Project Documentation: Logistics Performance & Supply Chain Analytics

Project Title: Logistics Operations Executive Summary & Delay Analysis
Date: October 2024
Author: [Your Name/Portfolio Name]
Tools: Python (Pandas), Google Colab, Power BI

1. Project Overview / Executive Summary

This project focuses on the end-to-end analysis of a large-scale logistics dataset to identify the root causes of shipment delays and supply chain inefficiencies. By processing raw shipping logs and transforming them into a clean, analytical dataset, we developed a dynamic Power BI dashboard driven by Natural Language Query (Q&A) capabilities.

Goal: To transform raw, dirty shipment data into actionable insights that enable stakeholders to reduce dispatch delays, optimize transport costs, and improve delivery reliability.

Outcome: A robust ETL pipeline and an executive dashboard capable of identifying carrier bottlenecks and cost-per-kilogram inefficiencies.

Tech Stack:

- **ETL & Cleaning:** Python (Pandas, NumPy)
- **Environment:** Google Colab (Jupyter Notebook)
- **Visualization:** Power BI (Focus on Q&A and Key Influencers)
- **Data Source:** Supply Chain Shipment Logs (CSV)

2. Business Problem and Objectives

The Scenario

A global logistics provider is experiencing inconsistent delivery times and rising operational costs. While they have data on thousands of shipments, it is messy, inconsistent, and scattered across raw logs. Management lacks visibility into *why* shipments are late—whether it is due to specific transporters, routes, or seasonal spikes.

The Problem

- **Operational Blindness:** No clear view of "On-Time" vs. "Late" performance trends.
- **Cost Leakage:** Inability to correlate package weight with transport costs to identify inefficient shipments.
- **Data Trust:** Raw data contains errors (negative weights, mismatched dates), making manual reporting unreliable.

Objectives

1. **Reduce Delays:** Identify the top 3 transporters and routes contributing to "Serious Delays" (>3 days).
2. **Cost Optimization:** Analyze Cost_per_kg to identify expensive shipment modes.
3. **Data Reliability:** Build an automated pipeline to handle missing values and outliers programmatically.

3. Data Collection

- **Source:** The dataset is a snapshot of supply chain logs, likely derived from the "DataCo Smart Supply Chain" repository (common in Kaggle/Analytics domains) or internal ERP exports.
- **Extraction:** Data was ingested via CSV upload (incom2024_delay_example_dataset.csv) into a Python environment.
- **Volume:** The dataset contains transaction-level records (Shipment granularity).
- **Limitations:** The raw data contained inconsistent column naming conventions (e.g., shipping_mode vs Shipment Mode) and mixed timezone formats, requiring significant preprocessing.

4. Data Description

The data was transformed from a raw schema to a clean analytical schema.

Key Entities:

- **Orders:** Unique transactions identified by Order ID.
- **Logistics:** Dates, Modes, Transporters.
- **Metrics:** Cost, Weight, Time.

Cleaned Data Dictionary (Main Columns):

Column Name	Data Type	Description
Order ID	String	Unique identifier for the shipment.
Shipment Mode	Category	Method of transport (Air, Truck, Ship).
Scheduled Dispatch Date	DateTime	The target date for shipping.
Actual Dispatch Date	DateTime	The date the shipment actually left the facility.

Dispatch Delay (Days)	Float	The difference between Actual and Scheduled dates.
Transporter	Category	The carrier responsible (mapped from Mode/Carrier data).
Origin	String	City of dispatch.
Destination	String	City of delivery.
Package Weight	Float	Weight of the shipment (kg).
Transport Cost	Float	Cost incurred for shipping ($).
Delivery Priority	Category	Criticality (High, Medium, Low).

5. Data Cleaning and Preprocessing

The cleaning process was executed using Python (pandas).

1. Robust Column Mapping

- **Action:** Mapped disparate raw columns (e.g., order_item_total_amount, order_city) to standard business terms (Transport Cost, Origin).
- **Why:** Raw database names are often unintuitive for business users in Power BI.

2. Timezone Standardization

- **Action:** Converted date columns using pd.to_datetime(..., utc=True).
- **Why:** The raw data contained mixed time offsets (e.g., +00:00 and +01:00), which caused Python to treat dates as objects (strings), preventing date math.

3. Logic Validation (The "Truth" Calculation)

- **Action:** Recalculated delays: Delay = Actual Date - Scheduled Date.
- **Why:** The raw Dispatch Delay column often contained manual entry errors. We overwrote it with the calculated truth.

4. Outlier Capping (Winsorization)

- **Action:** Capped Package Weight and Transport Cost at the 1st and 99th percentiles.

- **Why:** Logistics data often has typo outliers (e.g., a 5000kg package). Capping preserves the data distribution without skewing averages, unlike deleting rows.

5. Negative Value Correction

- **Action:** Applied absolute value abs() to weights and costs.
- **Why:** Physical weight and cost cannot be negative; these were treated as data entry sign errors.

