Telemetry Analytics on Snowflake

DISSERTATION

Submitted in partial fulfillment of the requirements of the

MTech Data Science and Engineering Degree Programme

by

GAURAV SHARMA

2021SA04043

Under the supervision of

VEDAVYAS CHINTAKUNTA

Head of Engineering

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

March 2024

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

Certificate

This is to certify that the Dissertation entitled *Telemetry Analytics on Snowflake* and submitted by Mr. GAURAV SHARMA, ID No. 2021SA04043 in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him under my supervision.

____

(Signature of the Supervisor)	Name: VEDAVYAS CHINTAKUNTA
Place: HYDERABAD	Designation: Head of Engineering
Date: 10-MARCH-2024	

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

I SEMESTER 23-24

DSE CL ZG628T **DISSERTATION**

Dissertation Outline

BITS ID No.: 2021SA04043 **Name of Student:** GAURAV SHARMA

Name of Supervisor:	VEDAVYAS CHINTAKUNTA
Designation of Supervisor:	Head of Engineering
Qualification and Experience:	19 Years
Email ID of Supervisor:	vchintakunta@ivycomptech.com
Topic of Dissertation:	Telemetry Analytics on Snowflake

Name of First Examiner:
Designation of First Examiner:
Qualification and Experience:
Email ID of First Examiner:

Name of Second Examiner:
Designation of Second Examiner:
Qualification and Experience:
Email ID of Second Examiner:

Signature of Student
Date:

Signature of Supervisor
Date:

"Torture the data, and it will confess to anything."

~ Anonymous

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

FIRST SEMESTER 2023-24

DSECLZG628T **DISSERTATION**

Dissertation Title:	Telemetry Analytics on Snowflake
Name of Supervisor:	Mr. Vedavyas Chintakunta, Head of Engineering
Name of Student:	Gaurav Sharma
ID No. of Student:	2021SA04043

Abstract

This project constitutes an exploration in telemetry analytics by integrating the Snowflake Data Cloud, renowned for its cloud-native architecture. Leveraging the SQL-based querying capabilities of Snowflake, the research delves into analytics and Data Science practices. Beyond traditional methods, this endeavor employs advanced Data Science Techniques like machine learning, tokenization and sentiment analysis, along with Neural Model. These sophisticated approaches aim to extract profound insights and discern patterns from telemetry data.

The project meticulously adheres to robust data engineering practices, facilitating data integration for a cohesive and comprehensive analysis. The core objective is to realize tangible cost benefits and optimize resource allocation through the practice of right sizing.

The research outcomes showcase cost benefits and successful right sizing, highlighting the practical applicability of the devised methodologies. To enhance accessibility and comprehension, the project employs the use of visualization techniques for the intuitive presentation of findings. Ultimately, this technical research significantly contributes to the field of SQL based Telemetry analysis, reshaping methodologies, and establishing new benchmarks in the intricate landscape of telemetry analytics.

Keywords:

Snowflake, Data Cloud, Telemetry Analytics, Cost Optimization, SVM, Syntax Aware NLP, Feed Forward Neural Network, Data Engineering, Visualization.

Acknowledgements

I wish to extend my deepest appreciation to the cornerstones of my life, whose steadfast support and encouragement sustained me throughout the demanding process of this dissertation.

At the forefront is my beloved wife **Bharti Vatwani**, and my cherished son **Vivaan Sharma**. Their boundless love, understanding, and patience were my guiding lights, providing motivation during the entire journey. In their presence, challenges felt surmountable, and accomplishments were more joyous.

A sincere debt of thanks is owed to my parents, **Smt. Neeta Sharma,** and **Shri. Ram Kishore Sharma**. Their boundless love, understanding, and patience served as unwavering guiding lights, providing constant motivation through every step of the way.

I express sincere thanks to my supervisor **Mr. Vedavyas Chintakunta**, Head of Engineering, **Ivy Software Development Services Private Limited** and to my mentor **Mr. Abhishek Ambastha**, AVP- Engineering & Architecture AI, **Star Health and Allied Insurance Co. Ltd** whose expertise, guidance, and constructive feedback proved invaluable. I am truly grateful for the positive impact they have had on both my academic and professional development.

I would like to extend my sincere thanks to my Organization **Ivy software Development Private Limited** for support, resources, and encouragement throughout this endeavor.

Appreciation is also extended to the entire **faculty of BITS Pilani WILP** for fostering an intellectually stimulating environment. Their dedication to academic excellence enriched my learning experience and influenced my scholarly pursuits.

Lastly, heartfelt thanks go to **colleagues and friends** for their unwavering support, encouragement, and assistance. They have been the pillars of strength, encouragement, and inspiration throughout my academic and personal journey.

Contents

Introduction

What is telemetry?

Telemetry[1] is the automated process of collecting and transmitting data from remote or inaccessible sources to a central location for monitoring, analysis, and decision-making. This technology enables the real-time measurement and transmission of information, often in the form of electronic signals or sensor data, from a distant location to a receiving system. Telemetry systems are widely used in various fields, including aerospace, healthcare, automotive, telecommunications, and industrial processes.

Overall, telemetry plays a pivotal role in enabling remote monitoring, control, and analysis of systems, contributing to improved efficiency, safety, and decision-making in diverse applications.

Telemetry types

Telemetry data is crucial for monitoring and optimizing various aspects of their operations. Different telemetry types are collected based on specific requirements. Here are some key telemetry types:

Telemetry Data from IT Infrastructures:

- Transaction and error rates
- Response times
- CPU and memory usage
- Disk I/O
- Network throughput

User Telemetry Data:

- User interactions with product features
- Examples: button clicks, system logins, page views, error encounters

Network Telemetry Data:

- Bandwidth capacity monitoring
- Monitoring specific network ports
- Storage solution metrics

- Health of network devices (routers, switches)
- CPU and memory utilization of network devices
- Device uptime and temperature

Application Infrastructure Telemetry Data:

- Latency
- Transactions per second
- Database access and queries
- Application-generated errors
- Deployment-specific activities (e.g., deployment topology)
- Insights into operating systems, browser types/versions, and device details

Telemetry Data in Cloud Environments:

- Routing decisions
- Configuration changes
- Security group modifications
- Cloud usage-related data

Key tenets of telemetry analytics

The key tenets for near real-time telemetry[2] analytics for data pipelines are:

- Data and Metrics Collection
- Aggregation
- Anomaly detection
- Notification and resolution
- Maintaining persistence, and create visualization.

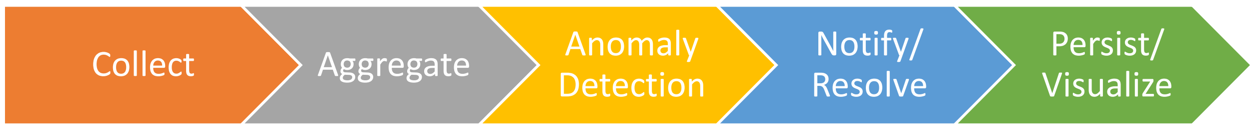


Figure 1. Key tenets of telemetry analytics

Telemetry vs Monitoring vs Observability

Telemetry, Monitoring, and Observability play essential roles in facilitating efficient and dependable software development and deployment. Although these terms share connections, they possess individual meanings and purposes. By discerning their unique distinctions and appreciating their combined significance, we can harness their capabilities to achieve a more profound understanding of systems, ultimately ensuring peak application performance and resilience.

Differentiators	**Telemetry**	**Monitoring**	**Observability**
Purpose	Measure and collect data for analysis	Track system performance and health	Understand system internals and behavior
Focus	Data collection and transmission	Overall system performance and metrics	Systems' current and future internal states
Scope	Data gathering for various purposes	Specific metrics and events	Comprehensive and deep understanding
Data	Raw and processed data for analysis	Aggregated data and metrics	Rich, granular and contextual data
Analysis	Statistical analysis and trend prediction	Rule-based or threshold-based detection	Exploratory and ad-hoc analysis
Complexity	Generalized, can be applied broadly	Primarily focused on individual components	Addresses complex distributed systems

It is important to emphasize that none of these elements independently ensures system reliability. Instead, they work in tandem, and a thorough strategy should encompass all three. Telemetry furnishes real-time data, monitoring identifies particular issues, and observability enables in-depth analysis of accumulated data for root cause identification, understanding system behavior, and optimizing performance.

The selection of appropriate telemetry tools depends on various factors, such as specific requirements, the characteristics of the monitored system or application, and the objectives of telemetry implementation.

Snowflake

Snowflake[4] is a cloud-based data warehousing platform that has gained significant traction for its innovative approach to data storage, processing, and analytics.

One of Snowflake's distinctive features is its architecture, which separates storage and compute resources. This separation allows users to independently scale these resources based on their specific needs, optimizing performance and cost-effectiveness. Snowflake operates as a fully managed service in popular cloud providers such as AWS, Azure, and Google Cloud Platform, eliminating the need for users to handle infrastructure management tasks.

Snowflake supports a variety of data types and integrates seamlessly with popular data visualization and business intelligence tools. Its unique multi-cluster, shared data architecture ensures that users can run multiple workloads concurrently without compromising performance.

Notably, Snowflake embraces a SQL-based querying language, making it accessible to users with SQL proficiency and enabling a smooth transition for traditional data warehouse users. The platform's commitment to security is evident through features such as end-to-end encryption, access controls, and comprehensive audit trails, ensuring the protection of sensitive data.

Snowflake Architecture components

Snowflake's architecture is designed to provide a cloud-based data warehousing solution that is scalable, flexible, and easy to use. The key components of Snowflake's common architecture include:

Cloud Storage:

Snowflake utilizes cloud-based object storage (e.g., Amazon S3, Azure Blob Storage) to store structured and semi-structured data in a scalable and cost-effective manner.

Elastic, multi-cluster compute:

Snowflake's architecture features virtual compute clusters, which are responsible for processing queries and performing computations on the data stored in cloud storage. The separation of compute and storage allows for independent scaling of these resources.

Cloud Services Layer:

The cloud services layer serves as the control plane, managing tasks such as authentication, authorization, and session management. It also facilitates communication between the user interface, compute clusters, and metadata store.

