EEP 153 Project 1 New Draft_

February 9, 2022

1 EEP 153 Project 1: Group Kitagawa

1.0.1 Countries of Investigation: The Baltic States (Estonia, Latvia, and Lithuania)

Description: Originally we wanted to see if there was convergence in population trends for countries who joined the EU in 2004 (mostly post-soviet states). What stood out to us most was a very similar pattern in the three Baltic states: their populations were steadily increasing until 1994, after which they experienced a steep decline. Interested in this abnormal trend, we decided to narrow our countries of interest down to Estonia, Latvia, and Lithuania, and see what exactly caused this pattern after the fall of the USSR. Preliminary research revealed that after the USSR collapsed, these countries' healthcare systems rapidly deteriorated while poverty spiked (along with poverty, homelessness, and drug use). The combined effect was a major Tuberculosis (and HIV) epidemic around 1994. Since we only have more comprehensive data on these states starting in 1995, we decided to see how economic indicators have determined population growth and TB incidence since then. Although GDPPC is a poor measure of poverty (doesn't account for distribution), it's the most accurate indicator of poverty we have for these countries. Evidence suggests that GDPPC is strongly correlated with population, and we maintain that public health is one intermediary through which national income determines population trends.

1.1 Import All Data Libraries

[1]:
```python
## uncomment lines below if installation is needed
!pip install wbdata
!pip install cufflinks
!pip install iso3166

import wbdata
import numpy as np
import plotly.offline as py
import plotly.graph_objs as go
import pandas as pd
from iso3166 import countries
import cufflinks as cf
cf.go_offline()
```

```
Requirement already satisfied: wbdata in /opt/conda/lib/python3.9/site-packages
(0.3.0)
Requirement already satisfied: tabulate>=0.8.5 in /opt/conda/lib/python3.9/site-
```

```
packages (from wbdata) (0.8.9)
Requirement already satisfied: requests>=2.0 in /opt/conda/lib/python3.9/site-
packages (from wbdata) (2.26.0)
Requirement already satisfied: appdirs<2.0,>=1.4 in
/opt/conda/lib/python3.9/site-packages (from wbdata) (1.4.4)
Requirement already satisfied: decorator>=4.0 in /opt/conda/lib/python3.9/site-
packages (from wbdata) (5.0.9)
Requirement already satisfied: charset-normalizer~=2.0.0; python_version >= "3"
in /opt/conda/lib/python3.9/site-packages (from requests>=2.0->wbdata) (2.0.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.9/site-packages (from requests>=2.0->wbdata) (2019.11.28)
Requirement already satisfied: idna<4,>=2.5; python_version >= "3" in
/opt/conda/lib/python3.9/site-packages (from requests>=2.0->wbdata) (2.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.9/site-packages (from requests>=2.0->wbdata) (1.25.7)
Requirement already satisfied: cufflinks in /opt/conda/lib/python3.9/site-
packages (0.17.3)
Requirement already satisfied: colorlover>=0.2.1 in
/opt/conda/lib/python3.9/site-packages (from cufflinks) (0.3.0)
Requirement already satisfied: pandas>=0.19.2 in /opt/conda/lib/python3.9/site-
packages (from cufflinks) (1.3.5)
Requirement already satisfied: numpy>=1.9.2 in /opt/conda/lib/python3.9/site-
packages (from cufflinks) (1.21.5)
Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.9/site-
packages (from cufflinks) (1.16.0)
Requirement already satisfied: setuptools>=34.4.1 in
/opt/conda/lib/python3.9/site-packages (from cufflinks) (58.2.0)
Requirement already satisfied: ipywidgets>=7.0.0 in
/opt/conda/lib/python3.9/site-packages (from cufflinks) (7.6.5)
Requirement already satisfied: plotly>=4.1.1 in /opt/conda/lib/python3.9/site-
packages (from cufflinks) (5.2.1)
Requirement already satisfied: ipython>=5.3.0 in /opt/conda/lib/python3.9/site-
packages (from cufflinks) (8.0.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.9/site-packages (from pandas>=0.19.2->cufflinks) (2.8.0)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.9/site-
packages (from pandas>=0.19.2->cufflinks) (2021.1)
Requirement already satisfied: ipykernel>=4.5.1 in
/opt/conda/lib/python3.9/site-packages (from ipywidgets>=7.0.0->cufflinks)
(6.7.0)
Requirement already satisfied: nbformat>=4.2.0 in /opt/conda/lib/python3.9/site-
packages (from ipywidgets>=7.0.0->cufflinks) (5.1.3)
Requirement already satisfied: jupyterlab-widgets>=1.0.0; python_version >=
"3.6" in /opt/conda/lib/python3.9/site-packages (from
ipywidgets>=7.0.0->cufflinks) (1.0.2)
Requirement already satisfied: ipython-genutils~=0.2.0 in
/opt/conda/lib/python3.9/site-packages (from ipywidgets>=7.0.0->cufflinks)
(0.2.0)
```

```
Requirement already satisfied: widgetsnbextension~=3.5.0 in
/opt/conda/lib/python3.9/site-packages (from ipywidgets>=7.0.0->cufflinks)
(3.5.2)
Requirement already satisfied: traitlets>=4.3.1 in
/opt/conda/lib/python3.9/site-packages (from ipywidgets>=7.0.0->cufflinks)
(5.1.1)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/lib/python3.9/site-
packages (from plotly>=4.1.1->cufflinks) (8.0.1)
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (0.7.5)
Requirement already satisfied: stack-data in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (0.1.4)
Requirement already satisfied: pexpect>4.3; sys_platform != "win32" in
/opt/conda/lib/python3.9/site-packages (from ipython>=5.3.0->cufflinks) (4.8.0)
Requirement already satisfied: matplotlib-inline in
/opt/conda/lib/python3.9/site-packages (from ipython>=5.3.0->cufflinks) (0.1.3)
Requirement already satisfied: pygments in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (2.11.2)
Requirement already satisfied: backcall in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (0.2.0)
Requirement already satisfied: black in /opt/conda/lib/python3.9/site-packages
(from ipython>=5.3.0->cufflinks) (22.1.0)
Requirement already satisfied: jedi>=0.16 in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (0.18.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/opt/conda/lib/python3.9/site-packages (from ipython>=5.3.0->cufflinks) (3.0.26)
Requirement already satisfied: decorator in /opt/conda/lib/python3.9/site-
packages (from ipython>=5.3.0->cufflinks) (5.0.9)
Requirement already satisfied: nest-asyncio in /opt/conda/lib/python3.9/site-
packages (from ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (1.5.4)
Requirement already satisfied: debugpy<2.0,>=1.0.0 in
/opt/conda/lib/python3.9/site-packages (from
ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (1.5.1)
Requirement already satisfied: jupyter-client<8.0 in
/opt/conda/lib/python3.9/site-packages (from
ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (7.1.2)
Requirement already satisfied: tornado<7.0,>=4.2 in
/opt/conda/lib/python3.9/site-packages (from
ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (6.1)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in
/opt/conda/lib/python3.9/site-packages (from
nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (4.4.0)
Requirement already satisfied: jupyter-core in /opt/conda/lib/python3.9/site-
packages (from nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (4.9.1)
Requirement already satisfied: notebook>=4.4.1 in /opt/conda/lib/python3.9/site-
packages (from widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (6.4.7)
Requirement already satisfied: pure-eval in /opt/conda/lib/python3.9/site-
packages (from stack-data->ipython>=5.3.0->cufflinks) (0.2.2)
```

```
Requirement already satisfied: asttokens in /opt/conda/lib/python3.9/site-
packages (from stack-data->ipython>=5.3.0->cufflinks) (2.0.5)
Requirement already satisfied: executing in /opt/conda/lib/python3.9/site-
packages (from stack-data->ipython>=5.3.0->cufflinks) (0.8.2)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.9/site-
packages (from pexpect>4.3; sys_platform != "win32"->ipython>=5.3.0->cufflinks)
(0.7.0)
Requirement already satisfied: pathspec>=0.9.0 in /opt/conda/lib/python3.9/site-
packages (from black->ipython>=5.3.0->cufflinks) (0.9.0)
Requirement already satisfied: mypy-extensions>=0.4.3 in
/opt/conda/lib/python3.9/site-packages (from black->ipython>=5.3.0->cufflinks)
(0.4.3)
Requirement already satisfied: click>=8.0.0 in /opt/conda/lib/python3.9/site-
packages (from black->ipython>=5.3.0->cufflinks) (8.0.3)
Requirement already satisfied: platformdirs>=2 in /opt/conda/lib/python3.9/site-
packages (from black->ipython>=5.3.0->cufflinks) (2.3.0)
Requirement already satisfied: typing-extensions>=3.10.0.0; python_version <
"3.10" in /opt/conda/lib/python3.9/site-packages (from
black->ipython>=5.3.0->cufflinks) (4.0.1)
Requirement already satisfied: tomli>=1.1.0 in /opt/conda/lib/python3.9/site-
packages (from black->ipython>=5.3.0->cufflinks) (2.0.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/opt/conda/lib/python3.9/site-packages (from
jedi>=0.16->ipython>=5.3.0->cufflinks) (0.8.3)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.9/site-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=5.3.0->cufflinks)
(0.2.5)
Requirement already satisfied: pyzmq>=13 in /opt/conda/lib/python3.9/site-
packages (from jupyter-
client<8.0->ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (22.3.0)
Requirement already satisfied: entrypoints in /opt/conda/lib/python3.9/site-
packages (from jupyter-
client<8.0->ipykernel>=4.5.1->ipywidgets>=7.0.0->cufflinks) (0.3)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/opt/conda/lib/python3.9/site-packages (from
jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (0.18.1)
Requirement already satisfied: attrs>=17.4.0 in /opt/conda/lib/python3.9/site-
packages (from
jsonschema!=2.5.0,>=2.4->nbformat>=4.2.0->ipywidgets>=7.0.0->cufflinks) (19.3.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.9/site-packages
(from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(3.0.3)
Requirement already satisfied: prometheus-client in
/opt/conda/lib/python3.9/site-packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(0.13.1)
Requirement already satisfied: Send2Trash>=1.8.0 in
/opt/conda/lib/python3.9/site-packages (from
```

```
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(1.8.0)
Requirement already satisfied: argon2-cffi in /opt/conda/lib/python3.9/site-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(21.3.0)
Requirement already satisfied: nbconvert in /opt/conda/lib/python3.9/site-
packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(6.4.0)
Requirement already satisfied: terminado>=0.8.3 in
/opt/conda/lib/python3.9/site-packages (from
notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks)
(0.13.1)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.9/site-
packages (from jinja2->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7
.0.0->cufflinks) (2.0.1)
Requirement already satisfied: argon2-cffi-bindings in
/opt/conda/lib/python3.9/site-packages (from argon2-cffi->notebook>=4.4.1->widge
tsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (21.2.0)
Requirement already satisfied: bleach in /opt/conda/lib/python3.9/site-packages
(from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
cufflinks) (4.1.0)
Requirement already satisfied: mistune<2,>=0.8.1 in
/opt/conda/lib/python3.9/site-packages (from nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.8.4)
Requirement already satisfied: testpath in /opt/conda/lib/python3.9/site-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->cufflinks) (0.5.0)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in
/opt/conda/lib/python3.9/site-packages (from nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.5.10)
Requirement already satisfied: pandocfilters>=1.4.1 in
/opt/conda/lib/python3.9/site-packages (from nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (1.5.0)
Requirement already satisfied: jupyterlab-pygments in
/opt/conda/lib/python3.9/site-packages (from nbconvert->notebook>=4.4.1->widgets
nbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (0.1.2)
Requirement already satisfied: defusedxml in /opt/conda/lib/python3.9/site-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->cufflinks) (0.7.1)
Requirement already satisfied: cffi>=1.0.1 in /opt/conda/lib/python3.9/site-
packages (from argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbexte
nsion~=3.5.0->ipywidgets>=7.0.0->cufflinks) (1.14.6)
Requirement already satisfied: webencodings in /opt/conda/lib/python3.9/site-
packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
ywidgets>=7.0.0->cufflinks) (0.5.1)
Requirement already satisfied: packaging in /opt/conda/lib/python3.9/site-
```

```
packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
ywidgets>=7.0.0->cufflinks) (21.3)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.9/site-
packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->
widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (2.20)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.9/site-packages (from packaging->bleach->nbconvert->noteb
ook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->cufflinks) (3.0.7)
Requirement already satisfied: iso3166 in /opt/conda/lib/python3.9/site-packages
(2.0.2)
```

```
/opt/conda/lib/python3.9/site-packages/geopandas/_compat.py:111: UserWarning:

The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS
version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both
will be slow.
```

