First Dataset

February 21, 2021

[1]:
```
import numpy as np
import pandas as pd
import pickle
from statsmodels import tsa
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels import multivariate
from statsmodels import regression
import scipy.stats as stats
from statsmodels.sandbox.regression import gmm
from statsmodels.sandbox.regression.gmm import GMM
import statsmodels.stats.diagnostic as smd
from statsmodels.tsa.adfvalues import mackinnonp, mackinnoncrit
from statsmodels.tsa.stattools import adfuller
import statsmodels.tsa.api as smt
from statsmodels.tsa.vector_ar.hypothesis_test_results import␣
↪CausalityTestResults
from statsmodels.tsa.vector_ar.var_model import VAR, VARProcess, VARResults
from statsmodels.tsa.vector_ar.vecm import VECM, coint_johansen, select_order
import statsmodels.tsa.arima_model as am
from statsmodels.regression.rolling import RollingOLS

from tabulate import tabulate

import datetime as dt
from dateutil.relativedelta import relativedelta
from datetime import timedelta


import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, MinuteLocator
from matplotlib.ticker import PercentFormatter


import os
import warnings
```

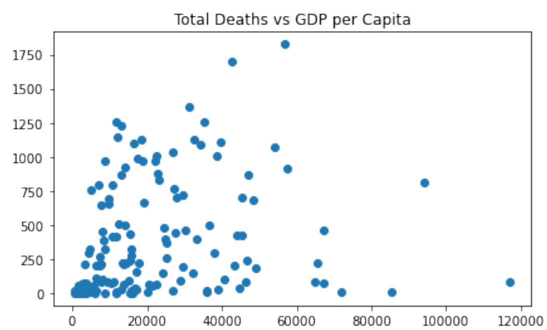
```
from scipy.optimize import minimize, brute
from arch import arch_model


warnings.filterwarnings("ignore")
```

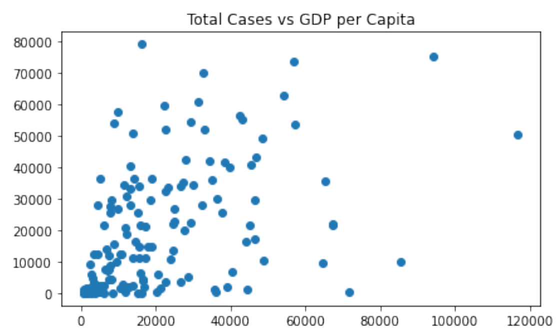
[2]:
```
covid_data = pd.read_csv('owid-covid-data.csv')
```

