Lab 9: Data Visualization & Dashboards

Z5008 Big Data Lab · MTech Data Science & AI

Dr. Innocent Nyalala
innocent@iitmz.ac.in

School of Engineering & Science
IIT Madras Zanzibar

11 May 2026

Outline

Why Visualization Matters for Big Data

The Insight Gap

Raw numbers in a DataFrame tell you *facts*.
A well-crafted chart tells you a *story*.

East Africa e-commerce context:

- 100K+ orders/day across Kenya, Tanzania, Uganda, Rwanda
- Mobile money (M-Pesa, Airtel Money) dominates payments
- Seasonal spikes: Ramadan, Christmas, school-fees months
- Rural vs. urban purchasing patterns differ sharply

Lab 9 Toolkit

Layer	Tool
Static/pub	Matplotlib, Seaborn
Interactive	Plotly Express
Custom GO	Graph Objects
Dashboard	Dash
Geospatial	Folium, px.choropleth
BI/Ops	Superset, Grafana

Learning Goals

- Aggregate first, then plot
- Choose the right chart type
- Build interactive dashboards

The Big Data Trap

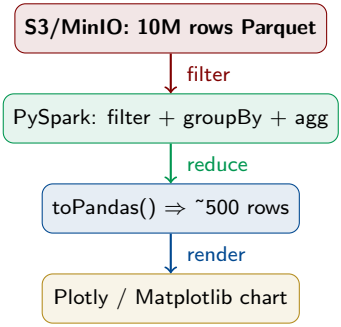
The Aggregation-First Principle

Cardinal Rule

Aggregate in Spark. Plot with Pandas.
Never move millions of rows to the driver.

Safe pipeline:

1. Filter & select relevant columns in PySpark
2. `groupBy().agg()` to collapse to summary
3. `.limit(N)` as an absolute safety net
4. `.toPandas()` — result is now small
5. Plot with Plotly / Matplotlib



Memory budget:
1 row ≈ 200 bytes ⇒
100K rows ≈ 20 MB (safe)

SparkSession Setup for Lab 9

```python
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

spark = SparkSession.builder \
    .appName("Lab9-Visualization") \
    .master("spark://spark-master:7077") \
    .config("spark.hadoop.fs.s3a.endpoint",          "http://minio:9000") \
    .config("spark.hadoop.fs.s3a.access.key",        "minioadmin") \
    .config("spark.hadoop.fs.s3a.secret.key",        "minioadmin") \
    .config("spark.hadoop.fs.s3a.path.style.access","true") \
    .config("spark.hadoop.fs.s3a.impl",
            "org.apache.hadoop.fs.s3a.S3AFileSystem") \
    .config("spark.sql.shuffle.partitions", "8") \
    .getOrCreate()

spark.sparkContext.setLogLevel("WARN")

# Load the 100K-row orders dataset from Lab 4
orders = spark.read.parquet("s3a://warehouse/lab04/orders_parquet/")
orders.printSchema()
print(f"Total rows: {orders.count():,}")
```

Safe toPandas() Patterns

```python
# SAFE PATTERN 1 — aggregate then convert
daily_rev = (
    orders
    .withColumn("date", F.to_date("order_timestamp"))
    .groupBy("date", "country")
    .agg(
        F.sum("order_value_usd").alias("revenue"),
        F.count("*").alias("num_orders")
    )
    .orderBy("date")
)
pdf_daily = daily_rev.toPandas()      # ~50 rows — fine
print(pdf_daily.shape)

# SAFE PATTERN 2 — exploration sample (never production)
pdf_sample = orders.sample(fraction=0.01, seed=42).toPandas()

# DANGEROUS — do NOT do this with large tables
# pdf_bad = orders.toPandas()         # moves ALL rows to driver!

# SAFE PATTERN 3 — categorical aggregation
pdf_cat = (
    orders
```

