CDL Testing Automation Tool

Manual data validation:

Validation of data between two different objects takes a lot of time if done manually. There are various scenarios that needs to be executed in order to make sure that the object is created correctly. All the checks need to be done one by one and the evidences need to be captured manually.
Manual data validation has below issues associated with it:

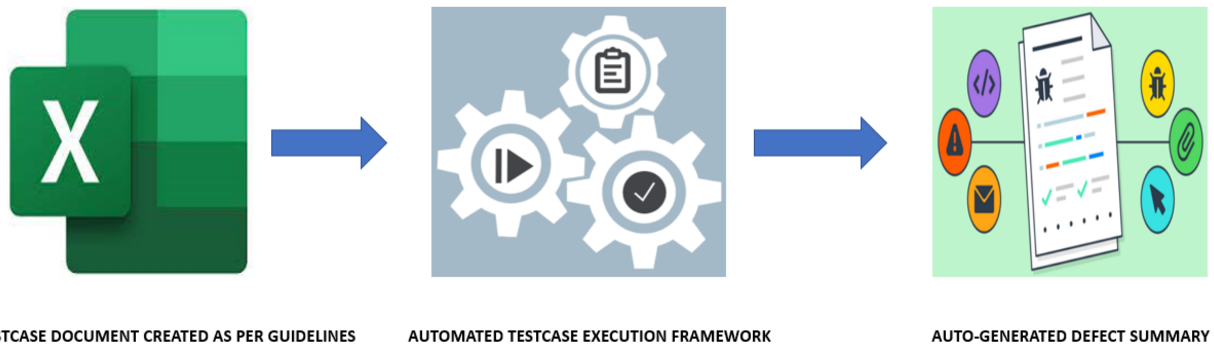
- Less execution speed
- Loss of accuracy due to human error
- Low testcase coverage
- As the capturing of evidences take time, RCA on issues gets delayed
- In case of regression, test execution time is not saved as it needs to be done manually
- Gauging the impact of each failed testcase on the application being tested is difficult

Proposed Solution:

We decided upon creating a framework programmed to ingest testcase documents, perform requested checks and validations and generate a defect summary with details of testcases failed, thereby costing us minimal time and effort.
The framework was developed on PySpark which runs on client cluster and basically leverages various PySpark and Pandas functions and libraries to pull requested data from source tables and tables developed in CDL environment, performs user requested sanity checks or validations and generates an output defect summary Excel file. The summary file highlights details of each failed testcase. It also enlists various numeric metrics derived from data validations performed, which can be used to understand the impact of defect caught by the testcase.
We just need to provide two input excel files, one of which will contain all the test cases that needs to be executed and the other for setting up a Spark session.

High Level Design:

Our framework basically leverages various PySpark functions and libraries to tackle the drawbacks we face in manual testing and increase the efficiency and speed of testing for our day-to-day implementation. It follows a simple procedure: ingest the testcase document and a Spark Session setup file in the framework, trigger its execution, then download and analyze the automatically generated defect summary.


TESTCASE DOCUMENT CREATED AS PER GUIDELINES
AUTOMATED TESTCASE EXECUTION FRAMEWORK
AUTO-GENERATED DEFECT SUMMARY

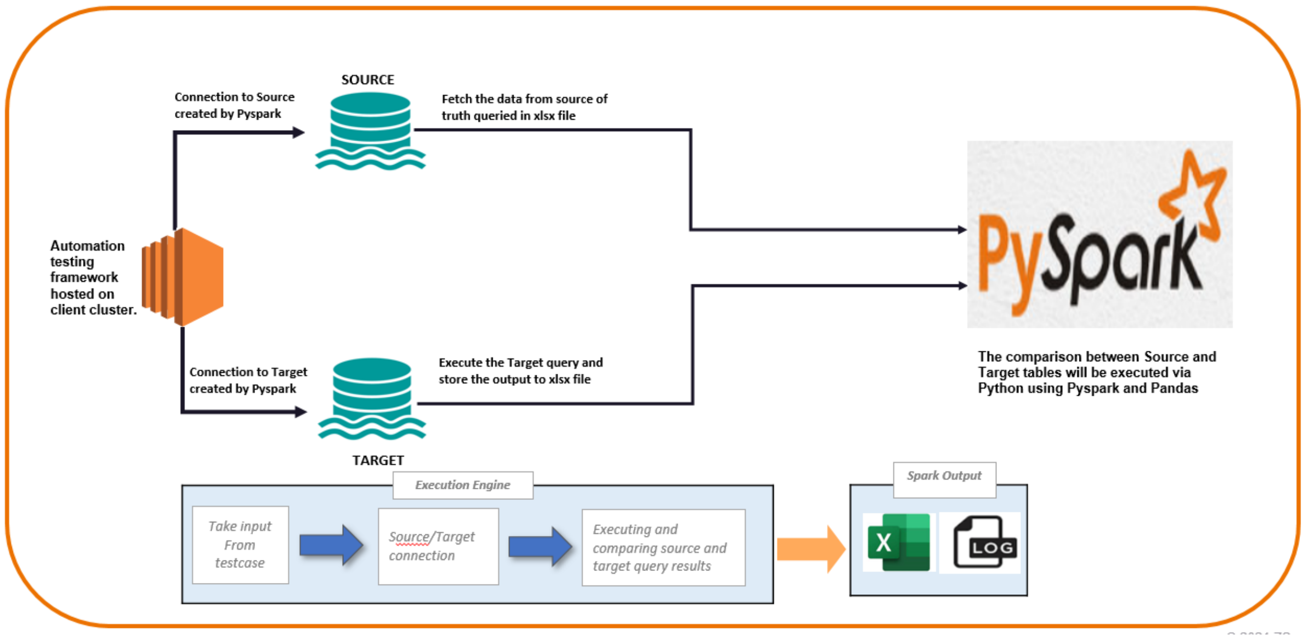
General Requirements:

- Access to requisite tables granted
- To run the automation framework in PySpark, we need to run the framework from a dedicated queue or Service account
- In order to ensure that the defect summary generation is seamless and does not face any crashes or failures, we have to be mindful about following the guidelines for creating each testcase following a framework friendly approach. Please refer to the attached document for further information on how to populate each field for a given testcase.
- All the required files to be present in the same location- the executables (.xlsx testcase document and .xlsx Spark Session setup file), .py automation code file

Compatible Checks:

	DQM CHECKS
Operations	**Description**
Null Check	The framework will check whether the specified column contains NULL values. The test will fail if the column contains NULL values
Uniqueness Check	The framework will check whether the records contains duplication on the columns specified. The test will fail if the column contains duplicate values
Referential Integrity Check	The framework will check whether the column follows all foreign key constraints.
Numeric Check	The framework will check whether the column contains numeric values or not. The test will fail if column contains nonnumeric values
Non-Numeric Check	The framework will check whether the column contains nonnumeric values or not. The test will fail if column contains numeric values
Valid Value check	The framework will check whether column contains valid value
Data Check	The framework will scan through the queried table and return the requested data set. User can pass any arbitrary query to this utility and pull required records from a table
	DATA VALIDATION CHECKS
Operations	**Description**
Data Comparison Check	The framework will check if data matches between source and target table. To perform this check, the grain should be provided
Count check	The framework will check whether the count matches for the source and target tables.

Workflow:


SOURCE
Connection to Source created by Pyspark
Fetch the data from source of truth queried in xlsx file
Automation testing framework hosted on client cluster.
PySpark
The comparison between Source and Target tables will be executed via Python using Pyspark and Pandas
Connection to Target created by Pyspark
Execute the Target query and store the output to xlsx file
TARGET
Execution Engine
Take input From testcase
Source/Target connection
Executing and comparing source and target query results
Spark Output
LOG

<u>Note: Please find the detailed manuals on end-to-end framework setup, creating testcase documents and Spark Session setup file in the "ATTACHMENTS" section.</u>

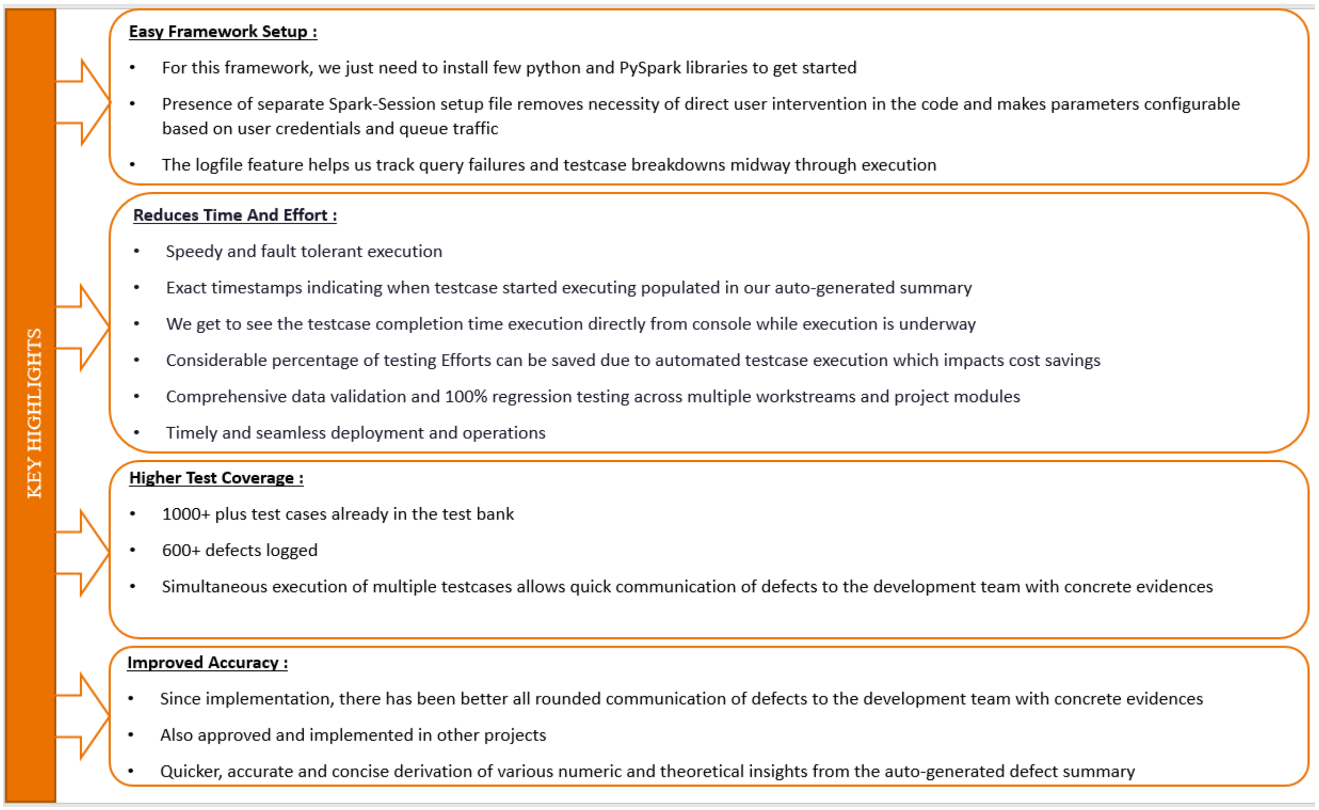
Defect Summary Analysis:

For each failed testcase, a worksheet populated with sample defective (with respect to the testcase failed) records is appended to the defect summary, ensuring quick communication of defects to the responsible team and faster resolution, retesting and closing of the defects.
The name of the worksheet appended for a failed testcase will be: ***[Test Case ID]_[Step Description]***
The primary worksheet comprising the testcases also have the factual columns populated with counts, match percentages etc.

Tools and Technologies Used:

- Python
- Spark

Benefits:



KEY HIGHLIGHTS

Easy Framework Setup :

- For this framework, we just need to install few python and PySpark libraries to get started
- Presence of separate Spark-Session setup file removes necessity of direct user intervention in the code and makes parameters configurable based on user credentials and queue traffic
- The logfile feature helps us track query failures and testcase breakdowns midway through execution

Reduces Time And Effort :

- Speedy and fault tolerant execution
- Exact timestamps indicating when testcase started executing populated in our auto-generated summary
- We get to see the testcase completion time execution directly from console while execution is underway
- Considerable percentage of testing Efforts can be saved due to automated testcase execution which impacts cost savings
- Comprehensive data validation and 100% regression testing across multiple workstreams and project modules
- Timely and seamless deployment and operations

Higher Test Coverage :

- 1000+ plus test cases already in the test bank
- 600+ defects logged
- Simultaneous execution of multiple testcases allows quick communication of defects to the development team with concrete evidences

Improved Accuracy :

- Since implementation, there has been better all rounded communication of defects to the development team with concrete evidences
- Also approved and implemented in other projects
- Quicker, accurate and concise derivation of various numeric and theoretical insights from the auto-generated defect summary

To set up the CDL Testing Automation Tool in your project, feel free to reach out to Shinjini Sanyal or Shubham Choudhury