A UNIFIED FIELD MODEL (UFM) FOR INTEGRATING SILOED DATASETS IN THE NIGER DELTA OIL AND GAS INDUSTRY: A DATA PROCESSING PIPELINE FOR THE PETROLEUM ENGINEERING RESEARCH DATASETS (PERD)

BY

OLOGBO EMMANUEL CHUKWUNEKU

ENG2006442

DEPARTMENT OF PETROLEUM ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF BENIN

BENIN CITY

NOVEMBER 2025

A UNIFIED FIELD MODEL (UFM) FOR INTEGRATING SILOED DATASETS IN THE NIGER DELTA OIL AND GAS INDUSTRY: A DATA PROCESSING PIPELINE FOR THE PETROLEUM ENGINEERING RESEARCH DATASETS (PERD)

OLOGBO EMMANUEL CHUKWUNEKU
ENG2006442

A PROJECT SUBMITTED TO THE
DEPARTMENT OF PETROLEUM ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE
AWARD OF BACHELOR OF ENGINEERING (B.ENG)
DEGREE IN
PETROLEUM ENGINEERING

DEPARTMENT OF PETROLEUM ENGINEERING
FACULTY OF ENGINEERING
UNIVERSITY OF BENIN
BENIN CITY

NOVEMBER 2025

CERTIFICATION

This is to certify that this project was carried out by OLOGBO EMMANUEL CHUKWUNEKU of the Department of Petroleum Engineering with matriculation number ENG2006442 in partial fulfillment of the requirements for the Award of the Degree, Bachelor of Engineering (B.ENG)

PROF. KELANI O. BELLO — **DATE**
(PROJECT SUPERVISOR)

DR. O. A. TAIWO — **DATE**
(PROJECT COORDINATOR)

ENGR. DR. IKPONMWOSA OHENHEN — **DATE**
(HEAD OF DEPARTMENT)

PROF. KEVIN CHINWUBA IGWILO — **DATE**
(EXTERNAL SUPERVISOR)

DEDICATION

This thesis is dedicated to God Almighty, who made it possible for me to complete the study successfully. This work is dedicated to my father, mother, siblings, alongside my professors and lecturers who have taught me that the best kind of knowledge to have is that which is learned for its own sake and have been a major source of motivation in this academic journey.

ACKNOWLEDGEMENT

Firstly, to Jehovah God Almighty who sustains life, I would like to express my utmost gratitude for granting me the opportunity, strength, wisdom, and grace to complete my project successfully. Without His divine guidance and favor, this accomplishment would not have been possible.

My sincere appreciation goes to my supervisors, Professor Kelani Bello (PhD, MEng, BEng), Dr. Seun Taiwo (PhD, MEng, BEng), and especially Engr. Ame Enosolease (MEng, BEng), for their unwavering support and commitment to the completion of this project. Their mentorship, professional feedback, and guidance made this work possible, and I am truly grateful for the sleepless nights and weekend meetings they dedicated to this process.

Special thanks to every member of my family, the Ologbo family, for being my backbone, my support system, and for sponsoring my education. They have always provided a safe space for me throughout my academic journey, catering to every one of my academic and emotional needs.

I would also like to thank all my lecturers for training me academically and for helping to develop my character during these years in the Department of Petroleum Engineering. In no particular order, I would like to acknowledge Prof. Ogbeide Paul, Prof. Oduwa David, Dr. Sunny, Dr. Wale, Engr. Alonge, and Engr. Emmanuel for their support and contributions toward my education and character development.

I am deeply grateful to my mentors, Engr. Ezekiel Agberen, Engr. Gabriel Oyakhirome, Engr. Prince Abangwu, Engr. Austine Erinomo, Engr. Flo Chinyere, and others, for their invaluable support and guidance on my academic and professional journey. Their mentorship has been essential in shaping my path.

I would also like to acknowledge my friends Caleb and Great-Nelson for being true friends and brothers to me through both the good and the challenging times. Your friendship has been a source of strength.

I extend my gratitude to all my coursemates with whom I have shared this academic journey from 100 level till the end. It has been a pleasure learning and growing together with all of you. Especially my buddies, Damilare, Sele and Moses.

Finally, I acknowledge myself, Chukwuneku Emmanuel Ologbo, for showing up every day, even when it was hard, for putting in the work, for not bowing to pressure, for perseverance, discipline, and patience throughout this academic journey.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The oil and gas industry in the Niger Delta faces a significant challenge with heterogeneous datasets fragmented across multiple platforms and stored in diverse file formats, a situation that impedes efficient research and operational optimization. To address this problem, this study develops a Unified Field Model (UFM) designed to aggregate, harmonize, and standardize these varied petroleum engineering datasets, including well logs, seismic data, and production logs, using scalable cloud storage and data processing pipelines. The core of the research involves creating a durable and adaptable data structure capable of handling both structured and unstructured data while preserving relational attributes. This process is supported by rigorous data quality assurance techniques, such as feature engineering, anomaly detection, and petroleum engineering domain-specific imputation strategies. This UFM then serves as the foundation for a web-based data brokerage platform, known as Petroleum Engineering Research Datasets (PERD), which enables researchers and industry operators in the Niger Delta to efficiently access high-quality petroleum datasets. This study provides a foundational improvement for the sector, enhancing operational efficiency, improving data interoperability, and allowing for the better use of computational tools to tackle complex Petroleum engineering challenges, thereby improving study reproducibility and performance in the region.

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background to the Study

The oil and gas industry, particularly in the Niger Delta region, has long struggled with the integration of diverse datasets, often leading to inefficiencies in both research and operational decision-making. The petroleum engineering field generates massive volumes of heterogeneous data, ranging from seismic readings and well logs to production logs and sensor data. Unfortunately, these data types often exist in disparate formats, including structured datasets such as CSV and Excel files, semi-structured data like LAS and SEG-Y files, and unstructured data like PDFs and free text reports.

This fragmentation of data has become a significant barrier to the effective analysis and decision-making that is vital for the efficient operation of the industry. As datasets come from multiple sources, with varied structures and standards, integrating them for meaningful analysis becomes an arduous task. Traditional database systems are ill-suited to handle such diverse and complex data types, often leading to inconsistent results and prolonged decision-making cycles.

The relevance of this study lies in addressing these integration challenges. A unified data model capable of homogenizing disparate datasets is essential for enhancing data accessibility and promoting data-driven decision-making. A Unified Field Model (UFM), which integrates and standardizes these diverse datasets, could transform how petroleum engineering research is conducted, leading to more efficient workflows, improved reservoir management, and enhanced operational strategies.

This research aims to develop a robust, scalable platform that can aggregate, harmonize, and deliver high-quality petroleum engineering datasets. By focusing on the local context of the Niger Delta, it will address both the technical and operational challenges unique to this region, providing a framework for the broader oil and gas industry.

1.2 Problem Statement

Effective petroleum engineering research and the successful deployment of AI and ML models are fundamentally constrained by the lack of reliable, accessible, and integrated datasets. Researchers and engineers spend a significant portion of their time on data cleaning, preparation, and integration rather than analysis, investing substantial effort to resolve inconsistencies and anomalies. This data bottleneck is increased by the diverse, non-standardized, and often proprietary formats used across the industry.

This problem is particularly evident in the Niger Delta region. While global data repositories like Volve exist, their content is generic and fails to capture the complex and specific geological formations unique to the Niger Delta. Furthermore, researchers in Nigeria face significant operational barriers in accessing commercially and nationally sensitive petroleum data, which is governed by strict corporate and national policies.

Finally, traditional database technologies, such as relational database management systems (RDBMS), are ill-equipped to handle the complex, hierarchical, and semi-structured nature of petroleum data. Their rigid schemas and inability to natively store unstructured data make them unsuitable for the agile and flexible data models required by modern analytics.

Therefore, a critical gap exists for a specialized data solution tailored to the Niger Delta. This solution must be capable of harmonizing heterogeneous data, ensuring high quality, and providing a scalable, secure, and accessible platform for researchers to advance data-driven studies specific to the region's unique geology.

1.3 Aim and Objectives

The aim of this research is to design a scalable, web-based data brokerage platform (PERD) that aggregates, harmonizes, and delivers high-quality heterogeneous petroleum datasets to researchers, specifically tailored for the local context of the Niger Delta region.

Project Objectives:

- To design a Unified Field Model (UFM) that is robust, flexible, and scalable with a data structure capable of harmonizing heterogeneous petroleum datasets for efficient storage while maintaining relational attributes.
- To architect and provision an appropriate cloud storage solution to store, serve, and query UFM data quickly in an online environment.
- To ensure data quality by performing feature engineering and relationship analysis on a pilot dataset, thereby informing the design of the automated data cleaning and validation pipeline.
- To investigate suitable data imputation strategies for a pilot dataset leveraging Petroleum Engineering (PE) domain knowledge.
- To establish a comprehensive and standardized set of columns for the pilot dataset by researching all essential and expected columns in petroleum production datasets, enabling accurate data mapping and the development of column-specific cleaning rules.

1.4 Significance of the Study

This study provides a practical solution to the persistent data interoperability challenges in the petroleum industry. By developing and validating a Unified Field Model (UFM), this research offers a pathway to unlock siloed data, directly contributing to the improvement of data management for petroleum engineering research.

This work is significant for several key stakeholders:

- **For Researchers:** It provides a model for accessing harmonized, high-quality, and reliable datasets, which enables more efficient data exchange, supports reproducible research, and accelerates the development of new models.
- **For the Nigerian Oil and Gas Sector:** The UFM serves as a critical framework and reference for future digital transformation initiatives. It directly addresses the data access and quality barriers that are specific to Nigeria and the Niger Delta region.
- **For AI and ML Adoption:** The UFM acts as a foundational layer for integrating advanced analytics. By delivering clean, standardized, and analysis-ready data, it enables the practical application of AI and machine learning tools to solve complex petroleum engineering problems.

1.5 Scope of the Study

The scope of this project covers the design, development, and validation of the Unified Field Model (UFM) and its supporting data processing pipeline. The research encompasses:

1. **Model Design:** The design of a robust UFM utilizing JSON as the core data structure to handle heterogeneous and hierarchical petroleum data.
2. **Architecture:** The implementation of a scalable, cloud-native architecture using Google Cloud Firestore as the NoSQL document database, supported by Cloud Run and Pub/Sub for automated data ingestion pipelines.
3. **Data Processing:** The development of Python scripts to process, clean, transform, and standardize pilot datasets. This includes specific procedures for handling .grdecl (grid) files and .xlsx (production spreadsheet) files.
4. **Data Quality and Validation:** The application of feature engineering, Exploratory Data Analysis (EDA), and domain-specific imputation strategies using a pilot dataset from the Volve field (referred to as the Saturn Field dataset).

5. **Anonymization:** The development and testing of a data anonymization protocol using UUIDs to protect sensitive information while maintaining data structure.

The study focuses on establishing the proof-of-concept for the data pipeline and architecture. It does not extend to the development of final AI/ML models, but rather prepares the data foundation that these models require.

1.6 Limitations

While this research successfully validates the UFM concept, it is subject to the following limitations:

- **Dataset Source:** The validation was performed using the publicly available Volve field dataset. Although this dataset is comprehensive, it originates from the North Sea and does not fully represent the unique geological complexities and specific data challenges characteristic of the target Niger Delta region.
- **File Format Scope:** The proof-of-concept primarily focused on harmonizing structured spreadsheet files (`.xlsx`, `.csv`) and semi-structured grid files (`.grdecl`). Other critical and complex industry formats, such as `.las` (well logs) and `.segy` (seismic data), were not included in the scope of this project and are recommended for future work.
- **Scalability Testing:** The cloud architecture performance tests for concurrent uploads were validated with a maximum of 10 devices. While the results were positive, this limited test phase does not fully simulate the potential load of a large-scale, industry-wide deployment with thousands of data sources.

CHAPTER TWO

2.0 LITERATURE REVIEW

Modern advances in petroleum engineering encompass a dynamic landscape shaped by digital transformation, the integration of artificial intelligence (AI) and machine learning (ML), industrial internet of things (IIoT), digital twin technology, advanced robotics, cloud and edge computing, blockchain, and sophisticated imaging systems. Recent reports project that, in 2025 and beyond, trends such as generative AI for operational optimization, emissions reduction, predictive maintenance, smart exploration using IoT sensors, and asset integrity monitoring are revolutionizing the field, improving safety, efficiency, and sustainability objectives for operators worldwide (AlphaSense, 2025; CapTech Consulting, 2024; AVEVA, 2024; StartUs Insights, 2025).

These innovations in reservoir management, simulation, and predictive modeling, especially when ML is applied, depend fundamentally on access to high-quality, high-volume datasets. The success of ML and AI initiatives is anchored in data that is accurate, reliable, and representative. Poorly curated data inevitably results in inaccurate or biased predictions, undermining trust in outcomes. Literature affirms that robust data enhances the performance, fairness, and robustness of analytic models, while the lack of data integration can introduce error, bias, and misinterpretation (Kotwel, 2023; Adata.pro, 2020; Prometeia, 2020; Ayadata.ai, 2020).

Effective research within petroleum engineering is constrained without reliable datasets. Studies in several fields, including petroleum, emphasize that model training, theoretical development, and complex simulation are limited unless quality, completeness, and consistency of data are guaranteed. Challenges encountered in the oil and gas sector—such as missing values, data inconsistency, and extensive cleaning needs—delay research workflows and data-driven decision-making. As noted by authorities in the domain, “high-quality data is

essential to the development and refinement of AI applications because it directly affects the accuracy, reliability, and fairness of model outputs" (Adata.pro, 2020; Mehdi Mohammadpoor, 2018; Anomalo, 2020).

Researchers have long faced hurdles in accessing high-quality petroleum data due to sensitivity, ownership, and heterogeneity of data sources. Drilling records, reservoir performance datasets, and production logs are typically considered sensitive commercial or national assets. These data often exist in proprietary formats and require careful management, governed by strict policies to safeguard intellectual property and national security interests. In Nigeria, researchers confront significant operational barriers stemming from rigorous data governance and corporate acquisition procedures imposed to protect the nation's oil and gas resources. Literature also highlights that security, privacy, and format compatibility are consistent challenges for big data usage in oil and gas, compounding access issues for the research community (Petroleum Agency SA, 2024; FileCloud, 2023; Gong et al., 2018).

Furthermore, researchers spend significant time on data cleaning, preparation, and integration. Before analysis can commence, substantial effort is invested in resolving inconsistencies—such as those arising from data being routed through multiple agencies or reported in disparate formats—thereby slowing innovation. This struggle is intensified in the petroleum sector, where formats range from structured CSV/Excel files to unstructured PDFs and well logs, and where datasets may be incomplete or carry anomalies (Adata.pro, 2020; JPT/Society of Petroleum Engineers, 2019; Kumari et al., 2023; Gong et al., 2018).

While global dataset repositories like UNISIM, The Well, and Volve offer valuable resources for petroleum research, their content is often generic and does not cover the complex and specific formations characteristic of regions such as the Niger Delta. The unique geology and

stratigraphy of the Delta create additional hurdles, reinforcing the case for regional or bespoke data solutions.

The complex hierarchy, structure, and evolving parameters present in petroleum datasets expose the limitations of traditional relational database management systems (RDBMS). Industry practice reveals the prevalence of formats including .csv, .xls, .dlis (well logs), .las, .segy (seismic), .pdf, and shapefile (GIS)—each designed for specific contexts and rarely interoperable out-of-the-box. The heterogeneity and lack of standardization complicate integration and analysis. Relational databases, with their rigid schemas and focus on structured, tabular data, struggle to manage this diversity, posing well-documented challenges in handling unstructured and hierarchical datasets, scaling horizontally, and maintaining schema flexibility (Milvus, 2025; Atlan, 2023; Gong et al., 2018).

A robust solution must accommodate both structured and unstructured data, maintaining relations among disparate input files, and supporting the inclusion of new fields without major schema modifications. Examples of unstructured data prevalent in oil and gas include scanned technical reports, handwritten field notes, and equipment status updates stored as free text or image files. The sensitivity of these datasets is underpinned by their commercial value, national security implications, and competitive advantage for operators, making security and access control a top priority (Milvus, 2025; Kumari et al., 2023).

Relational databases are often unsuitable as they do not naturally support unstructured data, are not compact when dealing with large, varied file types, and have rigid schemas which hinder agile model development and integration of emerging data fields. Binary Large Object (Blob) storage, while effective for raw data, lacks native update mechanisms and does not support relational querying or direct relational mapping needed for analytic workflows. XML databases can store unstructured data, but data representation is verbose and they are not compact for

high-volume industrial workflows. Moreover, XML lacks out-of-the-box relational support, compared to modern document databases (Milvus, 2025; Atlan, 2023; FileCloud, 2023).