Production-Quality Matplotlib Charts

```python
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import seaborn as sns

sns.set_theme(style="whitegrid", palette="muted", font_scale=1.1)
COLORS = {"KE":"#C5A059","TZ":"#0a1628","UG":"#00964F","RW":"#E53935"}

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Left: grouped bar — revenue per country
pdf_ctry = pdf_daily.groupby("country")["revenue"].sum().reset_index()
axes[0].bar(pdf_ctry["country"],
            pdf_ctry["revenue"] / 1e6,
            color=[COLORS[c] for c in pdf_ctry["country"]],
            edgecolor="white", linewidth=0.8)
axes[0].yaxis.set_major_formatter(
    mtick.FuncFormatter(lambda x, _: f"${x:.1f}M"))
axes[0].set_title("Total Revenue by Country",
                  fontsize=13, fontweight="bold")
axes[0].set_xlabel("Country"); axes[0].set_ylabel("Revenue (USD)")

# Right: line chart — daily orders
for cty, grp in pdf_daily.groupby("country"):
```

Seaborn: Heatmap & KDE Distribution

```
import seaborn as sns, pandas as pd
import matplotlib.pyplot as plt

# Pivot: country x payment_method order counts
pivot = pdf_cat.pivot_table(
    values="num_orders", index="country",
    columns="payment_method", aggfunc="sum", fill_value=0
)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Heatmap     annotated with actual counts
sns.heatmap(pivot, annot=True, fmt=",d", cmap="YlOrRd",
            linewidths=0.5, ax=ax1,
            cbar_kws={"label": "Order Count"})
ax1.set_title("Orders: Country x Payment Method", fontweight="bold")
ax1.set_xlabel("Payment Method"); ax1.set_ylabel("Country")

# KDE     distribution of order values per country
for cty in ["KE","TZ","UG","RW"]:
    sub = pdf_sample[pdf_sample["country"] == cty]
    sns.kdeplot(sub["order_value_usd"], ax=ax2,
                label=cty, fill=True, alpha=0.25, linewidth=2)
```

Plotly Express: Grouped & Stacked Bar Charts

```python
import plotly.express as px

COLOR_MAP = {"KE":"#C5A059","TZ":"#0a1628","UG":"#00964F","RW":"#E53935"}

# Grouped bar     revenue per category per country
fig = px.bar(
    pdf_cat, x="category", y="revenue", color="country",
    barmode="group",       # swap to "stack" for stacked
    title="Revenue by Category and Country     East Africa E-commerce",
    labels={"revenue":"Revenue (USD)","category":"Product Category"},
    color_discrete_map=COLOR_MAP,
    text_auto=".2s",       # auto-annotate bars
    height=500
)
fig.update_layout(
    legend_title="Country",
    plot_bgcolor="white",
    xaxis=dict(gridcolor="#eeeeee"),
    yaxis=dict(gridcolor="#eeeeee", tickprefix="$"),
    font=dict(family="Inter, sans-serif", size=12)
)
fig.show()
```

Plotly Express: Line & Scatter

```python
# Line chart with markers      daily revenue trend
fig_line = px.line(
    pdf_daily, x="date", y="revenue", color="country",
    markers=True, line_dash="country",
    title="Daily Revenue Trend     East Africa E-commerce 2025",
    labels={"revenue":"Revenue (USD)","date":"Date"},
    color_discrete_map=COLOR_MAP,
    height=430
)
fig_line.update_traces(line_width=2.5)
fig_line.update_layout(plot_bgcolor="white")
fig_line.show()

# Scatter      order value vs delivery days
# colour = country, size = basket size, symbol = payment method
fig_scat = px.scatter(
    pdf_sample, x="delivery_days", y="order_value_usd",
    color="country", size="basket_size",
    hover_name="order_id", symbol="payment_method",
    color_discrete_map=COLOR_MAP,
    trendline="ols",           # linear regression overlay per group
    opacity=0.65,
    title="Order Value vs Delivery Time (sampled)",
```

