```
from google.colab import files
import io
import os
from tqdm import tqdm
import csv

import statsmodels.api as sm
import statsmodels.formula.api as smf
import math
from sklearn.linear_model import LinearRegression

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
from scipy import linalg as LA
import operator
from xlwt import Workbook
import xlwt

import plotly.express as px

import plotly.graph_objects as go
from plotly.subplots import make_subplots

zzz=np.random.randint(1,100)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
root_path = 'drive/My Drive/ECI Research/Spring 2020/Data/'
```

```
Mounted at /content/drive/
```

Import Data

```
root_path = 'drive/My Drive/ECI Research/Spring 2020/Data/'

# Read input
#uploaded = files.upload()
df = pd.read_csv(root_path+'2013-2018 cleaned/2013clean.csv')
df1_cl= pd.read_csv(root_path+'2013-2018 cleaned/2013clean.csv', index_col=0)
#uploaded = files.upload()
```

```
df2 = pd.read_csv(root_path+'2013-2018 cleaned/2018clean.csv')
df2_cl= pd.read_csv(root_path+'2013-2018 cleaned/2018clean.csv', index_col=0)

df3 = pd.read_csv(root_path+'2013-2018 cleaned/2014clean.csv')
df3_cl= pd.read_csv(root_path+'2013-2018 cleaned/2014clean.csv', index_col=0)

df4 = pd.read_csv(root_path+'2013-2018 cleaned/2015clean.csv')
df4_cl= pd.read_csv(root_path+'2013-2018 cleaned/2015clean.csv', index_col=0)

df5 = pd.read_csv(root_path+'2013-2018 cleaned/2016clean.csv')
df5_cl= pd.read_csv(root_path+'2013-2018 cleaned/2016clean.csv', index_col=0)

df6 = pd.read_csv(root_path+'2013-2018 cleaned/2017clean.csv')
df6_cl= pd.read_csv(root_path+'2013-2018 cleaned/2017clean.csv', index_col=0)

df7 = pd.read_csv(root_path+'GDP4.csv')
df7_cl= pd.read_csv(root_path+'GDP4.csv', index_col=0)
df7 = df7.drop(columns = ['Country Code', 'Indicator Name'])
df7_cl = df7_cl.drop(columns = ['Country Code', 'Indicator Name'])
```

```
def df_to_matrix(dfa):
  raw_matrixa = dfa.to_numpy()
  matrixa = [list(dfa.columns)]

  for row in raw_matrixa:
    if np.min(row[1::]) > 0:
      matrixa.append(list(row))

  return matrixa
```

```
matrix = df_to_matrix(df)
matrix2 = df_to_matrix(df2)
matrix3 = df_to_matrix(df3)
matrix4 = df_to_matrix(df4)
matrix5 = df_to_matrix(df5)
matrix6 = df_to_matrix(df6)
matrix7 = df_to_matrix(df7)
```

ECI & PCI

MCP

```
#compute mcp
```

```
def compute_mcp(matrix):
  product_list = matrix[0][1::]
  country_list = []
  for i in range(1,len(matrix)):
    country_list.append(matrix[i][0])

  country_sum = []
  for i in range(1,len(matrix)):
    country_sum.append(np.sum(matrix[i][1::]))

  product_sum = []
  for i in range(1,len(matrix[0])):
    product = 0
    for j in range(1,len(matrix)):
        product += matrix[j][i]
    product_sum.append(product)
  total_sum = np.sum(product_sum)

  mcp = []

  for i in range (1,len(matrix)):
    country_row = []
    for j in range(1,len(matrix[i])):
      try:
        percentage = matrix[i][j]/country_sum[i-1]
      except:
        print(country_list[i-1])
      total_percentage = product_sum[j-1]/total_sum
      if percentage > total_percentage:
        country_row.append(1)
      else:
        country_row.append(0)
    mcp.append(country_row)

  return country_list,product_list,mcp,product_sum,country_sum
```

MCC

```
#compute mcc
def compute_mcc(mcp):
  mcp2 = mcp
  country_sum = []
  for i in range(0,len(mcp)):
    country_sum.append(np.sum(mcp[i]))

  product_sum = []
  for i in range(0,len(mcp[0])):
    product = 0
    for j in range(0,len(mcp)):
      product += mcp[j][i]
```

```
    product_sum.append(product)
  total_sum = np.sum(product_sum)

  mcc = []
  for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp)):
      temp_sum = 0
      for k in range(0,len(mcp[0])):
        try:
          value = (mcp[i][k] * mcp2[j][k]/(country_sum[i]*product_sum[k]))
          if np.isinf(value) or np.isnan(value):
            temp_sum += 0
          else:
            temp_sum += value
        except:
          pass
      country_row.append(temp_sum)
    mcc.append(country_row)
  return mcc
```

MPP

```
# compute mpp
def compute_mpp(mcp):
  mcp2 = mcp
  country_sum = []
  for i in range(0,len(mcp)):
    country_sum.append(np.sum(mcp[i]))

  product_sum = []
  for i in range(0,len(mcp[0])):
    product = 0
    for j in range(0,len(mcp)):
      product += mcp[j][i]
    product_sum.append(product)
  total_sum = np.sum(product_sum)

  mpp = []
  for i in range(0,len(mcp[0])):
    product_row = []
    for j in range(0,len(mcp[0])):
      temp_sum = 0
      for k in range(0,len(mcp)):
        try:
          temp_sum += (mcp[k][i] * mcp2[k][j]/(country_sum[k]*product_sum[i]))
        except:
          pass
      product_row.append(temp_sum)
    mpp.append(product_row)
```

```
  return mpp
```

ECI

```
#compute ECI
def compute_eci(mcc):
  mcc = np.array(mcc)

  e_vals, e_vecs = LA.eig(mcc)
  eigenvectors = e_vecs.tolist()
  eigen = []
  for eigenvector in eigenvectors:
    temp = []
    for col in eigenvector:
      temp.append(col.real)
    eigen.append(temp)

  vector = []
  for i in range(0,len(eigen)):
    vector.append(eigen[i][1])

  eci = []
  if vector[0] < 0:
    for item in vector:
      eci.append((item-np.mean(vector))/np.std(vector))
  else:
    for item in vector:
      eci.append((-item+np.mean(vector))/np.std(vector))
  return eci
```

PCI

```
#compute PCI
def compute_pci(mpp):
  mpp = np.array(mpp)
  e_vals, e_vecs = LA.eig(mpp)
  eigenvectors = e_vecs.tolist()
  eigen = []
  for eigenvector in eigenvectors:
    temp = []
    for col in eigenvector:
      temp.append(col.real)
    eigen.append(temp)

  vector = []
  for i in range(0,len(eigen)):
    vector.append(eigen[i][1])

  pci = []
```

```
  pci = []
  if vector[len(vector)-2] > 0:
    for item in vector:
      pci.append((item-np.mean(vector))/np.std(vector))
  else:
    for item in vector:
      pci.append((-item+np.mean(vector))/np.std(vector))
  return pci
```

```
country_list,product_list,mcp,product_sum,country_sum = compute_mcp(matrix)
country_list_18,product_list_18,mcp_18,product_sum_18,country_sum_18 = compute_mcp(matrix2)

mcc = compute_mcc(mcp)
mpp = compute_mpp(mcp)

mcc_18 = compute_mcc(mcp_18)
mpp_18 = compute_mpp(mcp_18)


eci = compute_eci(mcc)
pci = compute_pci(mpp)

eci_18 = compute_eci(mcc_18)
pci_18 = compute_pci(mpp_18)

#check ECI
eci_dict = {}
for i in range(len(country_list)):
  eci_dict[country_list[i]]=eci[i]
sorted_eci_dict = sorted(eci_dict.items(), key=operator.itemgetter(1))

#check PCI
pci_dict = {}
for i in range(len(product_list)):
  pci_dict[product_list[i]]=pci[i]
sorted_pci_dict = sorted(pci_dict.items(), key=operator.itemgetter(1))

#check ECI
eci_dict_18 = {}
for i in range(len(country_list_18)):
  eci_dict_18[country_list_18[i]]=eci_18[i]
sorted_eci_dict_18 = sorted(eci_dict_18.items(), key=operator.itemgetter(1))

print(sorted_eci_dict)
#for item in sorted_pci_dict:
 # print(item)
```

```
[('Algeria', -2.1904871551096545), ('Kuwait', -2.1904871551096545), ('Venezuela', -2.190
```

