Advice

Based on the finding I created the following classes:

- Cleaner: to clean the data `preprocessing.py` .
- Selector and Aggregator: to select data and create aggregates `preparation.py` .
- Dashboard for visualization `main.py`

Data Understanding

Only analyzes the dataset of The Netherlands

- Part 1
  Life Expectancy at Birth
- Part 2
  Country Codes
- Part 3
  Population (POP)
- Part 4
  Morticd10 part 1 - 5

This notebook will describe the different data files and how they are built.

I'll use the following files:

- life_expectancy_at_birth.xlsx
- country_codes
- pop
- Morticd10_part1
- Morticd10_part2
- Morticd10_part3
- Morticd10_part4
- Morticd10_part5

In [1]:
```
from pathlib import Path
import re

import pandas as pd
import numpy as np
```

In [2]:
```
# use config yaml
datadir = Path('../data')
```

1) Life Expectancy at Birth

Source: https://www.who.int/data/maternal-newborn-child-adolescent-ageing/indicator-explorer-new/mca/life-expectancy-at-birth

Indicator name: Life expectancy at birth (years)

The indicator reflects the overall mortality level of a population. Age groups include:

- Children, adolescents, adults and elderly.

Definition: The average number of years that a newborn could expect to live, if he or she were to pass through life exposed to the sex- and age-specific death rates prevailing at the time of his or her birth, for a specific year, in a given country, territory, or geographic area.

WHO used a specific method of estimation which can deviate from the official estimation of the Member State. I compared the data and in practise the deviations are very small (< 0.5). The WHO also provides more data, thus it outweights the cons.

i) Meaning of columns

In [4]:
```
df = pd.read_excel(Path(datadir, 'netherlands_life_exp.xlsx'))
df = df.sort_values('Year')
df.head(3)
```

Out[4]:

	Year	WHO Region	World Bank Income Group	Country ISO Code	Country	Sex	Global	Value
0	1950	Europe	High income	NLD	Netherlands	Both sexes	NaN	71.411
1	1950	Europe	High income	NLD	Netherlands	Female	NaN	72.615
3	1950	Europe	High income	NLD	Netherlands	Male	NaN	70.236

Column names & their definitions:

Year: the year of focus according to the Gregorian calendar.
WHO region: the continent in the world.
World Bank Income Group: economic state of the *Country*.
Country ISO Code: the ISO code of the *Country*.
Country: the country of focus.
Sex: a categorical value representing the biological sex, consisting of *Both sexes*, *Female*, and *Male*.
Global: *meaning unknown and will not be used*.
Value: the life expectancy in that *Year*.

ii) Characteristics of Values

Based on the description of the columns above, the following columns are important for further analysis:

- General
- Year
- Sex
- Value

General

In [5]:
```
df.shape
```

Out[5]:
```
(450, 8)
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
Year                          0
WHO Region                    0
World Bank Income Group       0
Country ISO Code              0
Country                       0
Sex                           0
Global                      450
Value                         0
dtype: int64
```

In general there are 450 rows and 8 columns.There are no missing values that are important.

Every year consists of three rows, consisting of a row for male, female and both sexes data.

Year

In [7]:
```
df['Year'].dtype
```

Out[7]:
```
dtype('int64')
```

In [8]:
```
df['Year'].unique().size
```

Out[8]:
```
150
```

In [9]:
```
# Check the lowest and highest year
min_y = df['Year'].min()
max_y = df['Year'].max()

# Check if the years are sequential
if (np.arange(min_y, max_y + 1) == df['Year'].unique()).all():
    print('The years range from {} to {}.'.format(min_y, max_y))
else:
    print('THE YEARS ARE NOT SEQUENTIAL. THERE ARE MISSING YEARS!')
```

```
The years range from 1950 to 2099.
```

Sex

In [10]:
```
df['Sex'].dtype
```

Out[10]:
```
dtype('O')
```

In [11]:
```
df['Sex'].unique().size
```

Out[11]:
```
3
```

In [12]:
```
df['Sex'].unique()
```

Out[12]:
```
array(['Both sexes', 'Female', 'Male'], dtype=object)
```

The `Sex` column can have one of the following values: `Both sexes` , `Female` , `Male` .

