Introduction:

This write-up is on different approaches we can take to transform a messy dataset into a tidy dataset. We need to do that because when data is loaded in integrated development environment such as 'python', it does not appear as it does in an excel file. In excel file data is organized in a different manner which necessarily may not be appropriate for analysis. Excel is a good tool for conducting some very basic analysis but if we want to delve deeper into it and conduct analysis on big data then we need other tools. So, the first step is to get our dataset ready.

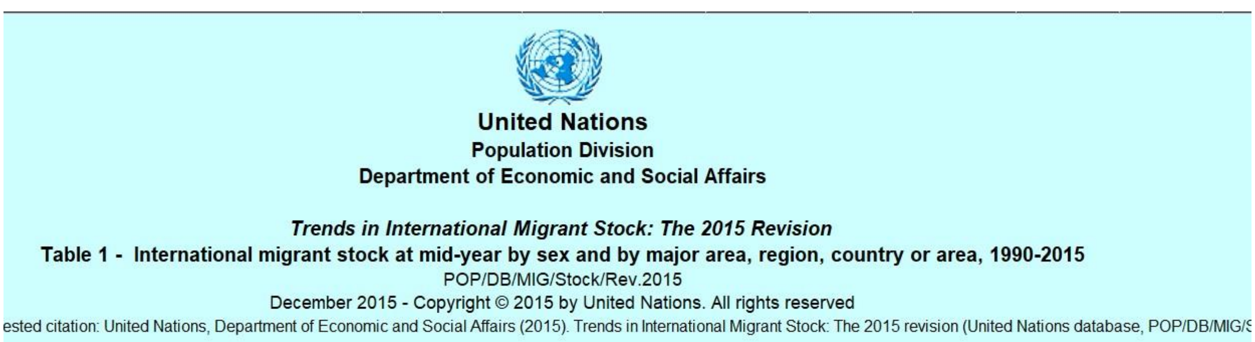
Description:

UN Dataset contains data on trends of international migrant stock. It is a revised version for year 2015. It has data for every five years starting from 1990. It has six different tables. First five columns are repeating for all of the six tables. Contents of each table are listed below.

TABLE	TITLE
Table 1	International migrant stock at mid-year by sex and by major area, region, country or area, 1990-2015
Table 2	Total population at mid-year by sex and by major area, region, country or area, 1990-2015 (thousands)
Table 3	International migrant stock as a percentage of the total population, 1990-2015
Table 4	Female migrants as a percentage of the international migrant stock by major area, region, country or area, 1990-2015
Table 5	Annual rate of change of the migrant stock by sex and by major area, region, country or area, 1990-2015 (percentage)
Table 6	Estimated refugee stock at mid-year by major area, region, country or area, 1990-2015
ANNEX	Classification of countries and areas by major area and region
NOTES	NOTES

Principle 1: Column headers are values, not variable names.

When we first read the excel file into our dataframe, the first thing we notice is that it's does not have any headers. Since the excel file has a header or banner on each of the sheets, sharing a screenshot below:


United Nations
Population Division
Department of Economic and Social Affairs

Trends in International Migrant Stock: The 2015 Revision

Table 1 - International migrant stock at mid-year by sex and by major area, region, country or area, 1990-2015

POP/DB/MIG/Stock/Rev.2015

ested citation: United Nations, Department of Economic and Social Affairs (2015). Trends in International Migrant Stock: The 2015 revision (United Nations database, POP/DB/MIG/S

When we read it in excel it looks like this:

```python
df = pd.read_excel(io="UN_MigrantStockTotal_2015.xlsx", sheet_name= 'Table 1', index_col=False)
print(display(df.head(20)))
```

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 13
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
3	NaN	NaN	NaN	NaN	United Nations	NaN	NaN	NaN	NaN	NaN	...	NaN
4	NaN	NaN	NaN	NaN	Population Division	NaN	NaN	NaN	NaN	NaN	...	NaN
5	NaN	NaN	NaN	NaN	Department of Economic and Social Affairs	NaN	NaN	NaN	NaN	NaN	...	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
7	NaN	NaN	NaN	NaN	Trends in International Migrant Stock: The 201...	NaN	NaN	NaN	NaN	NaN	...	NaN
8	NaN	NaN	NaN	NaN	Table 1 - International migrant stock at mid-...	NaN	NaN	NaN	NaN	NaN	...	NaN
9	NaN	NaN	NaN	NaN	POP/DB/MIG/Stock/Rev.2015	NaN	NaN	NaN	NaN	NaN	...	NaN
10	NaN	NaN	NaN	NaN	December 2015 - Copyright © 2015 by United Nat...	NaN	NaN	NaN	NaN	NaN	...	NaN
11	NaN	NaN	NaN	NaN	Suggested citation: United Nations, Department...	NaN	NaN	NaN	NaN	NaN	...	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
13	Sort\norder	Major area, region, country or area of destina...	Notes	Country code	Type of data (a)	International migrant stock at mid-year (both ...	NaN	NaN	NaN	NaN	...	NaN
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	...	2000
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	...	87884839
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	...	50536796

We can see in the above screenshot that actual column title is in 13th row. We can drop the first 13 rows as it just contains header data translated into different columns. No important information will be lost.

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 13	Unnamed: 14	Un
13	Sort\norder	Major area, region, country or area of destina...	Notes	Country code	Type of data (a)	International migrant stock at mid-year (both ...	NaN	NaN	NaN	NaN	...	NaN	NaN	
14	NaN	NaN	NaN	NaN	NaN	1990	1995	2000	2005	2010.0	...	2000	2005	
15	1	WORLD	NaN	900	NaN	152563212	160801752	172703309	191269100	221714243.0	...	87884839	97866674	1146
16	2	Developed regions	(b)	901	NaN	82378628	92306854	103375363	117181109	132560325.0	...	50536796	57217777	640
17	3	Developing regions	(c)	902	NaN	70184584	68494898	69327946	74087991	89153918.0	...	37348043	40648897	505
18	4	Least developed countries	(d)	941	NaN	11075966	11711703	10077824	9809634	10018128.0	...	5361902	5383009	54
		Less developed												

Once we do this, our dataset already looks a lot clearer. I kept the 13th row because it contains important information on gender and year which I will be using in my code to rename columns.

At this point, I will drop first column "Unnamed: 0" because it just contains sort order which we don't need as we already have 'index' for numbered list and if we want to sort in ascending, descending or alphabetical order we can use built-in functions.

I will also drop country, notes and type of data for now, but I will come back to these two columns later. Just to simplify the cleaning process I will be dropping them in this step. It will look like this after dropping extra columns:

	country_code	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12	Unnamed: 13	Unnamed: 14	Unnamed: 15
13	Country code	International migrant stock at mid-year (both ...	NaN	NaN	NaN	NaN	NaN	International migrant stock at mid-year (male)	NaN	NaN	NaN	NaN
14	NaN	1990	1995	2000	2005	2010.0	2015.0	1990	1995	2000	2005	2010.0
15	900	152563212	160801752	172703309	191269100	221714243.0	243700236.0	77747510	81737477	87884839	97866674	114613714.0
16	901	82378628	92306854	103375363	117181109	132560325.0	140481955.0	40263397	45092799	50536796	57217777	64081077.0
17	902	70184584	68494898	69327946	74087991	89153918.0	103218281.0	37484113	36644678	37348043	40648897	50532637.0
18	941	11075966	11711703	10077824	9809634	10018128.0	11951316.0	5843107	6142712	5361902	5383009	5462714.0
19	934	59105261	56778501	59244124	64272611	79130668.0	91262036.0	31641006	30501966	31986141	35265888	45069923.0
20	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

I've added an if-condition for Table 4 as it does not have a Type of data column.

```
#rename 3rd column as country code
df.rename({'Unnamed: 3' : 'country_code' }, axis = 'columns' , inplace = True)

#drop sort order, notes and type of data columns
df.drop(columns=['Unnamed: 0' , 'Unnamed: 1' , 'Unnamed: 2'] , inplace = True)

#condition because table 2 does not have type of data so we need to skip for table 2
if(key!=dict_['Table 2']):
    df.drop(columns=['Unnamed: 4'] , inplace = True)
```

