xlsFilter – Initial idea, code and approach by Shubham A. Dekatey (<https://github.com/deka1105>)

Prerequisites:

|  |  |
| --- | --- |
| pandas | * Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet * Ordered and unordered (not necessarily fixed-frequency) time series data. * Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels * Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure |
|  | To write a single object to an Excel .xlsx file it is only necessary to specify a target file name. To write to multiple sheets it is necessary to create an ExcelWriter object with a target file name, and specify a sheet in the file to write to. |
| numpy | * a powerful N-dimensional array object * sophisticated (broadcasting) functions * tools for integrating C/C++ and Fortran code * useful linear algebra, Fourier transform, and random number capabilities * and much more |
| datetime | * Decoupling between operations on date and time objects and their representation. * Access to different calendars, for input parsing and output formatting. * Access to different time representations, also for input and output. * Infinite precision in converting between different time representations. |
| time | This module provides various time-related functions. For related functionality, see also the [datetime](https://docs.python.org/2/library/datetime.html#module-datetime) and [calendar](https://docs.python.org/2/library/calendar.html#module-calendar) modules. |
| openpyxl | openpyxl is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files.  It was born from lack of existing library to read/write natively from Python the Office Open XML format.  All kudos to the PHPExcel team as openpyxl was initially based on PHPExcel. |
| operator | The [operator](https://docs.python.org/2/library/operator.html#module-operator) module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y. The function names are those used for special class methods; variants without leading and trailing \_\_ are also provided for convenience.  The functions fall into categories that perform object comparisons, logical operations, mathematical operations, sequence operations, and abstract type tests. |
| cufflinks | This library binds the power of [plotly](http://www.plot.ly/) with the flexibility of [pandas](http://pandas.pydata.org/) for easy plotting. |
| plotly.offline | Plotly at its core is a data visualization toolbox. Under every plotly graph is a JSON object, which is a dictionary like data structure. Simply by changing the values of some keywords in this object, we can get vastly different and ever more detailed plots. |
| seaborn | * A dataset-oriented API for examining relationships between multiple variables * Specialized support for using categorical variables to show observations or aggregate statistics * Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data * Automatic estimation and plotting of linear regression models for different kinds dependent variables * Convenient views onto the overall structure of complex datasets |

This library was a result of a very basic task of getting count of certain fields out of an excel sheet, which can be accomplished y default functions of pandas.dataframe. The challenge was the data was very heterogeneous yet in the same class. Some fields which were intended to be nominal fields has variation upto 50 different labels, which were representation for only 10-15 unique labels. Data fields with 50 labels is hard to handle and analyse, nonetheless those were redundant labels.

|  |  |
| --- | --- |
| Black Chair | Chair |
| White desk | Desk |
| Blue Bottle | Bottle |
| Blue Chair | Chair |
| White Bottle | Bottle |
| Black Desk | Desk |
| White Desk | Desk |
| White chair | Chair |

As the above table demonstrates the kind of data explained above, the first column of the table had 8 distinctly identified labels, where as they could be easily mapped down to just 3 distinct labels (if we were to analyze on basis of object) as shown in next column. This was possible only by knowning the smaller subset of labels which were needed to be mapped. In this case [Chair, Desk, Bottle], finding these labels in primary(first) column and storing the output into Secondary(second) column or directly operating is the back bone of this library.

To analyze any data we require count of distinct labels across various fields of the dataframe. Getting count is again an easy task with pre defined pandas.dataframe functions, but getting the exclusive inclusion count or getting an exclusive exclusion count is again a challenge for pre defined functions.



1. \_\_init\_\_(self, src, log, xls, ind)

Constructor of the class, initializing global variables of the class.

Define four path variables,

|  |  |
| --- | --- |
| src | Path for xlsx file with the filename and extension |
| xls | Path to a directory where the output xlsx sheets would be stored |
| log | Path to a directory where logs for the operations would be stored for review |
| clog | Path for xlsx file with the filename and extension, this file essentially has three fields which can be referred directly instead of drilling down into logs or output directories |

>>>from xlsModule import xlsxFilter

Imports ‘xlsxFilter’ class from ‘xlsModule’ python file.

