Analysis And Batch Processing Of Transactional Data Using Cloud Computing

Prepared By:

Kudumula Sri Charan Reddy

TABLE OF CONTENTS

.

1. Abstract

Using AWS cloud services, the proposed project describes an effective batch analysis and ETL procedure for transactional data. Using Amazon S3 for storage, AWS Glue for ETL processes, and a data warehouse (like Amazon Redshift), the solution simplifies the extraction, transformation, and loading of raw transactional data.

Data migration between data repositories is made easier using AWS Glue, an extract, transform, and load service that is entirely handled by Amazon. It streamlines the process of finding, organising, and modifying data from a variety of sources. Proposed model is automated and scalable architecture guarantees flexibility with a wide range of data sources and formats. After being converted, the data is put into a data warehouse where it can be used with Amazon Athena for reliable batch analysis.

Athena to execute SQL queries on S3 data. Users just pay for the queries they run, so infrastructure management is not necessary. Large and complicated datasets are handled by Athena automatically scaling. SQL queries can be utilised by analysts to obtain significant insights from organised data.

Using Amazon Quick Sight to provide stakeholders with user-friendly dashboards and reports is the last phase in the visualisation process. Organisations are given a more efficient way to extract actionable knowledge from their transactional datasets thanks to this end-to-end pipeline, which guarantees scalability, automation, and cost effectiveness.

.

2. Introduction

Today's businesses create vast amounts of transactional data, which can be used to get useful insights in the age of digital transformation. To successfully process and analyse this data, a dependable and scalable system is required. This project utilizes Amazon Web Services (AWS) cloud capabilities to provide a comprehensive cloud-based technique for batch processing and transactional data analysis. The data pipeline is automated and streamlined through the use of AWS services, including Amazon S3 for storage, AWS Glue for Extract, Transform, Load, Amazon Redshift for data warehousing, and Amazon Athena for query processing.

This technology makes it easier for businesses to extract, convert, and load raw transactional data into data warehouses for large-scale SQL query analysis. Organizations can also utilize Amazon Quick Sight to develop engaging dashboards and reports for stakeholders, assuring the delivery of essential business information. The proposed solution provides a comprehensive pipeline for processing and analysing big datasets with low infrastructure administration, providing scalability, automation, and cost effectiveness.

.

3. Literature Review

The rapid increase in data volumes has driven extensive research into cloud-based solutions for data processing and management, particularly in transactional data analysis. Several studies have contributed foundational insights that have informed and shaped the design of this project.

Salesforce's

Salesforce's use of tools like Batch Apex and Queue-ableApex has demonstrated how cloud platforms can handle large volumes of data by processing it in manageable chunks (Smith et al., 2021). While Salesforce focuses on optimizing data operations within its ecosystem, these methods highlight the importance of scalable solutions in handling big data, a concept central to our project. By utilizing AWS services such as Amazon S3 and AWS Glue, our project builds upon this idea of processing large datasets efficiently, but with the added flexibility of a more diverse set of data sources and formats.

Amazon Web Services

Amazon Web Services (AWS) has become a dominant force in cloud computing, offering scalable, secure, and cost-effective solutions for a variety of industries (Jones et al., 2022). Research into AWS's cloud services, such as AWS Lake Formation and Multi-Factor Authentication, has shown how businesses can maintain high levels of data security and management. This has directly influenced the security considerations in our project, which utilizes AWS's infrastructure to ensure that data is securely stored and processed while providing robust scalability.

Accordia

Accordia et al. (2023) explored methods to optimize both cost and performance in cloud systems, identifying ways to improve efficiency through smart prediction and automation. This research has been instrumental in our project's focus on automation and cost-effectiveness, as we employ AWS Glue for automated ETL operations and Athena for pay-per-query data analysis. These services reduce the need for extensive infrastructure

management while keeping operational costs low, aligning with the optimization strategies suggested in prior studies.

Big data analytics

Moreover, research into big data analytics in the cloud (Lee et al., 2023) has highlighted the benefits of scalability, flexibility, and cost savings. This body of work emphasized future opportunities for real-time data processing, enhanced security, and the integration of machine learning. These findings have influenced our approach by encouraging the adoption of Amazon Athena for scalable SQL query execution and Amazon Quick Sight for real-time data visualization, ensuring that our project not only processes large datasets but also provides actionable insights quickly and efficiently.

.

4. Problem Identification & Objectives

The project described in the provided file is centred on the analysis and batch processing of transactional data using AWS cloud computing services. Here's a breakdown of Problem Identification and Objectives based on the project details:

Problem Identification

Organizations face challenges in handling and processing large transactional datasets due to:

1. **Scalability Issues:** Managing extensive datasets efficiently and scaling resources as data volume grows.
2. **Data Transformation Complexity**: Automated tools are needed to transform raw data into usable formats.
3. **Infrastructure Management Costs:** High costs and resource management challenges for storage and computation.
4. **Secure and Reliable Data Processing:** Requirement for secure, encrypted, and reliable systems to handle sensitive data.
5. **Insights and Visualization Need:** Difficulty in providing stakeholders with real-time, actionable insights from complex data.

Project Objectives

1. **Automate Data Processing:** Develop an end-to-end automated ETL process using AWS Glue, from data ingestion to transformation.
2. **Scalable Storage:** Utilize Amazon S3 for cost-effective, scalable storage of raw transactional data.
3. **Simplify Query Execution:** Leverage Amazon Athena for serverless SQL querying, reducing the need for infrastructure management.

4. **Enhanced Data Visualization:** Use Amazon QuickSight to create dashboards and reports, enabling decision-makers to access insights easily.

5.Existing System Overview

1. **On-Premises Infrastructure**:

   Traditional data processing systems often rely on on-premises hardware for data storage and computation. This approach requires significant capital investment in servers, storage devices, and networking infrastructure.

   As data volumes grow, organizations face challenges scaling their infrastructure quickly and efficiently, leading to resource shortages or underutilization.

2. **Manual ETL Processes**:

   The existing systems commonly employ manual or semi-automated ETL (Extract, Transform, Load) processes, which require dedicated personnel to manage data ingestion, transformation, and loading.

   These manual processes are prone to errors, time-consuming, and inefficient when handling large, complex datasets.

3. **Data Processing Delays**:

   Batch processing in traditional systems can be slow, especially when handling large volumes of data. Without optimized, parallel processing capabilities, existing systems struggle to process data in real-time or near real-time.

   This often results in delays in data availability for analysis, impacting decision-making timelines.

4. **Limited Flexibility and Scalability**:

   Traditional systems often lack flexibility in integrating various data formats and sources. Scaling up typically requires adding more hardware, which can be costly and time-intensive.

Integration with third-party tools for reporting and visualization is often limited, resulting in inadequate data analysis and visualization capabilities.

5. **High Maintenance and Operational Costs**:

   Maintaining and upgrading on-premises infrastructure is costly, requiring constant monitoring, physical security, and specialized personnel to manage hardware and software.

   Operational costs can escalate quickly as data processing needs increase, particularly when complex computations and storage expansion are necessary.

.

6.Proposed System

System Architecture Overview

1. **Data Ingestion and Storage** :

   Component: Amazon S3

   Role: Serves as the primary storage for raw transactional data. S3 provides scalable, cost-effective storage that can accommodate diverse data formats from multiple sources.

   Data Flow: Incoming raw data is directly ingested into Amazon S3 buckets, making it readily accessible for the ETL process.

2. **Data Extraction, Transformation, and Loading**:

   Component: AWS Glue

   Role: Acts as the ETL engine, automating the extraction, transformation, and loading of data from S3 into Amazon Redshift. AWS Glue also automates the process of data cataloging, classifying, and organizing.

   Data Flow: AWS Glue extracts raw data from S3, applies transformation scripts to standardize and clean the data, and then loads the transformed data into the Redshift data warehouse.

3. **Data Warehousing and Storag**:

   Component: Amazon Redshift

   Role: Provides a fully managed data warehouse where transformed data is stored, allowing complex querying and analysis of large datasets.

   Data Flow: Transformed data from AWS Glue is loaded into Redshift, where it can be queried for advanced analytics and reporting.

4. **Serverless Querying** :

   Component: Amazon Athena

   Role: Enables serverless SQL querying of data stored in Amazon S3, offering a cost-effective and infrastructure-free solution for querying unstructured data without loading it into Redshift.

   Data Flow: Users can query S3 data directly with SQL through Athena, retrieving data on-demand for specific analyses or ad-hoc reporting needs.

5. **Data Visualization** :

   Component: Amazon QuickSight

   Role: Provides interactive dashboards and reports for visualizing insights derived from data in Redshift or S3.

   Data Flow: QuickSight accesses data from Redshift and S3 (via Athena) to create user-friendly visualizations and dashboards, allowing stakeholders to gain actionable insights from transactional data.

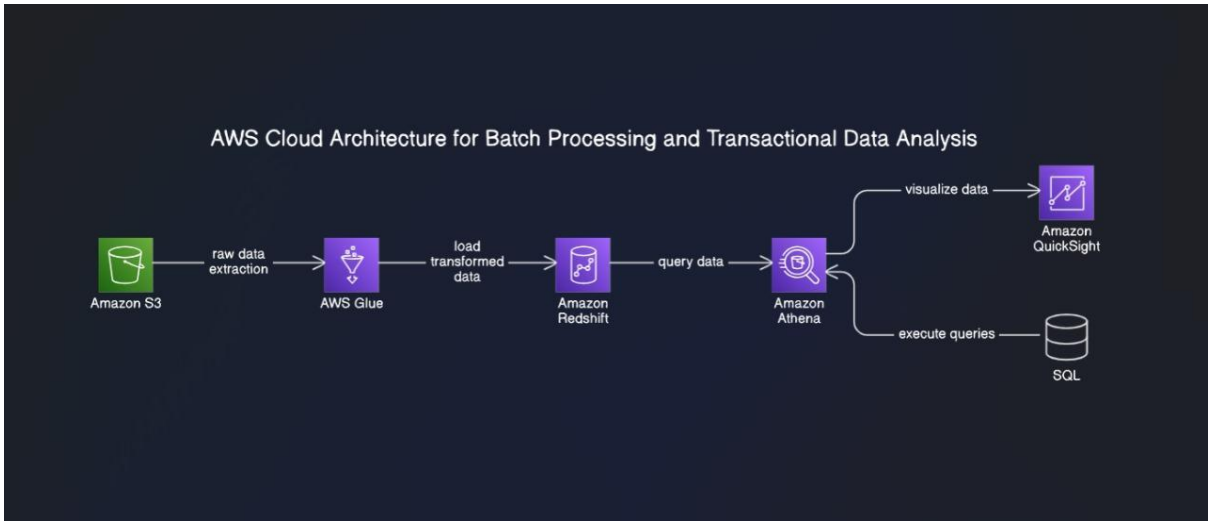
Data Flow Summary

1. **Data Ingestion**: Raw data is ingested and stored in Amazon S3.
2. **ETL Process**: AWS Glue extracts data from S3, transforms it, and loads it into Amazon Redshift.
3. **Data Storage**: Transformed data is warehoused in Amazon Redshift.
4. **Querying and Analysis**: SQL-based querying can be performed on Redshift or directly on S3 using Amazon Athena.
5. **Visualization and Reporting**: Amazon QuickSight visualizes the data for end-users.

Architecture Diagram

- **Amazon S3** as the initial data storage layer.
- **AWS Glue** positioned to connect S3 and transform data into **Amazon Redshift**.
- **Amazon Athena** providing direct query access to data in S3.
- **Amazon QuickSight** linked to both Redshift and Athena.

7.System Architecture



1. **Amazon S3**:
    - **Role**: Stores raw transactional data.
    - **Flow**: Data is extracted from various sources and stored here.
2. **AWS Glue**:
    - **Role**: Performs ETL operations on the data from S3.
    - **Flow**: Transforms the raw data and prepares it for analysis.
3. **Amazon Redshift**:
    - **Role**: Serves as the data warehouse for storing transformed data.
    - **Flow**: Data is loaded here for deeper analysis.
4. **Amazon Athena**:
    - **Role**: Allows for SQL-based querying on data stored in S3.
    - **Flow**: Users can execute queries directly on S3 data without needing to manage infrastructure.

.

5. **Amazon QuickSight**:
    - **Role**: Provides visualization and reporting capabilities.
    - **Flow**: Generates dashboards and reports for stakeholders based on processed data.
6. **Batch Processing**:
    - **Role**: Efficiently processes large volumes of data.
    - **Flow**: Enables the analysis of transactional datasets in batches, reducing operational costs.

Overall Workflow:

- Data is ingested into **Amazon S3**.
- **AWS Glue** performs the ETL process to transform the data.
- The transformed data is stored in **Amazon Redshift**.
- **Amazon Athena** allows for querying of both S3 and Redshift data.
- **Amazon QuickSight** visualizes the insights derived from the data.
- Batch processing ensures efficient handling of large datasets.

.

8.Tools/Technologies Used

1. **Amazon S3 (Simple Storage Service)**:

   Purpose: Primary storage solution for raw data ingestion.

   Functionality: Scalable, secure, and cost-effective storage with high durability. S3 supports a variety of data formats and enables easy data access for ETL operations.

   Benefits: Provides elastic storage for large datasets, facilitates data backup, and integrates seamlessly with other AWS services.

2. **AWS Glue**:

   Purpose: Manages the ETL (Extract, Transform, Load) process.

   Functionality: AWS Glue automates data discovery, categorization, transformation, and loading, reducing manual efforts in data preparation. It also includes a data catalog feature to organize and keep track of data.

   Benefits: Fully managed and serverless, providing scalability and flexibility to handle dynamic ETL requirements.

3. **Amazon Redshift**:

   Purpose: Acts as the main data warehouse.

