Data Quality Whistler

Release 0.0.1

Naresh Kumar

Oct 01, 2021

CONTENTS

CHAPTER

ONE

DQ ANALYZER

class dq_whistler.analyzer.**DataQualityAnalyzer**(*data: Union[pyspark.sql.dataframe.DataFrame, pandas.core.frame.DataFrame]*, *config: List[Dict[str, str]]*)

Analyzer class responsible for taking `JSON` dict and executing it on the columnar data

Parameters

- **data** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Dataframe/Series containing the data
- **config** (`List[Dict[str, str]]`) – The array of dicts containing config for each column

analyze() → str

Returns `JSON` string containing stats for multiple columns

Return type `str`

class dq_whistler.analyzer.**NpEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

CHAPTER

TWO

CONSTRAINTS

class dq_whistler.constraints.constraint.**Constraint**(*constraint: Dict[str, str]*, *column_name: str*)

Defines the base Constraint class

constraint_name()

Returns The name of the constraint

Return type str

execute_check(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Dict[str, str]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns

The dict containing the final output for one constraint Example Output:

```
{
    "name": "eq",
    "values", 5,
    "constraint_status": "failed/success",
    "invalid_count": 21,
    "invalid_values": [4, 6, 7, 1]
}
```

Return type dict[str, str]

get_column_name()

Returns The name of the column for which the Constraint instance was created

Return type str

abstract get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe containing failed cases for a constraint

Return type pyspark.sql.DataFrame

get_sample_invalid_values(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → List

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns A list containing the invalid values as per the given constraint

Return type list

CHAPTER

THREE

NUMERIC CONSTRAINTS

class dq_whistler.constraints.number_type.**Between**(*constraint: Dict[str, str]*, *column_name: str*)
Between constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"between",
        "values": [3, 4]
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is between [2, 8], then the dataframe will have rows where values are not in between [2, 8] (i.e only invalid cases)

Return type pyspark.sql.DataFrame|pandas.core.series.Series

class dq_whistler.constraints.number_type.**Equal**(*constraint: Dict[str, str]*, *column_name: str*)
Equal constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"eq",
        "values": 5
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with `invalid cases` as per the constraint, for ex: if constraint is `eq` to `5`, then the dataframe will have rows where values are `!= 5` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.number_type.**GreaterThan**(*constraint: Dict[str, str]*, *column_name: str*)

GreaterThan constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"gt",
        "values": 5
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `gt 5`, then the dataframe will have rows where values are `<= 5` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.number_type.**GreaterThanEqualTo**(*constraint: Dict[str, str]*, *column_name: str*)

GreaterThanEqualTo constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"gt_eq",
        "values": 5
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `gt_eq` to `5`, then the dataframe will have rows where values are `< 5` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.number_type.**IsIn**(*constraint: Dict[str, str]*, *column_name: str*)

IsIn constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"is_in",
        "values": [1, 2, 3]
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is is_in [1, 2, 3], then the dataframe will have rows where values are in [1, 2, 3] (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.number_type.**LessThan**(*constraint: Dict[str, str]*, *column_name: str*)

LessThan constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"lt",
        "values": 5
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is lt 5, then the dataframe will have rows where values are >= 5 (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.number_type.**LessThanEqualTo**(*constraint: Dict[str, str]*, *column_name: str*)

LessThanEqualTo constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"lt_eq",
        "values": 5
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is lt_eq to 5, then the dataframe will have rows where the values are > 5 (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.number_type.**NotBetween**(*constraint: Dict[str, str]*, *column_name: str*)

NotBetween constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_between",
        "values": [3, 5]
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is not_between [2,8], then the dataframe will have rows where values are in between [2, 8] (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.number_type.**NotEqual**(*constraint: Dict[str, str]*, *column_name: str*)

NotEqual constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"nt_eq",
        "values": 5
}
```

- **column_name** (*str*) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is nt_eq to 5, then the dataframe will have rows where values are = 5 (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.number_type.**NotIn**(*constraint: Dict[str, str]*, *column_name: str*)

NotIn constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_in",
        "values": [1, 2, 3]
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is "not_in" [1, 2, 3], then the dataframe will have rows where values are in [1, 2, 3] (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

CHAPTER

FOUR

STRING CONSTRAINTS

class dq_whistler.constraints.string_type.**Contains**(*constraint: Dict[str, str]*, *column_name: str*)
Contains constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"contains",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is contains "abc", then the dataframe will have rows where values does not contains "abc" (i.e only invalid cases)

Return type pyspark.sql.DataFrame|pandas.core.series.Series

class dq_whistler.constraints.string_type.**EndsWith**(*constraint: Dict[str, str]*, *column_name: str*)
EndsWith constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"ends_with",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `ends_with "abc"`, then the dataframe will have rows where values `does not ends with "abc"` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.string_type.**Equal**(*constraint: Dict[str, str]*, *column_name: str*)

Equal constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"eq",
        "values": "abc"
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `eq` to `"abc"`, then the dataframe will have rows where values are `!= "abc"` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.string_type.**IsIn**(*constraint: Dict[str, str]*, *column_name: str*)

IsIn constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"is_in",
        "values": ["abc", "xyz"]
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `is_in ["abc", "xyz"]`, then the dataframe will have rows where values `are not in ["abc", "xyz"]` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.string_type.**NotContains**(*constraint: Dict[str, str]*, *column_name: str*)

NotContains constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_contains",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is not_contains abc, then the dataframe will have rows where values contains "abc" (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.string_type.**NotEndsWith**(*constraint: Dict[str, str]*, *column_name: str*)

NotEndsWith constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_ends_with",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is not_ends_with "abc", then the dataframe will have rows where values ends with "abc" (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.string_type.**NotEqual**(*constraint: Dict[str, str]*, *column_name: str*)

NotEqual constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"nt_eq",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is nt_eq to "abc", then the dataframe will have rows where values are == "abc" (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.string_type.**NotIn**(*constraint: Dict[str, str]*, *column_name: str*)

NotIn constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_in",
        "values": ["abc", "xyz"]
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with invalid cases as per the constraint for ex: if constraint is not_in ["abc", "xyz"], then the dataframe will have rows where values are in ["abc", "xyz"] (i.e only invalid cases)

Return type pyspark.sql.DataFrame | pandas.core.series.Series

class dq_whistler.constraints.string_type.**NotStartsWith**(*constraint: Dict[str, str]*, *column_name: str*)

NotStartsWith constraint class that extends the base Constraint class

Parameters

- **constraint** (Dict[str, str]) – The dict representing a constraint config

```
{
        "name":"not_starts_with",
        "values": "abc"
}
```

- **column_name** (str) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (pyspark.sql.DataFrame | pandas.core.series.Series) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `not_starts_with "abc"`, then the dataframe will have rows where values `starts with "abc"` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.string_type.**Regex**(*constraint: Dict[str, str]*, *column_name: str*)

Regex constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"regex",
        "values": "^[A-Za-z]$"
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `regex ^[A-Za-z]$`, then the dataframe will have rows where values `does not` satisfies the regex `^[A-Za-z]$` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

class dq_whistler.constraints.string_type.**StartsWith**(*constraint: Dict[str, str]*, *column_name: str*)

StartsWith constraint class that extends the base Constraint class

Parameters

- **constraint** (`Dict[str, str]`) – The dict representing a constraint config

```
{
        "name":"starts_with",
        "values": "abc"
}
```

- **column_name** (`str`) – The name of the column for constraint check

get_failure_df(*data_frame: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*) → Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]

Parameters **data_frame** (`pyspark.sql.DataFrame | pandas.core.series.Series`) – Column data

Returns The dataframe with `invalid cases` as per the constraint for ex: if constraint is `starts_with "abc"`, then the dataframe will have rows where values `does not starts with "abc"` (i.e only invalid cases)

Return type `pyspark.sql.DataFrame | pandas.core.series.Series`

CHAPTER

FIVE

PROFILERS

class dq_whistler.profiler.column_profiler.**ColumnProfiler**(*column_data: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*, *config: Dict[str, Any]*)

Base class for column profiler

add_constraint(*constraint:* dq_whistler.constraints.constraint.Constraint)

Adds an instance of `Constraint` to the the parent list of constraints for this profiler

Parameters **constraint** (dq_whistler.constraints.constraint.Constraint) – An instance of `Constraint` class

get_column_config() → Dict[str, Any]

Returns The data quality config for the column

Return type `Dict[str, Any]`

get_column_info() → str

Returns

The column info for which the instance has been created Sample output:

```
str({
    "fields":[
        {
            "metadata":{},
            "name":"col_name",
            "nullable":True,
            "type":"string"
        }
    ],
    "type":"struct"
})
```

Return type `str`

get_constraints_config() → List[Dict[str, str]]

Returns The array containing the constraints for the column

Return type `List[Dict[str, str]]`

get_custom_constraint_check() → List[Dict[str, str]]

Returns

An array containing the output of each of the constraint for a column Sample Output:

```
[
    {
        "name": "eq",
        "values", 5,
        "constraint_status": "failed/success",
        "invalid_count": 21,
        "invalid_values": [4, 6, 7, 1]
    }...
]
```

Return type List[Dict[str, str]]

get_null_count() → int

Returns Count of null values in a column data

Return type int

get_quality_score() → float

Returns Overall quality score of a column

Return type float

get_topn() → Dict[str, Any]

Returns

Dict containing the top 10 values along with their counts Sample Output:

```
{
    "value1": count1,
    "value2": count2
}
```

Return type Dict[str, Any]

get_total_count() → int

Returns Count of total values in a column data

Return type int

get_unique_count() → int

Returns Count of unique values in a column data

Return type int

prepare_df_for_constraints() → None

Prepares a dataframe by doing pre validations

abstract run() → Dict[str, Any]

Returns The final stats of the column containing null count, total count, regex count, invalid rows, quality score etc.

Return type Dict[str, Any]

CHAPTER

SIX

NUMERIC PROFILER

class dq_whistler.profiler.number_profiler.**NumberProfiler**(*column_data: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*, *config: Dict[str, str]*)

Class for Numeric datatype profiler

get_max_value() → float

Returns Max value of the column data

Return type float

get_mean_value() → float

Returns Mean value of the column data

Return type float

get_min_value() → float

Returns Min value of the column data

Return type float

get_stddev_value() → float

Returns Standard deviation value of the column value

Return type float

run() → Dict[str, Any]

Returns

The final dict with all the metrics of a numeric column Example Output:

```
{
        "total_count": 100,
        "null_count": 50,
        "unique_count": 20,
        "topn_values": {"1": 24, "2": 25},
        "min": 2.0,
        "max": 30.0,
```

(continues on next page)

(continued from previous page)

```
        "mean": 18.0,
        "stddev": 5.0,
        "quality_score": 0,
        "constraints": [
                {
                        "name": "eq",
                        "values", 5,
                        "constraint_status": "failed/success",
                        "invalid_count": 21,
                        "invalid_values": [4, 6, 7, 1]
                }
        ]
}
```

Return type `Dict[str, Any]`

CHAPTER

SEVEN

STRING PROFILER

class dq_whistler.profiler.string_profiler.**StringProfiler**(*column_data: Union[pyspark.sql.dataframe.DataFrame, pandas.core.series.Series]*, *config: Dict[str, str]*)

Class for String datatype profiler

run() → Dict[str, Any]

Returns

The final dict with all the metrics of a string column Example Output:

```
{
        "total_count": 100,
        "null_count": 50,
        "unique_count": 20,
        "topn_values": {"abc": 24, "xyz": 25},
        "quality_score": 0,
        "constraints": [
                {
                        "name": "eq",
                        "values", "abc",
                        "constraint_status": "failed/success",
                        "invalid_count": 21,
                        "invalid_values": ["xy", "ab", "abcd"]
                }
        ]
}
```

Return type Dict[str, Any]

PYTHON MODULE INDEX

d

INDEX

I

L

M

N

P

R

S