

ts_xterisation_rca

October 17, 2019

Time Series Characterisation

Table of Contents

- 1 Time Series
- 2 Importing Libraries for time series forecasting
- 3 Importing data
- 4 Data Preprocessing and Visualization
 - 4.1 Stationarity
 - 4.1.1 ACF and PACF plots
 - 4.1.2 Plotting Rolling Statistics
 - 4.1.3 Augmented Dickey-Fuller Test
 - 4.2 Making Time Series Stationary
 - 4.2.1 Transformations
 - 4.2.1.1 Log Scale Transformation
 - 4.2.1.2 Other possible transformations:
 - 4.2.2 Techniques to remove Trend - Smoothing
 - 4.2.2.1 Moving Average
 - 4.2.2.2 Exponentially weighted moving average:
 - 4.2.3 Further Techniques to remove Seasonality and Trend
 - 4.2.3.1 Differencing
 - 4.2.3.2 Decomposition

DATA SOURCES

This project makes use of a time series of export data from **The Observatory of Economic Complexity(OEC)** based on the SITC (Standard International Trade Classification)** from 1962 - 2000, with data from **The Center for International Data** from Robert Feenstra. The more recent data 2001 - 2017, is sourced from **UN COMTRADE**.

Full OEC trade datasets are also available from a data dump on **SITC Product 4 Digit**

1 Importing Libraries for time series forecasting

```
[98]: #Run this for quick setup  
#!/usr/bin/env python3  
#!pip install -r requirements.txt
```

```
[99]: import numpy as np  
import pandas as pd  
pd.set_option('display.expand_frame_repr', False)  
pd.set_option('display.max_columns', 500)  
pd.set_option('display.width', 1000)  
  
import statsmodels.robust  
import statsmodels.tsa.stattools as tsa  
  
from statsmodels import api as sm  
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf  
from statsmodels.tsa.stattools import adfuller  
from statsmodels.tsa.seasonal import seasonal_decompose  
  
from statsmodels.graphics.tsaplots import plot_acf  
from statsmodels.graphics.tsaplots import plot_pacf  
from pylab import rcParams  
  
from math import sqrt  
  
import matplotlib  
import matplotlib.pyplot as plt  
matplotlib.colors  
matplotlib.rcParams['axes.labelsize'] = 14  
matplotlib.rcParams['xtick.labelsize'] = 12  
matplotlib.rcParams['ytick.labelsize'] = 12  
matplotlib.rcParams['text.color'] = 'k'  
import seaborn as sns  
  
from random import random  
  
import warnings; import os  
warnings.filterwarnings("ignore")  
plt.style.available  
plt.style.use('bmh')  
import os
```

2 Importing data

- Dataset: Trade data:
 - location_id: / - product_id: SITC 4 digit code system
 - year: from 1963 to 2017 - export_value: / - import_value: - / - export_rca:

- is_new: / - cog: / - distance: / - normalized_distance: / - normalized_cog:
- normalized_pci: / - export_rpop: / - sitc_eci: / - sitc_coi: / - pci: / - location_code: / - location_name_short_en:
- sitc_product_code: / - sitc_product_name_short_en: / - Unit: arbitrary

```
[100]: product = 'engine_parts' # Create a dictionary to store product code and sitc_code
experiment = 'rca_tests'

def load_data(csv_name, trade_form, target_value, code_system, sitc_code, ts_type):
    '''
        Ingests SITC multi-time series trade data in csv format, filters for one product code, removes zero values for target value, screens target values and renames target columns & creates dataframes for the target columns

    Parameters:
    -----
    csv:
    trade_form:
    target_value:
    code_system:
    sitc_code:
    ts_type:
    '''

    data = pd.read_csv(csv_name) #, header=None
    # TO DO: Create for loop to read sitc code from dictionary
    dframe = data.loc[data[code_system]==sitc_code]
    dframe.rename(columns={'location_name_short_en': 'exporter', 'export_rca': 'rca'}, inplace=True)
    dframe = dframe[[ts_type, trade_form, target_value]]
    dframe = dframe[dframe[trade_form] != 'Undeclared Countries']
    dframe[target_value].replace('', 0, inplace=True)
    dframe = dframe[dframe[target_value] != 0]
    dframe = dframe[dframe[target_value] >= 0.09]
    dframe = dframe[dframe[target_value] <= 20]
    return dframe
```

```
[101]: xdf = load_data('sitc4digit_year.csv', 'exporter', 'rca', 'sitc_product_code', 7132, 'year')
xdf.columns
```

```
[101]: Index(['year', 'exporter', 'rca'], dtype='object')
```

```
[102]: exporters = xdf['exporter'].nunique()
time = xdf['year'].nunique()
unknown = xdf['exporter'].loc[xdf['exporter']=='Undeclared Countries'].count()
```

```
ctotal = xdf.exporter.count()
zerorca = xdf['rca'][xdf['rca']==0].count()
nulls = xdf['rca'].isnull().sum().sum()
```

```
[103]: print(f'Shape of the dataframe: {xdf.shape}')
print(f'The data covers {time} years from {xdf.year.min()} to {xdf.year.max()} for {exporters} countries')
print(f'The rca values range from {xdf.rca.min()} to {xdf.rca.max()}'')
print(f'Countries not declared are {unknown} out of {ctotal}')
print(f'Zero export values: {zerorca}')
print(f'Null export values: {nulls}')
```

```
Shape of the dataframe: (1939, 3)
The data covers 56 years from 1962 to 2017 for 188 countries
The rca values range from 0.09004033 to 17.575996
Countries not declared are 0 out of 1939
Zero export values: 0
Null export values: 0
```

3 Data Preprocessing and Visualization

Converting to datetime format:

```
[104]: xdf['year'] = pd.to_datetime(xdf['year'], format='%Y')
xdf.columns
```

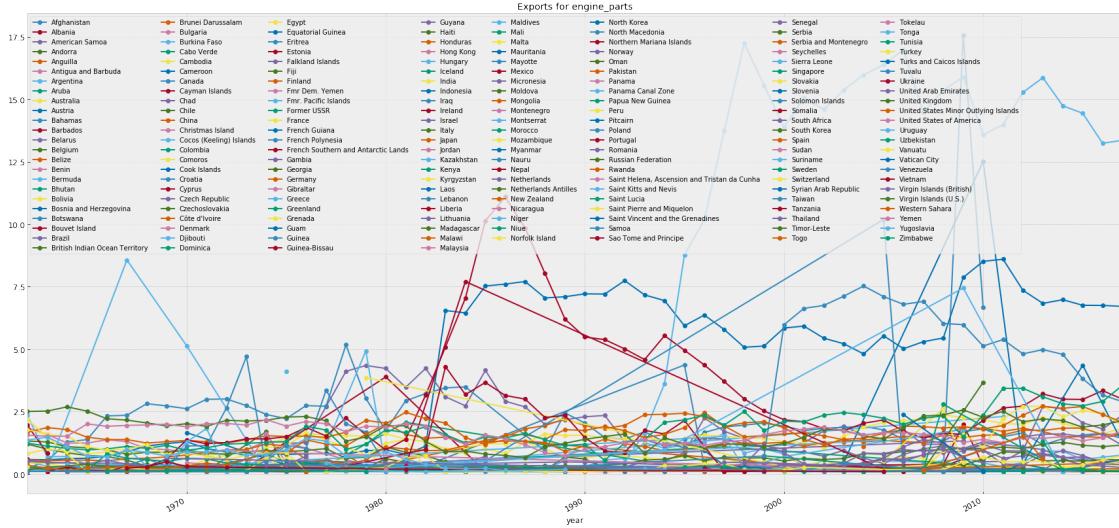
```
[104]: Index(['year', 'exporter', 'rca'], dtype='object')
```

```
[105]: #xdf = xdf.drop(columns=["rca"])
#xdf.columns
```

```
[106]: #sns.scatterplot(x='year',y='rca',data=xdf)
#sns.pairplot(xdf)
os.getcwd()
```

```
[106]: '/home/webber/Documents/PhD/time_series_trade'
```

```
[107]: fig, ax = plt.subplots(figsize=(30,15))
for key, grp in xdf.groupby(['exporter']):
    ax = grp.plot(ax=ax, kind='line', x='year', y='rca', label=key, marker='o', title='Exports for {}'.format(product))
    ax.legend(ncol=8, loc='best')
    target = f'images/{product}/{experiment}'
    savefile = os.path.join(os.getcwd(), target, f'Exports for {product}')
plt.savefig(savefile)
```



```
[108]: xdf.describe(include='all')
```

```
[108]:
```

	year	exporter	rca
count	1939	1939	1939.000000
unique	56	188	NaN
top	1975-01-01 00:00:00	Spain	NaN
freq	54	56	NaN
first	1962-01-01 00:00:00	NaN	NaN
last	2017-01-01 00:00:00	NaN	NaN
mean	NaN	NaN	1.193243
std	NaN	NaN	2.062163
min	NaN	NaN	0.090040
25%	NaN	NaN	0.175712
50%	NaN	NaN	0.513809
75%	NaN	NaN	1.408146
max	NaN	NaN	17.575996

A majority of countries have rca values below the mean value of 0.636 a few countries have once-off or intermittently high values while most values are zero. These countries were excluded from the dataset and treated as outliers by limiting rca values to below 20. An example is Norfolk Islands shown below

```
[109]: rdf = xdf.set_index('year')
#Norfolk = pd.DataFrame(rdf['rca'].loc[rdf['exporter']=='Norfolk Island'])
#Norfolk.to_csv('norfolk.csv')
norfolk = pd.read_csv('norfolk.csv')
norfolk.head()
```

```
[109]:
```

	year	rca
0	1990-01-01	114.697680

```
1 2001-01-01    0.125032
2 2005-01-01    0.526648
3 2008-01-01    2.614124
4 2009-01-01    0.350387
```

```
[110]: df_grps = xdf.set_index('year')
df_grps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1939 entries, 2010-01-01 to 1973-01-01
Data columns (total 2 columns):
exporter    1939 non-null object
rca         1939 non-null float64
dtypes: float64(1), object(1)
memory usage: 45.4+ KB
```

```
[129]: df_grps = xdf.groupby([xdf.year.name, xdf.exporter.name]).mean().unstack()
df_grps = df_grps.apply(pd.to_numeric, errors='coerce').fillna(0, ↴
    downcast='infer')
df_grps.columns = df_grps.columns.droplevel()
df_grps.head()
```

```
exporter Afghanistan Albania American Samoa Andorra Anguilla Antigua and
Barbuda Argentina Aruba Australia Austria Bahamas Barbados Belarus
Belgium Belize Benin Bermuda Bhutan Bolivia Bosnia and Herzegovina
Botswana Bouvet Island Brazil British Indian Ocean Territory Brunei
Darussalam Bulgaria Burkina Faso Cabo Verde Cambodia Cameroon Canada
Cayman Islands Chad Chile China Christmas Island Cocos (Keeling) Islands
Colombia Comoros Cook Islands Croatia Cyprus Czech Republic
Czechoslovakia Côte d'Ivoire Denmark Djibouti Dominica Egypt Equatorial
Guinea Eritrea Estonia Falkland Islands Fiji Finland Fmr Dem. Yemen
Fmr. Pacific Islands Former USSR France French Guiana French Polynesia
French Southern and Antarctic Lands Gambia Georgia Germany Gibraltar
Greece Greenland Grenada Guam Guinea Guinea-Bissau Guyana Haiti
Honduras Hong Kong Hungary Iceland India Indonesia Iraq Ireland
Israel Italy Japan Jordan Kazakhstan Kenya Kyrgyzstan Laos
Lebanon Liberia Lithuania Madagascar Malawi Malaysia Maldives Mali
Malta Mauritania Mayotte Mexico Micronesia Moldova Mongolia Montenegro
Montserrat Morocco Mozambique Myanmar Nauru Nepal Netherlands
Netherlands Antilles New Zealand Nicaragua Niger Niue Norfolk Island
North Korea North Macedonia Northern Mariana Islands Norway Oman
Pakistan Panama Panama Canal Zone Papua New Guinea Peru Pitcairn
Poland Portugal Romania Russian Federation Rwanda Saint Helena, Ascension
and Tristan da Cunha Saint Kitts and Nevis Saint Lucia Saint Pierre and
Miquelon Saint Vincent and the Grenadines Samoa Sao Tome and Principe
Senegal Serbia Serbia and Montenegro Seychelles Sierra Leone Singapore
Slovakia Slovenia Solomon Islands Somalia South Africa South Korea
```

Spain Sudan Suriname Sweden Switzerland Syrian Arab Republic Taiwan
 Tanzania Thailand Timor-Leste Togo Tokelau Tonga Tunisia Turkey Turks
 and Caicos Islands Tuvalu Ukraine United Arab Emirates United Kingdom
 United States Minor Outlying Islands United States of America Uruguay
 Uzbekistan Vanuatu Vatican City Venezuela Vietnam Virgin Islands (British)
 Virgin Islands (U.S.) Western Sahara Yemen Yugoslavia Zimbabwe

year

1962-01-01	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.000000	0.580496	0.148060	0.0	0.0
0.473824	0.0	0.000000	2.348061	0.0	0.0		0.0
0.0	0.0	0.0			0.0		0.197875
0.000000	0.263002		0.0	0.0	0.208462	0.230185	0.0
0.0	0.0	0.0		0.0		0.0	0.0
0.0	0.0	0.000000		0.0	1.192845	0.173982	1.848899
0.000000	0.0	0.0		0.0	0.0	0.0	0.0
0.110766	0.000000		0.290738		0.0	0.127338	0.820721
0.0	0.0				0.0	0.0	0.0
1.674000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.148012	0.502856	0.0	0.141510	0.0
0.108328	0.0	1.321232	0.569255	0.000000	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.000000	0.456691	0.000000		0.0	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.602013	0.491603
0.0	0.0	0.0	0.000000	0.0		0.0	0.0
0.0		0.0	0.554667	0.107453	0.000000	0.120572	
0.135594		0.0	0.0	0.0	0.000000	0.000000	0.0
0.0	0.0				0.0		0.0
0.0		2.183242				0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.000000	
0.265236	0.0	0.0		0.0	1.950224		0.0
0.157027	0.350309	0.0	0.0	1.234533	1.777193		0.0
0.100061	0.0	0.0		0.0	0.0	0.0	0.000000
0.0		0.0	0.0	0.0		0.0	
2.501614			0.0			1.556420	
0.0	0.0	0.280693		0.0	0.0	0.0	
0.0		0.0		0.0	0.0	1.354641	0.0
1963-01-01		0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0	0.158385	0.567719	0.112449	0.0	0.0
0.439567	0.0	0.180027	0.688449	0.0	0.0		0.0
0.0	0.0	0.0			0.0		0.000000
0.096400	0.000000		0.0	0.0	0.000000	0.515274	0.0
0.0	0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.090395		0.0	1.000416	0.000000	1.581009
0.170049	0.0	0.0		0.0	0.0	0.0	0.0
0.000000	0.136433	0.000000			0.0	0.189882	0.992000
0.0		0.0			0.0	0.0	0.0
1.856449	0.0	0.0	0.0	0.0	0.0	0.000000	0.0

0.0	0.0	0.0	0.168285	0.300612	0.0	0.137130	0.0	0.0
0.000000	0.0	1.280610	0.538054	0.000000	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0
0.000000	0.420222	0.178182	0.0	0.000000	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.476203	0.0
0.0	0.0	0.0	0.288871	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.468919	0.167411	0.000000	0.129274	0.0	0.0
0.000000	0.0	0.0	0.0	0.000000	0.101756	0.0	0.0	0.0
0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0
0.245206	0.0	0.0	0.0	0.0	0.837886	0.0	0.0	0.0
0.262759	0.235374	0.0	0.0	1.117450	1.586234	0.0	0.0	0.0
0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2.519554	0.0	0.0	0.0	0.0	1.511864	0.0	0.0	0.0
0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.474863	0.0	0.0	0.0
1964-01-01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.173532	0.613761	0.000000	0.0	0.0	0.0
0.458269	0.0	0.157290	0.702798	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.276851	0.0	0.0
0.109635	0.108313	0.0	0.0	0.000000	1.282494	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.194742	0.0	0.846037	0.000000	1.237742	0.0	0.0
0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000	0.096689	0.000000	0.0	0.0	0.205255	0.922372	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.775067	0.0	0.0	0.0	0.0	0.0	0.095132	0.0	0.0
0.0	0.0	0.0	0.118344	0.287945	0.0	0.157611	0.0	0.0
0.000000	0.0	1.093685	0.554424	0.000000	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.105220	0.0	0.0
0.000000	0.271898	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.434068	0.0
0.0	0.0	0.0	0.166111	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.443444	0.000000	0.136051	0.000000	0.0	0.0
0.130246	0.0	0.0	0.0	0.284105	0.274675	0.0	0.0	0.0
0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
0.0	0.091854	0.0	0.000000	0.0	0.0	0.386595	0.0	0.0
0.114891	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0
0.000000	0.136077	0.0	0.0	1.024864	1.506083	0.0	0.0	0.0
0.102161	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2.688499	0.0	0.0	0.0	0.0	1.513119	0.0	0.0	0.0
0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.000000	0.0	1.095493	0.0	0.0	0.0

1965-01-01		0.0	0.0		0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.186220	0.630304	0.093531		0.0	0.0
0.406301		0.0	0.229610	0.379873	0.0	0.0		0.0
0.0		0.0	0.0			0.0		0.224930
0.175400		0.000000		0.0	0.0	0.000000	1.378049	0.0
0.0	0.0	0.0		0.0			0.0	0.0
0.0	0.0	0.000000		0.0	0.740515		0.000000	1.239821
0.000000		0.0	0.0		0.0	0.0	0.0	0.0
0.000000	0.104563		0.166414			0.0	0.239299	0.797259
0.0		0.0				0.0	0.0	0.0
1.471533		0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0		0.0	0.000000	0.242567	0.0	0.179250	0.0
0.000000		0.0	1.048762	0.555957	0.440842		0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0	0.164524	0.0
0.097703	0.140666		0.000000	0.0	0.000000		0.0	0.0
0.0	0.0		0.0	0.0	0.0	0.0	0.000000	0.427324
0.0	0.0		0.0	0.000000	0.0		0.0	0.0
0.0			0.0	0.399739	0.000000	0.000000	0.161863	
0.416909			0.0	0.0	0.0	0.281856	0.137746	0.0
0.0	0.0					0.0		0.0
0.0		0.097421					0.0	0.0
0.0	0.000000	0.0			0.0	0.0	0.000000	
0.153407		0.0	0.0		0.0	0.000000		0.0
0.383884	0.245488		0.0	0.0	0.964977	1.095849		0.0
0.000000		0.0	0.0		0.0	0.0	0.0	0.000000
0.0		0.0	0.0		0.0			0.0
2.517870				0.0			2.004086	
0.0	0.0	0.000000		0.0	0.0	0.0		
0.0		0.0		0.0	0.0	0.716946		0.0
1966-01-01		0.0	0.0		0.0	0.0	0.0	
0.0	0.0	0.0	0.184314	0.709252	0.000000		0.0	0.0
0.332594		0.0	0.000000	0.000000	0.0	0.0		0.0
0.0	0.0	0.0				0.0		0.113258
0.090633		0.094281		0.0	0.0	0.000000	2.324608	0.0
0.0	0.0	0.0		0.0			0.0	0.0
0.0	0.0	0.000000		0.0	0.698982		0.000000	1.368511
0.000000		0.0	0.0		0.0	0.0	0.0	0.0
0.000000	0.144323		0.000000			0.0	0.258296	0.883454
0.0		0.0				0.0	0.0	0.0
1.378909		0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0		0.0	0.000000	0.154819	0.0	0.125569	0.0
0.000000		0.0	0.965186	0.524825	0.193478		0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.0	0.132050	0.0
0.000000	0.181963		0.000000	0.0	0.094698		0.0	0.0
0.0	0.0		0.0	0.0	0.0	0.000000	0.433133	
0.0	0.0		0.0	0.000000	0.0		0.0	0.0
0.0		0.0	0.555139	0.000000	0.109164	0.000000		

0.275759		0.0	0.0	0.0	0.294532	0.123265	0.0	
0.0	0.0				0.0			0.0
0.0		0.000000				0.0	0.0	
0.0	0.000000	0.0		0.0	0.0	0.0	0.000000	
0.799839	0.0	0.0		0.0	0.000000		0.0	
0.291808	0.248212	0.0	0.0	1.003457	0.952820			0.0
0.103350	0.0	0.0		0.0	0.0	0.0	0.0	0.118707
0.0		0.0	0.0	0.0			0.0	
2.211097				0.0			1.909151	
0.0	0.0	0.000000		0.0	0.0	0.0		
0.0		0.0		0.0	0.0	0.687152		0.0

```
[130]: df_grps = df_grps.rename(columns={'Fmr. Pacific Islands': 'Pacific Islands'})
df_grps['Russia'] = df_grps['Former USSR']+df_grps['Russian Federation']
df_grps['Yemen'] = df_grps['Fmr Dem. Yemen']+df_grps['Yemen']
df_grps.head()
```

Afghanistan Albania American Samoa Andorra Anguilla Antigua and Barbuda Argentina Aruba Australia Austria Bahamas Barbados Belarus Belgium Belize Benin Bermuda Bhutan Bolivia Bosnia and Herzegovina Botswana Bouvet Island Brazil British Indian Ocean Territory Brunei Darussalam Bulgaria Burkina Faso Cabo Verde Cambodia Cameroon Canada Cayman Islands Chad Chile China Christmas Island Cocos (Keeling) Islands Colombia Comoros Cook Islands Croatia Cyprus Czech Republic Czechoslovakia Côte d'Ivoire Denmark Djibouti Dominica Egypt Equatorial Guinea Eritrea Estonia Falkland Islands Fiji Finland Fmr Dem. Yemen Pacific Islands Former USSR France French Guiana French Polynesia French Southern and Antarctic Lands Gambia Georgia Germany Gibraltar Greece Greenland Grenada Guam Guinea Guinea-Bissau Guyana Haiti Honduras Hong Kong Hungary Iceland India Indonesia Iraq Ireland Israel Italy Japan Jordan Kazakhstan Kenya Kyrgyzstan Laos Lebanon Liberia Lithuania Madagascar Malawi Malaysia Maldives Mali Malta Mauritania Mayotte Mexico Micronesia Moldova Mongolia Montenegro Montserrat Morocco Mozambique Myanmar Nauru Nepal Netherlands Netherlands Antilles New Zealand Nicaragua Niger Niue Norfolk Island North Korea North Macedonia Northern Mariana Islands Norway Oman Pakistan Panama Panama Canal Zone Papua New Guinea Peru Pitcairn Poland Portugal Romania Russian Federation Rwanda Saint Helena, Ascension and Tristan da Cunha Saint Kitts and Nevis Saint Lucia Saint Pierre and Miquelon Saint Vincent and the Grenadines Samoa Sao Tome and Principe Senegal Serbia Serbia and Montenegro Seychelles Sierra Leone Singapore Slovakia Slovenia Solomon Islands Somalia South Africa South Korea Spain Sudan Suriname Sweden Switzerland Syrian Arab Republic Taiwan Tanzania Thailand Timor-Leste Togo Tokelau Tonga Tunisia Turkey Turks and Caicos Islands Tuvalu Ukraine United Arab Emirates United Kingdom United States Minor Outlying Islands United States of America Uruguay Uzbekistan Vanuatu Vatican City Venezuela Vietnam Virgin Islands (British)

	Virgin Islands (U.S.)	Western Sahara	Yemen	Yugoslavia	Zimbabwe	Russia
year						
1962-01-01	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.000000	0.580496	0.148060	0.0
0.473824	0.0	0.000000	2.348061	0.0	0.0	0.0
0.0	0.0	0.0			0.0	0.197875
0.000000	0.263002	0.0	0.0	0.208462	0.230185	0.0
0.0	0.0	0.0	0.0		0.0	0.0
0.0	0.0	0.000000	0.0	1.192845	0.173982	1.848899
0.000000	0.0	0.0	0.0	0.0	0.0	0.0
0.110766	0.000000	0.290738		0.0	0.127338	0.820721
0.0	0.0			0.0	0.0	0.0
1.674000	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.148012	0.502856	0.0	0.141510
0.108328	0.0	1.321232	0.569255	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.000000	0.456691	0.000000	0.0	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.602013	0.491603
0.0	0.0	0.0	0.000000	0.0	0.0	0.0
0.0	0.0	0.0	0.554667	0.107453	0.000000	0.120572
0.135594		0.0	0.0	0.0	0.000000	0.0
0.0	0.0			0.0		0.0
0.0		2.183242			0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.000000
0.265236	0.0	0.0		0.0	1.950224	0.0
0.157027	0.350309	0.0	0.0	1.234533	1.777193	0.0
0.100061	0.0	0.0		0.0	0.0	0.000000
0.0	0.0	0.0	0.0	0.0		0.0
2.501614			0.0			1.556420
0.0	0.0	0.280693	0.0	0.0	0.0	
0.0		0.0		0.0	0.290738	1.354641
0.127338						
1963-01-01	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.158385	0.567719	0.112449	0.0
0.439567	0.0	0.180027	0.688449	0.0	0.0	0.0
0.0	0.0	0.0			0.0	0.000000
0.096400	0.000000	0.0	0.0	0.000000	0.515274	0.0
0.0	0.0	0.0	0.0		0.0	0.0
0.0	0.0	0.090395	0.0	1.000416	0.000000	1.581009
0.170049	0.0	0.0		0.0	0.0	0.0
0.000000	0.136433	0.000000		0.0	0.189882	0.992000
0.0	0.0	0.0		0.0	0.0	0.0
1.856449	0.0	0.0	0.0	0.0	0.0	0.000000
0.0	0.0	0.0	0.168285	0.300612	0.0	0.137130
0.000000	0.0	1.280610	0.538054	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.000000	0.420222	0.178182	0.0	0.000000	0.0	0.0

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.476203	
0.0	0.0	0.0	0.288871	0.0	0.0	0.0	0.0	0.0	
0.0		0.0	0.468919	0.167411	0.000000	0.129274			
0.000000		0.0	0.0	0.0	0.000000	0.101756	0.0		
0.0	0.0				0.0			0.0	
0.0		0.000000				0.0	0.0		
0.0	0.000000	0.0		0.0	0.0	0.0	0.000000		
0.245206	0.0	0.0		0.0	0.837886		0.0		
0.262759	0.235374	0.0	0.0	1.117450	1.586234			0.0	
0.000000	0.0	0.0		0.0	0.0	0.0	0.000000		
0.0		0.0	0.0	0.0	0.0		0.0		
2.519554			0.0			1.511864			
0.0	0.0	0.000000		0.0	0.0	0.0			
0.0		0.0		0.0	0.000000	1.474863	0.0		
0.189882									
1964-01-01		0.0	0.0		0.0	0.0	0.0		
0.0	0.0	0.0	0.173532	0.613761	0.000000		0.0	0.0	
0.458269	0.0	0.157290	0.702798		0.0	0.0		0.0	
0.0	0.0	0.0				0.0		0.276851	
0.109635	0.108313		0.0	0.0	0.000000	1.282494		0.0	
0.0	0.0	0.0		0.0		0.0	0.0	0.0	
0.0	0.0	0.194742		0.0	0.846037		0.000000	1.237742	
0.000000	0.0	0.0		0.0	0.0	0.0		0.0	
0.000000	0.096689		0.000000		0.0	0.205255	0.922372		
0.0		0.0			0.0	0.0	0.0		
1.775067	0.0	0.0	0.0	0.0	0.0	0.095132		0.0	
0.0	0.0	0.0	0.118344	0.287945		0.0	0.157611	0.0	0.0
0.000000	0.0	1.093685	0.554424	0.000000		0.0	0.0		0.0
0.0	0.0	0.0	0.0		0.0	0.105220		0.0	
0.000000	0.271898		0.000000	0.0	0.000000		0.0	0.0	0.0
0.0	0.0	0.0		0.0	0.0	0.000000		0.434068	
0.0	0.0		0.0	0.166111	0.0		0.0	0.0	
0.0			0.0	0.443444	0.000000	0.136051	0.000000		
0.130246		0.0	0.0	0.0	0.284105	0.274675	0.0		
0.0	0.0				0.0			0.0	
0.0		0.000000				0.0	0.0		
0.0	0.091854	0.0		0.0	0.0	0.0	0.386595		
0.114891	0.0	0.0		0.0	0.000000		0.0		
0.000000	0.136077	0.0	0.0	1.024864	1.506083			0.0	
0.102161	0.0	0.0		0.0	0.0	0.0	0.0	0.000000	
0.0		0.0	0.0	0.0	0.0		0.0		
2.688499			0.0			1.513119			
0.0	0.0	0.000000		0.0	0.0	0.0			
0.0		0.0		0.0	0.000000	1.095493	0.0		
0.205255									
1965-01-01		0.0	0.0		0.0	0.0	0.0		
0.0	0.0	0.0	0.186220	0.630304	0.093531		0.0	0.0	

0.406301	0.0	0.229610	0.379873	0.0	0.0		0.0
0.0	0.0	0.0	0.0		0.0	0.0	0.224930
0.175400	0.000000		0.0	0.0	0.000000	1.378049	0.0
0.0	0.0	0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.000000		0.0	0.740515	0.000000	1.239821
0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.000000	0.104563		0.166414		0.0	0.239299	0.797259
0.0	0.0				0.0	0.0	0.0
1.471533	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.000000	0.242567	0.0	0.179250	0.0
0.000000	0.0	1.048762	0.555957	0.440842		0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.164524	0.0
0.097703	0.140666	0.000000		0.0	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.427324
0.0	0.0		0.0	0.000000	0.0	0.0	0.0
0.0			0.0	0.399739	0.000000	0.000000	0.161863
0.416909		0.0	0.0	0.0	0.281856	0.137746	0.0
0.0	0.0				0.0		0.0
0.0			0.097421			0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.000000	
0.153407	0.0	0.0		0.0	0.000000		0.0
0.383884	0.245488	0.0	0.0	0.964977		1.095849	0.0
0.000000	0.0	0.0		0.0	0.0	0.0	0.000000
0.0		0.0	0.0	0.0			0.0
2.517870				0.0			2.004086
0.0	0.0	0.000000		0.0	0.0	0.0	
0.0		0.0		0.0	0.166414	0.716946	0.0
0.239299							
1966-01-01		0.0	0.0		0.0	0.0	0.0
0.0	0.0	0.0	0.184314	0.709252	0.000000	0.0	0.0
0.332594	0.0	0.000000	0.000000		0.0	0.0	
0.0	0.0	0.0				0.0	0.113258
0.090633	0.094281		0.0	0.0	0.000000	2.324608	0.0
0.0	0.0	0.0		0.0		0.0	0.0
0.0	0.0	0.000000		0.0	0.698982	0.000000	1.368511
0.000000	0.0	0.0		0.0	0.0	0.0	0.0
0.000000	0.144323		0.000000		0.0	0.258296	0.883454
0.0		0.0			0.0	0.0	0.0
1.378909	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
0.0	0.0	0.0	0.000000	0.154819	0.0	0.125569	0.0
0.000000	0.0	0.965186	0.524825	0.193478		0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.132050	0.0
0.000000	0.181963		0.000000	0.0	0.094698	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.433133
0.0	0.0		0.0	0.000000	0.0	0.0	0.0
0.0			0.0	0.555139	0.000000	0.109164	0.000000
0.275759		0.0	0.0	0.0	0.294532	0.123265	0.0

```

0.0      0.0          0.0
0.0      0.000000     0.000000
0.0  0.000000     0.0          0.0
0.799839      0.0      0.0          0.0  0.000000
0.291808  0.248212     0.0      0.0  1.003457    0.952820
0.103350      0.0      0.0          0.0  0.0
0.0          0.0      0.0          0.0
2.211097          0.0          0.0
0.0      0.0  0.000000     0.0          0.0
0.0          0.0          0.0  0.000000
0.258296

```

```
[131]: df_grps['Virgin Islands (British)'].describe()
```

```

[131]: count      56.000000
       mean      0.025265
       std       0.114184
       min       0.000000
       25%      0.000000
       50%      0.000000
       75%      0.000000
       max      0.710457
Name: Virgin Islands (British), dtype: float64

```

```

[132]: df_grps = df_grps.drop(['Fmr Dem. Yemen','Norfolk Island','Former
→USSR','Russian Federation','Virgin Islands (U.S.)','Virgin Islands
→(British)'],axis=1)
df_grps.loc[:, df_grps.any()]
#df_grps.T[(df_grps !=0).any()].T
#df.T[(df != 0).any()].T
df_grps.tail()

```

```

[132]: exporter Afghanistan Albania American Samoa Andorra Anguilla Antigua and
Barbuda Argentina Aruba Australia Austria Bahamas Barbados Belarus
Belgium Belize Benin Bermuda Bhutan Bolivia Bosnia and Herzegovina
Botswana Bouvet Island Brazil British Indian Ocean Territory Brunei
Darussalam Bulgaria Burkina Faso Cabo Verde Cambodia Cameroon Canada
Cayman Islands Chad Chile China Christmas Island Cocos (Keeling)
Islands Colombia Comoros Cook Islands Croatia Cyprus Czech Republic
Czechoslovakia Côte d'Ivoire Denmark Djibouti Dominica Egypt Equatorial
Guinea Eritrea Estonia Falkland Islands Fiji Finland Pacific Islands
France French Guiana French Polynesia French Southern and Antarctic Lands
Gambia Georgia Germany Gibraltar Greece Greenland Grenada Guam
Guinea Guinea-Bissau Guyana Haiti Honduras Hong Kong Hungary Iceland
India Indonesia Iraq Ireland Israel Italy Japan Jordan Kazakhstan
Kenya Kyrgyzstan Laos Lebanon Liberia Lithuania Madagascar Malawi
Malaysia Maldives Mali Malta Mauritania Mayotte Mexico Micronesia

```

Moldova Mongolia Montenegro Montserrat Morocco Mozambique Myanmar Nauru
 Nepal Netherlands Netherlands Antilles New Zealand Nicaragua Niger
 Niue North Korea North Macedonia Northern Mariana Islands Norway Oman
 Pakistan Panama Panama Canal Zone Papua New Guinea Peru Pitcairn Poland
 Portugal Romania Rwanda Saint Helena, Ascension and Tristan da Cunha Saint
 Kitts and Nevis Saint Lucia Saint Pierre and Miquelon Saint Vincent and the
 Grenadines Samoa Sao Tome and Principe Senegal Serbia Serbia and Montenegro
 Seychelles Sierra Leone Singapore Slovakia Slovenia Solomon Islands
 Somalia South Africa South Korea Spain Sudan Suriname Sweden
 Switzerland Syrian Arab Republic Taiwan Tanzania Thailand Timor-Leste Togo
 Tokelau Tonga Tunisia Turkey Turks and Caicos Islands Tuvalu Ukraine
 United Arab Emirates United Kingdom United States Minor Outlying Islands
 United States of America Uruguay Uzbekistan Vanuatu Vatican City Venezuela
 Vietnam Western Sahara Yemen Yugoslavia Zimbabwe Russia

year

2013-01-01	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.922299	0.159312	0.239121	6.832908	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.974185			0.0		0.0
0.0	0.0	0.0	0.0	1.165582	0.0	0.0
0.193630		0.0			0.0	0.0
0.0	0.0	0.0	0.714487		0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.0
0.0	0.119975		0.0	2.249058	0.0	0.0
0.202853	0.0	0.0	1.743767		0.0	0.0
0.000000	0.0		0.0	0.0	0.0	0.0
0.0	0.182095	0.204121	0.0	0.0	0.0	0.0
0.0	0.0	0.189121	0.0	0.000000	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.206668	0.0
0.000000	0.000000		0.0	0.0	3.205015	0.1143
0.000000		0.0		0.0	0.0	0.0
0.0	0.0			0.0	0.0	0.0
0.0	0.14196	0.0	0.0	4.980607	0.0	2.712464
0.000000		0.0		0.0		0.0
0.000000	0.0		0.148612	0.0	0.0	0.0
0.0	0.0	0.0	2.714529		0.000000	0.0
0.764177	0.523549	2.707670	0.0	0.0	3.089318	0.0
0.0	0.0	0.0	1.592067		0.0	0.000000
0.523032		0.000000	0.0	0.0		0.0
2.205432			0.0			1.426788
0.0	1.295589	0.0	0.0	0.0	0.0	0.605617
0.0	0.0	0.0	0.124788			
2014-01-01	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.908908	0.000000	0.171495	6.979731	0.0	0.0
0.0	0.0	0.0	0.0	0.0		0.0
0.0	0.654322			0.0		0.0
0.0	0.0	0.0	0.0	1.408975	0.0	0.0
					0.117061	

0.190012		0.0		0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.936517	0.0	0.0	0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.132461	0.0
0.0	0.100854		0.0	2.026040	0.0		0.0
0.153136	0.0	0.0	1.836504	0.0	0.0	0.0	0.000000
0.105583	0.0		0.0	0.0	0.0	0.0	14.744479
0.0	0.294822	0.202422	0.0	0.0	0.0	1.264634	1.497386
0.0	0.0	0.000000	0.0	0.202216	0.0	0.574145	0.0
0.0	0.0	0.0	0.0	0.0	0.000000	2.994922	0.00000
0.000000	0.126584		0.0	0.0	0.0	0.0	0.0
0.095539		0.0		0.0	0.0	0.0	0.148033
0.0		0.0		0.0	0.0	0.0	0.0
0.0	0.000000	0.0	0.0	4.790739	0.0	2.605891	0.0
0.000000		0.0		0.0			0.0
0.000000	0.0		0.217231	0.0	0.0		0.0
0.0	0.0	0.0	2.288254	0.0		0.146686	0.0
0.641001	0.539583	2.629673	0.0	0.0	2.811413		0.0
0.0	0.0	0.0	1.604119	0.0	0.0	0.0	0.111355
0.265322		0.000000	0.0	0.0			0.189593
2.137144				0.0			1.492936
0.0	0.167461	0.0	0.0	0.0	0.0	0.0	0.000000
0.0	0.0	0.0	0.000000				
2015-01-01		0.0	0.0		0.0	0.0	0.0
0.0	0.580215	0.000000	0.140385	6.756086	0.0	0.0	0.552766
0.0	0.0	0.0	0.0	0.0	0.0		0.0
0.0	0.682905			0.0		0.0	0.0
0.0	0.0	0.0	0.0	1.561581		0.0	0.0
0.194272		0.0			0.0	0.0	0.0
0.0	0.0	0.0	0.735749		0.0	0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.000000	0.0
0.0	0.096099		0.0	1.805890	0.0		0.0
0.000000	0.0	0.0	1.836762	0.0	0.0	0.0	0.000000
0.000000	0.0		0.0	0.0	0.0	0.0	14.446037
0.0	0.283549	0.253028	0.0	0.0	0.0	1.147398	1.494958
0.0	0.0	0.000000	0.0	0.093389	0.0	0.369918	0.0
0.0	0.0	0.0	0.0	0.0	0.000000	2.985930	0.00000
0.000000	0.000000		0.0	0.0	0.0	0.0	0.0
0.094815		0.0		0.0	0.0	0.0	0.000000
0.0		0.0			0.0	0.0	0.0
0.0	0.000000	0.0	0.0	3.812788	0.0	2.051538	0.0
0.112618		0.0		0.0			0.0
4.354257	0.0		0.000000	0.0	0.0		0.0
0.0	0.0	0.0	2.544445	0.0		0.000000	0.0
0.307165	0.513409	2.702458	0.0	0.0	2.732172		0.0
0.0	0.0	0.0	1.502105	0.0	0.0	0.0	0.000000
0.373071		0.000000	0.0	0.0			0.0
1.876296			0.0			1.329156	

0.0	0.115978	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
0.0	0.0	0.0	0.0	0.000000				
2016-01-01		0.0	0.0		0.0	0.0	0.0	0.0
0.0	0.672384	0.000000	0.104019	6.748956		0.0	0.0	0.502346
0.0	0.0	0.0	0.0	0.0	0.0		0.0	0.0
0.0	0.499971			0.0			0.0	0.0
0.0	0.0	0.0	0.0	1.628511		0.0	0.0	0.099739
0.203942		0.0			0.0	0.0	0.0	
0.0	0.0	0.0	0.466085		0.0		0.0	0.0
0.0	0.123184	0.0		0.0	0.0	0.000000		0.0
0.0	0.105873		0.0	1.622226		0.0		0.0
0.000000	0.0	0.0	1.569836		0.0	0.0	0.0	0.000000
0.000000	0.0		0.0	0.0	0.0	0.0	0.0	13.246020
0.0	0.298770	0.448373	0.0	0.0	0.0	1.099201	1.463453	0.0
0.0	0.0	0.379580	0.0	0.000000	0.0	0.319781		0.0
0.0	0.0	0.0	0.0	0.0	0.000000	3.335884		0.0000
0.439548	0.000000		0.0	0.0		0.0	0.0	0.0
0.000000		0.0		0.0		0.0	0.0	0.000000
0.0		0.0		0.0		0.0	0.0	0.0
0.0	0.000000	0.0	0.0	3.161107		0.0	1.805244	0.0
0.000000		0.0		0.0			0.0	
0.000000	0.0		0.000000		0.0	0.0		0.0
0.0	0.0	0.0	2.404885		0.0		0.000000	0.0
0.405759	0.570792	2.384861	0.0		0.0	2.903786		0.0
0.0	0.0	0.0	1.601655		0.0	0.0	0.0	0.000000
0.560052		0.000000	0.0	0.0				0.129795
1.965101				0.0			1.449604	
0.0	0.000000	0.0	0.0		0.0	0.0		0.000000
0.0	0.0	0.0	0.000000					
2017-01-01		0.0	0.0		0.0	0.0	0.0	
0.0	0.760461	0.000000	0.000000	6.712914		0.0	0.0	0.318911
0.0	0.0	0.0	0.0	0.0	0.0		0.0	0.0
0.0	0.462394			0.0			0.0	0.0
0.0	0.0	0.0	0.0	1.422437		0.0	0.0	0.102930
0.207034		0.0			0.0	0.0	0.0	
0.0	0.0	0.0	0.568301		0.0		0.0	0.0
0.0	0.000000	0.0		0.0	0.0	0.000000		0.0
0.0	0.120428		0.0	1.814529		0.0		0.0
0.000000	0.0	0.0	1.595636		0.0	0.0	0.0	0.000000
0.000000	0.0		0.0	0.0	0.0	0.0	0.0	13.363432
0.0	0.361842	0.517201	0.0	0.0	0.0	1.153072	1.580102	0.0
0.0	0.0	0.518294	0.0	0.000000	0.0	0.253549		0.0
0.0	0.0	0.0	0.0	0.0	0.000000	2.997151		0.0000
0.000000	0.000000		0.0	0.0		0.0	0.0	0.0
0.105286		0.0		0.0		0.0	0.0	0.000000
0.0		0.0		0.0		0.0	0.0	0.0
0.0	0.000000	0.0	0.0	2.894686		0.0	2.357659	0.0

```

0.000000          0.0          0.0          0.0
1.238149    0.0          0.000000    0.0    0.0          0.0
0.0          0.0          0.0  1.955962    0.0          0.000000    0.0
0.377278    0.567139  2.069166    0.0    0.0  3.549535    0.0
0.0    0.0          0.0  1.143190    0.0    0.0          0.0  0.000000  0.138986
0.739315          0.115158    0.0    0.0          0.0          0.0          0.0
2.133351          0.0          0.0          0.0          0.0          0.0  1.729653
0.0    0.000000    0.0          0.0          0.0          0.0          0.000000
0.0          0.0          0.0  0.092718

```

```
[133]: def remove_dup_cols(frame):
    keep_names = set()
    keep_icols = list()
    for icol, name in enumerate(frame.columns):
        if name not in keep_names:
            keep_names.add(name)
            keep_icols.append(icol)
    return frame.iloc[:, keep_icols]
```

```
[134]: #remove_dup_cols(df_grps)
```

Setting index as the datetime column for easier manipulations:

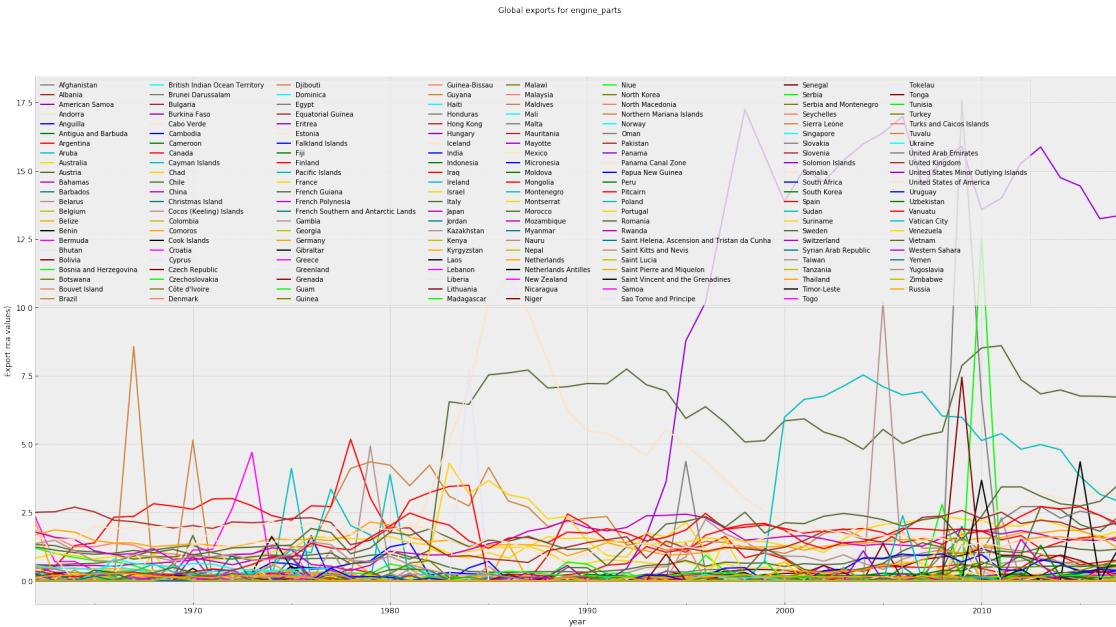
Image storage

```
[135]: location = os.getcwd()
target = f'images/{product}/{experiment}'
path = os.path.join(location, target)
```

```
[136]: #df_pivoted = xdf.
    ↪pivot_table(index='year',columns='exporter',values='export_value',aggfunc='sum')
fig1, ax1 = plt.subplots(figsize=(30,15))
#https://matplotlib.org/3.1.0/gallery/color/named_colors.html
colors = [
    "gray", "brown", "darkviolet", "bisque", "b", "g", "r", "c", "gold", "darkolivegreen", "m", "teal", "r",
    "olive", "salmon", "peru", "aqua"]
label='global trends in export rca values for engine parts'
df_grps.plot(ax=ax1,label=label,color=colors)
ax1.set_ylabel('Export rca values')
fig1.suptitle('Global exports for {}'.format(product))

plt.legend(ncol=8,loc='best')

root = os.path.join(path, 'export rca trends')
plt.savefig(str(root))
#.plot(legend=False)#subplots=True, legend=False)
```



```
[137]: # Adapted from https://stackoverflow.com/questions/30942755/
      ↪plotting-multiple-time-series-after-a-groupby-in-pandas
# Modified from https://www.programcreek.com/python/example/98021/matplotlib.
      ↪dates.YearLocator
from matplotlib.dates import YearLocator, MonthLocator, DateFormatter

def plot_multiple_time_series(dframe, ts_label, grp_label, values_label,
    ↪product, experiment=experiment, figsize=(30,15), title=None):
    """
    Plots multiple time series on one chart by first grouping time series based
    ↪on series names from column in dataframe
    """

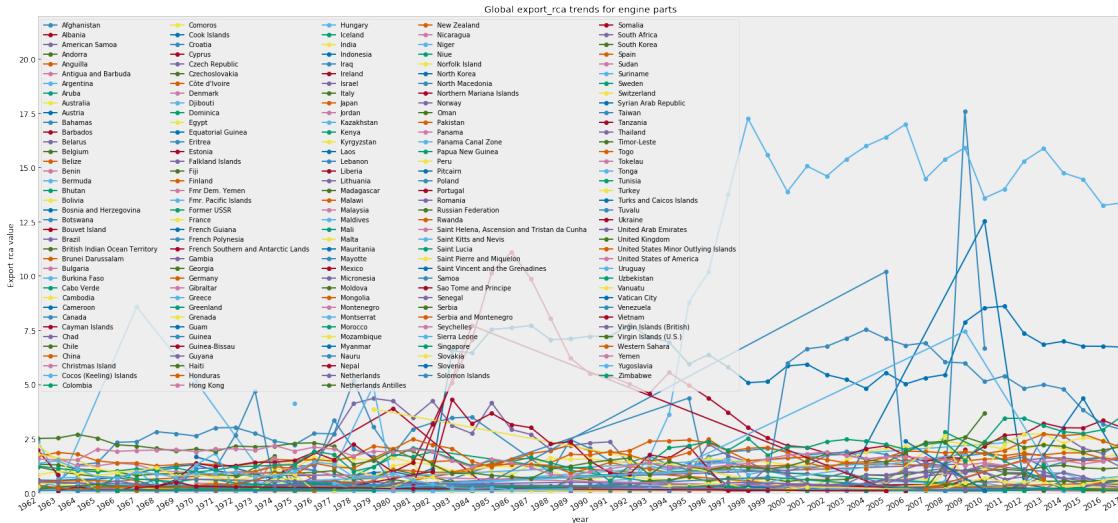
    Parameters:
    -----
    dframe : timeseries Pandas dataframe
    ts_label : string
        The name of the df column that has the datetime timestamp x-axis values.
    grp_label : string
        The column name in dframe for groupby method.
    values_label : string
        The column name in dframe for the y-axis.
    figsize : tuple of two integers
        Figure size of the resulting plot, e.g. (20, 7)
    title : string
        Optional title
    ...
    
```

```

xtick_locator = YearLocator()
xtick_dateformatter = DateFormatter('%Y')
fig, ax = plt.subplots(figsize=figsize)
location = os.getcwd()
target = f'images/{product}/{experiment}'
path = os.path.join(location, target)
for key, grp in dframe.groupby([grp_label]):
    ax = grp.plot(ax=ax, kind='line', x=ts_label, y=values_label, u
    ↪label=key, marker='o')
    ax.xaxis.set_major_locator(xtick_locator)
    ax.xaxis.set_major_formatter(xtick_dateformatter)
    ax.autoscale_view()
    ax.legend(ncol=5, loc='upper left')
    _ = plt.xticks(rotation=0, )
    _ = plt.grid()
    _ = plt.xlabel('year')
    _ = plt.ylabel('Export rca value')
    _ = plt.ylim(0, dframe[values_label].max() * 1.25)
if title is not None:
    _ = plt.title(title)
savefile = os.path.join(path,label)
plt.savefig(str(savefile))
_ = plt.show()

```

[138]: plot_multiple_time_series(xdf, 'year', 'exporter', 'rca', u
↪'engine_parts', experiment, title='Global export_rca trends for engine parts')



Reviewing plots of the density of observations will provide further insight into the structure of the data: - Is the distribution perfectly Gaussian (normal distribution)? - Skewness: which direction is the distribution? - If not Gaussian and skew, transformations will be necessary

prior to modelling

```
[139]: exporters = list(df_grps.columns)
```

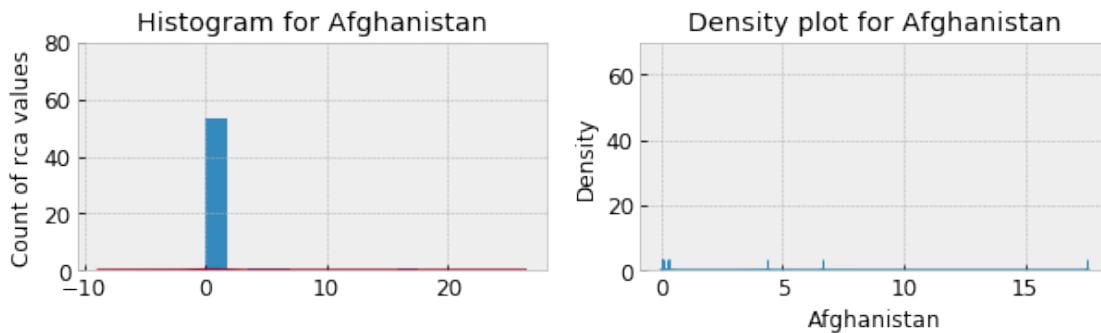
Target folder to store individual images

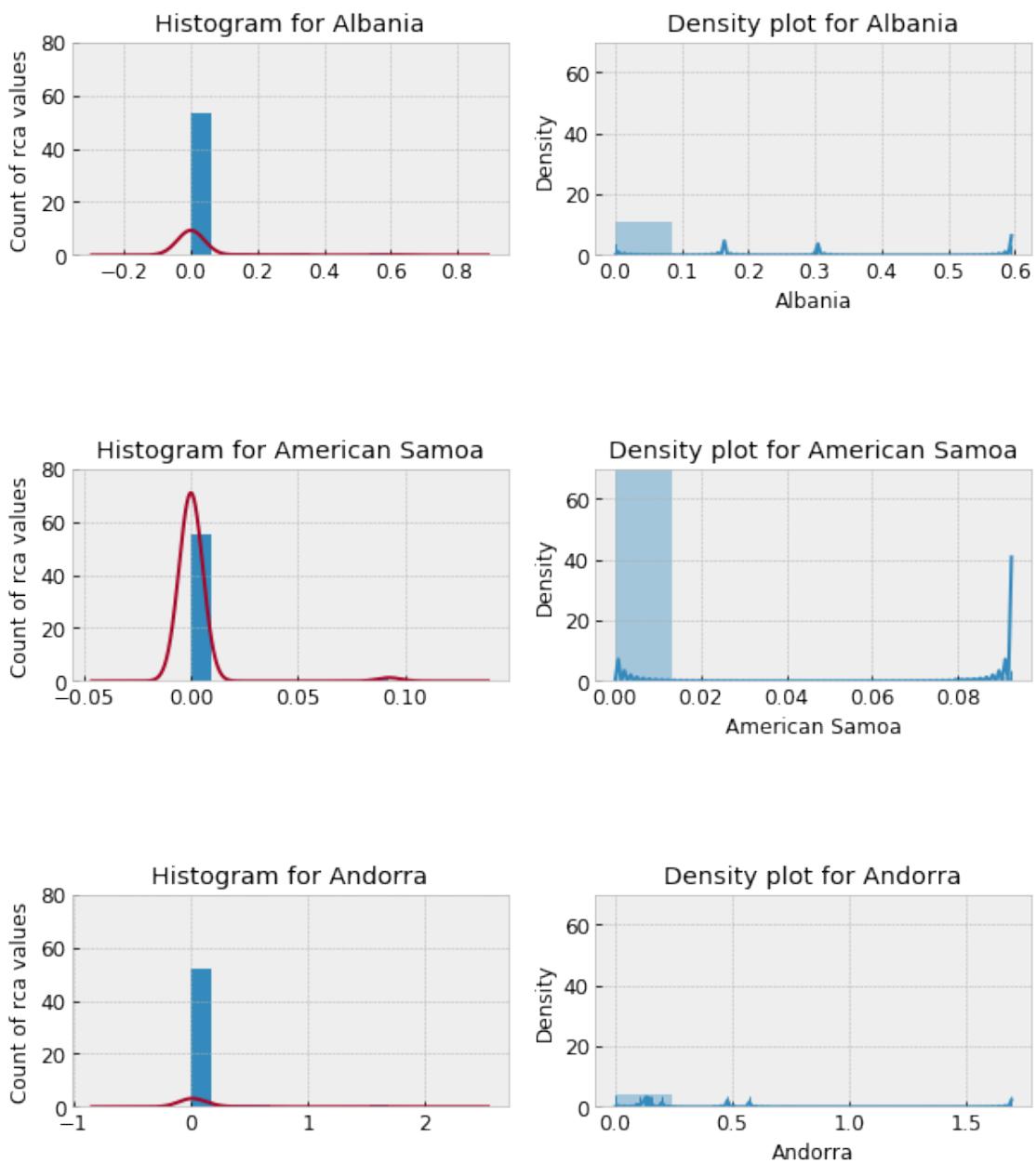
```
[140]: def save_images(ts_groups,analysis):
    """
    analysis: can be from these options
    ->['decompose','acf_pacf','adf_test','rolling_stats']
    """
    path = os.getcwd()+f'/images/{product}/{experiment}/{analysis}'
    savefile = os.path.join(path, ts_groups[exporter].name)
    return savefile
```

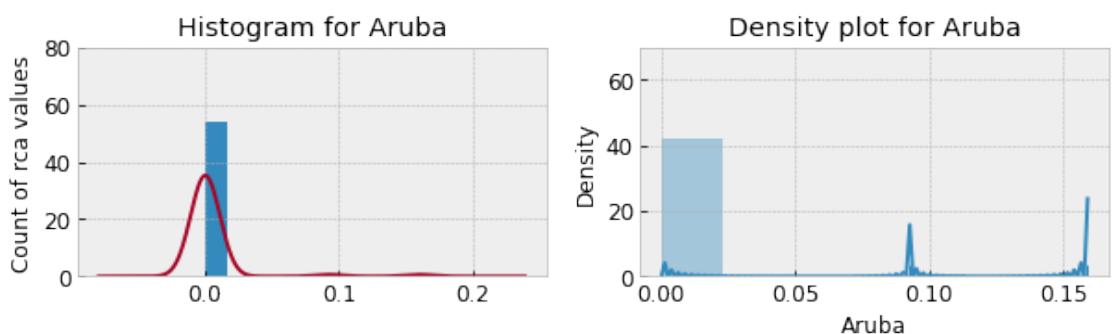
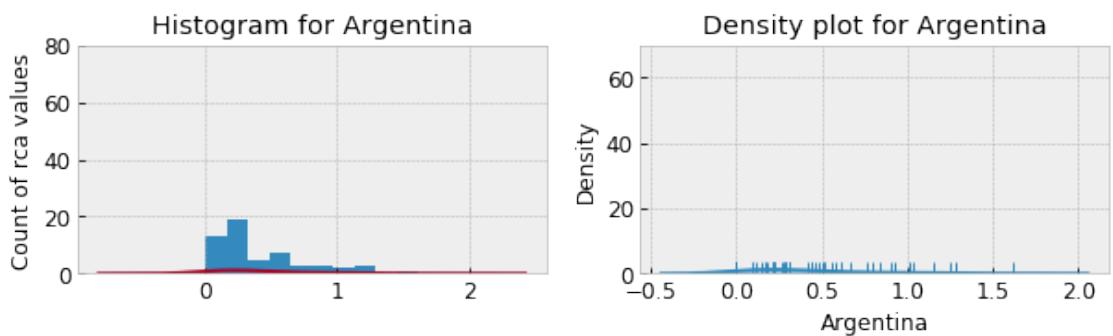
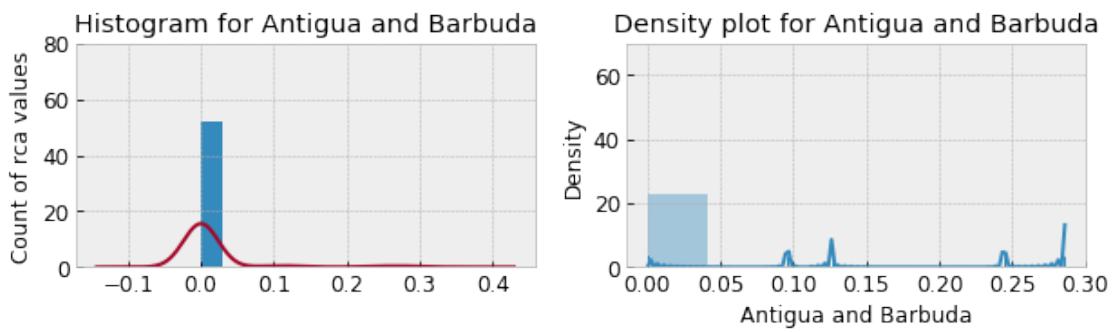
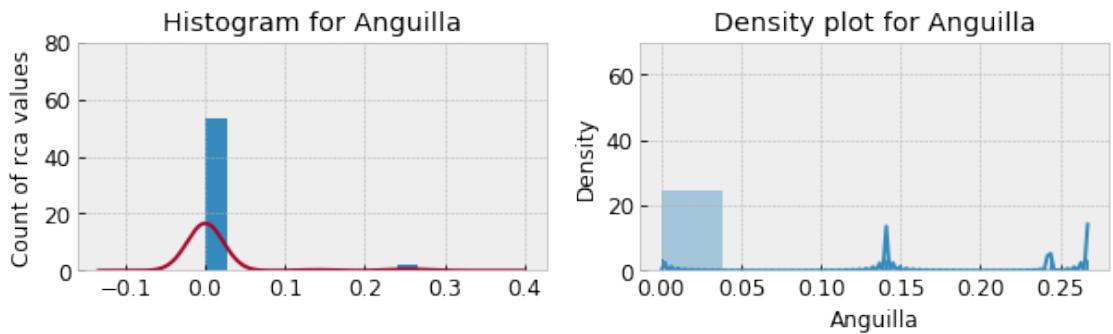
Plot of histograms, kernel density estimation (KDE) and distribution plots to show skewness in data for each country

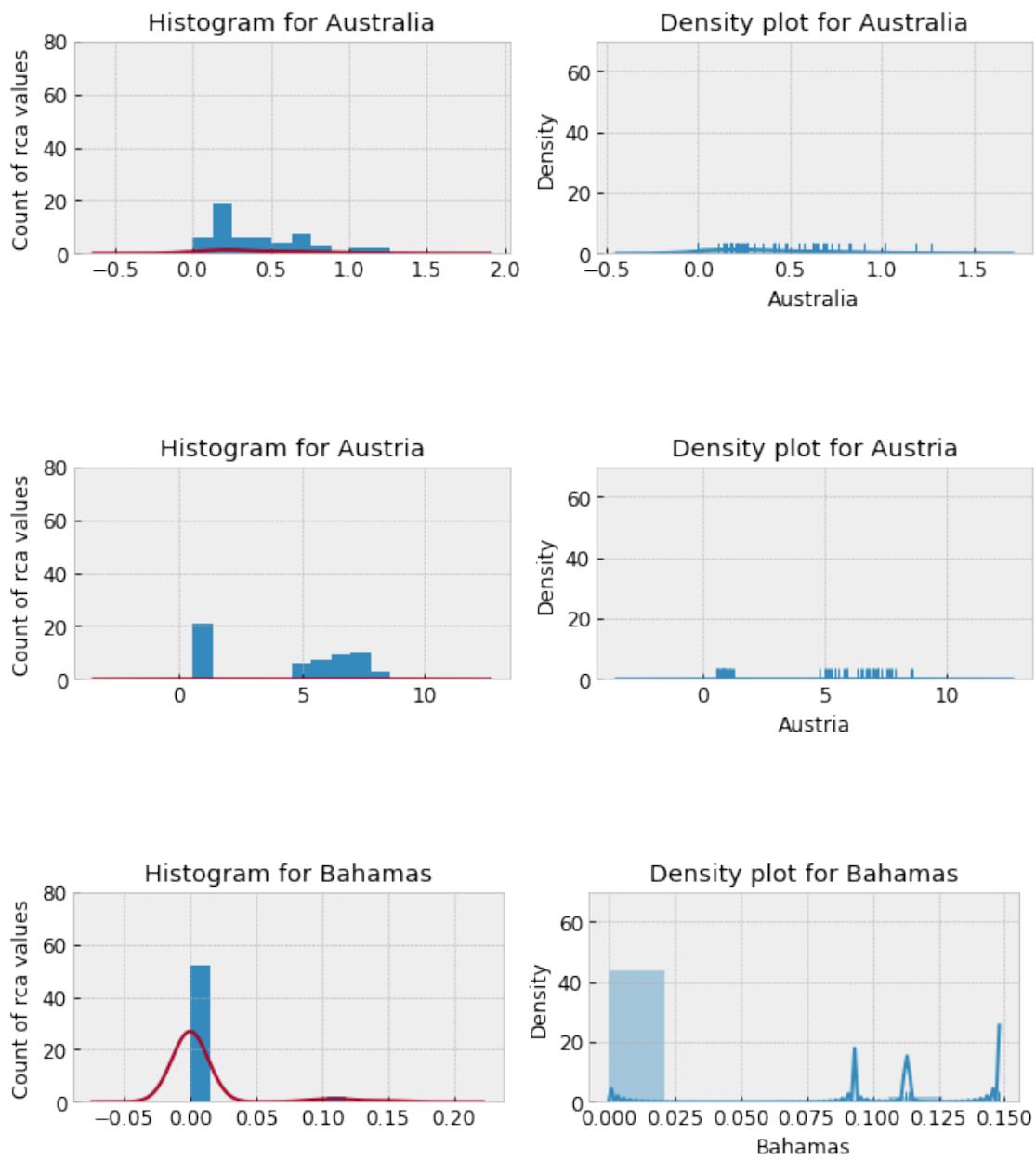
```
[141]: for exporter in exporters:
    root = save_images(df_grps,'rca_histograms')
    plt.figure(figsize=(10,5));
    plt.subplot(221)
    plt.title(f'Histogram for {exporter}')
    df_grps[exporter].plot(kind='hist')
    df_grps[exporter].plot(kind='kde')
    plt.ylabel('Count of rca values')
    #plt.xlim(1962,2016)
    plt.ylim(0,80)

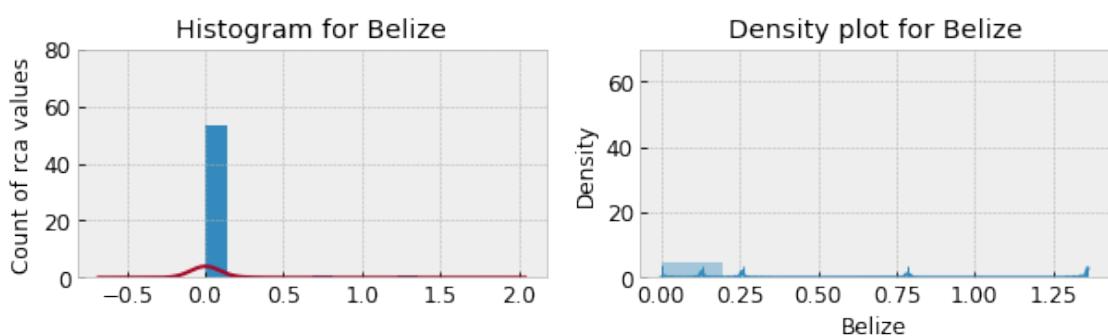
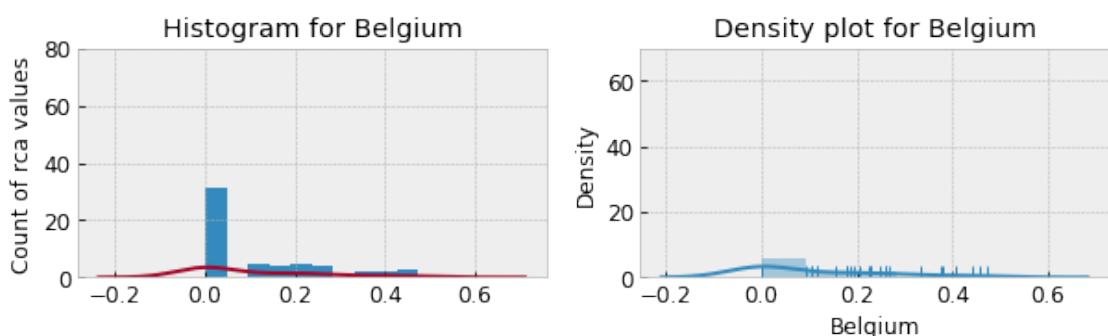
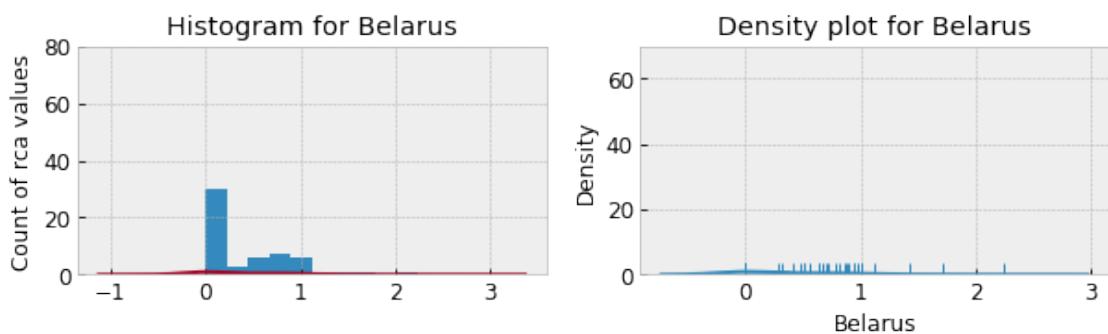
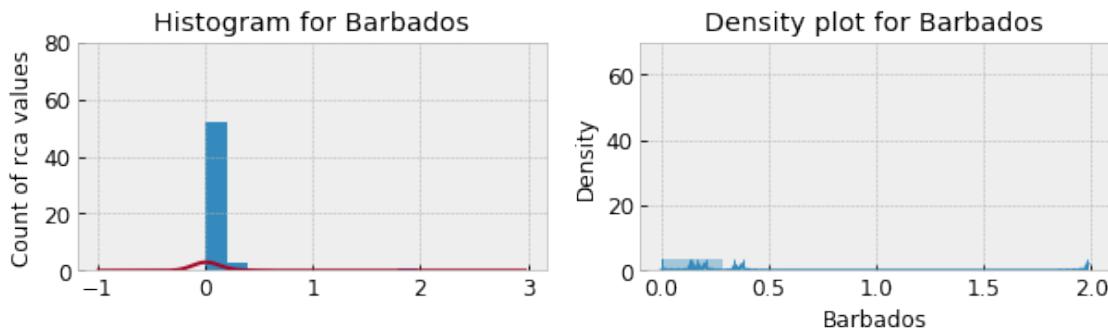
    plt.subplot(222)
    plt.title(f'Density plot for {exporter}')
    sns.distplot(df_grps[exporter],hist=True,rug=True)
    plt.ylabel('Density')
    plt.ylim(0,70)
    plt.savefig(str(root))
    plt.show()
```