Global Services Layer:

Snowflake's architecture features a global services layer that delivers a seamless, interconnected experience across worldwide regions and cloud environments. This framework promotes unified governance, ensures business continuity, and fosters collaboration within and between organizations.

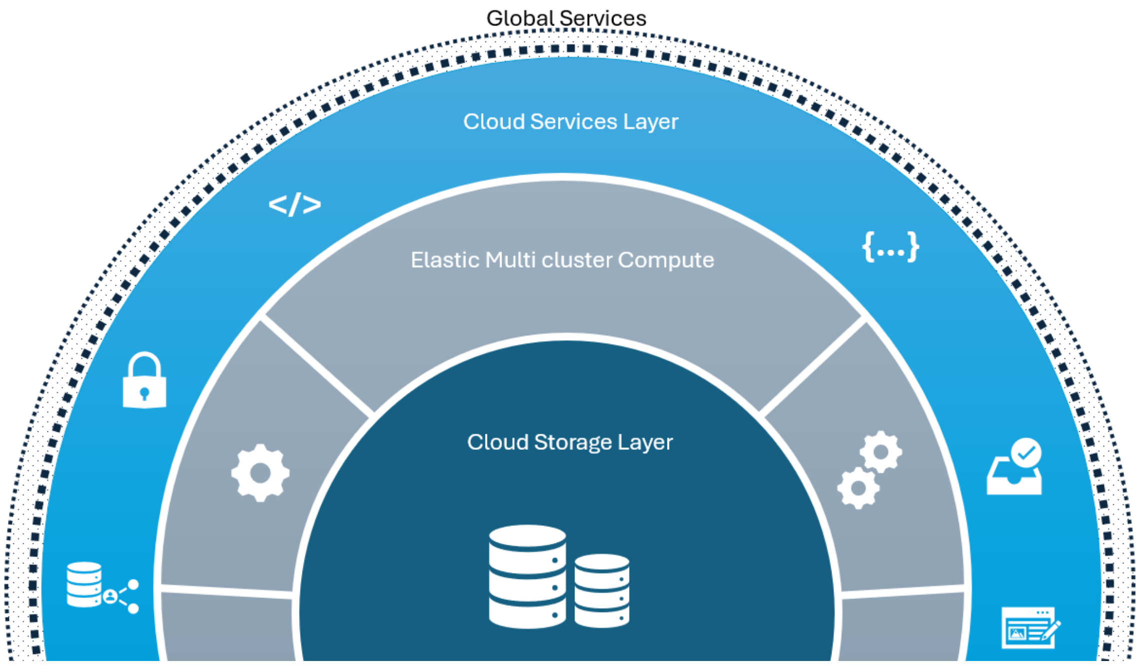


Figure 2. Snowflake Architecture (reference: snowflake.com)

Snowflake Telemetry Data

Snowflake primarily focuses on cloud data warehousing and does not directly provide telemetry data in the traditional sense. In the context of Snowflake, monitoring and logging functionalities are crucial for performance analysis, security, and auditing.

Snowflake provides a variety of features related to monitoring and managing the platform:

Snowflake Information Schema:

The Information Schema in Snowflake allows users to query metadata about their Snowflake account, providing insights into tables, views, and other objects.

Query History:

Users can access a history of executed queries, allowing for performance analysis and optimization. Query history includes details such as query text, execution time, and resource usage.

Account Usage:

Snowflake provides reports on account usage, detailing resource consumption, user activity, and warehouse usage.

Snowflake's UI and Web Interface:

The Snowflake web interface offers dashboards and visualizations for monitoring key performance indicators, query performance, and overall account health.

Snowflake Support for Third-Party Tools:

Users can integrate Snowflake with third-party monitoring and analytics tools to create custom dashboards and alerts based on specific telemetry requirements.

Data

Temporal filtering

To enhance the precision and relevance of machine learning (ML) analysis on the Snowflake Monitoring dataset. The chosen date range spans from **January 1, 2024, to February 10, 2024**, with a primary focus on refining the dataset for optimal analytical outcomes.

Reasoning

1. **Temporal Precision:**
   The deliberate selection of data from January 15, 2024, to February 2, 2024, aligns with the objective of achieving temporal precision in the ML analysis. This specificity ensures that the model is trained on recent and contextually relevant information within the query history.

2. **Relevance to Query Behavior:**
   Recognizing the dynamic nature of query behaviors, the chosen date range encapsulates a timeframe during which potential shifts, patterns, and trends in query execution are more likely to manifest.

3. **Quality Assurance:**
   Data quality is paramount in ML analysis. By selecting a specific temporal window, the analysis on a period expected to exhibit high data quality, reducing the impact of potential outliers or inconsistencies associated with older data.

4. **Efficiency in Analysis:**
   The temporal filtering approach contributes to the efficiency of the ML analysis. It streamlines the dataset, facilitating a more manageable computational load and expediting the training process. This, in turn, enhances the efficiency of the overall analysis.

5. **Functional Viability:**
   The temporal filtering approach is the most optimum in terms of data usability. This is the period where rigorous User Acceptance Testing, functional volumetric testing from various analytical tools like Power BI, MicroStrategy, Kyvos, Data Robot is also happening.

High Level Architecture/Data Flow

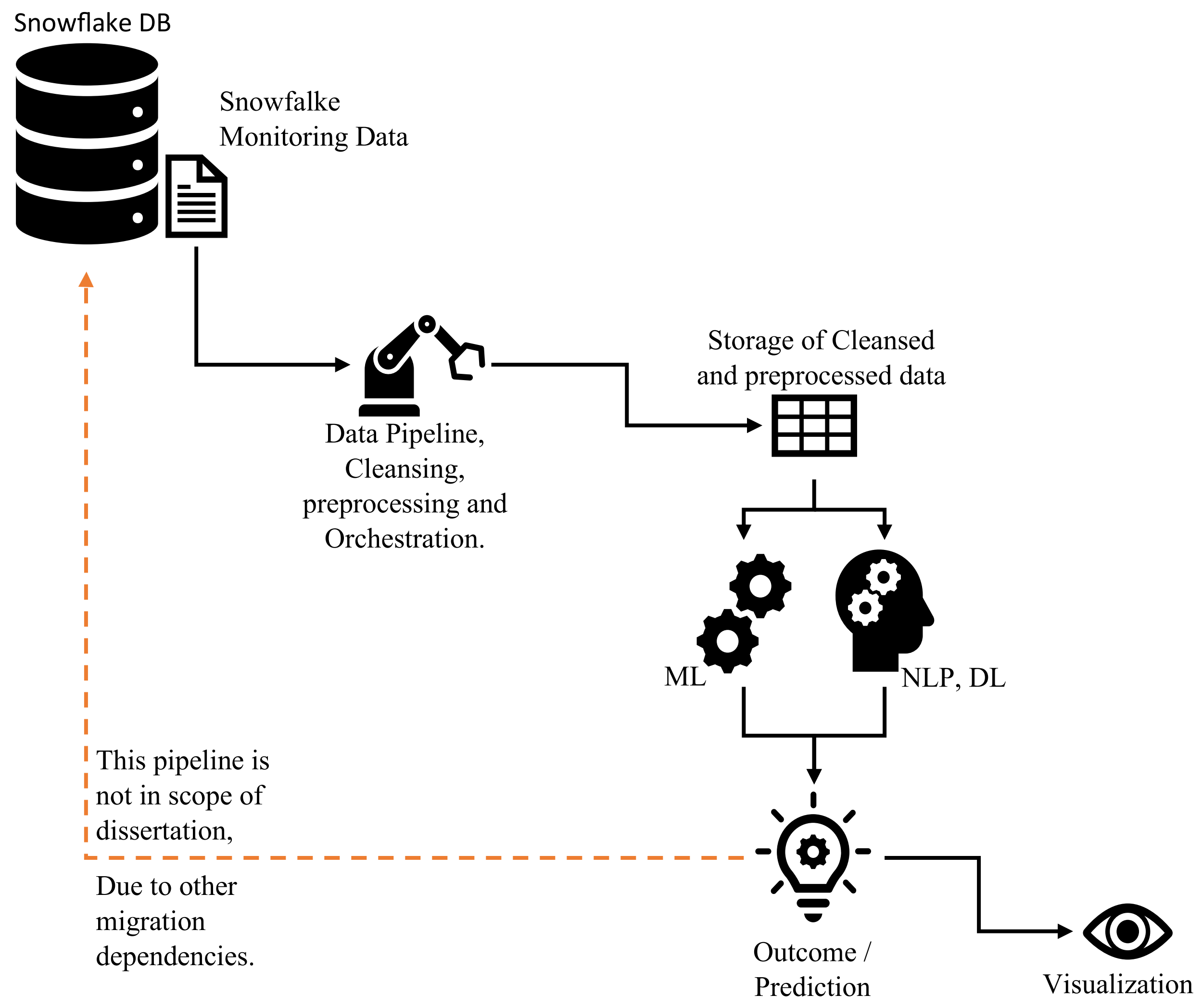

Snowflake DB
Snowfalke Monitoring Data
Data Pipeline, Cleansing, preprocessing and Orchestration.
Storage of Cleansed and preprocessed data
ML
NLP, DL
This pipeline is not in scope of dissertation,
Due to other migration dependencies.
Outcome / Prediction
Visualization

Figure 3. High Level Data Flow

Data Collection

Data is collected from snowflake monitoring tables.

SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY

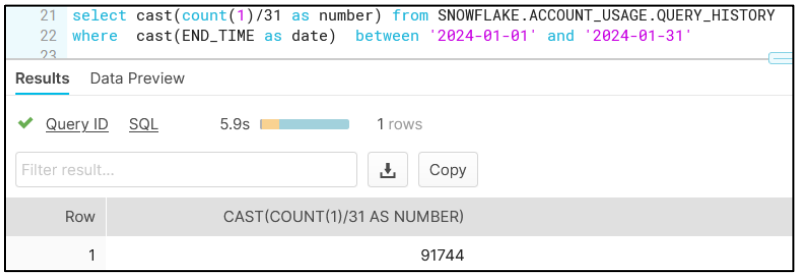
Metadata

QUERY_HISTORY

https://docs.snowflake.com/en/sql-reference/account-usage/query_history

Volume

Average Daily Volume QUERY_HISTORY



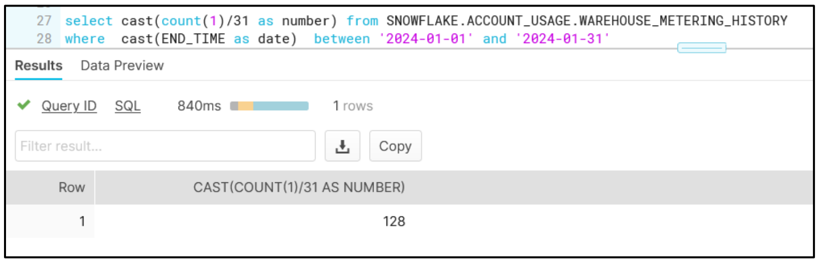
```sql
select cast(count(1)/31 as number) from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where  cast(END_TIME as date)  between '2024-01-01' and '2024-01-31'
```

Results   Data Preview

✓ Query ID   SQL   5.9s   1 rows

Row	CAST(COUNT(1)/31 AS NUMBER)
1	91744

Average Daily Volume WAREHOUSE_METERING_HISTORY



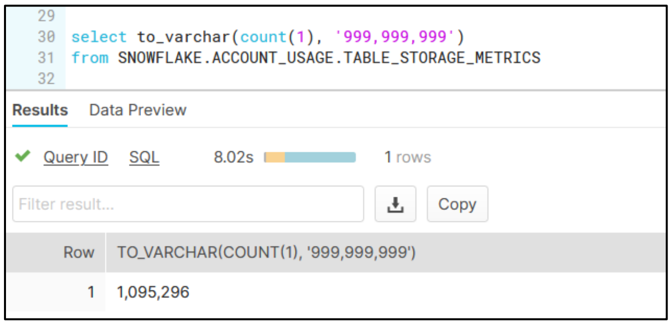
```sql
select cast(count(1)/31 as number) from SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
where  cast(END_TIME as date)  between '2024-01-01' and '2024-01-31'
```

Results   Data Preview

✓ Query ID   SQL   840ms   1 rows

Row	CAST(COUNT(1)/31 AS NUMBER)
1	128

Total Volume WAREHOUSE_METERING_HISTORY



```sql
select to_varchar(count(1), '999,999,999')
from SNOWFLAKE.ACCOUNT_USAGE.TABLE_STORAGE_METRICS
```

Results   Data Preview

✓ Query ID   SQL   8.02s   1 rows

Row	TO_VARCHAR(COUNT(1), '999,999,999')
1	1,095,296

Exploratory Data Analysis

Part 1, SQL based EDA.

1. **Total Number of queries in QUERY_HISTORY per warehouse.**

```
select count(1), WAREHOUSE_NAME
from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where cast(END_TIME as date) between '2024-01-01' and '2024-01-31'
group by all order by count(1) desc;
```

```
   COUNT(1)      WAREHOUSE_NAME
     1315392     WH_XX_XXXX
     1050791     WH_XXXXX_XXXXXXXXXXX
      194303
      108315     WH_XXXXX_DEV
       46059     WH_XX_XXXXX
       40081     WH_XX_XXXXX2
       32992     WH_XXXXX_M
       24543     WH_XX_XXXX1
       14508     WH_XX_XXXREAD
        5379     WH_XX_XXXX1
        4754     WH_XX_XXXXX
        3805     WH_ADMIN
        2460     WH_XXXXX_XS
         530     WH_XX_XXXX1
         142     WH_XXXXX
          12     COMPUTE_SERVICE_WH_%
```

Values masked to suppress company interpretation.

Outcome: Data points with NULL warehouse and COMPUTE_SERVICE warehouse (marked RED) will be excluded as a part of preprocessing, as they have no relevance in the analytics.

Reason: Queries with NULL warehouse are metadata queries and have no direct impact on the utilization of snowflake elastic compute. Additionally, queries with COMPUTE_SERVICES are automatically generated from snowflake internal processing.

2. **Top 10 queries in terms of utilization rate**

```
select count(1) as Num_Queries, sum(TOTAL_ELAPSED_TIME/(60*60)) as
Total_Time_in_Hr, QUERY_TYPE
from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where cast(END_TIME as date) between '2024-01-01' and '2024-01-31'
group by all order by Total_Time_in_Hr/Num_Queries desc
limit 10;
```

```
NUM_QUERIES     TOTAL_TIME_IN_HR        QUERY_TYPE
7913            50538                   MULTI_STATEMENT
7963            40413                   MERGE
62302           215691                  CALL
24210           46147                   UPDATE
505037          827773                  SELECT
37613           59301                   COPY
19196           23016                   DELETE
199226          210922                  INSERT
69013           37640                   CREATE_TABLE_AS_SELECT
11187           1605                    TRUNCATE_TABLE
```

Outcome: The above metric provides significant patterns and hypothesis for usage of top 10 QUERY_TYPE as indicated above for Feature Engineering and Decision Support.

Reason: The significance of such queries for the overall cost of compute consumption is significantly higher than rest of the QUERY_TYPES

3. **Failed Query Analysis**

```
select count(1), EXECUTION_STATUS
from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where cast(END_TIME as date) between '2024-01-01' and '2024-01-31'
group by all order by count(1) desc;
```

```
COUNT(1)     EXECUTION_STATUS
2829603      SUCCESS
14462        FAIL
1            INCIDENT
```

```
select count(1) as Num_Queries, sum(TOTAL_ELAPSED_TIME/(60*60)) as
Total_Time_in_Hr, EXECUTION_STATUS
from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
where cast(END_TIME as date) between '2024-01-01' and '2024-01-31'
group by all order by Total_Time_in_Hr desc;
```

```
NUM_QUERIES    TOTAL_TIME_IN_HR    EXECUTION_STATUS
2829603        1505608.98          SUCCESS
14462          66695.46888         FAIL
1              438.119167          INCIDENT
```

Outcome: The above metric provides a number of SUCCESS and FAILED SQLs. For this use case, we shall only consider SUCCESS and FAIL executions.

Reason: INCIDENT query is an outlier.

Data Export

```
ctx = snowflake.connector.connect(
    user='XXXXXXXXXXXXXX',
    password='*************',
    account='xx#####.xxxxxxxxxxxx',
    )
cs = ctx.cursor()

# Specify the path for the CSV files
base_csv_file_path = 'Data/Query_History_'

# Loop through each day from January 1st to January 31st
start_date = datetime.date(2024, 1, 1)
end_date = datetime.date(2024, 1, 31)

current_date = start_date
while current_date <= end_date:
    # Format the date for the query
    formatted_date = current_date.strftime('%Y-%m-%d')
    file_date = current_date.strftime('%Y%m%d')

    # Execute query to fetch data
    query  =  f"select  *  from  SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY  where
cast(END_TIME as date) = '{formatted_date}' \
    and     WAREHOUSE_NAME     is     NOT     NULL     and     QUERY_TYPE     in
('MULTI_STATEMENT','MERGE','CALL','UPDATE','SELECT','COPY'\
    ,'DELETE','INSERT','CREATE_TABLE_AS_SELECT','TRUNCATE_TABLE')"
    cs.execute(query)

    # Fetch the results
    results = cs.fetchall()

    # Specify the path for the CSV file
    csv_file_path = f'{base_csv_file_path}{file_date}.csv'

    # Write results to a CSV file with UTF-8 encoding
    with open(csv_file_path, 'w', newline='', encoding='utf-8') as csvfile:
        csv_writer = csv.writer(csvfile)

        # Write header
        csv_writer.writerow([desc[0] for desc in cs.description])

        # Write data
        csv_writer.writerows(results)

    # Move to the next day
    current_date += datetime.timedelta(days=1)


# Close Snowflake connection
cs.close()
ctx.close()
```

