LTIMindtree

WHITEPAPER

Data Warehouse Testing in the Cloud Era

Strategies to Ensure Accuracy, Integrity, and Scalability

This whitepaper outlines a modern strategy for validating cloud data warehouses (DWHs) through extract, transform, load (ETL), and business intelligence (BI) testing practices. It highlights key test types, architectural differences, automation techniques, and best practices to ensure data integrity and trustworthy analytics.

Author

Shweta Tripathi

LTIMindtree

Table of Contents

Executive Summary

Ensuring data integrity - The importance and practices of data warehouse testing

Data warehouse testing ensures data accuracy, reliability, and consistency in business reporting and analytics. As organizations rely more heavily on data-driven decisions, validating the correctness of data pipelines, especially in cloud-based architectures, has become critical.

01 What goes into a complete data warehouse testing strategy

ETL testing validates data extraction, transformation, and load processes. BI testing checks that business logic, dashboards, and reporting outputs are accurate and consistent. Together, they ensure end-to-end trust in the data pipeline.

02 How cloud data warehouses differ from traditional setups

Cloud-based data warehouses (e.g., Redshift, BigQuery, Azure Synapse) offer scalable, serverless storage and processing. They differ from traditional, on-premises systems in structure and flexibility, requiring a tailored testing approach. Extract, load, transform (ELT) is also gaining preference in cloud environments, where raw data is loaded first and transformed later.

03 What to focus on when testing cloud data pipelines

To test cloud DWHs effectively, teams should validate ingestion strategies, refresh mechanisms, data transformation logic, metadata, and performance. Emphasis on automation, regression, and version control is key.

A well-implemented testing strategy improves confidence in analytics, reduces the risk of reporting errors, and accelerates time to insight, making data a true business asset.

Why does testing your data warehouse matter?

In today's data-driven enterprises, ensuring warehouse data's reliability, accuracy, and consistency is non-negotiable. Data warehouse testing is pivotal in validating data pipelines, from source to report, to deliver trustworthy results that align with business rules and analytics goals.

This whitepaper is designed for QA leads, data engineers, data architects, and testing consultants working with traditional and cloud-based data warehouses. It provides guidance on how to build scalable testing strategies that ensure clean, complete, and governed data throughout the lifecycle.

Unlike traditional application testing, data warehouse testing covers the entire data pipeline, including ETL and business intelligence layers. It verifies that no data is lost or altered during transformation, and that BI dashboards and reports reference the correct data columns with precision, instilling trust in downstream decisions.

The paper also introduces ELT as an increasingly common pattern in cloud environments, where raw data is ingested first and then transformed within the warehouse itself. ETL and ELT require different validation techniques, which are explored in this guide.

ETL vs ELT - Key differences

Aspect	ETL (Extract, Transform, Load)	ETL testing focus	ELT (Extract, Load, Transform)	ELT testing focus
Order of operations	Transforms data before loading it into the warehouse	Verify the transformed data's accuracy and integrity before the target load	Loads raw data into the warehouse and transforms it afterward	Verify raw data integrity before transformation
Data formats	Typically handles clean, structured data	Validate table/column data types as per model specs	Accepts raw and mixed format data	Check data completeness and structure
Efficiency	Slower for large datasets (transformations before load)	Evaluate performance under load	Faster in cloud; loads raw data first	Ensure scalability and performance under transformation
Use cases	Ideal for highly governed, structured environments	Testing varies by project scope	Preferred in high-volume, cloud-native setups	Testing varies by use case and architecture
Popularity	Considered a traditional data pipeline model	Vary for projects	Increasingly common in cloud deployments	Vary for projects

Can ETL and ELT be implemented together?

Many organizations adopt a hybrid strategy that leverages both ETL and ELT processes. For instance, ETL may be used for high-priority data sources requiring strict transformation before loading.

At the same time, ELT is preferred for large or unstructured datasets where transformation is more efficient post-ingestion. This flexible approach allows organizations to optimize performance and maintain data integrity across diverse workloads.

ETL testing can be broadly classified into four main types

01

New system testing

Validates data sourced from multiple origins when a new data warehouse or pipeline is implemented.

02

Migration testing

Verifies the accuracy and completeness of data migrated from legacy systems or external sources to the data warehouse.

03

Change testing

Ensures that newly added data or structural changes to the data warehouse do not impact existing data or reporting.

