# fill\_missing\_values

**Description**

* **Purpose**: To fill or remove missing values in a specified column of the DataFrame.
* **Functionality**: Checks if a DataFrame is loaded, verifies the column's existence, and then applies the chosen method ('remove', 'mode', 'mean', 'median', 'specific') to handle missing values.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if an invalid method or specific value is provided.

**Arguments**

1. **column\_name (str)**: The name of the column in the DataFrame to process.
2. **method (str)**: Method for handling missing values. Options include 'remove', 'mode', 'mean', 'median', 'specific'.
3. **specific\_value (numeric, optional)**: A specific numeric value for filling missing values, used only if method='specific'.

**UI Requirements**

1. **Column Selector**: An interface element like a dropdown to select the column to be processed.
2. **Method Selector**: A dropdown or radio buttons to choose the method for handling missing values.
3. **Specific Value Input**: An input field for the specific value, enabled only when the 'specific' method is selected.
4. **Error Display**: A section to display errors in a user-friendly manner (e.g., 'Dataframe not loaded', 'Column name not found', 'Invalid specific value').
5. **Execute Button**: A button to run the function after selecting the DataFrame, column, and method.
6. **Output Display**: A section to show the modified DataFrame or a message if no missing values were found.

# remove\_outliers

**Description**

* **Purpose**: To remove or modify outliers in all numeric columns of a DataFrame based on standard deviation.
* **Functionality**: Iterates over each numeric column, calculates the mean and standard deviation, and replaces values outside the specified standard deviation range with NaN.
* **Error Handling**: Raises an error if no DataFrame is loaded.

**Arguments**

1. **sd (float)**: The number of standard deviations used to define an outlier. Defaults to 3.0.

**UI Requirements**

1. **Standard Deviation Input**: An input field for the user to specify the number of standard deviations for identifying outliers.
2. **Error Display**: An area to display errors, like 'Dataframe not loaded', in a clear and user-friendly manner.
3. **Execute Button**: A button to run the function after selecting the DataFrame and specifying the standard deviation.
4. **Output Display**: A section to view the DataFrame after outlier removal or modification.

# normalize\_data

**Description**

* **Purpose**: To normalize data in a specified DataFrame column using min-max scaling or z-score normalization.
* **Functionality**: Checks if a DataFrame is loaded, verifies the column's existence, and applies the selected normalization method.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if an invalid normalization method is chosen.

**Arguments**

1. **column\_name (str)**: The name of the column to be normalized.
2. **method (str)**: The normalization method to use. Options are 'min-max' and 'z-score'.

**UI Requirements**

1. **DataFrame Input**: A method to input or select a DataFrame, potentially through file upload or a DataFrame viewer/editor.
2. **Column Selector**: An interface element like a dropdown to choose the column to be normalized.
3. **Method Selector**: A dropdown or radio buttons to select the normalization method ('min-max' or 'z-score').
4. **Error Display**: An area to display errors such as 'Dataframe not loaded', 'Column name not found', or 'Invalid method'.
5. **Execute Button**: A button to initiate the normalization process after the DataFrame, column, and method are chosen.
6. **Output Display**: A section to view the DataFrame post-normalization.

# rename\_column

**Description**

* **Purpose**: To rename a column in a DataFrame to a new specified name.
* **Functionality**: Validates whether the DataFrame is loaded, ensures the new column name is a string, checks if the old column name exists, and then renames the column.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the new column name is not a string, or if the old column name does not exist.

**Arguments**

1. **old\_name (str)**: The current name of the column to be renamed.
2. **new\_name (str)**: The new name for the column.

**UI Requirements**

1. **Old Name Input/Select Field**: An input field for the user to enter the current name of the column/select
2. **New Name Input Field**: An input field for the user to enter the new name for the column.
3. **Error Display**: A section to display errors (e.g., 'Dataframe not loaded', 'new\_name must be a string', 'Column name not found').
4. **Execute Button**: A button to initiate the column renaming process after the DataFrame and column names are specified.
5. **Output Display**: A section to view the DataFrame after the column has been renamed.

# remove\_special\_characters

**Description**

* **Purpose**: To clean a text column in a DataFrame by removing any special characters, leaving only alphanumeric characters and spaces.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified column exists and is of string type, and then removes special characters using regular expressions.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if the column is not a text column.

**Arguments**

**column\_name (str)**: The name of the text column from which to remove special characters.

**UI Requirements**

1. **Column Selector**: An interface element, like a dropdown menu, for selecting the text column to be processed.
2. **Error Display**: An area to show errors such as 'Dataframe not loaded', 'Column name not found', or 'Column is not a text column'.
3. **Execute Button**: A button to initiate the removal of special characters after the DataFrame and column are specified.
4. **Output Display**: A section to display the modified DataFrame or to indicate that the process is complete.

# change\_column\_type

**Description**

* **Purpose**: To change the data type of a specific column in a DataFrame to a new, specified type.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified column exists, and then changes its data type to the new specified type.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if there's an issue during the data type conversion.

**Arguments**

1. **column\_name (str)**: The name of the column for which the data type is to be changed.
2. **new\_type**: The new data type to be assigned to the column. This should be a valid pandas data type.

**UI Requirements**

1. **DataFrame Input**: A method to input or select a DataFrame, possibly through file upload or a DataFrame viewer/editor.
2. **Column Selector**: An interface element (like a dropdown menu) for selecting the column whose data type needs to be changed.
3. **New Type Selector/Input**: An interface element to select or input the new data type. This could be a dropdown of common pandas data types or a text input field.
4. **Error Display**: A section to display errors (e.g., 'Dataframe not loaded', 'Column name not found', 'Error during data type change').
5. **Execute Button**: A button to start the data type conversion process after the DataFrame, column, and new type are specified.
6. **Output Display**: A section to view the DataFrame after the column's data type has been changed.

# trim\_whitespace

**Description**

* **Purpose**: To clean a text column in a DataFrame by trimming any leading and trailing whitespace.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified column exists and is of string type, and then trims whitespace from the column.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if the column is not a text column.

**Arguments**

**column\_name (str)**: The name of the text column from which to trim whitespace.

**UI Requirements**

1. **Column Selector**: An interface element, like a dropdown menu, for selecting the text column to be processed.
2. **Error Display**: An area to show errors such as 'Dataframe not loaded', 'Column name not found', or 'Column is not a text column'.
3. **Execute Button**: A button to initiate the trimming of whitespace after the DataFrame and column are specified.
4. **Output Display**: A section to display the modified DataFrame or to indicate that the process is complete.

# label\_encode

**Description**

* **Purpose**: To convert categorical values in a specified column of a DataFrame into numerical labels.
* **Functionality**: Checks if a DataFrame is loaded, verifies if the specified column exists and is categorical, and then applies label encoding.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, or if the column is not categorical.

**Arguments**

**column\_name (str)**: The name of the column to be label encoded.

**UI Requirements**

1. **DataFrame Input**: A method to input or select a DataFrame, such as through file upload or a DataFrame viewer/editor.
2. **Column Selector**: An interface element, like a dropdown menu, for selecting the categorical column to be label encoded.
3. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'Column name not found', or 'Column is not a categorical column'.
4. **Execute Button**: A button to start the label encoding process after the DataFrame and column are specified.
5. **Output Display**: A section to view the DataFrame after the label encoding has been applied to the column.

# bin\_numeric\_to\_categorical

**Description**

* **Purpose**: To categorize numeric data by binning it into specified ranges, transforming a numeric column into a categorical one.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified column exists and is numeric, and then applies binning based on the provided bins and optional labels.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, if the column is not numeric, or if the bins or labels are incorrectly specified.

**Arguments**

1. **column\_name (str)**: The name of the numeric column to convert.
2. **bins (list)**: The edges defining the bins. This should be a list of numbers.
3. **labels (list, optional)**: Labels for the bins. The length should be one less than the number of bins.

**UI Requirements**

1. **Column Selector**: An interface element, like a dropdown menu, for selecting the numeric column to be binned.
2. **Bins Input Field**: An input field for specifying the bin edges.
3. **Labels Input Field**: An optional input field for specifying labels for the bins.
4. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'Column name not found', 'Column is not numeric', or errors related to bin/label specifications.
5. **Execute Button**: A button to start the binning process after the DataFrame, column, bins, and optional labels are specified.
6. **Output Display**: A section to view the DataFrame after binning has been applied to the colum

# remove\_columns

**Description**

* **Purpose**: To remove one or more specified columns from a DataFrame.
* **Functionality**: Validates if a DataFrame is loaded, ensures that the input is a list of column names, checks if the specified columns exist, and then removes them.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the input is not a list, or if any specified column does not exist.

**Arguments**

**columns\_to\_remove (list)**: A list of column names to be removed from the DataFrame.

**UI Requirements**

1. **DataFrame Input**: A method to input or select a DataFrame, such as through file upload or a DataFrame viewer/editor.
2. **Columns Selector**: An interface element for selecting multiple columns to be removed. This could be a multi-select dropdown or a series of checkboxes.
3. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'columns\_to\_remove should be a list', or 'Column name not found'.
4. **Execute Button**: A button to initiate the removal of selected columns after the DataFrame and columns are specified.
5. **Output Display**: A section to view the DataFrame after the specified columns have been removed.

# extract\_datetime\_components

**Description**

* **Purpose**: To extract specific components like year, month, day, hour, minute, or second from a datetime column and add them as new columns in the DataFrame.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified column exists and is a datetime type, then extracts the specified components, converting months to their text representation and interpreting 'day' as 'day of the week'.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, if the column is not a datetime type, or if invalid components are specified.

**Arguments**

**column\_name (str)**: The name of the datetime column.

**components (list)**: A list of datetime components to extract. Valid options include 'year', 'month', 'day', 'hour', 'minute', 'second'.

**UI Requirements**

1. **Column Selector**: An interface element, like a dropdown menu, for selecting the datetime column.
2. **Components Selector**: A multi-select interface element for choosing the datetime components to extract. This could be a series of checkboxes or a multi-select dropdown.
3. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'Column name not found', 'Column is not a datetime column', or 'Invalid component'.
4. **Execute Button**: A button to start the extraction process after the DataFrame, column, and components are specified.
5. **Output Display**: A section to view the DataFrame after the datetime components have been extracted and added as new columns.

"extract\_datetime\_components": {

"example\_datetime\_column": [

"year",

"month",

"day",

"hour",

"minute",

"second"

]

}

# replace\_substring

**Description**

* **Purpose**: To replace occurrences of a specific substring with a new substring within a text column of a DataFrame.
* **Functionality**: Checks if a DataFrame is loaded, verifies if the specified column exists and is of text type, and then performs the substring replacement.
* **Error Handling**: Raises errors if no DataFrame is loaded, if the specified column doesn't exist, if the column is not a text column, or if there's an issue during the replacement process.

**Arguments**

1. **column\_name (str)**: The name of the text column in which to replace the substring.
2. **old\_substring (str)**: The substring to be replaced.
3. **new\_substring (str)**: The new substring to replace the old substring.

**UI Requirements**

1. **Column Selector**: An interface element, like a dropdown menu, for selecting the text column for substring replacement.
2. **Old Substring Input Field**: An input field for entering the old substring to be replaced.
3. **New Substring Input Field**: An input field for entering the new substring.
4. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'Column name not found', or 'Column is not a text column'.
5. **Execute Button**: A button to initiate the substring replacement process after the DataFrame, column, and substrings are specified.
6. **Output Display**: A section to view the DataFrame after the replacement has been applied to the column.

"replace\_substring": {

"example\_text\_column": {

"old\_substring": "example\_old\_substring",

"new\_substring": "example\_new\_substring"

}

}

# apply\_pca

**Description**

* **Purpose**: To reduce the dimensionality of data by applying PCA on selected columns, keeping a specified number of principal components.
* **Functionality**: Validates if a DataFrame is loaded, checks if the specified columns exist, extracts the relevant data, performs PCA, and appends the principal components to the DataFrame while dropping the original columns.
* **Error Handling**: Raises errors if no DataFrame is loaded, if any specified column doesn't exist, or if there's an issue during the PCA process.

**Arguments**

1. **columns (list)**: List of column names on which to apply PCA.
2. **n\_components (int, optional)**: The number of principal components to retain. If **None**, all components are kept.

**UI Requirements**

1. **DataFrame Input**: A method to input or select a DataFrame, such as through file upload or a DataFrame viewer/editor.
2. **Columns Selector**: An interface element for selecting multiple columns on which PCA is to be applied. This could be a multi-select dropdown or a series of checkboxes.
3. **Number of Components Input Field**: An optional input field for specifying the number of principal components to retain.
4. **Error Display**: A section to display errors, such as 'Dataframe not loaded', 'Column name not found', or errors related to the PCA process.
5. **Execute Button**: A button to start the PCA process after the DataFrame, columns, and number of components are specified.
6. **Output Display**: A section to view the DataFrame after PCA has been applied, including the new principal component columns.

"apply\_pca": {

"columns": [

"example\_column1",

"example\_column2"

],

"n\_components": "example\_number\_of\_components"

}