Data Engineering 2: Big Data Architectures

Project Title: Prototype of Data Pipeline

By: Kshema Iliger, Spoorti Basarkod Math

TABLE OF CONTENTS

ABSTRACT

INTRODUCTION

In today's data-driven world, the ability to efficiently process and analyze large volumes of data is crucial for gaining actionable insights. As the amount of data generated continues to grow exponentially, organizations face the challenge of transforming raw data into meaningful information that can drive strategic decisions. This project focuses on developing a prototype data pipeline using the ETL (Extract, Transform, Load) process on a cloud platform to address these challenges. Specifically, the project targets the analysis of real-time petrol prices across various cities in Uttar Pradesh, a region where petrol price fluctuations significantly impact consumers, businesses, and policymakers. The primary challenge addressed by this project is the efficient and scalable handling of diverse and dynamic data sources to facilitate real-time analysis and decision-making. Petrol prices can change frequently due to a variety of factors, including supply and demand dynamics, geopolitical events, and regulatory changes. Traditional methods of tracking petrol prices often involve manual data collection and static reporting, which are not only time-consuming but also prone to inaccuracies and delays. This hampers the ability of stakeholders to respond swiftly to market changes, making it imperative to have a more automated, reliable, and real-time data processing solution.

APPLICATION PROBLEM

Petrol price variations that are frequent and unpredictable present substantial issues for policymakers and consumers in Uttar Pradesh. These differences complicate the process of making well-informed decisions by having an impact on household finances, business operations, and economic planning. The current petrol price tracking and analysis systems are frequently antiquated, ineffective, and unable to provide real-time data. In order to provide stakeholders with timely and data-driven decision-making, this project addresses the requirement for a scalable and effective system to continually collect, process, and visualize real-time petrol pricing data across many cities.

SUMMARY OF OWN APPROACH

Our approach involves a systematic and structured ETL process designed to harness the power of cloud computing. The extraction phase utilizes a Python script to make API calls, fetching up-to-date petrol price data for multiple cities in Uttar Pradesh. This data, characterized by its variability and volume, is ingested into Google Cloud's BigQuery, a robust and scalable data warehouse solution. In BigQuery, the data undergoes transformation processes to ensure it is clean, normalized, and aggregated appropriately. This involves removing inconsistencies, handling missing values, and performing necessary calculations to derive meaningful metrics such as average prices and price changes over time. Once the data is transformed and stored in an accessible format, the final step of the pipeline involves visualization. We leverage Tableau, a powerful data visualization tool, to create interactive dashboards that allow users to explore the data in a dynamic and intuitive manner. These dashboards present petrol price trends and fluctuations across different cities, providing stakeholders with valuable insights into regional price dynamics. The visualizations are designed to be user-friendly, offering various filtering options to drill down into specific time periods or compare prices between different cities.

SUMMARY OF OWN RESULTS

The results of this project demonstrate the effectiveness and efficiency of a cloud-based ETL pipeline in managing and analyzing large datasets. The implementation of the ETL process on Google Cloud's BigQuery ensures that the data is processed in a scalable and reliable environment, capable of handling large volumes of real-time data with minimal latency. The interactive Tableau dashboards offer a comprehensive view of petrol price trends, highlighting significant increases and decreases across the region. These visualizations not only facilitate a deeper understanding of petrol price dynamics but also enable stakeholders to make informed decisions based on real-time data. Furthermore, this project highlights the potential for expanding the scope of the data pipeline to include additional regions or integrate more complex analytical models. Future work could involve incorporating historical data to perform trend analysis over extended periods, as well as implementing predictive analytics to forecast future petrol prices based on current trends and external factors. By doing so, the data pipeline could provide even more valuable insights, helping to anticipate market changes and inform policy decisions.

CHAPTER 1: INTRODUCTION

DESCRIPTION OF APPLICATION DOMAIN

Fuel prices are a critical component of the economy, affecting transportation costs, inflation rates, and overall economic stability. In countries like India, where transportation heavily relies on fuel, even small fluctuations in fuel prices can have a significant impact on the cost of goods and services. Accurate and timely information on fuel prices is essential for various stakeholders, including government agencies, businesses, and consumers.The transportation sector is particularly sensitive to changes in fuel prices. For logistics companies, fuel costs represent a significant portion of their operating expenses. Similarly, public transportation services, which many people rely on for their daily commute, are directly affected by fuel price variations. Thus, having up-to-date information on fuel prices helps these entities plan their operations more effectively.Furthermore, consumers also benefit from knowing the current fuel prices. It allows them to budget their expenses and make informed decisions about travel and consumption. Policymakers can use this data to understand market trends and make regulatory decisions that can help stabilize the economy.

PROBLEM ON APPLICATION LEVEL

The main problem at the application level is the lack of accessible and up-to-date information on fuel prices. Traditional methods of collecting and disseminating fuel price information are often slow and prone to errors. For example, manually tracking prices from different sources can lead to outdated or incomplete data. Additionally, there is often a lack of a centralized platform where this information can be easily accessed and analyzed. Without timely and accurate data, businesses may face challenges in managing their fuel expenses, and consumers may not be able to optimize their fuel purchases. Policymakers may also struggle to make informed decisions due to a lack of reliable data.

BENEFITS OF A SOLUTION ON APPLICATION LEVEL

Implementing an automated data pipeline for retrieving and processing real-time fuel prices offers several benefits:

1. **Timeliness**: Automated systems can retrieve and process data in real-time, ensuring that users always have access to the latest information.
2. **Accuracy**: By eliminating manual data entry, the likelihood of errors is significantly reduced, leading to more reliable data.

3. **Efficiency**: Automation reduces the time and effort required to collect and process data, freeing up resources for other tasks.
4. **Accessibility**: A centralized platform makes it easier for stakeholders to access and analyze the data they need.

PROBLEM ON TECHNICAL LEVEL

The technical challenges of setting up a robust data pipeline for fuel prices involve several aspects:

1. **Data Integration**: Integrating with APIs to fetch real-time data can be complex. Ensuring that the API connection is stable and handles errors gracefully is crucial.
2. **Data Accuracy**: Ensuring the accuracy of the data received from the API is vital. This involves validating the data and handling any discrepancies.
3. **Data Transformation**: The raw data from the API may need to be transformed into a format suitable for analysis. This includes cleaning the data, handling missing values, and standardizing units.
4. **Data Loading**: Efficiently loading the transformed data into a data warehouse like BigQuery requires careful planning and optimization to handle large volumes of data.
5. **Data Visualization**: Creating meaningful and interactive visualizations that convey the insights effectively requires expertise in tools like Tableau.
6. **Error Handling and Monitoring**: Implementing robust error handling and monitoring mechanisms to ensure the pipeline runs smoothly and any issues are promptly addressed.

TECHNICAL SOLUTION IDEA

The proposed solution involves developing a comprehensive data pipeline using ETL (Extract, Transform, Load) processes. Here’s a brief overview of each step:

1. **Extraction**: Data is extracted from the fuel price API using Python. This involves setting up an API client, making requests to the API, and handling the responses. Error handling mechanisms are implemented to manage any issues with the API connectivity or data retrieval.

2. **Transformation**: The extracted data undergoes several transformation steps. This includes cleaning the data to remove any inconsistencies, converting units to a standard format, and adding any necessary derived fields. Python scripts are used to perform these transformations.
3. **Loading**: The transformed data is then loaded into BigQuery, a cloud-based data warehouse. This step involves creating appropriate tables in BigQuery and using optimized methods to load large volumes of data efficiently.
4. **Visualization**: Once the data is available in BigQuery, it is connected to Tableau for visualization. Various dashboards and reports are created in Tableau to provide insights into fuel price trends across different cities in Uttar Pradesh. These visualizations help users quickly understand the data and make informed decisions.

This ETL pipeline ensures that the fuel price data is always up-to-date, accurate, and accessible for analysis and decision-making.

CHAPTER 2: RELATED WORK

TRADITIONAL DATA WAREHOUSING

Manual Data Collection and Static Reporting: Earlier, organizations relied heavily on manual data collection processes followed by static reporting using spreadsheets and traditional databases. While this method ensured data was recorded, it was inefficient, error-prone, and lacked real-time capabilities.

ON-PREMISE ETL TOOLS

ETL Software Solutions: Companies have used on-premise ETL tools like Informatica, Talend, and Apache Nifi to automate the extraction, transformation, and loading of data. These tools provided a structured way to handle data from multiple sources but often required significant hardware resources and were not easily scalable.

CLOUD-BASED DATA WAREHOUSING

AWS Redshift, Azure Synapse Analytics, and Google BigQuery: With the advent of cloud computing, many organizations transitioned to cloud-based data warehousing solutions. These platforms offer scalable storage and compute resources, enabling the handling of large volumes of data with better performance and lower costs. They also support real-time data ingestion and querying capabilities.

REAL-TIME DATA STREAMING AND PROCESSING

Apache Kafka and Apache Spark: For real-time data processing, technologies like Apache Kafka (for data streaming) and Apache Spark (for real-time data processing) have been widely adopted. Kafka acts as a distributed streaming platform, handling high-throughput data ingestion, while Spark processes this data in real-time, making it suitable for applications needing immediate insights.

API INTEGRATION FOR REAL-TIME DATA

Python and RESTful APIs: Similar to our approach, many solutions involve using programming languages like Python to automate API calls for real-time data extraction. Libraries such as requests in Python facilitate easy interaction with web APIs, enabling the continuous fetching of live data.

DATA VISUALIZATION PLATFORMS

Tableau, Power BI, and Looker: For visualization, platforms like Tableau, Power BI, and Looker provide powerful tools to create interactive dashboards. These tools are integrated with data warehouses and real-time data streams to offer dynamic and up-to-date visual representations of data.

CHAPTER 3: DATASET

DESCRIPTION OF DATA SOURCE

The primary data source for this project is the API provided by the "Petrol and Diesel Price in India" service. This API offers real-time information on fuel prices across various cities in India, including Uttar Pradesh. The API endpoint used for this project is:
API URL: Petrol and Diesel Price in India
The API provides a comprehensive dataset that includes the following key attributes for each city:

1. City: The name of the city where the fuel price is being reported.
2. Petrol Price: The current price of petrol in the city.
3. Price Variation: The change in petrol price compared to the previous report.