[3]:
```
data_death = covid_data.dropna(subset =␣
 ↪['total_deaths_per_million','gdp_per_capita'])
countries = data_death.location.unique()
deaths = []
cases = []
gdp = []
tests = []
stringency = []
density = []
population = []
poverty = []
for c in countries:
    country_data = data_death[data_death.location == c]
    deaths.append(country_data.total_deaths_per_million.iloc[-1])
    gdp.append(country_data.gdp_per_capita.mean())
    cases.append(country_data.total_cases_per_million.iloc[-1])
    tests.append(country_data.total_tests_per_thousand.iloc[-1])
    stringency.append(country_data.stringency_index.mean())
    density.append(country_data.population_density.mean())
    population.append(country_data.population.mean())
    poverty.append(country_data.extreme_poverty.mean())
ans = pd.
 ↪DataFrame([countries,deaths,gdp,cases,tests,stringency,density,population,poverty]).
 ↪T
ans.columns = ['location','total_deaths_per_million','gdp',
              ␣
 ↪'total_cases_per_million','total_tests_per_thousand','stringency_index','density','populati
              'poverty']
fig,axs = plt.subplots(1, 2,figsize = (15,4))
axs[0].plot(ans['gdp'],ans['total_deaths_per_million'],'o',)
axs[0].set_title('Total Deaths vs GDP per Capita')
axs[1].plot(ans['gdp'],ans['total_cases_per_million'],'o')
axs[1].set_title('Total Cases vs GDP per Capita')
display(ans.sort_values(by = 'total_deaths_per_million')[:10])
```

```
             location total_deaths_per_million      gdp  \
25            Burundi                    0.168  702.225
49            Eritrea                    0.282  1510.46
```

```
102          Mongolia                    0.305  11840.8
150          Tanzania                    0.352   2683.3
164           Vietnam                     0.36  6171.88
151          Thailand                    0.931  16277.7
120  Papua New Guinea                    1.006  3823.19
53               Fiji                    2.231  8702.98
32              China                    3.326  15308.7
17              Benin                    3.629  2064.24

    total_cases_per_million total_tests_per_thousand stringency_index  \
25                   70.811                      NaN          13.8689
49                  353.031                      NaN              NaN
102                 398.988                      NaN              NaN
150                   8.521                      NaN          28.6787
164                  15.379                      NaN          61.6633
151                 128.453                      NaN          54.6951
120                  89.303                      NaN          44.4667
53                    54.66                      NaN          50.1615
32                   66.869                      NaN          71.9987
17                  268.164                      NaN           47.207

     density   population poverty
25   423.062  1.18908e+07    71.7
49    44.304  3.54643e+06     NaN
102     1.98  3.27829e+06     0.5
150   64.699  5.97342e+07    49.1
164  308.127  9.73386e+07       2
151  135.132      6.98e+07     0.1
120    18.22  8.94703e+06     NaN
53    49.562       896444     1.4
32   147.674  1.43932e+09     0.7
17     99.11  1.21232e+07    49.6
```





0.1 Why we picked Europe

[4]:
```
import plotly.graph_objs as go
import plotly as py
import plotly.io as pio
pio.renderers.default = 'iframe'
```

[5]:
```
data = dict (
    type = 'choropleth',
    locations = ans['location'],
    locationmode='country names',
    z=ans['total_deaths_per_million'],
    colorbar_title = "Total Deaths Per Million",colorscale = 'Reds')


fig = go.Figure(data=[data])
fig.update_layout(title=go.layout.Title(text="Total Deaths Per Million Map",
                                        font=go.layout.title.Font(size=15)))
fig.show()
```

[6]:
```
data = dict (
    type = 'choropleth',
    locations = ans['location'],
    locationmode='country names',
    z=ans['total_cases_per_million'],
    colorbar_title = "Total Cases Per Million",colorscale = 'Reds')


fig = go.Figure(data=[data])
fig.update_layout(title=go.layout.Title(text="Total Cases Per Million Map",
                                        font=go.layout.title.Font(size=15))
                 )
fig.show()
```

[7]:
```
density = ans[['location','density']]

data = dict (
    type = 'choropleth',
    locations = ans['location'],
    locationmode='country names',
    z=ans['gdp'],
    colorbar_title = "GDP per Capita",colorscale = 'Reds')


fig = go.Figure(data=[data])
fig.update_layout(title=go.layout.Title(text="GDP per Capita Map",
                                        font=go.layout.title.Font(size=15))
```

```
                 )
fig.show()
```

[8]:
```
data = dict (
    type = 'choropleth',
    locations = ans['location'],
    locationmode='country names',
    z=ans['stringency_index'],
    colorbar_title = "Average Stringency Index",colorscale = 'Reds')


fig = go.Figure(data=[data])
fig.update_layout(title=go.layout.Title(text="Stringency Index Map",
                                        font=go.layout.title.Font(size=15))
                 )
fig.show()
```

