INF 1340 Mid-Term Project

Introduction

During the first half of this course, we have learned several different aspects of human-centered data science. They include different research methods, research ethics, various biases to be avoided, and some codes to help us analyse dataset. One of the key components of analysing data is to make sure that the dataset is clean so that future analysis can be conducted at ease. In class, we have learned five basic principles about tidy data. They are, firstly, column headers should not be values but variable names; secondly, variables should not be stored in one column; variables need to be in cells, not rows and columns, and each cell represents the values of its respected column; fourthly, each table column needs to have a singular data type; and lastly, a single observational unit must be in one table. We were given the UN Migrant Stock Total dataset to clean for this mid-term project. The following paragraph will talk about the dataset and potential issues with it.

The UN Migrant Stock Total dataset contains 9 tables. The first, and last two tables are Contents, Annex, and Notes, so we disregard these three tables. This will leave us with 6 tables remaining, with actual data. Thus, we will focus on these three tables. On the surface, there are some obvious problems with these tables. Such as the tables have sex and year as variables in the header. Furthermore, some of these tables are sharing the same unit of measurement. These tables should be combined to conserve space and easier for later. Therefore, we will use the techniques learned from this class to clean this dataset.

Methods

I have uploaded the whole xslx data to Jupyter and created a Python notebook to operate on this dataset. I have also imported Pandas, and Numpy packages for this data cleaning job.

Moreover, I have also installed openpyxl using !pip install function in Python. The following is a screenshot of these steps.

```
import pandas as pd
import numpy as np
#importing necessary packages for data cleaning
```

```
!pip install openpyxl
#installing openpyxl for Jupyter to be able to read xlsx files directly
```

```
Requirement already satisfied: openpyxl in /opt/conda/lib/python3.8/site-packages (3.0.10)
Requirement already satisfied: et-xmlfile in /opt/conda/lib/python3.8/site-packages (from openpyxl) (1.1.0)
```

After that, we set the displayed column to be maximum so it will be easier for us to see all the variables later. We will also import the data file into Python, as the following.

```
pd.set_option('display.max_columns', None)
#easier to read all the columns within Jupyter
```

```
UN_dataset = pd.ExcelFile('UN_MigrantStockTotal_2015.xlsx')
#importing the UN dataset to Python
```

Furthermore, I have noticed that all the tables contain 15 rows of titles for that table. So, I excluded these rows for each table. The following is the code for Table 1, but the same procedures are applied to all the tables.

```
df1 = pd.read_excel(UN_dataset, 'Table 1')
#loading the first table with actual data
df1.head(20)
#noticing that thr first couple rows are just used to display the title of this tabular
```

We will then do some data wrangling with each table, respectively. But because those are more focused on the five principles, we will talk about them in the following section.

Five Tidy Data Principles

Principle #1: Column headers should not be values but variable names.

This principle tells us that if we have variables in the columns, we should transform them so that the variables are in cells, and column headers contain only variable names.

This principle is violated in all the tables. We will use Table 1 as an example. Noticed that the header rows look like the following, after eliminating the title rows.

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12	Ur
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	2015.0	1990	1995	

(This is not exhaustive as the table has too many columns).

The table does not have a header yet, so we will first manipulate the rows with actual headers to the header place. I did the following, so the headers are in the header row now.

```
df1_header = df1.iloc[13]
#locating the actual headers in Table 1
df1 = df1.iloc[14:]
#remove the unnecessary rows from Table 1
df1.columns = df1_header
#setting the correct headers for the new Table 1
df1.head()
#displaying the updated Table 1 with correct headers
```

The result of the above codes looks like the following (this is also done to all the tables).

13	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)	NaN	NaN	NaN	NaN	NaN	International migrant stock at mid-year (male)	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	2015.0	1990	1995	200(
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	243700236.0	77747510	81737477	8788483(
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	140481955.0	40263397	45092799	5053679(
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	103218281.0	37484113	36644678	3734804(
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	11951316.0	5843107	6142712	536190(

Now with the correct headers, it is obvious that both the years (which are not even in the header) and gender are values, but they are in the columns, instead of in cells. Because years are under gender now, we will separate Table 1 into three different parts, each part will contain the data for each gender, or for both sexes, with all the years. We used the following codes to do so.

```
#because Table 1 is showing the measurement for total, male, and female for each country, each year. This violated Prir
df1_part1 = df1.iloc[:,0:11]
df1_part2 = df1.iloc[:,[0,11,12,13,14,15,16]]
df1_part3 = df1.iloc[:,[0,17,18,19,20,21,22]]
#choosing the correct columns for each part
df1_part1.head()
#displaying part1
```

After applying the previous codes, Table 1 is now divided into df1_part1, df1_part2, and df1_part3. I will paste df1_part1 as an example below.

