Introduction

This report discusses the steps taken to clean an Excel dataset published by the United Nation (UN) Population Division in 2015. The aforementioned UN dataset has information about the stock of international migrants across 232 countries. The UN dataset contains six tables with different types of migrant stock data, as well as other supplementary tables such as content, annex, and notes. Upon the initial assessment of the dataset, it has examples of violations of the "Tidy Data" principles first proposed by Hadley Wickham in his paper published in the *Journal of Statistical Software* in 2014. In his article, he discussed five principles to organize data in datasets, and ways to clean what he referred to as "messy data" – data structures that do not fit the tidy data framework (Wickham, 2014, p. 4).

To transform the existing messy UN dataset into a tidy dataset, I used Python and functions of its two libraries - Pandas and NumPy - to clean all six tables. This report will detail my approach to cleaning the tables following the order of Wickham's tidy data principles. Screenshots will be used to illustrate the steps taken; Python codes are available in the Jupyter notebook attached.

General Data Cleanliness Issues in the UN Dataset

Before delving into the discussions of tidy data, it is important to note that steps were also taken to clean stylistic issues in the dataset that may obstruct further analysis without proper cleaning. For example, all six tables in the dataset contain 14 empty rows when loaded into Python, as the original Excel file has a document header (e.g. UN logo, table name, copyright disclaimer etc.) that spreads across the top 14 rows in each table. Therefore, I used `df.iloc[]` to extract rows containing the actual data based on index positions.

Because of the document header, Python is not able to detect meaningful column headers by default. Therefore, new column headers need to be assigned to each column to indicate its attribute to avoid misinterpretation. I used the `df.rename(columns={})` function to assign a new set of column headers for each table.

Moreover, some cells (intersections of a row and a column) are populated by ".." in columns where numerical values are expected. Such issue requires cleanring as the symbol messes up Python's interpretation of values in that column, as the programming language recognizes it as a string. The mix of strings and numerical values in one column bars operations such as `nlargest()`, which was used to extract high-level insights. Therefore, I used `df.replace()` to replace ".." to "nan", which means "not a number" in Numpy, then converted the data type of relevant columns from object to float. I chose not to replace the ".." values with zeros as they refer to different meanings: ".." could be interpreted as "not available" or "no record" but zero means no such observation at all.

Tidy Data Issue 1: Column Headers Are Values, Not Variable Names

Wickham (2014) argues that column headers should be used to store variables, which are "underlying attribute[s]" that denote what the data represent (p. 3). Subsequently, values that are observations, meaning that they represent measurements of the same unit should be stored as rows (Wickham, 2014, p. 3). I will use Table 4 in the UN dataset to illustrate this violation in this report, but the same issue also appears in other tables in the dataset.

Table 4 records the percentage of female migrants out of the international migrant stock from 1990 to 2015. This table contains three main attributes: geographical region, year, and

percentage of female migrant. However, in the table, values of the attribute “Year” (e.g., 1990, 1995……2015) are stored across multiple column headers. That means instead of having all values listed under one column, they are scattered across six columns. Therefore, this table does not follow a tidy data structure and transformation needs to be done to restore its tidiness.

To clean the dataset, I used the `pandas.melt()` function to unpivot all column headers with values of the attribute “Year” into rows within a single column as these values represent the same variable. As illustrated in *figure 2*, values of the attributes “Year” are now transposed from column headers into one column titled “Sex_Year”, and the percentages are now organized in another column titled “Percentage of International Migrant Stock”.

I noticed that the new column “Sex_Year” does not meet the tidy data framework, but this issue will be discussed in more detail in the next section under tidy data principle 2. Otherwise, this structure fits the tidy data principle 1 as all column headers are variable names, not values. Similar operations are done to all tables in the UN dataset as this violation of tidy data is apparent in all of them.