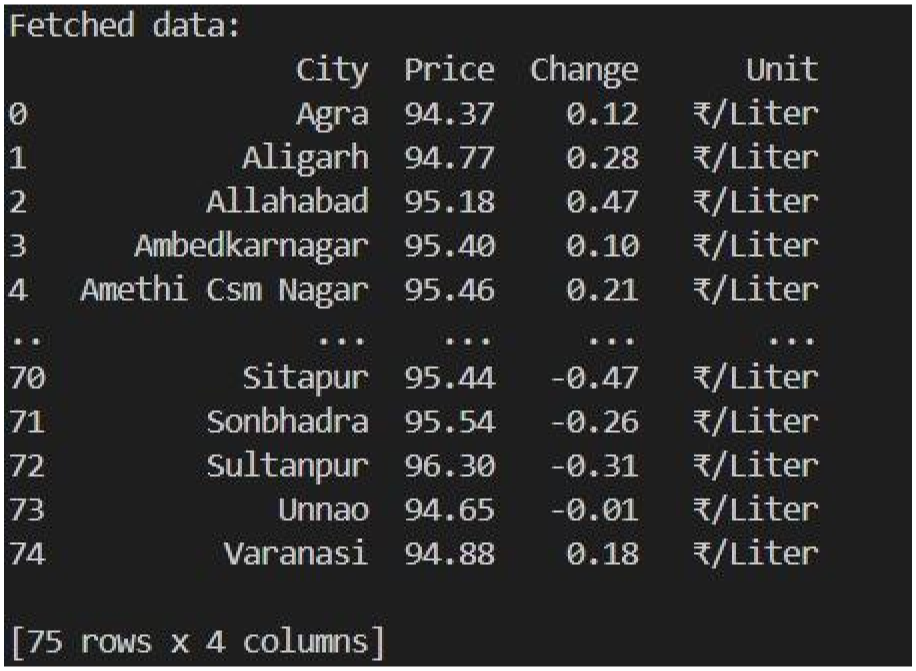
4. Unit: The unit of measurement for the petrol price (typically per liter).

DATA COLLECTION METHOD

The data collection process involves extracting data from the API using Python. The following steps outline the method used:

1. API Integration:
    - A Python script is developed to interact with the API. This script makes HTTP GET requests to the API endpoint to retrieve the latest fuel prices for all cities in Uttar Pradesh.
    - The API requires an API key for authentication, which is included in the request headers.
2. Handling API Responses:
    - The API responses are received in JSON format. The Python script parses the JSON data to extract relevant attributes such as city name, petrol price, price variation, and unit.
    - Error handling mechanisms are implemented to manage any issues with API connectivity or data retrieval. This includes retry logic for failed requests and logging errors for further analysis.
3. Data Cleaning and Preprocessing:
    - Once the data is extracted, it undergoes a series of cleaning and preprocessing steps. This includes:
        - Removing Duplicates: Ensuring that there are no duplicate entries in the dataset.
        - Handling Missing Values: Addressing any missing values in the dataset by either filling them with appropriate defaults or discarding incomplete records.
        - Standardizing Units: Ensuring that the price units are consistent across all records for accurate comparison and analysis.
4. Storing Data:
    - The cleaned and preprocessed data is then stored in Google BigQuery, a cloud-based data warehouse. This is done using the BigQuery API, which allows for efficient data loading and storage.
    - Appropriate schemas and tables are created in BigQuery to store the fuel price data, ensuring that the data is organized and easily accessible for analysis.

By automating the data collection process, we ensure that the dataset is continuously updated with the latest information, providing an accurate and up-to-date view of fuel prices across different cities in Uttar Pradesh.



```
Fetched data:
                City  Price  Change      Unit
0               Agra  94.37    0.12  ₹/Liter
1            Aligarh  94.77    0.28  ₹/Liter
2          Allahabad  95.18    0.47  ₹/Liter
3      Ambedkarnagar  95.40    0.10  ₹/Liter
4   Amethi Csm Nagar  95.46    0.21  ₹/Liter
..               ...    ...     ...       ...
70           Sitapur  95.44   -0.47  ₹/Liter
71         Sonbhadra  95.54   -0.26  ₹/Liter
72         Sultanpur  96.30   -0.31  ₹/Liter
73             Unnao  94.65   -0.01  ₹/Liter
74          Varanasi  94.88    0.18  ₹/Liter

[75 rows x 4 columns]
```

Fig: Data fetched from API call.

CHAPTER 4: SOLUTION

APPROACH 1

Initial Data Pipeline using GCP Pub/Sub

In the initial phase of our project, we aimed to construct a robust and scalable data pipeline using Google Cloud Platform (GCP) Pub/Sub. Pub/Sub is a messaging service that allows asynchronous communication between independent applications. Our plan was to utilize Pub/Sub for publishing and subscribing to real-time fuel price data, facilitating seamless data flow from the API to BigQuery.
Implementation Steps:

1. Data Extraction: We developed a Python script to extract fuel price data from the API. This script was designed to make regular API calls and retrieve the latest data on petrol prices across various cities in Uttar Pradesh.
2. Publishing Data to Pub/Sub: The extracted data was then published to a Pub/Sub topic. Each piece of data was encapsulated in a message and sent to the topic, enabling other services to subscribe and receive updates in real-time.

3. Subscribing and Loading Data into BigQuery: A Pub/Sub subscription was created to listen to the topic. This subscriber service was intended to receive the published messages, process the data, and load it into BigQuery for further analysis.

Challenges Encountered: During the implementation of this approach, we encountered significant challenges, particularly related to permissions and service account configurations. Despite thorough configuration attempts, we faced persistent permission errors when trying to write data from the Pub/Sub subscriber to BigQuery. These errors were primarily due to insufficient permissions on the service account used by the Pub/Sub subscriber.

Error Resolution Attempts:

- Permission Adjustments: We attempted to modify the service account permissions to grant the necessary roles for accessing and writing to BigQuery. However, these changes did not resolve the issues.
- Service Account Configuration: Various configurations of the service account were tested, including creating new service accounts with different sets of permissions. Unfortunately, none of these configurations allowed successful data writing to BigQuery.

Conclusion: After multiple attempts to troubleshoot and resolve the permission issues, we determined that the approach using GCP Pub/Sub was not feasible within the scope of our project. The persistent errors and the time required to resolve them led us to rule out this method. Instead, we decided to explore alternative approaches that could provide a more straightforward and reliable solution for our data pipeline needs.

This experience highlighted the importance of thorough permission and configuration management in cloud-based services and guided us in selecting more suitable technologies for our subsequent attempts.

APPROACH 2

Using Dataflow and Cloud Storage

Following the challenges faced with the initial GCP Pub/Sub approach, we explored a second approach leveraging Google Cloud Dataflow and Cloud Storage. Dataflow is a fully managed service for stream and batch processing, which we intended to use for moving data from Pub/Sub to BigQuery, with intermediate storage in Cloud Storage.

Implementation Steps:

- Data Extraction and Publishing:

    - Similar to the first approach, a Python script was used to extract fuel price data from the API and publish it to a Pub/Sub topic. This setup ensured the continuity of data retrieval and publication.
- Dataflow Pipeline Setup:
    - A Dataflow pipeline was configured to subscribe to the Pub/Sub topic, process the incoming data, and write it to BigQuery. Dataflow provides the capability to handle complex data processing tasks, making it suitable for our requirements.
    - The pipeline was designed to read messages from the Pub/Sub subscription, parse the JSON data, transform it as needed, and then write the results to BigQuery.
- Intermediate Storage in Cloud Storage:
    - To enhance reliability and facilitate troubleshooting, intermediate data storage was implemented using Google Cloud Storage. The data extracted from Pub/Sub was first written to Cloud Storage in JSON format before being processed by Dataflow and loaded into BigQuery.

CHALLENGES ENCOUNTERED

Despite the improved design, we faced significant challenges during the implementation of this approach.

1. Writing to BigQuery:
    - Similar to the previous approach, we encountered errors when attempting to write data to BigQuery. The Dataflow jobs consistently failed with permission errors, despite extensive troubleshooting and permission adjustments on the service accounts.
2. Schema Issues:
    - Another major challenge was defining and writing the correct schema for the JSON data in BigQuery. The schema needs to match the structure of the JSON data precisely, but discrepancies between the JSON structure and the BigQuery schema led to persistent errors.
    - Efforts to dynamically generate and apply the schema within the Dataflow pipeline were met with limited success. The complexity of the JSON data and the need for precise schema definitions proved to be a significant hurdle.

ERROR RESOLUTION ATTEMPTS

- Permission Adjustments: Multiple iterations of adjusting the service account permissions were attempted, similar to the first approach. This included granting extensive roles and permissions to ensure Dataflow could write to BigQuery.
- Schema Validation: Various methods to validate and correct the schema were explored. This included manually defining the schema in BigQuery, using schema inference tools, and modifying the JSON data structure to align with BigQuery's requirements.

Conclusion: After extensive efforts to troubleshoot the issues with writing to BigQuery and handling schema definitions, it became evident that this approach also faced significant limitations. The persistent errors and the complexity involved in managing the schema led us to abandon this method.

These challenges highlighted the need for a more streamlined and less error-prone solution for our data pipeline. Consequently, we decided to explore other alternatives that could provide a more reliable and straightforward path to achieving our project goals.

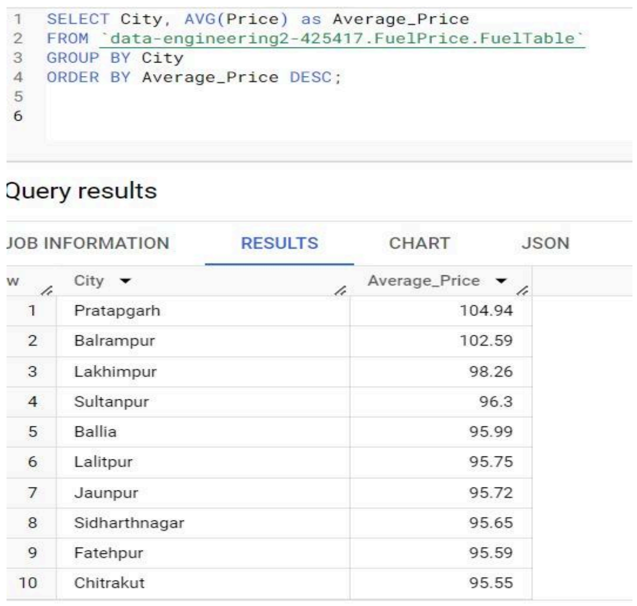
Approach 3: Successful Implementation Using Python, BigQuery, and Tableau

After facing challenges with the previous approaches involving GCP Pub/Sub and Dataflow, we successfully implemented a data pipeline using Python for data extraction, Google BigQuery for data warehousing, and Tableau for data visualization. This approach proved to be effective and reliable for our use case.

Implementation Steps:

1. Data Extraction and API Integration:
    - A Python script was developed to extract fuel price data from the API. The API endpoint used provided detailed information on petrol prices across various cities in Uttar Pradesh.
    - The script made periodic API calls to ensure that the data was always up-to-date.
2. Data Loading into BigQuery:
    - The extracted data was transformed into a suitable format for storage in BigQuery. This involved creating a DataFrame from the JSON response and ensuring that the data types were correctly set.
    - The data was then loaded into a BigQuery table. The schema for the table included fields for city, price, price change, and unit. A function was implemented to check if the table existed and create it if it did not.
    - The Python script was configured to run in a loop, fetching new data at regular intervals and appending it to the BigQuery table.

3. Performing queries on the data:
    - SELECT City, AVG(Price) as Average_Price
      FROM data-engineering2-425417.FuelPrice.FuelTable
      GROUP BY City
      ORDER BY Average_Price DESC;



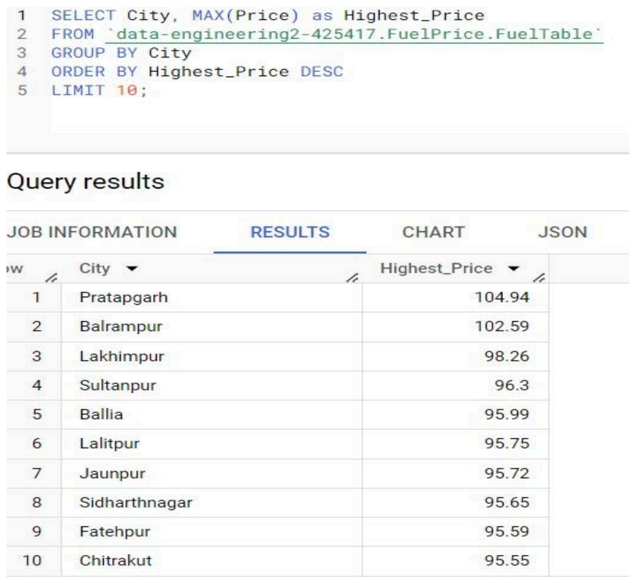
```
SELECT City, AVG(Price) as Average_Price
FROM `data-engineering2-425417.FuelPrice.FuelTable`
GROUP BY City
ORDER BY Average_Price DESC;
```

Query results

JOB INFORMATION RESULTS CHART JSON

w	City	Average_Price
1	Pratapgarh	104.94
2	Balrampur	102.59
3	Lakhimpur	98.26
4	Sultanpur	96.3
5	Ballia	95.99
6	Lalitpur	95.75
7	Jaunpur	95.72
8	Sidharthnagar	95.65
9	Fatehpur	95.59
10	Chitrakut	95.55

    - SELECT City, MAX(Price) as Highest_Price
      FROM data-engineering2-425417.FuelPrice.FuelTable
      GROUP BY City
      ORDER BY Highest_Price DESC
      LIMIT 10;



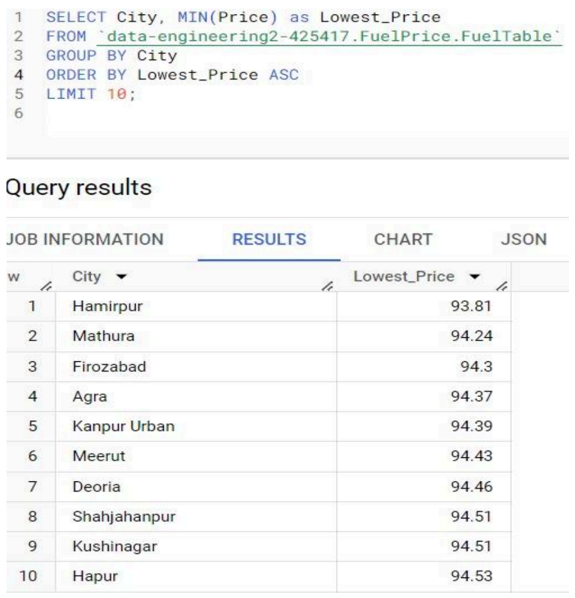
```
SELECT City, MAX(Price) as Highest_Price
FROM `data-engineering2-425417.FuelPrice.FuelTable`
GROUP BY City
ORDER BY Highest_Price DESC
LIMIT 10;
```

Query results

JOB INFORMATION RESULTS CHART JSON

w	City	Highest_Price
1	Pratapgarh	104.94
2	Balrampur	102.59
3	Lakhimpur	98.26
4	Sultanpur	96.3
5	Ballia	95.99
6	Lalitpur	95.75
7	Jaunpur	95.72
8	Sidharthnagar	95.65
9	Fatehpur	95.59
10	Chitrakut	95.55

- SELECT City, MIN(Price) as Lowest_Price
  FROM data-engineering2-425417.FuelPrice.FuelTable
  GROUP BY City
  ORDER BY Lowest_Price ASC
  LIMIT 10;

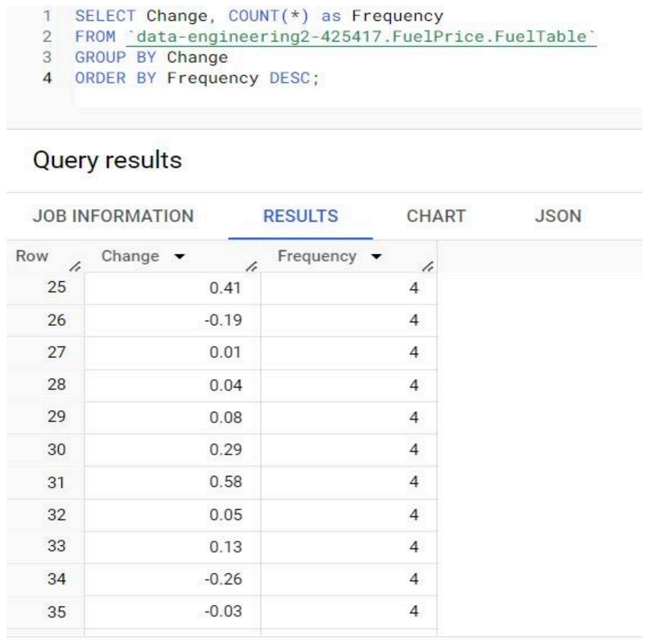


```
SELECT City, MIN(Price) as Lowest_Price
FROM `data-engineering2-425417.FuelPrice.FuelTable`
GROUP BY City
ORDER BY Lowest_Price ASC
LIMIT 10;
```

Query results

w	City	Lowest_Price
1	Hamirpur	93.81
2	Mathura	94.24
3	Firozabad	94.3
4	Agra	94.37
5	Kanpur Urban	94.39
6	Meerut	94.43
7	Deoria	94.46
8	Shahjahanpur	94.51
9	Kushinagar	94.51
10	Hapur	94.53

- SELECT Change, COUNT(*) as Frequency
  FROM `data-engineering2-425417.FuelPrice.FuelTable`GROUP BY Change
  ORDER BY Frequency DESC;

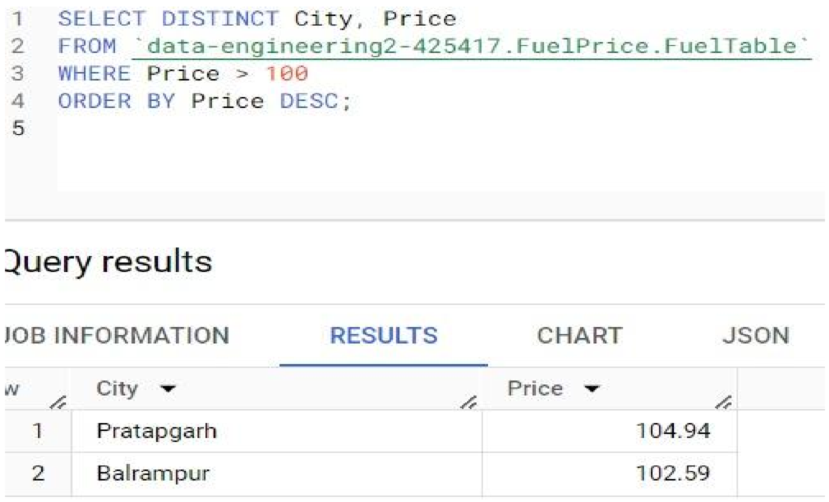


```
SELECT Change, COUNT(*) as Frequency
FROM `data-engineering2-425417.FuelPrice.FuelTable`
GROUP BY Change
ORDER BY Frequency DESC;
```

Query results

Row	Change	Frequency
25	0.41	4
26	-0.19	4
27	0.01	4
28	0.04	4
29	0.08	4
30	0.29	4
31	0.58	4
32	0.05	4
33	0.13	4
34	-0.26	4
35	-0.03	4

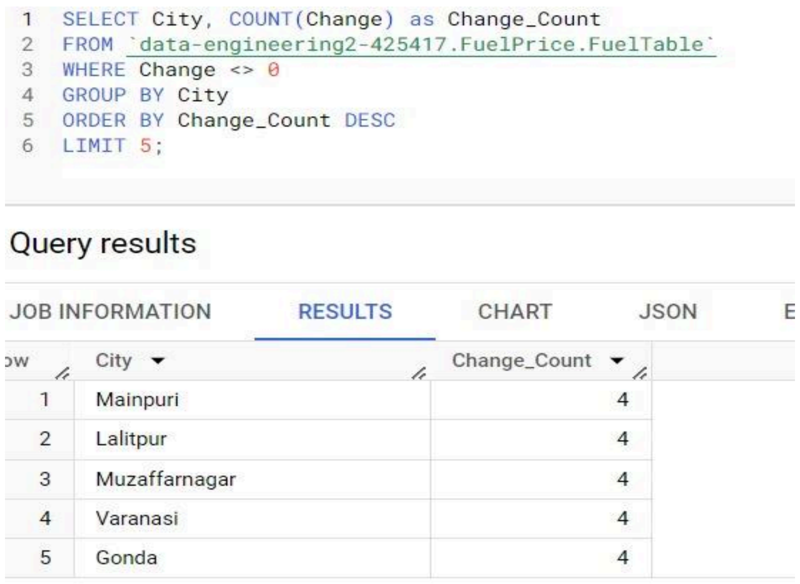
- Above 100
  SELECT DISTINCT City, Price
  FROM data-engineering2-425417.FuelPrice.FuelTable
  WHERE Price > 100
  ORDER BY Price DESC;



```
SELECT DISTINCT City, Price
FROM `data-engineering2-425417.FuelPrice.FuelTable`
WHERE Price > 100
ORDER BY Price DESC;
```

Row	City	Price
1	Pratapgarh	104.94
2	Balrampur	102.59

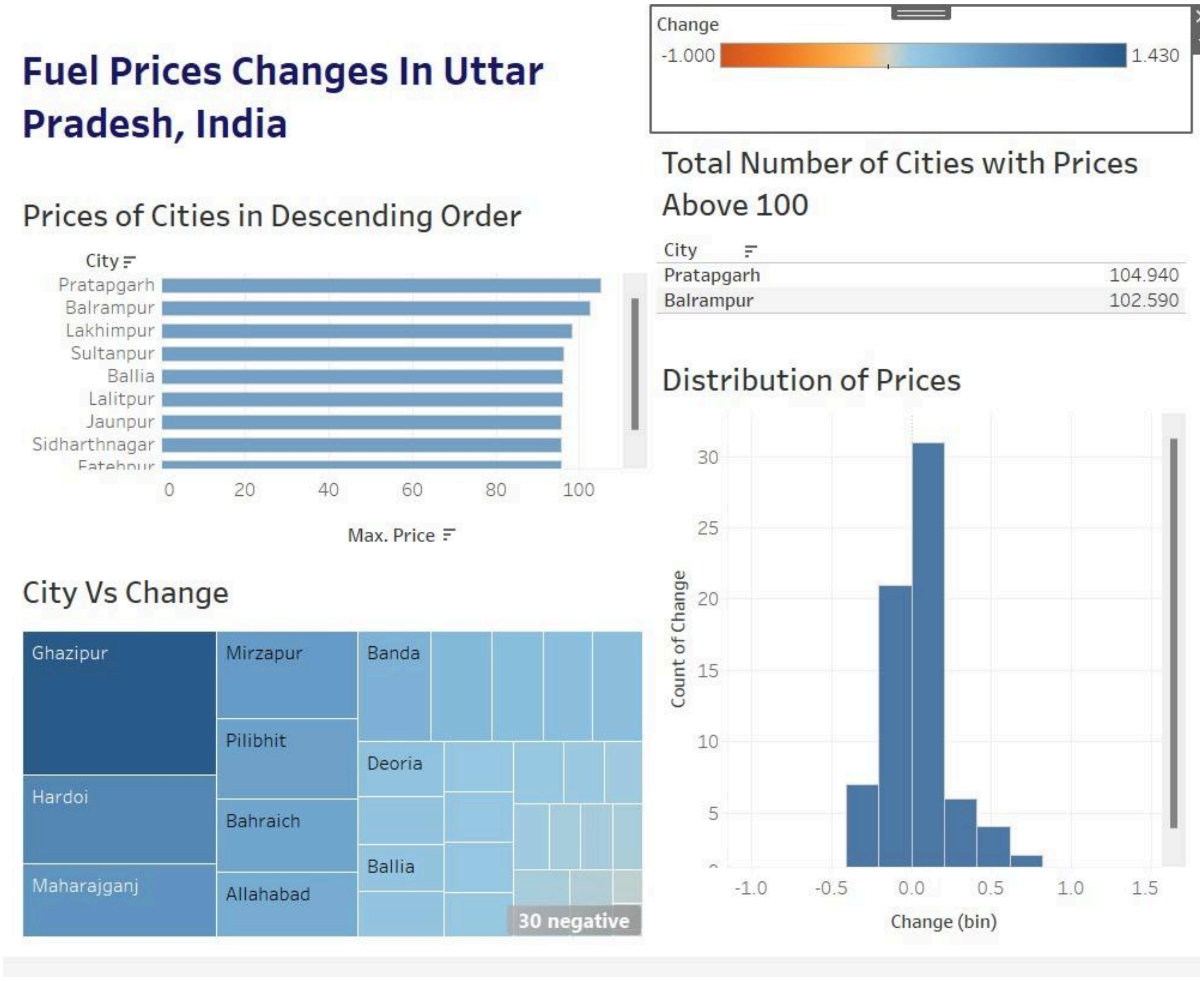
- SELECT City, COUNT(Change) as Change_Count
  FROM data-engineering2-425417.FuelPrice.FuelTable
  WHERE Change <> 0
  GROUP BY City
  ORDER BY Change_Count DESC
  LIMIT 5;



```
SELECT City, COUNT(Change) as Change_Count
FROM `data-engineering2-425417.FuelPrice.FuelTable`
WHERE Change <> 0
GROUP BY City
ORDER BY Change_Count DESC
LIMIT 5;
```

Row	City	Change_Count
1	Mainpuri	4
2	Lalitpur	4
3	Muzaffarnagar	4
4	Varanasi	4
5	Gonda	4

4. Data Visualization with Tableau:
   - BigQuery was connected to Tableau, enabling the creation of interactive dashboards and visualizations. Various visualizations were designed to display the trends and distributions of fuel prices across different cities.
   - These visualizations included bar charts showing the prices in descending order, a histogram for the distribution of price changes, and a heat map for city-wise price changes.



Total Number of Cities with Prices Above 100

City	
Pratapgarh	104.940
Balrampur	102.590

Data Visualization in Tableau: The final step was to visualize the processed data using Tableau. The visualizations provided insights into fuel price trends and changes across different cities in Uttar Pradesh. The attached screenshot showcases some of the visualizations created:

- Prices of Cities in Descending Order: A bar chart showing the maximum fuel prices in descending order.

- Total Number of Cities with Prices Above 100: A summary table listing cities with prices above ₹100.
- Distribution of Prices: A histogram showing the distribution of fuel price changes.
- City vs Change: A heat map depicting the variations in fuel prices across different cities.

These visualizations enabled stakeholders to easily interpret and analyze the fuel price data, making informed decisions based on the insights provided.

CHAPTER 5: SUMMARY AND OUTLOOK

SUMMARY

The project aimed to create a robust and efficient data pipeline for retrieving, processing, and visualizing real-time fuel price data for various cities in Uttar Pradesh, India. We experimented with multiple approaches to develop a reliable solution, and through iterative problem-solving, we successfully implemented a functional data pipeline using Python, Google BigQuery, and Tableau.

Project Overview:

1. Objective: The primary objective was to build a data pipeline that could fetch real-time fuel price data from an API, store and process this data in BigQuery, and visualize the results in Tableau. This would provide stakeholders with up-to-date insights into fuel price fluctuations across different cities.
2. Approaches:
    - Approach 1: The initial approach involved using GCP Pub/Sub to publish and subscribe to fuel price data, with the data being written to BigQuery. However, persistent permission issues led us to abandon this method.
    - Approach 2: The second approach leveraged Dataflow and Cloud Storage for ingesting data from Pub/Sub into BigQuery. Despite improving the design, we faced significant challenges with writing data to BigQuery and handling schema definitions.
    - Approach 3: The final and successful approach involved directly using Python to extract data from the API, load it into BigQuery, and then visualize it using Tableau.

Implementation Details:

- Data Extraction: A Python script was developed to make API calls and retrieve real-time fuel price data. This script was configured to run periodically, ensuring that the data remained current.

- Data Processing and Storage: The extracted data was formatted and cleaned before being loaded into BigQuery. We created a schema for the BigQuery table to store city names, fuel prices, price changes, and units. The script checked for the existence of the table and created it if necessary.
- Visualization: The data in BigQuery was connected to Tableau for visualization. Various dashboards and charts were created to depict the fuel price trends and changes, providing valuable insights to stakeholders.

Results:

- The pipeline successfully fetched, processed, and stored real-time data, ensuring that the latest information was always available for analysis.

OUTLOOK

Future Work and Improvements:

1. Enhanced Data Quality and Validation:
    - Implement additional data validation checks to ensure the accuracy and consistency of the data before it is loaded into BigQuery. This could include checks for outliers, missing values, and anomalies in the data.
    - Develop automated alerts and notifications for any discrepancies or issues detected during data extraction and processing.
2. Scalability and Performance Optimization:
    - Optimize the data pipeline to handle larger volumes of data and increased API call frequencies. This could involve parallel processing and more efficient data handling techniques.
    - Explore the use of advanced features in BigQuery, such as partitioning and clustering, to improve query performance and reduce costs.
3. Extended Data Sources:
    - Integrate additional data sources to provide a more comprehensive view of fuel prices and related factors. This could include data on diesel prices, historical fuel prices, and external factors such as crude oil prices and currency exchange rates.
    - Use machine learning models to predict future fuel price trends based on historical data and external factors.
4. User Experience and Accessibility:
    - Enhance the user interface of the Tableau dashboards to make them more intuitive and user-friendly. This could include the addition of interactive filters, drill-down capabilities, and customizable views.
    - Provide users with the ability to export data and visualizations for offline analysis and reporting.

5. Real-time Analytics and Reporting:
    - Implement real-time analytics to provide instant insights and updates on fuel price changes as they occur. This could involve streaming data processing techniques and real-time data visualization tools.
    - Develop automated reporting features to generate periodic reports on fuel price trends and changes, which can be distributed to stakeholders.
6. Security and Compliance:
    - Ensure that the data pipeline complies with relevant data protection regulations and industry standards. This could involve implementing encryption, access controls, and auditing mechanisms.
    - Regularly review and update security measures to protect the data from unauthorized access and breaches.

Long-term Vision:

- Integration with Other Systems: Integrate the data pipeline with other systems and applications used by stakeholders, such as ERP systems, financial management tools, and business intelligence platforms. This would enable seamless data flow and more comprehensive analysis.
- Geographic Expansion: Expand the scope of the data pipeline to cover fuel prices in other states and regions, providing a broader view of fuel price trends across the country.
- Community and Collaboration: Collaborate with other organizations and communities to share data and insights, fostering a collaborative approach to understanding and addressing fuel price fluctuations.

In conclusion, this project has demonstrated the potential of using modern data engineering tools and techniques to build a reliable and efficient data pipeline for real-time fuel price analysis. The successful implementation of the pipeline has provided valuable insights and has laid the foundation for future enhancements and expansions. By continually improving and scaling the solution.

SL. NO.	**TOPICS**	**AUTHORSHIP**
1.	ABSTRACT	Spoorti
2.	CHAPTER 1	Kshema
3.	CHAPTER 2	Spoorti
4.	CHAPTER 3	Kshema
5.	CHAPTER 4	Spoorti / Kshema
6.	CHAPTER 5	Kshema

BIBLIOGRAPHY

1. https://cloud.google.com/pubsub/docs

2. https://cloud.google.com/bigquery/docs

3. https://www.youtube.com/watch?v=TqvPEf5iJ2o&t=655s

4. https://rapidapi.com/blog/accordions/api-key/

5. https://cloud.google.com/pubsub/docs/samples/pubsub-create-cloud-storage-subscription?hl=en#pubsub_create_cloud_storage_subscription-python

6. https://youtu.be/Lj2XAVVhd8k?si=ARt1yDtOZNY3M166

7. https://youtu.be/faexdxtZGno?si=Uszpa0iH72DzYEKB