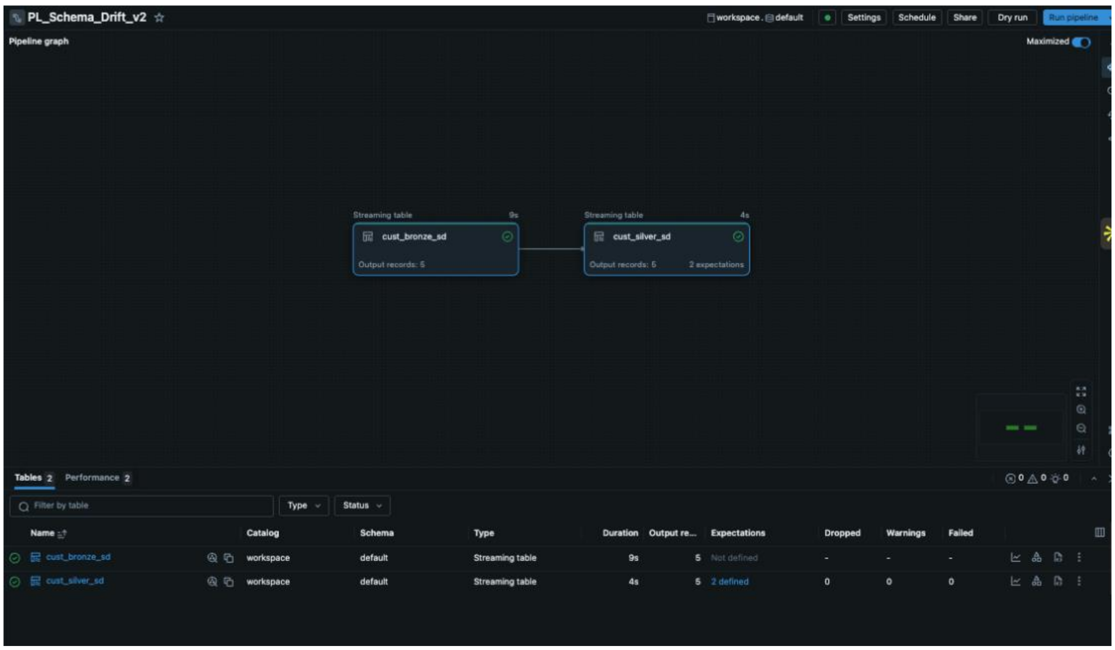
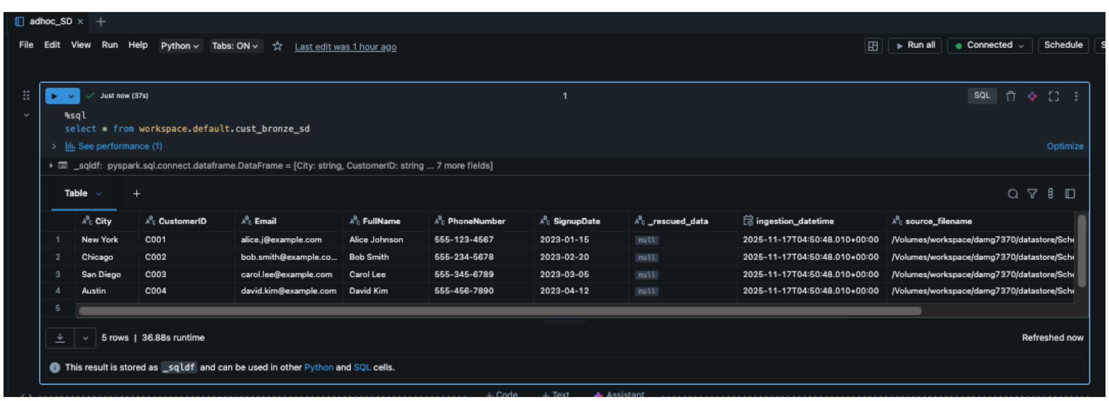
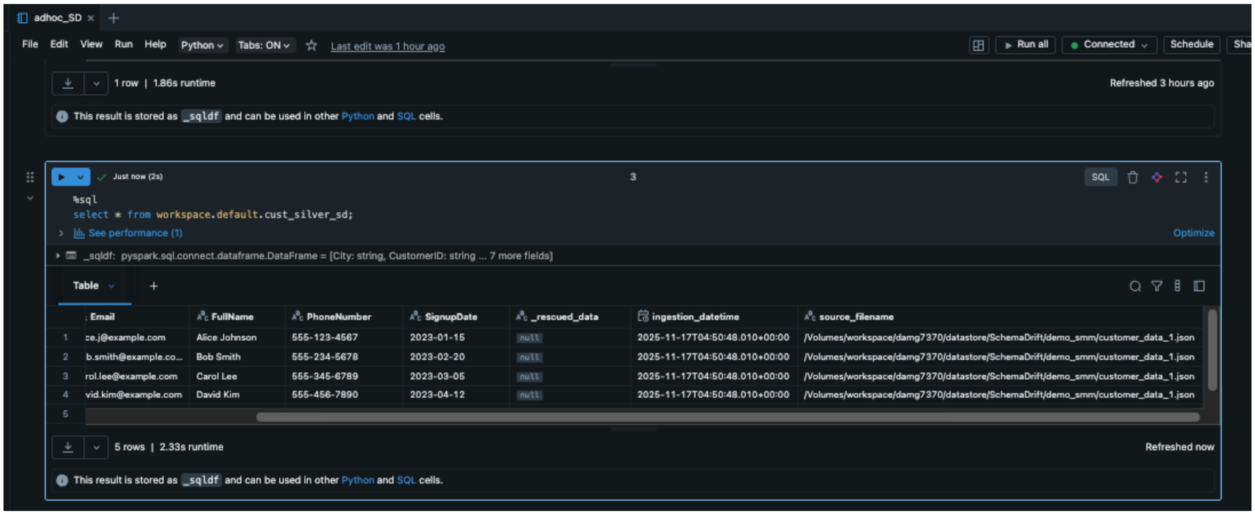
Schema Drift Replication Group 11