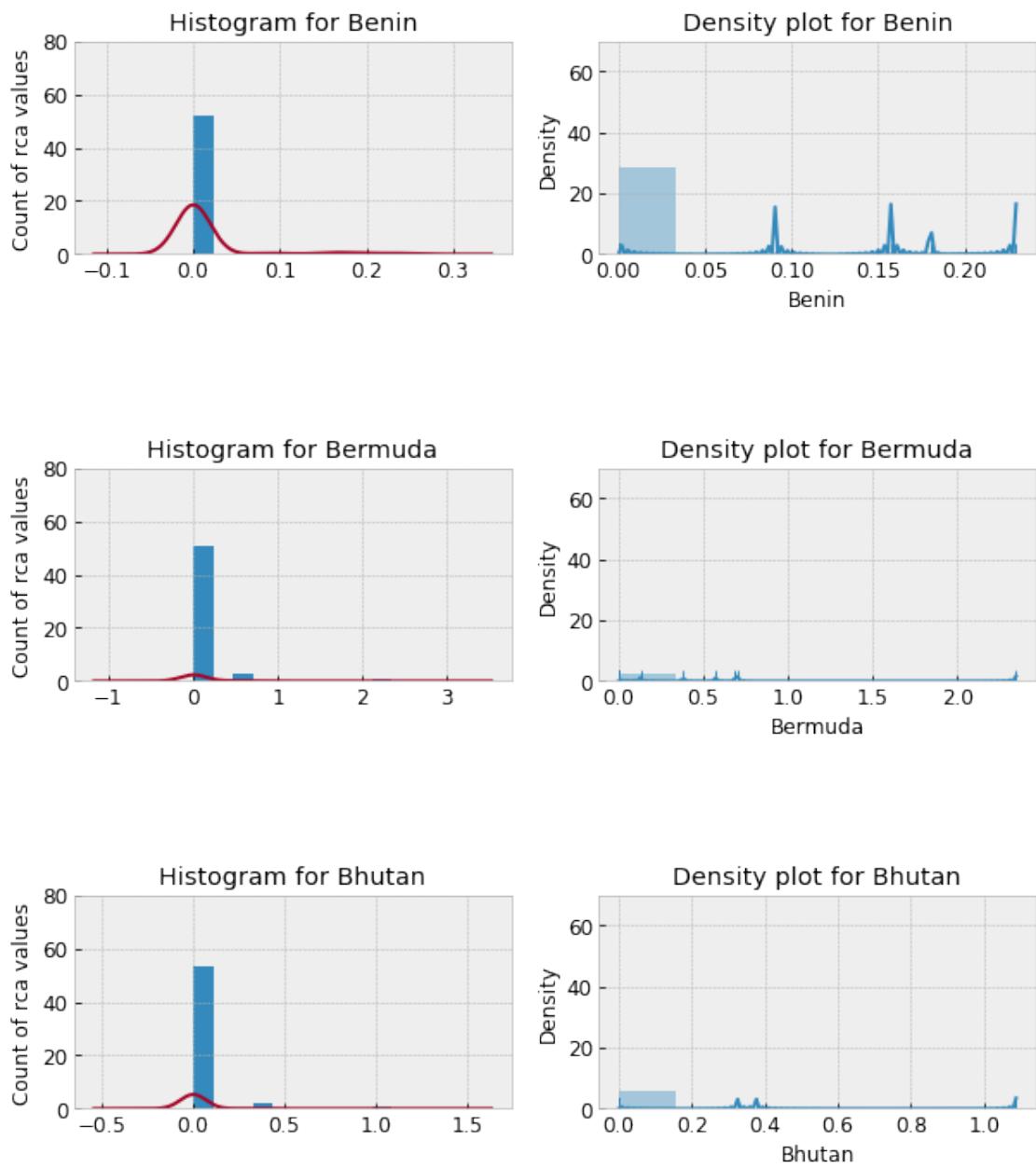


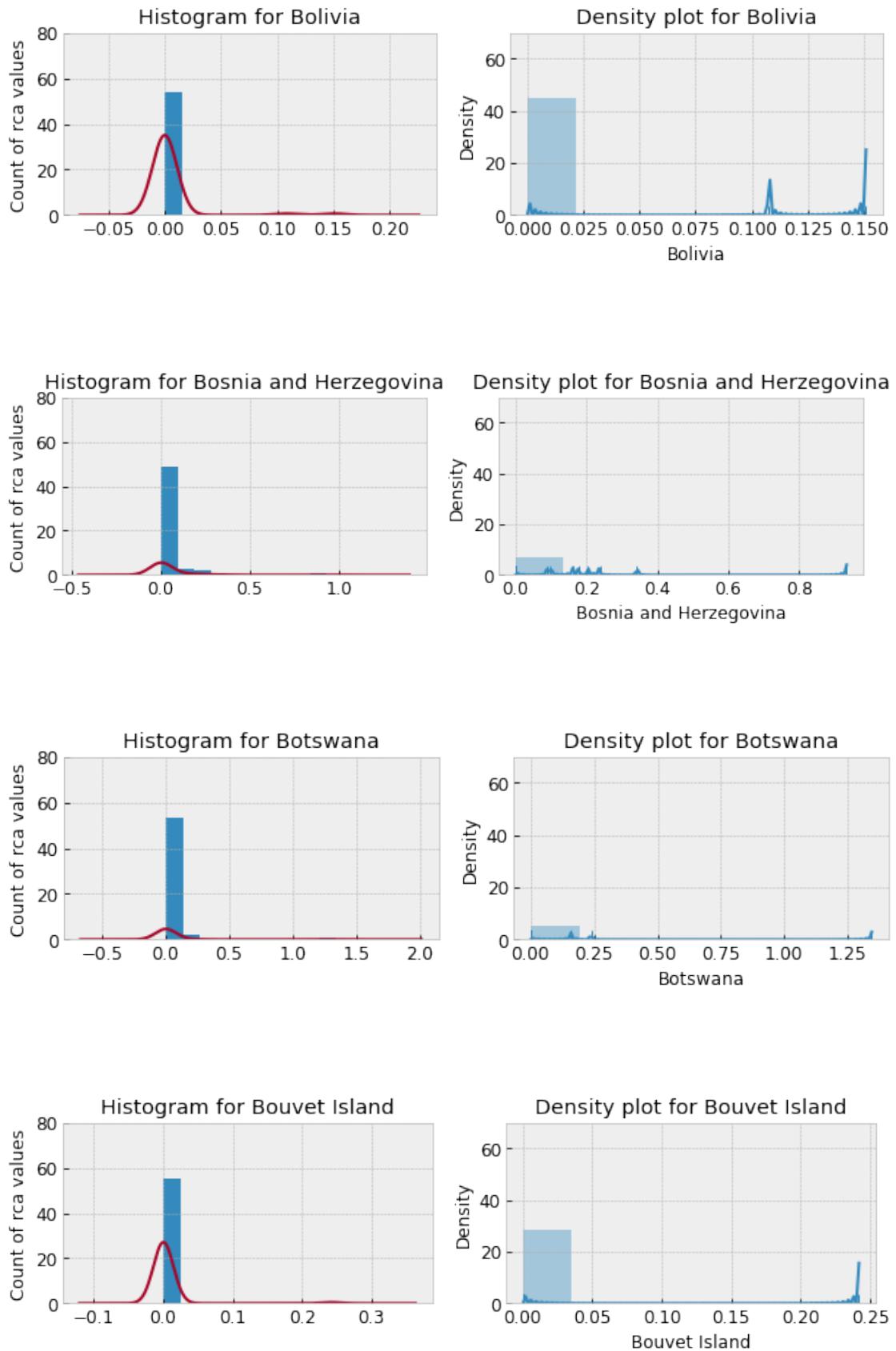


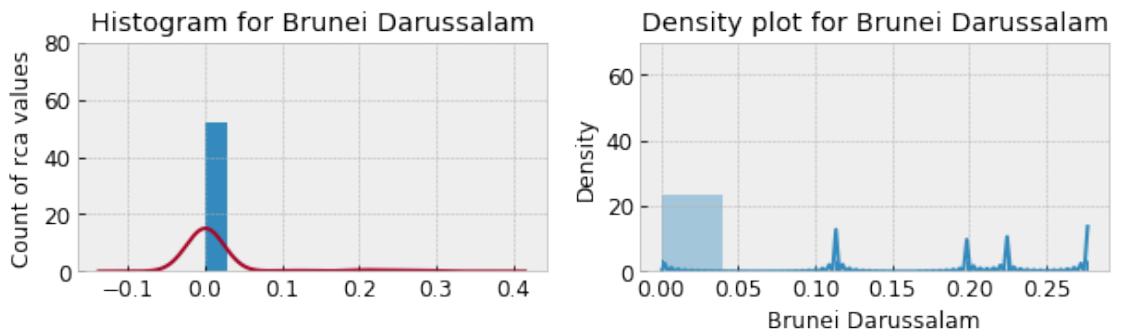
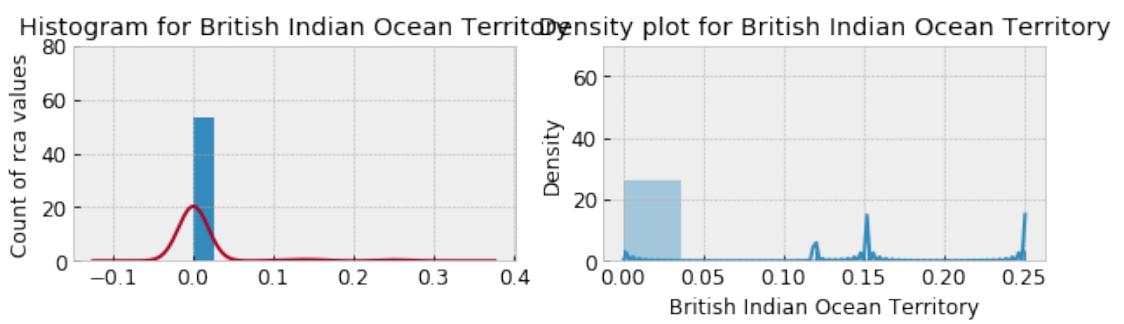
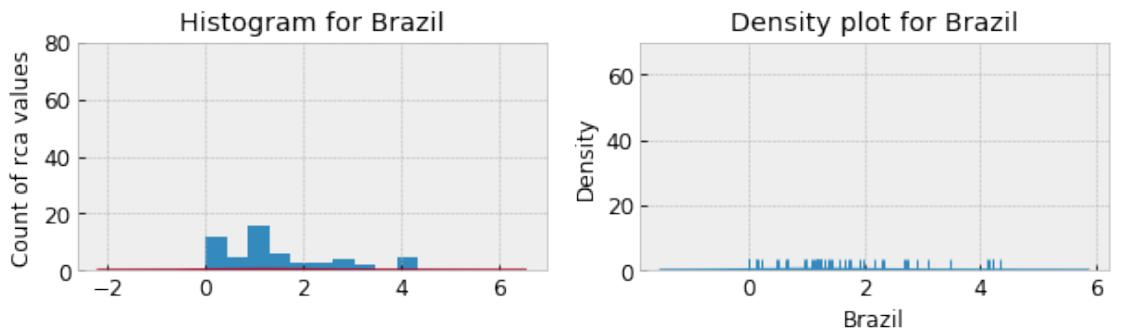


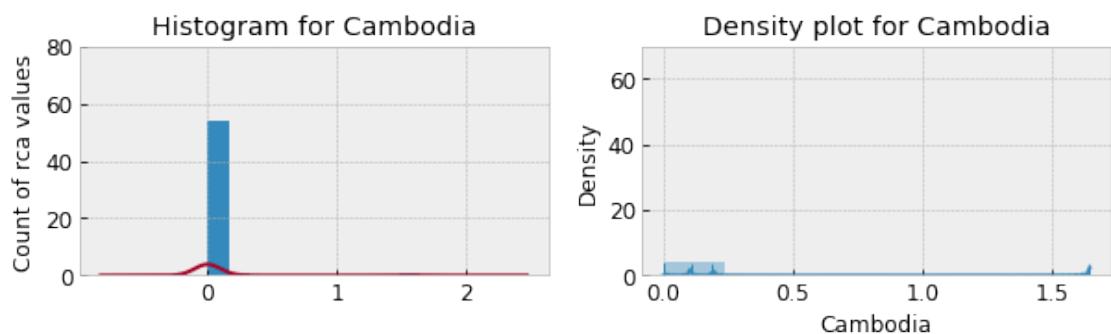
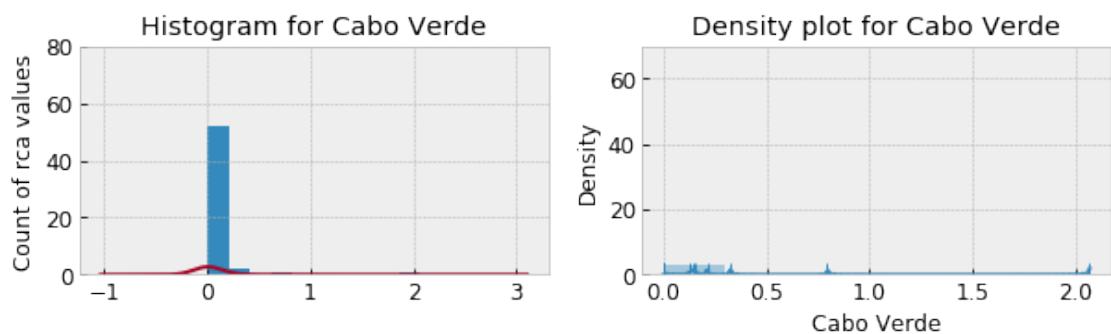
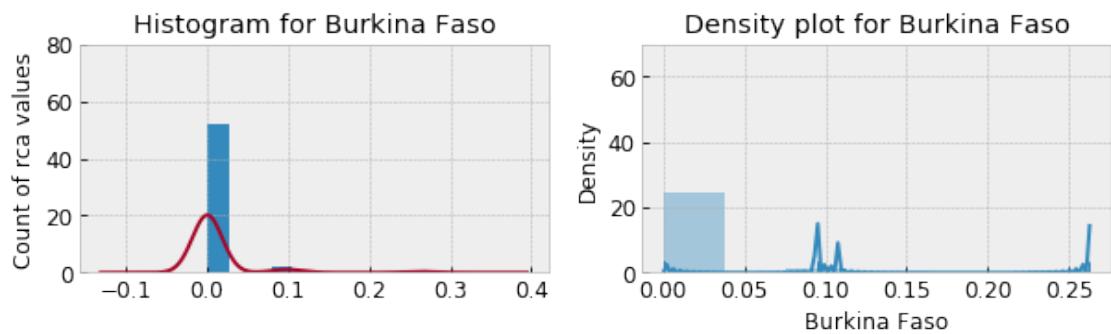
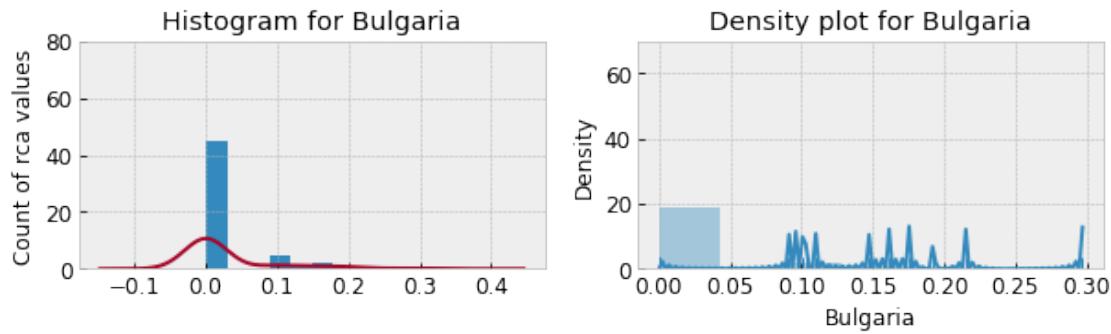


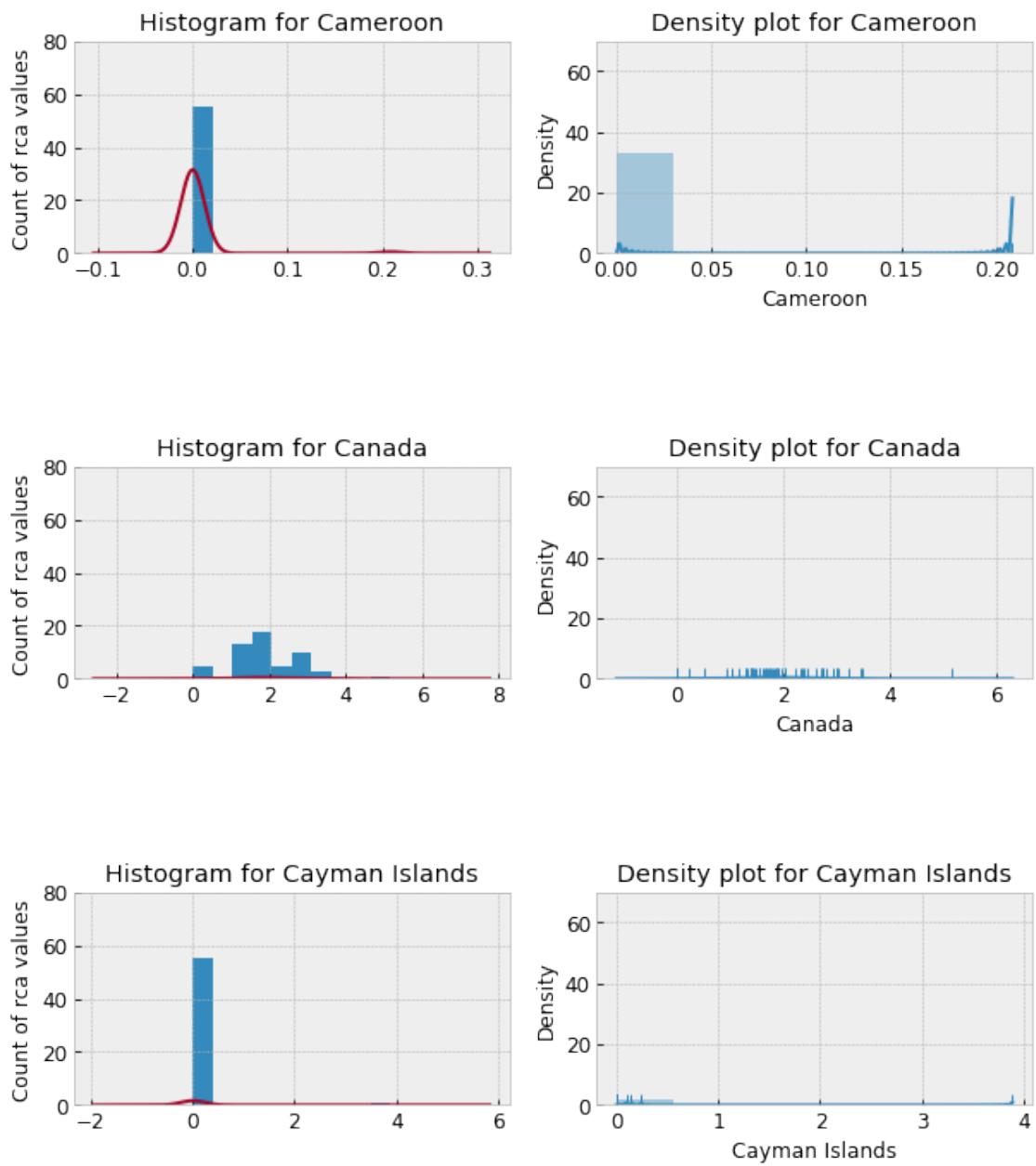


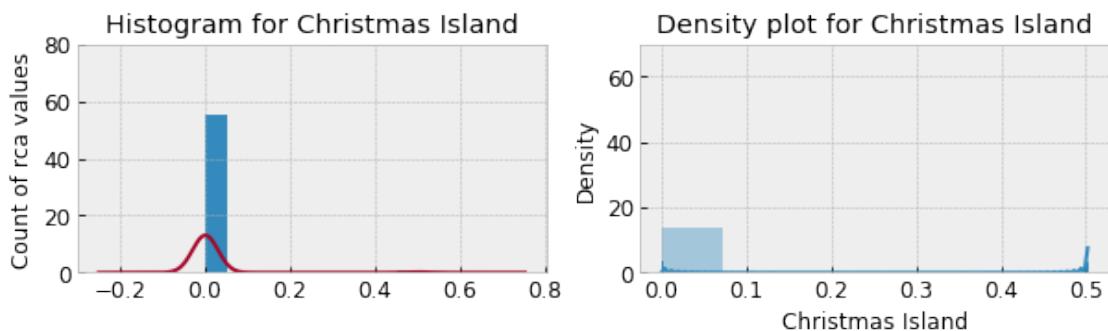
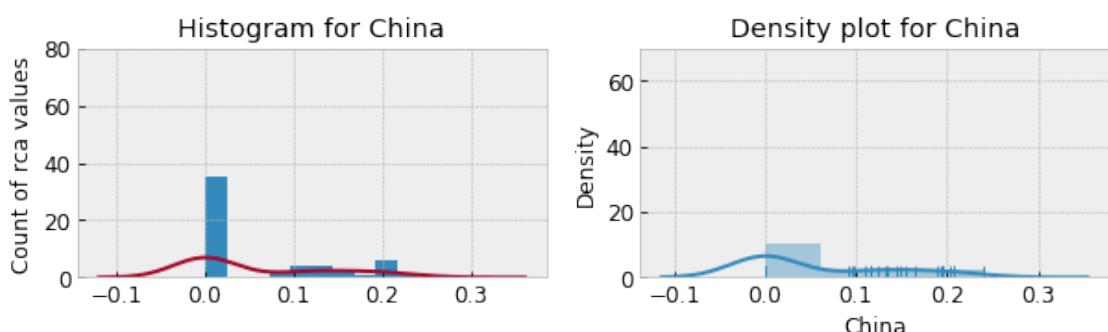
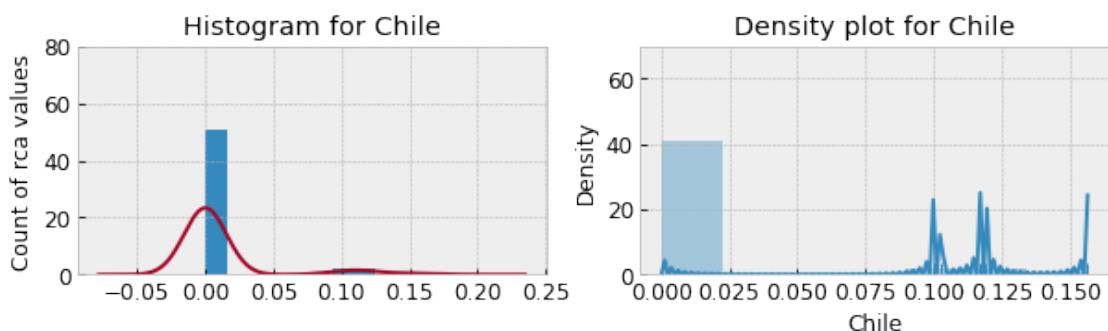
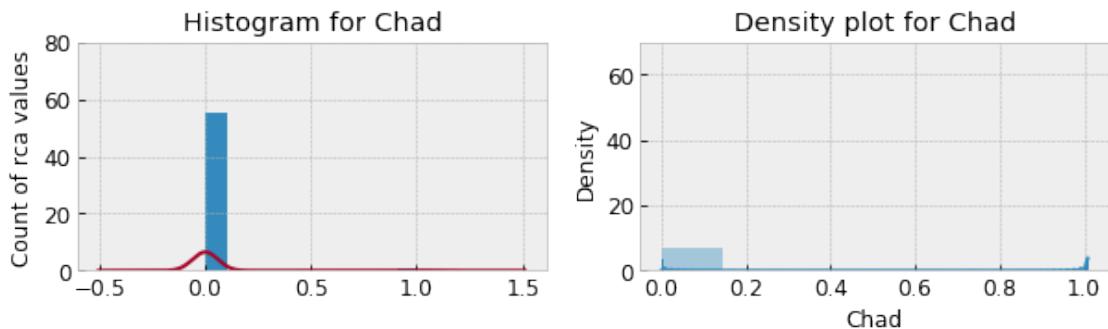


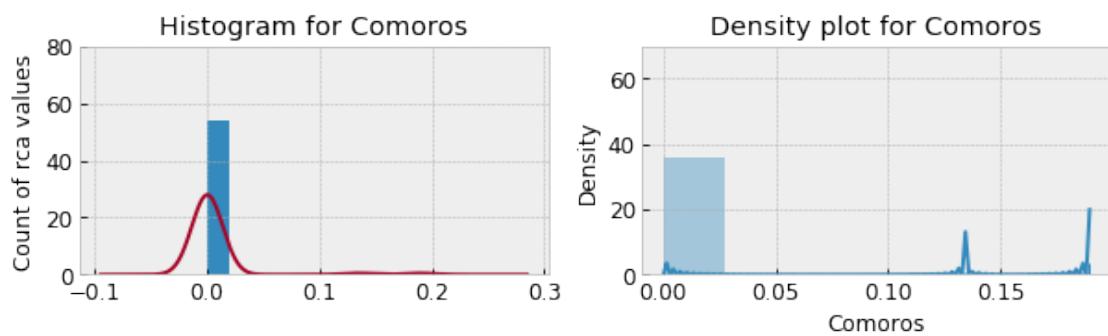
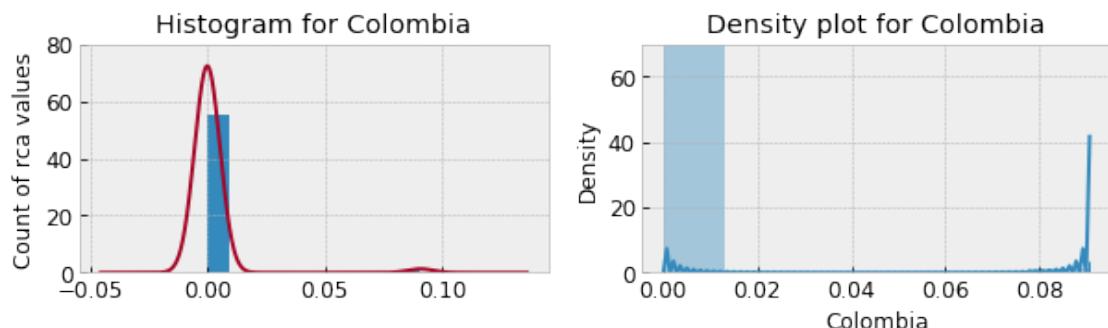
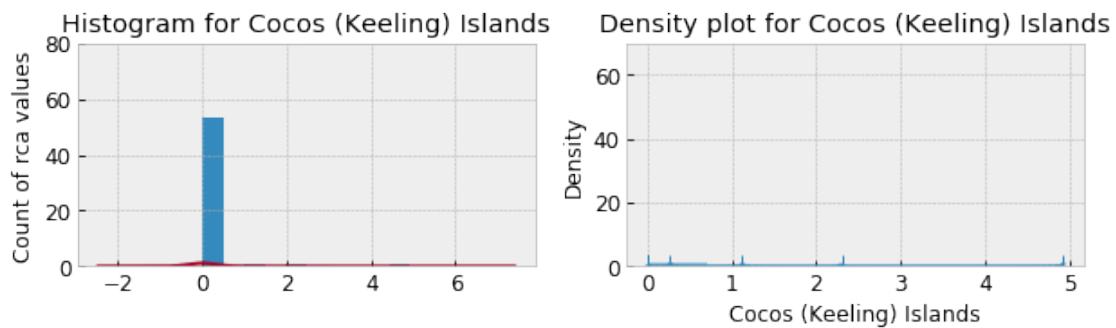


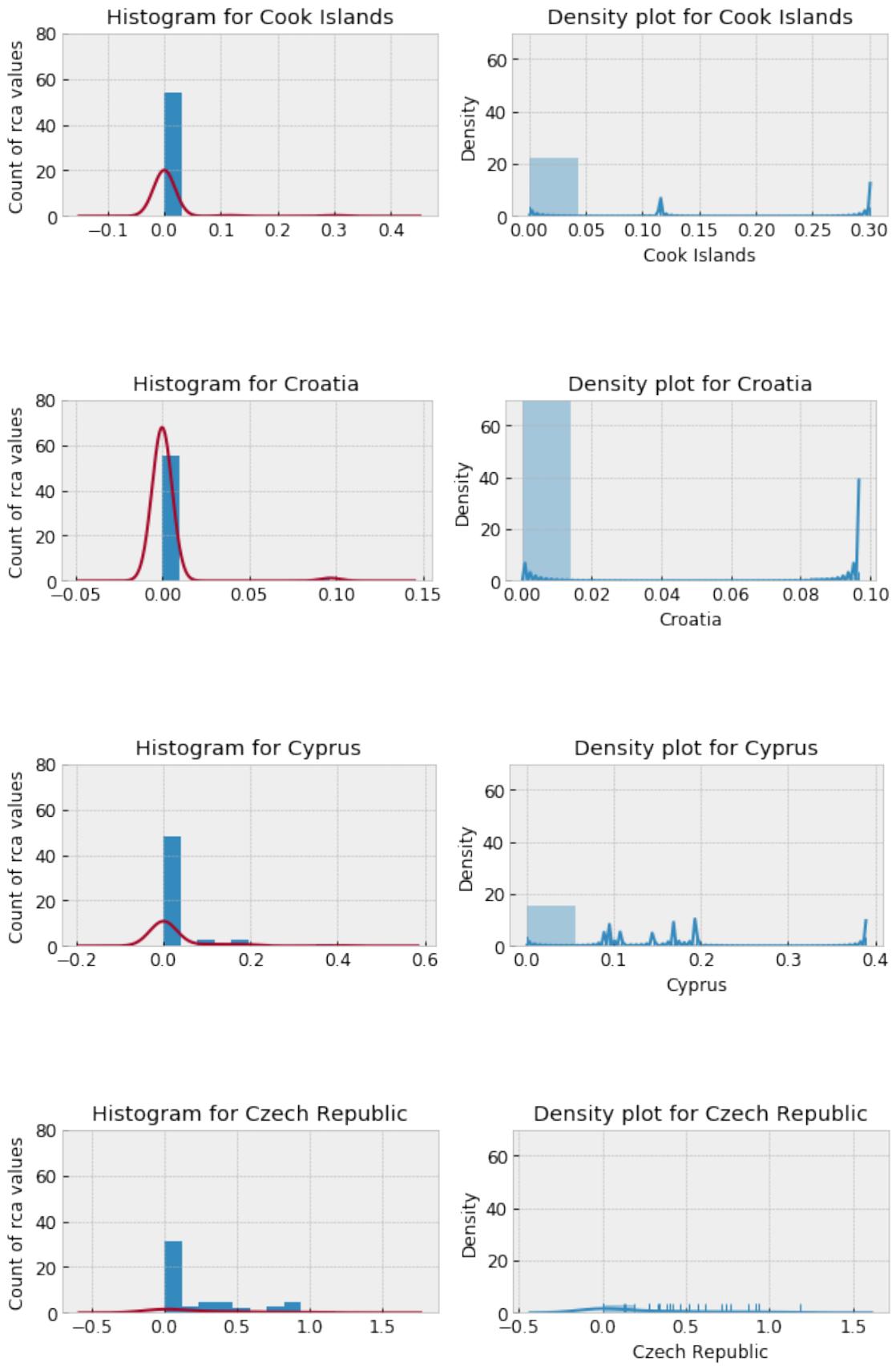


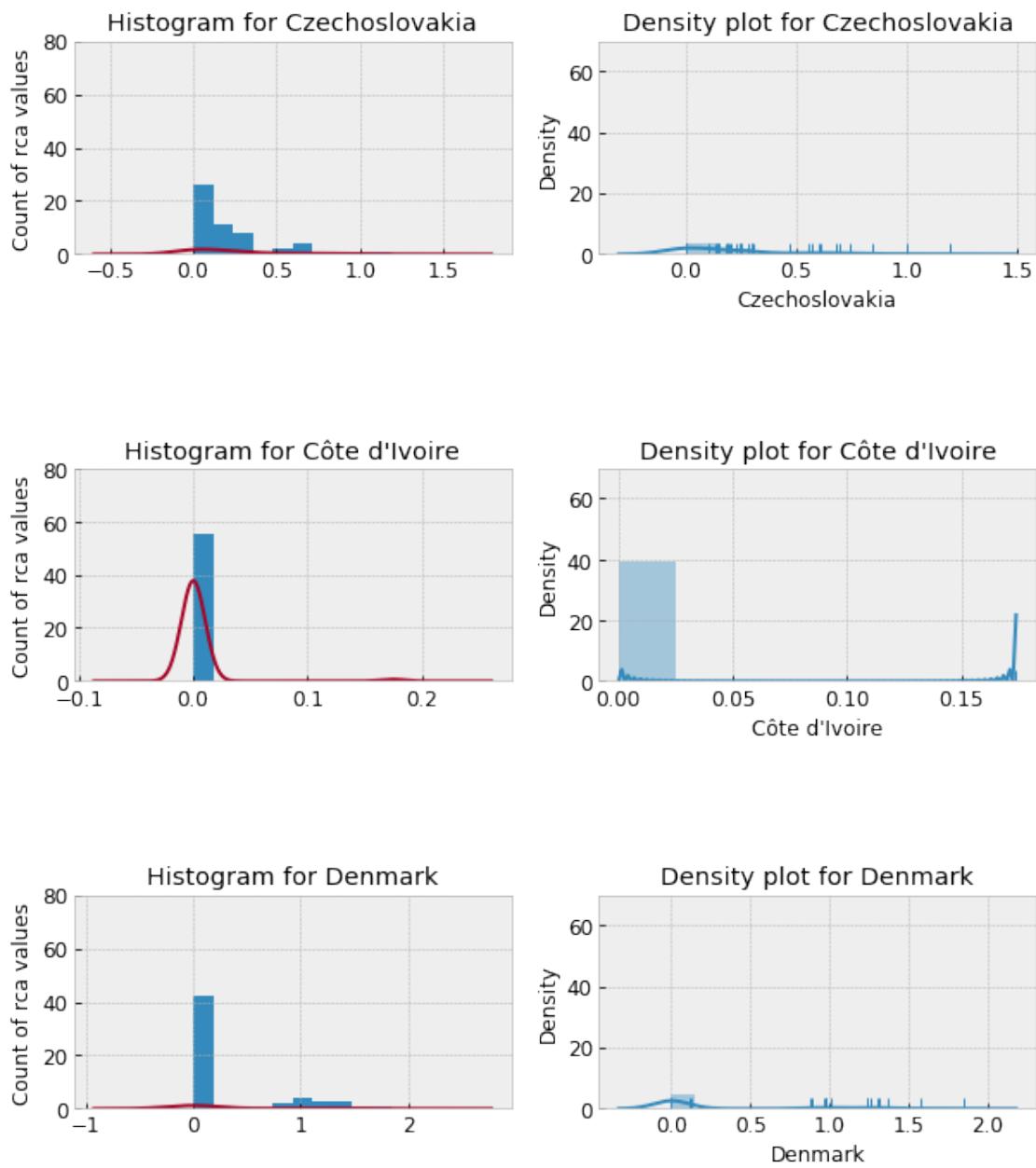


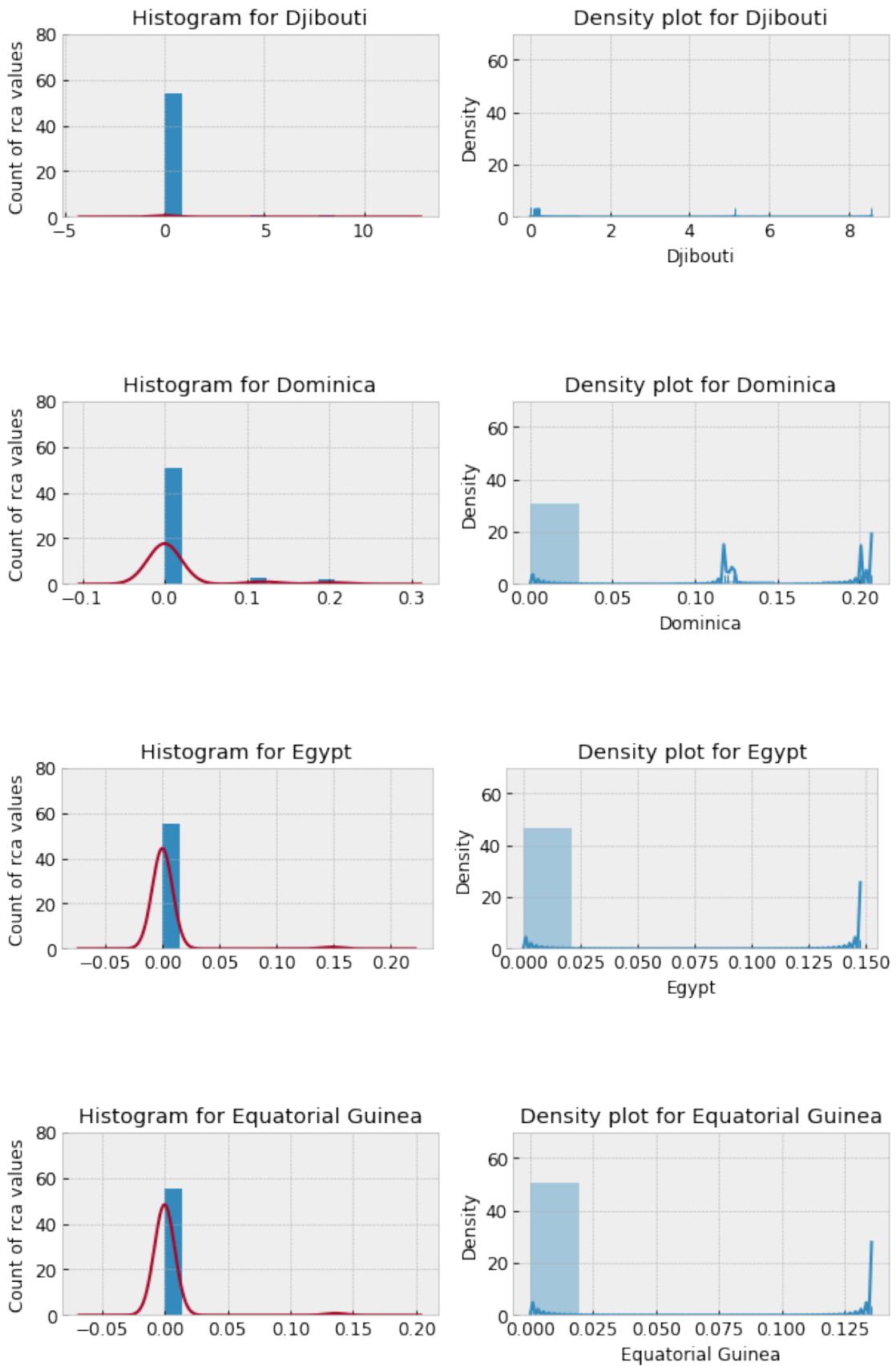


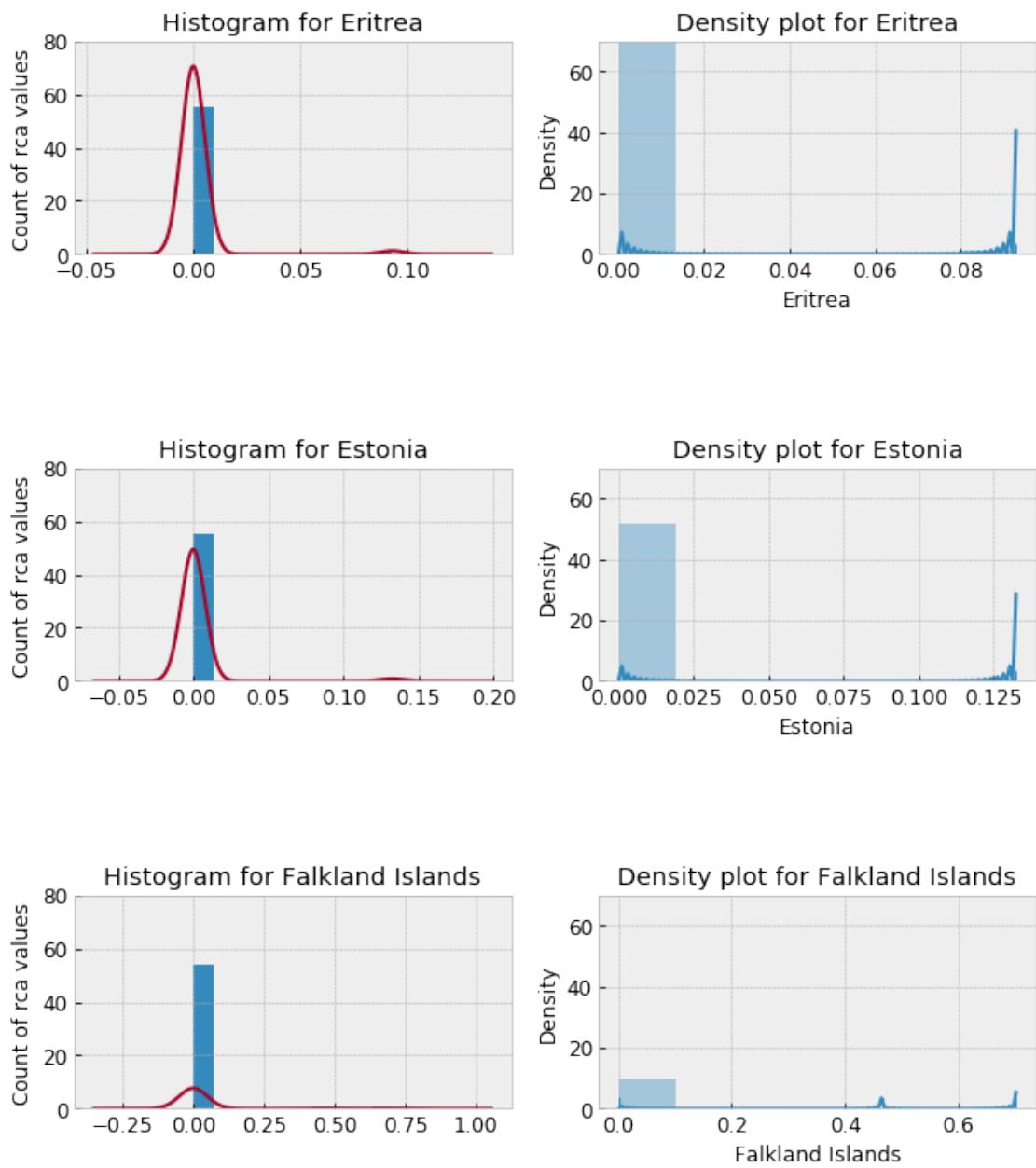


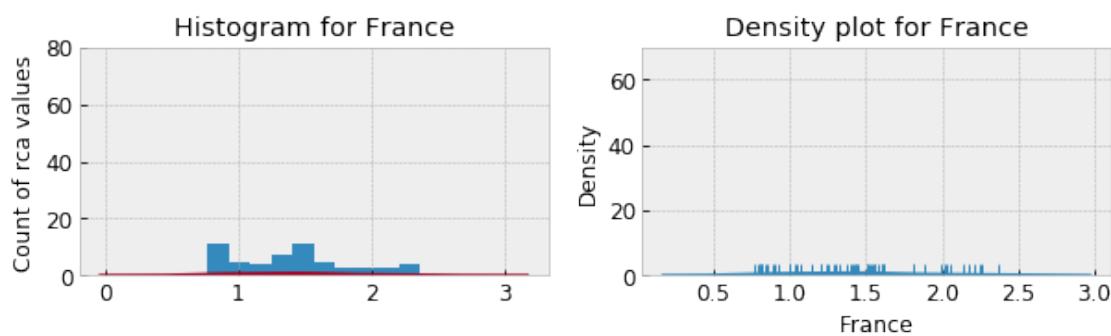
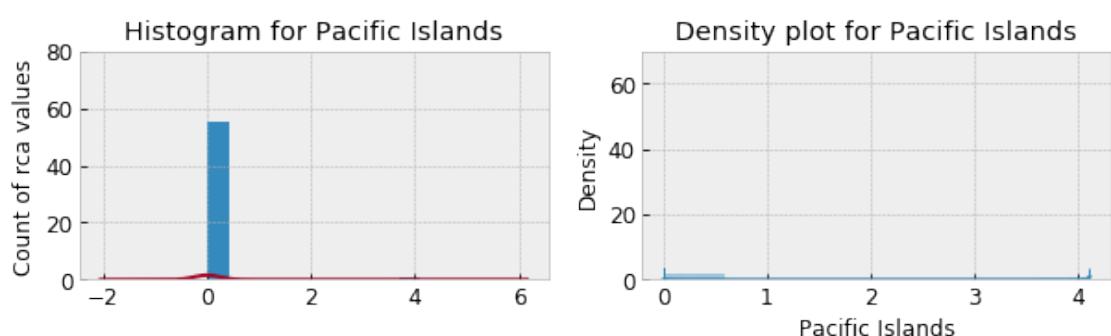
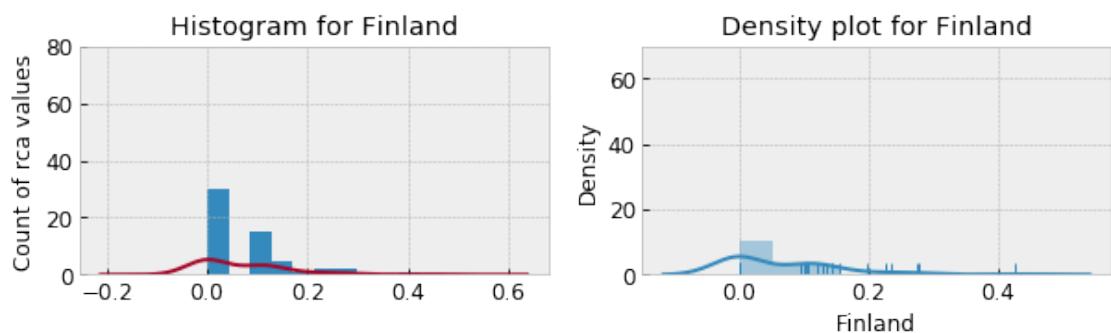
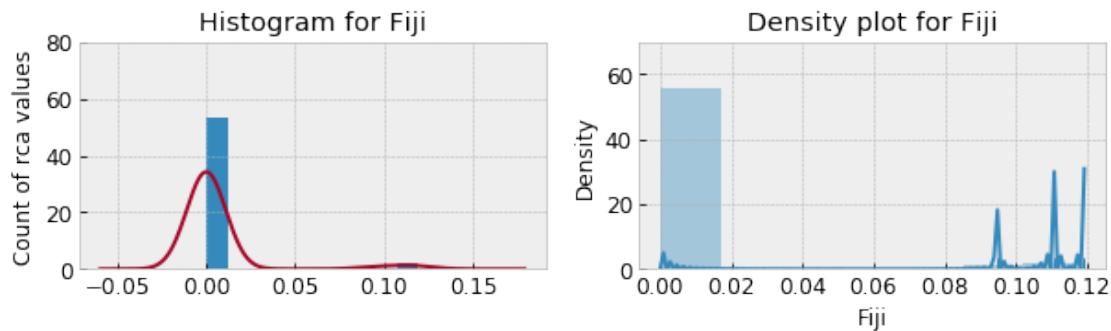


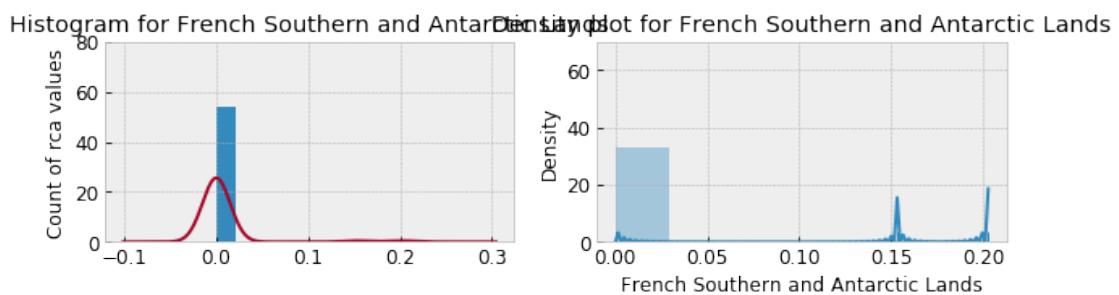
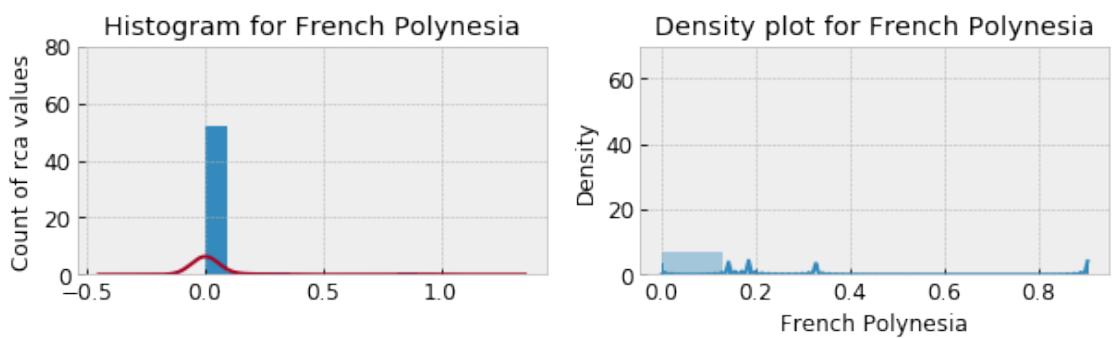
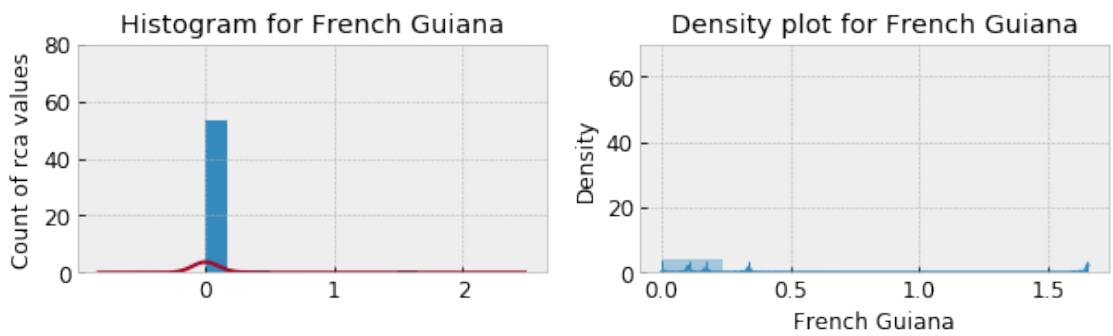


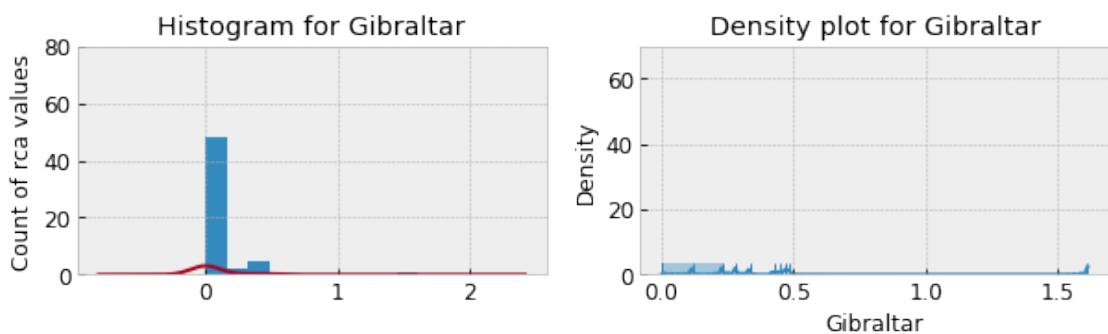
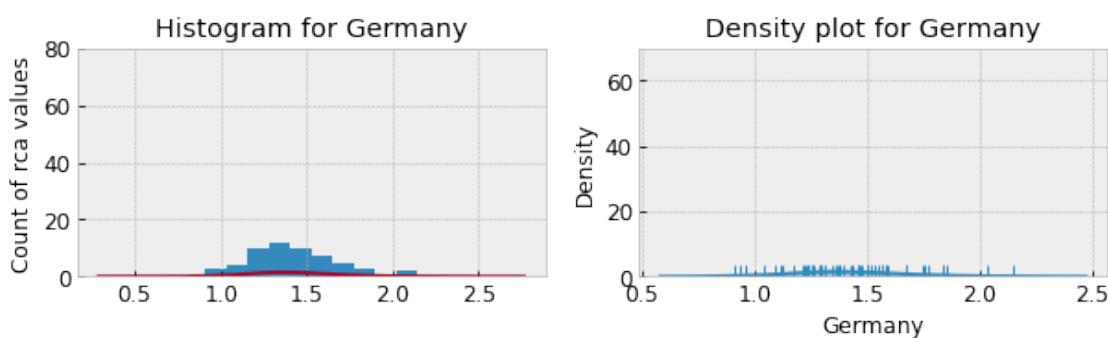
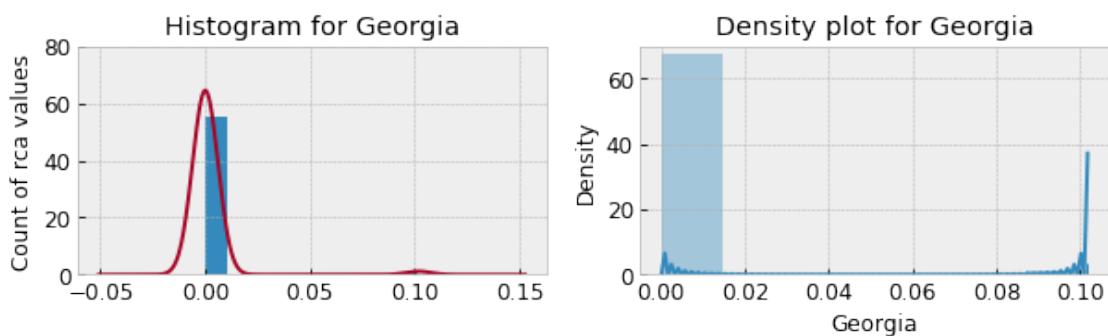
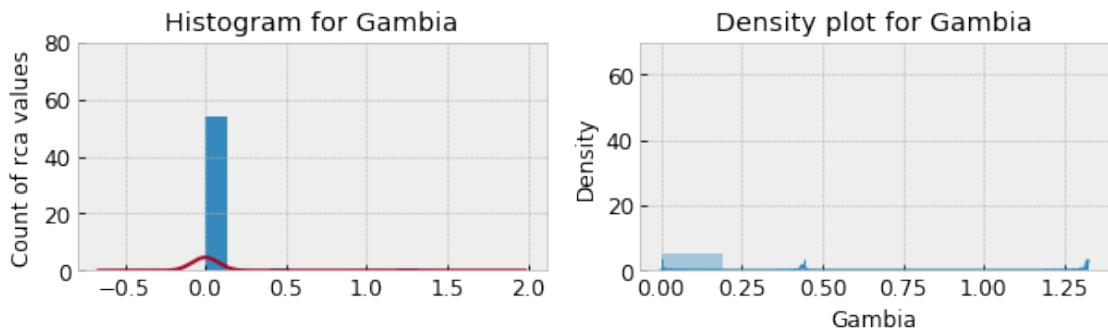


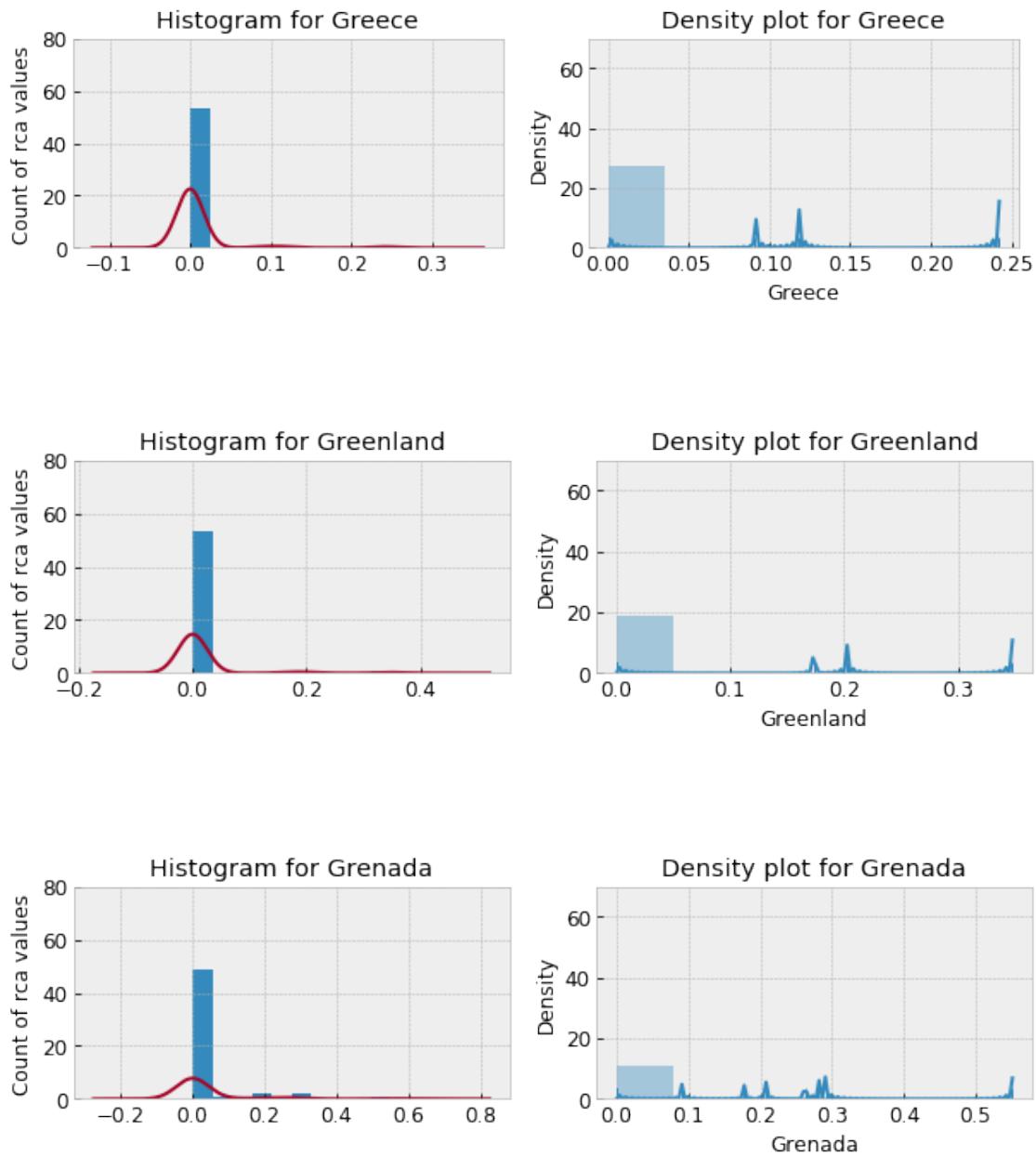


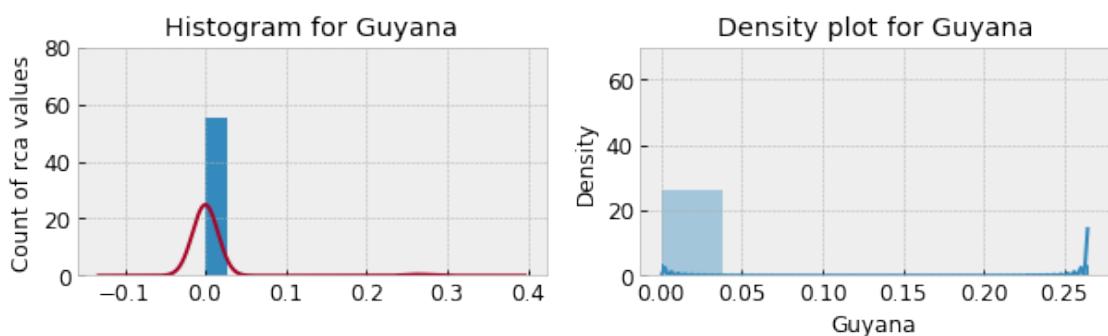
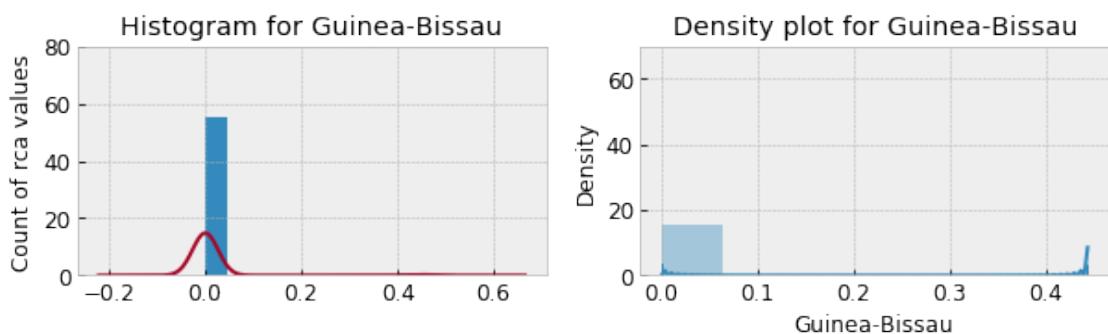
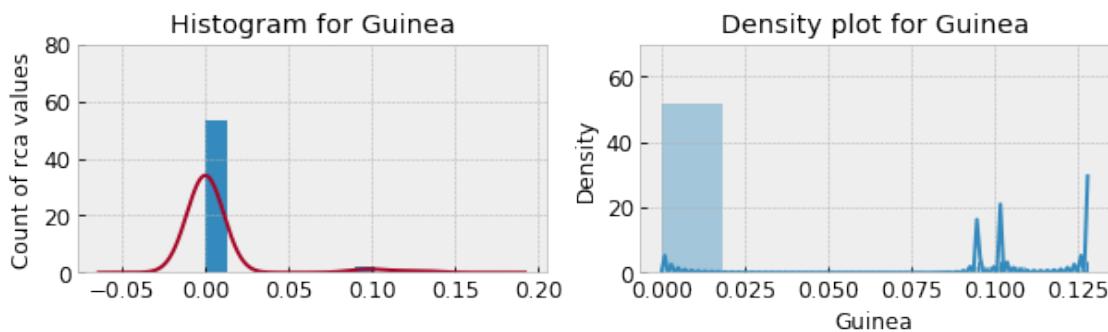
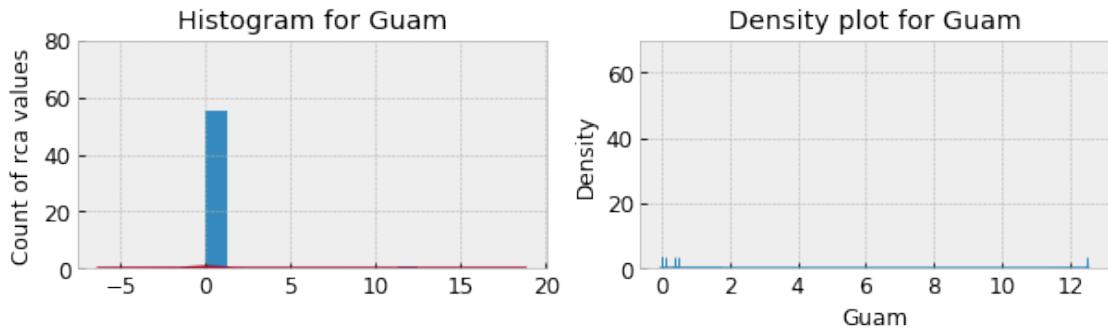


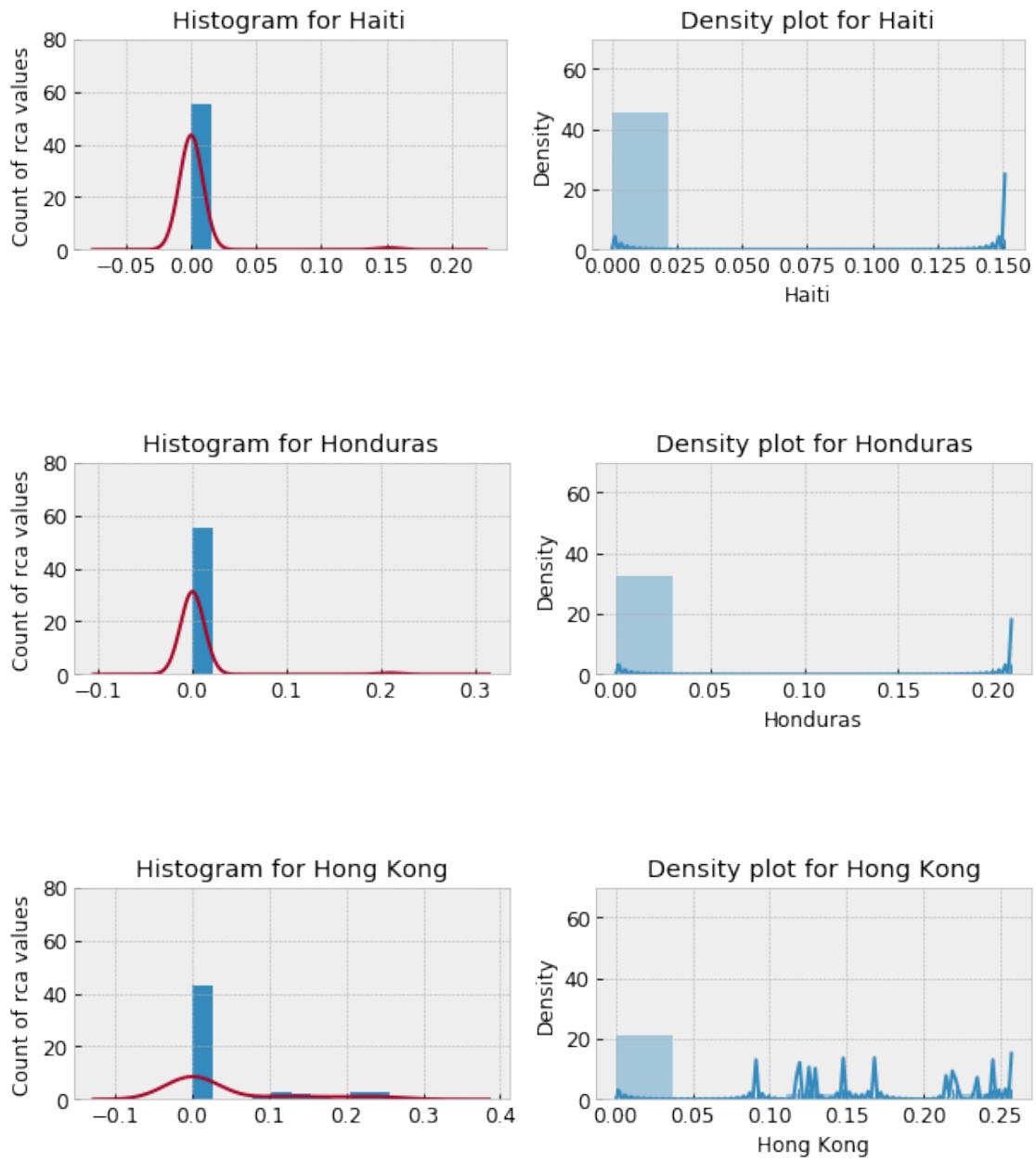


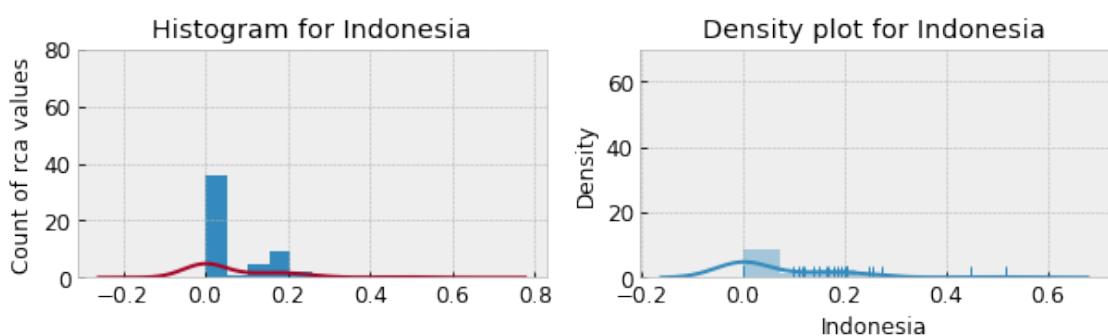
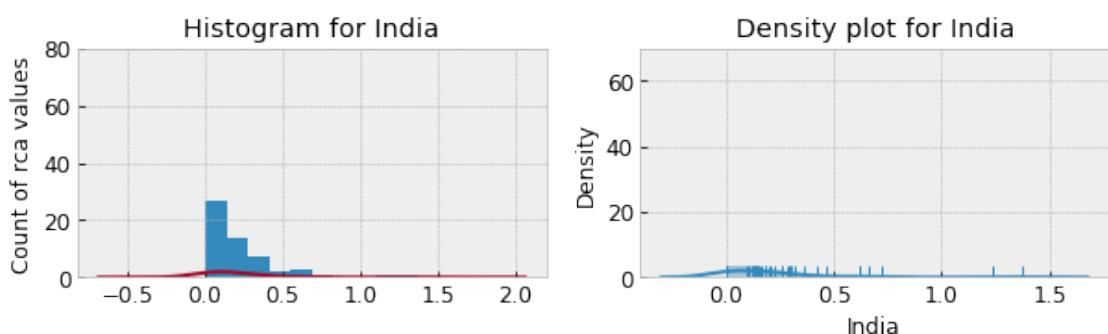
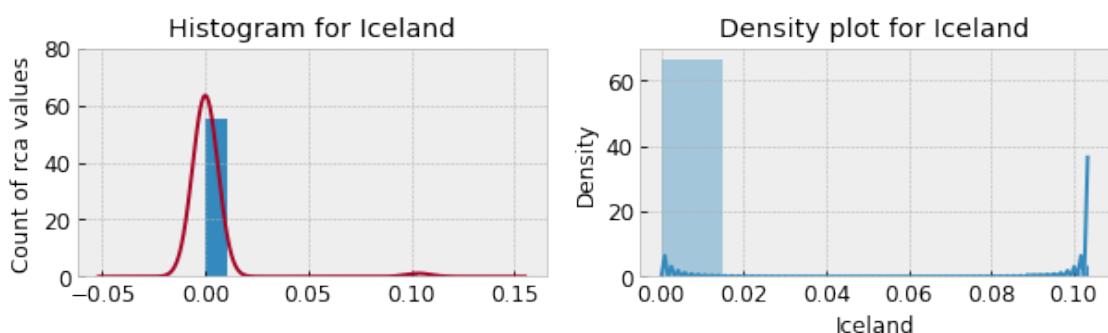
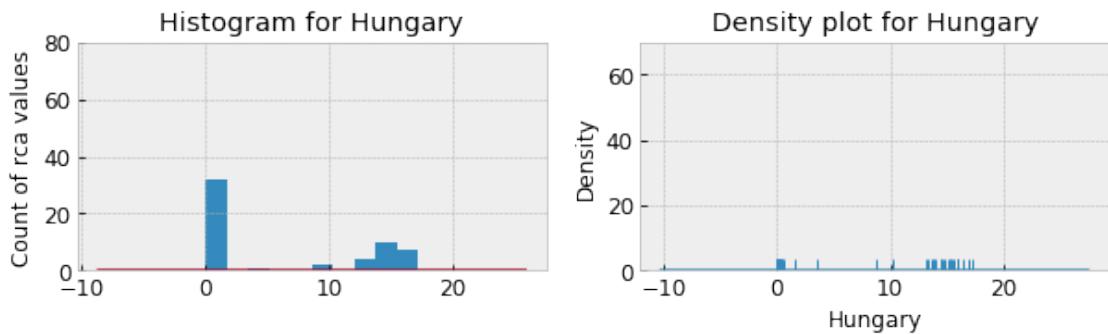


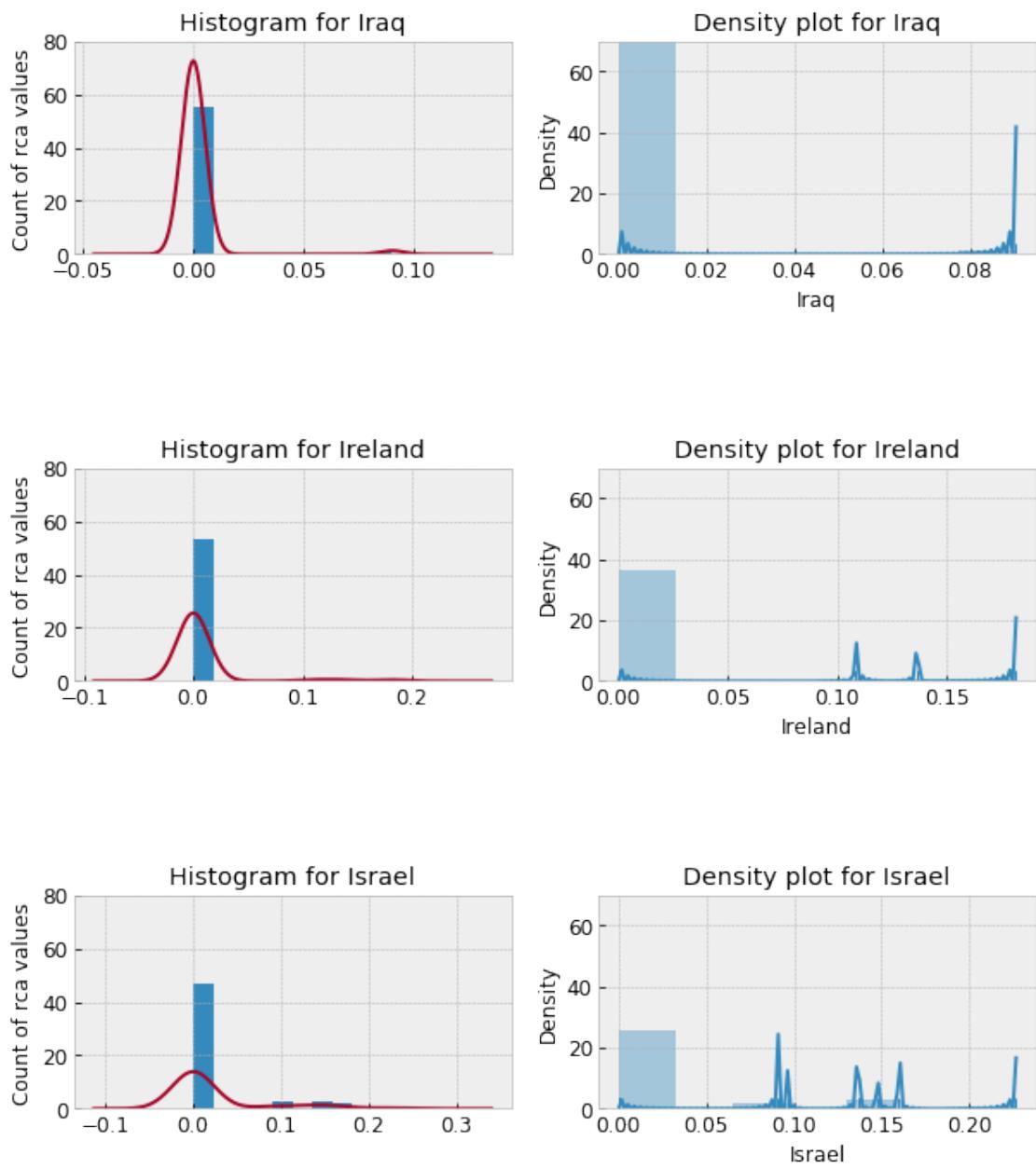


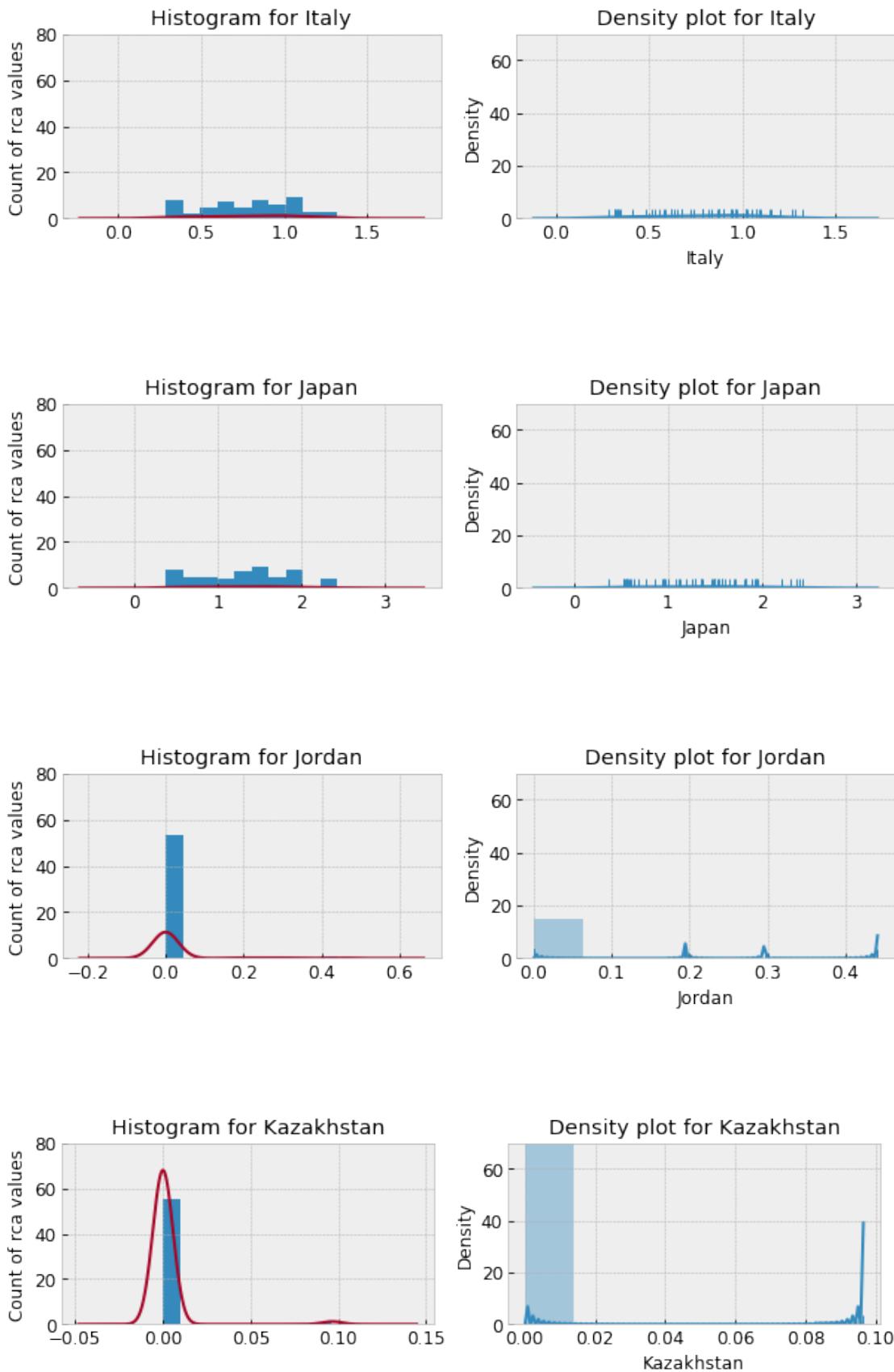


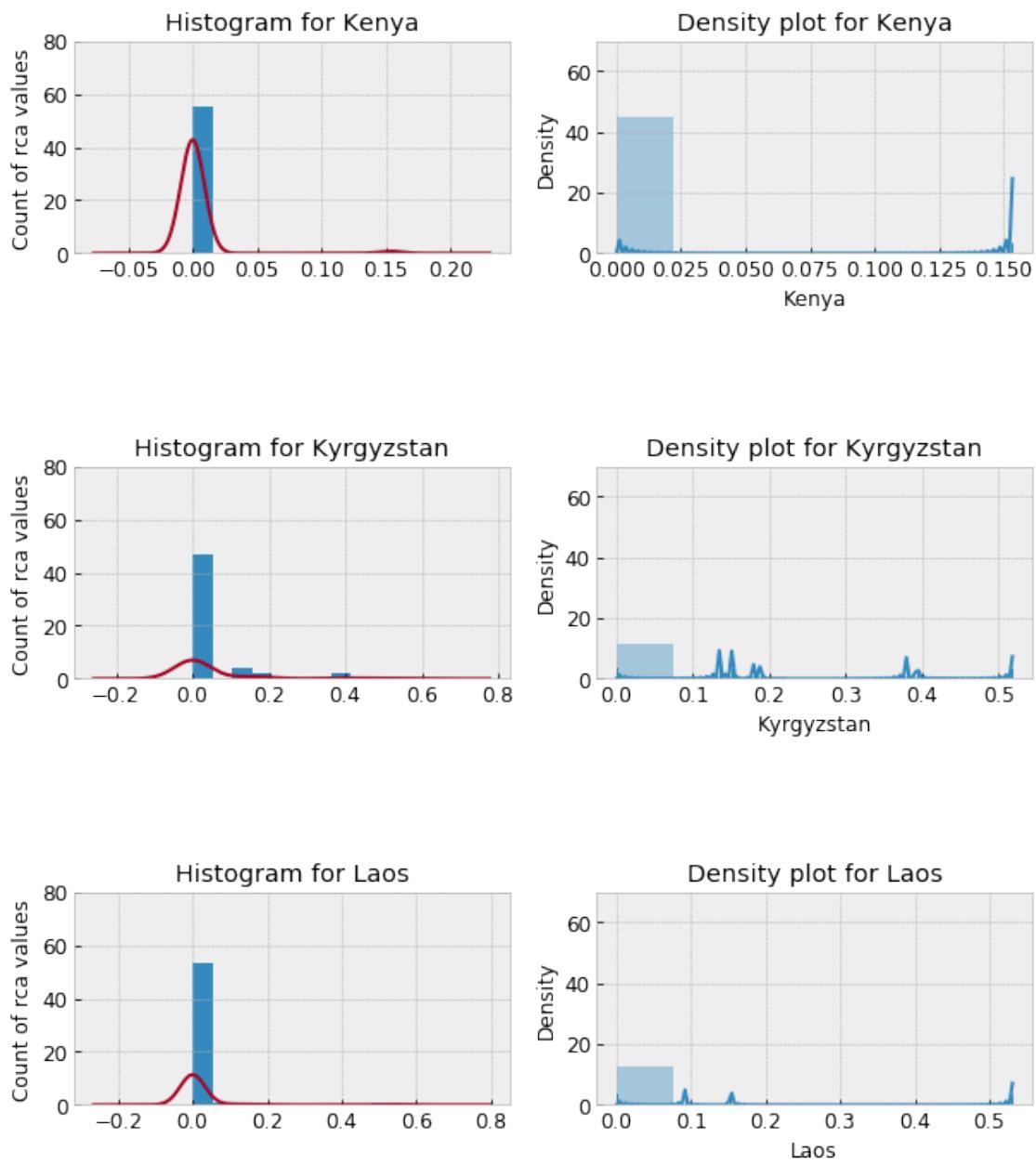


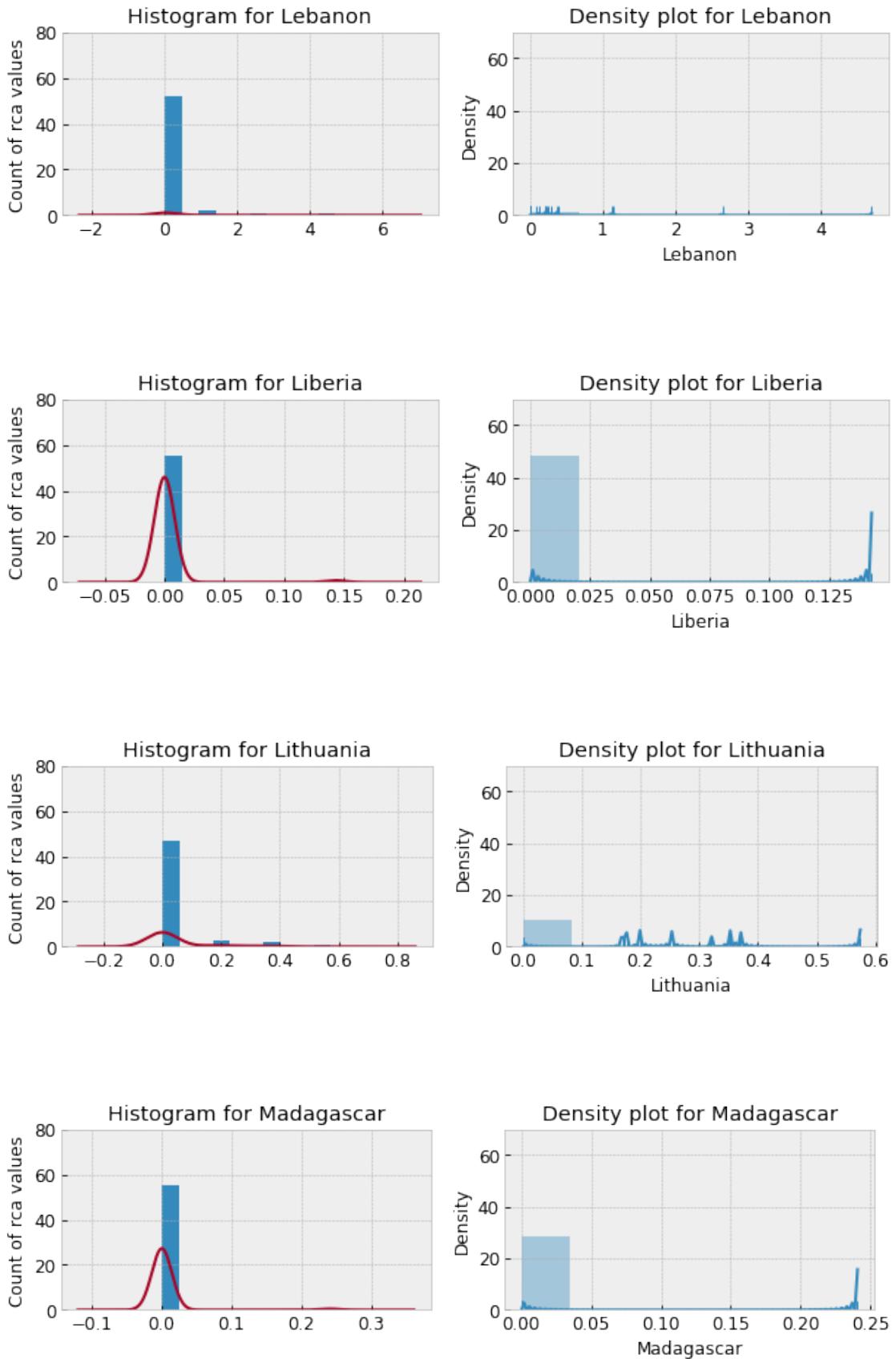


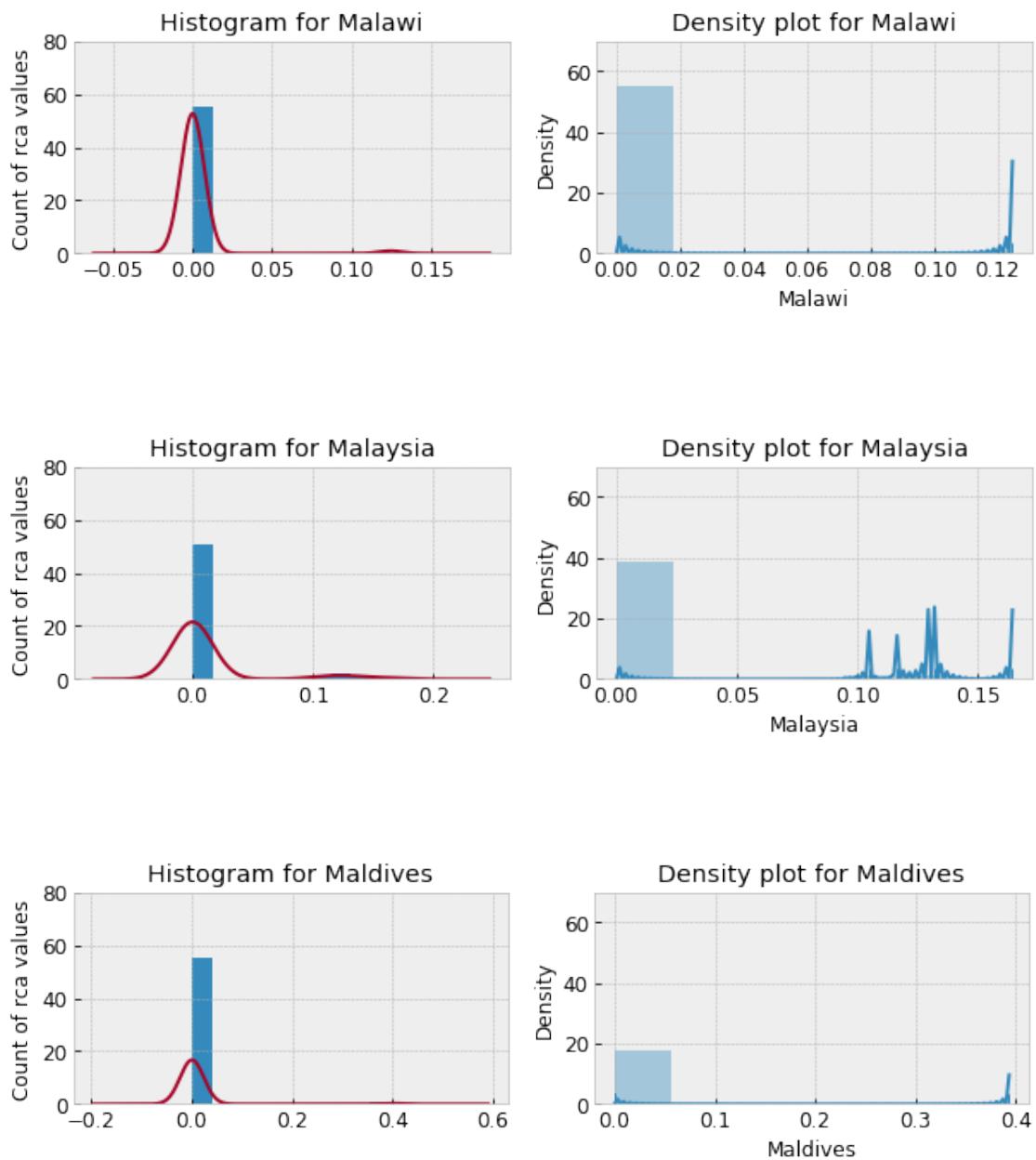


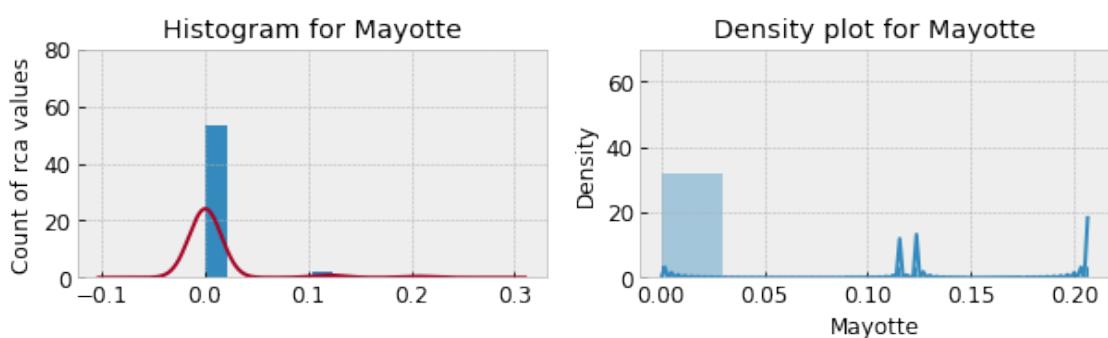
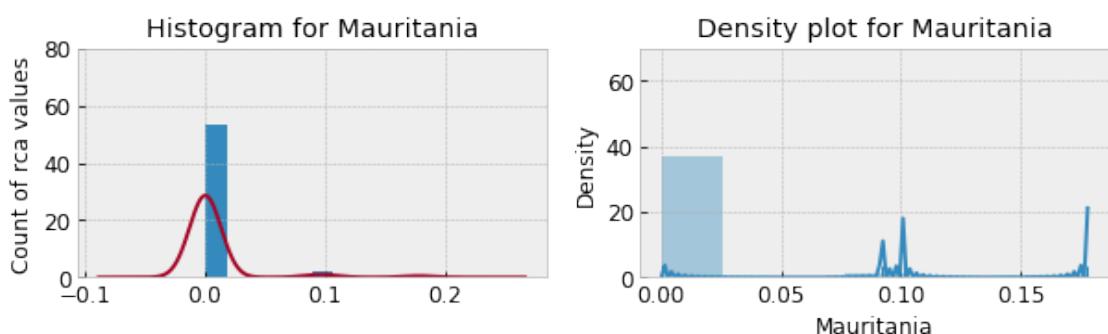
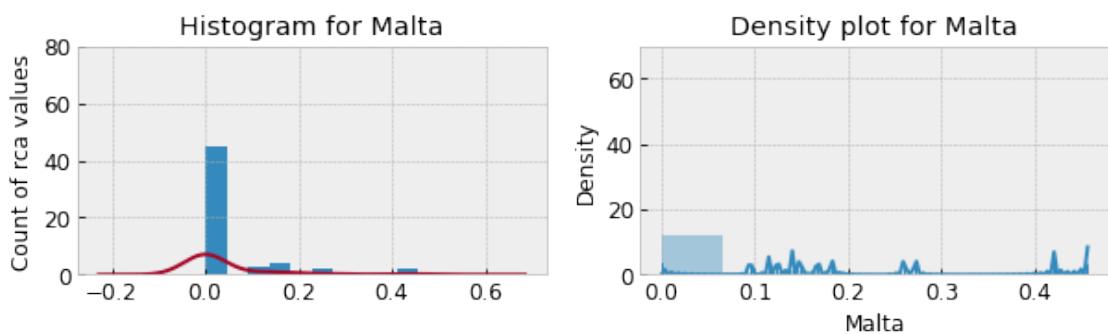
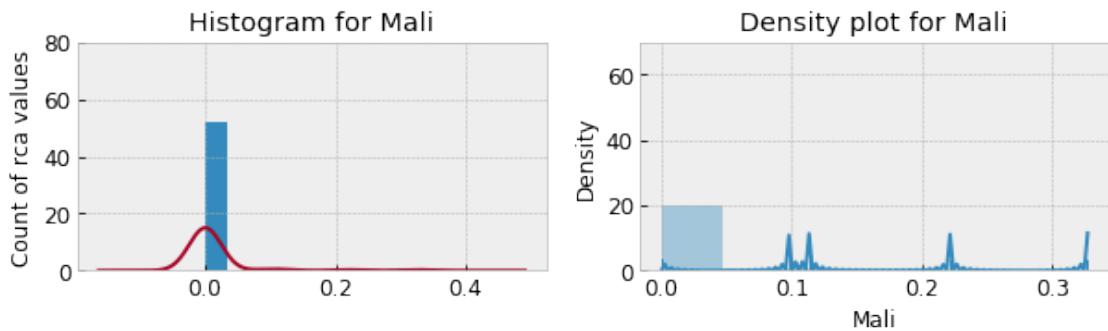


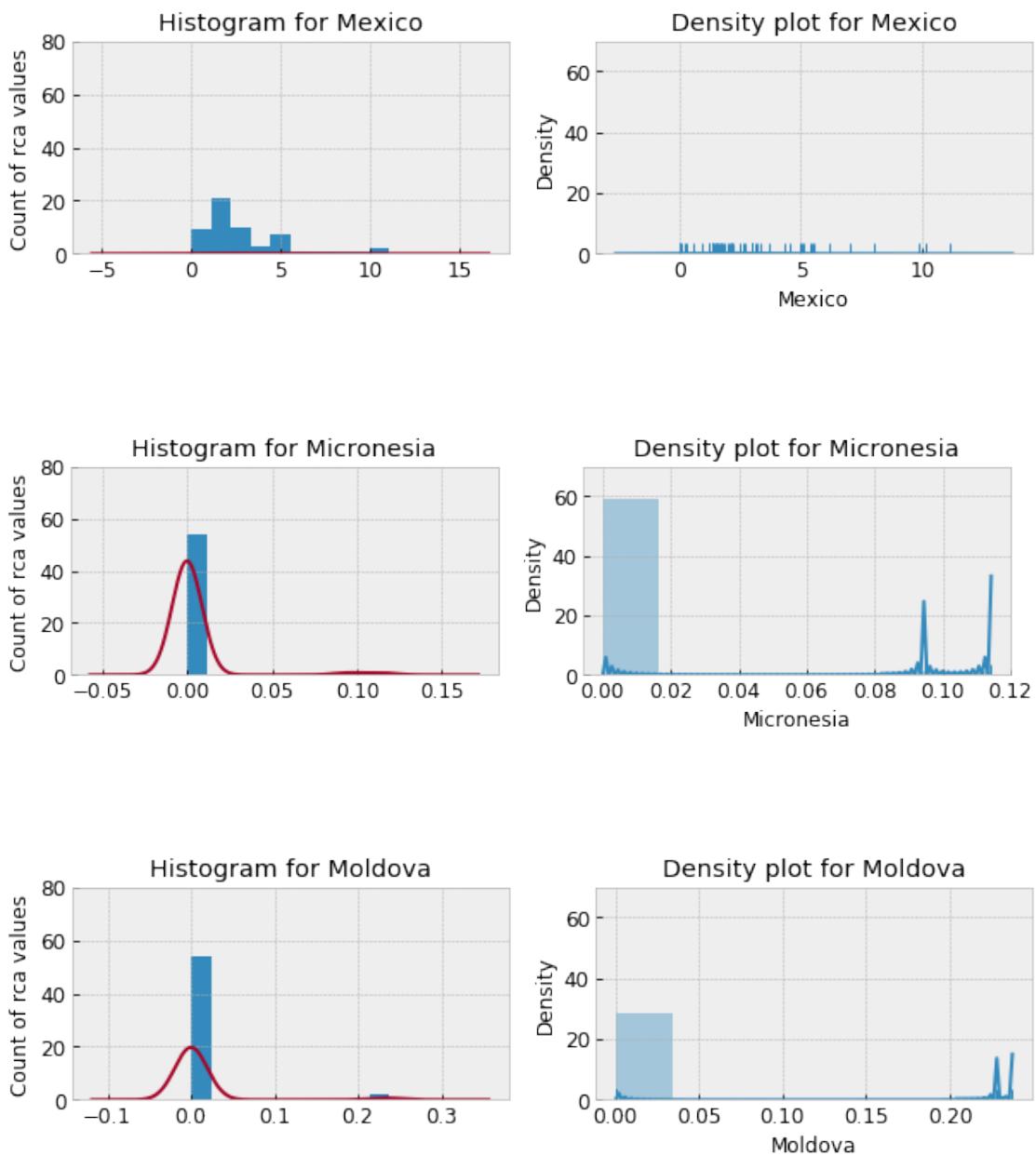


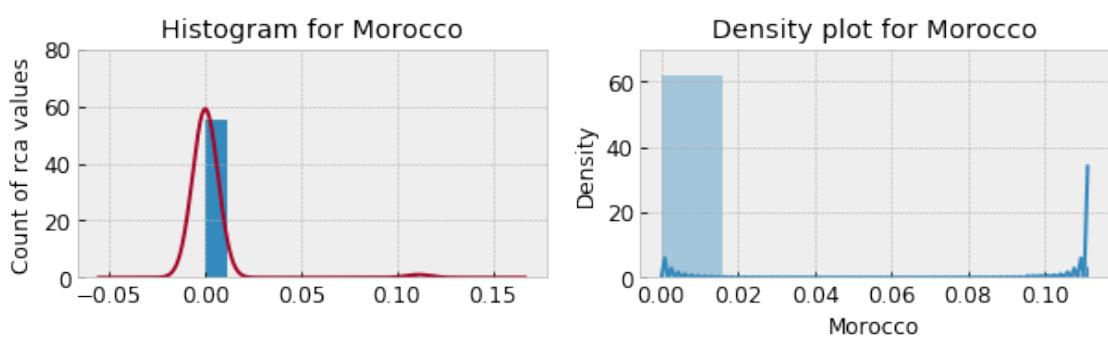
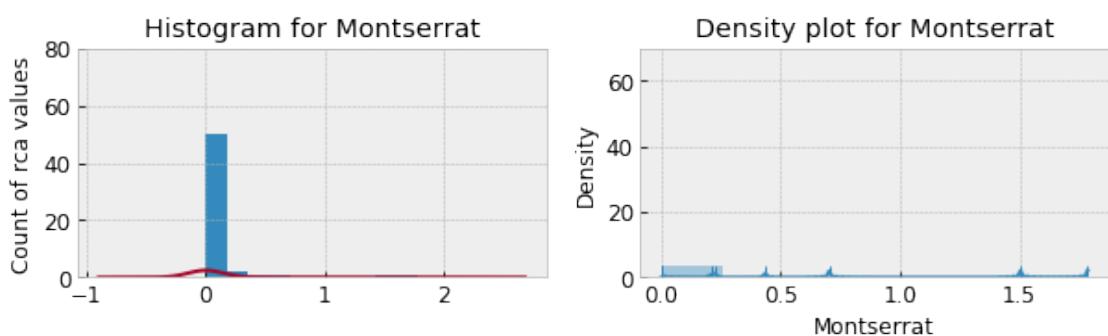
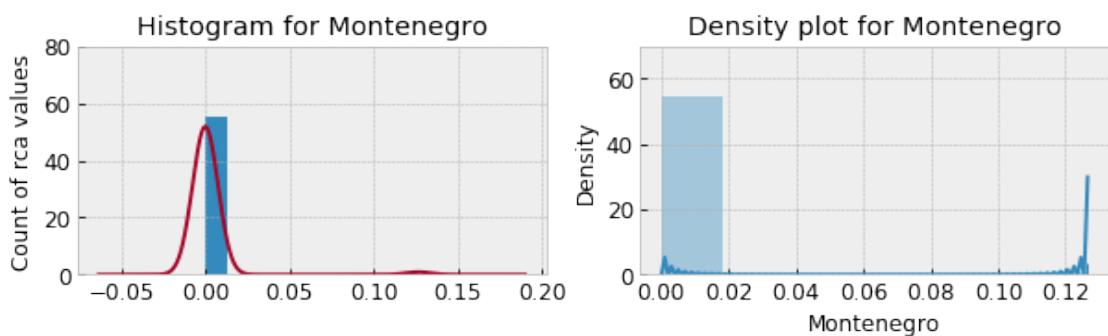
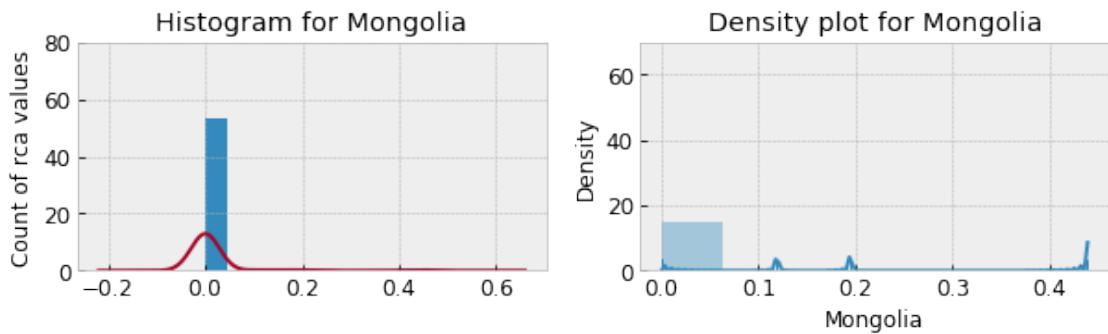


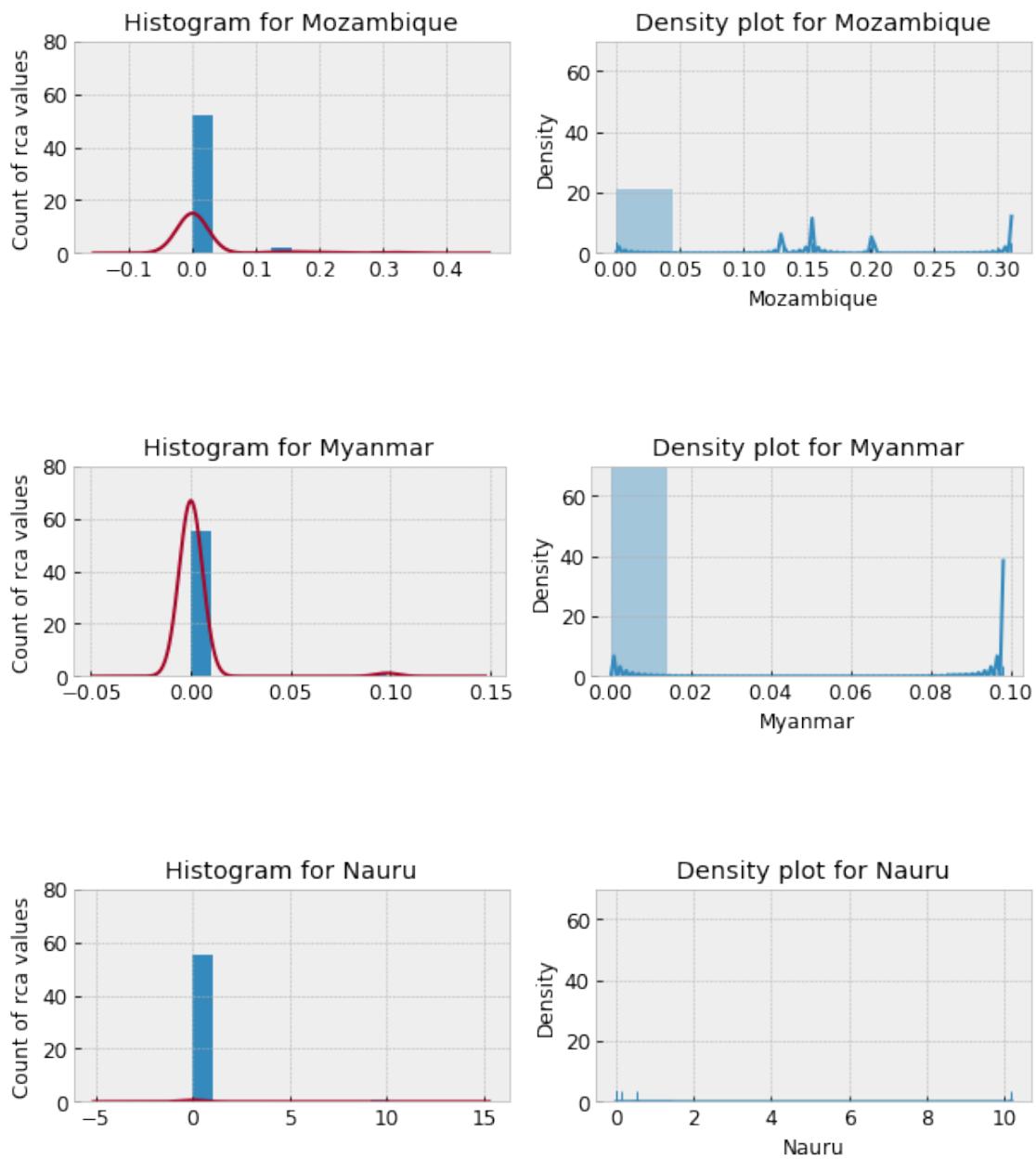


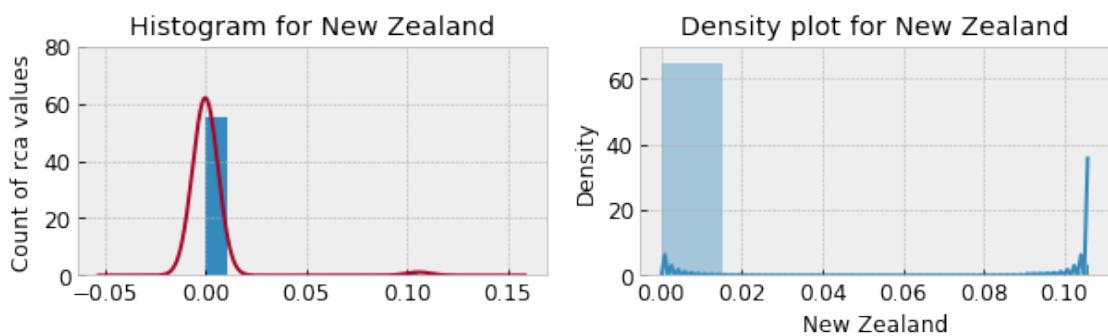
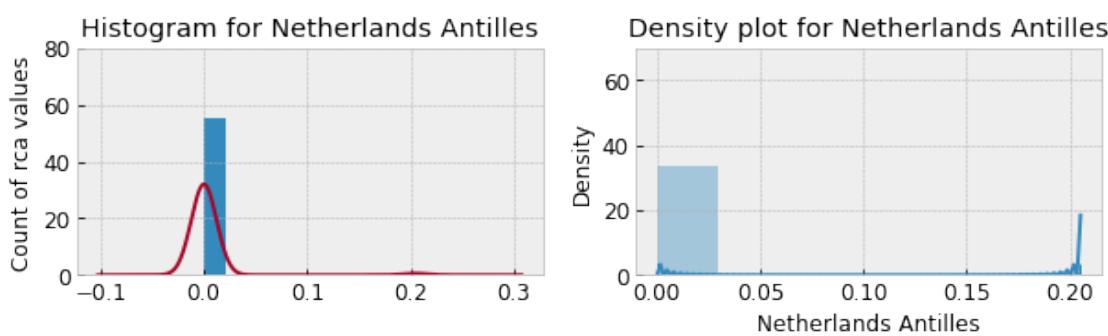
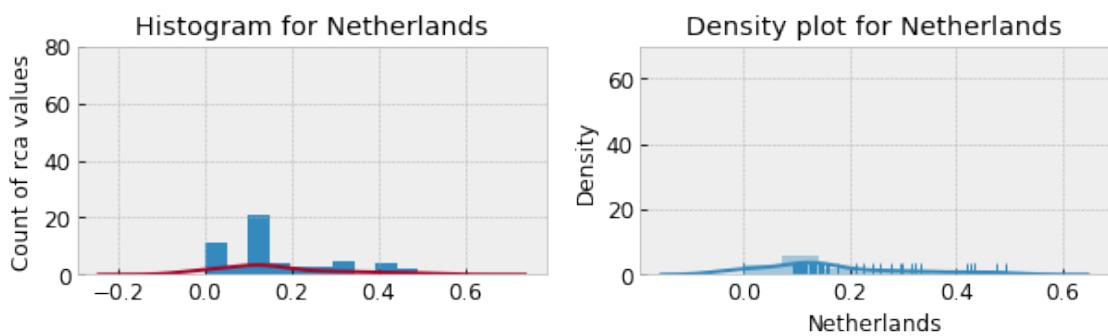
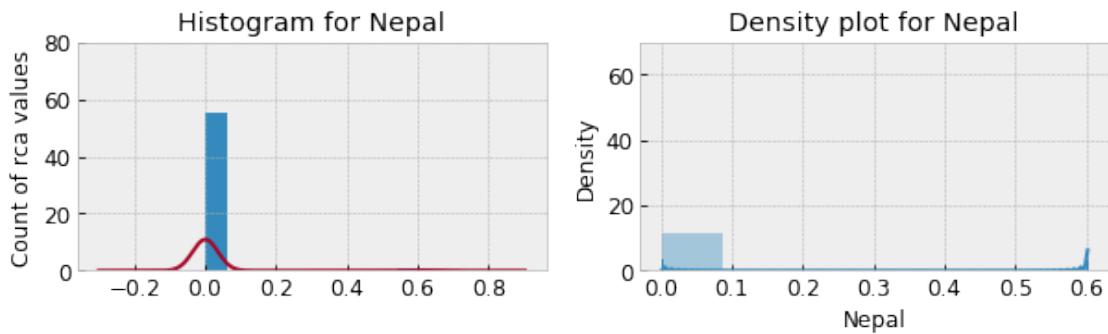


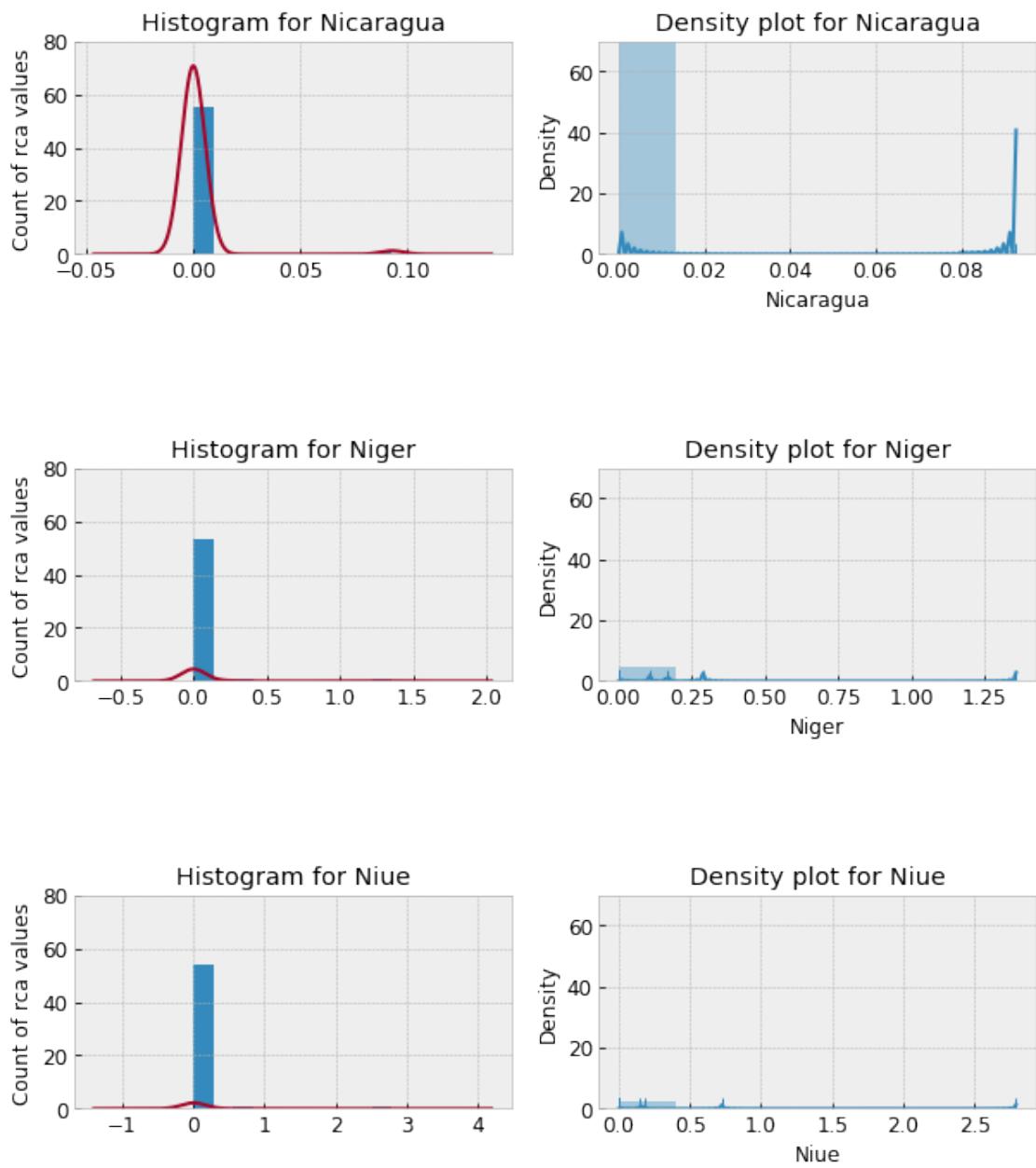


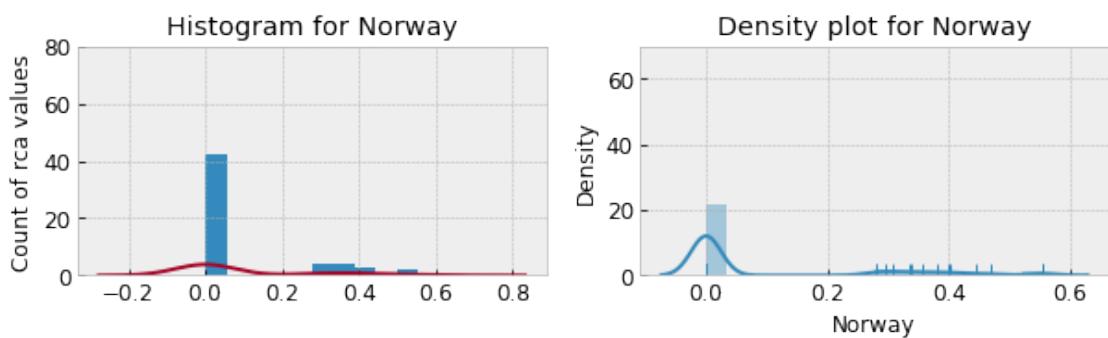
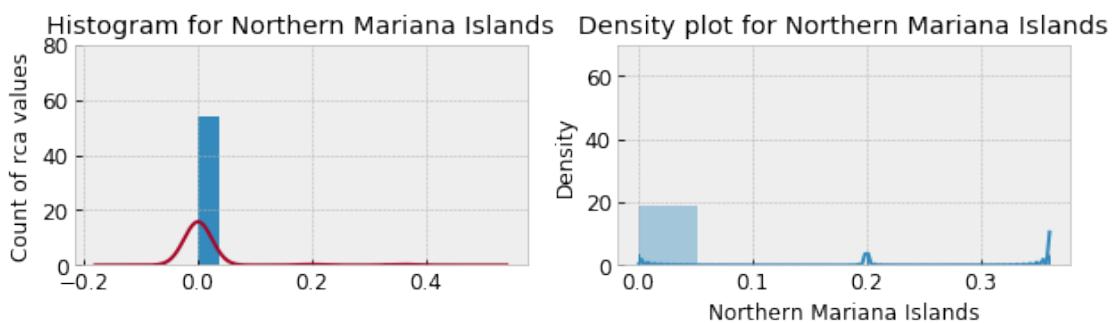
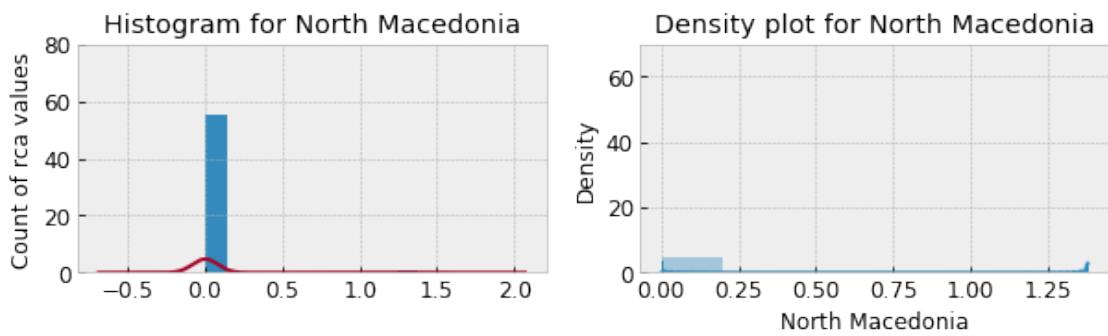
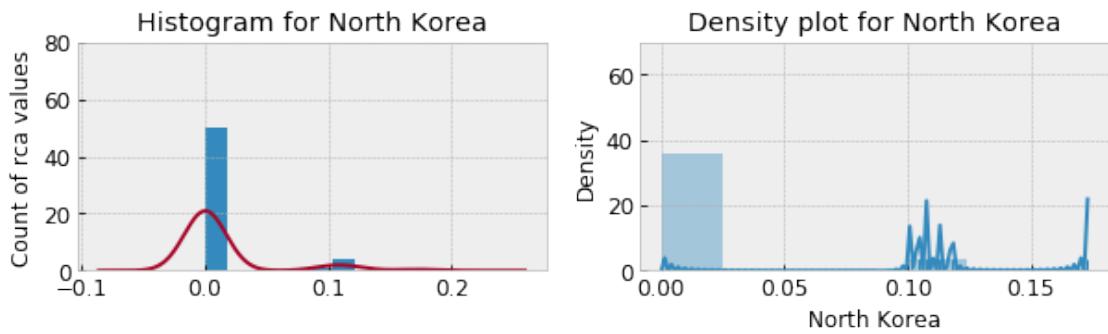


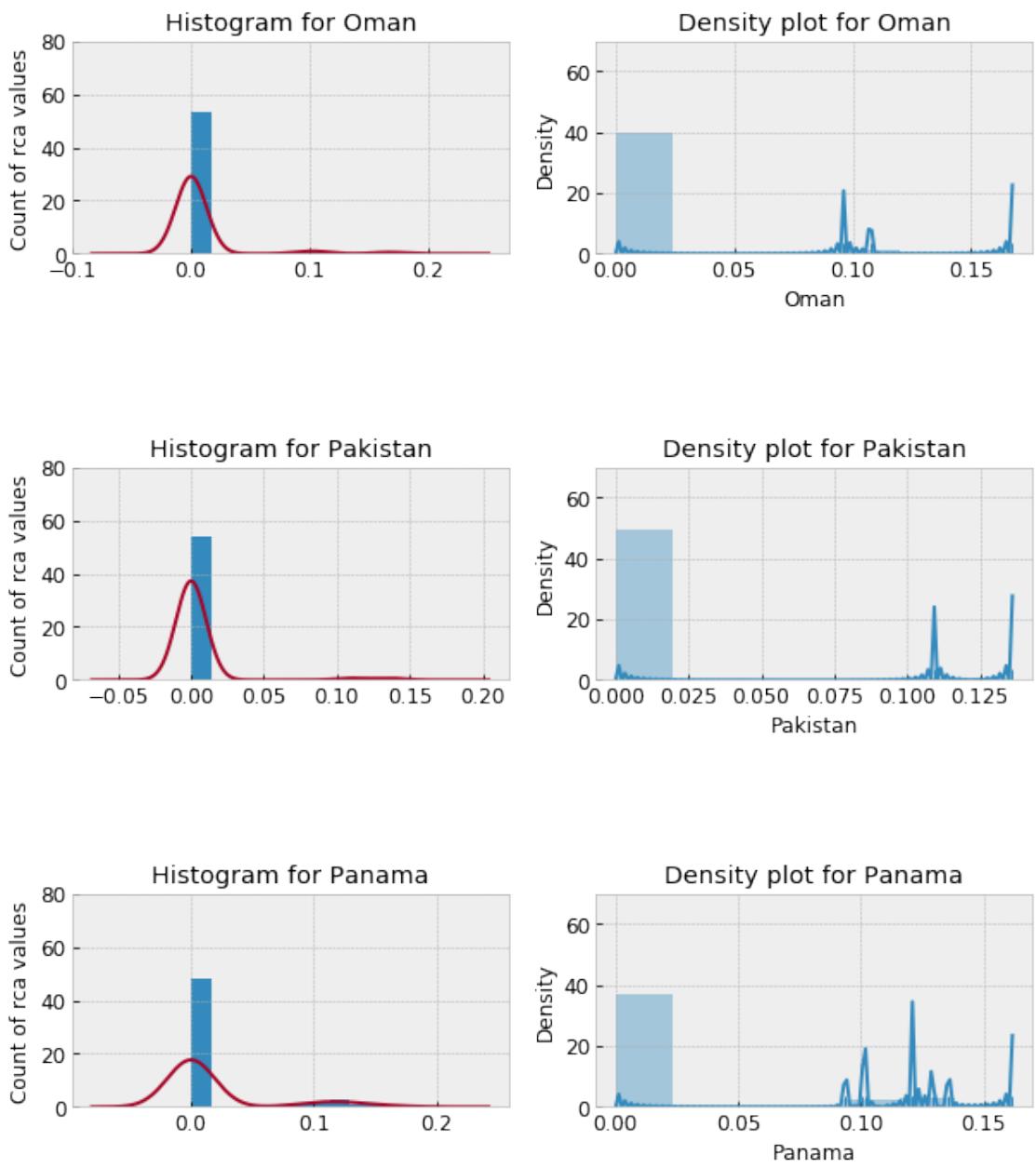


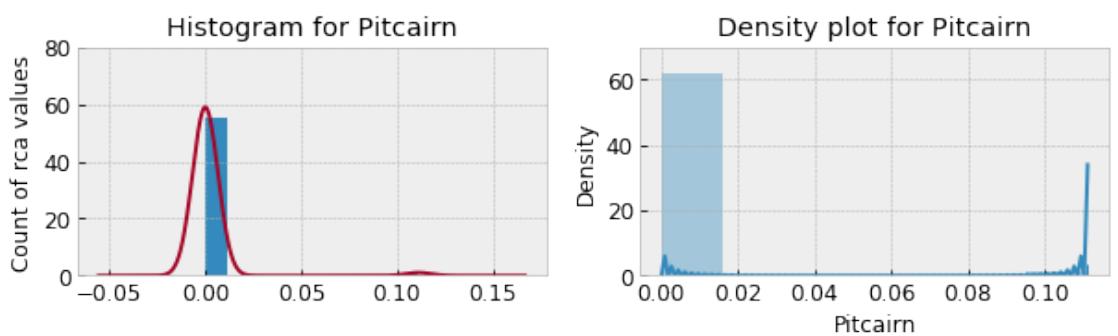
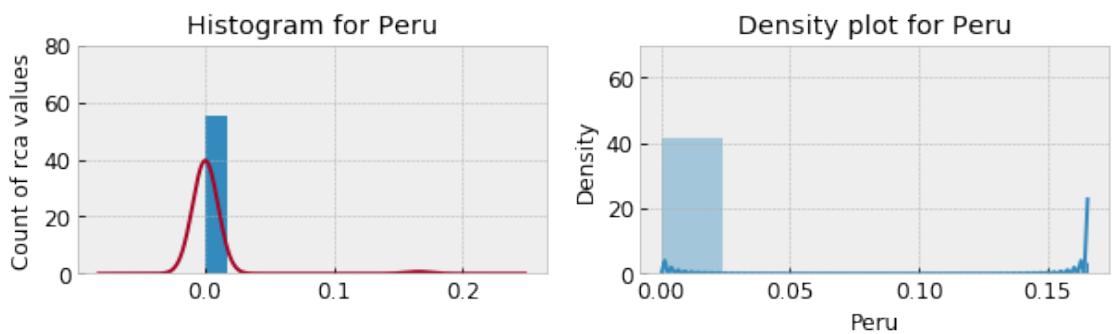
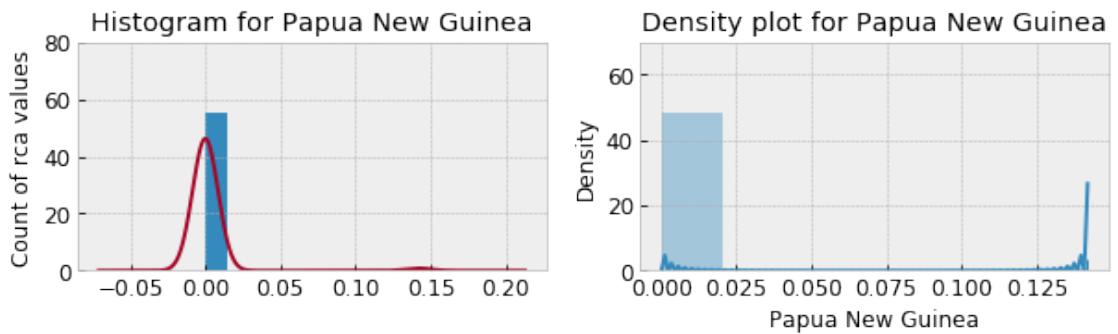
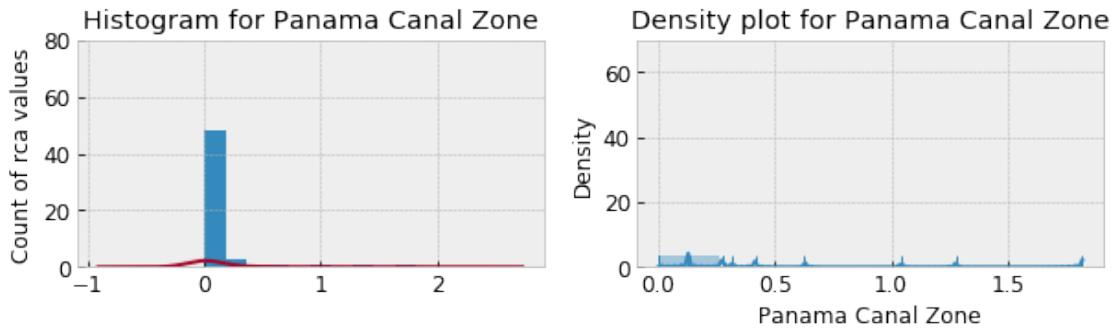


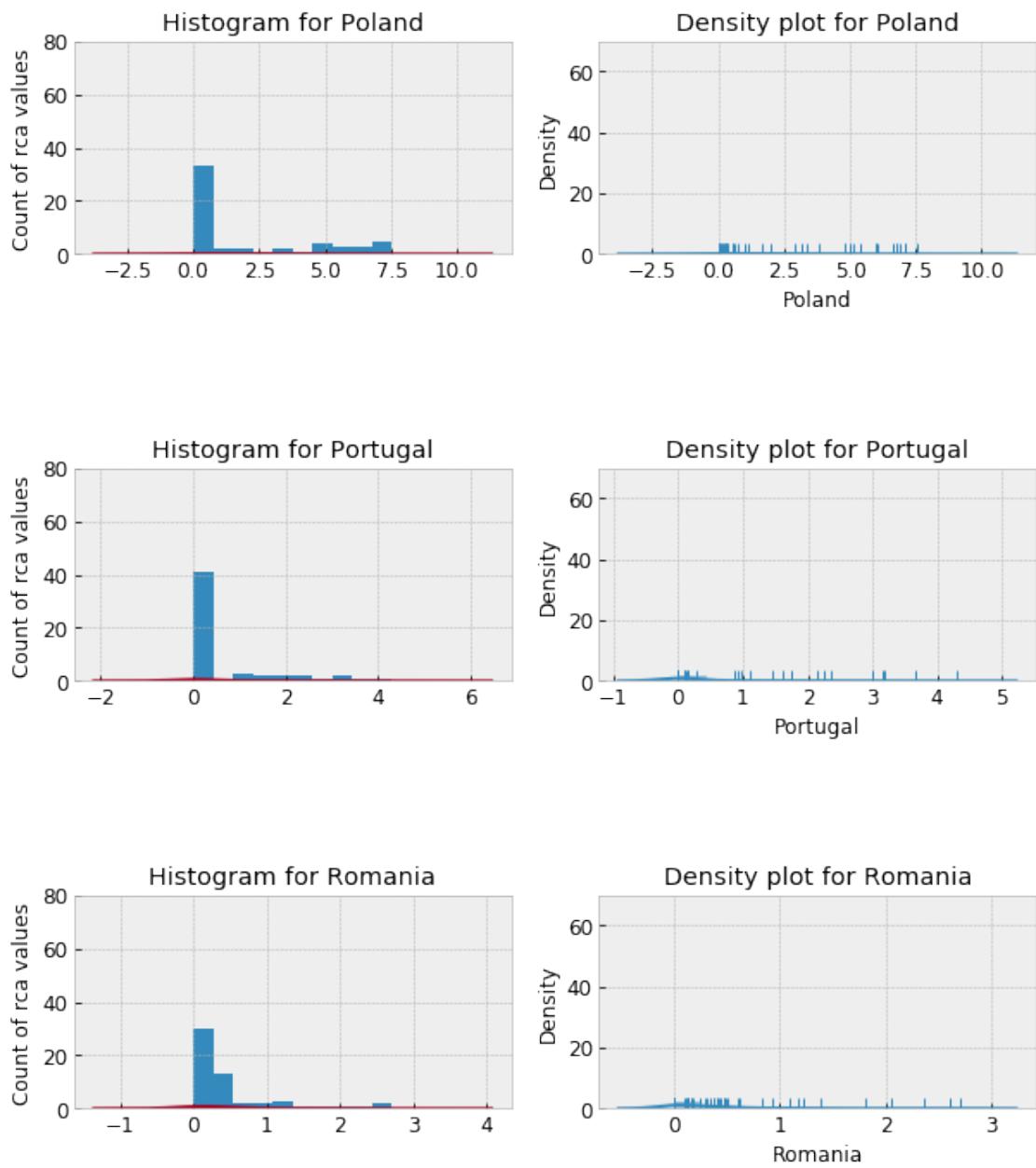


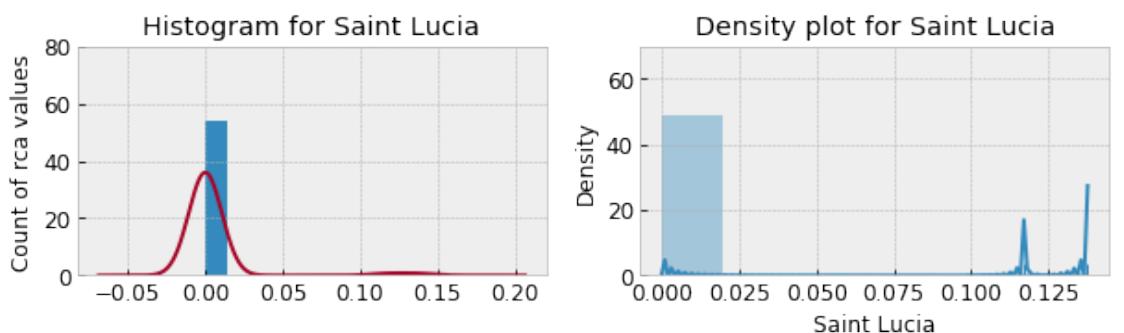
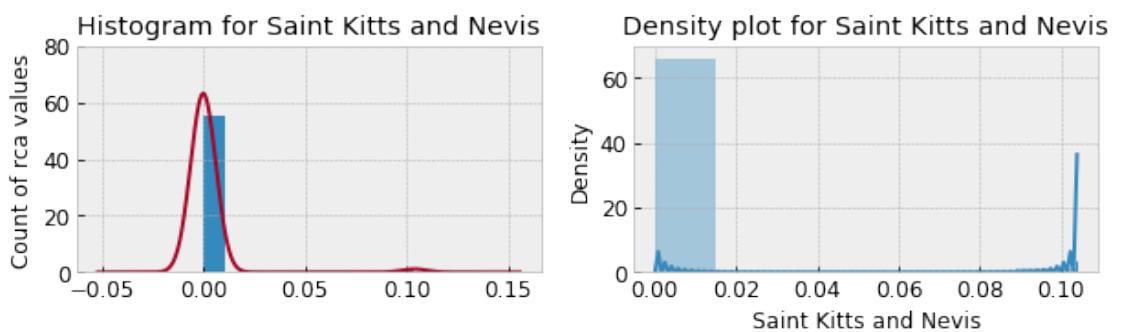
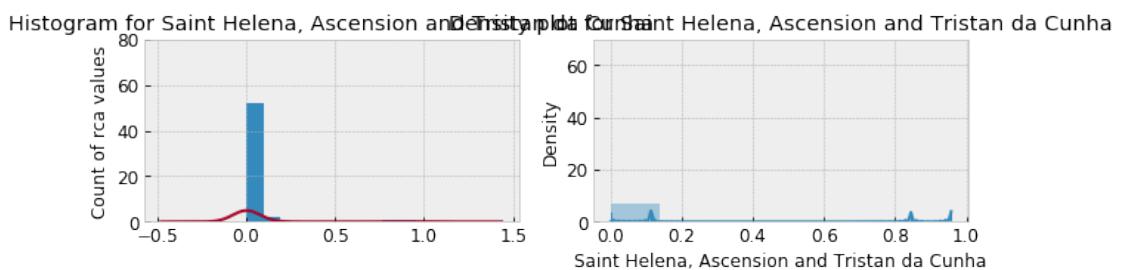
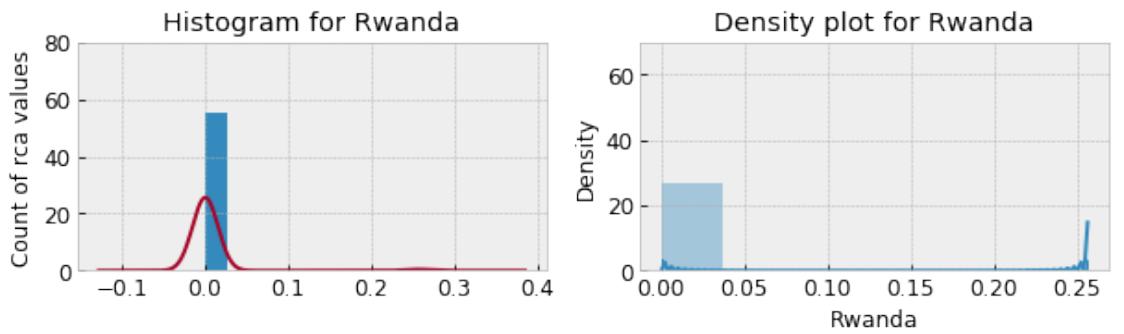