[2]:
```python
#if need be, use the comand below to get ISO alpha-3 country code
#example:
countries.get('estonia').alpha3
countries.get('latvia').alpha3
countries.get('lithuania').alpha3
```

[2]: 'LTU'

[3]:
```python
#Or, use the following command to get the country name, given its ISO3 code
#example
countries.get('EST').name
```

[3]: 'Estonia'

For the following statistical exploration, we are using **source 40: Population estimates and projections** and a few indicators within this database. To see all possible datasets, use `wbdata.get_source()`; to see all indicators in a dataset, use `wbdata.get_indicator(source=SOURCE_id)`.

1.2 [#A] Population DataFrames

A function that returns a `pandas` DataFrame indexed by Region or Country and Year, with columns giving counts of people in different age-sex groups.

Input Parameters: - **sex**: a str ('Female','Male', or 'All') - **age_range**: a tuple with a lower bound and a higher bound, between 0 to 100 - **place**: a str (the ISO 3166-1 alpha-3 code of a country or region) - **year** (optional): an int between 1960 to 2020; if no year is specified, the returned dataframe contains staticstics from 1960 to 2020

[4]:
```python
def population_df(sex, age_range, place, year = None):
```

```
def age_generate(age_range):
    lower = 5* round(age_range[0] / 5)
    upper = 5* round(age_range[1] / 5)
    age_code = []
    if upper < 80:
        while lower < upper:
            age_code.append(f"{lower:02d}{lower+4:02d}")
            lower += 5
        return age_code
    else:
        while lower < 79:
            age_code.append(f"{lower:02d}{lower+4:02d}")
            lower += 5
        age_code.append('80UP')
        return age_code

age_list = age_generate(age_range)

def label_generate(sex, age_ls):
    prefix = sex[:2].upper()
    indicators = {}
    for i in range(len(age_ls)):
        indicators[f"SP.POP.{age_ls[i]}.{prefix}"] = f"{sex} ages␣
↪{age_ls[i][:2]}-{age_ls[i][2:4]}"
    return indicators

if sex != "All":
    indicator_labels = label_generate(sex, age_list)
else:
    male_labels = label_generate("Male", age_list)
    female_labels = label_generate("Female", age_list)

    male_labels.update(female_labels)
    indicator_labels = male_labels

pdf = wbdata.get_dataframe(indicator_labels, country = place)

def clean_df(df, year = None):
    df.reset_index(inplace=True)
    df['date'] = df['date'].astype(int)
    df.set_index('date', inplace = True)
    df['Total Population in Given Range'] = df.loc[:].sum(axis=1)
    if year != None:
        df.query(f'date=={year}', inplace = True)

clean_df(pdf, year)
return pdf
```

One testing example:

[5]:
```
world_female = population_df("Female", (47,100), "WLD", year = 2002)
world_female
```

[5]:
```
      Female ages 45-49  Female ages 50-54  Female ages 55-59  \
date
2002        172436167.0        144642454.0        111974477.0

      Female ages 60-64  Female ages 65-69  Female ages 70-74  \
date
2002         96861604.0         83341490.0         66225590.0

      Female ages 75-79  Female ages 80-UP  Total Population in Given Range
date
2002         48357921.0         50298053.0                      774137756.0
```

1.2.1 1.Population Tabulation for Estonia

[6]:
```
estonia = population_df("All", (1,100), "EST")
estonia
```

[6]:
```
      Male ages 00-04  Male ages 05-09  Male ages 10-14  Male ages 15-19  \
date
2021              NaN              NaN              NaN              NaN
2020          35955.0          38207.0          38866.0          32086.0
2019          35743.0          38648.0          38094.0          30917.0
2018          34999.0          39377.0          36908.0          29869.0
2017          34419.0          39930.0          35509.0          29256.0
…                 …                …                …                …
1964          50217.0          49909.0          50254.0          48789.0
1963          49920.0          49550.0          49847.0          46057.0
1962          49498.0          49221.0          49164.0          43385.0
1961          48955.0          48885.0          47977.0          41537.0
1960          48637.0          48536.0          46365.0          40944.0

      Male ages 20-24  Male ages 25-29  Male ages 30-34  Male ages 35-39  \
date
2021              NaN              NaN              NaN              NaN
2020          29666.0          46429.0          53894.0          47694.0
2019          31675.0          47720.0          53040.0          46754.0
2018          34090.0          48714.0          51707.0          46136.0
2017          36343.0          49672.0          50112.0          45845.0
…                 …                …                …                …
1964          44629.0          52926.0          49113.0          45928.0
1963          45889.0          51739.0          48961.0          42656.0
1962          47756.0          50167.0          48892.0          38997.0
```

```
1961         49332.0          48848.0          48271.0          35489.0
1960         50195.0          48177.0          46848.0          32526.0

      Male ages 40-44  Male ages 45-49  …  Female ages 40-44  \
date                                    …
2021              NaN              NaN  …                NaN
2020          45834.0          49185.0  …            43041.0
2019          46441.0          47285.0  …            43813.0
2018          47221.0          44763.0  …            44849.0
2017          47711.0          42507.0  …            45701.0
…                …                …  …                  …
1964          30396.0          24949.0  …            47361.0
1963          27900.0          26256.0  …            43257.0
1962          25979.0          28080.0  …            39105.0
1961          24835.0          29788.0  …            36018.0
1960          24598.0          31043.0  …            34720.0

      Female ages 45-49  Female ages 50-54  Female ages 55-59  \
date
2021                NaN                NaN                NaN
2020            47508.0            39096.0            46754.0
2019            46207.0            40161.0            47497.0
2018            44370.0            42117.0            47681.0
2017            42799.0            44239.0            47757.0
…                  …                  …                  …
1964            34924.0            46112.0            44101.0
1963            36964.0            46153.0            43593.0
1962            40014.0            45573.0            43164.0
1961            42823.0            44875.0            42713.0
1960            44708.0            44463.0            42266.0

      Female ages 60-64  Female ages 65-69  Female ages 70-74  \
date
2021                NaN                NaN                NaN
2020            47282.0            49451.0            32731.0
2019            48013.0            46798.0            33582.0
2018            48711.0            43915.0            34982.0
2017            48784.0            41924.0            35841.0
…                  …                  …                  …
1964            40546.0            34178.0            26836.0
1963            39678.0            33274.0            26310.0
1962            38830.0            32497.0            25925.0
1961            38021.0            31862.0            25611.0
1960            37352.0            31435.0            25358.0

      Female ages 75-79  Female ages 80-UP  Total Population in Given Range
date
```

```
2021              NaN            NaN                           0.0
2020          36274.0        58673.0                     1331060.0
2019          36135.0        57184.0                     1326899.0
2018          35906.0        55565.0                     1321977.0
2017          36110.0        53696.0                     1317383.0
…                  …          …                              …
1964          19395.0        16906.0                     1277083.0
1963          18934.0        16462.0                     1258856.0
1962          18495.0        15883.0                     1241625.0
1961          18013.0        15206.0                     1225079.0
1960          17457.0        14525.0                     1211540.0

[62 rows x 35 columns]
```