Value

In [13]:
```
df['Value'].dtype
```

Out[13]:
```
dtype('float64')
```

In [14]:
```
df['Value'].unique().size
```

Out[14]:
```
448
```

In [15]:
```
min_v = df['Value'].min()
max_v = df['Value'].max()

min_i = df['Value'].argmin()
max_i = df['Value'].argmax()

print('Lowest life expectancy measured: {} in {}'.format(min_v, df['Year'].iloc[min_i]))
print('Highest life expectancy measured: {} in {}'.format(max_v, df['Year'].iloc[max_i]))
```

```
Lowest life expectancy measured: 70.236 in 1950
Highest life expectancy measured: 92.776 in 2099
```

The life expectancy was lowest in 1950 and will be expected to be the highest in 2099. WHO has extrapolated the life expectancy to 2099. This means, it is just an estimation and if values in the future will be used, then it must be used with caution.

The `Value` column name is not descriptive enough. It is advices to change it to `Life expectancy at birth` .

CONCLUSION:

Every year that is measured consists of 3 rows:

- Row 1 contains the life expectancy for Males and Females together (mean).
- Row 2 contains the life expectancy for Females.
- Row 3 contains the life expectancy for Males.

There are *NO* missing values.

The years range from `1950` till `2099` using extrapolation.

The `Value` column is not descriptive. Change the name to `Life expectancy at birth` .

---

2) Country Codes

- Part 1
  Life Expectancy at Birth
- Part 2
  Country Codes
- Part 3
  Population (POP)
- Part 4
  Morticd10 part 1 - 5

In [16]:
```
df = pd.read_csv(Path(datadir, 'country_codes'))
df.head(3)
```

Out[16]:

	country	name
0	1010	Algeria
1	1020	Angola
2	1025	Benin

Column names & their definitions:

country: The country number.
name: the name of the country.

In [17]:
```
df[df['name'] == 'Netherlands']
```

Out[17]:

	country	name
186	4210	Netherlands

CONCLUSION:

The Netherlands has country code `4210` .

---

3) Population POP

- Part 1
  Life Expectancy at Birth
- Part 2
  Country Codes
- Part 3
  Population POP
- Part 4
  Morticd10 part 1 - 5

In [18]:
```
df = pd.read_csv(Path(datadir, 'pop'))
df.head(3)
```

Out[18]:

	Country	Admin1	SubDiv	Year	Sex	Frmat	Pop1	Pop2	Pop3	Pop4	...	Pop18	Pop19	Pop20	Pop21	Pop22	Pop23
0	1060	NaN	NaN	1980	1	7	137100.0	3400.0	15800.0	NaN	...	NaN	5300.0	NaN	2900.0	NaN	NaN
1	1060	NaN	NaN	1980	2	7	159000.0	4000.0	18400.0	NaN	...	NaN	6200.0	NaN	3400.0	NaN	NaN
2	1125	NaN	NaN	1955	1	2	5051500.0	150300.0	543400.0	NaN	...	110200.0	51100.0	41600.0	14300.0	11800.0	25300.0

3 rows × 33 columns

Column names & their definitions:

Country: the country code, as defined in part 2.
Admin1: specified pertinent. If blank, data refers to the country.
SubDiv: Category of data, Annex Table 2 in the manual. If blank, data refers to the country.
Year: The year to which data refer.
Sex: categorical value where 1 means male and 2 means female.
Frmat: Age-group format for breakdown, see Annex Table 1 in the manual.
Pop1: Population at all ages.
Pop2: Population at age 0.
Pop3: Poplatuion at age 1.
Pop4: Poplatuion at age 2.
Pop5: Poplatuion at age 3.
Pop6: Poplatuion at age 4.
Pop7: Poplatuion at age 5-9.
Pop8: Poplatuion at age 10-14
Pop9: Poplatuion at age 15-19
Pop10: Poplatuion at age 20-24
Pop11: Poplatuion at age 25-29
Pop12: Poplatuion at age 30-34
Pop13: Poplatuion at age 35-39.
Pop14: Poplatuion at age 40-44
Pop15: Poplatuion at age 45-49
Pop16: Poplatuion at age 50-54
Pop17: Poplatuion at age 55-59
Pop18: Poplatuion at age 60-64
Pop19: Poplatuion at age 65-69
Pop20: Poplatuion at age 70-74
Pop21: Poplatuion at age 75-79
Pop22: Poplatuion at age 80-84
Pop23: Poplatuion at age 85-89
Pop24: Poplatuion at age 90-94
Pop25: Poplatuion at age 95 and over
Pop26: Poplatuion at age unspecified
Lb: Live births. *What this actually means is unknown.

ii) Characteristics of Values

Based on the description of the columns above, the following columns are important for further analysis:

- General
- Year
- Sex
- Pop1, we are only interested in the whole population.

