Kumaravelu Sarabojimahalingam (Cognizant)

COGNIZANT TECHNOLOGY SOLUTIONS LTD

**Easy and quick approach to develop complex pivot table reports using Python**

Table of Contents

[Objective 2](#_Toc486332279)

[Target audience 2](#_Toc486332280)

[Approach 2](#_Toc486332281)

[Assumptions/Constraints 2](#_Toc486332282)

[Load data from any data source to a dataframe 2](#_Toc486332283)

[Fetch records from a RDBMS database 2](#_Toc486332284)

[Load the data to a dataframe 2](#_Toc486332285)

[Code excerpt 3](#_Toc486332286)

[Areas where Python scores over manual method 3](#_Toc486332287)

[Analyze the characteristics of a pivot table and opt for an approach 3](#_Toc486332288)

[Approach-1: Pivot table doesn’t have multi-level columns & doesn’t include any weighted average calculations 4](#_Toc486332289)

[Steps involved 4](#_Toc486332290)

[Code excerpt 6](#_Toc486332291)

[Areas where Python scores over manual method 6](#_Toc486332292)

[Approach-2: Pivot table doesn’t have multi-level columns & includes weighted average calculations 7](#_Toc486332293)

[Steps involved 7](#_Toc486332294)

[Code excerpt 11](#_Toc486332295)

[Areas where Python scores over manual method 13](#_Toc486332296)

[Approach-3: Pivot table has multi-level columns & doesn’t include any weighted average calculations 13](#_Toc486332297)

[Steps involved 13](#_Toc486332298)

[Code excerpt 16](#_Toc486332299)

[Approach-4: Pivot table has multi-level columns & also include weighted average calculations 17](#_Toc486332300)

[Steps involved 17](#_Toc486332301)

[Code excerpt 23](#_Toc486332302)

[Observations and conclusion 24](#_Toc486332303)

# Objective

The objective of this paper is to delineate step by step approach that is easy and quick to develop complex pivot table like reports using Python. It also lists down the areas where Python scores over the limitations/constraints associated with manual creation of these reports in Microsoft excel. Pandas, Numpy, Openpyxl are the Python modules/packages (open source) considered in this approach

# Target audience

* Any technology specialist who manually and frequently creates large # of reports that extensively uses pivot tables
* Any technology specialist interested to understand the nitty-gritties of the pivot table reporting features available in Python programming language.

# Approach

Data extracted from DB2 RDBMS database (using ibm\_db package) is loaded to pandas dataframe data structure. Loaded data is munged before pivot tables are created. Finally, styles are applied to the pivot tables and the XLS report is generated

# Assumptions/Constraints

* Reports that are generated/tested are based out of the data spanning to a max of ~60K excel rows
* Tabular form is the report layout considered in this approach

# Load data from any data source to a dataframe

### Fetch records from a RDBMS database

* Connect to DB2
* Prepare the SQL. Execute and fetch the resultant using fetch\_tuple ibm\_db method
* Append the fetched tuple to a list

### Load the data to a dataframe

* Using ‘from\_records’ pandas method load the data from the list to a dataframe
* Ignore the index but specify appropriate column names
* Cast each of the dataframe columns to appropriate data types
* Strip any trailing spaces, prefix/suffix any characters, remove any unwanted characters/bytes
* Using ‘to\_excel’ pandas method save the dataframe to an excel sheet

### Code excerpt



### Areas where Python scores over manual method

* In most of the shops - pivot table reports are generated using data spread across RDBMS data sources like Oracle, DB2, mySQL etc., In the manual method even though we can connect to different databases, writing complex queries and lookups is not quite easy and efficient as implementing in Python. With plethora of drivers and Python packages available from the leading database vendors – data extraction has turned simpler resulting in faster Time to Market
* Munging of dataframe data is easier than to manual method

# Analyze the characteristics of a pivot table and opt for an approach

Based on the characteristics of pivot table the development approach varies. Please see below for the 4 approaches using which any kind of pivot table can be generated using Pandas. The sequence of steps listed within each approach should be followed in the same order