Hrishi Pal
Seamus McAvoy
Atharva Gadgil

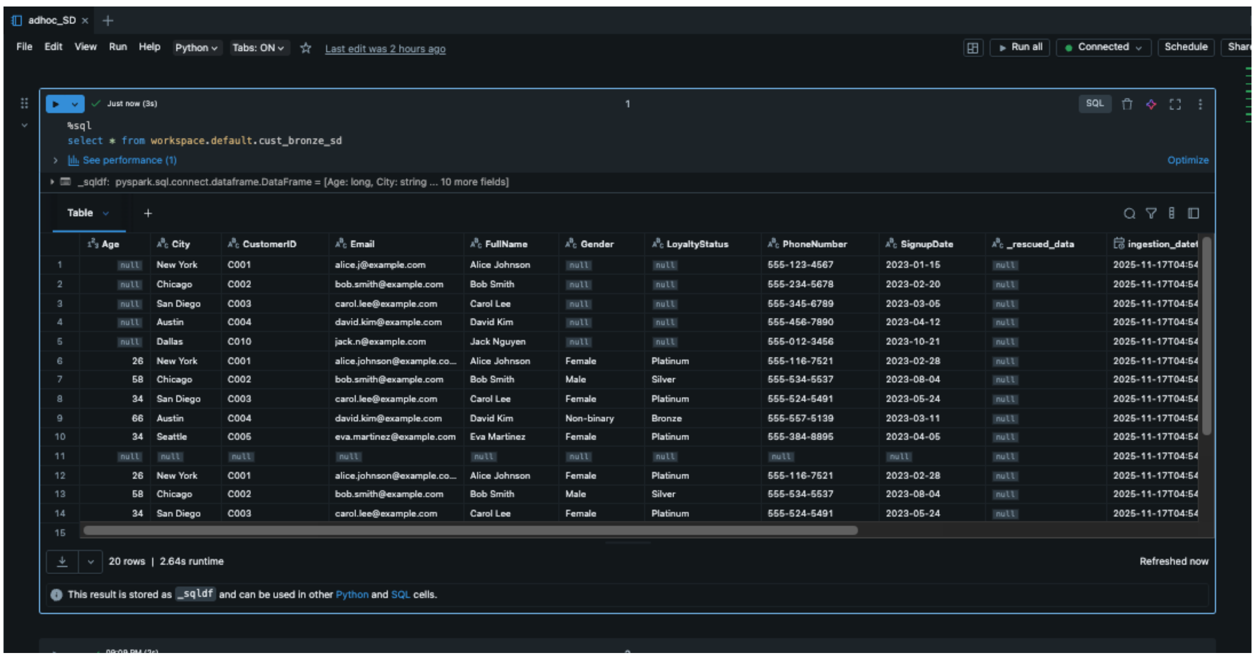
Plain Implementation



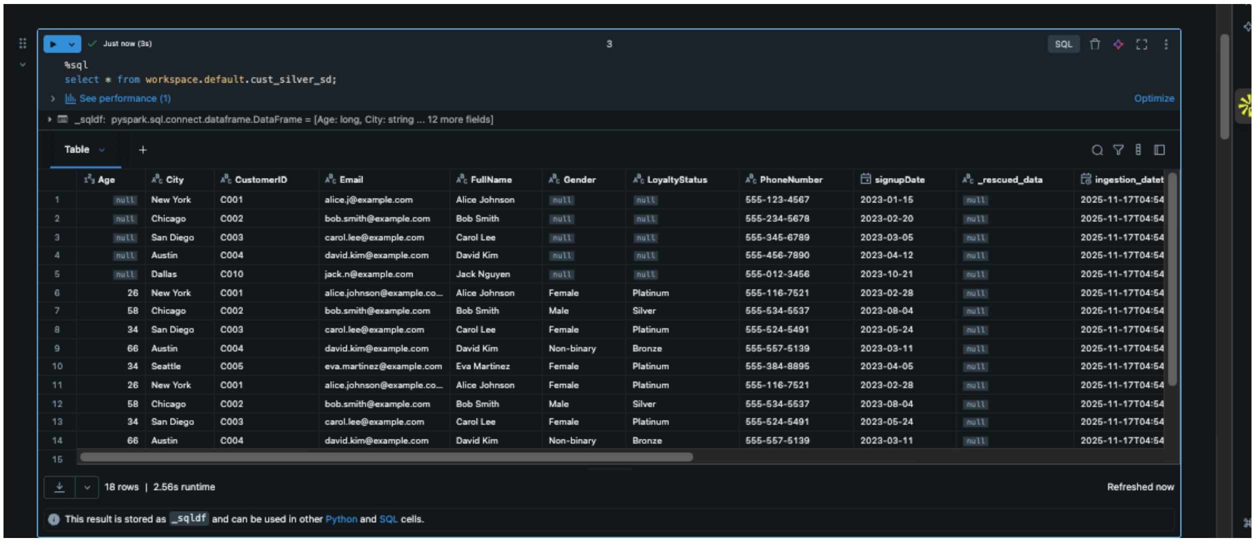




Customer_Data_2.json-



DataType Handling



```sql
%sql
select * from workspace.default.cust_silver_sd;
```

See performance (1)

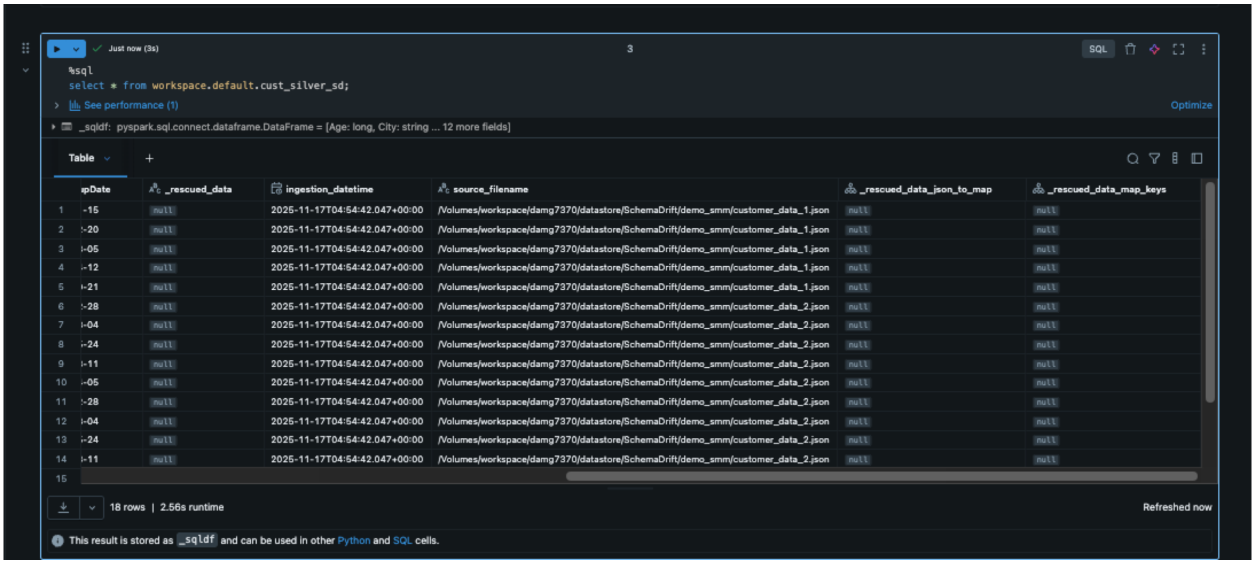
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [Age: long, City: string ... 12 more fields]