General

In [19]:
```
nl = df[df['Country'] == 4210]
```

In [20]:
```
nl.head()
```

Out[20]:

	Country	Admin1	SubDiv	Year	Sex	Frmat	Pop1	Pop2	Pop3	Pop4	...	Pop18	Pop19	Pop20	Pop21	Pop22
7055	4210	NaN	NaN	1950	1	1	5041000.0	117200.0	121100.0	127600.0	...	183300.0	147700.0	111500.0	68500.0	32700.0
7056	4210	NaN	NaN	1950	2	1	5072500.0	110700.0	114200.0	120700.0	...	193300.0	157000.0	119800.0	76100.0	38300.0
7057	4210	NaN	NaN	1951	1	1	5114800.0	115200.0	116800.0	121000.0	...	186400.0	151600.0	113800.0	71800.0	33800.0
7058	4210	NaN	NaN	1951	2	1	5149500.0	108800.0	110400.0	114100.0	...	197500.0	161400.0	122700.0	79300.0	39500.0
7059	4210	NaN	NaN	1952	1	1	5171900.0	115600.0	114300.0	116100.0	...	189800.0	155100.0	116200.0	75100.0	35300.0

5 rows × 33 columns

In [21]:
```
nl.shape
```

Out[21]:
```
(138, 33)
```

In [22]:
```
nl.isna().sum()
```

Out[22]:
```
Country      0
Admin1     138
SubDiv     138
Year         0
Sex          0
Frmat        0
Pop1         0
Pop2         0
Pop3         0
Pop4         0
Pop5         0
Pop6         0
Pop7         0
Pop8         0
Pop9         0
Pop10        0
Pop11        0
Pop12        0
Pop13        0
Pop14        0
Pop15        0
Pop16        0
Pop17        0
Pop18        0
Pop19        0
Pop20        0
Pop21        0
Pop22        0
Pop23        0
Pop24      100
Pop25      100
Pop26        0
Lb           0
dtype: int64
```

The columns that are important do not have missing values.

The dataframe consists of two rows for each year, where one row given information about males and the other about females.

Year

In [23]:
```
nl['Year'].head(10)
```

Out[23]:
```
7055    1950
7056    1950
7057    1951
7058    1951
7059    1952
7060    1952
7061    1953
7062    1953
7063    1954
7064    1954
Name: Year, dtype: int64
```

In [24]:
```
# Check the lowest and highest year
min_y = nl['Year'].min()
max_y = nl['Year'].max()

# Check if the years are sequential
if (np.arange(min_y, max_y + 1) == nl['Year'].unique()).all():
    print('The years range from {} to {}.'.format(min_y, max_y))
else:
    print('THE YEARS ARE NOT SEQUENTIAL. THERE ARE MISSING YEARS!')
```

```
The years range from 1950 to 2018.
```

Sex

In [25]:
```
nl['Sex'].dtype
```

Out[25]:
```
dtype('int64')
```

In [26]:
```
nl['Sex'].unique()
```

Out[26]:
```
array([1, 2], dtype=int64)
```

Where 1 means male, and 2 means female. It is possible to change this to categorical values as a string with a mapping.

Pop1

In [27]:
```
nl['Pop1'].dtype
```

Out[27]:
```
dtype('float64')
```

In [28]:
```
# Total population in 2010
# CBS returns 16 580 000
# The World Bank returns 16 620 000
nl[nl['Year'] == 2010]['Pop1'].sum()
```

Out[28]:
```
16615394.0
```

There is a slight divergence between the data from WHO and CBS, but it is negligible.

CONCLUSION:

Every year consists of two rows, namely:

- Row 1 represents the male population.
- Row 2 represents the female population.

Year ranges from `1950` to `2018` .

'Sex' consists of a mapping: 1 -> Male, 2 -> Female.

---

4) Morticd10

- Part 1
  Life Expectancy at Birth
- Part 2
  Country Codes
- Part 3
  Population POP
- Part 4
  Morticd10

In total, there are five files that make up the mortality rate.