Graph databases provide support for unstructured data and enable modeling of relationships through nodes and edges, satisfying requirements for direct updateability and relational support, but can be less compact—a result of their storage model—which is noted in major industry use cases (Gong et al., 2018; PLoS ONE, 2018).

JSON, on the other hand, offers a solution that aligns with the industry's needs. It is well-suited for representing semi-structured and unstructured data, is lightweight, supports schema flexibility, offers natural relational mappings (via nested documents), and is human-readable for easy debugging and maintenance. JSON files can directly model hierarchical petroleum data—such as wells, reservoirs, and time-series logs—in a compact representation. The widespread use of JSON in oil and gas for sharing drilling metrics, production updates, and operational data indicates its acceptance within modern digital workflows. Adaptation of JSON also streamlines integration with analytic platforms and cloud storage solutions, enabling scalable, secure remote access—a critical industry priority in the era of distributed and digitalized assets (JSON, 2025; Gong et al., 2018; PLoS ONE, 2018).

Cloud adoption has become standard practice in petroleum engineering, meeting needs for scalability, rapid deployment, and remote access—with security, data encryption, and compliance supported by best-in-class providers. Google's Firestore is one such database-as-a-service designed for cloud-native operations, combining schema flexibility, nested document support, security, and efficient querying. Firestore is natively designed to store data in JSON-like formats, supporting structured, semi-structured, and unstructured data, and facilitating integration with cloud analytics pipelines (Firestore, 2025; Firebase, 2025).

CHAPTER THREE

3.0 METHODOLOGY

3.1 Design of the Unified Field Model (UFM)

The first step in this methodology involves the design of a robust, flexible, and scalable Unified Field Model (UFM). The UFM is aimed at harmonizing heterogeneous datasets in the petroleum industry, ensuring efficient storage while maintaining relational attributes.

Data Sources and Compatibility Evaluation:

- Petroleum datasets exist in various formats like *.grdecl*, *.egrid*, *.csv*, *.xls*, *.pdf*, etc., and each dataset requires an understanding of its unique structure and metadata.
- The challenge lies in creating a model that can handle such diversity and support relational mappings between the different types of data.
- A conceptual model is developed using JSON to represent key entities, their attributes, and their relationships. This model incorporates Entity-Relationship (ER) mapping and ontology-based schemas.

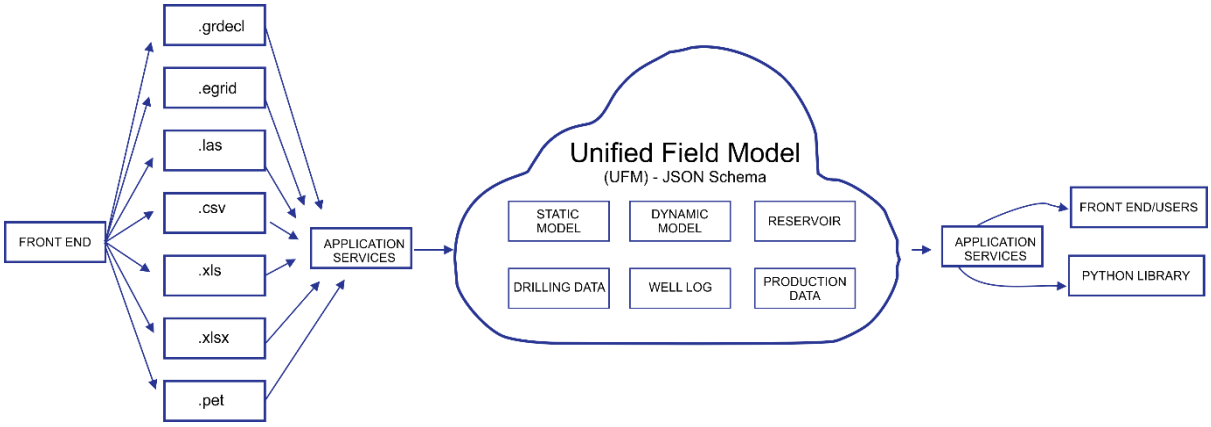

Figure 3-1: UFM Structure

Choosing JSON for the Unified Data Structure:

- Why JSON?: JSON is chosen because it supports flexibility, schema evolution, and efficient storage of complex and nested data structures common in petroleum datasets. Its ability to represent hierarchical data and relationships (e.g., wells within reservoirs, production logs within wells) is crucial for the model's success.
- JSON also eliminates the need for complex joins, typically required in SQL databases, which are inefficient for handling semi-structured datasets.

Characteristic	JSON (Document)	Graph (DB)	XML (Document)	SQL Table (RDBMS)	Blob (Binary Data)
Human Readable	Yes	No	Yes	Yes	No
Unstructured Data Support	Yes	Yes	Yes	No	Yes
Direct Updatability	Yes	Yes	Yes	Yes	No
Compact	Yes	No	No	No	Yes
Schema Flexibility/Evolution	Yes	Yes	Yes	No	Yes
Relational Support	Yes	Yes	No	Yes	No
Simple to Learn	Yes	No	Yes	No	No
Directly Usable with Firestore	Yes	No	No	No	No

Table 3-1: Comparison of the characteristics of various data storage technologies.

System Architecture and Evaluation:

- Performance tests with pilot data from the Saturn Field confirm that JSON-based architecture maintains relational connections and supports low-latency queries, validating its suitability for the UFM.

3.2 Cloud Storage Architecture for UFM Data

The next stage of the project focuses on the architecture and provisioning of an appropriate cloud storage solution that can store, serve, and query UFM data efficiently.

Key Requirements:

- The cloud storage solution must accommodate large volumes of heterogeneous petroleum data, ensuring scalability, low latency, and high availability.
- Google Cloud Firestore is chosen for its distributed, object-based storage model, which can efficiently store JSON documents. Firestore's low-latency and automatic scaling features are critical for handling concurrent queries and large datasets.

System Design:

- Firestore functions as the primary NoSQL datastore, organizing each field's data into collections of lightweight JSON documents, which optimizes query speed.
- Data transfer pipelines are automated using Cloud Run and Pub/Sub to ensure efficient, batch-upload handling.

Figure 3-2: Cloud Architecture

3.3 Data Quality Assurance and Feature Engineering

The third stage focuses on ensuring the quality of data by performing feature engineering and relationship analysis. This process utilizes a pilot dataset from the Saturn Field to evaluate data consistency, detect anomalies, and inform the design of an automated data cleaning and validation pipeline.

Data Preprocessing:

- Raw datasets are pre-processed using Python, which systematically separates numerical and non-numerical columns for targeted analysis.
- Exploratory Data Analysis (EDA) is performed to identify relationships and correlations within the dataset, visualized using tools like Pandas, Seaborn, and Matplotlib. This helps detect anomalies and understand how various data attributes interact.

Data Cleaning and Transformation:

- Numerical and categorical columns undergo specific cleaning processes, such as handling missing values, filling blank cells with the dominant value, and ensuring pressure and temperature columns maintain logical relationships.
- For non-numerical fields, string values are converted to UTF-8 encoding and standardized for consistency, such as adjusting capitalization or handling extra whitespace.

Data Formatting:

- Each column is converted into standardized, machine-readable formats like datetime, float, and geographic coordinates. This enables easier analysis, query execution, and geospatial analysis for future use.

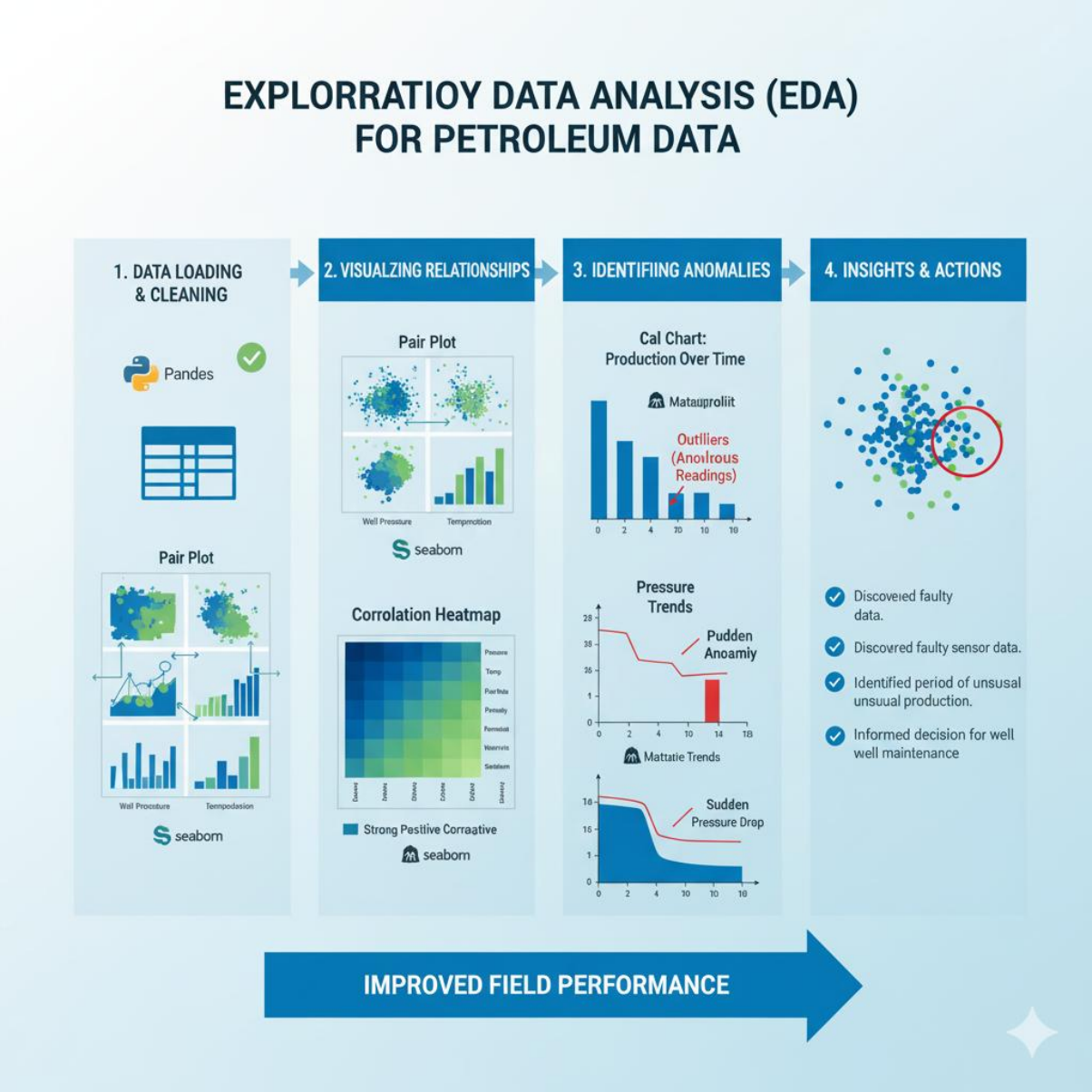

Figure 3-3: EDA process

3.4 Data Imputation Strategies Using Petroleum Engineering Knowledge

The fourth stage investigates data imputation strategies tailored to the petroleum domain. This involves leveraging domain-specific knowledge to restore missing or incomplete data while preserving the relationships and characteristics inherent in petroleum data.

Identifying Relationships and Missing Data:

- The dataset is analyzed for missing values and structural anomalies. Strong correlations between variables like Onstream Hours and Oil Produced are explored to guide imputation strategies.
- Pair plots and other visual tools help identify correlations, clusters, and outliers, which are addressed using imputation methods informed by petroleum engineering principles.

Imputation Techniques:

- A systematic approach is adopted to recover missing values based on domain knowledge. For example, missing production values are imputed based on associated metrics, such as the proportional relationship between production volumes and operating hours.
- Outliers are removed using statistical methods like the Interquartile Range (IQR), ensuring that only valid data is retained.

3.5 Column Standardization and Development of Cleaning Rules

The final stage of the methodology involves standardizing the dataset's columns and developing column-specific cleaning rules. This process ensures that the pilot dataset adheres to industry standards and improves data quality for future research and analysis.

Research and Standardization:

- An extensive review of industry-standard data models, like the PPDM schema, is conducted to identify essential columns commonly found in petroleum production datasets. These include well identifiers, production volumes, reservoir characteristics, and operational parameters.
- Columns are categorized into relevant groups such as identifiers, temporal records, and production metrics.

Data Mapping and Cleaning Rules:

- A detailed mapping process is established to ensure all identified columns are included in the dataset. Column-specific cleaning rules are implemented to ensure data consistency and integrity. For example, missing production data is filled conditionally, and spatial coordinates are standardized for geospatial analysis.
- This standardized dataset serves as the foundation for future automation, analysis, and visualization tools.

CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

4.1 Design of the Unified Field Model (UFM)

The design and implementation of the Unified Field Model (UFM) employed a JSON-based architecture, ensuring the seamless integration of various petroleum datasets. The model was particularly effective in managing complex relationships between different datasets such as wells, reservoirs, and production logs. A series of tests with pilot data demonstrated that the UFM could effectively handle heterogeneous formats (`.grdecl`, `.csv`, `.egrid`, `.pdf`) without compromising data structure or query performance.

JSON's flexibility and hierarchical structure were key to ensuring the scalability and ease of data retrieval, far surpassing the capabilities of traditional relational databases and XML in the context of this project.

Below is a preview of a pilot `.grdecl` file, which contains critical information on the 3D computational grid and the physical properties of each cell, integral for reservoir flow simulations. Key properties extracted include:

- **Porosity (PORO)**: The fraction of the rock volume that is void space, allowing for fluid storage.
- **Permeability (PERMX, PERMY, PERMZ)**: The rock's ability to transmit fluids in the X, Y, and Z directions, respectively.
- **Net-to-Gross (NTG)**: The ratio of reservoir rock to total rock volume.
- **Active/Inactive Cells (ACTNUM)**: Flags indicating which cells are part of the flow model.

The following Python script was developed to extract and store these values into JSON format:

```
 5.  5. def extract_grdecl_data(input_file_path, output_dir):
 6.  6.     input_base = os.path.splitext(os.path.basename(input_file_path))[0]
 7.  7.
 8.  8.     suffix = 1
 9.  9.     while True:
10. 10.         output_file_name = f"{input_base}{suffix:02d}.txt"
11. 11.         output_file_path = os.path.join(output_dir, output_file_name)
12. 12.         if not os.path.exists(output_file_path):
13. 13.             break
14. 14.         suffix += 1
15. 15.
16. 16.     data_dict = {
17. 17.         "Textual Data": [],
18. 18.         "Numerical Data": []
19. 19.     }
20. 20.
21. 21.     try:
22. 22.         with open(input_file_path, 'r') as file:
23. 23.             lines = file.readlines()
24. 24.
25.  25.                 filtered_lines  =  [line  for  line  in  lines  if  line.strip()  and  not
line.startswith(("#", "!"))]
26. 26.
27. 27.         for line in filtered_lines:
28. 28.             if line.strip():
29. 29.                 if any(char.isdigit() for char in line):
30. 30.                     try:
31. 31.                         values = list(map(float, line.split()))
32. 32.                         data_dict["Numerical Data"].append(values)
33. 33.                     except ValueError:
34. 34.                         data_dict["Textual Data"].append(line.strip())
35. 35.                 else:
36. 36.                     data_dict["Textual Data"].append(line.strip())
37. 37.
38. 38.         with open(output_file_path, 'w') as output_file:
39. 39.             output_file.write("Textual Data:\n")
40. 40.             for line in data_dict["Textual Data"]:
41. 41.                 output_file.write(f"{line}\n")
42. 42.
43. 43.             output_file.write("\nNumerical Data:\n")
44. 44.             for row in data_dict["Numerical Data"]:
45. 45.                 output_file.write(f"{row}\n")
46. 46.
47. 47.         print(f"Data extraction complete. Extracted data saved to: {output_file_path}")
48. 48.
49. 49.         json_output_file_name = f"{input_base}{suffix:02d}.json"
50. 50.         json_output_file_path = os.path.join(output_dir, json_output_file_name)
51. 51.
52. 52.         with open(json_output_file_path, 'w') as json_file:
53. 53.             json.dump(data_dict, json_file, indent=4)
54. 54.
55. 55.         print(f"JSON data saved to: {json_output_file_path}")
56. 56.
57. 57.     except FileNotFoundError:
58. 58.         print(f"Error: The file '{input_file_path}' was not found. Please check the file
path.")
59. 59.     except Exception as e:
60. 60.         print(f"An error occurred: {e}")
```