	Sort_order	Major area, region, country or area of destination	Notes	Country_code	Type_of_data(a)	1990	1995	2000	2005	2010	2015
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	2015.0
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	243700236.0
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	140481955.0
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	103218281.0
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	11951316.0

Noticing that the first row now is empty or with redundant year information as we have already moved the headers to the header row, thus we eliminate the first row by following:

```
df1_part1 = df1_part1.iloc[1:]
#removing the top line with years
df1_part1.head()
#displaying the updated df
```

The completed first part for Table 1 (df1_part1) is shown as below. This same procedure is done to all the tables except Table 4 where it has only one part. This is only showing df1_part1 as an example.

	Sort_order	Major area, region, country or area of destination	Notes	Country_code	Type_of_data(a)	1990	1995	2000	2005	2010	2015
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	2015.0
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	243700236.0
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	140481955.0
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	103218281.0
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	11951316.0

Because we have done dividing the tables with gender information (male, female, both), we have removed the variable from the headers. We will transform them into cells as a variable value, but for now we will continue with the second principle.

Principle #2: Multiple variables should not be stored in one column.

This principle asks us to make sure that each column only contains one piece of information. If the column has more than one piece of information, we should divide the information into different columns.

We have found that one of the variables is name as “Major area, region, country or area of destination”. In this column, it contains different variables, namely major area, region, and country or area of destination. These are different variables, and they should not be stored in the same column. Therefore, we should split them into different columns. Fortunately, they are already sorted in the original dataset (i.e., countries form the same major areas are all grouped together), making our job a bit easier. To split this column, we first need to get all the values from the column out as we will later assign them to different columns. The code looks like the following:

```
#We have noticed that for column "Major area, region, country or area of destination", it contains more than 1 variable
major_area = ['WORLD', 'Africa', 'Asia', 'Europe', 'Latin America and the Caribbean','Northern America', 'Oceania']
region = ['Developed regions','Developing regions','Sub-Saharan Africa','Eastern Africa','Middle Africa','Northern Afri
         'Southern Africa','Western Africa','Central Asia','Eastern Asia','South-Eastern Asia','Western Asia','Eastern
          'Northern Europe','Southern Europe','Western Europe','Caribbean','Central America','South America',
          'Australia and New Zealand','Melanesia','Micronesia','Polynesia']
Developing_region = df1.iloc[4:6]['Major area, region, country or area of destination'].values
Eastern_Africa = df1.iloc[9:29]['Major area, region, country or area of destination'].values
Middle_Africa = df1.iloc[30:39]['Major area, region, country or area of destination'].values
Northern_Africa = df1.iloc[40:47]['Major area, region, country or area of destination'].values
Southern_Africa = df1.iloc[48:53]['Major area, region, country or area of destination'].values
Western_Africa = df1.iloc[54:71]['Major area, region, country or area of destination'].values
Central_Asia = df1.iloc[73:78]['Major area, region, country or area of destination'].values
Eastern_Asia = df1.iloc[79:86]['Major area, region, country or area of destination'].values
South_Eastern_Asia = df1.iloc[87:98]['Major area, region, country or area of destination'].values
Southern_Asia = df1.iloc[99:108]['Major area, region, country or area of destination'].values
Western_Asia = df1.iloc[109:127]['Major area, region, country or area of destination'].values
Eastern_Europe = df1.iloc[129:139]['Major area, region, country or area of destination'].values
Northern_Europe = df1.iloc[140:153]['Major area, region, country or area of destination'].values
Southern_Europe = df1.iloc[154:170]['Major area, region, country or area of destination'].values
Western_Europe = df1.iloc[171:180]['Major area, region, country or area of destination'].values
Caribbean = df1.iloc[182:208]['Major area, region, country or area of destination'].values
Central_America = df1.iloc[209:217]['Major area, region, country or area of destination'].values
South_America = df1.iloc[218:232]['Major area, region, country or area of destination'].values
Australia_and_NewZealand = df1.iloc[240:242]['Major area, region, country or area of destination'].values
Melanesia = df1.iloc[243:248]['Major area, region, country or area of destination'].values
Micronesia = df1.iloc[249:256]['Major area, region, country or area of destination'].values
Polynesia = df1.iloc[257:266]['Major area, region, country or area of destination'].values
Northern_America = df1.iloc[233:238]['Major area, region, country or area of destination'].values
```

We will then define a function to help us assign values to region, and major area, respectively. Defining a function here can be useful as we can call this same function for later tables. First, we define the function to return us the correct values for the new “region” column:

```python
def get_region(aa):
    if ((aa in major_area) or (aa in region)):
        return aa
    elif (aa in Developing_region):
        return 'Developing region'
    elif (aa in Eastern_Africa):
        return 'Eastern Africa'
    elif (aa in Middle_Africa):
        return 'Middle Africa'
    elif (aa in Northern_Africa):
        return'Northern Africa'
    elif (aa in Southern_Africa):
        return 'Southern Africa'
    elif (aa in Western_Africa):
        return 'Western Africa'
    elif (aa in Central_Asia):
        return 'Central Asia'
    elif (aa in Eastern_Asia):
        return 'Eastern Asia'
    elif (aa in South_Eastern_Asia):
        return 'South Eastern Asia'
    elif (aa in Southern_Asia):
        return 'Southern Asia'
    elif (aa in Western_Asia):
        return 'Western Asia'
    elif (aa in Eastern_Europe):
        return 'Eastern Europe'
```

```python
    elif (aa in Northern_Europe):
        return 'Northern Europe'
    elif (aa in Southern_Europe):
        return 'Southern Europe'
    elif (aa in Western_Europe):
        return 'Western Europe'
    elif (aa in Caribbean):
        return 'Caribbean'
    elif (aa in Central_America):
        return 'Central America'
    elif (aa in Australia_and_NewZealand):
        return 'Australia and New Zealand'
    elif (aa in South_America):
        return 'South America'
    elif (aa in Melanesia):
        return 'Melanesia'
    elif (aa in Micronesia):
        return 'Micronesia'
    elif (aa in Polynesia):
        return 'Polynesia'
    elif (aa in Northern_America):
        return 'Northern America'
```

Having the region ready for each country, we will then use the lists of regions and their relationships with their major area to get the values for the new “major area” column. We first define the relationships between all the regions and major areas:

```python
WORLD =['Developed regions','Developing regions','Sub-Saharan Africa']
Africa = ['Eastern Africa','Middle Africa','Northern Africa','Southern Africa','Western Africa']
Asia = ['Central Asia','Eastern Asia','South-Eastern Asia','Southern Asia','Western Asia']
Europe = ['Eastern Europe','Northern Europe','Southern Europe','Western Europe']
Caribbeannn =['Caribbean','Central America','South America']
Oceania =['Australia and New Zealand','Melanesia','Micronesia','Polynesia']
```

Then we use a similar function to get the values for major area:

```
def get_major_area(aa):
    if (aa in major_area):
        return aa
    elif (aa in WORLD):
        return 'WORLD'
    elif (aa in Africa):
        return 'Africa'
    elif (aa in Asia):
        return 'Asia'
    elif (aa in Europe):
        return 'Europe'
    elif (aa in Caribbeannn):
        return 'Latin America and the Caribbean'
    elif (aa in Oceania):
        return 'Oceania'
    elif (aa in Developing_region):
        return 'WORLD'
    elif (aa in Eastern_Africa):
        return 'Africa'
    elif (aa in Middle_Africa):
        return 'Africa'
    elif (aa in Northern_Africa):
        return'Africa'
    elif (aa in Southern_Africa):
        return 'Africa'
    elif (aa in Western_Africa):
        return 'Africa'
    elif (aa in Central_Asia):
        return 'Asia'
    elif (aa in Eastern_Asia):
        return 'Asia'
```

```
    elif (aa in South_Eastern_Asia):
        return 'Asia'
    elif (aa in Southern_Asia):
        return 'Asia'
    elif (aa in Western_Asia):
        return 'Asia'
    elif (aa in Eastern_Europe):
        return 'Europe'
    elif (aa in Northern_Europe):
        return 'Europe'
    elif (aa in Southern_Europe):
        return 'Europe'
    elif (aa in Western_Europe):
        return 'Europe'
    elif (aa in Caribbean):
        return 'Latin America and the Caribbean'
    elif (aa in Central_America):
        return 'Latin America and the Caribbean'
    elif (aa in Australia_and_NewZealand):
        return 'Oceania'
    elif (aa in South_America):
        return 'Latin America and the Caribbean'
    elif (aa in Melanesia):
        return 'Oceania'
    elif (aa in Micronesia):
        return 'Oceania'
    elif (aa in Polynesia):
        return 'Oceania'
    elif (aa in Northern_America):
        return 'Oceania'
```