1.2.2 2.Population Tabulation for Latvia

[7]:
```
latvia = population_df("All", (1,100), "LVA")
latvia
```

[7]:
```
      Male ages 00-04  Male ages 05-09  Male ages 10-14  Male ages 15-19  \
date
2021              NaN              NaN              NaN              NaN
2020          59925.0          49127.0          52985.0          44197.0
2019          58850.0          50228.0          52323.0          42416.0
2018          55814.0          52010.0          51139.0          41015.0
2017          52243.0          53552.0          49810.0          40618.0
…                 …               …               …               …
1964          88837.0          86560.0          82018.0          78943.0
1963          88274.0          85297.0          79586.0          78689.0
1962          87265.0          83712.0          77203.0          79005.0
1961          86263.0          81744.0          75230.0          79594.0
1960          85321.0          79341.0          73783.0          80035.0

      Male ages 20-24  Male ages 25-29  Male ages 30-34  Male ages 35-39  \
date
2021              NaN              NaN              NaN              NaN
2020          36453.0          61794.0          73341.0          61917.0
2019          40800.0          65950.0          72516.0          61746.0
2018          46700.0          69232.0          71254.0          62184.0
2017          53107.0          71707.0          70146.0          62868.0
…                 …               …               …               …
1964          86033.0          91040.0          86198.0          77874.0
1963          86497.0          89092.0          85364.0          72389.0
1962          86663.0          87172.0          84378.0          66355.0
1961          86367.0          85676.0          82473.0          60438.0
1960          85581.0          84506.0          79206.0          55001.0
```

```
      Male ages 40-44  Male ages 45-49  …  Female ages 40-44  \
date                                    …
2021              NaN              NaN  …                NaN
2020          60104.0          65153.0  …            61626.0
2019          61515.0          64440.0  …            63369.0
2018          63329.0          63244.0  …            65426.0
2017          65176.0          62350.0  …            67425.0
…                   …                …  …                  …
1964          51448.0          41289.0  …            85169.0
1963          46610.0          44442.0  …            76476.0
1962          42679.0          48802.0  …            67559.0
1961          40319.0          53053.0  …            60940.0
1960          39842.0          56251.0  …            57987.0

      Female ages 45-49  Female ages 50-54  Female ages 55-59  \
date
2021                NaN                NaN                NaN
2020            69069.0            63062.0            79216.0
2019            68405.0            65722.0            79230.0
2018            67246.0            69543.0            78401.0
2017            66490.0            73311.0            77687.0
…                     …                  …                  …
1964            58620.0            82839.0            77779.0
1963            63247.0            82848.0            77347.0
1962            70022.0            81558.0            76984.0
1961            76184.0            80136.0            76241.0
1960            79944.0            79139.0            74779.0

      Female ages 60-64  Female ages 65-69  Female ages 70-74  \
date
2021                NaN                NaN                NaN
2020            73811.0            73039.0            48278.0
2019            73982.0            68519.0            51092.0
2018            74428.0            63818.0            55618.0
2017            74020.0            60780.0            59544.0
…                     …                  …                  …
1964            71231.0            56381.0            44667.0
1963            68834.0            54805.0            43760.0
1962            66245.0            53637.0            43007.0
1961            63807.0            52708.0            42347.0
1960            61677.0            51816.0            41686.0

      Female ages 75-79  Female ages 80-UP  Total Population in Given Range
date
2021                NaN                NaN                              0.0
2020            60901.0            81037.0                        1901548.0
2019            59146.0            82141.0                        1913822.0
```

```
2018          56812.0       83146.0                      1927174.0
2017          55888.0       82930.0                      1942249.0
…                …             …                            …
1964          32341.0       30683.0                      2240627.0
1963          31531.0       30174.0                      2210919.0
1962          30847.0       29382.0                      2181585.0
1961          30252.0       28302.0                      2152682.0
1960          29620.0       26915.0                      2120981.0

[62 rows x 35 columns]
```

1.2.3 3.Population Tabulation for Lithuania

[8]:
```python
lithuania = population_df("All", (1,100), "LTU")
lithuania
```

[8]:
```
      Male ages 00-04  Male ages 05-09  Male ages 10-14  Male ages 15-19  \
date
2021              NaN              NaN              NaN              NaN
2020          76730.0          82275.0          63163.0          64723.0
2019          76760.0          77629.0          62751.0          66162.0
2018          76790.0          73484.0          63697.0          68381.0
2017          77383.0          70874.0          65379.0          72512.0
…                 …                …                …                …
1964         146762.0         139478.0         121031.0         112532.0
1963         146367.0         135523.0         118956.0         113720.0
1962         145534.0         131777.0         117513.0         115215.0
1961         144014.0         128372.0         116527.0         116229.0
1960         140974.0         125472.0         116052.0         116368.0

      Male ages 20-24  Male ages 25-29  Male ages 30-34  Male ages 35-39  \
date
2021              NaN              NaN              NaN              NaN
2020          70274.0          98520.0          99409.0          80264.0
2019          77620.0         101721.0          95743.0          80006.0
2018          85933.0         102728.0          92092.0          80930.0
2017          94239.0         102309.0          89762.0          82844.0
…                 …                …                …                …
1964         114613.0         113025.0         111993.0          99662.0
1963         114455.0         112854.0         111100.0          94795.0
1962         113574.0         112956.0         109658.0          89288.0
1961         112255.0         112925.0         107188.0          83276.0
1960         111033.0         112479.0         103640.0          77065.0

      Male ages 40-44  Male ages 45-49  …  Female ages 40-44  \
date                                    …
2021              NaN              NaN  …                NaN
```

```
2020          82258.0          96679.0  …            83208.0
2019          85182.0          95768.0  …            87653.0
2018          88342.0          94190.0  …            92309.0
2017          91505.0          93784.0  …            96909.0
…                …             …   …              …
1964          70751.0          48724.0  …            98335.0
1963          64604.0          48656.0  …            89447.0
1962          58905.0          50022.0  …            80517.0
1961          54268.0          52679.0  …            73552.0
1960          51175.0          56512.0  …            69829.0

      Female ages 45-49  Female ages 50-54  Female ages 55-59  \
date
2021                NaN                NaN                NaN
2020           103231.0            99414.0           131402.0
2019           102864.0           104592.0           128045.0
2018           101611.0           111287.0           122793.0
2017           101579.0           117536.0           118488.0
…                   …                  …                …
1964            68121.0            85918.0            84095.0
1963            71179.0            86793.0            83196.0
1962            76269.0            86542.0            82309.0
1961            81068.0            85759.0            81315.0
1960            84179.0            85005.0            80161.0

      Female ages 60-64  Female ages 65-69  Female ages 70-74  \
date
2021                NaN                NaN                NaN
2020           108258.0           102640.0            71714.0
2019           106197.0            96053.0            73224.0
2018           105212.0            89487.0            76694.0
2017           103797.0            85426.0            80225.0
…                    …                  …                …
1964            75382.0            60675.0            42778.0
1963            73933.0            58085.0            42223.0
1962            72388.0            55619.0            41961.0
1961            70547.0            53522.0            41466.0
1960            68473.0            51953.0            40459.0

      Female ages 75-79  Female ages 80-UP  Total Population in Given Range
date
2021                NaN                NaN                              0.0
2020            79711.0           127154.0                        2794701.0
2019            78193.0           125227.0                        2794137.0
2018            76341.0           123142.0                        2801542.0
2017            76104.0           120407.0                        2828402.0
…                    …                  …                             …
```

```
1964            31363.0            26508.0                               2935203.0
1963            29948.0            25226.0                               2898949.0
1962            28176.0            24096.0                               2863351.0
1961            26129.0            23131.0                               2823548.0
1960            23939.0            22377.0                               2778550.0

[62 rows x 35 columns]
```

1.3 [#A] Population Statistics

Using the previously defined `population_df` function, a `python` function named `population` is created to return a string with population statistics in the following form:

- In [year], there are [number] of [people/males/females] aged [low] to [high] living in [the world/region/country].

Input Parameters: - **year**: a int between 1960 to 202 - **sex**: a str ('Female','Male', or 'All') - **age_range**: a tuple with a lower bound and a higher bound, between 0 to 100 - **place**: a str (the ISO 3166-1 alpha-3 code of a country or region)

[9]:
```python
def population(year_defined, sex, age_range, place):
    df = population_df(sex, age_range, place, year = year_defined)
    population = df['Total Population in Given Range'].iloc[0]
    return population
```

One testing examples:

[10]:
```python
population(2002, "Female", (47,100), "WLD")
```

[10]: 774137756.0

1.3.1 1.Population of Estonia in 1990

[11]:
```python
population(1990,"All", (1,100), "EST")
```

[11]: 1569173.0

1.3.2 2.Population of Latvia in 1990

[12]:
```python
population(1990,"All", (1,100), "LVA")
```

[12]: 2663151.0

1.3.3 3.Population of Lithuania in 1990

[13]:
```python
population(1990,"All", (1,100), "LTU")
```

[13]: 3697836.0

[ ]:

1.3.4 Additional Function

In additin, a **python** function named **population_statement** is created to return a string with population statistics in the following form:

- In [year], there are [number] of [people/males/females] aged [low] to [high] living in [the world/region/country].

Input Parameters: - **year**: a int between 1960 to 202 - **sex**: a str ('Female','Male', or 'All') - **age_range**: a tuple with a lower bound and a higher bound, between 0 to 100 - **place**: a str (the ISO 3166-1 alpha-3 code of a country or region)

[14]:
```python
def population_statement(year_defined, sex, age_range, place):
    df = population_df(sex, age_range, place, year = year_defined)
    if sex == "All":
        ppl = 'people'
    else:
        ppl = f"{sex.lower()}s"
    lower = age_range[0]
    higher = age_range[1]
    if place == 'WLD':
        country_name = 'the world'
    else:
        country_name = countries.get(place).name
    population = df['Total Population in Given Range'].iloc[0]

    answer = f"In {year_defined}, there are {population} {ppl} aged {lower} to␣
 ↪{higher} living in {country_name} by approximation."

    return answer
```

One testing examples:

[15]:
```python
population_statement(2002, "Female", (47,100), "WLD")
```

[15]: 'In 2002, there are 774137756.0 females aged 47 to 100 living in the world by approximation.'

[ ]:

1.4 [#B] Population Pyramids

1.4.1 Static Population Pyramid Function

The `pop_pyramid` funciton returns a static population pyramid for either the population of a single year or the populations between a year range with a specified increment of years.