- Morticd10_part1
- Morticd10_part2
- Morticd10_part3
- Morticd10_part4
- Morticd10_part5

i) Meaning of columns

All files contain the same columns.

In [29]:
```
df = pd.read_csv(Path(datadir, 'Morticd10_part1'))
df.head(3)
```

```
C:\Users\ddlat\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeWarning: Columns (4) hav
e mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[29]:

	Country	Admin1	SubDiv	Year	List	Cause	Sex	Frmat	IM_Frmat	Deaths1	...	Deaths21	Deaths22	Deaths23	Deaths24	Deaths25
0	1125	NaN	NaN	2000	103	A00	1	2	8	2	...	0	0.0	0.0	NaN	NaN
1	1125	NaN	NaN	2000	103	A01	1	2	8	27	...	0	0.0	1.0	NaN	NaN
2	1125	NaN	NaN	2000	103	A02	1	2	8	3	...	0	0.0	1.0	NaN	NaN

3 rows × 39 columns

In [30]:
```
df['Year']
```

Out[30]:
```
0          2000
1          2000
2          2000
3          2000
4          2000
           ...
1021813    2002
1021814    2002
1021815    2002
1021816    2002
1021817    2002
Name: Year, Length: 1021818, dtype: int64
```

Column names & their definitions:

Country: the country code, as defined in part 2.
Admin1: specified pertinent. If blank, data refers to the country.
SubDiv: Category of data, Annex Table 2 in the manual. If blank, data refers to the country.
Year: The year to which data refer.
List: List of ICD revision used, see Annex Table 2 in the manual.
Cause: Code of Cause of Death.
Sex: categorical value where 1 is male, 2 is female, and 9 is unspecified.
Frmat: Age-group format breakdown of deaths, see Annex Table 1 in the manual.
IM_Frmat: Age format for breakdown of infant deaths (0 year). see Annex Table 1 in the manual.
Deaths1: Deaths at all ages.
Deaths2: Deaths at age 0.
Deaths3: Deaths at age 1.
Deaths4: Deaths at age 2.
Deaths5: Deaths at age 3.
Deaths6: Deaths at age 4.
Deaths7: Deaths at age 5-9.
Deaths8: Deaths at age 10-14.
Deaths9: Deaths at age 15-19.
Deaths10: Deaths at age 20-24.
Deaths11: Deaths at age 25-29.
Deaths12: Deaths at age 30-34.
Deaths13: Deaths at age 35-39.
Deaths14: Deaths at age 40-44.
Deaths15: Deaths at age 45-49.
Deaths16: Deaths at age 50-54.
Deaths17: Deaths at age 55-59.
Deaths18: Deaths at age 60-64.
Deaths19: Deaths at age 65-69.
Deaths20: Deaths at age 70-74.
Deaths21: Deaths at age 75-79.
Deaths22: Deaths at age 80-84.
Deaths23: Deaths at age 85-89.
Deaths24: Deaths at age 90-94.
Deaths25: Deaths at age 95 years and above
Deaths26: Deaths at age unspecified
IM_Deaths1: Infant deaths at age 0 day
IM_Deaths2: Infant deaths at age 1-6 days
IM_Deaths3: Infant deaths at age 7-27 days
IM_Deaths4: Infant deaths at age 28-364 days

ii) Characteristics of Values

Based on the description of the columns above, the following Factors are important for further analysis:

- General
- Year
- List
- Sex
- Cause
- Deaths1

As the data consists of 5 part, they should be concatenated.

In [31]:
```
df1 = pd.read_csv(Path(datadir, 'Morticd10_part1'))
df2 = pd.read_csv(Path(datadir, 'Morticd10_part2'))
df3 = pd.read_csv(Path(datadir, 'Morticd10_part3'))
df4 = pd.read_csv(Path(datadir, 'Morticd10_part4'))
df5 = pd.read_csv(Path(datadir, 'Morticd10_part5'))

data = [df1, df2, df3, df4, df5]
```

```
C:\Users\ddlat\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeWarning: Columns (4) hav
e mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\Users\ddlat\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeWarning: Columns (2) hav
e mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [32]:
```
# select The Netherlands
for i, d in enumerate(data):
    data[i] = d[d['Country'] == 4210]
```