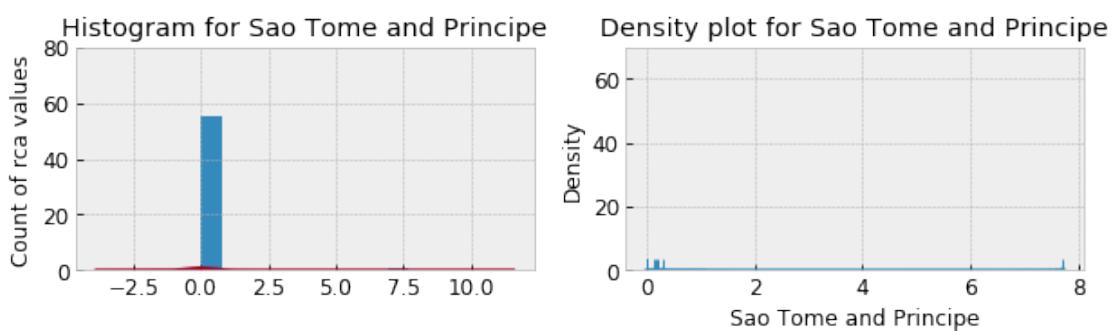
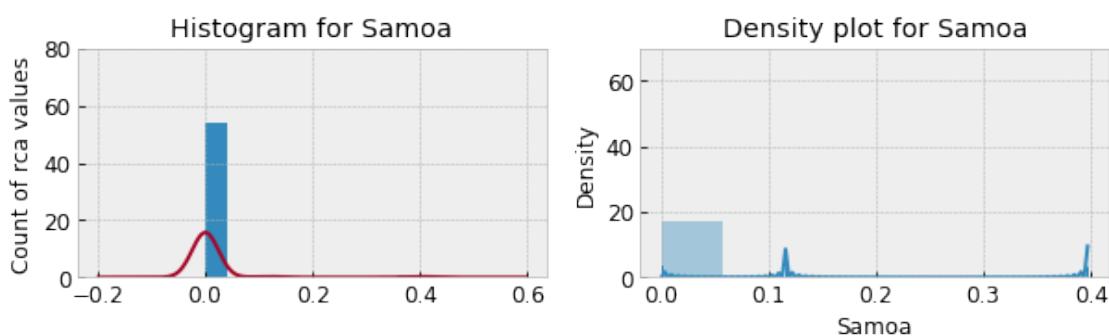
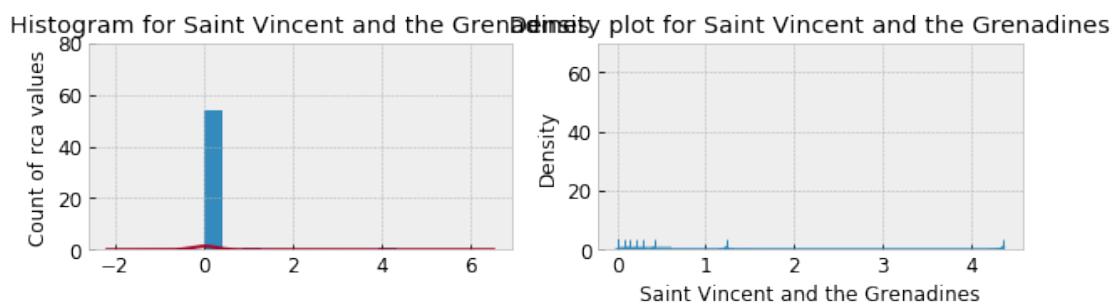
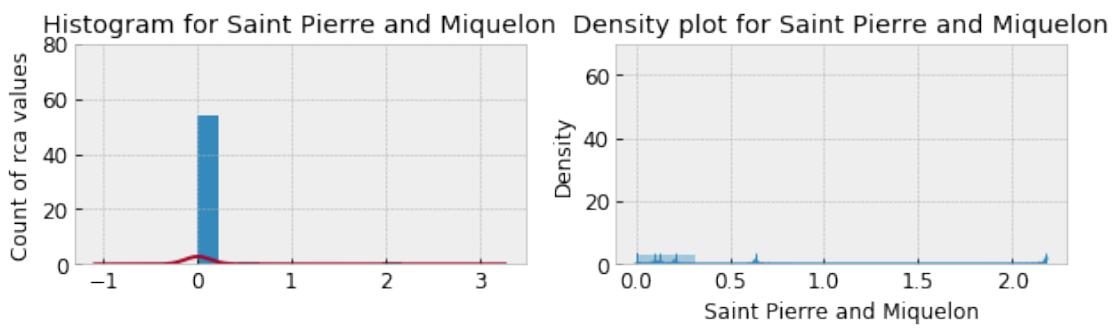


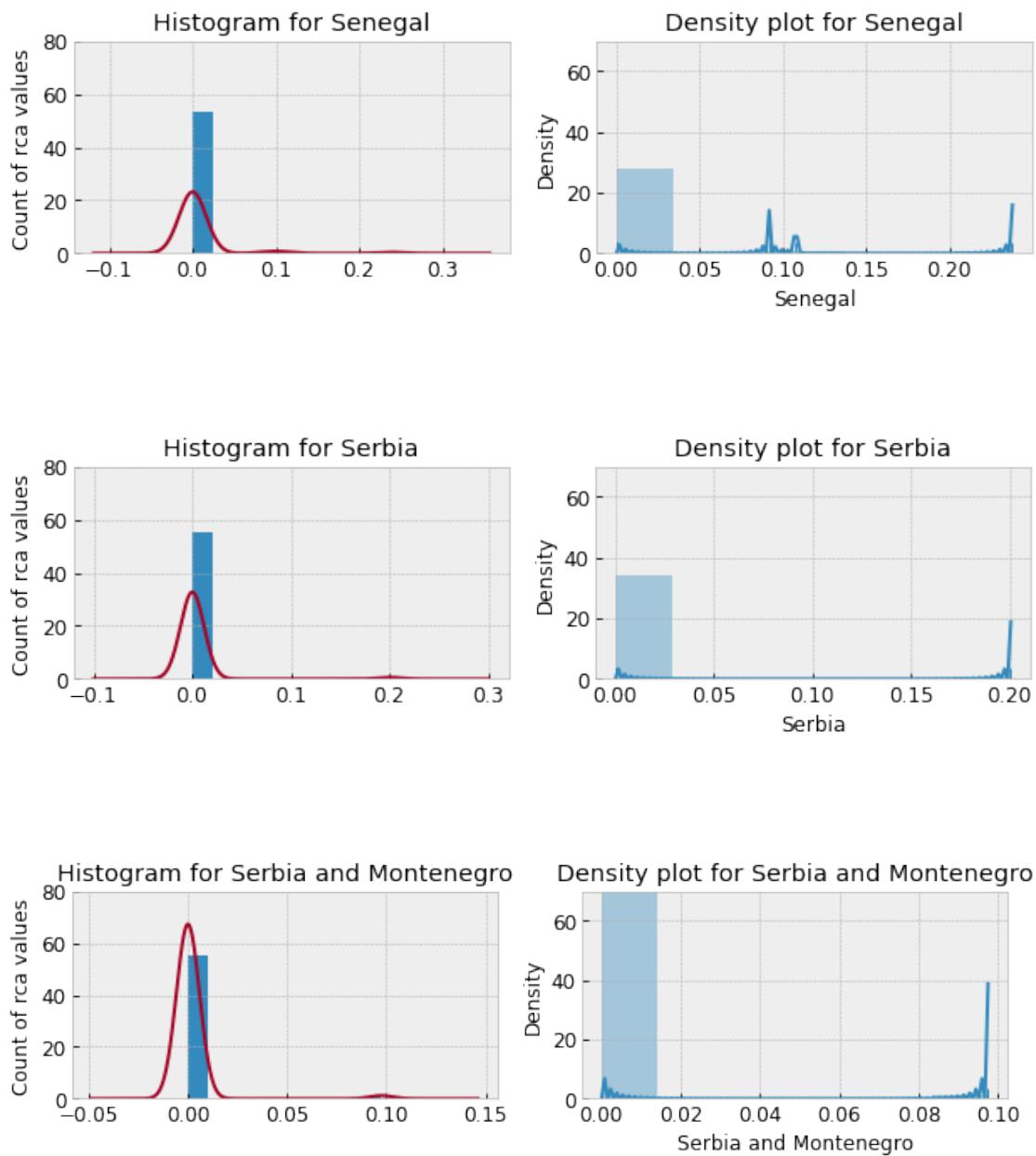


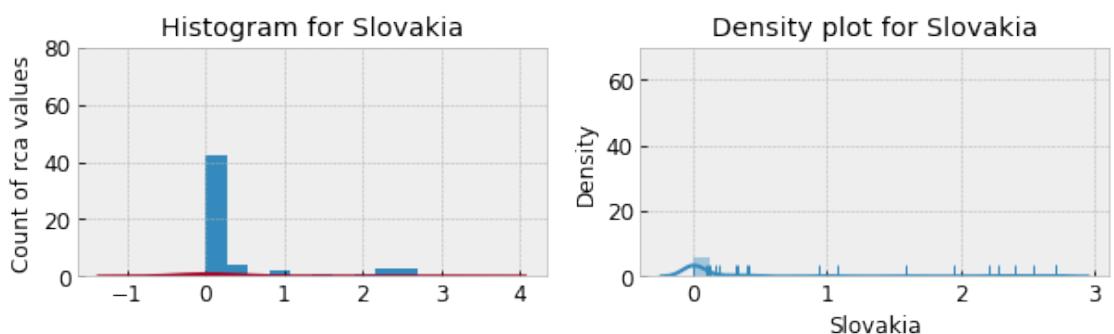
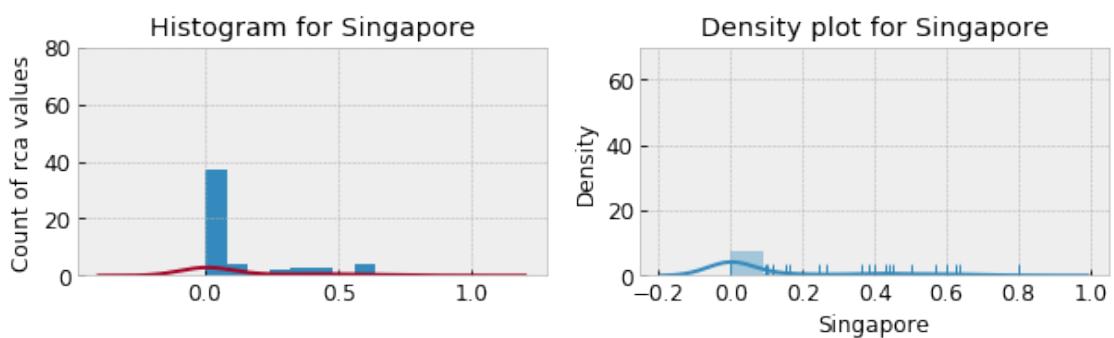
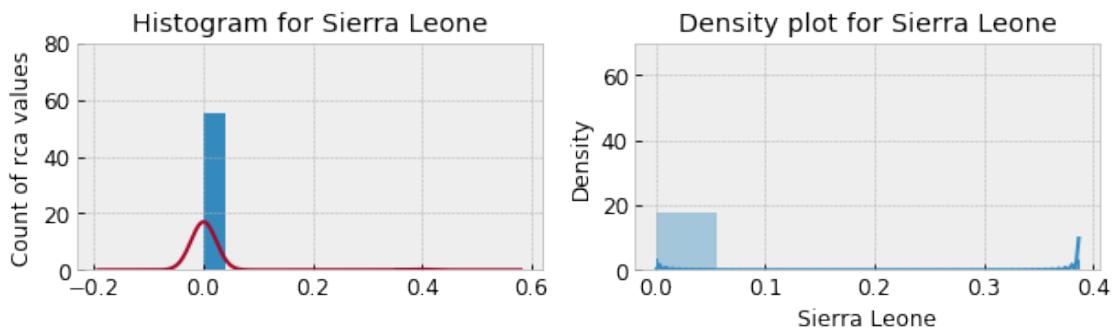
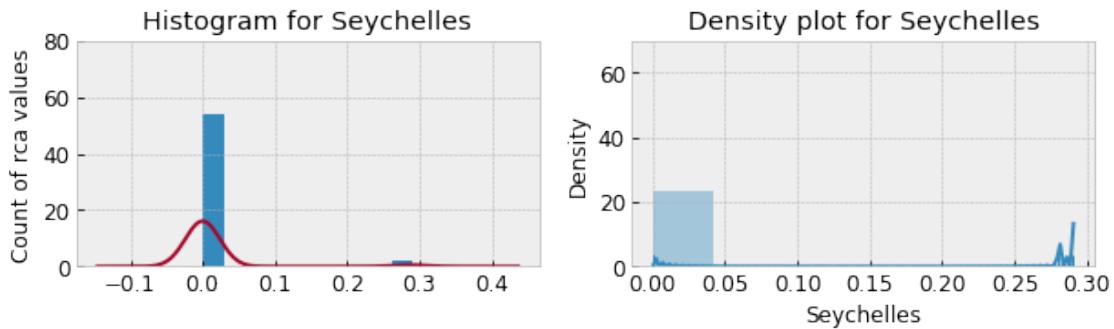


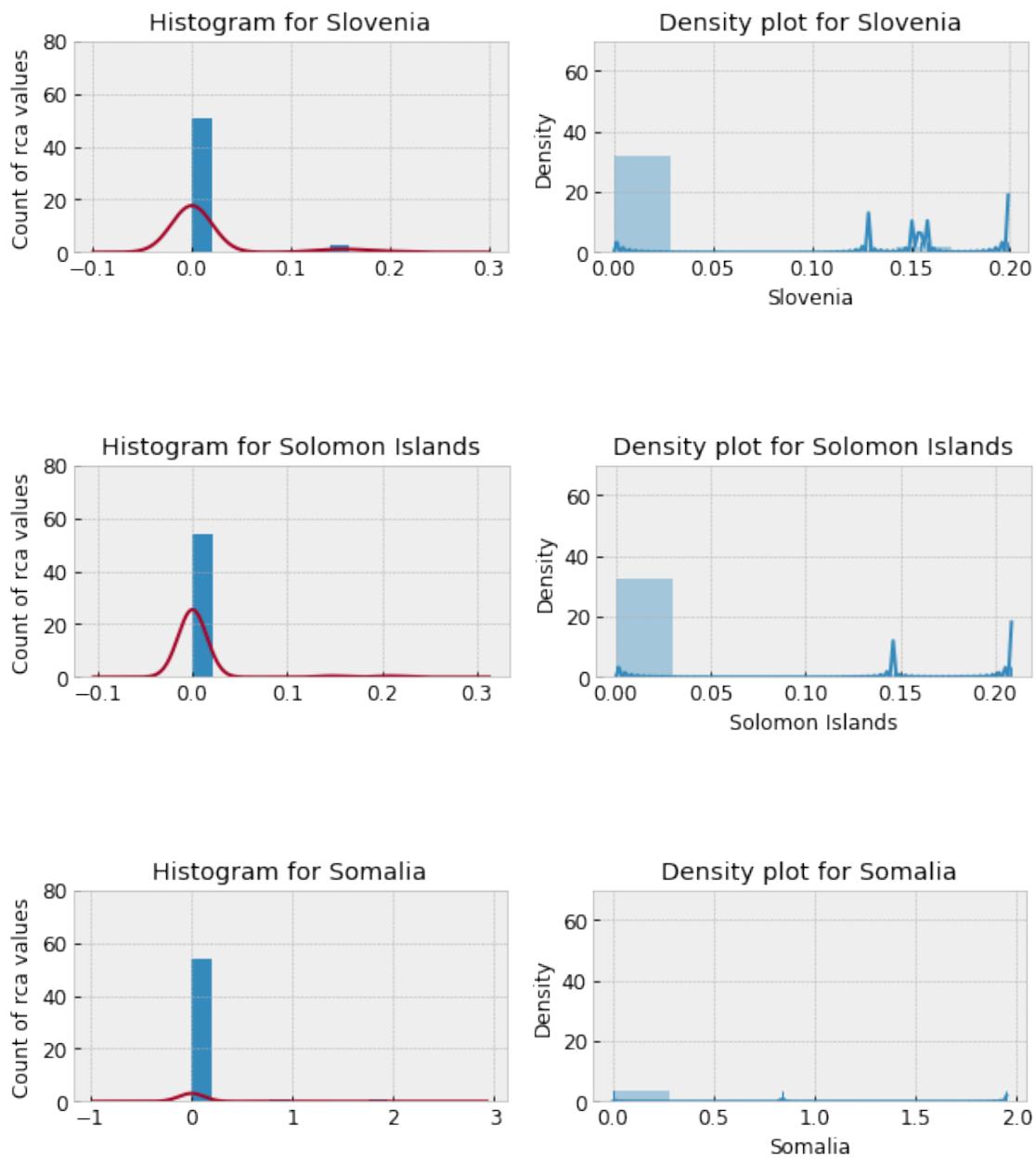


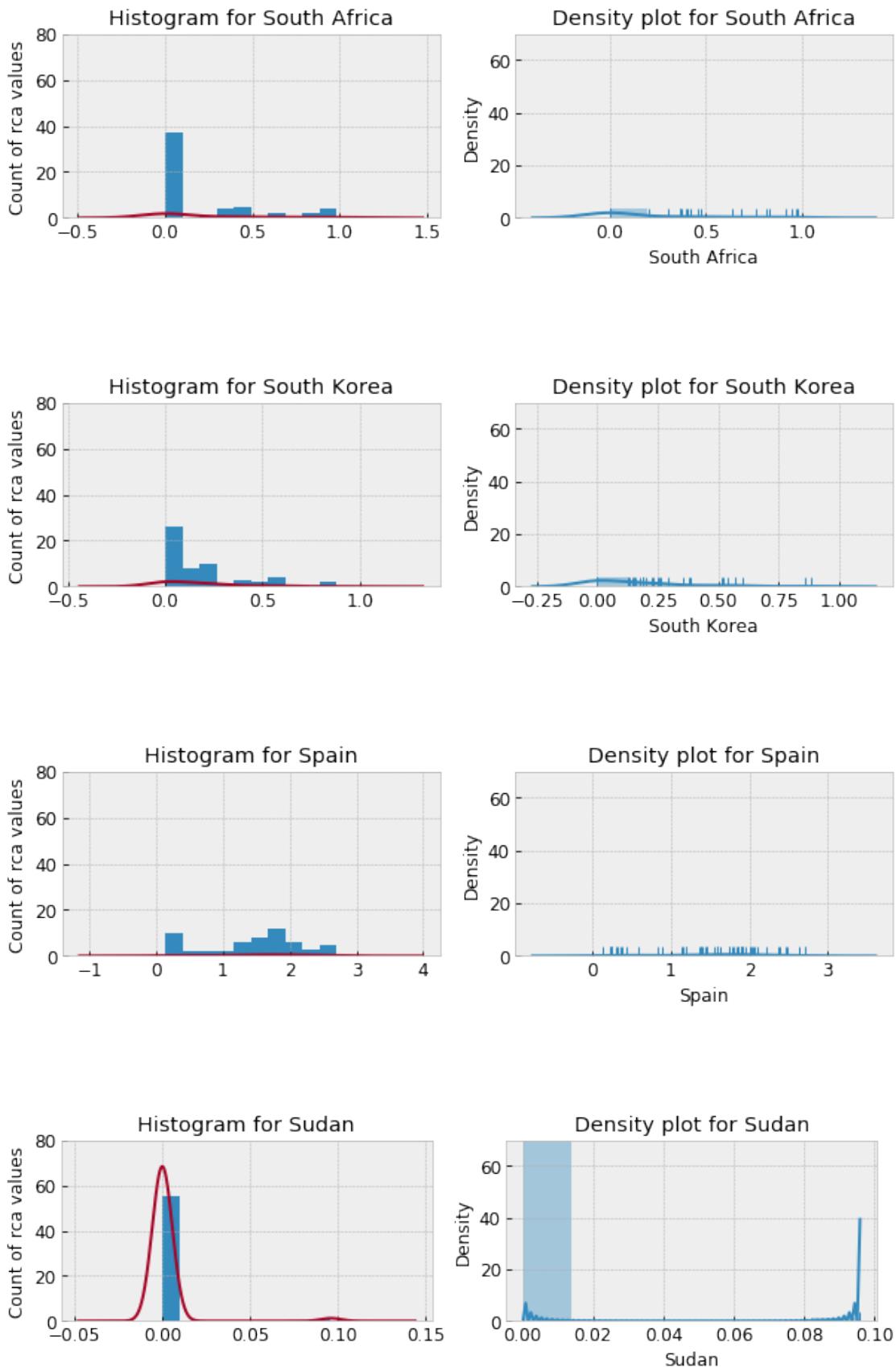


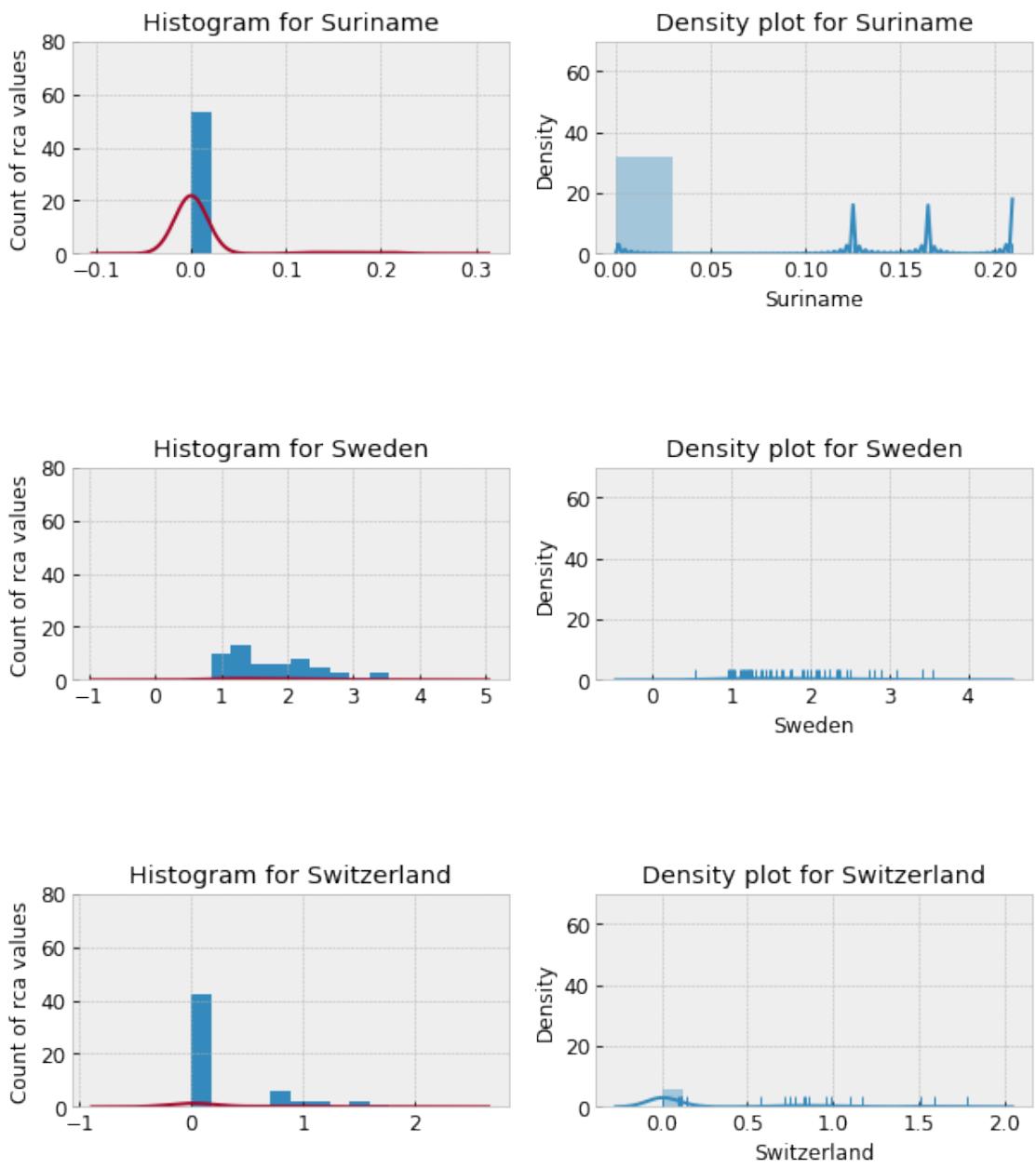


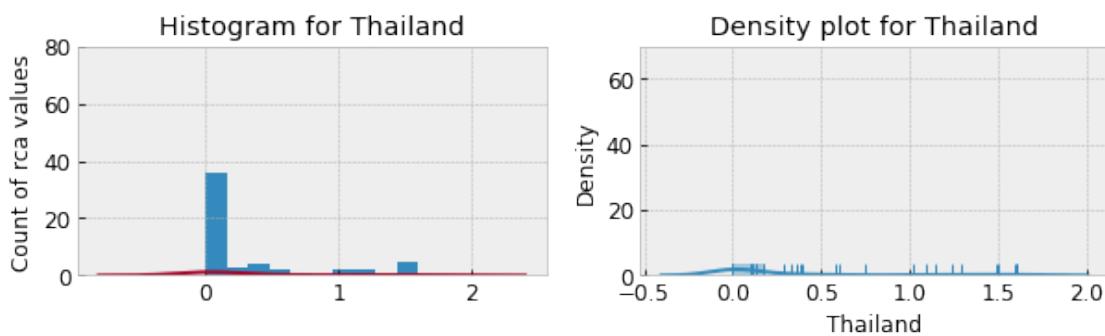
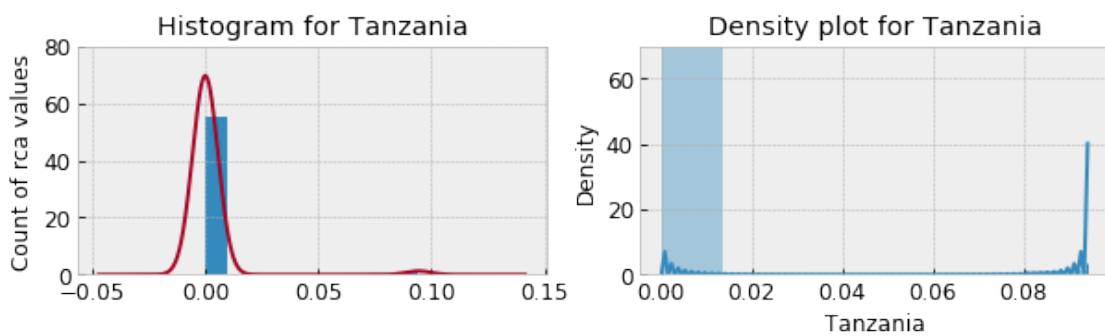
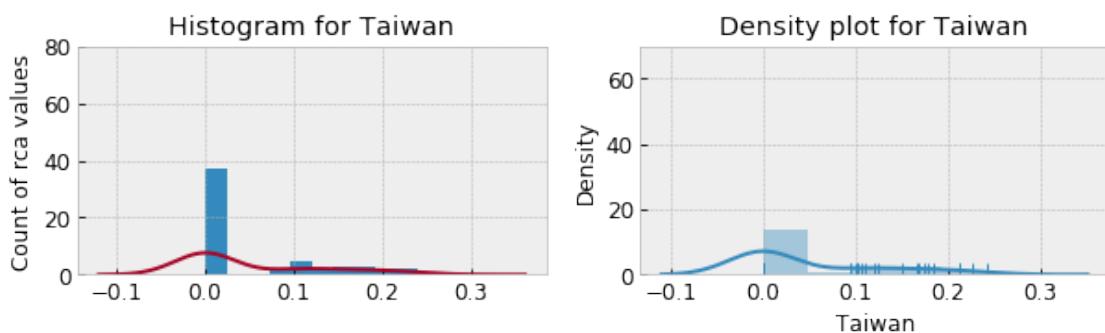
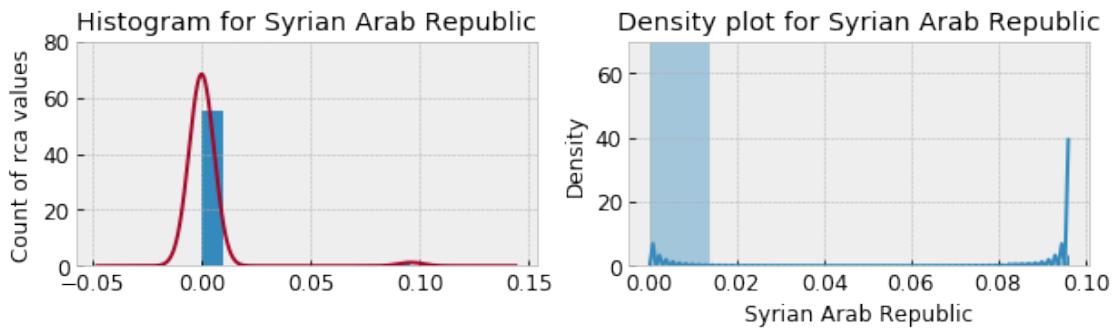


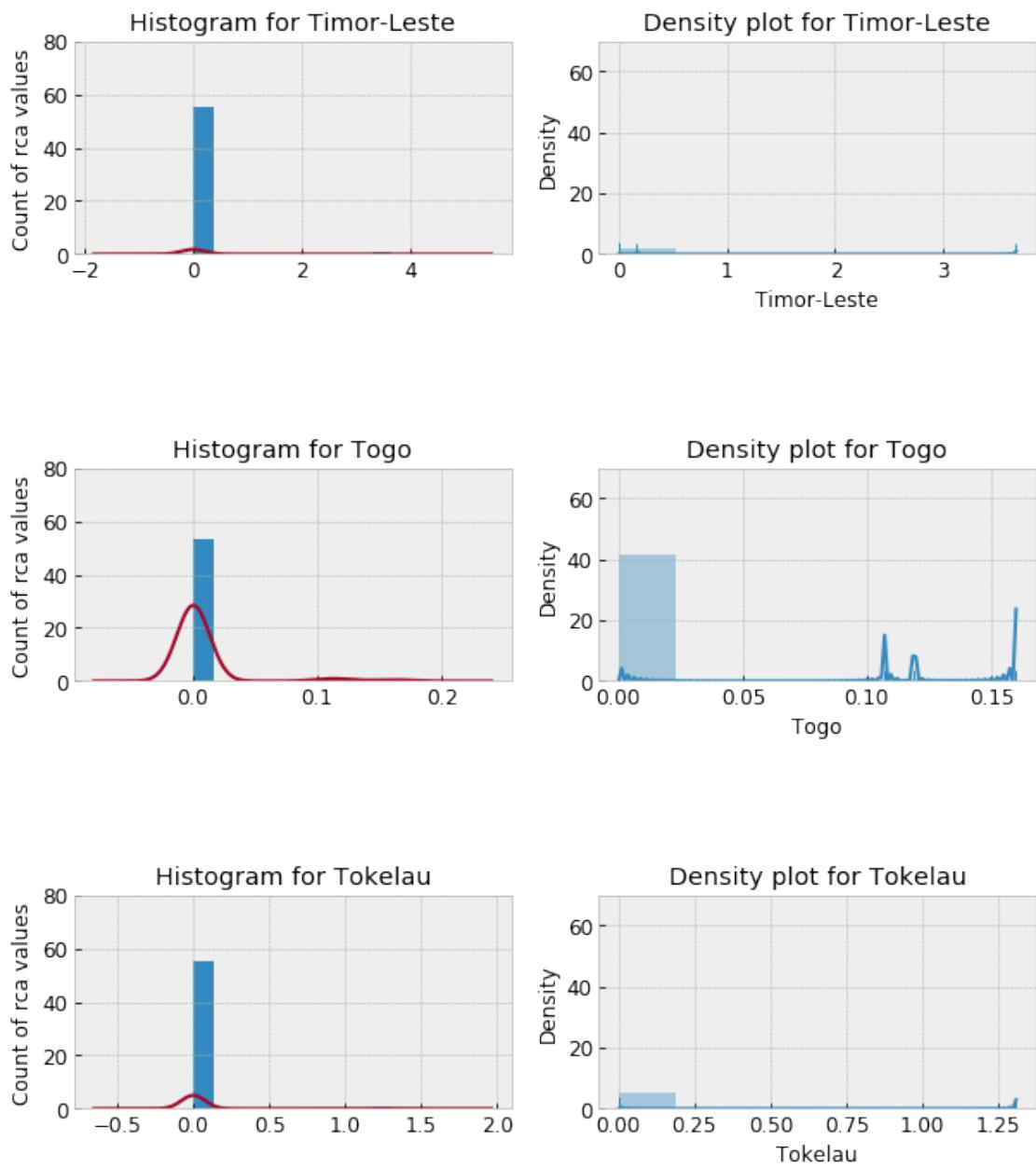


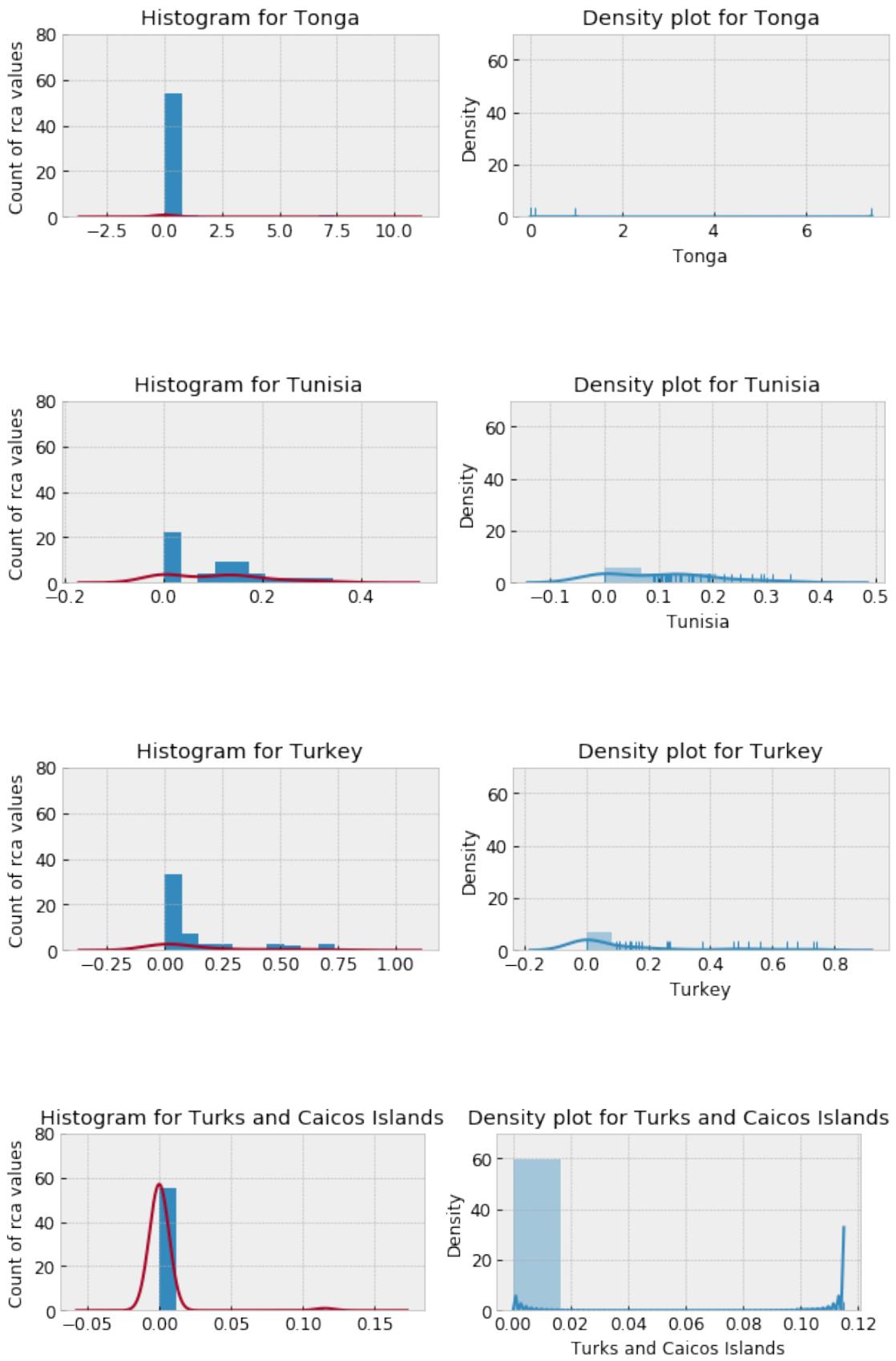


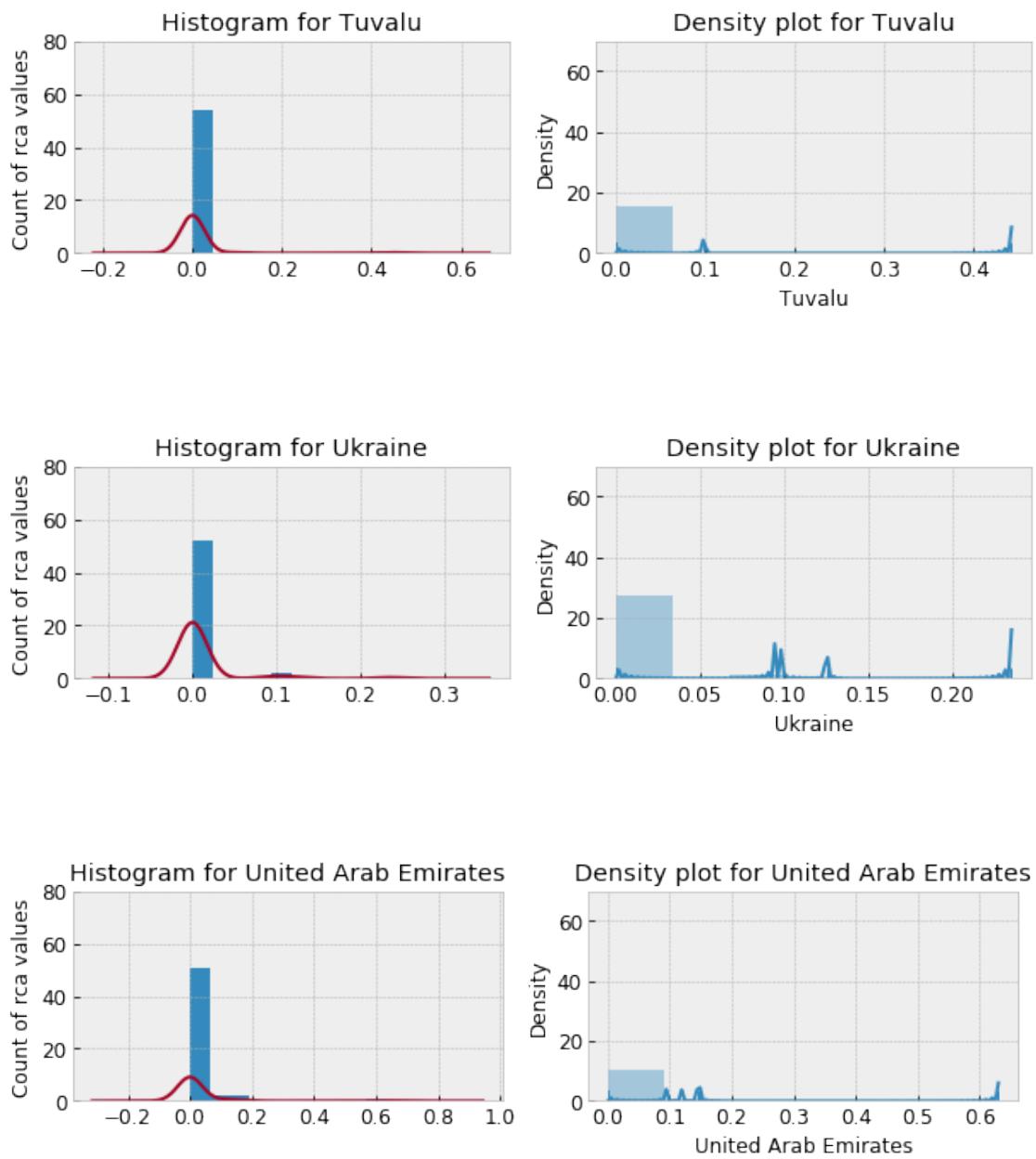


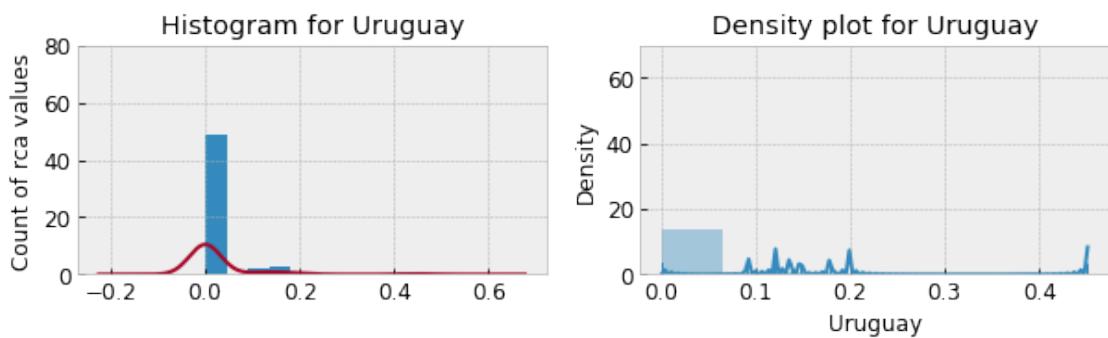
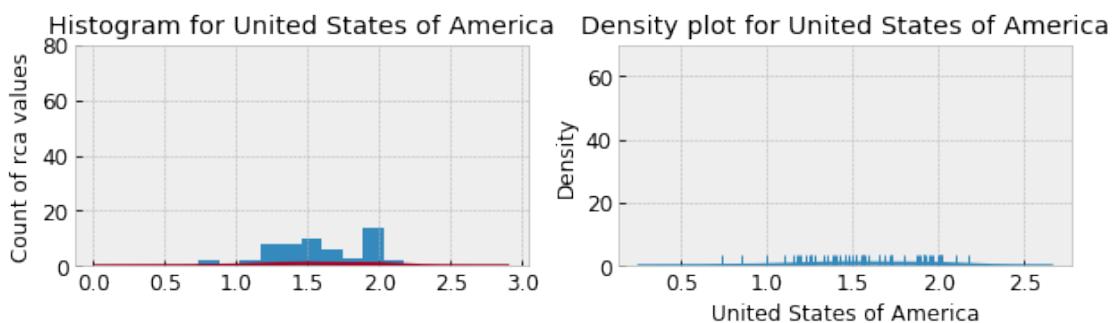
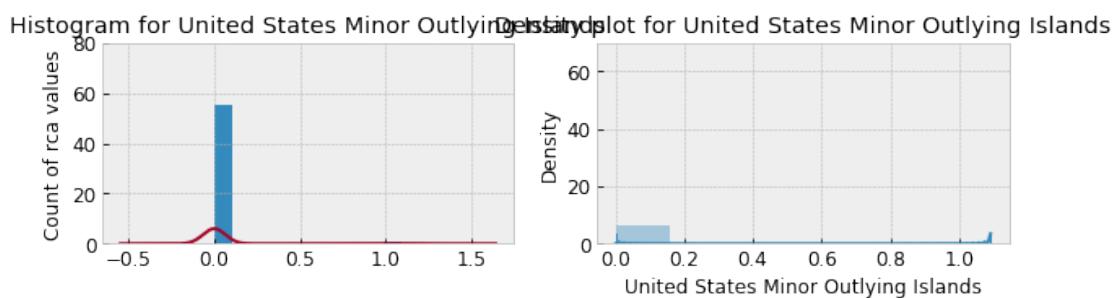
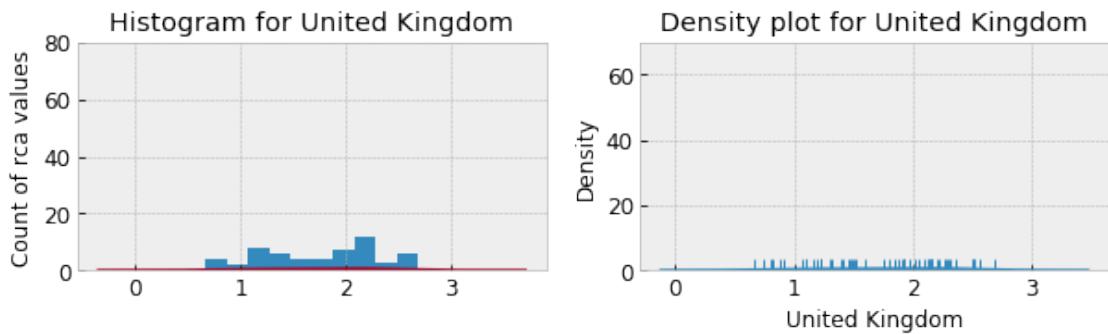


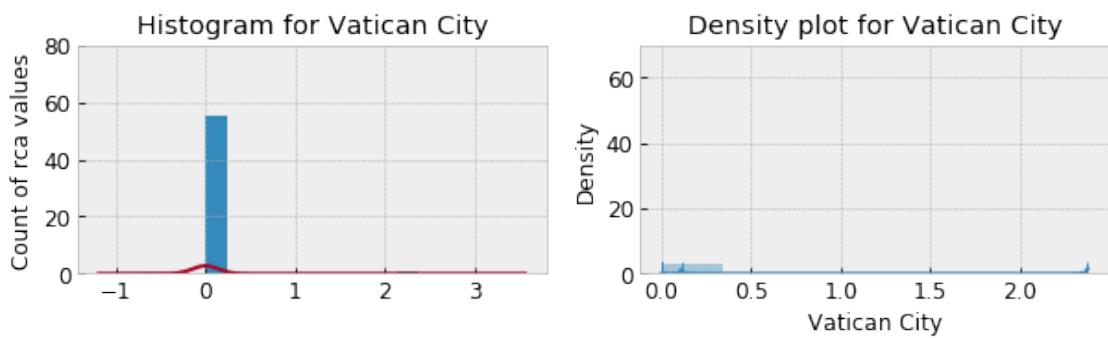
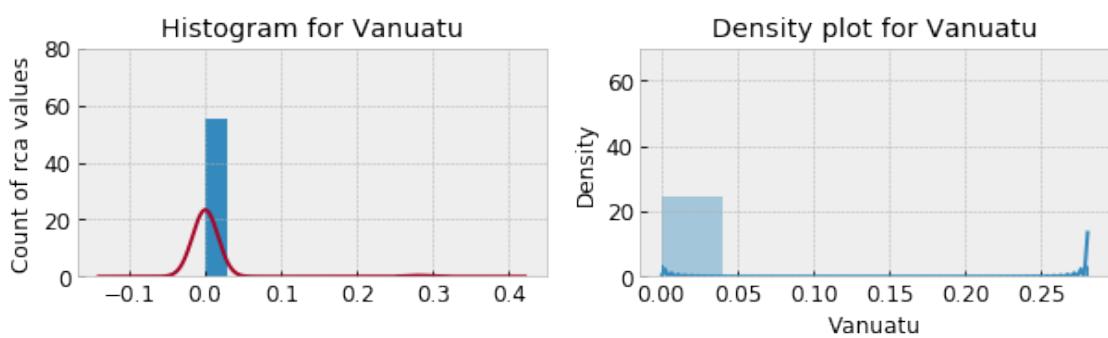
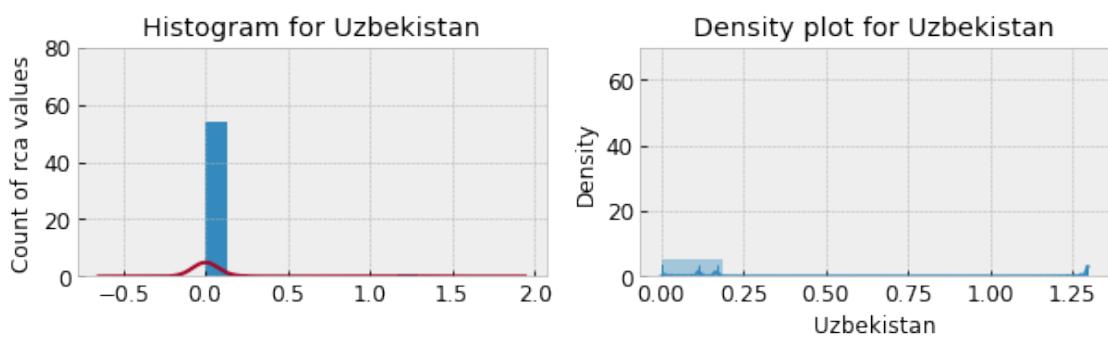


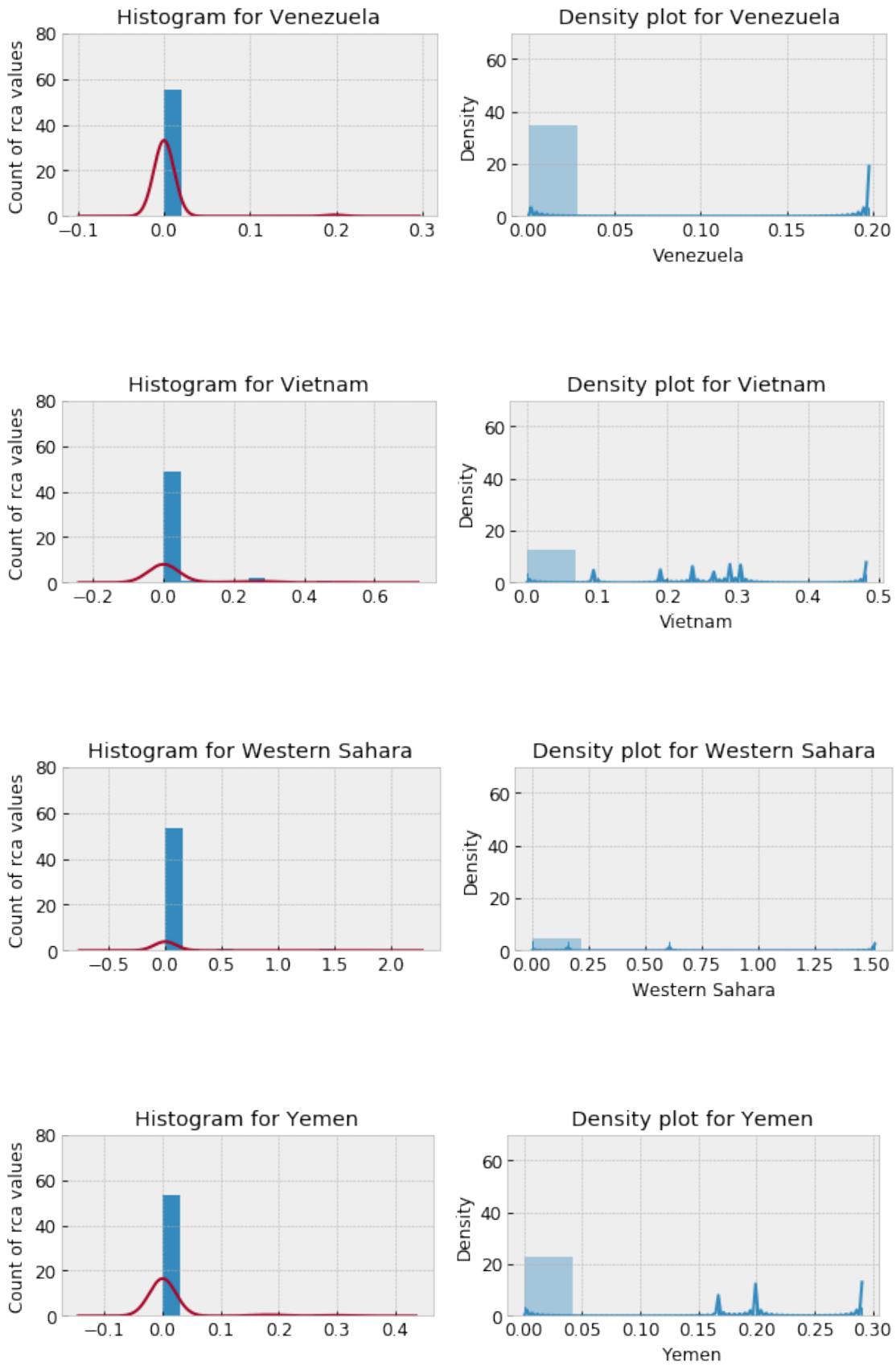


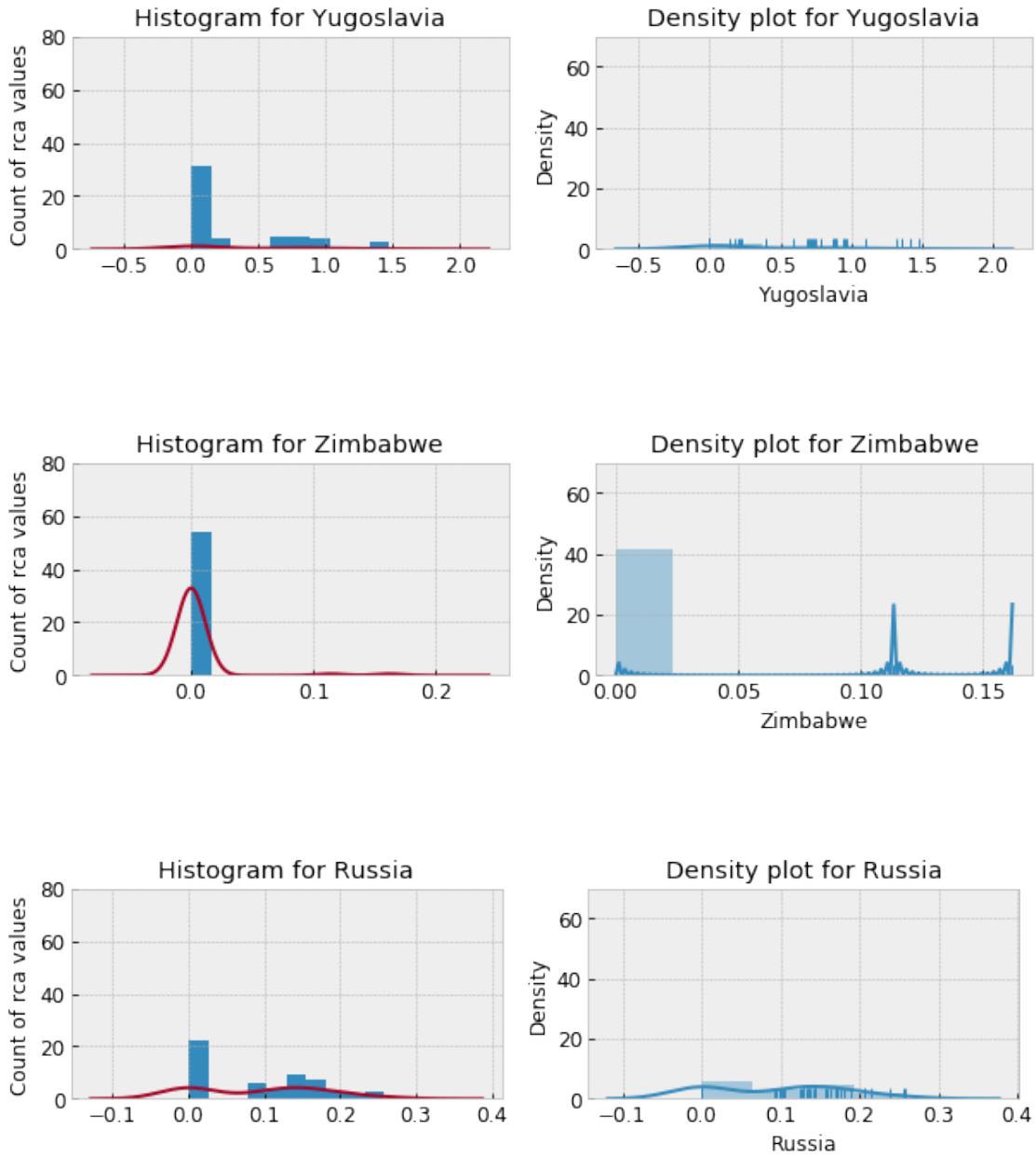












Box and Whisker Plots:

No high resolution in dataset to check: - what time series have similar median values across different years - whether there is a steady increase in the spread, or middle 50% of the data (boxes) over time - best model if considering seasonality

```
[142]: df_grps.info()
```

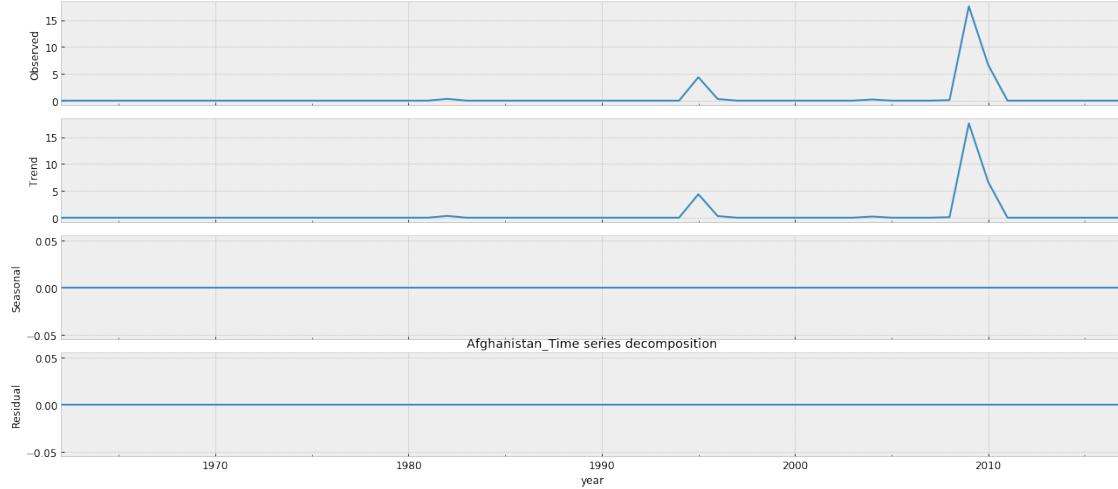
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 56 entries, 1962-01-01 to 2017-01-01
Columns: 183 entries, Afghanistan to Russia
dtypes: float64(183)
memory usage: 80.5 KB
```

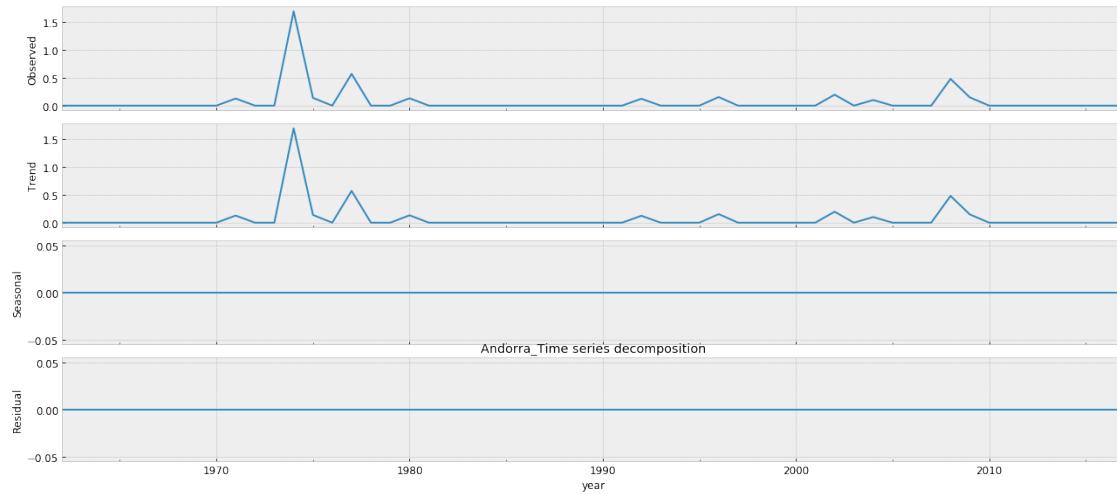
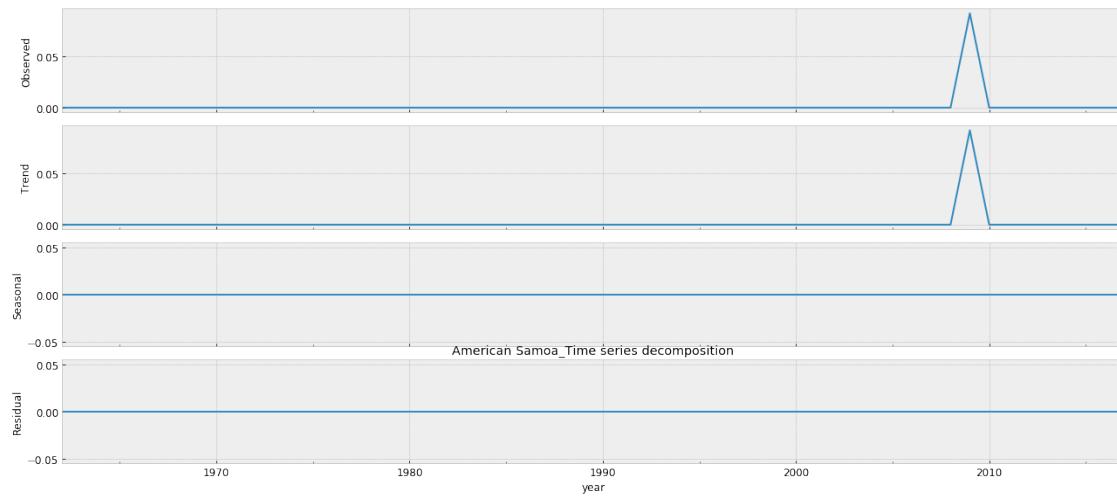
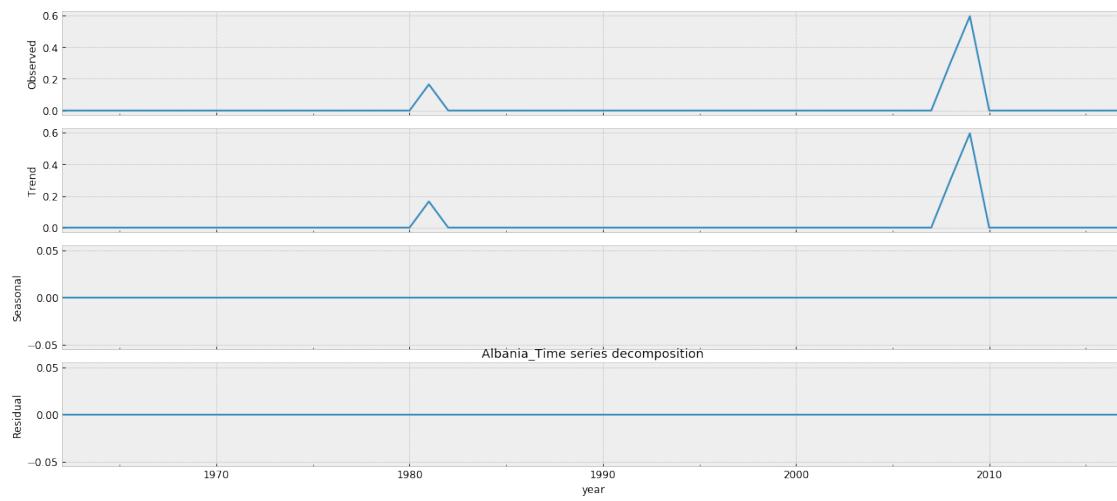
Decomposing using statsmodel: - statsmodels was used to perform a decomposition of the time series.
- Decomposition deconstructs a time series into several components, each representing one of the underlying categories of patterns.
- Statsmodels reveals the trend, seasonality, and residual components of our data.

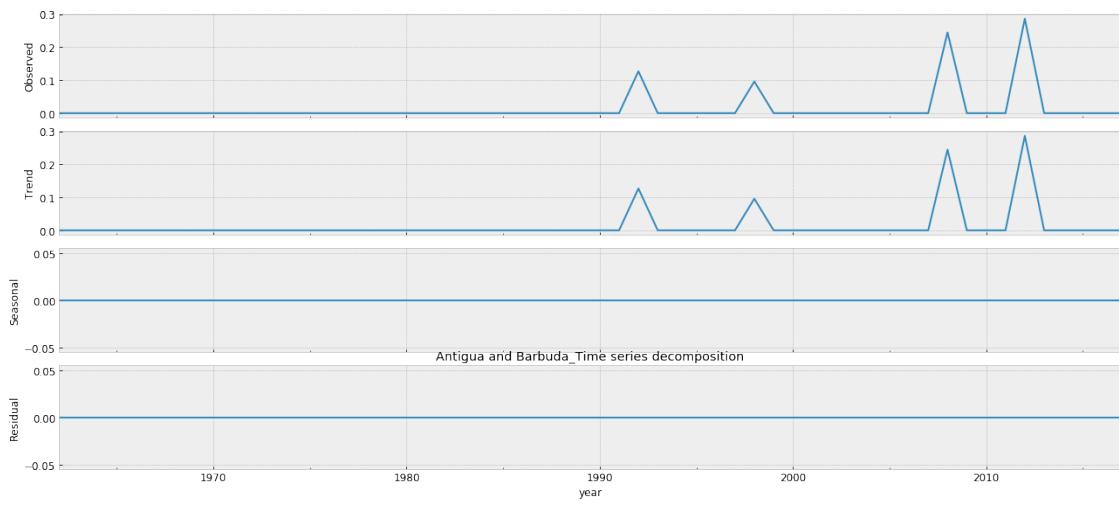
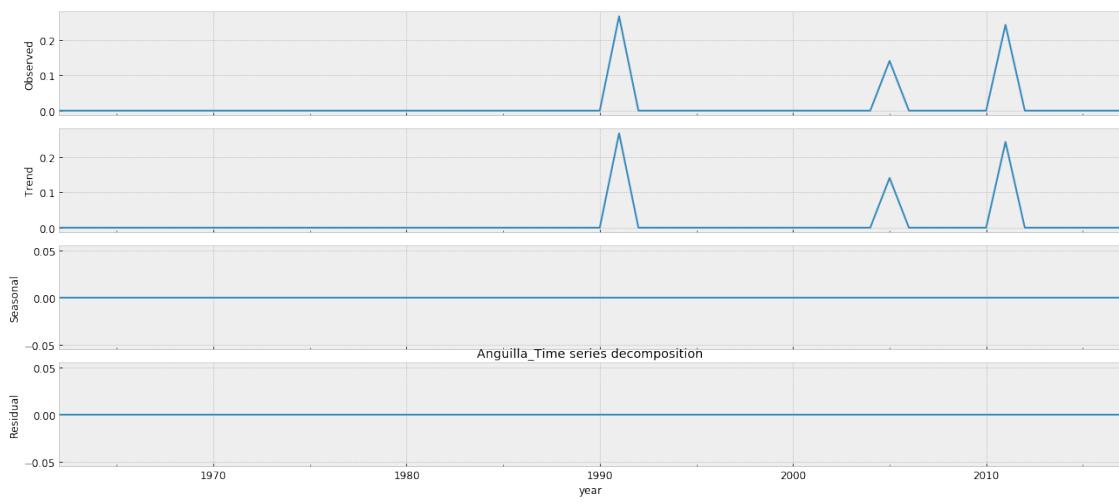
```
[146]: countries = df_grps.columns
```

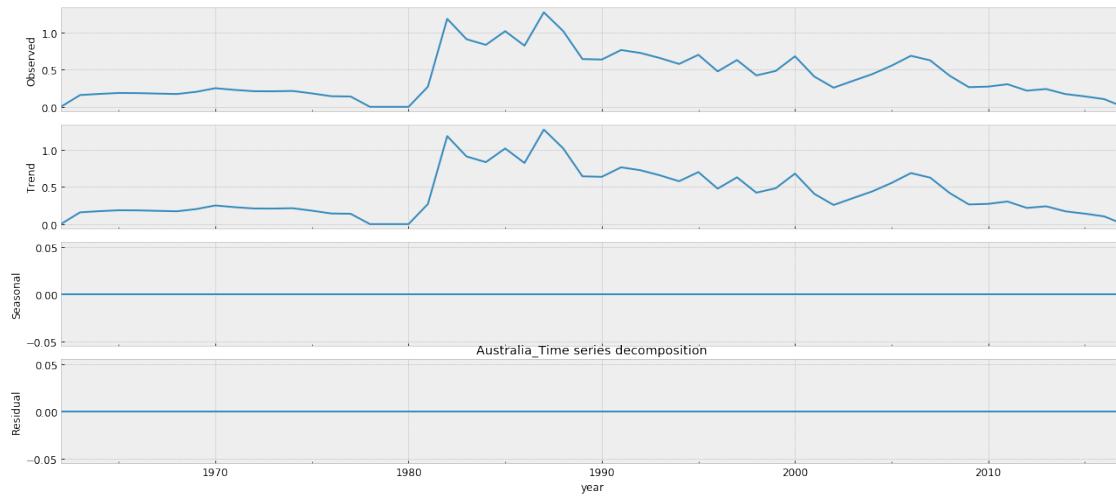
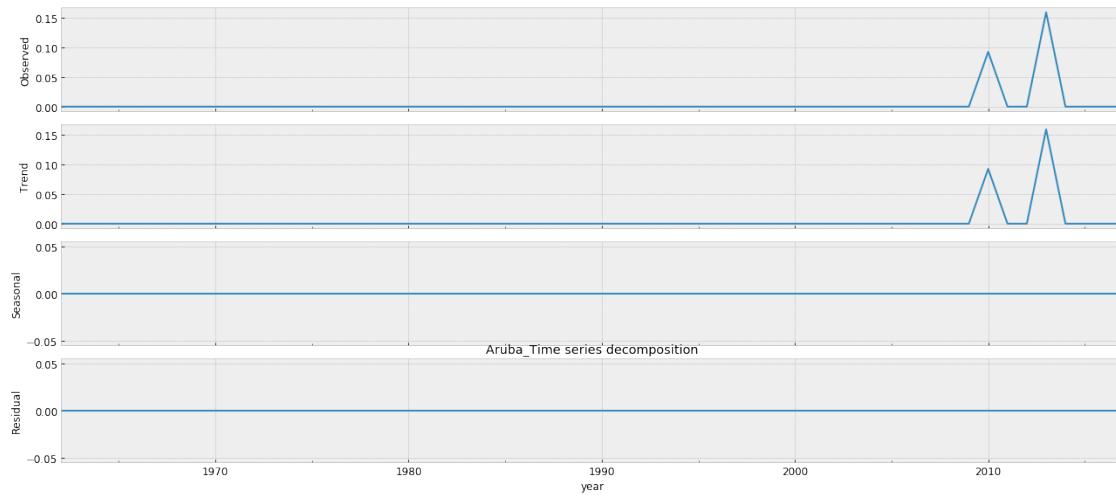
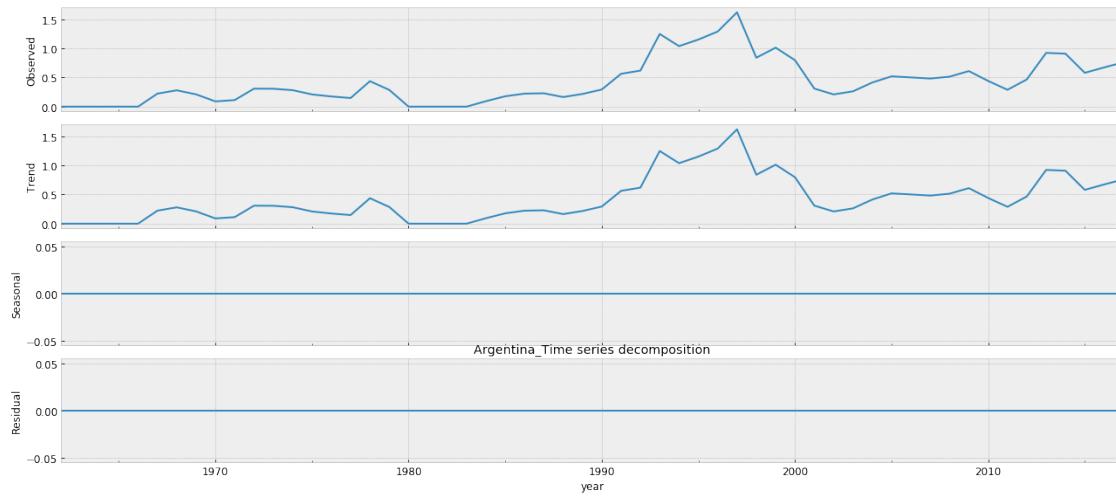
Decomposition

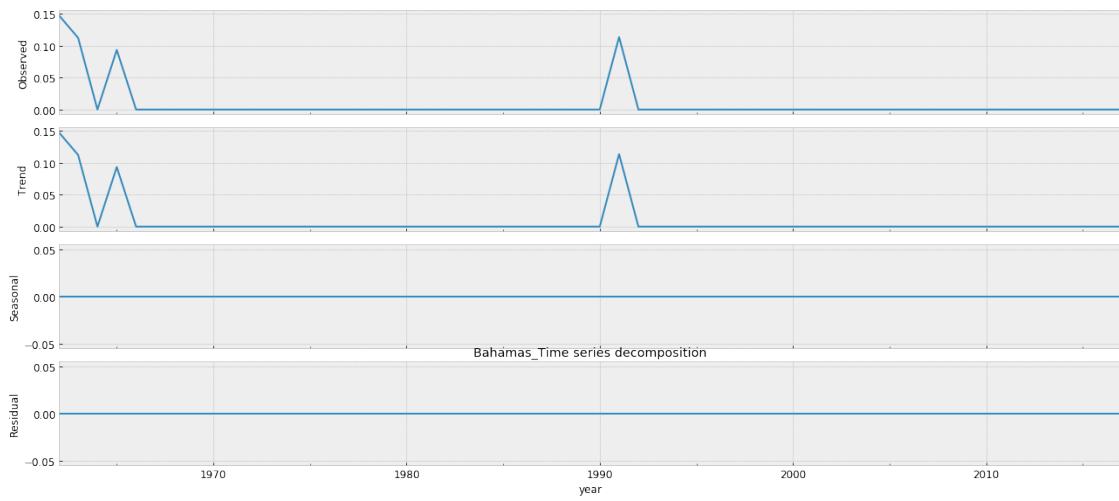
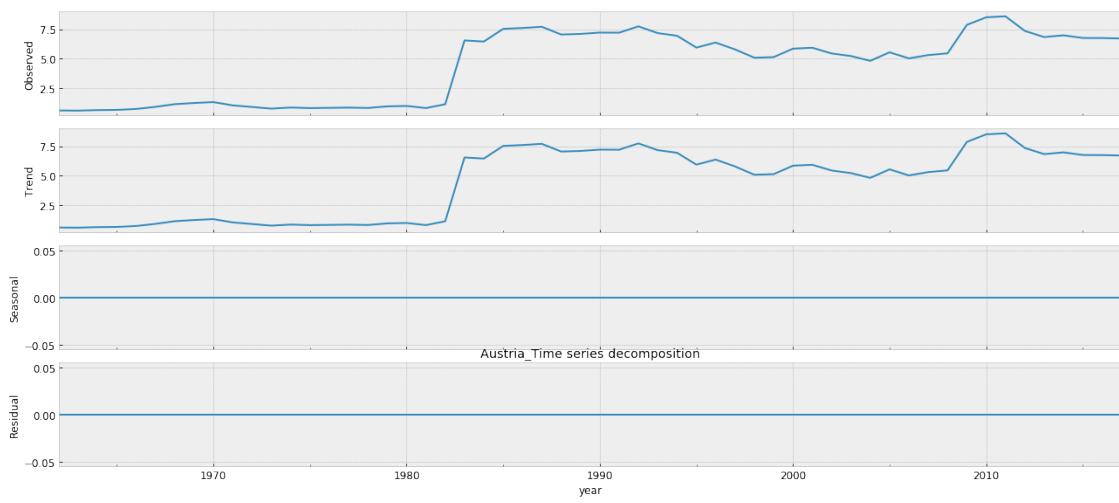
```
[147]: for country in countries:
    savefile = save_images(df_grps, 'decomposition')
    decomposition = sm.tsa.seasonal_decompose(df_grps[country], model='additive')
    fig = decomposition.plot()
    plt.title(f'{df_grps[country].name}_Time series decomposition', loc='center')
    rcParams['figure.figsize'] = (18, 8)
    plt.savefig(str(savefile))
    plt.show()
```

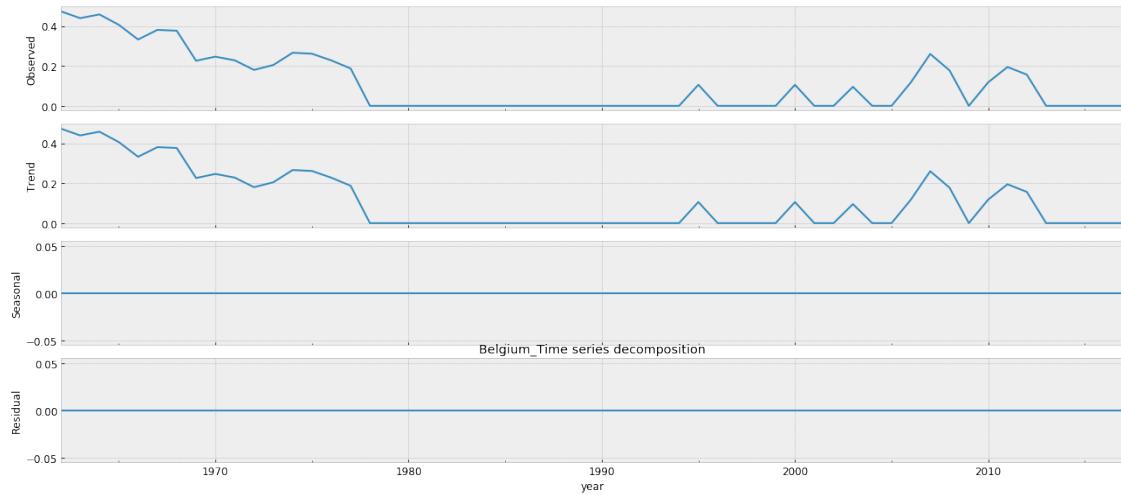
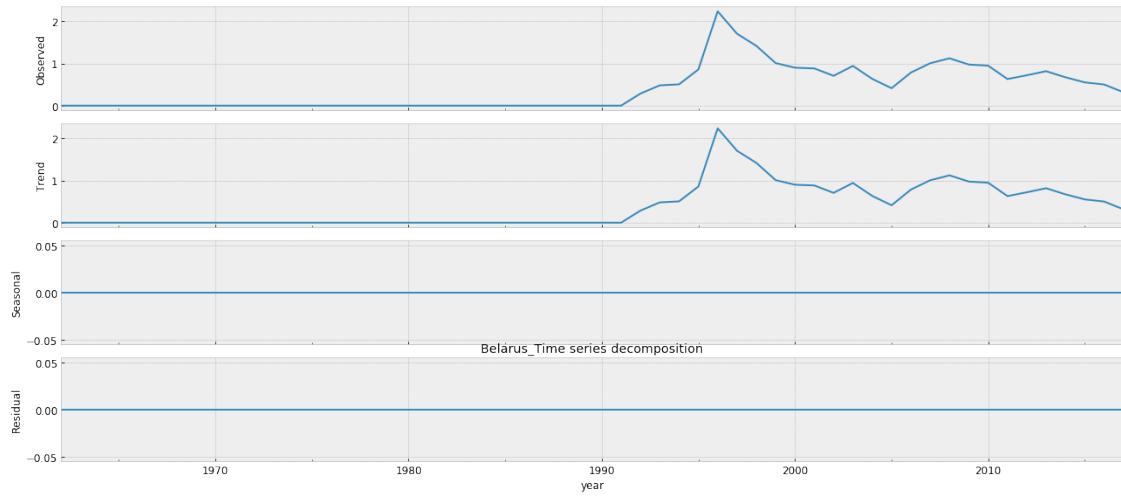
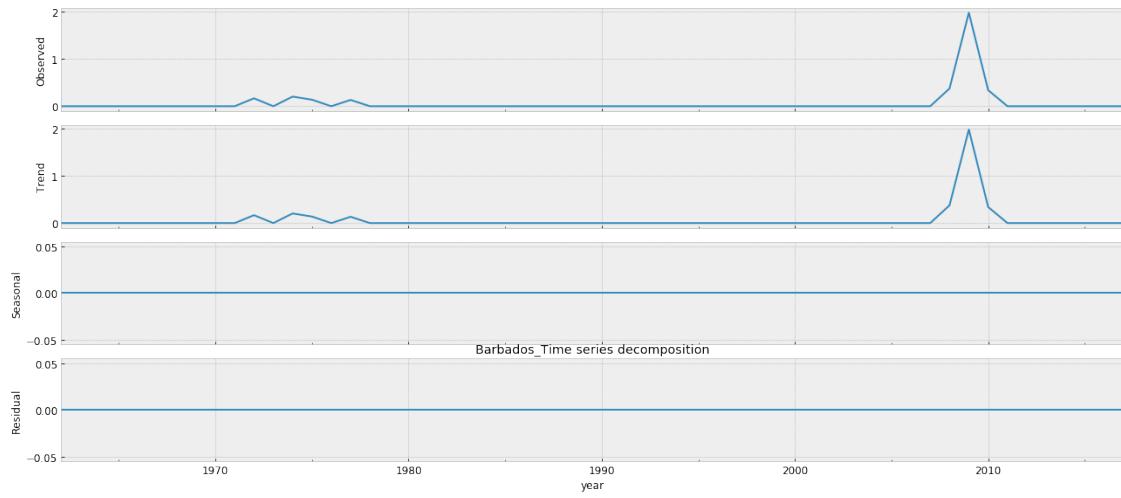


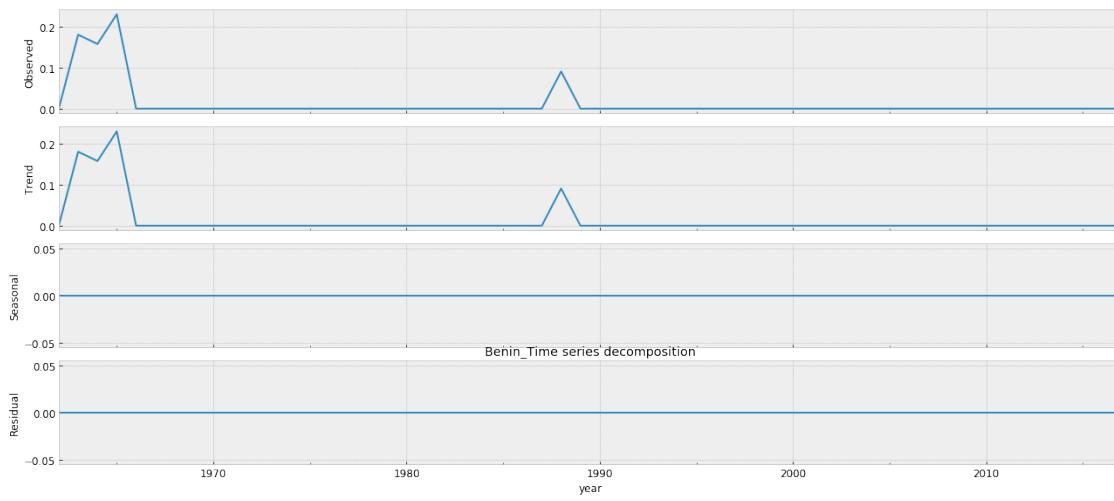
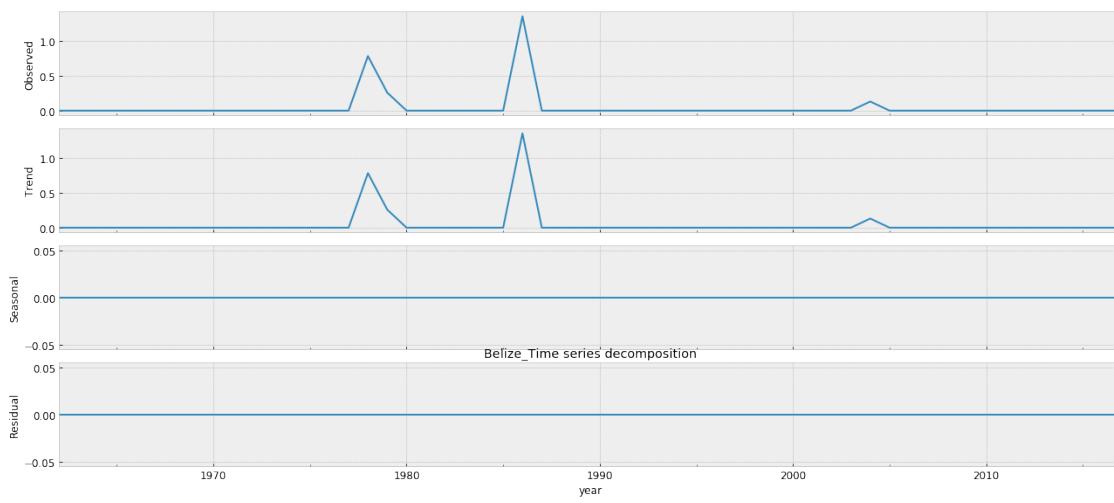


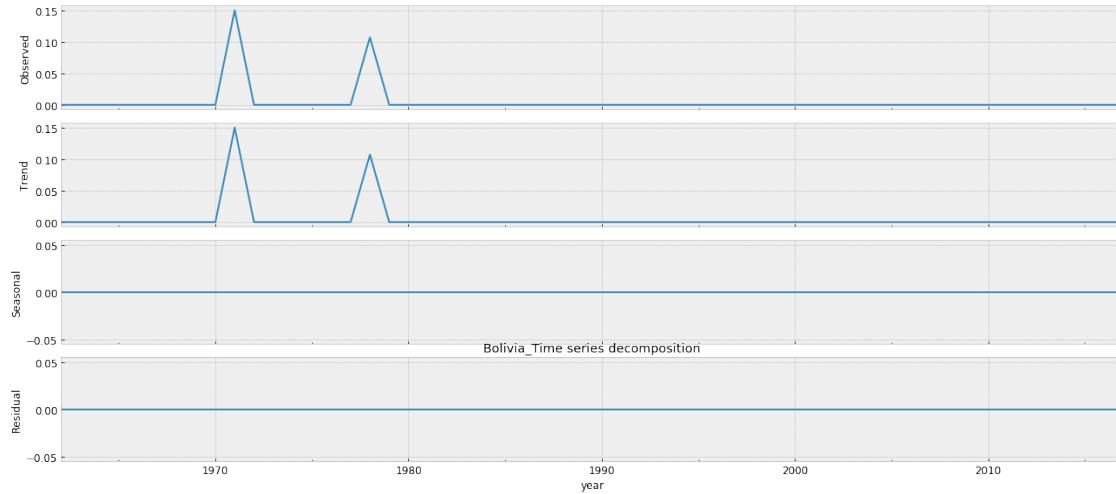
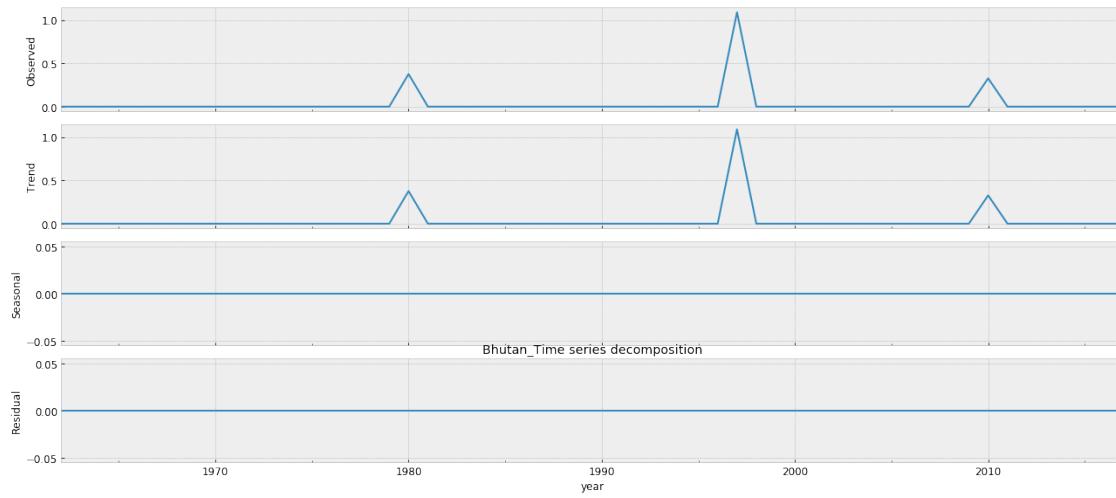
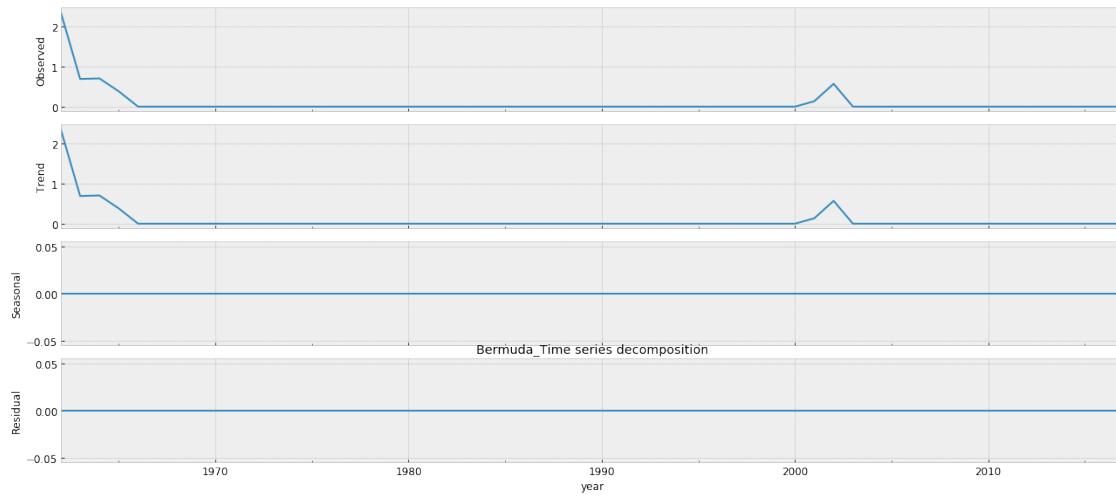


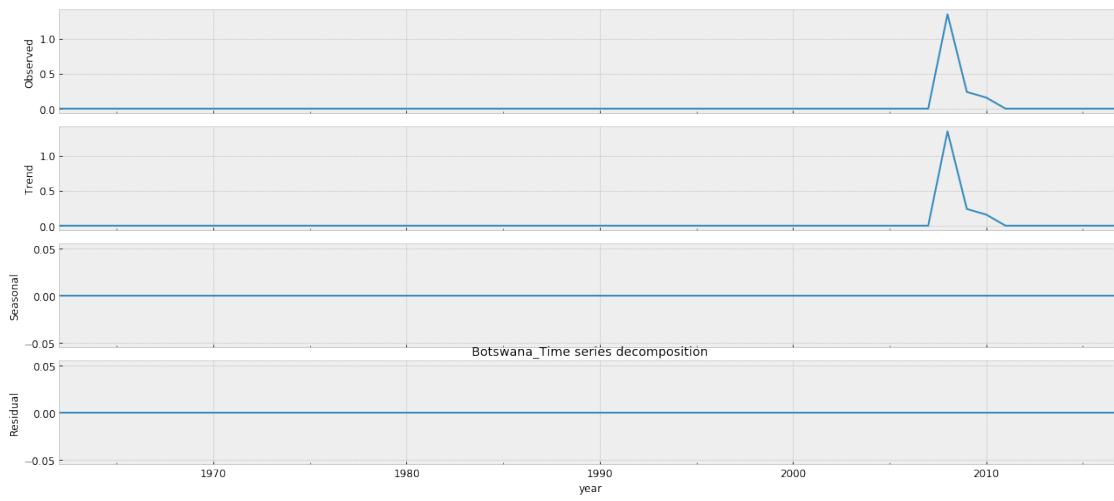
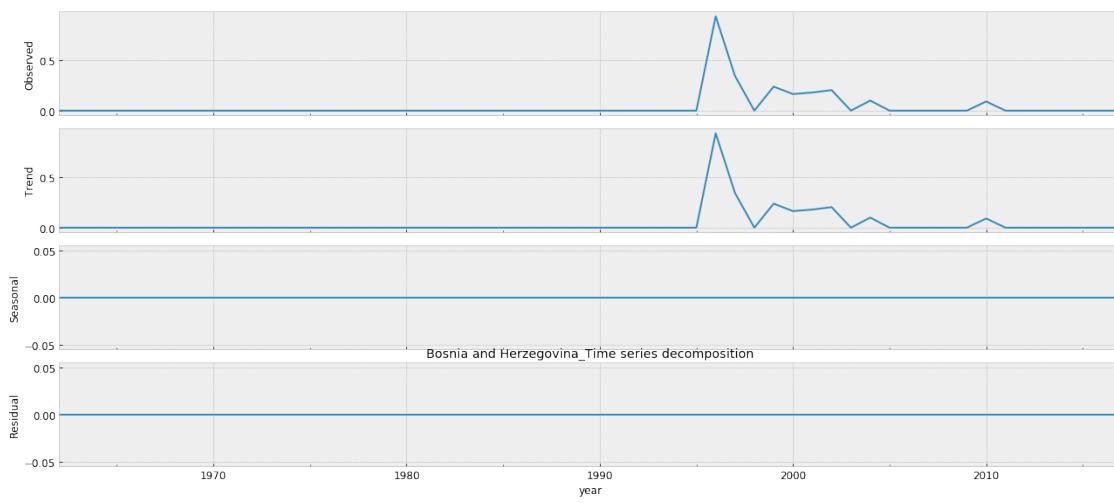


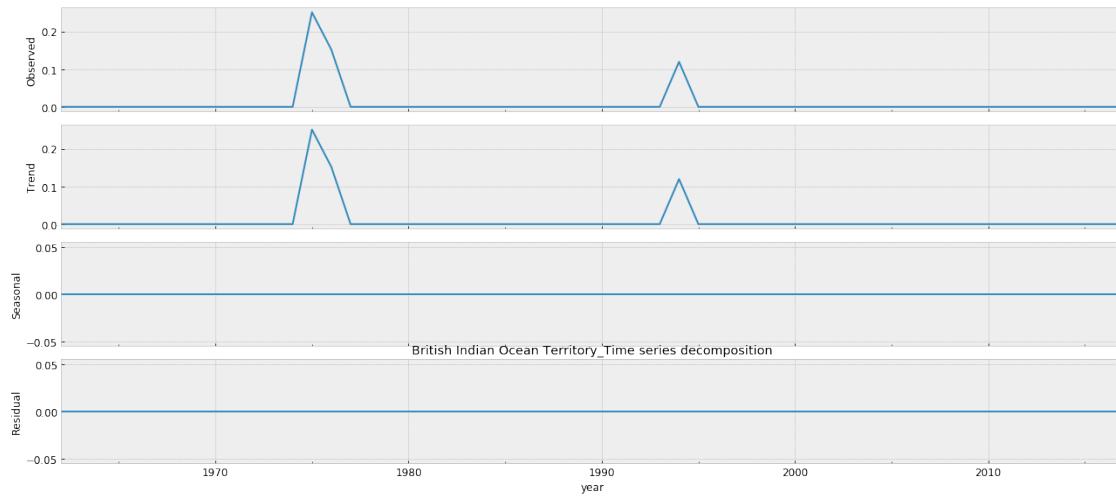
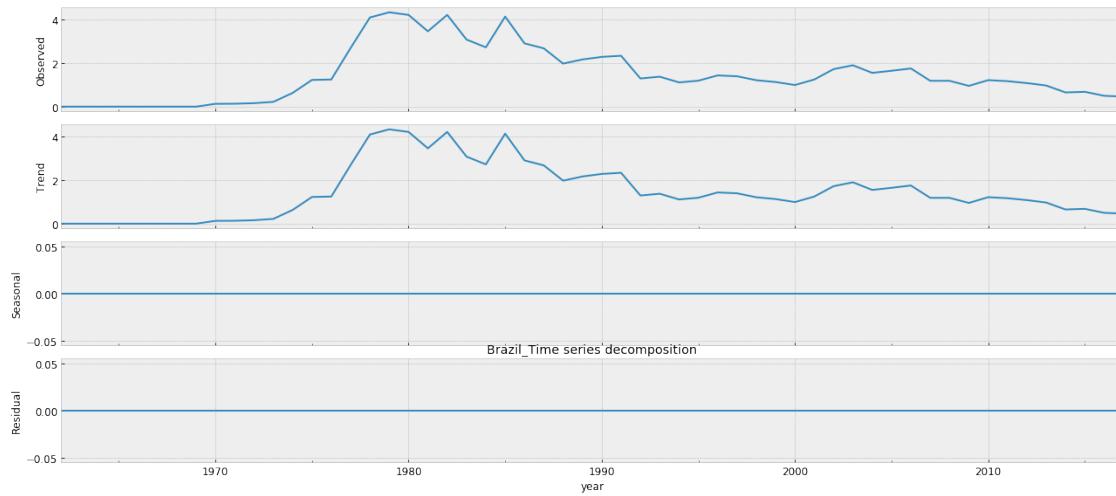
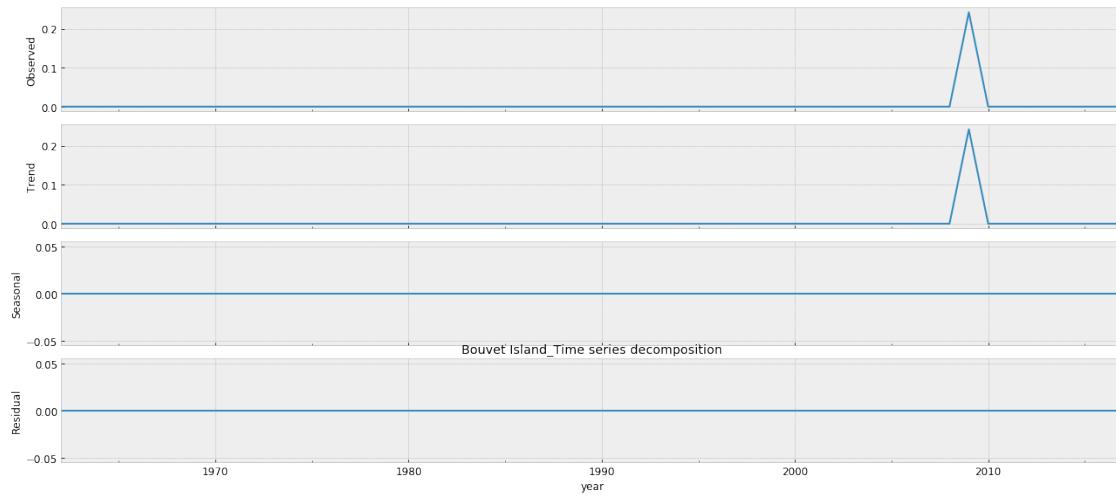


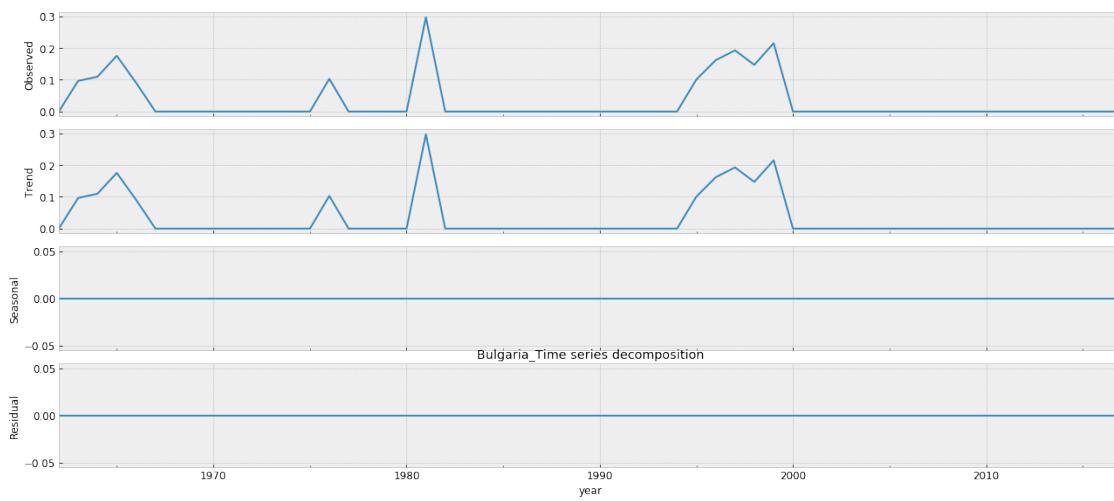