Next, we can see in excel file, each table has a main heading and sub-headings. Both headings contain important information that should be value and not variables. The top one gives us the gender and the second one gives us the year. Now we run **rename_columns** function.

```
def rename_columns(df1):
    #label read the value from df where the column header from excel file is stored
    label = df1.iloc[0][df1.columns[1]]

    #loop starting from second column till end because we dont want to rename country code
    lst = list(df1.columns[1:len(df1.columns)])

    for each in lst:
        #condition to check if first index(column header) is still the same
        #when value at 0 index changes label will be updated with the new value
        if(pd.notnull(df1.iloc[0][each])):
            label = df1.iloc[0][each]

        #code to rename column
        df1.rename({each : str(df1.iloc[1][each])+str(label).lower()} , axis = 'columns' , inplace = True)

    return df1
```

So, when I run rename function it will rename each ‘unnamed’ column based on a **label** which is the top heading in excel file stored in first row in our dataframe. This label is concatenated with the year which is taken from the second row and stored as column name.

	country_code	1990international migrant stock at mid-year (both sexes)	1995international migrant stock at mid-year (both sexes)	2000international migrant stock at mid-year (both sexes)	2005international migrant stock at mid-year (both sexes)	2010.0international migrant stock at mid-year (both sexes)	2015.0international migrant stock at mid-year (both sexes)	1990international migrant stock at mid-year (male)	1995inte migran mid-y
13	Country code	International migrant stock at mid-year (both ...	NaN	NaN	NaN	NaN	NaN	International migrant stock at mid-year (male)	
14	NaN	1990	1995	2000	2005	2010.0	2015.0	1990	
15	900	152563212	160801752	172703309	191269100	221714243.0	243700236.0	77747510	
16	901	82378628	92306854	103375363	117181109	132560325.0	140481955.0	40263397	
17	902	70184584	68494898	69327946	74087991	89153918.0	103218281.0	37484113	
18	941	11075966	11711703	10077824	9809634	10018128.0	11951316.0	5843107	
19	934	59105261	56778501	59244124	64272611	79130668.0	91262036.0	31641006	
20	947	14690319	15324570	13716539	13951086	15496764.0	18993986.0	7745306	
21	903	15690623	16352814	14800306	15191146	16840014.0	20649557.0	8279564	
22	910	5964031	5022742	4844795	4745792	4657063.0	6129113.0	3071189	

We can also run **df.columns** to quickly check if all the columns have been correctly renamed.

```
Index(['country_code',
       '1990international migrant stock at mid-year (both sexes)',
       '1995international migrant stock at mid-year (both sexes)',
       '2000international migrant stock at mid-year (both sexes)',
       '2005international migrant stock at mid-year (both sexes)',
       '2010.0international migrant stock at mid-year (both sexes)',
       '2015.0international migrant stock at mid-year (both sexes)',
       '1990international migrant stock at mid-year (male)',
       '1995international migrant stock at mid-year (male)',
       '2000international migrant stock at mid-year (male)',
       '2005international migrant stock at mid-year (male)',
       '2010.0international migrant stock at mid-year (male)',
       '2015.0international migrant stock at mid-year (male)',
       '1990international migrant stock at mid-year (female)',
       '1995international migrant stock at mid-year (female)',
       '2000international migrant stock at mid-year (female)',
       '2005international migrant stock at mid-year (female)',
       '2010.0international migrant stock at mid-year (female)',
       '2015.0international migrant stock at mid-year (female)'],
      dtype='object')
```

Next, comes the fun part where we transform the table from a wider one to a longer one. All the values in the column names that should be inside the table will be transferred. We will use melt function for this transformation.

```
#use melt function to change orientation of data
#two new columns will be formed tmp will contain all the column headers
#key value that we stored in dict_ will form the other columns

df_v1 = pd.melt(df, id_vars = 'country_code',
                value_vars = list(df.columns[1:len(df.columns)]),
                var_name = "tmp" ,
                value_name = key)
```

Now our data set looks like this. We can immediately notice the increase in number of rows.

