Unraveling separatism

A longitudinal study into the causes of separatist protest among communal groups in the period of 1990 to 2006.

Gereproduceerd met Python

Naam: Mart Groenen

Datum: 18-06-2020

Inleiding

De volgende variabelen heb ik in mijn masterscriptie geanalyseerd met behulp van SPSS:

Afhankelijke variabele:

- prot: Separatist protest (0 – 5, quasi-interval)

Onafhankelijke variabelen:

- protnearby: Separatist protest in nearby countries (quasi-interval: 0 - 5)
- kindred: Kindred group dispersion (ordinaal: 0 - 3)
- refugees: Refugee flows (ratio)
- ecdis: Economic discrimination (ordinaal: 0 – 4)
- poldis: Political discrimination (ordinaal: 0 – 4)
- intkinref: Interactie-effect tussen kindred en refugees
- intdis: Interactie-effect tussen ecdis en poldis

Controlevariabelen:

- democracy: Democracy score (interval: -10 – 10)
- GDP: GDP per capita (ratio)

De gebruikte dataset is panel data. Dit is data met meerdere niveaus. Andere gangbare termen zijn hierarchical data en nested data. De unit of analysis is de bevolkingsgroep. Dit is het laagste niveau. De andere niveau's zijn het land waar de bevolkingsgroep deel van uitmaakt, de regio waar het land binnen valt en het jaar van observeren (1990-2006). Ik heb een mixed model gebruikt om de panel data te analyseren. De uiteindelijke resultaten van deze analyse zagen er zo uit (model 3):

Table 6. *Random effects models with dependent variable separatist protest*

Variable	*Variable code*	*Model 1*	*Model 2*	*Model 3*
Nearby separatist protest	protnearby	0.711***		0.669***
Kindred group dispersion	kindred	0.223		0.224
Natural logarithm of refugees	logrefugees	0.065*		0.071*
Economic discrimination	ecdis		0.127#	0.110
Political discrimination	poldis		0.246***	0.210***
Polity IV democracy score	democracy	0.032***	0.033***	0.033***
Natural logarithm of GDP per capita	logGDP	-0.020		-0.014
Interaction: kindred group dispersion with natural logarithm of refugees	intkinref	-0.027		-0.026
Interaction: economic discrimination with political discrimination	intdis		-0.059*	-0.052*
N		1836	1836	1836

# p < 0,1; * p < 0.05; ** p < 0.01; *** p < 0.001

De dataset die ik gebruikt heb voor mijn analyse heb ik zelf samengesteld uit een aantal verschillende bronnen, namelijk:

- **Minorities at Risk (MAR):** Bevat variabelen omtrent separatisme voor bevolkingsgroepen binnen landen binnen regio's, per jaar geobserveerd. De meeste benodigde data staat hierin, deze dataset vormt dus de basis voor de uiteindelijke dataset.
- **UNHCR Statistical Database:** Bevat absolute aantallen vluchtelingen aanwezig in specifieke landen, per jaar.
- **Polity IV:** Bevat scores voor het niveau van democratie per land, per jaar.
- **World Bank:** Houdt jaarlijks de GDP per capita van vrijwel alle landen bij.

Voordat al deze data geanalyseerd kan worden moet er nog behoorlijk wat gebeuren ("data wrangling"). Het grootste deel van dit project bestaat dan ook uit het met behulp van Python en verschillende libraries de data omvormen tot een bruikbaar geheel. Voor dit project heb ik gebruik gemaakt van Jupyter Notebook.

Data wrangling

Minorities at Risk

Bij de MAR-data horen een aantal uitdagingen. Allereerst is het isoleren van de data die we nodig hebben belangrijk. Alle jaartallen van voor 1990 en alle variabelen die we niet gebruiken moeten uit de MAR-data geknipt worden. Daarnaast heb ik tijdens het schrijven van mijn scriptie besloten geen ontbrekende waarden in mijn dataset te willen. Uiteindelijk moeten dus voor alle bevolkingsgroepen alle variabelen zijn ingevuld voor ieder jaar. Missing values moeten dus weggehaald worden.

Een ander obstakel is het feit dat het MAR project vanaf 2004 een nieuwe fase is ingegaan. Dit betekent dat we voor de jaren 1990 – 2006 gebruik maken van twee datasets die weliswaar veel op elkaar lijken maar toch verschillen bevatten. Deze verschillen zullen worden gladgestreken waarna de datasets samengevoegd kunnen worden.

Om te beginnen zal de dataset met de data tot en met 2003 ingelezen worden. Dit gebeurt als volgt:

Input:
```
import numpy as np
import pandas as pd

mar2003_raw = pd.read_csv('mar2003.csv', sep=';', low_memory=False)
mar2003_raw.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7820 entries, 0 to 7819
Columns: 455 entries, adgained to V455
dtypes: int64(296), object(159)
memory usage: 27.1+ MB
```

We hebben nu dus met succes DataFrame ‘mar2003’ aangemaakt; deze bevat 7820 rijen en 455 kolommen.

Van al deze kolommen hebben we slechts een beperkt aantal nodig. Met behulp van indexing kunnen deze worden geisoleerd.

Input:
```
vars2003 = ['numcode', 'ccode', 'group', 'country', 'region', 'year',
            'prot', 'ecdis', 'poldis']

mar2003 = mar2003_raw.reindex(columns=vars2003)
mar2003.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7820 entries, 0 to 7819
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   numcode  7820 non-null   object
 1   ccode    7820 non-null   object
 2   group    7820 non-null   object
 3   country  7820 non-null   object
 4   region   7820 non-null   int64
 5   year     7820 non-null   int64
 6   prot     7820 non-null   int64
 7   ecdis    7820 non-null   int64
 8   poldis   7820 non-null   int64
dtypes: int64(5), object(4)
memory usage: 550.0+ KB
```

Nu is het belangrijk om alle kolommen op te schonen. Tijdens het schrijven van mijn scriptie heb ik besloten om enkel te werken met volledige informatie, aangezien de dataset groot genoeg was om dit te realiseren. Data van voor 1990, data met ontbrekende waarden en data die slordig of foutief gecodeerd is zal uit de DataFrame gehaald worden.

De volgende waarden voor jaartallen zijn in de DataFrame aanwezig:

Input:
```
unique_years = mar2003.year.unique()
unique_years.sort()
unique_years
```

Output:
```
array([ -99,    0,    1,    2,    3,    4,    5,    6, 1920, 1923, 1940,
       1945, 1950, 1952, 1955, 1960, 1965, 1970, 1975, 1980, 1985, 1986,
       1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997,
       1998, 1999, 2000, 2001, 2002, 2003], dtype=int64)
```

De waarden -99 tot en met 6 worden gebruikt om allerlei ontbrekende en afwijkende data aan te geven. Deze waarden evenals alle jaartallen onder 1990 moeten worden verwijderd:

Input:
```
mar2003 = mar2003[mar2003['year'] > 1989]

unique_years = mar2003.year.unique()
unique_years.sort()
unique_years
```

Output:
```
array([1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003], dtype=int64)
```

Hierna is het tijd om alle observaties met ontbrekende of afwijkende data te verwijderen (zie inleiding voor acceptabele waarden).