04

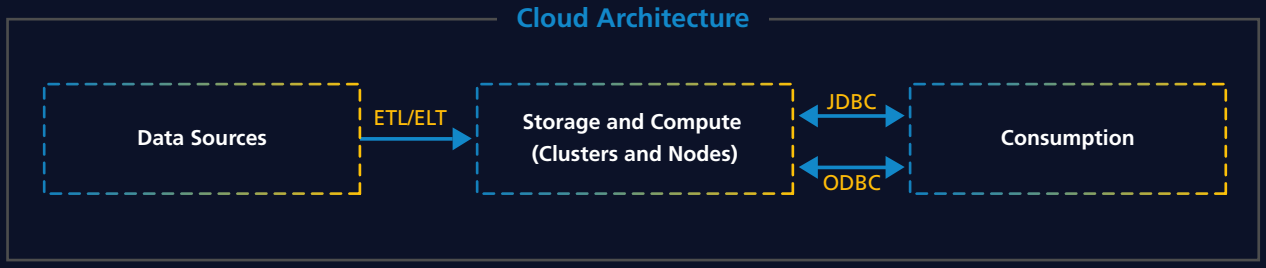
Report testing

Confirms that data displayed in BI reports is accurate by validating calculations, KPIs, and business logic against source data.

What is new in data warehousing: Cloud DWH

Cloud data warehouse

Cloud-based data warehouses like Amazon Redshift, Azure, and Google BigQuery offer scalable, serverless environments managed entirely by cloud providers. They eliminate the need for upfront infrastructure investment and provide flexible, on-demand resource allocation.



These platforms typically follow a three-tier architecture:

Data Sources

Storage and compute (clusters/nodes)

Consumption (BI dashboards, APIs, analytics tools)

Design evolution: The Medallion architecture

Cloud platforms often implement medallion architecture, a layered design pattern that improves data quality across stages:

Bronze
Raw, unprocessed data (no schema)

Silver
Cleaned and enriched data (schema applied)