0.2 Europe

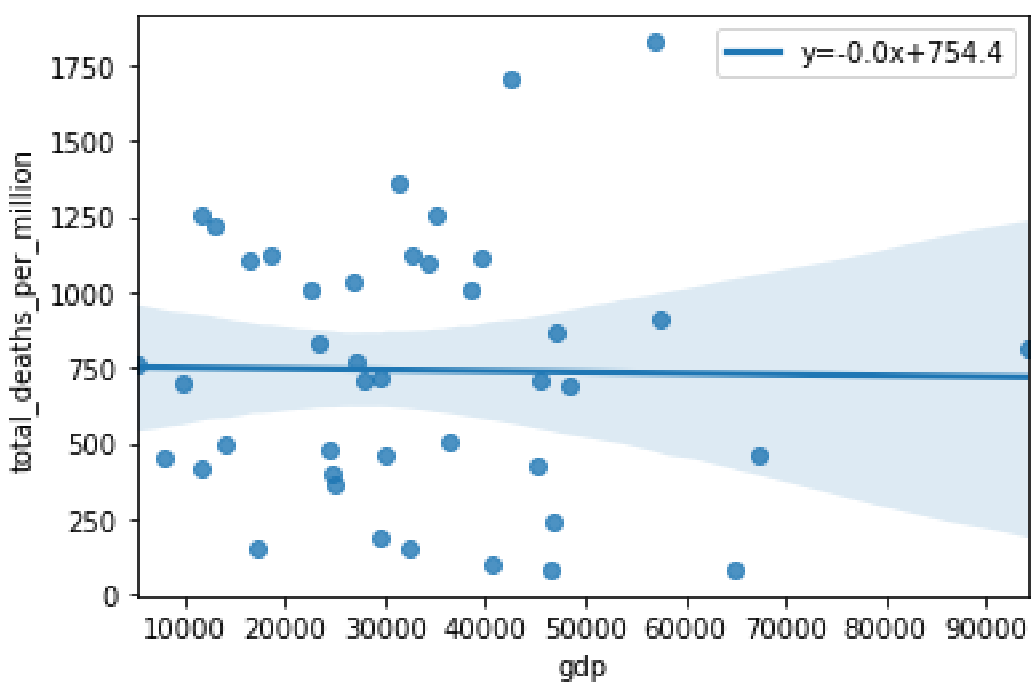
[9]:
```
eu = covid_data[covid_data.continent == 'Europe']
```

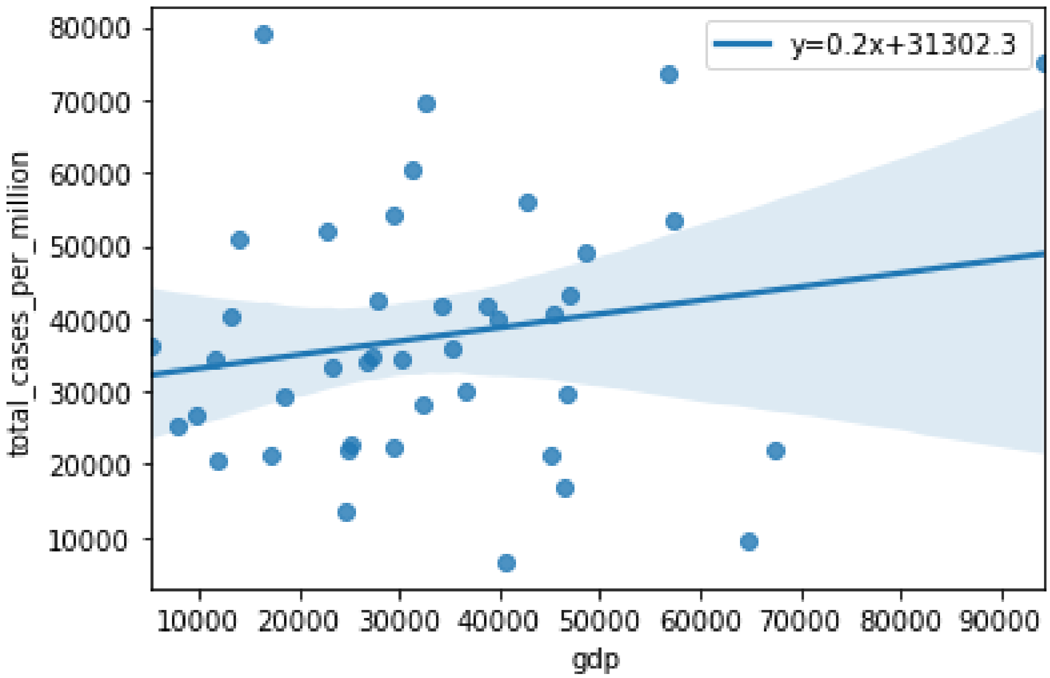
[10]:
```
data_death = eu.dropna(subset = ['total_deaths_per_million','gdp_per_capita'])
countries = data_death.location.unique()
deaths = []
cases = []
gdp = []
tests = []
stringency = []
density = []
population = []
poverty = []
for c in countries:
    country_data = data_death[data_death.location == c]
    deaths.append(country_data.total_deaths_per_million.iloc[-1])
    gdp.append(country_data.gdp_per_capita.mean())
    cases.append(country_data.total_cases_per_million.iloc[-1])
    tests.append(country_data.total_tests_per_thousand.iloc[-1])
    stringency.append(country_data.stringency_index.mean())
    density.append(country_data.population_density.mean())
    population.append(country_data.population.mean())
    poverty.append(country_data.extreme_poverty.mean())
ans_eu = pd.
 ↪DataFrame([countries,deaths,gdp,cases,tests,stringency,density,population,poverty]).
 ↪T
ans_eu.columns = ['location','total_deaths_per_million','gdp',
```

```
                ↪'total_cases_per_million','total_tests_per_thousand','stringency_index','density','populati
                'poverty']
```

