Yang Xuan (Student ID: 1003907427)
INF 1340
2022/11/04
Midterm Data Cleaning Project Writeup

1. Introduction

In Human-centered Data Science, data cleaning and wrangling is one of the most critical processes in any data science project. The cleaning process is crucial because data scientists make decisive judgment calls during each processing stage of the operation, in which different methods would be imposed onto the dataset. To make the final data product simple and straightforward enough for subsequent use, data cleaning would provide a helpful data package that is in a usable shape.

Typically, data cleaning aims to create a data frame that refers to "tidy data," which is commonly perceived by data scientists as following the "Five Principles of Tidy Data ."The five principles lay a framework that guides toward an ideal final result of any potential dataset. In this midterm project, the projected dataset that we are cleaning is the "United Nation Migrant Stock (1990-2015)" spreadsheet. The original EXCEL sheet contains nine sheets, including six tables (Table 1 to Table 6) with data information and three with only contextual information (CONTENTS, ANNEX, and NOTES). To clean the dataset, I applied the Five Principles of Tidy Data to guide through my thinking process of identifying problems across the six sheets. I primarily used the '*pandas*' function, *for loop*, *if* function, *process_table* function, *lambda* function, and *merge* function. After the clearing, I combined the six tables into one comprehensive table. With additional judgments of methods, the following report will be divided into three sections: Methodology and Results, Discussion, and Conclusion.

2. Methodology and results

2.1 Importing and Data Preparation

Before I conducted the cleaning process, I prepared the dataset at the stage of importing the spreadsheets into the Google Colab Notebook platform. Using the '*pandas*' function, instead of the entire spreadsheet, I read multiple sheets from the EXCEL sheet into a dictionary and used rows 14 and 15 as the header of the variables (columns). As for now, the data information was stored in the dictionary, meaning I could not call or command the table to do any coding function. Therefore, I created six separate data frames from the dictionary using *pandas.df.get* and named each data frame df1, df2, df3, df4, df5, and df6. Getting the six tables (instead of operating on the large spreadsheet) would make it easier for me to access each row, column, and cell to conduct the following steps.

```
# Read multiple sheet from UNdata excel sheet into a dictionary, read-in line 14 and 15 as the header of the row
UNdata_dict_df = pd.read_excel("UN_MigrantStockTotal_2015.xlsx",
                       sheet_name=['Table 1', 'Table 2', 'Table 3', 'Table 4', 'Table 5', 'Table 6'],
                       header = [14,15])
```

2.2 Problem Identification and Resolution

After data preparation, I printed out each table to observe the structure of the data frame and identify problems. By observation, each table shares a similar pattern in that the first five columns contain the same information, including '*Sort/order,' 'Major area, region, country or area of destination,' 'Notes,' 'Country code,' and 'Type of data (a)* .' Other columns include the specific data information regarding international migrant stock in absolute value and percentage terms. When I imported the six tables into Google Colab, I read rows 14 and 15 as the header for the variables. As a result, the combination of rows 14 and 15 became the column names. The 'unnamed' columns will be adjusted with appropriate methods.

	Sort\norder	Major area, region, country or area of destination	Notes	Country code	Type of data (a)	International migrant stock at mid-year (both sexes)					...	International migra (male)	
	Unnamed: 0_level_1	Unnamed: 1_level_1	Unnamed: 2_level_1	Unnamed: 3_level_1	Unnamed: 4_level_1	1990	1995	2000	2005	2010	...	2000	2005
0	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243	...	87884839	97866674
1	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325	...	50536796	57217777
2	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918	...	37348043	40648897
3	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128	...	5361902	5383009
4	5	Less developed regions excluding least develop...	NaN	934	NaN	59105261	56778501	59244124	64272611	79130668	...	31986141	35265888
5	6	Sub-Saharan Africa	(e)	947	NaN	14690319	15324570	13716539	13951086	15496764	...	7210452	7444048
6	7	Africa	NaN	903	NaN	15690623	16352814	14800306	15191146	16840014	...	7856358	8231437
7	8	Eastern Africa	NaN	910	NaN	5964031	5022742	4844795	4745792	4657063	...	2480584	2529460
8	9	Burundi	NaN	108	B R	333110	254853	125628	172874	235259	...	61094	84805

2.2.1 Problem 1: Define Table Function for Repetitive Cleaning

As I mentioned in the previous section, six similar-structured sheets contain the same first seven columns. Therefore, I defined a process table using the *'def'* function to store each cleaning step I made. The first step that was stored in the process table was the action of dropping 'Sort/n order' and 'Notes.' Since *Sort order* and *Notes* do not contain information that is directly useful for data analysis, I will drop both columns using the *pandas.dataframe.drop* function as I move to the next cleaning stage.

```
def process_table(df):
  # Drop sort/order, notes
  df = df.drop(["Sort\norder", "Notes"], axis=1)
```

2.2.2 Problem 2: The area of Destination and Major Regions reside in the Same Column (Principle 2)

By observing the data frame, the first problem I noticed is that the column "Major area, region, country or area of destination" contains more than one variable within one column. By reading the country code, we learned that significant areas (e.g., developed regions, developing regions, Africa, Eastern Africa, etc.) were coded with numbers that were larger than or equal to 900. Specific countries were coded with numbers that were smaller than 900 (e.g., Burundi, Canada, United States of America, etc.). Essentially, the level of region and country should be two variables, whereas "major regions" was used to specify where the part of each country belongs. As the two variables serve different purposes, it was not appropriate to put them in the same column.

Principle 2 of Tidy Data states, "*Multiple variables should not be stored in one column* ."Hence, I created a new column, "Regions," to store the information separately. I created three empty columns: *last_region*, Regions, and kept_rows. "Last_region" refers to the temporary storage of regions with country codes greater than or equal to 900. "Regions" refers to the actual column for major regions*. "Kept_rows" would return the countries with country codes smaller than 900 to the original row*.

After I defined the columns, I coded a *for loop* to sort the process to separate the variables. For rows (each observational unit) in data frame 1, if the country code of the observation is greater than or equal to 900, then the observation would be stored to last_region. Otherwise, the observational unit would go to *kept_rows*, where the countries (country code <900) will be stored. The process will then continue looping until the last observational unit (row 232) is sorted by the *for a loop*.

After I sorted out major regions and countries based on the country codes, I used the *df.append* function to aggregate the sorting results back to the columns I wanted to keep in the original sheets by filling the column with the sorted information. For countries with a country code smaller than 900 (*kept_rows*), I used the *df.iloc* function to locate the rows. For regions with a country code greater than 900 (*last_region*), I replaced the content with data that were stored within the variable of *regions*. Hence, observations of last_region were stored to regions and observations of kept_rows were stored back to 'Major area, region, country or area of destination.'

At this point, I have successfully created a separate column called 'region' and kept the observational units with country codes less than 900 within the original column. Each row has a designated column with the region it is in.

```
# Remove regions that are not a country, and add the region to be a field in it
last_region = None
regions = []
kept_rows = []
for row in df1.iterrows():
  if row[1]["Country code"]["Unnamed: 3_level_1"] >= 900:
    last_region = row[1]["Major area, region, country or area of destination"]["Unnamed: 1_level_1"]
    continue
  regions.append(last_region)
  kept_rows.append(row[0])
df = df.iloc[kept_rows]
df.insert(0, "Regions", regions, allow_duplicates=True)
```

2.2.3 Problem 3: Improper Column Name and Flatten

The next problem I tackled was correcting each table's column header. When I imported the EXCEL sheets, I read both row 14 and 15 as the column headers. As a result, several column headers would include two names, such as 'Sort\norder, Unnamed: 0_level_1'. Therefore, I used the df.rename function to assign new column headers to each variable to make the header make more sense.