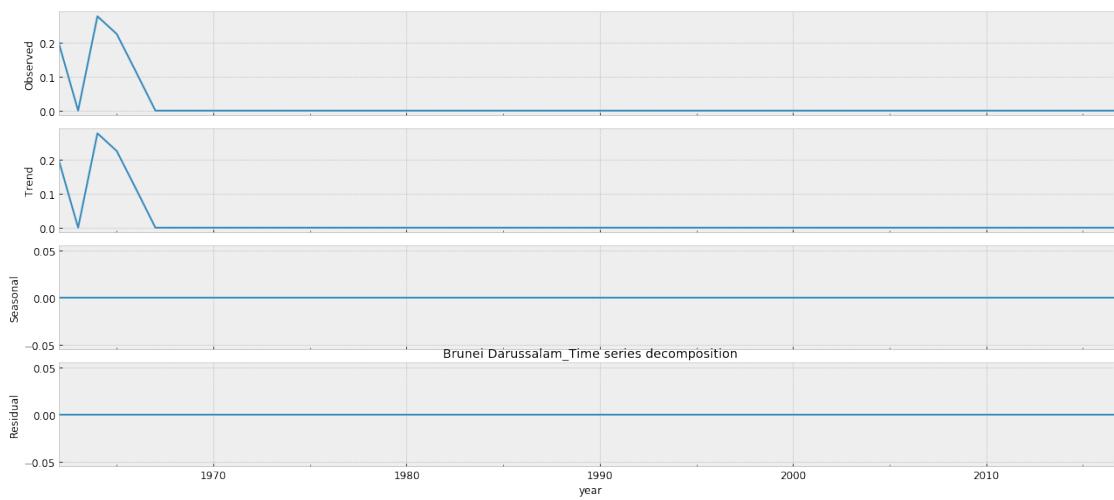


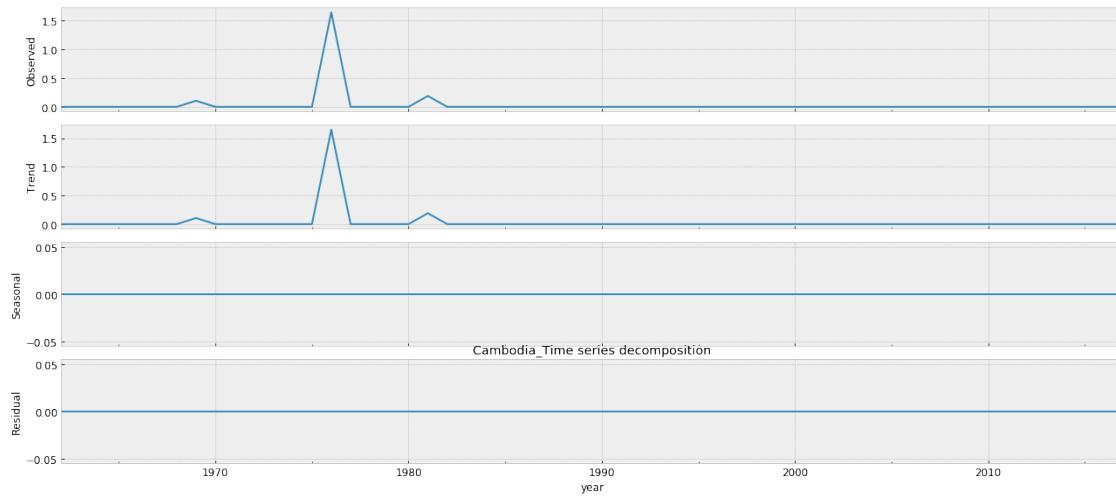
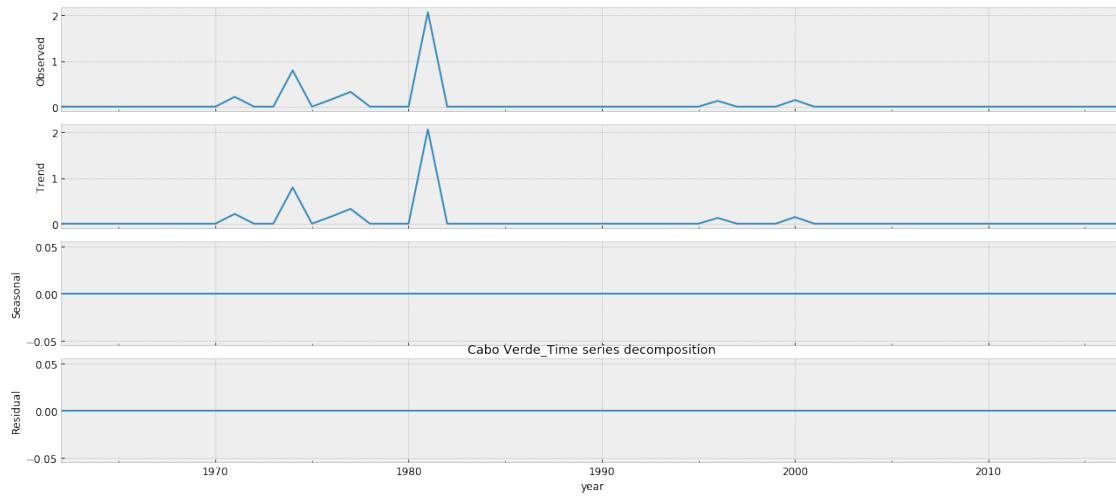
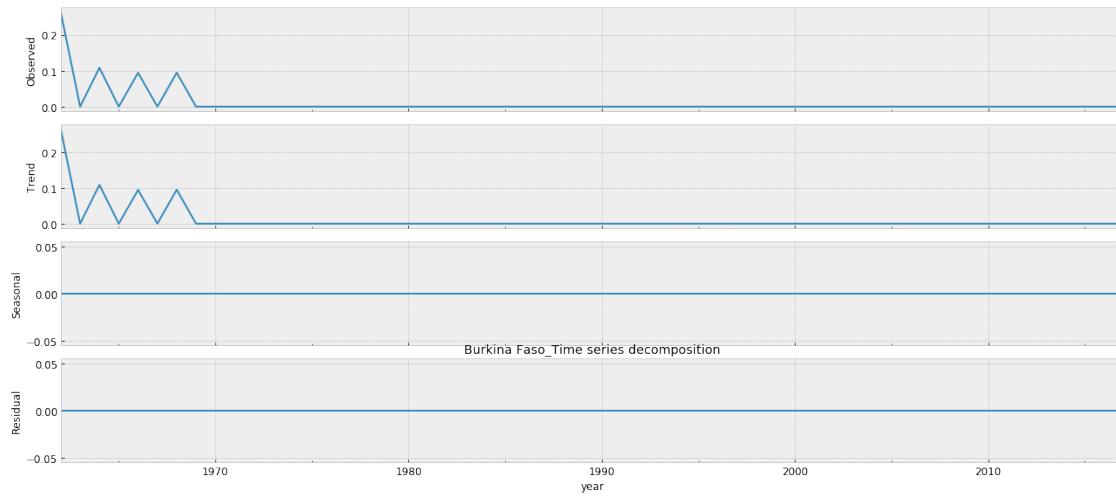


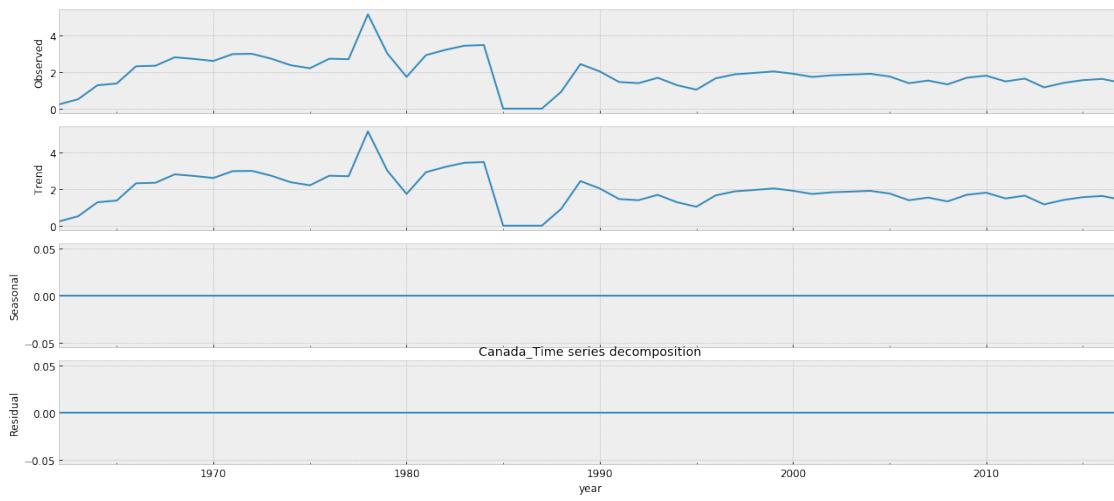
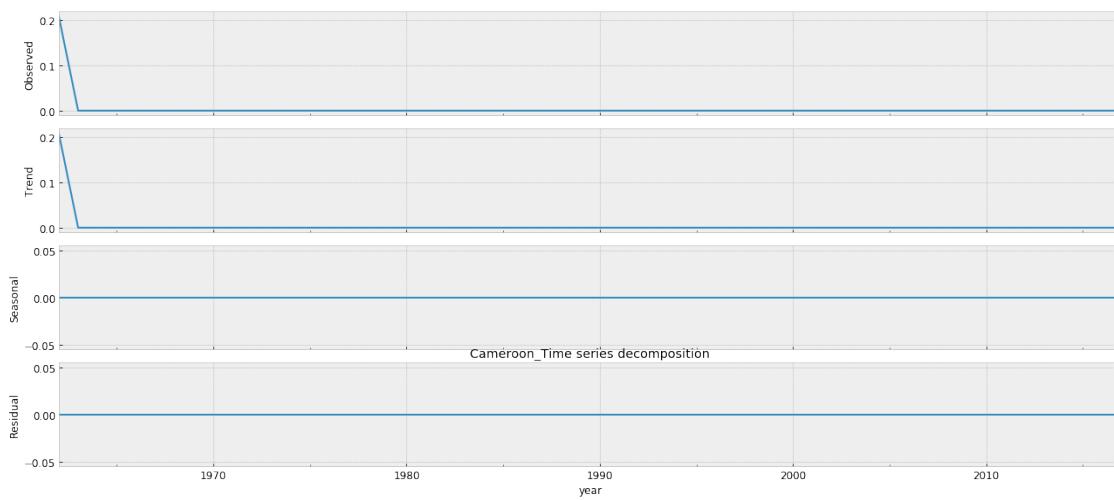


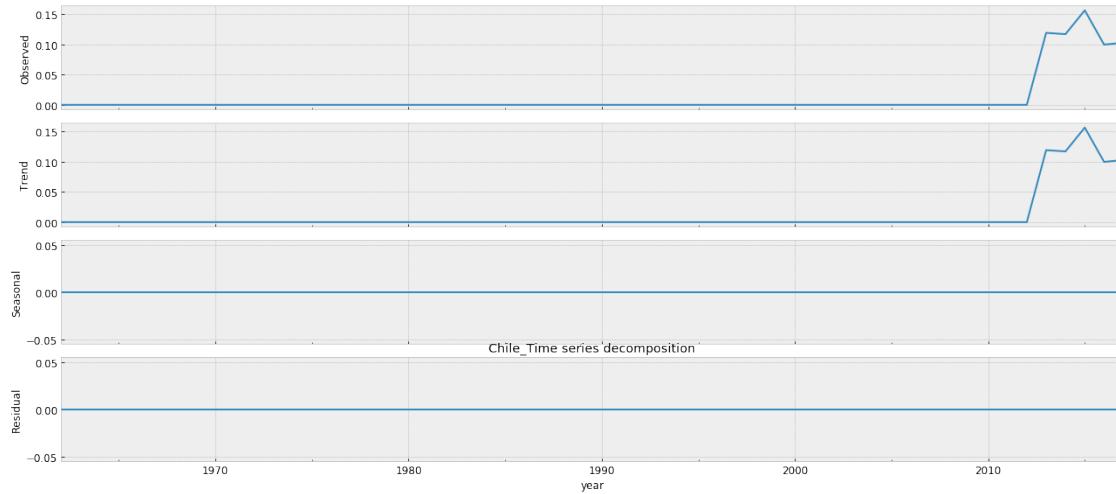
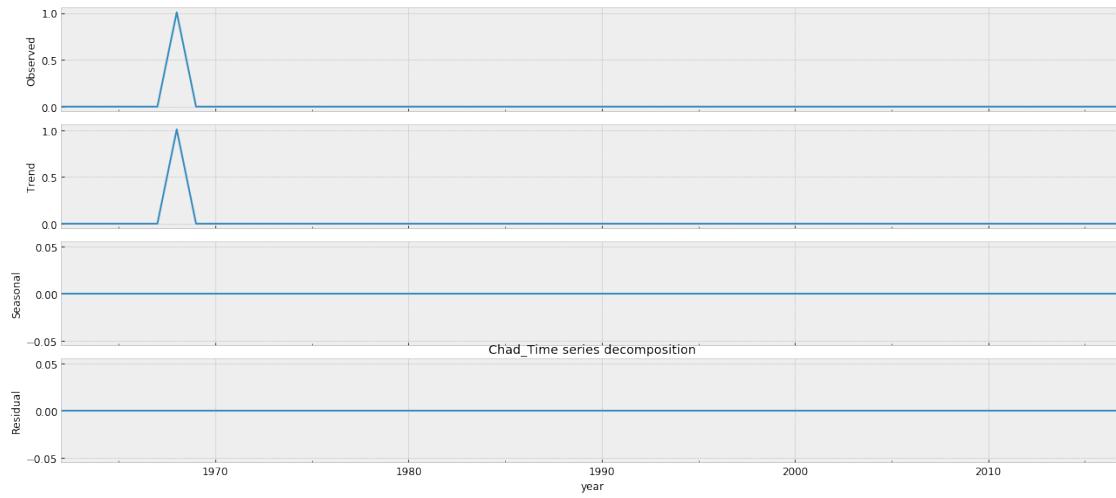
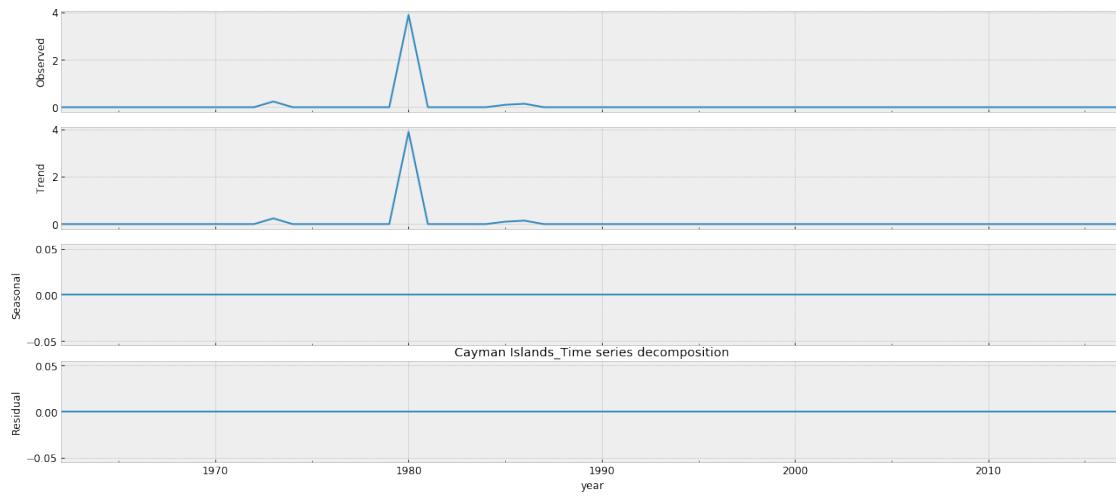


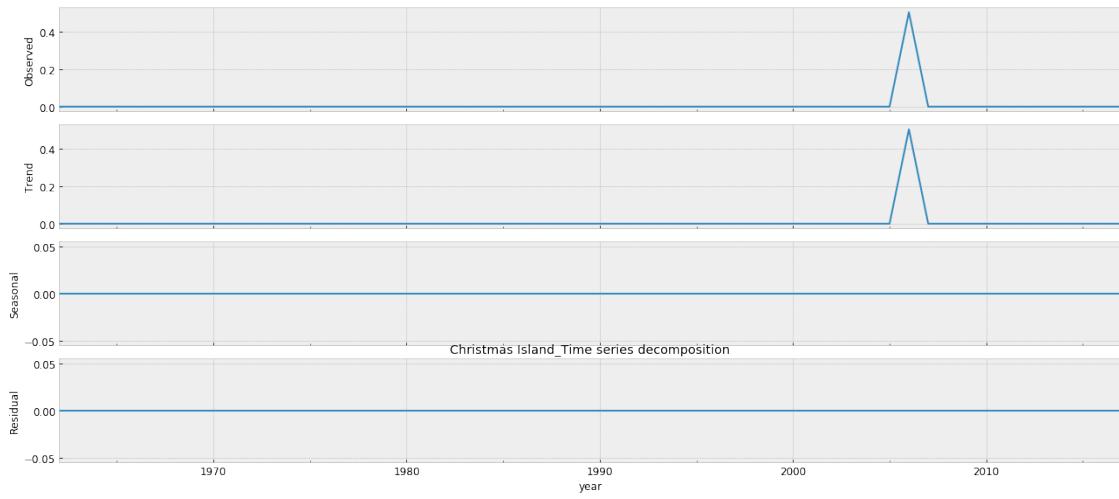
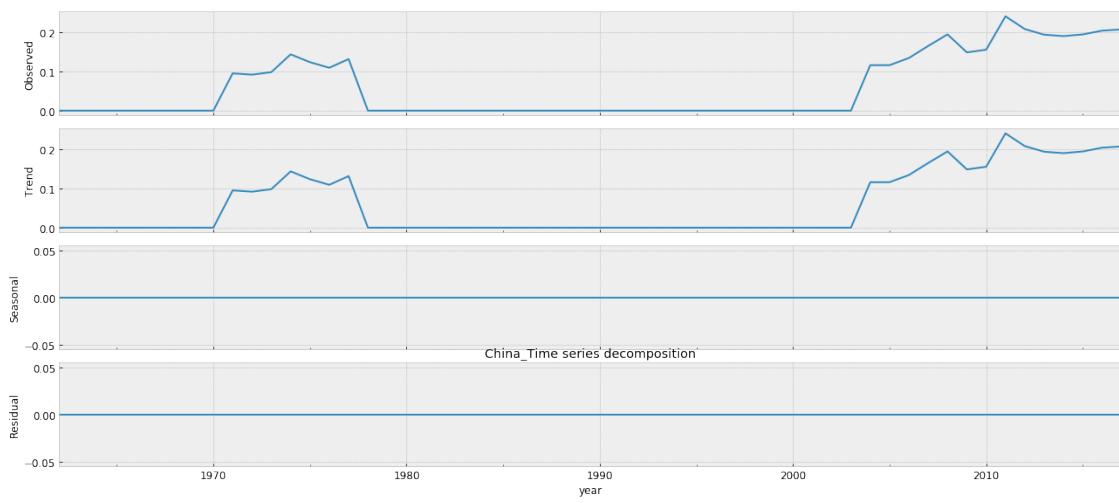


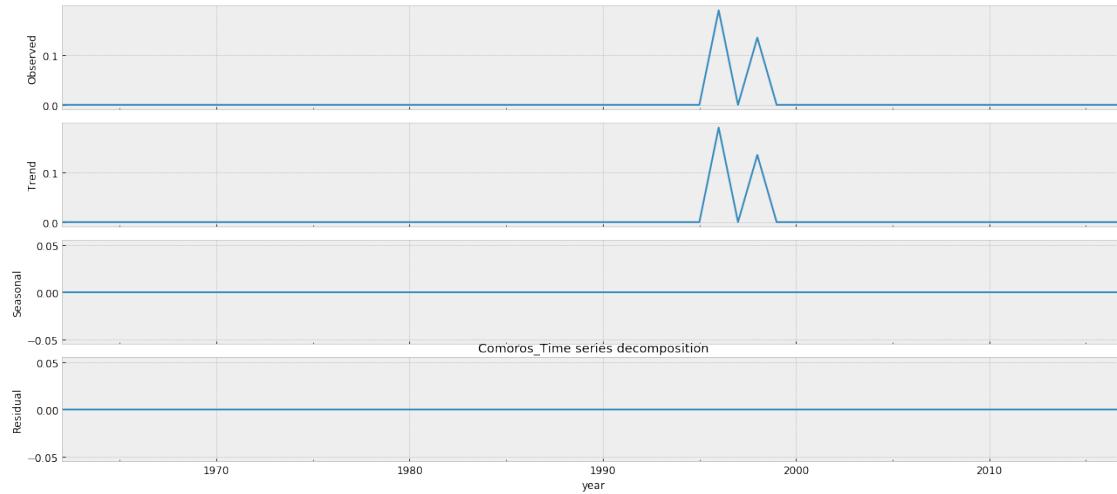
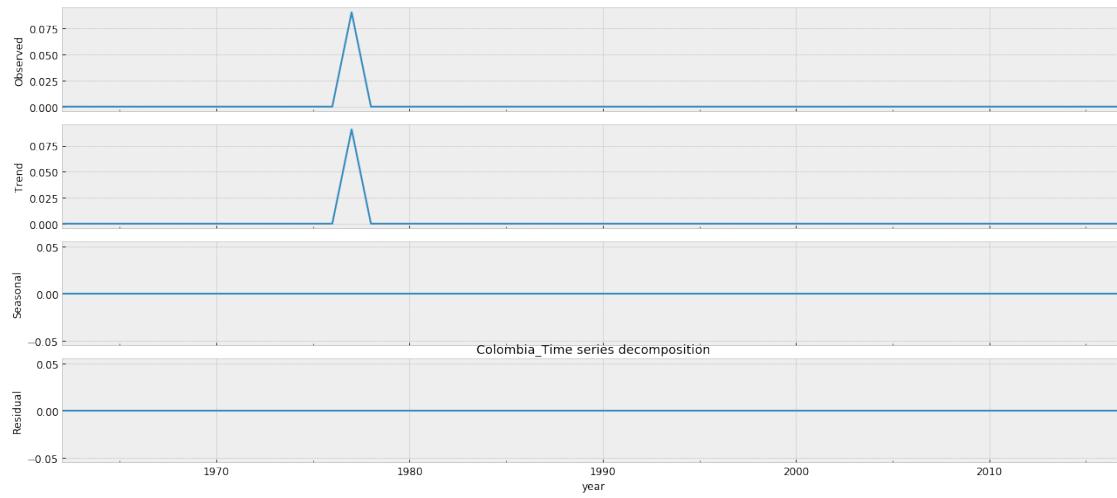
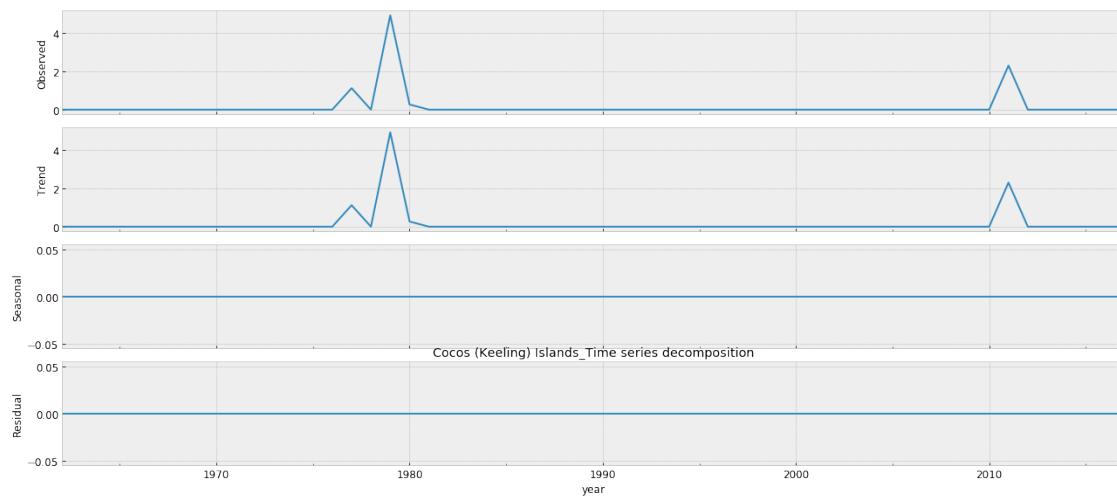


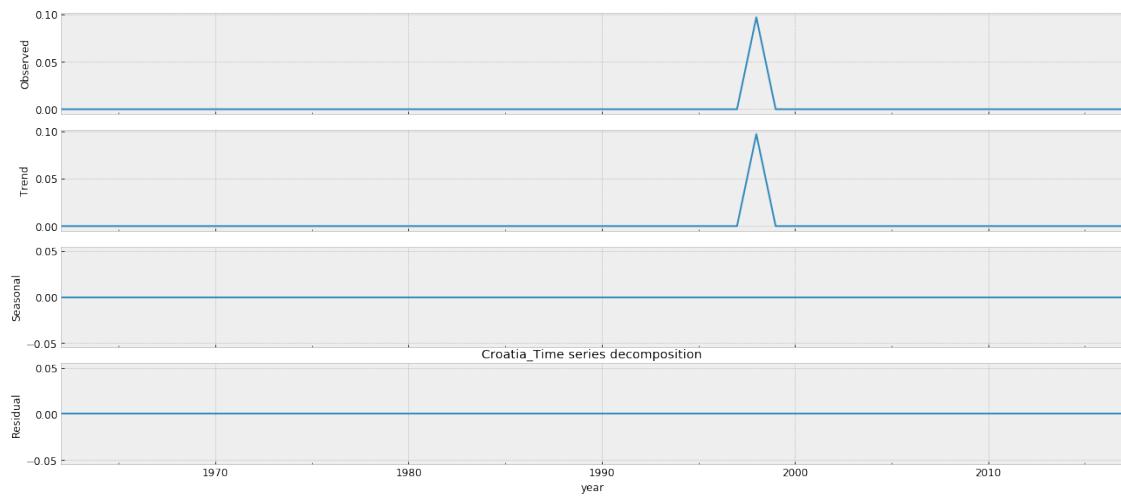
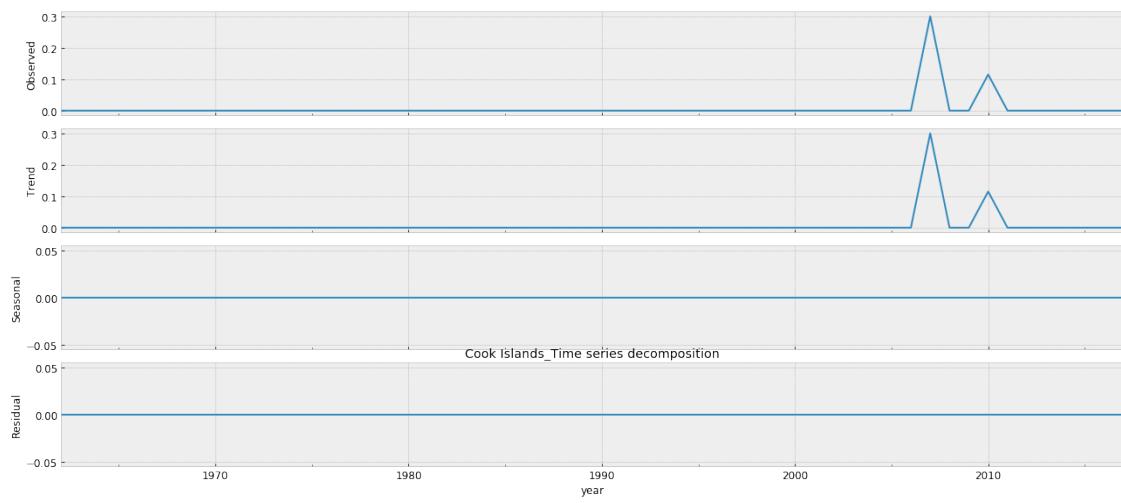


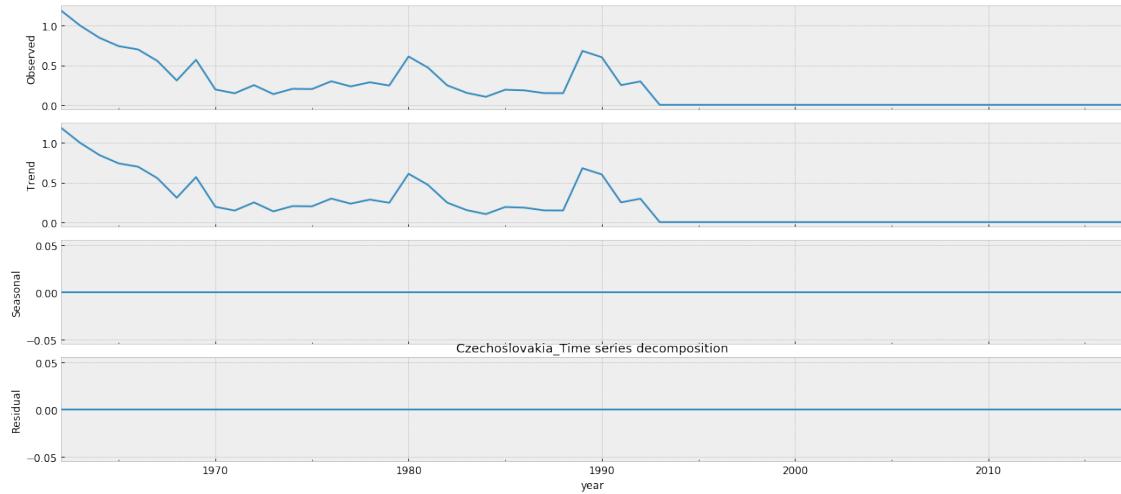
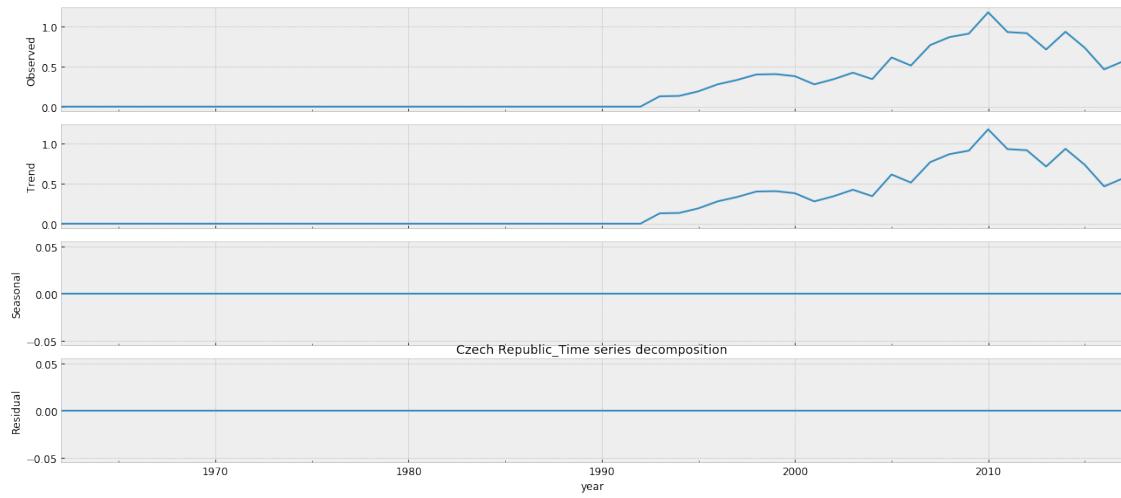
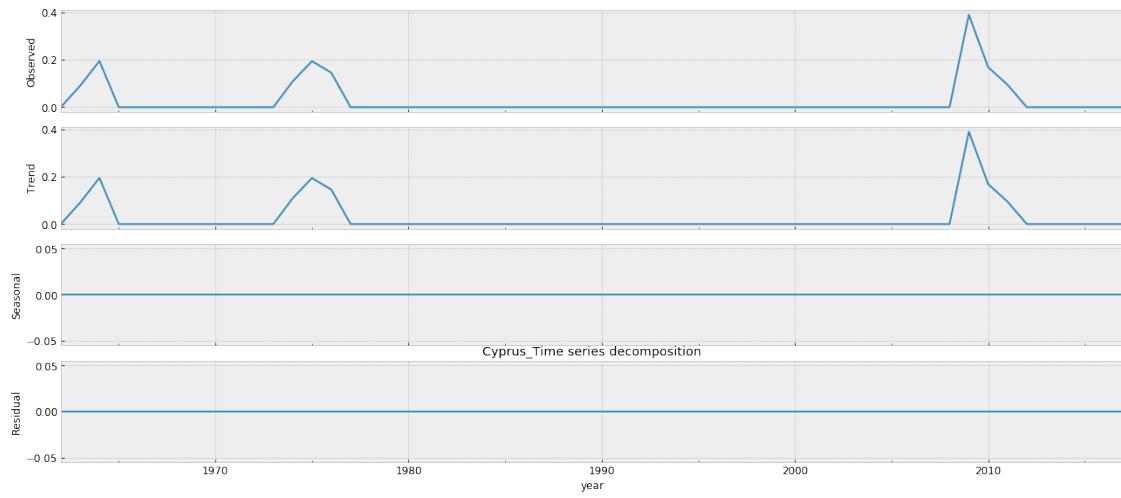


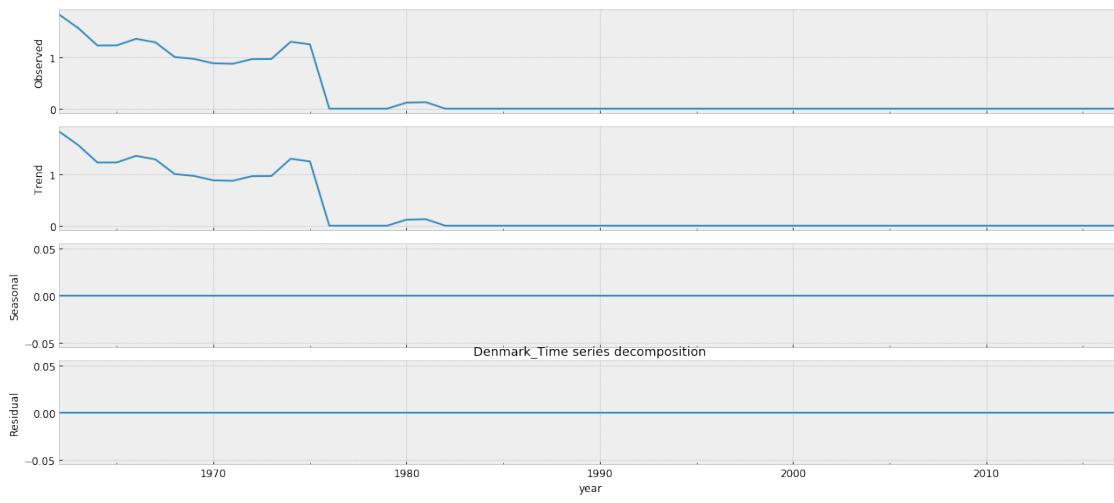
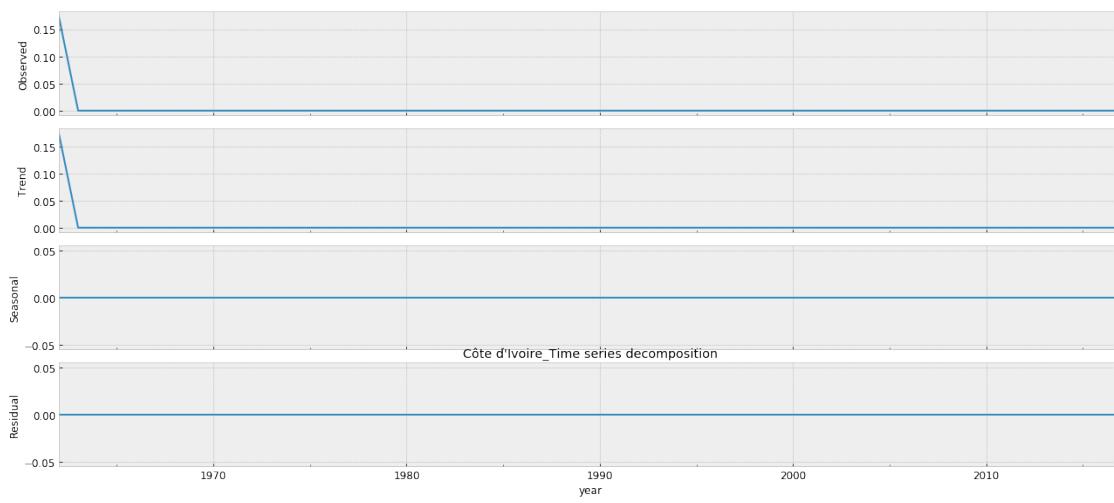


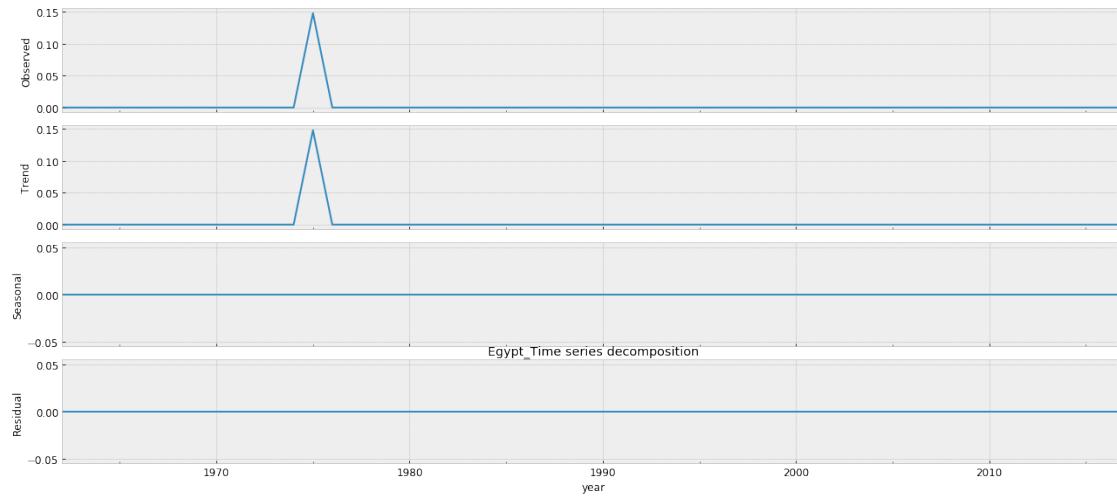
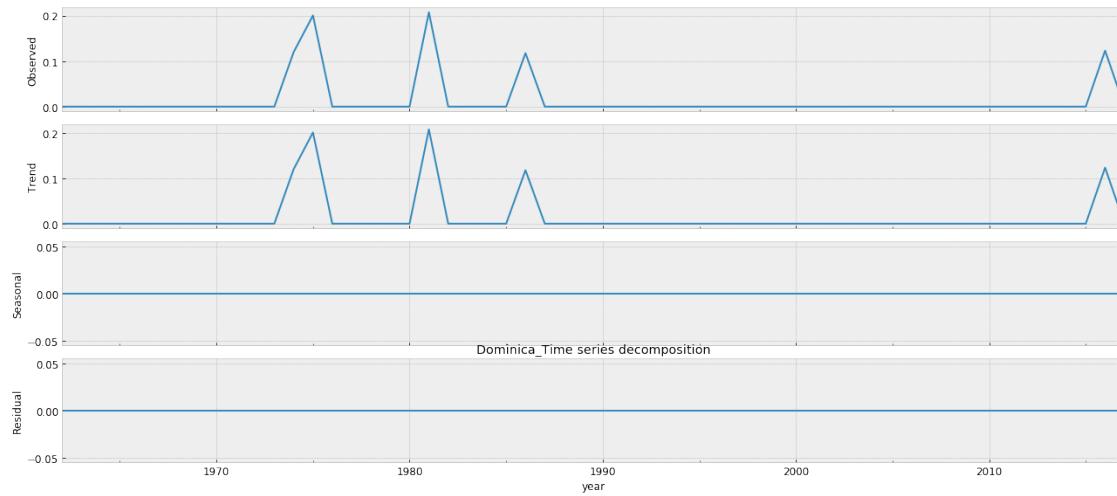
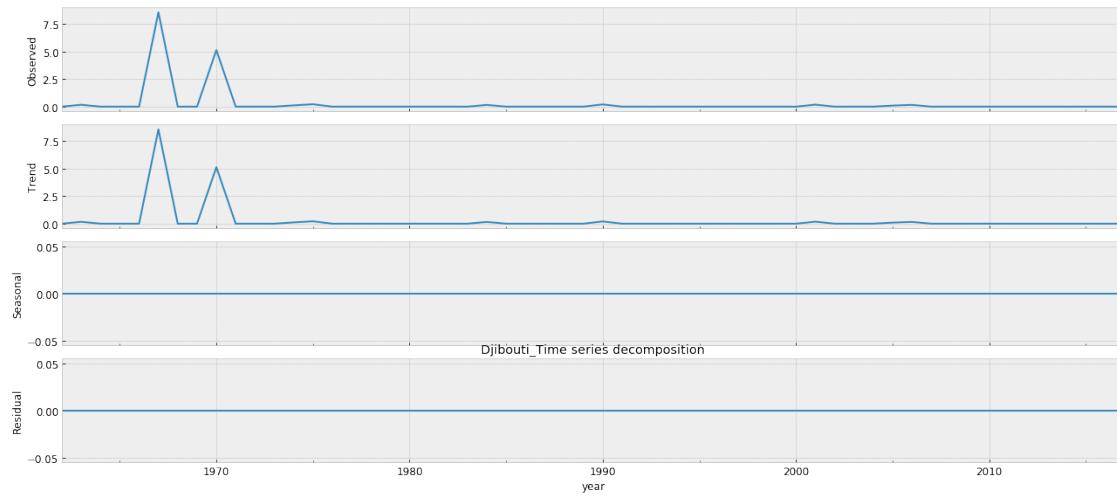


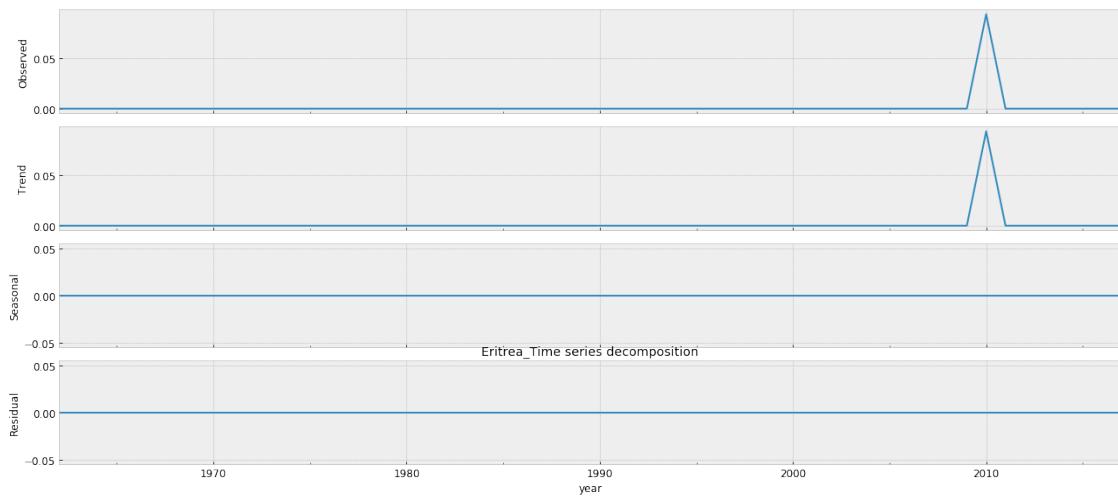
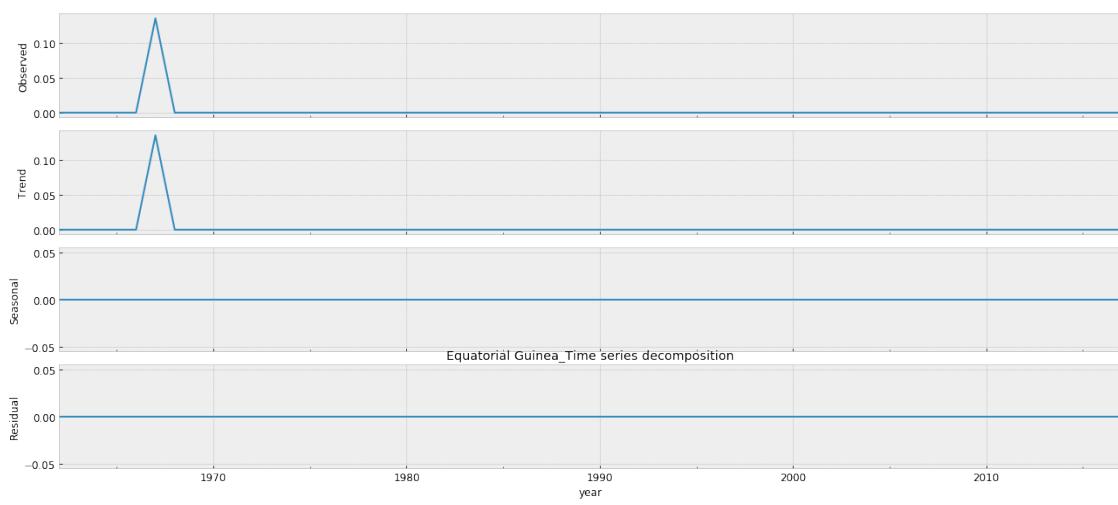


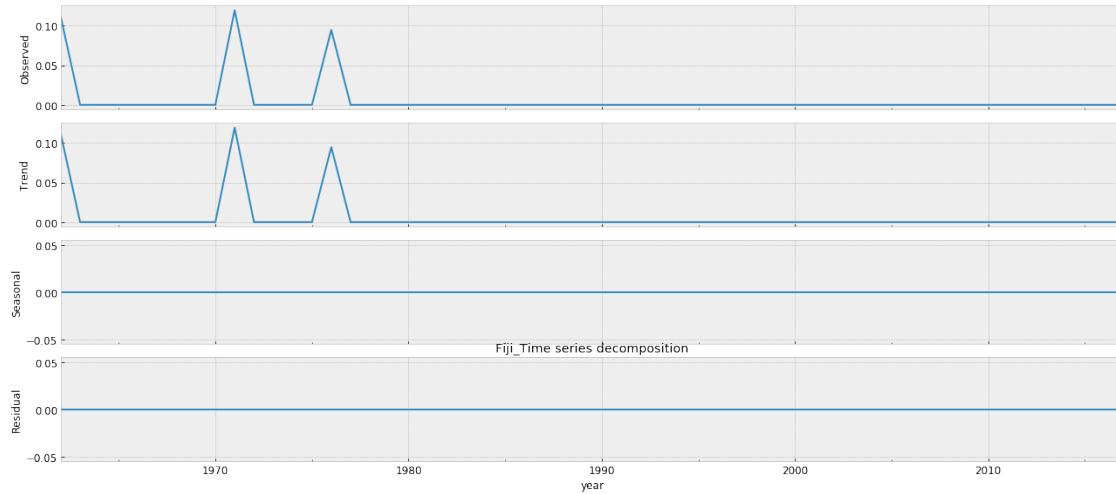
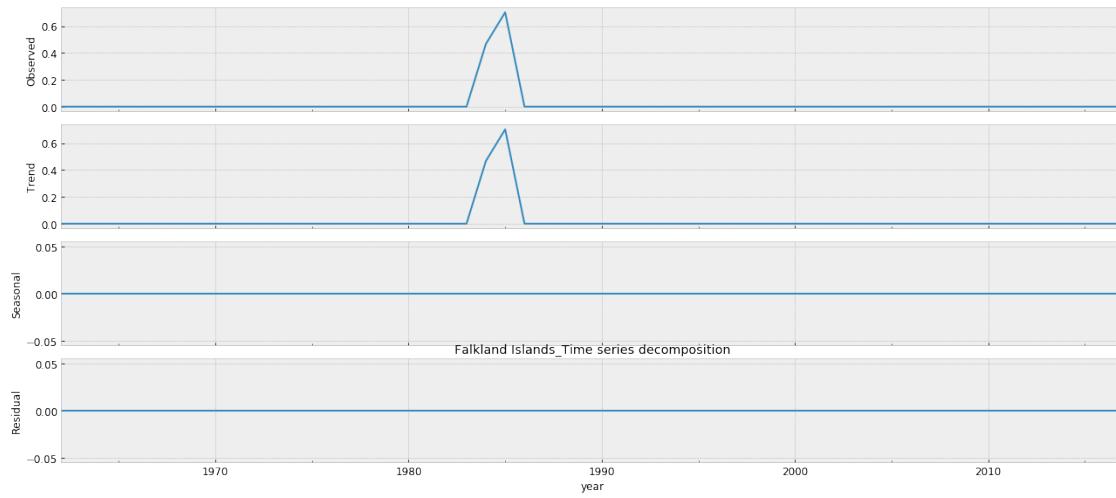
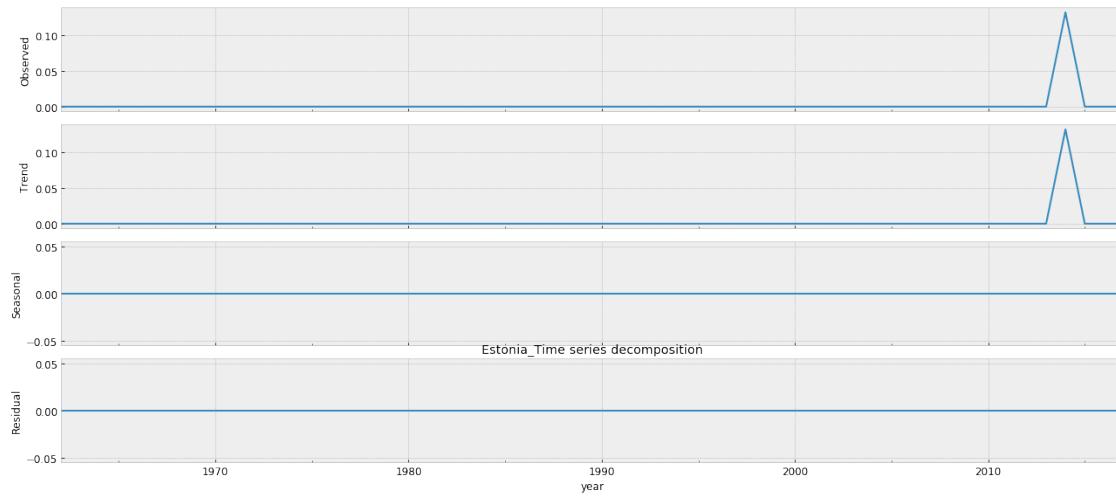


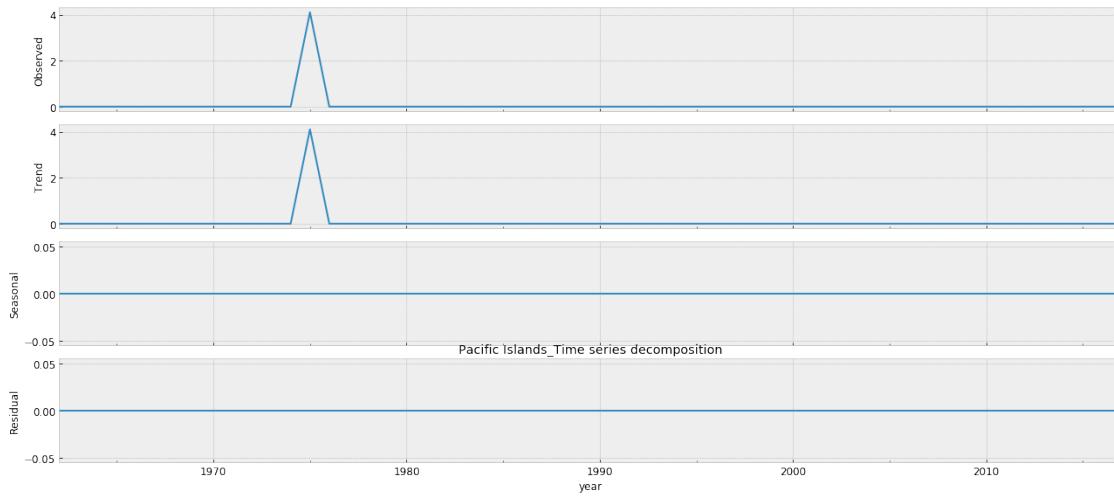
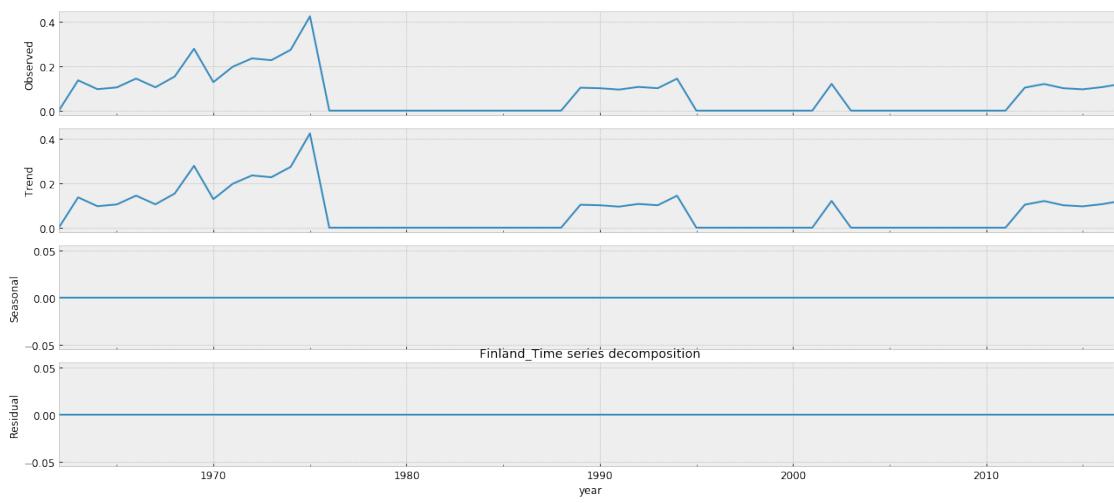


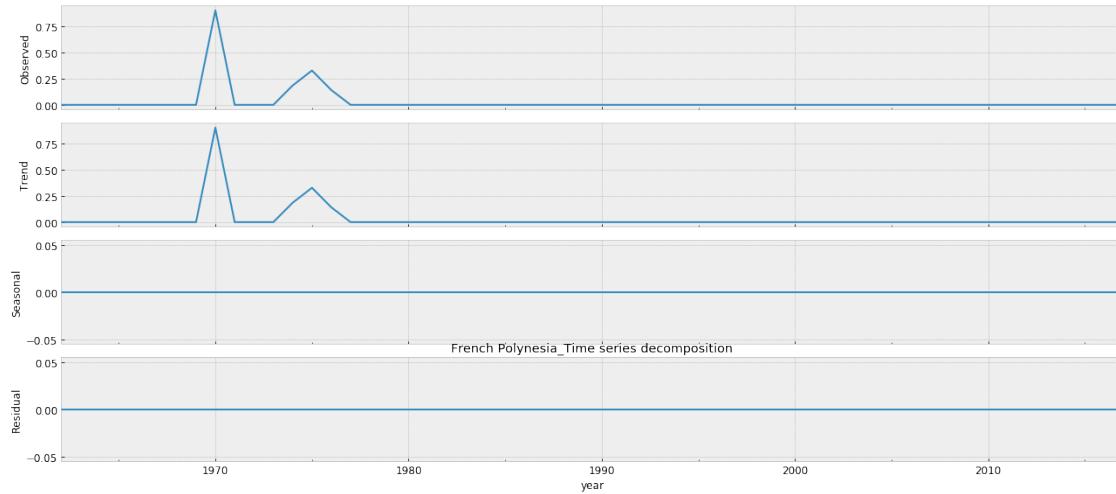
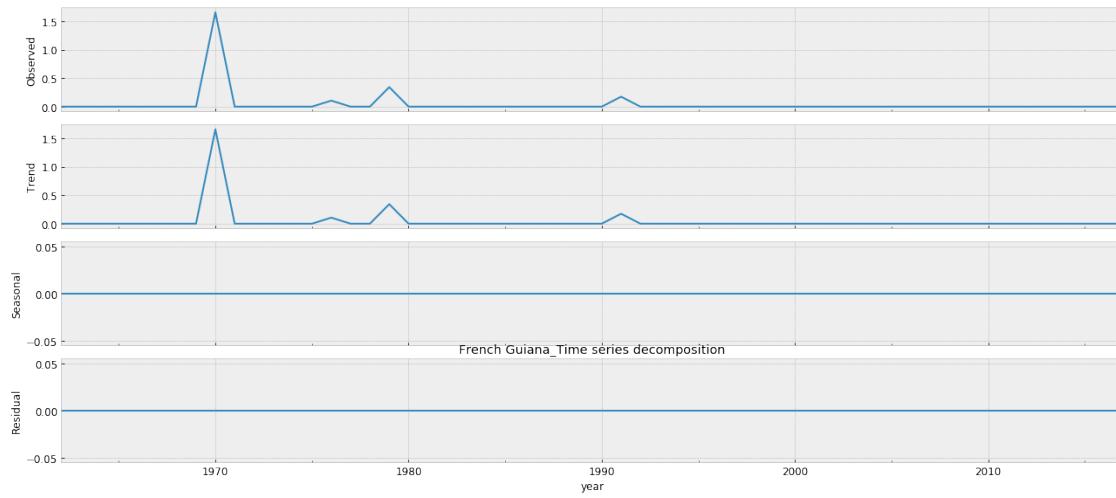
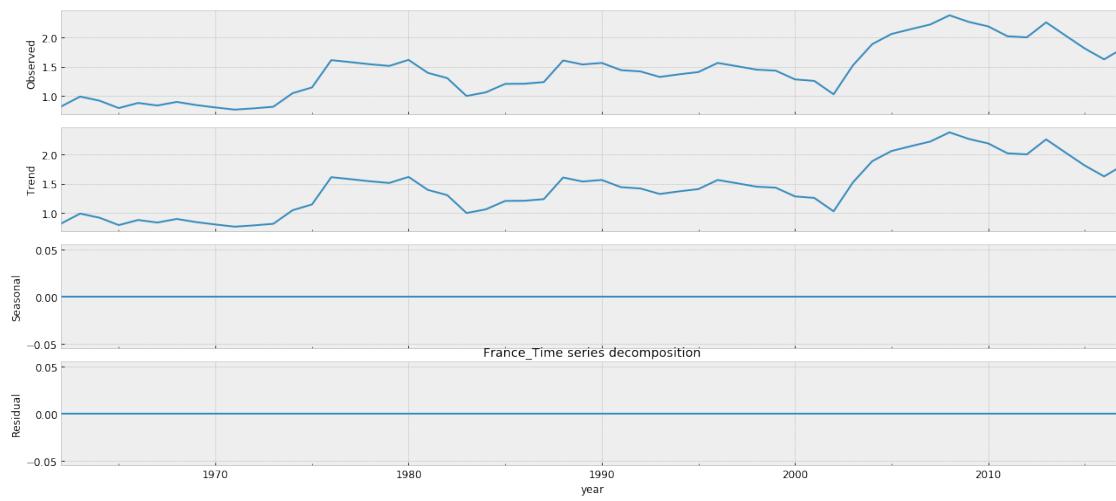


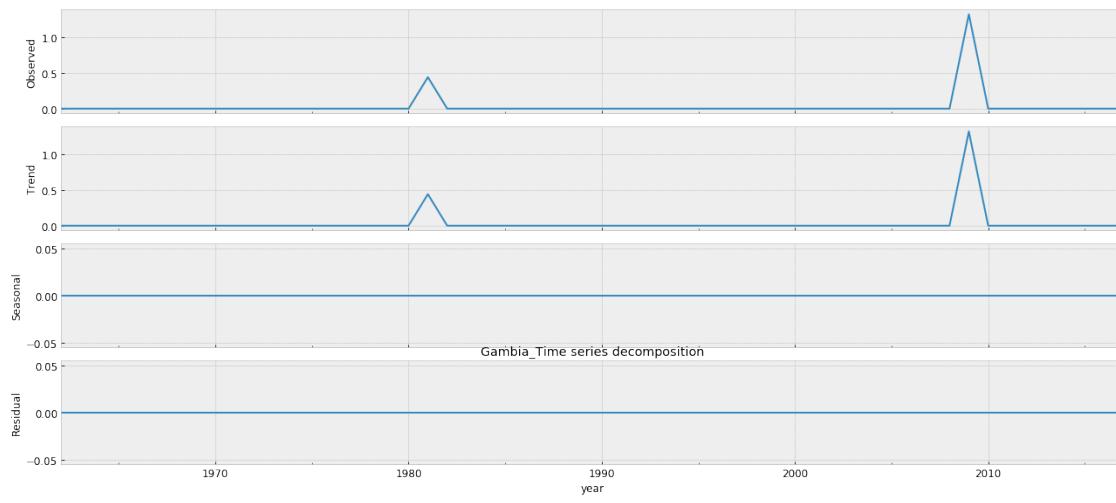
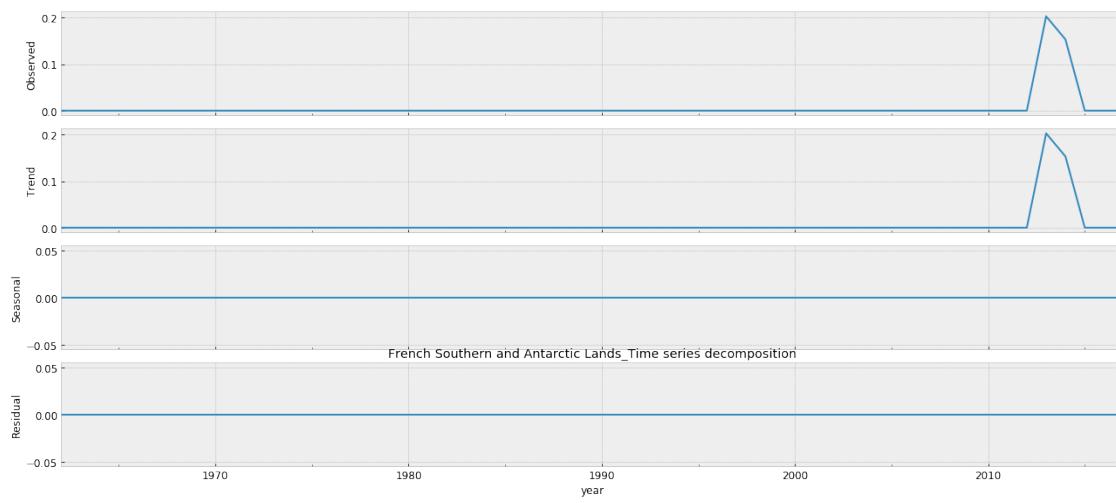


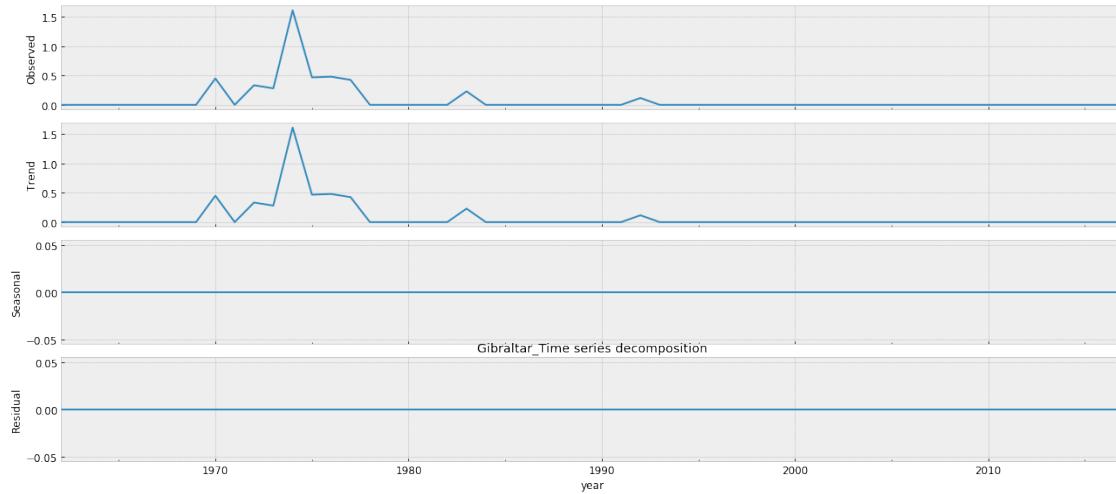
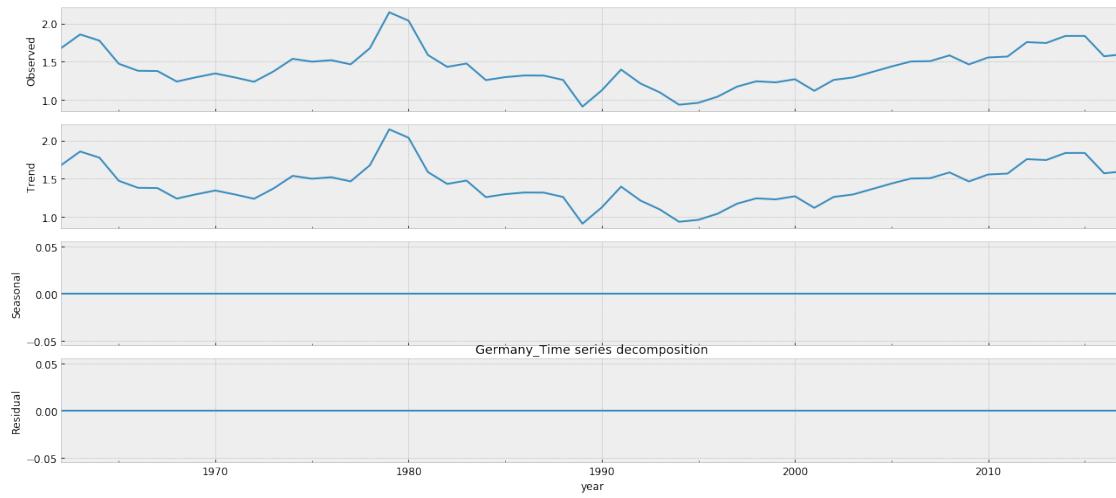
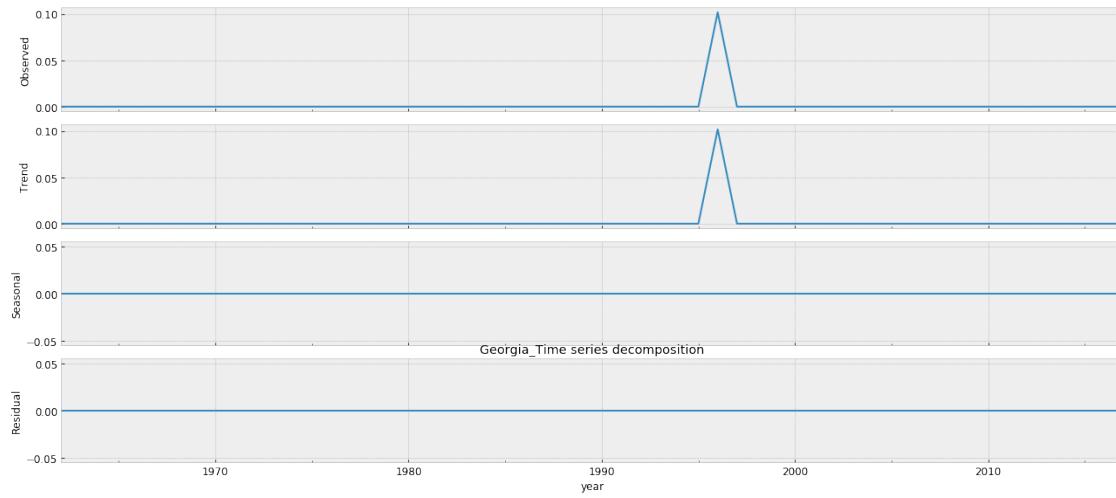


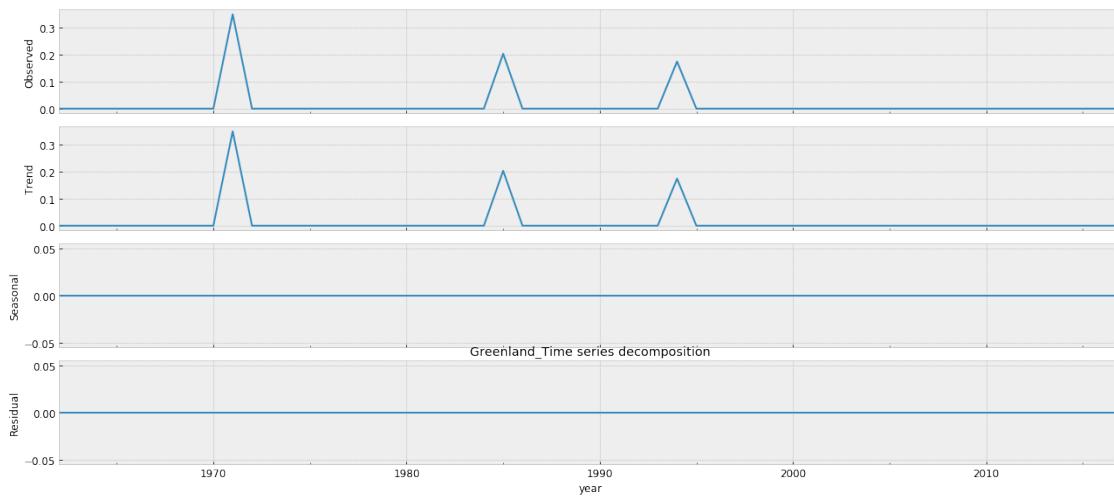
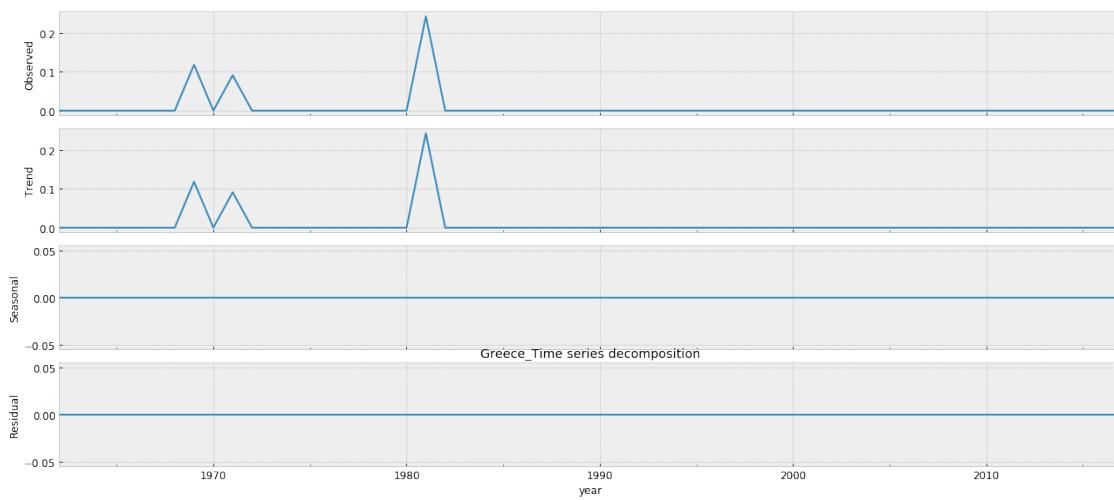


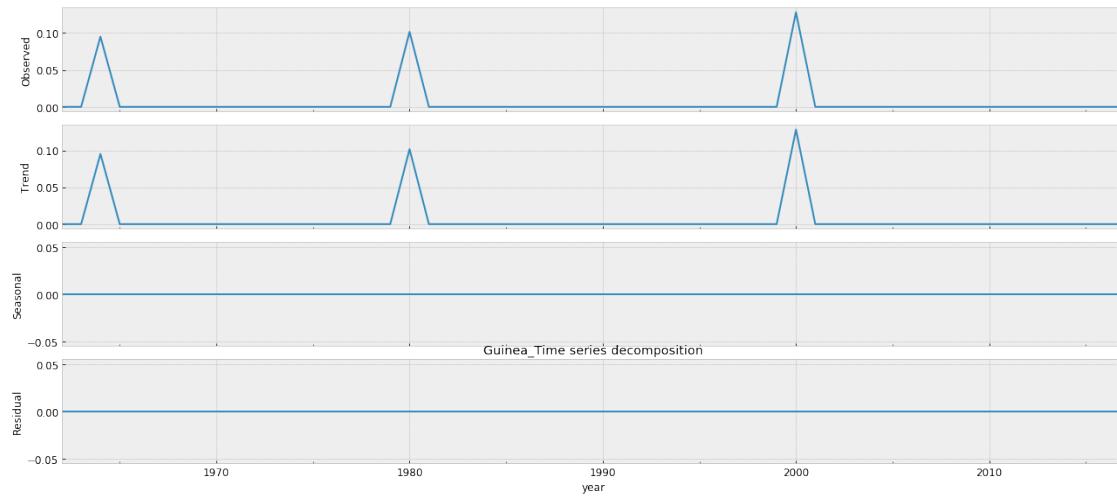
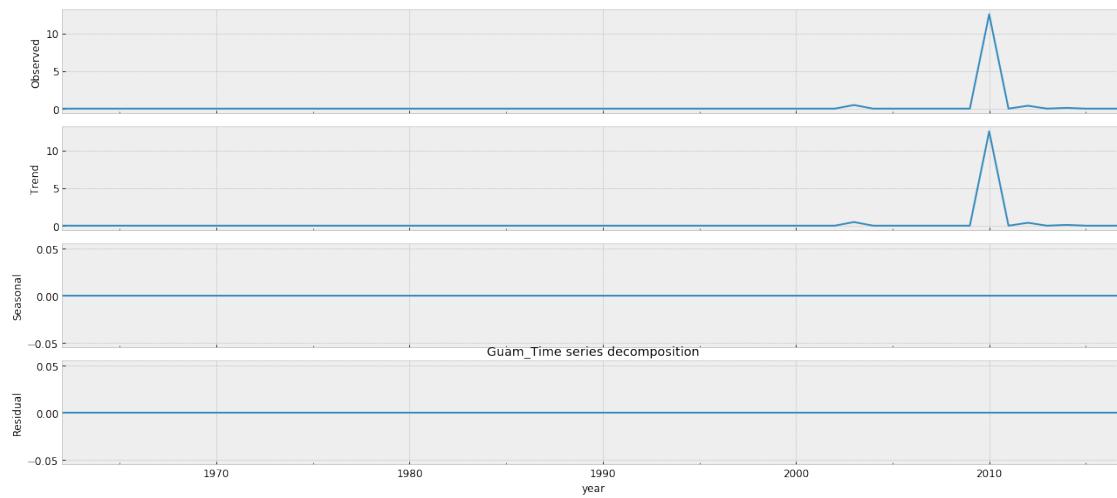
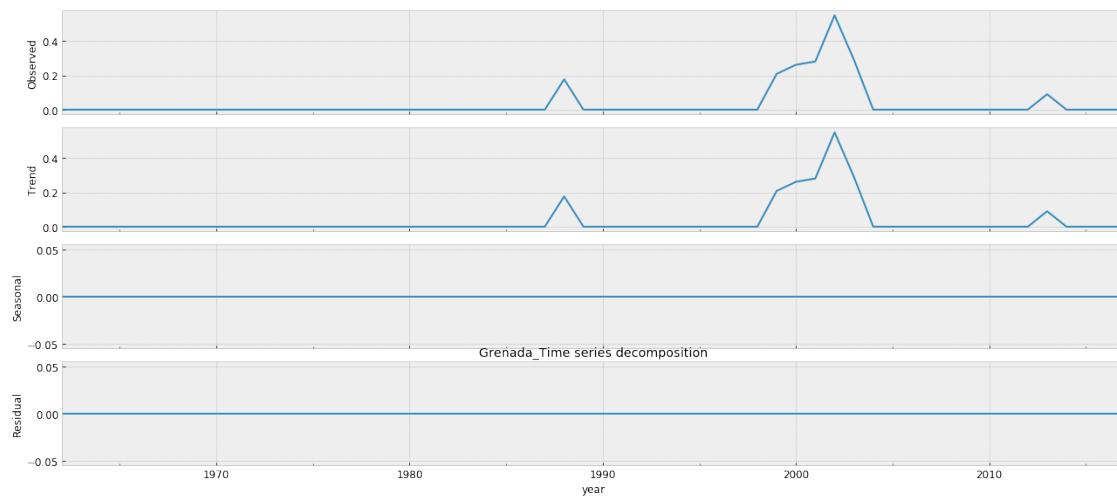


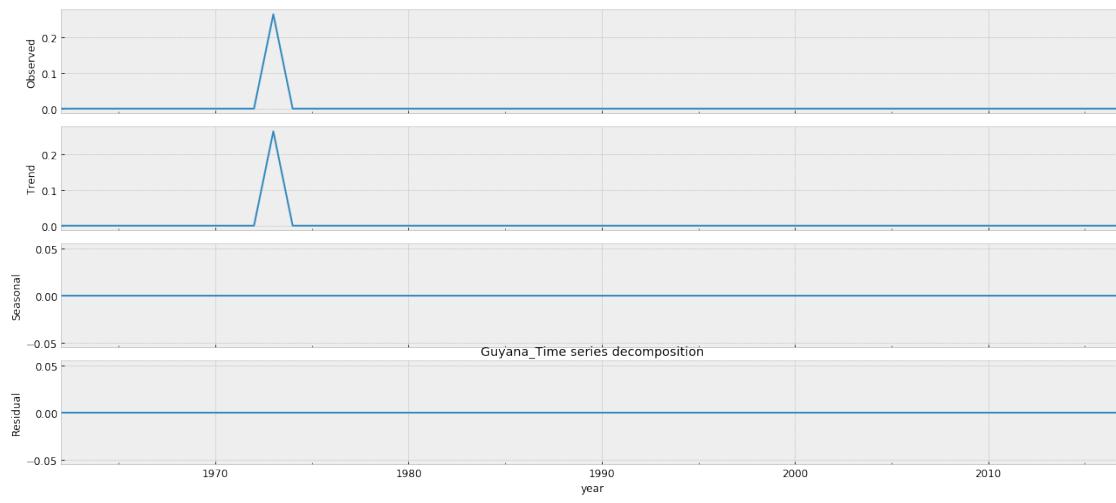
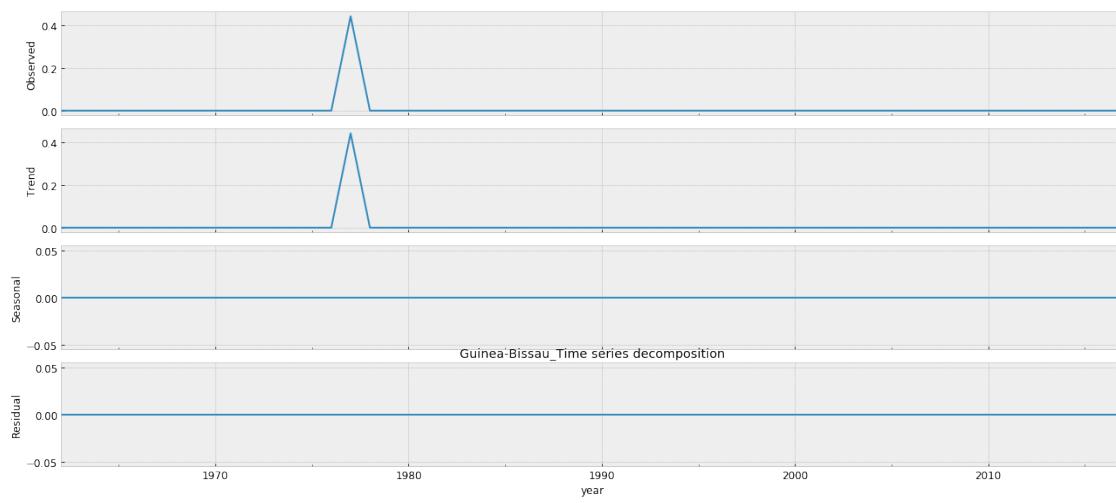


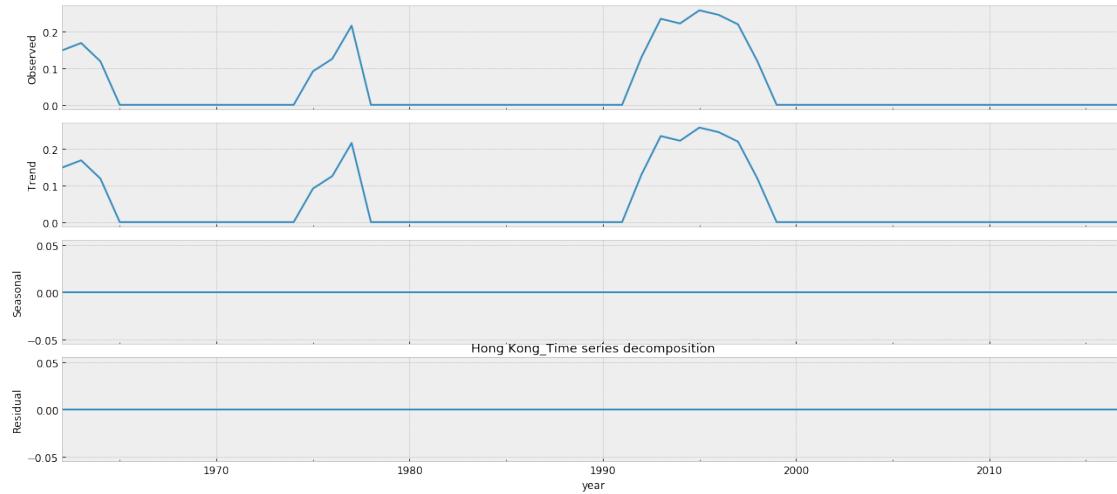
