Data quality in data engineering

As a data engineer, going into new projects often means a lot of data discovery and data profiling to meet the requirement of a specific use case. Having worked in the Databricks environment for the last couple of years, Databricks has introduced some excellent features for the discovery and profiling tasks, which makes this initial work a breeze.

One of the aspects of the data discovery stage is to ensure the quality of data coming in meets the specifications of the business requirements. This is very common and we often hear of businesses not having the confidence in part due to the quality and cleanliness of the data.

The Databricks profiling feature gives us visibility of how the data could look by providing, for example, a count of missing value in columns, data types, uniqueness of data, mean and median of numerical columns, and other notable data discoveries.

Getting this level of information is essential for data. However, ensuring data quality during the continuous flow of data ingestion is critical so that outputs are accurate and do not break in the event of incorrect ingestion.

While many solutions are available to ensure data quality, one solution often used in Python is Great Expectations. This short article does not intend to teach the length and breadth of Great Expectations but it does show how it could be of use in an automated way through embedding within a Databricks notebook.

A typical scenario in which I would use Great Expectations in a notebook is combining the results from Great Expectations to create a dataframe describing the effects of each constraint that the values are tested against.

```
import pandas as pd

pandas_df = df.select("*").toPandas()
ge_df = ge.dataset.PandasDataset(pandas_df)

ge_df.expect_column_values_to_be_of_type("tpep_pickup_datetime_parsed", 'datetime64')
ge_df.expect_column_values_to_not_be_null(column="passenger_count")
ge_df.expect_column_values_to_be_in_set(column='payment_type', value_set=[1,2,3,4])
ge_df.expect_column_values_to_be_in_set(column='passenger_count', value_set=[1,2,3,4,5,6], row_condition="tip_amount>5", condition_parser='pandas')
ge_df.expect_column_values_to_be_between(
    column="congestion_surcharge", min_value=0, max_value=1000
)
```

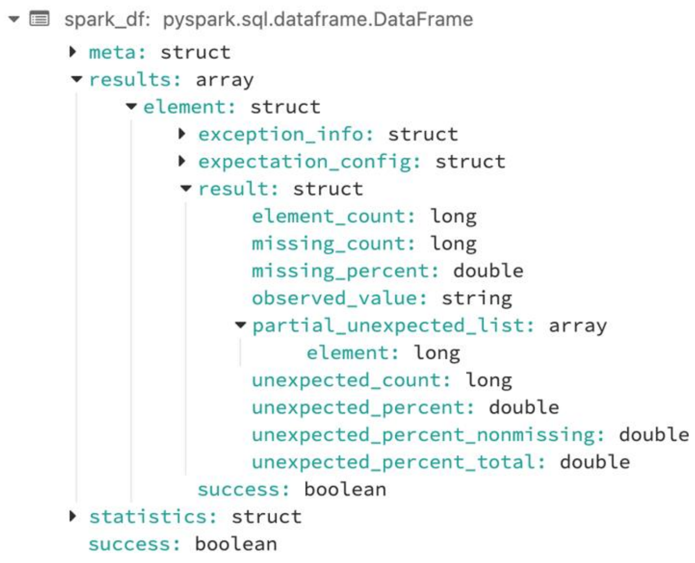
The above returns a JSON formatted result set which is then converted as a spark dataframe.

```
import json

results_validate = ge_df.validate().to_json_dict()
json_result = json.dumps(results_validate)

spark_df = spark.read.json(sc.parallelize([json_result]))
```

The resulting dataframe will have three columns, and our interest here is the 'results' columns which consist of arrays of the nested JSON structure shown below:



```
spark_df: pyspark.sql.dataframe.DataFrame
    meta: struct
    results: array
        element: struct
            exception_info: struct
            expectation_config: struct
            result: struct
                element_count: long
                missing_count: long
                missing_percent: double
                observed_value: string
                partial_unexpected_list: array
                    element: long
                unexpected_count: long
                unexpected_percent: double
                unexpected_percent_nonmissing: double
                unexpected_percent_total: double
            success: boolean
    statistics: struct
    success: boolean
```