Part 2, Python based EDA.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 537488 entries, 0 to 14344
Data columns (total 70 columns):
 #   Column                                          Non-Null Count   Dtype
---  ------                                          --------------   -----
 0   QUERY_ID                                        537488 non-null  object
 1   QUERY_TEXT                                      537484 non-null  object
 2   DATABASE_ID                                     502351 non-null  float64
 3   DATABASE_NAME                                   502351 non-null  object
 4   SCHEMA_ID                                       462803 non-null  float64
 5   SCHEMA_NAME                                     462803 non-null  object
 6   QUERY_TYPE                                      537488 non-null  object
 7   SESSION_ID                                      537488 non-null  int64
 8   USER_NAME                                       537488 non-null  object
 9   ROLE_NAME                                       537484 non-null  object
 10  WAREHOUSE_ID                                    537488 non-null  int64
 11  WAREHOUSE_NAME                                  537488 non-null  object
 12  WAREHOUSE_SIZE                                  344333 non-null  object
 13  WAREHOUSE_TYPE                                  537488 non-null  object
 14  CLUSTER_NUMBER                                  344328 non-null  float64
 15  QUERY_TAG                                       19140 non-null   object
 16  EXECUTION_STATUS                                537488 non-null  object
 17  ERROR_CODE                                      751 non-null     float64
 18  ERROR_MESSAGE                                   751 non-null     object
 19  START_TIME                                      537488 non-null  object
 20  END_TIME                                        537488 non-null  object
 21  TOTAL_ELAPSED_TIME                              537488 non-null  int64
 22  BYTES_SCANNED                                   537488 non-null  int64
 23  PERCENTAGE_SCANNED_FROM_CACHE                   537488 non-null  float64
 24  BYTES_WRITTEN                                   537488 non-null  int64
 25  BYTES_WRITTEN_TO_RESULT                         537488 non-null  int64
 26  BYTES_READ_FROM_RESULT                          537488 non-null  int64
 27  ROWS_PRODUCED                                   463482 non-null  float64
 28  ROWS_INSERTED                                   537488 non-null  int64
 29  ROWS_UPDATED                                    537488 non-null  int64
 30  ROWS_DELETED                                    537488 non-null  int64
 31  ROWS_UNLOADED                                   537488 non-null  int64
 32  BYTES_DELETED                                   537488 non-null  int64
 33  PARTITIONS_SCANNED                              537488 non-null  int64
 34  PARTITIONS_TOTAL                                537488 non-null  int64
 35  BYTES_SPILLED_TO_LOCAL_STORAGE                  537488 non-null  int64
 36  BYTES_SPILLED_TO_REMOTE_STORAGE                 537488 non-null  int64
 37  BYTES_SENT_OVER_THE_NETWORK                     537488 non-null  int64
 38  COMPILATION_TIME                                537488 non-null  int64
 39  EXECUTION_TIME                                  537488 non-null  int64
 40  QUEUED_PROVISIONING_TIME                        537488 non-null  int64
 41  QUEUED_REPAIR_TIME                              537488 non-null  int64
 42  QUEUED_OVERLOAD_TIME                            537488 non-null  int64
 43  TRANSACTION_BLOCKED_TIME                        537488 non-null  int64
 44  OUTBOUND_DATA_TRANSFER_CLOUD                    0 non-null       float64
 45  OUTBOUND_DATA_TRANSFER_REGION                   0 non-null       float64
 46  OUTBOUND_DATA_TRANSFER_BYTES                  537488 non-null  int64
 47  INBOUND_DATA_TRANSFER_CLOUD                   0 non-null       float64
 48  INBOUND_DATA_TRANSFER_REGION                  0 non-null       float64
 49  INBOUND_DATA_TRANSFER_BYTES                   537488 non-null  int64
 50  LIST_EXTERNAL_FILES_TIME                      537488 non-null  int64
 51  CREDITS_USED_CLOUD_SERVICES                   537488 non-null  float64
 52  RELEASE_VERSION                               537488 non-null  object
 53  EXTERNAL_FUNCTION_TOTAL_INVOCATIONS           537488 non-null  int64
 54  EXTERNAL_FUNCTION_TOTAL_SENT_ROWS             537488 non-null  int64
 55  EXTERNAL_FUNCTION_TOTAL_RECEIVED_ROWS         537488 non-null  int64
 56  EXTERNAL_FUNCTION_TOTAL_SENT_BYTES            537488 non-null  int64
 57  EXTERNAL_FUNCTION_TOTAL_RECEIVED_BYTES        537488 non-null  int64
 58  QUERY_LOAD_PERCENT                            344333 non-null  float64
 59  IS_CLIENT_GENERATED_STATEMENT                 537488 non-null  bool
 60  QUERY_ACCELERATION_BYTES_SCANNED              537488 non-null  int64
 61  QUERY_ACCELERATION_PARTITIONS_SCANNED         537488 non-null  int64
 62  QUERY_ACCELERATION_UPPER_LIMIT_SCALE_FACTOR  537488 non-null  int64
 63  TRANSACTION_ID                                537488 non-null  int64
 64  CHILD_QUERIES_WAIT_TIME                       537488 non-null  int64
 65  ROLE_TYPE                                     537484 non-null  object
 66  QUERY_HASH                                    527656 non-null  object
 67  QUERY_HASH_VERSION                            537488 non-null  int64
 68  QUERY_PARAMETERIZED_HASH                      527656 non-null  object
 69  QUERY_PARAMETERIZED_HASH_VERSION              537488 non-null  int64
dtypes: bool(1), float64(12), int64(38), object(19)
memory usage: 287.6+ MB
None
```

Visualize the relationship between two variables (e.g., partitions scanned and total elapsed time)

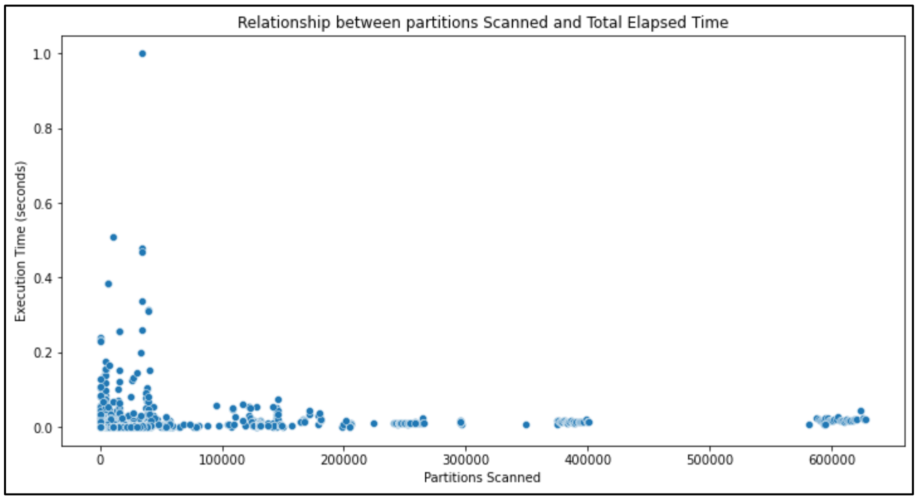


Figure 4. Relationship between partitions scanned and total elapsed time.

Correlation matrix of various data attributes

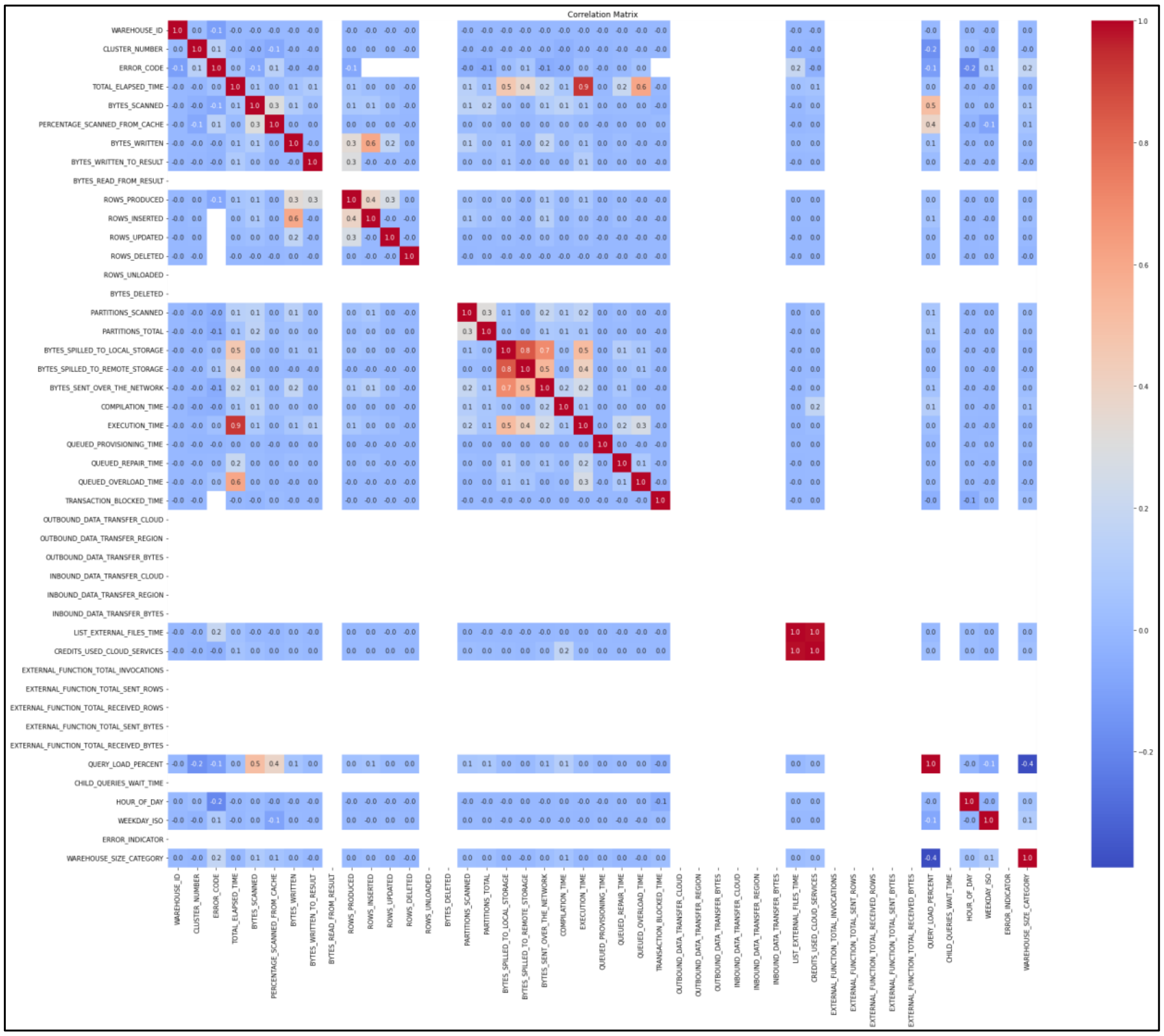


Figure 5. Correlation matrix within various data attributes.

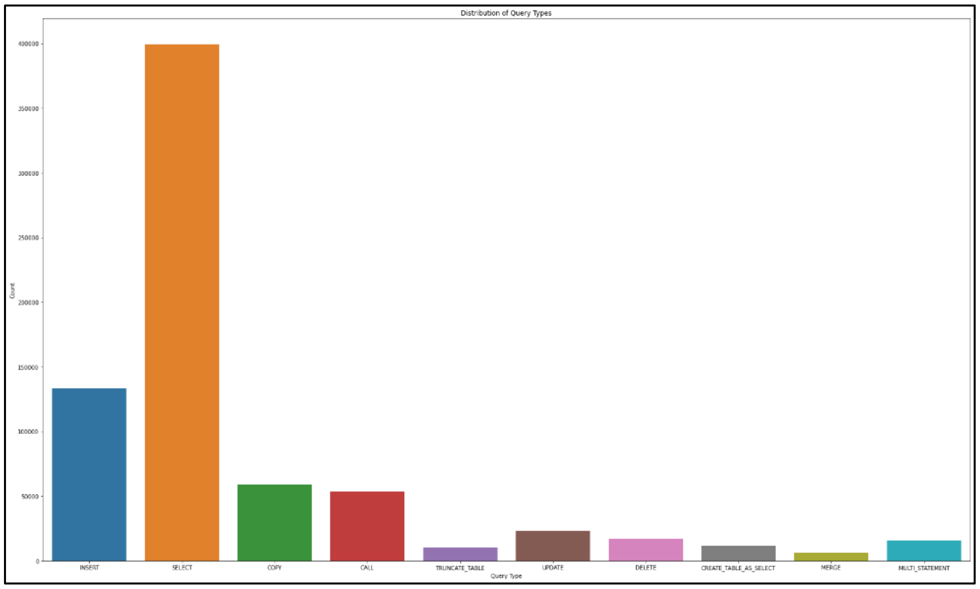


Figure 6. Distribution of Query Types.

Distribution of Warehouse Sizes

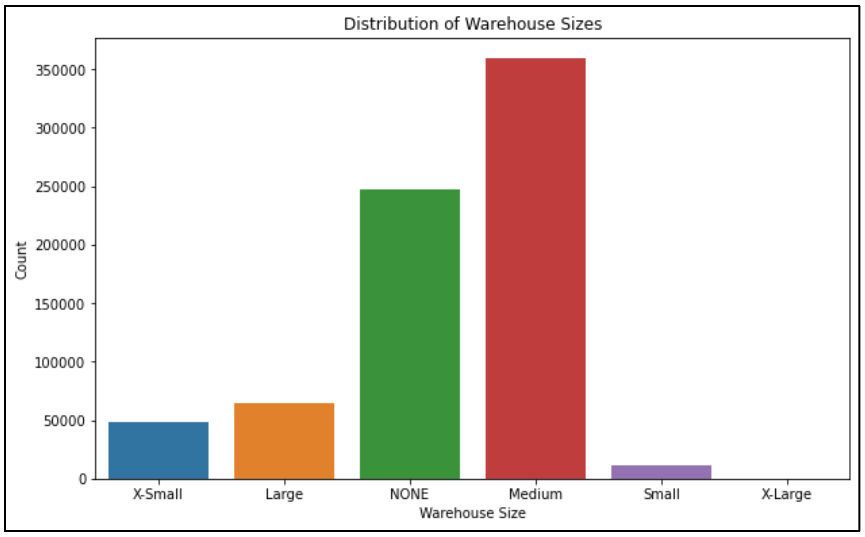


Figure 7. Distribution of Query Types.

Preprocessing

Normalization

Normalization plays a pivotal role in data preprocessing, particularly when managing features with varying scales or units. Its primary objective is to standardize the data onto a comparable scale, thereby avoiding the undue influence of specific features over others during the modeling process.

Performed Z-score Standardization (Standard Scaling) in the input data set.

Formula: $$X_{standardized} = \frac{X-\mu}{\sigma}$$

Where: μ: mean and σ: standard deviation

Feature selection

Feature selection can be an important part of the data preprocessing pipeline because it helps simplify the dataset, reduces dimensionality, and can improve the efficiency and performance of machine learning models. Unnecessary Columns are dropped which have no primary significance in the scope of this project work.

Type Casting

Datetime Conversion

Converting timestamp or string representations of dates and times to datetime objects for time-based analysis or modeling.

```
# Convert the timestamp column to datetime format
df['START_TIME'] = pd.to_datetime(df['START_TIME'])
df['END_TIME'] = pd.to_datetime(df['END_TIME'])
```

Outlier Handling

Outlier handling in data preprocessing serves the purpose of addressing and alleviating the influence of outliers on statistical analysis and machine learning models. Outliers, characterized as data points significantly deviating from the majority, have the potential to distort analysis outcomes and impact model performance.

Winsorizing

Winsorizing is a statistical approach where extreme values (outliers) within a dataset are replaced by values that are less extreme. The primary aim is to mitigate the impact of outliers on statistical analysis and modeling while retaining a certain degree of their presence in the dataset.

Feature Scaling

The main objective of feature scaling is to standardize or normalize the range of independent variables or features within the input data.

Min Max Scaling

Min-Max Scaling, or Min-Max Normalization, is a widely employed method in data preprocessing for scaling numerical features to a designated range. The objective of Min-Max Scaling is to adjust the data so that it fits within a predefined interval, commonly [0, 1]. This normalization approach proves advantageous in ensuring that all features carry equal weight in the analysis or modeling process, preventing the undue influence of specific features due to disparities in scale.

Formula: $X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$

Where, X is the original value of feature,

X_{min} is the minimum value of the feature in the dataset.

X_{max} is the maximum value of the feature in the dataset.

Feature Engineering

Feature engineering is actively conducted within SQL execution on the SNOWFLAKE.ACCOUNT_USAGE table. Performing feature engineering at the source offers several advantages:

1. **Efficiency:**
Feature engineering at the source allows for the creation and transformation of features directly within the SQL query. This reduces the need for additional processing steps in downstream applications or analysis tools.

2. **Optimized Query Performance:**
By incorporating feature engineering into the SQL execution, it can leverage the database's processing power and optimization capabilities. This can lead to more efficient query performance compared to extracting raw data and performing feature engineering later in the data processing pipeline.

3. **Reduced Data Movement:**
Feature engineering at the source minimizes the need to transfer large volumes of raw data to external environments for further processing. This reduces network overhead and accelerates the overall data processing workflow.

4. **Data Governance and Consistency:**
Feature engineering at the source ensures that all downstream applications, reports, and analyses benefit from a consistent set of features.

5. **Resource Optimization:**
Leveraging the capabilities of the underlying database for feature engineering can be resource efficient. It optimizes resource utilization by distributing the workload between the database and other processing components.

6. **Integration with Business Logic:**
Feature engineering at the source allows for the integration of domain-specific business logic directly into the SQL queries. This ensures that features are derived with a deep understanding of the business context, improving the relevance and quality of the engineered features.

Data Filtering

During the EDA phase it was observed that the SQLs which were using WAREHOUSE were the only ones contributing to the consumption of elastic compute in snowflake. Within the SQL query, filter has been applied to include only the data which have warehouse usage.

Additional outcome from EDA phase was the QUERY_TYPE, which was contributing more towards compute consumption, Within the SQL query, filter has been applied to include only the top consuming Query types.

Temporal Feature Engineering

Temporal feature engineering involves deriving new features or insights from time-related information in the dataset.

Feature Name: 'HOUR_OF_DAY'
Engineering Operation: Extraction of the hour component from the 'START_TIME' column.

Feature Name: 'HOUR_OF_DAY'
Engineering Operation: Extraction of the hour component from the 'START_TIME' column.

Binary Feature Engineering

Binary feature engineering involves creating a new feature that takes on only two possible values, typically representing the presence or absence of a certain condition or characteristic in the data.

Feature Name: 'ERROR_INDICATOR'
Engineering Operation: Transformation of the 'EXECUTION_STATUS' column values into binary indicators (0 or 1) based on whether the query execution status is 'FAILED' or not.

Time Series Analysis

Daily Query Count

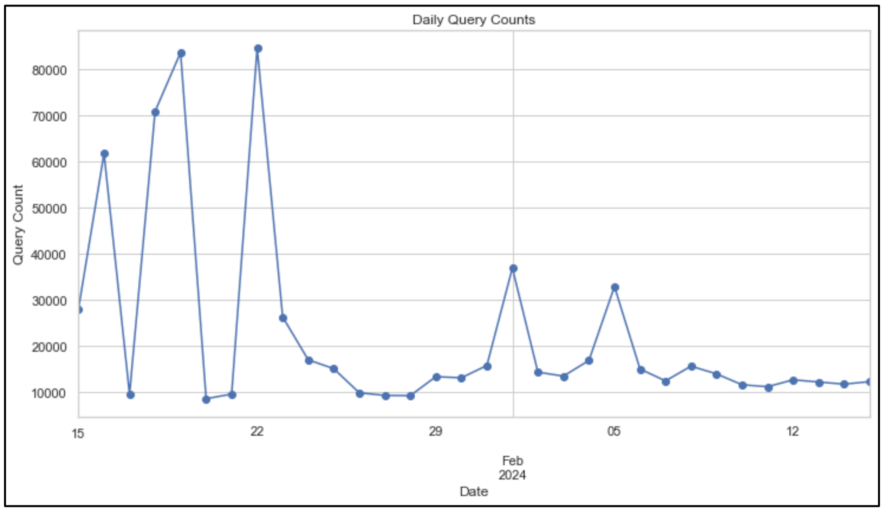
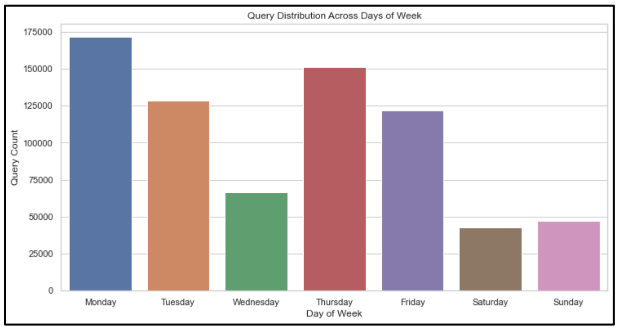


Figure 8. Time Series Analysis - Daily Query Count

The above time series graph illustrates notable fluctuations in query counts throughout January, primarily attributed to the occurrence of multiple holidays during that month. Subsequent to January, the daily query count displays reduced variability starting from February onwards.
This is also the reason for choosing February data for Model training.

Day of Week and Hour of Week Analysis



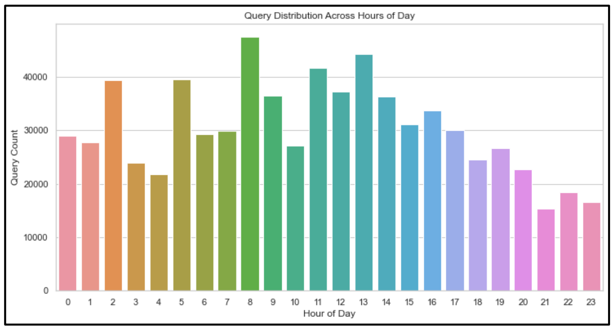


Figure 9(a) Day of week Analysis

Figure 9(b) Hour of day Analysis.

ML Model

Goal:

The primary goal of the ML model is to predict whether a query execution will result in success or failure based on various features related to the query.

Support Vector Machine (SVM) Model

Support Vector Machine (SVM) Model is chosen in this case due to characteristic of the data and goal of ML exercise.

Here are some reasons:

Feature Dimensionality:

Since the dataset contains a moderate to high number of features related to query execution, SVM can handle high-dimensional spaces effectively. This is advantageous when dealing with complex relationships between features.

Non-Linearity in Data:

The decision boundary between success and failure appears complex and non-linear, SVM can handle this with kernel functions. The ability to model non-linear relationships makes SVM suitable for scenarios with intricate decision boundaries.

Handling Imbalanced Classes:

SVM can handle imbalanced datasets by assigning different misclassification costs to different classes. This is beneficial if there is a significant class imbalance in the success and failure of query executions.

NLP Techniques

Tokenization

Tokenization involves dividing a text into smaller units, referred to as tokens. These tokens may consist of words, subwords, or individual characters, depending on the desired level of granularity. Tokenization constitutes a fundamental stage in natural language processing (NLP) and text analysis.

In the current data frame, we extract the QUERY_TEXT and perform word tokenization on the df using sqlparse[5].

Syntax Aware tokenization using sqlparse

Syntax-aware tokenization refers to the process of tokenizing text while considering the syntactic structure or grammar of the language. In the context of SQL queries or programming languages, syntax-aware tokenization involves breaking down the input text into meaningful tokens while considering the language's syntax rules.

Syntax aware tokenization using sqlparse is being performed in this dissertation.

sqlparse is a Python library that serves as a non-validating SQL parser. Designed to offer capabilities in parsing, splitting, and formatting SQL statements, the sqlparse module presents three straightforward functions at the module level. These functions are intended to assist in common tasks associated with working on SQL statements.

split():	Split function splits multiple SQL statements identified by occurrence of semicolon.
format()	SQL statements can be semantically written with the help of format function. This makes SQL more readable.
parse()	Parse function parses the SQL and returns the tuple of statement instance.

Since the scope of this dissertation is snowflake QUERY_HISTORY data, the tokenization of QUERY_TEXT is performed using sqlparse library.