Figure 1: Table 4 Before `pandas.melt()` *operation*

Out[19]:

	Sort\norder	Major Area/Region/Country	Notes	Country Code	Type of Data	F_1990	F_1995	F_2000	F_2005	F_2010	F_2015
15	1	WORLD	NaN	900	NaN	49.03915	49.16879	49.112244	48.832993	48.305660	48.249769
16	2	Developed regions	(b)	901	NaN	51.123977	51.149024	51.113307	51.171501	51.658932	51.866687
17	3	Developing regions	(c)	902	NaN	46.592099	46.500135	46.128444	45.134297	43.319780	43.327078
18	4	Least developed countries	(d)	941	NaN	47.261155	47.571664	46.826689	45.157406	45.499573	45.942752
19	5	Less developed regions excluding least develop...	NaN	934	NaN	46.466684	46.279022	46.009598	45.130768	43.043672	42.984398

Figure 2: Table 4 After the `pandas.melt()` *operation*

Out[46]:

	Sort\norder	Major Area/Region/Country	Notes	Country Code	Type of Data	Sex_Year	Percentage of International Migrant Stock
0	1	WORLD	NaN	900	NaN	F_1990	49.03915
1	2	Developed regions	(b)	901	NaN	F_1990	51.123977
2	3	Developing regions	(c)	902	NaN	F_1990	46.592099
3	4	Least developed countries	(d)	941	NaN	F_1990	47.261155
4	5	Less developed regions excluding least develop...	NaN	934	NaN	F_1990	46.466684

Tidy Data Issue 2: Multiple Variables Stored In One Column

According to Wickham (2014), each variable should be in its own column (p. 4) to keep the dataset tidy. However, in the UN dataset, we can see that in Table 1, 2, 3, and 5, the attributes “Sex” and “Year” are stored in the same column headers (e.g., Male Migrant Stock in 1990, etc.). As illustrated in the previous section, after the initial step of unpivoting column headers containing values of “Sex” and “Year” into a new column, values within that new “Sex_Year” column would consist of two attributes (*figure 3*). This is a violation of the tidy data principle 2 as each column should only contain one type of variable. Table 1 will be used in this section as an example to illustrate the issue and steps taken to address that.

Figure 3: Column "Sex_Year" contains information related to two attributes

Out[48]:

	Sort/Order	Major Area/Region/Country	Notes	Country Code	Type of Data	Sex_Year	Migrant Stock
0	1	WORLD	NaN	900	NaN	M_1990	77747510
1	2	Developed regions	(b)	901	NaN	M_1990	40263397
2	3	Developing regions	(c)	902	NaN	M_1990	37484113
3	4	Least developed countries	(d)	941	NaN	M_1990	5843107
4	5	Less developed regions excluding least develop...	NaN	934	NaN	M_1990	31641006
5	6	Sub-Saharan Africa	(e)	947	NaN	M_1990	7745306

As seen in *figure 3*, values of the attribute "Sex" (e.g., M, F, Both Sexes) as well as values of the attribute "Year" (e.g. 1990, 1995, 2000……etc.) are stored in the same column "Sex_Year" and separated by an underscore. These two variables need to be split and stored in separate columns to meet the tidy data framework. I used the `str.split()` function in Python to split the two strings – values that indicates sex and year – into two separate columns. The underscore was indicated as the delimiter `"_"`, and the `expand = True` parameter in the `str.split()` function allows Python to return two columns after the split (see figure 4).

Figure 4: Values of Sex and Year are separated into two columns

Out[49]:

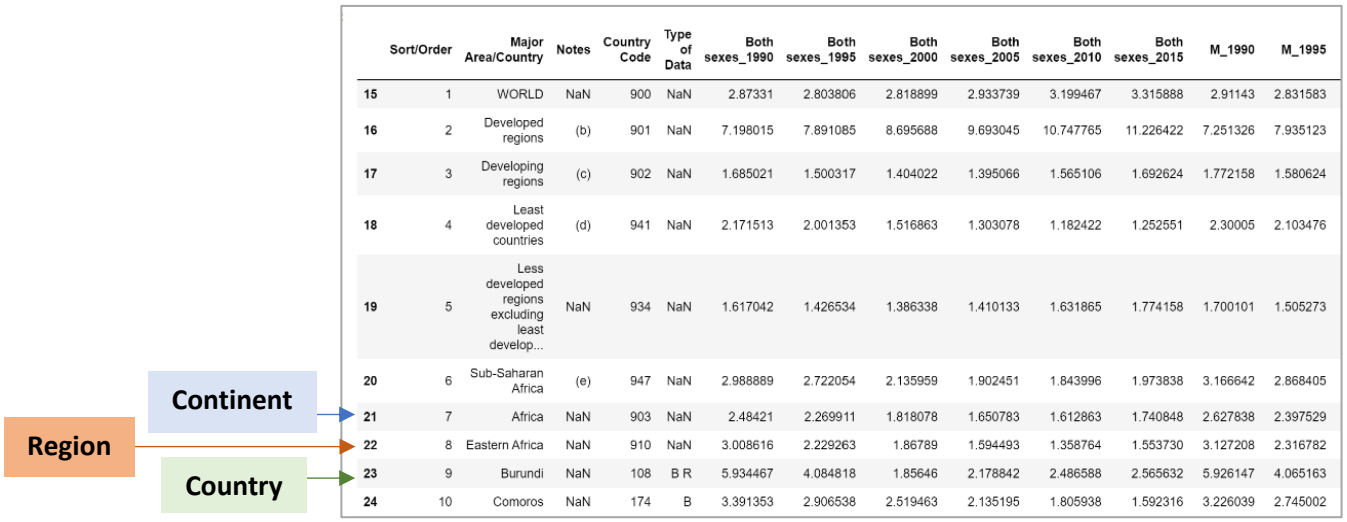
	Sort/Order	Major Area/Region/Country	Notes	Country Code	Type of Data	Migrant Stock	Sex	Year
0	1	WORLD	NaN	900	NaN	77747510	Male	1990
1	2	Developed regions	(b)	901	NaN	40263397	Male	1990
2	3	Developing regions	(c)	902	NaN	37484113	Male	1990
3	4	Least developed countries	(d)	941	NaN	5843107	Male	1990
4	5	Less developed regions excluding least develop...	NaN	934	NaN	31641006	Male	1990

After the split, I renamed the label "M", F" to "Male" and 'Female" to clearly indicate the values they represent using the `df.replace()` function. The previous column "Sex_Year" was also removed using the `df.drop()` function to improve readability of the table. The same cleaning procedures were applied to Table 2, 3, 4 and 5 to separate variables "Sex" and "Year" into two columns. *figure 4* better fits the tidy data principle as each column contains only one variable.

Tidy Data Issue 3: Variables Are Stored In Both Rows And Columns

Another issue that makes a dataset untidy is having variables stored in both rows and columns (Wickham, 2014, p. 9). As seen on *figure 5*, all tables in the UN dataset store major areas and regions, such as continent names (e.g., Africa) and regions (e.g., Eastern Africa) in the same column as the countries (e.g., Burundi). This is a violation of the tidy data principle as these three variables should be in their respective columns. Combining this issue with the fact that year and gender values are being stored in the column headers, important information is scattered across rows and columns in the data table, making it hard to analyze computationally (see *figure 5)*. Table 3 is used in this section as an example to illustrate the steps taken to clean the messy data, and the same steps are applied to clean all tables in the UN dataset.

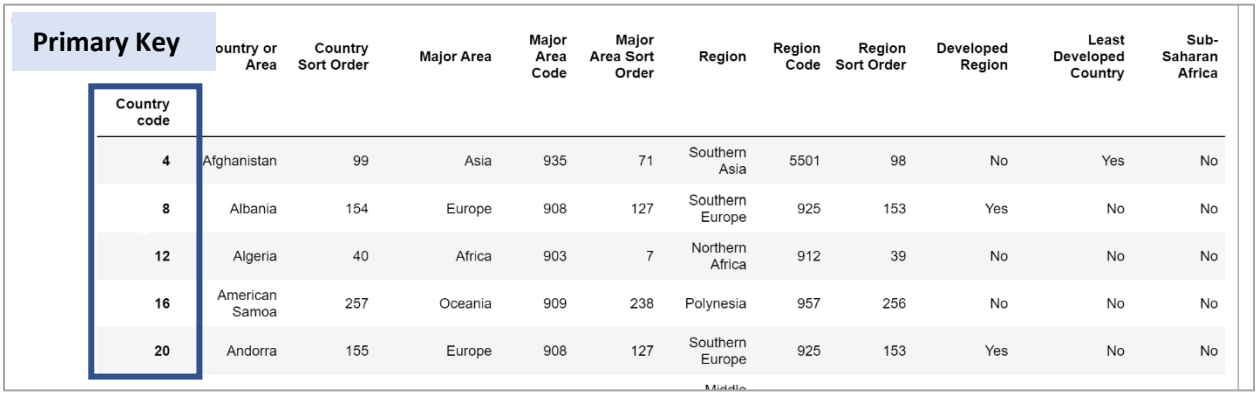
Figure 5: Continents and region are stored in the same column as countries in Table 3



	Sort/Order	Major Area/Country	Notes	Country Code	Type of Data	Both sexes_1990	Both sexes_1995	Both sexes_2000	Both sexes_2005	Both sexes_2010	Both sexes_2015	M_1990	M_1995
15	1	WORLD	NaN	900	NaN	2.87331	2.803806	2.818899	2.933739	3.199467	3.315888	2.91143	2.831583
16	2	Developed regions	(b)	901	NaN	7.198015	7.891085	8.695688	9.693045	10.747765	11.226422	7.251326	7.935123
17	3	Developing regions	(c)	902	NaN	1.685021	1.500317	1.404022	1.395066	1.565106	1.692624	1.772158	1.580624
18	4	Least developed countries	(d)	941	NaN	2.171513	2.001353	1.516863	1.303078	1.182422	1.252551	2.30005	2.103476
19	5	Less developed regions excluding least develop...	NaN	934	NaN	1.617042	1.426534	1.386338	1.410133	1.631865	1.774158	1.700101	1.505273
20	6	Sub-Saharan Africa	(e)	947	NaN	2.988889	2.722054	2.135959	1.902451	1.843996	1.973838	3.166642	2.868405
21	7	Africa	NaN	903	NaN	2.48421	2.269911	1.818078	1.650783	1.612863	1.740848	2.627838	2.397529
22	8	Eastern Africa	NaN	910	NaN	3.008616	2.229263	1.86789	1.594493	1.358764	1.553730	3.127208	2.316782
23	9	Burundi	NaN	108	B R	5.934467	4.084818	1.85646	2.178842	2.486588	2.565632	5.926147	4.065163
24	10	Comoros	NaN	174	B	3.391353	2.906538	2.519463	2.135195	1.805938	1.592316	3.226039	2.745002

First, I used `pandas.melt()` to unpivot the “Year” and “Sex” values in the existing column headers and used the `str.split()` to separate values of each attribute to two columns: “Year” and “Sex”. The next step was to separate continents, regions, and countries into three separate columns. There are many ways to do so, but I chose to make use of the geographical information available in the annex table as that would be the simplest and fastest way. As seen in *figure 6*, the annex table follows the tidy data structure where continents, regions, and countries are organized in three separate columns. This is exactly the structure I would like to implement in all data tables in the UN dataset. Thus, I decided to perform an inner join operation using Python to merge both tables, achieving the ideal tidy data structure.

Figure 6: The annex table follows the tidy data structure where each geographical variable is stored in its own column



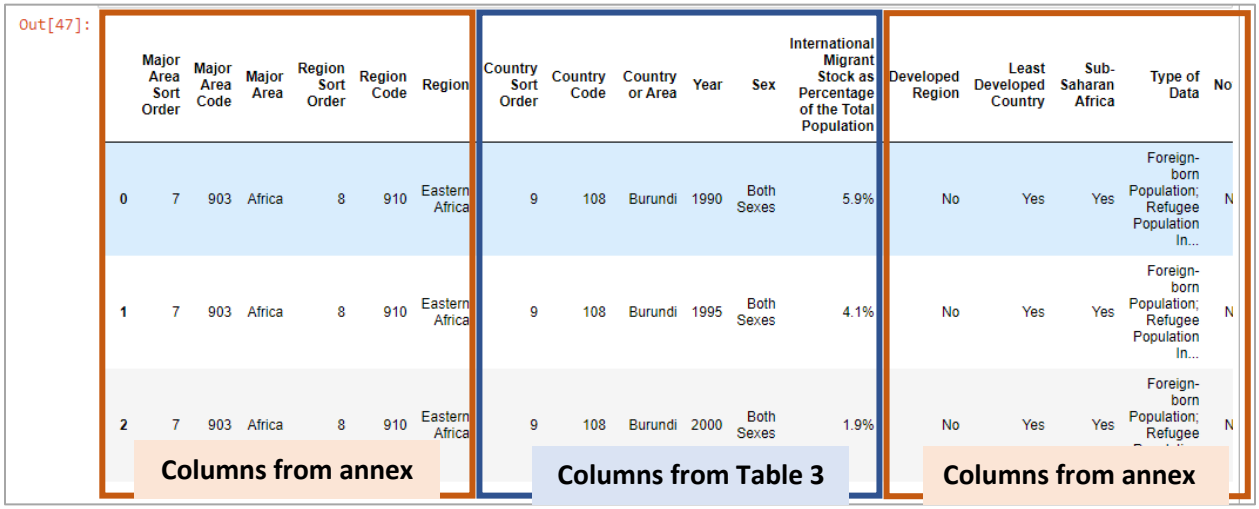
Country code	ountry or Area	Country Sort Order	Major Area	Major Area Code	Major Area Sort Order	Region	Region Code	Region Sort Order	Developed Region	Least Developed Country	Sub-Saharan Africa
4	Afghanistan	99	Asia	935	71	Southern Asia	5501	98	No	Yes	No
8	Albania	154	Europe	908	127	Southern Europe	925	153	Yes	No	No
12	Algeria	40	Africa	903	7	Northern Africa	912	39	No	No	No
16	American Samoa	257	Oceania	909	238	Polynesia	957	256	No	No	No
20	Andorra	155	Europe	908	127	Southern Europe	925	153	Yes	No	No

Inner join is a method that combines two tables though a common key organized in a column and returns a new table that only keeps matching records from both sides. I recognized that each country has its unique country code in the UN dataset, and this country code is present in the annex table and all data tables. I decided to make the country code as the index of the annex table using the `df.set_index()` function. Then, I performed an inner join using the

`df.join(how = inner)` function, with country code in the annex table as the primary key, and the country code column in Table 3 as the foreign key.

Following the inner join, only rows with country codes that are present in the “Country Code” column of both Table 3 and the annex would be kept in the new table. Rows containing the continents and regions are removed in the process from the country column, as their codes are not country codes present in the “Country Code” column of both tables. Still, these geographical regions are being retained in the new table following the join. It is because the inner join has brought in columns containing information on “Major Area” (continents) and “Region”, and other geographical information (e.g. Developed Region) from the annex table, as shown in figure 7.

Figure 7: Newly joined Table 3 with geographical information from the annex table



	Major Area Sort Order	Major Area Code	Major Area	Region Sort Order	Region Code	Region	Country Sort Order	Country Code	Country or Area	Year	Sex	International Migrant Stock as Percentage of the Total Population	Developed Region	Least Developed Country	Sub-Saharan Africa	Type of Data	No
0	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	1990	Both Sexes	5.9%	No	Yes	Yes	Foreign-born Population; Refugee Population In...	N
1	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	1995	Both Sexes	4.1%	No	Yes	Yes	Foreign-born Population; Refugee Population In...	N
2	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	2000	Both Sexes	1.9%	No	Yes	Yes	Foreign-born Population; Refugee	N

The newly joined Table 3 fits the tidy data principles. Different geographical information is organized in respective columns. Other variables such as “Sex” and “International Migrant Stock as Percentage of Total Population”, are structured as individual columns as well. This operation was applied to all tables in the UN dataset, separating countries, continents, and regions into different columns.

Tidy Data Issue 4: Multiple Types in One Table

The last tidy data violation found in the UN dataset is storing multiple data types in one table (Wickham, 2014, p. 11). This is only evident in Table 6 of the UN dataset, where it documents three different variables, including “Estimated refugees stock at mid-year (both sexes)”, “Refugees as percentage of the international migrant stock”, and “Annual rate of change of the refugee stock”. These three variables have different observational units, which are number, percentage, and rate of change respectively. Wickham (2014) argues that “each type of observational unit should be stored in its own table” (p. 11), hence, having different observational units stored in a single table is a clear violation of the tidy data framework.

Therefore, to clean the UN dataset, I split Table 6 into three datasets, one for each variable. Following the steps discussed in previous sections to make the data tidy, I used `pandas.melt()` to unpivot column headers that contain values into the: “Indicator_Year” and values they follow into the “Value” column. The first column contains the type of variable and the year, and the second column contains the actual value of the data (i.e., number of refugees,

percentage, or rate of change). Then, I split the indicator and year into two separate columns, followed by an inner join operation to separate continents and regions into their respective columns, as discussed in the preceding section.

Figure 8: Three variables and values with different obervational units are stored in Table 6

	Major Area Sort Order	Major Area Code	Major Area	Region Sort Order	Region Code	Region	Country Sort Order	Country Code	Country or Area	Year	Indicator	Value	Developed Region	Least Developed Country	Sub-Saharan Africa
0	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	1990	MYStock	267929	No	Yes	Yes
6	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	1990	RefugeesPercent	80.43259	No	Yes	Yes
12	7	903	Africa	8	910	Eastern Africa	9	108	Burundi	90-95	ARChange	-3.390926	No	Yes	Yes

As seen in figure 8, three variables are stored in the same column “Indicator”, and their respective values with different observational units are stored in the column “Value”. This structure is cleaner than the original table which enables me to separate the table into three datasets based on the value in the “Indicator” column. I selected rows that record each indicator and put them in a new table using this Python code: `df_IndicatorX = df[df[“Indicator”] == “Indicator_X”]`. After doing the same operation for each of the three indicators, observations that belong to each indicator are stored in separate tables, which fits the tidy data principles Wickham (2014) proposes.

Discussion & Other Considerations

Following the cleaning of the data, I decided to do some high-level analysis of the data to understand how a tidy dataset would aid in the analysis. I soon learned that having the right data type is very important as mathematical Python operations such as `nlargest()` or `nsmallest()` only work with values that are integers or floats. I decided to go back and change the data type in my code to convert values such as count, rate of change, and percentage from the Python data type “object” to “float”. I did not do this step until I realized the severity and implications of the issue, and therefore I took an interactive approach to go back and add changing data type to my codes. This process confirms what we have been learning throughout the entire semester that data science is an iterative process. The data cleaning process could, and should be re-visited, if the data were not in a state that would enable accurate analysis computationally.

There are two things I would like to achieve but was not able to in this mid-term project. First, there are opportunities for speeding up repetitive tasks with building custom functions in Python. For example, doing an inner join with the annex table is a consistent step that needs to be done for all six tables, but I was doing it one by one in this assignment. I will explore the use of

functions in Python in the next assignment as I get more familiar with the programming language with continuous practice.

Second, I am unsure of how to best handle the “Type of Data” and “Notes” columns that contain important information about the data. In this assignment, I re-coded the abbreviations in “Type of Data” column with the actual labels to indicate what kind of migrant demographics the count/percentage has included. I have largely left these two columns unchanged as I am unsure of the best way to handle them. I deliberately chose not to delete them because I recognized that this information is important in helping others understand the data.

Conclusion

The UN dataset in its original format violates four tidy data principles proposed by Wickham (2014). Key functions in the Pandas library such as `pd.melt` and `str.split()` were used to restructure the schema of the tables and split multiple variables into their respective columns. Referencing other tables containing information about the data set, such as the annex table, I was able to separate continents and regions from the countries’ column and into two individual columns respectively. Thus, I was able to conduct a high-level analysis in the “Discussion” section to unpack information related to international migrants more precisely at a country level. Additional data analysis and visualizations are required to fully capitalize the richness of the UN dataset, and the tidy UN dataset will allow for more effective and accurate ones. In terms of improvement to the current workflow, I aim to build functions in Python to complete repetitive steps to build visualizations in the next project.

Reference

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*(10).