Input:
```
mar2003.numcode.unique()
mar2003.ccode.unique()
mar2003.group.unique()
mar2003.country.unique()
mar2003.region.unique()
```

Output:
Acceptabele waarden

Input:
```
mar2003.prot.unique())
```

Output:
```
array([  3,   2,   1,   0,   4,   5, -99,   6], dtype=int64)
```

Input:
```
mar2003 = mar2003[mar2003.prot != -99]
mar2003 = mar2003[mar2003.prot != 6]
mar2003.prot.unique()
```

Output:
```
array([3, 2, 1, 0, 4, 5], dtype=int64)
```

Input:
```
mar2003.ecdis.unique())
```

Output:
```
array([  1,   0,   4,   3,   2, -99], dtype=int64)
```

Input:
```
mar2003 = mar2003[mar2003.ecdis != -99]
mar2003.ecdis.unique()
```

Output:
```
array([1, 0, 4, 3, 2], dtype=int64)
```

Input:
```
mar2003.poldis.unique())
```

Output:
```
array([  1,   0,   2,   4,   3, -99], dtype=int64)
```

Input:
```
mar2003 = mar2003[mar2003.poldis != -99]
mar2003.poldis.unique()
```

Output:
```
array([1, 0, 2, 4, 3], dtype=int64)
```

Volledige informatie houdt in dat iedere bevolkingsgroep data zou moeten hebben voor ieder jaar van 1990 tot 2006. Door ontbrekende jaarlijke observaties en door het verwijderen van observaties met ontbrekende data is dit niet het geval. De volgende informatie toont dit aan:

Input:
```
mar2003.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    160
1991    179
1992    195
1993    200
1994    201
1995    199
1996    199
1997    201
1998    200
1999    202
2000    202
2001    277
2002    276
2003    276
Name: year, dtype: int64
```

Om enkel volledige tijdsreeksen mee te nemen moeten alle bevolkingsgroepen met ontbrekende jaren uit de data gefilterd worden. Hiervoor maak ik eerst een lijst met alle waarden van numcode (= unieke code voor bevolkingsgroep) die het volledige aantal observaties hebben. Daarna worden alle numcodes die niet in deze lijst staan uit de DataFrame gefilterd:

Input:
```
full_years = []
for numcode in mar2003.numcode.value_counts().index:
    if mar2003.numcode.value_counts()[numcode] == 14:
        full_years.append(numcode)

mar2003 = mar2003[mar2003['numcode'].isin(full_years)]
mar2003.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    146
1991    146
1992    146
1993    146
1994    146
1995    146
1996    146
1997    146
1998    146
1999    146
2000    146
2001    146
2002    146
2003    146
Name: year, dtype: int64
```

We hebben nu 146 observaties voor elk jaar, en in totaal moeten we dus ook 146 unieke bevolkingsgroepen hebben:

Input:
```
len(mar2003.numcode.unique())
```

Output:
```
146
```

Nu is het tijd om de aandacht te verleggen naar de data voor de jaren 2004-2006:

Input:
```
mar2006_raw = pd.read_csv('mar2006.csv', sep=';', low_memory=False)
vars2006 = ['numcode', 'ccode', 'VMAR_Group', 'country', 'VMAR_Region','year',
            'PROT', 'GC10', 'ECDIS', 'POLDIS']

mar2006 = mar2006_raw.reindex(columns=vars2006))
mar2006 = mar2006.rename(columns={'VMAR_Group': 'group', 'VMAR_Region': 'region',
                                  'PROT': 'prot', 'GC10': 'kindred',
                                  'ECDIS': 'ecdis', 'POLDIS': 'poldis'})
mar2006.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852 entries, 0 to 851
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   numcode  852 non-null    int64
 1   ccode    852 non-null    int64
 2   group    852 non-null    object
 3   country  852 non-null    object
 4   region   852 non-null    int64
 5   year     852 non-null    int64
 6   prot     852 non-null    int64
 7   kindred  852 non-null    int64
 8   ecdis    852 non-null    int64
 9   poldis   852 non-null    int64
dtypes: int64(8), object(2)
memory usage: 66.7+ KB
```

Bij deze DataFrame wordt dezelfde reeks handelingen uitgevoerd om te garanderen dat er alleen volledige informatie gebruikt wordt:

Input:
```
mar2006.numcode.unique()
mar2006.ccode.unique()
mar2006.group.unique()
mar2006.country.unique()
mar2006.region.unique()
mar2006.kindred.unique()
mar2006.poldis.unique()
```

Output:
Acceptabele waarden

Input:
```
mar2006.prot.unique()
```

Output:
```
array([  3,   4,   5,   1,   0,   2, -99], dtype=int64)
```

Input:
```
mar2006 = mar2006[mar2006.prot != -99]
mar2006.prot.unique()
```

Output:
```
array([3, 4, 5, 1, 0, 2], dtype=int64)
```

Input:
```
mar2006.ecdis.unique()
```

Output:
```
array([  1,   0,   4,   3,   2, -99], dtype=int64)
```

Input:
```
mar2006 = mar2006[mar2006.ecdis != -99]
mar2006.ecdis.unique()
```

Output:
```
array([1, 0, 4, 3, 2], dtype=int64)
```

Ook hier weer moeten volledige tijdsreeksen gegarandeerd worden:

Input:
```
mar2006.year.value_counts().sort_index(ascending=True)
```

Output:
```
2004    280
2005    281
2006    281
Name: year, dtype: int64
```

Input:
```
full_years = []
for numcode in mar2006.numcode.value_counts().index:
    if mar2006.numcode.value_counts()[numcode] == 3:
        full_years.append(numcode)

mar2006 = mar2006[mar2006['numcode'].isin(full_years)]
mar2006.year.value_counts().sort_index(ascending=True)
```

Output:
```
2004    277
2005    277
2006    277
Name: year, dtype: int64
```

Input:
```
len(mar2006.numcode.unique())
```

Output:
```
277
```

Met behulp van een index kunnen DataFrames samengevoegd worden. De variabele numcode heeft unieke waarden voor iedere bevolkingsgroep en tijdsreeks, en is daarmee de ideale kandidaat.