>>>src = "C:/Users/admin/Documents/DataSheets-RAW/sourceFile.xlsx"

>>>log = "C:/Users/admin/Documents/Jupyter-Excel-Exports/"

>>>xls = "C:/Users/admin/Documents/Logs/"

>>>ind = "C:/Users/admin/Documents/CentralLog.xlsx"

>>>redcon = xlsxFilter(src, log, xls, clog)

‘redcon’ is the constructor of the xlsxFilter class, intiallizing variables (src, log, xls, clog) for complete class to utilize when required.

>>>defcon = redcon.initiate('Report 1')

1. initiate(self, worksheet)

‘initiate’ method of the xlsxFilter class, accepts sheet name from the xlsx file, referred by the src variable, and returns the pandas.dataframe format of the same data and also prints first 5 rows of data using head().

Therefore, now ‘defcon’ has a dataframe of complete data.

|  |  |  |  |
| --- | --- | --- | --- |
| Floor | Assets | Available | Asset\_Nominal |
| First | Black Chair | Yes | Chair |
| Second | White desk | No | Desk |
| First | Blue Bottle | Yes | Bottle |
| Third | Blue Chair | Yes | Chair |
| Second | White Bottle | Yes | Bottle |
| First | Black Desk | Yes | Desk |
| Basement 2 | White Desk | No | Desk |
| Second | White chair | No | Chair |
| Basement | Black Chair | Yes | Chair |
| First | White desk | No | Desk |
| Second | Blue Bottle | Yes | Bottle |
| First | Blue Chair | Yes | Chair |
| Third | White Bottle | Yes | Bottle |
| Second | Black Desk | Yes | Desk |
| First | White Desk | No | Desk |
| Third | White chair | No | Chair |

>>>defcon1 = excon.nullRemover(defcon, [‘Floor’])

>>>defcon2 = excon.reduceData(defcon1, ‘Floor’, [‘Basement’, ‘Basement 2’], 'ex')

>>>defcon3 = excon.reduceData(defcon2, ' Available ', [‘Yes’], 'in')

>>>userList1 = [‘Chair’] #What to include

>>>userList2 = [‘Bottle’] #What to exclusively exclude

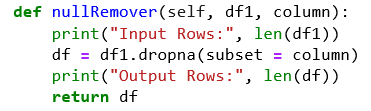
>>>qCode = "Qry0010"

>>>queryText = "Demonstration Query 1"

>>>defcon2 = excon.FilterDataFrameAdv(qCode, queryText, defcon3, ‘Floor’, Asset\_Nominal', userList1, userList2, 1, 'null')

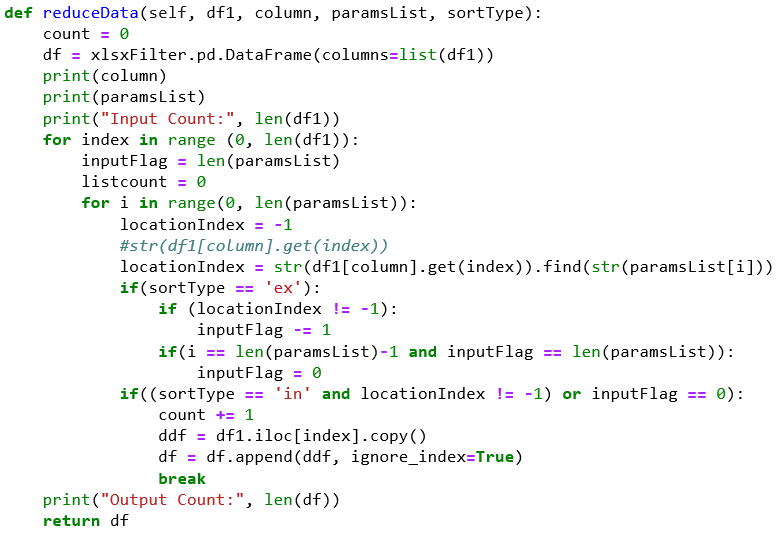
1. nullRemover(dataframe, list\_of\_ColumnNames) : return dataframe

Removes the complete row data with null values in the given column.



1. reduceData(dataframe, columnName, list\_of\_labels, tag) : return dataframe

Reduces the dataframe based on the ‘tag’ argument of the method, there are two tags, ‘in’ : Include, ‘ex’ : Exclude. This method returns a dataframe with only those rows which has/has not values for the labels as in the list\_of\_labels in the columnName column.



