**Raw Data to Processed Data**

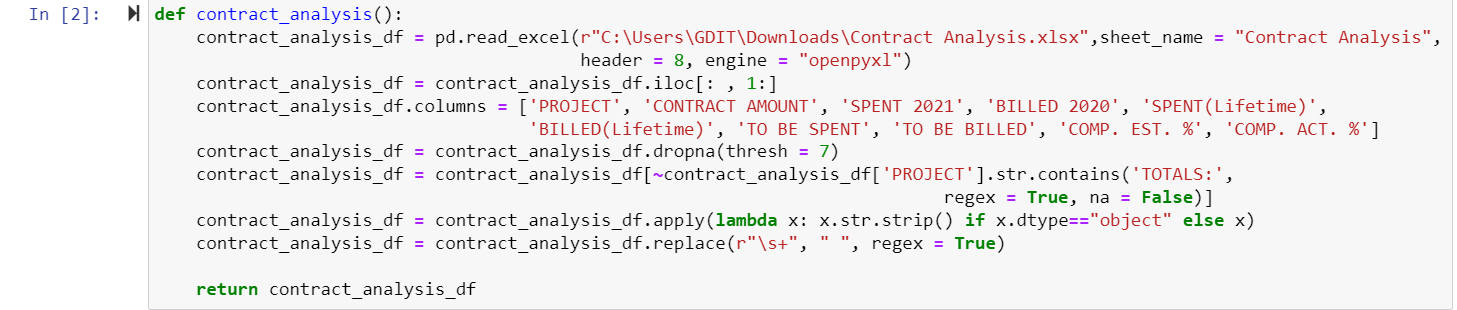
In this we read an excel file which contains raw data and perform some data processing and data cleansing operations, and then writing the processed data to a new excel file. We do this by writing the code in Jupyter Notebook and use different libraries such as NumPy, Pandas, Re and xlsxwriter.



First of all, we import the necessary libraries required to perform the different operations we want to execute. **Pandas** is a data manipulation library and is used to provide DataFrame and functions for reading, writing, and manipulating data. **Numpy** is a mathematical library used to perform mathematical functions on multidimensional arrays. **Re** is a regular expression library and provides functions and patterns for matching and manipulating data strings based on different patterns. **Warnings** is a module for handling warnings in Python.

The next library **openpyxl** is a library for reading and writing Excel files in Python. **Dateutil.parser** is a module that provides a parser for parsing dates and time strings. The **Time** library is used to calculate the time it took to execute the program. **Xlsxwriter** is a library used to create Excel files in Python. **Datetime** library is a module that provides classes for manipulating dates and times in Python. **OS** is a module for interacting with the operating system in Python. It provides functions for manipulating files, directories, and paths.

These libraries are included to provide various functionalities required for Data Analysis, date/time manipulation and reading/writing of Excel files.



In this code we first create a function named ‘contract\_analysis ()’ and we read the Excel file using the **pd.read\_excel()** function. The file is read by the specific path of the file. The sheet named “Contract Analysis” is read and the header starts at row 8 which means we are skipping the first 8 rows of the excel file.

In line 2 the **.iloc** function is used to access the DataFrame by integer-based indexing. The **‘:’** before the comma specifies that we want to select all rows of the DataFrame. After the comma the **‘1:’** specifies that we want to skip the first column and start from the second column (index 1). The new modified DataFrame is then reassigned to the same variable ‘contract\_analysis\_df’.

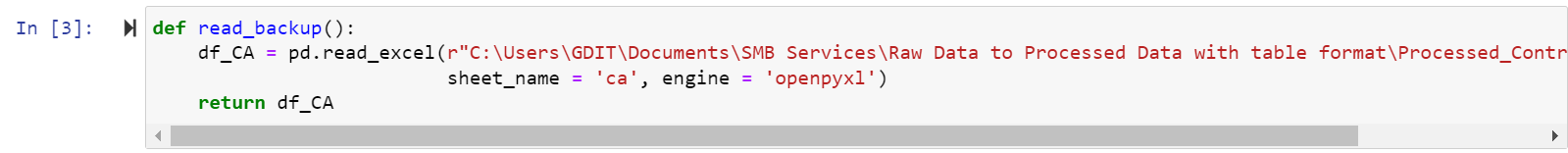
The third line assigns new column names to the DataFrame and the specific order in which we want them to appear.

In the next line we start data cleansing by first removing the non-null values and for that we use the **drop.na** function and we pass the parameter **thresh = 7** which removes rows from the DataFrame that have less than 7 non-null values.

The next line filters out rows from the DataFrame where the ‘PROJECT’ column contains the string ‘TOTALS:’. The **~** operator is used to negate the condition and then the DataFrame is then modified and only contains rows where the ‘PROJECT’ column does not contain the string ‘TOTALS:’.

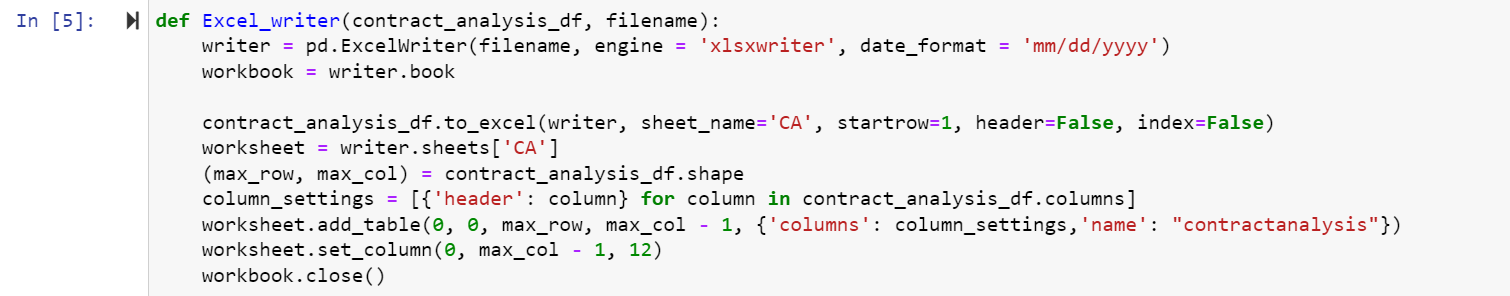
Here we apply a function to each element of the DataFrame and use a lambda function which removes whitespaces from string values. The **x.str.strip()** is the condition used where if the datatype of any value is string and for that we used “object” which means a string so if the condition is true, we trim the whitespace or else we return the original value and save it in the variable of the DataFrame.

In line 7 we replace consecutive or double whitespaces in a string with a single whitespace and for that we use a regular expression and set it to **regex = True** to find and replace the whitespace. The last line returns the modified DataFrame ‘contract\_analysis\_df’ from the function.



In the next cell we create a new function called **read\_backup()** and we create a new Dataframe and read an Excel file with the specified sheet. The reason for creating a backup excel file and storing it in a new DataFrame is to store a copy of important data or information in case the original file becomes corrupted or gets deleted. It provides an extra layer of protection against loss of data.

The last line returns the **‘df\_CA’** DataFrame as the output of the function.



First, we create a function called **Excel\_writer()** and pass the parameters **contract\_analysis\_df** and a string **filename**.

Next, we make an object called **writer** and use the function **pd.ExcelWriter()** to perform different operations on the Excel file. We pass the **filename** parameter as the Excel file to be written. The engine parameter is set to **xlsxwriter,** specifying the library to use for writing the Excel file. The date format is specified as well in the Excel file.

In the next line we make a new object **workbook** and store the **writer** object in it so that we can apply all the different functions we want to in the Excel file such as adding/removing sheets, setting workbook-level properties etc. By adding **writer.book** we can interact directly with the **workbook** object and perform various operations.

Next, we write the **contract\_analysis\_df** DataFrame to the Excel file using the **to\_excel()** function. It writes the data to the sheet name **‘ca’** in the Excel file. The **startrow** parameteris set to 1 which means that the data should start from row 2 in the Excel file since Excel row indexing starts from 1. The **header** parameter is set to **‘False’** which means that the column headers should not be included in the Excel File. The **index** column is set to **‘False’** specifying that the index column should not be included in the Excel file.

Next, we give a reference ‘**ca’** from the **writer** object and assign it to the **worksheet** variable which means that the functions will be applied on this specific sheet.