	Age	City	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	signupDate	_rescued_data	ingestion_datet
1	null	New York	C001	alice.j@example.com	Alice Johnson	null	null	555-123-4567	2023-01-15	null	2025-11-17T04:54
2	null	Chicago	C002	bob.smith@example.com	Bob Smith	null	null	555-234-5678	2023-02-20	null	2025-11-17T04:54
3	null	San Diego	C003	carol.lee@example.com	Carol Lee	null	null	555-345-6789	2023-03-05	null	2025-11-17T04:54
4	null	Austin	C004	david.kim@example.com	David Kim	null	null	555-456-7890	2023-04-12	null	2025-11-17T04:54
5	null	Dallas	C010	jack.n@example.com	Jack Nguyen	null	null	555-012-3456	2023-10-21	null	2025-11-17T04:54
6	26	New York	C001	alice.johnson@example.co...	Alice Johnson	Female	Platinum	555-116-7521	2023-02-28	null	2025-11-17T04:54
7	58	Chicago	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-534-5537	2023-08-04	null	2025-11-17T04:54
8	34	San Diego	C003	carol.lee@example.com	Carol Lee	Female	Platinum	555-524-5491	2023-05-24	null	2025-11-17T04:54
9	66	Austin	C004	david.kim@example.com	David Kim	Non-binary	Bronze	555-557-5139	2023-03-11	null	2025-11-17T04:54
10	34	Seattle	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-384-8895	2023-04-05	null	2025-11-17T04:54
11	26	New York	C001	alice.johnson@example.co...	Alice Johnson	Female	Platinum	555-116-7521	2023-02-28	null	2025-11-17T04:54
12	58	Chicago	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-534-5537	2023-08-04	null	2025-11-17T04:54
13	34	San Diego	C003	carol.lee@example.com	Carol Lee	Female	Platinum	555-524-5491	2023-05-24	null	2025-11-17T04:54
14	66	Austin	C004	david.kim@example.com	David Kim	Non-binary	Bronze	555-557-5139	2023-03-11	null	2025-11-17T04:54
15											