Gold
Business-ready aggregates, metrics, and KPIs

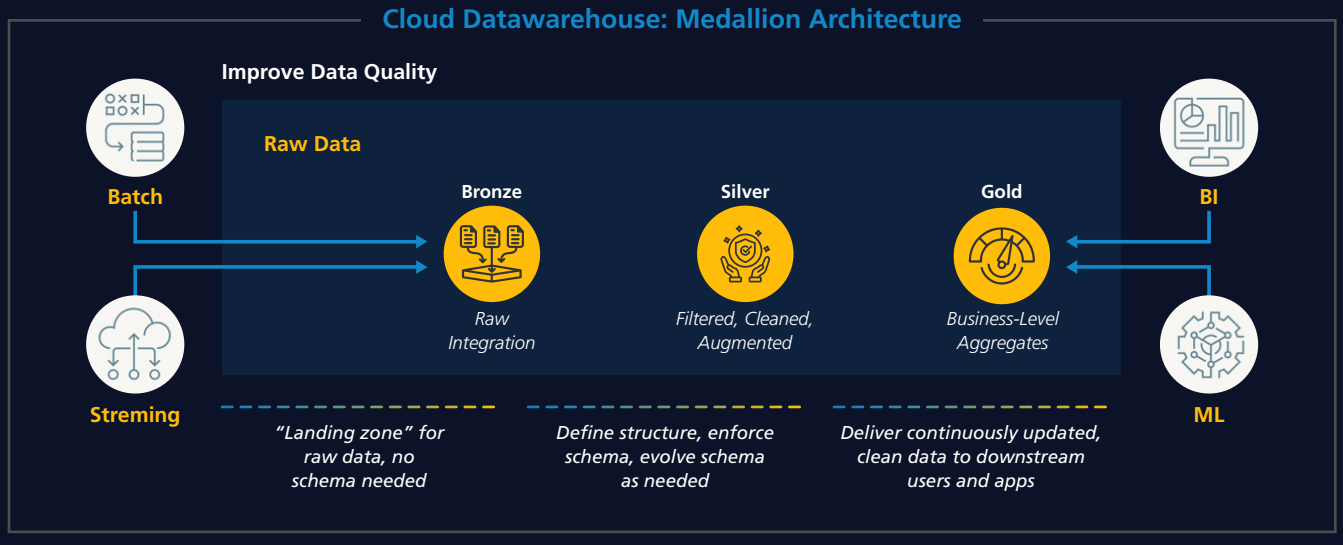


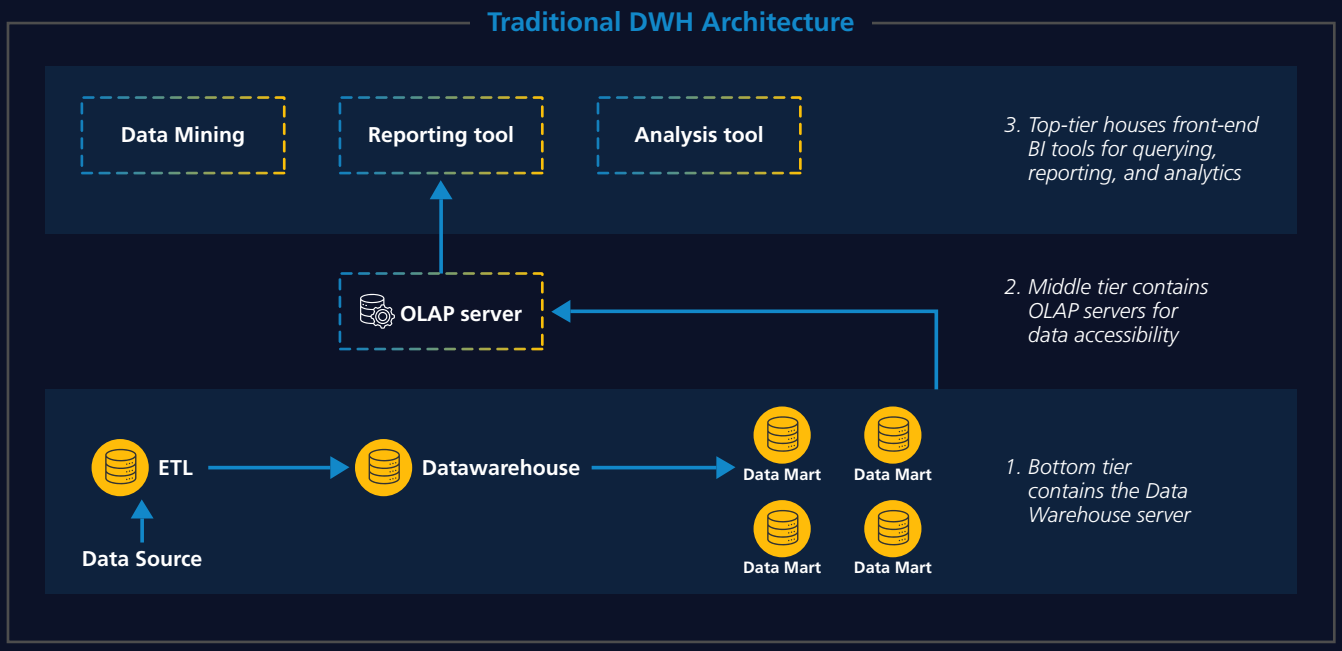
Image Source- https://conveyordata.com/portal-how-to-structure-data-for-self-service-data-teams?utm_source=linkedin-conveyor&utm_medium=referral

This "multi-hop" architecture enables more flexible, modular reporting and reduces time to insight.

Benefit Medallion-style Lakehouses allow organizations to store unstructured data without rigid schemas while enabling rich analytics downstream.

Traditional data warehousing: A quick recap

Traditional DWH environments require on-premises infrastructure such as servers, OLAP engines, and dedicated software layers. Architectures are structured in three tiers:



Bottom Data warehouse server and ETL processes

Middle OLAP servers for query access

Top BI tools for reporting, mining, and analytics

Data is pulled into a unified repository using ETL (Extract, Transform, and Load) tools.

Comparison: Traditional data warehouse vs cloud data warehouse

Cloud data warehouses	Cloud data warehouses
Scalable, serverless, cost-efficient	High initial cost and rigid infrastructure
Ideal for fast-growing or unstructured data	Best for governed, structured data pipelines
Supports real-time/near-real-time processing	Batch-heavy, slower pipelines
Potential cloud security concerns	High On-prem isolation reduces external riskscost and rigid infrastructure

Challenge

Key challenges in defining a cloud data testing strategy

Deriving an effective testing strategy for cloud-based data warehouses requires a deep understanding of source systems, data ingestion methods, and business transformation logic.