```
# Function for syntax-aware tokenization using sqlparse
def tokenize_sql(query):
    # Use sqlparse to tokenize SQL query
    parsed = sqlparse.parse(query)

    # Extract tokens from parsed SQL
    tokens = []
    for statement in parsed:
        for token in statement.tokens:
            # Add non-whitespace tokens to the list
            if not token.is_whitespace:
                tokens.append(str(token))

    return tokens

# Apply tokenization to each row in the QUERY_TEXT column
tokenized_data = text_data['QUERY_TEXT'].apply(tokenize_sql)
```

Sentiment Analysis

Sentiment analysis on the QUERY_TEXT data of Snowflake's QUERY_HISTORY can offer several advantages, providing valuable insights and enhancing the understanding of the queries and user interactions with the database.

- **Identifying Query Satisfaction:**

  Sentiment analysis helps determine the sentiment expressed in queries, allowing us to identify whether users are satisfied or dissatisfied with their interactions. Positive sentiments may indicate successful and satisfactory queries, while negative sentiments might suggest issues or frustrations.

- **Detecting Performance Issues:**

  Negative sentiments in queries might be indicative of performance issues or bottlenecks in the database. Analyzing sentiments can help identify queries associated with user dissatisfaction, enabling proactive performance optimization.

- **Improving Query Optimization:**

  Sentiment analysis can be used to categorize queries based on user satisfaction levels. By understanding which types of queries receive positive or negative sentiments, we can prioritize and focus on optimizing queries that are critical for user satisfaction.

- **Automated Issue Detection:**

  Integrating sentiment analysis into a monitoring system can enable automated detection of potential issues or anomalies in the database. Negative sentiments in queries may trigger alerts for further investigation.

- **Trend Analysis and Reporting:**

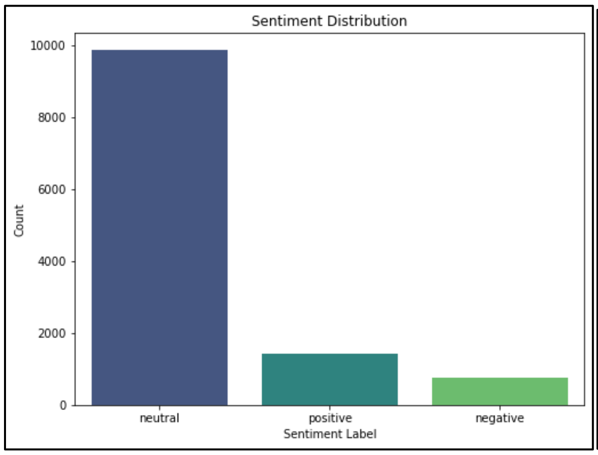
  Over time, sentiment analysis results can be used to identify trends and patterns in user sentiments. This information can be valuable for reporting and strategic decision-making.

- **Understanding Query Language Usage:**

Analyzing sentiments in queries can provide insights into the language and expressions users use. Understanding the natural language patterns can be helpful for improving documentation, tutorials, or user training materials.

- **Predictive Maintenance:**

By identifying negative sentiments associated with specific queries or patterns, we can proactively address potential issues before they become widespread, contributing to a more stable and user-friendly database environment.



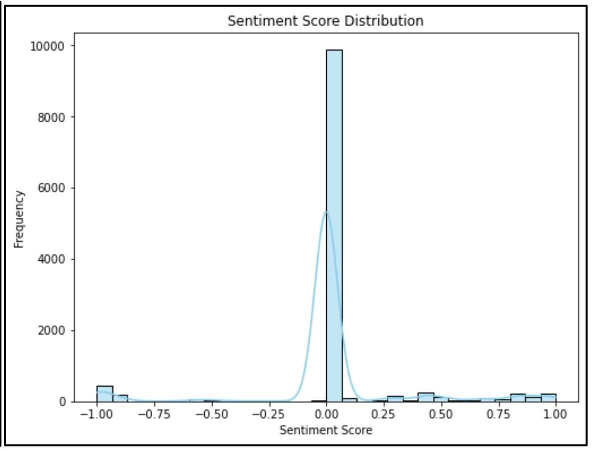


Figure 10(a) Sentiment Distribution

Figure 10(b) Sentiment Score distribution.

Deep Learning

Feed Forward Neural Network

For the given problem statement and based on input data, I am implementing a feedforward neural network also known as multilayer perceptron (MLP). Here are some characteristics of this type of neural model.

1. **Feedforward Structure:**
   Information travels in a unidirectional manner, moving from the input layer through any intervening hidden layers before culminating at the output layer. The network is designed without cycles.

2. **Multilayer Architecture:**
   The network architecture encompasses multiple strata, incorporating an input layer, one or more hidden layers, and a terminal output layer. Specifically, a single hidden layer is present in this configuration.

3. **Fully Connected (Dense) Layers:**
   Each neuron within a given layer establishes connections with every neuron in the subsequent layer. These inter-neuronal connections are encoded through weights within the linear transformation layers, also known as fully connected or dense layers.

4. **Activation Function:**
   ReLU (Rectified Linear Unit) is employed as the activation function within the hidden layer. ReLU introduces non-linearity to the model, augmenting the model's capacity to discern intricate relationships within the dataset.

5. **Output Layer:**
   The output layer generates the final predictions. In regression tasks, it typically has a single neuron, and for classification tasks, the number of neurons matches the number of classes.

```
# Define the neural network architecture
class WarehouseSizePredictor(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(WarehouseSizePredictor, self).__init__()
        # Define the first fully connected layer
        self.fc1 = nn.Linear(input_size, hidden_size)
        # Define the ReLU activation function
        self.relu1 = nn.ReLU()
        # Define the second fully connected layer
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        # Define the ReLU activation function for the second hidden layer
        self.relu2 = nn.ReLU()
        # Define the third fully connected layer
        self.fc3 = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x
```

The forward method defines the sequence of operations during the forward pass:

- **Input Layer (fc1):** This layer takes an input of size input_size and passes it to the first hidden layer. The linear transformation is applied using the weights and biases associated with this layer.
- **ReLU Activation (relu):** After the first linear transformation, the Rectified Linear Unit (ReLU) activation function is applied on each element. This introduces non-linearity to the model.
- **Second Hidden Layer (fc2):** The output from the first hidden layer is passed to the second hidden layer. This layer has the same number of neurons as the first hidden layer.
- **SELU Activation (selu):** After the second linear transformation, the Scaled Exponential Linear Unit (SELU) activation function is applied. SELU is a type of self-normalizing activation function. It overcomes the vanishing gradient over ReLU and learns faster and better than other activation functions.
- **Output Layer (fc3):** Finally, the output from the second hidden layer is passed to the output layer. This layer produces the final output with output_size neurons, which is typically one for regression problems.

The torch package encompasses data structures tailored for handling multi-dimensional tensors, coupled with the definition of mathematical operations specifically crafted for these tensors. Furthermore, it furnishes a plethora of utilities facilitating the streamlined serialization of Tensors and diverse data types. Beyond this, the package includes a range of practical and efficient utilities for various purposes.

Hyperparameters

Hyperparameters refer to the parameters established before the training process that are not learned directly from the data. Divergent from model parameters, hyperparameters are not derived from the training data; instead, they are predetermined. The selection of hyperparameters plays a pivotal role in shaping the model's learning dynamics, convergence behavior, and generalization performance.

In this case, the hyperparameters include –

Hidden Sizes (hidden_sizes):

This is a list containing different sizes for the hidden layer of the neural network. The model will be trained and evaluated for each hidden size in the list.

Learning Rates (lrs):

This is a list containing different learning rates for the optimizer. The learning rate controls the step size during optimization. The model will be trained and evaluated for each learning rate in the list.

Number of Epochs (num_epochs):

This hyperparameter defines the number of times the entire dataset is passed through the neural network during training.

Optimizer (Adam):

The Adam optimizer is used with varying learning rates. The Adam optimizer adjusts the learning rates of each parameter individually.

Loss Function (MSELoss):

Mean Squared Error (MSE) loss is used, which measures the average squared difference between predicted and actual values.

manual hyperparameter search is performed by iterating through all combinations of hidden sizes and learning rates. It evaluates each combination on the test set and selects the combination that results in the lowest test loss as the best hyperparameters. Finally, the final model is trained with the best hyperparameters.

Hidden Size	Learning Rate	Number of Epoch
8	0.01	1000
8	0.1	1000
8	1	1000
16	0.01	1000
16	0.1	1000
16	1	1000
32	0.01	1000
32	0.1	1000
32	1	1000
64	0.01	1000
64	0.1	1000
64	1	1000

Constraints

Data Limitations

1. **Limited Query Diversity:** As the current system is in phase of migration, the types of queries executed in the Snowflake database appear to be limited, leading to a lack of diversity in the query history data. A model trained on a narrow set of queries struggles to handle unseen queries.
2. **Imbalance in Query Frequency:** Certain types of queries are executed much more frequently than others. This leads to imbalances in the dataset, and the model appears to be biased toward more common query patterns.
3. **Impact of active QA:** With active QA being performed on the current data, there are lot of repetitive and small length SQL statements which do not give enough insights on the nature of processing.
4. **Changes in System Configuration:** Due to the nature of the overall project, changes in the Snowflake system configuration or architecture over time have impacted query performance and execution patterns.
5. **Queries processing from Cache:** Repetitive queries are often returned from cache (result cache or warehouse cache) rendering the consumption of warehouse to almost nil.
6. **One-time Queries:** There are significantly high number of one-time queries, from historical data migration and data fixes, which is resulting in skewness in the model.
7. **Influence of External Factors:** External factors such as changes in data volume, evaluation/testing of migration tools, tactical solutions to expedite migration influences query performance.

Sample Size and Selection

- The sample size taken during EDA phase was 1st January 2024 to 31 January 2024. However, due to volume and length of monitoring data the sample size for ML and Deep Learning Model was changed to 1st February 2024 to 9th February 2024.
- Due to post holidays reintegration, along with other holidays observed every week in January the Query History was not providing complete workload for an entire week.
- Additionally, the Models were not performing well for 1 month sample size, and had to be changed to a 9 day overall period.

Methodological Constraints

1. **Method of Data Collection:** The method of data export to csv and moving data outside snowflake in csv feeds was a methodological constraint.
2. **Variety of Data:** The variety of data collected for the exercise could have been improved, other monitoring data sets could have been collected for more comprehensive analysis and modelling.
3. **Scope Creep:** The dissertation scope was very vast to cover overall aspects of telemetry analytics. The topic could have been narrowed to avoid gradual and uncontrolled expansion.
4. **Lack of Reference:** There is a lack of references for the available analytics on telemetry data for a Data Warehouse system.

Technological Constraint

- Query Parsing libraries are built to parse ANSI standard SQLs. There is a technological constraint to parse snowflake SQLs using SQLParse[6]. The library has various issues, especially in parsing nested SQLs, Stored Procedures.
- Snowflake offers Snowpark[7] ML – and End to End Machine Learning module within snowflake. However, Snowpark is considerably new offering within snowflake and various features are only available in preview.

Time Constraints

- The duration for collecting Snowflake query history data was constrained.
- Existing work commitments and tight schedules in Q1-2024 to deliver multiple migration milestones had some impacts on the dissertation.
- Limited window of time to experiment with different model architectures, hyperparameters, or feature engineering approaches.

Results

Results from ML Model

```
Accuracy: 0.9985209027144691
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    107339
           1       0.00      0.00      0.00       159

    accuracy                           1.00    107498
   macro avg       0.50      0.50      0.50    107498
weighted avg       1.00      1.00      1.00    107498
```

Accuracy:

0.9985: This represents a very high accuracy, which means the model correctly classified 99.85% of the data points.

Results from NLP

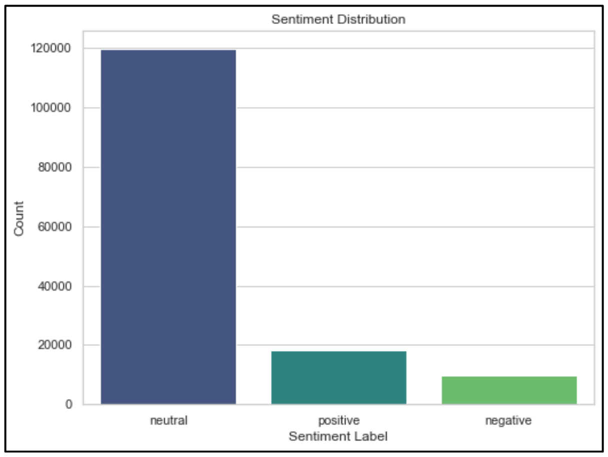


Figure 10(a) Sentiment Distribution

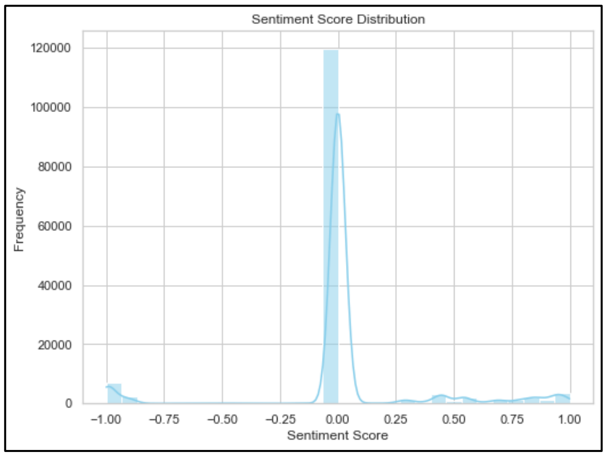


Figure 10(b) Sentiment Score distribution.

- The sentiment analysis predicts mostly neutral sentiments, this is probably due to presence of neutral statements like ‘INSERT’, ’UPDATE’, ’DELETE’, ’COPY’.
- Negative sentiment scores are more prominent than positive scores.

Results from Neural Model

