Data Engineering Practical – Banana Prices Pipeline

1. Objective

Ingest the latest DEFRA banana prices CSV from the GOV.UK dataset page into a Medallion architecture (Bronze → Silver → Gold) using Delta Lake tables. Bronze stores raw data plus lineage columns; Silver standardizes and cleans the schema; Gold produces business aggregates for reporting.

Source: GOV.UK – Banana prices statistical dataset (updated fortnightly).Link: https://www.gov.uk/government/statistical-data-sets/banana-prices

2. Medallion Design

Source (GOV.UK / DEFRA) → Bronze (Delta) → Silver (Delta) → Gold (Delta)

Source (GOV.UK / DEFRA):

Download latest bananas-YYYYMMDD.csv

Schedule aligned to refresh cadence

Store lineage metadata (url, run_id, time)

Bronze (Delta):

Raw + lineage columns

Append or overwrite depending on requirement

Silver (Delta):

Clean + standardize

Parse dates & types

Remove duplicates

Gold (Delta):

Avg price by year & origin

BI-ready table

Partition if needed

Lineage & Ops

Each run writes ingestion metadata columns: ingestion_timestamp, source_url, raw_file_path, and run_id. In production, schedule the job on Databricks Jobs (for example, daily) while upstream updates are fortnightly; if no new file is detected, the job can short-circuit without rewriting.

3. Source → Bronze Implementation (Databricks)

Below is interview-friendly, commented notebook code. It scrapes the dataset page, downloads the newest CSV, lands it in ADLS Gen2, and writes a Bronze Delta table.

3.1 Configuration

```
# GOV.UK dataset page that contains links to the latest 'bananas-YYYYMMDD.csv'

DATASET_PAGE_URL = "https://www.gov.uk/government/statistical-data-sets/banana-prices"

# ADLS Gen2 storage identifiers (Databricks + ABFSS)

# NOTE: Don't hardcode account keys in notebooks.

# Use Databricks Secrets (recommended) or managed identity.

STORAGE_ACCOUNT = "scgproject"

CONTAINER_BRONZE = "bronzedata"

# Base path in ADLS Gen2

BASE = f"abfss://{CONTAINER_BRONZE}@{STORAGE_ACCOUNT}.dfs.core.windows.net"

# Where we store the downloaded raw CSV (still part of Bronze, but "raw landing")

RAW_DIR = f"{BASE}/raw"

# Where the Bronze Delta table lives

BRONZE_DELTA_DIR = f"{BASE}/bronze/banana_prices"

print("DATASET_PAGE_URL =", DATASET_PAGE_URL)

print("RAW_DIR          =", RAW_DIR)

print("BRONZE_DELTA_DIR =", BRONZE_DELTA_DIR)
```

3.2 Discover latest CSV

```
# 2) Discover the latest CSV link on GOV.UK

# We scrape the dataset page and pick the newest file by the YYYYMMDD in the filename.

import re

import requests

html = requests.get(DATASET_PAGE_URL, timeout=30).text

# Example link format:
```

```
# https://assets.publishing.service.gov.uk/.../bananas-20260119.csv

links = re.findall(

    r"https://assets\.publishing\.service\.gov\.uk/[^\"]+/bananas-\d{8}\.csv",

html

)

if not links:

    raise Exception("No banana CSV links found on the dataset page")

# Sort by the date in the filename and take the newest

latest_csv_url = sorted(

    set(links),

    key=lambda u: int(re.search(r"bananas-(\d{8})\.csv", u).group(1))

)[-1]

latest_csv_url
```

3.3 Download to raw landing

```
# 3) Download the latest CSV into the 'raw' landing area

# In Databricks, we can write to ADLS with dbutils.fs.put.

# We also create a run_id to help with lineage & auditing.

import datetime

from datetime import timezone

dbutils.fs.mkdirs(RAW_DIR)

dbutils.fs.mkdirs(f"{BASE}/bronze")

run_id = datetime.datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%SZ")

raw_file_path = f"{RAW_DIR}/bananas_{run_id}.csv"

r = requests.get(latest_csv_url, timeout=60)

r.raise_for_status()

# The file is sometimes encoded in Latin-1 / ISO-8859-1.

# Decode using latin-1 so we don't crash on special characters.

dbutils.fs.put(raw_file_path, r.content.decode("latin-1"), overwrite=True)
```

raw_file_path

3.4 Build Bronze Delta

```
# 4) Build Bronze Delta table (raw + lineage)

# Bronze keeps the dataset "as-is" but adds metadata columns for governance.

from pyspark.sql import functions as F

df_bronze = (

    spark.read

        .option("header", "true")

        .option("inferSchema", "true")

        .option("encoding", "ISO-8859-1")     # defensive: handle latin-1 text

        .csv(raw_file_path)

        # lineage / audit columns

        .withColumn("ingestion_timestamp", F.current_timestamp())

        .withColumn("source_url", F.lit(latest_csv_url))

        .withColumn("raw_file_path", F.lit(raw_file_path))

        .withColumn("run_id", F.lit(run_id))

)

display(df_bronze.limit(20))

# Write Bronze as Delta

(df_bronze.write

    .format("delta")

    .mode("overwrite")                  # overwrite for the practical; production may prefer append + partition

    .option("overwriteSchema", "true")

    .save(BRONZE_DELTA_DIR)

)

# Read back to prove it saved correctly

display(spark.read.format("delta").load(BRONZE_DELTA_DIR).limit(20))
```

4. Silver layer (clean)

```
# 5) Silver layer (clean + standardized)

# Typical Silver steps:

#  - standardize column names

#  - parse dates

#  - cast numeric price fields

#  - drop empty rows / duplicates

import re

from pyspark.sql import functions as F

from pyspark.sql.types import DoubleType

SILVER_DELTA_DIR = f"abfss://silver@{STORAGE_ACCOUNT}.dfs.core.windows.net/banana_prices"

dbutils.fs.mkdirs(f"abfss://silver@{STORAGE_ACCOUNT}.dfs.core.windows.net")

df_bronze_in = spark.read.format("delta").load(BRONZE_DELTA_DIR)

# 1) Clean column names (lowercase, underscores)

def clean_name(c):

    c = c.strip().lower()

    c = re.sub(r"[^a-z0-9]+", "_", c)

    return c.strip("_")

df = df_bronze_in

for c in df.columns:

    df = df.withColumnRenamed(c, clean_name(c))

# 2) Try to parse a date column if present

date_candidates = [c for c in df.columns if "date" in c]

if date_candidates:

    date_col = date_candidates[0]

    df = df.withColumn(date_col, F.to_date(F.col(date_col)))

# 3) Cast common price columns to double (adjust names if needed)

price_candidates = [c for c in df.columns if "price" in c or "pence" in c]
```

```
for pc in price_candidates:
df = df.withColumn(pc, F.regexp_replace(F.col(pc).cast("string"), ",", "").cast(DoubleType()))
# 4) Drop duplicates (based on all business columns; keep metadata)
df_silver = df.dropDuplicates()
(df_silver.write
   .format("delta")
   .mode("overwrite")
   .option("overwriteSchema", "true")
   .save(SILVER_DELTA_DIR)
display(spark.read.format("delta").load(SILVER_DELTA_DIR).limit(20))
print("Silver saved to:", SILVER_DELTA_DIR)
```

5. Gold layer (aggregates)

```
# 6) Gold layer (business aggregates)
# Gold is what BI tools usually consume.
# For this practical: average price by year and origin.

GOLD_DELTA_DIR = f"abfss://gold@{STORAGE_ACCOUNT}.dfs.core.windows.net/banana_price_trends"
dbutils.fs.mkdirs(f"abfss://gold@{STORAGE_ACCOUNT}.dfs.core.windows.net")
df_silver_in = spark.read.format("delta").load(SILVER_DELTA_DIR)
# Pick columns (adjust to your file after you inspect df_silver_in.columns)
# We'll try to auto-detect:
origin_col = next((c for c in df_silver_in.columns if "origin" in c), None)
date_col   = next((c for c in df_silver_in.columns if "date" in c), None)
price_col  = next((c for c in df_silver_in.columns if "price" in c), None)
if not (origin_col and date_col and price_col):
   raise Exception(
      f"Could not auto-detect required columns. Found origin={origin_col}, date={date_col}, price={price_col}. "
      "Please inspect df_silver_in.columns and set them manually."
   )
```

```
df_gold = (
    df_silver_in
        .withColumn("year", F.year(F.col(date_col)))
        .groupBy("year", origin_col)
        .agg(F.avg(F.col(price_col)).alias("avg_price"))
        .orderBy("year", origin_col)
)
(df_gold.write
    .format("delta")
    .mode("overwrite")
    .option("overwriteSchema", "true")
    .save(GOLD_DELTA_DIR)
display(spark.read.format("delta").load(GOLD_DELTA_DIR).limit(50))
print("Gold saved to:", GOLD_DELTA_DIR)
```