Testers must align validation scenarios with architectural complexity, data movement frequency, and downstream consumption needs. One critical area is data migration testing, especially during legacy-to-cloud transitions. Common challenges include:

01 **Data quality issues**

Poor assumptions, unvalidated conversions, and weak profiling can compromise data trustworthiness.

02 **Data mismatch issues**

Structural and format mismatches between legacy and target schemas may require remediation or redesign.

03 Data loss

Information may be lost during transfer due to system incompatibility or incomplete mapping.

04 Data volume

Migrating large datasets within tight downtime windows increases the risk of disruption.

Core ETL validation scenarios

To ensure data accuracy and pipeline reliability, the following validation techniques are typically applied across the ETL lifecycle:

Metadata & structure validation

01

Metadata testing

Ensures schema alignment with design specifications.

02

Data type & length checks

Verifies column definitions, formats, and truncation limits.

03

Constraint validation

Confirms the presence of keys, indexes, and referential integrity.

04

Naming standards testing

Validates naming consistency in schema metadata.

Data accuracy & completeness

01

Data completeness testing

Verifies all required records are loaded from the source to the target.

02

Record count validation

Compares row counts between systems to detect discrepancies.

03

Source-to-target comparison

Validates field-level accuracy across platforms.

04

Duplicate data checks

Identifies redundant or duplicate rows.

05

Data integrity checks

Flags orphaned or mislinked records in relational joins.

Transformation & reference validation

01

Data transformation testing

Confirms logic execution across transformations.

02

Lookup & reference data validation

Verifies lookup table mappings and conforming dimensions.

Incremental & performance testing

01

Incremental ETL validation

Ensures proper handling of delta loads and change data capture.

02

ETL performance testing

Measures batch or real-time pipeline performance to identify bottlenecks.

Detailed solution

Defining source systems and the ingestion testing approach

A Source can be a database, a SaaS-based application (an API endpoint), or a file storage with the data you want to analyze.

Understanding data origin and structure helps testers identify sources. This allows testers to know where the data comes from (e.g., databases, APIs, flat files like CSV or XLS) and how it's structured, which is essential for designing appropriate test cases.

Tester's responsibility

Creating effective test cases

Medallion-style Lakehouses allow organizations to store unstructured data without rigid schemas while enabling rich analytics downstream.

Understand data ingestion strategy, i.e., Extract layer test scenarios

Data ingestion can be done in two ways:

01 **Pull-based ingestion**

In pull-based ingestion, the ingestion code pulls data from the configured Source at a regular interval. This regular interval is termed the data load frequency.

The ETL tool/code fetches data from most SaaS and database sources configured with DB Tables or custom framework code.

02 **Push-based ingestion**

In push-based ingestion, the data lake's raw or landing layer acts as a receiver, and it is the source's responsibility to send or post data.

The following sources support the deferment of data ingestion:

- Google Drive
- DynamoDB
- S3
- Salesforce
- Databases (except log-based data fetching tasks)

Validation focus by ingestion method

Validation	Pull-based method	Push-based method
Data type validation	Ensure correct conversion during extraction and load (e.g., dates, numbers, strings)	Same as pull-based
Data format validation	Verify formats (e.g., dates, currencies) are consistent and compliant]	Same as pull-based
Data completeness	Check for missing or incomplete records during extraction	Check for missing or partial data in streamed payloads
Data mapping	Validate correct mapping from source to target fields	Not typically performed at the source level; validated post-ingestion only

Note

Mapping validation is typically not applicable at the source in push-based setups, since structure is often applied downstream.

Understand and identify test scenarios about data load frequency

What is data loading?

Data loading is the process of writing ingested data to the destination warehouse at defined intervals. These intervals are called data load frequencies and can vary based on system architecture, business requirements, and ingestion strategies.

Types of data load frequencies in cloud DWH

01 **Custom loading schedule**

A developer-defined schedule that runs at pre-set times (e.g., weekly, monthly). Often used for batch processing or archive loading.

02 **Real-time streaming or data stream model**

Data is ingested from a messaging queue and written directly to destination tables, requiring support for low-latency and temporary buffering.

03 **Daily scheduled loads**

Data is loaded at fixed points in the day, often aligned with business hours (e.g., every four hours, or every hour after peak time).

04 **Fixed interval syncs**

Loads occur every minute or hour based on a business-defined sync granularity. This is used for near-real-time updates when high availability is required.

Note

Streaming loads (real-time) may be written to S3 or staging layers before final ingestion, especially when latency exceeds message retention thresholds.

Applicable testing scenarios by load type

01 Validating custom load schedules

Key test goals

Confirm the completeness, timeliness, and accuracy of batch loads.

Catch scheduling conflicts or upstream dependencies.

Testing checklist

Understand the schedule- Identify timeframes, triggers, and batch frequency.

Identify dependencies- Assess linkage with other ETL jobs or sources.

Timeliness- Validate that data arrives within the defined window.

Completeness- Ensure all expected records are loaded.

Error handling- Test retry logic and failure capture.

Accuracy- Confirm data matches the source system.

02 Validating real-time load scenarios

Key test goals

Simulate varying data velocities and volumes.

Verify stream health and system responsiveness.

Testing checklist

Validate latency, message retention, and real-time alerting.

Confirm data accuracy, completeness, consistency, and timing.

Ensure real-time tools can notify stakeholders of data issues as they occur.

Types of data refresh methods in ETL pipelines

Data refresh refers to updating warehouse tables with new or changed data. Depending on business needs and pipeline architecture, data refresh can follow one of the following strategies:

01 Historical

This refers to one-time ingestion of all existing records in the source system as of the pipeline creation date (commonly referred to as Day 0 data). Once completed, historical loads are typically not repeated unless the pipeline is reset.

02 Full load method

In a full load strategy:

The destination table is completely reloaded at each run, often by dropping and recreating the table based on query results.

This eliminates the need to track changes via primary keys or timestamps.

Full loads are called "truncate and load" methods, especially in legacy systems.

However, this approach removes all existing data and constraints, which may require re-applying validation logic, indexes, or metadata definitions.

Use case Useful for staging tables or datasets where change tracking is not feasible.

03 Incremental method

Incremental refresh ingests only the newly created or updated records since the last successful run. It typically includes:

Insert/update/delete/upsert operations.

Use of audit columns like last_modified_date or CDC (change data capture).

Applicable testing

For Historical / Full load

- **Record count validation:**
  Ensure the number of records loaded into the target matches the source for a given load date.
- **Data coverage:**
  Validate that all required columns are present and populated.

- **Data type consistency:**
  Confirm column data types in the target mirror those in the source.
- **Primary key integrity:**
  Ensure uniqueness and match of identifiers across source and target.
- **Referential relationships:**
  Validate foreign key references and table joins post-load.
- **Duplicate check:**
  Identify any repeated rows violating uniqueness constraints.
- **Data volume consistency:**
  Confirm overall data footprint aligns with expectations.

For Incremental Load

Incremental load testing validates Change Data Capture (CDC) through timestamps, versioning, or update flags. Testing must focus on:

- **Timestamp-based testing logic**

1. Last load time:
   Establish the last successful load timestamp (e.g., March 10, 2025).
2. Source filter logic:
   Filter incoming data using fields like last_modified_date or created_date.
3. Validation:
   Ensure only records modified after the last load timestamp are ingested.

Example: If the last successful load was on March 10, 2025, the ETL process should load only records where last_modified_date > March 10, 2025.

- **Additional incremental validations**

1. Compare with load time:
   Check timestamps against the load window to confirm proper filtering.
2. Data sampling:
   Randomly compare loaded and source rows to identify anomalies.
3. Data reconciliation:
   Full field-level comparison for completeness of updates.

4. SQL-based assertions:
   Use queries to flag missing records, incorrect joins, or nulls in expected fields.

- **Verifying deletes in incremental loads**

Deletes are often the most overlooked part of incremental logic. They can be handled using:

1. Flag-based logic:
   Records are marked as "inactive" or soft-deleted using status columns.

2. Log table capture:
   A separate table logs deleted records by ID and timestamp, ensuring traceability.

Best practice: Incorporate delete validation into regression test cycles to prevent silent data loss.

Designing a data and report migration testing strategy

Data migration testing ensures data and reporting capabilities are accurately transferred from a legacy system to a modern platform. The goal is to validate that data integrity, completeness, and business logic are preserved while minimizing downtime and ensuring seamless user transition.

- **Preparation steps for migration testing**

1. Define source scope:
   Identify legacy data systems, volume, and critical domains.

2. Source-to-target mapping:
   Establish high-level mappings between old and new data structures across each business entity.

3. Field-level requirements:
   Document destination system specifications: field types, mandatory values, and validation rules.

4. Validation via mappings:
   Use mapping logic to ensure compliance at the field level (e.g., if a target field is mandatory, its source field must not be null).

5. Connection checks:
   Verify the migration platform's connectivity and compatibility between source and destination systems.

- **Data migration testing approach**

Approach 1:

1. Data:
   The comparison between legacy and migrated system data tables is apple to apple.

2. Reporting:
   Functional validation can be achieved using a legacy report vs a migrated data report comparison using the same set of filters and parameters.

Approach 2:

1. Compare migrated records to destination system-generated records and validate that migrated records are complete and of the appropriate context.

2. Validate the source system's data sets and queries.

3. Validate mappings between the source system fields and the destination system.

ETL testing best practices

To ensure your ETL validation process is scalable, accurate, and aligned with business needs, apply the following best practices, organized into three key focus areas:

Planning and strategy

Automating and versioning

Execution, validation, governance, and testing

Planning and strategy

01 Develop a comprehensive test plan

Define scope, objectives, timelines, and validation points across the ETL lifecycle. Align test coverage with your architecture roadmap and include batch and incremental scenarios.

02 Understand source systems and business logic

Deeply understand data lineage, transformation logic, and business-critical fields. This will improve test case relevance and ensure alignment with reporting requirements.

03 Establish requirement traceability

Maintain a traceability matrix (RTM) to map every test case to its requirement, ensuring complete lifecycle visibility and accountability.

Automating and versioning

04 Automate test cases

Use automation frameworks for regression, repetitive validations, and parallel execution. Automation improves consistency and accelerates test cycles.

05 Implement version control

Maintain version control for test scripts, datasets, and configuration files to support collaboration, rollback, and auditability.

Execution, validation, governance, and testing

06 Perform end-to-end testing

Validate the whole pipeline, from data extraction through transformation and loading, to ensure business logic integrity and completeness.

07 Validate transformation logic

Test transformation rules against business requirements to confirm output accuracy across all computed layers (staging, integration, reporting).

08 Conduct performance testing

Simulate load conditions to evaluate ETL throughput, latency, and system responsiveness under pressure.

09 Perform regular regression testing

Ensure new deployments or updates do not break existing data flows, table structures, or dependent reports.

10 Documentation and reporting

Maintain detailed logs of test results, known issues, and remediation steps. Documentation supports audits, performance reviews, and continuous improvement.

Automating ETL testing with PySpark framework (Databricks tech stack example)

Using the PySpark framework, source systems like Databricks, API, or CSV can be easily connected, and data can be read using PySpark commands. This data can be further stored in temporary views or data frames.

While all or selected columns can be added to source and target data frames, comparing these Python data frames expedites the validation process across layers and ensures up to 100% data validation.

Conclusion

Data warehouse testing is an indispensable process for modern businesses that rely on data analytics to derive complex insights. Companies can ensure their data's reliability, accuracy, and consistency by adopting comprehensive testing strategies and best practices.

A data warehouse testing strategy includes developing a comprehensive test plan, understanding the data and business logic, automating test cases, performing end-to-end testing, and conducting performance testing.

Regular regression testing is crucial to ensure new modifications do not introduce issues. By following these best practices, businesses can enhance their data analytics capabilities and make informed decisions based on trustworthy data.

About author

Shweta Tripathi
Data Agile Project Manager

With more than 16 years of experience, including 14 years in data warehouse (DWH) testing, Shweta Tripathi has had a versatile journey across various DWH testing projects. Her expertise includes production support profiles, operation acceptance testing, and functional ETL testing. Shweta is skilled at managing large-scale data, digital, and UX projects, streamlining release management, and enhancing team collaboration. Her certifications in SAFe 6.0 and Agile Data Scrum Master reflect her commitment to continuous improvement and adherence to industry best practices.

LTIMindtree

LTIMindtree is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 700 clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by 84,000+ talented and entrepreneurial professionals across more than 40 countries, LTIMindtree — a Larsen & Toubro Group company — solves the most complex business challenges and delivers transformation at scale. For more information, please visit **https://www.ltimindtree.com/**