Input Parameters: - **dataframe**: the name of the established dataframe for a specific country - **country_name**: A string (e.g. "Estonia") - **year**: a list - for a single year, it's in the form [year] - for multiple years, it's in the form [end_year, start_year, year_increments]

[16]:
```
def pop_pyramid(dataframe, country_name, year):
    py.init_notebook_mode(connected=True)

    #create titles for the two different forms of population pyramid
    def title_creator(country_name, year):
        if len(year) == 1:
            title = f""
            return title
        else:
            "Population Pyramid of Estonia, 1990-2010"
            title = f"Population Pyramid of {country_name}, {year[1]}-{year[0]}␣
↪({year[2]}-year Increments)"
            return title
    title_ = title_creator(country_name, year)

    layout = go.Layout(barmode='overlay',
                       yaxis=go.layout.YAxis(range=[0, 90], title='Age'),
                       xaxis=go.layout.XAxis(title='Population'),
                      title = title_)

    #create the age labels
    age_ranges = []
    for i in range(0,80,5):
        age_ranges.append(f"{i:02d}"+f"{i+4:02d}")
    age_ranges.append("80UP")

    #bins for one year
    if len(year) == 1:
        bins = [go.Bar(x = dataframe.loc[year[0],:].filter(regex="Male").values,
                       y = [int(s[:2])+1 for s in age_ranges],
                       orientation='h',
                       name='Men',
                       marker=dict(color='blue'),
                       hoverinfo=['x','y']
                       ),
                go.Bar(x = -dataframe.loc[year[0],:].filter(regex="Female").
↪values,
                       y=[int(s[:2])+1 for s in age_ranges],
```

```
                 orientation='h',
                 name='Women',
                 marker=dict(color='red'),
                 hoverinfo=['x','y']
                    )
             ]
    #bins for multiple years
    else:
        years = range(year[0],year[1], -year[2])
        bins = [go.Bar(x = dataframe.loc[year,:].filter(regex="Male").values,
               y = [int(s[:2])+1 for s in age_ranges],
               orientation='h',
               name='Men {:d}'.format(year),
               hoverinfo=['x','y'],
                 opacity=0.5,
                 marker=dict(color='green')
              )
        for year in years]

        bins += [go.Bar(x = -dataframe.loc[year,:].filter(regex="Female").
↪values,
                y=[int(s[:2])+1 for s in age_ranges],
                orientation='h',
                name='Women {:d}'.format(year),
                hoverinfo=['x','y'],
                  opacity=0.5,
                  marker=dict(color='orange')
               )
         for year in years]


    py.iplot(dict(data=bins, layout=layout))
```

1.4.2 Animated Population Pyramid Function

The `animate_pyramid` funciton returns an interactive population pyramid for the chosen country to illustrate the change of population over yearts.

Input Parameters:

[17]:
```
#import data libraries
import ipywidgets
from ipywidgets import interactive, HBox, VBox, fixed

def animate_pyramid(df, country_name, year):
    def animate_helper(df, country_name, year):
            py.init_notebook_mode(connected=True)
```

```
            age_ranges = []
            for i in range(0,80,5):
                age_ranges.append(f"{i:02d}"+f"{i+4:02d}")
            age_ranges.append("80UP")

            layout = go.Layout(barmode='overlay',
                               yaxis=go.layout.YAxis(range=[0, 90],␣
↪title='Age'),
                               xaxis=go.layout.XAxis(title='Population'),
                               title = f"Population Pyramid of {country_name},␣
↪{year}")
            bins = [go.Bar(x = df.loc[year,:].filter(regex="Male").values,
                           y = [int(s[:2])+1 for s in age_ranges],
                           orientation='h',
                           name='Men',
                           marker=dict(color='yellow'),
                           hoverinfo=['x','y']
                          ),
                    go.Bar(x = -df.loc[year,:].filter(regex="Female").values,
                           y=[int(s[:2])+1 for s in age_ranges],
                           orientation='h',
                           name='Women',
                           marker=dict(color='pink'),
                           hoverinfo=['x','y'])
                   ]
            py.iplot(dict(data=bins, layout=layout))
    return ipywidgets.interact(animate_helper, df = fixed(df), year = year,␣
↪country_name = country_name)
```

[ ]:

```

```

1.4.3 1. Estonia

[18]:
```
pop_pyramid(estonia, "Estonia", [2010])
pop_pyramid(estonia, "Estonia", [2020,1960,20])
animate_pyramid(estonia, "Estonia", (1960, 2020, 1))
```

```
interactive(children=(Text(value='Estonia', description='country_name'),␣
↪IntSlider(value=1990, description='ye…
```

[18]:
```
<function __main__.animate_pyramid.<locals>.animate_helper(df, country_name,
year)>
```

1.4.4 2. Latvia

[19]:
```
pop_pyramid(latvia, "Latvia", [2010])
pop_pyramid(latvia, "Latvia", [2020,1960,20])
animate_pyramid(latvia, "Latvia", (1960, 2020, 1))
```

```
interactive(children=(Text(value='Latvia', description='country_name'),␣
↪IntSlider(value=1990, description='yea…
```

[19]:
```
<function __main__.animate_pyramid.<locals>.animate_helper(df, country_name,
year)>
```

1.4.5 3. Lithuania

Population Pyramid from 1990 to 2010 (5-year Increments)

[20]:
```
pop_pyramid(lithuania, "Lithuania", [2010])
pop_pyramid(lithuania, "Lithuania", [2020,1960,20])
animate_pyramid(lithuania, "Lithuania", (1960, 2020, 1))
```

```
interactive(children=(Text(value='Lithuania', description='country_name'),␣
↪IntSlider(value=1990, description='…
```

[20]:
```
<function __main__.animate_pyramid.<locals>.animate_helper(df, country_name,
year)>
```

[ ]:
```

```

1.5 [#C] Population Maps of the Baltic States

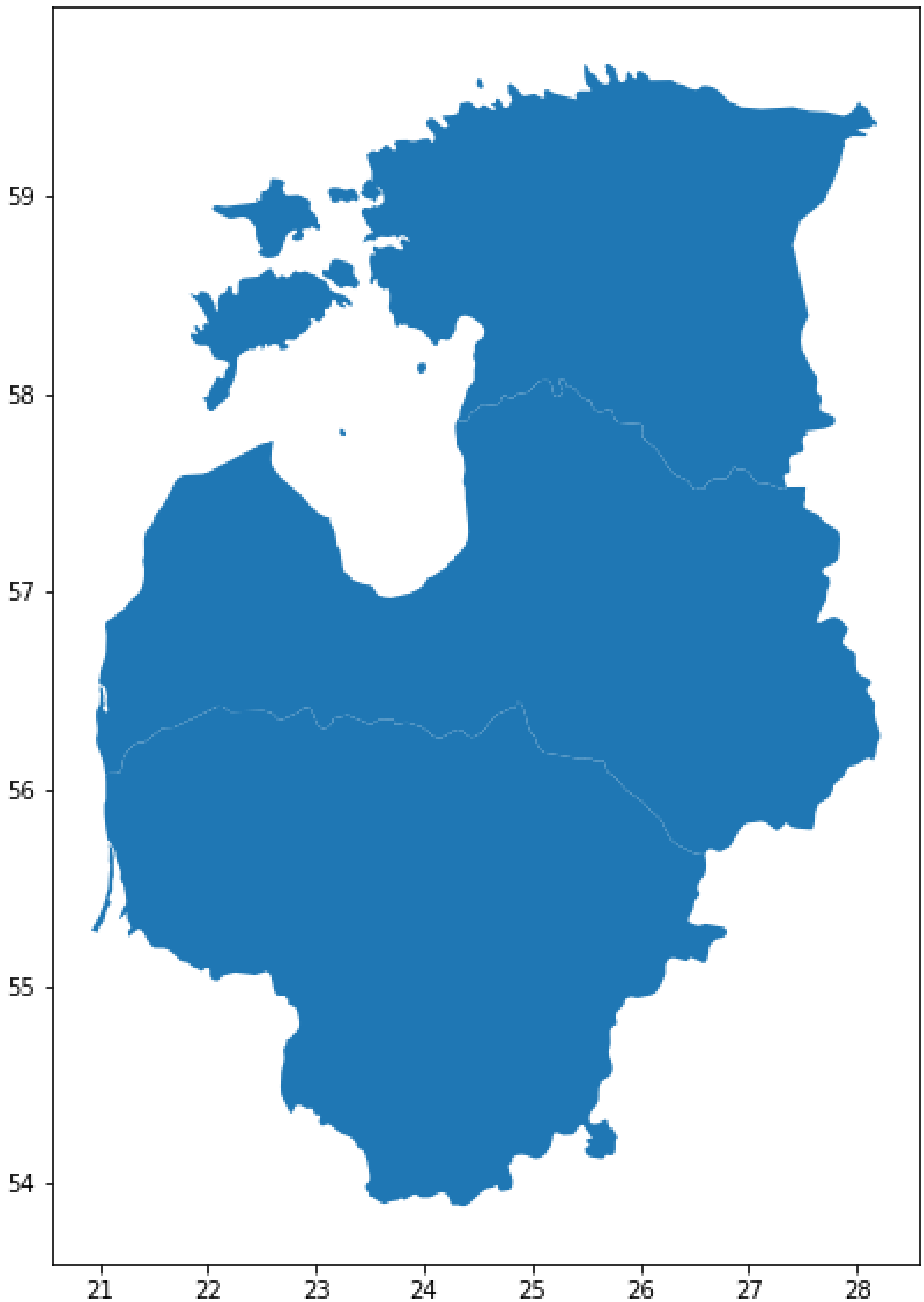
[21]:
```
# Install & import new data libraries
!pip install geopandas
!pip install pyshp


import matplotlib.pyplot as plt
import geopandas as gpd
import shapefile as shp
import seaborn as sns
```

```
Requirement already satisfied: geopandas in /opt/conda/lib/python3.9/site-
packages (0.10.2)
Requirement already satisfied: fiona>=1.8 in /opt/conda/lib/python3.9/site-
packages (from geopandas) (1.8.20)
Requirement already satisfied: shapely>=1.6 in /opt/conda/lib/python3.9/site-
packages (from geopandas) (1.8.0)
Requirement already satisfied: pyproj>=2.2.0 in /opt/conda/lib/python3.9/site-
packages (from geopandas) (3.3.0)
```

```
Requirement already satisfied: pandas>=0.25.0 in /opt/conda/lib/python3.9/site-
packages (from geopandas) (1.3.5)
Requirement already satisfied: click-plugins>=1.0 in
/opt/conda/lib/python3.9/site-packages (from fiona>=1.8->geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in /opt/conda/lib/python3.9/site-
packages (from fiona>=1.8->geopandas) (0.7.2)
Requirement already satisfied: attrs>=17 in /opt/conda/lib/python3.9/site-
packages (from fiona>=1.8->geopandas) (19.3.0)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.9/site-
packages (from fiona>=1.8->geopandas) (58.2.0)
Requirement already satisfied: click>=4.0 in /opt/conda/lib/python3.9/site-
packages (from fiona>=1.8->geopandas) (8.0.3)
Requirement already satisfied: six>=1.7 in /opt/conda/lib/python3.9/site-
packages (from fiona>=1.8->geopandas) (1.16.0)
Requirement already satisfied: munch in /opt/conda/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (2.5.0)
Requirement already satisfied: certifi in /opt/conda/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (2019.11.28)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.9/site-packages (from pandas>=0.25.0->geopandas) (2.8.0)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.9/site-
packages (from pandas>=0.25.0->geopandas) (2021.1)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.9/site-
packages (from pandas>=0.25.0->geopandas) (1.21.5)
Requirement already satisfied: pyshp in /opt/conda/lib/python3.9/site-packages
(2.1.3)
```

