Course: Big Data Analytics

Scalable Big Data Analytics on Yelp Dataset:

A Pipeline for Real-Time Sentiment Analysis

Presented by:
Sara Sbissi
Imen Kenza Chouaieb

Professor:
ABDELKADER Manel

Academic Year
2025 - 2026

February 11, 2026

Contents

Chapter 1

Introduction

In the modern digital economy, User-Generated Content has become one of the most influential drivers of business reputation and consumer decision-making. Online platforms like Yelp, Google reviews and TripAdvisor generate millions of customer reviews every day, which reflect real-time opinions about products, services and experiences. This large flow of feedback represents a valuable source of insight but also presents a significant challenge: data overhead. Manually analyzing large volumes of unstructured text data is no longer feasible. Thus, business analysts struggle to extract timely insights related to service quality, customer satisfaction, emerging complaints or market trends. As a result, critical indicators are unnoticed. These challenges create a strong need for scalable, automated, and intelligent data processing systems that are capable of transforming raw User-Generated Content to actionable business intelligence and insights.

The objective of this project is to design and implement a big data analytics pipeline that processes large-scale review data efficiently and extracts meaningful insights for business decision-making. More specifically, the project aims to achieve a distributed data pipeline capable of ingesting and processing millions of records, ensuring horizontal scalability, implement a streaming ingestion component that simulates real-time reviews, apply sentiment analysis to predict the business rating, and develop dashboards that convert processed data into actionable key performance indicators.

The project leverages the Yelp Open Dataset, a large-scale, real-world dataset released by Yelp for academic research purposes. The data volume is approximately 7 million reviews, 150 thousand businesses, and 200 thousand users.

Chapter 2

Dataset Profile

Yelp is an online platform that enables users to discover, review, and rate local businesses such as restaurants, cafes, retail stores, and service providers. Founded in 2004, Yelp has grown into one of the largest sources of User-Generated Content (UGC) related to local commerce. Users contribute written reviews, star ratings, and engagement signals, while businesses are described through detailed metadata such as location, categories, and operational attributes. From a data perspective, Yelp represents a large-scale, real-world ecosystem where consumer opinions, business characteristics, and user behavior intersect. In this project, the focus was on three dataset files: business, user, and reviews. The tip, checkin and photo files were excluded to reduce pipeline complexity. All dataset files are distributed as line-delimited JSON, enabling efficient parallel ingestion and transformation using Spark's DataFrame API.

Feature Name	**Type**	**Example**
business_id	string	"tnhfDv5Il8EaGSXZGiuQGg"
name	string	"Garaje"
address	string	"475 3rd St"
city	string	"San Francisco"
state	string	"CA"
postal code	string	"94107"
latitude	float	37.7817529521
longitude	float	-122.39612197
stars	float	4.5
review_count	integer	1198
is_open	integer	1
attributes	object	{"RestaurantsTakeOut": true, "BusinessParking": {...}}
categories	array of strings	["Mexican", "Burgers", "Gastropubs"]
hours	object	{"Monday": "10:00-21:00", ...}

Table 2.1: Structure of business.json

LaTeX Code for user.json

Feature Name	Type	Example
review_id	string	"zdSx_SD6obEhz9VrW9uAWA"
user_id	string	"Ha3iJu77CxlrFm-vQRs_8g"
business_id	string	"tnhfDv5Il8EaGSXZGiuQGg"
stars	integer	4
date	string	"2016-03-09"
text	string	"Great place to hang out after work..."
useful	integer	0
funny	integer	0
cool	integer	0

Table 2.2: Structure of review.json

Feature Name	Type	Example
user_id	string	"Ha3iJu77CxlrFm-vQRs_8g"
name	string	"Sebastien"
review_count	integer	56
yelping_since	string	"2011-01-01"
friends	array of strings	["wqoXYLWmpkEH0YvTmHBsJQ", ...]
useful	integer	21
funny	integer	88
cool	integer	15
fans	integer	1032
elite	array of integers	
average_stars	float	4.31
compliment_hot	integer	339
compliment_more	integer	668
compliment_profile	integer	42
compliment_cute	integer	62
compliment_list	integer	37
compliment_note	integer	356
compliment_plain	integer	68
compliment_cool	integer	91
compliment_funny	integer	99
compliment_writer	integer	95
compliment_photos	integer	50

Table 2.3: Structure of user.json

Chapter 3

Technology

Our Big Data analytics pipeline leverages modern cloud-native technologies optimized for scalability, performance, and cost-efficiency. The architecture is designed around the Medallion Architecture (Bronze-Silver-Gold), implementing best practices for data lakehouse patterns while accommodating the constraints of a free-tier environment.

3.1 Core Platform: Databricks Free Edition

We selected Databricks Free Edition as our primary platform for several strategic reasons:

- Serverless compute eliminates infrastructure management overhead, allowing focus on data engineering
- Zero cost for free tier while providing production-grade features (Unity Catalog, Delta Lake, collaborative notebooks)
- Production-ready architecture: Although running on single-node, our code uses distributed computing patterns (partitioning, caching) that seamlessly scale to multi-node clusters without modification
- Development velocity: Instant startup times vs. 5–10 minute cluster provisioning in traditional Hadoop
- Limitation mitigation: The Single-node constraint (~15GB RAM) was addressed through batch processing and incremental loading strategies

3.2 Data Processing Engine: Apache Spark 3.5

Apache Spark presents a unified platform for batch processing, SQL analytics, and ML (MLlib), eliminating the need for multiple specialized tools.