   Functionality: Amazon Redshift is a fully managed data warehousing service optimized for online analytical processing (OLAP) queries on large datasets. It supports complex queries and analysis.

   Benefits: High performance, scalable storage and query capabilities, supporting quick access to structured and transformed data for analysis.

.

4. **Amazon Athena**:

   Purpose: Serverless querying tool for data in S3.

   Functionality: Allows SQL-based querying of raw data directly in S3, eliminating the need for ETL into Redshift for ad-hoc analysis.

   Benefits: Serverless and pay-per-query, reducing costs and infrastructure management efforts. Ideal for on-demand analysis of unstructured or semi-structured data.

5. **Amazon QuickSight**:

   Purpose: Data visualization and reporting tool.

   Functionality: Enables the creation of interactive dashboards and visualizations, allowing stakeholders to view insights in an easy-to-understand format.

   Benefits: Provides real-time, visually rich reporting and integrates with Redshift

6. **AWS Identity and Access Management (IAM)**:

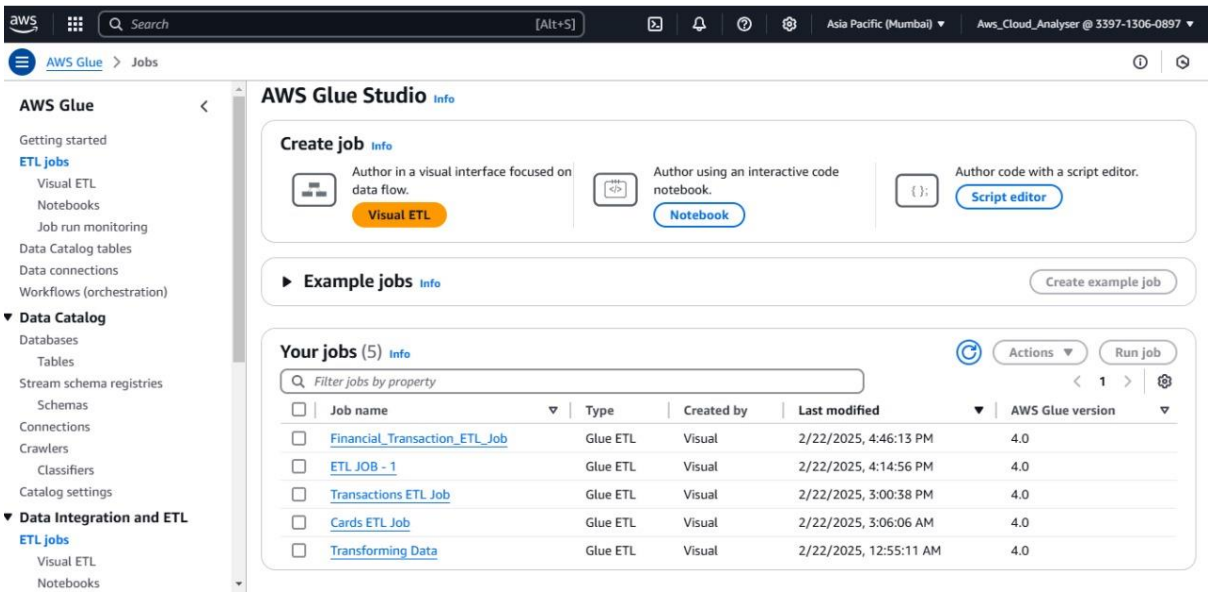
   Purpose: Security and access management.

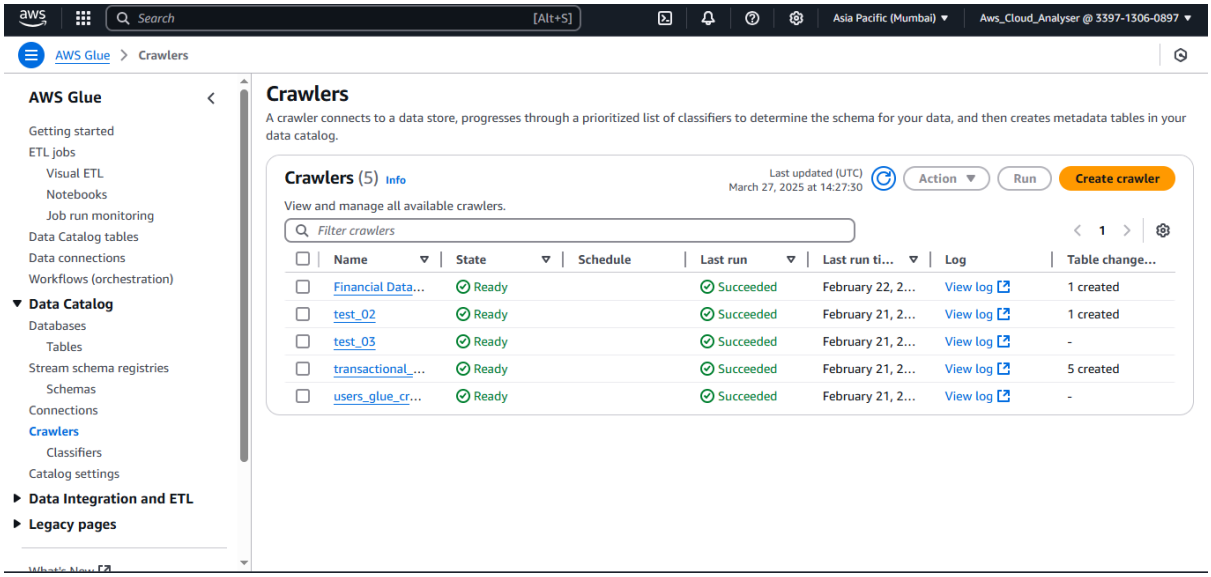
   Functionality: Manages user access and permissions, enforcing role-based access controls to secure sensitive data and services.

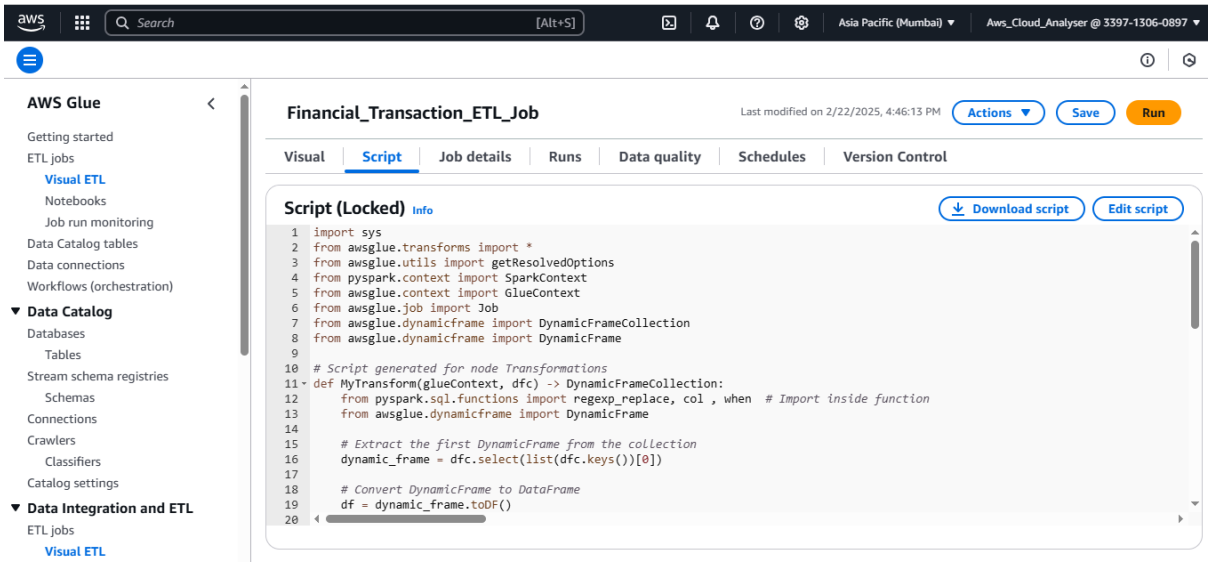
   Benefits: Ensures that only authorized users can access specific AWS resources, providing a high level of security across the system.

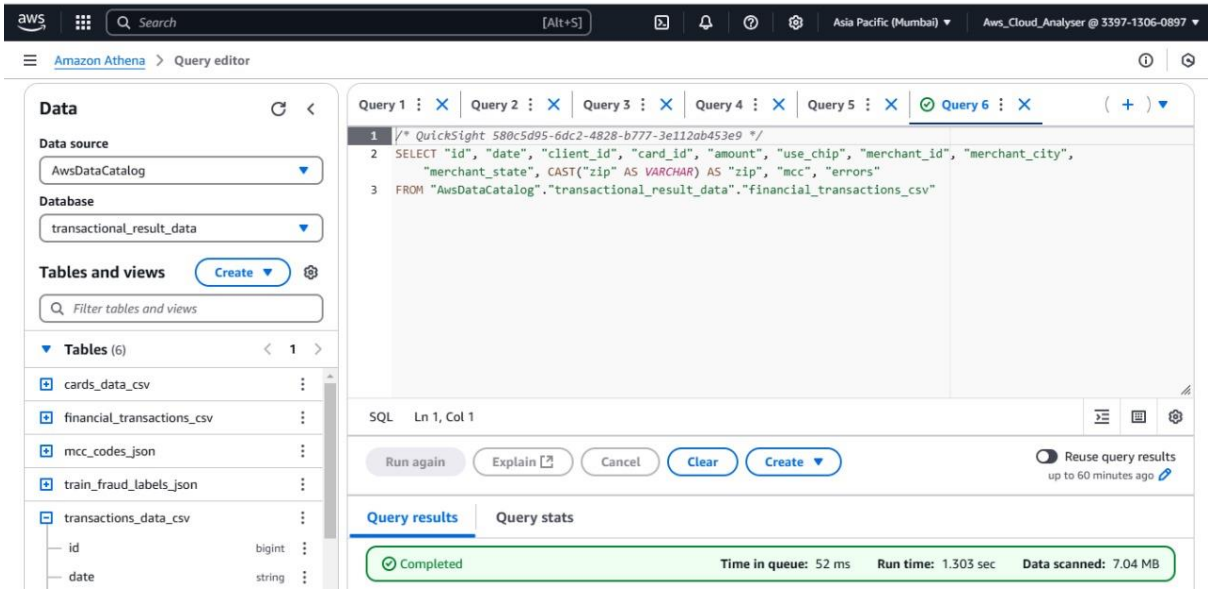
.

8.Implementation









ETL Jobs:

Financial_Transaction_ETL_Job.py

import sys

from awsglue.transforms import *

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

```
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrameCollection
from awsglue.dynamicframe import DynamicFrame


# Script generated for node Transformations
def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql.functions import regexp_replace, col , when  # Import inside function
    from awsglue.dynamicframe import DynamicFrame

    # Extract the first DynamicFrame from the collection
    dynamic_frame = dfc.select(list(dfc.keys())[0])

    # Convert DynamicFrame to DataFrame
    df = dynamic_frame.toDF()

    # Remove the '$' sign
   df = df.withColumn("amount",
                      regexp_replace(col("amount"), "[$]", ""))  #  Fixed syntax
    df = df.withColumn(
        "errors", when((col("errors").isNull()) | (col("errors") == ""),
"ERRORLESS").otherwise(col("errors"))
    )
```

```
    # Convert back to DynamicFrame
    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")  #  Added alias

    # Return the modified frame as a DynamicFrameCollection
    return DynamicFrameCollection({"transformed": transformed_dynamic_frame},
glueContext)

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)



AWSGlueDataCatalog_node1740222671880 =
glueContext.create_dynamic_frame.from_catalog(database="transactional_result_data",
table_name="financial_transactions_csv",
transformation_ctx="AWSGlueDataCatalog_node1740222671880")


Transformations_node1740222710697 = MyTransform(glueContext,
DynamicFrameCollection({"AWSGlueDataCatalog_node1740222671880":
AWSGlueDataCatalog_node1740222671880}, glueContext))
```

```
SelectFromÇollection_node1740222816732 =
SelectFromCollection.apply(dfc=Transformations_node1740222710697,
key=list(Transformations_node1740222710697.keys())[0],
transformation_ctx="SelectFromCollection_node1740222816732")



ChangeSchema_node1740222829746 =
ApplyMapping.apply(frame=SelectFromCollection_node1740222816732, mappings=[("id",
"long", "id", "long"), ("date", "string", "date", "string"), ("client_id", "long", "client_id",
"long"), ("card_id", "long", "card_id", "long"), ("amount", "string", "amount", "double"),
("use_chip", "string", "use_chip", "string"), ("merchant_id", "long", "merchant_id", "long"),
("merchant_city", "string", "merchant_city", "string"), ("merchant_state", "string",
"merchant_state", "string"), ("zip", "double", "zip", "double"), ("mcc", "double", "mcc",
"double"), ("errors", "string", "errors", "string")],
transformation_ctx="ChangeSchema_node1740222829746")


AWSGlueDataCatalog_node1740222920516 =
glueContext.write_dynamic_frame.from_catalog(frame=ChangeSchema_node174022282974
6, database="transactional_result_data", table_name="financial_transactions_csv",
additional_options={"enableUpdateCatalog": True, "updateBehavior":
"UPDATE_IN_DATABASE"},
transformation_ctx="AWSGlueDataCatalog_node1740222920516")


job.commit()
```

ETL JOB - 1.py:

```
import sys   .
from aws glue.transforms import *
from aws glue.utils import getResolvedOptions
from pyspark.context import SparkContext
from aws glue.context import GlueContext
from aws glue.job import Job
from aws glue. dynamic frame import DynamicFrameCollection
from aws glue. dynamic frame import DynamicFrame

def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql.functions import regexp_replace, col , when , trim # Import inside function
    from awsglue.dynamicframe import DynamicFrame

    # Extract the first DynamicFrame from the collection
    dynamic_frame = dfc.select(list(dfc.keys())[0])

    # Convert DynamicFrame to DataFrame
    df = dynamic_frame.toDF()

    # Remove the '$' sign
    df = df.withColumn("amount",
                       regexp_replace(col("amount"), "[$]", ""))  #  Fixed syntax
```

```
    df = df.withColumn(
        "errors", when((col("errors").isNull()) | (col("errors") == ""),
"ERRORLESS").otherwise(col("errors"))
    )

    mode_df = df.groupBy("merchant_state").count().orderBy(col("count").desc()).limit(1)
    mode_row = mode_df.collect()  # Collect result

    if mode_row:  # Ensure mode exists
        mode_value = mode_row[0]["merchant_state"]
    else:
        mode_value = "UNKNOWN"  # Default if no mode exists

    # Replace NULL values in 'merchant_state' with the mode value
    # Replace NULL, empty strings, "None", "NA", and "null" values
    df = df.withColumn(
        "merchant_state",
        when(
            (col("merchant_state").isNull()) |  # Check for NULL
            (trim(col("merchant_state")) == "") |  # Check for empty strings
            (col("merchant_state").isin("None", "NA", "null")),  # Handle common placeholders
            mode_value
        ).otherwise(col("merchant_state"))
```

) .

```
    # Convert back to DynamicFrame
    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")  # □ Added alias

    # Return the modified frame as a DynamicFrameCollection
    return DynamicFrameCollection({"transformed": transformed_dynamic_frame},
glueContext)

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)



AWSGlueDataCatalog_node1740218886598 =
glueContext.create_dynamic_frame.from_catalog(database="transactional_result_data",
table_name="transactions_data_csv",
transformation_ctx="AWSGlueDataCatalog_node1740218886598")


Transformations_node1740218923162 = MyTransform(glueContext,
DynamicFrameCollection({"AWSGlueDataCatalog_node1740218886598":
AWSGlueDataCatalog_node1740218886598}, glueContext))
```

.

```
SelectFromCollection_node1740218968248 =
SelectFromCollection.apply(dfc=Transformations_node1740218923162,
key=list(Transformations_node1740218923162.keys())[0],
transformation_ctx="SelectFromCollection_node1740218968248")
```

```
Change schema_node1740218973162 =
Applymapping.apply(frame=SelectFromCollection_node1740218968248, mappings=[("id",
"long", "id", "long"), ("date", "string", "date", "string"), ("client_id", "long", "client_id",
"long"), ("card_id", "long", "card_id", "long"), ("amount", "string", "amount", "string"),
("use_chip", "string", "use_chip", "string"), ("merchant_id", "long", "merchant_id", "long"),
("merchant_city", "string", "merchant_city", "string"), ("merchant_state", "string",
"merchant_state", "string"), ("zip", "double", "zip", "double"), ("mcc", "long", "mcc",
"long"), ("errors", "string", "errors", "string")],
transformation_ctx="ChangeSchema_node1740218973162")
```

```
AWS Glue DataCatalog_node1740219677959 =
glueContext.write_dynamic_frame.from_catalog(frame=ChangeSchema_node174021897316
2, database="transactional_result_data", table_name="transactions_data_csv",
additional_options={"enableUpdateCatalog": True, "updateBehavior":
"UPDATE_IN_DATABASE"},
transformation_ctx="AWSGlueDataCatalog_node1740219677959")
```

```
job.commit()
```

Transactions ETL Job.py :

```
import sys  .
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrameCollection
from awsglue.gluetypes import *
from awsglue.dynamicframe import DynamicFrame
from awsglue import DynamicFrame


def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql.functions import regexp_replace, col  # Import inside function
    from awsglue.dynamicframe import DynamicFrame

    # Extract the first DynamicFrame from the collection
    dynamic_frame = dfc.select(list(dfc.keys())[0])

    # Convert DynamicFrame to DataFrame
    df = dynamic_frame.toDF()

    # Remove the '$' sign
    df = df.withColumn("amount",
                       regexp_replace(col("amount"), "[$]", ""))  # □ Fixed syntax

    # Convert back to DynamicFrame
```

```
    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")  # □ Added alias


    # Return the modified frame as a DynamicFrameCollection

    return DynamicFrameCollection({"transformed": transformed_dynamic_frame},
glueContext)

# Script generated for node Filling empty/null values in error column with "ERRORLESS"

def MyTransform(glueContext, dfc) -> DynamicFrameCollection:

    from pyspark.sql.functions import when, col

    from awsglue.dynamicframe import DynamicFrame


    # Extract the first DynamicFrame from the collection

    dynamic_frame = dfc.select(list(dfc.keys())[0])


    # Convert DynamicFrame to DataFrame

    df = dynamic_frame.toDF()


    # Replace empty or NULL values in the "errors" column with "ERRORLESS"

    df = df.withColumn(

        "errors", when((col("errors").isNull()) | (col("errors") == ""),
"ERRORLESS").otherwise(col("errors"))

    )


    # Convert back to DynamicFrame

    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")


    # Return as a DynamicFrameCollection
```

```
    return DynamicFrameCollection({"transformed": transformed_dynamic_frame}, glueContext)

def _find_null_fields(ctx, schema, path, output, nullStringSet, nullIntegerSet, frame):
    if isinstance(schema, StructType):
        for field in schema:
            new_path = path + "." if path != "" else path
            output = _find_null_fields(ctx, field.dataType, new_path + field.name, output, nullStringSet, nullIntegerSet, frame)
    elif isinstance(schema, ArrayType):
        if isinstance(schema.elementType, StructType):
            output = _find_null_fields(ctx, schema.elementType, path, output, nullStringSet, nullIntegerSet, frame)
    elif isinstance(schema, NullType):
        output.append(path)
    else:
        x, distinct_set = frame.toDF(), set()
        for i in x.select(path).distinct().collect():
            distinct_ = i[path.split('.')[-1]]
            if isinstance(distinct_, list):
                distinct_set |= set([item.strip() if isinstance(item, str) else item for item in distinct_])
            elif isinstance(distinct_, str) :
                distinct_set.add(distinct_.strip())
            else:
                distinct_set.add(distinct_)
        if isinstance(schema, StringType):
            if distinct_set.issubset(nullStringSet):
                output.append(path)
```

```
    elif isinstance(schema, IntegerType) or isinstance(schema, LongType) or
isinstance(schema, DoubleType):
        if distinct_set.issubset(nullIntegerSet):
            output.append(path)
  return output


def drop_nulls(glueContext, frame, nullStringSet, nullIntegerSet, transformation_ctx) ->
DynamicFrame:
  nullColumns = _find_null_fields(frame.glue_ctx, frame.schema(), "", [], nullStringSet,
nullIntegerSet, frame)
  return DropFields.apply(frame=frame, paths=nullColumns,
transformation_ctx=transformation_ctx)



  from pyspark.sql.functions import regexp_replace, col, when   # Import inside function
  from awsglue.dynamicframe import DynamicFrame

  # Extract the first DynamicFrame from the collection
  dynamic_frame = dfc.select(list(dfc.keys())[0])

  # Convert DynamicFrame to DataFrame
  df = dynamic_frame.toDF()

  # Remove the '$' sign
  df = df.withColumn("amount",
                     regexp_replace(col("amount"), "[$]", ""))  #  Fixed syntax

  # Replace empty or NULL values in the "errors" column with "ERRORLESS"
```

```
    df = df.withColumn(
        "errors", when((col("errors").isNull()) | (col("errors") == ""),
"ERRORLESS").otherwise(col("errors"))
    )

    # Convert back to DynamicFrame
    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")  # □ Added alias

    # Return the modified frame as a DynamicFrameCollection
    return DynamicFrameCollection({"transformed": transformed_dynamic_frame},
glueContext)

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)



AWS Glue DataCatalog_node1740208363528 =
glueContext.create_dynamic_frame.from_catalog(database="transactional_result_data",
table_name="transactions_data_csv",
transformation_ctx="AWSGlueDataCatalog_node1740208363528")


# Script generated for node Removing $  sign

Removingsign_node1740208651059 = MyTransform(glueContext,
DynamicFrameCollection({"AWSGlueDataCatalog_node1740208363528":
AWSGlueDataCatalog_node1740208363528}, glueContext))
```

.

```
CustomTransform_node1740216099253 = MyTransform(glueContext,
DynamicFrameCollection({"AWSGlueDataCatalog_node1740208363528":
AWSGlueDataCatalog_node1740208363528}, glueContext))
```

```
FillingemptynullvaluesinerrorcolumnwithERRORLESS_node1740209860059 =
MyTransform(glueContext,
DynamicFrameCollection({"SelectFromCollection_node1740208769073":
SelectFromCollection_node1740208769073}, glueContext))
```

```
# Script generated for node Drop Null Fields

DropNullFields_node1740216538034 = drop_nulls(glueContext,
frame=SelectFromCollection_node1740216558298, nullStringSet={"", "null"},
nullIntegerSet={}, transformation_ctx="DropNullFields_node1740216538034")
```

```
SelectFromCollection_node1740215841919 =
SelectFromCollection.apply(dfc=FillingemptynullvaluesinerrorcolumnwithERRORLESS_no
de1740209860059,
key=list(FillingemptynullvaluesinerrorcolumnwithERRORLESS_node1740209860059.keys(
))[0], transformation_ctx="SelectFromCollection_node1740215841919")


job.commit()
```

Cards ETL Job.py

```
import sys

from awsglue.transforms import *
```

```
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrameCollection
from awsglue.dynamicframe import DynamicFrame


    from pyspark.sql.functions import regexp_replace, col  # Import inside function
    from awsglue.dynamicframe import DynamicFrame

    # Extract the first DynamicFrame from the collection
    dynamic_frame = dfc.select(list(dfc.keys())[0])

    # Convert DynamicFrame to DataFrame
    df = dynamic_frame.toDF()

    # Remove the '$' sign
    df = df.withColumn("credit_limit",
                       regexp_replace(col("credit_limit"), "[$]", ""))  #  Fixed syntax

    # Convert back to DynamicFrame
    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext,
"transformed_df")  #  Added alias
```

.

```
    # Return the modified frame as a DynamicFrameCollection

    return DynamicFrameCollection({"transformed": transformed_dynamic_frame},
glueContext)

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)




CustomTransform_node1740167981351 = MyTransform(glueContext,
DynamicFrameCollection({"AmazonS3_node1740167087745":
AmazonS3_node1740167087745}, glueContext))



SelectFromCollection_node1740168152998 =
SelectFromCollection.apply(dfc=CustomTransform_node1740167981351,
key=list(CustomTransform_node1740167981351.keys())[0],
transformation_ctx="SelectFromCollection_node1740168152998")
```

ChangeSchema_node1740167875839 = ApplyMapping.apply(frame=SelectFromCollection_node1740168152998, mappings=[("id", "string", "id", "long"), ("client_id", "string", "client_id", "long"), ("card_brand", "string", "card_brand", "string"), ("card_type", "string", "card_type", "string"), ("card_number", "string", "card_number", "long"), ("expires", "string", "expires", "string"), ("cvv", "string", "cvv", "long"), ("has_chip", "string", "has_chip", "string"), ("num_cards_issued", "string", "num_cards_issued", "long"), ("credit_limit", "string", "credit_limit", "long"), ("acct_open_date", "string", "acct_open_date", "string"), ("year_pin_last_changed", "string", "year_pin_last_changed", "long"), ("card_on_dark_web", "string", "card_on_dark_web", "string")], transformation_ctx="ChangeSchema_node1740167875839")

AWSGlueDataCatalog_node1740173744394 = glueContext.write_dynamic_frame.from_catalog(frame=ChangeSchema_node1740167875839, database="transactional_result_data", table_name="cards_data_csv", transformation_ctx="AWSGlueDataCatalog_node1740173744394")

job.commit()

Transforming Data.py:

import sys

```
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrameCollection
from awsgluedq.transforms import EvaluateDataQuality
from awsglue.dynamicframe import DynamicFrame

def MyTransform(glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql.functions import regexp_replace, col  # Import inside function
    from awsglue.dynamicframe import DynamicFrame

    # Extract the first DynamicFrame from the collection
    dynamic_frame = dfc.select(list(dfc.keys())[0])

    # Convert DynamicFrame to DataFrame
    df = dynamic_frame.toDF()



    # Remove the '$' sign
```

```
    df = df.withColumn("per_capita_income", regexp_replace(col("per_capita_income"), "[$]", ""))

    df = df.withColumn("yearly_income", regexp_replace(col("per_capita_income"), "[$]", ""))

    df = df.withColumn("total_debt", regexp_replace(col("per_capita_income"), "[$]", ""))


    # Convert back to DynamicFrame

    transformed_dynamic_frame = DynamicFrame.fromDF(df, glueContext)


    # Return the modified frame as a DynamicFrameCollection

    return DynamicFrameCollection({"transformed": transformed_dynamic_frame}, glueContext)

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)
```

```
# Default ruļeset used by all target nodes with data quality enabled

DEFAULT_DATA_QUALITY_RULESET = """

   Rules = [

      ColumnCount > 0

   ]

Removingsign_node1740151309242 = MyTransform(glueContext,
DynamicFrameCollection({"AmazonS3_node1740149510116":
AmazonS3_node1740149510116}, glueContext))



SelectFromCollection_node1740152460237 =
SelectFromCollection.apply(dfc=Removingsign_node1740151309242,
key=list(Removingsign_node1740151309242.keys())[0],
transformation_ctx="SelectFromCollection_node1740152460237")



ChangeSchema_node1740165044253 =
ApplyMapping.apply(frame=SelectFromCollection_node1740152460237, mappings=[("id",
"string", "id", "long"), ("current_age", "string", "current_age", "long"), ("retirement_age",
"string", "retirement_age", "long"), ("birth_year", "string", "birth_year", "long"),
("birth_month", "string", "birth_month", "long"), ("gender", "string", "gender", "string"),
("address", "string", "address", "string"), ("latitude", "string", "latitude", "double"),
("longitude", "string", "longitude", "double"), ("per_capita_income", "string",
"per_capita_income", "bigint"), ("yearly_income", "string", "yearly_income", "bigint"),
("total_debt", "string", "total_debt", "bigint"), ("credit_score", "string", "credit_score",
"long"), ("num_credit_cards", "string", "num_credit_cards", "long")],
transformation_ctx="ChangeSchema_node1740165044253")



EvaluateDataQuality().process_rows(frame=ChangeSchema_node1740165044253,
ruleset=DEFAULT_DATA_QUALITY_RULESET,
```

```
publishing_options={"dataQualityEvaluationContext":
"EvaluateDataQuality_node1740165114710", "enableDataQualityResultsPublishing": True},
additional_options={"dataQualityResultsPublishing.strategy": "BEST_EFFORT",
"observations.scope": "ALL"})

AmazonS3_node1740165465960 =
glueContext.write_dynamic_frame.from_options(frame=ChangeSchema_node1740165044253, connection_type="s3", format="json", connection_options={"path": "s3://transactional-data-bucket/transaction_data_results/", "partitionKeys": []},
transformation_ctx="AmazonS3_node1740165465960")


job.commit()
```

9. Results & Discussion

The project creates a system that can expand easily and handle large data volumes automatically using AWS cloud services. It uses Amazon S3 for data storage, AWS Glue for ETL (Extract, Transform, Load) tasks, and Amazon Athena for executing queries without needing separate servers. This setup addresses issues like managing large-scale data, improving processing speeds, and better infrastructure management.

Amazon QuickSight is added for interactive dashboards and real-time analytics, giving business stakeholders valuable insights. Security measures like IAM policies, encryption with AWS KMS, and monitoring through AWS CloudWatch ensure data safety and compliance with security requirements. Compared to traditional on-site data processing, this new AWS system shows major improvements in:

1. Scalability: By using the cloud, hardware limitations are eliminated, allowing smooth growth as data volume increases.

2. Automation: AWS Glue automates ETL processes, cutting down on manual work and reducing errors.

3. Cost Efficiency: The pay-as-you-go pricing model reduces costs by avoiding the need for large hardware investments.

4. Processing Speed: Amazon Athena's serverless querying significantly speeds up data analysis compared to traditional SQL systems.

5. Security and Reliability: AWS offers built-in encryption, access controls, and monitoring to maintain data security and prevent unauthorized access.

This study proves that the new system enhances data processing efficiency and reduces operational workload, making it ideal for businesses aiming to enhance their data pipelines.

10.Conclusion & Future Scope

The proposed system for analyzing and batch processing transactional data offers a comprehensive, cloud-based solution designed to meet the demands of modern data-driven organizations. Using AWS services such as Amazon S3 for scalable data storage, AWS Glue for automated ETL processing, Amazon Redshift for efficient data warehousing, Amazon Athena for serverless querying, and Amazon QuickSight for insightful visualizations, the system provides a streamlined and effective way to handle large datasets as well.

This architecture addresses the key challenges faced by traditional data processing systems, including scalability, automation, cost efficiency, and security. Through automation, the system minimizes manual intervention, improving data accuracy and consistency. It also allows for cost-effective processing, as users pay only for the resources they use, making it particularly beneficial for organizations looking to optimize expenses without sacrificing functionality.

Furthermore, robust security measures, including IAM, KMS, and SSL, ensure that data is protected at every stage, complying with industry standards for data privacy and security. The system's flexibility and scalability also ensure that it can adapt to evolving data processing needs and growth in data volume.

11.References

[1] Yang, C., Huang, Q., Li, Z., Liu, K., & Hu, F. (2016). *Big Data and Cloud Computing: Innovation Opportunities and Challenges*. *International Journal of Digital Earth*, 10(2), 122-139

[2] Muniswamaiah, M., Agerwala, T., & Tappert, C. (2014). *Big Data in Cloud Computing: Review and Opportunities*. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S0306437914001288

[3] Marańda, W., Poniszewska-Marańda, A., & Szymczyńska, M. (2022). *Data Processing in Cloud Computing Model on the Example of Salesforce Cloud*. *Information*, 13(2), 85. Retrieved from https://www.mdpi.com/2078-2489/13/2/85

[4] Berisha, B., & Mëziu, E. (2022). *Big Data Analytics in Cloud Computing: An Overview*.

[5] Diouf, P. S., & Ndiaye, S. (2018). *Variety of Data in the ETL Processes in the Cloud: State of the Art*.

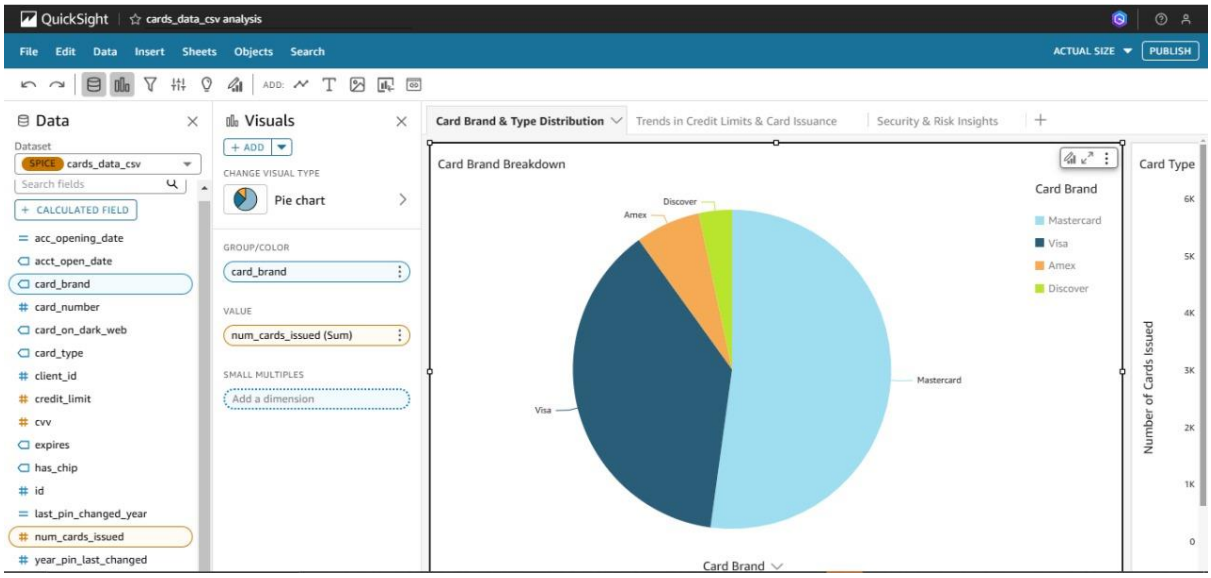
[6] Liu, Y., Xu, H., & Lau, W. C. (2023). *Cloud Configuration Optimization for Recurring Batch-Processing Applications*. Retrieved from https://www.researchgate.net/publication/368614466_Cloud_Configuration_Optimization_for_Recurring_Batch-Processing_Applications

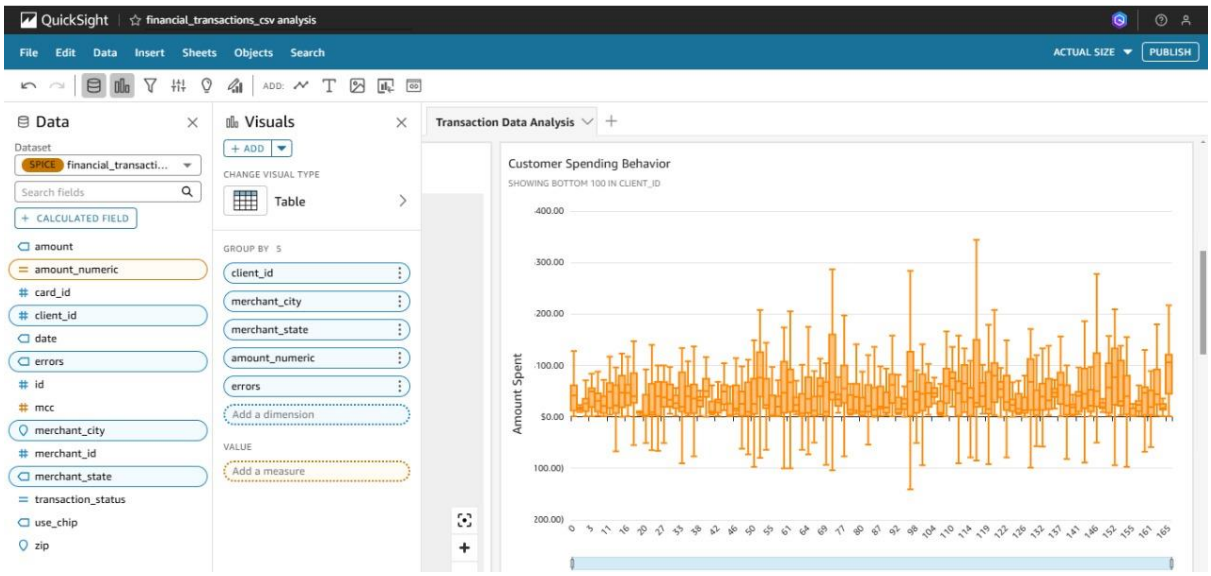
[7] Singh, T. (2021). *The Effect of Amazon Web Services (AWS) on Cloud Computing*. Retrieved from https://www.researchgate.net/publication/356809704_The_effect_of_Amazon_Web_Services_AWS_on_Cloud-Computing

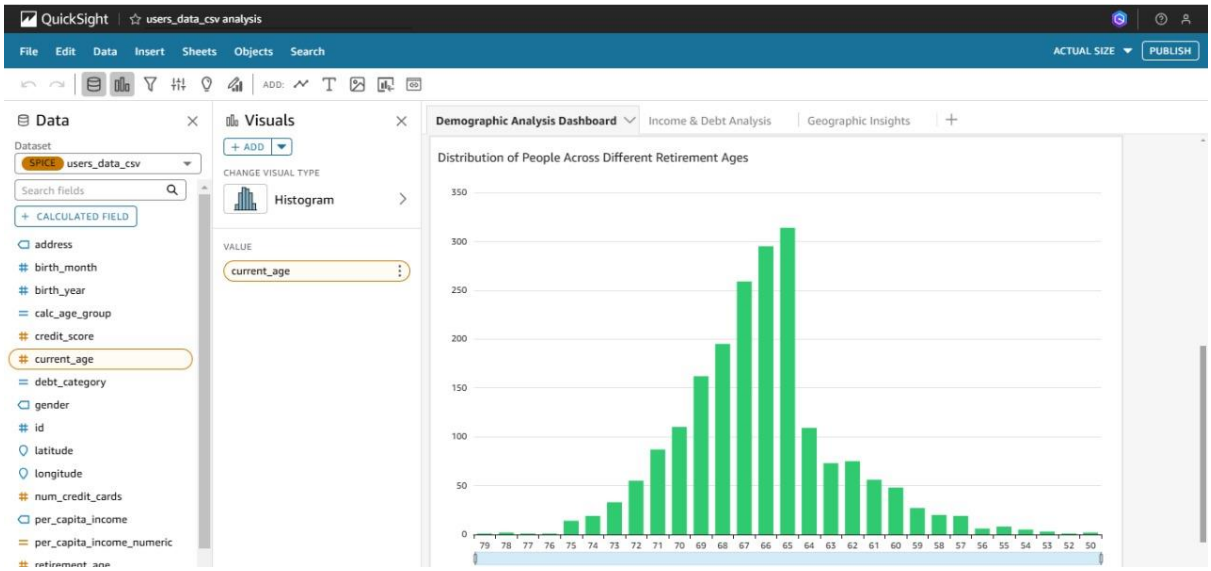
[8] Benjelloun, S., El Mehdi El Aissi, M., & Loukili, Y. (2021). *Big Data Processing: Batch-based Processing and Stream-based Processing*. Retrieved from https://ieeexplore.ieee.org/document/9268684

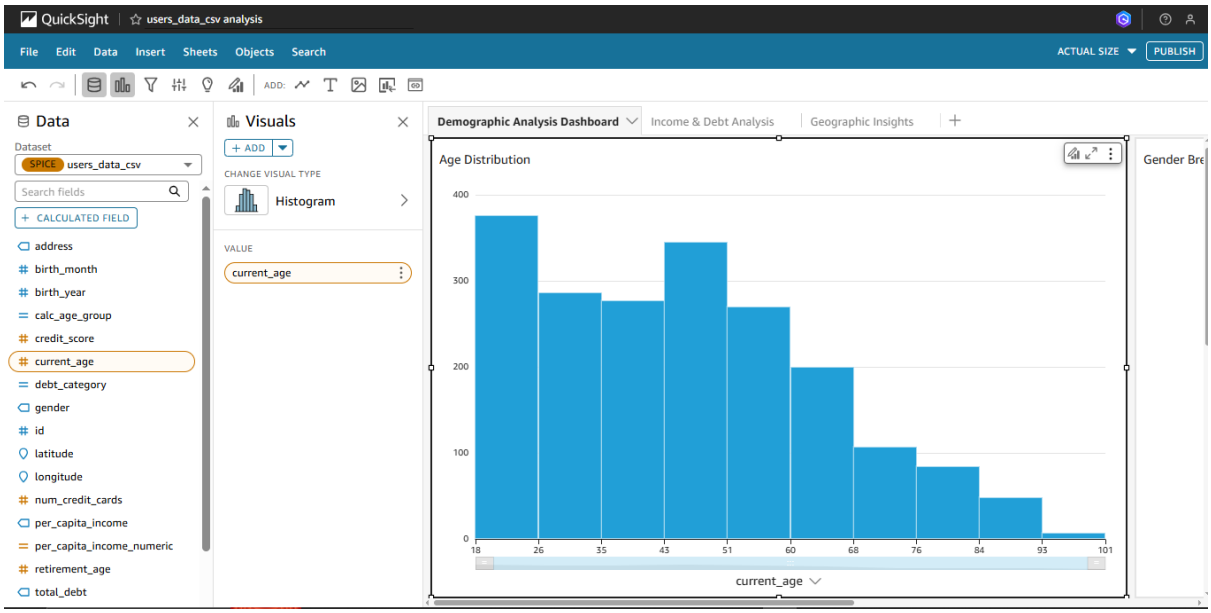
12.Output Screens

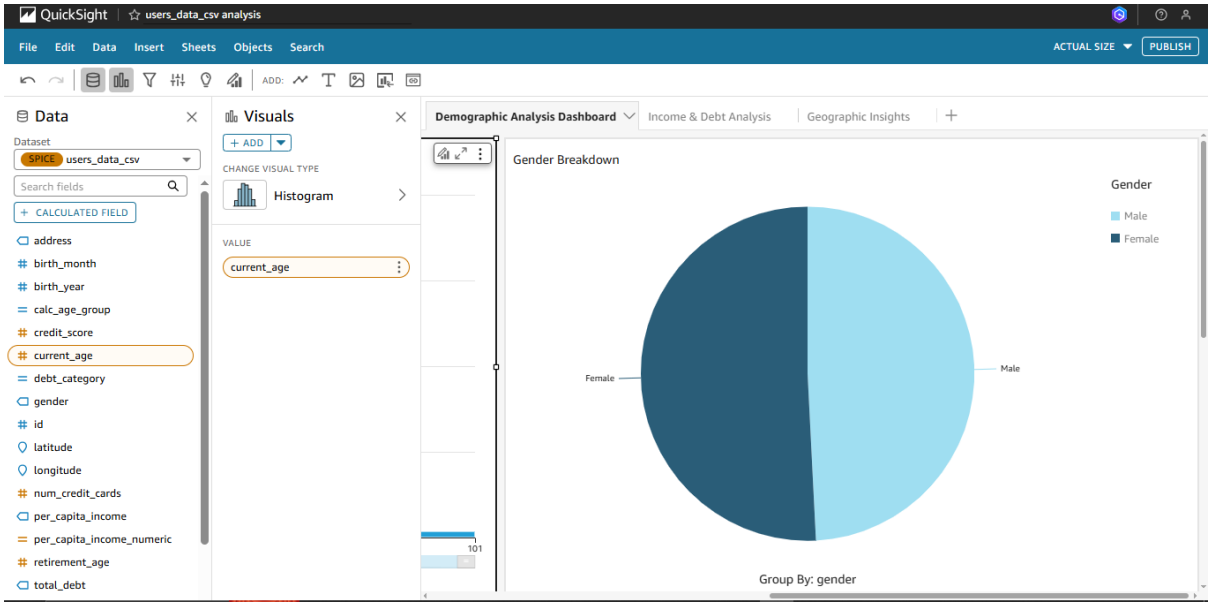
.

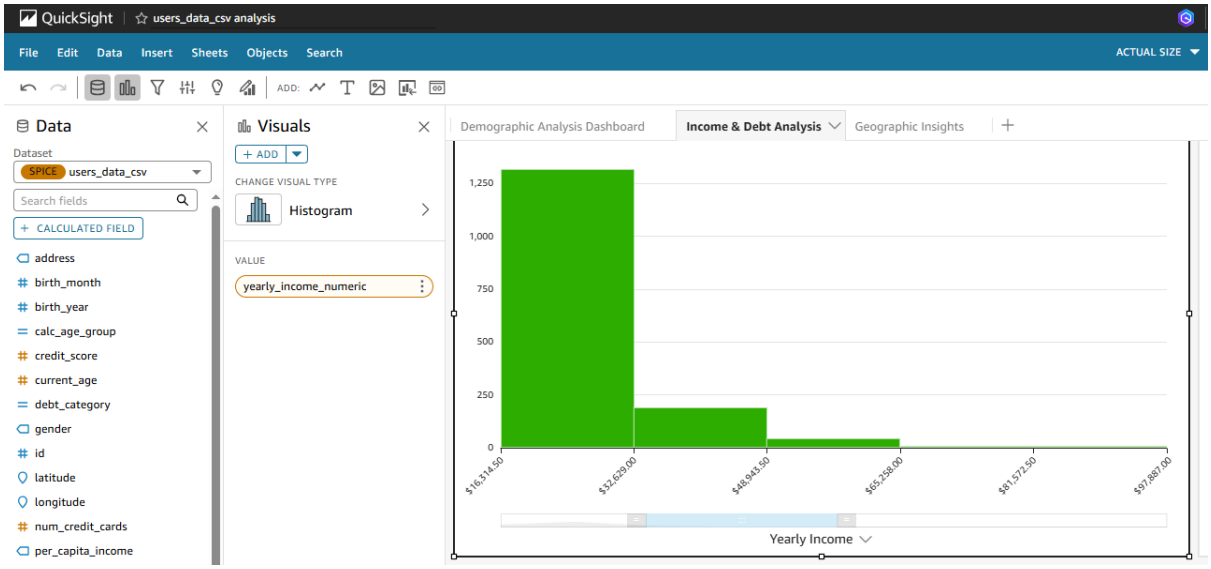


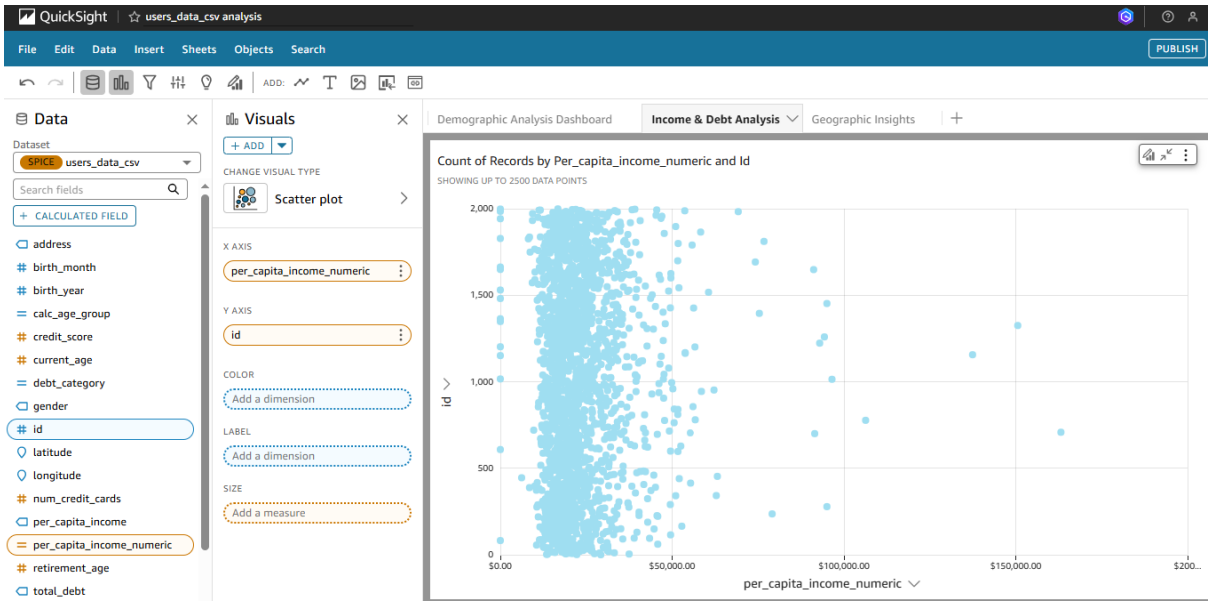


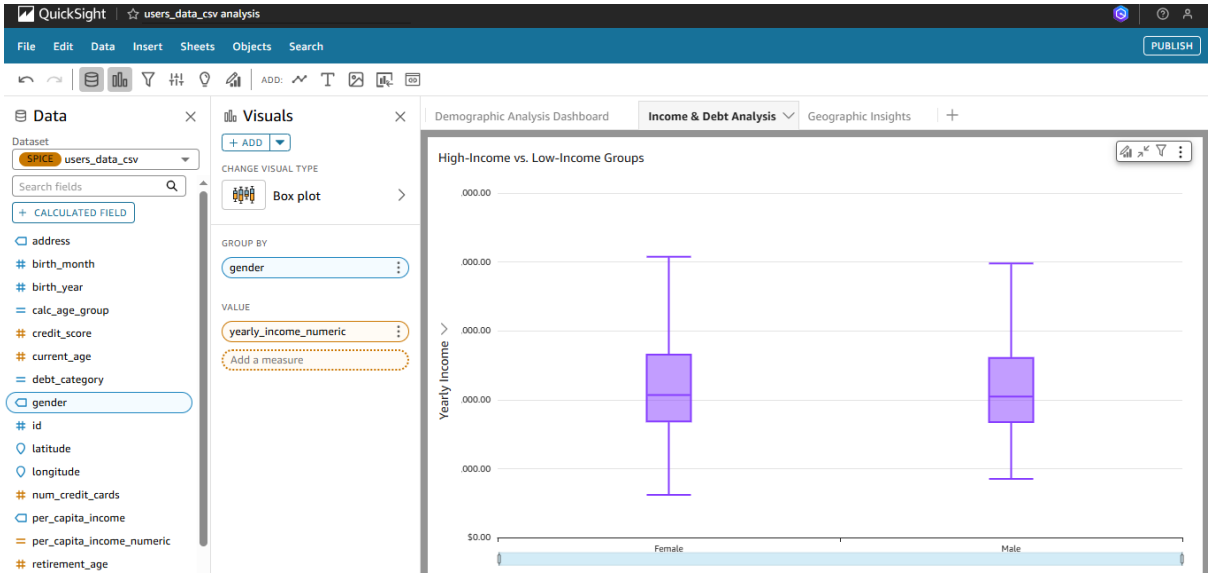

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
ACTUAL SIZE
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
Visuals
ADD
CHANGE VISUAL TYPE
Histogram
VALUE
current_age
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Distribution of People Across Different Retirement Ages
350
300
250
200
150
100
50
0
79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 50

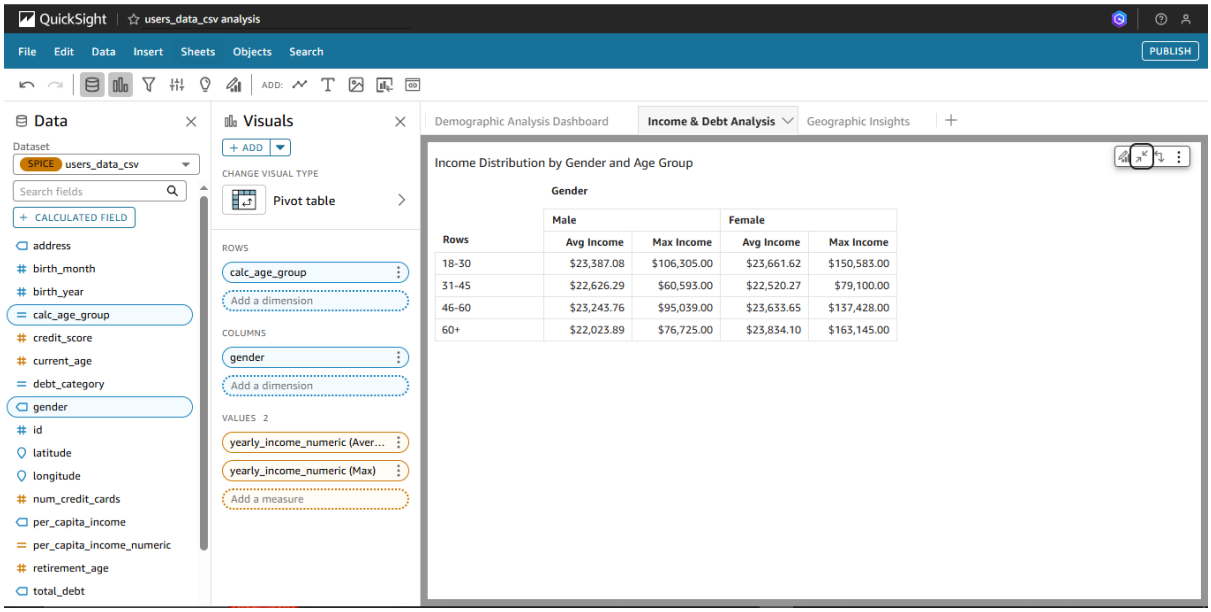

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
ACTUAL SIZE
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
total_debt
Visuals
ADD
CHANGE VISUAL TYPE
Histogram
VALUE
current_age
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Age Distribution
Gender Bre
400
300
200
100
0
18 26 35 43 51 60 68 76 84 93 101
current_age

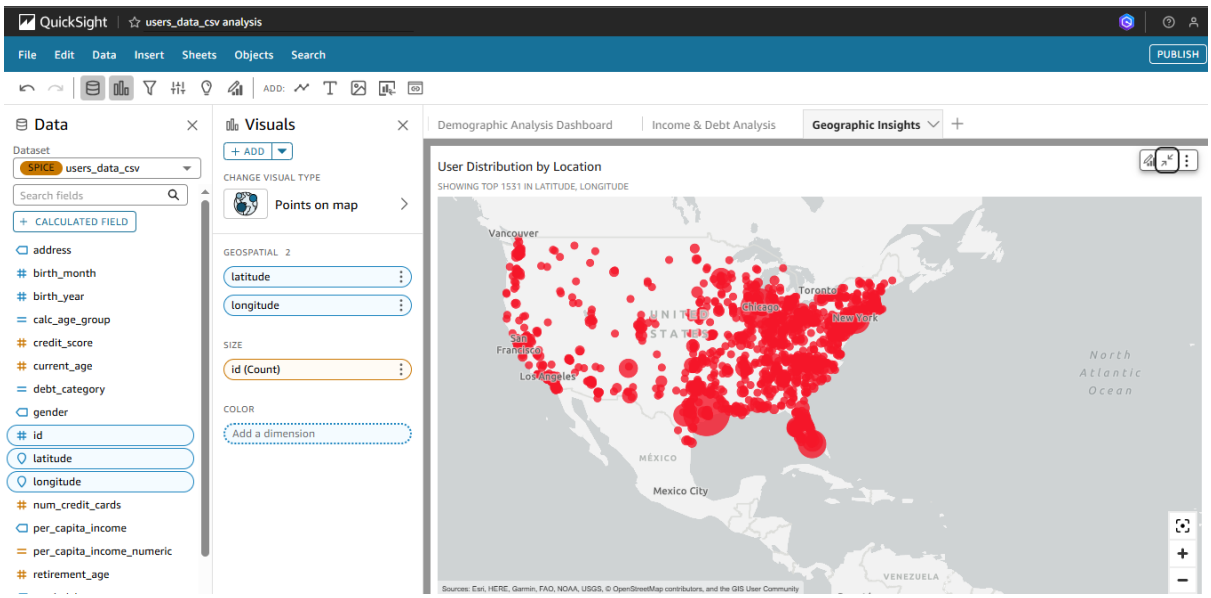

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
ACTUAL SIZE
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
total_debt
Visuals
ADD
CHANGE VISUAL TYPE
Histogram
VALUE
current_age
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Gender Breakdown
Gender
Male
Female
Female
Male
Group By: gender

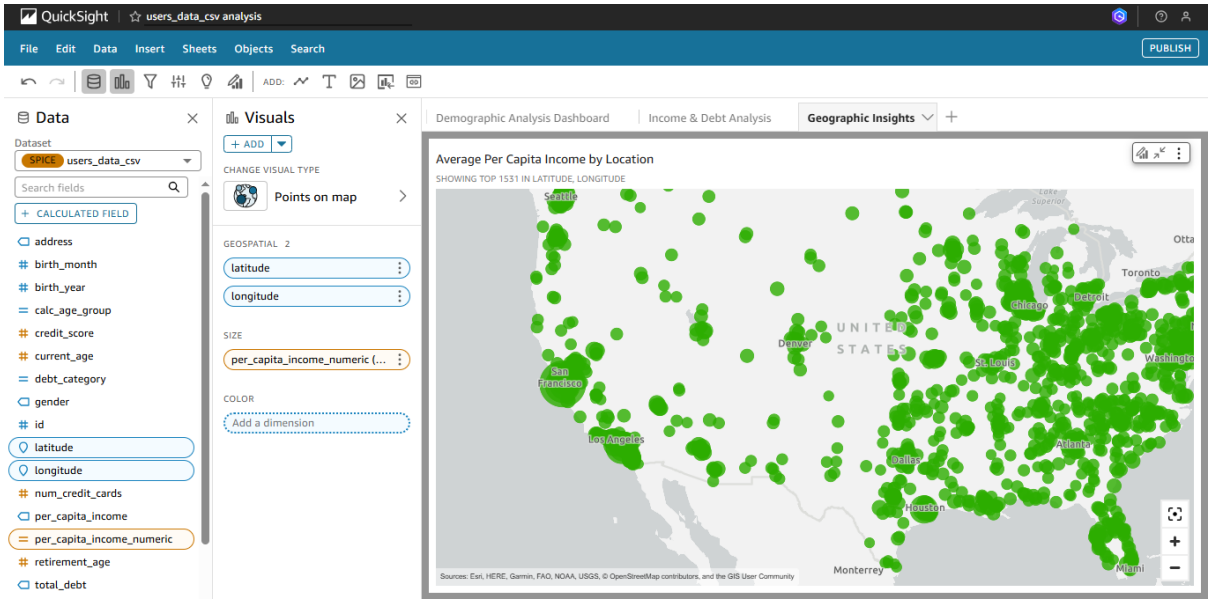

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
ACTUAL SIZE
Data
Dataset
SPICE users_data_csv
Search fields
CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
Visuals
ADD
CHANGE VISUAL TYPE
Histogram
VALUE
yearly_income_numeric
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
1,250
1,000
750
500
250
0
$16,314.50
$32,629.00
$48,943.50
$65,258.00
$81,572.50
$97,887.00
Yearly Income

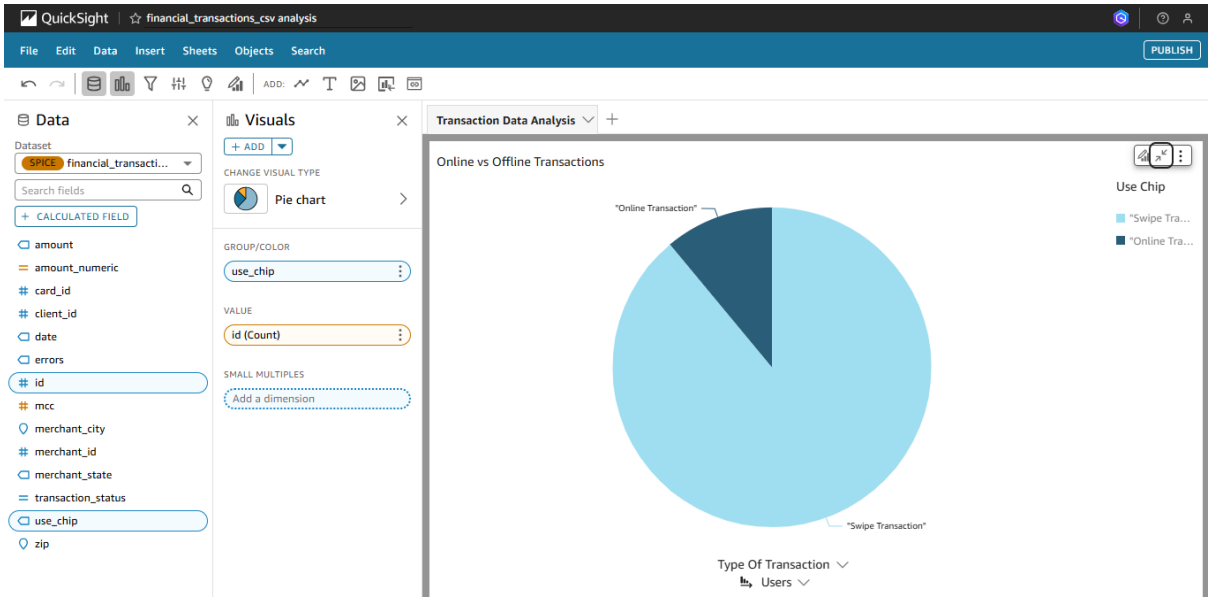

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
total_debt
Visuals
ADD
CHANGE VISUAL TYPE
Scatter plot
X AXIS
per_capita_income_numeric
Y AXIS
id
COLOR
Add a dimension
LABEL
Add a dimension
SIZE
Add a measure
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Count of Records by Per_capita_income_numeric and Id
SHOWING UP TO 2500 DATA POINTS
2,000
1,500
1,000
500
0
Id
$0.00
$50,000.00
$100,000.00
$150,000.00
per_capita_income_numeric

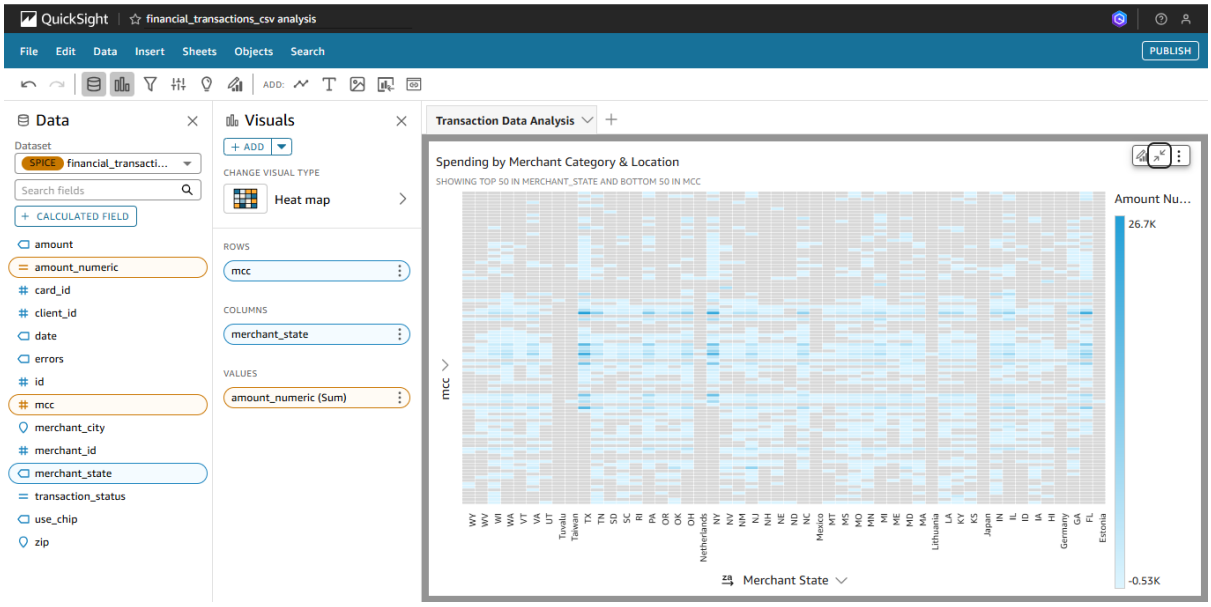

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
+ CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
Visuals
+ ADD
CHANGE VISUAL TYPE
Box plot
GROUP BY
gender
VALUE
yearly_income_numeric
Add a measure
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
High-Income vs. Low-Income Groups
Yearly Income
.000.00
.000.00
.000.00
.000.00
.000.00
$0.00
Female
Male

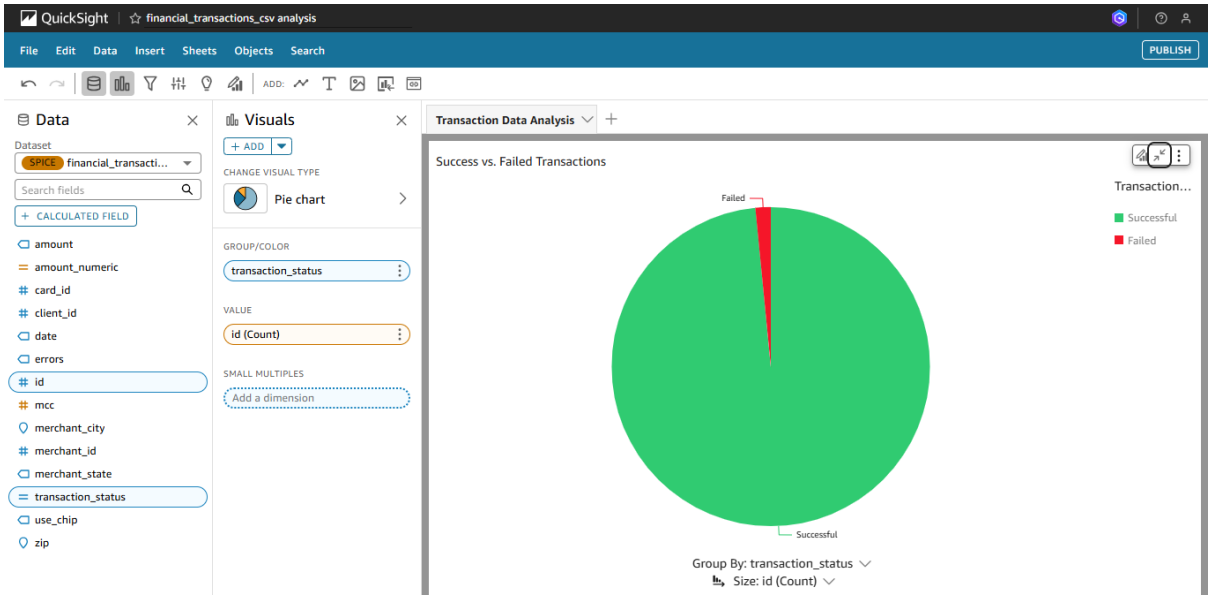

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
+ CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
total_debt
Visuals
+ ADD
CHANGE VISUAL TYPE
Pivot table
ROWS
calc_age_group
Add a dimension
COLUMNS
gender
Add a dimension
VALUES 2
yearly_income_numeric (Aver...
yearly_income_numeric (Max)
Add a measure
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Income Distribution by Gender and Age Group
Gender
Male
Female
Rows
Avg Income
Max Income
Avg Income
Max Income
18-30
$23,387.08
$106,305.00
$23,661.62
$150,583.00
31-45
$22,626.29
$60,593.00
$22,520.27
$79,100.00
46-60
$23,243.76
$95,039.00
$23,633.65
$137,428.00
60+
$22,023.89
$76,725.00
$23,834.10
$163,145.00

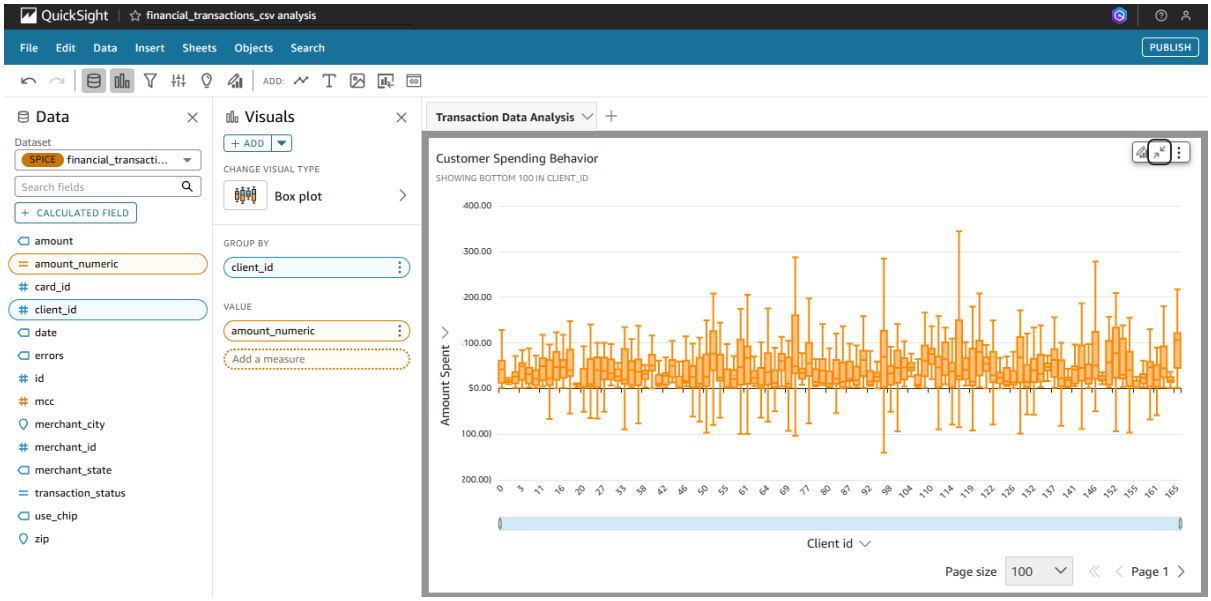

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
+ CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
Visuals
+ ADD
CHANGE VISUAL TYPE
Points on map
GEOSPATIAL 2
latitude
longitude
SIZE
id (Count)
COLOR
Add a dimension
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
User Distribution by Location
SHOWING TOP 1531 IN LATITUDE, LONGITUDE
Vancouver
Toronto
Chicago
New York
San Francisco
Los Angeles
UNITED STATES
MÉXICO
Mexico City
North Atlantic Ocean
VENEZUELA

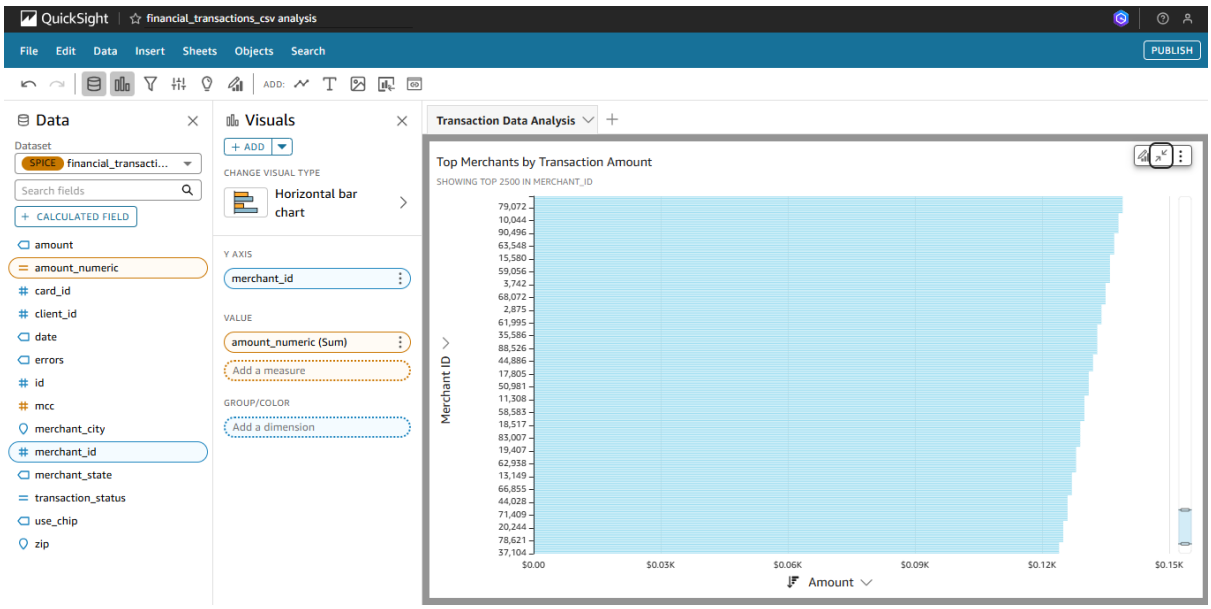

QuickSight
users_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE users_data_csv
Search fields
+ CALCULATED FIELD
address
birth_month
birth_year
calc_age_group
credit_score
current_age
debt_category
gender
id
latitude
longitude
num_credit_cards
per_capita_income
per_capita_income_numeric
retirement_age
total_debt
Visuals
+ ADD
CHANGE VISUAL TYPE
Points on map
GEOSPATIAL 2
latitude
longitude
SIZE
per_capita_income_numeric (...
COLOR
Add a dimension
Demographic Analysis Dashboard
Income & Debt Analysis
Geographic Insights
Average Per Capita Income by Location
SHOWING TOP 1531 IN LATITUDE, LONGITUDE
Seattle
Toronto
Chicago
Detroit
Denver
UNITED STATES
San Francisco
St. Louis
Los Angeles
Dallas
Atlanta
Houston
Monterrey
Miami
Sources: Esri, HERE, Garmin, FAO, NOAA, USGS, © OpenStreetMap contributors, and the GIS User Community

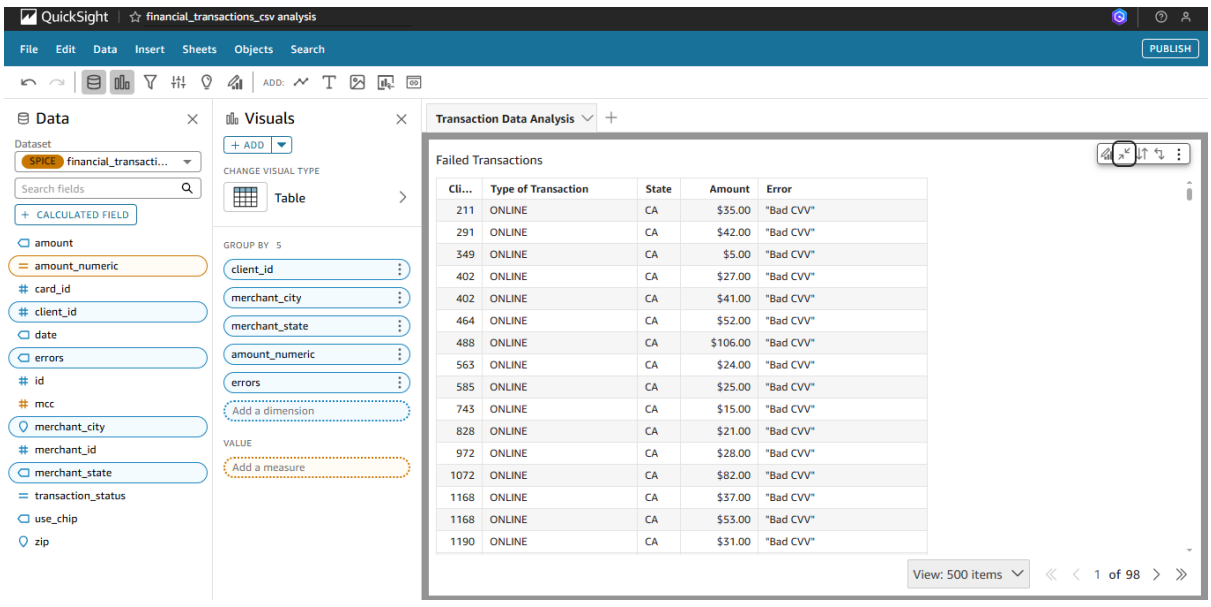

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
+ CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
+ ADD
CHANGE VISUAL TYPE
Pie chart
GROUP/COLOR
use_chip
VALUE
id (Count)
SMALL MULTIPLES
Add a dimension
Transaction Data Analysis
Online vs Offline Transactions
Use Chip
"Swipe Tra...
"Online Tra...
"Online Transaction"
"Swipe Transaction"
Type Of Transaction
Users

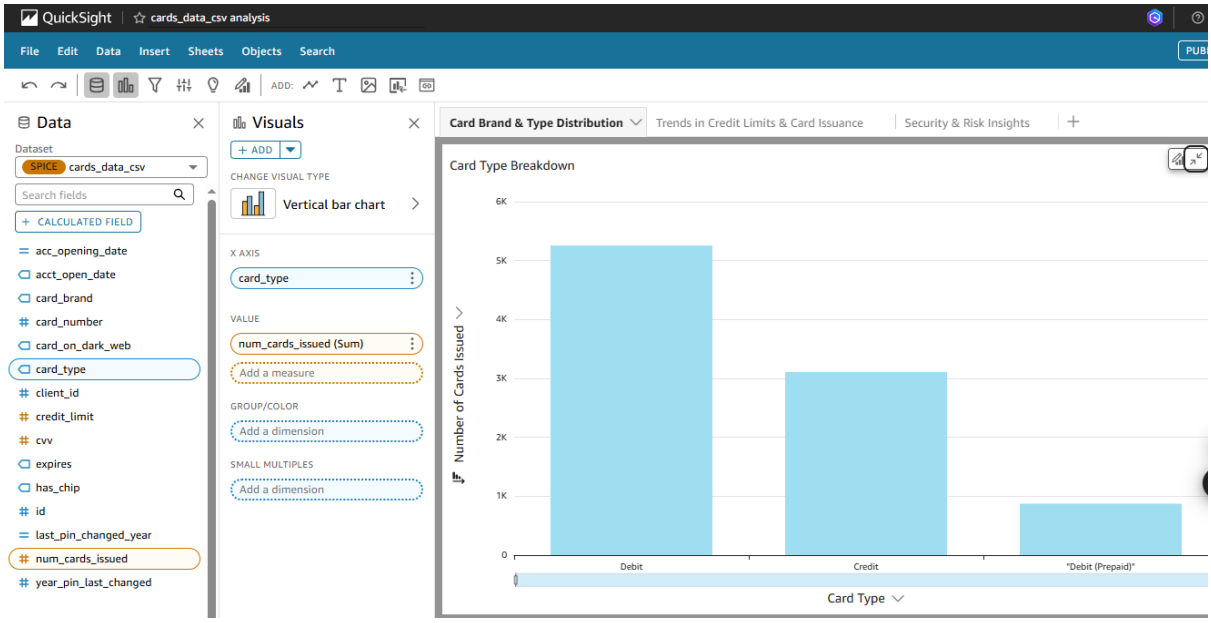

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
ADD
CHANGE VISUAL TYPE
Heat map
ROWS
mcc
COLUMNS
merchant_state
VALUES
amount_numeric (Sum)
Transaction Data Analysis
Spending by Merchant Category & Location
SHOWING TOP 50 IN MERCHANT_STATE AND BOTTOM 50 IN MCC
Amount Nu...
26.7K
-0.53K
mcc
Merchant State

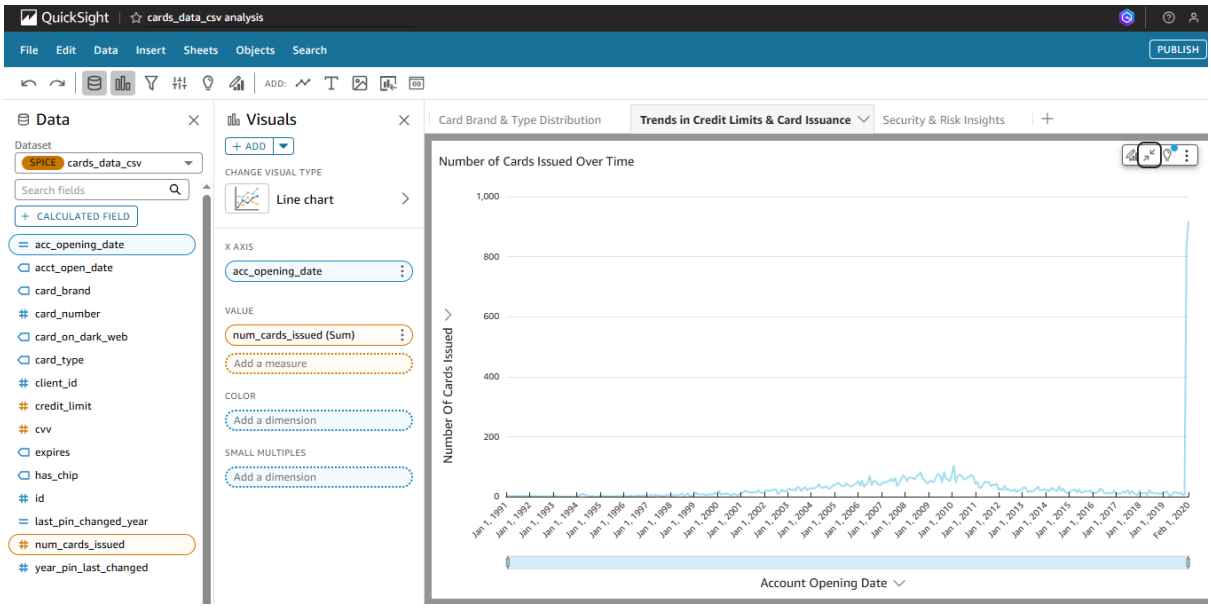

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
ADD
CHANGE VISUAL TYPE
Pie chart
GROUP/COLOR
transaction_status
VALUE
id (Count)
SMALL MULTIPLES
Add a dimension
Transaction Data Analysis
Success vs. Failed Transactions
Transaction...
Successful
Failed
Failed
Successful
Group By: transaction_status
Size: id (Count)

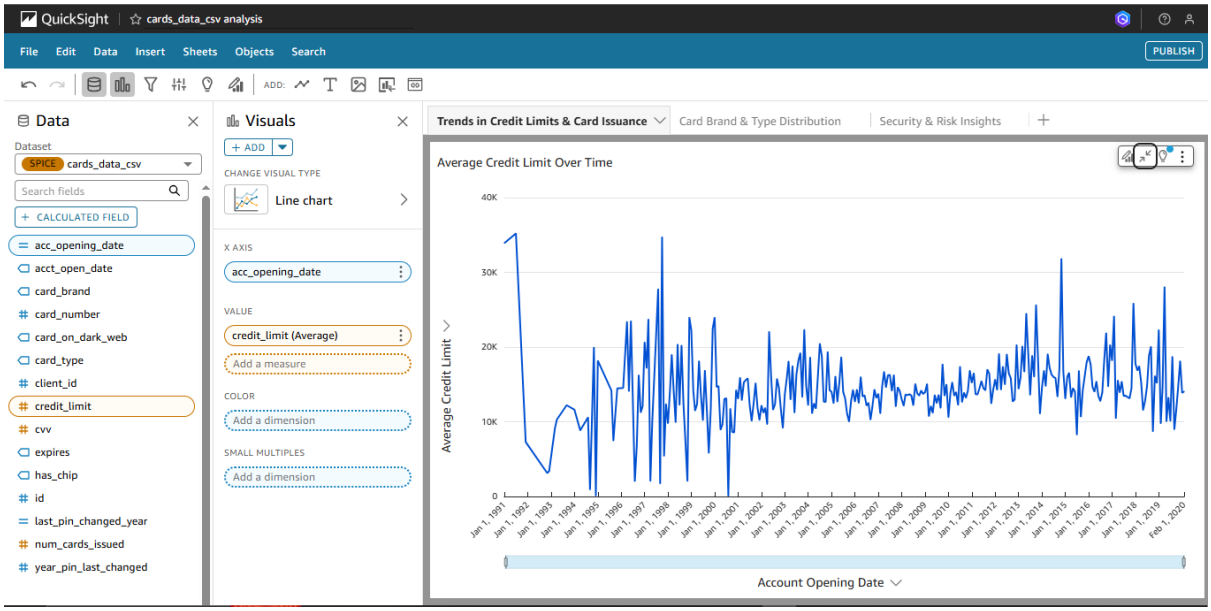

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
ADD
CHANGE VISUAL TYPE
Box plot
GROUP BY
client_id
VALUE
amount_numeric
Add a measure
Transaction Data Analysis
Customer Spending Behavior
SHOWING BOTTOM 100 IN CLIENT_ID
Amount Spent
Client id
Page size 100
Page 1

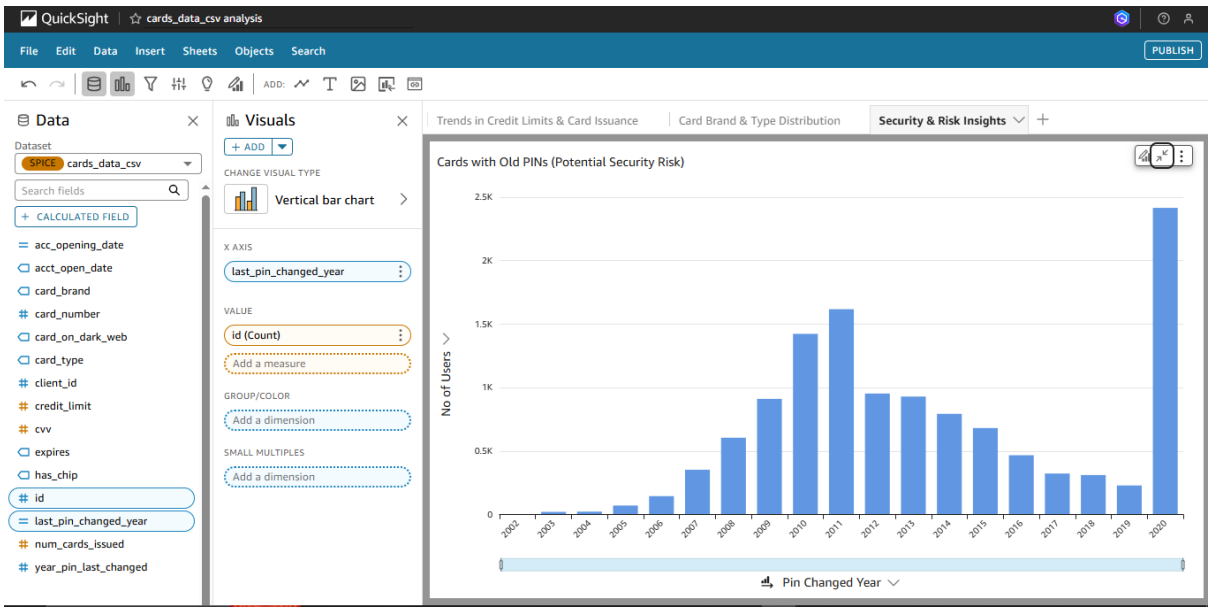

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
ADD
CHANGE VISUAL TYPE
Horizontal bar chart
Y AXIS
merchant_id
VALUE
amount_numeric (Sum)
Add a measure
GROUP/COLOR
Add a dimension
Transaction Data Analysis
Top Merchants by Transaction Amount
SHOWING TOP 2500 IN MERCHANT_ID
Merchant ID
Amount

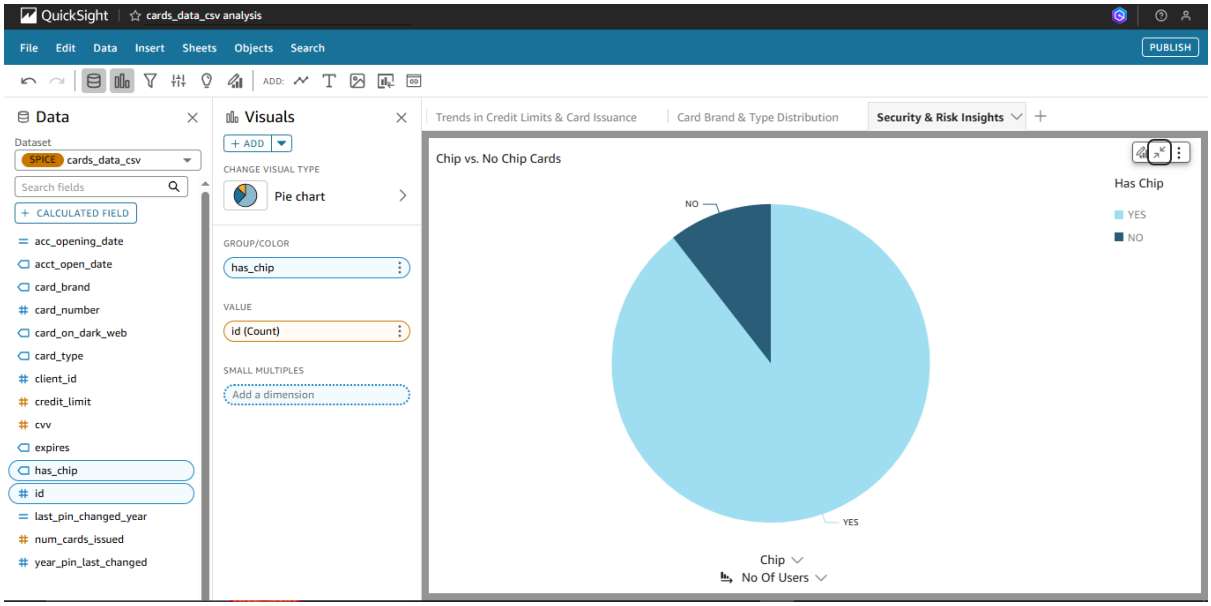

QuickSight
financial_transactions_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE financial_transacti...
Search fields
CALCULATED FIELD
amount
amount_numeric
card_id
client_id
date
errors
id
mcc
merchant_city
merchant_id
merchant_state
transaction_status
use_chip
zip
Visuals
ADD
CHANGE VISUAL TYPE
Table
GROUP BY 5
client_id
merchant_city
merchant_state
amount_numeric
errors
Add a dimension
VALUE
Add a measure
Transaction Data Analysis
Failed Transactions
Cli... Type of Transaction State Amount Error
211 ONLINE CA $35.00 "Bad CVV"
291 ONLINE CA $42.00 "Bad CVV"
349 ONLINE CA $5.00 "Bad CVV"
402 ONLINE CA $27.00 "Bad CVV"
402 ONLINE CA $41.00 "Bad CVV"
464 ONLINE CA $52.00 "Bad CVV"
488 ONLINE CA $106.00 "Bad CVV"
563 ONLINE CA $24.00 "Bad CVV"
585 ONLINE CA $25.00 "Bad CVV"
743 ONLINE CA $15.00 "Bad CVV"
828 ONLINE CA $21.00 "Bad CVV"
972 ONLINE CA $28.00 "Bad CVV"
1072 ONLINE CA $82.00 "Bad CVV"
1168 ONLINE CA $37.00 "Bad CVV"
1168 ONLINE CA $53.00 "Bad CVV"
1190 ONLINE CA $31.00 "Bad CVV"
View: 500 items
1 of 98


QuickSight
cards_data_csv analysis
File Edit Data Insert Sheets Objects Search
Data
Dataset
SPICE cards_data_csv
Search fields
CALCULATED FIELD
acc_opening_date
acct_open_date
card_brand
card_number
card_on_dark_web
card_type
client_id
credit_limit
cvv
expires
has_chip
id
last_pin_changed_year
num_cards_issued
year_pin_last_changed
Visuals
ADD
CHANGE VISUAL TYPE
Vertical bar chart
X AXIS
card_type
VALUE
num_cards_issued (Sum)
Add a measure
GROUP/COLOR
Add a dimension
SMALL MULTIPLES
Add a dimension
Card Brand & Type Distribution
Trends in Credit Limits & Card Issuance
Security & Risk Insights
Card Type Breakdown
Number of Cards Issued
6K
5K
4K
3K
2K
1K
0
Debit
Credit
"Debit (Prepaid)"
Card Type


QuickSight
cards_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE cards_data_csv
Search fields
CALCULATED FIELD
acc_opening_date
acct_open_date
card_brand
card_number
card_on_dark_web
card_type
client_id
credit_limit
cvv
expires
has_chip
id
last_pin_changed_year
num_cards_issued
year_pin_last_changed
Visuals
ADD
CHANGE VISUAL TYPE
Line chart
X AXIS
acc_opening_date
VALUE
num_cards_issued (Sum)
Add a measure
COLOR
Add a dimension
SMALL MULTIPLES
Add a dimension
Card Brand & Type Distribution
Trends in Credit Limits & Card Issuance
Security & Risk Insights
Number of Cards Issued Over Time
Number Of Cards Issued
1,000
800
600
400
200
0
Account Opening Date


QuickSight
cards_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE cards_data_csv
Search fields
CALCULATED FIELD
acc_opening_date
acct_open_date
card_brand
card_number
card_on_dark_web
card_type
client_id
credit_limit
cvv
expires
has_chip
id
last_pin_changed_year
num_cards_issued
year_pin_last_changed
Visuals
ADD
CHANGE VISUAL TYPE
Line chart
X AXIS
acc_opening_date
VALUE
credit_limit (Average)
Add a measure
COLOR
Add a dimension
SMALL MULTIPLES
Add a dimension
Trends in Credit Limits & Card Issuance
Card Brand & Type Distribution
Security & Risk Insights
Average Credit Limit Over Time
Average Credit Limit
Account Opening Date


QuickSight
cards_data_csv analysis
File Edit Data Insert Sheets Objects Search
PUBLISH
Data
Dataset
SPICE cards_data_csv
Search fields
CALCULATED FIELD
acc_opening_date
acct_open_date
card_brand
card_number
card_on_dark_web
card_type
client_id
credit_limit
cvv
expires
has_chip
id
last_pin_changed_year
num_cards_issued
year_pin_last_changed
Visuals
ADD
CHANGE VISUAL TYPE
Vertical bar chart
X AXIS
last_pin_changed_year
VALUE
id (Count)
Add a measure
GROUP/COLOR
Add a dimension
SMALL MULTIPLES
Add a dimension
Trends in Credit Limits & Card Issuance
Card Brand & Type Distribution
Security & Risk Insights
Cards with Old PINs (Potential Security Risk)
No of Users
Pin Changed Year


QuickSight
cards_data_csv analysis
File
Edit
Data
Insert
Sheets
Objects
Search
PUBLISH
Data
Dataset
SPICE cards_data_csv
Search fields
+ CALCULATED FIELD
acc_opening_date
acct_open_date
card_brand
card_number
card_on_dark_web
card_type
client_id
credit_limit
cvv
expires
has_chip
id
last_pin_changed_year
num_cards_issued
year_pin_last_changed
Visuals
+ ADD
CHANGE VISUAL TYPE
Pie chart
GROUP/COLOR
has_chip
VALUE
id (Count)
SMALL MULTIPLES
Add a dimension
Trends in Credit Limits & Card Issuance
Card Brand & Type Distribution
Security & Risk Insights
Chip vs. No Chip Cards
Has Chip
YES
NO
NO
YES
Chip
No Of Users