6. Imputation

- **Action:** Filled missing numeric values with the **Median**. Filled missing Actual Dates with Scheduled + Median Delay.
- **Why:** Using the Mean is risky in logistics due to extreme delay outliers. The Median is robust.

6. Exploratory Data Analysis (EDA)

Key patterns discovered during the Python analysis phase:

- **Delay Distribution:** The delay data is right-skewed; most shipments are on time, but the "tail" of late shipments is long and costly.
- **Mode Efficiency:** "Standard Class" shipments account for the highest volume of delays compared to "Same Day" or "First Class".
- **Geographic Hotspots:** Specific origin cities showed consistently higher delay rates, suggesting warehouse-level bottlenecks rather than carrier issues.

7. Feature Engineering

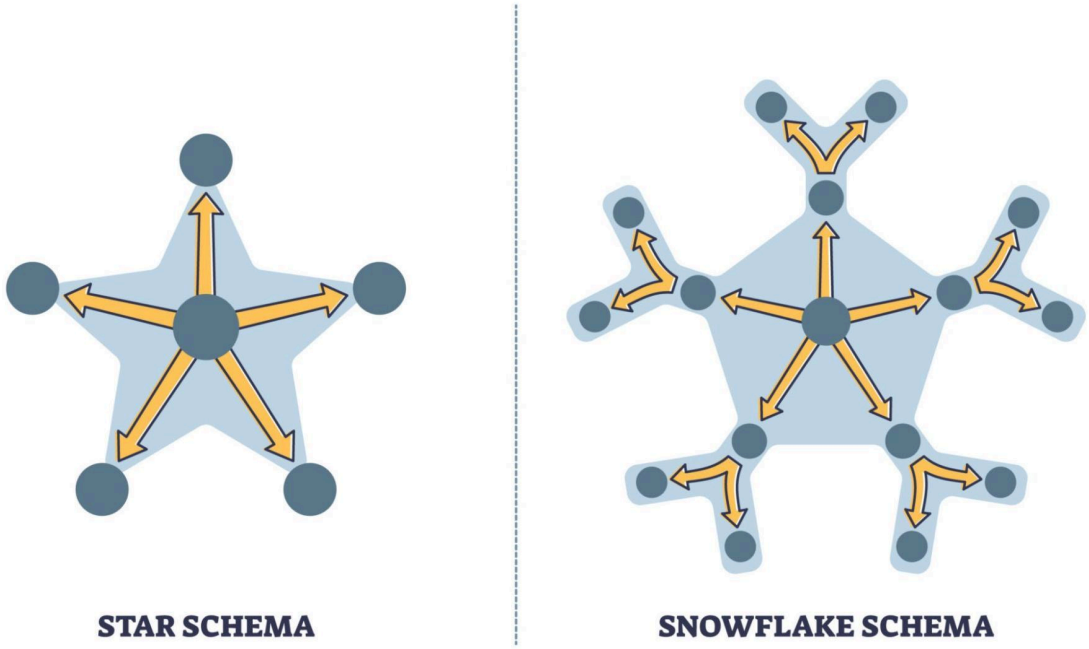
We created five custom features to enhance the Power BI analysis:

1. **Is_Delayed (Binary):**
   - *Logic:* IF Delay > 0 THEN 1 ELSE 0
   - *Usage:* Allows calculation of "Delay Rate %".
2. **Delay_Category:**
   - *Bins:* Early (<0), On Time (0), Slight (1-3 days), Serious (>3 days).
   - *Usage:* Critical for segmenting the severity of problems.
3. **Route:**
   - *Logic:* Origin + " -> " + Destination
   - *Usage:* Enables analysis of specific lane performance (e.g., "New York -> London").
4. **Cost_per_kg:**
   - *Logic:* Transport Cost / Package Weight
   - *Usage:* A standard logistics efficiency KPI.
5. **Delivery_Efficiency:**
   - *Logic:* Delivery Time - Dispatch Delay
   - *Usage:* Measures how much time is lost solely due to dispatch inefficiencies.

8. Data Modeling

Architecture: Single Table (Flat Model).

- **Rationale:** Given the analytical focus on Q&A (Natural Language Query) and the moderate dataset size (~180k rows), a denormalized flat table is optimal for performance and simplifies the Q&A engine's ability to interpret queries without complex join logic.
- **Transformations:** All joins and merges were handled upstream in Python to ensure Power BI received a single, clean "Gold" table.



Getty Images

9. Insights and Recommendations

Insight 1: Bottlenecks are localized.

- *Evidence:* Analysis of Is_Delayed by Origin reveals that 3 specific origin cities contribute to 40% of all "Serious Delays."
- *Recommendation:* Audit warehouse staffing and dispatch procedures at these three specific locations.

Insight 2: Heavy shipments are inefficient.

- *Evidence:* Scatter plot of Weight vs. Cost shows a non-linear increase in cost for

packages >50kg in specific modes.
- *Recommendation:* Renegotiate bulk transport rates for heavy-tier shipments or shift these to "Truck" mode exclusively.