18 rows | 2.56s runtime

Refreshed now

This result is stored as `_sqldf` and can be used in other Python and SQL cells.



```sql
%sql
select * from workspace.default.cust_silver_sd;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [Age: long, City: string ... 12 more fields]

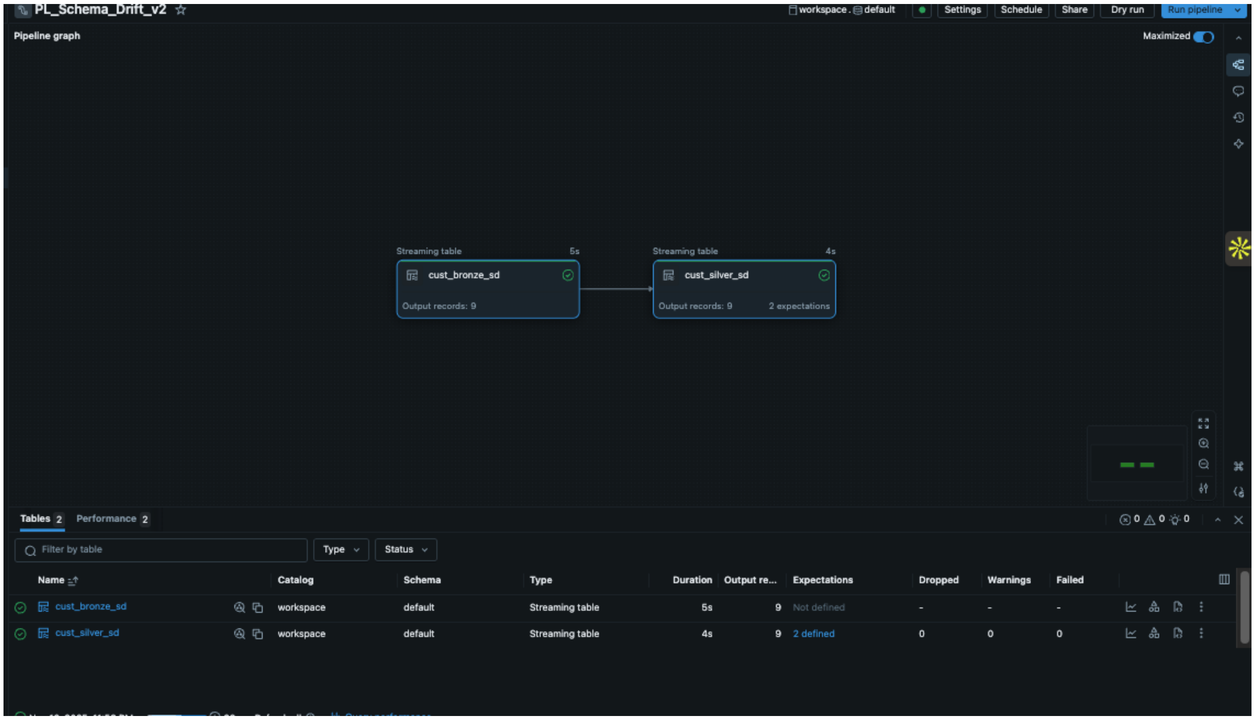
	ıpDate	_rescued_data	ingestion_datetime	source_filename	_rescued_data_json_to_map	_rescued_data_map_keys
1	-15	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_1.json	null	null
2	:-20	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_1.json	null	null
3	ı-05	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_1.json	null	null
4	ı-12	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_1.json	null	null
5	ı-21	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_1.json	null	null
6	:-28	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
7	ı-04	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
8	;-24	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
9	ı-11	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
10	;-05	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
11	:-28	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
12	ı-04	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
13	;-24	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
14	ı-11	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json	null	null
15						

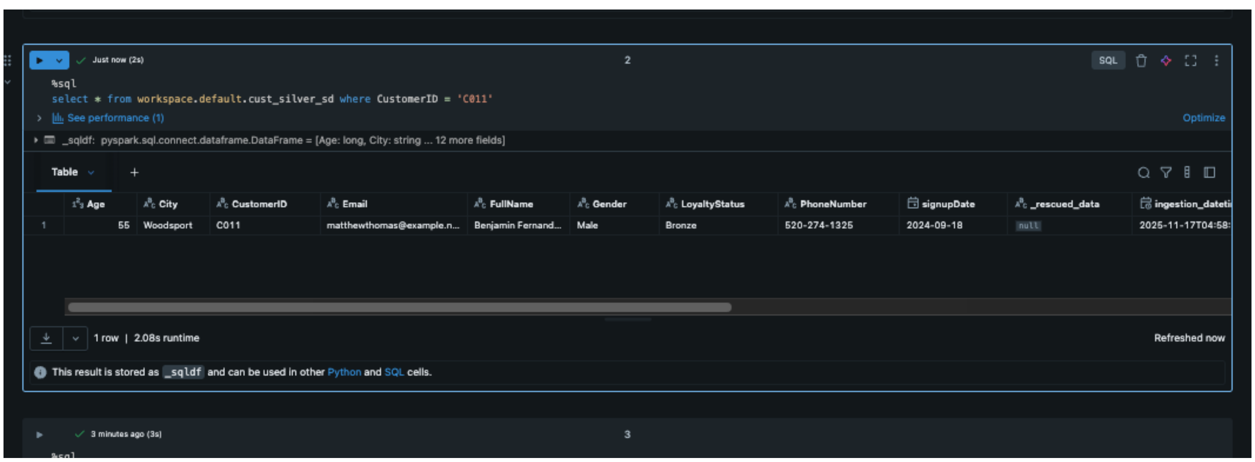
18 rows | 2.56s runtime

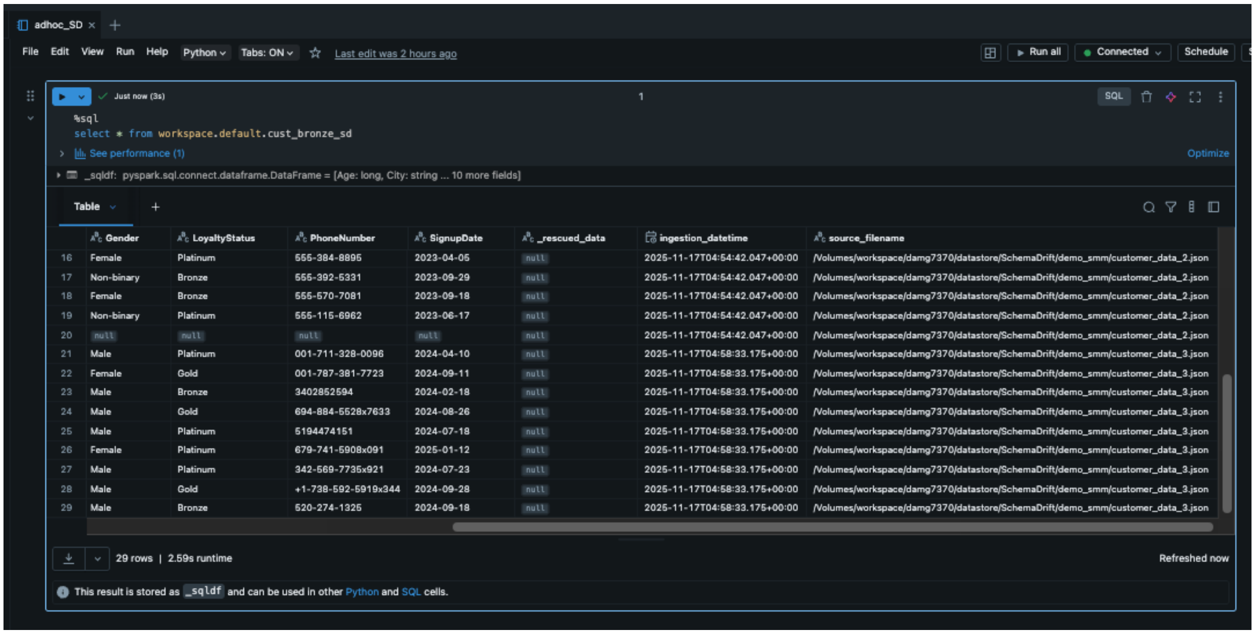
Refreshed now

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Customer_Data_3.json-

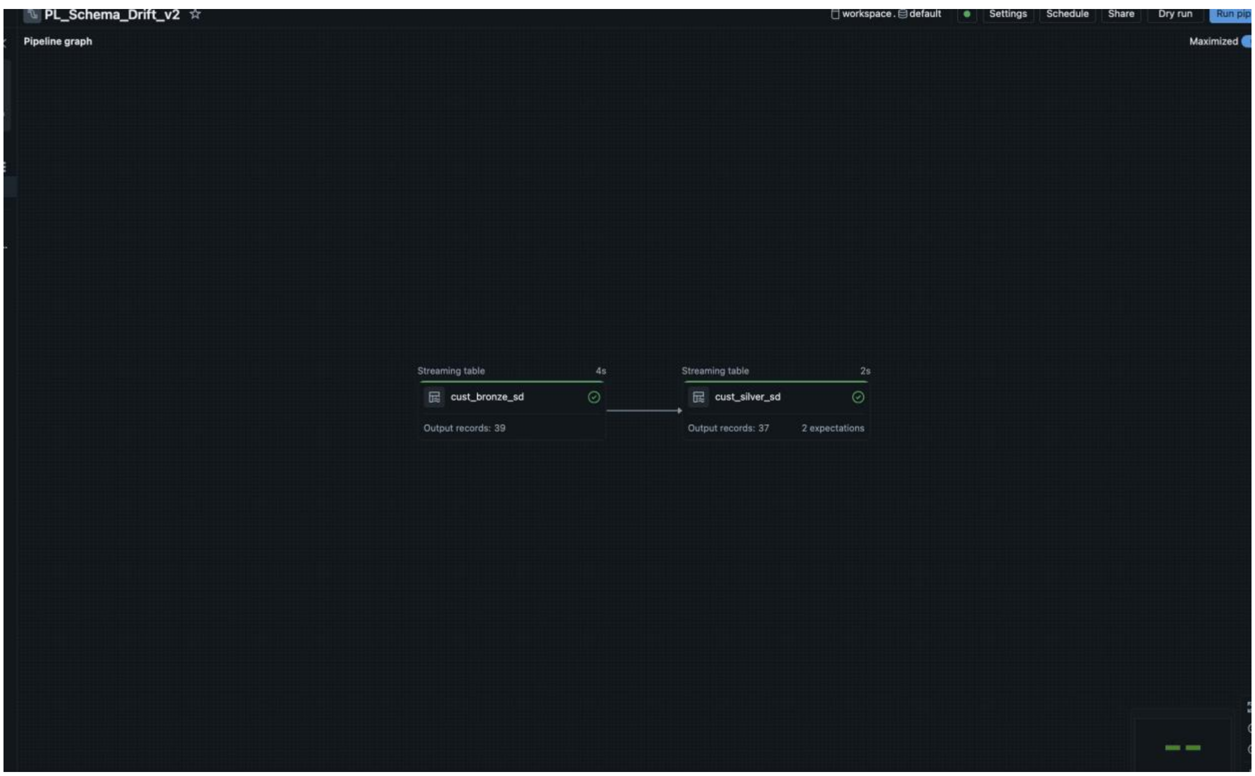




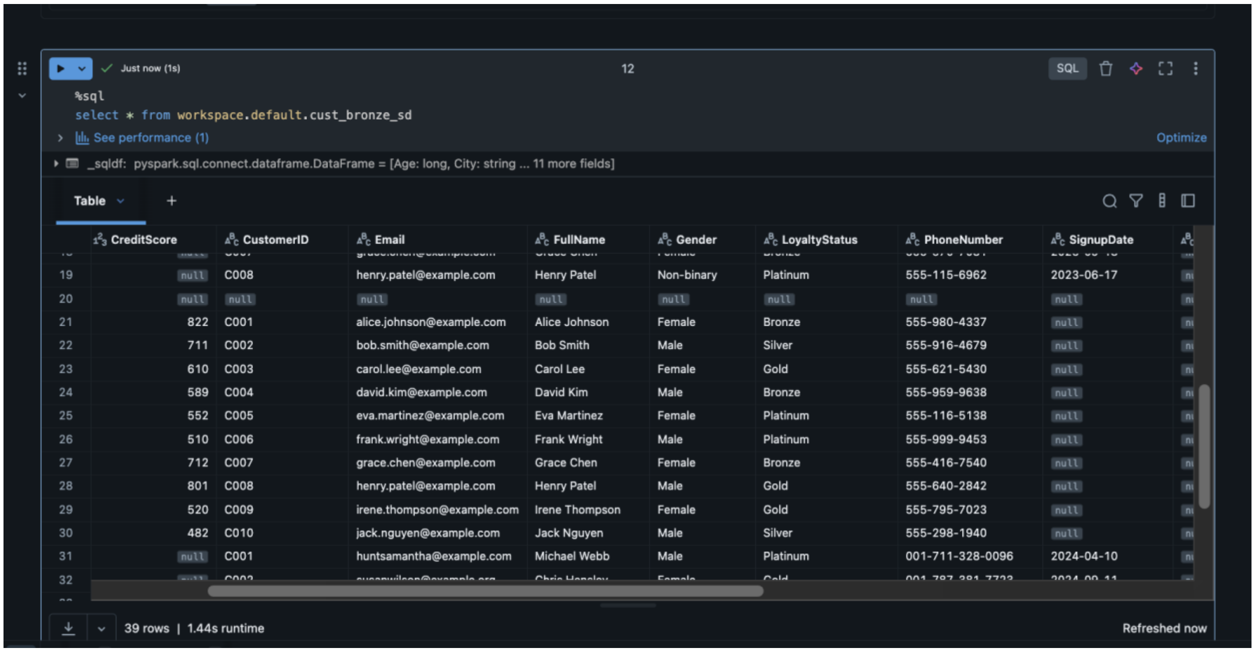


	Gender	LoyaltyStatus	PhoneNumber	SignupDate	_rescued_data	ingestion_datetime	source_filename
16	Female	Platinum	555-384-8895	2023-04-05	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json
17	Non-binary	Bronze	555-392-5331	2023-09-29	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json
18	Female	Bronze	555-570-7081	2023-09-18	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json
19	Non-binary	Platinum	555-115-6962	2023-06-17	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json
20	null	null	null	null	null	2025-11-17T04:54:42.047+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_2.json
21	Male	Platinum	001-711-328-0096	2024-04-10	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
22	Female	Gold	001-787-381-7723	2024-09-11	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
23	Male	Bronze	3402852594	2024-02-18	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
24	Male	Gold	694-884-5528x7633	2024-08-26	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
25	Male	Platinum	5194474151	2024-07-18	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
26	Female	Platinum	679-741-5908x091	2025-01-12	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
27	Male	Platinum	342-569-7735x921	2024-07-23	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
28	Male	Gold	+1-738-592-5919x344	2024-09-28	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json
29	Male	Bronze	520-274-1325	2024-09-18	null	2025-11-17T04:58:33.175+00:00	/Volumes/workspace/damg7370/datastore/SchemaDrift/demo_smm/customer_data_3.json

Missing CreditSore logic fix-



Bronze



```
%sql
select * from workspace.default.cust_bronze_sd
```

See performance (1)

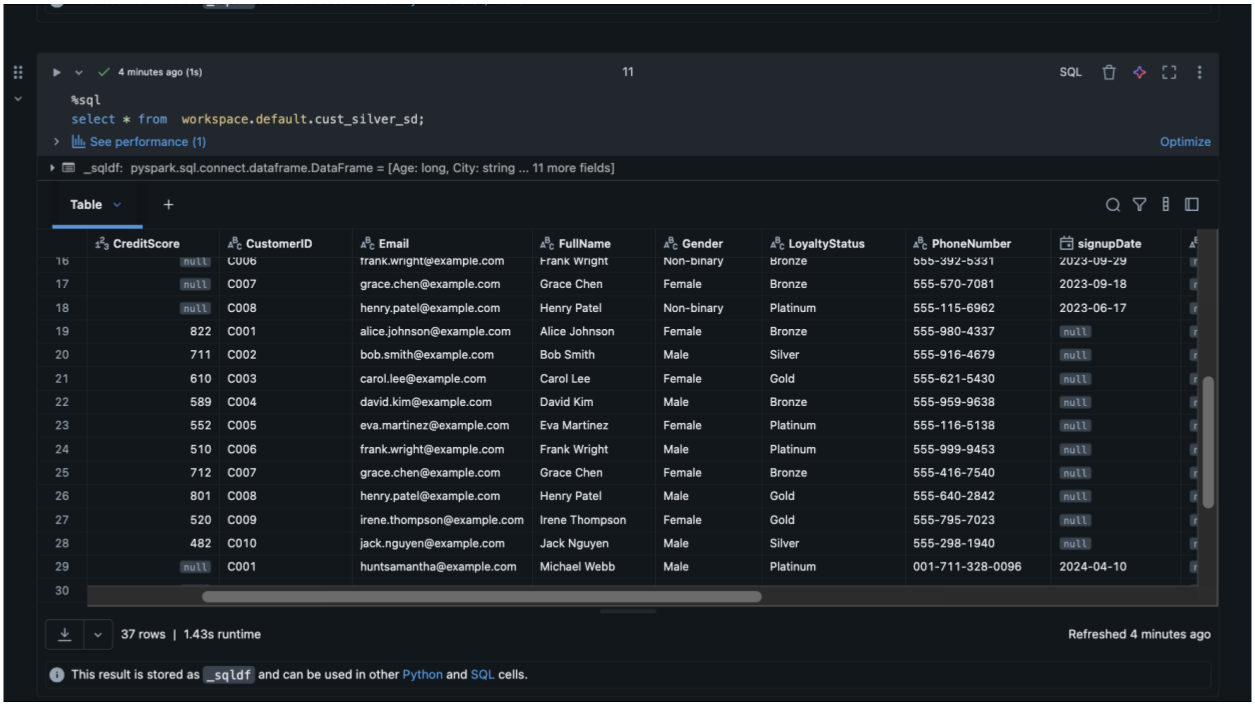
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [Age: long, City: string ... 11 more fields]

	CreditScore	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	SignupDate
19	null	C008	henry.patel@example.com	Henry Patel	Non-binary	Platinum	555-115-6962	2023-06-17
20	null	null	null	null	null	null	null	null
21	822	C001	alice.johnson@example.com	Alice Johnson	Female	Bronze	555-980-4337	null
22	711	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-916-4679	null
23	610	C003	carol.lee@example.com	Carol Lee	Female	Gold	555-621-5430	null
24	589	C004	david.kim@example.com	David Kim	Male	Bronze	555-959-9638	null
25	552	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-116-5138	null
26	510	C006	frank.wright@example.com	Frank Wright	Male	Platinum	555-999-9453	null
27	712	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-416-7540	null
28	801	C008	henry.patel@example.com	Henry Patel	Male	Gold	555-640-2842	null
29	520	C009	irene.thompson@example.com	Irene Thompson	Female	Gold	555-795-7023	null
30	482	C010	jack.nguyen@example.com	Jack Nguyen	Male	Silver	555-298-1940	null
31	null	C001	huntsamantha@example.com	Michael Webb	Male	Platinum	001-711-328-0096	2024-04-10

39 rows | 1.44s runtime

Refreshed now

Silver



