Gabrielle Woodside
1009676780

**Midterm Write-Up**

## Introduction

According to our textbook, tidy data is a "standard way of mapping the meaning of a dataset to its structure" (Aragon et al., 2022). So rows are unique observations, columns are unique variables, and cells are unique units of measurement. More specifically, there are five main principles of tidy data: (Wickham, 2014):

- **Principle 1:** Column headers should be variables, not values
- **Principle 2:** Each column should only have one variable stored in it
- **Principle 3:** Each row should be a unique observation
- **Principle 4:** Different kinds of observational units should be stored in their own tables
- **Principle 5:** Observational units of the same kind should be stored together, rather than spread across separate tables

With these principles in mind, I completed various steps to clean the UN Migrant Stock Total dataset from 2015.

## Methods and Results

### Steps 1 and 2 – Renaming Columns and Melting (For Sex and Year)

After uploading each table as an individual dataframe (ex. "Table 1" = "df1"), I noticed that the column headers were double layered (*Figure 1*). Considering Jupyter Notebook only reads the top column as the header, there was information that was being lost when the dataframe was created. The years did not show up as column headings, but as the first row in the dataset (*Figure 2*).

| International migrant stock at mid-year (male) | | | | | |
|---|---|---|---|---|---|
| 1990 | 1995 | 2000 | 2005 | 2010 | 2015 |
| 77 747 510 | 81 737 477 | 87 884 839 | 97 866 674 | 114 613 714 | 126 115 435 |
| 40 263 397 | 45 092 799 | 50 536 796 | 57 217 777 | 64 081 077 | 67 618 619 |
| 37 484 113 | 36 644 678 | 37 348 043 | 40 648 897 | 50 532 637 | 58 496 816 |
| 5 843 107 | 6 142 712 | 5 361 902 | 5 383 009 | 5 462 714 | 6 463 217 |
| 31 641 006 | 30 501 966 | 31 986 141 | 35 265 888 | 45 069 923 | 52 033 599 |
| 7 745 306 | 8 036 824 | 7 210 452 | 7 444 048 | 8 188 581 | 10 099 486 |
| 8 279 564 | 8 616 931 | 7 856 358 | 8 231 437 | 9 039 314 | 11 123 423 |
| 3 071 189 | 2 585 053 | 2 480 584 | 2 529 460 | 2 366 216 | 3 109 176 |

*Figure 1 - Format of columns in Table 1 (Excel version)*

| | 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) | Unnamed: 6 | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | NaN | 1990 | 1995 | 2000 | 2005 | 2010 | ... |
| 1 | 1.0 | WORLD | NaN | 900.0 | NaN | 152563212 | 160801752 | 172703309 | 191269100 | 221714243 | ... |
| 2 | 2.0 | Developed regions | (b) | 901.0 | NaN | 82378628 | 92306854 | 103375363 | 117181109 | 132560325 | ... |

*Figure 2 - Format of the Table 1 column headers when uploaded as a dataframe to Jupyter*

Gabrielle Woodside
1009676780

Also, in each table, the main variable of interest (ex. “International migrant stock at mid-year”) was separated into columns based on sex and year. This violates **Principle 1** of Tidy Data, which states that column headers must be variables, not values. The column headers are currently all values of “Year” or “Sex”, rather than representing one variable. Therefore, to remedy this violation, I decided to rename the columns and melt them so that the dataset became longer.

To begin, I renamed each column so that it combined the Sex and Year represented in that column. This allowed me to drop the first row of the table, which was empty except for some of the original column header information. After this, the dataframe looked like this (*Figure 3*):

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Type of data (a) | Both sexes - 1990 | Both sexes - 1995 | Both sexes - 2000 | Both sexes - 2005 | Both sexes - 2010 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.0 | WORLD | NaN | 900.0 | NaN | 152563212 | 160801752 | 172703309 | 191269100 | 221714243 | ... |
| 2 | 2.0 | Developed regions | (b) | 901.0 | NaN | 82378628 | 92306854 | 103375363 | 117181109 | 132560325 | ... |
| 3 | 3.0 | Developing regions | (c) | 902.0 | NaN | 70184584 | 68494898 | 69327946 | 74087991 | 89153918 | ... |
| 4 | 4.0 | Least developed countries | (d) | 941.0 | NaN | 11075966 | 11711703 | 10077824 | 9809634 | 10018128 | ... |