Now we have the two functions ready, we now apply them to our dataset:

```
new_table1['region'] = new_table1['Major area, region, country or area of destination'].apply(get_region)
new_table1['major_area'] = new_table1['Major area, region, country or area of destination'].apply(get_major_area)
new_table1 = new_table1.rename(columns={'Major area, region, country or area of destination': "country"})
new_table1 = new_table1[['Sort_order','major_area','region','country','Notes','Country_code','Type_of_data(a)','year','
new_table1.head()
```

The result looks like the following:

	Sort_order	major_area	region	country	Notes	Country_code	Type_of_data(a)	year	International_migrant_stock_at_mid-year(both_sexes)	International_migrant_stock_a year
0	1	WORLD	WORLD	WORLD	NaN	900	NaN	1990	152563212	777
1	2	WORLD	Developed regions	Developed regions	(b)	901	NaN	1990	82378628	402
2	3	WORLD	Developing regions	Developing regions	(c)	902	NaN	1990	70184584	374
3	4	WORLD	Developing region	Least developed countries	(d)	941	NaN	1990	11075966	58
4	5	WORLD	Developing region	Less developed regions excluding least develop...	NaN	934	NaN	1990	59105261	316

Now the problematic column is divided into three different columns. We have successfully fixed the violations of Principle#2. (This is done after I fixed Principle#1 and Principle#3, but for the sake of the order of the principles, I have reordered the sequence in this write-up).

Principle #3: Variables need to be in cells, not rows and columns.

This principle states that if any variables are put in rows, or columns, instead of being in cells, we should reform the data frame so that the values are in cells.

Originally, the tables had two rows of headers. We have moved one row to the actual header row to form a proper header. However, year is still stored in the first row now. Currently, the data looks like this:

13	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)	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	2015.0
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	243700236.0
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	140481955.0
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	103218281.0
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	11951316.0

We will move the year value to the header, and then remove the first row so that Principle#3 will no longer be violated. We use the following code:

```
cols = ['Sort_order','Major area, region, country or area of destination','Notes','Country_code','Type_of_data(a)','199
#giving the correct names to the all the headers
df1_part1.set_axis(cols, axis=1,inplace=True)
#replacing the previous headers to the ones we just made
df1_part1.head()
#displaying the updated df
```

```
df1_part1 = df1_part1.iloc[1:]
#removing the top line with years
df1_part1.head()
#displaying the updated df
```

The issue with the “year” row is now fixed, and the data looks like this:

	Sort_order	Major area, region, country or area of destination	Notes	Country_code	Type_of_data(a)	1990	1995	2000	2005	2010	2015
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	243700236.0
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	140481955.0
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	103218281.0
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	11951316.0
19	5	Less developed regions excluding least develop...	NaN	934	NaN	59105261	56778501	59244124	64272611	79130668.0	91262036.0

Vola! Principle#3 is now conformed.

Principle #4: Each table column needs to have a singular data type.

Principle#4 states that if data types are different, they should be in different tables, instead of the same one. I will combine Principle#5 to illustrate what I have done for the dataset to meet this requirement.

Principle #5: A single observational unit must be in one table.

The last principle basically suggests that if we have different groups of data in the same observational unit, we should merge them into the same table. For this project, we have divided all the different tables into various sub-parts already. Therefore, we will merge the ones that meet the requirement of Principle#5, and leave the remaining separate, according to Principle#4.

The whole dataset measured immigrants and refugee information based on country, by gender. However, some of the data is measured by year (a time frame), whereas the others were measured on a 5-year time basis (a time duration). So I have decided for this project (without the analysing part) I will merge all the parts with the same time frame (year with year, 5-year with 5-year) according to Principle#5, and leave these tow data frames separate according to Principle#4.

Before merging all the different sub-parts, I need to finish what I left in Principle#1, to put gender information into a new column, and assign the correct value to the column. I used the code below (the same is done to all the sub-parts, this one is pasted here as an example):

```
tidy_table2 = new_table2.melt(
    id_vars = ['Sort_order', 'major_area', 'region', 'country', 'Notes', 'Country_code', 'year'],
    var_name = "gender_break_down",
    value_name = "Total_population_at_mid-year"
)
tidy_table2 = tidy_table2.replace(to_replace=['Total_population_of_both_sexes_at_mid-year(thousands)','Total_male_popul
tidy_table2.head()
```

The result now looks like this:

	Sort_order	major_area	region	country	Notes	Country_code	year	gender_break_down	Total_population_at_mid-year
0	1	WORLD	WORLD	WORLD	NaN	900	1990	both	5309667.699
1	2	WORLD	Developed regions	Developed regions	(b)	901	1990	both	1144463.062
2	3	WORLD	Developing regions	Developing regions	(c)	902	1990	both	4165204.637
3	4	WORLD	Developing region	Least developed countries	(d)	941	1990	both	510057.629
4	5	WORLD	Developing region	Less developed regions excluding least develop...	NaN	934	1990	both	3655147.008

Notice that now the table has a new column called "gender_break_down", and it contains the gender information about that particular observation.

Having done that, to merge the sub-parts, firstly I will merge them together directly if they were from the same table. The code looks like the following:

```
new_table1 = pd.merge(tidy_df1_part1, tidy_df1_part2, left_on=['Sort_order', 'year'], right_on=['Sort_order', 'year'])
#merging df1_part1 and df1_part2
new_table1 = pd.merge(new_table1, tidy_df1_part3, left_on=['Sort_order', 'year'], right_on=['Sort_order', 'year'])
#merging all three parts
new_table1.head(30)
#displaying the merged table
```

The merged Table 1 is shown as an example below, with Principle#1, #2, and #3 conformed (the left part, including the "sort_order", "major_area", "region" columns are cut out for this picture to save space:

country	Notes	Country_code	Type_of_data(a)	year	International_migrant_stock_at_mid-year(both_sexes)	International_migrant_stock_at_mid-year(male)	International_migrant_stock_at_mid-year(female)
WORLD	NaN	900	NaN	1990	152563212	77747510	74815702
eveloped regions	(b)	901	NaN	1990	82378628	40263397	42115231
veloping regions	(c)	902	NaN	1990	70184584	37484113	32700471
Least eveloped countries	(d)	941	NaN	1990	11075966	5843107	5236216
Less eveloped regions xcluding least evelop...	NaN	934	NaN	1990	59105261	31641006	27464255

Then we will combine all the tables with the same time unit into the same table. I used merge function and joint them one by one, and an example is shown below:

```
tidy_table6_1['gender_break_down'] = "both"
#adding a new column into tidy_table4, and assign value "female" to all the cells as this table shows female data only
cols = ['Sort_order','major_area','region','country','Notes','Country_code','Type_of_data(a)','year','Estimated_refugee
tidy_table6_1.set_axis(cols, axis=1,inplace=True)
#renaming the columns as "both sexes" is now moved into a separate column
tidy_table6_1.head()
#displaying the dataframe for future comparison
```

Another example of code is shown when certain sub-parts have fewer rows because they only contain data with one gender_break_down value ("how= 'left'"):

```
merged_temp3 = merged_temp2.merge(tidy_table4, how = 'left', on = ['Sort_order','major_area','region','country','Notes'
merged_temp3.tail()
#df1.merge(df2, how='inner', on='a')
#displaying the last rows to match the results. Perfectly merged!やったね!
```

So far, all 5 principles are checked and met, with our new dataset.