First, I used *df.rename* to delete the three irrelevant 'Unnamed' headers with a blank space *("Unnamed: 1_level_1": " ", "Unnamed: 3_level_1": " ",* and *"Unnamed: 4_level_1": " ")*. Then, I renamed the column "Major area, region, country or area of destination" to "Area of Destination", which directly spoke to the data information stored within the column. Last but not least, I need to combine the headers for 'year' and the statistics of international migrant stock into new header names. To do that, I used the *df.column[.join]* function to merge the two parts. The result from this function will be applied across the six tables.

```
# Column rename and flatten
df.rename(columns={"Unnamed: 1_level_1": "", "Unnamed: 3_level_1" :"", "Unnamed: 4_level_1":""}, inplace=True)
df.columns = [''.join([str(v) for v in col]) for col in df.columns]
df.rename(columns={"Major area, region, country or area of destination":"Area of Destination"}, inplace=True)
```

2.2.4 Problem 4: Missing Values

By observing the six tables, I spotted missing values in all six tables. The missing values were represented by '..' in all tables. The missing values are the population in Table 1 and Table 2. From Table 3 to Table 6, the missing values are the population percentage. I would replace the missing values with 0. I replaced missing values with 0 because there was valid data being stored in the same row where the missing values appeared. I did not want to drop the rows with missing values since other data might be valuable in the data set. Hence, I coded a *lambda* function to replace the missing values with 0. The *lambda* function was embedded within for a loop. For the '..' in each column, the lambda function defines that the missing values will be replaced with 0, whereas (else) the rest data will remain in the same column and cell. Then the last step was to return the results in the original data frame.

```
# Define '..' as missing value, replace missing value with 0, apply the function across 6 tables
for column in df.columns:
  df[column] = df[column].apply(lambda x : x if x != ".." else 0)
return df
```

2.2.5 Problem 5: Applying Changes across 6 Tables with Process Table

The next problem is applying the changes I made to Table 1 across the rest of the five tables. At the beginning of the data cleaning process, I intentionally used the *def* function to define a 'process table', which preserved a complete cycle of the steps I coded to clean one table. In summary, the process table included 1) dropping 'Sort/norder' and 'Notes' column; 2) removing regions that are not a country and adding a column only including regions; 3) renaming columns and flattening the columns; and 4) defining '..' as missing values and replacing them with 0.

To apply the changes across the six tables, I saved each table into a new data frame (*dfn_process = process_rable(dfn)*). Ideally, the steps stored in the process table should be applied to all six tables, yet the changes should also work effectively. By looking at each table's columns, the table's conversion was successfully executed. The 'Area of Destination' and 'Region' columns were separated for all tables with a precise match. Also, the total population and the percentage of the population in each table were properly matched with the year. Thus, they became new headers for each column. Each table contained 232 observational units and 22 columns.

Regions	Area of Destination	Country code	Type of data (a)	International migrant stock at mid-year (both sexes)1990
Eastern Africa	Burundi	108	B R	333110
Eastern Africa	Comoros	174	B	14079
Eastern Africa	Djibouti	262	B R	122221
Eastern Africa	Eritrea	232	I	11848
Eastern Africa	Ethiopia	231	B R	1155390
Eastern Africa	Kenya	404	B R	297292
Eastern Africa	Madagascar	450	C	23917
Eastern Africa	Malawi	454	B R	1127724
Eastern Africa	Mauritius	480	C	3613

2.2.6 Problem 6: Merging 6 Tables into 1 (Principle 5)

The last problem is merging the six tables into one data frame. The 5th principle of tidy data indicates that "*A single observational unit should not be stored in multiple tables* ."The observational unit in this data set is each row, which refers to each country (and its relevant information). While the first three columns for each table remained the same, the fourth (and beyond) contained unique data on the international migrant stock each year. Therefore, I would preserve "Area of Destination," "Country code," and "Type of data (a)" as the first three rows for the merged table and add the rest of the columns from each table to the merged table. To merge the tables, I used the *reduce()* function, a *lambda* function, and *pd.merge* function to perform the merge operation. The new merged table was named "*merged_df* ."By printing out the shape of *merged_df*, the new table contained 232 observational units (the same as the separated tables) and 100 columns.