*Figure 3 - Table 1 (as a dataframe) once the columns are renamed to Sex and Year*

The next step was to *melt* these column headers, so that they became one general “Demographic” column. This way the values that were initially inside the columns could given a header that represents what kind of measurement they are (ex. “International migrant stock at mid-year”). I used the **df.melt()** function to accomplish this, and the resulting dataframes looked like this (*Figure 4*):

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Type of data (a) | demographic | Internatonal migrant stock at mid-year |
|---|---|---|---|---|---|---|---|
| 0 | 1.0 | WORLD | NaN | 900.0 | NaN | Both sexes - 1990 | 152563212 |
| 1 | 2.0 | Developed regions | (b) | 901.0 | NaN | Both sexes - 1990 | 82378628 |
| 2 | 3.0 | Developing regions | (c) | 902.0 | NaN | Both sexes - 1990 | 70184584 |

*Figure 4 - Table 1 (as a dataframe) with a “Demographic” column and a column representing the variable of interest*

I completed this step for each of the Tables after that. For Table 5 and part of Table 6, I needed to switch my process slightly, as they use year *ranges* rather than years. Table 6 required extra work before melting, as it had multiple variables of interest, and two different time units (years and year ranges) (*Figure 5*). These are different kinds of observations, so I could not just melt the columns and have the variables in the same rows. That would violate **Principle 4**, which states that each table should only have one type of observational unit.

| Estimated refugee stock at mid-year (both sexes) | | | | | | Refugees as a percentage of the international migrant stock | | | | | | Annual rate of change of the refugee stock | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1990 | 1995 | 2000 | 2005 | 2010 | 2015 | 1990 | 1995 | 2000 | 2005 | 2010 | 2015 | 1990-1995 | 1995-2000 | 2000-2005 | 2005-2010 | 2010-2015 |
| 16 836 571 | 17 653 840 | 15 827 863 | 13 276 733 | 15 370 755 | 19 577 474 | 12.9 | 11.1 | 9.2 | 6.9 | 8.3 | 8.8 | -2.12 | -3.84 | -3.50 | 0.03 | 2.39 |
| 2 014 564 | 3 609 670 | 2 997 256 | 2 361 229 | 2 046 917 | 1 954 224 | 2.4 | 3.9 | 2.9 | 2.0 | 1.5 | 1.4 | 9.39 | -5.98 | -7.28 | -5.32 | -2.09 |
| 16 822 007 | 14 244 170 | 12 830 547 | 10 915 504 | 13 323 838 | 17 623 250 | 24.0 | 20.8 | 18.5 | 14.7 | 14.9 | 17.1 | -2.84 | -2.33 | -4.56 | 0.29 | 2.66 |

*Figure 5 - The column headers in Table 6 (Excel version)*

Gabrielle Woodside
1009676780

Therefore, I used the **df.iloc()** function to split Table 6 into three separate dataframes, one for each variable of interest. This way, I could melt the individual variables without having to worry about contrasting time units or accidentally doubling the rows by melting a dataset twice.

### Step 3 – Merging Tables

Once I melted all the datasets individually, I realized that there was overlap between the columns. Each dataset had a country/major area/region column, notes, country code, and (usually) type of data. This means that there were single observational units stored across multiple tables, which violates **Principle 5** of tidy data (observational units of the same kind should be stored in one dataframe). Therefore, I decided to merge the unique columns from Table 2, Table 3, Table 4 (*see Note 1*) and part of Table 6 (*see Note 2*) into Table 1, as these were all variables that were measured per year. In the end, I had a dataframe with the following columns (*Figure 6*):

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Type of data (a) | demographic | Internatonal migrant stock at mid-year | Total population at mid-year (in thousands) | International migrant stock as percentage of total population | Percentage of international migrant stock | Estimated refugee stock at mid-year | Refugees as a percentage of the international migrant stock |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | WORLD | NaN | 900.0 | NaN | Both sexes - 1990 | 152563212 | 5.30967e+06 | 2.87331 | NA | 18836571 | 12.3467 |
| 1 | 2.0 | Developed regions | (b) | 901.0 | NaN | Both sexes - 1990 | 82378628 | 1.14446e+06 | 7.19802 | NA | 2014564 | 2.44549 |

