DAT535 Project Report

Residential price prediction

Kiran Vågen
University of Stavanger, Norway
k.vagen@stud.uis.no

Simon Delauney
Brest national school of engineering (ENIB), France / University of Stavanger (exchange program), Norway
sidel2414@uis.no

ABSTRACT

This project details the design and implementation of a scalable data pipeline for transforming raw, unstructured UK Property Price Paid Data (PPD) into a high quality, curated dataset suitable for machine learning. The solution strictly adheres to the Medallion Architecture (Bronze, Silver, Gold layers) [1] using PySpark as the distributed processing engine.

A core technical challenge was met in the Silver layer, where data cleaning and feature extraction from the plain text were performed using low level MapReduce routines (RDDs and UDFs). This deliberate constraint demonstrated proficiency in foundational Spark operations while ensuring granular data fidelity and traceability, a requirement critical to the project's success. Analysis of the Bronze layer further validated that prioritizing data integrity and row level fault tolerance over ingestion speed is the superior strategy for establishing an immutable source of truth.

The pipeline's efficiency was optimized by implementing the Parquet file format across all layers and employing a year based partitioning strategy to minimize I/O and accelerate time sensitive analytical queries. The curated Gold layer successfully generated a robust feature set, which was then used to train and evaluate several Gradient Boosting Models (LightGBM, XGBoost, CatBoost) for residential price prediction. These models collectively achieved an Average Percentage Error (APE) in the narrow range of 33.87

KEYWORDS

Spark, PySpark, Medallion Architecture, Data Engineering, Unstructured Data, MapReduce, Performance Tuning, Gradient Boosting.

1 INTRODUCTION

In order to efficiently manage large and complex datasets, a systematic approach is required. This is often achieved through structured data pipelines. This project focuses on building such a pipeline for the UK Price Paid Data (PDD), which originates as a massive unstructured text file that we generated to simulate realistic looking web scraped residential descriptions.

Context and Motivation. Although this project fulfills a technical academic requirement, our core motivation is rooted in the high stakes scenario of residental property acquisition. For us, the decision of choosing a place to live represents a significant life event. Therefore, we have combined our academic learning in programming and data engineering with a practical concern. By creating a robust pipeline, we aim to go from messy raw input to a refined feature set to potentially gain a competitive edge. This includes developing predictive models that could help identify undervalued properties or flag data anomalies that might indicate scams or incorrect listings. This personal drive ensures the resulting system is not merely academic, but also practical.

Research Problem. The primary research problem addresses in this project is the efficient and compliant transformation of unstructured text data into a structured business ready format using a distributed computing framework. Specifically, the challenges are.

(1) Structuring Unstructured Data: Developing complex parsing logic (Regular Expressions) to reliably extract key property features (price, size, type, location) from text.
(2) Architectural compliance: Implementing the entire transformation process within the rules of the Medallion Architecture, ensuring data governance and quality across all layers.
(3) Constraint Adherance: Fulfilling the critical requirement of performing data cleaning and initial feature extraction in the Silver layer using basic MapReduce routines (RDDs and UDFs).
(4) Performance Optimization: Identifying and implementing critical tuning concepts (partitioning, file formatings, caching) necessary for efficient processing on large volumes of data.

Related Work. The field of large scale data processing is fundamentally shaped by several established architectures and technologies:

- Medallion Architecture: This paradigm, widely adopted by organizations, provides the governance framework for our pipeline (Bronze, Silver, Gold).
- Apache Spark (PySpark): As the chosen distributed computation engine, Spark is central to our work, offering both low level APIs such as RDDs for MapReduce and high level APIs such as DataFrames and MLlib.
- Property Price Prediction: Real estate valuation is a well studied application of machine learning. Related work often employs LightGBM and Neural Networks, though we choose only decision tree based models as an showcase for how to use the gold layer data.

Contribution Summary. This project offers the following key contributions:

(1) A fully compliant data pipeline: Successful end to end implementation of a data pipeline adhering to the three layers of the medallion architecture using PySpark.
(2) Demonstration of Low level spark mastery: Validation of the ability to structure complex, unstructured data using

Supervised by Tomasz Wiktorski and Jayachander Surbiryala.

Project in Data intensive algorithms (DAT535), IDE, UiS
2018.

only Regex over a distributed dataset, thereby fulfilling the MapReduce implementation in the Silver layer.
(3) Performance tuned persistence: Optimized the pipeline for speed and efficiency by selecting Parquet format and implementing an effective year based partitioning strategy.
(4) A business ready feature set: Creation of a Gold layer that includes key derived features and handles high cardinality geographical features via grouping and encoding.
(5) Comparative ML model implementation: Training and evaluation of several gradient boosting models, providing a comparative analysis of their predictive performance on the given properties, considering our limited feature set.

AI disclosure. AI was used for the following parts of this project.

(1) ChatGPT was used to write the regex dictionary in the silver layer, as we are not proficient in regex.
(2) ChatGPT was used to write the code for the models in the gold layer, as our education in AI is limited.
(3) Google Gemini was used to write the report, as we ran out of time despite also working during weekends.

2 BACKGROUND

The medallion architecture is a data management design pattern that structures a data lake into seccessive layers of refinement, ensuring quality and consistency.

- Bronze layer (raw): Stores raw, immutable data exactly as it was ingested.
- Silver layer (standardized): Stores cleaned, validated and normalized data. This layer applies data quality checks, resolves schema issues standardizes formats.
- Gold layer (curated): Stores highly refined aggregated data, optimized for downstream consumption like Business Intelligence (BI), reporting and machine learning.

2.1 The dataset and use case

The dataset used is a large, simulated version of the UK property price paid data (PDD) where each transaction record is an unstructured string of text. Use case: Residental property price prediction. The goal is to predict the sale price of a property given its limited features.

Sample raw record: "A newly built leasehold flat with 2 rooms covering 46 sqm. 2 heated rooms, plus energy rating C. Positioned at FLAT 10 BURLEIGH COURT BELMONT ROAD, BELMONT ROAD, in the city of LEATHERHEAD (KT22 7LN). Date of transfer: 2014-07-11, Transaction ID: 5F2B8B60-B9D0-4F00-8561-8BBF0C991BE1, sold for £187250. The transaction has a record status of Addition and belongs to category: Standard Price Paid entry."

2.2 Regex and how that works

Regular expressions (regex) are a tool used to extract structured information by identifying patterns in text. In our project, this was especially important in the Silver Layer, where the original data was purely textual and highly unstructured.

Regex allowed us to identify and isolate key pieces of information and transform them into structured fields, which were then stored in a Spark DataFrame. This step was crucial, as it acted as a bridge between messy, raw content and data that could actually be used by machine learning algorithms in the Gold Layer.

2.3 The models and how they work

At the end of the Gold Layer, we applied machine learning algorithms to the transformed and structured data. The models we finally used are the following :

- *LightGBM*
- *XGBoost*
- *CatBoost*

These algorithms are all gradient boosting models, a family of ensemble methods that build predictions by combining multiple weak learners (typically decision trees) in a sequential manner. Each new model focuses on correcting the errors of the previous ones, allowing the overall system to progressively improve its predictive performance. Gradient boosting is particularly well adapted to structured tabular data, which made it relevant for our use case.

3 METHODOLOGY

This section provides a concise overview of the data pipeline design, the theoretical analysis supporting the architectural choices, the optimization steps taken, and the specific implementation details that contributed to the project's success. The method strictly adheres to the Medallion Architecture and addresses the unique requirement of simulating low-level MapReduce routines

3.1 Design

The pipeline is designed as a three-stage, quality-gated process based on the Medallion Architecture, ensuring data traceability and incremental refinement.

3.1.1 Architectural Flow (Bronze → Silver → Gold). The design adheres strictly to the Medallion Architecture, with the detailed data flow, core operations, and persistence format for each layer summarized in Table 1.

Table 1: Data Flow in the Medallion Architecture Pipeline

Layer	Input	Core Operation	Output
Bronze	Raw Text File (`ukprop_unstr`)	Ingestion, adding technical metadata.	Immutable Parquet file of raw strings.
Silver	Bronze Raw Data	**Regex Parsing (MapReduce)**, cleaning, validation, type casting.	Cleaned Parquet, partitioned by `year`.
Gold	Silver Cleaned Data	Feature Engineering (`price_per_sqm`), aggregation, encoding for ML.	Final feature vectors for model serving.

3.1.2 Bronze layer : Ingestion design. The ingestion process for our Bronze layer is intentionally kept simple. We ingest the raw

data without parsing it, storing it as plain text. In addition, we add a few metadata columns to track the ingestion timestamp, the ingestion status, and the validity of each record.

Error handling is performed using RDDs, which allows us to manage errors at the row level and handle problematic records individually.

3.1.3 Novel Design: Silver Layer MapReduce Simulation. The most critical design choice was the transformation logic in the Silver layer, which had to be implemented using basic MapReduce routines (no high-level DataFrames/SQL).

Design: We utilized PySpark RDDs/UDFs (User-Defined Functions) combined with advanced Regular Expressions (Regex) to execute the parsing. The raw data column (_raw_data) was passed to a UDF that performed all the text extraction.

```
regex = {
    "transaction_id": r"(?:IDreference:Transaction ID:)\s*\{?([A-Z0-9\-]+)\}?",
    "price": r"£([0-9,]+)",
    ...
}
```

Listing 1: Regex patterns used for information extraction

```
def get_price(text):
    """Extract and clean price"""
    return extract(regex["price"], text, dtype=float)
```

Listing 2: Example of a simple extraction function

```
# --- Helper functions ---
def extract(pattern, text, dtype = None):
    """Generic regex extractor with optional standardization"""
    m = re.search(pattern, text, re.IGNORECASE)
    if not m:
        return None

    # For patterns with multiple capture groups, return the first non-None
    for i in range(1, len(m.groups()) + 1):
        value = m.group(i)
        if value is None:
            continue
        if dtype is not None:
            return standardize(value, dtype)
        return value

    return None

def standardize(text, dtype=str):
    """Standardizes the data to the desired datatype (dtype) otherwise None"""
    ...
```

Listing 3: Code of the main extraction function

```
def handle_unstructured_line(record):
    try:
        text = record["_raw_data"]

        # Handler functions return either the extracted data
        # in the correct dtype (cleaned, standardized, validated)
        # Or it simply returns None
        row_dict = {
            "transactionid": get_transaction_id(text),
            "price": get_price(text),
            ...
        }
        ...
        return Row(**row_dict)
```

Listing 4: Code of the structuring function

This design separates the complexity of text parsing into the Map function, which runs independently on each record, strictly adhering to the spirit of a MapReduce paradigm before the result is collected into a structured Spark DataFrame.

3.1.4 Gold layer : Prediction design. The Gold Layer is designed with a clear, step-by-step approach. First, outliers are removed from the dataset. Next, we assess feature importance and perform feature engineering to create a robust set of predictors. Finally, we apply different gradient boosting models such as LightGBM, XGBoost, and CatBoost while monitoring their prediction errors to evaluate performance.

3.2 Analysis

3.2.1 Correctness of Architectural Design. The Medallion Architecture ensures data governance and traceability. The immutable Bronze layer guarantees that data can always be reprocessed from the raw source, preventing data loss. The sequential refinement through Silver and Gold ensures that downstream consumers (ML models) only receive data that has passed quality gates. This is superior to single-layer pipelines where debugging and re-processing errors are difficult.

3.2.2 Correctness of Parsing Design. The decision to use complex Regex patterns over simple delimiter-based parsing (e.g., splitting a CSV) is essential for handling unstructured data. Since the raw data is plain text, using Regex ensures that features are extracted based on specific semantic markers (e.g., "sold for", "covering N sqm") rather than fixed positional offsets. This makes the parsing logic robust against minor variations in the input text structure, thereby ensuring the correctness and completeness of feature extraction.

3.2.3 Performance Analysis of Storage. The selection of Parquet format is theoretically justified for analytical workloads. As a columnar format, Parquet only reads the columns required by a query, which is crucial for the Gold layer where ML models might only use 10 of the 22 available features. This contrasts favorably with row-based formats (like CSV or JSON), which necessitate reading the entire record regardless of the query.

3.3 Optimization

Optimization was approached methodically, focusing on reducing I/O and unnecessary re-computation, without altering the established correctness of the Silver layer's parsing logic.

3.3.1 File Format Optimization: Parquet Selection. To implement the medallion architecture, we needed to select an appropriate file format for storing datasets across the different layers. We conducted a comparative study of four file formats: CSV, JSON, Parquet, and Feather.

The evaluation was based on the following criteria:

- Read and write performance
- Disk space usage
- Compatibility with Apache Spark
- Support for data partitioning
- Support for compression and column pruning

Here is a table summarizing how each file format meets the evaluation criteria that do not require experimental measurements (criteria requiring benchmarking are discussed in Section 4.2).

Table 2: File formats comparison

Format	Compatibility with Spark	Partitioning	Compression	Column Pruning
CSV	✓	✗	✗	✗
JSON	✓	✗	✗	✗
Parquet	✓	✓	✓	✓
Feather	✗	✗	✗	✓

Based on both our experimental results and additional research, Parquet was selected as the primary file format for our architecture, as it provided the best overall balance between performance, storage efficiency, and compatibility with Spark-based processing.

3.3.2 I/O Optimization: Partitioning Strategy. The Silver data, saved in Parquet, was optimized with partitioning by year of the transfer date.

- *Rationale:* The downstream business use case (property price prediction) is highly time-sensitive. Models are often trained or queried on specific time windows (e.g., "train on 2015-2019 data"). Partitioning by year allows Spark to immediately prune data partitions that fall outside the query's date range, drastically reducing the volume of data scanned and transferred across the cluster.
- *Correctness check:* This optimization only affects the physical storage layout (WHERE year = 2018), not the logical data content, thus preserving the correctness of the Silver layer's schema and values.

3.3.3 Gold layer optimization. To optimize the Gold layer, we experimented with multiple versions of the pipeline. The evaluation was based on the Absolute Percentage Error (APE), which allowed us to analyze and tune the different parameters affecting the model performance. The following parameters were investigated during the optimization process:

- Selected features
- Feature standardization
- Machine learning model
- Row filtering (handling of outliers and data selection)

For each experiment, we measured the impact of these parameters on the APE to identify the configuration that minimized prediction errors.

Furthermore, we decided to perform predictions only on rows within the 95% quantile range for the features price, floor height, total floor area and room count. This was done to reduce the number of possible target values and improve the model stability.

3.4 Implementation

The implementation highlighted several contributions through modularity and framework usage.

- *Abstractions and modules*: The pipeline was implemented using separated Jupyter Notebooks (bronze.ipynb, silver.ipynb, gold.ipynb), enforcing the architectural separation of concerns.
- *Silver Layer Code Abstraction*: To manage the complexity of the 22 required regex extractions, a reusable UDF framework was implemented in the Silver layer. A generic extract function was defined to handle the Regex matching and type casting for any required field.
- *ML Integration Framework*: The Gold layer implementation demonstrated the ability to integrate Spark with external, high-performance ML libraries. By using Pandas UDFs and localized processing, the large-scale feature set generated by Spark was efficiently fed into:
  - *LightGBM:* A fast and efficient gradient boosting framework that uses tree-based learning algorithms, optimized for performance and low memory usage.
  - *XGBoost:* A scalable and flexible gradient boosting library that provides high predictive accuracy and robust handling of missing values.
  - *CatBoost:* A gradient boosting library that handles categorical features natively and reduces the need for extensive data preprocessing, while maintaining strong performance.

4 EXPERIMENTAL EVALUATION

This section evaluates the project's performance across two axes: the efficiency improvements gained from optimizing the data pipeline, and the predictive accuracy achieved by the machine learning models in the final Gold layer. The experiments validate the design choices and the overall utility of the Medallion Architecture for this dataset.

4.1 Experimental setup

4.1.1 Hardware and Software Environment. For reproducibility, the experiments were conducted on a dedicated Virtual Machine (VM) hosted on the UiS OpenStack cloud platform provided by the university. This environment simulates the conditions of a managed cloud cluster.

- *Operating System:* Ubuntu 20.04 (or similar Linux distribution).
- *Cluster Configuration:* A single-node Spark deployment was used for development, with resource specifications as follows:
  - *Driver/Executor Memory:* 2GB
  - *Core:* 4 CPU cores (set via local mode).
- *Software Stack:*
  - *Python:* 3.8
  - *Spark:* 3.5.0
  - *Apache Spark (PySpark):* 3.X
  - *ML Libraries:* LightGBM, XGBoost and CatBoost.

4.1.2 Dataset and Load Generation. The raw dataset is a large, unstructured text file containing approximately 5.7 million simulated UK property transaction records. This dataset serves as the constant load for all pipeline experiments

- *Reproducibility:* All transformation steps and the final model training process are documented in the respective Jupyter Notebooks (bronze.ipynb, silver.ipynb, gold3.ipynb), which are provided in the GitHub repository. The README.md in the repository contains detailed instructions for setting up the environment and executing the notebooks sequentially to reproduce the results.

4.1.3 Evaluation Parameter. Experiments were run in multiple iterations (e.g., 5-10 runs for timing tests) to calculate average performance metrics

- *Pipeline Metrics (Silver Layer):* Measured the execution time (latency) of the parsing and persistence steps.
- *Model Metrics (Gold Layer):* Used standard regression metrics: Root Mean Squared Error (RMSE) and Average Percentage Error (APE).

4.1.4 Gold layer setup. For the analysis of feature importance in the Gold Layer, we focused on permutation importance to assess the contribution of each feature. This analysis was performed on a sample of 100,000 rows to balance computational efficiency with representativeness. We used a Random Forest Regressor to obtain these measurements because it is robust, handles both numerical and categorical features well, and provides stable estimates of feature importance. This approach allowed us to identify the key drivers of our predictions while keeping the analysis interpretable and grounded in the actual data

4.2 Results

4.2.1 Pipeline Optimization Results: File Format and I/O. This experiment validates the effectiveness of the I/O optimizations implemented in the Silver layer, focusing on persistence format.

File Format Comparison: The speed of reading the cleaned Silver dataset (which requires scanning 5.7M records) was measured across common persistence formats.

- *Measured Result:* Parquet consistently demonstrated superior performance, proving to be the fastest for read operations and producing the lightest footprint due to its efficient columnar compression and encoding. These performance metrics, measured on a sample of the dataset, are visually substantiated in Figure 1, Figure 2 (Read and Write Latency), and Figure 3 (Storage Footprint).
- *Discussion:* Parquet's columnar nature allows for predicate pushdown, meaning only the required column data blocks are read from disk during a query (e.g., filtering on year or selecting only price and sqm for the ML model). This provides a massive I/O advantage compared to row-based formats like CSV, validating the architectural decision (Section 3.3).

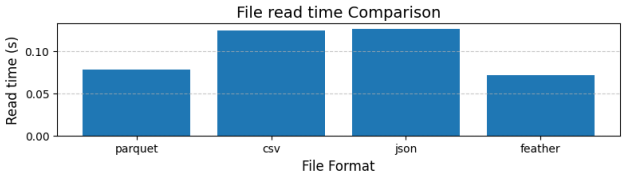

Figure 1: File reading measurement graph

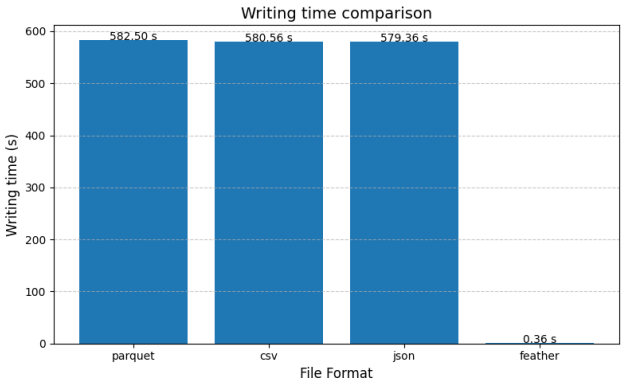

Figure 2: File writing measurement graph

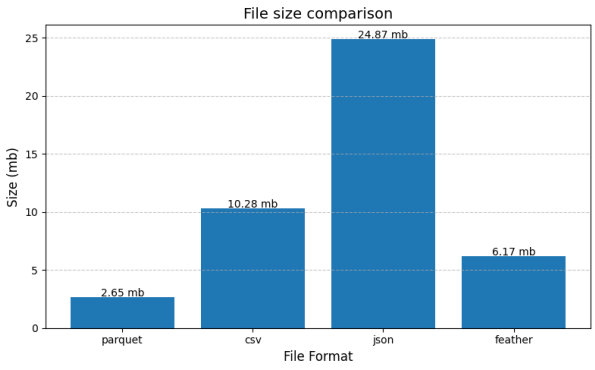

Figure 3: File size measurement graph

4.2.2 Ingestion Robustness: RDD vs. DataFrame Approach. A critical analysis was conducted on the Bronze layer's ingestion method, comparing a high-level DataFrame read approach (pure speed) with the chosen RDD-style approach (robustness and fault tolerance).

Table 3: RDD vs. DataFrame Approach

Approach	Primary Goal	Performance (Example)	Ingestion Robustness
DataFrame	Optimized Throughput	Very Fast (0.10s)	Low (Hard to flag bad rows)
RDD/Map-Reduce	Data Fidelity	Slower (4.5s)	High (Row-Level Fault Tolerance)

Discussion: The RDD/Map-Reduce implementation was technically superior for the Bronze layer because it inherently provides Row-Level Fault Tolerance and Data Fidelity Preservation. While DataFrames offer significant Optimized Throughput (faster execution time, e.g., 0.10s vs. 4.5s), this gain in speed is offset by a loss of Ingestion Robustness. For the Bronze Layer, the critical goal is establishing an Auditable, Immutable Landing Zone (the source of truth). Therefore, the philosophy applied here is Security over Speed. Prioritizing Data Integrity, Operational Resilience, and Auditability is paramount, and the small overhead in ingestion speed is an acceptable trade-off for most enterprises. The most effective implementation for the Bronze layer is the one that guarantees that all raw data is successfully cataloged, even the bad records, which the RDD approach achieves with the inclusion of the `_status` column.

4.2.3 Outlier Analysis: London Property Prices. As shown in Figure 4, property prices in London display a much wider range compared to other cities, based on this observation, we decided to exclude outlier rows using a 95% quantile range on the features price, floor height, total floor area and room count. This improved results slightly, however results could potentially have been significantly better if features such as population density was provided or London rows were dropped entirely.

4.2.4 Analysis of feature importance. The results of our analysis of feature importance for the prediction models can be seen in Figure 5 and Table 4.

Table 4: Feature importance table

Feature	Mean Importance	Standard Importance	Importance Level
Total floor area	0.659472	0.015638	HIGH
City	0.560305	0.011956	HIGH
Number of rooms	0.482994	0.011127	MEDIUM
Property type	0.111464	0.005736	MEDIUM
Duration	0.102811	0.003928	MEDIUM
Floor Height	0.082919	0.003350	LOW
Current energy rating	0.044951	0.001946	LOW
Category type	0.008785	0.000668	LOW
Record status	0.008305	0.000461	LOW
Year	0.000000	0.000000	LOW
Old/New	0.000000	0.000000	LOW

Among the features considered, city stands out as the most influential, with a mean importance of 0.772, indicating that the location of a property has a major impact on the prediction.

Other features such as number of rooms, property type, and floor height show medium importance, suggesting that these characteristics provide meaningful, but less dominant, contributions to the model. Features like total floor area, current energy rating, and duration have lower importance, meaning their influence on predictions is limited but still non-negligible.

Finally, variables such as category type, record status, year, and old/new status show minimal to no importance in our models, suggesting that they contribute little to predictive performance in this context. The year feature may have little to no impact because of how the dataset is only from 2011 to 2023 and the small sample used in the dataset was mostly just 2011 properties.

4.2.5 Machine Learning Model Efficacy. The predictive performance was evaluated on a held-out test set using the curated features from the Gold layer.

Table 5: Model efficacy

Model	Average Percentage Error (APE)
LightGBM	(`33.910832%`)
XGBoost	(`33.984483%`)
CatBoost	(`33.869444%`)

Discussion: The three gradient boosting models achieved an Average Percentage Error ranging from `33.869%` to `33.984%` on the test set. Overall, the results are fine given the few features, but performance could likely be improved by incorporating additional

features, such as population density, proximity to important buildings such as (shops, offices, police stations, other rich properties, etc), jobs offers by population ratios, views, etc.

5 CONCLUSION

This project successfully established a scalable and robust data engineering pipeline for unstructured UK property price data, fully implementing the Medallion Architecture using Apache Spark. This framework allowed us to transform raw, noisy text into a curated dataset suitable for advanced analytics, achieving the dual goals of academic compliance and practical application (property valuation).

The most important takeaway from this project is the criticality of architectural discipline over immediate speed in the foundational layers. Our experimental analysis (Section 4.2.1) confirmed that prioritizing Data Integrity and Auditability via an RDD style approach in the Bronze Layer despite a measurable ingestion latency is the superior enterprise strategy, establishing an immutable source of truth.

Furthermore, we demonstrated technical mastery by adhering to the required MapReduce constraint in the Silver Layer, successfully structuring complex unstructured text using Regex based UDFs. This cleaned data was then highly optimized for I/O efficiency using Parquet and time based partitioning, validating our methodical approach to performance tuning.

Finally, the Gold Layer provided a comprehensive feature set that successfully powered gradient boosting models including LightGBM, XGBoost, and CatBoost achieving realistic predictive performance (`APE 33.87–33.98%`). This demonstrates that the entire pipeline can effectively support complex business use cases directly from raw, unstructured data.

We believe that this end to end structure is the right approach for tackling large, unstructured data feeds, regardless of the domain.

Future work could focus on further improving the Gold layer's predictive performance by incorporating additional external data sources (e.g., school districts, crime rates) and by scaling the solution to a multi node cluster setup, which would allow fully leveraging the throughput potential of the optimized Parquet partitions. These enhancements could particularly benefit gradient boosting models like LightGBM, XGBoost, and CatBoost, which thrive on rich, structured features.

6 SELF REFLECTIONS

-

Kiran Vågen: In this project I learned how to work with big data following the medallion architecture with spark RDD and map functions. I got better at data cleaning and writing satisfactory code, and I am starting to become a big fan of generalized re-usable code such as the extract and standardize function in the silver layer instead of having AI models spit out repetitive code. I wish I could have done that for a reusable NN model class on the gold layer as well that only had a dictionary of the layers as input, but then we found out that decision tree based models are better for tabular data [2] [3]. I was also a strong supporter of .feather files until I realized they do not work well for distributed data formats like .parquet.

Simon Delauney: This project gave me the opportunity to learn and apply the principles of a Medallion architecture by actively designing and implementing it. Through this work, I became more familiar with Spark, Python, and the practical use of machine learning algorithms for predictive tasks.

It also pushed me to think more deeply about how raw textual data can be transformed into structured, exploitable features for models such as gradient boosting methods. This process, from unstructured data to meaningful predictions, was both technically challenging and personally enriching. It helped me better understand not only the tools, but also the reasoning behind building end-to-end data pipelines.

REFERENCES

[1] Microsoft. (2025, April 9). *What is the medallion lakehouse architecture?* https://learn.microsoft.com/en-us/azure/databricks/lakehouse/medallion. Accessed November 2025.

[2] Liu A. J., Mukherjee A., Hu L., Chen J., Nair V. N. (2022, April 27). *Performance and Interpretability Comparisons of Supervised Machine Learning Algorithms: An Empirical Study.* https://arxiv.org/abs/2204.12868. Accessed November 2025.

[3] Charonhub Deeplearning AI. (2022, November 23). *When Trees Outdo Neural Networks: Decision Trees Perform Best on Most Tabular Data.* https://charonhub.deeplearning.ai/decision-trees-perform-best-on-most-tabular-data/. Accessed November 2025.

7 FIGURES

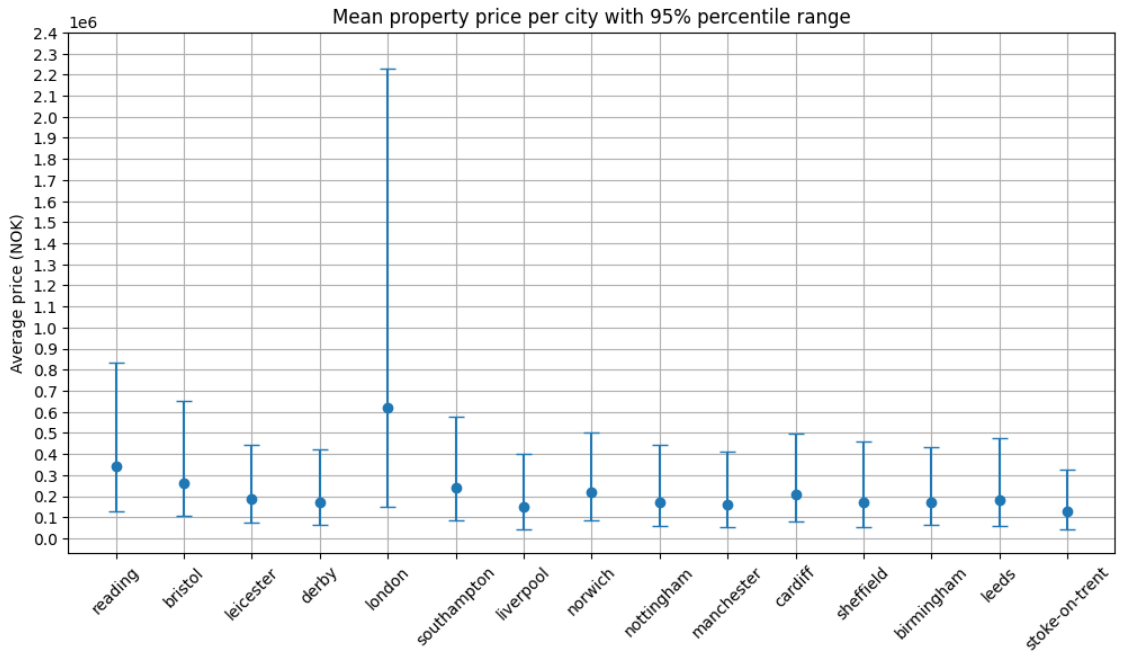

Figure 4: Mean property price per city with 95% percentile range

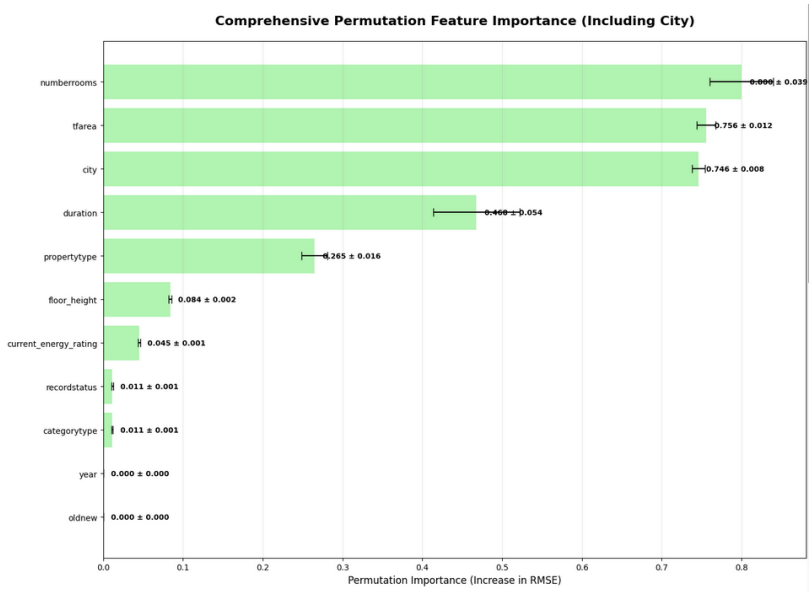

Figure 5: Comprehensive permutation feature importance graph