Results

The whole dataset is cleaned using the 5 principles of tidy data. The result is consisted of two tables. Within each table, each row is a unique measurement, and each column is a unique variable with only one piece of information. Each cell is a unique observational value, and each table has only one type of data.

Table 1

Table 1 contains all the information with time frame being a single year. The data frame looks like the following (there are more columns on the right, cut here to conserve space):

	Sort_order	major_area	region	country	Notes	Country_code	Type_of_data(a)	year	gender_break_down	International_migrant_stock_at_mid-year	Total
0	1	WORLD	WORLD	WORLD	NaN	900	NaN	1990	both	152563212	
1	2	WORLD	Developed regions	Developed regions	(b)	901	NaN	1990	both	82378628	
2	3	WORLD	Developing regions	Developing regions	(c)	902	NaN	1990	both	70184584	
3	4	WORLD	Developing region	Least developed countries	(d)	941	NaN	1990	both	11075966	
4	5	WORLD	Developing region	Less developed regions excluding least develop...	NaN	934	NaN	1990	both	59105261	
...	...	...	...	...	...	...	...	...	...	...	
4765	261	Oceania	Polynesia	Samoa	NaN	882	B	2015	female	2460.0	
4766	262	Oceania	Polynesia	Tokelau	NaN	772	B	2015	female	254.0	
4767	263	Oceania	Polynesia	Tonga	NaN	776	B	2015	female	2604.0	
4768	264	Oceania	Polynesia	Tuvalu	NaN	798	C	2015	female	63.0	
4769	265	Oceania	Polynesia	Wallis and Futuna Islands	NaN	876	B	2015	female	1411.0	

Table 2

Table 2 is like Table 1, with year column measured by a 5-year time frame:

	Sort_order	major_area	region	country	Notes	Country_code	Type_of_data(a)	year	gender_break_down	Annual_rate_of_change_of_migrant_stock
0	1	WORLD	WORLD	WORLD	NaN	900	NaN	1990-1995	both	1.051865
1	2	WORLD	Developed regions	Developed regions	(b)	901	NaN	1990-1995	both	2.275847
2	3	WORLD	Developing regions	Developing regions	(c)	902	NaN	1990-1995	both	-0.487389
3	4	WORLD	Developing region	Least developed countries	(d)	941	NaN	1990-1995	both	1.118175
4	5	WORLD	Developing region	Less developed regions excluding least develop...	NaN	934	NaN	1990-1995	both	-0.803244
...	...	...	...	...	...	...	...	...	...	...
3970	261	Oceania	Polynesia	Samoa	NaN	882	B	2010-2015	female	-0.545343
3971	262	Oceania	Polynesia	Tokelau	NaN	772	B	2010-2015	female	2.60325
3972	263	Oceania	Polynesia	Tonga	NaN	776	B	2010-2015	female	2.526318
3973	264	Oceania	Polynesia	Tuvalu	NaN	798	C	2010-2015	female	-1.819436
3974	265	Oceania	Polynesia	Wallis and Futuna Islands	NaN	876	B	2010-2015	female	0.516899

Discussion

So far in the course, we have learned many aspects of human-centered data science. From how to conduct research, how to choose data, to how to analyze them properly, how to present your results, and furthermore, how to be ethical. During this data cleaning process, I feel that many of these topics are more relatable than what I have imagined when I was just reading the textbook. Questions such as “Why do I need to combine these two tables?”, “What should I record when I manipulate the tables?”, and “How was this data collected?” frequently came to my mind. These five principles of tidy data can serve as a great guideline for our data cleaning project. But when we eventually start the cleaning job, much more need to be considered. It is not as straight-forward as “following this and do this”, but rather “should I do this and why so” types of scenarios. Yet I have not even started the analysis part.

I strongly agree with the textbook, mentioning that we should always reflect on what we are doing, why we are doing it, and how we can do it better at all stages of data projects. Keep a detailed log on what we have done to the data and the rationales behind the decision. Because for this project, I must reference back, multiple times, to what I have done before and improve on the methods or reconsider the decisions I have made. All these helped me to go back to my previous decisions, reflect, and improve on my codes and reasoning, which in turn improved the overall quality of this data cleaning project.