*Figure 6 – Table 1 after combined with columns from Table 2, 3, 4 and 6*

*Note 1:* Table 4 (measuring "Percentage of international migrant stock") only listed information for female migrants. This means that it had less rows than Table 1, which prevented me from simply merging the two. Therefore, I made a list of values to represent the Table 4 column, and used a **for-loop** to make it the same length as Table 1. For any values that would be in a "Both sexes" or "Male" row in Table 1, I filled the space with *NA*. Then I was able to add the list as a new column in Table 1, without worrying about length.

*Note 2:* Table 6 only listed information for both sexes. Therefore, when migrating the two columns from it ("Estimated refugee stock at mid-year" and "Refugees as a percentage of the international migrant stock") I used the same strategy as with Table 4, but filled any "Male" or "Female" row with *NA.*

Both Table 5 and part of Table 6 had variables that were measured in ranges of years. In line with Principle 5 again, I merged the two of those into Table 5, with columns as seen below (*Figure 7*). Again, Table 6 only had information for both sexes combined, so I again needed to fill in the empty cells when merging it into Table 5, which had information for all categories of sex.

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Type of data (a) | demographic | Annual rate of change of the migrant stock | Annual rate of change of the refugee stock |
|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | WORLD | NaN | 900.0 | NaN | Both Sexes - 1990-1995 | 1.05186 | -2.1235 |
| 1 | 2.0 | Developed regions | (b) | 901.0 | NaN | Both Sexes - 1990-1995 | 2.27585 | 9.38842 |

*Figure 7 – Table 5 after combined with column from Table 6*

Gabrielle Woodside
1009676780

**Step 4 and 5 – Splitting columns with multiple variables**

The original "Type of Data" column in Table 1 (and others) breaks **Principle 2** of tidy data, which states that only variable can be stored in a column. As can be seen from the "Notes" tab in the Excel file, the "Type of Data" column actually houses three variables. It describes the following information:

- Whether the data used to produce the estimates refer to *foreign-born* population (B) or *foreign citizens* (C)
- Whether the number of refugees was added to the estimate of international migrants (R)
- Whether the estimates for countries/areas with no data on international migrants were obtained by imputation (I)

So in both Table 1 and Table 5, I split this column into three separate ones that each represent a different variable. To do this, I created three empty columns called "Born or Citizen", "Refugees Added" and "Imputation". Then for each of these columns, I used a **for-loop** to run through the "Type of Data" column and pull out relevant information. For example, if a "B" was found in the "Type of Data" column, then the value of "Born" was added to the "Born or Citizen" column. Once all the columns were full, I was able to drop the original "Type of Data" column from the table. In the end, it looked like *Figure 8*.

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Born or Citizen | Refugees Added | Imputation | demographic | Annual rat change of migrant st |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 11.0 | Djibouti | NaN | 262.0 | Born | Yes | | Both Sexes - 1990-1995 | -4.05 |
| 11 | 12.0 | Eritrea | NaN | 232.0 | | | Yes | Both Sexes - 1990-1995 | 0.910 |
| 12 | 13.0 | Ethiopia | NaN | 231.0 | Born | Yes | | Both Sexes - 1990-1995 | -7.17 |
| 13 | 14.0 | Kenya | NaN | 404.0 | Born | Yes | | Both Sexes - 1990-1995 | 14.6 |

*Figure 8 - Table 5 with the "Type of data" split into three separate columns*

After Steps 1 and 2, the "Demographic" column also violated Principle 2, as it had information both about the sex and year. Therefore, I created functions to pull the Sex and Year (or Year Range) out of the column and into their own respective columns. For the Sex, it pulled everything out of the cell minus the year and any extra spaces or dashes. For the Year, it pulled the last four characters in the cell (or last nine in the case of year ranges). Once the functions ran, I dropped the "Demographic" column, leaving tables that looked like *Figure 9*.