Plotly Express: Sunburst & Treemap

```python
# Sunburst      country -> category -> payment_method
fig_sun = px.sunburst(
    pdf_cat,
    path=["country","category","payment_method"],
    values="revenue",
    color="revenue",
    color_continuous_scale="YlOrRd",
    title="Revenue Hierarchy: Country -> Category -> Payment",
    height=560
)
fig_sun.update_traces(textinfo="label+percent entry")
fig_sun.show()

# Treemap      same hierarchy, different layout
fig_tree = px.treemap(
    pdf_cat,
    path=["country","category"],
    values="revenue",
    color="num_orders",
    color_continuous_scale="Viridis",
    title="Treemap: Revenue (tile size) vs Orders (colour)",
    height=500
)
```

Plotly Express: Box, Histogram, Choropleth

```python
# Box plot      order value distribution with notch (CI on median)
fig_box = px.box(
    pdf_sample, x="country", y="order_value_usd",
    color="country", points="outliers", notched=True,
    color_discrete_map=COLOR_MAP,
    title="Order Value Distribution by Country",
    labels={"order_value_usd":"Order Value (USD)"}
)
fig_box.show()

# Histogram      delivery days, overlaid by country
fig_hist = px.histogram(
    pdf_sample, x="delivery_days", color="country",
    nbins=30, barmode="overlay", opacity=0.60,
    color_discrete_map=COLOR_MAP,
    title="Delivery Days Distribution by Country"
)
fig_hist.show()

# Choropleth      African country-level revenue
fig_choro = px.choropleth(
    pdf_country, locations="iso3", color="revenue",
    locationmode="ISO-3", scope="africa",
```

Plotly Graph Objects: Manual Control

```python
import plotly.graph_objects as go

fig = go.Figure()

for country in ["KE","TZ","UG","RW"]:
    mask = pdf_cat["country"] == country
    fig.add_trace(go.Bar(
        name=country,
        x=pdf_cat[mask]["category"],
        y=pdf_cat[mask]["revenue"],
        text=pdf_cat[mask]["revenue"].map(lambda v: f"${v/1e3:.0f}K"),
        textposition="inside",
        marker_color=COLOR_MAP[country],
        marker_line_width=0.5,
        marker_line_color="white",
    ))

fig.update_layout(
    barmode="stack",
    title=dict(text="Stacked Revenue by Category",
               font=dict(size=18, family="Inter")),
    legend=dict(orientation="h", yanchor="bottom",
                y=1.02, xanchor="right", x=1),
```

Plotly Subplots: make_subplots

```python
from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Revenue Trend","Orders by Country",
                    "Payment Methods Share","Category Heatmap"),
    vertical_spacing=0.13, horizontal_spacing=0.09
)

# (1,1)     Line chart
fig.add_trace(go.Scatter(
    x=pdf_daily["date"], y=pdf_daily["revenue"],
    mode="lines+markers", name="Revenue", line_color="#C5A059"), row=1, col=1)

# (1,2)     Bar chart
fig.add_trace(go.Bar(
    x=pdf_country["country"], y=pdf_country["num_orders"],
    name="Orders", marker_color="#0a1628"), row=1, col=2)

# (2,1)     Pie
fig.add_trace(go.Pie(
    labels=pdf_pay["method"], values=pdf_pay["count"],
```

Plotly Animations: animation_frame

```python
# Prepare monthly aggregation
pdf_monthly = (
    orders
    .withColumn("month", F.date_format("order_timestamp","yyyy-MM"))
    .groupBy("month","country")
    .agg(F.sum("order_value_usd").alias("revenue"),
         F.count("*").alias("num_orders"),
         F.avg("delivery_days").alias("delivery_days_avg"))
    .orderBy("month")
    .toPandas()
)

# Animated bar chart      revenue by country over months
fig_anim = px.bar(
    pdf_monthly, x="country", y="revenue", color="country",
    animation_frame="month",       # <- this drives the animation
    animation_group="country",
    range_y=[0, pdf_monthly["revenue"].max() * 1.15],
    color_discrete_map=COLOR_MAP,
    title="Monthly Revenue Animation --- East Africa 2025",
    height=500
)
# Slow down playback
```