- Performance: In-memory processing delivers 10–100x speedup vs. disk-based MapReduce
- Scalability: Horizontal scaling from single machine to thousands of nodes with fault tolerance through RDD lineage
- Rich ecosystem: DataFrames API, Spark SQL (ANSI compliant), MLlib, Structured Streaming

3.3 Storage Layer: Delta Lake

Delta Lake provides an ACID-compliant storage layer on top of cloud object storage, addressing critical limitations of traditional data lakes:

- ACID transactions ensure data integrity during concurrent read/write operations
- Schema enforcement prevents data quality issues through automatic validation
- Time travel enables querying historical data versions and rollback capabilities
- Performance optimizations: Z-ordering for faster queries, automatic file compaction

3.4 Data Governance: Unity Catalog

Unity Catalog provides enterprise-grade data governance capabilities, essential for production data platforms:

- Centralized governance with fine-grained access control and audit logging
- Metadata management provides single source of truth for all data assets
- Replaces deprecated DBFS FileStore with governed Volumes storage in free tier
- Compliance-ready with data lineage tracking and retention policies

3.5 Machine Learning

For machine learning tasks we used libraries built natively on Spark. Spark MLlib handled scalable feature engineering and predictive modeling. Using native Spark libraries ensures the ML models can scale horizontally as the dataset grows.

3.6 Scalability Pathway

Our technology choices ensure a clear migration path to production:

Current (Free Tier)	**Production Scaling**	**Notes**
Single-node (1 driver)	Multi-node cluster (1 driver + N workers)	Horizontal scalability
15GB RAM	Hundreds of GB per node	Memory expansion
~12 min processing time	~2–3 min (estimated 10-node cluster)	Performance gains
Manual execution	Orchestrated via Databricks Workflows	Automation
Free Edition	Standard/Premium tier with SLA guarantees	Enterprise readiness

Chapter 4

Data Ingestion Strategy

The ingestion phase establishes the "Bronze" layer of the Medallion architecture, transforming raw external files into governed cloud assets.

4.1 Governance & Namespace Configuration

To ensure organized data management, a dedicated catalog and schema (`yelp_project.raw_data`) were established within the Unity Catalog. This structure provides a centralized namespace that aligns with enterprise governance standards, allowing for fine-grained access control and clear separation between raw ingestion and downstream transformations.

4.2 Volume-Based Landing Zone

A Unity Catalog Volume (`yelp_reviews`) was defined as the primary storage container for the raw JSON files. By utilizing Volumes instead of the legacy DBFS FileStore, the project adopts the modern Databricks standard for managing non-tabular data, providing a secure and high-performance landing zone that is decoupled from the compute resources.

4.3 Dataset Partitioning and Chunking

To bypass browser-based upload constraints and manage memory efficiently, the large-scale Yelp datasets were divided into smaller, manageable chunks. This pre-processing step ensured a reliable transfer of the 8M+ record dataset into the cloud environment without risking session timeouts or upload failures, effectively preparing the data for parallel processing.

4.4 Schema-Validated PySpark Ingestion

Rather than relying on schema inference, which can be compute-intensive and prone to errors, data was read using PySpark with pre-defined, explicit schemas. This "Schema-on-Read" approach guarantees that data types (such as timestamps and nested arrays) are correctly interpreted from the moment of ingestion, ensuring structural integrity before the data reaches the Silver layer.

Chapter 5

Data Preprocessing: The Medallion Architecture

To ensure scalability, data quality and analytical reliability, this project adopted the Medallion architecture as the foundation of its data preprocessing pipeline. This approach structures data processing into progressive layers: Bronze, Silver, and Gold. Each of these layers is responsible for a specific level of refinement. Raw Yelp dataset files are first ingested into the Bronze layer in their original JSON format,enabling efficient large-scale ingestion. Subsequent preprocessing steps in the Silver layer focus on data cleansing. Finally, the Gold layer produces analytics-ready datasets through joining the datasets based on the shared IDs. This layered design improves data governance, simplifies pipeline maintenance, and supports incremental data processing in a distributed Spark environment.

5.1 Business Dataset Preprocessing

Due to its nested structure and heterogeneous field types, the business dataset required systematic preprocessing before it can be used for analytical queries or downstream joins with review data.

5.1.1 Bronze Layer

The focus of this layer is high-fidelity data capture from the source.

- Spark's schema inference was utilized to capture the complex, nested structure of some fields.

```
business_schema = StructType([
StructField("business_id", StringType(), True),
StructField("name", StringType(), True),
StructField("address", StringType(), True),
StructField("city", StringType(), True),
StructField("state", StringType(), True),
StructField("postal_code", StringType(), True),
StructField("latitude", DoubleType(), True),
StructField("longitude", DoubleType(), True),
StructField("stars", DoubleType(), True),
StructField("review_count", IntegerType(), True),
StructField("is_open", IntegerType(), True),
StructField("attributes", MapType(StringType(), StringType()), True),
StructField("categories", StringType(), True),
StructField("hours", MapType(StringType(), StringType()), True)
])
```

Figure 5.1: Business data schema

- The data is persisted as a Delta table to enable efficient columnar storage and initial partitioning.

```
business_bronze = spark.read \
.schema(business_schema) \
.json("/Volumes/workspace/default/yelp-reviews/yelp_academic_dataset_business.json")

business_bronze.write \
.format("delta") \
.mode("overwrite") \
.option("mergeSchema", "true") \
.saveAsTable("bronzebusiness")
```

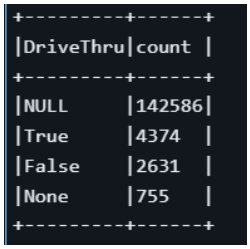
Figure 5.2: Business bronze layer delta table

5.1.2 Silver Layer

In the Silver layer, the data is refined, cleaned, and standardized to improve data quality.

- A check of the presence of null values to maintain high data integrity
- Data quality checks are performed to identify and remove duplicate business IDs, ensuring each entity is unique.
- The JSON-stylecolumns are flattened into individual columns to allow their preprocessing.
  - **Attributes Feature**: This is the most complex nested object. It contains a variable set of key-value pairs representing business features.
    - *Payment & Accessibility*: BusinessAcceptsCreditCards, BusinessAcceptsBitcoin, WheelchairAccessible, ByAppointmentOnly.
    - *Dining Specifics*: RestaurantsTakeOut, RestaurantsDelivery, RestaurantsReservations, RestaurantsTableService, RestaurantsPriceRange2, RestaurantsAttire, RestaurantsGoodForGroups .
    - *Atmosphere & Amenities*: HasTV, OutdoorSeating, WiFi, Alcohol, NoiseLevel, Caters, HappyHour, DogsAllowed, DriveThru, GoodForKids .

∗ *Nested Dictionaries*: BusinessParking (garage, street, validated, lot, valet), Ambience (romantic, intimate, classy, hipster, divey, touristy, trendy, upscale, casual), and GoodForMeal (dessert, latenight, lunch, dinner, brunch, breakfast) .



DriveThru	count
NULL	142586
True	4374
False	2631
None	755

Figure 5.3: Drive thru attribute counts

– **Hours feature**: Maps days of the week to operating intervals (e.g., "9:0-18:0") .

8	tUFrWirKiKi_TAnsVWINQQ	Monday	8:0-22:0	8:0	22:0
9	tUFrWirKiKi_TAnsVWINQQ	Tuesday	8:0-22:0	8:0	22:0
10	tUFrWirKiKi_TAnsVWINQQ	Wednesday	8:0-22:0	8:0	22:0
11	tUFrWirKiKi_TAnsVWINQQ	Thursday	8:0-22:0	8:0	22:0
12	tUFrWirKiKi_TAnsVWINQQ	Friday	8:0-23:0	8:0	23:0
13	tUFrWirKiKi_TAnsVWINQQ	Saturday	8:0-23:0	8:0	23:0
14	tUFrWirKiKi_TAnsVWINQQ	Sunday	8:0-22:0	8:0	22:0

Figure 5.4: Example of business working hours

– **Categories feature**: A nested list of tags.

Top categories:

category	count
Restaurants	52268
Food	27781
Shopping	24395
Home Services	14356
Beauty & Spas	14292
Nightlife	12281
Health & Medical	11890
Local Services	11198
Bars	11065
Automotive	10773
Event Planning & Services	9895
Sandwiches	8366
American (Traditional)	8139
Active Life	7687
Pizza	7093
Coffee & Tea	6703
Fast Food	6472

Figure 5.5: The top categories

- The identified quality issues are the missing values in categories and in hours

```
Quality issues summary:
+------------------+-----+
|issue             |count|
+------------------+-----+
|missing_hours     |23120|
|missing_categories|103  |
+------------------+-----+
```

Figure 5.6: Business dataset quality issues

- The silver layer results was also saved as a Delta table

5.2 Review Dataset Preprocessing

The review dataset represents the most critical component of the Yelp Open Dataset, as it contains the unstructured textual content used for sentiment analysis and predictive modeling. Due to the large volume of records and the presence of free-text fields, careful preprocessing is required to ensure both data quality and analytical usability.

5.2.1 Bronze layer

- A strict schema was applied during ingestion to ensure data types are consistent across millions of records.

```python
review_schema = StructType([
    StructField("review_id", StringType(), True),
    StructField("user_id", StringType(), True),
    StructField("business_id", StringType(), True),
    StructField("stars", DoubleType(), True),
    StructField("useful", IntegerType(), True),
    StructField("funny", IntegerType(), True),
    StructField("cool", IntegerType(), True),
    StructField("text", StringType(), True),
    StructField("date", StringType(), True)
])
```

Figure 5.7: Review dataset schema

- An _ingest_ts (timestamp) column was added to track when the data was processed and the dataset was saved as a Delta table

5.2.2 Silver layer

- A null-check was performed on key columns to identify gaps in the data

- To maintain data integrity, reviews containing identical text posted by the same user on the same date for the same business were excluded from the dataset.

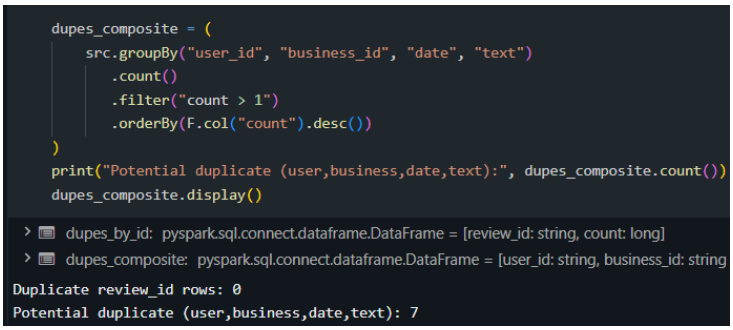


Figure 5.8: Duplicate reviews

- The useful, funny and cool counts were standardized using the absolute value function to handle data entry anomalies or negative values.

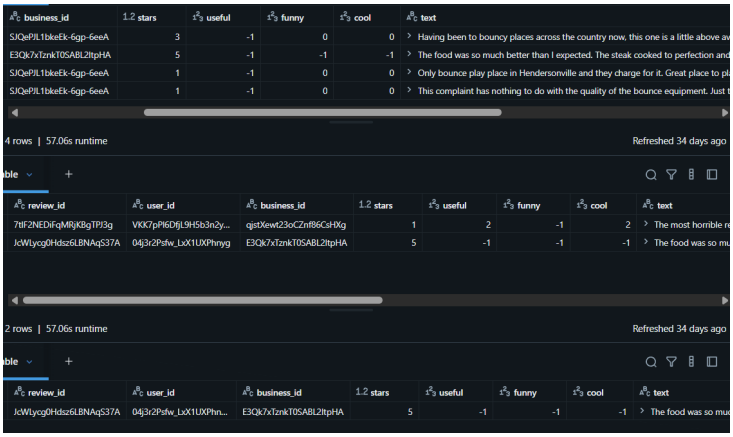


Figure 5.9: Review dataset's numerical attributes before preprocessing

```
src = (
    src
    .withColumn("useful", F.abs(F.col("useful")))
    .withColumn("funny",  F.abs(F.col("funny")))
    .withColumn("cool",   F.abs(F.col("cool")))
)

print("Before:", src.count(), "After:", src.count())
src.select(
    F.min("useful"), F.min("funny"), F.min("cool")
).show()
```

```
src: pyspark.sql.connect.dataframe.DataFrame = [review_id: string, user_id: strin
fore: 6990271 After: 6990271
+----------+----------+---------+
in(useful)|min(funny)|min(cool)|
+----------+----------+---------+
         0|         0|        0|
+----------+----------+---------+
```

Figure 5.10: Review dataset's numerical attributes after preprocessing

- Text features were extracted by trimming leading and trailing whitespaces and calculating a text_len feature to analyze the correlation between review length and engagement metrics.

```
Jan 07, 2026 (7s)

stats = (
    src  # your reviews DataFrame/table
    .withColumn("text_trim", F.trim("text"))
    .withColumn("text_len", F.length("text_trim"))
    .select(
        F.min("text_len").alias("min_text_len"),
        F.max("text_len").alias("max_text_len"),
        F.avg("text_len").alias("avg_text_len")
    )
)

stats.show(truncate=False)
```

```
stats: pyspark.sql.connect.dataframe.DataFrame = [min_text_le
+------------+------------+-----------------+
|min_text_len|max_text_len|avg_text_len     |
+------------+------------+-----------------+
|1           |5114        |567.8276896560949|
+------------+------------+-----------------+
```

Figure 5.11: Review dataset's text preprocessing

- The processed reviews were saved in Delta table format enabled, allowing for schema evolution and ACID transactions—critical for the scalability test requirements.

5.3 User Dataset Preprocessing

The transformation of the user dataset from raw JSON to a refined Silver-layer table represents a critical stage in the data engineering lifecycle. This process ensures that unstructured strings are converted into a high-performance, queryable schema.

5.3.1 Bronze Layer: Standardized Ingestion

The Bronze layer acts as a high-fidelity landing zone for raw data ingestion. To ensure ingestion accuracy and schema consistency, a strict schema was enforced using PySpark's `StructType`. This approach prevents data corruption by guaranteeing that each attribute, ranging from `average_stars` to `yelping_since`, conforms to its expected data type at read time.

```
# Bronze Layer

user_schema = StructType([
    StructField("average_stars", DoubleType(), True),
    StructField("compliment_cool", LongType(), True),
    StructField("compliment_cute", LongType(), True),
    StructField("compliment_funny", LongType(), True),
    StructField("compliment_hot", LongType(), True),
    StructField("compliment_list", LongType(), True),
    StructField("compliment_more", LongType(), True),
    StructField("compliment_note", LongType(), True),
    StructField("compliment_photos", LongType(), True),
    StructField("compliment_plain", LongType(), True),
    StructField("compliment_profile", LongType(), True),
    StructField("compliment_writer", LongType(), True),
    StructField("cool", LongType(), True),
    StructField("elite", StringType(), True),
    StructField("fans", LongType(), True),
    StructField("friends", StringType(), True),
    StructField("name", StringType(),True),
    StructField("review_count", LongType(), True),
    StructField("useful", LongType(), True),
    StructField("user_id", StringType()),
    StructField("yelping_since", StringType())
  ])
```

Figure 5.12: Schema Enforcement

Schema Enforcement An explicit schema comprising 21 fields was defined to correctly handle nested long and string data types. This eliminated implicit type inference errors and ensured consistent parsing of semi-structured JSON records.

Temporal Metadata An ingestion timestamp column (`_ingest_ts`) was appended to each record to enable data lineage tracking and recency-based validation.

Storage Layer The ingested dataset was persisted as a Delta table (`bronzeuser`), leveraging ACID transaction guarantees and Parquet-backed storage to ensure reliability and scalable read performance.

5.3.2 Silver Layer: Data Refinement and Normalization

The transition from the Bronze to the Silver layer involved extensive data cleaning, normalization, and feature engineering to prepare the dataset for analytical workloads and machine learning use cases.

```
# normalising arrays
from pyspark.sql.functions import when, split, trim, size, expr

# normalize friends: convert "None" or "" to empty array
src= src.withColumn("friends_arr",
                    when((col("friends").isNull()) | (col("friends") == "None") | (col("friends") == ""), expr("array()"))
                    .otherwise(split(col("friends"), r",\s*")))

# trim spaces and drop empty strings in friends_arr
from pyspark.sql.functions import transform, filter
src = src.withColumn("friends_arr", transform("friends_arr", lambda x: trim(x)))
src = src.withColumn("friends_arr", filter("friends_arr", lambda x: (x != "") & (x.isNotNull())) )
src = src.withColumn("friend_count", size("friends_arr"))
```

Figure 5.13: Data Refinement and Normalization

Structural Normalization

A primary challenge within the Yelp user dataset is the representation of list-like attributes (e.g., `friends` and `elite`) as comma-separated strings.

- **Array Conversion:** The `split()` and `expr()` functions were used to convert flat strings and literal `"None"` values into proper Spark array structures.
- **Data Sanitization:** `transform()` and `filter()` operations were applied to trim whitespace and remove empty or invalid entries.
- **Feature Engineering:** Derived numerical features (`friend_count` and `elite_count`) were introduced to quantify user influence without requiring runtime array computations.
- **Type Casting:** The `yelping_since` attribute was converted from string format into a high-precision `TimestampType`.

Deduplication and Data Integrity

To ensure a single, authoritative representation of each user, multiple integrity checks were applied:

```
#  drop duplicate user_id rows, keep first occurrence
src= src.dropDuplicates(["user_id"])

negatives = src.filter(
    (col("review_count") < 0) |
    (col("fans") < 0) |
    (col("useful") < 0) |
    (col("funny") < 0) |
    (col("cool") < 0)
).limit(5)
neg_count = negatives.count()

invalid_stars = src.filter((col("average_stars") < 0) | (col("average_stars") > 5)).limit(5)
```

Figure 5.14: Deduplication and Data Integrity

- **Entity Resolution:** Duplicate records were eliminated using `dropDuplicates()` on the `user_id` primary key.
- **Logical Validation:** Records containing illogical negative values in engagement metrics (e.g., `review_count`, `fans`, `useful`) were filtered out.

5.4 Gold Layer: Analytical Data Products

The Gold layer represents the final stage of the Medallion architecture, where curated Silver datasets are transformed into query-ready tables optimized for business intelligence and analytics.

5.4.1 Master Review Denormalization Strategy

While the Silver layer maintains normalized structures to minimize redundancy, analytical dashboards benefit from denormalized schemas. A unified *Master Review* table was therefore constructed.

Join Strategy The review dataset (fact table) was left-joined with the business and user datasets (dimension tables). This enriched each review with contextual attributes such as business location and user engagement history.

Selective Projection Only high-value analytical attributes (e.g., city, categories, user review count, and account age) were retained. Low-impact metadata was excluded to reduce table size and improve query latency in Databricks SQL Dashboards.

5.4.2 Technical Execution and Optimization

The denormalization process was executed using PySpark's DataFrame API to efficiently process over 8 million review records.

- **Aliasing and Namespace Safety:** Dataset aliases were used to prevent column name collisions and preserve distinct primary keys (`business_id`, `user_id`).
- **Delta Lake Persistence:** The final dataset was saved as `gold_review_master` in Delta format, ensuring schema enforcement, transactional guarantees, and optimized Parquet-based storage for downstream BI consumption.

Chapter 6

Machine Learning: sentiment analysis

The primary objective of this component is to automate the processing of millions of Yelp reviews by predicting the sentiment associated with user-generated text. By classifying reviews into specific categories, businesses can identify critical service failures or exceptional performance without manual reading. Within our pipeline, the text attribute serves as the independent variable (input), while the stars column is transformed into a label for a multiclass classification task. This allows the system to generate a quantitative score for businesses in real-time as new reviews are ingested via the streaming pipeline.
While advanced Natural Language Processing (NLP) frameworks like SparkNLP or pretrained transformer models such as BERT and GloVe 100 offer high accuracy, they proved incompatible with the Databricks Free Version (Community Edition). These models require significant driver memory and often necessitate the installation of custom JAR files and heavy Python dependencies that exceed the resource quotas of the free tier. Furthermore, the "serverless" nature of the free tier limits persistent caching and memory management, making the standard Spark CrossValidator extremely unstable. Consequently, we opted for the native PySpark MLlib library, which is optimized for distributed processing on Spark clusters and provides a reliable foundation for feature engineering and classification at scale.
To identify the most effective classifier, we implemented and compared two distinct algorithms: Logistic Regression and Random Forest. We chose Logistic Regression for its computational efficiency and its proven ability to handle high-dimensional sparse data typical of text processing (Bag-of-Words). In parallel, we utilized a Random Forest classifier to capture potential non-linear relationships between word frequencies. Both models shared a common preprocessing pipeline consisting of:

- StringIndexer: converting the star ratings into numerical labels.
- Tokenizer: breaking sentences into individual words and remove punctuation.
- StopWordsRemover: filtering out common words (e.g., "the", "a", "is") that carry little semantic value.
- HashingTF & IDF: converting text into a numerical feature vector using Term Frequency-Inverse Document Frequency (TF-IDF) logic.