In [33]:
```
# concatenate them into one
df = pd.concat(data).sort_values('Year')
```

General

In [34]:
```
df.shape
```

Out[34]:
```
(72376, 39)
```

In [35]:
```
df[df['Sex'] == 1].shape[0] + df[df['Sex'] == 2].shape[0]
```

Out[35]:
```
72376
```

In [36]:
```
df.isna().sum()
```

Out[36]:
```
Country           0
Admin1        72376
SubDiv        72376
Year              0
List              0
Cause             0
Sex               0
Frmat             0
IM_Frmat          0
Deaths1           0
Deaths2           0
Deaths3           0
Deaths4           0
Deaths5           0
Deaths6           0
Deaths7           0
Deaths8           0
Deaths9           0
Deaths10          0
Deaths11          0
Deaths12          0
Deaths13          0
Deaths14          0
Deaths15          0
Deaths16          0
Deaths17          0
Deaths18          0
Deaths19          0
Deaths20          0
Deaths21          0
Deaths22          0
Deaths23          0
Deaths24          0
Deaths25          0
Deaths26          0
IM_Deaths1        0
IM_Deaths2     6512
IM_Deaths3     6512
IM_Deaths4     6512
dtype: int64
```

The columns that are important do not have missing values.

Year

In [37]:
```
df['Year'].dtype
```

Out[37]:
```
dtype('int64')
```

In [38]:
```
df['Year'].unique()
```

Out[38]:
```
array([1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006,
       2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017,
       2018], dtype=int64)
```

In [39]:
```
# Check the lowest and highest year
min_y = df['Year'].min()
max_y = df['Year'].max()

# Check if the years are sequential
if (np.arange(min_y, max_y + 1) == df['Year'].unique()).all():
    print('The years range from {} to {}.'.format(min_y, max_y))
else:
    print('THE YEARS ARE NOT SEQUENTIAL. THERE ARE MISSING YEARS!')
```

```
The years range from 1996 to 2018.
```

List

List of ICD revision used, see Annex Table 2 in the manual.

In [40]:
```
df['List'].dtype
```

Out[40]:
```
dtype('O')
```

In [41]:
```
df['List'].unique()
```

Out[41]:
```
array(['10M', '104', 104], dtype=object)
```

`104` is saved as two seperate datatypes, namely as an `int` and a `str` . Therefore, convert the column to type `str` .

Two types of revisions are used, namelijk the `10M` and `104` revision. The revisions have the following description according to the documentation:

- `10M` : CD10 3 and 4 (detailed) character list. Pls note that when a 4th character code is given it is therefore not included in a 3 character code. All records are mutually exclusive.
- `104` : ICD10 4 (detailed) character list

Sex

categorical value where 1 is male, 2 is female, and 9 is unspecified.

In [42]:
```
df['Sex'].dtype
```

Out[42]:
```
dtype('float64')
```

In [43]:
```
df['Sex'].unique()
```

Out[43]:
```
array([1., 2.])
```

The Netherlands has data for male and female sexes.

Cause

Code of Cause of Death.

These codes can be found in the ICD

In [44]:
```
df['Cause'].dtype
```

Out[44]:
```
dtype('O')
```

In [45]:
```
df['Cause'].unique().size
```

Out[45]:
```
5243
```

There are in total 5243 reported causes of death. Only the noncommunicable diseases are relevant to answer the research question.

Code source for noncommunicable disease: https://www.euro.who.int/__data/assets/pdf_file/0007/350278/Fact-sheet-SDG-NCD-FINAL-25-10-17.pdf

In fact, a non-communicable disease may also be defined as a non-transmissible disease. This disease can be chronic or acute. In this research the focus is on chronic non-comunicable diseases that causes relatively the highest mortality rates.

It is not clearly defined when a disease is chronic or not. For this reason, all diseases that are specifically annotated as 'Acute' are left out.

The diseases have the following codes:

- Cardiovascular disease: ICD-10 codes `I05-I99`
- Cancer: ICD-10 codes `C00-C97`
- Diabetes mellitus: ICD-10 codes `E10-E13`
- Chronic respiratory diseases: ICD-10 codes `J40-47`
- Diseases of digestive system: ICD-10 codes `K00-K93`

The codes are formatted with 3 characters or 4 characters. In this case, we only need the first 3 characters.