```
%sql
select * from  workspace.default.cust_silver_sd;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [Age: long, City: string ... 11 more fields]

	CreditScore	CustomerID	Email	FullName	Gender	LoyaltyStatus	PhoneNumber	signupDate
16	null	C006	frank.wright@example.com	Frank Wright	Non-binary	Bronze	555-392-5331	2023-09-29
17	null	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-570-7081	2023-09-18
18	null	C008	henry.patel@example.com	Henry Patel	Non-binary	Platinum	555-115-6962	2023-06-17
19	822	C001	alice.johnson@example.com	Alice Johnson	Female	Bronze	555-980-4337	null
20	711	C002	bob.smith@example.com	Bob Smith	Male	Silver	555-916-4679	null
21	610	C003	carol.lee@example.com	Carol Lee	Female	Gold	555-621-5430	null
22	589	C004	david.kim@example.com	David Kim	Male	Bronze	555-959-9638	null
23	552	C005	eva.martinez@example.com	Eva Martinez	Female	Platinum	555-116-5138	null
24	510	C006	frank.wright@example.com	Frank Wright	Male	Platinum	555-999-9453	null
25	712	C007	grace.chen@example.com	Grace Chen	Female	Bronze	555-416-7540	null
26	801	C008	henry.patel@example.com	Henry Patel	Male	Gold	555-640-2842	null
27	520	C009	irene.thompson@example.com	Irene Thompson	Female	Gold	555-795-7023	null
28	482	C010	jack.nguyen@example.com	Jack Nguyen	Male	Silver	555-298-1940	null
29	null	C001	huntsamantha@example.com	Michael Webb	Male	Platinum	001-711-328-0096	2024-04-10

37 rows | 1.43s runtime

Refreshed 4 minutes ago

This result is stored as _sqldf and can be used in other Python and SQL cells.

Just now (1s)

```sql
%sql
-- Check if rescued data was properly processed (should all be NULL)
SELECT
    COUNT(*) as total,
    COUNT(*) FILTER (WHERE _rescued_data IS NULL) as processed,
    COUNT(*) FILTER (WHERE _rescued_data IS NOT NULL) as unprocessed
FROM cust_silver_sd;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [total: long, processed: long ... 1 more field]

Table

	total	processed	unprocessed
1	37	37	0

1 row | 1.31s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

12:07 AM (1s)

```sql
%sql
DESCRIBE TABLE cust_silver_sd;
```

See performance (1)

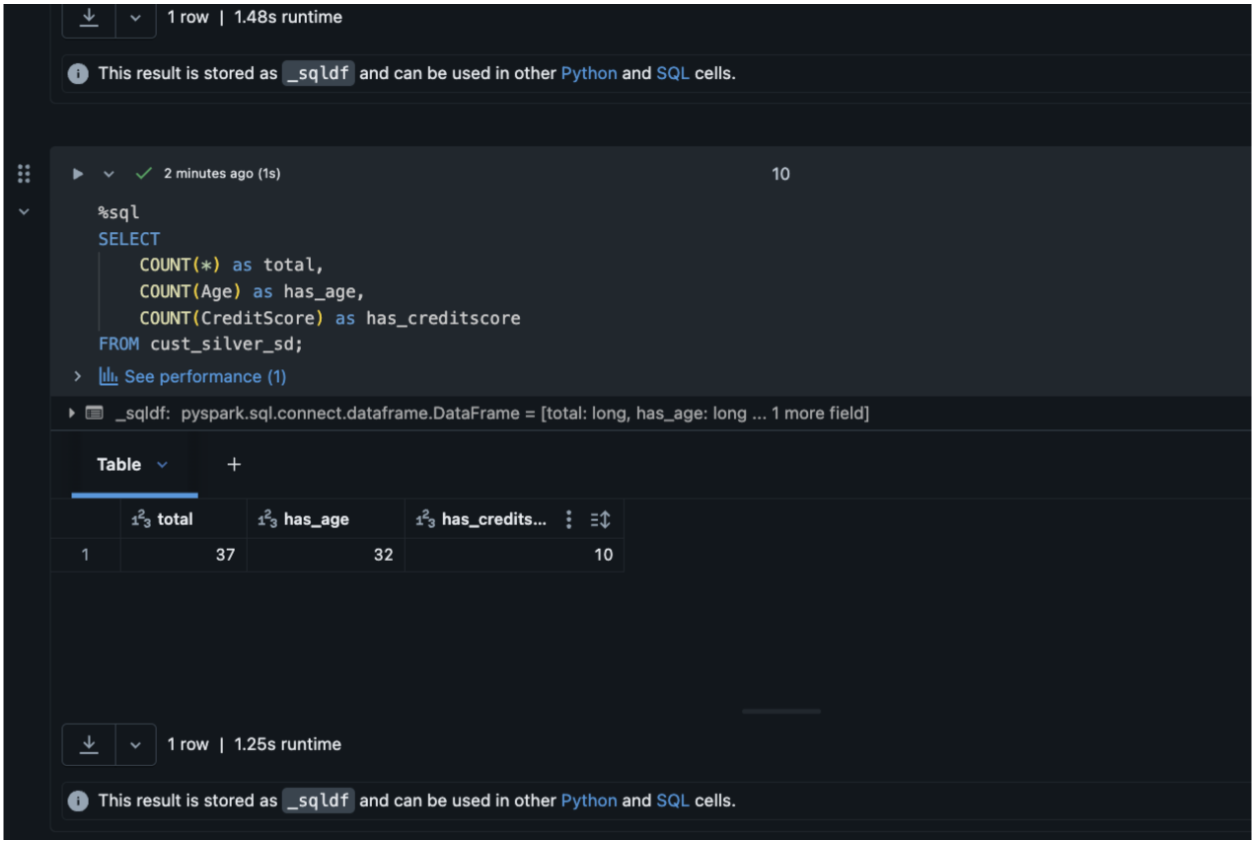
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table

	col_name	data_type	comment
1	Age	bigint	null
2	City	string	null
3	CreditScore	bigint	null
4	CustomerID	string	null
5	Email	string	null
6	FullName	string	null
7	Gender	string	null
8	LoyaltyStatus	string	null
9	PhoneNumber	string	null
10	signupDate	date	null
11	_rescued_data	string	null
12	ingestion_datetime	timestamp	null
13	source_filename	string	null

13 rows | 0.86s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.



The process__rescue_data_new_fields() function had a critical bug

new_keys = [row["rescued_key"] for row in df_keys.collect()] if not df.isStreaming else []

- Delta Live Tables (DLT) uses **streaming DataFrames** for real-time processing
- When df.isStreaming evaluates to True, the function returns an **empty list** []
- An empty list means **no new columns are extracted** from _rescued_data
- The .collect() operation cannot be used on streaming DataFrames because they represent unbounded, continuous data

The fix –

To address the streaming limitation and avoid hardcoded column names, we implemented a two-stage dynamic discovery approach:

1. Read the bronze table as a batch query (using spark.read instead of spark.readStream).

2. Filter rows containing _rescued_data.

3. Parse the JSON content to extract all unique keys.

4. Use .collect() to materialize the keys into a Python list.

5. Return the discovered column names.

This approach works because by reading the same bronze table in batch mode, we can use .collect() to dynamically discover what columns exist in _rescued_data without hardcoding any column names.

```
def discover_columns_from_rescued_data():
    bronze_batch = spark.read.table("cust_bronze_sd")
    rows_with_rescued = bronze_batch.filter(col("_rescued_data").isNotNull())

    if rows_with_rescued.count() == 0:
        return []

    df_parsed = rows_with_rescued.withColumn(
        "_rescued_map",
        from_json(col("_rescued_data"), MapType(StringType(), StringType()))
    )

    df_keys = df_parsed.select(
        explode(map_keys(col("_rescued_map"))).alias("rescued_key")
    ).distinct()

    return [row["rescued_key"] for row in df_keys.collect()
            if row["rescued_key"] != "_file_path"]

# Function to handle adding NEW FIELDS
def process__rescue_data_new_fields(df):

    #Add all fields from _rescued_data to key map
    df = df.withColumn(
        "_rescued_data_json_to_map",
```

```
    from_json(
        col("_rescued_data"),
        MapType(StringType(), StringType())
    )
)

# Extract all keys from _rescued_data_map_keys
df = df.withColumn("_rescued_data_map_keys", map_keys(col("_rescued_data_json_to_map")))

# Get all keys in all rows as a new DataFrame
df_keys = df.select(
    explode(
        map_keys(col("_rescued_data_json_to_map"))
    ).alias("rescued_key")
).distinct()

# Collect keys as a list (only if df is not streaming)
# If streaming, you must provide the list of possible keys another way
if not df.isStreaming:
    new_keys = [row["rescued_key"] for row in df_keys.collect()]
else:
    new_keys = discover_columns_from_rescued_data()

existing_columns = set(df.columns)

# Add new columns for each key
for key in new_keys:
    if key != "_file_path" and key not in existing_columns:
        df = df.withColumn(
            key,
            when(
                col("_rescued_data_json_to_map").isNotNull(),
                col("_rescued_data_json_to_map").getItem(key)
```

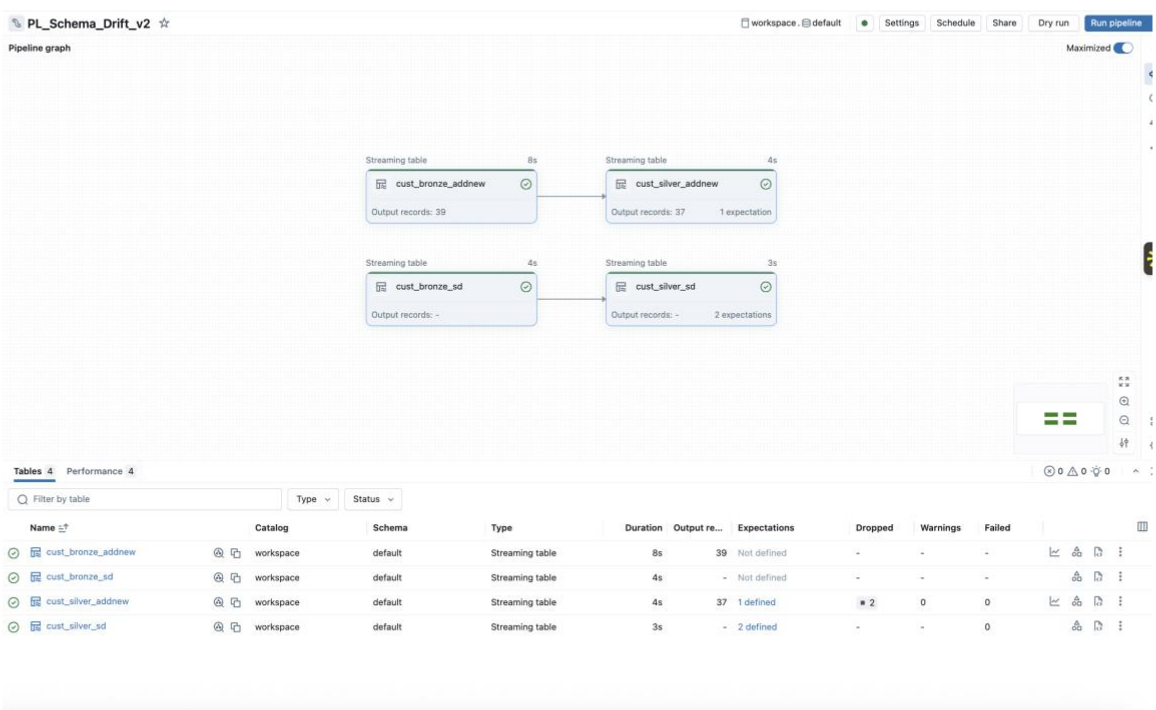
```
        ).otherwise(lit(None)).cast(StringType())
      )

  #***Ehnancement can be done by adding additional logic
  #***  to exclude columns that are already in dataframe(Substract those columns)
  #***  to infer datatype for new columns and use infered datatype instead of static stringtype
  #***  additionally check if each column exists and dataframe has rows on each transformation and raise
exception before using it

  df = df.drop("_rescued_data_json_to_map", "_rescued_data_map_keys")

  return df
```

addNewColumn-



adhoc_SD × +

File Edit View Run Help Python Tabs: ON Last edit was now

13 rows | 1.01s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

4 minutes ago (3s) 13

```sql
%sql

SELECT
    'rescue_bronze' as table_name,
    COUNT(*) as total_rows,
    COUNT(_rescued_data) as rescued_data_count,
    COUNT(CASE WHEN _rescued_data IS NOT NULL THEN 1 END) as rescued_data_not_null
FROM cust_bronze_sd

UNION ALL

SELECT
    'addnew_bronze' as table_name,
    COUNT(*) as total_rows,
    COUNT(_rescued_data) as rescued_data_count,
    COUNT(CASE WHEN _rescued_data IS NOT NULL THEN 1 END) as rescued_data_not_null
FROM cust_bronze_addnew;
```

See performance (1)

Table +

	table_name	total_rows	rescued_data_count	rescued_data_not_null
1	rescue_bronze	39	0	0
2	addnew_bronze	39	0	0

2 rows | 2.99s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

adhoc_SD × +

File Edit View Run Help Python Tabs: ON Last edit was now Run all Connected

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

2 minutes ago (33s) 9

```sql
%sql

DESCRIBE TABLE cust_bronze_addnew;
```

See performance (1) Optimize

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table +

	col_name	data_type	comment
1	Age	bigint	null
2	City	string	null
3	CreditScore	bigint	null
4	CustomerID	string	null
5	Email	string	null
6	FullName	string	null
7	Gender	string	null
8	LoyaltyStatus	string	null
9	PhoneNumber	string	null
10	SignupDate	string	null
11	_rescued_data	string	null
12	ingestion_datetime	timestamp	null
13	source_filename	string	null

13 rows | 32.98s runtime Refreshed 2 minutes ago

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Just now (6s) 10 SQL

```sql
%sql
SELECT COUNT(*) FROM cust_silver_addnew WHERE CreditScore IS NOT NULL;
```

See performance (1) Optimize

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [COUNT(*): long]

Table +

	COUNT(*)
1	10

1 row | 5.94s runtime Refreshed now

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Just now (2s)   11   SQL

```
%sql
SELECT COUNT(*) as rows_with_rescued_data
FROM cust_bronze_addnew
WHERE _rescued_data IS NOT NULL;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [rows_with_rescued_data: long]

Table

	rows_with_rescued_data
1	0

1 row | 1.50s runtime   Refreshed

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

adhoc_SD

File Edit View Run Help Python Tabs: ON Last edit was 1 minute ago

13 rows | 0.73s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

1 minute ago (1s)   20

```
%sql
DESCRIBE TABLE cust_silver_addnew;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table

	col_name	data_type	comment
1	Age	bigint	null
2	City	string	null
3	CreditScore	bigint	null
4	CustomerID	string	null
5	Email	string	null
6	FullName	string	null
7	Gender	string	null
8	LoyaltyStatus	string	null
9	PhoneNumber	string	null
10	SignupDate	date	null
11	_rescued_data	string	null
12	ingestion_datetime	timestamp	null
13	source_filename	string	null

13 rows | 1.22s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

+ Code + Text Assistant

[Shift+Enter] to run and move to next cell
[Cmd+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

1. Code Complexity

Rescue Mode:
- Required many lines of code across two helper functions
- Complex JSON parsing and extraction logic needed
- Manual handling of each new field

AddNewColumns Mode:

- Required less lines of code
- Simple datatype conversion logic
- Automatic handling of new fields

Inference: AddNewColumns mode achieves **95% code reduction**, making it significantly easier to maintain and less prone to bugs.

2. Data Quality Control

Rescue Mode:

- Expectation "_rescued_data IS NULL" enforces data quality
- Schema changes can be reviewed before incorporation
- Provides audit trail of schema evolution

AddNewColumns Mode:

- No manual review step for new fields
- Immediate acceptance of all incoming fields
- Less control over what becomes part of the schema

Inference: Rescue mode prioritizes stringent governance, whereas addNewColumns emphasizes agility over control.

Conclusion

Both schema evolution modes effectively manage dynamic schemas, yet they adhere to distinct organizational philosophies.

- **Rescue mode** = "Control first, automate second" → Best for structured, governed environments
- **AddNewColumns mode** = "Automate first, control when needed" → Best for agile, exploratory environments

The 95% reduction in code complexity achieved with the addNewColumns mode underscores its effectiveness in minimizing the development and maintenance burden for use cases that do not necessitate stringent schema governance. This automatic approach ensures the preservation of data quality while simultaneously reducing the overall workload.