[11]:
```
ans_eu.iloc[:,1:] = ans_eu.iloc[:,1:].astype('float')
```

[12]:
```
slope, intercept, r_value, p_value, std_err = stats.linregress(ans_eu["gdp"].
 ↪astype('float'),ans_eu["total_deaths_per_million"].astype('float'))

sns.regplot(x=ans_eu["gdp"].astype('float'),
 ↪y=ans_eu["total_deaths_per_million"].astype('float'),
              line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
plt.legend()

plt.show()

slope, intercept, r_value, p_value, std_err = stats.linregress(ans_eu["gdp"].
 ↪astype('float'),

 ↪y=ans_eu["total_cases_per_million"].astype('float'))
sns.regplot(x=ans_eu["gdp"].astype('float'),
 ↪y=ans_eu["total_cases_per_million"].astype('float'),
              line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
plt.legend()

plt.show()
```





0.3 Days between first 100 cases and 1000 cases per million in EU

[30]:
```
countries = eu.location.unique()
iso_code = eu.iso_code.unique()
c = countries[0]
country_data = eu[eu.location== c].reset_index()
```

[32]:
```
day_of_100 = []
day_of_1000 = []
human_index = []
policy_stringency_100 = []
policy_stringency_1000 = []
pop_density = []
icu_per_mil = []
hosp_per_mil = []
for c in countries:
    country_data = eu[eu.location== c].reset_index()
    country_data['day_of_100'] = 0
    country_data['day_of_1000'] = 0
    try:
```