1. FilterDataFrameAdv(self, qCode, queryText, defcon2, primarycolumn, secondaryField, exclusiveInclude, exclusiveExclude, exportFlag, xlsxFilePath) : return dataframe

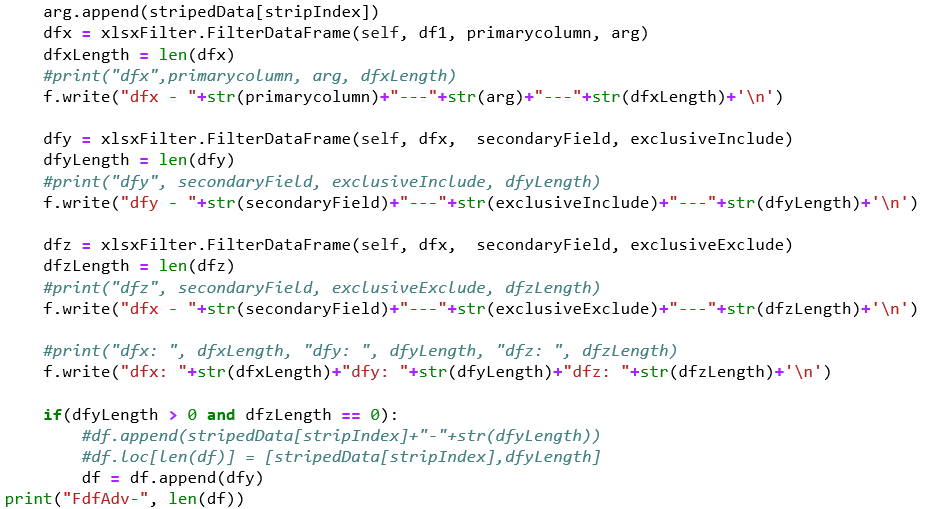
|  |  |
| --- | --- |
| qCode, queryText | Query Code, and query to log in the clog file, for better mapping and indexing |
| Defcon2 | Dataframe |
| primaryColumn | The column on which it is needed to operate |
| secondaryField | The column from which you will filter labels |
| exclusiveInclude | List of labels to include with unique primaryColumn Labels |
| exclusiveExclude | List of labels to exclude with unique primaryColumn Labels, if even one row matches the exclusiveExclude list elements, the unique primaryColumn label is dropped out |
| exportFlag | Integer 1 represents, you want export the output dataframe to excel sheet |
| xlsxFilePath | Path of the directory where you want your output excel sheet should be stored |

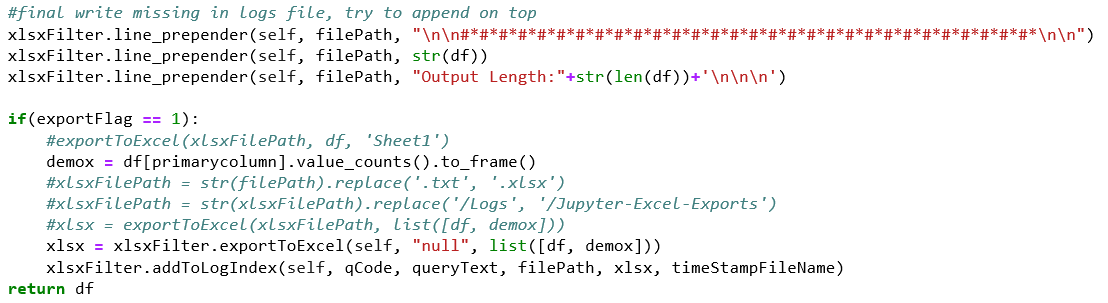
The method finds all the unique labels in the primaryColumn as you want to add all other paramenters or fields on the basis of this column. Then complete dataframe is checked for the occurance of exclusiveInclude List objects in the secondaryField itratively for each distinct label of primaryColumn. Then cross checks for exclusiceExclude by same methodology, if results of exclusiveExclude is positive or not zero, the distinct label of primaryColumn is discarded. Else the output of exclusiveInclude is concatted to final output of the method.

This method also maintains textual log files and excel output files, as well as central log files. The output is divided into two separate excel sheets, one which has all the rows of source dataframe, and another with just the distinct label of primaryColumn and the count of them.

To achieve these operations the method calls some other methods which are: FilterDataFrame, line\_prepender, exportToExcel, addToLogIndex.



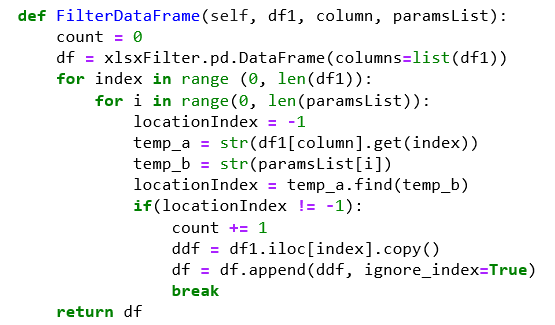




1. FilterDataFrame(self, defcon2, column, paramsList) : return dataframe

|  |  |
| --- | --- |
| Defcon2 | Dataframe of the data |
| Column | The field name in which to check the paramsList |
| paramsList | List of parameters/labels |

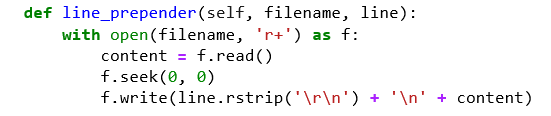
This method returns the same dataframe structure, with removing the rows for which the ‘column’ label has substring in the paramsList.



1. line\_prepender(self, filename, line)

|  |  |
| --- | --- |
| Filename | Text File path in which the operation is to be performed |
| Line | String which has to be appended in the text file |

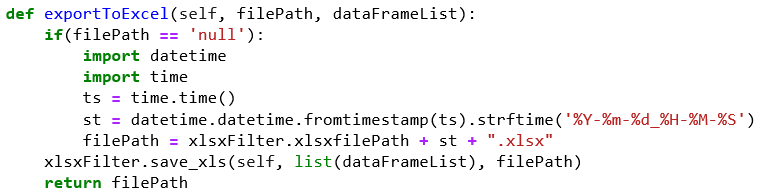
This function appends ‘line’ at the End of the File (EOF). You can use several string or string patterns to make your text file (for this library a log file) more organized and structured.



1. exportToExcel(self, filePath, dataFrameList) : Return String

|  |  |
| --- | --- |
| filePath | Excel File or directory path in which the operation is to be performed |
| dataframList | List of dataframes which will be stored distinctly in different excel workbooks |

If the excel file is not at the filePath location, a new excel file will be created and list of dataframes which will be stored distinctly in different excel workbooks.

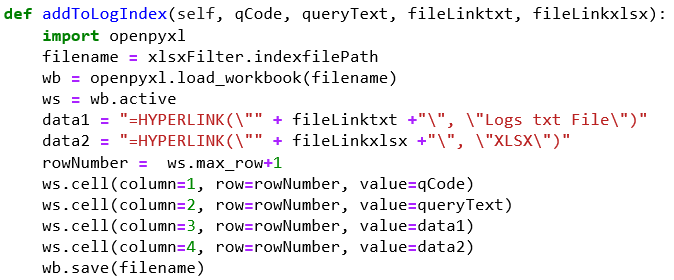


1. addToLogIndex(self, qCode, queryText, fileLinktxt, fileLinkxlsx)

|  |  |
| --- | --- |
| qCode | User defined string text |
| queryText | User defined string query |
| fileLinktxt | Filepath to the text log file |
| fileLinkxlsx | Filepath to the output excel file |

This method deals with central log excel file, whose path was passed to constructor in ‘clog’ variable. Logs in a new row when invoked, he qCode is introduced for better organization of the central logging and queryText for better details, and links to the path of txt log file and output excel file.

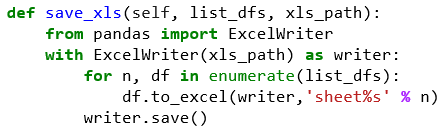
Note : This function only modifies existing central log excel file, if not found will throw an error and you will need to create new (blank preferred) excel file, and provide path to ‘clog’ variable.



1. save\_xls(self, list\_dfs, xls\_path)

|  |  |
| --- | --- |
| list\_dfs | List of multiple dataframes |
| xls\_path | Path to the excel file in which these dataframes are to be stored |

This method stores multiple dataframes in one single excel file but in individual sheets.

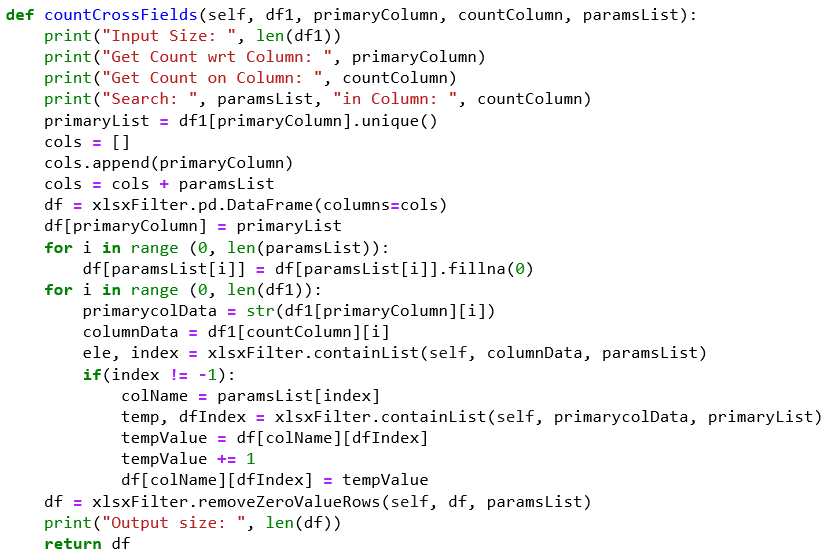


1. countCrossFields(self, df1, primaryColumn, countColumn, paramsList) : return dataframe

|  |  |
| --- | --- |
| Df1 | Dataframe |
| primaryColumn | Name of the column according which count will be extracted |
| countColumn | Name of the column across which counts will be done as per paramsList |
| paramsList | List of elements which are to be counted |

This method returns a different format of dataframe than the one which is provided through the argument. The output dataframe has Number of paramsList elements + 1 column fields, i.e. primaryColumn + paramsList.

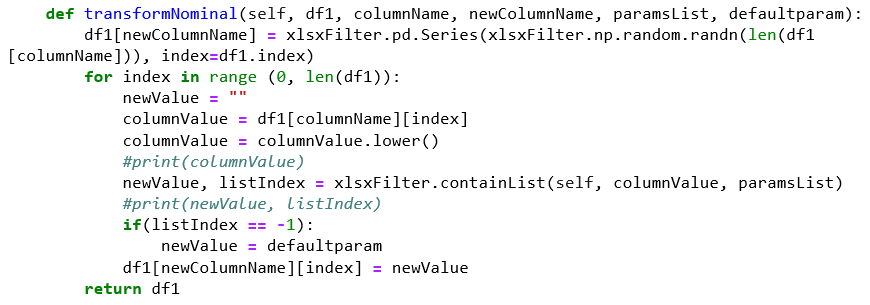
First all the unique labels in primaryColumn are identified and added to the output dataframe, the remaining dataframe cells are filled with 0 (integer). For each distinct primaryColumn label, the datasheet is itrated, checking countColumn labels matching for substring with the elements of the paramsList. As a match is found, the count of that cell [primaryColumn label, element matched] is incremented by 1.



1. def transformNominal(self, df1, columnName, newColumnName, paramsList, defaultparam) : return dataframe

|  |  |
| --- | --- |
| Df1 | Dataframe |
| columnName | Existing column name |
| newColumnName | New column name which will hold nominal values |
| paramsList | List of strings which will be nominal values |
| defaultparam | Nominal value which will be assigned as default |

A new column is created as ‘newColumnName’, ‘columnName’ labels matching for substring with the elements of the paramsList. As a match is found, the element from paramsList is assigned in newColumnName, else defaultparam is assigned.



1. countCrossFieldsNominal(self, df1, primaryColumn, countColumn, paramsList) ) : return dataframe

|  |  |
| --- | --- |
| Df1 | Dataframe |
| primaryColumn | Name of the column according which count will be extracted |
| countColumn | Name of the column across which counts will be done as per paramsList containing Nominal labels |
| paramsList | List of elements which are to be counted |

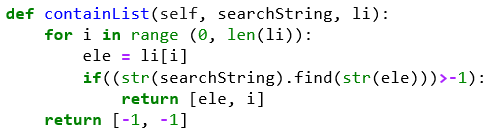
This method is similar to countCrossFields, but it doesn’t need to search for substring, as the countColumn is nominal, this method just checks if the label of countColumn in paramsList which returns index of element matched, and then increments the count in respective column, which is index + 1. But if the element is not found in the list, you get -1, and terminate the current row and jump to next.



1. containList(self, searchString, li) : return[list element, element Index]

|  |  |
| --- | --- |
| searchString | Element which has to be searched in the list |
| Li | List in which element has to be searched |

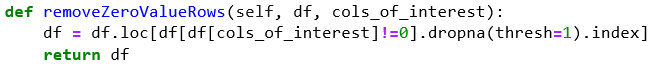
This method itrattively checks each element of list to return list element as well as its index if found else -1.



1. removeZeroValueRows(self, df, cols\_of\_interest) : return dataframe

|  |  |
| --- | --- |
| Df | Dataframe |
| Cols\_of\_interest | List of column names |

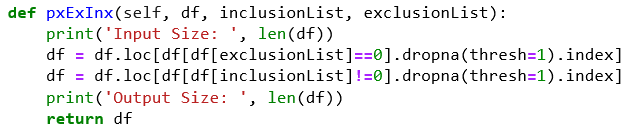
Removing the rows which has labels as 0 (integer) in there corresponding class/column.



1. pxExInx(self, df, inclusionList, exclusionList) : return dataframe

|  |  |
| --- | --- |
| df | Dataframe |
| inclusionList | List of columns whose value should be above 0 |
| exclusionList | List of columns whose value should be 0 |

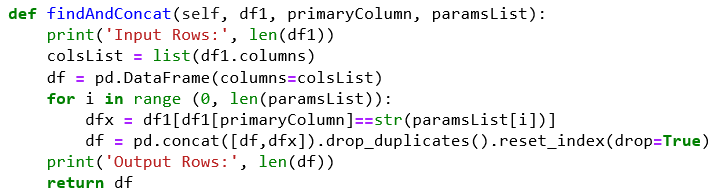
This method is only possible if all the column names in inclusionList and exclusionList have integer data. If so, then it will provide output with those rows which has zero for exclusionList elemets as column names.



1. findAndConcat(self, df1, primaryColumn, paramsList) : return dataframe

|  |  |
| --- | --- |
| Defcon2 | Dataframe of the data |
| Column | The field name in which to check the paramsList |
| paramsList | List of parameters/labels |

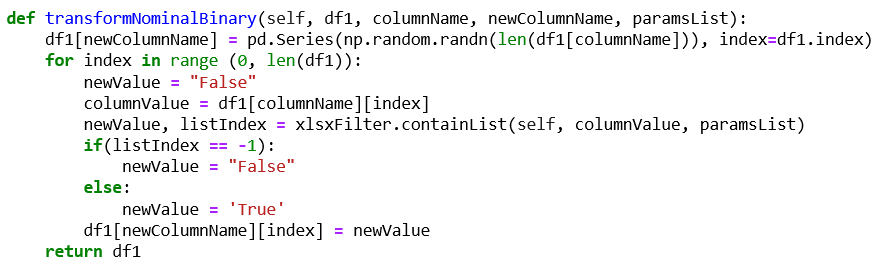
This function is similar to FilterDataFrame but more efficient and faster.



1. transformNominalBinary(self, df1, columnName, newColumnName, paramsList, defaultparam) : return dataframe

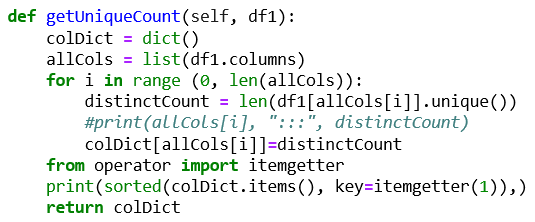
|  |  |
| --- | --- |
| Df1 | Dataframe |
| columnName | Existing column name with nominal values |
| newColumnName | New column name which will hold binary values |
| paramsList | List of strings which will be marked as True |

This function is similar to transformNominal if the labels of the columnName column are elements of paramsList, they will me marked as ‘True’ in newColumnName.



1. getUniqueCount(self, df1) : return dict

A dataframe is provided as an argument to this method and it returns a dictionary of all column names as keys, and count of unique labels in that column as values.



1. vizualizeDf(self, df):

This method will create an interactive plot for the given dataframe.

Note: The dataframe being supplied to the method should have integer values in all the columns except one.