During the experimental phase, Logistic Regression consistently outperformed Random Forest in terms of both execution time and accuracy. Specifically, the Logistic Regression model achieved a test accuracy of 65%, whereas the Random Forest model only reached

46.59%.
However, due to the mentioned memory limitations of the Databricks Community Edition, we could not use the automated CrossValidator for hyperparameter optimization, as the repeated shuffling and lack of caching caused cluster failures. Instead, we implemented a Manual Tuning Strategy. We split the data into three subsets—70% Training, 15% Validation, and 15% Testing—and manually iterated through a custom ParamGrid. This allowed us to observe the model's behavior while adjusting the following parameters:

- maxIter: To ensure the model reached convergence without overfitting.
- regParam (Regularization): To prevent overfitting by penalizing large coefficients.
- elasticNetParam: To balance between L1 and L2 regularization. By tracking the MulticlassClassificationEvaluator metrics on the validation set for each iteration, we were able to isolate the "Best Model" configuration and subsequently evaluate its performance on the final unseen test set, ensuring a robust and reproducible result despite the platform's hardware constraints.

The computational overhead required for the fine-tuning process exceeded available hardware capabilities, culminating in a runtime exception and system termination before the search algorithm could converge on the global optimum.

Chapter 7

Real-time Streaming

The entry point of the streaming architecture utilizes Databricks Auto Loader to ingest raw JSON data from the cloud volume. Traditional file-based streaming often struggles with the listing bottlenecks when dealing with millions of records; however, Auto Loader bypasses this by using a scalable directory listing or file notification service. In the implementation, we configured the stream to use .option("cloudFiles.format", "json") and enabled .option("cloudFiles.inferColumnTypes", "true"). This setup allows the pipeline to automatically detect and adapt to schema changes in the raw Yelp reviews without manual intervention, ensuring that the Bronze layer remains a high-fidelity, complete mirror of the incoming raw data.

To manage the complexity of a multi-stage streaming pipeline, we implemented Delta Live Tables (DLT). This declarative framework allowed us to define our data flow using Python decorators like @dlt.table, which simplifies the underlying Spark logic into a manageable DAG (Directed Acyclic Graph). DLT manages the task orchestration, cluster sizing, and error handling automatically. In the DLT pipeline, the bronze_yelp_raw table serves as the foundation, streaming data directly into the Silver layer. By using DLT, we moved toward a triggered execution model that ensures our tables are always up-to-date with the latest review data.

The transition from Bronze to Silver is where the raw stream is refined into a clean, queryable state. We utilized DLT's expectations to enforce data quality, specifically using `@dlt.expect_or_drop` to ensure that any record lacking a review_id is filtered out before reaching the analytics stage. Technically, this subsection of the pipeline performs critical type-casting which ensures that downstream machine learning models and dashboards receive standardized, high-quality data, fulfilling the project's Data Cleaning evaluation criteria.

A critical requirement for any Big Data system is the ability to recover from failures without data loss. We implemented manual Structured Streaming queries with a defined checkpointLocation to track the "offsets" of the data being processed. By storing these offsets in a Delta Lake directory, the system can resume exactly where it left off in the event of a cluster restart. Additionally, we utilized the .trigger(availableNow=True) configuration. This allowed us to process all available data in the stream as a single micro-batch, providing the scalability of streaming with the cost-efficiency of batch processing.

The final stage of the streaming pipeline is the Gold layer, which provides real-time business value. Using `dlt.read_stream`, we created an aggregated table, gold_business_stats, which performs running calculations of average stars and the number of reviews grouped by business_id. Because this is a streaming aggregation, Spark maintains the interme-

diate state, allowing the dashboard to reflect the most current business ratings as soon as a new review is ingested. This layer transforms the high-volume stream into a low-volume, high-value dataset ready for consumption by dashboards, directly answering the core business challenge of tracking reputation at scale.

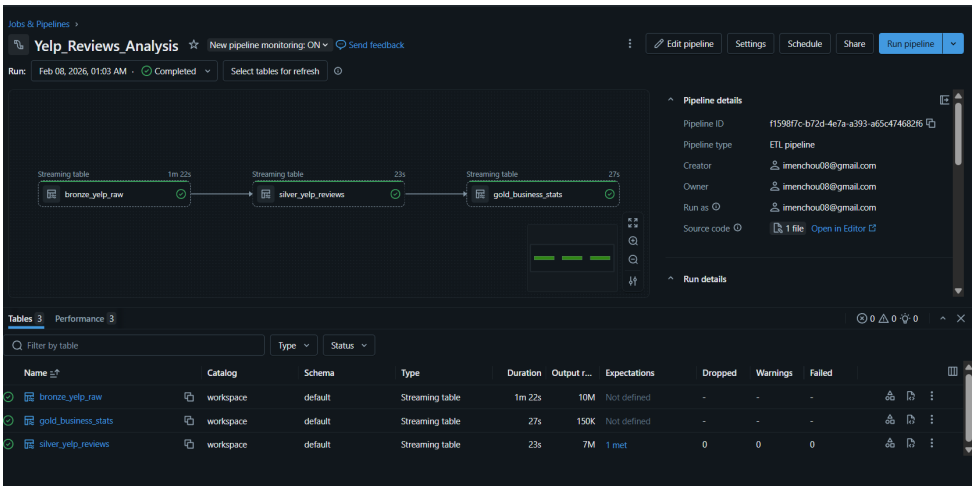


Figure 7.1: Real-time streaming pipeline

Chapter 8

Visualization and Performance Assessment

8.1 Dashboards

The analytical insights are presented through three specialized Databricks SQL Dashboards, each targeting a specific entity within the ecosystem. By utilizing the delta lake tables, these dashboards provide near-instantaneous query responses.

8.1.1 Business Overview Dashboard

This dashboard provides a macro-level view of the market landscape, focusing on geographic distribution and category dominance.

Key KPIs: Tracks 119.7K active businesses across the dataset with a global average

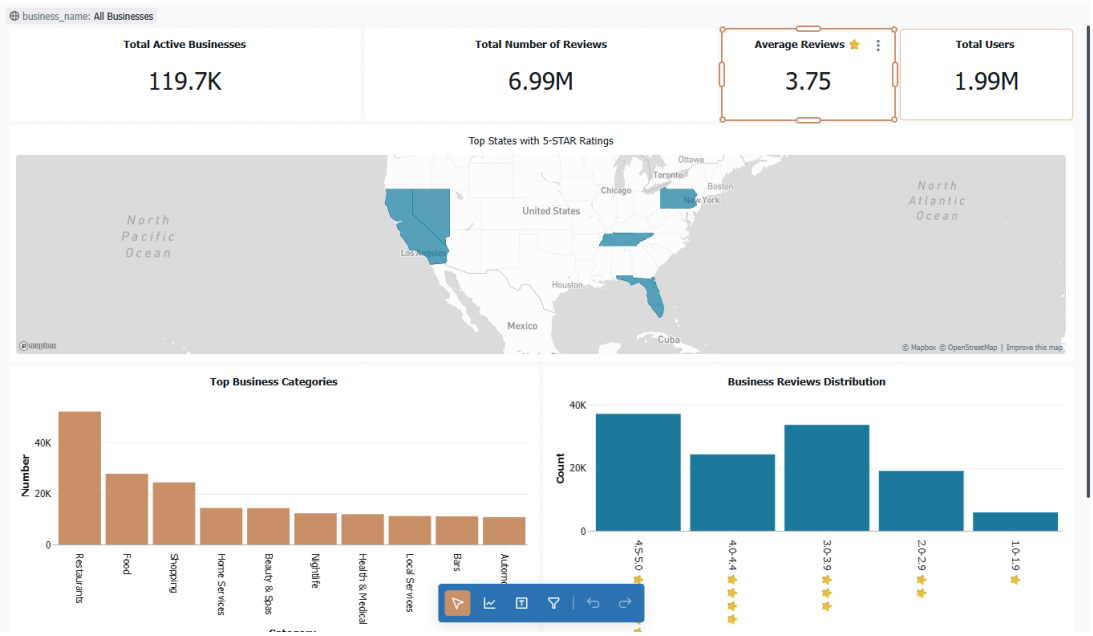


Figure 8.1: Business Overview Dashboard

rating of 3.75.

Market Analysis: High-performing states (e.g., California, Florida, Pennsylvania) are

identified via a geospatial map of 5-star ratings.
Industry Trends: The "Top Business Categories" bar chart reveals that Restaurants and Food services represent the highest volume of businesses, guiding potential market entry strategies.

8.1.2 Reviews Dashboard

This dashboard is designed for granular performance tracking. It features a Global Filter (e.g., for “Trader Joe’s”) that allows for both aggregate market analysis and individual business deep-dives.

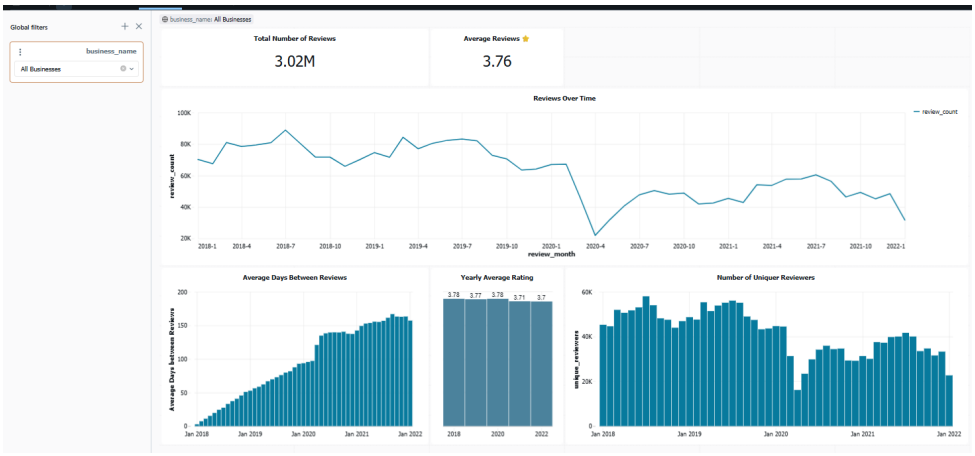


Figure 8.2: Reviews Dashboard

Temporal Trends The “Reviews Over Time” line chart tracks engagement volatility, helping businesses identify seasonal peaks or specific periods of service decline.

Customer Loyalty Metrics like “Average Days Between Reviews” provide a proxy for customer retention and visit frequency.

Rating Drift The “Yearly Average Rating” bar chart visualizes long-term quality trends, allowing stakeholders to see if recent operational changes have positively impacted customer satisfaction.

Filter Application : Case for Trader Joe’s

8.1.3 User Engagement Dashboard

This dashboard segments the 1.99M users to identify the influencers driving platform content.

User Segmentation A donut chart classifies the community into Casual, Regular, and Power Users (100+ reviews), revealing that while casual users are the majority, power users drive a disproportionate amount of content.

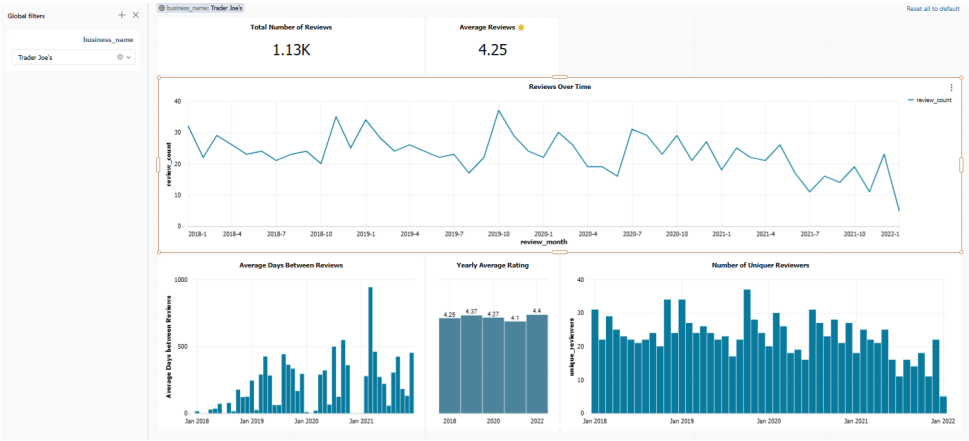


Figure 8.3: Trader Joe's Dashboard

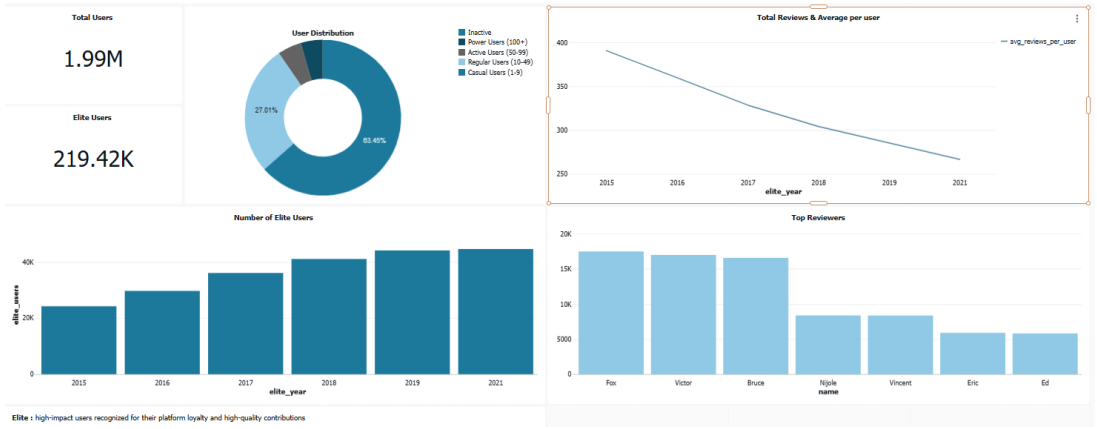


Figure 8.4: User Engagement

Elite Insights Tracks the growth of the Elite User population over time. Despite the total number of Elite users increasing, the "Average Reviews per User" line indicates a downward trend in individual output from this group since 2015.

Leaderboards Identifies "Top Reviewers" by name (e.g., Fox, Victor, Bruce) to highlight the platform's most significant individual contributors.

By implementing native Databricks SQL Dashboards on top of our Delta Lake Layer, we successfully bridged the gap between complex big data engineering and actionable business intelligence. The inclusion of global parameters allows non-technical stakeholders to filter 6.99 million reviews down to a single business in real-time.

Chapter 9

Conclusion

This project successfully established a production-ready data engineering foundation using Apache Spark and Delta Lake to process millions of records into a reliable Silver-layer "Single Version of Truth." By integrating automated sentiment intelligence via PySpark MLlib, the pipeline transforms raw text into quantitative scores through a TF-IDF and Logistic Regression framework. Despite the hardware constraints of the Databricks Community Edition, a strategic approach to manual hyperparameter tuning and a structured 70/15/15 data split ensured robust model validation without exceeding memory quotas. This technical rigor, combined with high-value feature engineering, such as tracking Elite user influence and monthly rating drift, culminated in interactive SQL Dashboards that bridge the gap between complex data processing and executive decision-making for over 119,000 businesses. Looking forward, the architecture is positioned for expansion into real-time structured streaming, advanced NLP migration using transformer models, and predictive analytics to forecast business churn by correlating sentiment drift with historical closure patterns.

To further evolve this platform, future iterations could include migrating the existing logic to a multi-node production cluster to test performance gains on even larger datasets (e.g., 50M+ records).