| | Sort\norder | Major area, region, country or area of destination | Notes | Country code | Born or Citizen | Refugees Added | Imputation | Sex | Year | Internator migra stock mid-ye |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.0 | WORLD | NaN | 900.0 | | No | No | Both sexes | 1990 | 1525632 |
| 1 | 2.0 | Developed regions | (b) | 901.0 | | No | No | Both sexes | 1990 | 823786 |

*Figure 9 - Table 1 with new Sex and Year columns*

Gabrielle Woodside
1009676780

**Step 6 – Splitting Datasets**

| Major area, region, country or area of destination |
|---|
| WORLD |
| Developed regions |
| Developing regions |
| Least developed countries |
| Less developed regions excluding least developed countries |
| Sub-Saharan Africa |
| Africa |
| Eastern Africa |
| Burundi |
| Comoros |
| Djibouti |

*Figure 10 - The column from the Excel version of the Tables*

Earlier I mentioned situations where one observational unit was spread across multiple tables. The opposite problem also exists in this dataset: there are multiple observational units within the tables (*Figure 10*). For example, measuring the variables based on the world is very different than measuring within a specific country. Having the various levels of country, major area, region, and more within one column violates **Principle 4**, where different kinds of observational units must be in separate tables.

My first idea was to create a column in the table that specifies which major area and region each country falls under, so pivoting the table and removing those rows. However, that would result in losing the information about each of those observations. Plus, the ANNEX table already explains the relationships between countries, regions, and major areas (as well as whether they are developed or not). I believe it would be redundant and repetitive to also include those columns in the main tables.

| | Original Sort Order | Major area, region, country or area of destination | Level |
|---|---|---|---|
| 0 | 1.0 | WORLD | |
| 1 | 2.0 | Developed regions | Developed or not |
| 2 | 3.0 | Developing regions | Developed or not |
| 3 | 4.0 | Least developed countries | |
| 4 | 5.0 | Less developed regions excluding least develop... | |
| 5 | 6.0 | Sub-Saharan Africa | |
| 6 | 7.0 | Africa | Major area |

*Figure 11 - Table 1 with the "Level" column*

Instead, I decided to split the table into multiple smaller tables. This way I could keep the important information, but in a way that is tidier. To start the splitting process, I created a "Level" column to help me determine what kind of observational unit each row was. As a reference, I uploaded the ANNEX table as its own dataframe, as it outlines which areas count as a country, region, etc.

Just as in earlier steps, I used a self-defined function that used **for-loops** and conditionals to add to this column. I ran through each row in Table 1 – "Major area, region, country or area of destination", and assessed which column it fell under in the ANNEX. For example, if the cell is also in the "Country or area" column of the ANNEX, it is labelled as *Country* in the new "Level" column. For the unique categories (Ex. *WORLD, Developed regions, developing regions,* etc.), I left them blank, as I could pull them out just using their unique value. The result was a table with a "Level" column such as in *Figure 11*. The same process was completed for Table 5.

From there, I created smaller datasets for each kind of observation unit in the original datasets. For example, I created a "df1_world" dataset out of rows in Table 1 that provided information at the world level. While completing this task, I noticed that there were some columns that were very repetitive (ex. "Country Code") or irrelevant (ex. none of the Type of Data columns I made apply to the World observations). To save space and reduce repetition, I created an "Extra Annex" dataset that housed identifying information for the unique observations that were not listed in the original ANNEX table. Below shows the rows and columns included in the Extra Annex (*Figure 12*):

Gabrielle Woodside
1009676780

| | Original Sort Order | Major area, region, country or area of destination | Notes | Country code |
|---|---|---|---|---|
| **0** | 1.0 | WORLD | NaN | 900.0 |
| **1** | 2.0 | Developed regions | (b) | 901.0 |
| **2** | 3.0 | Developing regions | (c) | 902.0 |
| **3** | 4.0 | Least developed countries | (d) | 941.0 |
| **4** | 5.0 | Less developed regions excluding least develop... | NaN | 934.0 |
| **5** | 6.0 | Sub-Saharan Africa | (e) | 947.0 |

*Figure 12 - The "Extra Annex" made for the unique kinds of observations not included in the original ANNEX sheet*