The image below (Figure 4-1) shows an expanded view of the Porosity values extracted from the JSON document, capturing the number of occurrences of each value.The code interpreted

and represented the number of times each porosity value is repeated in the file which typically has this structure.

$$N * Value$$

$$Where\ N\ =\ the\ number\ of\ times\ the\ value\ exists$$

$$Value$$

$$= The\ actual\ value\ of\ the\ parameter\ which\ can\ be\ porosity, permeability, etc.$$

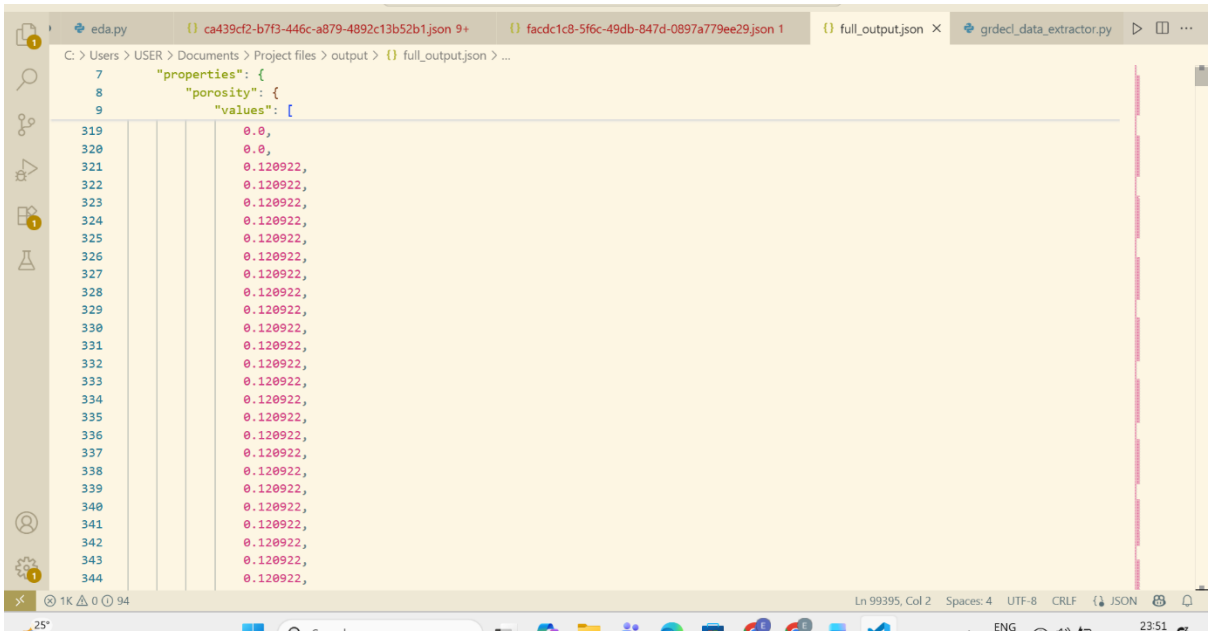

Figure 4-1: Expanded view of Porosity values captured within the JSON document

Following this, Figure 4-2 illustrates the collapsed view of the static grid file structure after conversion to the JSON document, outlining the overall structure and extracted parameters.

Figure 4-2: Collapsed view of static grid file structure after conversion to JSON document

4.2 Handling Spreadsheet Files (CSV, XLSX)

Spreadsheet formats like `.csv`, `.xlsx`, and `.xls` are integral in the petroleum industry, primarily for managing and analyzing operational data. The pilot production dataset was tested by converting it from its original Excel format to JSON. This transformation was crucial for enabling efficient querying and data integrity maintenance as new records were added. The conversion process was automated, allowing for easy updates without disturbing the structure of the underlying file.

The Python script that managed the conversion was as follows:

```
1
 2. def export_to_files(df, original_filename):
 3.     """
 4.     Exports the cleaned DataFrame to Excel and Firestore-ready JSON format.
 5.     """
 6.     if df is None:
 7.         print("No data to export due to previous errors.")
 8.         return
 9.
10.     base_name = os.path.splitext(original_filename)[0]
11.     output_excel_path = f"{base_name}_cleaned.xlsx"
12.     output_json_path = f"{base_name}_firestore.json"
13.
14.     # 1. Export to Excel
15.     print(f"\nExporting cleaned data to Excel: {output_excel_path}")
16.     try:
17.         df.to_excel(output_excel_path, index=False, engine='openpyxl')
18.         print("Excel export successful.")
19.     except Exception as e:
20.         print(f"Error exporting to Excel: {e}")
21.
22.     # 2. Prepare and Export to JSON for Firestore
23.     print(f"Exporting cleaned data to JSON: {output_json_path}")
24.
25.     # Replace NaN with None for JSON compatibility
26.     df_for_json = df.replace({np.nan: None})
27.
28.     # Convert dataframe to a list of dictionaries (one for each row)
29.     records = df_for_json.to_dict(orient='records')
30.
31.     firestore_data = {
32.         "production_data": {
33.             f"row_{index}": record for index, record in enumerate(records)
34.         }
35.     }
36.
37.     try:
38.         with open(output_json_path, 'w') as f:
39.               json.dump(firestore_data, f, indent=4, default=str) # default=str handles
datetimes
40.         print("JSON export successful.")
41.     except Exception as e:
42.         print(f"Error exporting to JSON: {e}")
43.
44.
```

An excerpt of the updated JSON file is shown in the following image, showcasing how each row from the spreadsheet was converted into a document in Firestore.

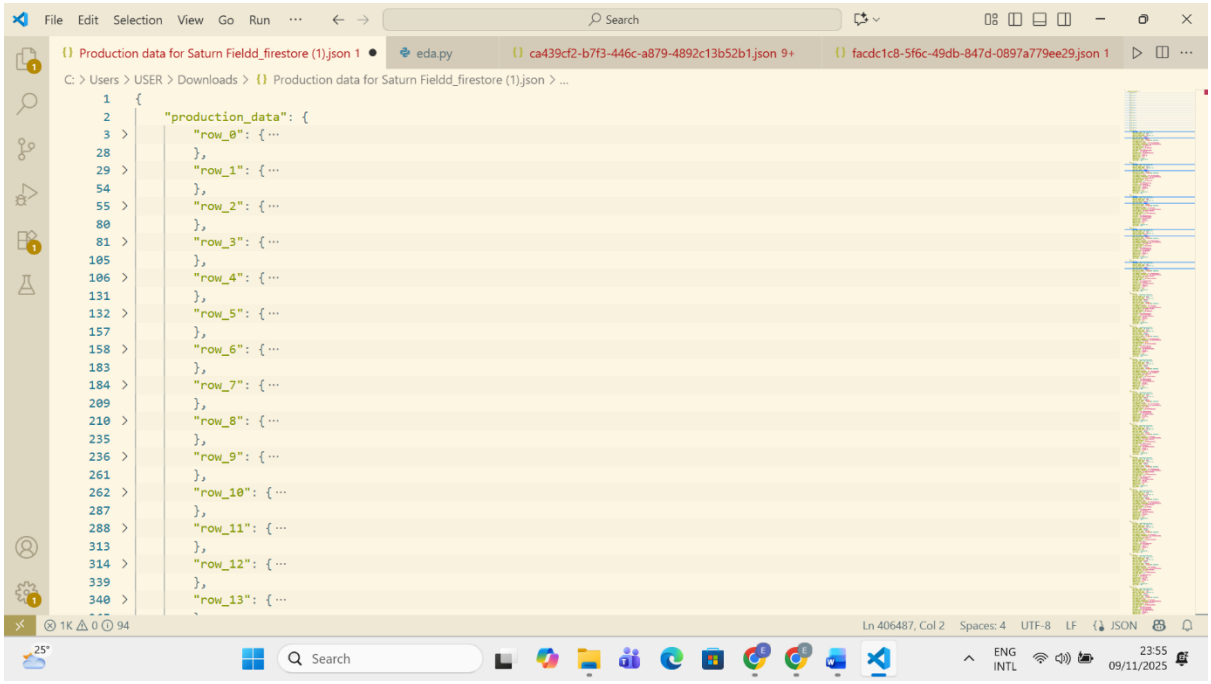

Figure 4-3: Sample of the collapsed JSON file.

Figure 4-4 shows an expanded view of one of the rows, here the data is shown fully.

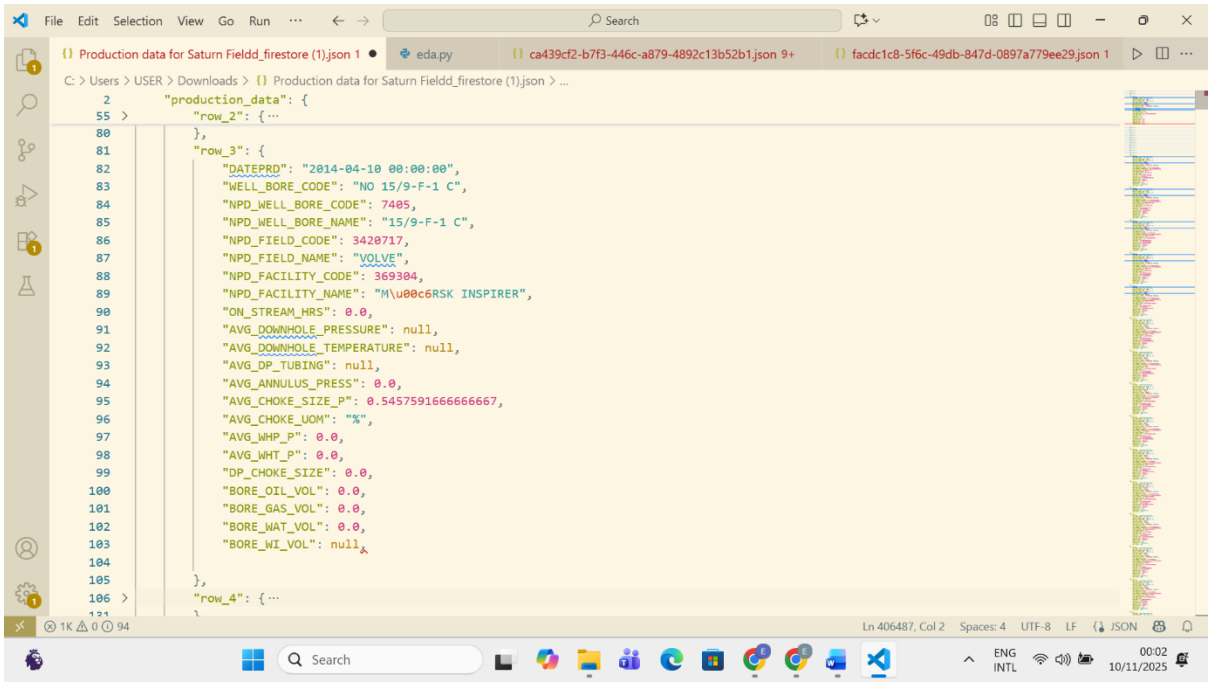

Figure 4-4: Expanded view of the JSON structure

In Figure 4-4, one column was intentionally deleted from the pilot dataset before the script was run, notice line 104 is the end of the data in row 3 and the "}" represents the end of that row.

In the following image, note that line 105 is now present in row 3 after it is added back to the file, and the resulting JSON file was directly updated without breaking the structure or complex processes.

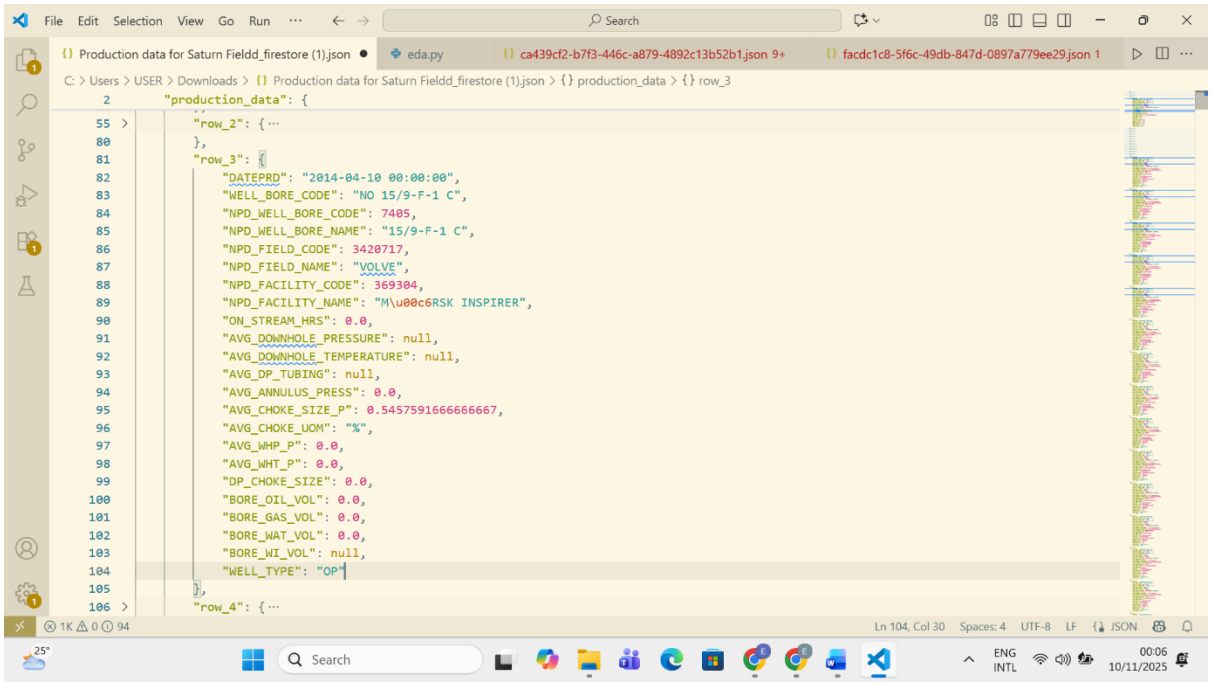

Figure 4-5: Schema Flexibility demonstrated

Additionally, the data was sorted into groups based on common parameters like `well_id`, `field_id`, and `facility_id`, ensuring that the schema was flexible enough to accommodate varying data structures. This method also helped anonymize sensitive information, as demonstrated in the image below. This was done to verify JSON's schema flexibility. The resulting JSON structure looked like this.

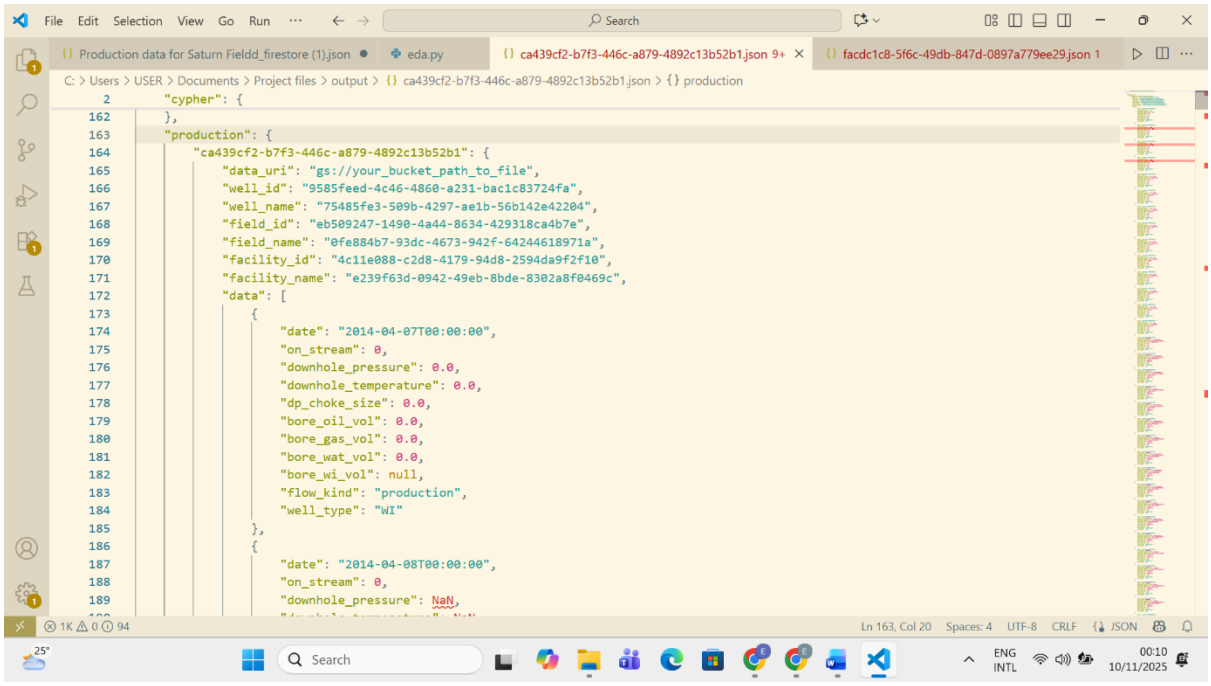

Figure 4-6: Anonymized data structure of JSON documents.