```
Best Hyperparameters: {'hidden_size': 64, 'lr': 0.01, 'epochs': 1000, '
test_loss': 4.912156105041504}
Epoch [100/1000], Loss: 5.9330
Epoch [200/1000], Loss: 5.3488
Epoch [300/1000], Loss: 5.1727
Epoch [400/1000], Loss: 5.0970
Epoch [500/1000], Loss: 5.0952
Epoch [600/1000], Loss: 5.0164
Epoch [700/1000], Loss: 5.0630
Epoch [800/1000], Loss: 4.9691
Epoch [900/1000], Loss: 5.0374
Epoch [1000/1000], Loss: 4.9526
Test Loss: 4.9369
```

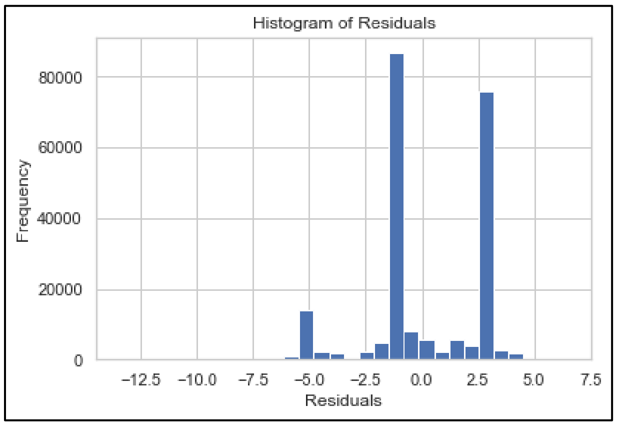


Figure 11(a) Histogram of Residuals

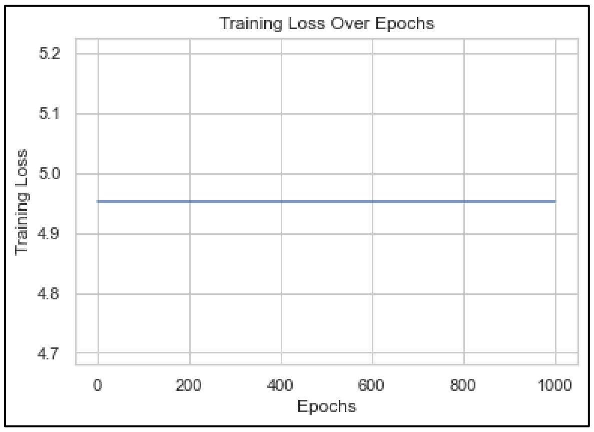


Figure 11(b) Training loss over Epochs

Histogram of Residuals:

The histogram displays the distribution of residuals. Residuals are the difference between the actual values of a variable and the predicted values from a model.

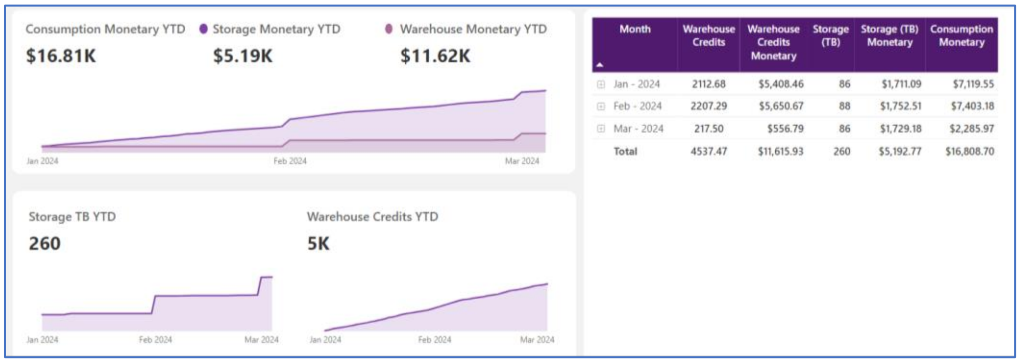
The highest frequency of residuals appears to be in the range of -2.5 to 0, which suggests that the model's predictions were generally close to the actual values.

The distribution appears to be slightly skewed to the right. This means that there are more positive residuals than negative residuals.

Training Loss over Epochs:

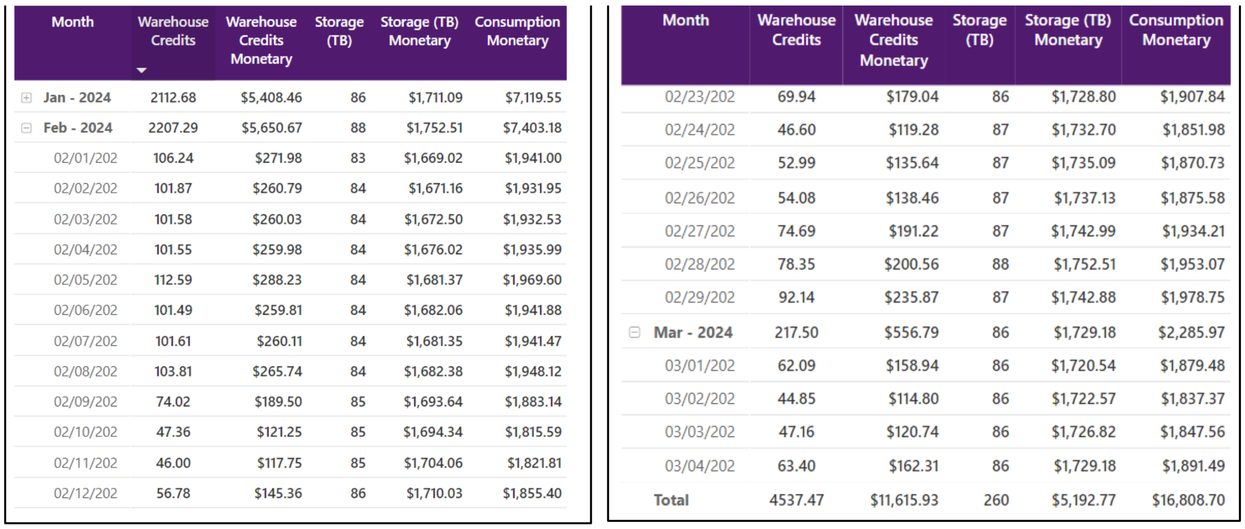
General Trend: The training loss appears to be decreasing over the epochs, which is a typical and desired outcome. This indicates that the model is gradually learning and improving its performance on the training data.

Visual representation of Results (Visualization)



Month	Warehouse Credits	Warehouse Credits Monetary	Storage (TB)	Storage (TB) Monetary	Consumption Monetary
⊞ Jan - 2024	2112.68	$5,408.46	86	$1,711.09	$7,119.55
⊞ Feb - 2024	2207.29	$5,650.67	88	$1,752.51	$7,403.18
⊞ Mar - 2024	217.50	$556.79	86	$1,729.18	$2,285.97
Total	4537.47	$11,615.93	260	$5,192.77	$16,808.70

Figure 12 Power BI Visualization for monthly trends in consumption



Month	Warehouse Credits	Warehouse Credits Monetary	Storage (TB)	Storage (TB) Monetary	Consumption Monetary
⊞ Jan - 2024	2112.68	$5,408.46	86	$1,711.09	$7,119.55
⊟ Feb - 2024	2207.29	$5,650.67	88	$1,752.51	$7,403.18
02/01/202	106.24	$271.98	83	$1,669.02	$1,941.00
02/02/202	101.87	$260.79	84	$1,671.16	$1,931.95
02/03/202	101.58	$260.03	84	$1,672.50	$1,932.53
02/04/202	101.55	$259.98	84	$1,676.02	$1,935.99
02/05/202	112.59	$288.23	84	$1,681.37	$1,969.60
02/06/202	101.49	$259.81	84	$1,682.06	$1,941.88
02/07/202	101.61	$260.11	84	$1,681.35	$1,941.47
02/08/202	103.81	$265.74	84	$1,682.38	$1,948.12
02/09/202	74.02	$189.50	85	$1,693.64	$1,883.14
02/10/202	47.36	$121.25	85	$1,694.34	$1,815.59
02/11/202	46.00	$117.75	85	$1,704.06	$1,821.81
02/12/202	56.78	$145.36	86	$1,710.03	$1,855.40

Month	Warehouse Credits	Warehouse Credits Monetary	Storage (TB)	Storage (TB) Monetary	Consumption Monetary
02/23/202	69.94	$179.04	86	$1,728.80	$1,907.84
02/24/202	46.60	$119.28	87	$1,732.70	$1,851.98
02/25/202	52.99	$135.64	87	$1,735.09	$1,870.73
02/26/202	54.08	$138.46	87	$1,737.13	$1,875.58
02/27/202	74.69	$191.22	87	$1,742.99	$1,934.21
02/28/202	78.35	$200.56	88	$1,752.51	$1,953.07
02/29/202	92.14	$235.87	87	$1,742.88	$1,978.75
⊟ Mar - 2024	217.50	$556.79	86	$1,729.18	$2,285.97
03/01/202	62.09	$158.94	86	$1,720.54	$1,879.48
03/02/202	44.85	$114.80	86	$1,722.57	$1,837.37
03/03/202	47.16	$120.74	86	$1,726.82	$1,847.56
03/04/202	63.40	$162.31	86	$1,729.18	$1,891.49
Total	4537.47	$11,615.93	260	$5,192.77	$16,808.70

Figure 13: Metrics before and after Model Deployment

Interpretation of visualization

There is a slight reduction in the consumption of warehouse credits (elastic compute consumption). During the first week of February 2024, the average daily consumption for the said warehouse was marked at ~102 credits per day. This is the same period for which the Temporal Filtering for Models has been applied.

Towards the end of February, the daily consumption has been observed at ~80 credits per day. This marks about 20% reduction in the warehouse consumption.

This benefit cannot be directly quantified into this dissertation exercise, as there are various external factors involved in reduction of compute usage. Key ones to note - Phase wise production migration, reduction in QA and full volumetric activities, migration of Reporting application.

Future Work

1. **Data Constraint Optimization:**
   Implement post-go-live modeling strategies following the attainment of Snowflake maturity to mitigate constraints related to data volume. Execute modeling procedures on a more refined dataset to enhance efficiency and effectiveness. Also improve the telemetry logging[8] in snowflake functions and procedures.

2. **Dynamic Model Revision and Generalization:**
   Adopt a model revision approach that accommodates frequent changes in the dataset. Implement dynamic model generalization techniques to ensure adaptability to evolving data patterns, enhancing the model's resilience to variations in the underlying dataset.

3. **Enhanced Data Processing with Snowpark ML Libraries:**
   Redefine data processing and collection methodologies by leveraging Snowpark ML libraries. Integrate Snowpark ML with other machine learning techniques to minimize data movement, optimize processing efficiency and reduce dependencies on external data sources.

4. **Incorporation of Diverse Monitoring Data:**
   Expand the dataset by incorporating a broader range of monitoring data sources. Enhance data analysis comprehensiveness by integrating diverse datasets, including system logs, user interactions, and performance metrics, providing a more holistic foundation for modeling.

5. **Real-time Data Integration:**
   Implement real-time data integration mechanisms to facilitate model testing. Utilize streaming technologies or event-driven architectures to ensure continuous ingestion of live data, allowing for immediate model evaluation and validation against real-world scenarios.

6. **Discrete Data Filtration for Bias Reduction:**
   Introduce discreet data filtration strategies to minimize one-time queries and mitigate bias in the dataset. Implement refined filtering mechanisms to exclude transient or non-representative data, contributing to a more unbiased and robust model training process.

7. **Experimentation with Multiple Models:**
   Broaden the scope of experimentation by testing various machine learning models. Explore different architectures, algorithms, and hyperparameter configurations to identify the models that yield optimal performance on the Snowflake query history data. Compare and contrast the results to inform model selection for future analytics.

Conclusion

Embarking on a comprehensive data science journey within the realm of Snowflake query history involves a systematic and multi-faceted approach. The process initiates with meticulous data capture, capturing a trove of historical queries that serve as the foundation for subsequent analyses. The captured data then undergoes a thorough preprocessing phase, incorporating filtering mechanisms and feature engineering techniques. This step is pivotal in ensuring that the data is both relevant and refined, setting the stage for meaningful insights.

A unique facet of this endeavor lies in the application of SQL-aware tokenization, achieved through tools like sqlparse. This innovative technique transforms raw SQL queries into structured tokens, facilitating a deeper understanding of linguistic patterns and syntactic structures embedded in the queries. This not only enhances the data's interpretability but also sets the stage for more sophisticated natural language processing (NLP) analyses.

Following the tokenization process, sentiment analysis is applied to gauge the emotional tone within the queries. Uncovering sentiments expressed in queries provides valuable insights into user attitudes and sentiments, which can be instrumental in understanding user experiences and shaping system improvements.

The pinnacle of this data science journey is the deployment of a neural model, exemplified by the WarehouseSizePredictor. Leveraging features like BYTES_SCANNED and PARTITIONS_SCANNED, this model predicts optimal warehouse sizes based on historical patterns. By optimizing resource utilization, organizations can ensure efficiency and cost-effectiveness in their data management strategies.

This integrated approach, spanning data preprocessing, NLP tokenization, sentiment analysis, and neural modeling, underscores the synergy of various data science techniques. It empowers organizations to extract actionable insights from their Snowflake QUERY history, facilitating strategic decision-making and enhancing overall data management efficiency. The collaborative nature of these methodologies establishes a robust framework for organizations to navigate the complexities of their data landscape and glean valuable insights for informed decision-making.

Future of Telemetry

The growing demand for insights derived from data has propelled advancements in data collection, analysis, and visualization. As systems evolve in complexity and cloud environments become more widespread, the significance of telemetry will amplify, facilitating enhanced monitoring, observability, and overall system reliability. Integrating telemetry with artificial intelligence (AI) and machine learning (ML) holds the potential for sophisticated analytics, thereby enhancing anomaly detection and predictive capabilities.

References

1. https://www.splunk.com/en_us/blog/learn/what-is-telemetry.html
2. https://aws.amazon.com/blogs/architecture/boosting-resiliency-with-an-ml-based-telemetry-analytics-architecture/
3. https://last9.io/blog/observability-vs-telemetry-vs-monitoring/
4. https://www.snowflake.com/blog/managing-snowflakes-compute-resources/
5. https://sqlparse.readthedocs.io/en/latest/
6. https://github.com/andialbrecht/sqlparse/issues
7. https://docs.snowflake.com/en/developer-guide/snowpark-ml/index
8. https://docs.snowflake.com/developer-guide/logging-tracing/logging