	country_code	tmp	IMS
0	Country code	1990international migrant stock at mid-year (b...	International migrant stock at mid-year (both ...
1	NaN	1990international migrant stock at mid-year (b...	1990
2	900	1990international migrant stock at mid-year (b...	152563212
3	901	1990international migrant stock at mid-year (b...	82378628
4	902	1990international migrant stock at mid-year (b...	70184584
5	941	1990international migrant stock at mid-year (b...	11075966
6	934	1990international migrant stock at mid-year (b...	59105261
7	947	1990international migrant stock at mid-year (b...	14690319
8	903	1990international migrant stock at mid-year (b...	15690623
9	910	1990international migrant stock at mid-year (b...	5964031
10	108	1990international migrant stock at mid-year (b...	333110
11	174	1990international migrant stock at mid-year (b...	14079
12	262	1990international migrant stock at mid-year (b...	122221
13	232	1990international migrant stock at mid-year (b...	11848
14	231	1990international migrant stock at mid-year (b...	1155390
15	404	1990international migrant stock at mid-year (b...	297292
16	450	1990international migrant stock at mid-year (b...	23917
17	454	1990international migrant stock at mid-year (b...	1127724
18	480	1990international migrant stock at mid-year (b...	3613
19	175	1990international migrant stock at mid-year (b...	15229

Table 1 contains figures on International migrant stock which is why the value table is renamed as **'IMS'.**

This dictionary contains what column from each table will be renamed as when it is melted down into rows.

```
#intialize a dictionary for all the tables as keys and information we extract from each
#table as its value
#IMS is International migrant stock
#RMS refugee migrant stock

dict_ = {'Table 1' : 'IMS' ,
         'Table 2' : 'total_population',
         'Table 3' : 'IMS_total_population' ,
         'Table 4' : 'female_IMS',
         'Table 5' : 'IMS_ROC' ,
         'Table 6' : ['RMS' , 'RMS_IMS','RMS_ROC']}
```

Principle 2: Multiple variables are stored in one column.

Next, our new **'tmp'** column violates the second principle. It contains two different variables which is sex and year. We will split this column into two different columns for sex and year.

```
#extract year from tmp column
df_v1['year']  = df_v1.tmp.str.extract(pat = '([0-9-]+)')

#extract sex from tmp column
df_v1['sex']  = df_v1.tmp.str.extract(pat = '(?i)(both|male|female)')
```

Using regular expression and extract function we will separate the two values. After doing this step we will drop the **tmp** column.

	country_code	IMS	year	sex
0	Country code	International migrant stock at mid-year (both ...	1990	both
1	NaN	1990	1990	both
2	900	152563212	1990	both
3	901	82378628	1990	both
4	902	70184584	1990	both
...	...	...	...	...
4801	882	2460.0	2015	female
4802	772	254.0	2015	female
4803	776	2604.0	2015	female
4804	798	63.0	2015	female
4805	876	1411.0	2015	female

4806 rows × 4 columns

..

These steps are followed for all the six tables. I have written them in a function “clean_table”, which is called for each of the tables using a for loop.

Lst_ stores a list of all the cleaned dataframes.

```
for each in lst_:
    print(display(each))
```

	country_code	total_population	year	sex
0	Country code	Total population of both sexes at mid-year (th...	1990	both
1	NaN	1990	1990	both
2	900	5309667.699	1990	both
3	901	1144463.062	1990	both
4	902	4165204.637	1990	both
...	...	...	...	...
4801	882	93.584	2015	female
4802	772	..	2015	female
4803	776	52.931	2015	female
4804	798	..	2015	female
4805	876	..	2015	female

For Table 6, the first does not contain values in fact those are the actual variables but second has all the years. I divided the dataframe in 3 parts and cleaned and melted each of the frames separately.

```
for key in dict_:
    df = pd.read_excel(io="UN_MigrantStockTotal_2015.xlsx", sheet_name= key , index_col=False)

    if(key == 'Table 6'):
        df1 = clean_table(df[list(df.columns[0:10])].copy(), dict_[key][0])
        df2 = pd.merge(df2,df1 , on = ['country_code' , 'year' , 'sex'], how = 'left')

        df1 = clean_table(df[list(df.columns[[0,1,2,3,4,11,12,13,14,15,16]])].copy() , dict_[key][1])
        df2 = pd.merge(df2,df1 , on = ['country_code' , 'year'], how = 'left').drop(columns = ['sex_y'])

        df1 = clean_table(df[list(df.columns[[0,1,2,3,4,17,18,19,20,21]])].copy() , dict_[key][2])
        flag = True

    else:
        df1 = clean_table(df , dict_[key])
```

The tables would look like this:

	country_code	RMS	year	sex
0	Country code	Estimated refugee stock at mid-year (both sexes)	1990	both
1	NaN	1990	1990	both
2	900	18836571	1990	both

None

	country_code	RMS_IMS	year	sex
0	Country code	Refugees as a percentage of the international ...	1990	NaN
1	NaN	1990	1990	NaN
2	900	12.346732	1990	NaN

None

	country_code	RMS_ROC	year	sex
0	Country code	Annual rate of change of the refugee stock	1990-1995	NaN
1	NaN	1990-1995	1990-1995	NaN
2	900	-2.123497	1990-1995	NaN

None

Principle 3: Variables are stored in both rows and columns.

For this principle I'll come back to the two columns I dropped earlier. So, 'notes' column has all these references which are explained in 'NOTES' sheet. By reading the two, the 'ANNEX' sheet made a lot of sense. All the variables that were discussed in notes were made into columns in ANNEX.

It made sense to join the two tables based on **country_code** instead of writing a lot of code to separate the variables in notes column into individual 'columns'

I cleaned annex in a similar pattern like how I cleaned other tables, and it looks like this.

	country_code	country	major_area	major_area_code	region	region_code	developed_region	least_developed_country	sub_saharan_africa
0	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No
1	8	Albania	Europe	908	Southern Europe	925	Yes	No	No
2	12	Algeria	Africa	903	Northern Africa	912	No	No	No
3	16	American Samoa	Oceania	909	Polynesia	957	No	No	No
4	20	Andorra	Europe	908	Southern Europe	925	Yes	No	No
...	...	...	...	...	...	...	...	...	...
227	876	Wallis and Futuna Islands	Oceania	909	Polynesia	957	No	No	No
228	732	Western Sahara	Africa	903	Northern Africa	912	No	No	No
229	887	Yemen	Asia	935	Western Asia	922	No	Yes	No
230	894	Zambia	Africa	903	Eastern Africa	910	No	Yes	Yes
231	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes

232 rows × 9 columns

Annex has three different columns for country, region and major area. These three different variable were confined to a single column in excel which was violating the 3rd principle. Annex also have three different codes for country, region and major area. These are not our indexes, but it might be important for analysis, so we are keeping these. I got rid of three **sort_order** columns because they weren't giving an important information and also if we want to group our data we can always use **group by** built-in function.

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
Eritrea
Ethiopia
Kenya
Madagascar
Malawi
Mauritius
Mayotte
Mozambique
Réunion
Rwanda
Seychelles
Somalia
South Sudan
Uganda
United Republic of Tanzania
Zambia
Zimbabwe
Middle Africa
Angola

When we join our table with annex using left-join that means only the country codes in the annex table will be considered. We will instantly lose all the extra rows of derived data on regions and major area.

	country_code	country	major_area	major_area_code	region	region_code	developed_region	least_developed_country	sub_saharan_africa	
0	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No	
1	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No	
2	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No	
3	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No	
4	4	Afghanistan	Asia	935	Southern Asia	5501	No	Yes	No	10
...	...	...	...	...	...	...	...	...	...	
4171	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes	
4172	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes	
4173	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes	
4174	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes	17
4175	716	Zimbabwe	Africa	903	Eastern Africa	910	No	No	Yes	17

4176 rows × 12 columns

Our data is referring to different countries and those countries have further attributes such as region, major area, economic condition etc. So, if we want to analyze it according to region, we can do it this way:

Secondly, we can also observe here that **developed_region** and **least_developed_region** is both actually values for how a country is classified. A country can be either more developed or less developed or least developed. I combined these two-column using simple **replace** function.

```
#make one column for underdeveloped and developed
final_table = pd.merge(tmp, datatype, on = 'country_code' , how = 'left')
final_table.replace({'developed_region' : {'Yes' : 'more' , 'No' : 'less'}}, inplace = True)
final_table.replace({'least_developed_country' : {'Yes' : 'least' , 'No' : 'less'}}, inplace = True)
final_table['country_classification'] = final_table['developed_region'] + final_table['least_developed_country']
final_table['country_classification'].replace('lessleast', 'least', inplace=True)
final_table['country_classification'].replace('moreless', 'more', inplace=True)
final_table['country_classification'].replace('lessless', 'less', inplace=True)
final_table.drop(columns = ['developed_region' , 'least_developed_country'] , inplace = True)
```

After running these lines our data would look like this,

IMS	year	sex	total_population	IMS_total_population	female_IMS	RMS	RMS_IMS	foreign_pop_status	refugee_incl	imputation	country_classification
7686	1990	both	12067.57	0.478025	NaN	25	0.043338	born	no	no	least
1522	1995	both	16772.522	0.426424	NaN	19605	27.411146	born	no	no	least
5917	2000	both	19701.94	0.385328	NaN	0	0	born	no	no	least
7300	2005	both	24399.948	0.357788	NaN	32	0.036655	born	no	no	least
46.0	2010	both	27962.207	0.365658	NaN	6434.0	6.292667	born	no	no	least
...	...	...	...	...	...	...	...	...	...	...	...
5214	1995	female	5877.504	3.151236	42.950564	NaN	0.119195	born	yes	no	less
5198	2000	female	6280.133	2.805641	42.970825	NaN	1.006485	born	yes	no	less
3723	2005	female	6548.18	2.57664	42.965625	NaN	1.129381	born	yes	no	less
24.0	2010	female	7068.861	2.417985	42.957493	NaN	1.114627	born	yes	no	less
87.0	2015	female	7915.194	2.166555	42.993637	NaN	1.353086	born	yes	no	less

Thirdly, at this point at will come back to notes column as that contains important information about what is included in data that could help us make sense of analysis. So, I will read type of data column and translate reference code into different columns. From notes we know that B and C refer to foreign-born and foreign citizen respectively, R refer to whether refugee data is included or not and I refer to imputation.

We run **get_data_type** function,

```python
def get_data_type(df):
    df.drop(df.index[0:23] , inplace = True)
    df.rename({'Unnamed: 3' : 'country_code' , 'Unnamed: 4' : 'type_of_data'} , axis = 'columns' , inplace = True)
    df1 = df[['country_code' , 'type_of_data']]
    df1['foreign_pop_status']  = df1.type_of_data.str.extract(pat = '(B|C)')
    df1['refugee_incl']  = df1.type_of_data.str.extract(pat = '(R)')
    df1['imputation']  = df1.type_of_data.str.extract(pat = '(I)')
    df1.replace({'foreign_pop_status' : {'B' : 'born' , 'C' : 'citizen'}}, inplace = True)
    df1.replace({'refugee_incl' : {'R' : 'yes' , np.nan : 'no'}}, inplace = True)
    df1.replace({'imputation' : {'I' : 'yes' , np.nan : 'no'}} , inplace = True)
    df1.drop(columns = 'type_of_data' , inplace = True)
    df1.reset_index(drop = True , inplace = True)
    return df1
```

This will extract reference keyword and make separate columns for each attribute of data.

MS_IMS	foreign_pop_status	refugee_incl	imputation	country
).043338	born	no	no	
7.411146	born	no	no	
0	born	no	no	
).036655	born	no	no	
5.292667	born	no	no	
...	...	...	...	
).119195	born	yes	no	
1.006485	born	yes	no	
1.129381	born	yes	no	
1.114627	born	yes	no	
1.353086	born	yes	no	

After combining these three columns with our main table it will look like this.

Principle 4: Multiple types of observational units are stored in the same table.

In Table 6, we observed that rate of change in international migrant stock is stored in the same table. Change of rate tells us a change over a given range of years. It can not be the part of the same table as it would be contradicting to the other 'year' column we have. So, we make a separate for the two rate of changes, one for refugee migrant stock and the other one for international migrant stock.

```
if(df3.empty):
    df3 = df1
    flag = False
else:
    df3 = pd.merge(df3,df1, on = ['country_code' , 'year'], how = 'left').drop(columns = ['sex_y'])
    df3.rename({'sex_x' : 'sex'} , axis = 'columns', inplace = True)
    flag = False
?.rename({'sex_x' : 'sex'} , axis = 'columns', inplace = True)
```

Here df3 will store data on rate of change variable.

	country_code	IMS_ROC	year	sex	RMS_ROC
0	4	4.299812	1990-1995	both	128.99347
1	4	1.192711	1995-2000	both	..
2	4	2.794196	2000-2005	both	..
3	4	3.160624	2005-2010	both	102.911692
4	4	26.37988	2010-2015	both	50.501739
...	...	...	...	...	...
3475	716	-7.890123	1990-1995	female	-110.036176
3476	716	-0.998071	1995-2000	female	42.669158
3477	716	-0.867001	2000-2005	female	2.304118
3478	716	0.259214	2005-2010	female	-0.262999
3479	716	0.065769	2010-2015	female	3.877364

3480 rows × 5 columns

Principle 5: A single observational unit is stored in multiple tables.

All the tables have a single observational unit such as table 1 contains estimates for International migrant stock and table 2 contains total population, in order to conduct analysis, we will have to combine the two tables.

For instance, if we want to see which year the stock for IMS changed drastically? Was it due to a decrease in female population?

Another reason for joining/merging all the tables is that it will be quicker for analysis, unlike relational databases, in analysis we must see data from different angles and perspectives so keeping data in small individual units would mean that every time we want to conduct some research, we will have to combine 2 or 3 tables. Join is an expensive operation as it compares each row of one dataframe with another so depending on the size of the dataframe could take long. Therefore, I joined columns from each table with one another.

	country_code	country	major_area	major_area_code	region	region_code	sub_saharan_africa	IMS	year	sex	total_population	IMS_
0	4	Afghanistan	Asia	935	Southern Asia	5501	No	57686	1990	both	12067.57	
1	4	Afghanistan	Asia	935	Southern Asia	5501	No	71522	1995	both	16772.522	
2	4	Afghanistan	Asia	935	Southern Asia	5501	No	75917	2000	both	19701.94	
3	4	Afghanistan	Asia	935	Southern Asia	5501	No	87300	2005	both	24399.948	
4	4	Afghanistan	Asia	935	Southern Asia	5501	No	102246.0	2010	both	27962.207	
...	...	...	...	...	...	...	...	...	...	...	...	
4171	716	Zimbabwe	Africa	903	Eastern Africa	910	Yes	185214	1995	female	5877.504	
4172	716	Zimbabwe	Africa	903	Eastern Africa	910	Yes	176198	2000	female	6280.133	
4173	716	Zimbabwe	Africa	903	Eastern Africa	910	Yes	168723	2005	female	6548.18	
4174	716	Zimbabwe	Africa	903	Eastern Africa	910	Yes	170924.0	2010	female	7068.861	
4175	716	Zimbabwe	Africa	903	Eastern Africa	910	Yes	171487.0	2015	female	7915.194	

4176 rows × 19 columns

Discussion:

After following the five principles of tidy data, I am left with just two final tables. One for rate of change and the other one consists of data on International Migrant Stock and Refugee Migrant Stock. This table gives us different attributes of International Migrant Stock.

Table 1:

	Column	Data	Description
1.	column_code	Alphanumeric	Identifier for each country
2.	country	String	List of countries
3.	major_area	String	Major area each country belong to
4.	major_area_code	Alphanumeric	Identifier for a major area
5.	region	String	Region each country belongs to
6.	region_code	Alphanumeric	Identifier for a region
7.	sub_saharan_africa	Yes/no	Whether a country is part of sub-saharan Africa?
8.	IMS	Decimal	Estimate of International Migrant Stock
9.	Year	Numeric	Year
10.	sex	String	Gender (male/female/both)
11.	total_population	Numeric	Population of stock
12.	IMS_total_population	Numeric	IMS as a percentage of total population
13.	female_IMS	Numeric	IMS female population as a percentage of total population
14.	RMS	Numeric	Estimates on Refugee Migrant Stock
15.	RMS_IMS	Numeric	Refugee as a percentage of international migrant stock
16.	foreign_pop_status	String	Whether population was foreign born or foreign citizen?
17.	refugee_incl	String	Were refugee included in the estimates of international migrant stock?
18.	Imputation	String	Whether estimates for a country having no data on the number of international migrants were obtained by imputation? Yes/Np
19.	country_classification	String	Whether a country is least developed, less developed or more developed.

Table 2:

	Column	Data	Description
1.	Country_code	Alphanumeric	Identifier for each country
2.	IMS_ROC	Numeric	rate of change in international migrant stock
3.	year	Numeric	Range of years over which rate of change observed
4.	Sex	String	Male/female/both
5.	RMS_ROC	numeric	Rate of change in refugee migrant stock

Conclusion:

We can observe that following tidy data principles transforms data set and prepares it for analysis. It is an entirely different way of looking at a dataset. However, I have also concluded that there's a lot more to do with a dataset and this is only the first step. For instance, one question that came to my mind was how do we deal with nan values?

There are many more steps we take to inform uniformity and consistency in data set. But by following these five steps the following steps make more sense.