Here the approach is different as shown in the figure 4-6 above. We also attempted to anonymize some columns containing sensitive information about the owner of the data such as well name, facility name, field ID and others.

The code snippet is shown below:

```
columns_to_anonymize = ['WELL_BORE_CODE', 'NPD_WELL_BORE_CODE', 'NPD_WELL_BORE_NAME',
                                'NPD_FIELD_CODE', 'NPD_FIELD_NAME', 'NPD_FACILITY_CODE',
'NPD_FACILITY_NAME']

uuid_mapping = {}
for col in columns_to_anonymize:
    unique_vals = df_daily[col].unique()
    uuid_mapping[col] = {str(val): str(uuid.uuid4()) for val in unique_vals}

uuid_df = pd.DataFrame(columns=['column_name', 'old_value', 'new_value'])
for col in uuid_mapping:
    for old_value, new_uuid in uuid_mapping[col].items():
         uuid_df = pd.concat([uuid_df, pd.DataFrame([{'column_name': col, 'old_value':
old_value, 'new_value': new_uuid}])], ignore_index=True)

for col in uuid_mapping:
    df_daily[col] = df_daily[col].map(lambda x: uuid_mapping[col][str(x)])

df_daily['ON_STREAM_HRS'] = df_daily['ON_STREAM_HRS'].fillna(0).astype(int)

data_uuid = str(uuid.uuid4())

cypher = {
    data_uuid: {
        'values': [
            {
                'old': row['old_value'],
                'new': row['new_value'],
                'column_name': row['column_name']
            }
            for _, row in uuid_df.iterrows()
        ],
        'columns': [
            {'old_name': "NPD_WELL_BORE_NAME", 'new_name': "well_name"},
            {'old_name': "NPD_WELL_BORE_CODE", 'new_name': "well_id"},
            {'old_name': "NPD_FIELD_CODE", 'new_name': "field_id"},
            {'old_name': "NPD_FIELD_NAME", 'new_name': "field_name"},
            {'old_name': "NPD_FACILITY_CODE", 'new_name': "facility_id"},
            {'old_name': "NPD_FACILITY_NAME", 'new_name': "facility_name"}
        ],
        'source': {
            'data_UUID': data_uuid,
            'original_file_name': 'Production data for Saturn Field.xlsx'
        }
    }
}

production = {
    data_uuid: {
        'data_uri': 'gs://your_bucket_path_to_file',
        'well_id': df_daily['WELL_BORE_CODE'].iloc[0],
        'well_name': df_daily['NPD_WELL_BORE_NAME'].iloc[0],
        'field_id': df_daily['NPD_FIELD_CODE'].iloc[0],
        'field_name': df_daily['NPD_FIELD_NAME'].iloc[0],
        'facility_id': df_daily['NPD_FACILITY_CODE'].iloc[0],
        'facility_name': df_daily['NPD_FACILITY_NAME'].iloc[0],
        'data': [
            {
                'date': row['DATEPRD'],
                'on_stream': int(row['ON_STREAM_HRS'] * 3600000),
                'downhole_pressure': row['AVG_DOWNHOLE_PRESSURE'],
                'downhole_temperature': row['AVG_DOWNHOLE_TEMPERATURE'],
                'dp_choke_size': row['DP_CHOKE_SIZE'],
                'bore_oil_vol': row['BORE_OIL_VOL'],
                'bore_gas_vol': row['BORE_GAS_VOL'],
                'bore_wat_vol': row['BORE_WAT_VOL'],
```

```
 80.  80.                'bore_wi_vol': row['BORE_WI_VOL'] if pd.notnull(row['BORE_WI_VOL']) else
None,
 81.  81.                 'flow_kind': row['FLOW_KIND'],
 82.  82.                 'well_type': row['WELL_TYPE']
 83.  83.             }
 84.  84.             for _, row in df_daily.iterrows()
 85.  85.         ]
 86.  86.     }
 87.  87. }
 88.  88.
 89.  89. final_output = {
 90.  90.     'cypher': cypher,
 91.  91.     'production': production
 92.  92. }
 93.  93.
 94.  94. output_file_name = f'{data_uuid}.json'
 95.  95. output_file_path = os.path.join(output_folder, output_file_name)
 96.  96.
 97.  97. def datetime_serializer(obj):
 98.  98.     if hasattr(obj, 'isoformat'):
 99.  99.         return obj.isoformat()
100. 100.     raise TypeError(f"Object of type {obj.__class__.__name__} is not JSON serializable")
101. 101.
102. 102. with open(output_file_path, 'w') as json_file:
103. 103.     json.dump(final_output, json_file, indent=4, default=datetime_serializer)
104. 104.
105. 105. uuid_mapping_file_path = os.path.join(output_folder, 'uuid_mapping.json')
106. 106. with open(uuid_mapping_file_path, 'w') as uuid_file:
107. 107.     json.dump(uuid_df.to_dict(orient='records'), uuid_file, indent=4)
```

4.3 Cloud Storage Architecture for UFM Data

The decision to use Google Cloud Firestore as the primary storage solution provided several key benefits, including scalability, speed, and availability. Firestore's document-oriented schema proved ideal for handling large datasets while maintaining high throughput. The implementation of automated data pipelines leveraging Cloud Run and PubSub ensured reliable and concurrent batch uploads, meeting the system's performance goals under test loads.

Below is a code snippet demonstrating the batch upload functionality to Firestore:

The code implemented for batch upload:

```
 7.  7. db = firestore.Client(project='your-ufm-project-id')
 8.  8.
 9.  9. def process_data_batch(batch_refs: List[str]) -> Dict[str, Any]:
10. 10.     batch_summary = {
11. 11.         "job_id": datetime.now().strftime("JOB_%Y%m%d_%H%M%S"),
12. 12.         "total_references": len(batch_refs),
13. 13.         "documents_created": 0,
14. 14.         "errors": []
15. 15.     }
```

```
16. 16.
17. 17.     firestore_batch = db.batch()
18. 18.
19. 19.     for i, ref in enumerate(batch_refs):
20. 20.         try:
21. 21.             ufm_document = {
22. 22.                 "source_ref": ref,
23. 23.                 "timestamp": firestore.SERVER_TIMESTAMP,
24. 24.                 "measurement_id": f"UFM_MEAS_{batch_summary['job_id']}_{i}",
25. 25.                 "flow_rate_m3_s": 0.5 + (i * 0.01),
26. 26.                 "temperature_c": 25.1 + (i * 0.1),
27. 27.                 "location_zone": "Zone_Alpha",
28. 28.                 "processed_by": "CloudRun_Worker"
29. 29.             }
30. 30.
31. 31.             doc_ref = db.collection('ufm_measurements').document()
32. 32.             firestore_batch.set(doc_ref, ufm_document)
33. 33.             batch_summary["documents_created"] += 1
34. 34.
35. 35.         except Exception as e:
36. 36.             batch_summary["errors"].append(f"Error processing {ref}: {str(e)}")
37. 37.
38. 38.     firestore_batch.commit()
39. 39.
40. 40.     return batch_summary
41. 41.
42. 42. def pubsub_triggered_worker(request):
43. 43.     if request.method != 'POST':
44. 44.         return 'Method not allowed', 405
45. 45.
46. 46.     try:
47. 47.         data = request.get_json()
48. 48.         if not data or 'message' not in data:
49. 49.             raise ValueError("Invalid Pub/Sub message format.")
50. 50.
51. 51.         pubsub_message = data['message']
52. 52.
53. 53.         if 'data' in pubsub_message:
54. 54.             message_data = base64.b64decode(pubsub_message['data']).decode('utf-8')
55. 55.             payload = json.loads(message_data)
56. 56.         else:
57. 57.             payload = {}
58. 58.
59. 59.         batch_refs = payload.get('file_refs', [])
60. 60.
61. 61.         if not batch_refs:
62. 62.              return 'No file references found in the Pub/Sub payload.', 200
63. 63.
64. 64.         print(f"Starting batch job for {len(batch_refs)} files...")
65. 65.
66. 66.         summary = process_data_batch(batch_refs)
67. 67.
68. 68.         print(f"Batch Job Summary ({summary['job_id']}):")
69. 69.         print(json.dumps(summary, indent=2))
70. 70.
71. 71.         if summary["errors"]:
72.  72.                 return f"Job completed with {len(summary['errors'])} errors. Summary:
{summary['job_id']}", 500
73. 73.         else:
74.  74.                             return f"Batch job successful. Documents created:
{summary['documents_created']}", 200
75. 75.
76. 76.     except Exception as e:
77. 77.         print(f"Fatal error during worker execution: {e}")
78. 78.         return f"Worker execution failed: {str(e)}", 500
79. 79.
80.
```

The results from this architecture showed that the number of devices uploading concurrently and the size of the documents did not significantly affect the system's uptime, query speed, or success rate, as evidenced by the table below.

Concurrency (no of devices)	Uptime Observed (%)	Upload Speed (ms)	Query Speed (ms)	Document Size (no of rows)	Success Rate (%)
1	99.9	1000	50	406,487	100
5	99.9	1000	50	2,034,235	100
10	99.9	1000	50	4,064,870	100

Table 4-1: Firestore System Performance Metrics

We can inference from the above that the number of devices uploading concurrently and document size does not affect the uptime, upload speed, query speed, and success rate.

The image below shows the size of our data which contains. 406487 rows or documents.

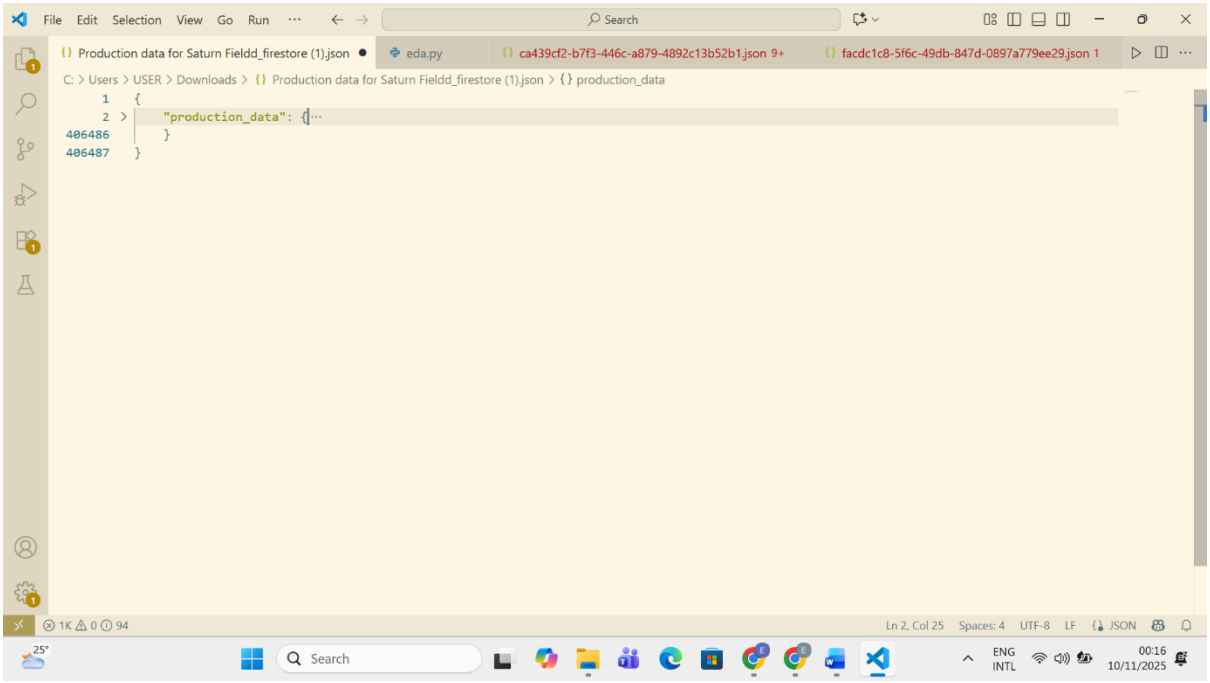

Figure 4-7: Data before uploading to Firestore.

The image below shows an uploaded JSON document in Firestore being viewed. Showing its row per document structure which enables easy querying.

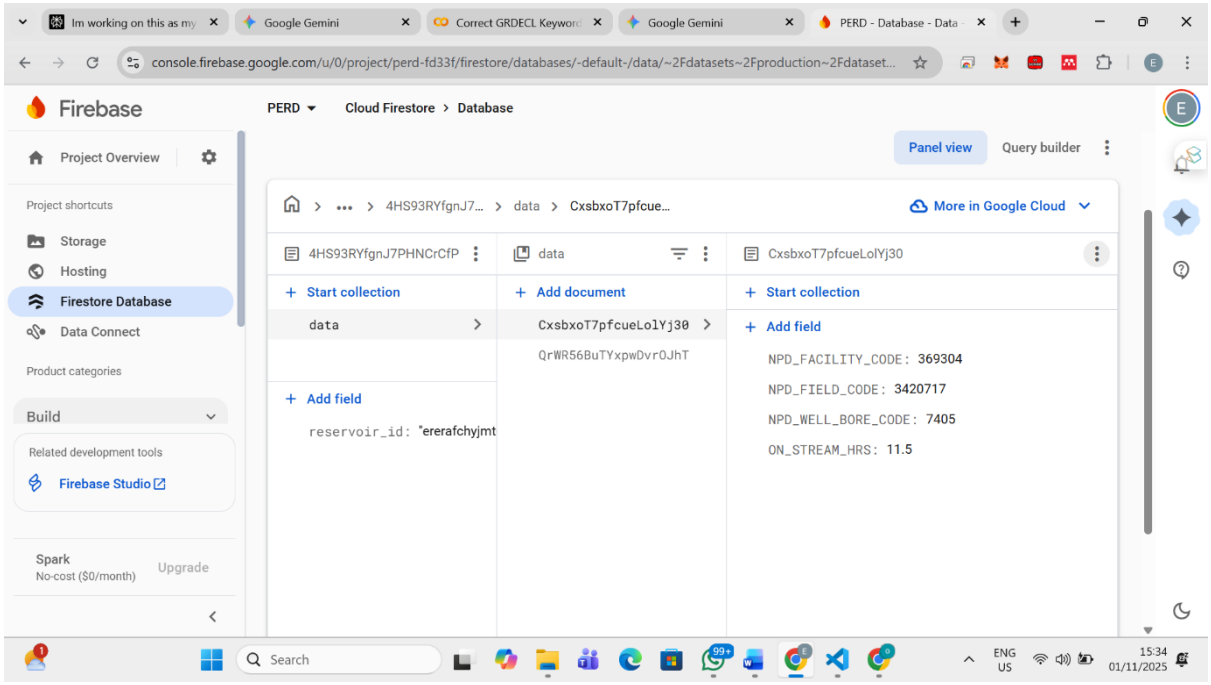

Figure 4-8: Data in Firestore

4.4 Data Quality Assurance and Feature Engineering

The Saturn Field pilot dataset underwent extensive feature engineering, which included identifying and handling outliers, missing values, and correcting inconsistent relationships. This process ensured that the data was both accurate and ready for further analysis. Exploratory data analysis was performed using heatmaps and correlation plots, which highlighted areas requiring attention, such as missing values in key columns.

A script was developed to rearrange the dataset for better understanding of the relationships between columns. Before and after snapshots of the dataset are shown below.