Section 2 Graphs (ECI and GDP)

```
def get_eci_dict(mat):
  country_list,product_list,mcp,product_sum,country_sum = compute_mcp(mat)
  mcc = compute_mcc(mcp)
  eci = compute_eci(mcc)
  eci_dict = {}
  for i in range(len(country_list)):
    eci_dict[country_list[i]]=eci[i]
  sorted_eci_dict = sorted(eci_dict.items(), key=operator.itemgetter(1))
  return sorted_eci_dict
```

```
eci_2013 = get_eci_dict(matrix)
eci_2014 = get_eci_dict(matrix3)
eci_2015 = get_eci_dict(matrix4)
eci_2016 = get_eci_dict(matrix5)
eci_2017 = get_eci_dict(matrix6)
eci_2018 = get_eci_dict(matrix2)
```

```
from collections import defaultdict
eci_list = [eci_2013, eci_2014, eci_2015, eci_2016,eci_2017, eci_2018]
eci_13to18_dict = defaultdict(list)
i = 0
for li in eci_list:
  for entry in li:
    eci_13to18_dict[entry[0]].append((2013+i, entry[1]))
  i += 1
eci_13to18_dict['United Kingdom']
```

```
     [(2013, 0.9222510373986949),
      (2014, 1.4592817624318128),
      (2015, 1.2928925410690086),
      (2016, 1.2587814161126576),
      (2017, 1.5508429006607576),
      (2018, 1.3052209031255437)]
```

```
gdp_13to18_dict = defaultdict(list)
i = 2013
while i <= 2018:
  for country in df7_cl.index:
    gdp_13to18_dict[country].append((i, df7_cl.loc[country, str(i)]))
  i += 1
```

```
gdp_13to18_dict['Bangladesh']
```

```
     [(2013, 463000000000.0),
```

```
      (2014, 501000000000.0),
      (2015, 539000000000.0),
      (2016, 584000000000.0),
      (2017, 638000000000.0),
      (2018, 705000000000.0)]
```

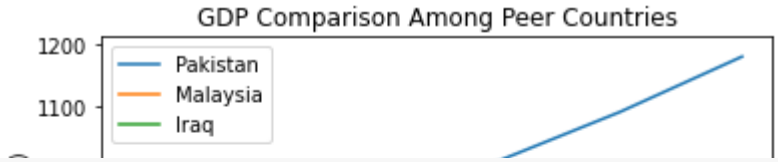
Average Annual Growth Rate

```
def aagrate(nums):
  tot = 0
  i = 0
  while i < len(nums) - 1:
    tot += (((nums[i+1]-nums[i])/nums[i]) * 100)
    i += 1
  return tot/len(nums)
```

GDP Peer Comparison Graph

```
def gdp_comparison_graph(cnames):
  plt.ylabel('GDP in USD (billions)')
  for country_name in cnames:
    if country_name == 'Plurinational State of Bolivia':
      label = 'Bolivia'
    elif country_name == 'Norway, Svalbard and Jan Mayen':
      label = 'Norway'
    elif country_name == 'Russian Federation':
      label = 'Russia'
    elif country_name == 'Republic of Korea':
      label = 'South Korea'
    else:
      label = country_name
    plt.plot([x[0] for x in gdp_13to18_dict[country_name]],[float(x[1])/(10**9) for x in gdp_
  plt.legend()
  plt.title('GDP Comparison Among Peer Countries')
  plt.show()
```

```
gdp_comparison_graph(['Pakistan', 'Malaysia', 'Iraq'])
```




```
def printgdps(cns):
  for cn in cns:
    print(round(aagrate([x[1] for x in gdp_13to18_dict[cn]]),2))

printgdps(['Pakistan', 'Malaysia', 'Iraq', 'Vietnam'])
```

```
     7.75
     1.31
     7.18
     1.46
```

ECI Peer Comparison Graph

```
def eci_comparison_graph(cnames):
  plt.ylabel('ECI')
  for country_name in cnames:
    if country_name == 'Plurinational State of Bolivia':
      label = 'Bolivia'
    elif country_name == 'Norway, Svalbard and Jan Mayen':
      label = 'Norway'
    elif country_name == 'Russian Federation':
      label = 'Russia'
    else:
      label = country_name
    plt.plot([x[0] for x in eci_13to18_dict[country_name]],[x[1] for x in eci_13to18_dict[cou
  plt.legend()
  plt.title('ECI Comparison Among Peer Countries')
  plt.show()
```

```
eci_comparison_graph(['Pakistan', 'Malaysia'])
```

ECI Comparison Among Peer Countries

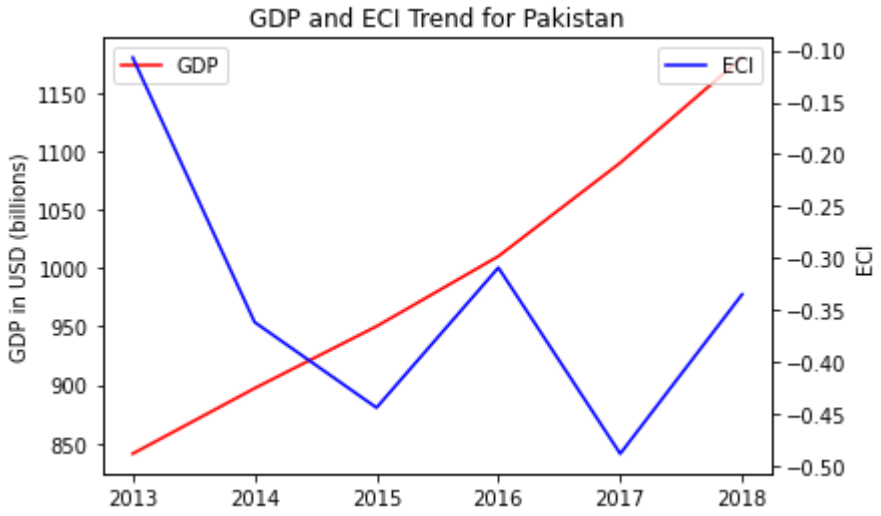
```
def eci_change(cns):
  for country_name in cns:
    print(round((eci_13to18_dict[country_name][len(eci_13to18_dict[country_name])-1][1]-eci_1

eci_change(['Pakistan', 'Malaysia'])
```

```
-212.75
13.03
```

GDP Growth and ECI Trend Graph

```
def gdp_and_eci_graph(country_name):
  fig, ax1 = plt.subplots()
  ax1.set_ylabel('GDP in USD (billions)')
  ax1.plot([x[0] for x in gdp_13to18_dict[country_name]],[float(x[1])/(10**9) for x in gdp_13
  ax2 = ax1.twinx()
  ax2.set_ylabel('ECI')
  ax2.plot([x[0] for x in eci_13to18_dict[country_name]],[x[1] for x in eci_13to18_dict[count
  ax1.legend()
  ax2.legend()
  plt.title('GDP and ECI Trend for ' + country_name)
  plt.show()
```

```
gdp_and_eci_graph('Pakistan')
```



Proximity, Distance, and RID

Proximity

```
def compute_proximity(mcp,product_sum):
  proximity = []
  for i in range(0,len(mcp[0])):
    product_row = []
    for j in range(0,len(mcp[0])):
      temp_sum = 0
      for k in range(0,len(mcp)):
        try:
          temp_sum += mcp[k][i] * mcp[k][j]
        except:
          pass
      p0 = product_sum[i]
      p1 = product_sum[j]
      if p0 < p1:
        large = p1
      else:
        large = p0
      product_row.append(temp_sum/large)
    proximity.append(product_row)
  return proximity
```

Distance

```
#compute distance
def compute_distance(mcp,proximity):
  distance = []
  for i in range(0,len(mcp)):
    product_row = []
    for j in range(0,len(mcp[0])):
      sum1 = 0
      for k in range(0,len(mcp[0])):
        if k != j:
          try:
            sum1 += mcp[i][k]*proximity[j][k]
          except:
            pass
      sum2 = 0
      for k in range(0,len(mcp[0])):
        if k != j:
          try:
            sum2 += proximity[j][k]
          except:
            pass
      product_row.append(1-sum1/sum2)
    distance.append(product_row)
  return distance
```

RID

```
def compute_adi(df2,country_list,distance):
  raw_matrix = df2.to_numpy()
  matrix = [list(df2.columns)]
  for row in raw_matrix:
    if sum(row[1::]) > 0:
      matrix.append(list(row))

  country_list2,product_list2,mcp2,product_sum2,country_sum2 = compute_mcp(matrix)
  proximity2 = compute_proximity(mcp2,product_sum2)
  distance2 = compute_distance(mcp2,proximity2)

  adi = {}
  for i in range(len(country_list)):
    for j in range(len(country_list2)):
      if country_list[i] == country_list2[j]:
        adi[country_list[i]] = np.array(distance2[j]) - np.array(distance[i])
  return adi, country_list2,product_list2
```

```
proximity = compute_proximity(mcp,product_sum)
proximity_18 = compute_proximity(mcp_18,product_sum_18)

distance = compute_distance(mcp,proximity)
adi, country_list2,product_list2  = compute_adi(df2,country_list,distance)
```

```
print(adi)
```

```
     {'Bulgaria': array([-0.05656147,  0.10781709,  0.08260293,  0.05362079,  0.09987232,
            -0.00448736,  0.09543358,  0.11876785,  0.04276899,  0.0579003 ,
            -0.00403864, -0.05881344,  0.00660341,  0.09788831,  0.00554464,
             0.0473999 , -0.04144822,  0.07104513]), 'Myanmar': array([-0.01122475,  0.000
             0.01204726, -0.08499633,  0.01759321, -0.01438475,  0.00319003,
             0.09903481,  0.07602243,  0.09087697,  0.04320619,  0.06412416,
            -0.0162516 ,  0.05986468, -0.05767907]), 'Burundi': array([-0.2001496 , -0.221
            -0.18175333, -0.17334847, -0.16390731, -0.2015974 , -0.16130462,
            -0.19766996, -0.12774187, -0.08174553, -0.14447051, -0.12425716,
            -0.2170069 , -0.0856746 , -0.30312362]), 'Canada': array([-0.04701712, -0.0425
            -0.07308198, -0.0476176 , -0.04503776, -0.06959114, -0.02223379,
            -0.07097157, -0.02962156,  0.03289839, -0.06458142, -0.04894972,
            -0.0263207 , -0.00846888, -0.06238256]), 'Chile': array([-0.00149814,  0.03142
            -0.01154326,  0.00267891, -0.0029534 , -0.01132138,  0.00821169,
            -0.0016459 ,  0.00431624,  0.0502621 , -0.03948242, -0.01071289,
            -0.02916391,  0.03961106,  0.06477578]), 'China': array([-0.01783163, -0.02721
            -0.03850733, -0.02241205, -0.03507308, -0.00751196, -0.01014312,
            -0.02898244, -0.06663006, -0.09189158, -0.01539533, -0.03953489,
            -0.00653897, -0.09203746, -0.00168595]), 'Colombia': array([-0.23573749, -0.25
            -0.25556051, -0.15334505, -0.20787381, -0.24712053, -0.2158682 ,
            -0.27229612, -0.22385598, -0.17074261, -0.18389807, -0.19656361,
            -0.22715398, -0.1748891 , -0.34359837]), 'Costa Rica': array([-0.03020222,  0.
            -0.02289988, -0.04637893, -0.02402343, -0.02869434, -0.00247713,
            -0.05292096, -0.07067365, -0.03183293, -0.00034533, -0.06381919,
            -0.03077065,  0.04355424, -0.00626835]), 'Croatia': array([-0.19903622,  0.013
            -0.12477774, -0.12652956, -0.16650168, -0.15061249, -0.09253064,
```

```
-0.11575376, -0.08555777, -0.05761213, -0.17717831, -0.10089965,
-0.12644372, -0.05381548, -0.02127805]), 'Cyprus': array([0.42402327, 0.272044
0.2515025 , 0.29571837, 0.40097291, 0.36072208, 0.35426937,
0.27594243, 0.21179171, 0.16853018, 0.30639161, 0.27977124,
0.33867159, 0.16628579, 0.31043177]), 'Andorra': array([0.12034509, 0.10429861
0.10567956, 0.10186636, 0.10827731, 0.13206528, 0.07342851,
0.10481381, 0.02722374, 0.06115478, 0.09566338, 0.10968116,
0.08761818, 0.06734776, 0.10194874]), 'Czechia': array([ 0.14014429,  0.132228
 0.03050484,  0.05581585, -0.03690101,  0.11225901,  0.17752048,
 0.07237294, -0.00115229, -0.07166728,  0.11519304,  0.06096066,
 0.07715542, -0.08020791,  0.091986  ]), 'Denmark': array([0.23318789, 0.07330
0.19711342, 0.13954848, 0.24998319, 0.22443999, 0.19527627,
0.23824793, 0.15090425, 0.16555416, 0.24625494, 0.1802945 ,
0.22501396, 0.1305283 , 0.24607875]), 'Ecuador': array([-0.01525811,  0.010365
-0.04084809, -0.00393348,  0.01276632, -0.02558579, -0.00800222,
-0.0187567 , -0.02109478,  0.03659035,  0.01479335,  0.00215795,
-0.02705628,  0.01640005, -0.02062388]), 'El Salvador': array([-0.02344631, -0
 0.00397038, -0.05290443,  0.00196409, -0.04997111, -0.03182329,
-0.04543521, -0.0809991 ,  0.00896517,  0.00380929, -0.04233962,
-0.0418791 , -0.04539141,  0.03727997]), 'Estonia': array([0.14525713, 0.15344
0.11789929, 0.14208308, 0.20293912, 0.09029771, 0.12145015,
0.1256444 , 0.12264546, 0.14162094, 0.13799765, 0.17510163,
0.12460895, 0.23260295, 0.13967349]), 'Fiji': array([-0.23434228, -0.21593045,
-0.20993768, -0.11870083, -0.06171041, -0.2319822 , -0.25457345,
-0.23450802, -0.21273245, -0.13398598, -0.22279094, -0.14649486,
-0.15318841, -0.1350988 , -0.2259982 ]), 'Finland': array([-0.03294429, -0.036
-0.08456267, -0.07203049, -0.03277021, -0.04487129, -0.03314209,
-0.05162969, -0.13080635, -0.13634775, -0.08401182, -0.04317774,
-0.05897797, -0.12183537, -0.03019487]), 'France, Monaco': array([ 0.00103559,
-0.00718584,  0.00383823,  0.01146846,  0.02791331,  0.03416042,
 0.01003544,  0.01150348,  0.00468962,  0.02368909, -0.00441401,
 0.00319861, -0.01622082,  0.14222512]), 'Georgia': array([ 0.04168255,  0.061
```

```
new_conts = ['United Kingdom', 'Ireland', 'Sweden', 'Denmark', 'Israel', 'Lebanon', 'India',
```

```
df_ = pd.DataFrame()
df_['Category'] = product_list_18
df_['RID'] = adi[new_conts[13]]
df_.sort_values(by="RID")
```

	Category	RID
5	Chemical Manufacturing	-0.073082
10	Metals	-0.070972
8	Clothing and Accessories	-0.069591
13	Weapons	-0.064581
2	Food and Preparations	-0.062557
17	Services and Utilities	-0.062383
3	Energy Drilling and Mining	-0.053457
14	Daily Instruments	-0.048950
6	Raw Manufacturing	-0.047618
0	Animals and By-products	-0.047017

Sequential Strategy

Opportunity Gain

```
def compute_opportunity_gain(mcp,proximity,pci):
  opportunity_gain = []
  for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
      gain = 0
      for k in range(0,len(mcp[0])):
        temp = 0
        for l in range(0,len(mcp[0])):
          temp += proximity[l][k]
        gain += (proximity[j][k]/temp)*(1-mcp[i][k])*pci[k]
      gain = gain - (1-distance[i][j]) * pci[j]
      if mcp[i][j] == 1:
        country_row.append(0)
      else:
        country_row.append(gain)
      #country_row.append(gain)
    opportunity_gain.append(country_row)

  opportunity_gain = np.array(opportunity_gain)
  return opportunity_gain
```

Marginal Opportunity Gain

```
def compute_marginal_opportunity_gain(mcp,opportunity_gain,country_sum,product_sum,matrix):
```

```
def compute_marginal_opportunity_gain(mcp,opportunity_gain,country_sum,product_sum,matrix):
  marginal = []
  total_sum = np.sum(product_sum)
  for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
      if mcp[i][j] == 1:
        country_row.append(0)
      else:
        export = matrix[i+1][j+1]
        country_export = country_sum[i]
        product_export = product_sum[j]
        export_increase = (product_export*country_export/total_sum)-export
        country_row.append(opportunity_gain[i][j]*export/export_increase)
    marginal.append(country_row)

  marginal = np.array(marginal)
  return marginal
```

```
opportunity_gain = compute_opportunity_gain(mcp,proximity,pci)
opportunity_gain_18 = compute_opportunity_gain(mcp_18,proximity_18,pci_18)

marginal_opportunity_gain = compute_marginal_opportunity_gain(mcp,opportunity_gain,country_su

#for i in range(len(country_list)):
#  print(country_list[i],opportunity_gain[i])
```

```
# For computation of sequential strategies, refer to sequential strategy.py
```

```
# For sensitivity analysis, refer to sensitivity.ipynb
```

```
# For computation of SOI, soi.ipynb
```

```

```

SDV & SIDS

```
def read_country(filename):
    f = open(filename,'r', encoding='utf-8')
    reader = csv.reader(f)
    raw_matrix = []
    for row in reader:
        temp = []
        for i in range(0,len(row)):
            try:
                temp.append(float(row[i]))
```

```
                temp.append(float(row[i]))
            except ValueError:
                temp.append(row[i])
        raw_matrix.append(temp)
    f.close()
    return raw_matrix

def rank(score_dict):
    rank_list = []
    score_list = []
    for item in score_dict:
        score_list.append(score_dict[item])
    while score_list:
        maximum = max(score_list)
        for item in score_dict:
            if score_dict[item] == maximum:
                rank_list.append(item)
        score_list = [score for score in score_list if score!= maximum]
    return rank_list


def sdv_and_sids(country, year):
    #Read the raw matrix
    raw_matrix = read_country(f"{year}.csv")
    raw_matrix = [row for row in raw_matrix if np.all(row[1:]==np.zeros(len(row)-1)) == False
    #compute mcp
    product_list = raw_matrix[0][1::]
    country_list = []
    for i in range(1,len(raw_matrix)):
        country_list.append(raw_matrix[i][0])

    country_sum = []
    for i in range(1,len(raw_matrix)):
        country_sum.append(np.sum(raw_matrix[i][1::]))

    product_sum = []
    for i in range(1,len(raw_matrix[0])):
        product = 0
        for j in range(1,len(raw_matrix)):
            product += raw_matrix[j][i]
        product_sum.append(product)
    total_sum = np.sum(product_sum)

    #print(product_list)
    #print(country_list)

    mcp = []

    for i in range (1,len(raw_matrix)):
        country_row = []
        for j in range(1,len(raw_matrix[i])):
            percentage = raw_matrix[i][j]/country_sum[i-1]
```

```
            total_percentage = product_sum[j-1]/total_sum
            if percentage > total_percentage:
                country_row.append(1)
            else:
                country_row.append(0)
        mcp.append(country_row)

    #compute mpp
    country_sum2 = []
    for i in range(0,len(mcp)):
        country_sum2.append(np.sum(mcp[i]))

    product_sum2 = []
    for i in range(0,len(mcp[0])):
        product = 0
        for j in range(0,len(mcp)):
            product += mcp[j][i]
        product_sum2.append(product)

    mpp = []
    for i in range(0,len(mcp[0])):
        product_row = []
        for j in range(0,len(mcp[0])):
            temp_sum = 0
            for k in range(0,len(mcp)):
                try:
                    temp_sum += (mcp[k][i] * mcp[k][j]/(country_sum2[k]*product_sum2[i]))
                except:
                    pass
            product_row.append(temp_sum)
        mpp.append(product_row)

    #compute PCI
    mpp = np.array(mpp)

    e_vals, e_vecs = LA.eig(mpp)
    eigenvectors = e_vecs.tolist()
    eigen = []
    for eigenvector in eigenvectors:
        temp = []
        for col in eigenvector:
            temp.append(col.real)
        eigen.append(temp)

    vector = []
    for i in range(0,len(eigen)):
        vector.append(eigen[i][1])

    pci = []
    if vector[len(vector)-2] > 0:
        for item in vector:
            pci.append((item-np.mean(vector))/np.std(vector))
```

```
    else:
        for item in vector:
            pci.append((-item+np.mean(vector))/np.std(vector))

    #compute proximity
    proximity = []
    for i in range(0,len(mcp[0])):
        product_row = []
        for j in range(0,len(mcp[0])):
            temp_sum = 0
            for k in range(0,len(mcp)):
                try:
                    temp_sum += mcp[k][i] * mcp[k][j]
                except:
                    pass
            p0 = product_sum2[i]
            p1 = product_sum2[j]
            if p0 < p1:
                large = p1
            else:
                large = p0
            product_row.append(temp_sum/large)
        proximity.append(product_row)

    #compute actual distance
    distance = []
    for i in range(0,len(mcp)):
        product_row = []
        for j in range(0,len(mcp[0])):
            sum1 = 0
            for k in range(0,len(mcp[0])):
                try:
                    sum1 += (1-mcp[i][k])*proximity[j][k]
                except:
                    pass
            sum2 = 0
            for k in range(0,len(mcp[0])):
                try:
                    sum2 += proximity[j][k]
                except:
                    pass
            if mcp[i][j] == 1:
                product_row.append(0)
            else:
                product_row.append(sum1/sum2)
        distance.append(product_row)

    #compute opportunity gain
    opportunity_gain = []
    for i in range(0,len(mcp)):
        country_row = []
```

```
    for j in range(0,len(mcp[0])):
        gain = 0
        for k in range(0,len(mcp[0])):
            temp = 0
            for l in range(0,len(mcp[0])):
                temp += proximity[l][k]
            gain += (proximity[j][k]/temp)*(1-mcp[i][k])*pci[k]
        gain = gain - (1-distance[i][j]) * pci[j]
        if mcp[i][j] == 1:
            country_row.append(0)
        else:
            country_row.append(gain)
    opportunity_gain.append(country_row)

#compute marginal opportunity gain
marginal = []
for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
        if mcp[i][j] == 1:
            country_row.append(0)
        else:
            export = raw_matrix[i+1][j+1]
            country_export = country_sum[i]
            product_export = product_sum[j]
            export_increase = (product_export*country_export/total_sum)-export
            country_row.append(opportunity_gain[i][j]*country_export/export_increase)
    marginal.append(country_row)


#computing sequential strategy
strategy = []

for i in range(0,len(country_list)):
    if country in country_list[i]:
        country_index = i
og = np.copy(opportunity_gain[country_index])
sdv = dict(zip(product_list, og))
sdv = sorted(sdv.items(), key=operator.itemgetter(1), reverse=True)
sdv_colors = []
for i in range(len(sdv)):
    if sdv[i][1]>0:
      sdv_colors.append(44)
    elif sdv[i][1]==0:
      sdv_colors.append(27)
    else:
      sdv_colors.append(1)
#print("Before Iteration: ")
#for i in range(0,len(marginal[country_index])):
#    print(product_list[i], ": ", marginal[country_index][i])
#print(" ")
```

```
    temp = []
    temp.append("Before Iteration")
    posi = {}
    zero = []
    nega = {}
    colors = []
    color = [1]
    for i in range(0,len(marginal[country_index])):
        if marginal[country_index][i] > 0:
            posi[product_list[i]] = marginal[country_index][i]
        elif marginal[country_index][i] == 0:
            zero.append(product_list[i])
        else:
            nega[product_list[i]] = marginal[country_index][i]
    posi = rank(posi)
    nega = rank(nega)
    temp.extend(posi)
    color.extend([44 for i in range(len(posi))])
    temp.extend(zero)
    color.extend([27 for i in range(len(zero))])
    temp.extend(nega)
    color.extend([1 for i in range(len(nega))])
    strategy.append(temp)
    colors.append(color)

    ite_round = 0
    prev_industry = []

    while max(marginal[country_index]) > 0 and ite_round < 20:
        ite_round += 1

        value = max(marginal[country_index])
        for i in range(0,len(marginal[0])):
            if marginal[country_index][i] == value:
                industry_index = i
        industry = product_list[industry_index]
        prev_industry.append(industry_index)

        #calculate new export
        export = raw_matrix[country_index+1][industry_index+1]
        country_export = country_sum[country_index]
        product_export = product_sum[industry_index]
        required_export = product_export*country_export/total_sum
        export_increase = required_export-export

        #print("increase ", industry, " in ", country, " by ", export_increase, " USD")

        raw_matrix[country_index+1][industry_index+1] = required_export*1.3

        #calculate everything again
        country_sum = []
        for i in range(1,len(raw_matrix)):
```

```
for i in range(1,len(raw_matrix)):
    country_sum.append(np.sum(raw_matrix[i][1::]))

product_sum = []
for i in range(1,len(raw_matrix[0])):
    product = 0
    for j in range(1,len(raw_matrix)):
        product += raw_matrix[j][i]
    product_sum.append(product)
total_sum = np.sum(product_sum)

mcp = []

for i in range (1,len(raw_matrix)):
    country_row = []
    for j in range(1,len(raw_matrix[i])):
        percentage = raw_matrix[i][j]/country_sum[i-1]
        total_percentage = product_sum[j-1]/total_sum
        if percentage > total_percentage:
            country_row.append(1)
        else:
            country_row.append(0)
    mcp.append(country_row)

#compute mpp
country_sum2 = []
for i in range(0,len(mcp)):
    country_sum2.append(np.sum(mcp[i]))

product_sum2 = []
for i in range(0,len(mcp[0])):
    product = 0
    for j in range(0,len(mcp)):
        product += mcp[j][i]
    product_sum2.append(product)

mpp = []
for i in range(0,len(mcp[0])):
    product_row = []
    for j in range(0,len(mcp[0])):
        temp_sum = 0
        for k in range(0,len(mcp)):
            try:
                temp_sum += (mcp[k][i] * mcp[k][j]/(country_sum2[k]*product_sum2[i]))
            except:
                pass
        product_row.append(temp_sum)
    mpp.append(product_row)

#compute PCI
mpp = np.array(mpp)
```

```
        e_vals, e_vecs = LA.eig(mpp)
        eigenvectors = e_vecs.tolist()
        eigen = []
        for eigenvector in eigenvectors:
            temp = []
            for col in eigenvector:
                temp.append(col.real)
            eigen.append(temp)

        vector = []
        for i in range(0,len(eigen)):
            vector.append(eigen[i][1])

        pci = []
        if vector[len(vector)-2] > 0:
            for item in vector:
                pci.append((item-np.mean(vector))/np.std(vector))
        else:
            for item in vector:
                pci.append((-item+np.mean(vector))/np.std(vector))

        #compute proximity
        proximity = []
        for i in range(0,len(mcp[0])):
            product_row = []
            for j in range(0,len(mcp[0])):
                temp_sum = 0
                for k in range(0,len(mcp)):
                    try:
                        temp_sum += mcp[k][i] * mcp[k][j]
                    except:
                        pass
                p0 = product_sum2[i]
                p1 = product_sum2[j]
                if p0 < p1:
                    large = p1
                else:
                    large = p0
                product_row.append(temp_sum/large)
            proximity.append(product_row)

        #compute actual distance
        distance = []
        for i in range(0,len(mcp)):
            product_row = []
            for j in range(0,len(mcp[0])):
                sum1 = 0
                for k in range(0,len(mcp[0])):
                    try:
                        sum1 += (1-mcp[i][k])*proximity[j][k]
                    except:
                        pass
```

```
                pass
        sum2 = 0
        for k in range(0,len(mcp[0])):
            try:
                sum2 += proximity[j][k]
            except:
                pass
        if mcp[i][j] == 1:
            product_row.append(0)
        else:
            product_row.append(sum1/sum2)
    distance.append(product_row)

#compute opportunity gain
opportunity_gain = []
for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
        gain = 0
        for k in range(0,len(mcp[0])):
            temp = 0
            for l in range(0,len(mcp[0])):
                temp += proximity[l][k]
            gain += (proximity[j][k]/temp)*(1-mcp[i][k])*pci[k]
        gain = gain - (1-distance[i][j]) * pci[j]
        if mcp[i][j] == 1:
            country_row.append(0)
        else:
            country_row.append(gain)
    opportunity_gain.append(country_row)

#compute marginal opportunity gain
marginal = []
for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
        if mcp[i][j] == 1:
            country_row.append(0)
        else:
            export = raw_matrix[i+1][j+1]
            country_export = country_sum[i]
            product_export = product_sum[j]
            export_increase = (product_export*country_export/total_sum)-export
            country_row.append(opportunity_gain[i][j]*country_export/export_increase)
    marginal.append(country_row)

#print("Iteration ", ite_round, ": ")
#for i in range(0,len(marginal[country_index])):
#    print(product_list[i], ": ", marginal[country_index][i])
#print(" ")

temp = []
```

```
        string = "Iteration " + str(ite_round)
        temp.append(string)
        posi = {}
        zero = []
        nega = {}
        color = [1]
        for i in range(0,len(marginal[country_index])):
            if marginal[country_index][i] > 0:
                posi[product_list[i]] = marginal[country_index][i]
            elif marginal[country_index][i] == 0:
                zero.append(product_list[i])
            else:
                nega[product_list[i]] = marginal[country_index][i]
        posi = rank(posi)
        nega = rank(nega)
        temp.extend(posi)
        color.extend([44 for i in range(len(posi))])
        temp.extend(zero)
        color.extend([27 for i in range(len(zero))])
        temp.extend(nega)
        color.extend([1 for i in range(len(nega))])
        strategy.append(temp)
        colors.append(color)

    strategy = np.array(strategy)
    strategy = np.transpose(strategy)
    colors = np.array(colors)
    colors = np.transpose(colors)
    #print(f'color shape is {colors.shape}')

    book = Workbook()
    sheet1 = book.add_sheet('Sheet 1')
    for i in range(0,len(strategy)):
        for j in range(len(strategy[0])):
          st = xlwt.easyxf('pattern: pattern solid;')
          st.pattern.pattern_fore_colour = colors[i][j]
          sheet1.write(i,j,strategy[i][j], st)
    book.save(f'sequential strategy of {country}.xls')

    book = Workbook()
    sheet1 = book.add_sheet('Sheet 1')
    for i in range(0,len(sdv)):
          st = xlwt.easyxf('pattern: pattern solid;')
          st.pattern.pattern_fore_colour = sdv_colors[i]
          sheet1.write(i, 0, sdv[i][0], st)
    book.save(f'sequential vector of {country}.xls')
    print(f'{country} completed')

def main():
    countries = ['Canada', 'Denmark', 'Chile', 'Ireland', 'Ghana', 'India', 'Kenya', 'Israel'

    for country in countries:
```

```
        sdv_and_sids(country, root_path+'2013-2018 cleaned/2013clean')
```

```
if __name__ == "__main__":
    main()   #Change the countries and year above to have different results
```

```
     /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:33: FutureWarning:

     elementwise comparison failed; returning scalar instead, but in the future will perform

     Canada completed
     Denmark completed
     Chile completed
     Ireland completed
     Ghana completed
     India completed
     Kenya completed
     Israel completed
     Lebanon completed
     Russia completed
     Pakistan completed
     Qatar completed
     Sweden completed
     United Kingdom completed
     Bangladesh completed
```

Greedy Algorithm

```
def greedy(country, year, percentage, num_part):
    df = pd.read_csv(root_path+'2013-2018 cleaned/2015clean.csv')
    raw_matrix = df.to_numpy()
    matrix = [list(df.columns)]

    for row in raw_matrix:
      if np.min(row[1::]) > 0:
        matrix.append(list(row))

    #print(len(raw_matrix))
    product_list = matrix[0][1::]
    #print(product_list)
    country_list = []
    for i in range(1,len(matrix)):
        country_list.append(matrix[i][0])
        if country in matrix[i][0]:
           country_index = i
           #print(country_index)
           total_export = np.sum(matrix[i][1::])

    budget = (percentage/100) * total_export
```

```
budget = (percentage/100) * total_export
print(f'Original total export is {total_export}, budget is {budget}')
mcp = compute_mcp(matrix)[2]
mcc = compute_mcc(mcp)
last_eci = compute_eci(mcc)[country_index-1]
print(f'ECI this year is {last_eci}')
each_amount = budget/num_part
original_vector = np.copy(matrix[country_index][1::])

total = 0
while total < budget:
    eci = []
    print(f'Progress: {round(total/budget, 2)}')
    for i in tqdm(range(1, len(matrix[0]))):
        matrix[country_index][i] += each_amount
        mcp = compute_mcp(matrix)[2]
        mcc = compute_mcc(mcp)
        eci.append(compute_eci(mcc)[country_index - 1])
        matrix[country_index][i] -= each_amount
    if  eci.count(max(eci)) != 1:
        #print('path 2')
        multiple = 2
        total_temp = total
        temp_each = multiple*each_amount
        while max(eci)<=last_eci and total_temp < budget:
            temp_each = multiple*each_amount
            print(f'Multiple = {multiple}, Raise each amount to {temp_each}')
            eci = []
            for i in tqdm(range(1, len(matrix[0]))):
                matrix[country_index][i] += temp_each
                mcp = compute_mcp(matrix)[2]
                mcc = compute_mcc(mcp)
                eci.append(compute_eci(mcc)[country_index - 1])
                matrix[country_index][i] -= temp_each
            multiple += 1
            total_temp = total + temp_each
        if max(eci)<=last_eci and total_temp == budget:
           print('ECI does not change any more, add to the most adjacent ')
           export_increase = []
           mcp, product_sum, country_sum = compute_mcp(matrix)[2:]
           total_sum = np.sum(product_sum)
           for j in range(0,len(mcp[0])):
              if mcp[country_index-1][j] == 1:
                  export_increase.append(0)
              else:
                  export = raw_matrix[country_index][j+1]
                  country_export = country_sum[country_index-1]
                  product_export = product_sum[j]
                  export_increase.append((product_export*country_export/total_sum)-export
           country_dist = np.array(export_increase)
           min_distance = np.where(country_dist > 0, country_dist, np.inf).argmin()
           matrix[country_index][min_distance + 1] += temp_each
```

```
            else:
               matrix[country_index][np.argmax(eci) + 1] += temp_each
               print(eci)
               print(f'After increasing, largest argument is {np.argmax(eci) + 1}, product is
               print(f'last_eci is {last_eci}, this_eci is {max(eci)}')
            total = total_temp
            last_eci = max(eci)
        else:
            #print('path 3')
            largest_component = np.argmax(eci) + 1
            print(eci)
            print(f'largest argument is {np.argmax(eci) + 1}, product is {product_list[np.arg
            print(f'last_eci is {last_eci}, this_eci is {max(eci)}')
            last_eci = max(eci)
            matrix[country_index][np.argmax(eci) + 1] += each_amount
            total += each_amount
    optimal_vector = np.array(matrix[country_index][1::]) - original_vector
    max_eci = last_eci
    return dict(zip(product_list,optimal_vector)), max_eci
```

```
greedy('Morocco', 2015, 10, 12)
```

```
 Original total export is 31004050436.0, budget is 3100405043.6000004
   0%|          | 0/18 [00:00<?, ?it/s]ECI this year is 0.4597544664749378
 Progress: 0.0
 100%|██████████| 18/18 [00:25<00:00,  1.44s/it]
   0%|          | 0/18 [00:00<?, ?it/s]Multiple = 2, Raise each amount to 516734173.93333
  39%|███▉      | 7/18 [00:10<00:15,  1.45s/it]
```

SOI

```
def read_it(filename):
    f = open(filename,'r', encoding='cp1252')
    reader = csv.reader(f)
    raw_matrix = []
    for row in reader:
        temp = []
        for i in range(0,len(row)):
            try:
                temp.append(float(row[i]))
            except ValueError:
                #print(row[i])
                temp.append(row[i])
        raw_matrix.append(temp)
    f.close()
    return raw_matrix
```

```
zzz+=1
def compute_soi_adi(adi, country_list, product_list,country_list2, product_list2, opportunity
    soi = {}
    if len(product_list) == len(product_list2):
        print("categorization verified")

    common = []
    for i in range(0,len(country_list)):
        if country_list[i] in country_list2:
            for j in range(0,len(country_list2)):
                if country_list[i] == country_list2[j]:
                    dependent = adi[country_list[i]]
                    independent = []
                    for item in opportunity_gain[i]:
                        independent.append(item)
                    dfs=  pd.DataFrame({'I': independent, 'D': dependent})
                    model = smf.ols(formula='D ~ I', data=dfs).fit()
                    r_2 = model.rsquared_adj
                    plt.scatter(independent, dependent)
                    plt.show()
                    print(model.cov_HC0)
                    print(country_list[i], "!!!")
                    print(country_list2[j], "???")
                    common.append(country_list[i])
                    soi[country_list[i]] =r_2

    matrix = open(root_path+"Rohan Testing/structual optimality index adi ("+str(zzz)+ ").csv
    writer = csv.writer(matrix)
    col_name = ["Country","SOI"]
    writer.writerow(col_name)
    for i in range(0,len(soi)):
       temp = [common[i],soi[common[i]]]
       writer.writerow(temp)
    matrix.close()
    return soi
```

```
def compute_soi_oppgain(country_list, product_list,country_list2, product_list2, opportunity_
    soi = {}
    if len(product_list) == len(product_list2):
        print("categorization verified")

    common = []
    for i in range(0,len(country_list)):
        if country_list[i] in country_list2:
            for j in range(0,len(country_list2)):
                if country_list[i] == country_list2[j]:
                    independent=[]
                    dependent=[]
                    for item in opportunity_gain2[j]:
                        #dependent.append([item])
                        dependent.append(item)
```

```
                    dependent.append(item)
                for item in opportunity_gain[i]:
                    #independent.append([item])
                    independent.append(item)
                # reg = LinearRegression()
                # reg.fit(independent, dependent)
                # r_2 = reg.score(independent, dependent)
                dfs=  pd.DataFrame({'I': independent, 'D': dependent})
                model = smf.ols(formula='D ~ I', data=dfs).fit()
                r_2 = model.rsquared_adj
                common.append(country_list[i])
                soi[country_list[i]] =r_2

    matrix = open(root_path+"Rohan Testing/structual optimality index oppgain.csv",'w',newlin
    writer = csv.writer(matrix)
    col_name = ["Country","SOI"]
    writer.writerow(col_name)
    for i in soi.keys():
        temp = [i,soi[i]]
        writer.writerow(temp)
    matrix.close()
    return soi
```

```
def compute_soi_exp(country_list, product_list,country_list2, product_list2, opportunity_gain
    soi = {}
    if len(product_list) == len(product_list2):
        print("categorization verified")
    for i in range(0,len(country_list)):
        if country_list[i] in country_list2:
            for j in range(0,len(country_list2)):
                if country_list[i] == country_list2[j]:
                    dependent = []
                    independent = []
                    for item in opportunity_gain[i]:
                        #independent.append([item])
                        independent.append(item)
                    #print(opportunity_gain2[i])
                    for item in exp_change.loc[country_list[i],:]:
                        #dependent.append([item])
                        dependent.append(item)
                    #reg = LinearRegression()
                    #reg.fit(independent, dependent)
                    #r_2 = reg.score(independent, dependent)
                    dfs=  pd.DataFrame({'I':independent, 'D': dependent})
                    model = smf.ols(formula='D ~ I', data=dfs).fit()
                    r_2 = model.rsquared_adj
                    soi[country_list[i]] = r_2

    matrix = open(root_path+"Rohan Testing/structual optimality index oppgain.csv",'w',newlin
    writer = csv.writer(matrix)
    col_name = ["Country","SOI"]
    writer.writerow(col_name)
```

```
    writer.writerow(col_name)
    for i in soi.keys():
        temp = [i,soi[i]]
        writer.writerow(temp)
    matrix.close()
    return soi
```

```
def compute_regress(soi, gdp_raw, country_list, country_list2):
    #comprehensive regression
    common=np.intersect1d(country_list, country_list2)
    independent = []
    dependent = []
    name = []
    common2= []
    growth={}
    for i in common:
        if i in gdp_raw.index.values and gdp_raw['2013'][i] != 0:
            pct_change=(float(gdp_raw['2018'][i])-float(gdp_raw['2013'][i]))/float(gdp_raw['2
            dependent.append(pct_change)
            independent.append(soi[i])
            growth[i]=pct_change
            common2.append(i)
    # plt.scatter(independent,dependent)
    # plt.title('SOI v GDP Growth')
    # plt.xlabel('SOI')
    # plt.ylabel('% Change in GDP')
    # plt.show()

    # print(np.setdiff1d(common,common2))
    # print()
    # print(common)
    # print()
    # print(common2)

    dfs=  pd.DataFrame({'Country': common2,'GDP':dependent, 'SOI': independent})
    #dfs.set_index('Country', inplace=True)
    model = smf.ols(formula='GDP ~ SOI', data=dfs).fit()


    fig = px.scatter(dfs, x="SOI", y="GDP",  text='Country', trendline="ols")
    fig.update_layout(
    title="Regression of GDP over SOI",
    yaxis_title="% Growth in GDP 2013-2018",
    xaxis_title="SOI",
    height=600,
    width=900,
    title_x=.5,
    font=dict(
        family="Times New Roman",
        size=15
        ) )
    #fig.update_traces(textposition='top center')
```

```
    #fig.update_traces(textposition='top center')
    fig.show()
    fig2 = plt.figure(figsize=(12,8))
    fig2 = sm.graphics.plot_regress_exog(model,'SOI', fig=fig2)
    fig3, ax = plt.subplots(figsize=(12,8))
    fig3 = sm.graphics.influence_plot(model, ax=ax, criterion='Cooks')
    return model, growth
```

```
def compute_opportunity_gain(mcp,proximity,pci):
  opportunity_gain = []
  for i in range(0,len(mcp)):
    country_row = []
    for j in range(0,len(mcp[0])):
      gain = 0
      for k in range(0,len(mcp[0])):
        temp = 0
        for l in range(0,len(mcp[0])):
          temp += proximity[l][k]
        gain += (proximity[j][k]/temp)*(1-mcp[i][k])*pci[k]
      gain = gain - (1-distance[i][j]) * pci[j]
      country_row.append(gain)
    opportunity_gain.append(country_row)

  opportunity_gain = np.array(opportunity_gain)
  return opportunity_gain

opportunity_gain = compute_opportunity_gain(mcp,proximity,pci)
opportunity_gain_18 = compute_opportunity_gain(mcp_18,proximity_18,pci_18)

marginal_opportunity_gain = compute_marginal_opportunity_gain(mcp,opportunity_gain,country_su
```

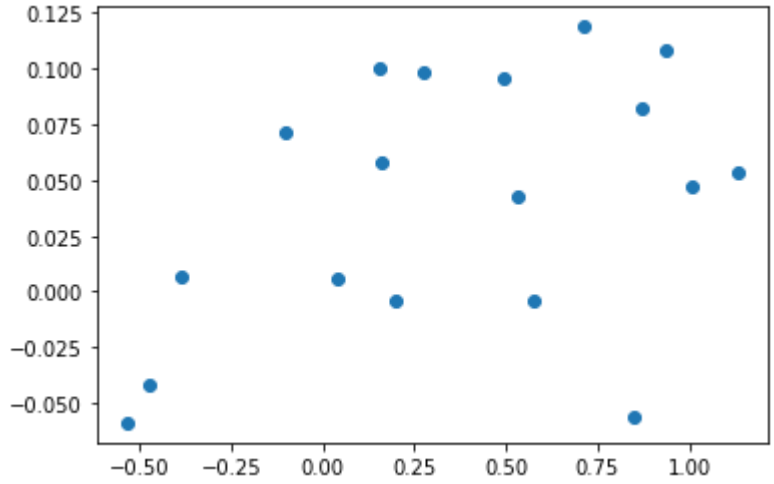
```
def soi_plotter(country, adi, country_list, product_list,country_list2, product_list2, opport
    if len(product_list) == len(product_list2):
            print("categorization verified")
    for i in range(0,len(country_list)):
        if country_list[i] in country_list2 and country_list[i]==country:
            for j in range(0,len(country_list2)):
                if country_list[i] == country_list2[j]:
                    dependent = adi[country_list[i]]
                    independent = []
                    for item in opportunity_gain[i]:
                        independent.append(item)
                    # print(independent)
                    # print(dependent)
                    # print(product_list)
                    # Create figure with secondary y-axis
                    fig = make_subplots(specs=[[{"secondary_y": True}]])

                    # Add traces
```

```
                    # Add traces
                    fig.add_trace(
                        go.Scatter(x=product_list, y=independent, name="Opportunity Gain"),
                        secondary_y=False,
                    )

                    fig.add_trace(
                        go.Scatter(x=product_list, y=dependent, name="RID"),
                        secondary_y=True,
                    )

                    # Add figure title
                    fig.update_layout(
                        title_text="SOI of "+country
                    )
                    fig.update_layout(
                        height=600,
                        width=1500,
                        title_x=.5,
                        font=dict(
                            family="Times New Roman",
                            size=15
                            ) )
                    # Set x-axis title
                    fig.update_xaxes(title_text="Product Category")

                    # Set y-axes titles
                    fig.update_yaxes(title_text="RID", secondary_y=False)
                    fig.update_yaxes(title_text="Opportunity Gain", secondary_y=True)

                    correlation_matrix = np.corrcoef(dependent, independent)
                    correlation_xy = correlation_matrix[0,1]
                    r_squared = correlation_xy**2
                    print("ASDFASDFASDFSADFASDFASDFASDF R^2: ", r_squared, "ASDFASDFASDFASDF)
                    fig.show()
```

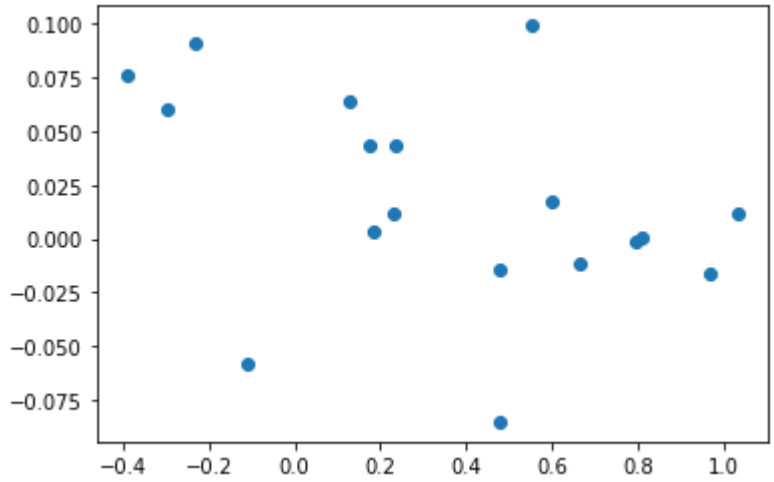
SOI w/ ADI

```
soi_adi= compute_soi_adi(adi, country_list, product_list,country_list2, product_list2, opport
soi_adi_list = sorted(soi_adi.items(), key=operator.itemgetter(1))
gdp_raw=pd.read_csv(root_path+'GDP4.csv', index_col=0).fillna(0)
print(gdp_raw)
reg, growth= compute_regress(soi_adi, gdp_raw, country_list, country_list2)
# print(len(country_list))
# print(len(country_list2))
# print(len(soi_adi_list))
reg.summary()
```

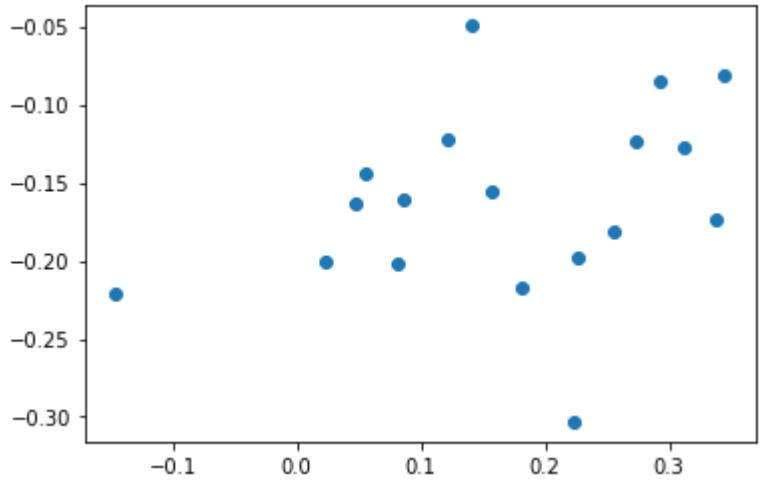
```
categorization verified
```




```
[[ 0.00016739 -0.00012406]
 [-0.00012406  0.00048335]]
Bulgaria !!!
Bulgaria ???
```




```
[[ 0.00019323 -0.00019632]
 [-0.00019632  0.00040471]]
Myanmar !!!
Myanmar ???
```




```
[[ 0.00016532 -0.00034645]
 [-0.00034645  0.00433213]]
Burundi !!!
Burundi ???
```

```
c='Pakistan'
sorted_x = sorted(soi_adi.items(), key=lambda kv: kv[1], reverse=True)
for i in range(len(sorted_x)):
  print(i, "  ", sorted_x[i])
soi_plotter(c, adi, country_list, product_list, country_list2, product_list2, opportunity_gai
print('SOI', soi_adi[c],'Growth', growth[c])
for i in growth.keys():
  print(i, "\t", growth[i])
```

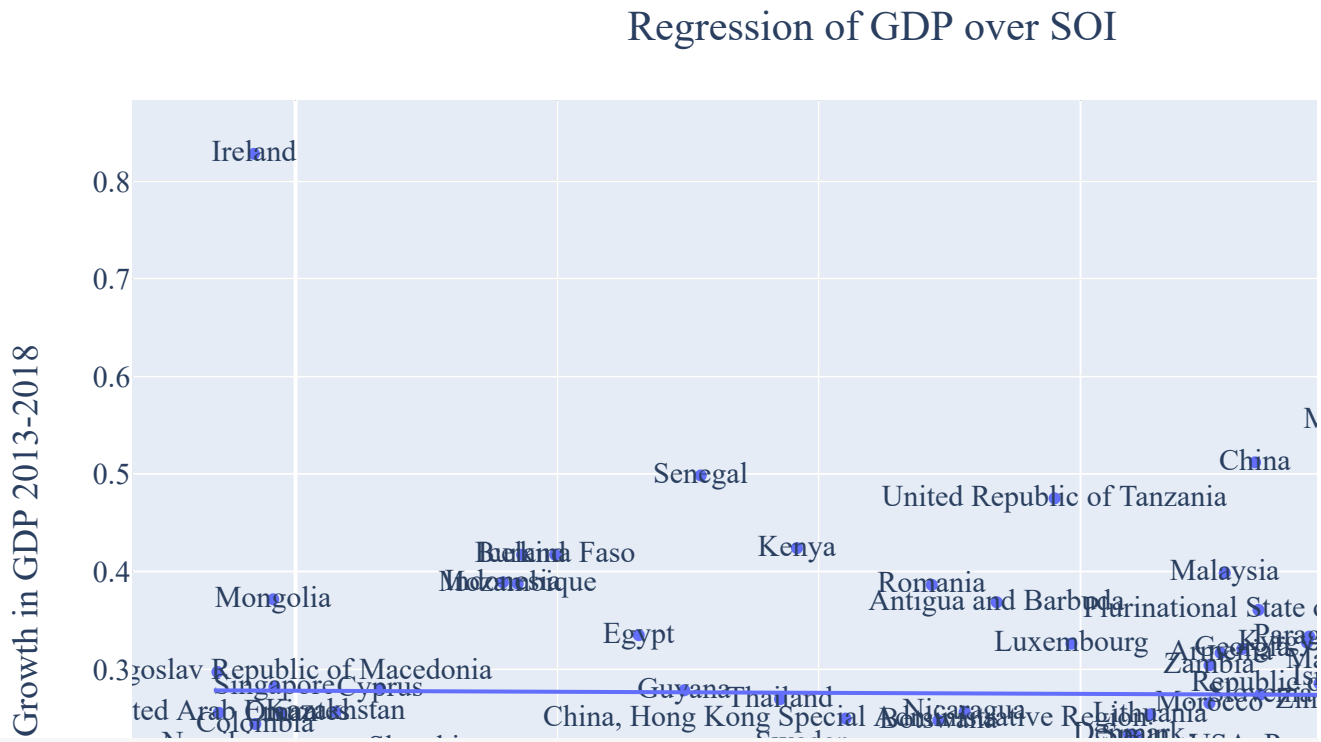
```
0    ('China', 0.5846572657612421)
1    ('Uganda', 0.4966286224252544)
2    ('United Republic of Tanzania', 0.4740806434579983)
3    ('Cyprus', 0.4607334988461864)
4    ('Samoa', 0.4549307778684747)
5    ('Serbia', 0.3944644801380698)
6    ('USA, Puerto Rico and US Virgin Islands', 0.35275937137141566)
7    ('Jordan', 0.33132962996221105)
8    ('Georgia', 0.3144112538404761)
9    ('Antigua and Barbuda', 0.3031302553390668)
10    ('Andorra', 0.3017645972441684)
11    ('Czechia', 0.29074297493740786)
```