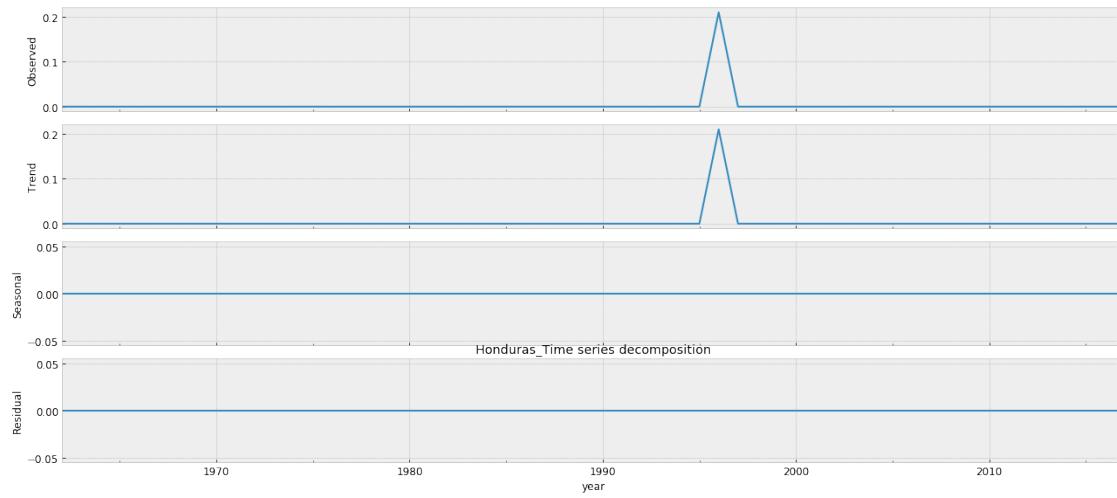
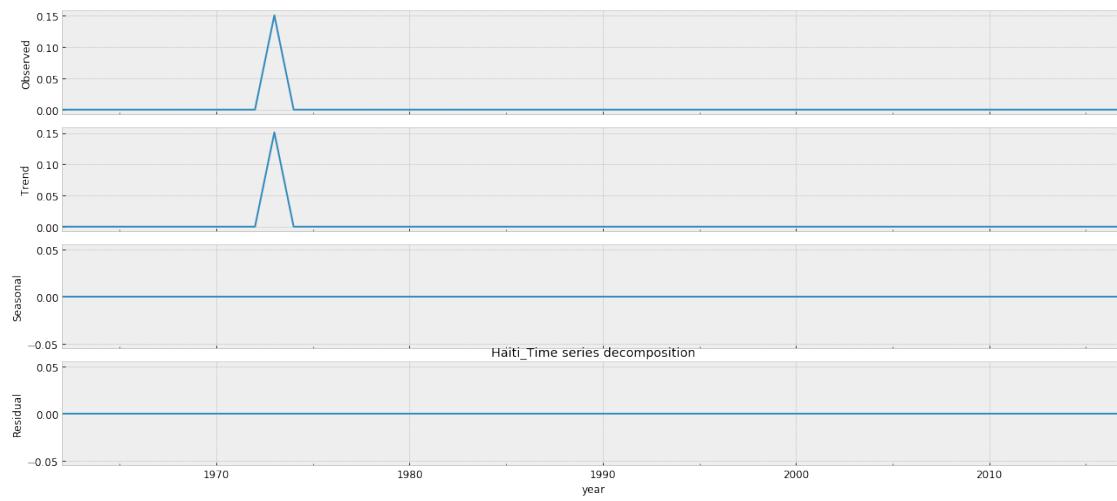


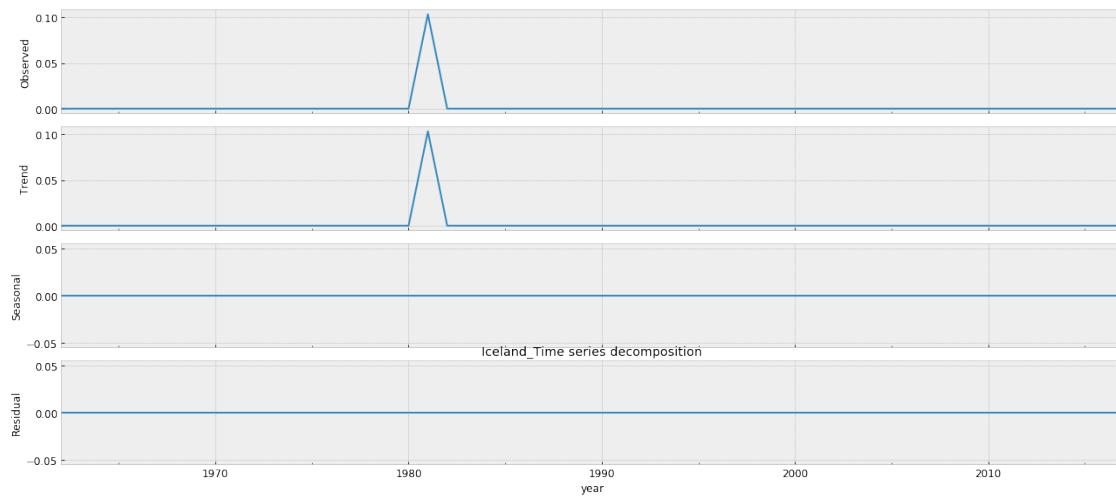
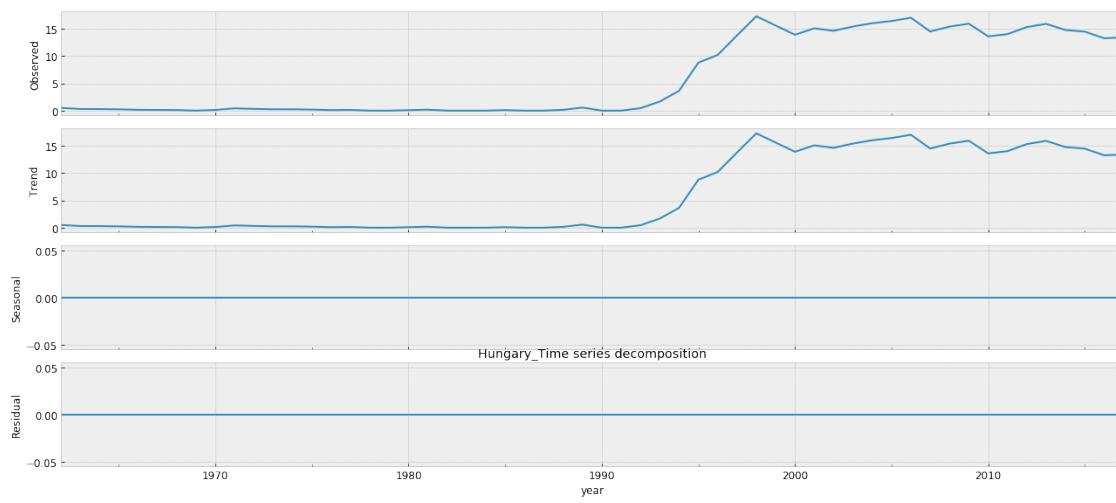


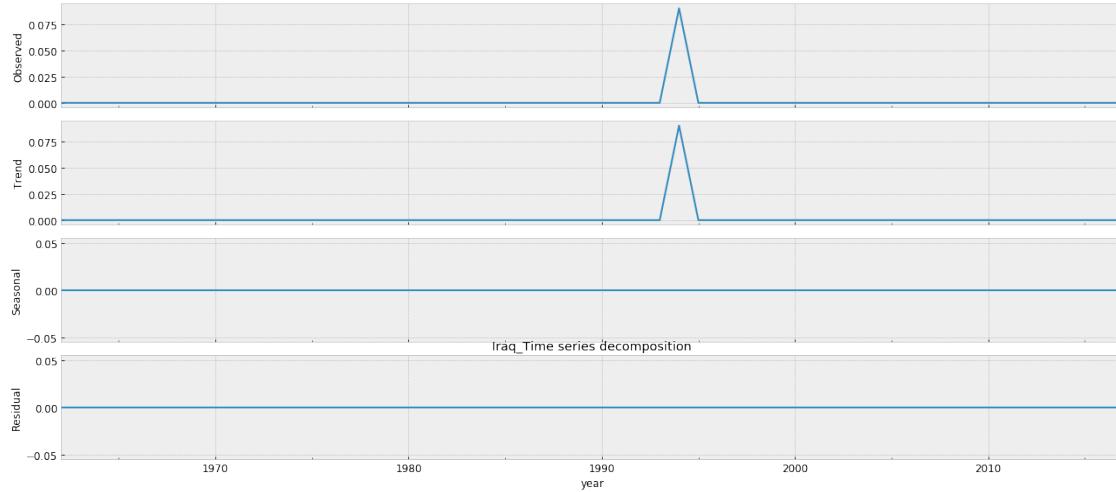
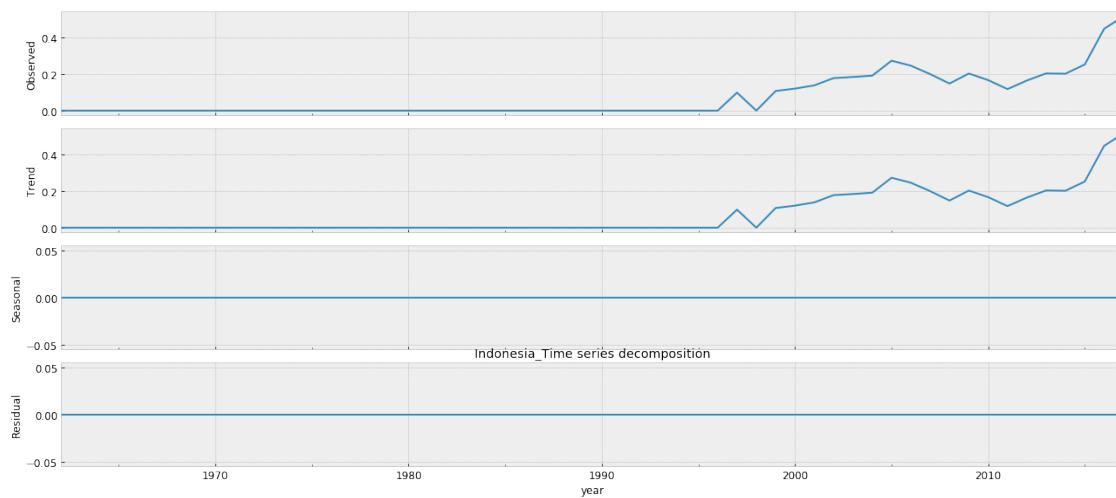
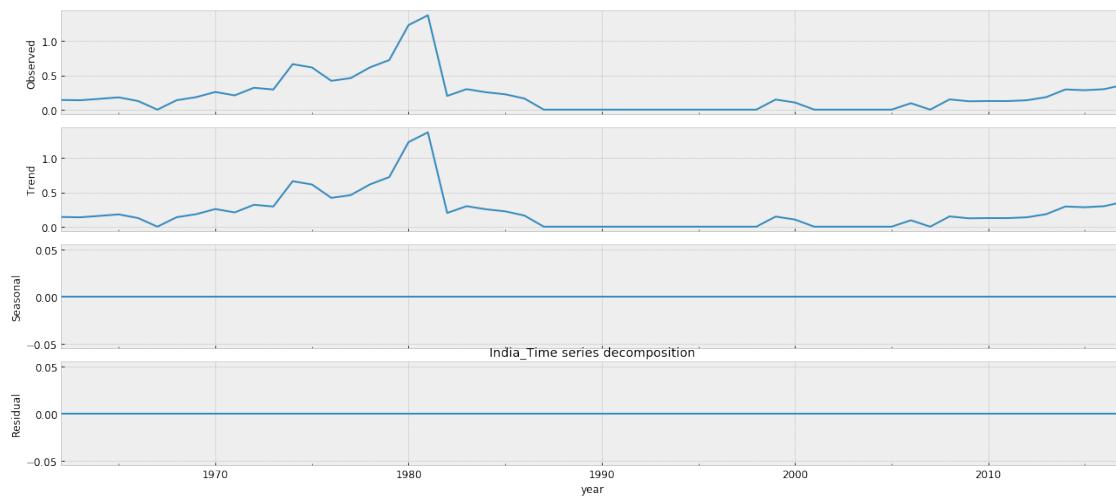


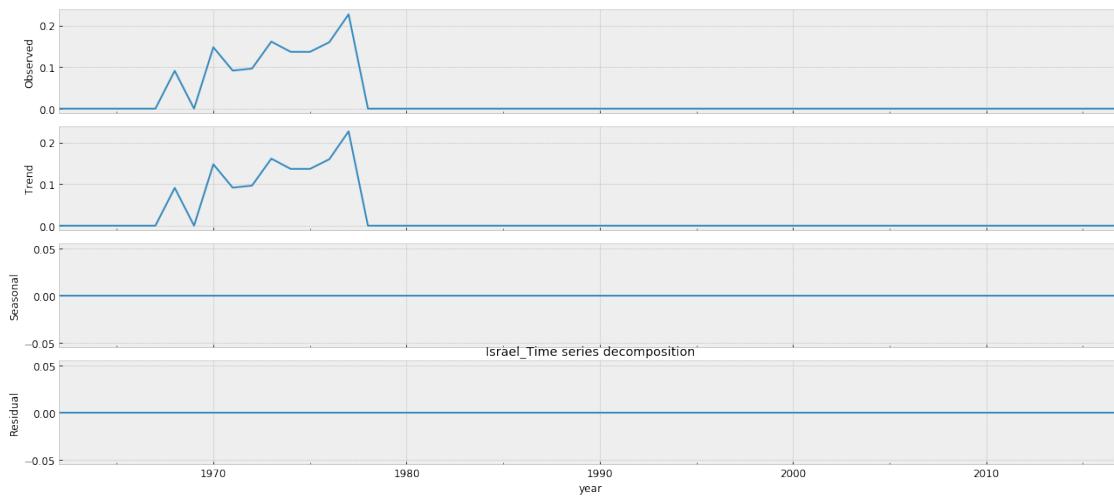
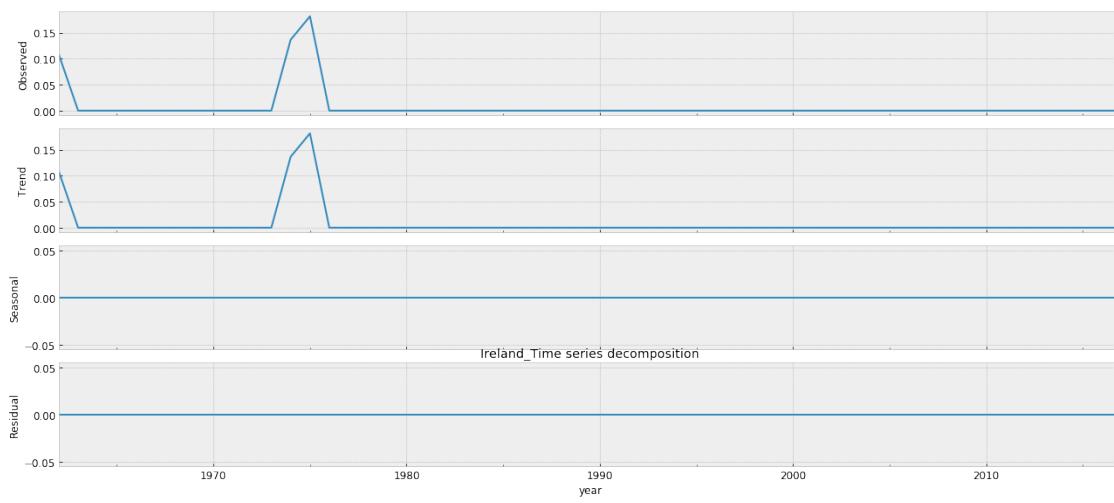


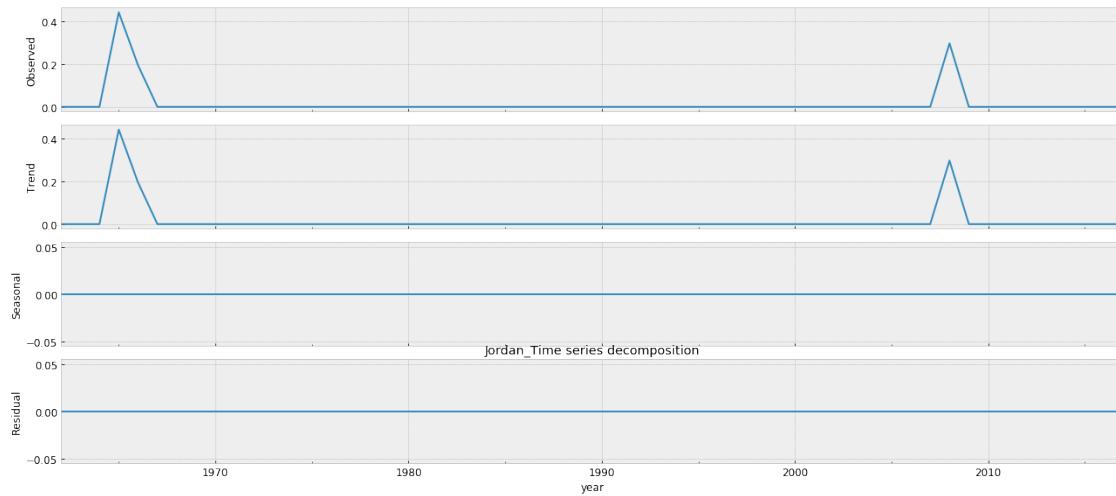
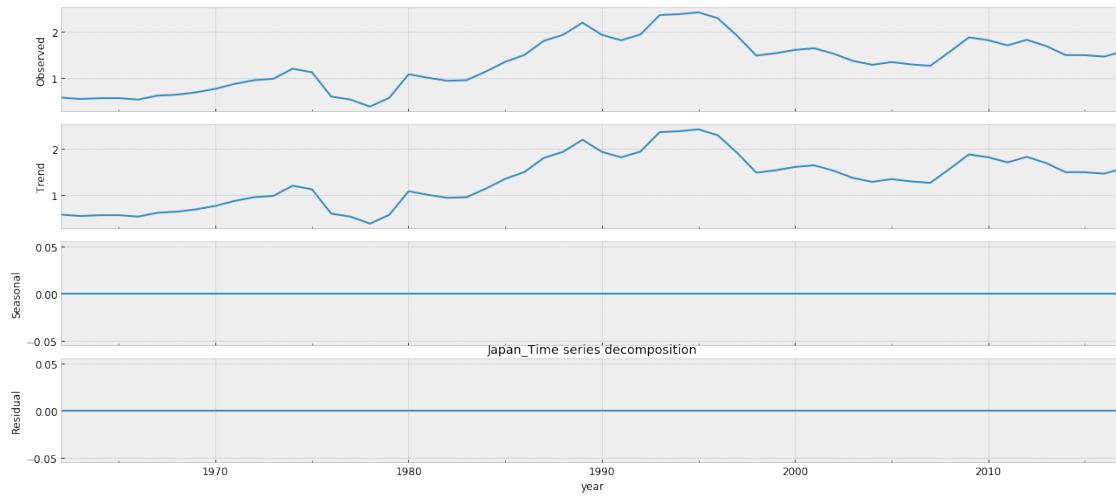
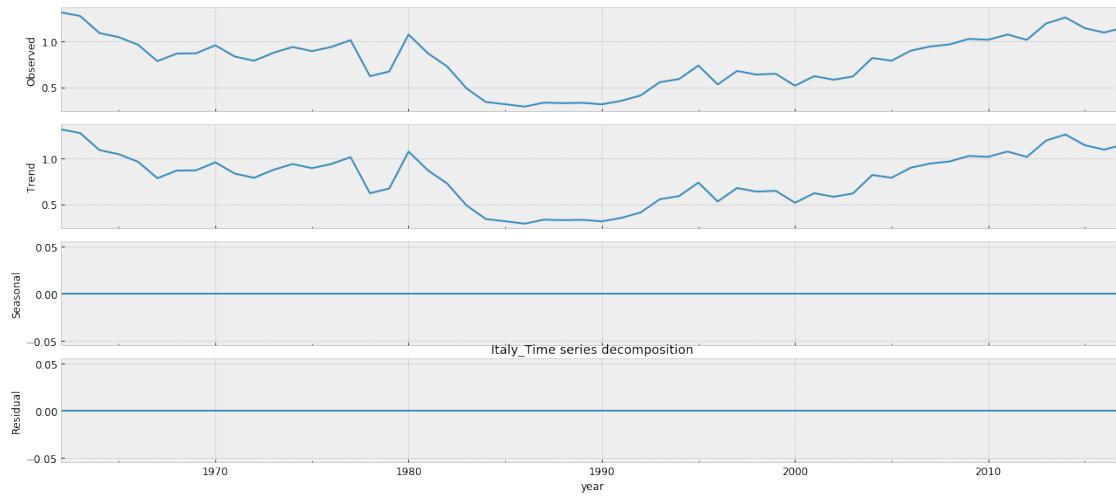


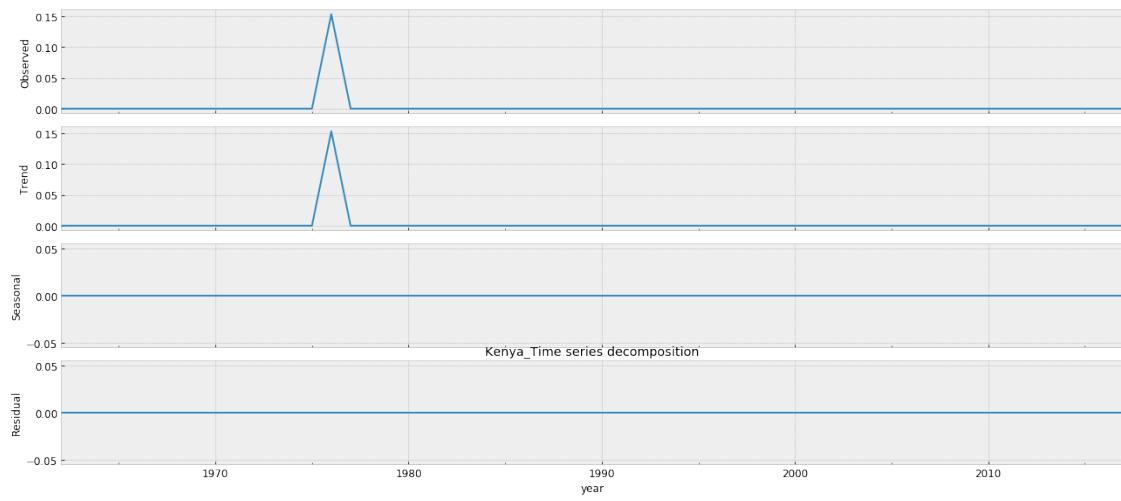
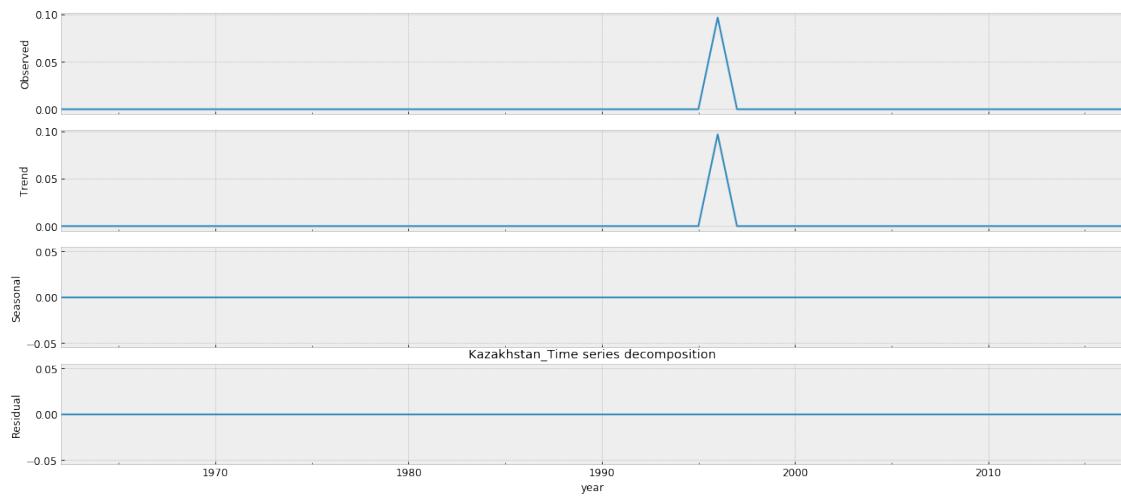


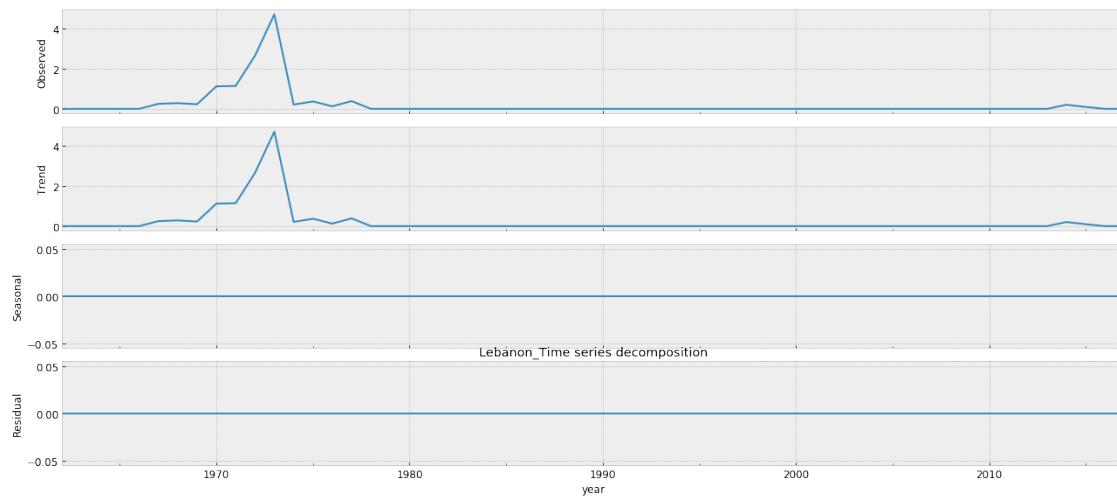
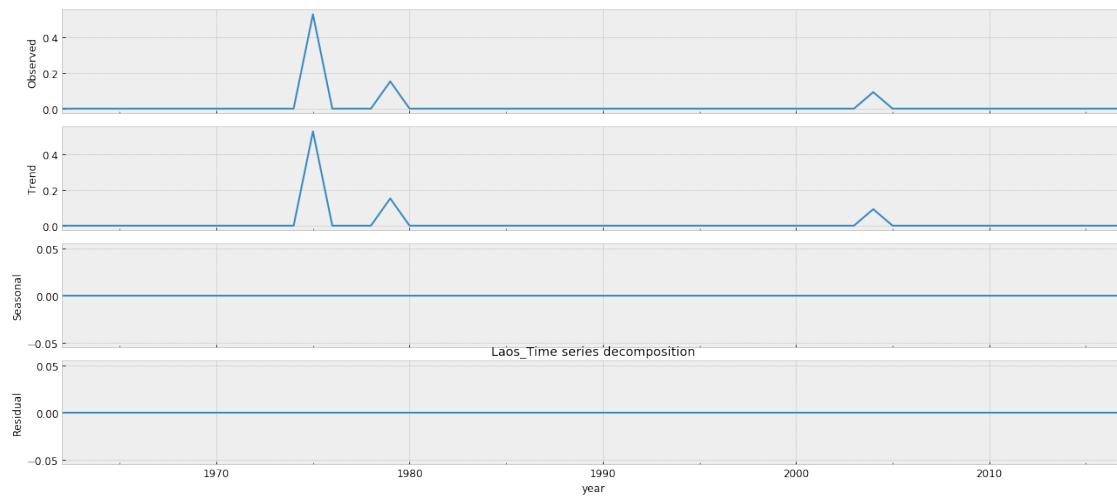
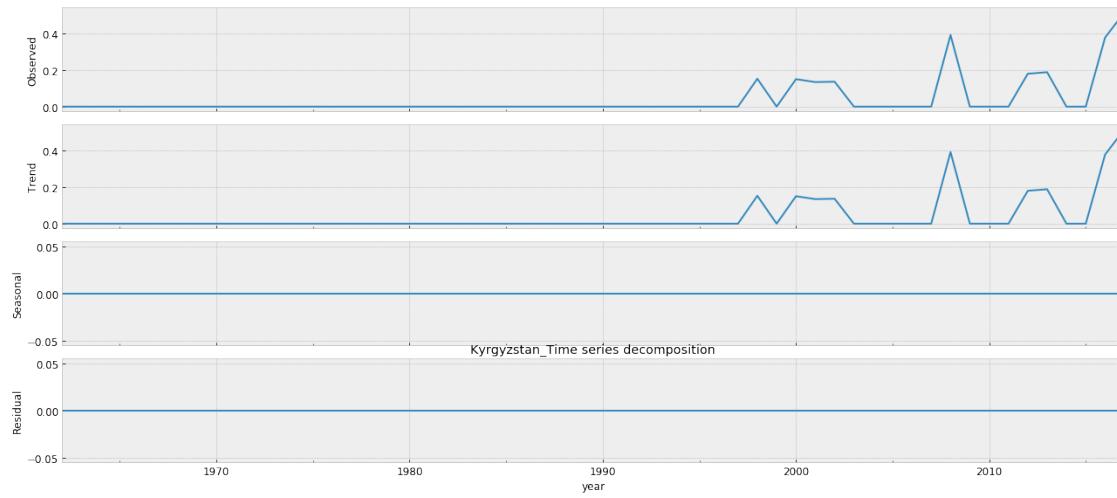


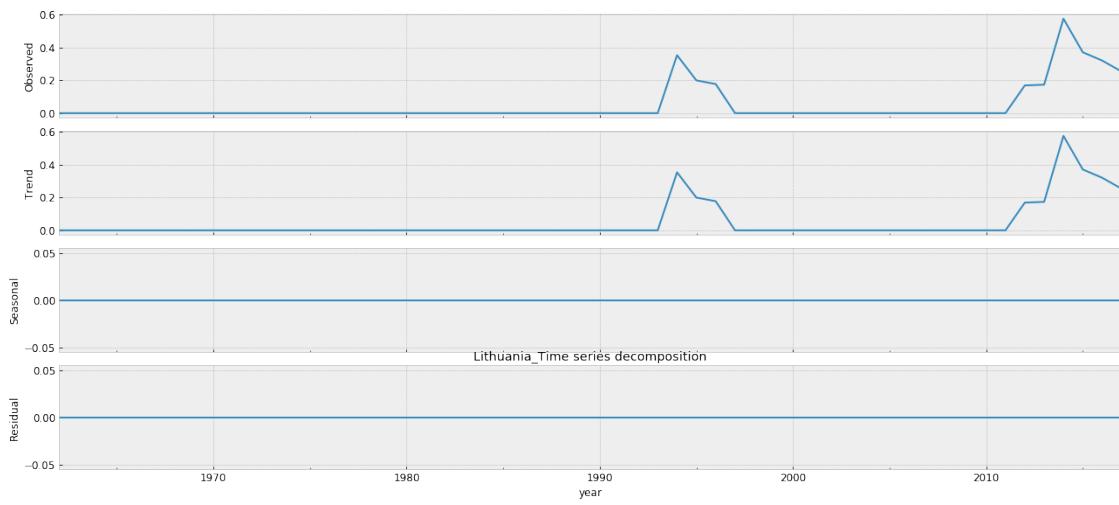
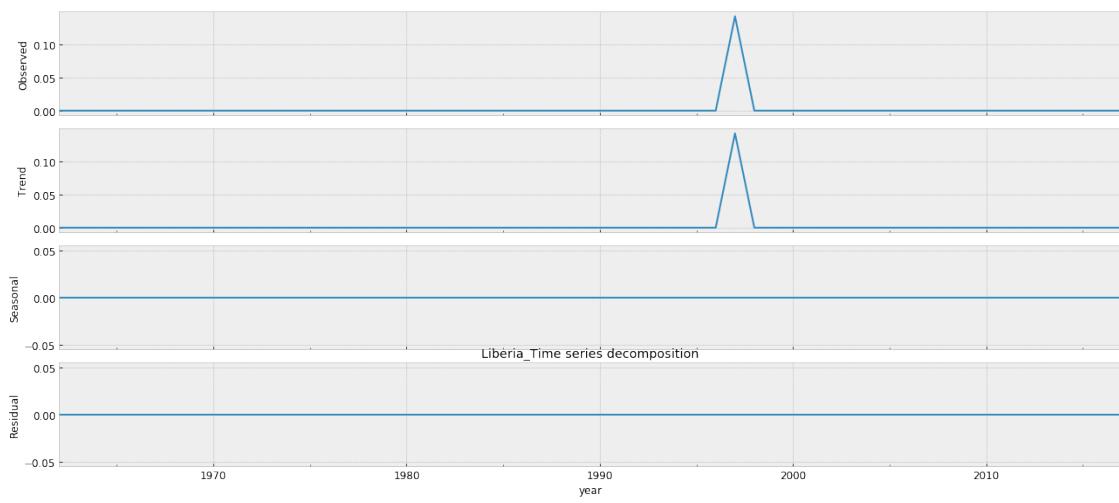


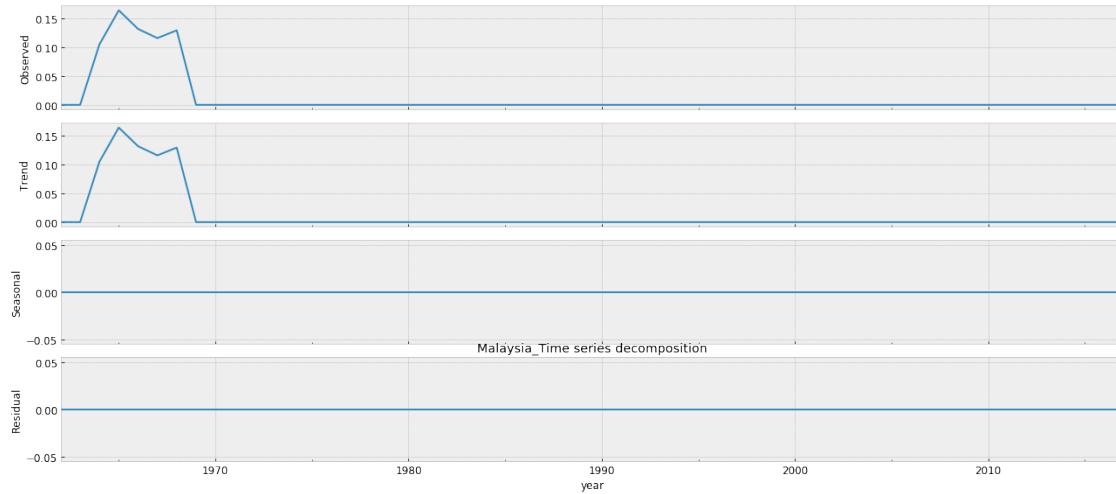
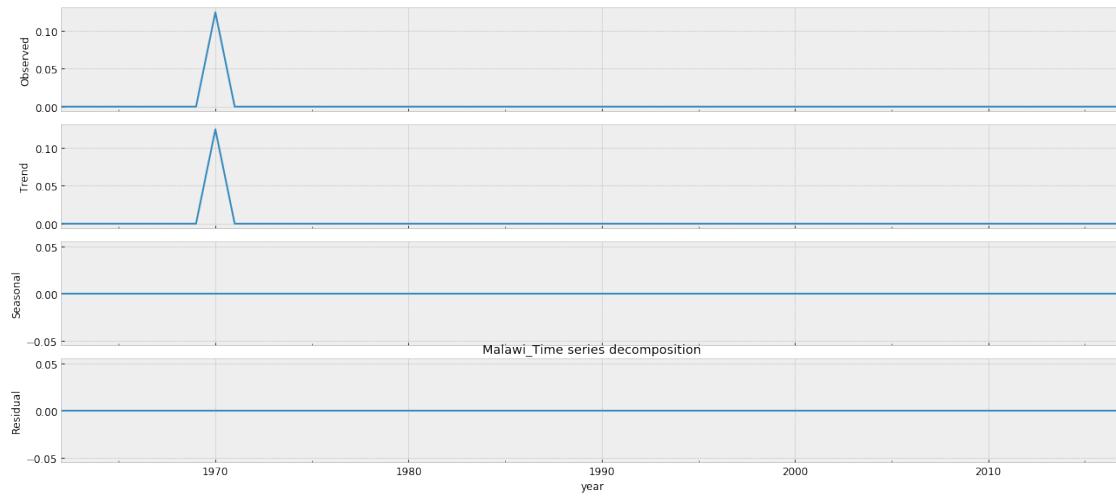
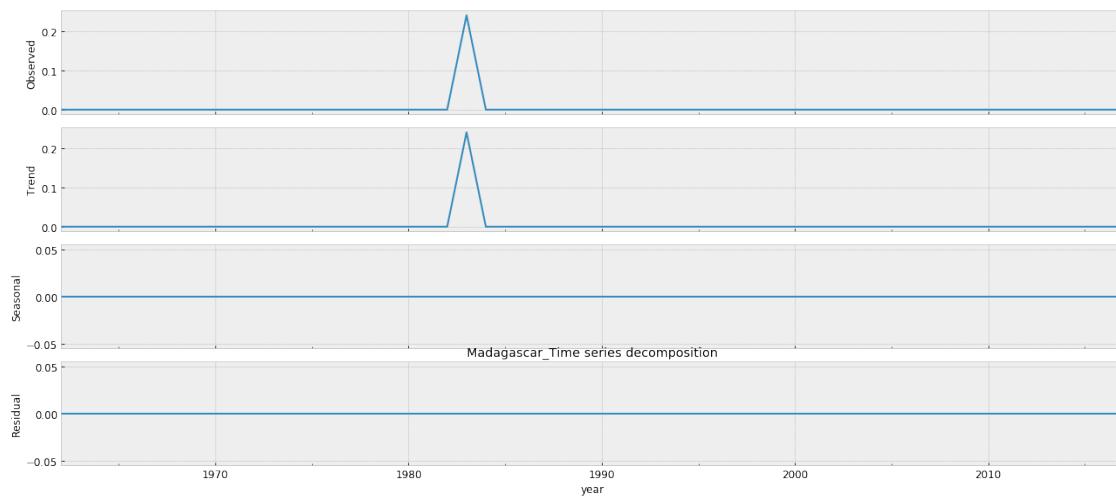


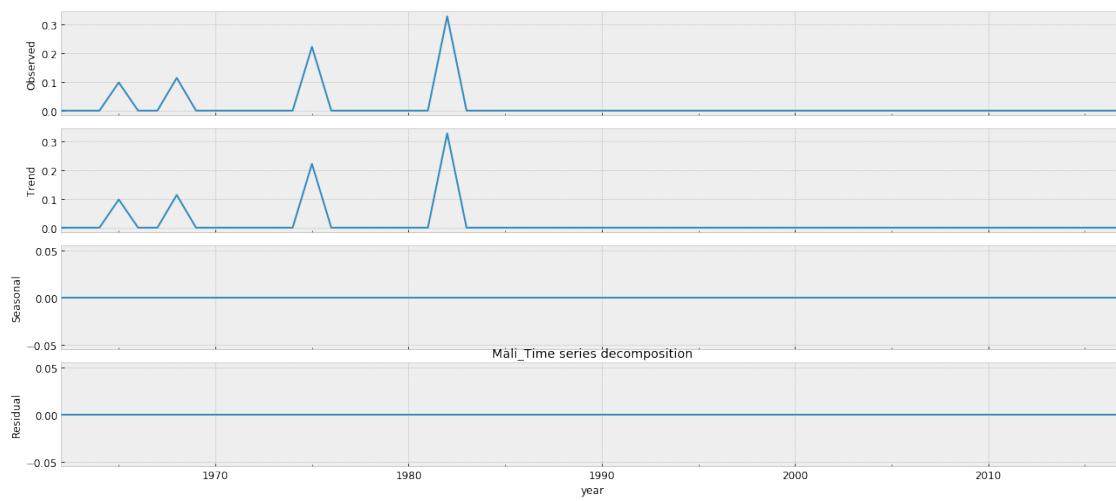
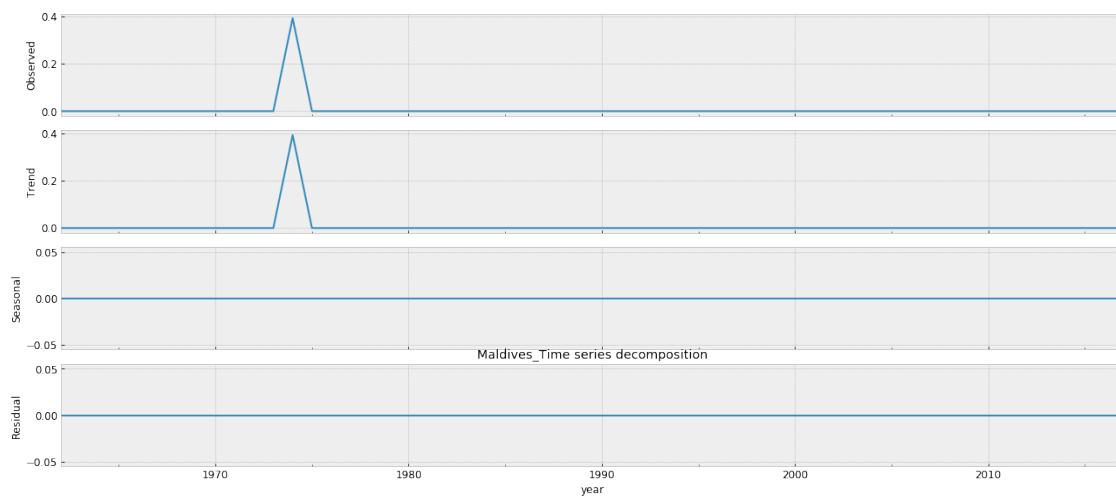


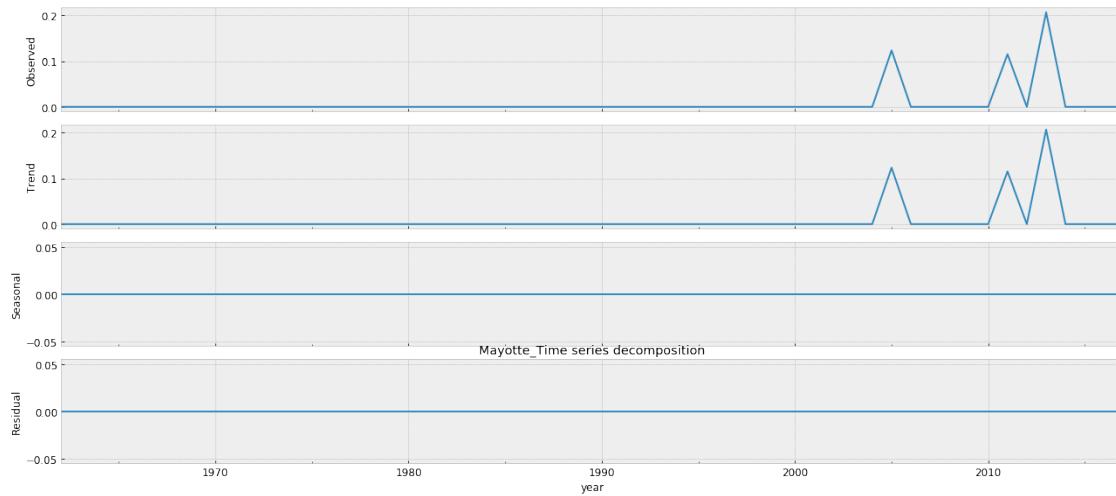
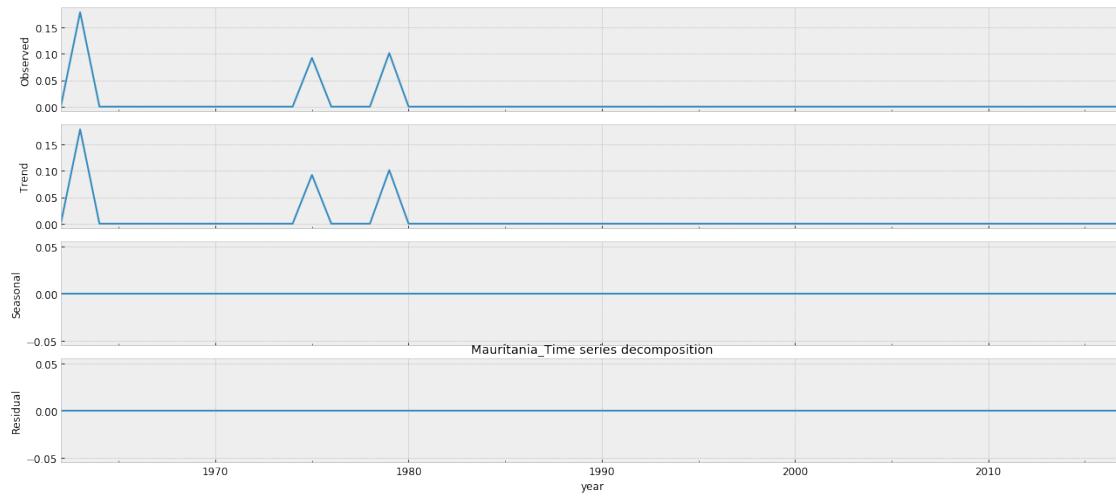
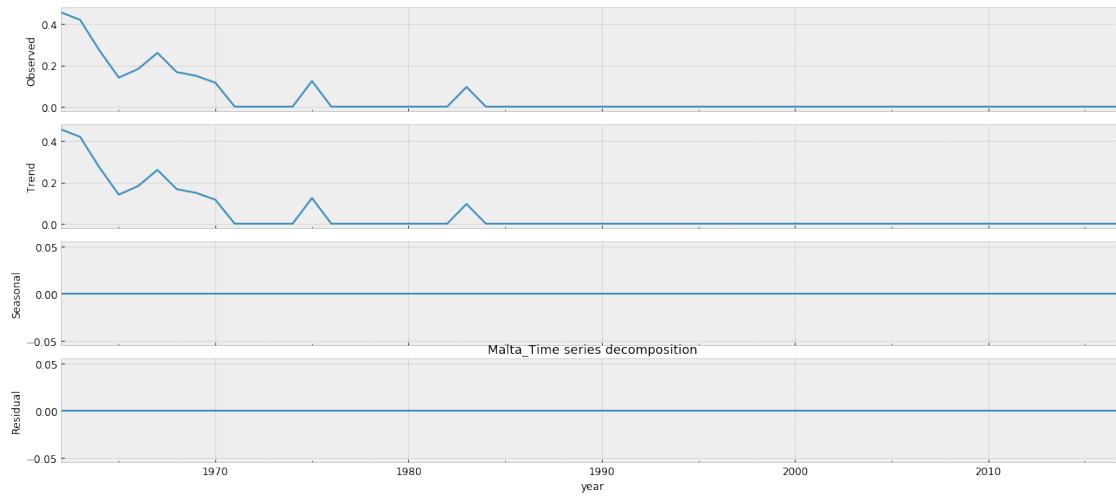


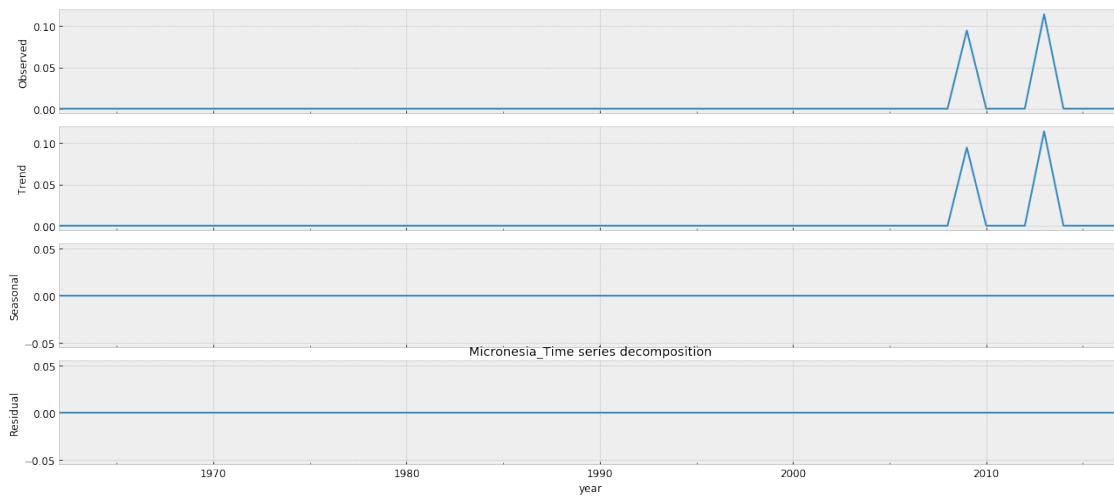
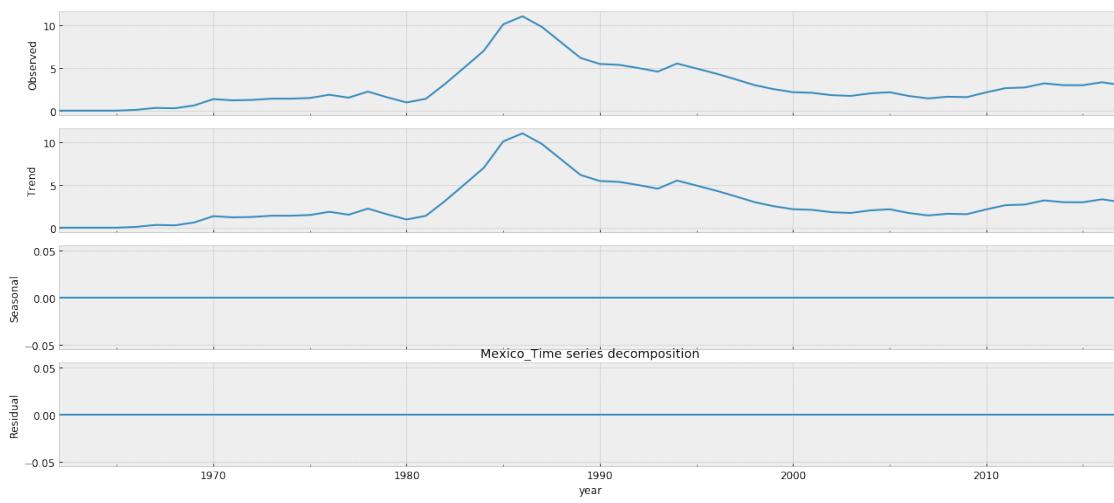


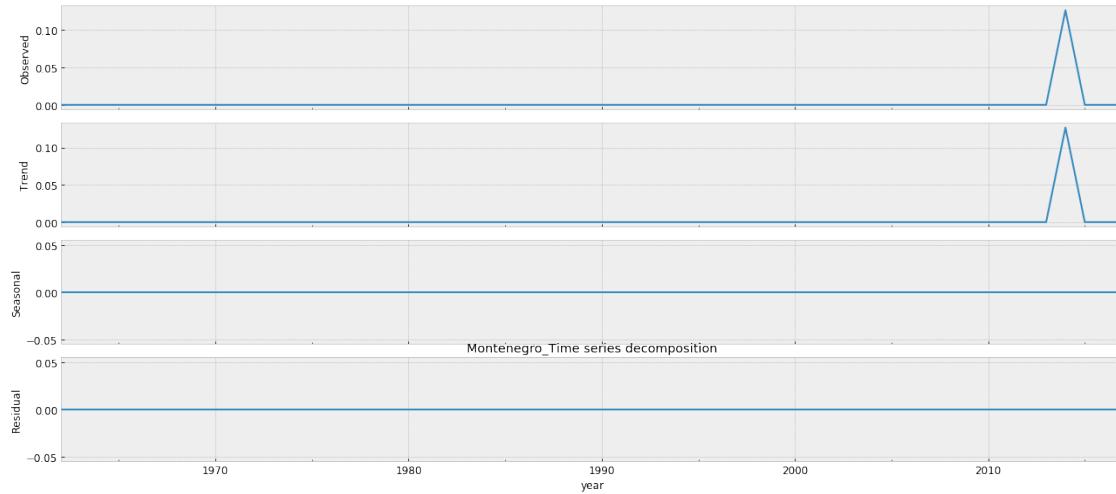
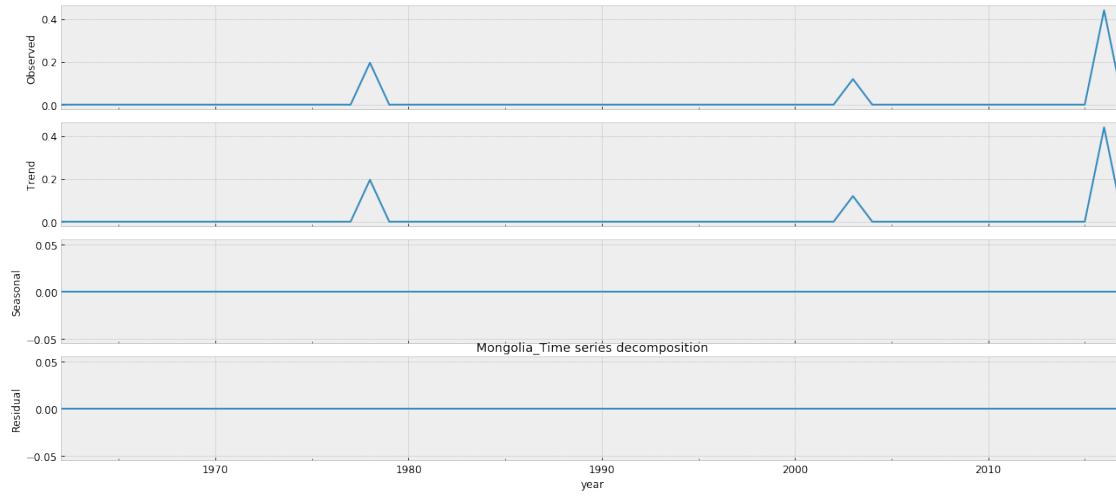
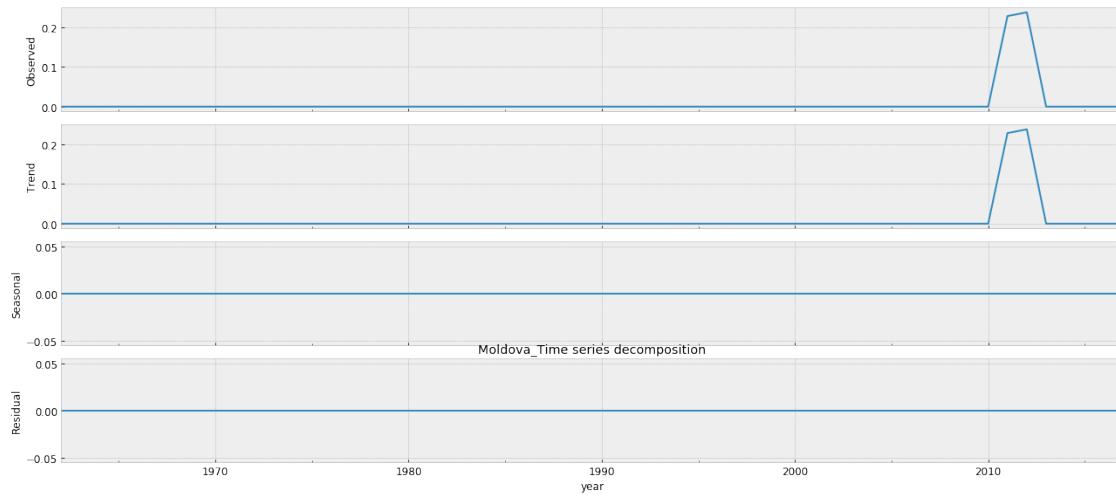


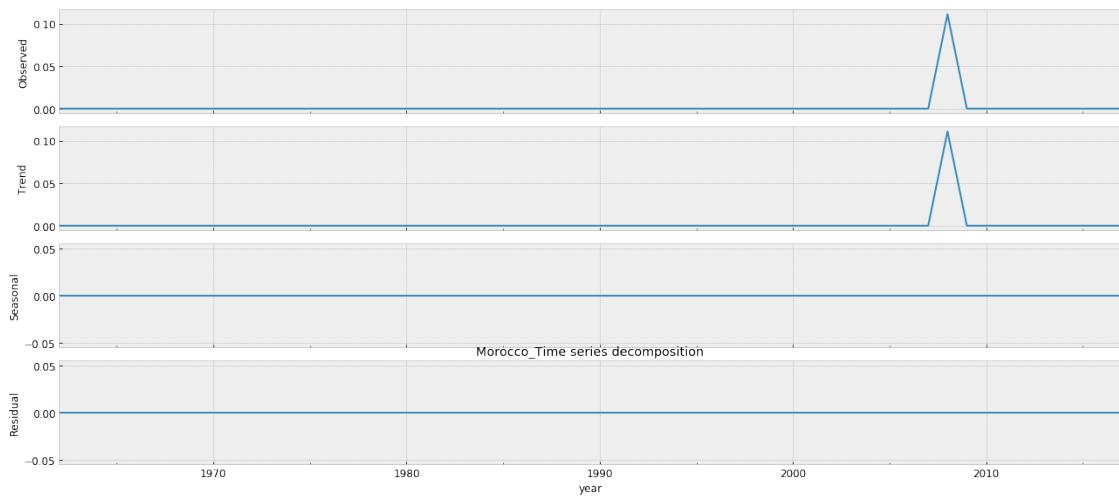
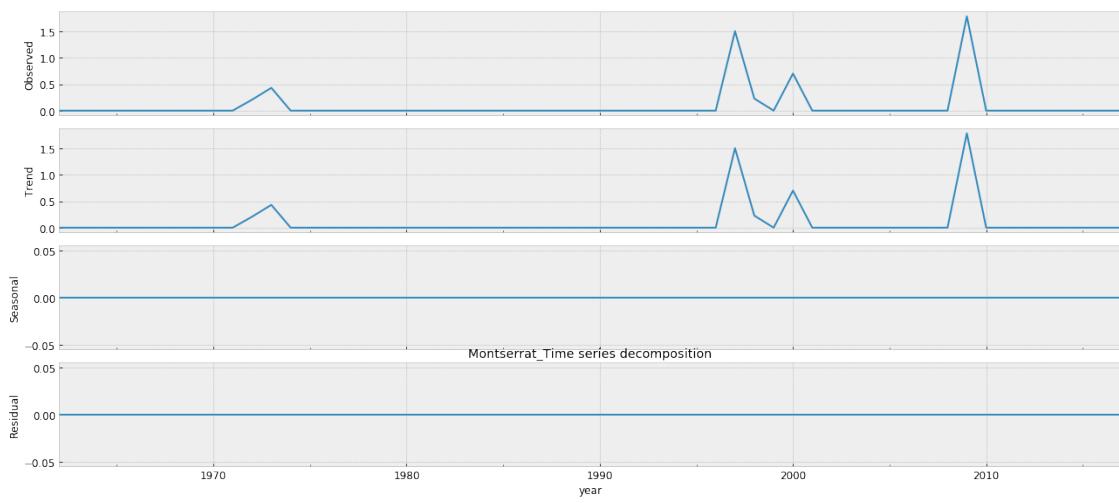


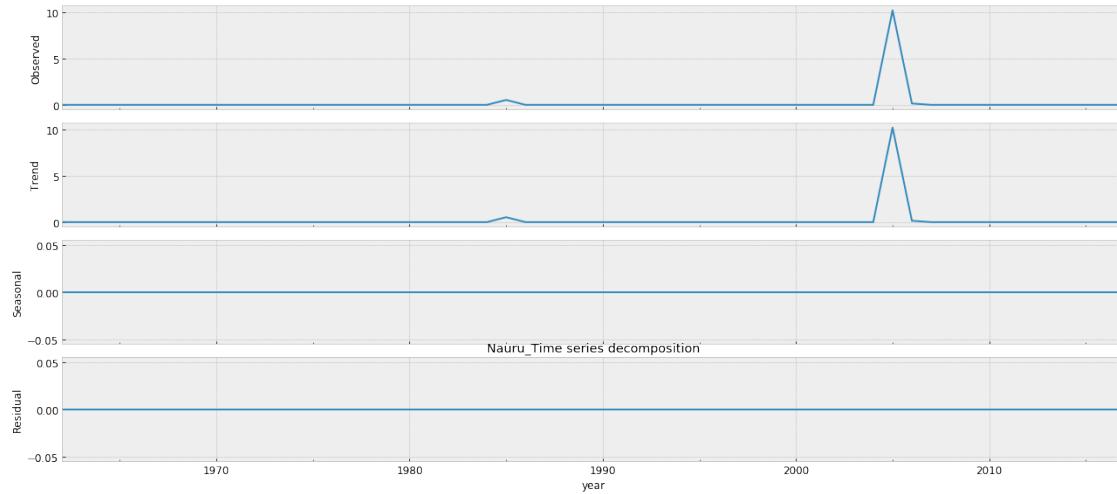
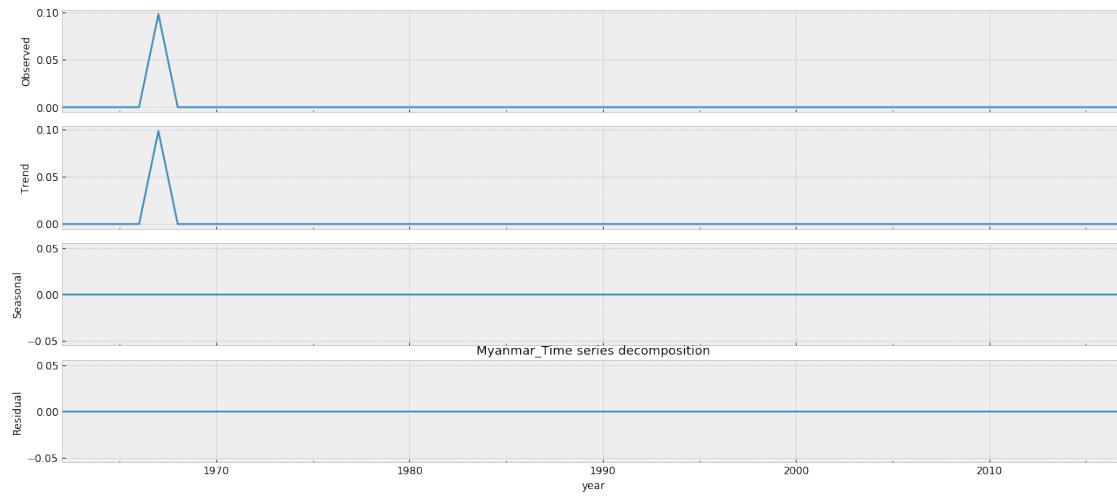
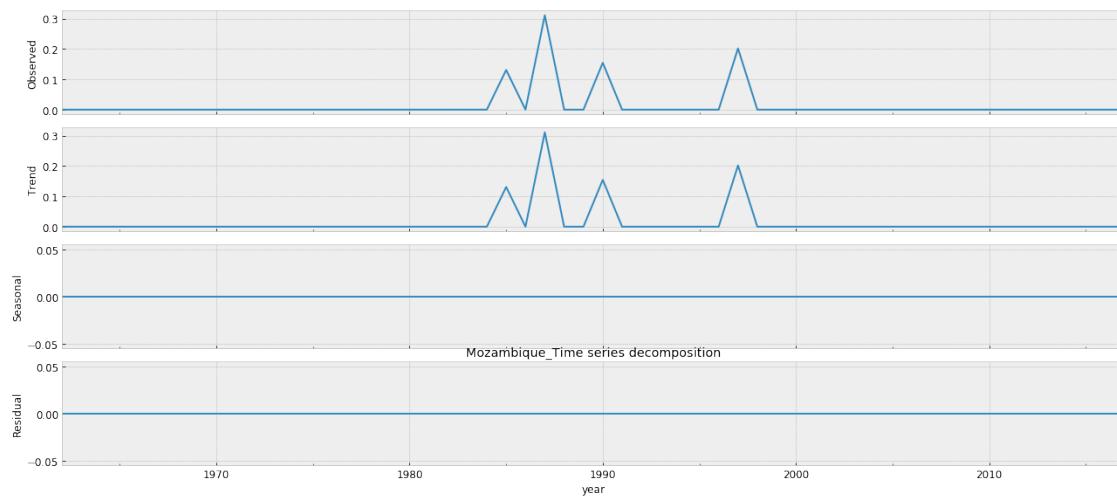


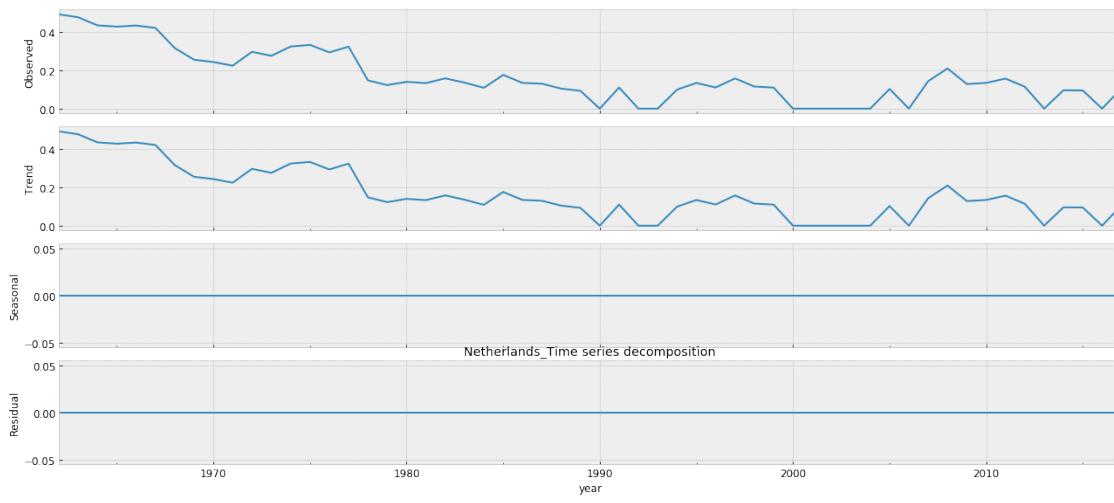
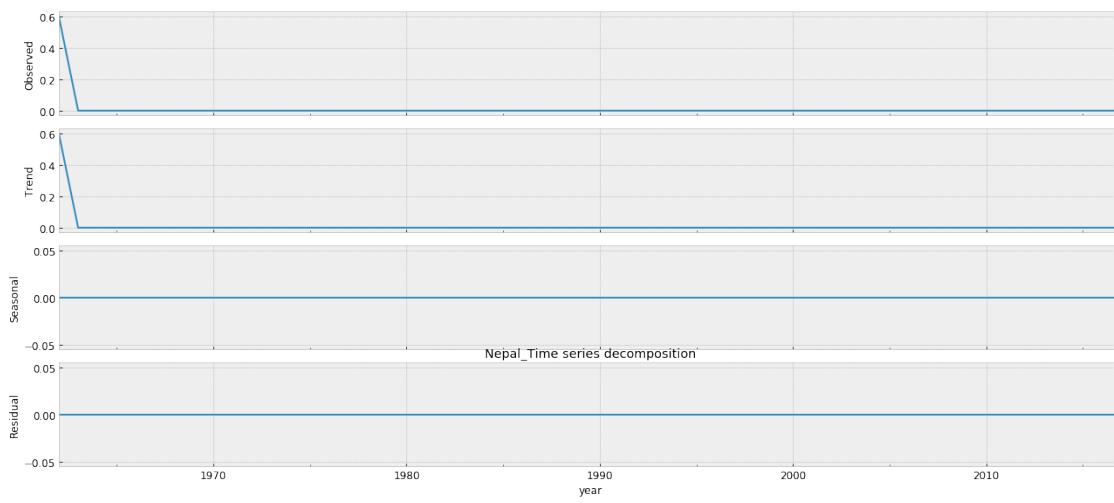


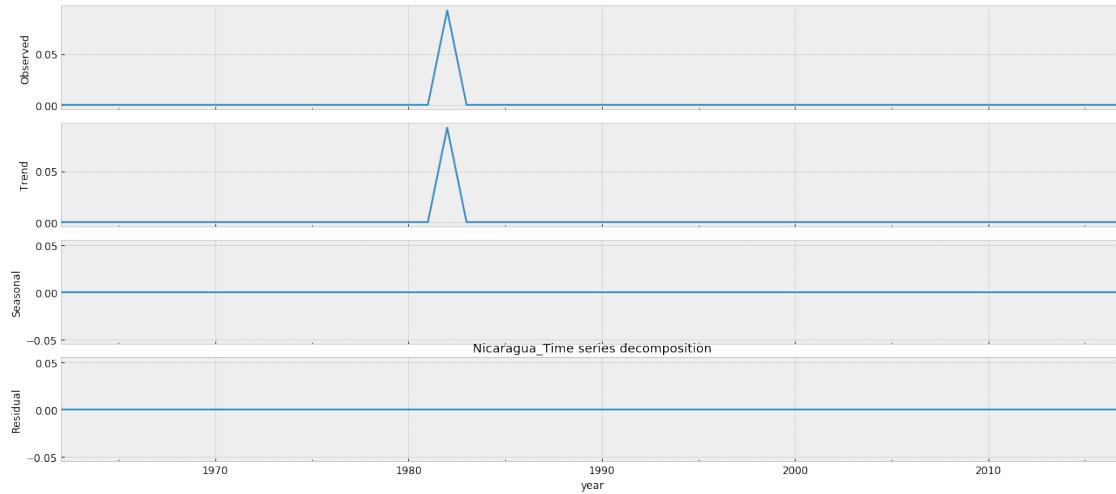
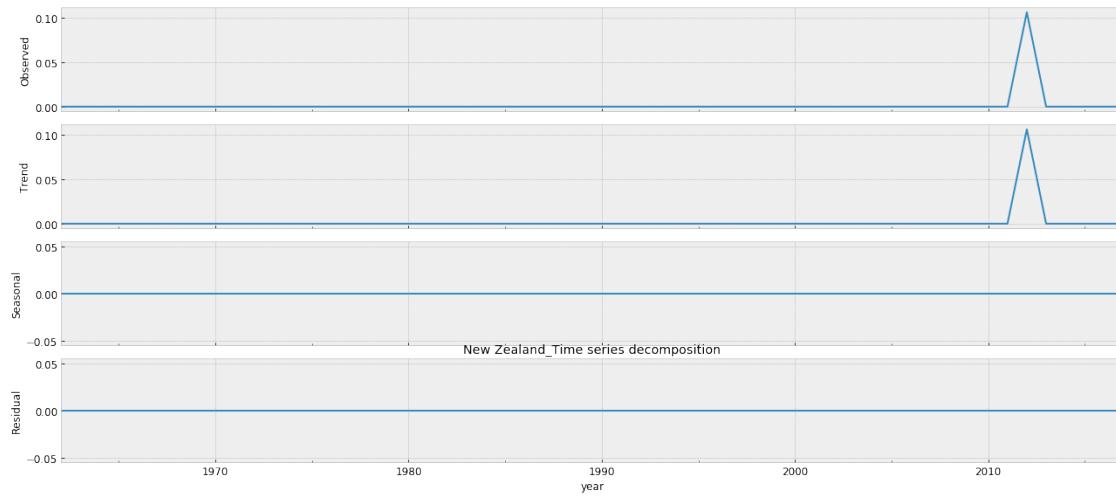
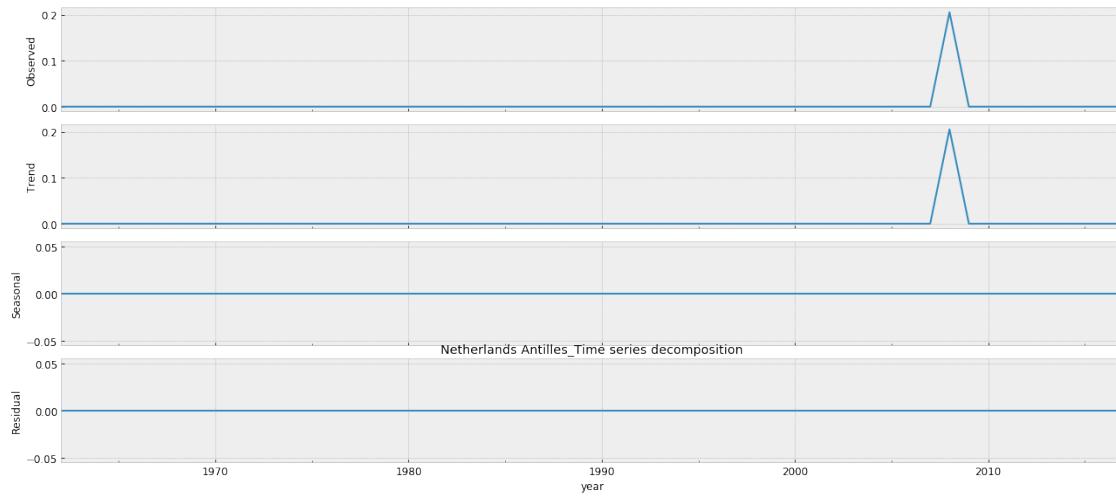


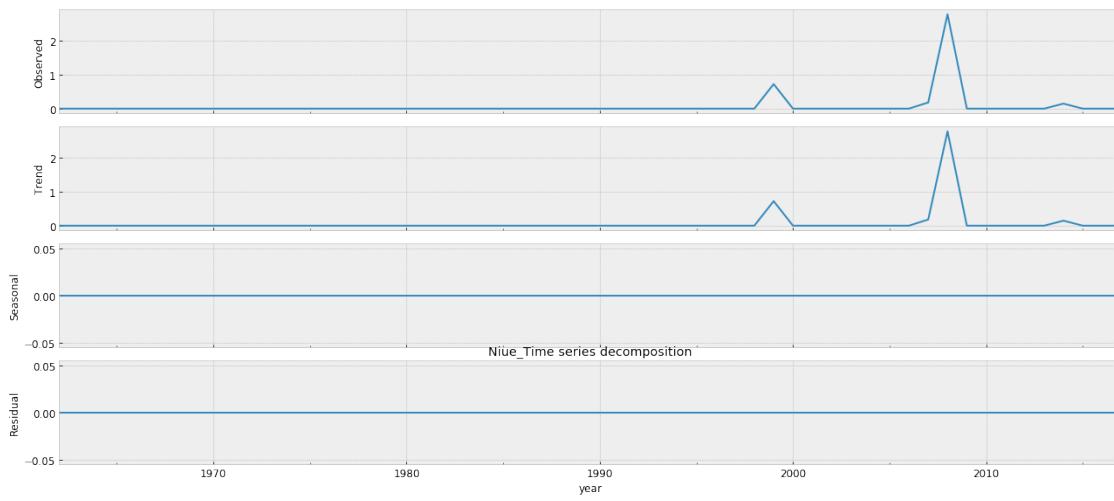
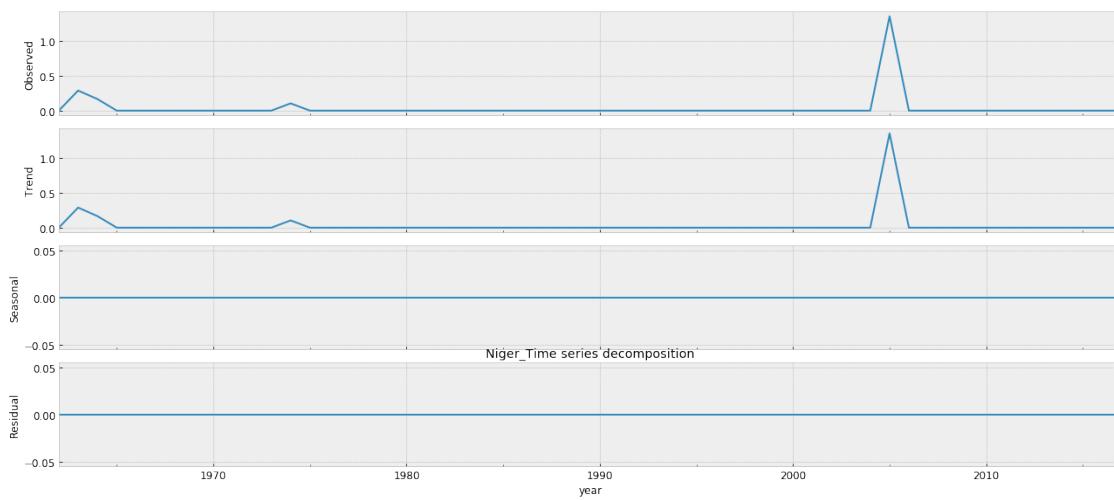


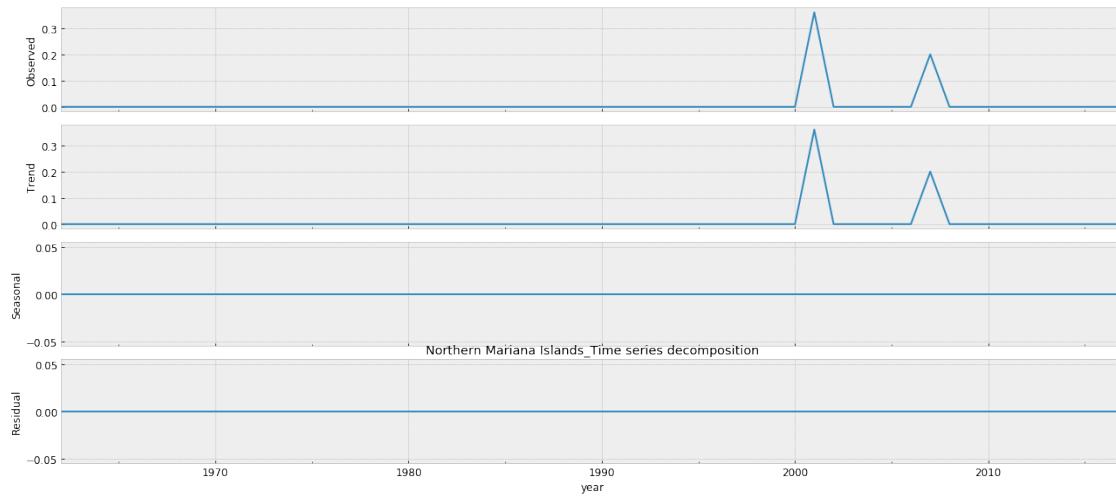
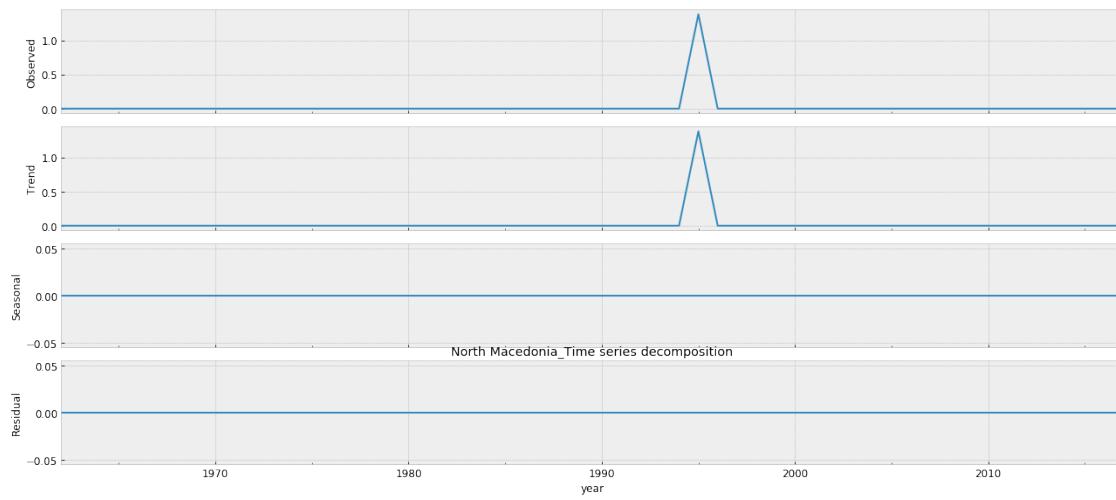
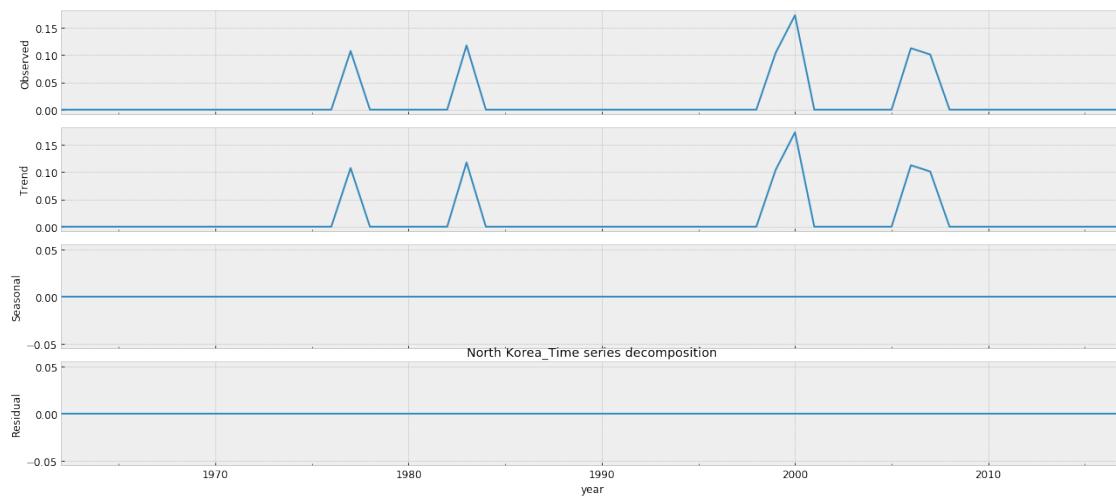


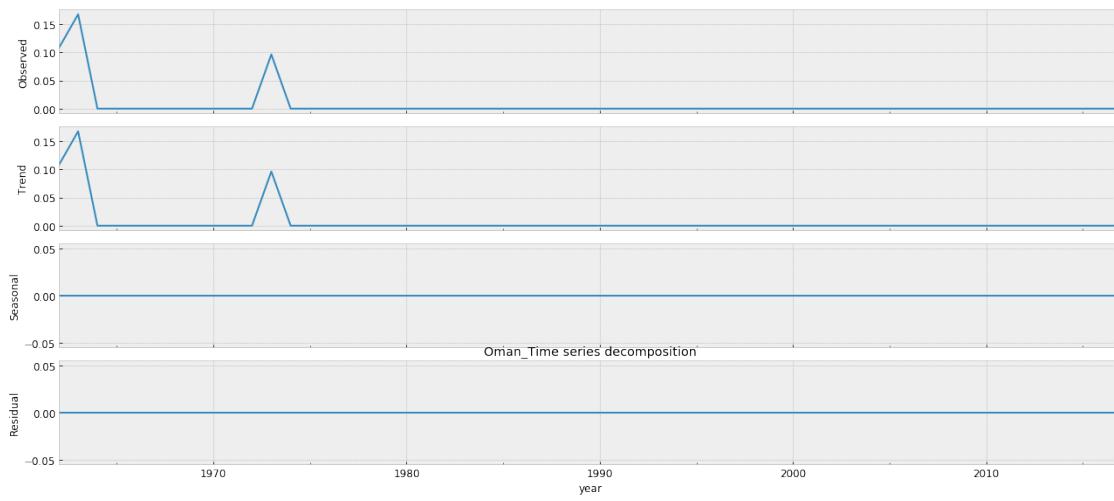