[24]:

```
baltics_path = "/home/jovyan//EEP153 Project I/baltic shapefiles/baltics␣
 ↪shapefiles"
map_df = gpd.read_file(baltics_path)
plt.rcParams['figure.figsize'] = [30,10]
map_df.plot();
```


59
58
57
56
55
54
21
22
23
24
25
26
27
28

[25]:
```
population = {"SP.POP.TOTL":"Population"}
baltic = {"EST":"Estonia",
          "LVA":"Latvia",
          "LTU":"Lithuania"}
```

[26]:
```
baltic_pop_df = wbdata.get_dataframe(population, country=baltic).squeeze()
baltic_pop_df = baltic_pop_df.unstack('country')
# Date index is of type string; change to integers
baltic_pop_df.index = baltic_pop_df.index.astype(int)
baltic_pop_df = baltic_pop_df.transpose()
baltic_pop_df.index.name
map_df.index.name
pop_map_df = map_df.merge(baltic_pop_df, left_on="NAME_ID", right_on="country",␣
 ↪how='outer')
pop_map_df
```

[26]:
```
          featurecla  scalerank  LABELRANK SOVEREIGNT SOV_A3  ADM0_DIF  LEVEL  \
0  Admin-0 map unit        0.0        5.0  Lithuania    LTU       0.0    2.0
1  Admin-0 map unit        0.0        6.0    Estonia    EST       0.0    2.0
2  Admin-0 map unit        0.0        5.0     Latvia    LVA       0.0    2.0
3               NaN        NaN        NaN        NaN    NaN       NaN    NaN

                TYPE      ADMIN ADM0_A3  …       2011       2012       2013  \
0  Sovereign country  Lithuania     LTU  …        NaN        NaN        NaN
1  Sovereign country    Estonia     EST  …  1327439.0  1322696.0  1317997.0
2  Sovereign country     Latvia     LVA  …  2059709.0  2034319.0  2012647.0
3                NaN        NaN     NaN  …  3028115.0  2987773.0  2957689.0

        2014       2015       2016       2017       2018       2019       2020
0        NaN        NaN        NaN        NaN        NaN        NaN        NaN
1  1314545.0  1315407.0  1315790.0  1317384.0  1321977.0  1326898.0  1331057.0
2  1993782.0  1977527.0  1959537.0  1942248.0  1927174.0  1913822.0  1901548.0
3  2932367.0  2904910.0  2868231.0  2828403.0  2801543.0  2794137.0  2794700.0

[4 rows x 223 columns]
```

[27]:
```
shape_columns = map_df[['SOVEREIGNT', 'ADMIN', 'geometry']]
merged = shape_columns.merge(baltic_pop_df, left_on="SOVEREIGNT",␣
 ↪right_on="country", how='outer')
merged
```

[27]:
```
  SOVEREIGNT      ADMIN                                           geometry  \
0  Lithuania  Lithuania  MULTIPOLYGON (((26.59453 55.66699, 26.60383 55…
1    Estonia    Estonia  MULTIPOLYGON (((24.30616 57.86819, 24.31666 57…
2     Latvia     Latvia  POLYGON ((27.35293 57.52760, 27.52817 57.52848…

        1960       1961       1962       1963       1964       1965  \
```

```
0  2778550.0  2823550.0  2863350.0  2898950.0  2935200.0  2971450.0
1  1211537.0  1225077.0  1241623.0  1258857.0  1277086.0  1294566.0
2  2120979.0  2152681.0  2181586.0  2210919.0  2240623.0  2265919.0

        1966  …       2011       2012       2013       2014       2015  \
0  3008050.0  …  3028115.0  2987773.0  2957689.0  2932367.0  2904910.0
1  1308597.0  …  1327439.0  1322696.0  1317997.0  1314545.0  1315407.0
2  2283217.0  …  2059709.0  2034319.0  2012647.0  1993782.0  1977527.0

        2016       2017       2018       2019       2020
0  2868231.0  2828403.0  2801543.0  2794137.0  2794700.0
1  1315790.0  1317384.0  1321977.0  1326898.0  1331057.0
2  1959537.0  1942248.0  1927174.0  1913822.0  1901548.0

[3 rows x 64 columns]
```

[28]:
```
merged = merged.set_index('SOVEREIGNT')
```

[29]:
```
landareas = pd.Series([25212, 17505, 24938], index = ['Lithuania', 'Estonia',␣
 ↪'Latvia'])
landareas
```

[29]:
```
Lithuania    25212
Estonia      17505
Latvia       24938
dtype: int64
```

[30]:
```
merged[2020] = merged[2020] / landareas
merged[2020]
```

[30]:
```
SOVEREIGNT
Lithuania    110.848009
Estonia       76.038675
Latvia        76.251023
Name: 2020, dtype: float64
```

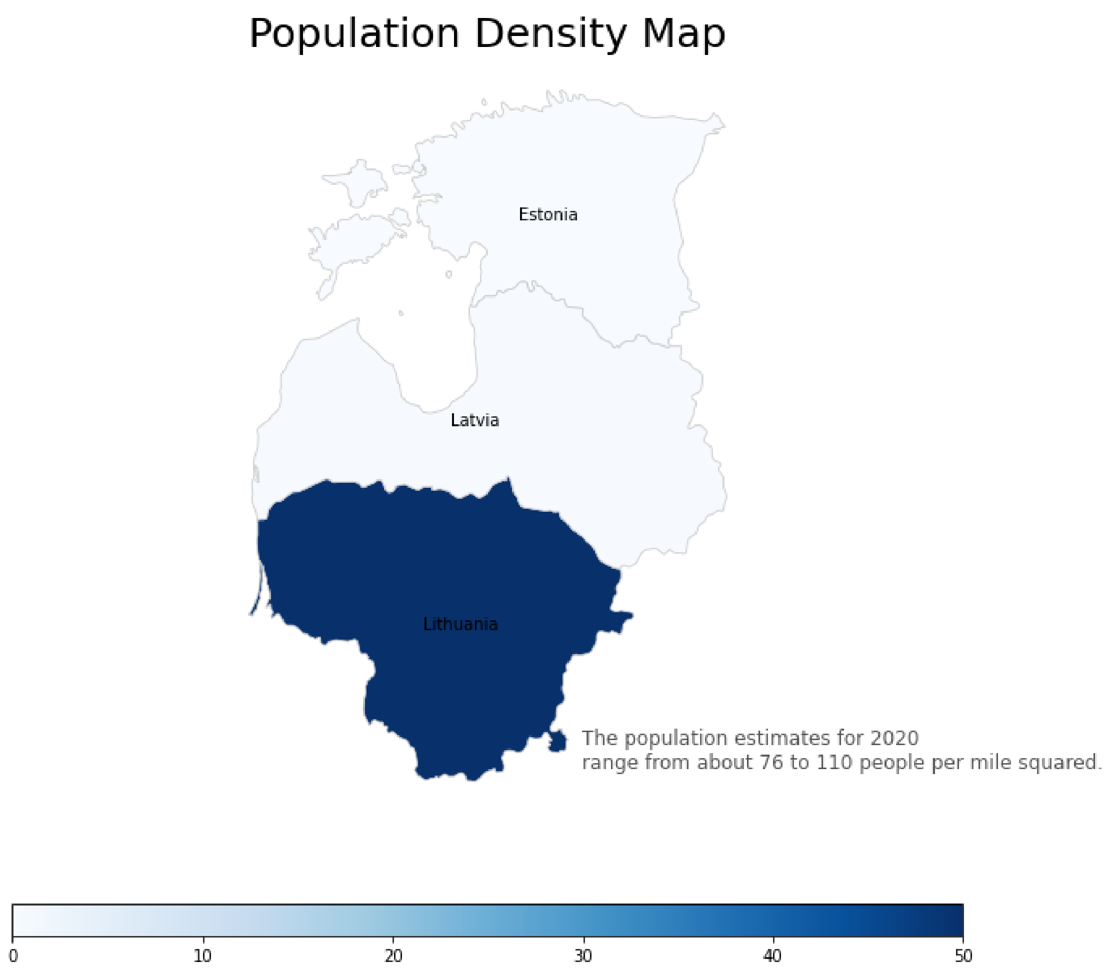
[31]:
```
merged
```

[31]:
```
                ADMIN                                          geometry  \
SOVEREIGNT
Lithuania   Lithuania  MULTIPOLYGON (((26.59453 55.66699, 26.60383 55…
Estonia       Estonia  MULTIPOLYGON (((24.30616 57.86819, 24.31666 57…
Latvia         Latvia  POLYGON ((27.35293 57.52760, 27.52817 57.52848…

                 1960       1961       1962       1963       1964       1965  \
SOVEREIGNT
Lithuania   2778550.0  2823550.0  2863350.0  2898950.0  2935200.0  2971450.0
```

```
Estonia      1211537.0  1225077.0  1241623.0  1258857.0  1277086.0  1294566.0
Latvia       2120979.0  2152681.0  2181586.0  2210919.0  2240623.0  2265919.0

                 1966       1967  …       2011       2012       2013  \
SOVEREIGNT                        …
Lithuania    3008050.0  3044400.0  …  3028115.0  2987773.0  2957689.0
Estonia      1308597.0  1318946.0  …  1327439.0  1322696.0  1317997.0
Latvia       2283217.0  2301220.0  …  2059709.0  2034319.0  2012647.0

                 2014       2015       2016       2017       2018       2019  \
SOVEREIGNT
Lithuania    2932367.0  2904910.0  2868231.0  2828403.0  2801543.0  2794137.0
Estonia      1314545.0  1315407.0  1315790.0  1317384.0  1321977.0  1326898.0
Latvia       1993782.0  1977527.0  1959537.0  1942248.0  1927174.0  1913822.0

                  2020
SOVEREIGNT
Lithuania    110.848009
Estonia       76.038675
Latvia        76.251023

[3 rows x 63 columns]
```