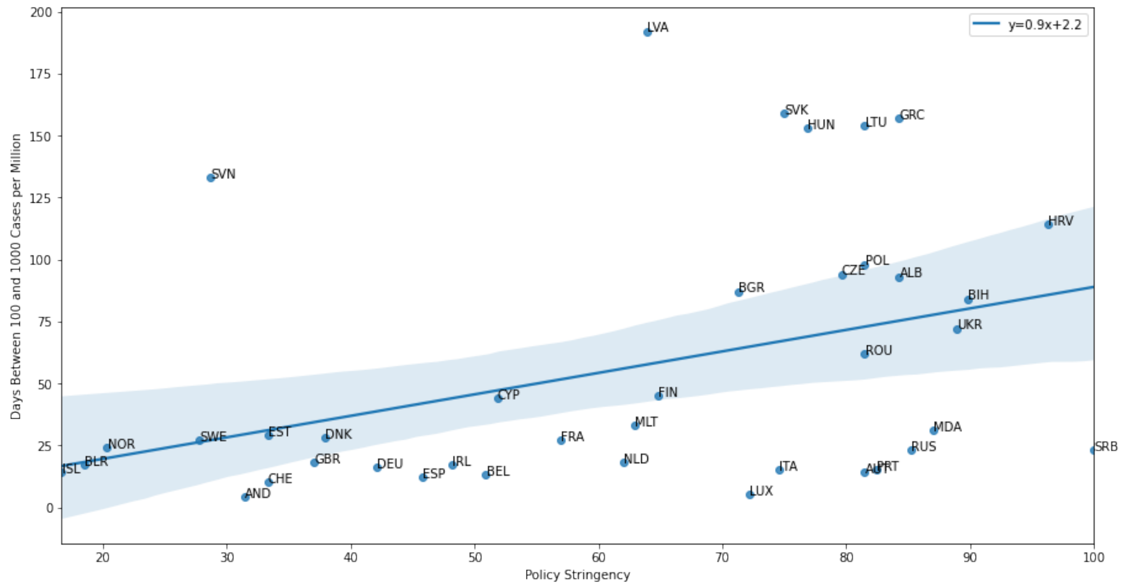
```
        day_100 = country_data[country_data.total_cases_per_million > 100].
↪first_valid_index()
        day_1000 = country_data[country_data.total_cases_per_million > 1000].
↪first_valid_index()
        day_of_100.append(day_100)
        day_of_1000.append(day_1000)
    except:
        day_of_100.append(None)
        day_of_1000.append(None)

    try:
        human_index.append(country_data.human_development_index.mean())
        policy_stringency_100.append(country_data.stringency_index.loc[day_100])
        policy_stringency_1000.append(country_data.stringency_index.
↪loc[day_1000])
        density.append(country_data.population_density.mean())
        icu_per_mil.append(None)
        hosp_per_mil.append(None)
    except:
        human_index.append(None)
        policy_stringency_100.append(None)
        policy_stringency_1000.append(None)
        density.append(None)
```

[33]:
```
ans = pd.
↪DataFrame([countries,day_of_100,day_of_1000,human_index,policy_stringency_100,
                  policy_stringency_1000,density,iso_code]).T
ans.columns =␣
↪['location','day_100','day_1000','human_index','policy_100','policy_1000','density','iso_co

ans['days_between'] = ans['day_1000'] - ans['day_100']
ans = ans.dropna()
```

[34]:
```
def label_point(x, y, val, ax):
    a = pd.concat({'x': x, 'y': y, 'val': val}, axis=1)
    for i, point in a.iterrows():
        ax.text(point['x']+.02, point['y'], str(point['val']))
```

[48]:
```
plt.figure(figsize = (15,8))
x = ans['policy_100'].astype('float')
y = ans['days_between'].astype('float')
val = ans.iso_code


slope, intercept, r_value, p_value, std_err = stats.
↪linregress(ans['policy_100'].astype('float'),
```

```
↪y=ans['days_between'].astype('float'))


ax = sns.regplot(x=ans['policy_100'].astype('float'), y=ans['days_between'].
↪astype('float'),
            line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
plt.legend()
label_point(x, y, val, plt.gca())
plt.xlabel("Policy Stringency")
plt.ylabel("Days Between 100 and 1000 Cases per Million")
plt.show()
```



[37]:
```
Y = ans['days_between'].astype('float')
X = ans['policy_100'].astype('float')
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.summary()
```

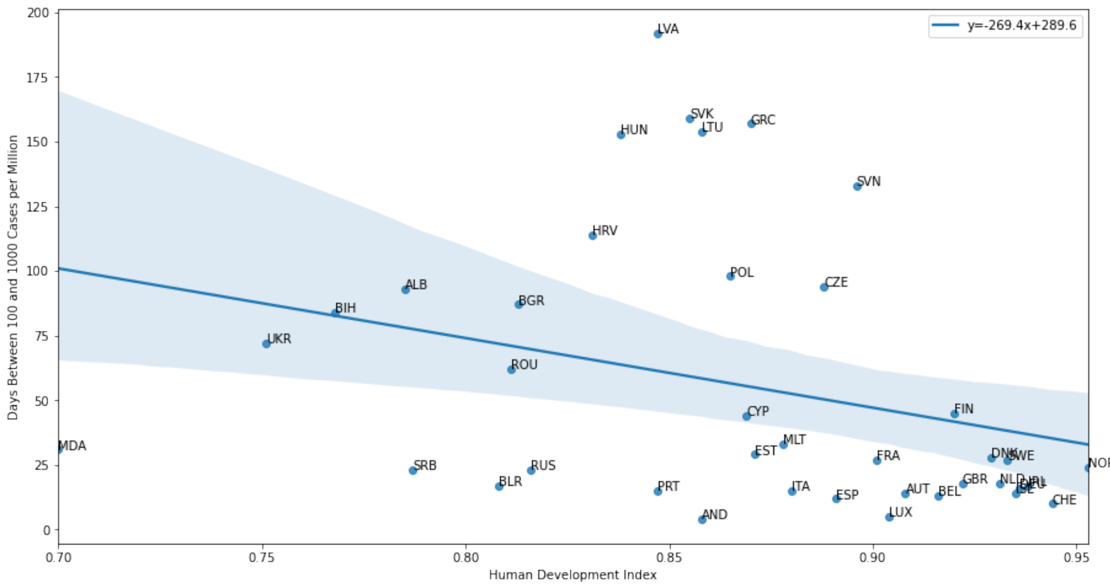
[37]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:           days_between   R-squared:                       0.154
Model:                            OLS   Adj. R-squared:                  0.132
Method:                 Least Squares   F-statistic:                     6.758
```

```
Date:                Sat, 20 Feb 2021   Prob (F-statistic):           0.0133
Time:                        18:48:13   Log-Likelihood:              -206.57
No. Observations:                  39   AIC:                           417.1
Df Residuals:                      37   BIC:                           420.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.1805     22.083      0.099      0.922     -42.564      46.925
policy_100     0.8674      0.334      2.600      0.013       0.191       1.543
==============================================================================
Omnibus:                        8.474   Durbin-Watson:                   1.643
Prob(Omnibus):                  0.014   Jarque-Bera (JB):                7.596
Skew:                           1.051   Prob(JB):                       0.0224
Kurtosis:                       3.504   Cond. No.                         184.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[49]:
```
ans.reset_index(inplace = True)
```

[52]:
```
plt.figure(figsize = (15,8))
x1 = ans['human_index'].astype('float')
y1 = ans['days_between'].astype('float')
val1 = ans.iso_code


slope, intercept, r_value, p_value, std_err = stats.
 ↪linregress(ans['human_index'].astype('float'),
                                                                        ␣

 ↪y=ans['days_between'].astype('float'))

ax = sns.regplot(x=ans['human_index'].astype('float'), y=ans['days_between'].
 ↪astype('float'),
            line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
plt.legend()
#label_point(x1, y1, val1, plt.gca())

for i, txt in enumerate(val1):
    #print(i)
    ax.annotate(txt, (x1[i], y1[i]))
plt.xlabel("Human Development Index")
plt.ylabel("Days Between 100 and 1000 Cases per Million")
```

```
plt.show()
```



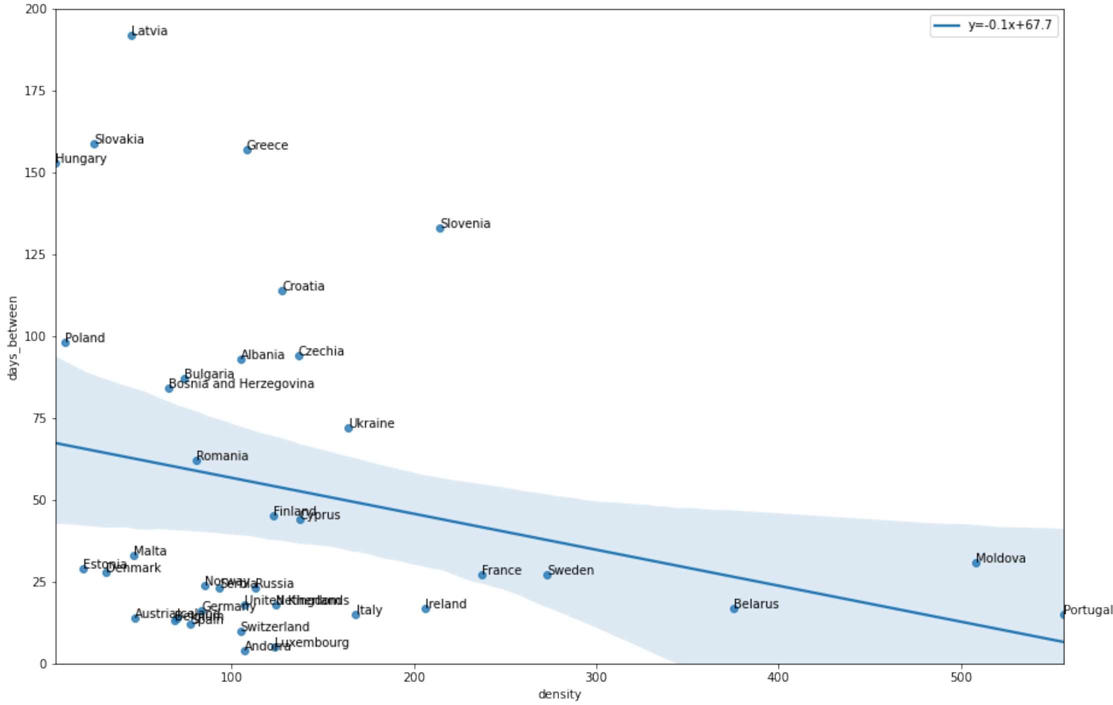
[21]:
```
Y = ans['days_between'].astype('float')
X = ans['human_index'].astype('float')
X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()
results.summary()
```

[21]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:           days_between   R-squared:                       0.091
Model:                            OLS   Adj. R-squared:                  0.066
Method:                 Least Squares   F-statistic:                     3.682
Date:                Sat, 20 Feb 2021   Prob (F-statistic):             0.0627
Time:                        18:45:27   Log-Likelihood:                -207.99
No. Observations:                  39   AIC:                             420.0
Df Residuals:                      37   BIC:                             423.3
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const         289.5727    122.132      2.371      0.023      42.110     537.035
human_index  -269.3963    140.388     -1.919      0.063    -553.850      15.058
```

```
==============================================================================
Omnibus:                        8.207   Durbin-Watson:                   1.585
Prob(Omnibus):                  0.017   Jarque-Bera (JB):                7.822
Skew:                           1.094   Prob(JB):                       0.0200
Kurtosis:                       3.169   Cond. No.                         29.9
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[22]:

```
ans_ = ans.sort_values(by = 'density')[:-1]#removing outlier
plt.figure(figsize = (15,10))
x1 = ans_['density'].astype('float')
y1 = ans_['days_between'].astype('float')
val1 = ans_.location


slope, intercept, r_value, p_value, std_err = stats.linregress(ans_['density'].
 ↪astype('float'),
                                                                              ␣
 ↪y=ans_['days_between'].astype('float'))

sns.regplot(x=ans_['density'].astype('float'), y=ans_['days_between'].
 ↪astype('float'),
            line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
plt.legend()
label_point(x1, y1, val1, plt.gca())
plt.ylim(0,200)

plt.show()
```


y=-0.1x+67.7
days_between
density
Latvia
Slovakia
Greece
Hungary
Slovenia
Croatia
Poland
Albania
Czechia
Bulgaria
Bosnia and Herzegovina
Ukraine
Romania
Finland
Cyprus
Malta
Estonia
Denmark
Moldova
France
Sweden
Russia
Ireland
Belarus
Italy
Portugal
Switzerland
Luxembourg
Andorra

[ ]: