iDataValidator Functions

[validatequestion 2](#_Toc174498331)

[validatecolumns 4](#_Toc174498332)

[checkinvalid 5](#_Toc174498333)

[checkblanks 6](#_Toc174498334)

[checknoblanks 6](#_Toc174498335)

[checkinvalid\_logic 7](#_Toc174498336)

[checkvalid\_logic 7](#_Toc174498337)

[checkmasking 8](#_Toc174498338)

[checkquestion\_mask 9](#_Toc174498339)

[checkrank 10](#_Toc174498340)

[checksum 11](#_Toc174498341)

[checksum100 12](#_Toc174498342)

[checkterm 13](#_Toc174498343)

# Structure

iDataValidator/

└── data/

├── generate/

├── logs/

├── validator\_functions/

├── utils/

├── survey\_model/

├── validators/

│ validators/

│ ├── \_\_init\_\_.py

│ ├── custom\_functions.py

│ ├── validator.py

│ ├── validatorx1.py

│ ├── validatorx2.py

**├── run.py**

**├── config.py**

# Validatesingle

**Signature: (questionid, datacols, datarow, valid\_values, optional\_cols, exclude\_cols, allowblanks, condition)**

Perform single-value validation on specified columns in a row of survey data. The function checks if the values

in `datacols` are valid according to the provided `valid\_values` list, optionally allows blanks, and excludes certain columns from validation if needed.

**Parameters:**

* questionid (str): The ID of the question being checked (used for logging purposes).
* datacols (list): A list of column names to be checked for valid single values.
* datarow (pd.Series): A single row from a pandas DataFrame, where the single-value validation is performed.
* valid\_values (list): A list of valid values that the data in `datacols` should match.
* optional\_cols (list, optional): A list of columns where blank values are allowed. Defaults to an empty list.
* exclude\_cols (list, optional): A list of columns to exclude from the single-value validation. These columns are removed from `datacols` before the check is performed. Defaults to an empty list.
* allowblanks (bool, optional): Whether blank values are allowed in the columns. Defaults to False.
* condition (bool, optional): A flag to enable or disable the check. If `True`, the single-value validation is performed. If `False`, the function only performs a blank check. Defaults to `True`.

*Example Usage:*

*# datarow is row of data*

*# Columns to validate*

*datacols = ['col1', 'col2', 'col3']*

*# Perform single-value validation allowing values from 1 to 5*

*validatesingle('Q1', datacols, datarow, valid\_values=[1, 2, 3, 4, 5])*

# checksum100

Signature: (questionid, datacols, datarow, exclude\_cols, condition, range\_param)

Perform a sum check on specified columns in a row of survey data to 100.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names whose values will be summed.

datarow (pd.Series): A single row from a pandas DataFrame, where the sum check is performed.

exclude\_cols (list, optional): A list of column names to exclude from the sum calculation. These columns

are removed from `datacols` before the check is performed. Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the sum check is performed.

If `False`, the function does nothing. Defaults to `True`.

Example Usage:

#

# Columns to check for sum

datacols = ['col1', 'col2', 'col3', 'col4']

# Perform the sum check to ensure the sum equals 100

checksum('Q1', datacols, datarow)

Module: /Users/prasad/github/iDataValidator/validator\_functions/checksum.py

Function: checksum

Signature: (questionid, datacols, datarow, exclude\_cols, condition, sum\_condition)

Docstring:

Perform a sum check on specified columns in a row of survey data. The function checks if the sum of the values in

`datacols` meets the condition specified in `sum\_condition` (e.g., '=100', '<50', '>20', 'range(90,110)').

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names whose values will be summed.

datarow (pd.Series): A single row from a pandas DataFrame, where the sum check is performed.

exclude\_cols (list, optional): A list of column names to exclude from the sum calculation. These columns

are removed from `datacols` before the check is performed. Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the sum check is performed.

If `False`, the function does nothing. Defaults to `True`.

sum\_condition (str, optional): A string representing the condition to check the sum against. This can be:

- '=X' to check if the sum equals X

- '<X' to check if the sum is less than X

- '>X' to check if the sum is greater than X

- 'range(X,Y)' to check if the sum is within the range X to Y (inclusive)

Defaults to '=100'.

Example Usage:

# Example datarow from a DataFrame

datarow = pd.Series({

'record': '001',

'col1': 50,

'col2': 30,

'col3': 20,

'col4': None

})

# Columns to check for sum

datacols = ['col1', 'col2', 'col3', 'col4']

# Perform the sum check to ensure the sum equals 100

checksum('Q1', datacols, datarow, sum\_condition='=100')

# Perform the sum check to ensure the sum is within the range 90 to 110

checksum('Q1', datacols, datarow, sum\_condition='range(90,110)')

Module: /Users/prasad/github/iDataValidator/validator\_functions/checkblanks.py

Function: checkblanks

Signature: (questionid, datacols, datarow, exclude\_cols, condition)

Docstring:

Performs a blank check on specified columns in a row of data. If any column that is expected to be blank contains a value,

an error is logged. Certain columns can be excluded from this check based on the provided list.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names that need to be checked for blank values in the data row.

datarow (pd.Series): A single row from a pandas DataFrame, represented as a pandas Series, that is being checked for blanks.

exclude\_cols (list, optional): A list of columns that should be excluded from the blank check. These columns will be removed from `datacols` before the check is performed. Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the blank check is performed. If `False`, the function skips the check. Defaults to `True`.

Returns:

None: The function logs an error through the `adderror` function if any non-blank values are found in columns where blanks are expected. No value is returned.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns to check for blank values

datacols = ['col1', 'col2', 'col3']

# Exclude 'col3' from the blank check

exclude\_cols = ['col3']

# Perform the blank check

checkblanks('Q1', datacols, datarow, exclude\_cols)

Module: /Users/prasad/github/iDataValidator/validator\_functions/checkcondition.py

Function: checkcondition

Signature: (value, condition)

Docstring:

Checks if a given value satisfies a condition based on various operators such as '=', 'in', 'range', '>', '<', etc.

Parameters:

value (int/float): The value to be checked against the condition.

condition (str): A string representing the condition to check against. Supported conditions include:

- '=': Equals (e.g., '=5' checks if value equals 5)

- 'in': In a list of values (e.g., 'in[1,2,3]' checks if value is 1, 2, or 3)

- 'range': Within a range (e.g., 'range(1,10)' checks if value is between 1 and 10, inclusive)

- '>': Greater than (e.g., '>5' checks if value is greater than 5)

- '<': Less than (e.g., '<10' checks if value is less than 10)

- '>=': Greater than or equal to (e.g., '>=5' checks if value is greater than or equal to 5)

- '<=': Less than or equal to (e.g., '<=10' checks if value is less than or equal to 10)

Returns:

bool: True if the value satisfies the condition, False otherwise.

Module: /Users/prasad/github/iDataValidator/validator\_functions/\_\_init\_\_.py

Module: /Users/prasad/github/iDataValidator/validator\_functions/checknonblanks.py

Function: checknonblanks

Signature: (questionid, datacols, datarow, exclude\_cols, condition)

Docstring:

Perform a non-blank check on specified columns in a row of survey data. The function checks if the specified columns

in `datacols` are non-null (not blank). If any column is found to be null, an error is logged.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names to be checked for non-blank (non-null) values.

datarow (pd.Series): A single row from a pandas DataFrame, where the non-blank check is performed.

exclude\_cols (list, optional): A list of column names to exclude from the non-blank check. These columns

are removed from `datacols` before the check is performed. Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the non-blank check is performed.

If `False`, the function does nothing. Defaults to `True`.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns to check for non-blank values

datacols = ['col1', 'col2', 'col3']

# Exclude 'col3' from the non-blank check

exclude\_cols = ['col3']

# Perform the non-blank check

checknonblanks('Q1', datacols, datarow, exclude\_cols)

Module: /Users/prasad/github/iDataValidator/validator\_functions/vwcheck.py

Function: vwcheck

Signature: (questionid, datarow, VW\_cols, condition)

Docstring:

Validates the VW (Value for Money) check based on the specified columns. The check ensures that the values

in the specified columns are in increasing order, with each successive value being greater than the previous.

Parameters:

questionid (str): The ID of the question being validated (used for logging purposes).

datarow (pd.Series): A single row from a pandas DataFrame, where the VW check is performed.

VW\_cols (list): A list of column names representing the order of "Too cheap" to "Too expensive".

The function checks that these values are in increasing order.

condition (bool, optional): A flag to enable or disable the check. If `True`, the VW check is performed.

Defaults to `True`.

Example Usage:

# Example datarow from a DataFrame

datarow = pd.Series({

'record': '001',

'too\_cheap': 50,

'affordable': 100,

'expensive': 150,

'too\_expensive': 200

})

# Columns to validate for VW check

VW\_cols = ['too\_cheap', 'affordable', 'expensive', 'too\_expensive']

# Perform the VW check

vwcheck('Q1', datarow, VW\_cols)

Module: /Users/prasad/github/iDataValidator/validator\_functions/validatemulti.py

Function: validatemulti

Signature: (questionid, datacols, datarow, valid\_values, optional\_cols, exclude\_cols, exclusive\_cols, at\_least, at\_most, allowblanks, required, condition)

Docstring:

Perform a multi-select validation on a set of columns in a row of survey data. This function checks if the selected

values are valid, enforces exclusive selection rules, and ensures that at least or at most a certain number of options

are selected. It also handles optional columns and blanks.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names to be checked for valid multi-select values.

datarow (pd.Series): A single row from a pandas DataFrame, where the multi-select validation is performed.

valid\_values (list, optional): A list of valid values for the multi-select columns. Defaults to [0,1].

optional\_cols (list, optional): Columns where blank values are allowed. Defaults to an empty list.

exclude\_cols (list, optional): Columns to exclude from the multi-select validation. These columns are removed

from `datacols` before the check is performed. Defaults to an empty list.

exclusive\_cols (list, optional): Columns that should behave exclusively (i.e., only one can be selected).

Defaults to an empty list.

at\_least (int, optional): The minimum number of selections required. Defaults to 1.

at\_most (int, optional): The maximum number of selections allowed. If set to -1, no limit is imposed. Defaults to -1.

allowblanks (bool, optional): Whether blank values are allowed in the multi-select columns. Defaults to False.

required (int, optional): A flag to enforce the requirement of at least one selection. Defaults to 1 (True).

condition (bool, optional): A flag to enable or disable the check. If `True`, the multi-select validation is

performed. If `False`, only a blank check is performed. Defaults to `True`.

Example Usage:

# Example datarow from a DataFrame

datarow = pd.Series({

'record': '001',

'col1': 1,

'col2': 0,

'col3': 1,

'col4': None

})

# Columns to validate for multi-select

datacols = ['col1', 'col2', 'col3', 'col4']

# Perform the multi-select validation with at least 1 and at most 2 selections

validatemulti('Q1', datacols, datarow, at\_least=1, at\_most=2, exclusive\_cols=['col1'], optional\_cols=['col4'])

Module: /Users/prasad/github/iDataValidator/validator\_functions/validatetext.py

Function: validatetext

Signature: (questionid, datacols, datarow, optional\_cols, exclude\_cols, exclusive\_cols, at\_least, at\_most, txt\_mnlen, txt\_mxlen, allowblanks, required, condition)

Docstring: None

Module: /Users/prasad/github/iDataValidator/validator\_functions/validatenumeric.py

Function: validatenumeric

Signature: (questionid, datacols, datarow, optional\_cols, exclude\_cols, exclusive\_cols, at\_least, at\_most, allowblanks, required, condition, range\_param)

Docstring:

Perform numeric validation on specified columns in a row of survey data. The function checks whether the values

in `datacols` fall within the range specified by `range\_param`, enforces exclusive selection rules, and ensures

that at least or at most a certain number of valid values are selected. It also handles optional columns and blanks.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (List[str]): A list of column names to be checked for valid numeric values.

datarow (pd.Series): A single row from a pandas DataFrame, where the numeric validation is performed.

optional\_cols (List[str], optional): A list of columns where blank values are allowed. Defaults to an empty list.

exclude\_cols (List[str], optional): A list of columns to exclude from the numeric validation. These columns

are removed from `datacols` before the check is performed. Defaults to an empty list.

exclusive\_cols (List[str], optional): Columns that should behave exclusively (i.e., only one can be selected).

Defaults to an empty list.

at\_least (int, optional): The minimum number of valid values required. Defaults to 1.

at\_most (int, optional): The maximum number of valid values allowed. If set to -1, no limit is imposed. Defaults to -1.

allowblanks (bool, optional): Whether blank values are allowed in the numeric columns. Defaults to False.

required (int, optional): A flag to enforce the requirement of at least one valid value. Defaults to 1 (True).

condition (bool, optional): A flag to enable or disable the check. If `True`, the numeric validation is

performed. If `False`, the function performs a blank check. Defaults to `True`.

range\_param (RangeTuple, optional): A tuple specifying the valid range for the numeric values (min, max).

Defaults to (0, 100).

Example Usage:

# Example datarow from a DataFrame

datarow = pd.Series({

'record': '001',

'col1': 75,

'col2': 105,

'col3': 50,

'col4': None

})

# Columns to validate for numeric values

datacols = ['col1', 'col2', 'col3', 'col4']

# Perform numeric validation with a valid range of 0 to 100

validatenumeric('Q1', datacols, datarow, range\_param=(0, 100))

Module: /Users/prasad/github/iDataValidator/validator\_functions/checkrank.py

Function: checkrank

Signature: (questionid, datacols, datarow, min\_rank\_value, max\_rank\_value, exclude\_cols, condition)

Docstring:

Perform a rank check on specified columns in a row of survey data. This function ensures that the ranks in

`datacols` are unique and within a specified range (from `min\_rank\_value` to `max\_rank\_value`).

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names to check for rank values.

datarow (pd.Series): A single row from a pandas DataFrame, where the rank check is performed.

min\_rank\_value (int, optional): The minimum allowed rank value. Defaults to 1.

max\_rank\_value (int, optional): The maximum allowed rank value. If not provided, it defaults to the number

of columns being checked.

exclude\_cols (list, optional): A list of column names to exclude from the rank check. These columns

are removed from `datacols` before the check is performed. Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the rank check is performed.

If `False`, the function does nothing. Defaults to `True`.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns to check for rank values

datacols = ['col1', 'col2', 'col3', 'col4']

# Perform the rank check

checkrank('Q1', datacols, datarow, min\_rank\_value=1, max\_rank\_value=4)

Module: /Users/prasad/github/iDataValidator/validator\_functions/backcheck\_single.py

Function: backcheck\_single

Signature: (questionid, qcol, datarow, cols\_to\_check, maskcondition, condition)

Docstring:

Perform a backcheck on a single question in a survey data row. This function verifies the value of a specified column

(`qcol`) and checks whether the corresponding value in `cols\_to\_check` satisfies a given condition.

Parameters:

questionid (str): The ID of the question being checked.

qcol (str): The column name from `datarow` to be validated (represents a question in the survey).

datarow (pd.Series): The row of survey data being processed. Each row is treated as a pandas Series.

cols\_to\_check (List[str]): A list of column names to backcheck against. The column to check is determined by the

value of `question\_val` in the `qcol` column.

maskcondition (Optional[str]): A condition string that the target value must satisfy (e.g., '>', '<', '==').

If not provided, no condition check will be applied.

condition (bool): A flag to enable or disable the check. If `True`, the function will run; otherwise, it will skip.

Raises:

None: Errors are logged through the `adderror` function, rather than raised directly.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns in 'cols\_to\_check' are ['col1', 'col2'] and the question column is 'qcol'.

# Check if the value in the corresponding column (cols\_to\_check[source - 1]) satisfies the condition '> 10'.

backcheck\_single('Q1', 'qcol', datarow, ['col1', 'col2'], maskcondition='> 10')

Module: /Users/prasad/github/iDataValidator/validator\_functions/checkexclusive.py

Function: checkexclusive

Signature: (questionid, datacols, datarow, exclusive\_cols, iszerovalid, condition, oneway)

Docstring:

Performs an exclusivity check on a set of columns in a row of survey data. It ensures that only one "exclusive" option

is selected, and optionally, that no other non-exclusive options are selected alongside it. This function can also

check for cases where no selection is made at all if the `oneway` flag is enabled.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datacols (list): A list of column names to check for selections. These represent the set of columns

where responses can be recorded.

datarow (pd.Series): A single row of data from a pandas DataFrame, where the exclusivity check is performed.

exclusive\_cols (list, optional): A list of column names that represent "exclusive" options.

These columns should not be selected along with any other columns.

Defaults to an empty list.

iszerovalid (bool, optional): Determines whether a value of 0 should be considered a valid selection.

If `True`, 0 is treated as a valid selection. If `False`, 0 is ignored.

Defaults to `True`.

condition (bool, optional): A flag to enable or disable the check. If `True`, the exclusivity check is

performed. If `False`, the function does nothing. Defaults to `True`.

oneway (bool, optional): If `True`, an additional check is performed to ensure that at least one selection

(exclusive or non-exclusive) is made. Defaults to `False`.

Returns:

None: The function logs errors through the `adderror` function if the exclusivity rules are violated.

No value is returned.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns representing possible selections, with 'col1' as exclusive

datacols = ['col1', 'col2', 'col3', 'col4']

exclusive\_cols = ['col1']

# Perform the exclusive check with 'col1' as the exclusive column

checkexclusive('Q1', datacols, datarow, exclusive\_cols, iszerovalid=False, condition=True)

Module: /Users/prasad/github/iDataValidator/validator\_functions/checkmasking.py

Function: checkmasking

Signature: (questionid, datarow, question\_cols, maskcond\_cols, maskcondition, always\_showcols, condition)

Docstring:

Perform a masking check on specified columns in a row of survey data. The function checks whether certain

columns (from `question\_cols`) should be shown or hidden based on conditions applied to corresponding columns

in `maskcond\_cols`. If any condition is violated, an error is logged.

Parameters:

questionid (str): The ID of the question being checked (used for logging purposes).

datarow (pd.Series): A single row from a pandas DataFrame, where the masking check is performed.

question\_cols (List[str]): A list of column names representing questions to be checked for masking.

maskcond\_cols (List[str]): A list of corresponding column names that contain the conditions to check for each question.

maskcondition (str, optional): The condition to check on `maskcond\_cols` (e.g., '=1', 'in[1, 2]', '>0').

Defaults to '=1'.

always\_showcols (List[str], optional): A list of column names that should always be shown, regardless of

the masking condition. These columns are excluded from the regular masking check.

Defaults to an empty list.

condition (bool, optional): A flag to enable or disable the check. If `True`, the masking check is performed.

If `False`, the function does nothing. Defaults to `True`.

Returns:

None: The function logs errors through the `adderror` function if the masking rules are violated.

No value is returned.

Example Usage:

# Assuming 'datarow' is a row from a pandas DataFrame:

# Columns to check for masking

question\_cols = ['A', 'B']

maskcond\_cols = ['ConditionCol1', 'ConditionCol2']

# Columns that should always be shown

always\_showcols = ['C', 'D']

# Perform the masking check

checkmasking('Q1', datarow, question\_cols, maskcond\_cols, '=1', always\_showcols) value")