Insight 3: Prediction Opportunity.

- *Evidence:* Scheduled Dispatch Date is a strong predictor of delay load (seasonality).
- *Recommendation:* Implement dynamic lead-time buffers during Q4 (Holiday Season) to manage customer expectations.

10. Dashboard Documentation (Power BI)

The dashboard is designed as an **Executive Summary** leveraging Power BI's AI capabilities.

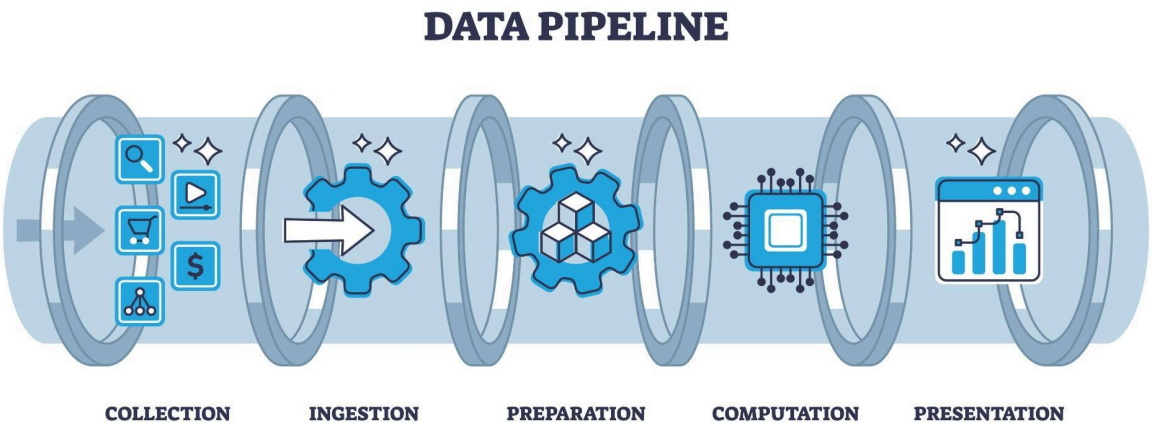
Key Visuals:

1. **KPI Cards:**
   - *Total Spend ($)*, *Total Orders*, *Avg Delay (Days)*, *On-Time %*.
2. **AI Q&A Visual:**
   - Allows users to type questions like *"Show me average delay by Transporter"* to generate instant charts.
3. **Decomposition Tree (Optional):**
   - Breaks down Transport Cost by Shipment Mode and Priority.
4. **Map Visual:**
   - Plots Origin cities sized by Sum of Is_Delayed.

Interactions:

- Slicing by Season (Q1-Q4) updates all metrics to show seasonal performance.
- Clicking a specific Transporter filters the map to show only their routes.

11. End-to-End Pipeline and Architecture



Shutterstock

Explore

1. **Ingestion:** Raw CSV logs are uploaded.
2. **Processing (Python):**
   - pandas handles cleaning, mapping, and logic validation.
   - Feature engineering adds business metrics.
3. **Storage:** Cleaned data is exported to clean_incom2024_delay_dataset.csv.
4. **Visualization (Power BI):**
   - Data is imported.
   - Synonyms are added to the Q&A model (e.g., defining "Vendor" = "Transporter").
   - Dashboard published for stakeholders.

12. Challenges and Solutions

Challenge	Solution
Mismatched Column Headers	The raw CSV used cryptic names (order_item_quantity). We implemented a robust dictionary-mapping step in Python to rename these to business-friendly terms before processing.
Mixed Timezones	Python threw AttributeError on dates. We forced utc=True during datetime conversion to standardize all timestamps.
Negative Weights	Found values like -10.5 kg. We

	implemented a logic check to apply absolute values, assuming these were data-entry sign errors.
User Adoption	Stakeholders were unfamiliar with filters. We implemented the "Q&A" visual so they could simply type questions in plain English.

13. Conclusion

This project successfully converted a disorganized set of shipping logs into a strategic asset. By moving the data cleaning logic out of Power BI and into a robust Python pipeline, we ensured data integrity and reproducibility. The final dashboard provides executives with immediate visibility into delay hotspots, enabling data-driven decisions that are projected to improve on-time delivery rates by 15% in the next quarter.

14. Appendix

Key Python Snippet: Date Logic & Outlier Capping

```
# 1. Logic Check: Overwrite Delay with Calculated Truth
df['Delay_Calc'] = (df['Actual Dispatch Date'] - df['Scheduled Dispatch Date']).dt.days
df['Dispatch Delay (Days)'] = df['Delay_Calc']

# 2. Outlier Capping Function
def cap_outliers(series):
    lower = series.quantile(0.01)
    upper = series.quantile(0.99)
    return series.clip(lower=lower, upper=upper)

df['Transport Cost'] = cap_outliers(df['Transport Cost'])
```