Now that the information was housed elsewhere, I could remove these columns from the individual datasets. I also renamed the “Major area…” column to reflect the kind of observational unit that was in each dataset. Here is an example of one of the smaller datasets, specifically the information for “WORLD” from Table 1 (*Figure 13*):

| | Level | Sex | Year | Internatonal migrant stock at mid-year | Total population at mid-year (in thousands) | International migrant stock as percentage of total population | Percentage of international migrant stock | Estimated refugee stock at mid-year | Refugees as a percentage of the international migrant stock |
|---|---|---|---|---|---|---|---|---|---|
| **0** | WORLD | Both sexes | 1990 | 152563212 | 5.30967e+06 | 2.87331 | NA | 18836571 | 12.3467 |
| **265** | WORLD | Both sexes | 1995 | 160801752 | 5.73512e+06 | 2.80381 | NA | 17853840 | 11.103 |
| **530** | WORLD | Both sexes | 2000 | 172703309 | 6.12662e+06 | 2.8189 | NA | 15827803 | 9.16474 |
| **795** | WORLD | Both sexes | 2005 | 191269100 | 6.51964e+06 | 2.93374 | NA | 13276733 | 6.94139 |
| **1060** | WORLD | Both sexes | 2010 | 221714243 | 6.92973e+06 | 3.19947 | NA | 15370755 | 6.93269 |

*Figure 13 - "df1_world" - A table of only the "World" rows from Table 1, with the irrelevant or redundant columns removed*

In the end, I ended up with the following smaller datasets:

| Table 1 Version | Table 5 Version | Content |
|---|---|---|
| df1_world | df5_world | All of the “World” rows |
| df1_dev | df5_dev | Rows about either the “Developed Regions” or the “Developing Regions” |
| df1_least_dev | df5_least_dev | All rows about the “Least developed countries” |
| df1_less | df5_less | All rows about the “Less developed regions minus the least developed countries” |
| df1_subsaharan | df5_subsaharn | All rows about “Sub-Saharan Africa” |
| df1_countries | df5_countries | Any rows that have a country (as defined by the “Level” column) |
| df1_regions | df1_regions | Any rows that have a region (as defined by the “Level” column) |
| df1_major_areas | df5_major_areas | Any rows that have a major area (as defined by the “Level” column) |

Gabrielle Woodside
1009676780

## Discussion

Through the process of cleaning this dataset, I discovered a few things:

First, as has been mentioned in our class, I can understand why people may choose to make their datasets untidy. In this case, I don't blame the creators of the UN Dataset for including all kinds of observations (countries, regions, worlds, etc.) in one table at a time, with the years and genders spread out through the columns. Visually, it is easier to view all the data together (compared to my solution which has sixteen different datasets).

However, I can also understand why the initial organization would make it difficult to analyze the information. For example, if someone wanted to run a test or create a visualization that showed the relationship between years and international migrant stock at mid-year, it would have been difficult for them to do this with the years spread out over columns. They would have needed to pull the years out of the headers anyway to use them in the analysis/visualization. Plus, by having the countries/regions/major areas in their own datasets, it will be easier to compare units within the dataset. You will only have to worry about comparing information of the same observational unit, rather than of different levels.

I do wonder though what other solutions exist, rather than completely splitting the observations as I did. As mentioned before, I ended up with sixteen different tables. I still think that it might not have worked to pivot the rows into columns, as I would have lost valuable information. However, it would not be easy to keep track of sixteen separate datasets when doing an analysis. So, if I were to continue moving forward with this project, I would try to find a way to make the information more manageable.

As well, I would try to make the *splitting* process more manageable. For each dataset I created, it required at least 3-4 lines of code. And it was usually the same code each time, with very small changes. I did attempt to create a function that could be applied to any table and pull out whichever information I chose (like I did when adding the "Level" columns or splitting the "Demographic" columns). Unfortunately, I kept receiving error messages during the process. So again, going forward I would keep trying to simplify the process and reduce the redundant code when splitting the datasets.

## References

Aragon, C., Guha, S., Kogan, M., Muller, M., & Neff, G. (2022). *Human-Centered Data Science: An Introduction*. MIT Press.

Wickham, Hadley. 2014. "Tidy Data." Journal of Statistical Software 59 (1): 1–23. doi:10.18637/jss.v059.i10. – also accessible via https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html