Input:
```
mar2003 = mar2003.set_index(['numcode'], drop=True)
mar2003.index
```

Output:
```
Index(['203', '203', '203', '203', '203', '203', '203', '203', '203', '203',
       ...
       '95002', '95002', '95002', '95002', '95002', '95002', '95002', '95002',
       '95002', '95002'],
      dtype='object', name='numcode', length=2044
```

Input:
```
mar2006 = mar2006.set_index(['numcode'], drop=True)
mar2006.index
```

Output:
```
Int64Index([  201,   201,   201,   202,   202,   202,   203,   203,   203,
             204,
            ...
            91001, 92001, 92001, 92001, 95001, 95001, 95001, 95002, 95002,
            95002],
           dtype='int64', name='numcode', length=831)
```

De laatste stap voor het samenvoegen van de DataFrames is ervoor zorgen dat de kolommen in beide DataFrames overeenkomende datatypes hebben. Bij de kolommen ccode en de index (numcode) is dit niet het geval.

Input:
```
mar2003.ccode = mar2003.ccode.astype('int64')
mar2003.index = mar2003.index.astype('int64')
```

Het samenvoegen van de DataFrames gaat als volgt:

Input:
```
mar_complete = pd.concat([mar2003,mar2006], axis=0).sort_values(by=['numcode',
                                                                    'year'])
mar_complete.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2875 entries, 201 to 95002
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   ccode    2875 non-null   int64
 1   group    2875 non-null   object
 2   country  2875 non-null   object
 3   region   2875 non-null   int64
 4   year     2875 non-null   int64
 5   prot     2875 non-null   int64
 6   ecdis    2875 non-null   int64
 7   poldis   2875 non-null   int64
 8   kindred  831 non-null    float64
dtypes: float64(1), int64(6), object(2)
memory usage: 224.6+ KB
```

Een probleem dat we eerder weggewerkt hebben komt nu weer terug. De twee DataFrames bevatten respectievelijk 146 en 277 unieke numcodes.

Input:
```
mar_complete.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    146
1991    146
1992    146
1993    146
1994    146
1995    146
1996    146
1997    146
1998    146
1999    146
2000    146
2001    146
2002    146
2003    146
2004    277
2005    277
2006    277
Name: year, dtype: int64
```

Dit betekent dat de nieuwe DataFrame niet enkel bestaat uit complete tijdsreeksen van 1990-2006. Om dit probleem te overkomen passen we de truc van eerder opnieuw toe. Omdat in het betreffende stuk code het aantal jaarlijkse observaties per numcode wordt bijgehouden, moet numcode tijdelijk weer als reguliere kolom in de DataFrame gevoegd worden.

Input:
```
mar_complete['numcode'] = mar_complete.index
full_years = []
for numcode in mar_complete.numcode.value_counts().index:
    if mar_complete.numcode.value_counts()[numcode] == 17:
        full_years.append(numcode)

mar_complete = mar_complete[mar_complete['numcode'].isin(full_years)]
mar_complete = mar_complete.drop('numcode', axis=1)
mar_complete.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    143
1991    143
1992    143
1993    143
1994    143
1995    143
1996    143
1997    143
1998    143
1999    143
2000    143
2001    143
2002    143
2003    143
2004    143
2005    143
2006    143
Name: year, dtype: int64
```

Het is je misschien opgevallen dat de variabele kindred alleen uit het tweede csv-bestand is meegenomen. Hierdoor zijn er alleen waarden aanwezig voor de jaren 2004-2006.

Input:
```
mar_complete.count()
```

Output:
```
ccode      2431
group      2431
country    2431
region     2431
year       2431
prot       2431
ecdis      2431
poldis     2431
kindred     429
dtype: int64
```

De reden hiervoor is dat deze variabele in de nieuwe fase van het MAR-project op een andere manier tot stand is gekomen; de oude manier had fouten en wordt niet langer als accuraat beschouwd. Aangezien de variabele kindred wijst op de aanwezigheid van verwante bevolkingsgroepen in naburige landen kan worden aangenomen dat er zeer weinig verandering plaats vindt in de periode 1990-2006. Om deze reden is besloten de waarden van 2004-2006 met terugwerkende kracht te gebruiken voor de periode 1990-2003. Daarbij wordt het datatype van kindred gecorrigeerd naar integer.

Input:
```
mar_complete.kindred = mar_complete.kindred.fillna(method='bfill')
mar_complete.kindred = mar_complete.kindred.astype('int64')
mar_complete.count()
```

Output:
```
ccode      2431
group      2431
country    2431
region     2431
year       2431
prot       2431
ecdis      2431
poldis     2431
kindred    2431
dtype: int64
```

De volgende stap in het proces is het aanmaken van de variabele protnearby. Dit is een variabele op landniveau die het gemiddelde van variable prot aangeeft van alle landen in de regio behalve het land in kwestie. De eerste stap is het optellen van alle waarden van prot per regio per jaar.

Input:
```
prot_region = mar_complete.groupby(['region', 'year']).sum().prot
prot_region = prot_region.rename("prot_region")
prot_region
```

Output:
```
region  year
0       1990    46
        1991    48
        1992    48
        1993    50
        1994    49
                ..
7       2002    51
        2003    43
        2004    35
        2005    36
        2006    40
Name: prot_region, Length: 102, dtype: int64
```

Hierna kan deze Series toegevoegd worden aan de DataFrame. In de code zie je wederom dat numcode tijdelijk terug wordt gebracht als kolom. Dit is om te voorkomen dat numcode als index verloren gaat bij de merge.

Input:
```
mar_complete = mar_complete.reset_index().merge(prot_region, how='left',
                                                on=['year',
                                                    'region']).set_index('numcode')
mar_complete.count()
```

Output:
```
ccode          2431
group          2431
country        2431
region         2431
year           2431
prot           2431
ecdis          2431
poldis         2431
kindred        2431
prot_region    2431
dtype: int64
```

De volgende stap is het berekenen van de gemiddelde waarde van prot per land per jaar.

Input:
```
prot_country = mar_complete.groupby(['ccode', 'year']).mean().prot
prot_country = prot_country.rename("prot_country")
prot_country
```

Output:
```
ccode  year
2      1990    3.0
       1991    3.0
       1992    2.0
       1993    3.0
       1994    3.0
              ...
950    2002    3.0
       2003    0.0
       2004    0.0
       2005    0.0
       2006    0.0
Name: prot_country, Length: 1309, dtype: float64
```

Ook deze Series wordt als kolom toegevoegd aan de DataFrame.

Input:
```
mar_complete = mar_complete.reset_index().merge(prot_country, how='left',
                                                on=['year',
                                                    'ccode']).set_index('numcode')
mar_complete.count()
```

Output:
```
ccode           2431
group           2431
country         2431
region          2431
year            2431
prot            2431
ecdis           2431
poldis          2431
kindred         2431
prot_region     2431
prot_country    2431
dtype: int64
```

Om per land, per jaar, de totale optelsom van prot te krijgen hoeven we enkel de waarden van prot_country af te trekken van de waarden van prot_region.

Input:
```
mar_complete['prot_region_minus'] = \
                          (mar_complete.prot_region - mar_complete.prot_country)
mar_complete.count()
```

Output:
```
ccode                2431
group                2431
country              2431
region               2431
year                 2431
prot                 2431
ecdis                2431
poldis               2431
kindred              2431
prot_region          2431
prot_country         2431
prot_region_minus    2431
dtype: int64
```

Het aantal numcodes per regio berekenen we met behulp van een pivot table:

Input:
```
prot_groupcount = mar_complete.reset_index()\
                 .pivot_table(index='numcode').reset_index()\
                 .region.value_counts()

prot_groupcount = prot_groupcount.rename("prot_groupcount")
prot_groupcount.index.name = 'region'
prot_groupcount.sort_index(ascending=True)
```

Output:
```
0      24
34      9
51     40
85     11
102    34
119    25
Name: prot_groupcount, dtype: int64
```

Toevoegen aan de DataFrame:

Input:
```
mar_complete = mar_complete.reset_index().merge(prot_groupcount, how='left',
                                                on='region').set_index('numcode')
mar_complete.count()
```

Output:
```
ccode                2431
group                2431
country              2431
region               2431
year                 2431
prot                 2431
ecdis                2431
poldis               2431
kindred              2431
prot_region          2431
prot_country         2431
prot_region_minus    2431
prot_groupcount      2431
dtype: int64
```

Tot slot moeten alle waarden van prot_region_minus gedeeld worden door de bijbehorende waarden van prot_groupcount. Hierna is de variabele protnearby eindelijk compleet.

Input:
```
mar_complete['protnearby'] = \
(mar_complete.prot_region_minus / mar_complete.prot_groupcount)

mar_complete.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2431 entries, 203 to 95002
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   ccode              2431 non-null   int64
 1   group              2431 non-null   object
 2   country            2431 non-null   object
 3   region             2431 non-null   int64
 4   year               2431 non-null   int64
 5   prot               2431 non-null   int64
 6   ecdis              2431 non-null   int64
 7   poldis             2431 non-null   int64
 8   kindred            2431 non-null   int64
 9   prot_region        2431 non-null   int64
 10  prot_country       2431 non-null   float64
 11  prot_region_minus  2431 non-null   float64
 12  prot_groupcount    2431 non-null   int64
 13  protnearby         2431 non-null   float64
dtypes: float64(4), int64(8), object(2)
memory usage: 284.9+ KB
```

Om de DataFrame overzichtelijk te houden mogen de kolommen die niet langer nodig zijn verwijderd worden.

Input:

```
mar_complete = mar_complete.drop(columns=['prot_region','prot_country',
                                          'prot_region_minus','prot_groupcount'])
mar_complete.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2431 entries, 203 to 95002
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   ccode       2431 non-null   int64  
 1   group       2431 non-null   object 
 2   country     2431 non-null   object 
 3   region      2431 non-null   int64  
 4   year        2431 non-null   int64  
 5   prot        2431 non-null   int64  
 6   ecdis       2431 non-null   int64  
 7   poldis      2431 non-null   int64  
 8   kindred     2431 non-null   int64  
 9   protnearby  2431 non-null   float64
dtypes: float64(2), int64(6), object(2)
memory usage: 208.9+ KB
```

Refugees

Hiermee zijn we klaar met de data afkomstig uit het MAR project. De volgende variabele, refugees, bestaat uit het aantal aanwezige vluchtelingen per land per jaar. Omdat refugees afkomstig is van een andere bron, is deze niet zonder meer te koppelen aan mar_complete. We moeten ervoor zorgen dat de waarden van country overeenkomen in refugees en mar_complete.

Eerst is het belangrijk om uberhaupt te zorgen dat alle waarden van country per ccode hetzelfde zijn. Aangezien deze waarden afkomstig zijn van twee fases van het MAR project zijn er verschillen. Zo staan de waarden van fase 1990-2003 allemaal in hoofdletters tussen aanhalingstekens (bijvoorbeeld: Canada - “CANADA”). Dit moet worden gladgestreken. Voor de goede orde doen we hetzelfde met de kolom group.

Input:
```
print(mar_complete.country.unique().size)
print(mar_complete.group.unique().size)
```

Output:
```
154
225
```

Input:
```
mar_complete.loc[mar_complete.country.str.contains('\"'), 'country'] = np.nan
mar_complete.country = mar_complete.country.fillna(method='bfill')

mar_complete.loc[mar_complete.group.str.contains('\"'), 'group'] = np.nan
mar_complete.group = mar_complete.group.fillna(method='bfill')

print(mar_complete.country.unique().size)
print(mar_complete.group.unique().size)
```

Output:
```
77
112
```

Nu kunnen we het csv-bestand van refugees inlezen en de relevante kolommen selecteren:

Input:
```
unhcr_raw = pd.read_csv('UNHCR refugees.csv', sep=',', low_memory=False)

unchr = unhcr_raw.reindex(columns=\
['Country or territory of asylum or residence',
 'Year',
 'Total refugees and people in refugee-like situations<sup>**</sup>'])

unchr = unchr.rename(columns=\
{'Country or territory of asylum or residence': 'country',
 'Year': 'year',
 'Total refugees and people in refugee-like situations<sup>**</sup>': 'refugees'})

unchr.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96064 entries, 0 to 96063
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   country   96064 non-null  object
 1   year      96061 non-null  float64
 2   refugees  96020 non-null  float64
dtypes: float64(2), object(1)
memory usage: 2.2+ MB
```

De kolom year loopt van 1975 tot en met 2016. Enkel de jaren 1990-2006 moeten meegenomen worden.

Input:

```
unchr = unchr[unchr['year'] > 1989]
unchr = unchr[unchr['year'] < 2007]
unchr.year.unique()
```

Output:

```
array([2006., 2005., 2004., 2003., 2002., 2001., 2000., 1999., 1998.,
       1997., 1996., 1995., 1994., 1993., 1992., 1991., 1990.])
```

Daarna worden alle rijen met ontbrekende waarden op refugees gefilterd.

Input:

```
unchr = unchr[unchr.refugees.notnull()]
unchr.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43636 entries, 48802 to 92475
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   country   43636 non-null  object
 1   year      43636 non-null  float64
 2   refugees  43636 non-null  float64
dtypes: float64(2), object(1)
memory usage: 1.3+ MB
```

In de oorspronkelijke dataset zijn de aantallen vluchtelingen per land per jaar ook nog eens verdeeld per land van herkomst. Deze moeten dus opgeteld worden. Een pivot table biedt uitkomst.

Input:

```
unchr = unchr.pivot_table(index=['country','year'],
                          aggfunc=np.sum).reset_index()
unchr.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2418 entries, 0 to 2417
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   country   2418 non-null   object
 1   year      2418 non-null   float64
 2   refugees  2418 non-null   float64
dtypes: float64(2), object(1)
memory usage: 56.8+ K
```

Voordat we verdergaan is het een goed idee om de datatypes van year en refugees te corrigeren.

Input:
```
unchr.year = unchr.year.astype('int64')
unchr.refugees = unchr.refugees.astype('int64')
unchr.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2418 entries, 0 to 2417
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   country   2418 non-null   object
 1   year      2418 non-null   int64
 2   refugees  2418 non-null   int64
dtypes: int64(2), object(1)
memory usage: 56.8+ KB
```

Net als bij de MAR-data moet hier ook weer gegarandeerd worden dat de data compleet is. Dat betekent: een jaarlijkse observatie per land. Dit is nu niet het geval.

Input:
```
unchr.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    112
1991    118
1992    132
1993    139
1994    142
1995    143
1996    143
1997    147
1998    146
1999    151
2000    150
2001    151
2002    147
2003    147
2004    148
2005    149
2006    153
Name: year, dtype: int64
```

Hiervoor kan weer dezelfde methode gebruikt worden die ook voor de MAR data gebruikt werd.

Input:
```
full_years = []
for country in unchr.country.value_counts().index:
    if unchr.country.value_counts()[country] == 17:
        full_years.append(country)

unchr = unchr[unchr['country'].isin(full_years)]
unchr.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    100
1991    100
1992    100
1993    100
1994    100
1995    100
1996    100
1997    100
1998    100
1999    100
2000    100
2001    100
2002    100
2003    100
2004    100
2005    100
2006    100
Name: year, dtype: int64
```

Input:
```
len(unchr.country.unique())
```

Output:
```
100
```

Om unchr en mar_complete te kunnen verbinden moeten de landen die niet in mar_complete aanwezig zijn verwijderd worden uit refugees. Dit kan als volgt gecontroleerd worden:

Input:
```
qualified_countries = []
disqualified_countries = []

for country in unchr.country.unique():
    if country in mar_complete.country.unique():
        qualified_countries.append(country)
    else:
        disqualified_countries.append(country)

print(len(qualified_countries))
print(len(disqualified_countries))
```

Output:
```
50
50
```

Er zijn dus in vijftig landen aanwezig in refugees die ook direct herkend worden in mar_complete. Een probleem is dat landsnamen vaak op verschillende manier gespeld of weergegeven kunnen worden. De enige oplossing is hier dan ook om te kijken naar de lijst disqualified_countries om handmatig landen eruit te pikken die toch aanwezig zijn in mar_complete.

Input:
```
remaining_countries = []
for country in mar_complete.country.unique():
    if country not in qualified_countries:
        remaining_countries.append(country)

np.sort(remaining_countries)
```

Output:
```
array(['Afghanistan', 'Albania', 'Bahrain', 'Bhutan', 'Bolivia',
       'Bulgaria', 'Burma', 'Dominican Republic', 'Fiji', 'Guyana',
       'Iran', 'Israel', 'Korea, South', 'Laos', 'Madagascar', 'Namibia',
       'Romania', 'Singapore', 'Somalia', 'South Africa', 'Sri Lanka',
       'Taiwan', 'Tanzania', 'United States of America', 'Venezuela',
       'Vietnam', 'Yugoslavia'], dtype='<U24')
```

De lijst remaining_countries geeft alle landen van mar_complete aan die niet in qualified_countries zijn beland en dus zijn overgebleven. Van deze array zijn Bolivia, Iran, Tanzania, United States of America, Venezuela en Vietnam in een andere vorm terug te vinden in disqualified_countries. Deze moeten dus alsnog meegenomen worden.

Input:
```
def rename_country(old, new):
    unchr.loc[unchr.country.str.contains(old, regex=False), 'country'] = new

rename_country('Bolivia (Plurinational State of)', 'Bolivia')
rename_country('Islamic Rep. of Iran', 'Iran')
rename_country('United Rep. of Tanzania', 'Tanzania')
rename_country('United States', 'United States of America')
rename_country('Venezuela (Bolivarian Republic of)', 'Venezuela')
rename_country('Viet Nam', 'Vietnam')

qualified_countries = []
disqualified_countries = []

for country in unchr.country.unique():
    if country in mar_complete.country.unique():
        qualified_countries.append(country)
    else:
        disqualified_countries.append(country)

print(len(qualified_countries))
print(len(disqualified_countries))
```

Output:
```
56
44
```

Nu kunnen alle landen in disqualified_countries uit refugees gefilterd worden.

Input:
```
unchr = unchr[unchr['country'].isin(qualified_countries)]
len(unchr.country.unique())
```

Output:
```
56
```

Om overzicht en leesbaarheid te bewaren wordt voor variabelen met erg uiteenlopende waarden wel eens het natuurlijk logaritme van die variabele gebruikt. Voor refugee_count is dit een goede optie.

Input:
```
print(unchr.refugees.min())
print(unchr.refugees.max())
```

Output:
```
9
4404995
```

Input:
```
unchr['logrefugees'] = np.log(unchr['refugees'])
print(unchr.logrefugees.min())
print(unchr.logrefugees.max())
```

Output:
```
2.1972245773362196
15.298249682277993
```

Nu de landen in refugees overeenkomen met die in mar_complete kan de kolom country in beide DataFrames gebruikt worden om de DataFrames samen te voegen.

Input:
```
final_data = mar_complete.reset_index().merge(unchr, how='right',
                                              on=['country', 'year'])
final_data.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1887 entries, 0 to 1886
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   numcode      1887 non-null   int64  
 1   ccode        1887 non-null   int64  
 2   group        1887 non-null   object 
 3   country      1887 non-null   object 
 4   region       1887 non-null   int64  
 5   year         1887 non-null   int64  
 6   prot         1887 non-null   int64  
 7   ecdis        1887 non-null   int64  
 8   poldis       1887 non-null   int64  
 9   kindred      1887 non-null   int64  
 10  protnearby   1887 non-null   float64
 11  refugees     1887 non-null   int64  
 12  logrefugees  1887 non-null   float64
dtypes: float64(2), int64(9), object(2)
memory usage: 206.4+ KB
```

Democracy

Om te beginnen wordt het csv-bestand ingelezen en worden hiervan de juiste kolommen geselecteerd.

Input:
```
polity_raw = pd.read_csv('Polity IV.csv', sep=';', low_memory=False)
polity = polity_raw.reindex(columns=['ccode', 'year', 'polity2'])
polity = polity.rename(columns={'polity2': 'democracy'})
polity.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17228 entries, 0 to 17227
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ccode      17228 non-null  int64 
 1   year       17228 non-null  int64 
 2   democracy  17228 non-null  object
dtypes: int64(2), object(1)
memory usage: 403.9+ KB
```

De volgende stap is het selecteren van de juiste jaren.

Input:
```
polity = polity[polity['year'] > 1989]
polity = polity[polity['year'] < 2007]
polity.year.unique()
```

Output:
```
array([1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006], dtype=int64)
```

Ontbrekende waarden op democracy bestaan uit een enkele spatie. Deze moeten verwijderd worden, waarna de variabele ook omgezet kan worden naar integer.

Input:
```
polity = polity[polity.democracy != ' ']
polity.democracy = polity.democracy.astype('int64')
polity.democracy.unique()
```

Output:
```
array([ -8,   0,  -7,   1,   5,   7,   9,  -2,  -3,   2,  -1,   8,   3,
        -6,  10, -10,  -9,   6,  -5,  -4,   4], dtype=int64)
```

Om volledige informatie te garanderen moeten ook hier alle observaties met incomplete data voor de jaren 1990-2006 verwijderd worden.

Input:
```
polity.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    144
1991    160
1992    160
1993    163
1994    162
1995    161
1996    161
1997    161
1998    161
1999    161
2000    161
2001    160
2002    161
2003    159
2004    160
2005    161
2006    163
Name: year, dtype: int64
```

Input:
```
full_years = []
for ccode in polity.ccode.value_counts().index:
    if polity.ccode.value_counts()[ccode] == 17:
        full_years.append(ccode)

polity = polity[polity['ccode'].isin(full_years)]
polity.year.value_counts().sort_index(ascending=True)
```

Output:
```
1990    134
1991    134
1992    134
1993    134
1994    134
1995    134
1996    134
1997    134
1998    134
1999    134
2000    134
2001    134
2002    134
2003    134
2004    134
2005    134
2006    134
Name: year, dtype: int64
```

Nu kan polity toegevoegd worden aan final_data.

Input:
```
final_data = final_data.merge(polity, how='inner', on=['ccode', 'year'])
final_data.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1785 entries, 0 to 1784
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   numcode      1785 non-null   int64  
 1   ccode        1785 non-null   int64  
 2   group        1785 non-null   object 
 3   country      1785 non-null   object 
 4   region       1785 non-null   int64  
 5   year         1785 non-null   int64  
 6   prot         1785 non-null   int64  
 7   ecdis        1785 non-null   int64  
 8   poldis       1785 non-null   int64  
 9   kindred      1785 non-null   int64  
 10  protnearby   1785 non-null   float64
 11  refugees     1785 non-null   int64  
 12  logrefugees  1785 non-null   float64
 13  democracy    1785 non-null   int64  
dtypes: float64(2), int64(10), object(2)
memory usage: 209.2+ KB
```

GDP per capita

Hier lopen we tegen een nieuwe uitdaging aan. Het csv-bestand voor GDP per capita is in een zogenaamd wide format. Dit is in tegenstelling tot de data waar we tot nu toe mee gewerkt hebben, die in long format is. In dit geval betekent het dat er slechts één rij per land is, met bijbehorende kolommen per jaar van 1960 tot en met 2017. Gelukkig kan een DataFrame als deze omgezet worden naar long format met pandas.melt.

Input:
```
worldbank_raw = pd.read_csv('GDP per capita.csv', sep=';',low_memory=False)
worldbank = worldbank_raw.drop(columns=['IndicatorName', 'IndicatorCode',
'CountryCode'])
worldbank = worldbank.melt(id_vars='CountryName')
worldbank = worldbank.reindex(columns=['CountryName', 'variable', 'value'])
worldbank = worldbank.rename(columns={'CountryName': 'country',
                                      'variable': 'year',
                                      'value': 'GDP'})
worldbank.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15312 entries, 0 to 15311
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   country  15312 non-null  object
 1   year     15312 non-null  object
 2   GDP      15312 non-null  object
dtypes: object(3)
memory usage: 359.0+ KB
```

Nu moeten alle jaartallen in de periode 1990-2006 geselecteerd worden. In de kolom year staat een @ voor ieder jaartal. Dit moet worden gecorrigeerd om met de kolom te kunnen werken.

Input:
```
worldbank.year = worldbank.year.str.lstrip('@').astype('int64')
worldbank = worldbank[worldbank['year'] > 1989]
worldbank = worldbank[worldbank['year'] < 2007]
worldbank.year.unique()
```

Output:
```
array([1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006], dtype=int64)
```

Ontbrekende waarden in de kolom GDP zijn aangegeven met een enkele spatie. Deze moeten verwijderd worden.

Input:
```
worldbank = worldbank[worldbank.GDP != ' ']
np.isin(" ", worldbank.GDP.unique())
```

Output:
```
array(False)
```

Hierna kunnen we wederom de landen met incomplete observaties voor 1990-2006 verwijderen.

Input:
```
worldbank.country.value_counts().sort_index(ascending=True)
```

Output:
```
Afghanistan            6
Albania               17
Algeria               17
American Samoa         5
Andorra               17
                     ..
West Bank and Gaza    13
World                 17
Yemen, Rep.           17
Zambia                17
Zimbabwe              17
Name: country, Length: 253, dtype: int64
```

Input:
```
full_years = []
for country in worldbank.country.value_counts().index:
    if worldbank.country.value_counts()[country] == 17:
        full_years.append(country)

worldbank = worldbank[worldbank['country'].isin(full_years)]
worldbank.country.value_counts().sort_index(ascending=True)
```

Output:
```
Albania               17
Algeria               17
Andorra               17
Angola                17
Antigua and Barbuda   17
                     ..
Vietnam               17
World                 17
Yemen, Rep.           17
Zambia                17
Zimbabwe              17
Name: country, Length: 216, dtype: int64
```

De kolom GDP moet omgezet worden naar datatype float. Hiervoor moeten echter de komma’s in de waarden van GDP omgezet worden naar punten, zodat deze waarden ook als getal geïnterpreteerd kunnen worden.

Input:
```
worldbank.GDP = worldbank.GDP.str.replace(',','.')
worldbank.GDP = worldbank.GDP.astype('float64')
worldbank.GDP.describe())
```

Output:
```
count      3672.000000
mean       7813.077023
std       13626.270943
min          65.011416
25%         609.116060
50%        2032.312954
75%        7672.450920
max      135535.002415
Name: GDP, dtype: float64
```

Hiermee zijn we aangekomen bij de laatste stap: het toevoegen van worldbank aan final_data. Net als bij refugees moeten hier ook landsnamen aan elkaar gekoppeld worden.

Input:
```
qualified_countries = []
disqualified_countries = []

for country in worldbank.country.unique():
    if country in final_data.country.unique():
        qualified_countries.append(country)
    else:
        disqualified_countries.append(country)

print(len(qualified_countries))
print(len(disqualified_countries))
```

Output:
```
47
169
```

Er zijn 47 landen aanwezig in worldbank die direct herkend worden in final_data. Ook hier moet weer handmatig gekeken worden of er in de lijst met 169 afgekeurde landen toch nog matchende landen met een afwijkende naam staan.

Input:
```
remaining_countries = []
for country in final_data.country.unique():
    if country not in qualified_countries:
        remaining_countries.append(country)

np.sort(remaining_countries))
```

Output:
```
array(['Dem. Rep. of the Congo', 'Egypt', 'Hungary', 'Iran',
       'United States of America', 'Venezuela'], dtype='<U24')
```

We moeten in disqualified_countries op zoek naar de landen die in remaining_countries staan. De volgende landen zijn gevonden en worden hernoemd:

Input:
```
def rename_country2(old, new):
    worldbank.loc[worldbank.country.str.contains(old, regex=False),'country'] = new

rename_country2('Congo, Dem. Rep.', 'Dem. Rep. of the Congo')
rename_country2('Egypt, Arab Rep.', 'Egypt')
rename_country2('United States', 'United States of America')
rename_country2('Venezuela, RB', 'Venezuela')

qualified_countries = []
disqualified_countries = []

for country in worldbank.country.unique():
    if country in final_data.country.unique():
        qualified_countries.append(country)
    else:
        disqualified_countries.append(country)

print(len(qualified_countries))
print(len(disqualified_countries))
```

Output:
```
51
165
```

Alle afgekeurde landen kunnen nu uit worldbank verwijderd worden.

Input:
```
worldbank = worldbank[worldbank['country'].isin(qualified_countries)]
len(worldbank.country.unique())
```

Output:
```
51
```

Tot slot kunnen we worldbank toevoegen aan final_data.

Input:
```
final_data = final_data.merge(worldbank, how='inner', on=['country', 'year'])
final_data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1734 entries, 0 to 1733
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   numcode      1734 non-null   int64  
 1   ccode        1734 non-null   int64  
 2   group        1734 non-null   object 
 3   country      1734 non-null   object 
 4   region       1734 non-null   int64  
 5   year         1734 non-null   int64  
 6   prot         1734 non-null   int64  
 7   ecdis        1734 non-null   int64  
 8   poldis       1734 non-null   int64  
 9   kindred      1734 non-null   int64  
 10  protnearby   1734 non-null   float64
 11  refugees     1734 non-null   int64  
 12  logrefugees  1734 non-null   float64
 13  democracy    1734 non-null   int64  
 14  GDP          1734 non-null   float64
dtypes: float64(3), int64(10), object(2)
memory usage: 216.8+ KB
```

Alles data die nodig is om de analyse uit te voeren is nu aanwezig. In het volgende hoofdstuk worden eerst een aantal aannames gecontroleerd, waarna de analyse wordt uitgevoerd.

Analyse

Bij een analyse met tijdreeksen moet er altijd geled worden op autocorrelation. Er is sprake van autocorrelation wanneer de waarde van een variabele op een bepaald tijdstip (deels) kan worden verklaard door de waarde van diezelfde variabele op het tijdstip ervoor. In ons geval: het is denkbaar dat een hoog niveau van prot in het ene jaar volgt op een hoog niveau van prot in het voorgaande jaar. Om dit probleem op te lossen wordt een lagged dependent variable geïntroduceerd. Dit is niets meer dan de variabele prot, waarvan alle waarden één jaar vooruit zijn geschoven. Door deze lagged variable mee te nemen wordt er op ieder observatiemoment in de analyse rekening gehouden met het voorgaande moment. Het nadeel hiervan is dat het eerste moment (1990) verloren zal gaan.

Input:

```
final_data['protlag'] = final_data.prot.shift(1)
final_data = final_data[final_data['year'] != 1990]
final_data.protlag = final_data.protlag.astype('int64')
final_data = final_data.reindex(columns=['numcode', 'group', 'ccode',
                                         'country', 'year', 'prot',
                                         'protlag','protnearby', 'kindred',
                                         'refugees','logrefugees', 'ecdis',
                                         'poldis', 'democracy', 'GDP', 'region'])
final_data.to_csv('final_data.csv', sep=';')
final_data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1632 entries, 1 to 1733
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   numcode      1632 non-null   int64  
 1   group        1632 non-null   object 
 2   ccode        1632 non-null   int64  
 3   country      1632 non-null   object 
 4   year         1632 non-null   int64  
 5   prot         1632 non-null   int64  
 6   protlag      1632 non-null   int64  
 7   protnearby   1632 non-null   float64
 8   kindred      1632 non-null   int64  
 9   refugees     1632 non-null   int64  
 10  logrefugees  1632 non-null   float64
 11  ecdis        1632 non-null   int64  
 12  poldis       1632 non-null   int64  
 13  democracy    1632 non-null   int64  
 14  GDP          1632 non-null   float64
 15  region       1632 non-null   int64  
dtypes: float64(3), int64(11), object(2)
memory usage: 216.8+ KB
```

De DataFrame final_data is nu helemaal compleet en daarom opgeslagen als csv-bestand. Het data wrangling-proces heeft in dit project een andere lengte opgeleverd dan bij mijn masterscriptie (1632 vs. 1728 rijen). Ik heb het proces zo nauw mogelijk gereproduceerd, maar door ontbrekende documentatie is er toch een ander resultaat uitgekomen. Dit kan in de analyse terug te zien zijn.

Intraclass correlation (ICC) is een soort correlatie dat optreedt bij waarden binnen groepen van een hiërarchische dataset. Het wordt als volgt berekend:

Input:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf

results_icc = smf.mixedlm('prot ~ numcode', data=final_data,
                          groups=final_data.numcode).fit()
results_icc.cov_re / (results_icc.cov_re + results_icc.scale)
```

Output:
```
0.456259
```

Deze waarde kan gezien worden als 45.6% van de totale variantie van de onafhankelijke variabele. Dit betekent dat 45.6% van de variantie van prot bepaald wordt door fixed effects: bevolkingsgroep (binnen land, regio). In een fixed effects model wordt gecontroleerd op fixed effects. Dit zou betekenen dat 45.6% van de totale variante als het ware wordt weggegooid en maar 54.4% verklaard kan worden. Een random effects/mixed linear model is daarom een betere optie.

Dit statistische model kan nu worden aangemaakt. De verwachting is dat hogere waarden op zowel refugees en kindred het effect van prot meer versterken dan die twee variabelen afzonderlijk opgeteld zouden doen. Eenzelfde verwachting bestaat bij de variabelen ecdis en poldis. In een mixed model kan hiermee rekening gehouden worden door een zogenaamd interactie-effect mee te nemen in de analyse. Deze interactie neemt de vorm aan van een variabele die bestaat uit de vermenigvuldiging van de relevante variabelen.

Input:
```
model = smf.mixedlm('prot ~ protlag + protnearby + kindred +\
                    logrefugees + kindred*logrefugees + ecdis + poldis +\
                    ecdis*poldis + democracy + GDP',
                    data=final_data, groups=final_data.numcode)
results = model.fit()
```

Hierna kan er getest worden op heteroscedasticiteit:

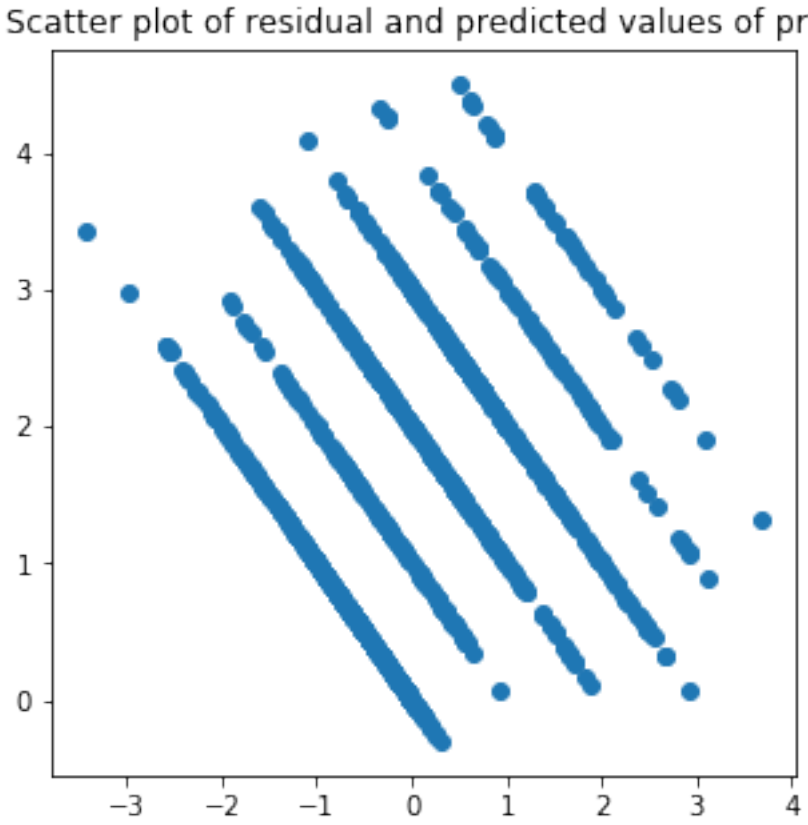
Input:
```
import matplotlib.pyplot as plt

pred_val = results.fittedvalues
true_val = final_data.prot.values.copy()
residual = true_val - pred_val

fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(residual, pred_val)
plt.title("Scatter plot of residual and predicted values of prot")
```

Output:

```
Text(0.5, 1.0, 'Scatter plot of residual and predicted values of prot')
```



Er is sprake van homoscedasticiteit wanneer de waarden in het scatterplot rond het midden geconcentreerd zijn en wanneer het scatterplot een grofweg ronde vorm aanneemt. Als de vorm van het scatterplot afwijkt en bijvoorbeeld een waaier vormt is er sprake van heteroscedasticiteit en moeten er maatregelen genomen worden. In dit geval lijkt er niets aan de hand te zijn en is er geen actie vereist.

De volgende test betreft multicollineariteit. Dit is een fenomeen waarbij twee of meer onafhankelijke variabelen sterk met elkaar correleren waardoor de waarde op één (deels) verklaard kan worden door de waarde op de ander. Dit kan het onduidelijk maken welke onafhankelijke variabele daadwerkelijk invloed heeft op de afhankelijke variabele. Een test die multicollineariteit kan aantonen heet Variance Intolerance Factor (VIF). Een vuistregel is dat VIF-waarden boven de 5 iets van multicollineariteit aangeven en dat VIF-waarden boven de 10 problematisch zijn. De VIF-waarden van alle relevante, originele variabelen zijn berekend.

Input:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vifvars = final_data.reindex(columns=['protnearby', 'kindred', 'refugees', 'ecdis',
'poldis'])

vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(vifvars.values, i) for i in
range(vifvars.shape[1])]
print(vif.set_index(vifvars.columns).round(2))
```

Output:

```
              VIF
protnearby   4.97
kindred      2.93
refugees     1.25
ecdis        5.00
poldis       4.39
```

Bij de variabelen protnearby, ecdis en poldis is waarschijnlijk dus iets van multicollineariteit aanwezig met de andere variabelen in de lijst. Deze waarden zijn echter niet zo hoog dat we ons zorgen hoeven te maken.

Nu alle aannames zijn getest kan eindelijk de resultaten van de analyse gepresenteerd worden. Hiervoor wordt het model gebruikt dat eerder aangemaakt is.

Input:

```
results.summary()
```

Output:

Model:	MixedLM	Dependent Variable:	prot
No. Observations:	1632	Method:	REML
No. Groups:	102	Scale:	1.0786
Min. group size:	16	Log-Likelihood:	-2537.1985
Max. group size:	16	Converged:	Yes
Mean group size:	16.0		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Intercept	-0.307	0.475	-0.647	0.518	-1.239	0.624
protlag	0.089	0.025	3.592	0.000	0.041	0.138
protnearby	0.628	0.091	6.906	0.000	0.449	0.806
kindred	-0.052	0.262	-0.200	0.841	-0.565	0.460
logrefugees	0.043	0.042	1.036	0.300	-0.039	0.126
kindred:logrefugees	-0.004	0.025	-0.151	0.880	-0.052	0.045
ecdis	0.163	0.071	2.301	0.021	0.024	0.302
poldis	0.248	0.066	3.789	0.000	0.120	0.377
ecdis:poldis	-0.076	0.025	-3.044	0.002	-0.125	-0.027
democracy	0.039	0.009	4.588	0.000	0.023	0.056
GDP	-0.000	0.000	-1.707	0.088	-0.000	0.000
numcode Var	0.799	0.125				

Deze tabel bevat de resultaten van de analyse van het statistisch model. Er zijn significante verbanden gevonden tussen afhankelijke variabele prot en onafhankelijke variabelen protnearby, ecdis en poldis evenals de interactie tussen ecdis en poldis.

Omdat de dataset van nu enigszins afwijkt van de dataset die ik bij mijn masterscriptie gemaakt heb, verschillen de resultaten ook tot op bepaalde hoogte. Het is echter wel zo dat de significante coefficienten in bovenstaande tabel sterk overeenkomen met de significante coefficienten in mijn originele analyse. Hieruit is te concluderen dat het project met succes is voltooid.