**Data Quality Check Framework Documentation**

**Table of Contents**

**1. Introduction**

1.1 Purpose

1.2 Scope

1.3 Audience

**2. Dependencies**

2.1 PySpark Installation

2.2 Configuration Setup

2.3 Additional Dependencies

**3. Files Overview**

3.1 config.json

3.2 data\_quality\_checks.py

3.3 main.py

3.4 test\_data\_quality\_checks.py

**4. Configuration (config.json)**

4.1 Overview

4.2 Parameters

**5. Data Quality Checks (data\_quality\_checks.py)**

5.1 Introduction

5.2 Check Functions

5.2.1 Completeness Check

5.2.2 Validity Check

5.2.3 Uniqueness Check

5.2.4 Range Check

5.2.5 Pattern Check

5.2.6 Format Check

5.2.7 Null Check

5.2.8 Dependency Check

5.2.9 Spelling and Case Sensitivity Check

5.2.10 Categorical Value Consistency Check

5.2.11 Inter-Column Relationship Check

5.2.12 Exclusion of Sensitive Information

5.2.13 Regression Testing

**6. Main Functionality (main.py)**

6.1 Framework Initialization

6.2 Loading Configuration

6.3 Spark Session Setup

6.4 Running Data Quality Checks

6.5 Logging Results

**7. Unit Testing (test\_data\_quality\_checks.py)**

7.1 Overview

7.2 Test Case Functions

7.2.1 Test Completeness Check

7.2.2 Test Validity Check

7.2.3 Test Uniqueness Check

7.2.4 Test Range Check

7.2.5 Test Pattern Check

7.2.6 Test Format Check

7.2.7 Test Null Check

7.2.8 Test Completeness Check

7.2.9 Test Validity Check

7.2.10 Test Uniqueness Check

7.2.11 Test Range Check

7.2.12 Test Pattern Check

7.2.13 Test Pattern Check

**8. Logics and Implementations**

**9. Conclusion**

**10. All Checks Detail**

**1. Introduction**

**1.1 Purpose**

The purpose of this framework is to automate and standardize the process of performing data quality checks using PySpark.

**1.2 Scope**

The framework covers a variety of data quality checks, providing flexibility to users to select and customize based on their specific requirements.

**1.3 Audience**

This documentation is intended for data engineers, data scientists, and quality assurance professionals working with PySpark.

**2. Dependencies**

**2.1 PySpark Installation**

Ensure PySpark is installed in your environment using:

pip install pyspark

**2.2 Configuration Setup**

Configure the framework using config.json for parameters such as input data paths, output path, log file path, and other required settings.

**2.3 Additional Dependencies**

Install any additional libraries required for specific quality checks or reporting features.

**3. Files Overview**

**3.1 config.json**

This file holds all configuration data required for the framework. Users can modify this file to customize the framework behaviour.

**3.2 data\_quality\_checks.py**

Contains all data quality check functions. This file should be extended with additional checks as needed.

**3.3 main.py**

The main functionality file that orchestrates the data quality checks based on the configuration provided.

**3.4 test\_data\_quality\_checks.py**

Unit test file containing test cases for each data quality check function.

**4. Configuration (config.json)**

**4.1 Overview**

The config.json file centralizes configuration parameters, promoting easy customization without modifying code.

**4.2 Parameters**

input\_data\_path: Path to the input data file or directory.

output\_data\_path: Path of the output data file or directory.

file\_type: Type of file going to process.

log\_data\_path: Path to store log files.

(It can be extended…!)

**5. Data Quality Checks (data\_quality\_checks.py)**

**5.1 Introduction**

This module contains various data quality check functions that can be used to validate different aspects of the input data.

**5.2 Check Functions**

**5.2.1 Completeness Check**

def completeness\_check(data\_frame):

# Implementation details

pass

**5.2.2 Validity Check**

def validity\_check(data\_frame):

# Implementation details

pass

**5.2.3 Uniqueness Check**

def uniqueness\_check(data\_frame):

# Implementation details

pass

**5.2.4 Range Check**

def range\_check(data\_frame):

# Implementation details

pass

**5.2.5 Pattern Check**

def pattern\_check(data\_frame):

# Implementation details

pass

**5.2.6 Format Check**

def format\_check(data\_frame):

# Implementation details

pass

**5.2.7 Null Check**

def null\_check(data\_frame):

# Implementation details

Pass

**5.2.8 Dependency Check**

def dependency\_check(data\_frame):

# Implementation details

Pass

**5.2.9 Spelling and Case Sensitivity Check**

def spelling\_and\_case\_sensitivity\_check(data\_frame):

# Implementation details

Pass

**5.2.10 Categorical Value Consistency Check**

def categorical\_value\_consistency\_check(data\_frame):

# Implementation details

Pass

**5.2.11 Inter-Column Relationship Check**

def inter-column\_relationship\_check(data\_frame):

# Implementation details

Pass

**5.2.12 Exclusion of Sensitive Information**

def exclusion\_of\_sensitive\_information\_check(data\_frame):

# Implementation details

Pass

**5.2.13 Regression Testing Check**

def regression\_testing\_check(data\_frame):

# Implementation details

Pass

**6. Main Functionality (main.py)**

**6.1 Framework Initialization**

from pyspark.sql import SparkSession

import json

# Read configuration

with open("config.json", "r") as config\_file:

config = json.load(config\_file)

# Initialize Spark session

spark = SparkSession.builder.appName("DataQualityFramework").getOrCreate()

**6.2 Loading Configuration**

input\_data\_path = config["input\_data\_path"]

output\_data\_path = config["output\_data\_path "]

file\_type = config["file\_type"]

log\_data\_path = config["log\_data\_path"]

**6.3 Spark Session Setup**

# Load data into DataFrame

data\_frame = spark.read.csv(input\_data\_path, header=True)

**6.4 Running Data Quality Checks**

# Run data quality checks

# Run Main Functionality

**6.5 Logging Results**

# Log results

# Store all check function log data in log file

# Create separate log files

**7. Unit Testing (test\_data\_quality\_checks.py)**

**7.1 Overview**

Unit test file for testing each data quality check function in data\_quality\_checks.py.

**7.2 Test Case Functions**

**7.2.1 Test Completeness Check**

def test\_completeness\_check():

# Test logic

Pass

**7.2.2 Test Validity Check**

def test\_validity\_check():

# Test logic

Pass

**7.2.3 Test Uniqueness Check**

def test\_uniqueness\_check():

# Test logic

Pass

**7.2.4 Test Range Check**

def test\_range\_check():

# Test logic

Pass

**7.2.5 Test Pattern Check**

def test\_pattern\_check():

# Test logic

Pass

**7.2.6 Test Format Check**

def test\_format\_check():

# Test logic

Pass

**7.2.7 Test Null Check**

def test\_null\_check():

# Test logic

Pass

**7.2.8 Test Dependency Check**

def test\_dependency\_check():

# Test logic

Pass

**7.2.9 Test Spelling and Case Sensitivity Check**

def test\_spelling\_and\_case\_sensitivity\_check():

# Test logic

Pass

**7.2.10 Test Categorical Value Consistency Check**

def test\_categorical\_value\_consistency\_check():

# Test logic

Pass

**7.2.11 Test Inter-Column Relationship Check**

def test\_inter-column\_relationship\_check():

# Test logic

Pass

**7.2.12 Test Exclusion of Sensitive Information**

def test\_exclusion\_of\_sensitive\_information\_check():

# Test logic

Pass

**7.2.13 Test Regression Testing Check**

def test\_regression\_testing\_check():

#

Pass

**8. Logics and Implementations**

Detailed logics and implementation details for each data quality check function are available in the data\_quality\_checks.py file. Refer to this file for a deeper understanding of the checks performed and customization options.

**9. Conclusion**

This documentation provides a comprehensive guide to setting up and using the Data Quality Check Framework using PySpark. Users can refer to the specific sections for detailed information on configuration, check functions, main functionality, unit testing, and the underlying logics and implementations.

**All Checks Detail**

**1. Completeness Check**

**Meaning:**

The completeness check assesses the percentage of missing values in the dataset. It ensures that a specified proportion of the data is present, with a threshold indicating the minimum acceptable completeness.

**Purpose:**

To guarantee that a high percentage of the expected data is available, preventing issues arising from missing information.

**Usage:**

The function calculates the completeness percentage of the input data\_frame.

Compares it against the specified threshold.

Returns a result indicating whether the completeness meets the defined standard.

**Example:**

1. Suppose there are 1000 records, and 950 of them have non-null values in a specific column.
2. Completeness threshold is set to 0.95.
3. The result will be True since (950 / 1000) >= 0.95

**2. Validity Check**

**Meaning:**

The validity check ensures that the data adheres to predefined rules or standards. It assesses whether the data is in the expected format and follows specific criteria.

**Purpose:**

To identify and correct data that deviates from defined standards, preventing inconsistencies.

**Usage:**

1. The function checks the validity of the input data\_frame.
2. Compares it against the specified threshold.
3. Returns a result indicating whether the validity meets the defined standard.

**Example:**

1. Assume 'age' is the column to be checked for validity.
2. Suppose there are 1000 records, and 980 of them have 'age' between 18 and 100.
3. Validity threshold is set to 0.98.
4. The result will be True since (980 / 1000) >= 0.98.

**3. Uniqueness Check**

**Meaning:**

The uniqueness check ensures that there are no duplicate records in the dataset. It assesses the proportion of unique records against the total number of records.

**Purpose:**

To identify and handle duplicate entries that might lead to incorrect analyses or biased results.

**Usage:**

The function evaluates the uniqueness of the input data\_frame.

Compares it against the specified threshold.

Returns a result indicating whether the uniqueness meets the defined standard.

**Example:**

1. Suppose there are 1000 records, and after dropping duplicates, there are 950 unique records.
2. Uniqueness threshold is set to 0.95.
3. The result will be True since (950 / 1000) >= 0.95.

**4. Range Check**

**Meaning:**

The range check assesses whether numerical data falls within predefined ranges. It ensures that data values are within acceptable limits.

**Purpose:**

To identify and address data points that fall outside expected ranges, preventing outliers.

**Usage:**

1. The function performs range checks on numerical columns in the input data\_frame.
2. Compares it against the specified threshold.
3. Returns a result indicating whether the range meets the defined standard.

**Example:**

1. Assume 'value' is the numerical column to be checked for the range.
2. Suppose there are 1000 records, and 980 of them have 'value' between 10 and 100.
3. Range threshold is set to 0.98.
4. The result will be True since (980 / 1000) >= 0.98.

**5. Pattern Check**

**Meaning:**

The pattern check assesses whether data adheres to predefined patterns or regular expressions. It ensures that data follows specified formats.

**Purpose:**

To identify and correct deviations from expected patterns, maintaining data consistency.

**Usage:**

1. The function checks the patterns of data in the input data\_frame.
2. Compares it against the specified threshold.
3. Returns a result indicating whether the pattern meets the defined standard.

**Example:**

1. Assume 'email' is the column to be checked for a specific email pattern.
2. Suppose there are 1000 records, and 950 of them have 'email' in a valid email format.
3. Pattern threshold is set to 0.95.
4. The result will be True since (950 / 1000) >= 0.95.

**6. Format Check**

**Meaning:**

The format check assesses whether data adheres to specified formats, including date formats, string formats, etc. It ensures data consistency in terms of formatting.

**Purpose:**

To identify and correct data that does not conform to expected formats, maintaining uniformity.

**Usage:**

1. The function checks the formats of data in the input data\_frame.
2. Compares it against the specified threshold.
3. Returns a result indicating whether the format meets the defined standard.

**Example:**

1. Assume 'date' is the column to be checked for a specific date format.
2. Suppose there are 1000 records, and 980 of them have 'date' in the 'YYYY-MM-DD' format.
3. Format threshold is set to 0.98.
4. The result will be True since (980 / 1000) >= 0.98.

**7. Null Check**

**Meaning:**

The null check assesses the percentage of null or missing values in the dataset. It ensures that a specified proportion of the data is non-null.

**Purpose:**

To identify and handle excessive null values that might impact analyses and processing.

**Usage:**

1. The function calculates the percentage of null values in the input data\_frame.
2. Compares it against the specified threshold.
3. Returns a result indicating whether the null check meets the defined standard.

**Example:**

1. Suppose there are 1000 records, and 50 of them have null values in a specific column.
2. Null threshold is set to 0.95.
3. The result will be True since (50 / 1000) <= 0.95.

**8. Dependency Check**

**Meaning:**

The threshold value determines the minimum acceptable percentage of non-null values in the 'dependent\_column' based on the values in the 'independent\_column'.

**Purpose:**

Ensures that the values in 'dependent\_column' are adequately populated based on the values in 'independent\_column'.

**Usage:**

1. The function calculates the percentage dependencies between two columns, ensuring that one column's values have a sufficient impact on the presence of values in another column.
2. Compares it against the specified threshold.

**Example:**

1. Suppose 'dependent\_column' represents customer emails, and 'independent\_column' is a flag indicating whether the customer has opted to receive emails
2. The threshold may be set to 0.95.
3. Indicating that at least 95% of customers who opted to receive emails should have a valid email in the 'dependent\_column.'

**9. Spelling and Case Sensitivity Check**

**Meaning:**

The threshold value sets the minimum acceptable percentage of records with consistent spelling and case sensitivity in 'text\_column'.

**Purpose:**

Ensures that text data in 'text\_column' adheres to expected spelling and case sensitivity standards.

**Usage:**

Validate that free-form text data is consistently spelled and formatted according to predefined expectations.

**Example:**

1. If 'text\_column' contains country names.
2. The threshold might be set to 0.98.
3. Indicating that at least 98% of the records should have country names consistently spelled and formatted.

**10. Categorical Value Consistency Check**

**Meaning:**

The threshold value sets the minimum acceptable percentage of records with values in 'category\_column' consistent with predefined categories.

**Purpose:**

Ensures that categorical data in 'category\_column' is limited to a predefined set of values.

**Usage:**

Validate that categorical values are consistent with expected categories, reducing the risk of unexpected or undefined values.

**Example:**

If 'category\_column' represents product types, the threshold might be set to 0.99, indicating that at least 99% of records should have values matching predefined product categories.

**11. Inter-Column Relationship Check**

**Meaning:**

The threshold value sets the minimum acceptable percentage of records where the relationship between 'column\_A' and 'column\_B' holds true.

**Purpose:**

Validates that a predefined relationship between two columns is consistent across the dataset.

**Usage:**

Ensure that the combined values of 'column\_A' and 'column\_B' adhere to a specific relationship or business logic.

**Example:**

If 'column\_A' represents product prices and 'column\_B' represents tax rates, the threshold might be set to 0.95, indicating that at least 95% of records should have a combined value consistent with the expected total cost.

**12. Exclusion of Sensitive Information:**

**Meaning:**

The threshold value sets the maximum acceptable percentage of records with non-null values in 'sensitive\_column,' indicating the maximum allowed presence of sensitive information.

**Purpose:**

Ensures that sensitive information is appropriately excluded or anonymized in the dataset.

**Usage:**

Validate compliance with privacy regulations by monitoring the presence of sensitive information in specified columns.

**Example:**

If 'sensitive\_column' contains customer social security numbers, the threshold might be set to 0.02, indicating that no more than 2% of records should have non-null values in 'sensitive\_column.'

**13. Regression Testing Check**

**Meaning:**

The threshold value sets the minimum acceptable percentage of records where the relationship between 'column\_C' and 'column\_D' holds true, indicating successful regression testing.

**Purpose:**

Validates that changes to the dataset or processing pipeline do not introduce new issues or break existing relationships.

**Usage:**

Conduct regression tests on data transformations to ensure that existing relationships between columns remain valid.

**Example:**

If 'column\_C' represents quantities and 'column\_D' represents double the quantity, the threshold might be set to 0.98, indicating that at least 98% of records should satisfy the relationship 'column\_C' \* 2 == 'column\_D.'

These explanations cover the function definitions, threshold values, purposes, usages, and examples for each of the specified data quality checks. Adjusting the threshold values allows customization based on the specific requirements and data characteristics of your use case.