Plotly Animated Bubble Chart

```python
# Animated bubble      delivery vs revenue, size = orders
fig_bubble = px.scatter(
    pdf_monthly,
    x="delivery_days_avg",
    y="revenue",
    animation_frame="month",
    animation_group="country",
    size="num_orders",
    color="country",
    hover_name="country",
    size_max=60,
    range_x=[0, pdf_monthly["delivery_days_avg"].max() * 1.2],
    range_y=[0, pdf_monthly["revenue"].max() * 1.2],
    color_discrete_map=COLOR_MAP,
    title="Delivery Performance vs Revenue (Animated Bubble)",
    height=500
)
fig_bubble.update_traces(marker=dict(opacity=0.75, line_width=1))
fig_bubble.show()
```

Design Tip

Always set range_x and range_y explicitly in animated charts; otherwise the axes rescale per

Dash: Architecture & Setup

Component Stack

dash — Flask-based Python server
dcc — Interactive widgets (Graph, Dropdown, Slider)
html — HTML layout building blocks
dash.dependencies — Input, Output, State

Request cycle:

1. User moves widget
2. Browser POSTs to Flask
3. Python callback runs
4. Returns serialised Plotly JSON
5. Browser re-renders chart

```
# Install in JupyterLab container
pip install dash dash-bootstrap-components

# docker-compose.yml      add port mapping
services:
  jupyterlab:
    ports:
      - "8888:8888"
      - "8050:8050"    # Dash app

# Run standalone app
python app.py
# http://localhost:8050

# Run inline in notebook
app.run(debug=True, port=8050,
        jupyter_mode="tab")
```

Port 8050

Dash default port. Already mapped in the Lab

Dash: Minimal App Layout

```
from dash import Dash, dcc, html, Input, Output, callback
import plotly.express as px
import pandas as pd

app = Dash(__name__)

COUNTRIES = ["All","KE","TZ","UG","RW"]

app.layout = html.Div([
    html.H1("East Africa E-commerce Dashboard",
            style={"textAlign":"center","color":"#800000",
                   "fontFamily":"Inter, sans-serif"}),

    html.Div([
        html.Label("Country:", style={"fontWeight":"bold"}),
        dcc.Dropdown(
            id="country-dd",
            options=[{"label":c,"value":c} for c in COUNTRIES],
            value="All", clearable=False,
            style={"width":"200px"}
        ),
    ], style={"display":"flex","alignItems":"center","gap":"10px",
              "padding":"10px 40px"}),
```

Dash: Callbacks — Input/Output Pattern

```python
@callback(
    Output("revenue-chart", "figure"),
    Output("orders-chart",  "figure"),
    Input("country-dd",     "value")
)
def update_charts(selected_country):
    if selected_country == "All":
        filtered = pdf_daily
    else:
        filtered = pdf_daily[pdf_daily["country"] == selected_country]

    fig_rev = px.line(
        filtered, x="date", y="revenue", color="country",
        title=f"Daily Revenue --- {selected_country}",
        markers=True, color_discrete_map=COLOR_MAP,
        labels={"revenue":"Revenue (USD)","date":"Date"}
    )
    fig_rev.update_layout(plot_bgcolor="white")

    fig_ord = px.bar(
        filtered.groupby("country")["num_orders"].sum().reset_index(),
        x="country", y="num_orders", color="country",
        title=f"Total Orders --- {selected_country}", Visualization
```

Dash: Advanced — RangeSlider, DatePicker, DataTable

```python
from dash import dash_table
import dash_bootstrap_components as dbc

app2 = Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
CATS = sorted(pdf_cat["category"].unique().tolist())

app2.layout = dbc.Container([
    dbc.Row([
        dbc.Col([html.Label("Category:"),
                 dcc.Dropdown(id="cat-dd", options=CATS,
                              multi=True, placeholder="All categories")], width=4),
        dbc.Col([html.Label("Order Value Range (USD):"),
                 dcc.RangeSlider(id="val-slider",
                     min=0, max=1000, step=25, value=[0,1000],
                     marks={0:"$0",250:"$250",500:"$500",1000:"$1K"})], width=5),
        dbc.Col([html.Label("Date Range:"),
                 dcc.DatePickerRange(id="date-range",
                     start_date="2025-01-01",
                     end_date="2025-12-31")], width=3),
    ], className="mb-3"),
    dbc.Row([
        dbc.Col(dcc.Graph(id="main-chart"), width=8),
        dbc.Col(dcc.Graph(id="pie-chart"),  width=4),
```

Folium: Bubble Map of East Africa

```
import folium
from folium.plugins import MarkerCluster

# City coordinates, country, and order counts
cities = {
    "Nairobi":       {"lat":-1.286,"lon": 36.817,"cty":"KE","orders":45000},
    "Mombasa":       {"lat":-4.043,"lon": 39.668,"cty":"KE","orders":18000},
    "Dar es Salaam":{"lat":-6.792,"lon": 39.208,"cty":"TZ","orders":38000},
    "Zanzibar City":{"lat":-6.165,"lon": 39.202,"cty":"TZ","orders": 8000},
    "Kampala":       {"lat": 0.347,"lon": 32.583,"cty":"UG","orders":22000},
    "Kigali":        {"lat":-1.943,"lon": 30.059,"cty":"RW","orders":15000},
}
COLORS = {"KE":"#C5A059","TZ":"#0a1628","UG":"#00964F","RW":"#E53935"}

m = folium.Map(location=[-2.5, 35.0], zoom_start=5,
               tiles="CartoDB positron")

for name, info in cities.items():
    radius = (info["orders"] / 1000) ** 0.55 * 5
    folium.CircleMarker(
        location=[info["lat"], info["lon"]], radius=radius,
        color=COLORS[info["cty"]], fill=True, fill_opacity=0.75,
        popup=folium.Popup(
```

Plotly Choropleth: African Country Data

```python
import plotly.express as px, pandas as pd

pdf_africa = pd.DataFrame({
    "iso3":        ["KEN","TZA","UGA","RWA","ETH","GHA","NGA","ZAF"],
    "country":     ["Kenya","Tanzania","Uganda","Rwanda",
                    "Ethiopia","Ghana","Nigeria","South Africa"],
    "revenue":     [2.1e6,1.8e6,0.9e6,0.5e6,0.3e6,0.7e6,1.2e6,0.4e6],
    "num_orders": [120000,95000,48000,28000,15000,38000,62000,18000],
    "mobile_pct": [78,72,65,81,45,60,55,40],  # % mobile-money
})

fig = px.choropleth(
    pdf_africa,
    locations="iso3", locationmode="ISO-3",
    color="revenue", scope="africa",
    hover_name="country",
    hover_data={"num_orders":":,","mobile_pct":":.0f%","revenue":":$,.0f"},
    color_continuous_scale="YlOrRd",
    title="E-commerce Revenue Across Africa --- 2025",
    labels={"revenue":"Revenue (USD)"},
    height=530
)
fig.update_geos(showcoastlines=True, coastlinecolor="white",
```

Apache Superset: Self-Serve BI

What is Apache Superset?

- Open-source BI & data exploration platform
- Connects to Spark Thrift Server, PostgreSQL, Druid, BigQuery, ...
- Drag-and-drop chart builder (no code for analysts)
- Built-in SQL IDE (`SQL Lab`)
- Role-based access control
- Embeddable iframes for portal integration

Add to docker-compose.yml:

```
superset:
  image: apache/superset:3.1.0
  ports: ["8088:8088"]
  environment:
    - SUPERSET_SECRET_KEY=lab9supersecret
  command: >
```

Superset Workflow

1. Settings → **Database Connections** → Add
2. Choose driver: Spark SQL, PostgreSQL, etc.
3. Create a **Dataset** from a table or SQL query
4. **Charts** → pick chart type, drag metrics
5. Build a **Dashboard** by arranging charts
6. Set refresh interval (e.g. every 30 s)

Spark Connection String

Grafana + Prometheus: Operational Dashboards

Use Cases in Big Data:

- Monitor Spark job stages, GC pauses, task failures
- Track cluster CPU/RAM/disk utilisation
- Alert when job exceeds SLA duration
- Real-time streaming pipeline throughput
- Kafka consumer lag monitoring

Sample PromQL queries:

```
# JVM heap used on executor nodes
jvm_memory_used_bytes{area="heap"}

# CPU usage rate (5-min window)
avg(rate(node_cpu_seconds_total
          {mode!="idle"}[5m])) * 100
```

Grafana Panel Types

Panel	Best for
Time series	Metrics over time
Stat	Single KPI value
Gauge	% utilisation
Bar chart	Category compare
Table	Tabular drill-down
Heatmap	Density over time
Logs	Log stream
Alerting	Threshold breach

Enable Spark Metrics

```
.config("spark.metrics.conf",
```

Big Data Visualization Principles

Aggregation Strategies

- **GROUP BY**: exact sums, counts, averages
- **Sampling**: `.sample(fraction=0.01)` for EDA
- **Binning**: histograms on bucketed values
- **Approximate**: `approx_count_distinct()`, HyperLogLog sketches
- **Window avg**: rolling means smooth noisy series

Overplotting Solutions

- Alpha transparency (`opacity=0.1`)

Colorblind-Safe Palette Rules

- Avoid red–green combos (affects ∼8% of men)
- Prefer **Okabe–Ito** or **Viridis**
- **Sequential** (magnitude): YlOrRd, Blues, Viridis
- **Diverging** (neg/pos): RdBu, PiYG, coolwarm
- **Categorical**: qualitative palettes, max 7–8 hues
- Test at coblis simulator

Data Ink Ratio (Tufte)

Chart Selection Guide

Question → Chart type

- **Comparison**: Bar chart, Radar
- **Trend over time**: Line, Area
- **Correlation**: Scatter, Bubble
- **Part-of-whole**: Pie / Donut ($\leq$5 slices), Treemap
- **Distribution**: Histogram, KDE, Box, Violin
- **Geographic**: Choropleth, Bubble map
- **Matrix**: Heatmap, Parallel coords
- **Flow/hierarchy**: Sankey, Treemap, Sunburst

When NOT to Use

Pie	>5 slices; not part-of-whole
3-D bar	Always misleading
Dual axis	Misleads scale comparison
Raw scatter	>10K points (overplot)
Line chart	Unordered categorical x-axis

Actionable vs. Decorative

Actionable dashboard:

- Answers a specific decision question
- Refreshes with live data

Lab 9 Tasks Summary

Today's Lab (2 hours 40 minutes)

1. **Pipeline** — Load orders parquet → `groupBy` → `toPandas()`
2. **Matplotlib/Seaborn** — 3-panel subplot: revenue bar, orders line, heatmap
3. **Plotly Express** — bar (grouped & stacked), line, scatter, sunburst, choropleth
4. **Subplots** — 2×2 executive dashboard with `make_subplots`
5. **Animation** — Monthly revenue animated bar chart with `animation_frame`
6. **Dash App** — Dropdown + RangeSlider + 2 synchronised callback charts
7. **Folium** — East Africa bubble map by city, saved as `.html`

Assignment (due Sun 17 May 23:59 EAT)

100 marks | 5 tasks | submit notebook + HTML exports to:
`s3a://submissions/week09/<rollno>/`

Grading Criteria

Correctness · Chart quality · Interactivity depth · Code clarity · Storytelling & reflection quality

Docker Service Reference

Active Services — Lab 9

Service	URL	Credentials	Purpose
JupyterLab	`localhost:8888`	token in logs	Notebooks
Spark Master	`localhost:8080`	—	Cluster UI
Spark RPC	`spark-master:7077`	—	Submit jobs
Spark UI	`localhost:4040`	—	Job monitoring
MinIO Console	`localhost:9001`	minioadmin/minioadmin	Object storage
Kafka Broker	`kafka:9092`	—	Streaming bus
Grafana	`localhost:3000`	admin/admin	Dashboards
Prometheus	`localhost:9090`	—	Metrics store

Data Locations

Orders dataset:

```
s3a://warehouse/lab04/orders_parquet/
```

Note

URLs may differ on the exam machines. Use `docker compose ps` and container logs to

Practical Exam (2 hours): End-to-End Big Data Pipeline

Goal

Build a **working mini data platform** from ingestion to dashboards: **Kafka** → **Spark** → **MinIO/Postgres** → **Prometheus/Grafana**.