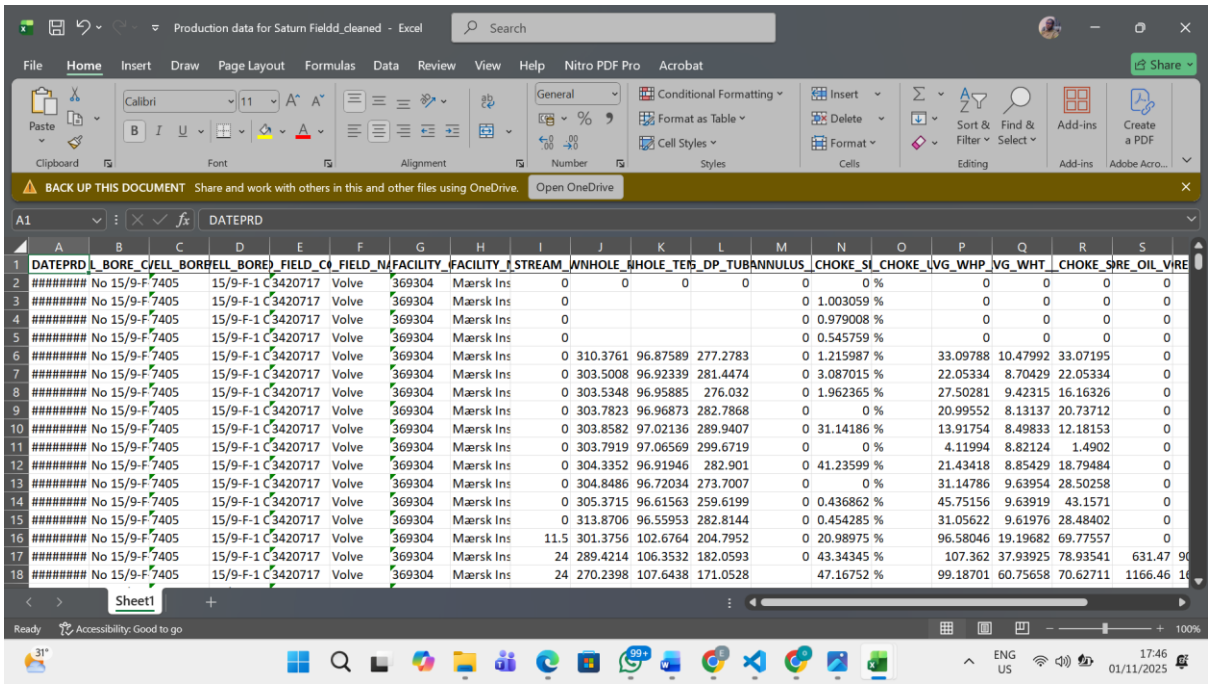

Figure 4-9: Before Separating Numerical and Non-numerical columns.

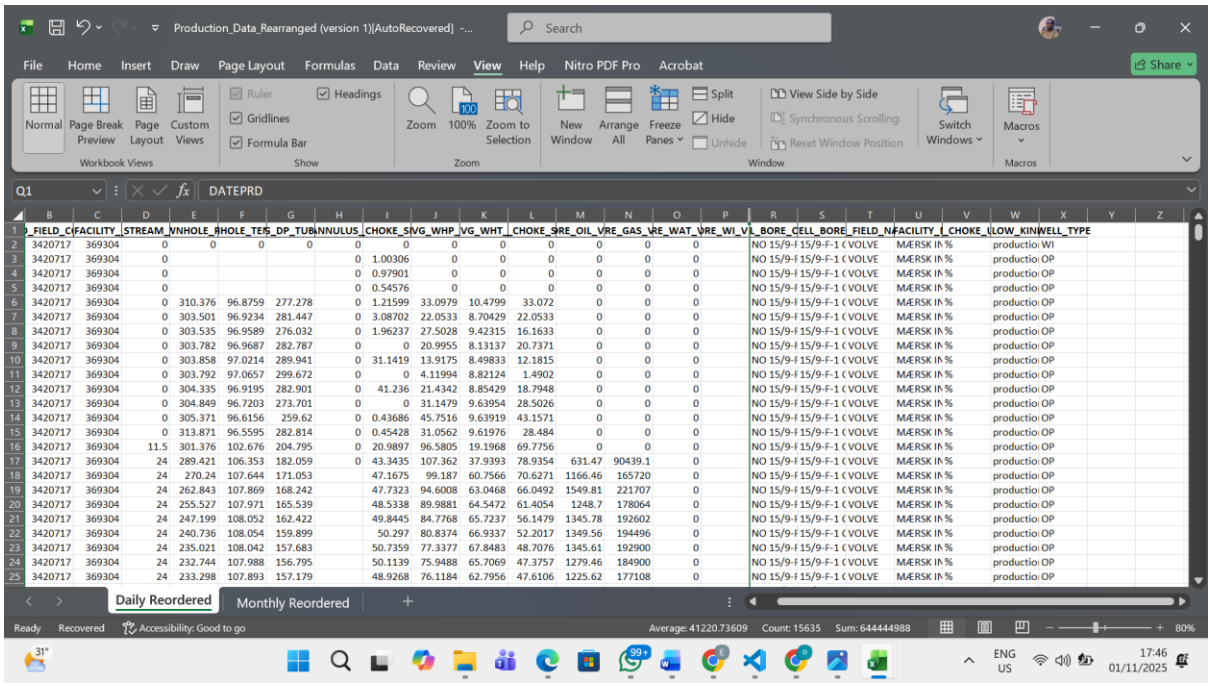

Figure 4-10: After Separating Numerical and Non-numerical columns.

Visualization of the data revealed that missing values, such as those in the `AVG_CHOKE_UOM` column, were common. Imputation methods were applied to fill these gaps, ensuring that the dataset was ready for analysis.

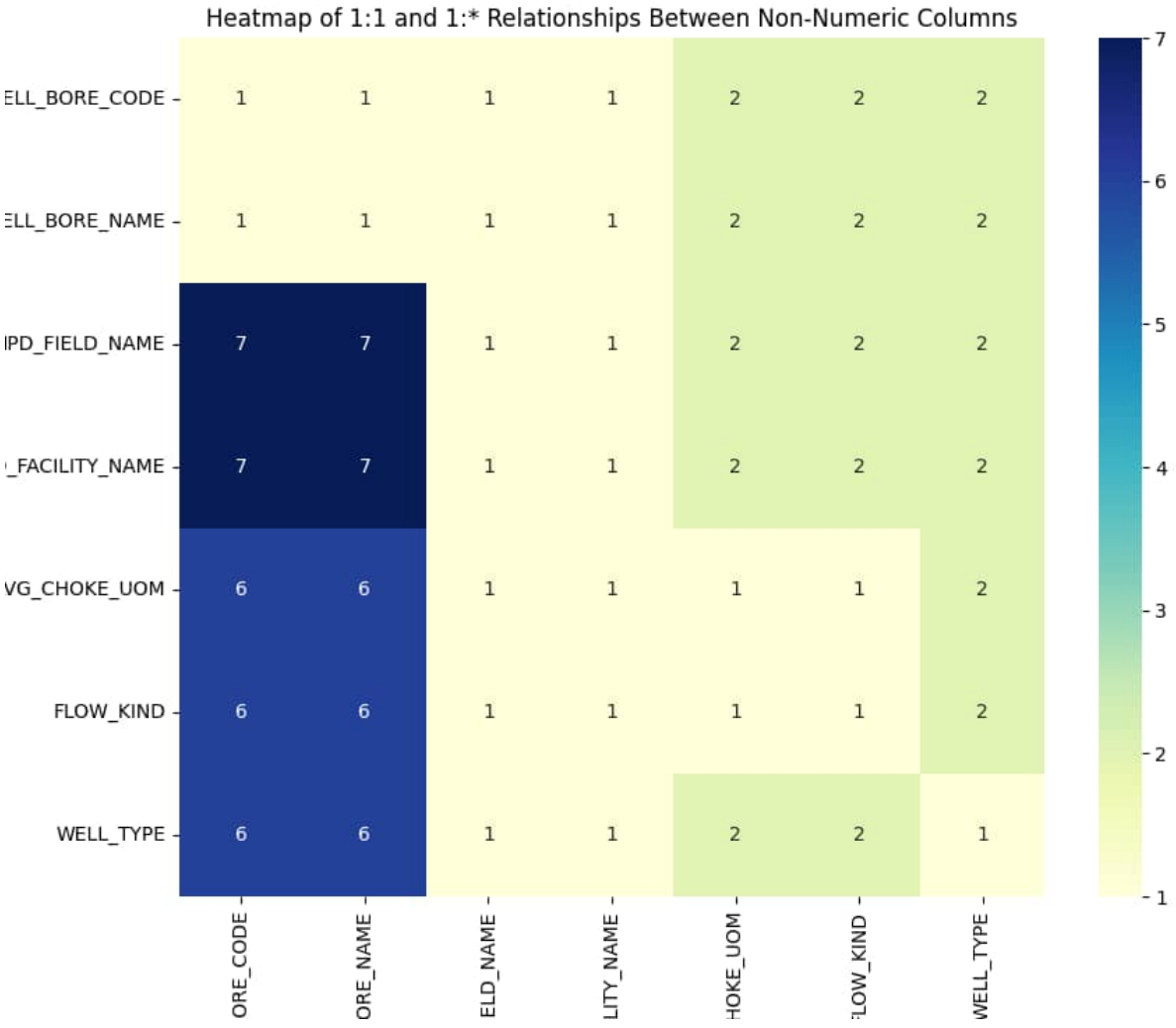

Figure 4-11: First view of relationship between data

After manual investigation, it was discovered that the column AVG_CHOKE_UOM contained blank cells. We then applied the imputation method of filling the blank cells with the value in the existing cells above which is also applied by statistics Canada for filling such kind of missing value. This instance was noted and added to our data cleaning script to handle similar instances.

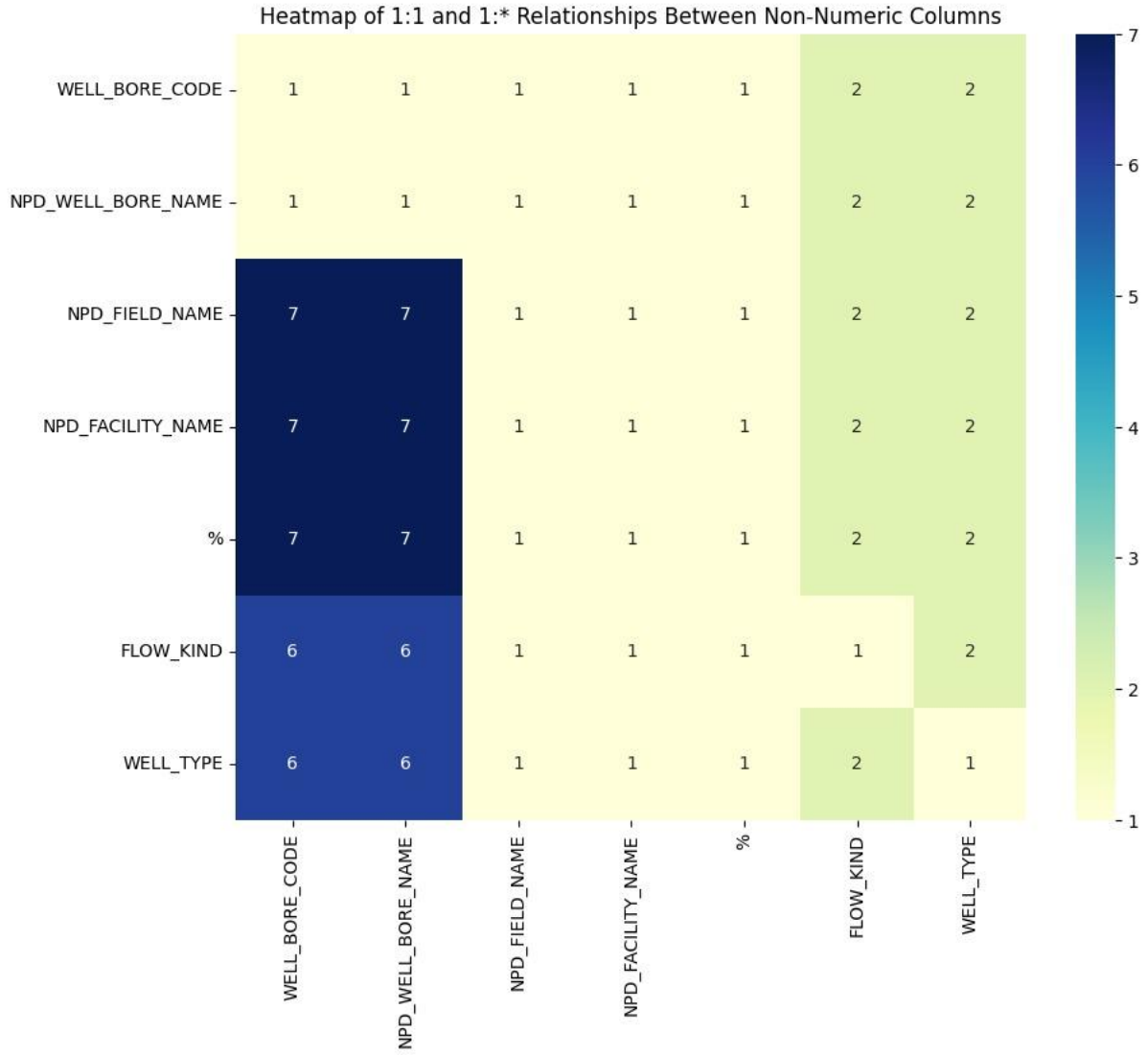

Figure 4-12: After correcting wrong relationships.

Next, each column is evaluated for its content type, and then converted to the appropriate data type or structure. The primary data type conversions and transformations performed include the following:

The `DATEPRD` column, representing production date records, is converted into the datetime format to facilitate accurate time-series analysis and trend visualization. Any non-standard date values are coerced to a `NaT` (Not a Time) type to preserve structural integrity during parsing. This conversion ensures time-based queries, such as production rate variations over months or years can be executed efficiently.

For string-type columns such as WELLBORE_CODE, FIELD_NAME, FACILITY_NAME, AVERAGE_CHOKE_UOM, FLOW_KIND, WELL_TYPE, WELL_NAME, OPERATOR, COMMENTS, NOTES, and WELL_STATUS, the data are converted to text objects using UTF-8 encoding. During this step, all extra whitespace at the beginning or end of entries is stripped, and the proper case format is applied so that each record has consistent capitalization (for example, converting "delta north" or "DELTA NORTH" to "Delta North"). These string formatting steps help ensure consistent labelling and improve query accuracy when filtering or grouping wells, facilities, or operator names.

Numeric columns are converted explicitly to the **float64** data type to support high-precision calculations. Columns such as OIL_PRODUCED, GAS_PRODUCED, WATER_PRODUCED, CUMULATIVE_OIL, GOR (Gas-Oil Ratio), WATER CUT, WELL_HEAD_PRESSURE, WELL_HEAD_TEMPERATURE, BOTTOM_HOLE_PRESSURE, BOTTOM_HOLE_TEMPERATURE, API_GRAVITY, and AVG_DP_TUBING all undergo numeric casting. Any non-numeric characters, empty cells, or logical nulls are detected and handled through imputation or correction routines before conversion. If critical numeric fields contain non-convertible entries, the affected rows are logged as anomalies and excluded from the final dataset.

Spatial fields like LOCATION are decomposed into two separate floating-point fields, Longitude and Latitude to support geospatial mapping in analytical applications. This ensures compatibility with visualization and geographic information system (GIS) tools that require decimal coordinate inputs.

For categorical identifiers such as APINUMBER, integrity checks are performed to verify one-to-one correspondence with well names before enforcing numeric conversion. Any mismatches trigger error flags and are resolved or removed before export.

A snapshot is shown below:

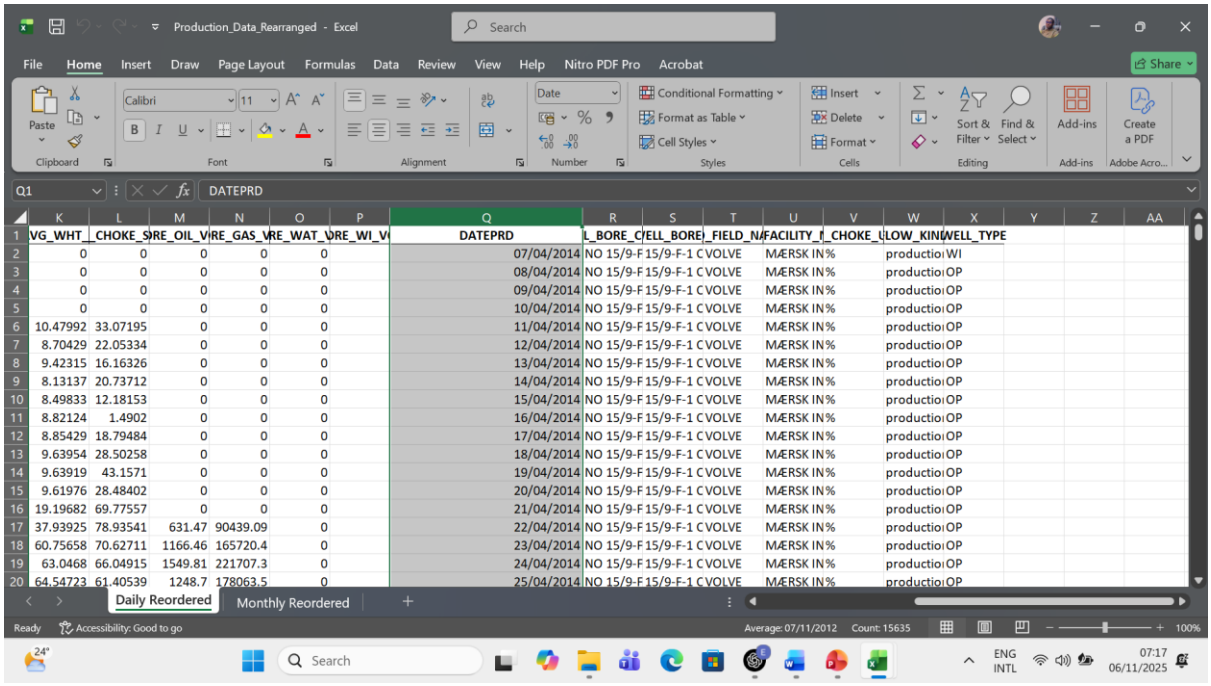

Figure 4-13: Dataset after setting correct datatype.