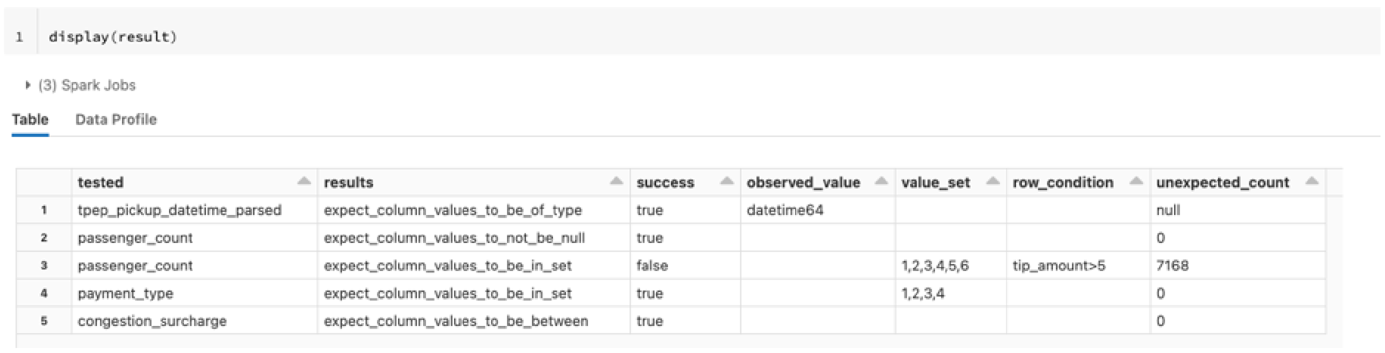
Selecting what we need to include as a curated dataframe, we can use Databricks built-in functions like arrays_zip and explode to tabularise the JSON data.

```
 from pyspark.sql import functions as F

result = (spark_df.withColumn("new", F.arrays_zip("results"))
              .withColumn("new", F.explode("new"))
              .withColumn('value_set', F.concat_ws(",", 
F.col("new.results.expectation_config.kwargs.value_set")))
              .select(F.col("new.results.expectation_config.kwargs.column").alias("tested"),

F.col("new.results.expectation_config.expectation_type").alias("results"),
                        F.col("new.results.success").alias("success"),
                        F.col("new.results.result.observed_value"),
                        F.col("value_set"),
                        F.col("new.results.expectation_config.kwargs.row_condition"),
                        F.col("new.results.result.unexpected_count")
                 ).na.fill('')
)
```

The expected result would look like this:



```
display(result)
```

(3) Spark Jobs

Table | Data Profile

	tested	results	success	observed_value	value_set	row_condition	unexpected_count
1	tpep_pickup_datetime_parsed	expect_column_values_to_be_of_type	true	datetime64			null
2	passenger_count	expect_column_values_to_not_be_null	true				0
3	passenger_count	expect_column_values_to_be_in_set	false		1,2,3,4,5,6	tip_amount>5	7168
4	payment_type	expect_column_values_to_be_in_set	true		1,2,3,4		0
5	congestion_surcharge	expect_column_values_to_be_between	true				0

The notebook can be used as part of a job workflow, to stop the workflow by raising an exception if the expected results are not met or can be used as a dataset in a visualisation or dashboard to show any exceptions to the user.

Newer versions of Great Expectations have incorporated more features that can be used with Apache Spark data frames.

With many organisations going through digital transformations and analytics capabilities that are getting more sophisticated, the risk and impact of inaccurate data can lead to biased and poor decision-making.

Added features in Delta Live Tables uses "expectations" to define quality constraints on the contents of a dataset. An expectation consists of a description, an invariant, and an action to take when a record fails to be consistent. The expectation can then be applied to queries using Python decorators or SQL constraint clauses.

These are fundamental requirements built into the Fujistu Data & AI Lakehouse Accelerator, which simplifies data ingestion with data accuracy, and rapidly delivers value from data.

If your business needs help with their data quality and accuracy, please contact a Fujitsu Data & AI specialist now.

Contact

Fujitsu Data & AI

+61 3 9924 3000