SOI w/ Opportunity Gain

```
15    ('Ireland', 0.20839412606037766)
```

```
soi_oppg= compute_soi_oppgain(country_list, product_list,country_list_18, product_list_18, op
print(sorted(soi_oppg.items(), key=operator.itemgetter(0)))
#print(np.intersect1d(country_list,country_list_18))
reg_opp= compute_regress(soi_oppg, gdp_raw, country_list, country_list_18)
reg_opp.summary()
```

```
categorization verified
[('Andorra', 0.9182616726552274), ('Antigua and Barbuda', 0.5360366195877627), ('Armenia
```




```
#print(df1_cl)
#print(df2_cl)
#print(df1_cl-df2_cl)
exp_change= (df1_cl-df2_cl)/df1_cl
soi_exp= compute_soi_exp(country_list, product_list,country_list_18, product_list_18, opportu
print(sorted(soi_exp.items(), key=operator.itemgetter(0)))
#print(np.intersect1d(country_list,country_list_18))
reg_exp= compute_regress(soi_exp, gdp_raw, country_list, country_list_18)
reg_exp.summary()
```

```
AttributeError                            Traceback (most recent call last)

----> 5 reg_opp.summary()
```

End

```
print(np.sort(list(soi_adi.keys())))
print(np.sort(country_list))
print(np.sort(country_list2))
```

```
['Albania' 'Andorra' 'Antigua and Barbuda' 'Armenia' 'Aruba' 'Austria'
 'Azerbaijan' 'Barbados' 'Bosnia Herzegovina' 'Botswana' 'Brazil'
 'Bulgaria' 'Burkina Faso' 'Burundi' 'Canada' 'Chile' 'China'
 'China, Hong Kong Special Administrative Region' 'Colombia' 'Costa Rica'
```

```
 'Croatia' 'Cyprus' 'Czechia' 'Denmark' 'Ecuador' 'Egypt' 'El Salvador'
 'Estonia' 'Fiji' 'Finland' 'France, Monaco' 'Gambia' 'Georgia' 'Germany'
 'Ghana' 'Greece' 'Greenland' 'Guyana' 'Iceland' 'India' 'Indonesia'
 'Ireland' 'Israel' 'Italy' 'Japan' 'Jordan' 'Kazakhstan' 'Kenya' 'Kuwait'
 'Kyrgyzstan' 'Latvia' 'Lebanon' 'Lithuania' 'Luxembourg' 'Madagascar'
 'Malaysia' 'Malta' 'Mauritius' 'Mexico' 'Mongolia' 'Montenegro' 'Morocco'
 'Mozambique' 'Myanmar' 'Namibia' 'Netherlands' 'New Zealand' 'Nicaragua'
 'Nigeria' 'Norway, Svalbard and Jan Mayen' 'Oman' 'Pakistan' 'Paraguay'
 'Peru' 'Philippines' 'Plurinational State of Bolivia' 'Poland' 'Portugal'
 'Qatar' 'Republic of Korea' 'Republic of Moldova' 'Romania'
 'Russian Federation' 'Samoa' 'Saudi Arabia' 'Senegal' 'Serbia'
 'Singapore' 'Slovakia' 'Slovenia' 'South Africa' 'Spain'
 'State of Palestine' 'Suriname' 'Sweden' 'Switzerland, Liechtenstein'
 'Thailand' 'The Former Yugoslav Republic of Macedonia' 'Turkey'
 'USA, Puerto Rico and US Virgin Islands' 'Uganda' 'Ukraine'
 'United Arab Emirates' 'United Kingdom' 'United Republic of Tanzania'
 'Uruguay' 'Zambia' 'Zimbabwe']
['Albania' 'Algeria' 'Andorra' 'Antigua and Barbuda' 'Armenia' 'Aruba'
 'Austria' 'Azerbaijan' 'Bahamas' 'Bangladesh' 'Barbados'
 'Bosnia Herzegovina' 'Botswana' 'Brazil' 'Bulgaria' 'Burkina Faso'
 'Burundi' 'Cambodia' 'Cameroon' 'Canada' 'Chile' 'China'
 'China, Hong Kong Special Administrative Region' 'Colombia' 'Costa Rica'
 'Croatia' 'Cyprus' 'Czechia' "Côte d'Ivoire" 'Denmark'
 'Dominican Republic' 'EU-28' 'Ecuador' 'Egypt' 'El Salvador' 'Estonia'
 'Ethiopia' 'Fiji' 'Finland' 'France, Monaco' 'French Polynesia' 'Gambia'
 'Georgia' 'Germany' 'Ghana' 'Greece' 'Greenland' 'Guatemala' 'Guinea'
 'Guyana' 'Iceland' 'India' 'Indonesia' 'Iran' 'Ireland' 'Israel' 'Italy'
 'Jamaica' 'Japan' 'Jordan' 'Kazakhstan' 'Kenya' 'Kuwait' 'Kyrgyzstan'
 "Lao People's Dem. Rep." 'Latvia' 'Lebanon' 'Lesotho' 'Lithuania'
 'Luxembourg' 'Madagascar' 'Malawi' 'Malaysia' 'Malta' 'Mauritius'
 'Mexico' 'Mongolia' 'Montenegro' 'Morocco' 'Mozambique' 'Myanmar'
 'Namibia' 'Nepal' 'Netherlands' 'New Caledonia' 'New Zealand' 'Nicaragua'
 'Niger' 'Nigeria' 'Norway, Svalbard and Jan Mayen' 'Oman'
 'Other Asia, not elsewhere specified' 'Pakistan' 'Panama' 'Paraguay'
 'Peru' 'Philippines' 'Plurinational State of Bolivia' 'Poland' 'Portugal'
 'Qatar' 'Republic of Korea' 'Republic of Moldova' 'Romania'
 'Russian Federation' 'Rwanda' 'Saint Kitts and Nevis' 'Saint Lucia'
 'Samoa' 'Saudi Arabia' 'Senegal' 'Serbia' 'Singapore' 'Slovakia'
 'Slovenia' 'South Africa' 'Spain' 'Sri Lanka' 'State of Palestine'
 'Suriname' 'Swaziland' 'Sweden' 'Switzerland, Liechtenstein' 'Thailand'
 'The Former Yugoslav Republic of Macedonia' 'Togo' 'Tonga' 'Tunisia'
 'Turkey' 'USA, Puerto Rico and US Virgin Islands' 'Uganda' 'Ukraine'
 'United Arab Emirates' 'United Kingdom' 'United Republic of Tanzania'
 'Uruguay' 'Venezuela' 'Viet Nam' 'Zambia' 'Zimbabwe']
['Afghanistan' 'Albania' 'Andorra' 'Angola' 'Antigua and Barbuda'
 'Argentina' 'Armenia' 'Aruba' 'Australia' 'Austria' 'Azerbaijan'
 'Bahrain' 'Barbados' 'Belarus' 'Belgium' 'Belize' 'Benin' 'Bermuda'
 'Bosnia Herzegovina' 'Botswana' 'Brazil' 'Brunei Darussalam' 'Bulgaria'
 'Burkina Faso' 'Burundi' 'Cabo Verde' 'Canada' 'Chile' 'China'
 'China, Hong Kong Special Administrative Region' 'Colombia' 'Costa Rica'
 'Croatia' 'Cyprus' 'Czechia' "Çte d'Ivoire" 'Denmark' 'Ecuador' 'Egypt'
 'El Salvador' 'Estonia' 'Fiji' 'Finland' 'France, Monaco' 'Gambia'
 'Georgia' 'Germany' 'Ghana' 'Greece' 'Greenland' 'Guyana' 'Hungary'
 'Iceland' 'India' 'Indonesia' 'Ireland' 'Israel' 'Italy' 'Japan' 'Jordan'
 'Kazakhstan' 'Kenya' 'Kuwait' 'Kyrgyzstan' 'Latvia' 'Lebanon' 'Lithuania'
 'Luxembourg' 'Madagascar' 'Malaysia' 'Maldives' 'Malta' 'Mauritius'
Ecuador          0.15606936416184972
```

```
Estonia          0.3042403437325716
Fiji     0.3438460010330347
Finland          0.18141592920353983
France, Monaco   0.16475095785440613
Gambia   0.2693685445957011
Georgia          0.3206506992130121
Germany          0.21212121212121213
Ghana    0.3557692307692308
Greece   0.10839160839160839
Guyana   0.2782962341840547
Iceland          0.41717861789054594
India    0.5601783060921248
Indonesia        0.3888888888888889
Ireland          0.8280542986425339
Israel   0.2909090909090909
Italy    0.1552511415525114
Japan    0.09054325955734406
Jordan   0.21807729222374195
Kazakhstan       0.2561576354679803
Kenya    0.424
Kuwait   0.09420289855072464
Kyrgyzstan       0.327188252636357
Latvia   0.2783033801140199
Lebanon          0.1381353873860886
```