaned_data.Temperature > -100) &
                                   (cleaned_data.Humidity <= 100) &
                                   (cleaned_data.Humidity >= 0))
```

- Save the cleaned data to Delta:

```
cleaned_data.write.format("delta").mode("overwrite").save("/path/to/delta/weather_cleaned")
```

Task 3: Data Transformation

1. Create Notebook for Data Transformation:

- Load cleaned data from Delta:

```
cleaned_data = spark.read.format("delta").load("/path/to/delta/weather_cleaned")
```

- Calculate average temperature and humidity per city:

```
from pyspark.sql.functions import avg
transformed_data = cleaned_data.groupBy("City").agg(
    avg("Temperature").alias("Avg_Temperature"),
    avg("Humidity").alias("Avg_Humidity")
)
```

- Save transformed data to Delta:

```
transformed_data.write.format("delta").mode("overwrite").save("/pathto/delta/weather_transformed”)
```

Task 4: Build and Run a Pipeline

1. Create a Databricks Pipeline:

- Sequential execution of notebooks:

```
try:
    dbutils.notebook.run("/path/to/ingest_weather_data", timeout_seconds=3600)
```

```
    dbutils.notebook.run("/path/to/clean_weather_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/transform_weather_data", timeout_seconds=3600)
  except Exception as e:
    print(f"Pipeline failed: {str(e)}")
```

2. Logging the status of each step:

- Use Python's `logging` library or Databricks utilities to log success/failure of each step.

Task 1: Customer Data Ingestion

1. CSV Data (Customer Transactions):

```
CustomerID,TransactionDate,TransactionAmount,ProductCategory
C001,2024-01-15,250.75,Electronics
C002,2024-01-16,125.50,Groceries
C003,2024-01-17,90.00,Clothing
C004,2024-01-18,300.00,Electronics
C005,2024-01-19,50.00,Groceries
```

2. Create Notebook for Ingesting Data:

- Define schema:

```
from pyspark.sql.types import StructType, StructField, StringType, DateType, FloatType
 customer_schema = StructType([
    StructField("CustomerID", StringType(), True),
    StructField("TransactionDate", DateType(), True),
    StructField("TransactionAmount", FloatType(), True),
    StructField("ProductCategory", StringType(), True)
 ])
```

- Read the CSV file with error handling:

```
 import os
 from pyspark.sql import SparkSession
 spark = SparkSession.builder.appName("CustomerDataIngestion").getOrCreate()
 file_path = "/path/to/customer_transactions.csv"
```

```
if os.path.exists(file_path):
    raw_customer_data = spark.read.csv(file_path, schema=customer_schema, header=True)
else:
    print(f"File {file_path} not found!")
```

- Save raw data to Delta:

```
raw_customer_data.write.format("delta").mode("overwrite").save("/path/to/delta/customer_raw")
```

Task 2: Data Cleaning

1. Create Notebook for Cleaning Data:

- Load raw data from Delta:

```
raw_customer_data = spark.read.format("delta").load("/path/to/delta/customer_raw")
```

- Remove duplicates and handle null values:

```
cleaned_customer_data = raw_customer_data.dropDuplicates()
cleaned_customer_data = cleaned_customer_data.filter(cleaned_customer_data.TransactionAmount.isNotNull())
```

- Save cleaned data to Delta: `cleaned_customer_data.write.format("delta").mode("overwrite").save("/path/to/delta/customer_cleaned")`

Task 3: Data Aggregation

1. Create Notebook for Data Aggregation:

- Load cleaned data from Delta:

```
cleaned_customer_data = spark.read.format("delta").load("/path/to/delta/customer_cleaned")
```

- Aggregate data by ProductCategory:

```
from pyspark.sql.functions import sum
aggregated_data = cleaned_customer_data.groupBy("ProductCategory").agg(
    sum("TransactionAmount").alias("Total_TransactionAmount")
)
```

- Save aggregated data to Delta:

```
aggregated_data.write.format("delta").mode("overwrite").save("/path/to/delta/customer_aggregated")
```

Task 4: Pipeline Creation

1. Create a Databricks Pipeline:

- Sequential execution of notebooks:

```
try:
    dbutils.notebook.run("/path/to/ingest_customer_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/clean_customer_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/aggregate_customer_data", timeout_seconds=3600)
except Exception as e:
    print(f"Pipeline failed: {str(e)}")
```

Task 5: Data Validation

1. Create Data Validation Notebook:

```
original_data = spark.read.format("delta").load("/path/to/delta/customer_cleaned")
aggregated_data = spark.read.format("delta").load("/path/to/delta/customer_aggregated")
```

- Calculate total transaction amounts:

```
total_original_transactions = original_data.agg(sum("TransactionAmount")).collect()[0][0]
total_aggregated_transactions = aggregated_data.agg(sum("Total_TransactionAmount")).
collect()[0][0]
```

- Compare the two sums:

```
if total_original_transactions == total_aggregated_transactions:
    print("Validation successful: Totals match")
else:
    print(f"Validation failed: {total_original_transactions} != {total_aggregated_transactions}")
```

Task 1: Product Inventory Data Ingestion

1. CSV Data (Product Inventory Information):

```
ProductID,ProductName,StockQuantity,Price,LastRestocked
P001,Laptop,50,1500.00,2024-02-01
P002,Smartphone,200,800.00,2024-02-02
P003,Headphones,300,100.00,2024-01-29
```

P004,Tablet,150,600.00,2024-01-30

P005,Smartwatch,100,250.00,2024-02-03

2. Create Notebook for Ingesting Data:

- Define schema:

```
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, FloatType, DateType
product_schema = StructType([
    StructField("ProductID", StringType(), True),
    StructField("ProductName", StringType(), True),
    StructField("StockQuantity", IntegerType(), True),
    StructField("Price", FloatType(), True),
    StructField("LastRestocked", DateType(), True)
])
```

- Read the CSV file with error handling:

```
import os
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("ProductDataIngestion").getOrCreate()
file_path = "/path/to/product_inventory.csv"
if os.path.exists(file_path):
    raw_product_data = spark.read.csv(file_path, schema=product_schema, header=True)
else:
    print(f"File {file_path} not found!")
```

- Save raw data to Delta:

```
raw_product_data.write.format("delta").mode("overwrite").save("/path/to/delta/product_inventory_raw")
```

Task 2: Data Cleaning

1. Create Notebook for Cleaning Data:

- Load raw data from Delta:

```
raw_product_data = spark.read.format("delta").load("/path/to/delta/product_inventory_raw")
```

- Clean the data:

  - Remove rows with null values in `StockQuantity` and `Price`:

  cleaned_product_data = raw_product_data.filter(

    (raw_product_data.StockQuantity.isNotNull()) &

    (raw_product_data.Price.isNotNull())

)

  - Ensure `StockQuantity` is greater than or equal to 0:

  cleaned_product_data = cleaned_product_data.filter(cleaned_product_data.StockQuantity >= 0)

- Save cleaned data to Delta:
cleaned_product_data.write.format("delta").mode("overwrite").save("/path/to/delta/product_inventory_cleaned")

Task 3: Inventory Analysis

1. Create Notebook for Inventory Analysis:

- Load cleaned data from Delta:

  cleaned_product_data = spark.read.format("delta").load("/path/to/delta/product_inventory_cleaned")

- Calculate total stock value for each product:

  from pyspark.sql.functions import col

  inventory_analysis = cleaned_product_data.withColumn(

    "TotalStockValue", col("StockQuantity") * col("Price")

)

- Identify products that need restocking (`StockQuantity < 100`):

  products_needing_restock = cleaned_product_data.filter(cleaned_product_data.StockQuantity < 100)

- Save analysis results to Delta:
inventory_analysis.write.format("delta").mode("overwrite").save("/path/to/delta/product_inventory_analysis")

products_needing_restock.write.format("delta").mode("overwrite").save("/path/to/delta/products_needing_restock")

Task 4: Build an Inventory Pipeline

1. Create a Databricks Pipeline:

- Sequential execution of notebooks using `dbutils.notebook.run()`:

```
try:
    dbutils.notebook.run("/path/to/ingest_product_inventory", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/clean_product_inventory", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/analyze_product_inventory", timeout_seconds=3600)
except Exception as e:
    print(f"Pipeline failed: {str(e)}")
```

2. Log progress and errors at each step to track success or failure.

Task 5: Inventory Monitoring

1. Create a Monitoring Notebook:

- Load inventory data:

```
inventory_data = spark.read.format("delta").load("/path/to/delta/product_inventory_cleaned")
```

- Check for products that need restocking (`StockQuantity < 50`):

```
low_stock_products = inventory_data.filter(inventory_data.StockQuantity < 50)
if low_stock_products.count() > 0:
    print("Alert: Some products need restocking!")
```

2. Send alerts using Databricks notebooks or external services like email/SMS for products below the threshold.

Task 1: Employee Attendance Data Ingestion

1. CSV Data (Employee Attendance Logs):

```
EmployeeID,Date,CheckInTime,CheckOutTime,HoursWorked
E001,2024-03-01,09:00,17:00,8
E002,2024-03-01,09:15,18:00,8.75
E003,2024-03-01,08:45,17:15,8.5
E004,2024-03-01,10:00,16:30,6.5
E005,2024-03-01,09:30,18:15,8.75
```

2. Create Notebook for Ingesting Data:

- Define schema:

```
from pyspark.sql.types import StructType, StructField, StringType, DateType, TimestampType, FloatType

attendance_schema = StructType([
    StructField("EmployeeID", StringType(), True),
    StructField("Date", DateType(), True),
    StructField("CheckInTime", StringType(), True),
    StructField("CheckOutTime", StringType(), True),
    StructField("HoursWorked", FloatType(), True)
])
```

- Read the CSV file with error handling:

```
import os
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("AttendanceDataIngestion").getOrCreate()
file_path = "/path/to/attendance_logs.csv"
if os.path.exists(file_path):
    raw_attendance_data = spark.read.csv(file_path, schema=attendance_schema, header=True)
else:
    print(f"File {file_path} not found!")
```

- Save raw data to Delta:

```
raw_attendance_data.write.format("delta").mode("overwrite").save("/path/to/delta/attendance_raw")
```

Task 2: Data Cleaning

1. Create Notebook for Cleaning Data:

- Load raw data from Delta:

```
raw_attendance_data = spark.read.format("delta").load("/path/to/delta/attendance_raw")
```

- Clean the data:

- Remove rows with null values in `CheckInTime`, `CheckOutTime`, or `HoursWorked`:

```
cleaned_attendance_data = raw_attendance_data.na.drop(subset=["CheckInTime", "CheckOutTime", "HoursWorked"])
```

- Ensure `HoursWorked` is correctly calculated:

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

```
    cleaned_attendance_data = cleaned_attendance_data.withColumn(
        "CalculatedHoursWorked",
        (unix_timestamp(col("CheckOutTime"), "HH:mm") - unix_timestamp(col("CheckInTime"), "HH:mm")) / 3600
    )
    cleaned_attendance_data = cleaned_attendance_data.filter(
        col("CalculatedHoursWorked") == col("HoursWorked")
    )
```

- Save cleaned data to Delta:

```
cleaned_attendance_data.write.format("delta").mode("overwrite").save("/path/to/delta/attendance_cleaned")
```

Task 3: Attendance Summary

1. Create Notebook for Attendance Summary:

- Load cleaned data from Delta:

```
cleaned_attendance_data = spark.read.format("delta").load("/path/to/delta/attendance_cleaned")
```

- Calculate total hours worked by each employee for the current month:

```
from pyspark.sql.functions import month, year
monthly_hours_worked = cleaned_attendance_data.groupBy("EmployeeID").agg(
    sum("HoursWorked").alias("TotalHoursWorked")
).filter(year("Date") == 2024).filter(month("Date") == 3)
```

- Find employees who worked overtime (more than 8 hours in a day):

```
overtime_employees = cleaned_attendance_data.filter(cleaned_attendance_data.HoursWorked > 8)
```

- Save attendance summary to Delta:

```
monthly_hours_worked.write.format("delta").mode("overwrite").save("/path/to/delta/attendance_summary")
overtime_employees.write.format("delta").mode("overwrite").save("/path/to/delta/overtime_employees")
```

Task 4: Create an Attendance Pipeline

1. Create Databricks Pipeline:

- Sequential execution of notebooks using `dbutils.notebook.run()`:

```
try:
    dbutils.notebook.run("/path/to/ingest_attendance_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/clean_attendance_data", timeout_seconds=3600)
    dbutils.notebook.run("/path/to/summarize_attendance_data", timeout_seconds=3600)
except Exception as e:
    print(f"Pipeline failed: {str(e)}")
```

2. Log progress and errors at each step to track success or failure.

Task 5: Time Travel with Delta Lake

1. Implement Time Travel:

```
previous_version = spark.read.format("delta").option("versionAsOf", 1).
load("/path/to/delta/attendance_raw")
```

- Use `DESCRIBE HISTORY` to inspect changes:

```
spark.sql("DESCRIBE HISTORY delta.`/path/to/delta/attendance_raw`").show()
```