* Approach-1: Pivot table doesn’t have multi-level columns & doesn’t include any weighted average calculations
* Approach-2: Pivot table doesn’t have multi-level columns & includes weighted average calculations
* Approach-3: Pivot table has multi-level columns & doesn’t include any weighted average calculations
* Approach-4: Pivot table has multi-level columns & also includes weighted average calculations

## Approach-1: Pivot table doesn’t have multi-level columns & doesn’t include any weighted average calculations

### Steps involved

#### Index, summarizations and grand total

* + Identify all those dataframe columns (from step-1) that doesn’t participate in the summarization (count/sum/average etc.,) and pass them to ‘index’ parameter in pivot\_table pandas method.
	+ Identify all those dataframe columns (from step-1) that participates in the summarization and pass them to ‘values’ parameter in pivot\_table pandas method
	+ Pass the summarization functions as a list to ‘aggfunc’ parameter of ‘pivot\_table’ pandas method
	+ To get the grand summarization set the margins parameter of ‘pivot\_table’ pandas method to True
	+ Build a dataframe (2) using the ‘pivot\_table’ pandas method



#### Sub totals calculation

* + Identify all those dataframe columns (from step-1) that must be grouped for subtotals and pass them to groupby pandas method.
	+ Each of the grouping will be of pandas data type – dtype



#### Concatenate subtotal dataframes to build the final pivot table

* + Convert each of the dtype to multi-index dataframe as below. The new dataframes should align with the dataframe (2) indexes/levels
		- Get the index values for each of the levels and then form a tuple using zip method
		- Pass the tuple list to MultiIndex.from\_tuples pandas method and build the index/dataframes
	+ Concatenate the newly formed dataframes to dataframe (2) to form a consolidated dataframe (3)
	+ Sort the indexes

#### % to calculation (calculated fields)

####

* + Percentages are calculated using ‘iloc’ method and division operators. The percentages will appear as new dataframe (3) columns.
	+ The % columns should be aptly named so that the new columns gets created adjacent to their respective summarized columns

#### Casting, formatting and cleansing the dataframe

* + Using ‘astype’ pandas method, each of the dataframe (3) series/columns are casted to the required data types.
	+ Using ‘apply’ pandas method and format python method, $ can be prefixed, numbers can be comma separated etc.., on the applicable dataframe (3) series.
	+ Any unwanted rows/columns in the dataframe (3) can be removed using ‘drop’ pandas method



### Code excerpt



### Areas where Python scores over manual method

* Calculated fields are easier and transparent using pandas than to manual method

## Approach-2: Pivot table doesn’t have multi-level columns & includes weighted average calculations

### Steps involved

#### Summarizations

* + Identify the indexes (dataframe series by which the values must be grouped by) of final pivot table
	+ Slice the needed dataframe columns (from step-1) and apply summarization functions one at a time using ‘groupby’ pandas method. Pass the indexes as a list to the groupby function. For eg: if count, sum and weighted average are values to be calculated - create a data type of dtype for each of the 3 functions.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (2)
	+ Reset the index of dataframe (2)



#### Grand total calculation

* + Using ‘deep copy’ pandas method - copy the dataframe (from step-1) to a new dataframe (3)
	+ Select all the series (that belongs to identified indexes) from dataframe (3) and replace all the values with a constant value (eg: ‘zAll’).
	+ Slice the modified dataframe column and apply summarization functions one at a time using ‘groupby’ pandas method. Pass the indexes as a list to the groupby function. For eg: if count, sum and weighted average are values to be calculated - create a data type of dtype for each of the 3 functions.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (4)
	+ Reset the index of dataframe (4)



#### Sub totals calculation

* + Identify the series corresponding to the columns by which the data needs to be grouped by
	+ Append a constant like ‘zAll’ to the rightmost column belonging to the grouping
	+ Slice the needed dataframe columns (from step-1) and apply summarization functions one at a time using ‘groupby’ pandas method. Pass the indexes as a list to the groupby function.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (6)
	+ Reset the index of dataframe (6)

#### Concatenate subtotal and grand total dataframes to build the final pivot table

* + Concatenate the newly formed grand total and sub-total dataframes to dataframe (2) to form a consolidated dataframe (8)
	+ Re-index, set the index and sort them

#### % to calculation (calculated fields)

####

* + Percentages are calculated using ‘iloc’ method and division operators. The percentages will appear as new dataframe (8) columns.
	+ The % columns should be aptly named so that the new columns gets created adjacent to their respective summarized columns

#### Casting, formatting and cleansing the dataframe

* + Using ‘astype’ pandas method, each of the dataframe (8) series/columns are casted to the required data types.
	+ Using ‘apply’ pandas method and format python method, $ can be prefixed, numbers can be comma separated etc.., on the applicable dataframe (8) series.
	+ Any unwanted rows/columns in the dataframe (8) can be removed using ‘drop’ pandas method



### Code excerpt





### Areas where Python scores over manual method

* Weighted calculated fields are easier and transparent using pandas than to manual method

## Approach-3: Pivot table has multi-level columns & doesn’t include any weighted average calculations

### Steps involved

#### Index, summarizations and grand total

* + Identify all those dataframe columns (from step-1) that doesn’t participate in the summarization (count/sum/average etc.,) and pass them to ‘index’ parameter in pivot\_table pandas method.
	+ Identify all those dataframe columns (from step-1) that participates in the summarization and pass them to ‘values’ parameter in pivot\_table pandas method
	+ Identify all those dataframe columns (from step-1) that participates in the summarization and pass them to ‘columns parameter in pivot\_table pandas method
	+ Pass the summarization functions as a list to ‘aggfunc’ parameter of ‘pivot\_table’ pandas method
	+ To get the grand summarization set the margins parameter of ‘pivot\_table’ pandas method to True
	+ Build a dataframe (2) using the ‘pivot\_table’ pandas method
	+ Swap the column levels if needed



#### Sub totals calculation

* + Identify all those dataframe columns (from step-1) that must be grouped for subtotals and pass them to groupby pandas method.
	+ Each of the grouping will be of pandas data type – dtype



#### Concatenate subtotal dataframes to build the final pivot table

* + Convert each of the dtype to multi-index dataframe as below. The new dataframes should align with the dataframe (2) indexes/levels
		- Get the index values for each of the levels and then form a tuple using zip method
		- Pass the tuple list to MultiIndex.from\_tuples pandas method and build the index/dataframes
	+ Concatenate the newly formed dataframes to dataframe (2) to form a consolidated dataframe (3)
	+ Sort the indexes

#### % to calculation (calculated fields)

####

* + Get the unique values for each column levels.
	+ Percentages are calculated by iterating over the appropriate column level and using ‘iloc’ method and division operators. The percentages will appear as new dataframe (3) columns.
	+ The % columns should be aptly named so that the new columns gets created adjacent to their respective summarized columns

#### Casting, formatting and cleansing the dataframe

* + Using ‘astype’ pandas method, each of the dataframe (3) series/columns are casted to the required data types.
	+ Using ‘apply’ pandas method and format python method, $ can be prefixed, numbers can be comma separated etc.., on the applicable dataframe (3) series.
	+ Any unwanted rows/columns in the dataframe (3) can be removed using ‘drop’ pandas method



### Code excerpt







## Approach-4: Pivot table has multi-level columns & also include weighted average calculations

### Steps involved

#### Summarizations

* + Slice the needed dataframe columns (from step-1) and apply summarization functions one at a time over the value fields using ‘groupby’ pandas method. For eg: if count, sum and weighted average are values to be calculated - create a data type of dtype for each of the 3 functions.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (2)
	+ Reset the index of dataframe (2)
	+ Identify the indexes from dataframe (2) and pass them to ‘index’ parameter in pivot\_table pandas method.
	+ Although summarization is complete - identify the summarization columns and pass them to ‘values’ parameter in pivot\_table pandas method.
	+ Pass the columns for which summarization have to be segregated to ‘columns’ parameter in pivot\_table pandas method
	+ Use numpy sum summarization function in aggfunc parameter generate a dataframe (3) with margins parameter set to false
	+ Swap the column levels of dataframe (3) if needed



#### Grand total calculation

* + Using ‘deep copy’ pandas method - copy the dataframe (from step-1) to a new dataframe (4)
	+ Identify the series corresponding to the indexes of the final pivot table from dataframe (4) and replace all the values with a constant value (eg: ‘zAll’).
	+ Slice the needed dataframe columns (from step-1) and apply summarization functions one at a time over the column fields using ‘groupby’ pandas method. For eg: if count, sum and weighted average are values to be calculated - create a data type of dtype for each of the 3 functions.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (5)
	+ Reset the index of dataframe (5)
	+ Identify the indexes from dataframe (5) and pass them to ‘index’ parameter in pivot\_table pandas method.
	+ Although summarization is complete - identify the summarization columns and pass them to ‘values’ parameter in pivot\_table pandas method.
	+ Pass the columns for which summarization have to be segregated to ‘columns’ parameter in pivot\_table pandas method
	+ Use numpy sum summarization function in aggfunc parameter generate a dataframe (6) with margins parameter set to false
	+ Swap the column levels of dataframe (6) if needed



#### Sub totals calculation

* + Identify the series corresponding to the columns by which the data needs to be grouped by
	+ Slice the needed dataframe columns (from step-1) and apply summarization functions one at a time using ‘groupby’ pandas method.
	+ Sum values can be calculated using ‘sum’ method on the grouped data
	+ Count values can be calculated using ‘count’ method on the grouped data
	+ For weighted average calculations, use numpy average function (with weights parameter) on the applicable dataframe series via apply method
	+ Convert the output generated using apply method to a dataframe using ‘DataFrame’ pandas method. Specify the column names accordingly
	+ Concatenate the dataframes generated after applying sum/count/apply methods to a single dataframe (7)
	+ Reset the index of dataframe (7)
	+ Identify the indexes from dataframe (7) and pass them to ‘index’ parameter in pivot\_table pandas method.
	+ Although summarization is complete - identify the summarization columns and pass them to ‘values’ parameter in pivot\_table pandas method.
	+ Pass the columns for which summarization have to be segregated to ‘columns’ parameter in pivot\_table pandas method
	+ Use numpy sum summarization function in aggfunc parameter generate a dataframe (8) with margins parameter set to false
	+ Swap the column levels of dataframe (8) if needed



#### Concatenate subtotal and grand total dataframes to build the final pivot table

* + Convert each of the subtotal dataframes and grand total dataframes to multi-index dataframe as below. The new dataframe should align with the dataframe (3) indexes/levels
		- Get the index values for each of the levels and then form a tuple using zip method
		- Pass the tuple list to MultiIndex.from\_tuples pandas method and build the index/dataframes
	+ Concatenate the newly formed dataframes to dataframe (2) to form a consolidated dataframe (9)
	+ Sort the indexes

#### % to calculation (calculated fields)

####

* + Get the unique values for each column levels.
	+ Percentages are calculated by iterating over the appropriate column level and using ‘iloc’ method and division operators. The percentages will appear as new dataframe (3) columns.
	+ The % columns should be aptly named so that the new columns gets created adjacent to their respective summarized columns

#### Casting, formatting and cleansing the dataframe

* + Using ‘astype’ pandas method, each of the dataframe (8) series/columns are casted to the required data types.
	+ Using ‘apply’ pandas method and format python method, $ can be prefixed, numbers can be comma separated etc.., on the applicable dataframe (8) series.
	+ Any unwanted rows/columns in the dataframe (8) can be removed using ‘drop’ pandas method



### Code excerpt







## Observations and conclusion

Following are the observations when using Python to generate pivot tables

1. If the data is prone to change each time the pivot table is to be generated then - choosing Python over manual method results in quicker time to Market

Viz.,

* If multiple sub-pivot table views are to be generated (from parent pivot table) based on the varying data (frequently) - then using Python comes handy as it eliminates the laborious effort involved with the manual generation
* If # of pivot table columns varies depending upon the data and if the Pivot table involves weighted or any other calculated columns then using Python makes the generation so simpler and easier than to the manual one.
* Excel stylings like applying borders, font variations, color highlights/fills, pattern fills can be quickly applied to the pivot tables than the manual method
1. The conditions to generate pivot table can be easily input when coupled with a GUI (like tkinter) and the XLS report generated can be placed at the user preferred location