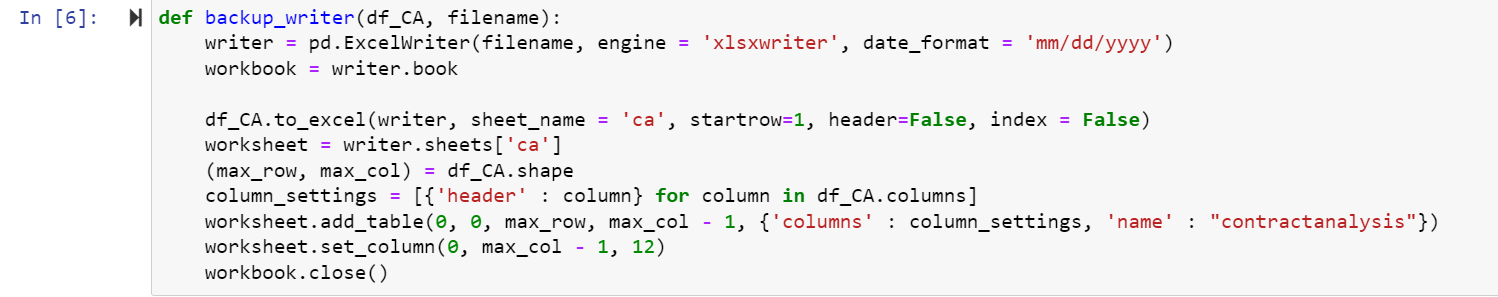
The **.shape** attribute of a pandas DataFrame tells us the number of rows and columns in an Excel file. By passing the parameters **‘max\_row’** and **‘max\_col’** the values of rows and columns are assigned to these variables and allows us to access and use them in the DataFrame.

Next, we create a variable **column\_settings** and this creates a list of dictionaries. Each dictionary has a single key value pair where the key is **‘header’** and the value is the column label itself. This represents the header name for a column in the Excel table. The list comprehension iterates over each column in **contract\_analysis\_df.columns.**

In the next line of code, the **add.table()** method is used on **worksheet** object. It allows you to create formatted tables within a worksheet in the excel file. The first two parameter **‘0, 0’** represent the starting cell of the table A1 which is the top left corner of the table. The third parameter **‘max\_row’** represents the maximum number of rows in the table. The fourth parameter **‘max\_col -1’** represents the maximum number of columns in the table minus one, we are skipping the first index of the column. The fifth parameter is a dictionary that contains additional settings for the table. The **‘columns’ : column\_settings** specify the column definitions for the table. The **name** function sets the name of the table to **“contractanalysis”**.

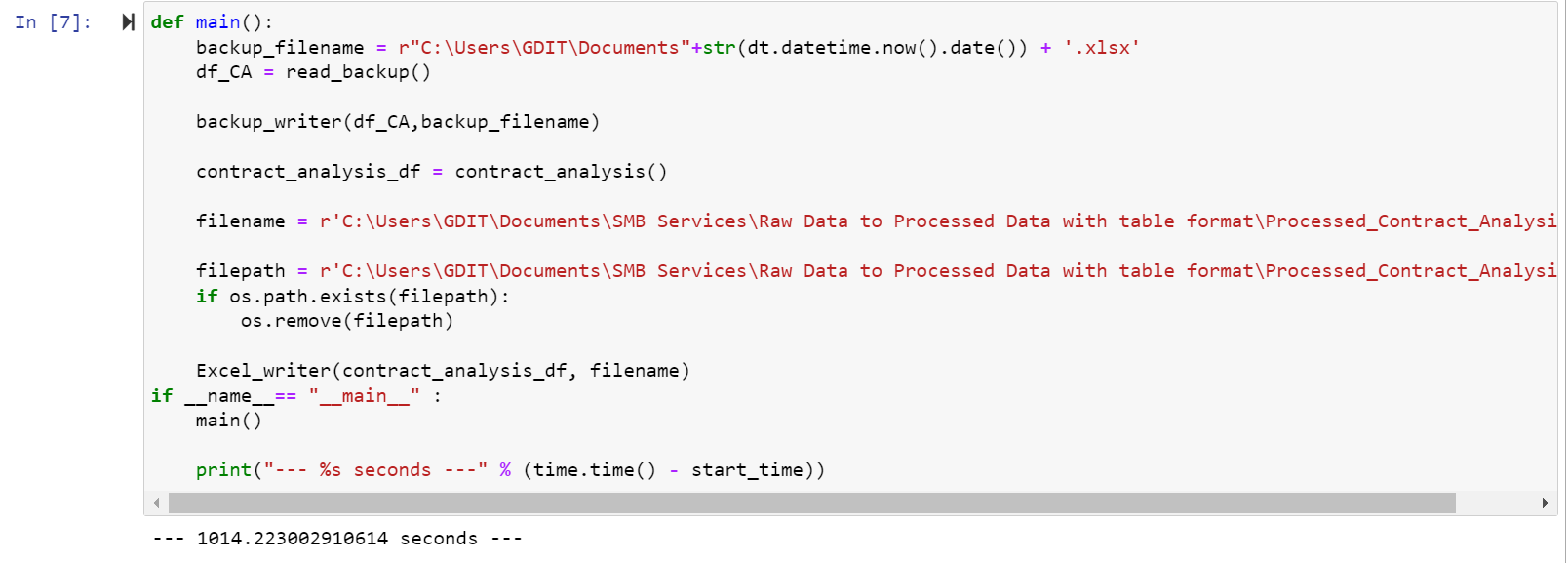
Next, we set the width of the columns in the Excel file and for that we use the **.set\_column()** function on the worksheet variable. The first parameter **‘0’** represents that the column starts from column A which is the left topmost column. The second parameter **‘max\_col -1’** specifies the ending column index for which the width will be set. The third parameter ‘12’ represents the width of the columns in the unit of characters.

The last line closes the workbook and ensures that the modified Excel File is properly saved.



Next, we create a function for the backup Excel file and pass the parameters ‘**df\_CA’** and the string **‘filename’**. We perform the same operations on the backup Excel such as reading it first and then storing it in the ‘**workbook’** variable. Then we write the **’df\_CA’** DataFrame to a new excel sheet and pass the same parameters and the same Excel sheet we want to perform operations on.

Next, we assign the maximum rows and columns of the Excel file to the new variables which we create and apply the necessary formatting we want apply on the columns. Then we create a new table in the Excel file and then save the workbook. This is done to store the copy of the data in a backup file incase there is loss of data or data corruption.



In the last cell of code, we make another function called the **main** function and we create a backup filename by concatenating the directory path of the file, current date(it is obtained using the **‘dt.datetime.now().date()+ ‘xlsx’’.** By using the current date in the backup filename, it helps ensure that each backup file has a unique name based on the date when the program is run.

The **’df\_CA = read\_backup()’** line calls the **‘read\_backup’** function to read the backup file from the specified Excel file and store it in the DataFrame **‘df\_CA’**.

The **‘backup\_writer(df\_CA, backup\_filename)’** calls the **‘backup\_writer()’** function to write the data from the DataFrame **‘df\_CA’** into the backup Excel File specified by **‘backup\_filename’**.

In the next line we call the function **‘contract\_analysis()’** and assign its return value to the **‘contract\_analysis\_df’** DataFrame.

Next, we assign the string filename with the path of the final processed data Excel file.

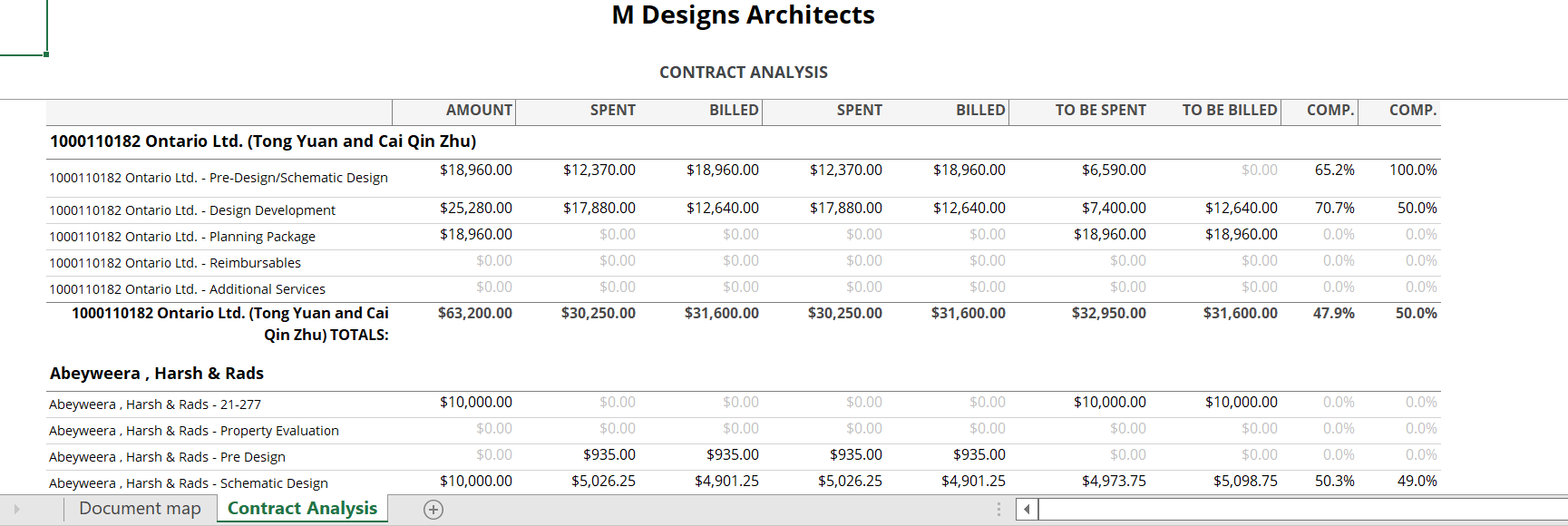
The next line assigns the filepath of the final processed data Excel file.

The next line of code checks if the file specified by **‘filepath’** exists or not using the **‘os.path.exists()’** function and then remove the file specified by **‘filepath’** using the **‘os.remove()’**. If the file exists it is deleted before creating a new one.

Then we call the **‘Excel\_writer()’** function to write the data from the **‘contract\_analysis\_df’** DataFrame into the final processed file named **‘filename’**.

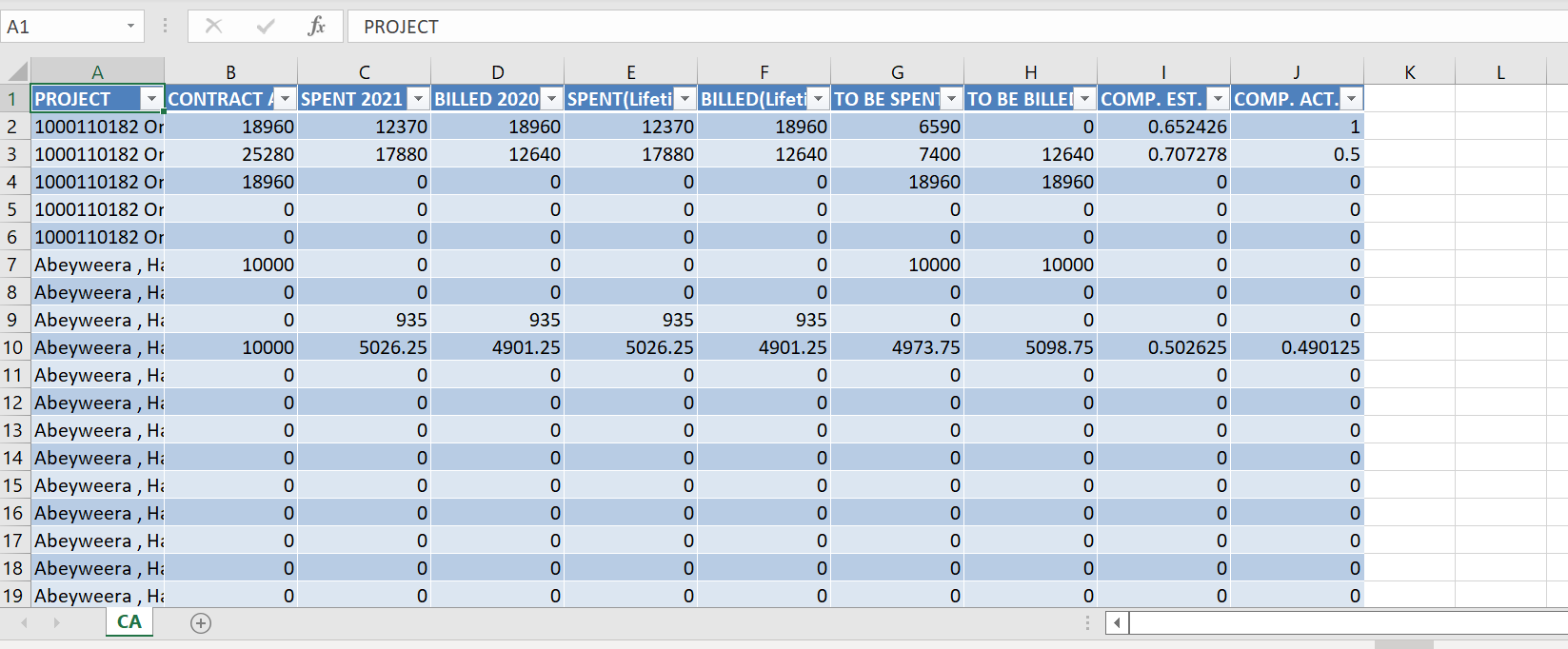
After that we call the main function which serves as the entry point of the program execution. The **‘main’** function is responsible for executing the operations to create a backup of the data, perform data cleansing, remove the existing processed data file, and generate a new formatted data processed file. The time it took for the program to execute is also printed at the end of the program.

**Raw Data File:**

****

This is a small snippet of the raw data file which was not formatted.

**Processed Data File:**

****

This the New Processed data file which has undergone the necessary operations and table formatting.