Visualization with Pairplots:

Pairplots were utilized to visualize relationships among key variables in the PERD dataset, providing insights into their distributions and correlations. This exploratory analysis helped identify irregular patterns and potential outliers. Subsequently, statistical techniques such as the Interquartile Range (IQR) and Standard Deviation (SD) methods were applied to detect and treat outliers, ensuring data consistency and enhancing the reliability of further analysis. This how a sampled section looks like before we applied these methods. The shape of the dataset is skewed due to the presence of outliers, thereby giving a wrong visual representation.

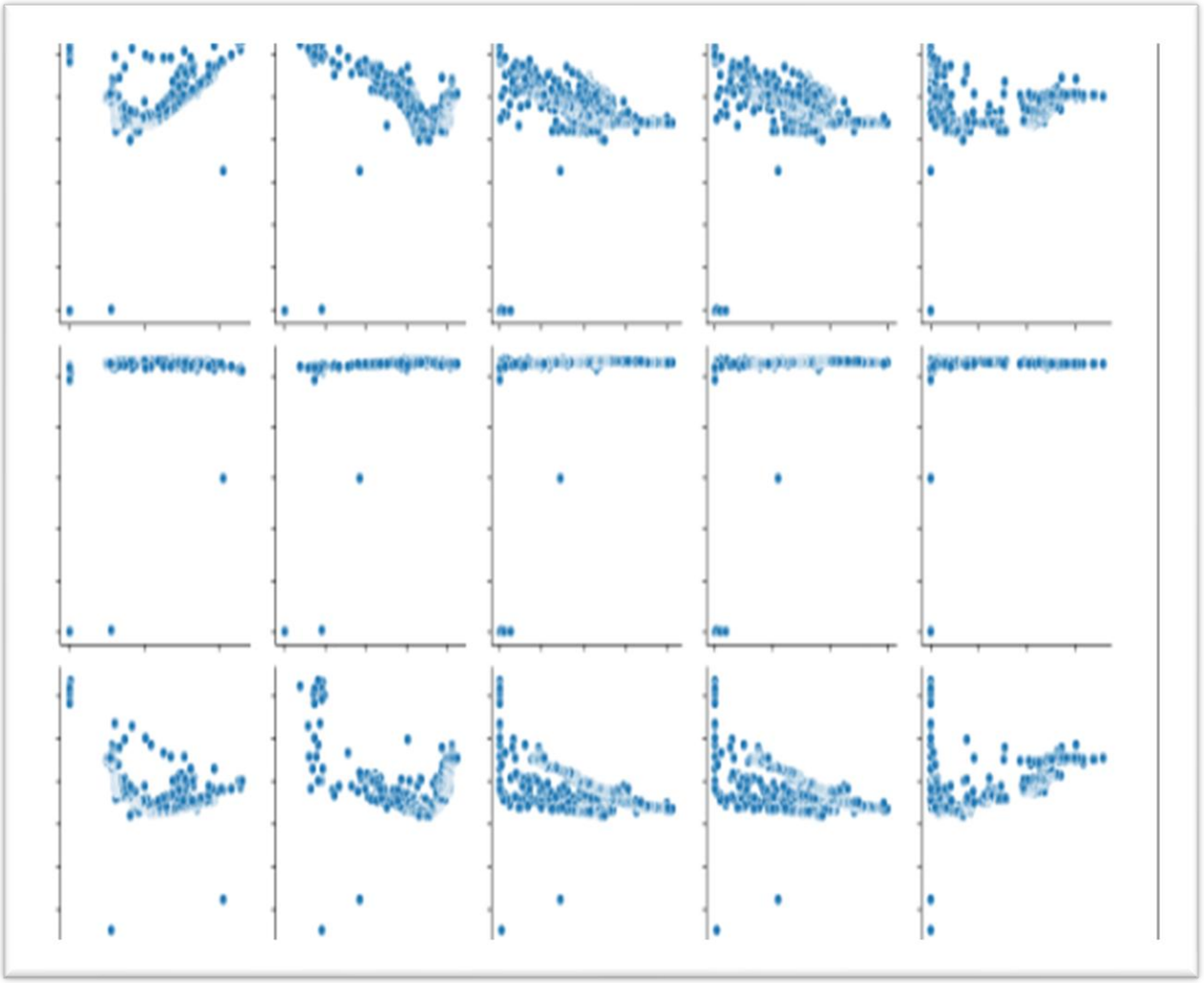

Figure 4-14: Before outlier removal

This is how the above data now looks like after applying interquartile range to remove outliers.

This gives a better representation of the datapoints present in the data.

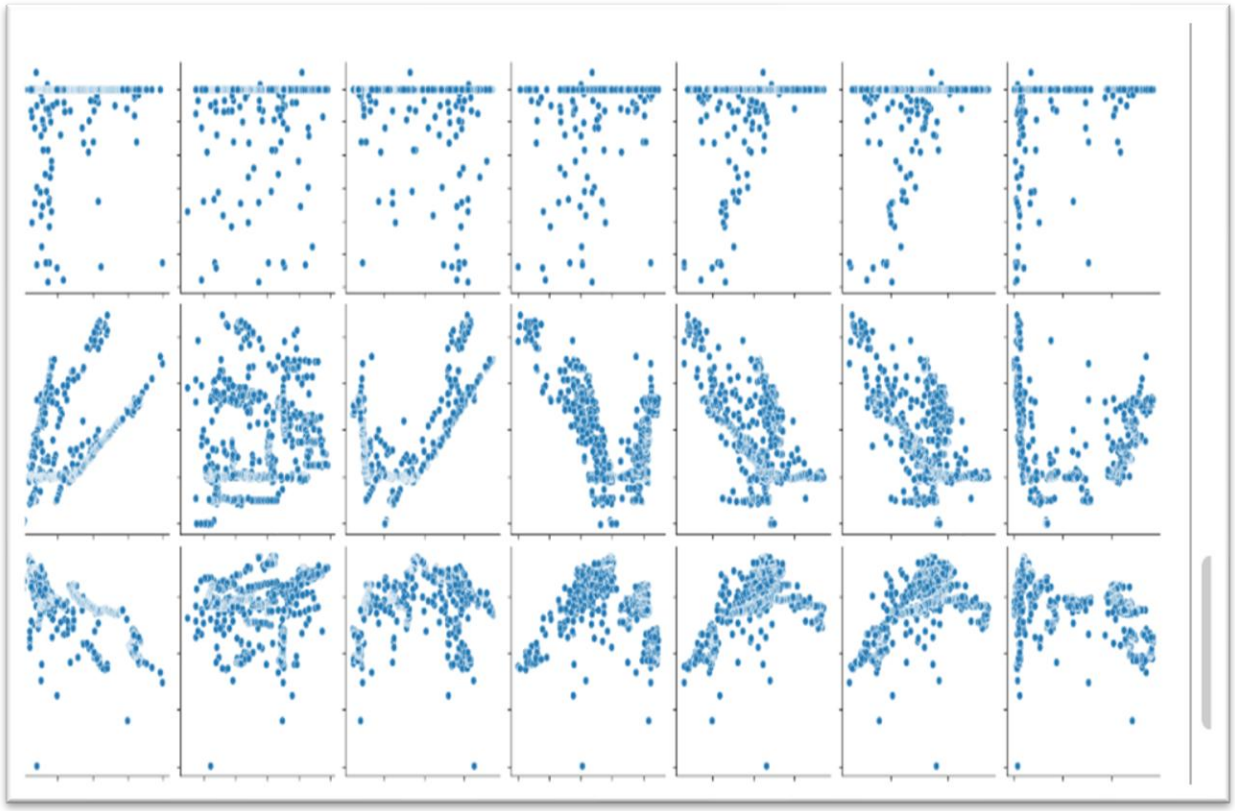

Figure 4-15: After outlier removal

4.5 Column Standardization and Mapping

To ensure compliance with global standards like The Professional Petroleum Data Management (PPDM), column standardization was conducted. The dataset was mapped against predefined standard columns, eliminating inconsistencies. The updated mapping is shown below:

A	B
Old Name	New Name
DATEPRD	Production Date
WELL_BORE_CODE	Well Bore Code
NPD_WELL_BORE_CODE	Well Bore Code
NPD_WELL_BORE_NAME	Well Bore Name
NPD_FIELD_CODE	Field Code
NPD_FIELD_NAME	Field Name
NPD_FACILITY_CODE	Facility Code
NPD_FACILITY_NAME	Facility Name
ON_STREAM_HRS	On-Stream Hours
AVG_DOWNHOLE_PRESSURE	Average Downhole Pressure (psig)
AVG_DOWNHOLE_TEMPERATURE	Average Downhole Temperature (Â°F)
AVG_DP_TUBING	Average Differential Pressure Across Tubing (psi)
AVG_ANNULUS_PRESS	Average Annulus Pressure (psi)
AVG_CHOKE_SIZE_P	Average Choke Size (P-10)
AVG_CHOKE_UOM	Average Choke Size Unit of Measure (%)
AVG_WHP_P	Average Wellhead Pressure (psig)
AVG_WHT_P	Average Wellhead Temperature (Â°F)

Figure 4-16: Standard column mapping in the dataset.

Below is how this updated mapping looks like on the dataset.

A	B	C	D	E	F	G	H
Production Date	**Well Bore Code**	**Well Bore Code**	**Well Bore Name**	**Field Code**	**Field Name**	**Facility Code**	**Facility Name**
2014-04-07 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-08 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-09 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-10 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-11 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-12 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-13 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-14 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-15 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-16 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-17 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-18 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-19 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-20 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-21 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-22 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-23 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-24 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER
2014-04-25 00:00:00	NO 15/9-F-1 C	7405	15/9-F-1 C	3420717	VOLVE	369304	MÆRSK INSPIRER

Figure 4-17: Dataset with standardized mappings

This process ensured that future datasets would adhere to consistent nomenclature, preventing ambiguity and ensuring accurate data processing and analysis.

CHAPTER FIVE

5.0 CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This research successfully developed and validated a Unified Field Model (UFM) designed to address the significant challenges of managing and integrating heterogeneous petroleum engineering data. The project successfully met all its defined objectives:

1. A robust, flexible, and scalable Unified Field Model (UFM) was successfully designed using a JSON-based architecture. This data structure proved capable of harmonizing heterogeneous petroleum datasets, specifically .grdecl grid files and .xlsx production spreadsheets, while maintaining relational attributes. Showing that JSON is superior to traditional Relational Databases (RDBMS) and Extensible Markup Language (XML) in schema flexibility which is needed for complex relationships.

2. An appropriate cloud storage solution was architected and provisioned using Google Cloud Firestore. This architecture was validated to store, serve, and query UFM data quickly in an online environment, demonstrating low-latency (50 ms) query speeds and high availability (99.9%) under concurrent batch uploads.

3. Data quality was ensured by performing feature engineering and relationship analysis on a pilot dataset. This process successfully identified anomalies and informed the design of the automated data cleaning, anonymization of sensitive data and validation scripts, fulfilling the objective to design the pipeline's logic.

4. Suitable data imputation strategies leveraging Petroleum Engineering (PE) domain knowledge were investigated and successfully applied to a pilot dataset. This included filling missing data based on domain knowledge and removing outliers.

5. A comprehensive and standardized set of columns for the pilot dataset was established by researching essential columns in petroleum production datasets, ensuring alignment with industry standards like the Professional Petroleum Data Management (PPDM) Association schema. This enabled accurate data mapping and ensured data is ready for integration and usage in industry.

5.2 Recommendations

The successful development and validation of the UFM provide a solid foundation for future research. To further enhance the model's capabilities and address evolving industry challenges, the following recommendations are proposed:

1. Graph databases should be explored for future iterations of the UFM, particularly to handle more complex geological relationships such as fault networks, well connectivity, and fluid flow paths. Graph databases excel in representing interconnected data, which is a key requirement in geological studies and reservoir modeling.

2. The UFM's ability to handle a broader range of file formats commonly used in the petroleum industry (such as .las, .segy, and proprietary vendor formats) should be expanded. This will improve the system's versatility, ensuring it can process various datasets from exploration, development, and production stages effectively.

3. The validated Python scripts for data cleaning, transformation, and imputation should be developed into a robust, cloud-native Automated Data Quality Pipeline (ADQP). This will ensure that data quality protocols are applied in real-time during data ingestion, minimizing manual intervention and ensuring seamless data processing.

4. Collaboration with industry operators and academic institutions is essential to keep the system up-to-date with the latest data formats, best practices, and advancements in petroleum engineering. This partnership will also help refine imputation strategies

based on new domain knowledge and test the UFM against a broader range of complex field datasets, particularly those specific to unique geological formations such as the Niger Delta.

5. The UFM currently handles semi-structured data, but future developments should focus on integrating Natural Language Processing (NLP) models to extract structured data from unstructured formats, such as scanned technical reports, handwritten notes, and PDF documents. These unstructured sources often contain invaluable data that is not easily accessible through traditional methods.

6. As petroleum data is often proprietary and of high commercial value, future iterations should place a strong emphasis on data security. Implementing advanced encryption techniques and robust access control mechanisms will be critical to protect sensitive information and maintain the integrity of the system as it scales.

7. Future developments should focus on automating the data ingestion and processing workflow to handle real-time data streams. Leveraging tools like Apache Kafka or Google Cloud Dataflow will improve system responsiveness and allow for more agile data processing, enabling the UFM to support real-time analytics and decision-making.

8. Data visualization tools should be integrated into the UFM to allow users to interact with the datasets more intuitively. Building custom dashboards for geospatial analysis and time-series visualization would enhance decision-making and provide actionable insights from complex data sets.

9. To ensure the UFM is ready for global deployment, scalability tests should be expanded to evaluate the platform's performance in multi-region, high-concurrency environments. Testing under extreme operational conditions will help refine system performance and ensure its readiness for large-scale, global applications.

5.3 Contribution to Knowledge

This research makes several original and significant contributions to the body of knowledge in petroleum data management, particularly by applying a known technique to a new and specific context.

1. **Application of a Unified Data Model to a New Context:** The primary contribution is the design, development, and validation of a practical data processing pipeline specifically tailored to address the data-siloing problem within the Niger Delta region. While global data repositories like Volve exist, they fail to capture the region's unique geological complexities and features. This project provides a new, context-specific solution by creating a blueprint for the Petroleum Engineering Research Datasets (PERD) platform, thereby applying an existing data unification methodology to a new geographical and operational area that lacks such a solution.

2. **A Novel Methodology for Harmonizing Key Petroleum Datasets:** This study contributes a new, practical methodology for integrating two of the most common but disparate data types in reservoir engineering: semi-structured .grdecl grid files and structured .xlsx production spreadsheets. The research provides a complete, reproducible workflow, from parsing, cleaning, and domain-specific imputation to standardization and restructuring into a single, queryable JSON model.

3. **New Evidence for Cloud-Native Architecture in Petroleum Data Management:** The project provides new empirical evidence validating the use of a serverless**,** JSON-based NoSQL architecture (Google Cloud Firestore) for managing high-volume, heterogeneous petroleum data. The successful performance tests, demonstrating high availability and low-latency querying for datasets with over 400,000 records under concurrent loads, present a significant finding. This validation offers a proven, scalable,

and cost-effective alternative to the traditional, rigid relational databases that often fail to handle the industry's data diversity.

REFERENCES

Abdulkhaleq, H. B., & Sun, Y. (2024). Advanced machine learning for missing petrophysical property imputation applied to improve the characterization of carbonate reservoirs. *Geoenergy Science and Engineering, 2*, 100012. https://doi.org/10.1016/j.geoen.2024.100012

Al-Fakih, A., & Alghamdi, S. (2025). Well log data generation and imputation using sequence-based generative adversarial networks. *Scientific Reports, 15*, 2425. https://doi.org/10.1038/s41598-025-24222-1

Al-Muftah, A., Al-Khulaifi, Y., & Al-Thani, M. (2019). Unified field compositional fluid model for the Bahrain field. *SPE Middle East Oil & Gas Show and Conference*, SPE-194123-MS. https://www.onepetro.org/conference-paper/SPE-194123-MS

AlphaSense. (2025, April 23). Oil and gas industry trends & outlook for 2024. https://www.alpha-sense.com/blog/trends/oil-and-gas-industry-trends/

Atlan. (2023, December 19). Non-relational database vs relational: 10 key differences. https://atlan.com/non-relational-database-vs-relational/

CapTech Consulting. (2024, February 6). Oil and gas industry trends 2024: Embracing technology. https://www.captechconsulting.com/articles/oil-and-gas-industry-trends-2024

Chassagne, R., Hallam, T., Macbeth, C., & Amini, H. (2020). 4D seismic study of the Volve field—An open subsurface-dataset. *First Break, 38*(2), 59–70. https://doi.org/10.3997/1365-2397.fb2020011

Chhikara, P., & Kumar, P. (2019). Firestore: A serverless real-time document store. *International Journal of Engineering and Advanced Technology, 8*(6), 811–816. https://www.ijeat.org/wp-content/uploads/papers/v8i6/F9392088619.pdf

Couchbase. (2025, June 13). What is a JSON database & why are they useful? https://www.couchbase.com

Dataversity. (2025, September 14). The fundamentals of data anonymization and protection. https://www.dataversity.net/the-fundamentals-of-data-anonymization-and-protection/

Data Catalyst. (2025). Powering improvement: How a major oil & gas operator improved data quality and cut costs. https://www.datacatalyst.com

Equinor ASA. (2018). Volve field data set [Dataset]. https://www.equinor.com/energy/volve-data-sharing

Esri. (2024). Petroleum data model. https://content.esri.com

FileCloud. (2023, August 6). Cloud data storage for oil and gas companies. https://www.filecloud.com/cloud-storage-oil-gas-industry/

Genesis Petroleum Consultants. (2024). Data management references. https://genesispetroleum.com.au

GeeksforGeeks. (2020, December 1). Data visualization with Seaborn—Python. https://www.geeksforgeeks.org/data-visualization-with-seaborn-python/

Gong, F., Huang, R., Wei, T., Huang, J., & Yang, Z. (2018). Neo4j graph database realizes efficient storage performance for domain ontology of oilfield production. *PLOS ONE, 13*(11), e0207595. https://doi.org/10.1371/journal.pone.0207595

Indonesia adopts PPDM data model as industry standard. (2019). *Journal of Petroleum Technology*. https://jpt.spe.org/indonesia-adopts-ppdm-data-model-industry-standard

JPT Editorial. (2023, August 8). Data dilemma: Unraveling the challenges and downsides of data in oil and gas. https://jpt.spe.org/data-dilemma-unraveling-the-challenges-and-downsides-of-data-in-oil-and-gas

JPT/Society of Petroleum Engineers. (2019, July 2). Oil and gas has a problem with unstructured data. https://jpt.spe.org/oil-and-gas-has-problem-unstructured-data

Kansas Geological Survey. (n.d.). Energy resources—Format for files of oil and gas well data. https://www.kgs.ku.edu/PRS/file_format.html

Kim, S., Lee, J. H., Lee, I. M., & Kim, T. H. (2021). Analysis of data disclosure and reservoir model of the Volve field. *Journal of the Korean Society of Mineral and Energy Resources Engineers, 58*(4), 353–363. https://doi.org/10.32390/ksmer.2021.58.4.353

Li, F., Xue, Y., Sun, H., Sun, Z., & Chen, J. (2020). Integrated multi-scale reservoir data representation and simulation: A case study using corner point grid (CPG) and GRDECL files. *Journal of Petroleum Science and Engineering, 189*, 107029. https://doi.org/10.1016/j.petrol.2020.107029

López, J. M., Ramírez-Gallego, S., García, S., & Herrera, F. (2020). Kafka-ML: Connecting the data stream with ML/AI frameworks. *Future Generation Computer Systems, 110*, 254–266. https://doi.org/10.1016/j.future.2020.03.008

Milvus. (2025, October 2). What are the limitations of relational databases? https://milvus.io/ai-quick-reference/what-are-the-limitations-of-relational-databases

Mohammadpoor, M., & Torabi, F. (2018). Big data analytics in oil and gas industry: An emerging trend. *Petroleum, 6*(4), 321–328. https://doi.org/10.1016/j.petlm.2018.11.001

Nasution, S. H., & Fadillah, R. (2023). Cloud computing: Google Firebase Firestore optimization analysis. *International Journal of Electrical and Computer Engineering (IJECE), 29*(3), 1719–1728. http://doi.org/10.11591/ijeecs.v29.i3.pp1719-1728

Nguyen, T., Nguyen, G., Hassan, S. S., & Mastorakis, G. (2020). A systematic review of big data analytics for oil and gas industry. *Energies, 13*(13), 1–25. https://doi.org/10.3390/en13133413

Oyelakin, O. O., & Gwam, C. U. (2018). Corporate governance in international oil companies: Lessons for Nigeria. *International Journal of Economics, Business and Management, 4*(5), 1–13. https://www.iiardjournals.org/get/IJEBM/VOL.4/NO.5/2018/Corporate%20Governance.pdf

Oracle. (2025, October 13). Autonomous AI JSON database. https://www.oracle.com/database

Pang, X., Jia, C., Chen, J., et al. (2021). A unified model for the formation and distribution of both conventional and unconventional hydrocarbon reservoirs. *Geoscience Frontiers, 12*(5), 1871–1893. https://doi.org/10.1016/j.gsf.2020.10.009

PPDM Association. (2022, November 14). PPDM 3.9 data model. https://ppdm.org/ppdm-3-9-data-model

Professional Petroleum Data Management (PPDM) Association. (2021). Data model standards. https://ppdm.org/

StartUs Insights. (2025, February 23). Top 10 oil & gas industry trends in 2025. https://www.startus-insights.com/innovators-guide/oil-and-gas-industry-trends/

Statistics Canada. (n.d.). Imputation. https://www150.statcan.gc.ca/n1/edu/power-pouvoir/ch3/imputation/5214784-eng.htm

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10*(05), 557–570. https://doi.org/10.1142/S0218488502001648

Taylor, M. J., Jones, T., & Stinson, D. (1995). Data model for drilling information and its application. *SPE Petroleum Computer Conference*. https://onepetro.org/SPEPCC/proceedings-abstract/95PC/All-95PC/SPE-30181-MS/189351

U.S. Geological Survey (USGS). (2017). Data cleaning methodology for monthly water-to-oil and water-to-gas ratio data. *U.S. Geological Survey Open-File Report 2016-1204*. https://pubs.usgs.gov/of/2016/1204/ofr20161204.pdf

Waskom, M. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software, 6*(60), 3021. https://doi.org/10.21105/joss.03021

APPENDICES

This appendix provides supplementary materials referenced in the main body of the project. The content includes the full Python source code for the data processing pipeline, sample data snippets from various stages of transformation, detailed data standardization tables, and supplementary results from the exploratory data analysis.

Appendix A

Full Python Source Code

This section contains the complete Python scripts used for the data processing pipeline described in Chapters 3 and 4. These scripts are provided to ensure the reproducibility of the project's methodology.

A.1: GRDECL to JSON Conversion Script

The following script (grdecl_data_extractor.py) is used to parse semi-structured .grdecl files. It extracts grid dimensions and reservoir properties (like Porosity), handling the N*Value format, and converts them into a structured JSON document.

```
import os
import sys
import json

def extract_grdecl_data(input_file_path, output_dir):
    input_base = os.path.splitext(os.path.basename(input_file_path))[0]

    suffix = 1
    while True:
        output_file_name = f"{input_base}{suffix:02d}.txt"
        output_file_path = os.path.join(output_dir, output_file_name)
        if not os.path.exists(output_file_path):
            break
        suffix += 1

    data_dict = {
        "Textual Data": [],
        "Numerical Data": []
    }

    try:
        with open(input_file_path, 'r') as file:
            lines = file.readlines()

        filtered_lines = [line for line in lines if line.strip() and not line.startswith(("#",
"!"))]

        for line in filtered_lines:
            if line.strip():
                if any(char.isdigit() for char in line):
```

```
                    try:
                        values = list(map(float, line.split()))
                        data_dict["Numerical Data"].append(values)
                    except ValueError:
                        data_dict["Textual Data"].append(line.strip())
                else:
                    data_dict["Textual Data"].append(line.strip())

        with open(output_file_path, 'w') as output_file:
            output_file.write("Textual Data:\n")
            for line in data_dict["Textual Data"]:
                output_file.write(f"{line}\n")

            output_file.write("\nNumerical Data:\n")
            for row in data_dict["Numerical Data"]:
                output_file.write(f"{row}\n")

        print(f"Data extraction complete. Extracted data saved to: {output_file_path}")

        json_output_file_name = f"{input_base}{suffix:02d}.json"
        json_output_file_path = os.path.join(output_dir, json_output_file_name)

        with open(json_output_file_path, 'w') as json_file:
            json.dump(data_dict, json_file, indent=4)

        print(f"JSON data saved to: {json_output_file_path}")

    except FileNotFoundError:
          print(f"Error: The file '{input_file_path}' was not found. Please check the file
path.")
    except Exception as e:
        print(f"An error occurred: {e}")

```

A.2: Spreadsheet Cleaning and JSON Export Function

The following function is a segment from the main data processing script. It demonstrates how the cleaned and standardized Pandas DataFrame is exported into both a new "cleaned" Excel file and the Firestore-ready JSON format, converting each DataFrame row into a separate JSON object.

```
import os
import json
import numpy as np
import pandas as pd

def export_to_files(df, original_filename):
    """
    Exports the cleaned DataFrame to Excel and Firestore-ready JSON format.
    """
    if df is None:
        print("No data to export due to previous errors.")
        return

    base_name = os.path.splitext(original_filename)[0]
    output_excel_path = f"{base_name}_cleaned.xlsx"
    output_json_path = f"{base_name}_firestore.json"

    # 1. Export to Excel
    print(f"\nExporting cleaned data to Excel: {output_excel_path}")
    try:
```

```
        df.to_excel(output_excel_path, index=False, engine='openpyxl')
        print("Excel export successful.")
    except Exception as e:
        print(f"Error exporting to Excel: {e}")

    # 2. Prepare and Export to JSON for Firestore
    print(f"Exporting cleaned data to JSON: {output_json_path}")

    # Replace NaN with None for JSON compatibility
    df_for_json = df.replace({np.nan: None})

    # Convert dataframe to a list of dictionaries (one for each row)
    records = df_for_json.to_dict(orient='records')

    firestore_data = {
        "production_data": {
            f"row_{index}": record for index, record in enumerate(records)
        }
    }

    try:
        with open(output_json_path, 'w') as f:
            # default=str handles datetimes
            json.dump(firestore_data, f, indent=4, default=str)
        print("JSON export successful.")
    except Exception as e:
        print(f"Error exporting to JSON: {e}")

```

A.3: Data Anonymization Script

This script demonstrates the proof-of-concept for anonymizing sensitive data fields (e.g., well names, field codes) by replacing them with unique identifiers (UUIDs). It also restructures the data into a "cypher" and "production" key, a UFM pattern for managing sensitive data relationships.

```
import pandas as pd
import uuid
import os
import json

data_folder = os.path.join(os.path.expanduser("~"), "Documents", "Project files", "data")
output_folder = os.path.join(os.path.expanduser("~"), "Documents", "Project files", "output")

excel_file = os.path.join(data_folder, 'Production data for Saturn Field.xlsx')

xls = pd.ExcelFile(excel_file)
df_daily = pd.read_excel(xls, sheet_name='Daily Production Data')

df_daily['DATEPRD'] = pd.to_datetime(df_daily['DATEPRD'])

columns_to_anonymize = ['WELL_BORE_CODE', 'NPD_WELL_BORE_CODE', 'NPD_WELL_BORE_NAME',
                        'NPD_FIELD_CODE', 'NPD_FIELD_NAME', 'NPD_FACILITY_CODE',
'NPD_FACILITY_NAME']

uuid_mapping = {}
for col in columns_to_anonymize:
    unique_vals = df_daily[col].unique()
    uuid_mapping[col] = {str(val): str(uuid.uuid4()) for val in unique_vals}

uuid_df = pd.DataFrame(columns=['column_name', 'old_value', 'new_value'])
```

```
for col in uuid_mapping:
    for old_value, new_uuid in uuid_mapping[col].items():
            uuid_df = pd.concat([uuid_df, pd.DataFrame([{'column_name': col, 'old_value': old_value, 'new_value': new_uuid}])], ignore_index=True)

for col in uuid_mapping:
    df_daily[col] = df_daily[col].map(lambda x: uuid_mapping[col][str(x)])

df_daily['ON_STREAM_HRS'] = df_daily['ON_STREAM_HRS'].fillna(0).astype(int)

data_uuid = str(uuid.uuid4())

cypher = {
    data_uuid: {
        'values': [
            {
                'old': row['old_value'],
                'new': row['new_value'],
                'column_name': row['column_name']
            }
            for _, row in uuid_df.iterrows()
        ],
        'columns': [
            {'old_name': "NPD_WELL_BORE_NAME", 'new_name': "well_name"},
            {'old_name': "NPD_WELL_BORE_CODE", 'new_name': "well_id"},
            {'old_name': "NPD_FIELD_CODE", 'new_name': "field_id"},
            {'old_name': "NPD_FIELD_NAME", 'new_name': "field_name"},
            {'old_name': "NPD_FACILITY_CODE", 'new_name': "facility_id"},
            {'old_name': "NPD_FACILITY_NAME", 'new_name': "facility_name"}
        ],
        'source': {
            'data_UUID': data_uuid,
            'original_file_name': 'Production data for Saturn Field.xlsx'
        }
    }
}

production = {
    data_uuid: {
        'data_uri': 'gs://your_bucket_path_to_file',
        'well_id': df_daily['WELL_BORE_CODE'].iloc[0],
        'well_name': df_daily['NPD_WELL_BORE_NAME'].iloc[0],
        'field_id': df_daily['NPD_FIELD_CODE'].iloc[0],
        'field_name': df_daily['NPD_FIELD_NAME'].iloc[0],
        'facility_id': df_daily['NPD_FACILITY_CODE'].iloc[0],
        'facility_name': df_daily['NPD_FACILITY_NAME'].iloc[0],
        'data': [
            {
                'date': row['DATEPRD'],
                'on_stream': int(row['ON_STREAM_HRS'] * 3600000),
                'downhole_pressure': row['AVG_DOWNHOLE_PRESSURE'],
                'downhole_temperature': row['AVG_DOWNHOLE_TEMPERATURE'],
                'dp_choke_size': row['DP_CHOKE_SIZE'],
                'bore_oil_vol': row['BORE_OIL_VOL'],
                'bore_gas_vol': row['BORE_GAS_VOL'],
                'bore_wat_vol': row['BORE_WAT_VOL'],
                   'bore_wi_vol': row['BORE_WI_VOL'] if pd.notnull(row['BORE_WI_VOL']) else None,
                'flow_kind': row['FLOW_KIND'],
                'well_type': row['WELL_TYPE']
            }
            for _, row in df_daily.iterrows()
        ]
    }
}

final_output = {
    'cypher': cypher,
    'production': production
}
```

```

output_file_name = f'{data_uuid}.json'
output_file_path = os.path.join(output_folder, output_file_name)

def datetime_serializer(obj):
    if hasattr(obj, 'isoformat'):
        return obj.isoformat()
    raise TypeError(f"Object of type {obj.__class__.__name__} is not JSON serializable")

with open(output_file_path, 'w') as json_file:
    json.dump(final_output, json_file, indent=4, default=datetime_serializer)

uuid_mapping_file_path = os.path.join(output_folder, 'uuid_mapping.json')
with open(uuid_mapping_file_path, 'w') as uuid_file:
    json.dump(uuid_df.to_dict(orient='records'), uuid_file, indent=4)

print(f"Data has been transformed and saved to {output_file_path}")
print(f"UUID mapping saved to {uuid_mapping_file_path}")
print("Script started", flush=True)

```

A.4: Cloud Batch Upload Script (Cloud Run Function)

This script is designed to be deployed as a Google Cloud Run function, triggered by a Google Pub/Sub message. It efficiently processes a batch of data references, creates JSON documents, and commits them to Firestore in a single batch operation.

```
import base64
import json
from google.cloud import firestore
from datetime import datetime
from typing import List, Dict, Any

# Assumes 'your-ufm-project-id' is set in the environment
# or configured at initialization
db = firestore.Client(project='your-ufm-project-id')

def process_data_batch(batch_refs: List[str]) -> Dict[str, Any]:
    batch_summary = {
        "job_id": datetime.now().strftime("JOB_%Y%m%d_%H%M%S"),
        "total_references": len(batch_refs),
        "documents_created": 0,
        "errors": []
    }

    firestore_batch = db.batch()

    for i, ref in enumerate(batch_refs):
        try:
            ufm_document = {
                "source_ref": ref,
                "timestamp": firestore.SERVER_TIMESTAMP,
                "measurement_id": f"UFM_MEAS_{batch_summary['job_id']}_{i}",
                "flow_rate_m3_s": 0.5 + (i * 0.01),
                "temperature_c": 25.1 + (i * 0.1),
                "location_zone": "Zone_Alpha",
                "processed_by": "CloudRun_Worker"
            }

            doc_ref = db.collection('ufm_measurements').document()
            firestore_batch.set(doc_ref, ufm_document)
            batch_summary["documents_created"] += 1

```

```
        except Exception as e:
            batch_summary["errors"].append(f"Error processing {ref}: {str(e)}")

    firestore_batch.commit()

    return batch_summary

def pubsub_triggered_worker(request):
    if request.method != 'POST':
        return 'Method not allowed', 405

    try:
        data = request.get_json()
        if not data or 'message' not in data:
            raise ValueError("Invalid Pub/Sub message format.")

        pubsub_message = data['message']

        if 'data' in pubsub_message:
            message_data = base64.b64decode(pubsub_message['data']).decode('utf-8')
            payload = json.loads(message_data)
        else:
            payload = {}

        batch_refs = payload.get('file_refs', [])

        if not batch_refs:
             return 'No file references found in the Pub/Sub payload.', 200

        print(f"Starting batch job for {len(batch_refs)} files...")

        summary = process_data_batch(batch_refs)

        print(f"Batch Job Summary ({summary['job_id']}):")
        print(json.dumps(summary, indent=2))

        if summary["errors"]:
                    return f"Job completed with {len(summary['errors'])} errors. Summary:
{summary['job_id']}", 500
        else:
           return f"Batch job successful. Documents created: {summary['documents_created']}",
200

    except Exception as e:
        print(f"Fatal error during worker execution: {e}")
        return f"Worker execution failed: {str(e)}", 500

```

A.5: Exploratory Data Analysis (EDA) Script

The full script used to generate the pairplots (Figures 4-14 and 4-15) and perform outlier removal using the Interquartile Range (IQR) method is consolidated here for reference. This script utilizes libraries such as seaborn and matplotlib for visualization and pandas for data manipulation.

```
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import numpy as np # Import numpy for percentile calculations

def generate_pair_plots(filepath, remove_outliers=True):
    """
    Reads production data, splits it by well bore name, and generates
    pair plots for numeric columns for each well.

    Args:
        filepath (str): The path to the CSV or Excel file containing the production data.
        remove_outliers (bool): Whether to remove outliers using the IQR method. Defaults to True.
    """
    try:
        # Load the dataset
        # Use read_excel for .xlsx files, and read_csv for .csv files
        if filepath.lower().endswith('.xlsx'):
            df = pd.read_excel(filepath)
        elif filepath.lower().endswith('.csv'):
            df = pd.read_csv(filepath)
        else:
            print(f"Error: Unsupported file format for {filepath}. Please provide a .csv or .xlsx file.")
            return

        print("Successfully loaded the dataset.")
    except FileNotFoundError:
        print(f"Error: The file was not found at {filepath}")
        return
    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
        return

    # --- Data Cleaning and Preparation ---

    # Select columns that are likely to be numeric and interesting for analysis
    # We are excluding columns that might be identifiers or have a constant value.
    numeric_cols = [
        'ON_STREAM_HRS',
        'AVG_DOWNHOLE_PRESSURE',
        'AVG_DOWNHOLE_TEMPERATURE',
        'AVG_DP_TUBING',
        'AVG_ANNULUS_PRESS',
        'AVG_WHP_P',
        'AVG_WHT_P',
        'BORE_OIL_VOL',
        'BORE_GAS_VOL',
        'BORE_WAT_VOL'
    ]

    # Convert selected columns to numeric, coercing errors into NaN (Not a Number)
    initial_numeric_cols = numeric_cols[:] # Create a copy to iterate over while modifying the original list
    for col in initial_numeric_cols:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')
        else:
              print(f"Warning: Column '{col}' not found in the dataframe. Removing from numeric_cols.")
              numeric_cols.remove(col)

    # Get the unique well bore names from the dataset
    if 'NPD_WELL_BORE_NAME' not in df.columns:
       print("Error: 'NPD_WELL_BORE_NAME' column not found in the dataframe. Cannot proceed.")
        return

    well_bore_names = df['NPD_WELL_BORE_NAME'].unique()
    print(f"Found {len(well_bore_names)} unique well bores: {', '.join(well_bore_names)}")

    # --- Plotting ---
```

```
69.
70.     # Loop through each unique well bore name
71.     for well_name in well_bore_names:
72.         print(f"\nProcessing data for well: {well_name}")
73.
74.         # Create a separate dataframe for the current well
75.         well_df = df[df['NPD_WELL_BORE_NAME'] == well_name].copy()
76.
77.         # Drop rows where any of the numeric columns have missing values
78.         # This is important for the pairplot function
79.         initial_rows_before_na_drop = well_df.shape[0]
80.         well_df.dropna(subset=numeric_cols, inplace=True)
81.         rows_removed_by_na = initial_rows_before_na_drop - well_df.shape[0]
82.           print(f"Removed {rows_removed_by_na} rows with missing numeric data for well:
{well_name}.")
83.
84.         # --- Outlier Removal (using IQR) ---
85.         if remove_outliers:
86.             Q1 = well_df[numeric_cols].quantile(0.25)
87.             Q3 = well_df[numeric_cols].quantile(0.75)
88.             IQR = Q3 - Q1
89.
90.             # Define bounds for outliers
91.             lower_bound = Q1 - 1.5 * IQR
92.             upper_bound = Q3 + 1.5 * IQR
93.
94.             # Create a boolean mask to identify outliers
95.              # A data point is an outlier if any of its numeric values are outside the IQR
bounds
96.             outlier_mask = ((well_df[numeric_cols] < lower_bound) | (well_df[numeric_cols] >
upper_bound)).any(axis=1)
97.
98.             # Remove rows identified as outliers
99.             initial_rows = well_df.shape[0]
100.            well_df = well_df[~outlier_mask].copy()
101.            removed_rows = initial_rows - well_df.shape[0]
102.
103.            print(f"Removed {removed_rows} outliers for well: {well_name} using IQR.")
104.        else:
105.            print(f"Skipping outlier removal for well: {well_name}.")
106.
107.        # Check if there is any data left to plot after cleaning and outlier removal
108.        if well_df.empty or well_df[numeric_cols].shape[0] < 2 or len(numeric_cols) < 2:
109.              print(f"Skipping plot for '{well_name}' due to insufficient data or numeric
columns after cleaning and filtering.")
110.            continue
111.
112.        print(f"Generating pair plot for {well_name}...")
113.
114.        # Create the pair plot
115.        # A pair plot shows scatter plots for relationships between variables
116.        # and histograms on the diagonal for the distribution of each variable.
117.        try:
118.            pair_plot = sns.pairplot(well_df[numeric_cols])
119.
120.            # Add a title to the entire plot
121.            title = f"Relationships Between Production Variables for Well: {well_name}"
122.            if not remove_outliers:
123.                title += " (Outliers Included)"
124.            pair_plot.fig.suptitle(title, y=1.02)
125.
126.            # Display the plot
127.            plt.show()
128.        except Exception as e:
129.            print(f"An error occurred while generating pair plot for {well_name}: {e}")
130.            continue
131.
132. if __name__ == '__main__':
133.     # The user should replace this filepath with the actual path to their data file.
134.     # Using the uploaded file name.
```

```
135.     daily_data_filepath = 'Production data for Saturn Field.xlsx'
136.     # To show the plot without outlier removal, set remove_outliers to False
137.     generate_pair_plots(daily_data_filepath, remove_outliers=False)
138.
```

Appendix B

Sample Data Snippets

This appendix provides text-based snippets of data at various stages of the processing pipeline. This complements the screenshots in Chapter 4 and provides a clearer, machine-readable view of the data structures.

B.1: Raw .grdecl File Sample

A representative sample from a .grdecl file, demonstrating the keyword-driven and N*Value (e.g., 99372*0.128922) format that the script in Appendix A.1 is designed to parse.

```
-- Sample GRDECL Data
-- This is a comment line
SPECGRID
16 100 16 /

COORDSYS
...
/

-- Porosity values for all cells
PORO
99372*0.128922  -- This represents 99372 cells with a porosity of 0.128922
12*0.0
4*0.128922
...
/

-- Permeability in the X direction
PERMX
10000*150.0
...
/

-- Active cell flags
ACTNUM
20000*1
15000*0
...
/

```

B.2: Sample Output UFM JSON (from .grdecl)

The corresponding JSON output from the script in Appendix A.1, based on the snippet in Figure 4-2. This shows the hierarchical structure created to store the extracted grid dimensions and properties.

```
{
  "grid_dimensions": {
    "NX": 16,
    "NY": 100,
    "NZ": 16
  },
  "properties": {
    "porosity": {
      "values": [
        0.0,
        0.0,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        0.128922,
        "..... (values continue for 99372 entries)"
      ],
      "status": "Extracted 99372 values"
    },
    "permeability": {
      "values": [],
      "status": "Extracted 0 values"
    },
    "saturation": {
      "values": [],
      "status": "Extracted 0 values"
    }
  },
  "notes": "This JSON contains all extracted property data."
}
```


B.3: Sample Output UFM JSON (from .xlsx)

This snippet, based on Figures 4-3 and 4-4, shows the final JSON structure for the production dataset. Each row from the spreadsheet is converted into a nested JSON object, keyed by row_..., which is the format uploaded to Firestore.


```
{
  "production_data": {
    "row_0": {
      "DATEPRD": "2014-04-07 00:00:00",
      "WELL_BORE_CODE": "NO 15/9-F-1 C",
      "NPD_WELL_BORE_CODE": 7405,
      "...": "..."
    },
    "row_1": {
      "DATEPRD": "2014-04-08 00:00:00",
      "WELL_BORE_CODE": "NO 15/9-F-1 C",
      "NPD_WELL_BORE_CODE": 7405,
      "...": "..."
    },
    "row_2": {
```

```
      "DATEPRD": "2014-04-09 00:00:00",
      "WELL_BORE_CODE": "NO 15/9-F-1 C",
      "NPD_WELL_BORE_CODE": 7405,
      "...": "..."
    },
    "row_3": {
      "DATEPRD": "2014-04-10 00:00:00",
      "WELL_BORE_CODE": "NO 15/9-F-1 C",
      "NPD_WELL_BORE_CODE": 7405,
      "NPD_WELL_BORE_NAME": "15/9-F-1 C",
      "NPD_FIELD_CODE": 3420717,
      "NPD_FIELD_NAME": "VOLVE",
      "NPD_FACILITY_CODE": 369304,
      "NPD_FACILITY_NAME": "MÆRSK INSPIRER",
      "ON_STREAM_HRS": 0.0,
      "AVG_DOWNHOLE_PRESSURE": null,
      "AVG_DOWNHOLE_TEMPERATURE": null,
      "AVG_DP_TUBING": null,
      "AVG_ANNULUS_PRESS": null,
      "AVG_CHOKE_SIZE_P": 0.5457591666666667,
      "AVG_CHOKE_UOM": "%",
      "AVG_WHP_P": 0.0,
      "AVG_WHT_P": 0.0,
      "DP_CHOKE_SIZE": 0.0,
      "BORE_OIL_VOL": 0.0,
      "BORE_GAS_VOL": 0.0,
      "BORE_WAT_VOL": 0.0,
      "BORE_WI_VOL": null
    },
    "row_4": {
      "DATEPRD": "2014-04-11 00:00:00",
      "...": "..."
    }
  }
}
```

Appendix C

Data Standardization

This appendix details the data governance and standardization rules applied to the pilot dataset. It provides a full mapping of original column names to the standardized UFM schema and a data dictionary defining the final fields.

C.1: Full Column Standardization and Mapping

The following table extends Figure 4-16, providing a comprehensive mapping of all relevant columns identified in the Saturn Field pilot production dataset to their new, standardized UFM names.

Old Name (Original)	New Name (Standardized)
DATEPRD	Production Date
WELL_BORE_CODE	Well Bore Code
NPD_WELL_BORE_CODE	Numeric Well Bore Code
NPD_WELL_BORE_NAME	Well Bore Name
NPD_FIELD_CODE	Field Code
NPD_FIELD_NAME	Field Name
NPD_FACILITY_CODE	Facility Code
NPD_FACILITY_NAME	Facility Name
ON_STREAM_HRS	On-Stream Hours
AVG_DOWNHOLE_PRESSURE	Average Downhole Pressure (psig)
AVG_DOWNHOLE_TEMPERATURE	Average Downhole Temperature (°F)
AVG_DP_TUBING	Average Differential Pressure Across Tubing (psi)
AVG_ANNULUS_PRESS	Average Annulus Pressure (psi)
AVG_CHOKE_SIZE_P	Average Choke Size (P-10)
AVG_CHOKE_UOM	Average Choke Size Unit of Measure (%)
AVG_WHP_P	Average Wellhead Pressure (psig)
AVG_WHT_P	Average Wellhead Temperature (°F)
DP_CHOKE_SIZE	Differential Pressure Choke Size (psi)
BORE_OIL_VOL	Bore Oil Volume (m^3)
BORE_GAS_VOL	Bore Gas Volume (Sm^3)
BORE_WAT_VOL	Bore Water Volume (m^3)
BORE_WI_VOL	Bore Water Injection Volume (m^3)
FLOW_KIND	Flow Kind
WELL_TYPE	Well Type

Table C1-1: Column mapping

C.2: Data Dictionary for UFM

This dictionary defines the data type and description for each standardized field within the Unified Field Model, ensuring clarity for future researchers and analysts.

Standardized Field	Data Type	Description
Production Date	Datetime	The specific date and time of the production record, standardized to ISO 8601 format.
Well Bore Code	String	The unique identifier for the wellbore. Mapped from WELL_BORE_CODE and NPD_WELL_BORE_CODE.
Well Bore Name	String	The common name of the wellbore (e.g., "15/9-F-1 C").
Field Code	String	The unique identifier code for the oil field.
Field Name	String	The common name of the oil field (e.g., "VOLVE").
Facility Code	String	The unique identifier code for the processing facility.
Facility Name	String	The common name of the facility (e.g., "MÆRSK INSPIRER").
On-Stream Hours	Float	The number of hours the well was operational and producing on the given date.
Average Downhole Pressure (psig)	Float	The average pressure measured at the bottom of the well, in pounds per square inch (gauge).
Average Downhole Temperature (°F)	Float	The average temperature measured at the bottom of the well, in degrees Fahrenheit.
Average DP Across Tubing (psi)	Float	The average differential pressure measured across the production tubing, in pounds per square inch.
Average Annulus Pressure (psi)	Float	The average pressure in the annulus of the well, in pounds per square inch.
Average Choke Size (%)	Float	The average choke valve opening size, expressed as a percentage or P-10.
Average Wellhead Pressure (psig)	Float	The average pressure measured at the wellhead, in pounds per square inch (gauge).
Average Wellhead Temperature (°F)	Float	The average temperature measured at the wellhead, in degrees Fahrenheit.

Bore Oil Volume (m^3)	Float	The volume of oil produced from the bore on the given date, in cubic meters.
Bore Gas Volume (Sm^3)	Float	The volume of gas produced from the bore on the given date, in standard cubic meters.
Bore Water Volume (m^3)	Float	The volume of water produced from the bore on the given date, in cubic meters.
Bore Water Injection Volume (m^3)	Float	The volume of water injected into the well on the given date, in cubic meters. null if not an injection well.
Flow Kind	String	A categorical value describing the type of flow (e.g., "production", "injection").
Well Type	String	A categorical value describing the type of well (e.g., "OP" - Oil Producer, "WI" - Water Injector).

Table C2-1: Data type dictionary