In [46]:

```
def generate_ICD_codes(lower, upper, symbol):
    codes = []
    for i in range(lower, upper+1, 1):
        if i < 10:
            codes.append(f'{symbol}0{i}')
        else:
            codes.append(f'{symbol}{i}')

    return np.array(codes)

def get_unique_codes(df, symbol, column_name='Cause'):
    return list(df[df[column_name].str.contains(symbol)][column_name].unique())

def test_codes(codes):
    """Some codes can have ascii symbols, but first test if they are integer only.
    That makes the code somewhat easier."""
    # remove the first symbol
    codes = [code[1:] for code in codes]

    try:
        [int(code) for code in codes]
        print('Codes are valid.')
    except ValueError as e:
        print('Expected integers: ', e)
    except:
        print('Something else went wrong.')

def convert_format(series, n):
    """Only keep the n first characters of the column"""
    return series.apply(lambda x: x[:3])

def find_codes(codes, series):
    mask = np.isin(codes, series)
    found = np.where(mask, codes, '')
    valid = [c for c in found if c != '']

    return valid

C_codes = generate_ICD_codes(0, 97, 'C')
I_codes = generate_ICD_codes(5, 99, 'I')
E_codes = generate_ICD_codes(10, 13, 'E')
J_codes = generate_ICD_codes(40, 47, 'J')
K_codes = generate_ICD_codes(0, 93, 'K')

causes_3 = convert_format(df['Cause'], 3)
causes_3 = causes_3.unique()
```

Cancer

In [47]:

```
data_C_codes = get_unique_codes(df, 'C')

test_codes(data_C_codes)
```

```
Codes are valid.
```

In [48]:

```
# Codes belonging to cancer
C_N = C_codes.size

C_valid = find_codes(C_codes, causes_3)

print("Found in total {} codes of {} possible codes for cancer.".format(len(C_valid), C_N))
```

```
Found in total 86 codes of 98 possible codes for cancer.
```

Cardiovascular disease

In [49]:

```
data_I_codes = get_unique_codes(df, 'I')

test_codes(data_I_codes)
```

```
Codes are valid.
```

In [50]:

```
I_N = I_codes.size

valid = find_codes(I_codes, causes_3)

print("Found in total {} codes of {} possible codes for cardiovascular disease.".format(len(valid), I_N))
```

```
Found in total 61 codes of 95 possible codes for cardiovascular disease.
```

Diabetes mellitus: ICD-10 codes E10-E13

In [51]:

```
data_E_codes = get_unique_codes(df, 'E')

test_codes(data_E_codes)
```

```
Codes are valid.
```

In [52]:

```
E_N = E_codes.size

valid = find_codes(E_codes, causes_3)

print("Found in total {} codes of {} possible codes for diabetes mellitus.".format(len(valid), E_N))
```

```
Found in total 4 codes of 4 possible codes for diabetes mellitus.
```

Chronic respiratory diseases: ICD-10 codes J40-47

In [53]:

```
data_J_codes = get_unique_codes(df, 'J')

test_codes(data_J_codes)
```

```
Codes are valid.
```

In [54]:

```
J_N = J_codes.size

valid = find_codes(J_codes, causes_3)

print("Found in total {} codes of {} possible codes for chronic respiratory diseases.".format(len(valid), J_N))
```

```
Found in total 8 codes of 8 possible codes for chronic respiratory diseases.
```

Diseases of digestive system: ICD-10 codes K00-K93

In [55]:

```
data_K_codes = get_unique_codes(df, 'K')

test_codes(data_K_codes)
```

```
Codes are valid.
```

In [56]:

```
K_N = K_codes.size

valid = find_codes(K_codes, causes_3)

print("Found in total {} codes of {} possible codes for diseases of digestive system.".format(len(valid), K_N))
```

```
Found in total 64 codes of 94 possible codes for diseases of digestive system.
```

CONCLUSION:

Every year consists of two rows, namely:

- Row 1 represents the male population.
- Row 2 represents the female population.

The years range from 1996 to 2018.

'Sex' consists of a mapping: 1 -> Male, 2 -> Female.

There are:

- 86 registered codes for cancer.
- 61 registered codes for cardiovascular disease.
- 4 registered codes for diabetes mellitus.
- 8 registered codes for chronic respiratory diseases.
- 64 registered codes for diseases of digestive system.

In [ ]: