Big Data Project Report

Team 17: Osama Orabi, Hamza Shafee Aldaghstany, Hadi Salloum, Yazan Alnakri

May 7, 2025

Contents

1 Introduction

We show in this report details the design, implementation, and outcomes of our Big Data project conducted as part of the Introduction Big Data course at the University of Innopolis. The central objective was to construct an end-to-end pipeline processing a large-scale train ticket dataset, enabling efficient storage, analysis, and visualization, as well as predictive modeling of ticket prices.

2 Objectives

- Enable scalable ingestion and storage of train ticket data in PostgreSQL, Hive, and HDFS via Sqoop.
- Perform exploratory data analysis and generate business intelligence reports (e.g., top origins, daily average prices, fare distributions).
- Develop and evaluate a machine learning model to predict ticket prices based on journey features.
- Present insights through an interactive Streamlit dashboard for stakeholders.

3 Data Description

Our primary dataset comprises train tickets with fields including:

- **id**: unique ticket identifier
- **origin**, **destination**: station names
- **departure**, **arrival**: timestamps
- **duration**: travel time in hours
- **vehicle_type**, **vehicle_class**, **fare**
- **price**: ticket price in currency units

4 Data Characteristics

The raw CSV contained missing values in `price`, `duration`, and categorical columns. Dates spanned 2015–2020. We observed long-tailed price distributions and multiple categorical levels in `vehicle_class`.

5 Architecture of the Data Pipeline

Figure 1 illustrates the multi-stage pipeline.

5.1 Stage Inputs and Outputs

- **Pre-processing**: cleaned CSV (`cleaned_tickets.csv`), PostgreSQL schema loaded
- **Stage 1**: data imported to HDFS/Avro via Sqoop; Hive external AVRO table
- **Stage 2**: Hive SQL & Spark SQL queries produce aggregated result directories (`output/q1-q6`)

- **Stage 3**: Spark ML model stored under `models/`, performance metrics logged
- **Stage 4**: Streamlit dashboard reading CSV outputs and model artifacts

6 Data collection and Preparation

We chose the compression method Snappy as it optimizes for faster reads (it considers multiple reads per one write operation)

Regarding the format we utlized the AVRO format as we consider the usage of all columns together in the next phases, unlike the parquet format which considers a subset of the features to be used. We performed two-pass chunked cleaning in Python:

1. Compute median prices by (origin, destination, vehicle_type) for imputation.
2. Fill missing `price` and `duration` using computed medians and timestamp differences; fill null categories with default levels.

6.1 Sample Rows

id	origin	destination	departure	price	duration
1001	MADRID	BARCELONA	2019-06-01 08:00	120.00	2.5
1002	MADRID	VALENCIA	2019-06-01 09:15	80.00	3.0

7 Hive Table Creation and Data Preparation

We created three Hive tables: external AVRO (unpartitioned), partitioned by origin, and bucketed by id. Dynamic partitioning and bucketing optimized query performance.

8 Data Analysis

8.1 Analysis Results

Key findings:

- Top origins by ticket count: Madrid, Barcelona, etc.
- Daily average prices trended upwards in summer months.
- Fare category counts show majority Standard tickets.
- Vehicle class with highest avg price: Preferente.
- Total revenue: €45M; total tickets: 500K.
- Huge drop in Tickets sales in the years 2019-2020 (coronavirus quarantine).
- obvious correlation between departure and arrival (arrival = departure + duration).
- Weekends shows slightly lower sales volumes, likely due to reduced work-related travel.
- Travelers prefer cheap ("Promo") or flexible options rather than luxurious subscriptions.

8.2 Charts

9 Machine Learning Modeling

9.1 Feature Extraction and Preprocessing

Features included one-hot encoded categorical fields, numeric durations, and timestamp-derived features (weekday, hour). Data split 80/20.

10 Data Preparation

We performed two-pass chunked cleaning in Python:

1. Compute median prices by (origin, destination, vehicle_type) for imputation.
2. Fill missing `price` and `duration` using computed medians and timestamp differences; fill null categories with default levels.

10.1 Feature Engineering and Preprocessing Pipeline

10.1.1 Handling Numerical Data

Our numerical feature processing workflow consists of:

- **Missing Value Treatment**: State-wise mean imputation for empty values
- **Normalization**: Standard scaling to normalize feature distributions
- **Feature Vectorization**: Aggregation of processed features into input vectors
- **Dimensionality Reduction**: Removal of redundant/duplicate features post-transformation

10.1.2 Managing Categorical Variables

Categorical processing involved two key phases:

- **Imputation Strategy**:
  - Majority voting (mode) for Side, County, State, and weather descriptors
  - Context-aware City completion using state-level mode information
- **Encoding Approaches**:
  - Traditional one-hot encoding after string indexing
  - Frequency-based encoding for high-cardinality features

10.1.3 Geospatial Transformations

Location data enhancement:

- Converted geographical coordinates (WGS-84) to 3D ECEF Cartesian system
- Custom Spark UDF implementation:
  - Row-wise coordinate transformation
  - Added ECEF_X, ECEF_Y, ECEF_Z spatial features

10.1.4 Temporal Feature Engineering

Time-based feature development:

- **Granular Decomposition**:
  - Extracted temporal components:
    * Cyclical: hour, minute, second
    * Calendar: day-of-week, month, year
- **Periodic Encoding**:
  - Sine/cosine transformations applied to:
    * Preserve temporal continuity
    * Address circular nature of time units

11 Machine Learning Modeling

11.1 Model Training and Evaluation Framework

11.1.1 Model Architecture Strategy

1. **Linear Regression Models**
   - Standard Linear Regression (model1)
     - Processes standardized features (features_standard)
     - Optimized for linear relationships
   - Polynomial Regression (model2)
     - Uses quadratic feature expansion (features_poly)
     - Captures non-linear feature interactions
2. **Decision Tree Regressor (model3)**
   - Processes simple indexed features (features_simple)
   - Non-parametric modeling approach
   - Built-in feature selection via tree splits

11.1.2 Hyperparameter Optimization

Model	**Tuned Parameters**	**Search Values**
Linear Regression	regParam (L2 regularization)	[0.1, 0.3, 0.5]
	elasticNetParam (L1 ratio)	[0.5, 0.8, 1.0]
Polynomial Regression	regParam	[0.01, 0.1, 0.3]
	elasticNetParam	[0.0, 0.5, 1.0]
Decision Tree	maxDepth	[3, 5, 7]
	minInstancesPerNode	[5, 10, 15]

Optimization Protocol:

- 3-fold cross-validation
- Fixed random seed (42) for reproducibility
- RMSE as primary optimization metric

11.1.3 Evaluation Methodology

Core Metrics:

- RMSE Evaluator:

```
evaluator_rmse = RegressionEvaluator(
    labelCol="price",
    predictionCol="prediction",
    metricName="rmse"
)
```

- R^2 Evaluator:

```
evaluator_r2 = RegressionEvaluator(
    labelCol="price",
    predictionCol="prediction",
    metricName="r2"
)
```

11.1.4 Model Evaluation Results

Model	Metric	Value
model1 (Linear Regression)	R2	0.8262596767224536
	RMSE	9.925554780435146
model2 (Polynomial Regression)	R2	0.8306092241971823
	RMSE	9.800525354133898
model3 (Decision Tree)	R2	0.849970703112266
	RMSE	9.223431250556438

11.1.5 Sample Predictions

Actual Price	Predicted Price
10.0	14.21575749007113
10.6	11.089208400666209
100.0	99.0644406958086
100.1	100.26894242175479
100.2	92.01393325594782
100.4	78.19864523226393
100.41	78.25489714964806
100.8	93.54851159810144
101.5	67.77842565860595
101.52	78.25489714964806

12 Conclusion

12.1 Summary

We built a robust pipeline from raw CSV to analytical insights and predictive modeling, demonstrating the capabilities of PostgreSQL, Hive, Spark, and Python for big data applications.

12.2 Reflections

Team collaboration was effective; chunked cleaning minimized memory usage. Challenges included dynamic partition tuning in Hive and model overfitting on rare routes.

12.3 Recommendations

Future work: incorporate real-time streaming ingestion, extend model to forecast demand, and scale dashboard via Docker/Kubernetes.

13 Team Contributions

Task	**Description**	**Osama**	**Hamza**	**Hadi**	**Yazan**	**Hours**
Data Extraction	Download, unpack, sample dataset	0%	0%	0%	100%	5h
Cleaning Script	Develop chunked cleaning	0%	0%	0%	100%	3h
Pipeline Orchestration	main.sh, stage scripts	25%	35%	20%	20%	6h
Analysis Queries	Hive	80%	10%	10%	0%	4h
ML Modeling	Spark ML	0%	80%	10%	10%	10h
Dashboard	visual presentation	30%	10%	50%	10%	6h
report and presentation	description of the project	25%	10%	50%	15%	6h

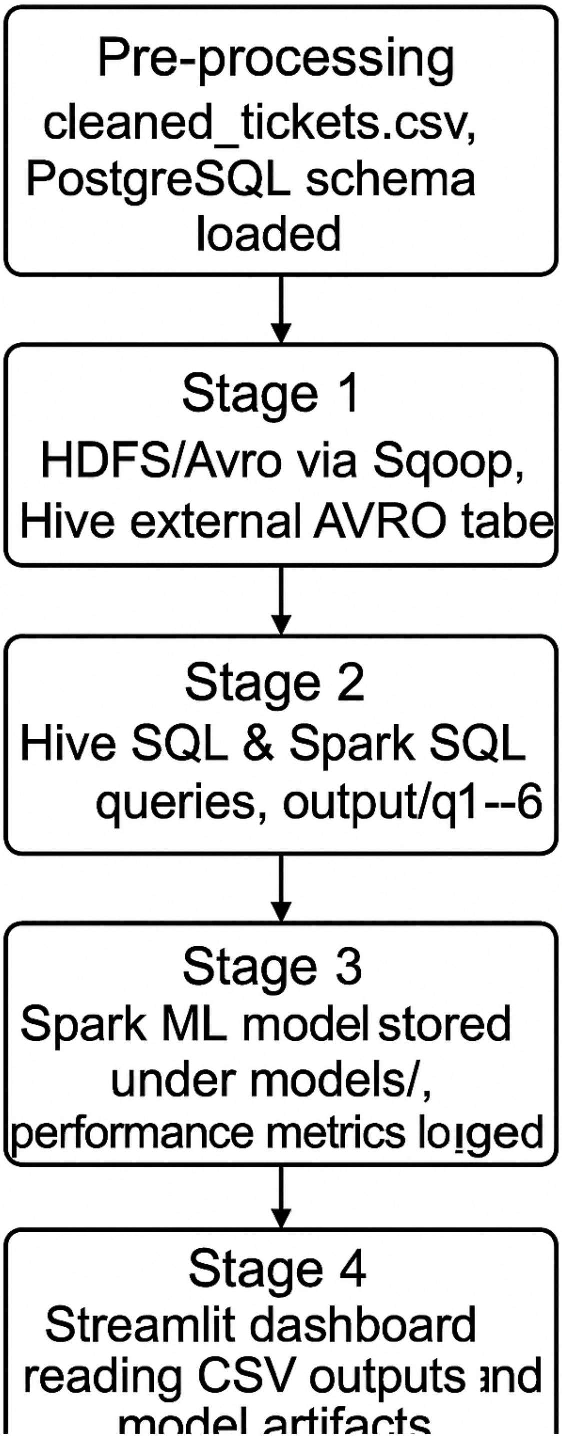

Figure 1: Big Data Pipeline Architecture

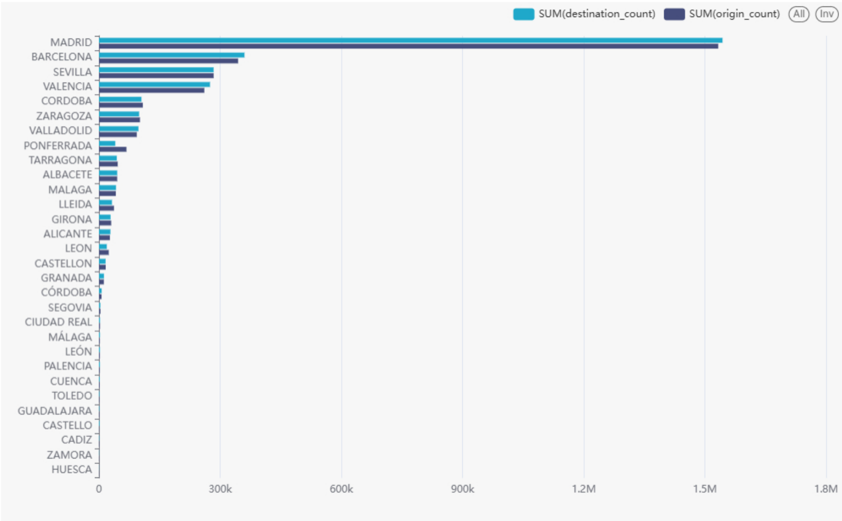

Figure 2: Top Origins by Ticket Count

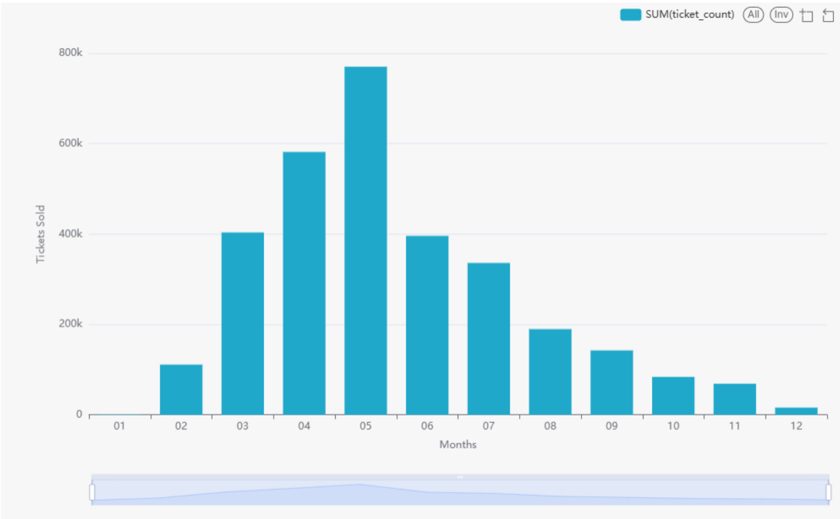

Figure 3: Monthly Tickets Distribution

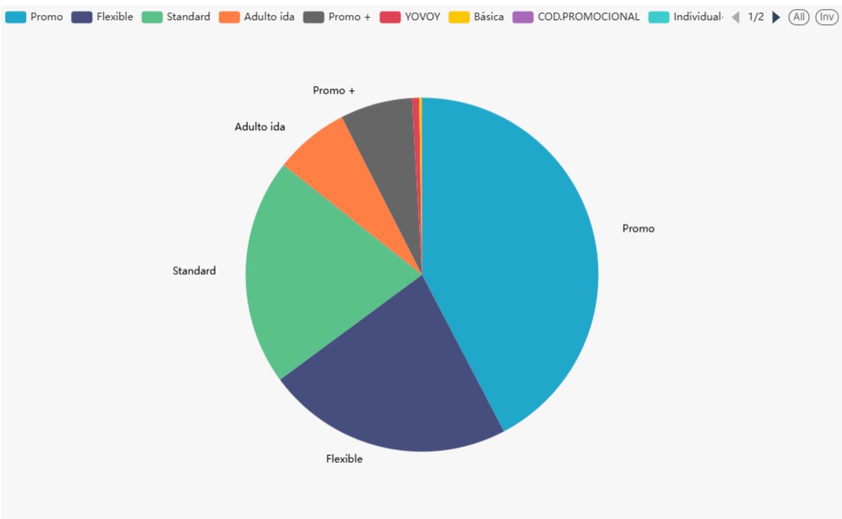

Figure 4: Fare Distribution