	Regions	Area of Destination	Country code	Type of data (a)_x	International migrant stock at mid-year (both sexes)1990	International migrant stock at mid-year (both sexes)1995	International migrant stock at mid-year (both sexes)2000	International migrant stock at mid-year (both sexes)2005	International migrant stock at mid-year (both sexes)2010	International migrant stock at mid-year (both sexes)2015
0	Eastern Africa	Burundi	108	B R	333110	254853	125628	172874	235259	286810
1	Eastern Africa	Comoros	174	B	14079	13939	13799	13209	12618	12555
2	Eastern Africa	Djibouti	262	B R	122221	99774	100507	92091	101575	112351
3	Eastern Africa	Eritrea	232	I	11848	12400	12952	14314	15676	15941
4	Eastern Africa	Ethiopia	231	B R	1155390	806904	611384	514242	567720	1072949
...	...	...	...	...	...	...	...	...	...	...
227	Polynesia	Samoa	882	B	3357	4694	5998	5746	5122	4929
228	Polynesia	Tokelau	772	B	270	266	262	258	429	487
229	Polynesia	Tonga	776	B	2911	3274	3684	4301	5022	5731
230	Polynesia	Tuvalu	798	C	318	263	217	183	154	141
231	Polynesia	Wallis and Futuna Islands	876	B	1402	1680	2015	2365	2776	2849

3. Discussion

The United Nations "Trends in International Migrant Stock" dataset contains the relevant data on the international migrant population from 1990 to 2015. The tables included valuable statistics regarding the trends in global stock. From the scope of the content of the data, it is reasonable to say that the data set is content-rich because each table is interconnected. Table 1 includes the international migrant stock at mid-year by sex and by major area from 1990 to 2015. Table 2 consists of the total population (in thousands) at mid-year by sex and by major area from 1990 to 2015. Table 3 is a derivative product of Table 1 and Table 2 because the international migrant stock as a percentage is calculated using Table 1 divided by Table 2.

Similarly, Table 4 (female migrants as a percentage of the international migrant stock by major area, 1990-2015) is calculated using the female migrants divided by international migrants (data all stored in Table 1).

The challenge that I faced when I was cleaning the data set was how to split the "major area" column. The header contains two parts of different information. I wanted to match the country with the region it belonged to. Therefore, I tested various methods, and the *for* loop worked perfectly. However, the cleaning process was based on my intuition and used the "tidy data principles'' as general guidelines. There was no specific purpose for the dataset to be used after cleaning (e.g., data visualization). The cleaning process would be more directional and straightforward if I had the context of the data set.

4. Conclusion

In conclusion, the project aims to clean the "United Nation Migrant Stock (1990-2015)" spreadsheet. In the clearing process, I used the "5 Principles of Tidy Data'' as the guideline for creating a tidy data frame. I made a process table to record each step I took to clean the first sheet (Table 1). Primarily, I separated the column of "Major area, region, country or area of destination" into "Area of Destination" and "Region ."I merged the headers of the columns (rows 14 and 15) for the statistical part of the table. Then, I replaced the missing values '..' with 0. Lastly, I applied all changes across the six tables by saving the processed table into new ones. After all the changes had been made, I combined the six tables as a comprehensive one. The final product looks neat and clean, consisting of 232 observational units and 100 columns. The general cleaning was challenging because there were difficulties in the process. As the planning and thinking process moved forward, I found solutions to the problems. However, if I knew the further use of the data set in the subsequent steps, the cleaning process would be more target oriented.