ETL Data Pipeline for Banking Loan Data Analysis

A Novel Data Pipeline for Efficient Cleaning and Transformation in a Streamlined Banking Loan Analysis

Group Members:

1. Name: Ishan Sinnarkar
   Student ID: 10453
   Email: ishanmahesh.sinnarkar2@mail.dcu.ie

2. Name: Atharva Patil
   Student ID: 11866
   Email: atharvarajkumar.patil2@mail.dcu.ie

3. Name: Varad Vaidya
   Student ID: 10545
   Email: varadanil.vaidya2@mail.dcu.ie

Source code link - CC_assignment (open with collab so that code along with output is visible)

Dataset Link: LoanStats_2018Q4.csv

Video Link: Cloud Technology Assignment - Video.mp4

INTRODUCTION:

In the modern, data-driven world, banking and financial sectors generate a huge amount of data day in and day out. Of those, loan data bears critical insights into decision-making, assessment of risk, and ways to improve customer experience. A well-designed ETL-Data Pipeline is required to understand this data. ELT-approach means the contemporary way of processing data when the raw data gets extracted from sources, loaded into some sort of centralized data storage system (like a cloud-based data warehouse), and then transformed if needed for analysis.

In this project, the ETL data pipeline is designed to facilitate the processing and analysis of bank loan data. The pipeline, taking advantage of cloud computing technologies-MS Azure-simplifies raw data extraction: a semi-structured CSV file. The data is loaded into a scalable and secure cloud-based storage solution, namely Azure Data Lake storage Gen2, and transformed with the help of Distributed data processing tool, namely Databricks, to prepare the data for analysis. In this way, stakeholders can derive actionable insights efficiently.

This project demonstrates how to construct an efficient and scalable data pipeline by combining Spark's distributed data processing with a practical dataset. It shows that cleaning, schema validation, and transformation of data are important steps in preparing the data for meaningful insights, whether the purpose is machine learning, reporting, or operational.

MOTIVATION:

In the big data era, an organization has to increasingly rely on efficient engineering of data and analysis for meaningful insights to drive data-driven decisions. This project shows our commitment to leveraging modern cloud-based technologies and distributed computing platforms to solve real-world data challenges. Among those, Apache Spark is a powerful tool for data processing and analysis, and we have utilized it on the Databricks platform to show its transformative potential.

By executing this project, we are targeting an understanding and putting into practice all basic principles of cloud-based technologies and, especially, the Microsoft Azure ecosystem for designing and constructing scalable, efficient, practical data solutions. The ETL pipeline we have developed shows how raw, semi-structured data can be extracted, cleaned, and checked and then prepared for actionable insights.

It indeed acts as a bed that showcases our learning abilities to use what is learned throughout this course related to cloud technologies. Practical vision showed clearly how different tools or technology, like Microsoft Azure with Databricks, has targeted specific data challenges, very vital in allowing us to further compare and evaluate those matched for particular tasks while at the same time getting enriched understanding of the whole Cloud ecosystem.

This is ultimately a project demonstrating the values of learning by doing. It reflects our passion for exploring the potential of cloud-based technologies and our commitment to excellence in data engineering.

In this work, we aim to develop this book so that people may better appreciate both the tools and the practice of data analysis and engineering that makes modern data analysis work and allow innovative solutions into the future.

RELATED WORK:

1. Get Data into Databricks - Simple ETL Pipeline:

The following work-related work entails the automation of the ETL process within Databricks. In detail, the workflow involves a Databricks notebook reading in some CSV files into a DataFrame and writing that DataFrame into a Delta Table, which ensures consistency and streamlined data processing. This approach, by the power of Spark, automated some of the most tedious steps-structured data format generation from the unstructured CSVs-so the data would be that much more prepared for loading, cleaning, and preparation to undergo analysis or machine learning tasks. It also provides the view of how Databricks integrates various data sources; thus, enabling effective processing and transformation of data automatically.

2. How to Create a SQL Table in Databricks from a CSV File & Loading CSV Files in Databricks:

The current example will show how to successfully load a CSV into Databricks and create, in SQL, tables from there for analysis. It would involve reading the CSV file using Spark's DataFrame API, performing any transformation or cleaning steps necessary, and then converting the DataFrame into a SQL table inside Databricks. This would now enable, in a really cumbersome manner, SQL-based analytics directly in the Databricks environment for large datasets, especially for complicated queries or aggregations. This approach underlines how Databricks has managed to integrate Spark with SQL in one go-massive transition from raw data to actionable insight in one place.

3. World Development Indicators Analytics Project in Apache Spark for Beginners Using Databricks:

This related work is among the use cases of Databricks for big data analysis in developmental analysis. The project describes how Apache Spark is used within Databricks to explore WDIs data, which is made up of a wide set of global economic, social, and environmental indicators. In this project, it will be shown how data analysis and visualization are done on large datasets by leveraging the distributed processing ability of Spark.

HOW DOES OUR WORK DIFFER?

Our project extends basic ETL processes to address unique challenges in semi-structured banking loan data, including cleaning fields with text descriptors-such as "36 months"-and numeric fields containing special characters, like percentages. We added schema validation at an advanced level that guarantees the integrity and scalability of the data, preparing the pipeline for downstream machine learning and advanced analytics. Whereas other projects on analytics were generalized, ours is an industry-specific project that will involve the transformation and analysis of financial data, such as complex metrics including debt-to-income ratios, loan grades, and current statuses of loans. We've designed the pipeline to provide actionable insights from the given data while mitigating the appearance of critical issues such as outliers. Our integration with Azure SQL Database went beyond SQL analytics to provide persistent storage, relational querying, and schema mapping for compatibility with Databricks. We used PySpark to process the data by applying a variety of techniques, including regular expressions, data type conversions, and caching. Working in this distributed environment on Databricks, cluster configuration and performance optimization due to Spark's lazy evaluation were also imperative. Such an approach allowed us to create a strong, scalable, and efficient pipeline that solves real-world financial data challenges while exploiting all the potential of cloud-based technologies

CHOICE OF TECHNOLOGIES:

For this project, we have utilized the following technologies to streamline the data pipeline:

1. Cloud Based Technology – Microsoft Azure:

Azure is Microsoft's multi-service cloud-based computing platform for computation, analytics, storage, and databases. It will eventually provide the enterprises with the much-needed flexibility, scalability, and security for which it is in such high demand today. Other important factors for which we have chosen Azure are cost and because various integrations like Databricks and SQL Server amicably in it support efficiently our ELT pipeline that processes and analyses big data.

2. Databricks:

Databricks is a unified analytics platform on Apache Spark, making big data processing and machine learning simpler. Since it is integrated with Azure, it offers scalable resources to perform distributed data processing and analysis. It was advanced data transformation, cleaning, and validation using Databricks to its full capacity, taking advantage of the collaborative environment in handling large datasets with much ease.

3. Structured Query Language (SQL):

SQL can be used to query, insert, update, and delete data in a relational database management system. The reason SQL was used in the project is because SQL becomes necessary when interfacing with the data has to be done; for example, doing some schema validations, doing transformations as necessary to get it into the right formats for eventual analysis.

4. Python:

Python is a general-purpose language which applied in data processing, data analysis, and machine learning. Featured with simplicity and readability, hence, Python can be used for scripting in the automation of tasks along the data pipeline. Python was combined with PySpark for data transformation in handling big data on Databricks.

5. ADLS:

ADLS provides a highly scalable, secure data lake in Azure to store petabytes of data in unstructured formats supported by them. Seamless integration with Azure analytics tools: We used ADLS for storing raw data in order to provide the ETL pipeline with a unified and secured place for storing raw data in a scalable manner.

6. Azure SQL Database:

Azure SQL Database, the managed relational database service that enables users to build their applications more quickly, more independently, with high availability, scalability, and security. It allows the execution or management of structured data using Structured Query Language queries. Besides, Azure has the virtue of being very harmonious with other services provided thereby. Data transformed and processed are stored in Azure SQL Database, which allows the effective querying and reporting of these latter.

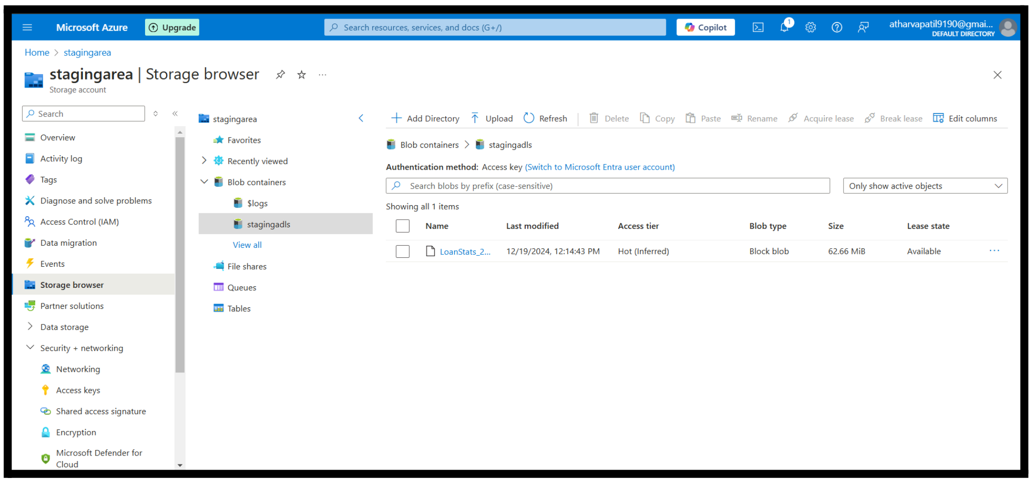
DATASET DESCRIPTION:

a) Source of the data:

Lending Club is a peer-to-peer lending firm that facilitates personal, small business, and auto refinancing loans. It has a dataset containing information regarding loan data for a single quarter, consisting of 128,000 rows and 145 columns, where each row describes a loan. In this dataset, the detail provided about the borrower profiles and loan details, coupled with the payment status of every single loan, is quite exhaustive.
Some of the key features in this dataset are loan amount, funded amount, term of the loan (36 or 60 months), interest rate, grade assigned by Lending Club, employment length, home ownership, loan purpose, and debt-to-income ratio. The dataset also shows the status of loans, whether current, delayed, or charged off-a non-recoverable loan.

b) Process of extraction/collection:

The CSV file of the dataset was downloaded and then we uploaded the dataset to the cloud storage i.e ADLS.



Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive) | Only show active objects

Showing all 1 items

Name	Last modified	Access tier	Blob type	Size	Lease state
LoanStats_2...	12/19/2024, 12:14:43 PM	Hot (Inferred)	Block blob	62.66 MiB	Available

We have stored the data in a hot tier so that it is accessible quickly and we have made the data geo redundant for better fault tolerance.

c) Data pre-processing – preparation and cleaning:

We have chosen Azure Databricks since it has a very seamless integration with Azure Data Lake Storage through the mounting of the storage directly onto the Databricks platform. The setup ensures a seamless flow of data, which enables the efficient access and processing of data. Azure Databricks is a combination of collaboration on Databricks with scalability and flexibility on Azure, hence suitable for big data workflows.

At the core of the platform is Apache Spark, the leading distributed big data processing framework. Apache Spark provides robust capabilities for large-scale data transformations, analytics, and machine learning in a distributed computing environment. Through Databricks' intuitive user interface and the robustness of Spark, we are able to achieve simplified data pipelines, increased efficiency, and productivity.

Lending Club is a peer-to-peer lending company that offers personal loans, small business loans, and auto refinancing. The platform connects borrowers who, in most instances, have poor credit histories and want quick access to funds with investors who can fund their loans. Borrowers request loans, such as a $20,000 loan, and their profiles are made available to potential investors. Investors can then review these profiles and decide if the risk is acceptable to them. Investors choose to invest; once a loan is funded, it is repaid through regular monthly installments in addition to interest charges. The interest rates on those loans are usually higher compared to the ones provided by banks due to the investors taking more risks.

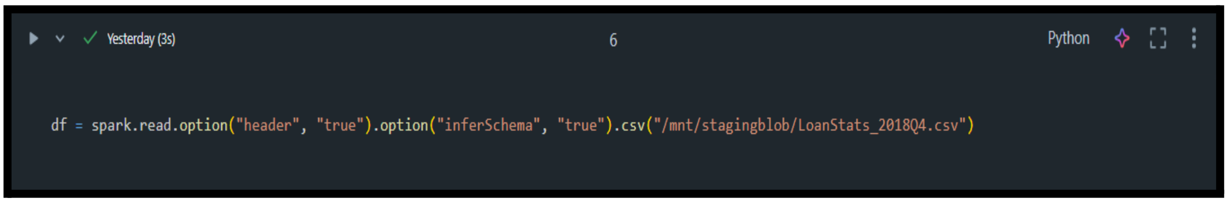
Many fields require cleaning and transformation for meaningful analysis. For example, fields like loan terms have text in them such as "months," which need to be stripped to make them numeric. Similarly, percentages in the interest rate and revolving utilization columns are stored as strings, requiring conversion to numeric formats. Employment length, which includes symbols such as "<" and "years," also needs to be cleaned.

DESCRIPTION OF DATA PROCESSING:

- Data Loading

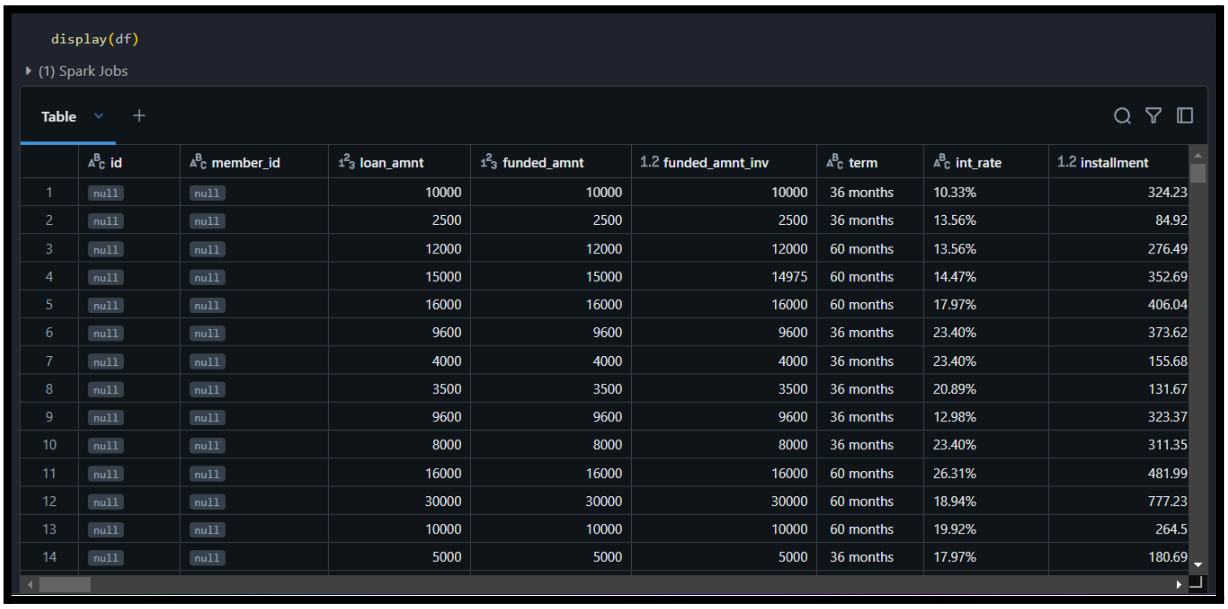
The dataset was loaded into a Spark DataFrame using:

df = spark.read.format("csv").option("inferSchema", "true").option("header", "true").load("/path/to/lending_club.csv")



```
df = spark.read.option("header", "true").option("inferSchema", "true").csv("/mnt/stagingblob/LoanStats_2018Q4.csv")
```

This ensured automatic detection of column types and correct interpretation of headers. Loading the dataset as a Spark DataFrame provided scalability for handling large volumes of data and flexibility for transformations and analysis.



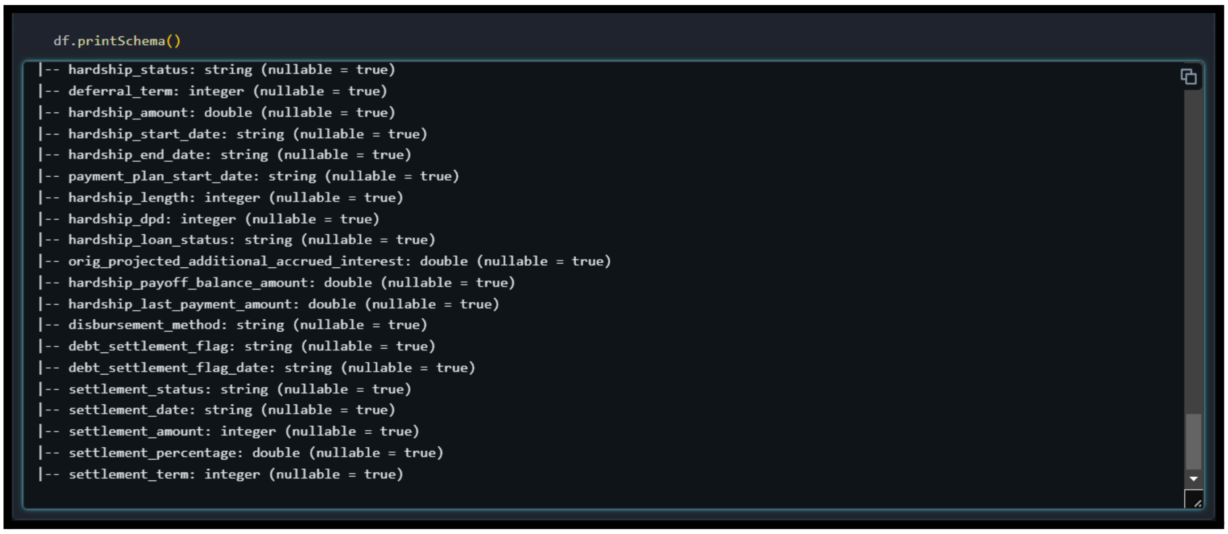
```
display(df)
```

(1) Spark Jobs

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment
1	null	null	10000	10000	10000	36 months	10.33%	324.23
2	null	null	2500	2500	2500	36 months	13.56%	84.92
3	null	null	12000	12000	12000	60 months	13.56%	276.49
4	null	null	15000	15000	14975	60 months	14.47%	352.69
5	null	null	16000	16000	16000	60 months	17.97%	406.04
6	null	null	9600	9600	9600	36 months	23.40%	373.62
7	null	null	4000	4000	4000	36 months	23.40%	155.68
8	null	null	3500	3500	3500	36 months	20.89%	131.67
9	null	null	9600	9600	9600	36 months	12.98%	323.37
10	null	null	8000	8000	8000	36 months	23.40%	311.35
11	null	null	16000	16000	16000	60 months	26.31%	481.99
12	null	null	30000	30000	30000	60 months	18.94%	777.23
13	null	null	10000	10000	10000	60 months	19.92%	264.5
14	null	null	5000	5000	5000	36 months	17.97%	180.69

- Schema Inspection: The schema was inspected using:

df.printSchema()



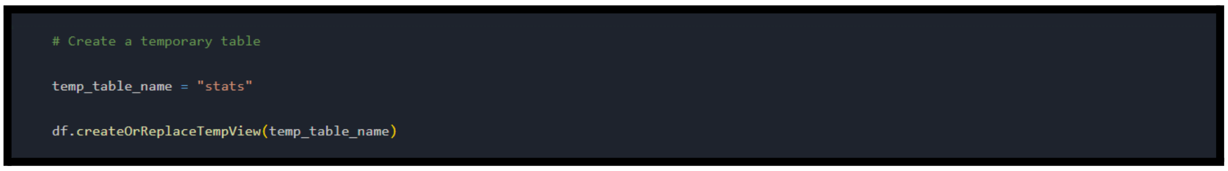
```
df.printSchema()
```

```
 |-- hardship_status: string (nullable = true)
 |-- deferral_term: integer (nullable = true)
 |-- hardship_amount: double (nullable = true)
 |-- hardship_start_date: string (nullable = true)
 |-- hardship_end_date: string (nullable = true)
 |-- payment_plan_start_date: string (nullable = true)
 |-- hardship_length: integer (nullable = true)
 |-- hardship_dpd: integer (nullable = true)
 |-- hardship_loan_status: string (nullable = true)
 |-- orig_projected_additional_accrued_interest: double (nullable = true)
 |-- hardship_payoff_balance_amount: double (nullable = true)
 |-- hardship_last_payment_amount: double (nullable = true)
 |-- disbursement_method: string (nullable = true)
 |-- debt_settlement_flag: string (nullable = true)
 |-- debt_settlement_flag_date: string (nullable = true)
 |-- settlement_status: string (nullable = true)
 |-- settlement_date: string (nullable = true)
 |-- settlement_amount: integer (nullable = true)
 |-- settlement_percentage: double (nullable = true)
 |-- settlement_term: integer (nullable = true)
```

This step verified the structure and data types of the columns, ensuring they matched processing requirements. Proper understanding of the schema was crucial for identifying necessary transformations and detecting potential data type issues early.

- Temporary Table Creation

To facilitate SQL-based analysis, the DataFrame was registered as a temporary table:

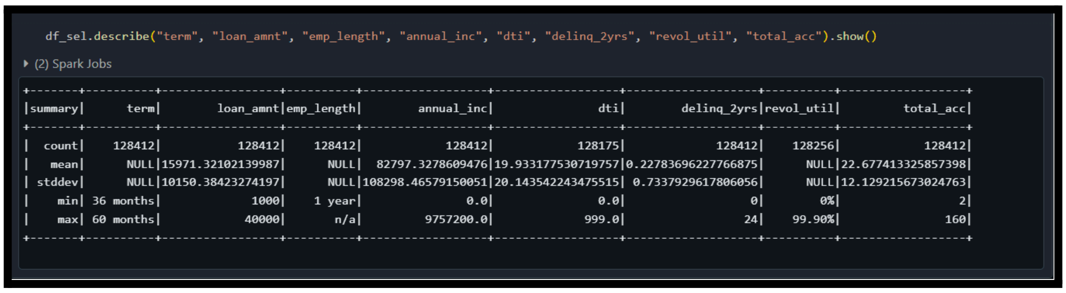


```
# Create a temporary table

temp_table_name = "stats"

df.createOrReplaceTempView(temp_table_name)
```

This allowed analysts to leverage Spark SQL for querying and exploring the data. Temporary tables acted as an interface between raw data and analytical queries, simplifying complex operations.

Descriptive statistics for numerical columns were generated using:

df.describe().show()



```
df_sel.describe("term", "loan_amnt", "emp_length", "annual_inc", "dti", "delinq_2yrs", "revol_util", "total_acc").show()
```

(2) Spark Jobs

```
+-------+---------+-----------------+----------+------------------+------------------+-------------------+----------+------------------+
|summary|     term|        loan_amnt|emp_length|        annual_inc|               dti|        delinq_2yrs|revol_util|         total_acc|
+-------+---------+-----------------+----------+------------------+------------------+-------------------+----------+------------------+
|  count|   128412|           128412|    128412|            128412|            128175|             128412|    128256|            128412|
|   mean|     NULL|15971.32102139987|      NULL|  82797.3278609476|19.933177530719757|0.22783696227766875|      NULL|22.677413325857398|
| stddev|     NULL|10150.38423274197|      NULL|108298.46579150051|20.143542243475515| 0.7337929617806056|      NULL|12.129215673024763|
|    min|36 months|             1000|    1 year|               0.0|               0.0|                  0|        0%|                 2|
|    max|60 months|            40000|       n/a|         9757200.0|             999.0|                 24|    99.90%|               160|
+-------+---------+-----------------+----------+------------------+------------------+-------------------+----------+------------------+
```

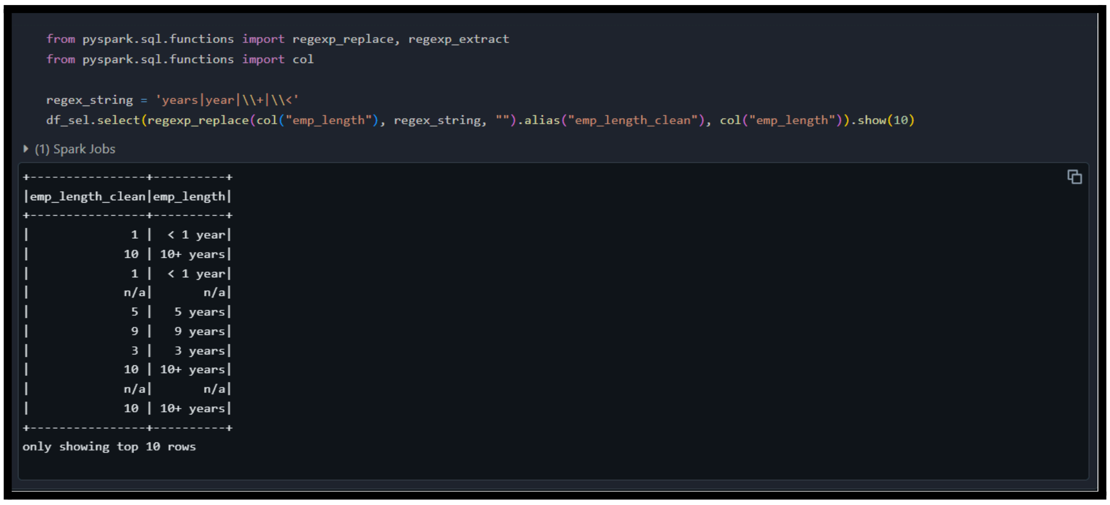
This provided key metrics such as count, mean, standard deviation, and maximum values. These insights highlighted potential outliers, skewed distributions, and missing data trends, forming the basis for further cleaning and analysis.

- Data Cleaning:

Key cleaning activities included:

1. Removing Special Characters: Unwanted characters, such as percentages or text descriptors, were removed to standardize column values:

from pyspark.sql.functions import regexp_replace

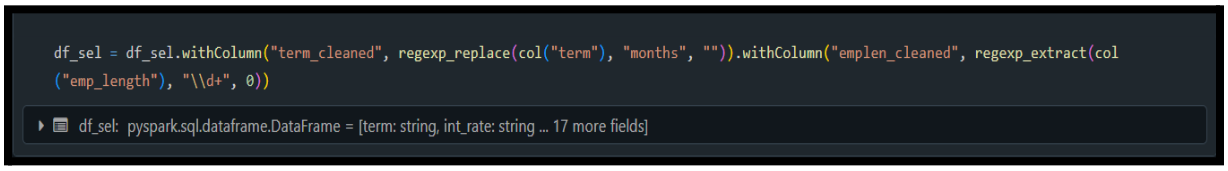


```
from pyspark.sql.functions import regexp_replace, regexp_extract
from pyspark.sql.functions import col

regex_string = 'years|year|\\+|\\<'
df_sel.select(regexp_replace(col("emp_length"), regex_string, "").alias("emp_length_clean"), col("emp_length")).show(10)
```

▸ (1) Spark Jobs

```
+----------------+----------+
|emp_length_clean|emp_length|
+----------------+----------+
|              1 |  < 1 year|
|             10 | 10+ years|
|              1 |  < 1 year|
|             n/a|       n/a|
|              5 |   5 years|
|              9 |   9 years|
|              3 |   3 years|
|             10 | 10+ years|
|             n/a|       n/a|
|             10 | 10+ years|
+----------------+----------+
only showing top 10 rows
```

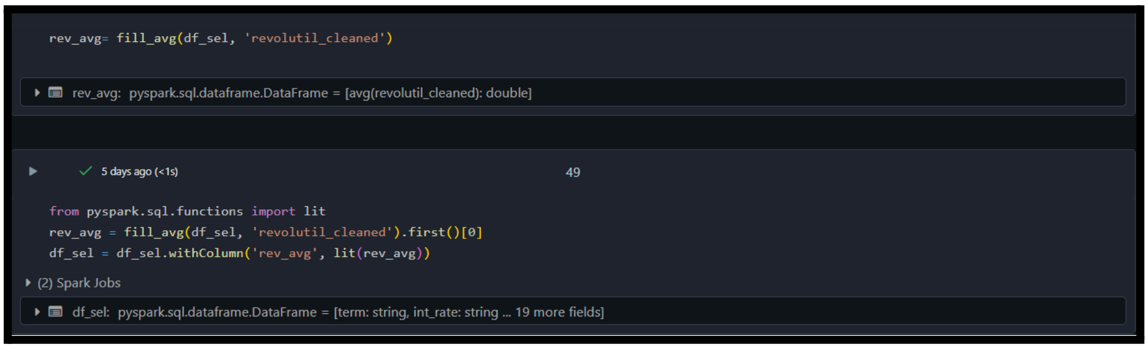
2. Converting Data Types: String columns were converted to numeric types to facilitate arithmetic operations and statistical analysis:



```
df_sel = df_sel.withColumn("term_cleaned", regexp_replace(col("term"), "months", "")).withColumn("emplen_cleaned", regexp_extract(col
("emp_length"), "\\d+", 0))
```

▸ df_sel: pyspark.sql.dataframe.DataFrame = [term: string, int_rate: string ... 17 more fields]

3. Imputing Missing Values: Missing data was addressed by calculating averages or using default values:



```
rev_avg= fill_avg(df_sel, 'revolutil_cleaned')
```

▸ rev_avg: pyspark.sql.dataframe.DataFrame = [avg(revolutil_cleaned): double]

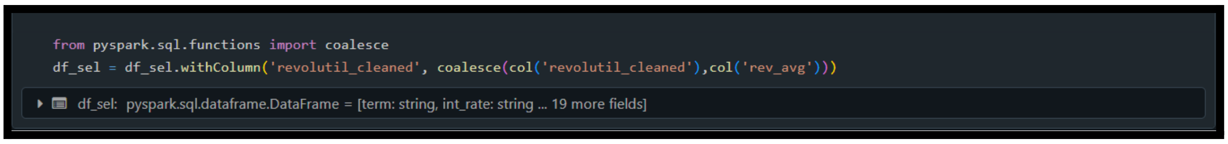
▸ ✓ 5 days ago (<1s) 49

```
from pyspark.sql.functions import lit
rev_avg = fill_avg(df_sel, 'revolutil_cleaned').first()[0]
df_sel = df_sel.withColumn('rev_avg', lit(rev_avg))
```

▸ (2) Spark Jobs

▸ df_sel: pyspark.sql.dataframe.DataFrame = [term: string, int_rate: string ... 19 more fields]

4. Merging Columns: Related columns (e.g., DTI and DTI Joint) were consolidated to simplify the dataset and reduce redundancy:

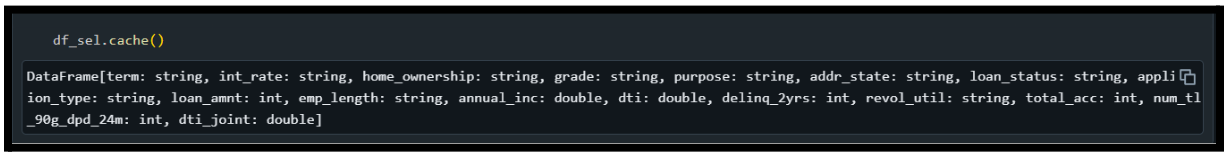


```
from pyspark.sql.functions import coalesce
df_sel = df_sel.withColumn('revolutil_cleaned', coalesce(col('revolutil_cleaned'),col('rev_avg')))
```

```
df_sel:  pyspark.sql.dataframe.DataFrame = [term: string, int_rate: string ... 19 more fields]
```

Frequent operations on the dataset were optimized by caching:

df.cache()



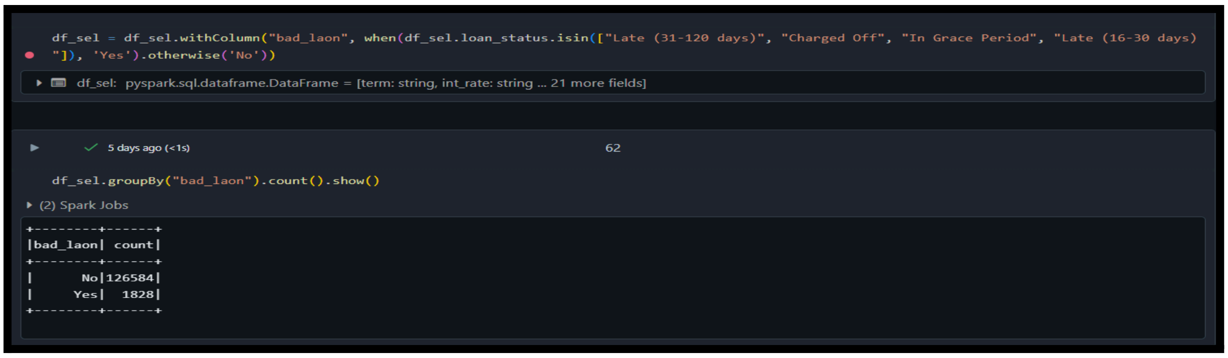
```
df_sel.cache()
```

```
DataFrame[term: string, int_rate: string, home_ownership: string, grade: string, purpose: string, addr_state: string, loan_status: string, appli
ion_type: string, loan_amnt: int, emp_length: string, annual_inc: double, dti: double, delinq_2yrs: int, revol_util: string, total_acc: int, num_tl
_90g_dpd_24m: int, dti_joint: double]
```

Caching stored the dataset in memory, significantly improving the performance of iterative queries and transformations. This was particularly useful for exploratory data analysis and repeated calculations.

Derived Insights and New Columns:

- Categorizing Loan Status: Loan statuses were standardized into two categories (Good Loan and Bad Loan) based on business rules:



```
df_sel = df_sel.withColumn("bad_laon", when(df_sel.loan_status.isin(["Late (31-120 days)", "Charged Off", "In Grace Period", "Late (16-30 days)
"]), 'Yes').otherwise('No'))
```

```
df_sel:  pyspark.sql.dataframe.DataFrame = [term: string, int_rate: string ... 21 more fields]
```

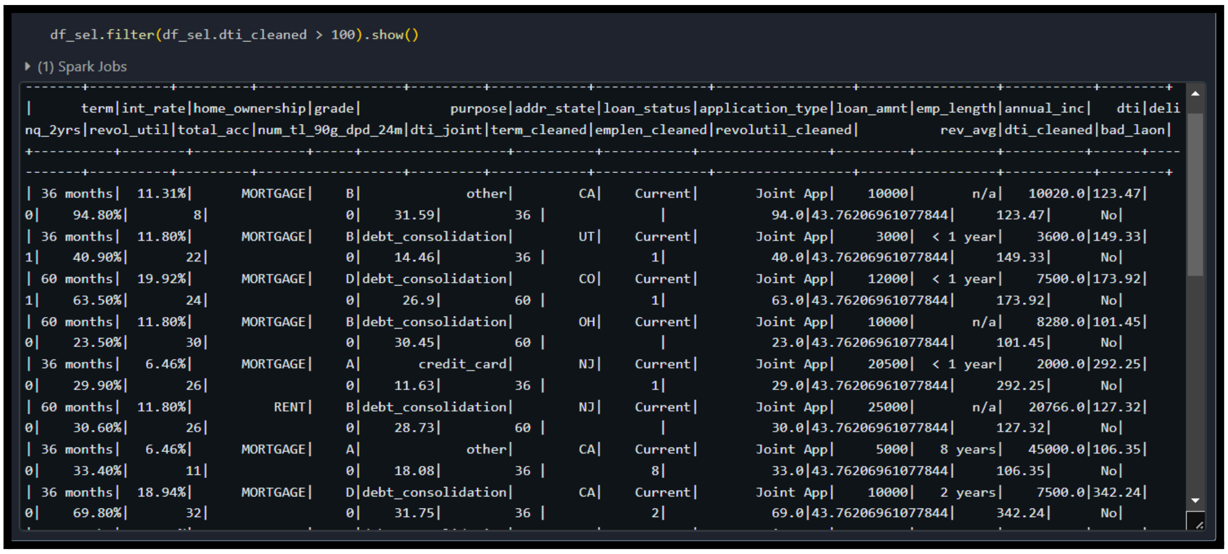
5 days ago (<1s) 62

```
df_sel.groupBy("bad_laon").count().show()
```

(2) Spark Jobs

```
+--------+------+
|bad_laon| count|
+--------+------+
|      No|126584|
|     Yes|  1828|
+--------+------+
```

- Flagging Risky Borrowers: Borrowers with high Debt-to-Income (DTI) ratios were flagged as potentially risky:



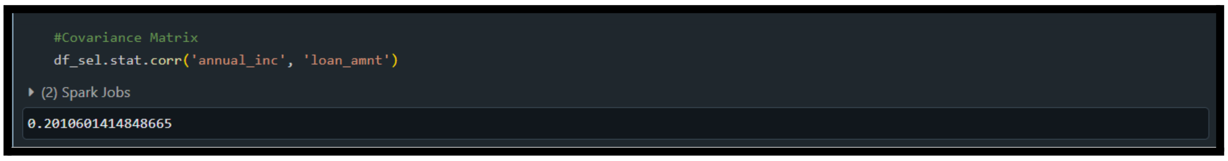
```
df_sel.filter(df_sel.dti_cleaned > 100).show()
```

(1) Spark Jobs

```
-------+---------+--------+----------------+--------+-----------+--------------+----------------+---------+----------+--------+
|      term|int_rate|home_ownership|grade|           purpose|addr_state|loan_status|application_type|loan_amnt|emp_length|annual_inc|   dti|deli
nq_2yrs|revol_util|total_acc|num_tl_90g_dpd_24m|dti_joint|term_cleaned|emplen_cleaned|revolutil_cleaned|          rev_avg|dti_cleaned|bad_laon|
+----------+--------+-------------+-----+------------------+----------+-----------+----------------+---------+----------+----------+------+----
-------+---------+--------+----------------+--------+-----------+--------------+----------------+---------+----------+--------+
| 36 months|  11.31%|     MORTGAGE|    B|             other|        CA|    Current|       Joint App|    10000|       n/a|   10020.0|123.47|
0|    94.80%|        8|                 0|    31.59|          36 |             |             94.0|43.76206961077844|     123.47|      No|
| 36 months|  11.80%|     MORTGAGE|    B|debt_consolidation|        UT|    Current|       Joint App|     3000|  < 1 year|    3600.0|149.33|
1|    40.90%|       22|                 0|    14.46|          36 |            1|             40.0|43.76206961077844|     149.33|      No|
| 60 months|  19.92%|     MORTGAGE|    D|debt_consolidation|        CO|    Current|       Joint App|    12000|  < 1 year|    7500.0|173.92|
1|    63.50%|       24|                 0|     26.9|          60 |            1|             63.0|43.76206961077844|     173.92|      No|
| 60 months|  11.80%|     MORTGAGE|    B|debt_consolidation|        OH|    Current|       Joint App|    10000|       n/a|    8280.0|101.45|
0|    23.50%|       30|                 0|    30.45|          60 |             |             23.0|43.76206961077844|     101.45|      No|
| 36 months|   6.46%|     MORTGAGE|    A|       credit_card|        NJ|    Current|       Joint App|    20500|  < 1 year|    2000.0|292.25|
0|    29.90%|       26|                 0|    11.63|          36 |            1|             29.0|43.76206961077844|     292.25|      No|
| 60 months|  11.80%|         RENT|    B|debt_consolidation|        NJ|    Current|       Joint App|    25000|       n/a|   20766.0|127.32|
0|    30.60%|       26|                 0|    28.73|          60 |             |             30.0|43.76206961077844|     127.32|      No|
| 36 months|   6.46%|     MORTGAGE|    A|             other|        CA|    Current|       Joint App|     5000|   8 years|   45000.0|106.35|
0|    33.40%|       11|                 0|    18.08|          36 |            8|             33.0|43.76206961077844|     106.35|      No|
| 36 months|  18.94%|     MORTGAGE|    D|debt_consolidation|        CA|    Current|       Joint App|    10000|   2 years|    7500.0|342.24|
0|    69.80%|       32|                 0|    31.75|          36 |            2|             69.0|43.76206961077844|     342.24|      No|
```

Analytical Techniques:

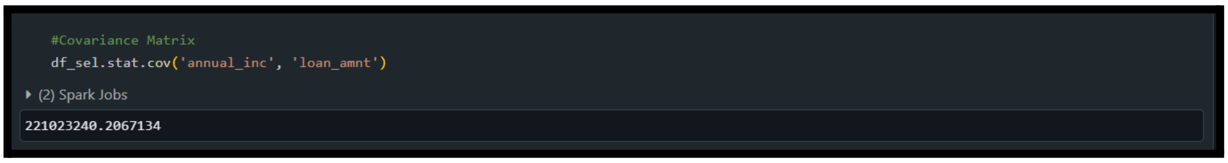
- Correlation and Covariance: Relationships between variables were explored to understand dependencies and inform feature selection:



```
#Covariance Matrix
df_sel.stat.corr('annual_inc', 'loan_amnt')
```

(2) Spark Jobs

```
0.2010601414848665
```

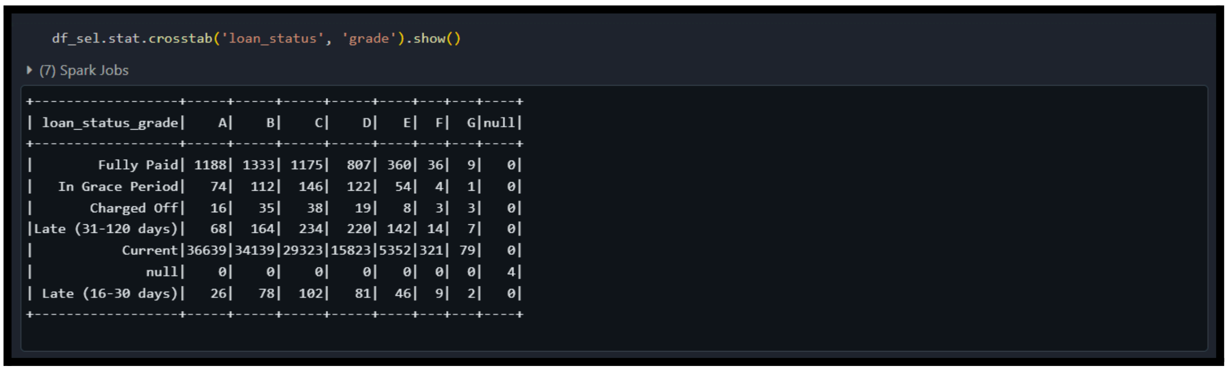


```
#Covariance Matrix
df_sel.stat.cov('annual_inc', 'loan_amnt')
```

(2) Spark Jobs

```
221023240.2067134
```

- Cross-tabulation: Relationships between categorical variables, such as loan status and grade, were analyzed using:



```
df_sel.stat.crosstab('loan_status', 'grade').show()
```

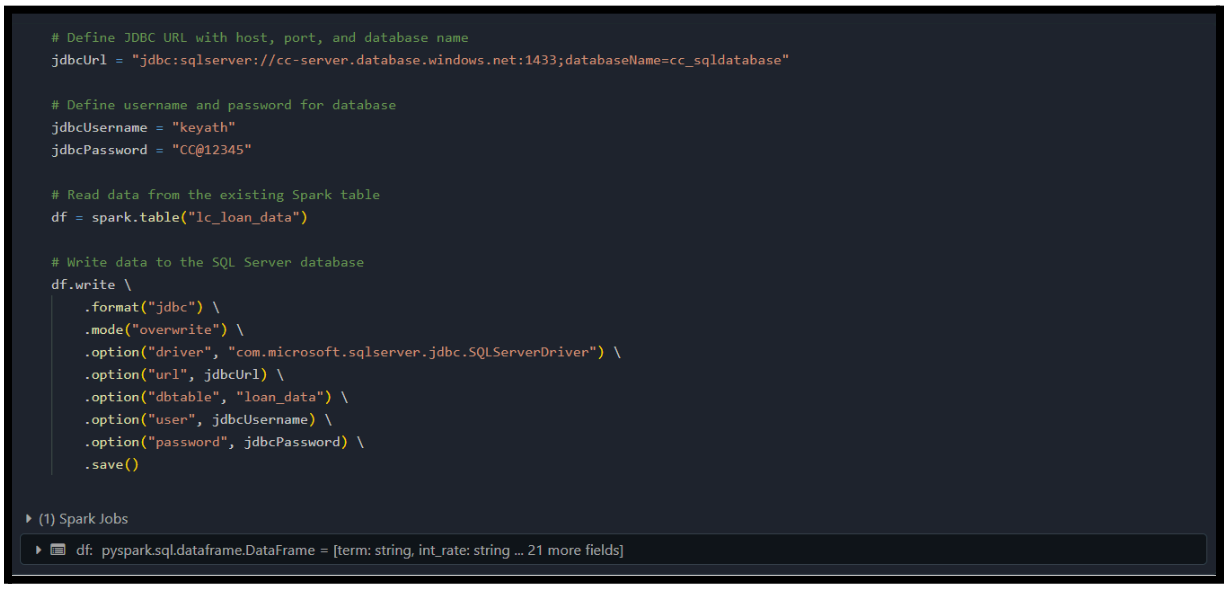
(7) Spark Jobs

```
+------------------+-----+-----+-----+-----+----+---+---+----+
| loan_status_grade|    A|    B|    C|    D|   E|  F|  G|null|
+------------------+-----+-----+-----+-----+----+---+---+----+
|        Fully Paid| 1188| 1333| 1175|  807| 360| 36|  9|   0|
|   In Grace Period|   74|  112|  146|  122|  54|  4|  1|   0|
|       Charged Off|   16|   35|   38|   19|   8|  3|  3|   0|
|Late (31-120 days)|   68|  164|  234|  220| 142| 14|  7|   0|
|           Current|36639|34139|29323|15823|5352|321| 79|   0|
|              null|    0|    0|    0|    0|   0|  0|  0|   4|
| Late (16-30 days)|   26|   78|  102|   81|  46|  9|  2|   0|
+------------------+-----+-----+-----+-----+----+---+---+----+
```

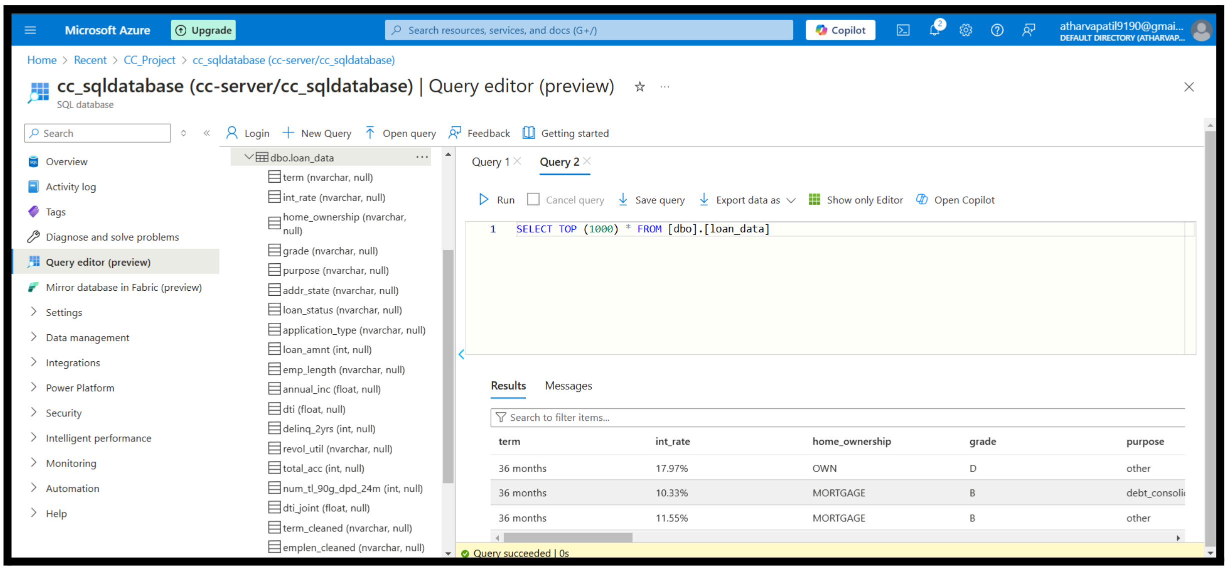
- Frequent Items: Frequently occurring categories were identified to prioritize key variables for analysis.

Data Storage

The cleaned and processed dataset was saved in Azure SQL database for easy querying and reporting of the output data.



```
# Define JDBC URL with host, port, and database name
jdbcUrl = "jdbc:sqlserver://cc-server.database.windows.net:1433;databaseName=cc_sqldatabase"

# Define username and password for database
jdbcUsername = "keyath"
jdbcPassword = "CC@12345"

# Read data from the existing Spark table
df = spark.table("lc_loan_data")

# Write data to the SQL Server database
df.write \
    .format("jdbc") \
    .mode("overwrite") \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .option("url", jdbcUrl) \
    .option("dbtable", "loan_data") \
    .option("user", jdbcUsername) \
    .option("password", jdbcPassword) \
    .save()

▸ (1) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [term: string, int_rate: string ... 21 more fields]
```



```
dbo.loan_data
  term (nvarchar, null)
  int_rate (nvarchar, null)
  home_ownership (nvarchar, null)
  grade (nvarchar, null)
  purpose (nvarchar, null)
  addr_state (nvarchar, null)
  loan_status (nvarchar, null)
  application_type (nvarchar, null)
  loan_amnt (int, null)
  emp_length (nvarchar, null)
  annual_inc (float, null)
  dti (float, null)
  delinq_2yrs (int, null)
  revol_util (nvarchar, null)
  total_acc (int, null)
  num_tl_90g_dpd_24m (int, null)
  dti_joint (float, null)
  term_cleaned (nvarchar, null)
  emplen_cleaned (nvarchar, null)
```

```sql
SELECT TOP (1000) * FROM [dbo].[loan_data]
```

term	int_rate	home_ownership	grade	purpose
36 months	17.97%	OWN	D	other
36 months	10.33%	MORTGAGE	B	debt_consoli
36 months	11.55%	MORTGAGE	B	other

Query succeeded | 0s

DEVELOPMENT OF THE PIPELINE:

i. Data extraction tool

Lending Club is a peer-to-peer loaning company that offers personal loans, small business loans, and auto refinancing. It connects borrowers, who possibly have poor credit histories and need quick access to funds, with investors who will fund their loans. A borrower can request a loan-for example, a $20,000 loan-and his or her profile is made available to potential investors. Investors can look through these profiles and decide whether the risk is acceptable to them. If they decide to invest, the loan gets funded, and the borrower repays the loan in monthly instalments along with interest charges. Interest rates on these loans are generally higher than those that traditional banks give, reflecting the increased risk for investors

The dataset for Lending Club is available in Kaggle and big enough to perform extensive analysis. A quarter of this dataset has been selected for faster processing and to understand the fundamentals of Spark. The dataset is in CSV format, with comma delimiters, and the first row contains headers, which Spark uses to infer the schema. Although the dataset is semi-structured and uncleaned in nature, it is downloaded to the system and uploaded first to a staging area. Here, the data is cleaned and transformed further before being actually stored in the target system, which could be an Azure SQL Database, for its further utilization and analysis.

ii. Data Processing tool

Apache Spark

Apache Spark is a powerful, distributed data processing engine at the heart of the data extraction and processing workflow mentioned in the transcript. Apache Spark is a powerful, distributed data processing engine that is designed to efficiently handle large volumes of data. The important features among others are distributed processing, meaning Spark divides large sets of data into smaller fragments and processes them on various worker nodes to ensure scalability and speed. In the same way, the DataFrame API from Spark provides a high-level abstraction for data manipulation-a sort of Pandas in Python-but optimized for distributed systems. Using functions such as spark.read.format("csv") with options like inferSchema=True and header=True, Spark can load and interpret structured data, including large CSV files, in an efficient manner. Automatic schema detection is also provided through the option of inferSchema in Spark, which removes the need to manually define the schema, especially for large and complex datasets. Additionally, Spark supports SQL queries. It enables users to work with data using SQL syntax by creating temporary views with createOrReplaceTempView. This flexibility lets users seamlessly switch between DataFrame operations and SQL queries. Another important feature of Spark is lazy evaluation: the execution plan (DAG) is only triggered when an action like show() or count() is called. This optimizes performance by avoiding superfluous computation and allows Spark to plan the most efficient execution strategy for data processing.

Databricks

Databricks is a collaborative environment to run Apache Spark, with many additional features to make one's workflow smooth. Databricks offers a very user-friendly notebook interface where the user can run Spark commands in various languages, such as Python, SQL, and Scala, in the same environment. This interface supports "magic commands," such as %sql, which let users specify the language of each cell and, therefore, easily combine different languages for data processing. One of the salient features of Databricks is that it allows users to cache data in executor memory using simple commands like df.cache(). This drastically optimizes performance, accelerating repeated operations on the same dataset-especially iterative tasks or complex workflows. Another strong feature of Databricks is that it integrated robust visualization tools directly into the notebook environment, making it quite easy for users to build graphs, charts, and other visualizations with data. These visualizations help to assess data trends and make decisions themselves without exporting data necessarily out of this environment. Boosted productivity is supported due to an integrated interactive environment with caching and visualizations on top, offering the Databricks advantage through efficient data exploration and analyses; thus, making Databricks irreplaceable in the armamentarium of a data engineer or analyst operating in a big data perspective.

CHALLENGES & LESSONS LEARNED:

Understanding what tools were appropriate for what and how they integrated into the Azure ecosystem took time to learn within the expansive suite of services that are available within Microsoft Azure. Much later, setting up the resources and ensuring proper access control, especially using RBAC and configuration of firewalls.
We had to establish seamless integration between Databricks, Azure Data Lake Storage, and Azure SQL Database by creating appropriate credentials and authentication methods such as Service Principals and OAuth. Another challenge was debugging any error in a distributed environment since working with Apache Spark involves asynchronous execution of tasks with limited visibility of logs at certain levels. Writing spark jobs for handling semi-structured data and understanding of partition, memory management. We learned the configuration and management of Databricks, including the configuration of worker and driver nodes for performance optimization. Understanding how Databricks integrates with other Azure services prepared us for making more robust and seamless pipelines in projects. Working with SQL and Azure SQL Database presented some unique challenges, especially when it comes to handling large datasets.

Efficient SQL scripting required avoiding redundant operations, ensuring schema validation before data uploads, and addressing integration challenges between Azure SQL Database and Databricks. Careful schema mapping and compatibility checks were crucial for transferring processed data, while query optimization techniques like indexing improved performance. This process enhanced our understanding of Azure SQL Database and its role in robust data pipelines.

Working with Python in Spark's distributed environment posed unique challenges, including deciphering complex tracebacks, maintaining modular and reusable code, and managing dependencies in Databricks. We gained proficiency in PySpark for handling large datasets and learned effective dependency management in shared environments.

Integrating Databricks with Azure SQL Database further emphasized the importance of schema mapping and query optimization for seamless data migration and performance.

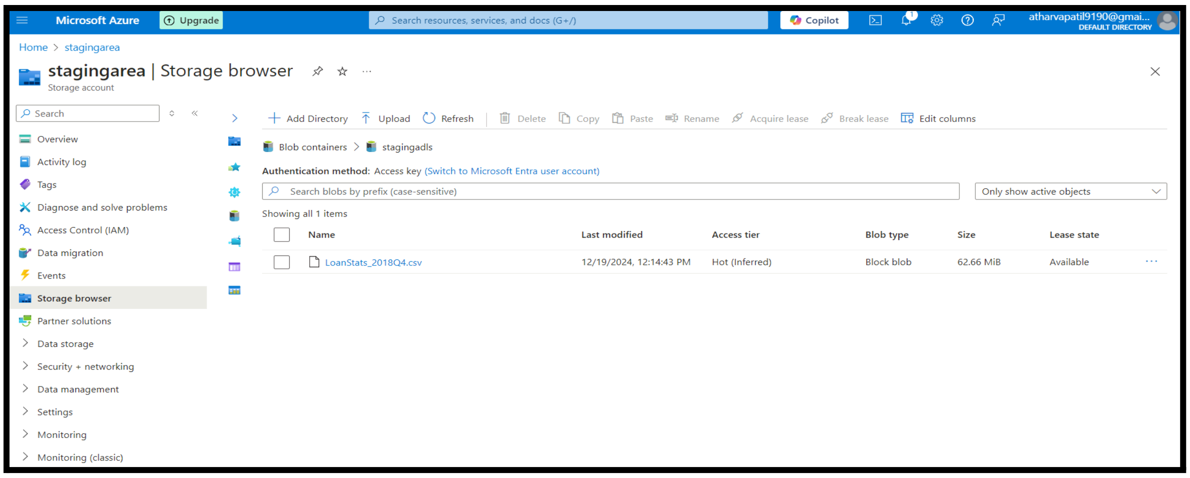
Moreover, there was also a challenge in managing scalability and concurrent connections that required being carefully configured and observed in order to meet the workload requirements. The research into scalability features such as elastic pools within Azure SQL Database empowered us in handling larger workloads. In the process of migrating data, the critical need for thorough validation at each point in the data pipeline became underlined.

REFERENCES:

- Azure Databricks documentation | Microsoft Learn
- **Cloud Technologies CSC1142 Dr. Michael Scriney**
- **#9. Azure Data Bricks - Read from CSV & Write to Az. SQL table**
- **Connect to Azure Data Lake Storage Gen2 and Blob Storage - Azure Databricks | Microsoft Learn**
- **Azure SQL Database documentation**
- **Deepak Goyal: https://adeus.azurelib.com/**

Screenshots of the Project -

1. **ADLS - loanstats_2018 is the raw file which is pushed in the adls for staging.**
2. **Azure Databricks set up (Transform)**



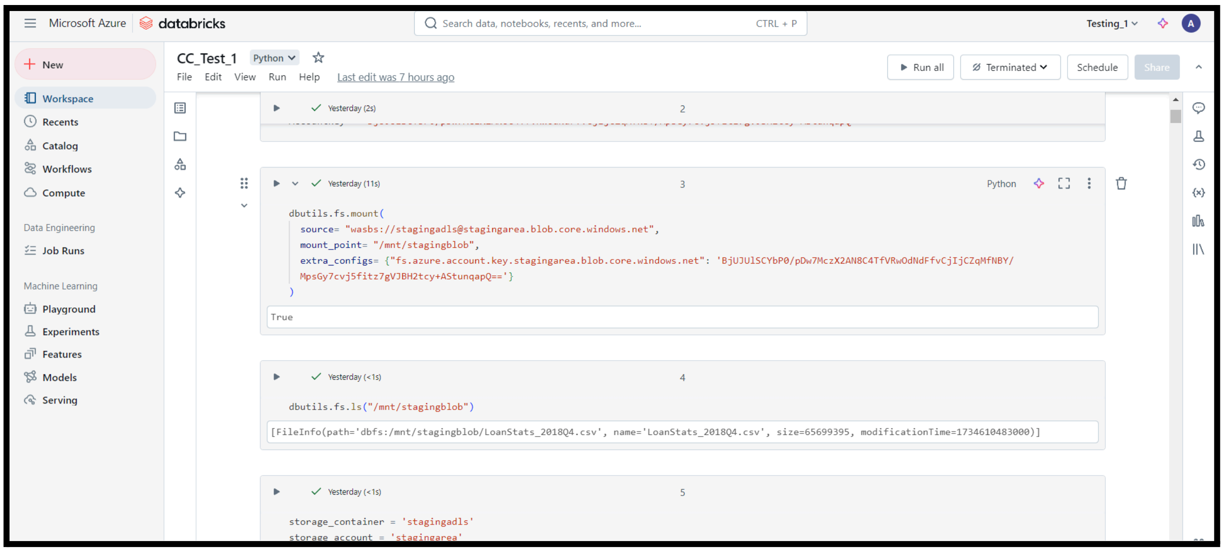
stagingarea | Storage browser

Blob containers > stagingadls

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 1 items

Name	Last modified	Access tier	Blob type	Size	Lease state
LoanStats_2018Q4.csv	12/19/2024, 12:14:43 PM	Hot (Inferred)	Block blob	62.66 MiB	Available



```
dbutils.fs.mount(
  source= "wasbs://stagingadls@stagingarea.blob.core.windows.net",
  mount_point= "/mnt/stagingblob",
  extra_configs= {"fs.azure.account.key.stagingarea.blob.core.windows.net": 'BjUJUlSCYbP0/pDw7MczX2AN8C4TfVRwOdNdFfvCjIjCZqMfNBY/
  MpsGy7cvj5fitz7gVJBH2tcy+AStunqapQ=='}
)
```

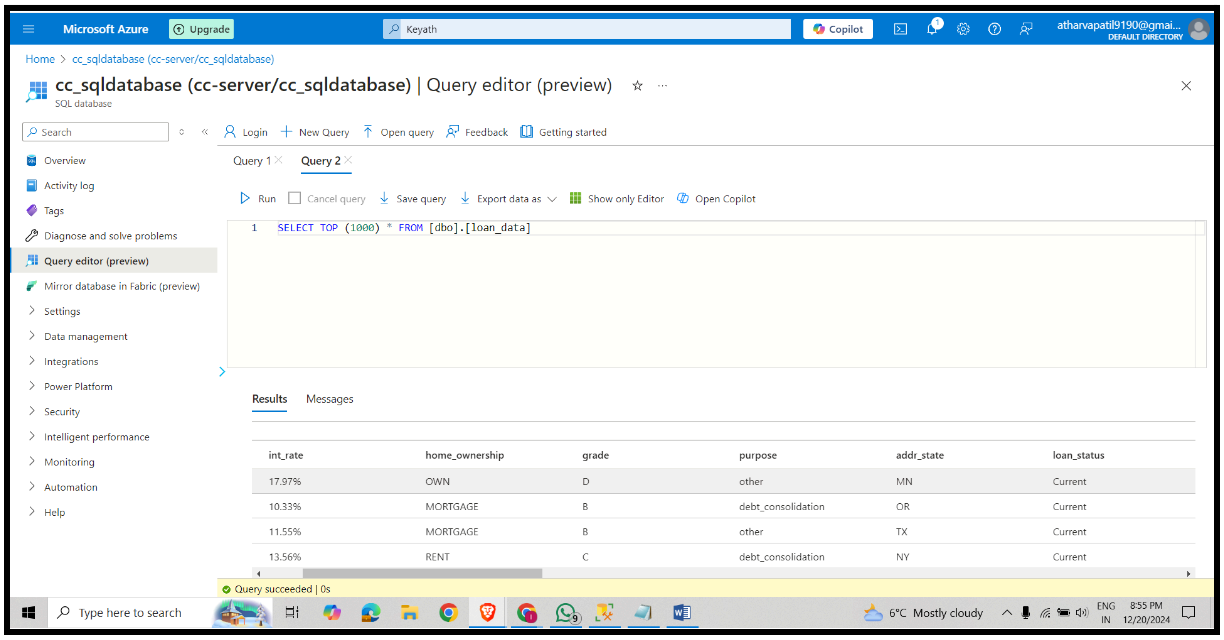
```
True
```

```
dbutils.fs.ls("/mnt/stagingblob")
```

```
[FileInfo(path='dbfs:/mnt/stagingblob/LoanStats_2018Q4.csv', name='LoanStats_2018Q4.csv', size=65699395, modificationTime=1734610483000)]
```

```
storage_container = 'stagingadls'
storage_account = 'stagingarea'
```

3. Azure SQL Database (Load)



```
SELECT TOP (1000) * FROM [dbo].[loan_data]
```

int_rate	home_ownership	grade	purpose	addr_state	loan_status
17.97%	OWN	D	other	MN	Current
10.33%	MORTGAGE	B	debt_consolidation	OR	Current
11.55%	MORTGAGE	B	other	TX	Current
13.56%	RENT	C	debt_consolidation	NY	Current

Query succeeded | 0s