What you are allowed to use

- Internet + any AI tools (must cite prompts in README)
- Any libraries already available in the lab Docker images

What you must demonstrate

- Services running (compose), data flowing, storage visible
- Streaming + batch/triggered job + monitoring dashboard

Deliverables (submit a ZIP)

- Notebook or scripts + an `exam_README.md`
- Your Kafka topic name(s) + schema
- Evidence: screenshots (MinIO, Grafana, Spark UI)
- Exported Grafana dashboard JSON

Exam Data: “Clickstream + Orders” (Provided Generator)

Input Stream

You will produce JSON events to Kafka topic `exam.clickstream.<rollno>`.

Event fields (minimum):

- `event_time` (ISO timestamp), `user_id`, `country`, `event_type`
- `product_id`, `category`, `price_usd`

Rule

Your pipeline must handle **bad records** (missing fields / wrong types): count them and store them to a `quarantine/` location.

Task A (25 marks): Kafka Ingestion + Streaming Storage

Build and prove:

1. A producer generates $\geq$ 2,000 events in 2–3 minutes.
2. Spark Structured Streaming reads Kafka.
3. Write raw events to MinIO as Parquet, partitioned by `dt=YYYY-MM-DD`.
4. Write quarantine (bad records) to `s3a://warehouse/exam/quarantine/`.

Evidence

Screenshot: Kafka topic has messages; MinIO shows new Parquet files.

Task B (25 marks): Real-Time Aggregation + Serving Store

Compute (streaming aggregation):

- KPI1: events/sec (throughput)
- KPI2: revenue/minute by country (sum of `price_usd` for `purchase`)
- KPI3: top-5 categories in last 10 minutes

Serve to PostgreSQL:

- Upsert a table `exam_kpis_<rollno>` every micro-batch

Rule

Your serving table must be queryable from SQL (show at least 2 queries).

Task C (20 marks): Triggered Batch Job + Data Quality Checks

Create a triggered job (choose one):

1. **Airflow DAG** that runs once and materialises a daily summary table, or
2. A **scheduled Spark job** (cron-style) that writes a batch summary.

Minimum data-quality checks:

- Null-check rate (`country`, `event_time`)
- Outlier check: `price_usd > 10000` (count)
- Duplicate check on (`user_id`, `event_time`, `product_id`)

Evidence

Screenshot: DAG run success (or job log) + output table/files exist.

Task D (20 marks): Prometheus Metrics + Grafana Dashboard

Expose at least 3 custom metrics (your choice), e.g.:

- `events_processed_total`
- `bad_records_total`
- `batch_duration_seconds`

Build a Grafana dashboard with 4 panels:

1. Throughput (time series)
2. Bad records (stat)
3. Revenue/min by country (time series)
4. Top categories (table)

Alert

Create one alert rule: bad records $>$ 1% for 2 minutes.

Submission Checklist (ZIP) + Marking Rubric

ZIP must contain:

- `exam_README.md` (setup, commands, AI prompt citations)
- `producer.py` (or notebook cell) + topic name
- `streaming_job.py` (or notebook) + checkpoint path
- `dq_job.py` or `dags/exam_dag.py`
- `grafana_dashboard.json`
- `screenshots/` (MinIO + Grafana + Spark UI)

Rubric

- Task A: 25
- Task B: 25
- Task C: 20
- Task D: 20
- Clarity (README + reproducibility): 10

Good luck!

Submit a reproducible ZIP (it should run on a clean machine).

Dr. Innocent Nyalala — innocent@iitmz.ac.in