[32]:

```python
variable = 2020
vmin, vmax = 0, 50
fig, ax = plt.subplots(1, figsize=(30,10))
ax.axis('off')
ax.set_title('Population Density Map', fontdict={'fontsize': '25', 'fontweight'␣
 ↪: '3'})
ax.annotate('The population estimates for 2020 \nrange from about 76 to 110␣
 ↪people per mile squared. ',xy=(26, 54),
            xycoords='data', fontsize=12,
            color='#555555')
sm = plt.cm.ScalarMappable(cmap='Blues', norm=plt.Normalize(vmin=vmin,␣
 ↪vmax=vmax))
sm.set_array([])
fig.colorbar(sm, orientation="horizontal", fraction=0.036, pad=0.1, aspect = 30)
merged.plot(column=variable, cmap='Blues', linewidth=0.8, ax=ax, edgecolor='0.
 ↪8')
merged['coords'] = merged['geometry'].apply(lambda x: x.representative_point().
 ↪coords[:])
merged['coords'] = [coords[0] for coords in merged['coords']]
for idx, row in merged.iterrows():
    plt.annotate(text=row['ADMIN'],␣
 ↪xy=row['coords'],horizontalalignment='center')
```



1.6 [#C] Other Indicators

[33]:
```
eumembers = {"ESP":"Spain",
              "AUT":"Austria",
              "EST":"Estonia",
              "FIN":"Finland",
              "GRC":"Greece",
              "LVA":"Latvia",
              "LTU":"Lithuania",
              "MLT":"Malta",
              "POL":"Poland",
              "PRT":"Portugal",
              "SVK":"Slovakia",
              "SVN":"Slovenia",
             "IRL":"Ireland",
             "BGR":"Bulgaria",
             "HRV":"Croatia",
```

```
            "ROU":"Romania",
            "CZE":"Czech Republic",
            "DNK":"Denmark",
            "HUN":"Hungary",
            "POL":"Poland",
            "SWE":"Sweden",
            "BEL":"Belgium",
            "CYP":"Cyprus",
            "FRA":"France",
            "DEU":"Germany",
            "ITA":"Italy",
            "LUX":"Luxembourg",
            "NLD":"The Netherlands"}

original_15 = {"ESP":"Spain",
                "AUT":"Austria",
                "FIN":"Finland",
                "GRC":"Greece",
                "PRT":"Portugal",
                "IRL":"Ireland",
                "DNK":"Denmark",
                "SWE":"Sweden",
                "BEL":"Belgium",
                "FRA":"France",
                "DEU":"Germany",
                "ITA":"Italy",
                "LUX":"Luxembourg",
                "NLD":"The Netherlands",
                "GBR":"United Kingdom"}

baltic = {"EST":"Estonia", "LVA":"Latvia", "LTU":"Lithuania"}
```

[34]:
```
def graph_indicator(indicator_dict, country_dict, graph_title, y_title):
    data = wbdata.get_dataframe(indicator_dict, country = country_dict).
 ↪squeeze().T
    data = data.unstack('country').dropna(axis=0, how='any')
    data.index = data.index.astype(int)
    data.iplot(title = graph_title, yTitle = y_title, xTitle = 'Year')
```

[35]:
```
def graph_log(indicator_dict, country_dict, graph_title, y_title):
    data = wbdata.get_dataframe(indicator_dict, country = country_dict).
 ↪squeeze().T
    data = data.unstack('country').dropna(axis=0, how='any')
    data.index = data.index.astype(int)
    np.log(data).diff().iplot(title = graph_title, yTitle = y_title, xTitle =␣
 ↪'Year')
```

[36]:
```
def eu_comparison_graph(indicator_dict, graph_title, y_title):
    all_eu = wbdata.get_dataframe(indicator_dict, country = eumembers)
    all_eu = all_eu.unstack('country').dropna(axis=0, how='any').
 ↪droplevel(level=0, axis = 1)
    all_eu['EU27'] = all_eu.sum(axis=1) / 27

    originals = wbdata.get_dataframe(indicator_dict, country = original_15)
    originals = originals.unstack('country').dropna(axis=0, how='any').
 ↪droplevel(level=0, axis = 1)
    all_eu['EU15'] = originals.sum(axis=1) / 15
    all_eu.index = all_eu.index.astype(int)

    comparison = all_eu[['Estonia', 'Latvia', 'Lithuania', 'EU27', 'EU15']]
    comparison.iplot(title = graph_title, yTitle = y_title, xTitle = 'Year')
```

[37]:
```
graph_indicator({"SP.POP.TOTL":"Population"}, baltic, "Baltic Population Peak␣
 ↪in 1990", "Population")
```

[38]:
```
graph_log({"SP.POP.TOTL":"Population"}, baltic, "Population Growth in the␣
 ↪Baltic Countries", "Growth Rate")
```

[39]:
```
eu_comparison_graph({"SP.DYN.AMRT.MA":"Male Mortality"}, "Male Mortality Rates␣
 ↪in the Baltics vs the EU", "Mortality Rate")
```

[40]:
```
eu_comparison_graph({"SP.DYN.AMRT.FE":"Female Mortality"}, "Female Mortality␣
 ↪Rates in the Baltics vs the EU", "Mortality Rate")
```

[41]:
```
eu_comparison_graph({"NY.GDP.PCAP.CD":"GDP per capita"}, "GDP per capita in the␣
 ↪Baltics vs the EU", "GDP per capita")
```

1.7 [#C] Other Indicators: Tuberculosis and Poverty

[42]:
```
adults = {"SP.POP.TOTL":"total pop",
               "SP.POP.GROW":"pop growth rate",
              "SP.DYN.AMRT.MA":"male mortality",
              "SP.DYN.AMRT.FE":"female mortality",
              "SP.POP.1564.FE.ZS":"% adult female",
              "SP.POP.TOTL.FE.ZS":"% female",
              "NY.GDP.PCAP.CD":"gdppc",
              "SP.DYN.TFRT.IN":"tfr",
          "SP.POP.0014.MA.IN": "Population ages 0-14, male",
          "SP.POP.1519.MA.5Y": "Population ages 15-19, male (% of male␣
 ↪population)",
          "SP.POP.TOTL.MA.IN": "Population, male",
          "SP.POP.0014.FE.IN": "Population ages 0-14, female",
```

```
          "SP.POP.1519.FE.5Y": "Population ages 15-19, female (% of female␣
 ↪population)",
           "SP.POP.TOTL.FE.IN": "Population, female",
          "SP.DYN.CDRT.IN": "crude death rate",
          "SP.POP.1564.FE.IN": "Population ages 15-64, female",
          "SP.POP.6064.FE":    "Population ages 60-64, female",
          "SP.POP.TOTL.FE.IN":  "Population, female",
          "SP.POP.TOTL.MA.IN":  "Population, male"
         }

baltic = {"EST":"estonia",
             "LVA":"latvia",
             "LTU":"lithuania"
            }

peqn = wbdata.get_dataframe(adults, country = baltic)
peqn['adult male pop'] = peqn['Population, male'] - (peqn['Population ages␣
 ↪0-14, male'] + (peqn['Population ages 15-19, male (% of male population)']/
 ↪100)*peqn['Population, male'])
peqn['adult female pop'] = peqn['Population, female'] - (peqn['Population ages␣
 ↪0-14, female'] + (peqn['Population ages 15-19, female (% of female␣
 ↪population)']/100)*peqn['Population, female'])
peqn['adult pop'] = peqn['adult male pop'] + peqn['adult female pop']

peqn['male mortality rate'] = peqn['male mortality']/1000
peqn['female mortality rate'] = peqn['female mortality']/1000
peqn['total mortality rate'] = (peqn['male mortality rate']*peqn['Population,␣
 ↪male'] + peqn['female mortality rate']*peqn['Population, female'])/
 ↪peqn['total pop']

peqn["% adult female"] = peqn["% adult female"]*peqn["% female"]/100
peqn['mompop'] = peqn['Population ages 15-64, female']-peqn['Population ages␣
 ↪60-64, female']
peqn['momshare'] = peqn['mompop']/peqn['total pop']

peqn.reset_index(inplace=True)
peqn['date'] = peqn['date'].astype(int)
peqn.set_index(['date', 'country'],inplace=True)
peqn = peqn.dropna(axis=0, how='any')

peqn = peqn.drop(columns = ['Population ages 0-14, male', 'Population ages␣
 ↪15-19, male (% of male population)', 'Population ages 0-14, female',␣
 ↪'Population ages 15-19, female (% of female population)', 'Population ages␣
 ↪15-64, female', 'Population ages 60-64, female'])
```

```
peqn_clean = peqn.drop(columns = ['male mortality', 'female mortality', '%␣
↪adult female', '% female', 'adult male pop', 'adult female pop', 'adult␣
↪pop', 'male mortality rate', 'female mortality rate'])
peqn_clean
```

[42]:

```
                total pop  pop growth rate         gdppc   tfr  \
date country
1995 Estonia    1436634.0        -1.785400   3134.389753  1.38
1996 Estonia    1415594.0        -1.475365   3380.926302  1.37
1997 Estonia    1399535.0        -1.140919   3682.952301  1.32
1998 Estonia    1386156.0        -0.960559   4093.392477  1.28
1999 Estonia    1390244.0         0.294482   4140.936602  1.30
…                   …              …            …    …
2015 Lithuania  2904910.0        -0.940754  14258.229335  1.70
2016 Lithuania  2868231.0        -1.270695  14998.125060  1.69
2017 Lithuania  2828403.0        -1.398322  16843.699655  1.63
2018 Lithuania  2801543.0        -0.954190  19176.812151  1.63
2019 Lithuania  2794137.0        -0.264704  19575.768481  1.61

                Population, male  Population, female  crude death rate  \
date country
1995 Estonia            664904.0            771730.0              14.5
1996 Estonia            655373.0            760221.0              13.4
1997 Estonia            649018.0            750517.0              13.3
1998 Estonia            644332.0            741824.0              14.0
1999 Estonia            647619.0            742625.0              13.4
…                          …                   …                 …
2015 Lithuania         1337721.0           1567189.0              14.4
2016 Lithuania         1322120.0           1546111.0              14.3
2017 Lithuania         1305326.0           1523077.0              14.2
2018 Lithuania         1294538.0           1507005.0              14.1
2019 Lithuania         1292438.0           1501699.0              13.7

                total mortality rate    mompop  momshare
date country
1995 Estonia                0.251556  447311.0  0.311360
1996 Estonia                0.216094  442208.0  0.312383
1997 Estonia                0.214128  437348.0  0.312495
1998 Estonia                0.224434  432662.0  0.312131
1999 Estonia                0.209563  433831.0  0.312054
…                              …         …        …
2015 Lithuania              0.158899  901910.0  0.310478
2016 Lithuania              0.155664  878890.0  0.306422
2017 Lithuania              0.140692  854187.0  0.302003
2018 Lithuania              0.136554  833138.0  0.297385
2019 Lithuania              0.132322  817016.0  0.292404
```

```
[71 rows x 10 columns]
```

[43]:
```
# Trying to replicate the equation for population in year t+1 given indicators␣
↪for year t

peqn_clean['surviving adults'] = (1 - (peqn_clean['crude death rate']/
↪1000))*peqn_clean['total pop']
peqn_clean['next yr predicted total pop'] = peqn_clean['surviving adults'] +␣
↪peqn_clean['mompop']*peqn_clean['tfr']

prediction = peqn_clean[['total pop', 'next yr predicted total pop']]
prediction = prediction.rename(columns = {'next yr predicted total pop': 'pred␣
↪total pop'})
pred_shift = prediction.groupby('country')['pred total pop'].shift()
pred_shift = pd.DataFrame(pred_shift)
prediction['pred total pop'] = pred_shift['pred total pop']


prediction['pop resid'] = prediction['pred total pop'] - prediction['total pop']
prediction = prediction.dropna(axis=0, how='any')
prediction
```

[43]:
```
                 total pop  pred total pop     pop resid
date country
1996 Estonia     1415594.0    2.033092e+06  6.174980e+05
1997 Estonia     1399535.0    2.002450e+06  6.029150e+05
1998 Estonia     1386156.0    1.958221e+06  5.720645e+05
1999 Estonia     1390244.0    1.920557e+06  5.303132e+05
2000 Estonia     1396985.0    1.935595e+06  5.386100e+05
…                      …             …             …
2015 Lithuania   2904910.0    4.392918e+06  1.488008e+06
2016 Lithuania   2868231.0    4.396326e+06  1.528095e+06
2017 Lithuania   2828403.0    4.312539e+06  1.484136e+06
2018 Lithuania   2801543.0    4.180564e+06  1.379021e+06
2019 Lithuania   2794137.0    4.120056e+06  1.325919e+06

[68 rows x 3 columns]
```

[44]:
```
import plotly.express as px
prediction.reset_index(inplace = True)

fig = px.line(prediction, x='date', y=['total pop', 'pred total pop'] ,␣
↪color='country')
fig.update_traces(textposition="bottom right")
fig.show()

#gap due to net migration (incomplete data), or did i fuck something up?
```

Is the residual between predicted total pop and actual total pop in year t+1 due to net migration (for which we have incomplete data) or did I mess something up? - Next step: compare residual to net migration for the three years in Estonia for which we have complete data

Looking at relationship between TB incidence, population, and income. Know income strongly associated with TB - when USSR collapsed, so did it's healthcare and welfare systems. High rates of poverty, homelessness, and drug use –> TB epidemic.

Wanted to use better indicators for poverty than gdppc, but we have most robust (and accurate) data on gdppc.

[45]:
```
ind = {"SH.TBS.INCD" : "Incidence of tuberculosis (per 100,000 people)",
        "NY.GDP.PCAP.CD":"GDP per capita",
        "SI.POV.LMIC": "Poverty headcount ratio at $3.20 a day (2011 PPP) (% of␣
 ↪population)",
       "SP.POP.TOTL":"Total population",
       "SP.POP.GROW":"Population Growth Rate",
       "SP.DYN.AMRT.FE":"Female Mortality"}

baltic = {"EST":"Estonia",
              "LVA":"Latvia",
              "LTU":"Lithuania"
         }

data = wbdata.get_dataframe(ind, country = baltic)

# Make years ints instead of strings
data.reset_index(inplace=True)
data['date'] = data['date'].astype(int)
data = data.dropna(axis=0, how='any')

data['Log GDP per capita'] = np.log(data['GDP per capita'])
data['Log TB incidence'] = np.log(data['Incidence of tuberculosis (per 100,000␣
 ↪people)'])
```

[46]:
```
import statsmodels.api as sm

#simple linear regression of log TB incidence on log gdppc, for all three␣
 ↪baltic states
y_3i = data['Log TB incidence']
X_3i = sm.add_constant(data[['Log GDP per capita']])
model_3i = sm.OLS(y_3i, X_3i)
results_3i = model_3i.fit(cov_type='HC1')
results_3i.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning:

In a future version of pandas all arguments of concat except for the argument
```

```
'objs' will be keyword-only
```

[46]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log TB incidence   R-squared:                       0.386
Model:                            OLS   Adj. R-squared:                  0.371
Method:                 Least Squares   F-statistic:                     28.93
Date:                Wed, 09 Feb 2022   Prob (F-statistic):           3.09e-06
Time:                        23:24:39   Log-Likelihood:                -16.367
No. Observations:                  44   AIC:                             36.73
Df Residuals:                      42   BIC:                             40.30
Df Model:                           1
Covariance Type:                  HC1
================================================================================
======
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const             12.6698      1.629      7.776      0.000       9.476
15.863
Log GDP per capita -0.9337     0.174     -5.378      0.000      -1.274
-0.593
==============================================================================
Omnibus:                        3.423   Durbin-Watson:                   0.321
Prob(Omnibus):                  0.181   Jarque-Bera (JB):                2.058
Skew:                          -0.295   Prob(JB):                        0.357
Kurtosis:                       2.121   Cond. No.                         307.
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

[47]:
```
#now, want to see how gdppc well predicts tb incidence in each individual␣
↪country, starting with estonia

est = data[data['country']=='Estonia']

y_3i = est['Log TB incidence']
X_3i = sm.add_constant(est[['Log GDP per capita']])
model_3i = sm.OLS(y_3i, X_3i)
results_3i = model_3i.fit(cov_type='HC1')
results_3i.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning:

In a future version of pandas all arguments of concat except for the argument
'objs' will be keyword-only

/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning:

kurtosistest only valid for n>=20 … continuing anyway, n=15
```

[47]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:      Log TB incidence   R-squared:                       0.634
Model:                           OLS   Adj. R-squared:                  0.606
Method:                Least Squares   F-statistic:                     43.80
Date:               Wed, 09 Feb 2022   Prob (F-statistic):           1.67e-05
Time:                       23:24:41   Log-Likelihood:                 1.4315
No. Observations:                 15   AIC:                             1.137
Df Residuals:                     13   BIC:                             2.553
Df Model:                          1
Covariance Type:                 HC1
================================================================================
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const                 12.5353      1.352      9.272      0.000       9.886
15.185
Log GDP per capita    -0.9590      0.145     -6.618      0.000      -1.243
-0.675
==============================================================================
Omnibus:                        0.836   Durbin-Watson:                   0.581
Prob(Omnibus):                  0.659   Jarque-Bera (JB):                0.374
Skew:                           0.378   Prob(JB):                        0.829
Kurtosis:                       2.840   Cond. No.                         310.
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

[48]:

```
lva = data[data['country']=='Latvia']
```

```python
y_3i = lva['Log TB incidence']
X_3i = sm.add_constant(lva[['Log GDP per capita']])
model_3i = sm.OLS(y_3i, X_3i)
results_3i = model_3i.fit(cov_type='HC1')
results_3i.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning:

In a future version of pandas all arguments of concat except for the argument
'objs' will be keyword-only

/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning:

kurtosistest only valid for n>=20 … continuing anyway, n=14
```

[48]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log TB incidence   R-squared:                       0.563
Model:                            OLS   Adj. R-squared:                  0.526
Method:                 Least Squares   F-statistic:                     29.51
Date:                Wed, 09 Feb 2022   Prob (F-statistic):           0.000152
Time:                        23:24:42   Log-Likelihood:                 5.2893
No. Observations:                  14   AIC:                            -6.579
Df Residuals:                      12   BIC:                            -5.300
Df Model:                           1
Covariance Type:                  HC1
================================================================================
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const                 10.4343      1.156      9.030      0.000       8.169
12.699
Log GDP per capita    -0.6894      0.127     -5.433      0.000      -0.938
-0.441
==============================================================================
Omnibus:                        0.083   Durbin-Watson:                   0.533
Prob(Omnibus):                  0.959   Jarque-Bera (JB):                0.106
Skew:                           0.085   Prob(JB):                        0.948
Kurtosis:                       2.608   Cond. No.                         330.
==============================================================================
```

```
Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

[49]:

```
ltu = data[data['country']=='Lithuania']

y_3i = ltu['Log TB incidence']
X_3i = sm.add_constant(ltu[['Log GDP per capita']])
model_3i = sm.OLS(y_3i, X_3i)
results_3i = model_3i.fit(cov_type='HC1')
results_3i.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning:

In a future version of pandas all arguments of concat except for the argument
'objs' will be keyword-only

/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning:

kurtosistest only valid for n>=20 … continuing anyway, n=15
```

[49]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log TB incidence   R-squared:                       0.462
Model:                            OLS   Adj. R-squared:                  0.420
Method:                 Least Squares   F-statistic:                     8.368
Date:                Wed, 09 Feb 2022   Prob (F-statistic):             0.0126
Time:                        23:24:43   Log-Likelihood:                 10.200
No. Observations:                  15   AIC:                            -16.40
Df Residuals:                      13   BIC:                            -14.98
Df Model:                           1
Covariance Type:                  HC1
================================================================================
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const                  7.9296      1.309      6.060      0.000       5.365
10.494
Log GDP per capita    -0.3999      0.138     -2.893      0.004      -0.671
-0.129
==============================================================================
```

```
Omnibus:                        0.340   Durbin-Watson:                   0.529
Prob(Omnibus):                  0.844   Jarque-Bera (JB):                0.465
Skew:                           0.256   Prob(JB):                        0.792
Kurtosis:                       2.306   Cond. No.                         319.
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

[50]:
```
#by incorporating binary indicators for country to our model, we can compare␣
 ↪the relationship in each country
#drop Lithuania from model --> use as baseline against which we compare Estonia␣
 ↪and Latvia

bdummies = pd.get_dummies(data['country'], prefix='', prefix_sep='')
data_wdummies = data[['country', 'date', 'Log TB incidence', 'Log GDP per␣
 ↪capita', 'Total population']].join(bdummies)

y_3g = data_wdummies['Log TB incidence']
X_3g = sm.add_constant(data_wdummies.drop(columns=['Log TB incidence',␣
 ↪'Lithuania', 'country', 'date', 'Total population']))
model_3g= sm.OLS(y_3g, X_3g)
results_3g = model_3g.fit(cov_type='HC1')
results_3g.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning:

In a future version of pandas all arguments of concat except for the argument
'objs' will be keyword-only
```

[50]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:       Log TB incidence   R-squared:                       0.826
Model:                            OLS   Adj. R-squared:                  0.813
Method:                 Least Squares   F-statistic:                     41.07
Date:                Wed, 09 Feb 2022   Prob (F-statistic):           2.75e-12
Time:                        23:24:44   Log-Likelihood:                 11.363
No. Observations:                  44   AIC:                            -14.73
Df Residuals:                      40   BIC:                            -7.590
Df Model:                           3
Covariance Type:                  HC1
================================================================================
```

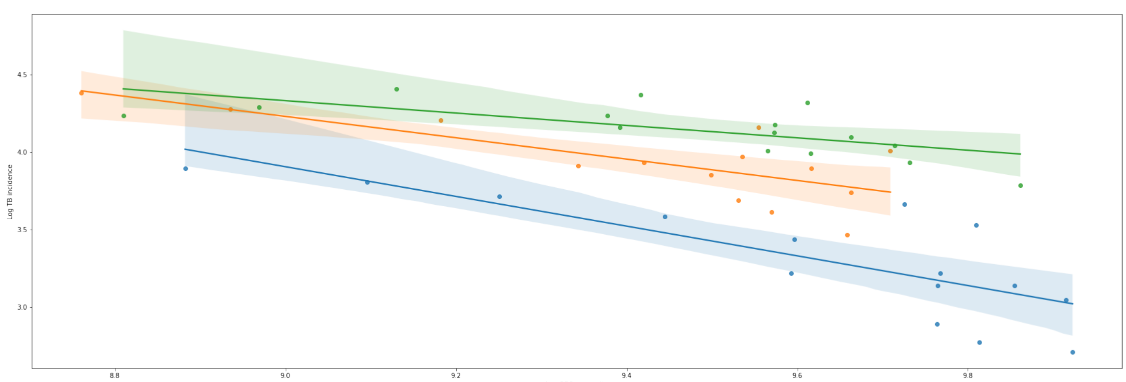
```
======
                         coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const                 10.7197      1.001     10.713      0.000       8.758
12.681
Log GDP per capita    -0.6946      0.104     -6.695      0.000      -0.898
-0.491
Estonia               -0.7262      0.068    -10.683      0.000      -0.859
-0.593
Latvia                -0.2370      0.064     -3.722      0.000      -0.362
-0.112
==============================================================================
Omnibus:                        0.642   Durbin-Watson:                   0.490
Prob(Omnibus):                  0.726   Jarque-Bera (JB):                0.283
Skew:                          -0.194   Prob(JB):                        0.868
Kurtosis:                       3.066   Cond. No.                         318.
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

[51]:

```python
#now we're just plotting some regression lines to visualize the relationships␣
↪in question

import seaborn as sns

sns.regplot(x='Log GDP per capita', y='Log TB incidence', data=est)
sns.regplot(x='Log GDP per capita', y='Log TB incidence', data=lva)
sns.regplot(x='Log GDP per capita', y='Log TB incidence', data=ltu)

#not sure how to add legend but i'll figure that out!
```

[51]:

```
<AxesSubplot:xlabel='Log GDP per capita', ylabel='Log TB incidence'>
```



[52]:
```
est.reset_index(inplace=True)
est.set_index(['date'],inplace=True)

est.iplot(kind='scatter', mode='markers', symbol='circle-dot', bestfit=True,
         x='Log GDP per capita',y='Log TB incidence',
         text=est.reset_index('date')['date'].values.tolist(),
         xTitle='Log GDP per capita',yTitle='Log TB incidence',
         title='Estonia')

lva.reset_index(inplace=True)
lva.set_index(['date'],inplace=True)

lva.iplot(kind='scatter', mode='markers', symbol='circle-dot', bestfit=True,
         x='Log GDP per capita',y='Log TB incidence',
         text=lva.reset_index('date')['date'].values.tolist(),
         xTitle='Log GDP per capita',yTitle='Log TB incidence',
         title='Latvia')


ltu.reset_index(inplace=True)
ltu.set_index(['date'],inplace=True)

ltu.iplot(kind='scatter', mode='markers', symbol='circle-dot', bestfit=True,
         x="Log GDP per capita",y='Log TB incidence',
         text=ltu.reset_index('date')['date'].values.tolist(),
         xTitle='Log GDP per capita',yTitle='Log TB incidence',
         title='Lithuania')
```

[53]:
```
data['Log Total Pop'] = np.log(data['Total population'])
est = data[data['country']=='Estonia']
lva = data[data['country']=='Latvia']
ltu = data[data['country']=='Lithuania']

est[['Log Total Pop', 'Log TB incidence', 'country', 'date']].iplot(
    x='date', y='Log Total Pop', mode='lines+markers', secondary_y = 'Log TB␣
 ↪incidence',
    secondary_y_title='Log TB incidence', xTitle='date', yTitle='Log Total Pop',
    text='country', title='Log total population and log TB incidence over time,␣
 ↪Estonia')

lva[['Log Total Pop', 'Log TB incidence', 'country', 'date']].iplot(
    x='date', y='Log Total Pop', mode='lines+markers', secondary_y = 'Log TB␣
 ↪incidence',
    secondary_y_title='Log TB incidence', xTitle='date', yTitle='Log Total Pop',
```

```
    text='country', title='Log total population and log TB incidence over time,␣
↪Latvia')

ltu[['Log Total Pop', 'Log TB incidence', 'country', 'date']].iplot(
    x='date', y='Log Total Pop', mode='lines+markers', secondary_y = 'Log TB␣
↪incidence',
    secondary_y_title='Log TB incidence', xTitle='date', yTitle='Log Total Pop',
    text='country', title='Log total population and log TB incidence over time,␣
↪Lithuania')
```

[ ]:

[ ]:

[ ]:

[ ]:

2 Ignore the Rest

[74]:
```
print(lithuania.query("date == 2008").sum(axis=0))
```

```
Male ages 00-04                     74531.0
Male ages 05-09                     77085.0
Male ages 10-14                    101177.0
Male ages 15-19                    121772.0
Male ages 20-24                    117971.0
Male ages 25-29                    105428.0
Male ages 30-34                    103289.0
Male ages 35-39                    109503.0
Male ages 40-44                    115974.0
Male ages 45-49                    119999.0
Male ages 50-54                    101240.0
Male ages 55-59                     79587.0
Male ages 60-64                     67884.0
Male ages 65-69                     62887.0
Male ages 70-74                     53403.0
Male ages 75-79                     36470.0
Male ages 80-UP                     28579.0
Female ages 00-04                   71248.0
Female ages 05-09                   72897.0
Female ages 10-14                   96098.0
Female ages 15-19                  117427.0
Female ages 20-24                  114910.0
Female ages 25-29                  103994.0
Female ages 30-34                  106152.0
```

```
Female ages 35-39                   114585.0
Female ages 40-44                   123256.0
Female ages 45-49                   132801.0
Female ages 50-54                   117014.0
Female ages 55-59                    99161.0
Female ages 60-64                    93649.0
Female ages 65-69                    96319.0
Female ages 70-74                    93576.0
Female ages 75-79                    78455.0
Female ages 80-UP                    89910.0
Total Population in Given Range    3198231.0
dtype: float64
```

[60]:

```python
import numpy as np

variable_new1 = {"SP.DYN.CBRT.IN":"Birth rate"
                 }

# Three letter codes come from wbdata.get_country()
countries_new1 = {
             "ESP":"Spain",
                      "AUT":"Austria",
                      "EST":"Estonia",
                      "FIN":"Finland",
                      "GRC": "Greece",
                      "LVA": "Latvia",
                      "LTU": "Lithuania",
                      "MLT": "Malta",
                      "POL": "Poland",
                      "PRT": "Portugal",
                      "SVK": "Slovakia",
                      "SVN":"Slovenia",
                    "USA":"US",
                    "CHN":"China"

            }

new = wbdata.get_dataframe(variable_new1, country = countries_new1).squeeze()

new = new.unstack('country')
# Date index is of type string; change to integers
new.index = new.index.astype(int)

# Differences (over time) in logs give us growth rates
new.iplot(title="EU Birth Rate",
                        yTitle="Rate",xTitle='Year')
```

```
[ ]:
```