Data Ingestion, Exploration & Visualization in Databricks

1. Data Exploration and Visualization in Databricks

Exploratory Data Analysis (EDA) Tools

Databricks provides built-in capabilities for EDA:

```
# Basic statistics
display(df.summary())

# Quick profiling
import pandas as pd
df.profile.show()

# Schema inspection
df.printSchema()
```

Visualization Options

1. **Built-in Display Function**

```
display(df)  # Auto-renders charts for numeric data
```

2. **Custom Visualizations**

```
# Matplotlib
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
df_pd = df.toPandas()
df_pd.plot(kind='bar', x='category', y='sales')
display()
```

3. **Advanced Dashboards**
    - Create → Dashboard
    - Add widgets and parameterized queries

📌 **Pro Tip:** Use displayHTML() for custom D3.js visualizations

2. Hands-on Exercise: Visualizing Data

Step-by-Step Example

```
# Load sample data
df = spark.read.csv("/databricks-datasets/COVID/covid-19-data/usa-
counties.csv", header=True)

# Basic visualization
display(df.groupBy("date").agg(sum("cases").alias("total_cases"))

# Advanced plot
from pyspark.sql.functions import to_date
cases_by_state = (df
  .withColumn("date", to_date("date"))
  .filter("date > '2022-01-01'")
  .groupBy("state", "date")
  .agg(sum("cases").alias("daily_cases"))

display(cases_by_state)
```

3. Mounting Cloud Storage

Mounting ADLS Gen2

python

```
configs = {
  "fs.azure.account.auth.type": "OAuth",
  "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
  "fs.azure.account.oauth2.client.id": "<app-id>",
  "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="key-
vault", key="databricks-sp-secret"),
  "fs.azure.account.oauth2.client.endpoint":
"https://login.microsoftonline.com/<tenant-id>/oauth2/token"
}

dbutils.fs.mount(
  source = "abfss://container@storage.dfs.core.windows.net/",
  mount_point = "/mnt/data_lake",
  extra_configs = configs
)
```

Mounting S3

python

```
aws_access_key = dbutils.secrets.get(scope="aws", key="access-key")
aws_secret_key = dbutils.secrets.get(scope="aws", key="secret-key")

dbutils.fs.mount(
  source = "s3a://bucket-name",
  mount_point = "/mnt/s3_data",
  extra_configs = {
    "fs.s3a.access.key": aws_access_key,
    "fs.s3a.secret.key": aws_secret_key
  }
)
```

4. Reading and Writing Data

Supported Formats

Format	Read Syntax	Write Syntax
CSV	spark.read.csv(path)	df.write.csv(path)
JSON	spark.read.json(path)	df.write.json(path)
Parquet	spark.read.parquet(path)	df.write.parquet(path)
Delta	spark.read.format("delta").load(path)	df.write.format("delta").save(path)

Specialized Operations

```
# Read from Delta with time travel
df = spark.read.format("delta").option("versionAsOf",
12).load("/mnt/delta/table")

# Write with partitioning
df.write.partitionBy("year",
"month").format("delta").save("/mnt/delta/partitioned")

# Create managed table
df.write.saveAsTable("managed_table")
```

5. Notebook Orchestration

Run Notebooks Programmatically

```
# Run with parameters
result = dbutils.notebook.run(
  "/Shared/processing",
  timeout_seconds=300,
  arguments={"input_path": "/mnt/raw/data", "output_path":
"/mnt/processed"}
)

# Chain executions
if result == "SUCCESS":
  dbutils.notebook.run("/Shared/next_step", 60)
```

6. Streaming Data Analysis

Streaming EDA Example

```
from pyspark.sql.functions import *

stream_df = (spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "kafka:9092")
  .option("subscribe", "transactions")
  .load()
  .select(from_json(col("value").cast("string"), schema).alias("data"))
  .select("data.*")
)

# Create streaming visualizations
display(stream_df.groupBy("product_id").agg(count("*").alias("transactions"
)))
```

Streaming to Delta Lake

```
(stream_df.writeStream
  .format("delta")
  .outputMode("append")
  .option("checkpointLocation", "/checkpoints/transactions")
  .start("/delta/transactions"))
```

Best Practices

1. **Storage Optimization**
    - Use Delta Lake for ACID transactions

- Compact small files: OPTIMIZE delta_table ZORDER BY (date)

2. **Visualization Tips**
   - Cache frequently used datasets: df.cache()
   - Use dashboard parameters for interactivity
3. **Performance**
   - Set spark.sql.shuffle.partitions appropriately
   - Use photon acceleration for faster SQL

End-to-End Data Flow