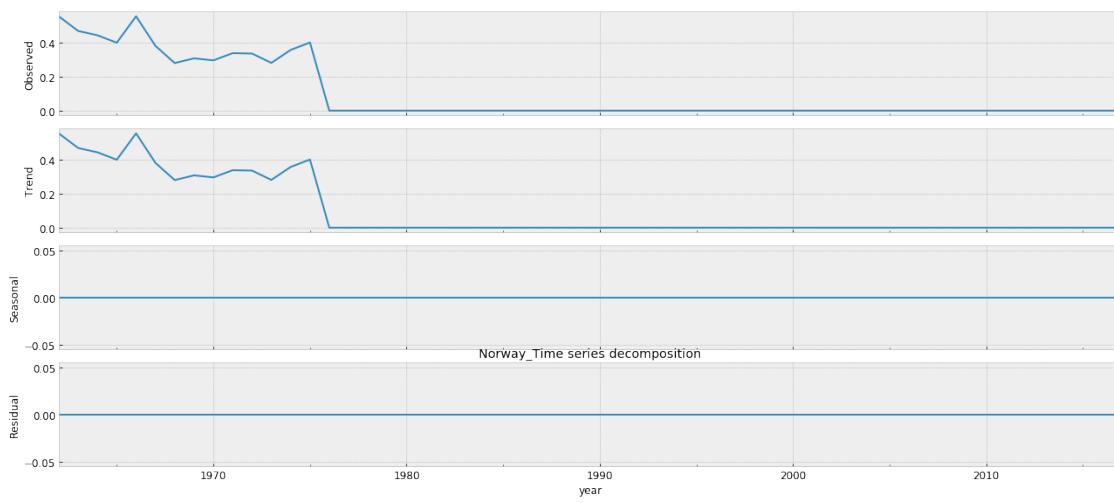


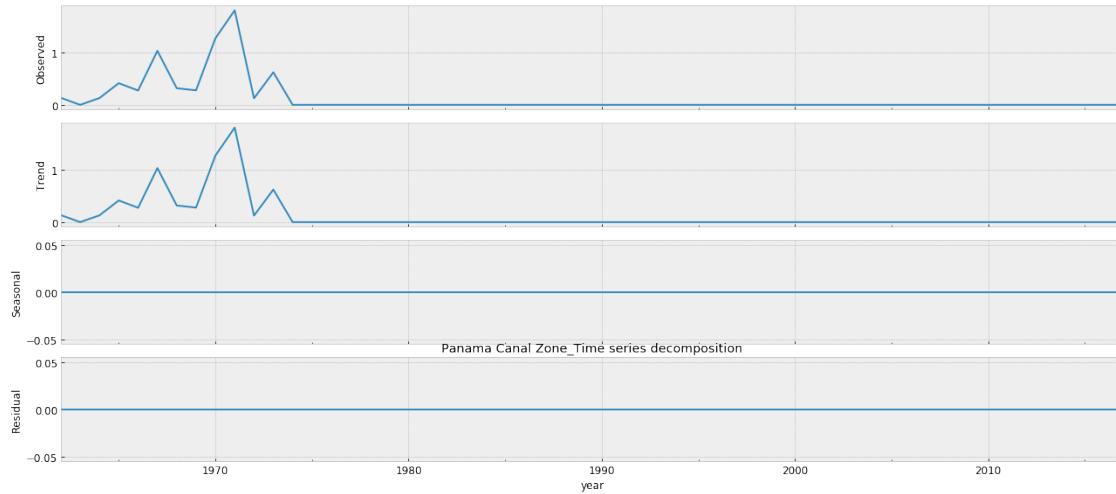
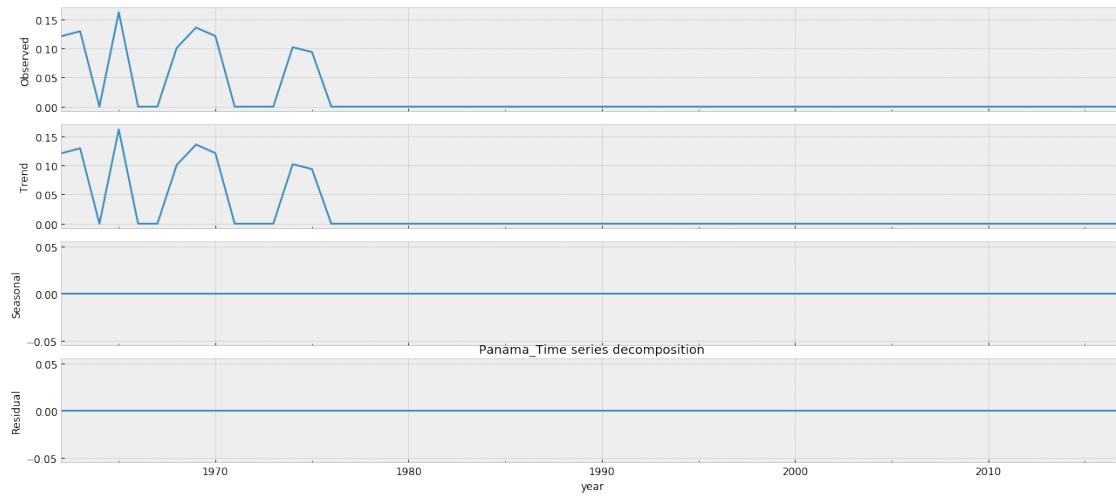
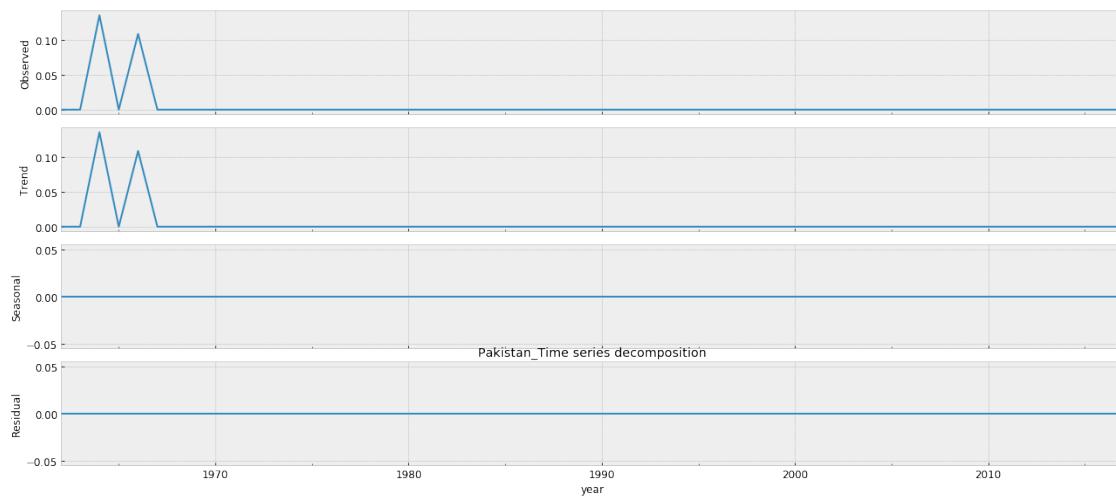


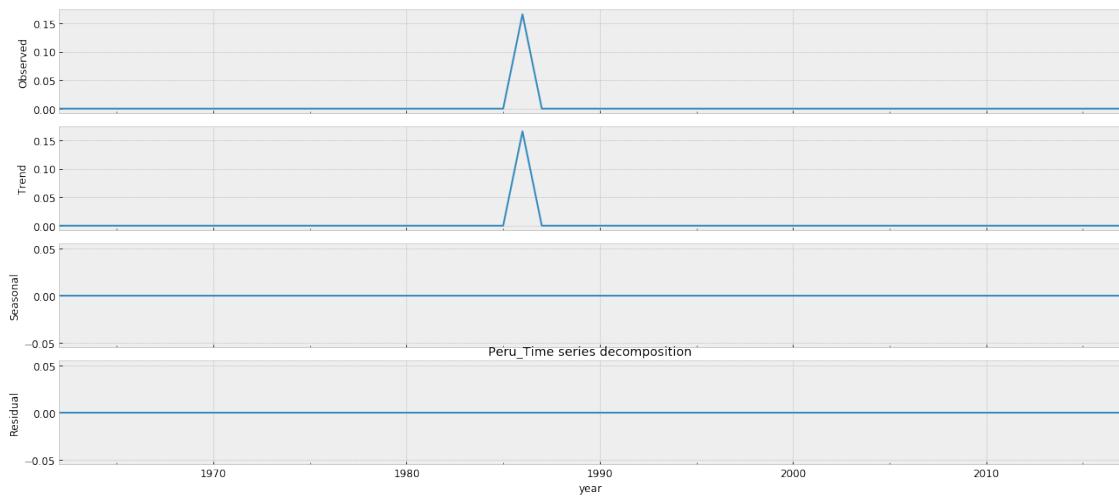
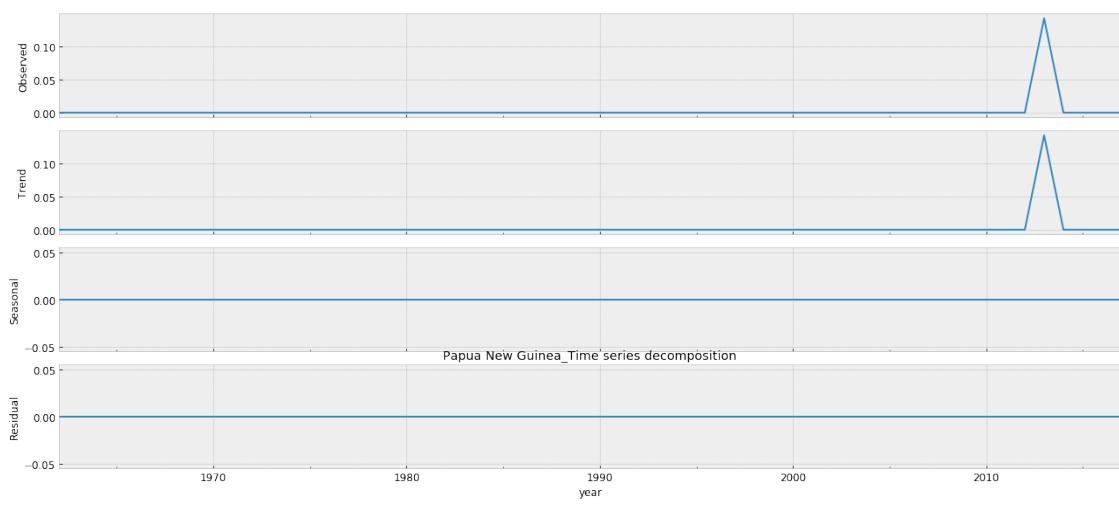


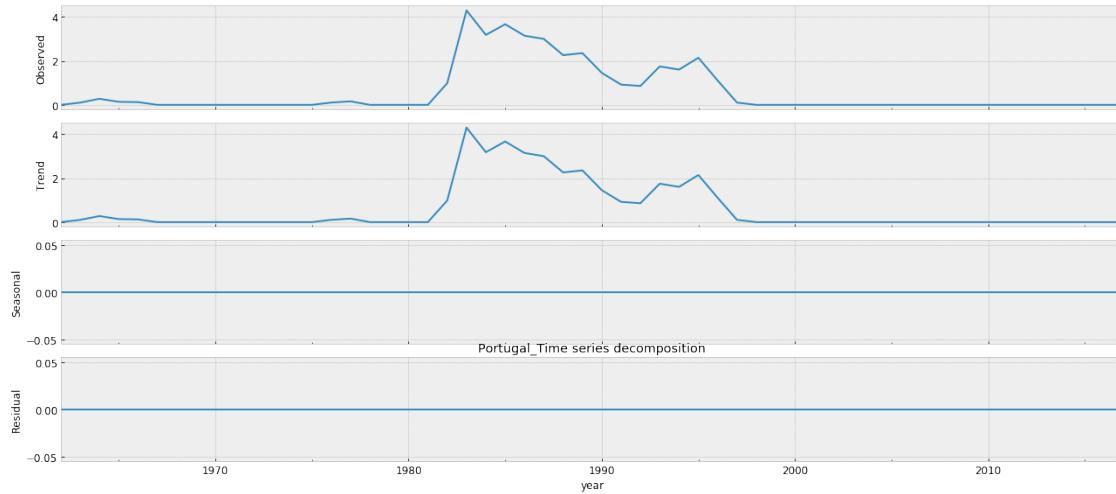
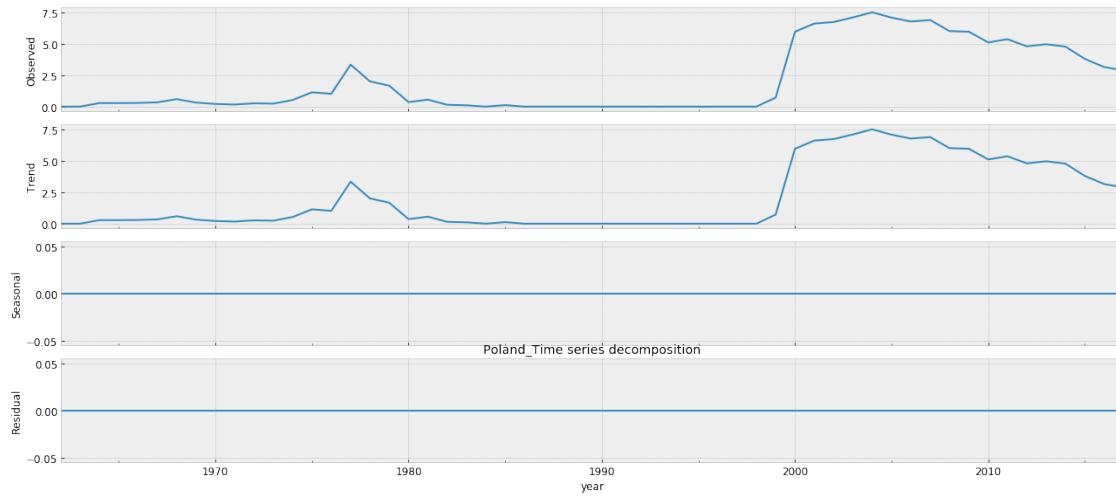
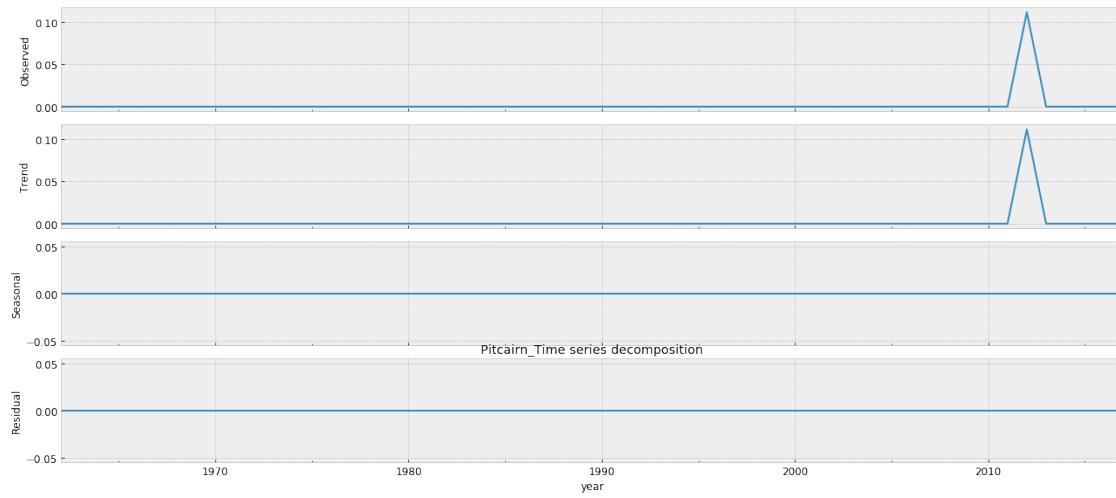


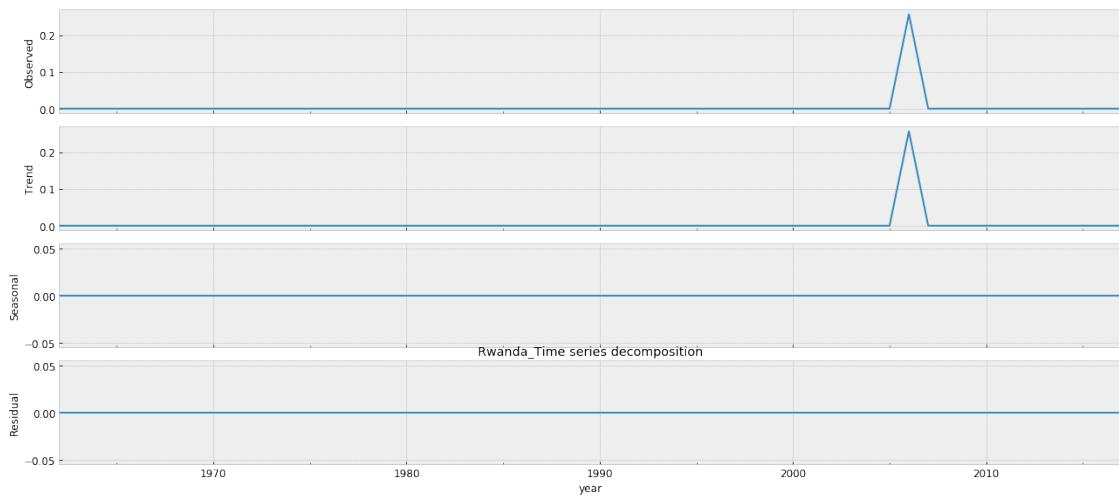
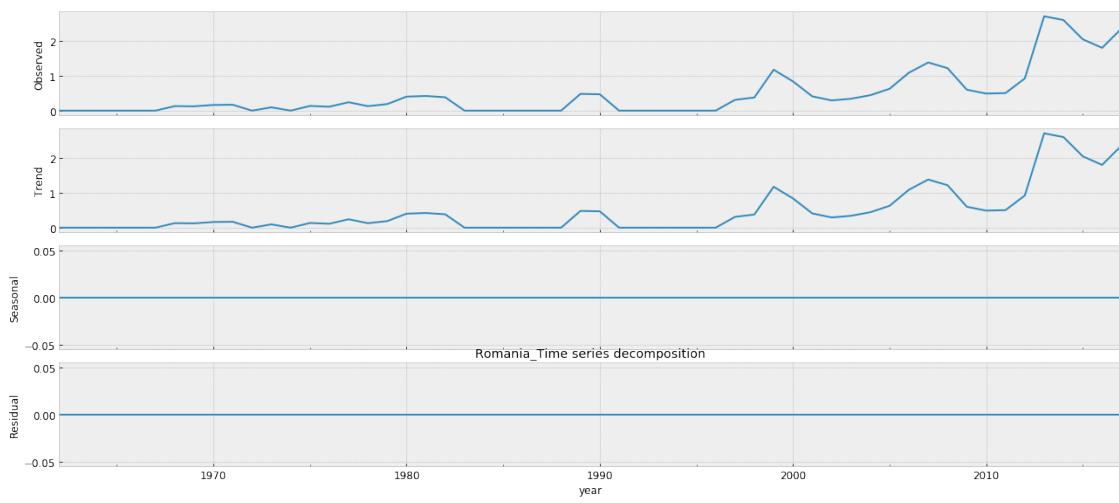


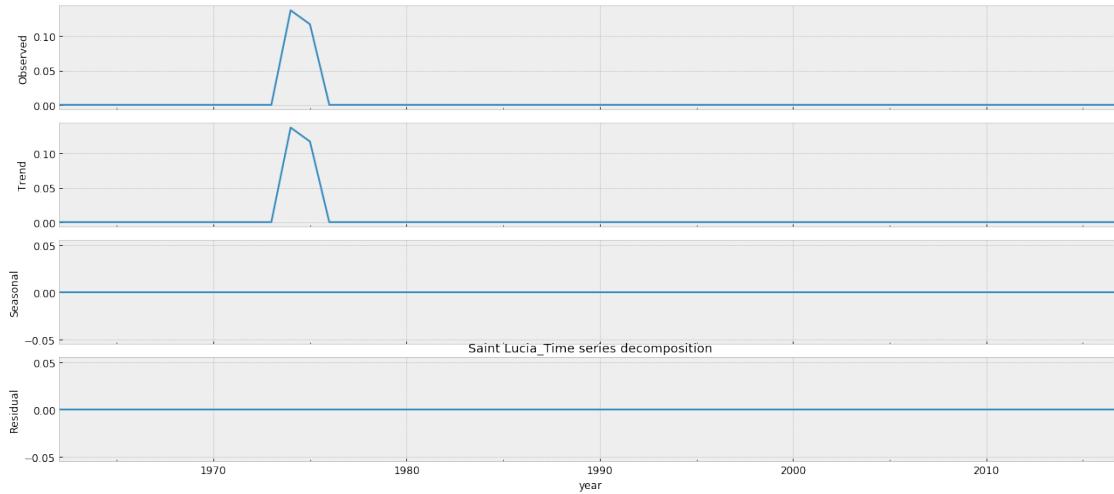
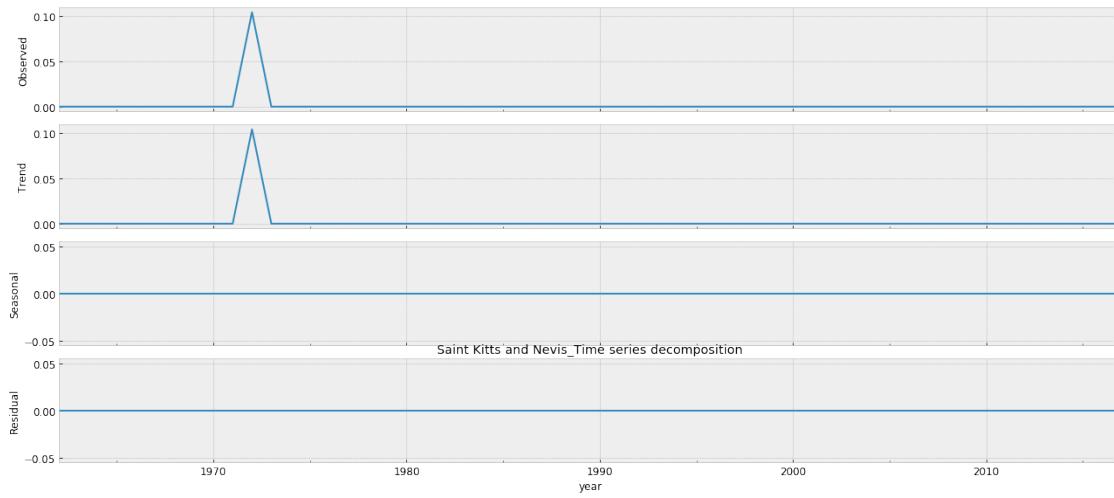
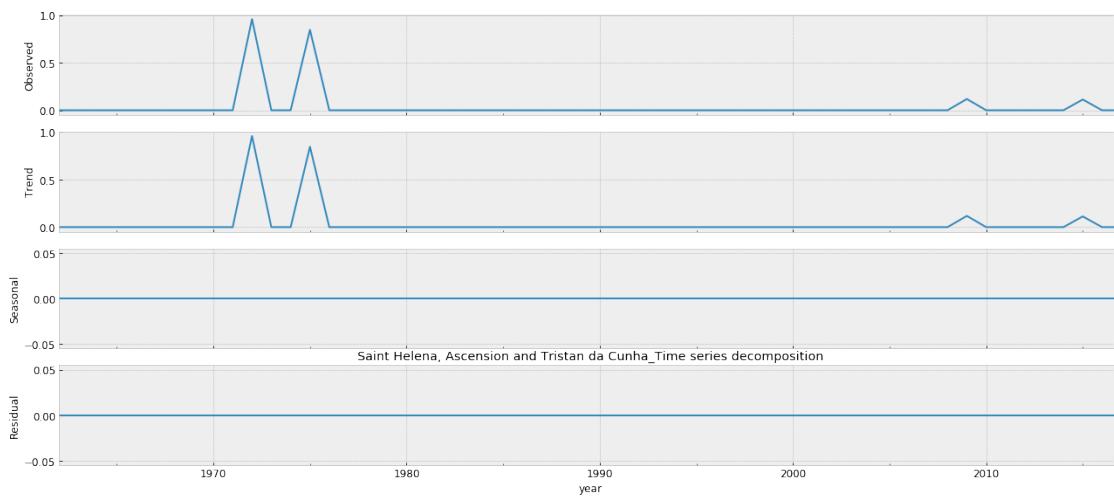


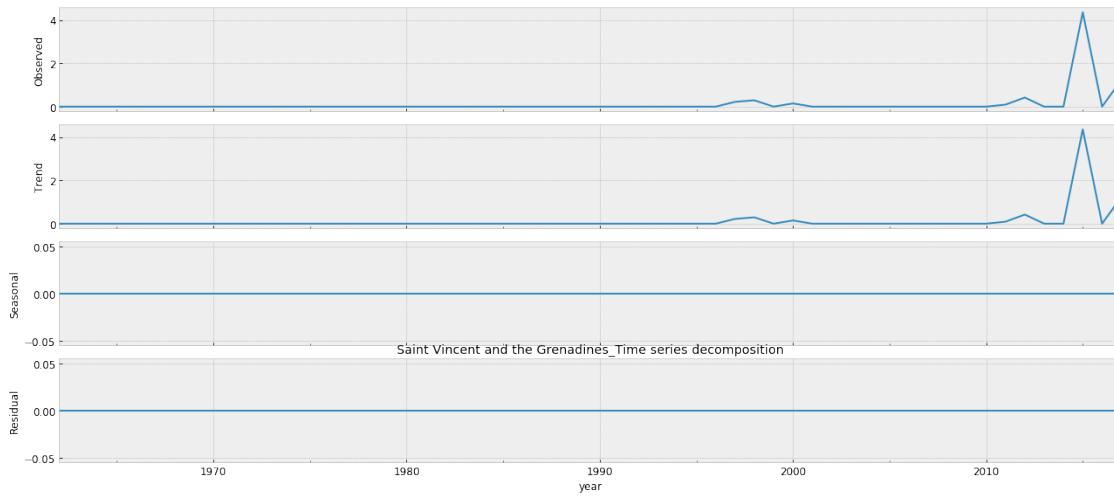
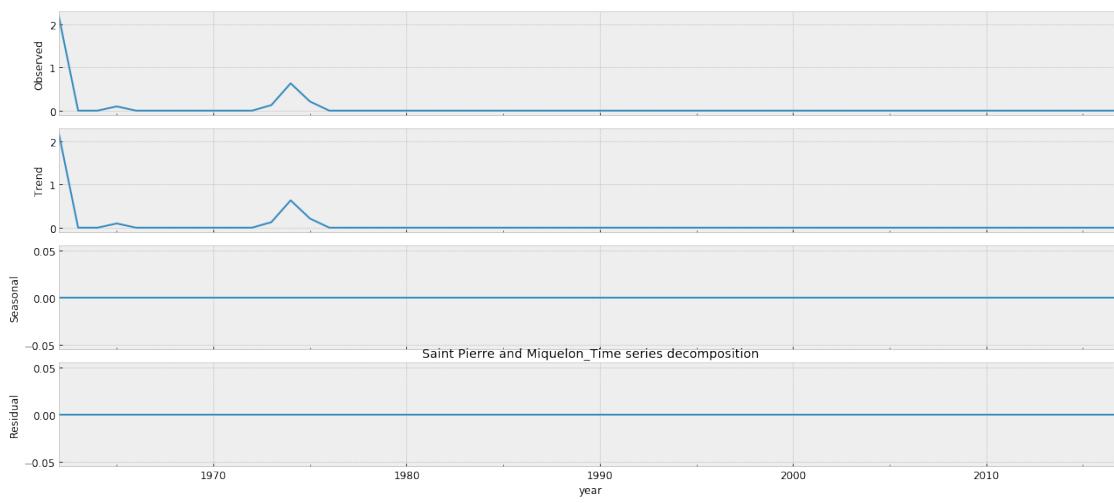


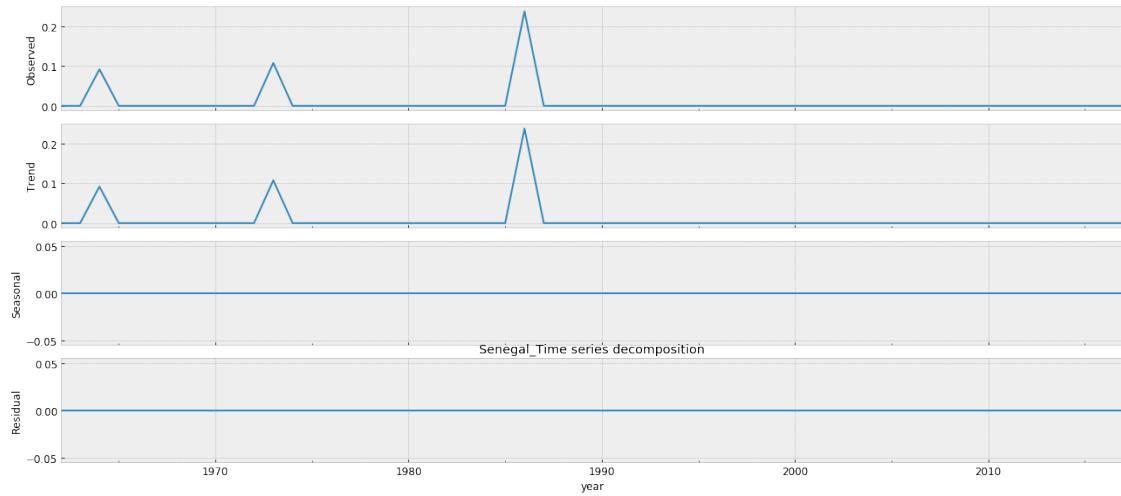
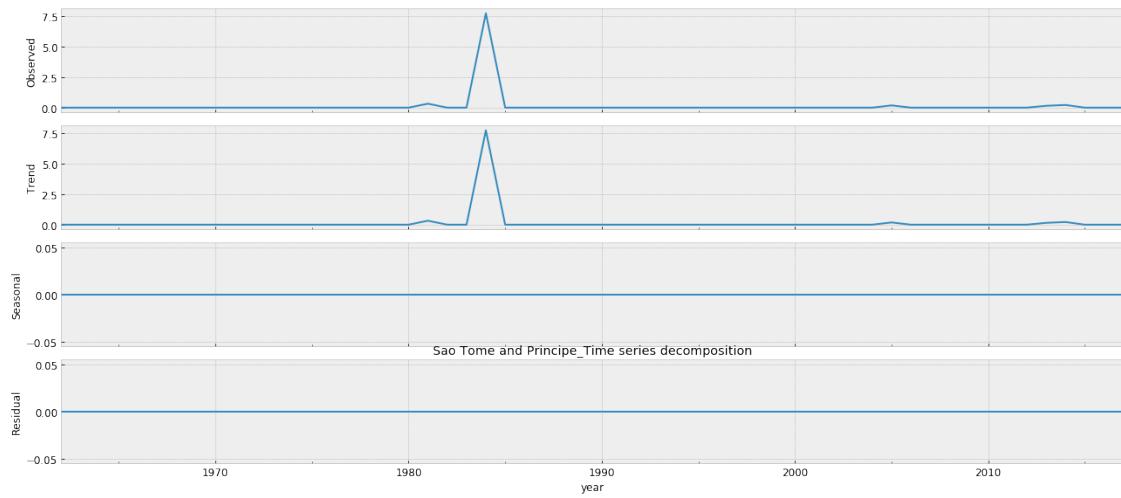
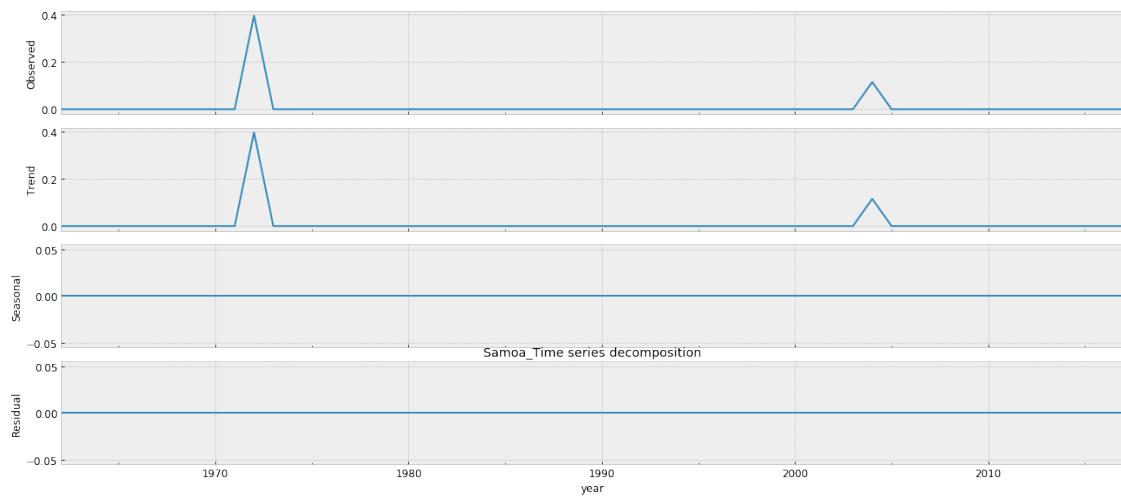


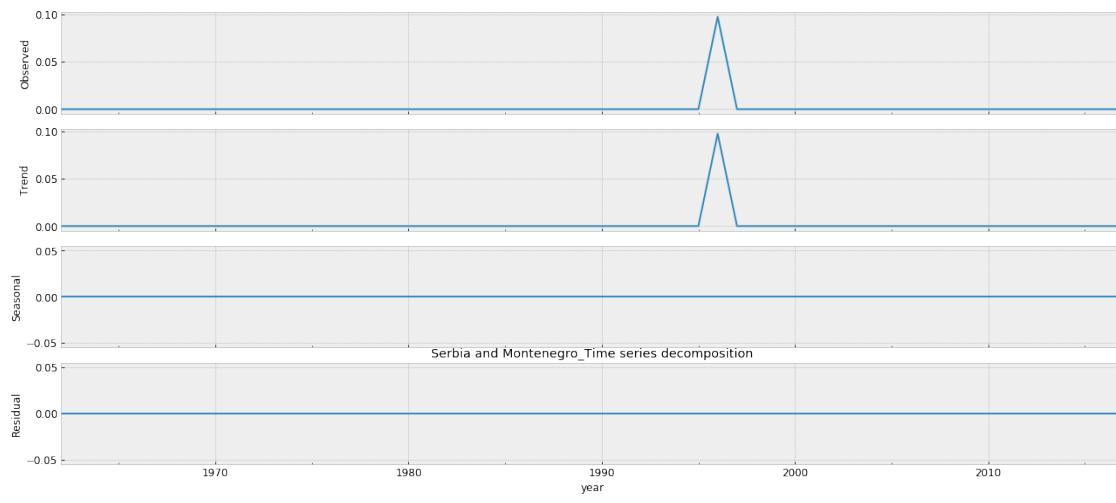
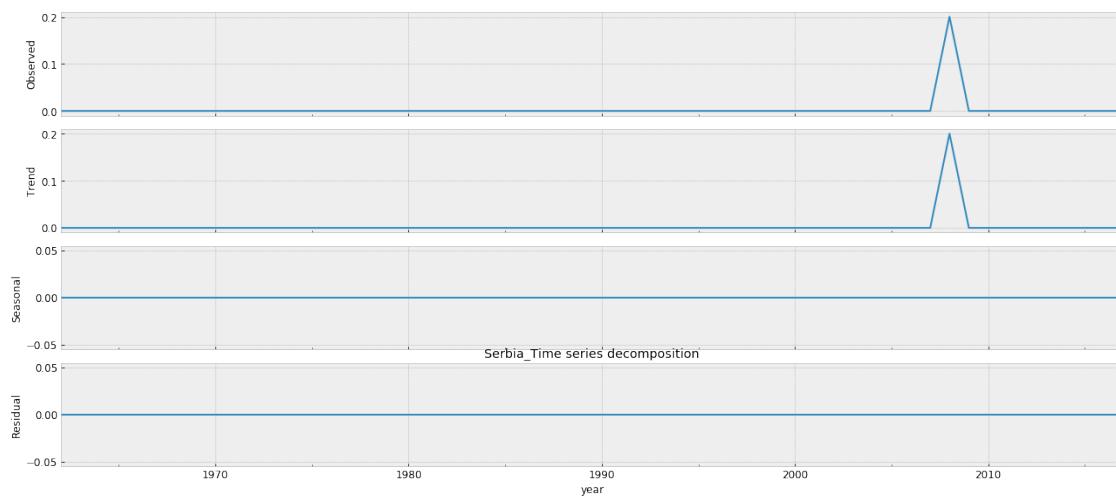


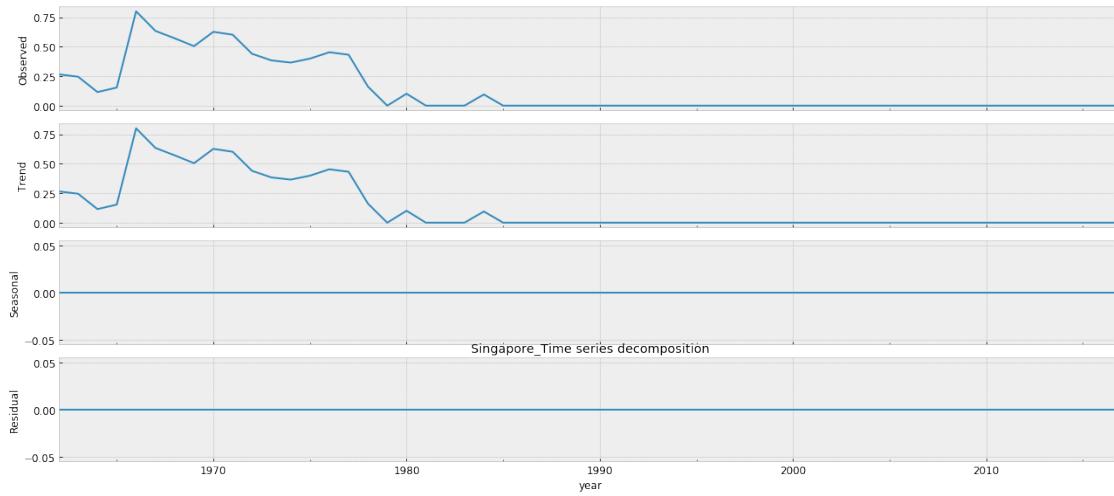
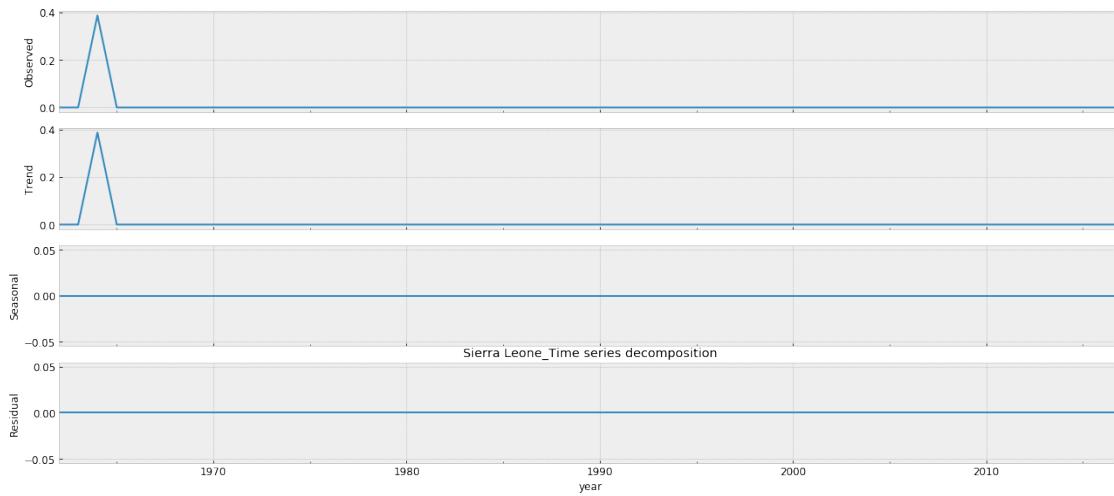
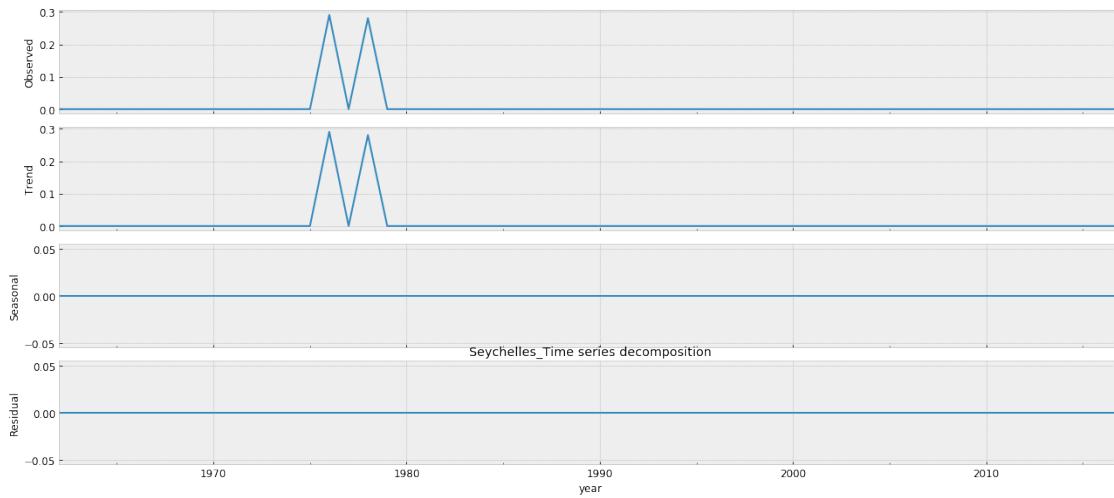


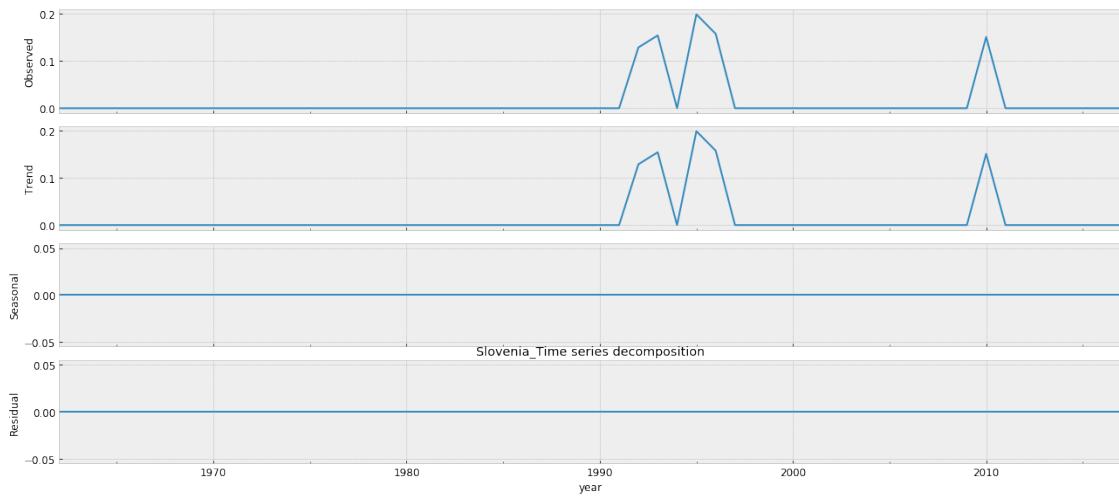
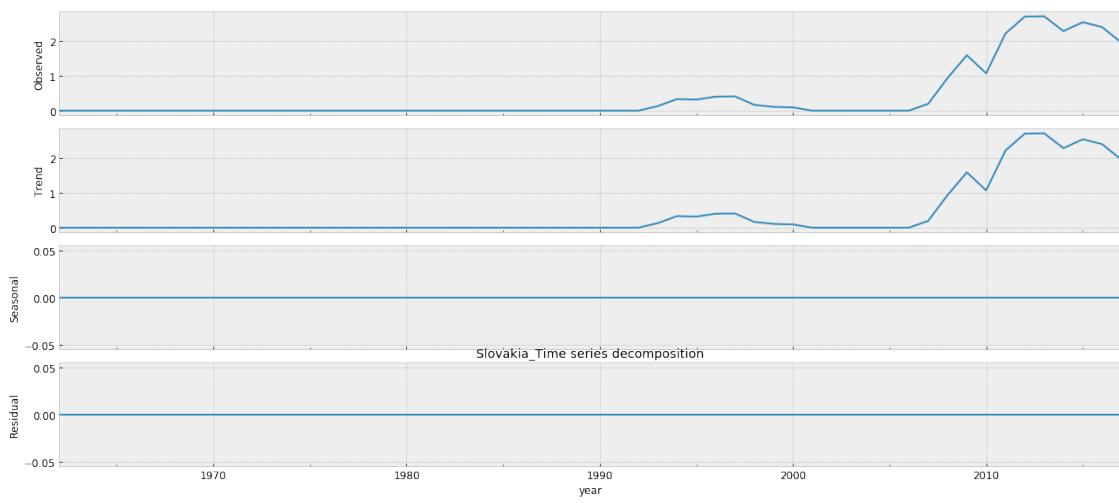


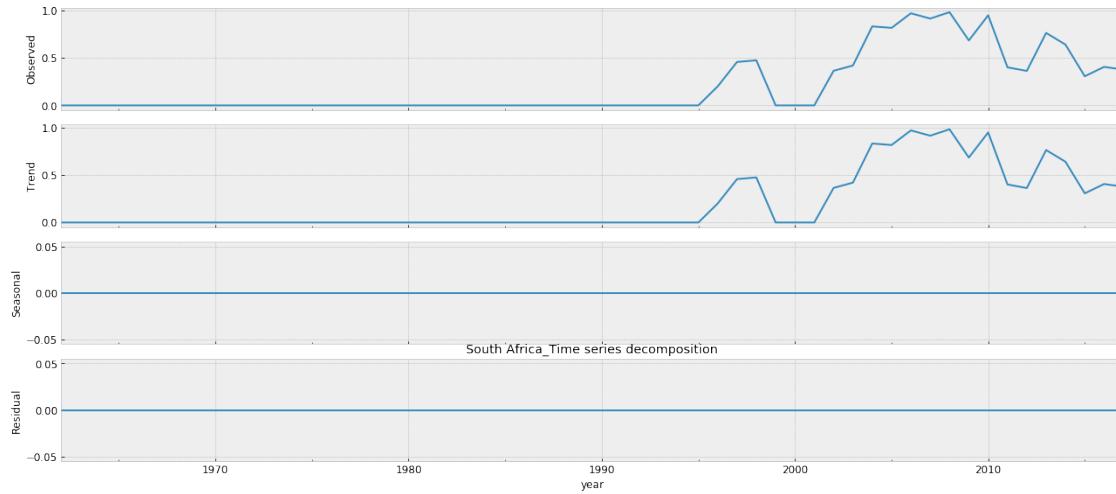
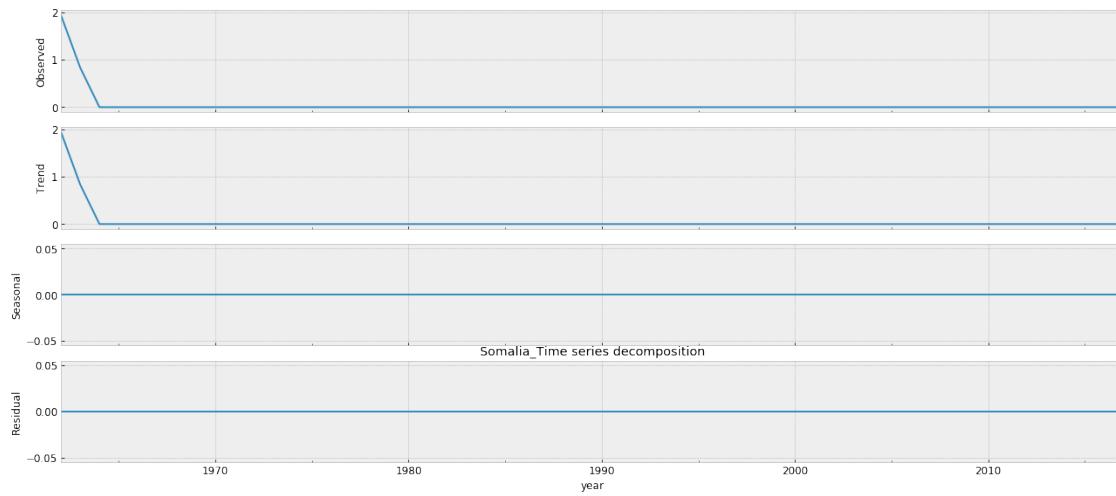
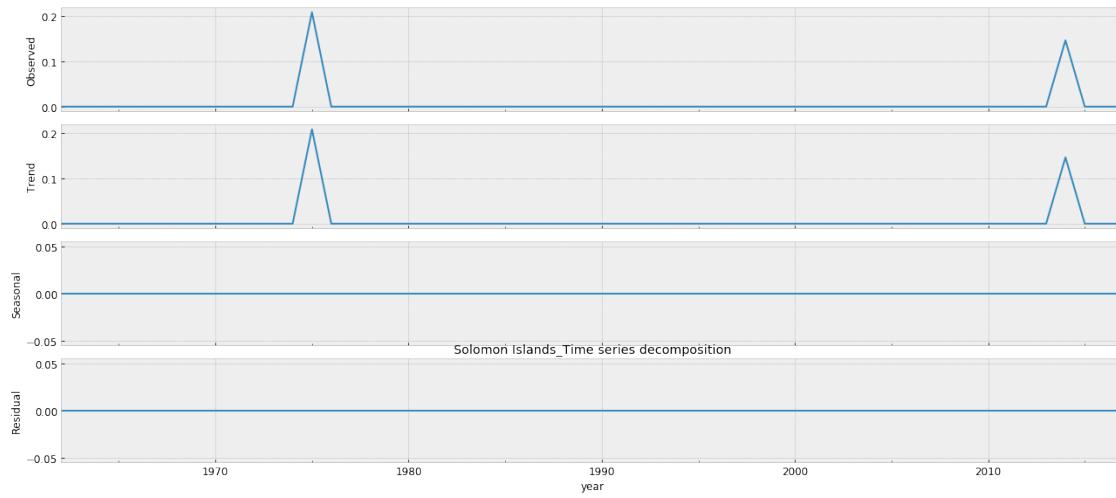


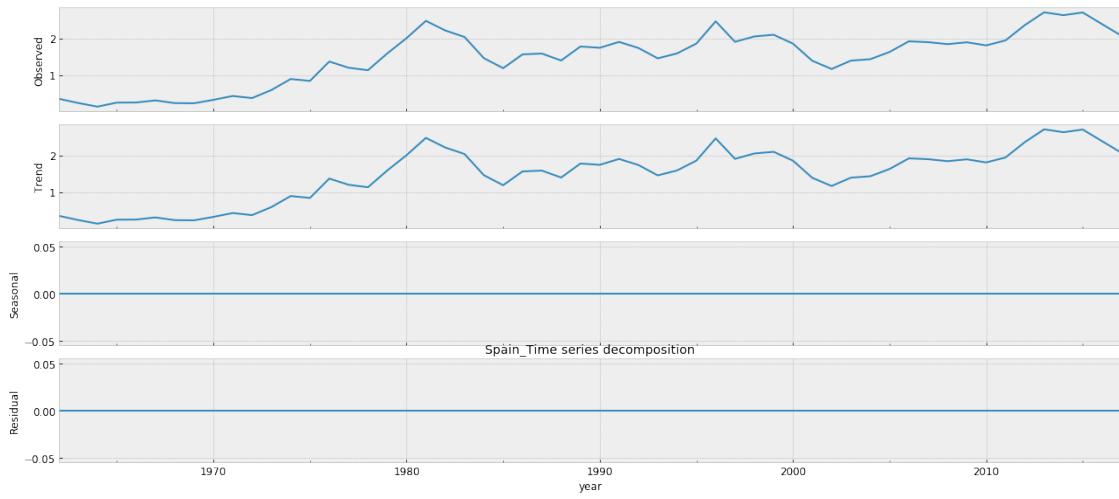
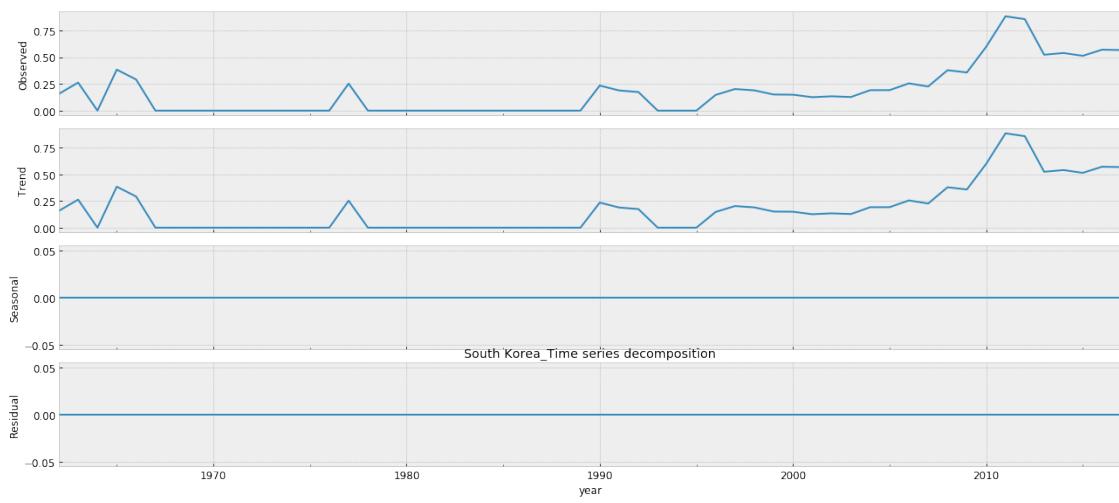


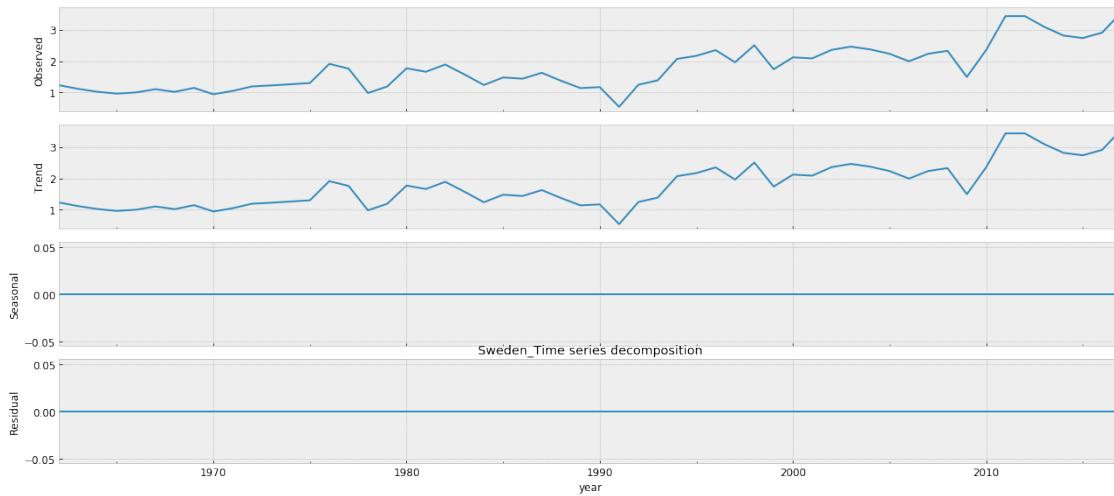
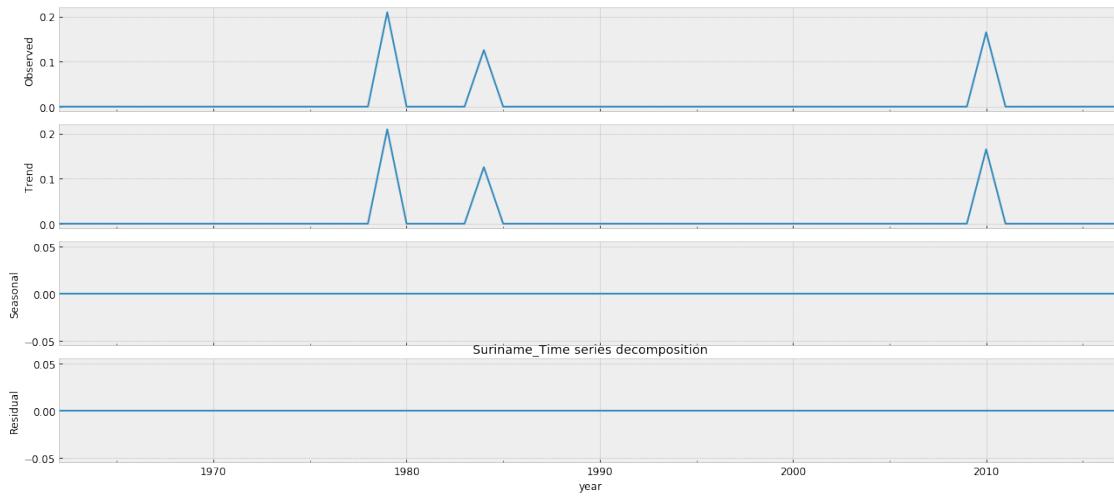
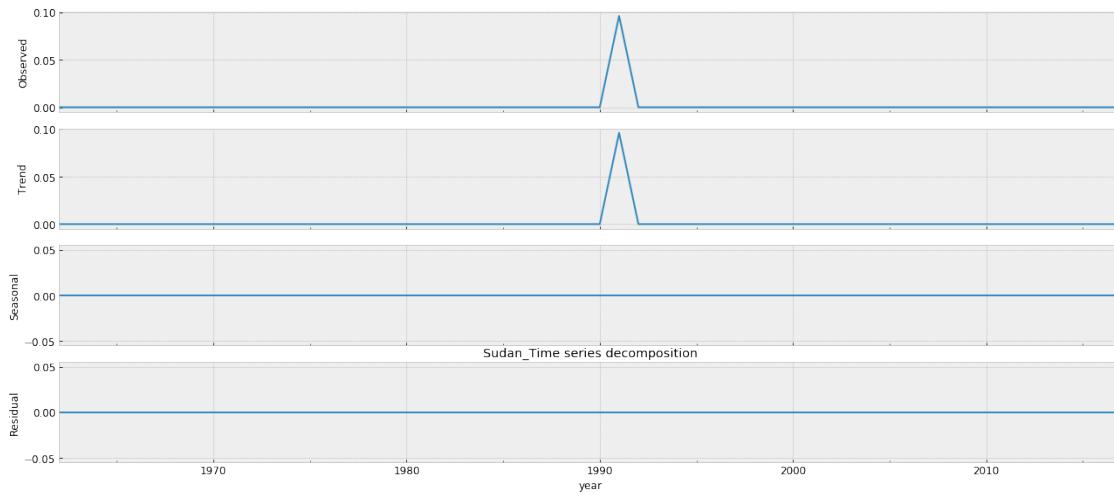


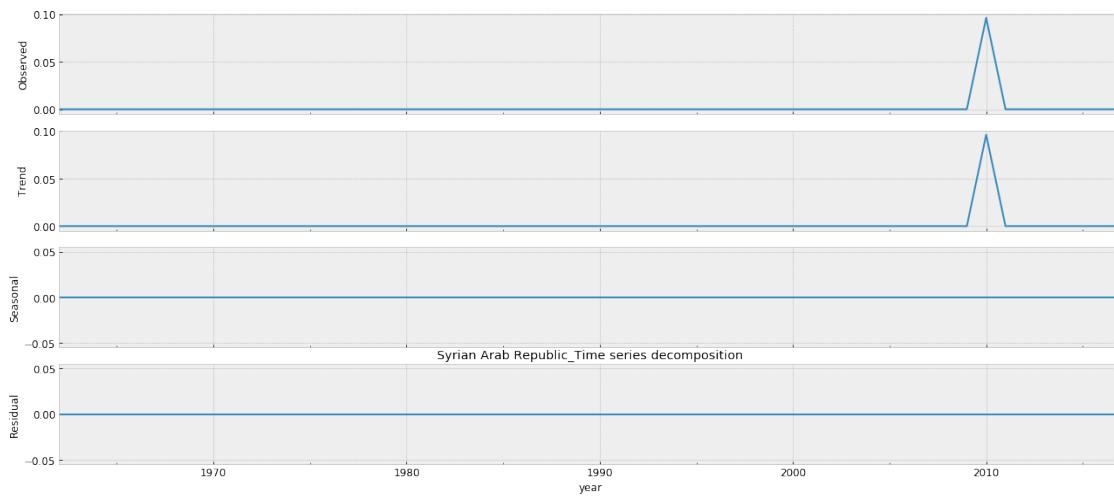
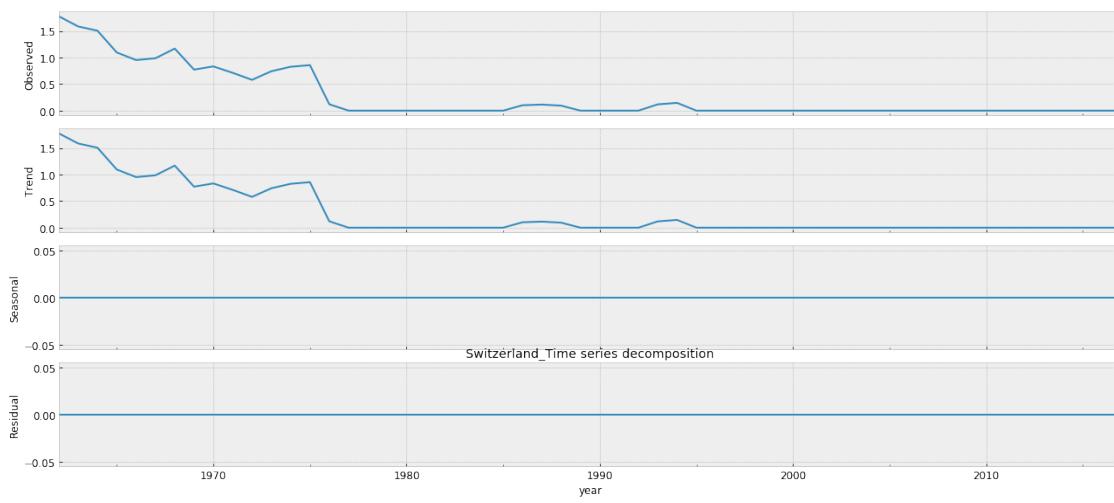


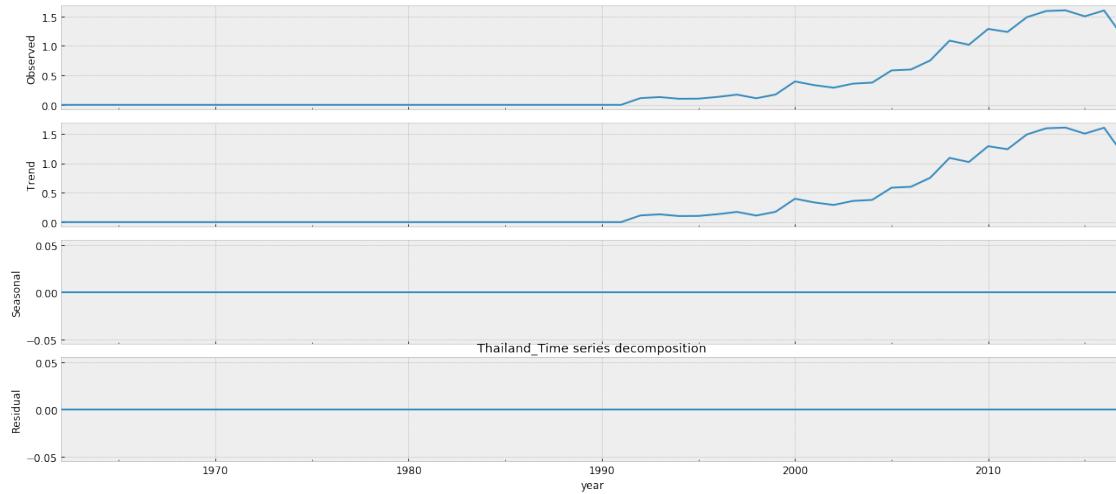
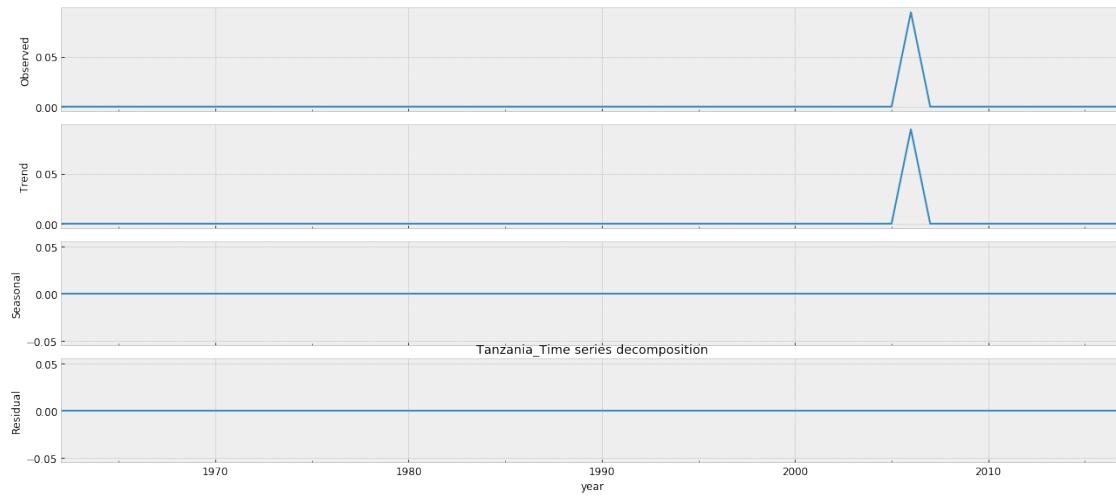
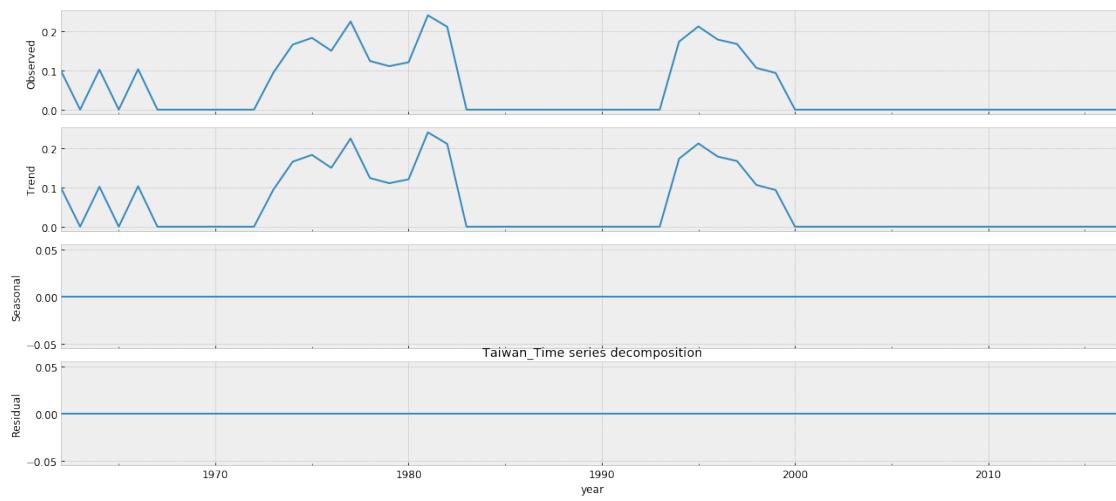


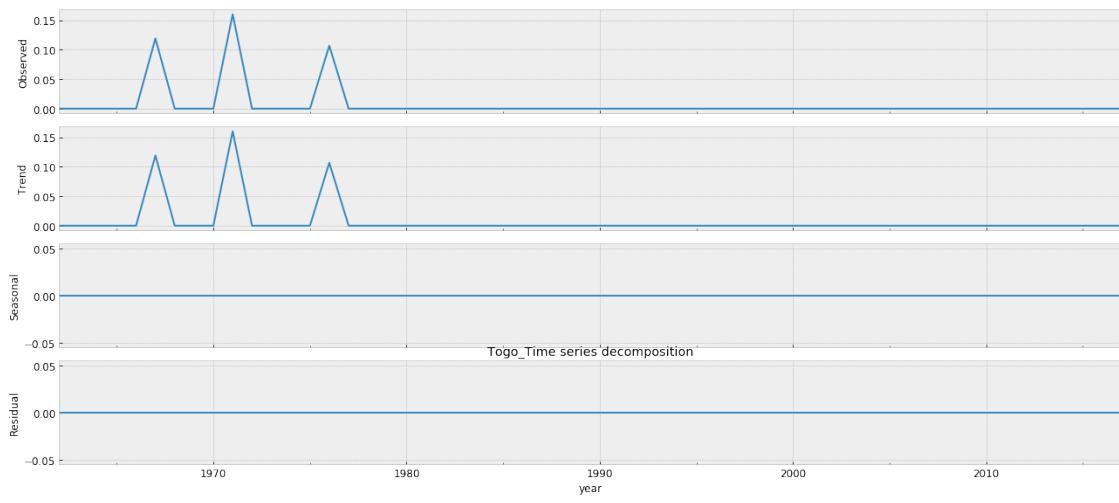
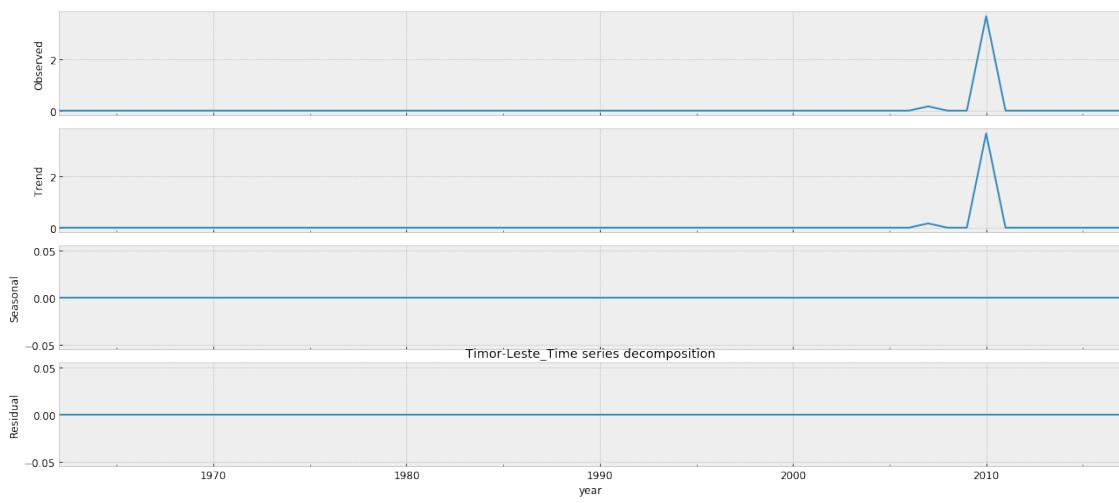


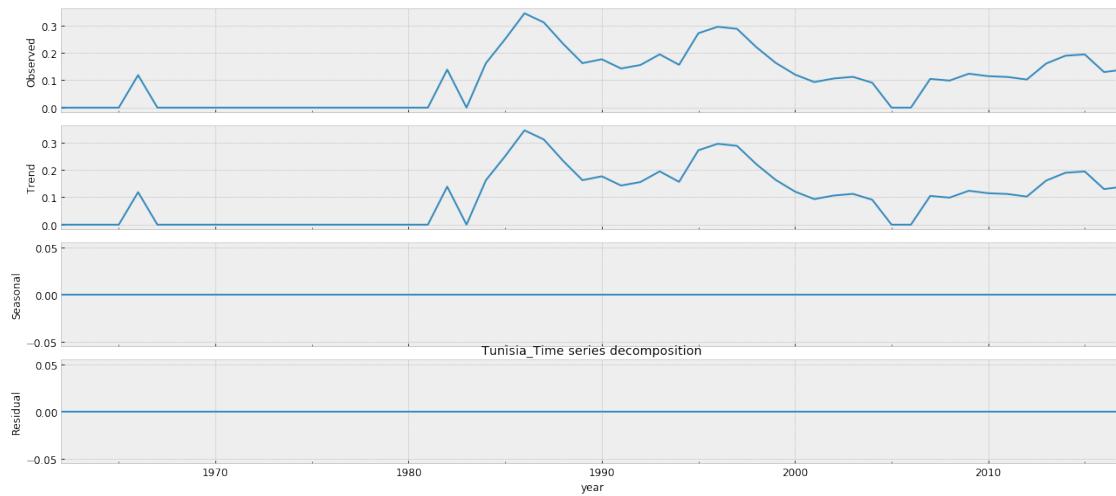
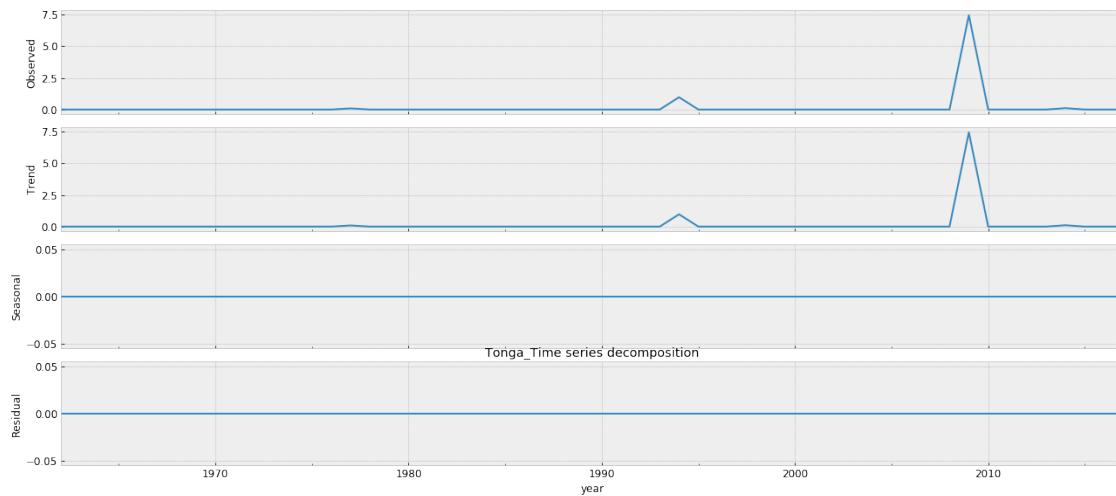
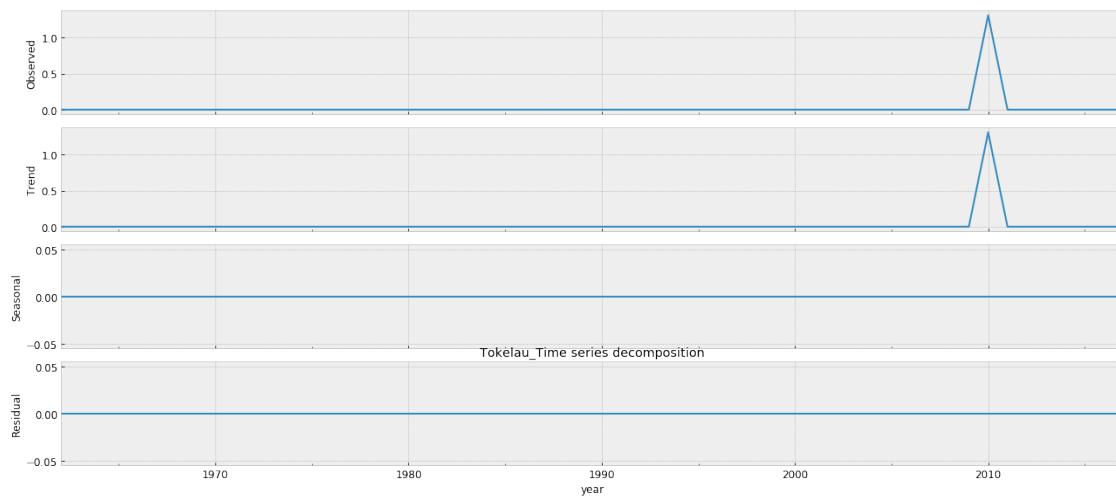


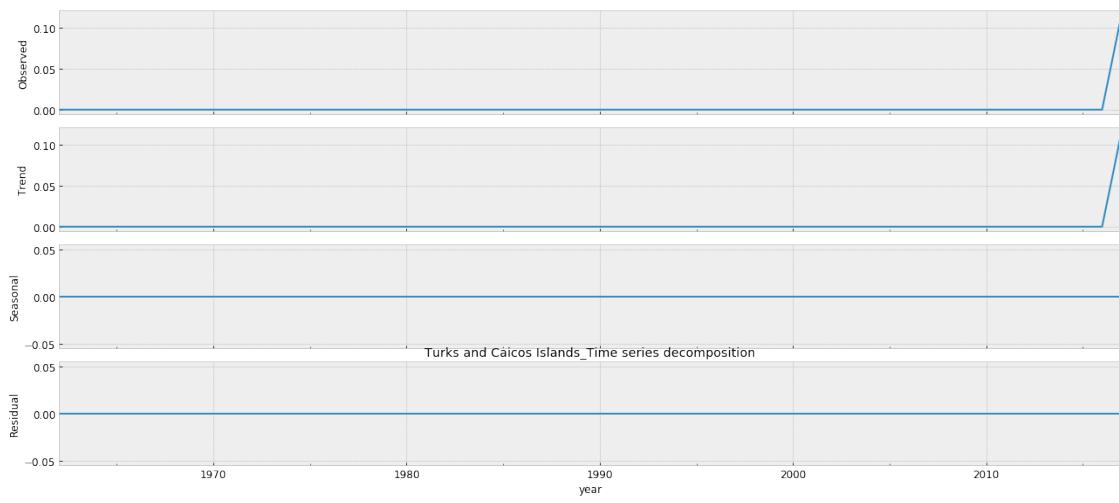
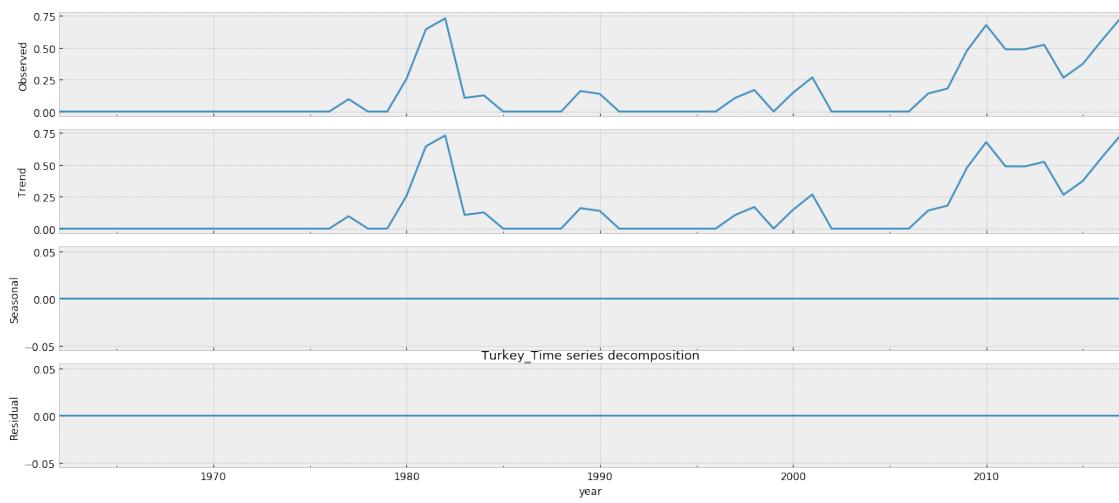


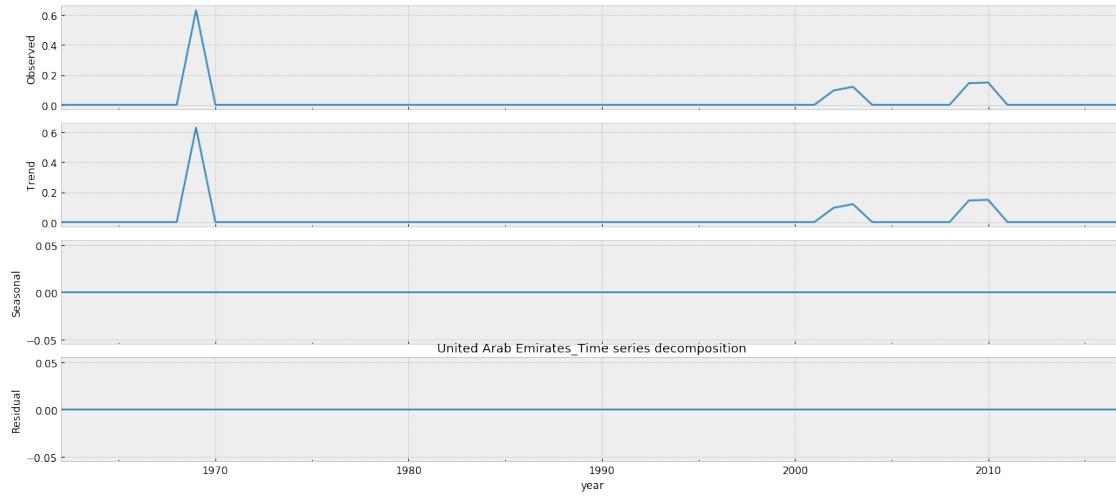
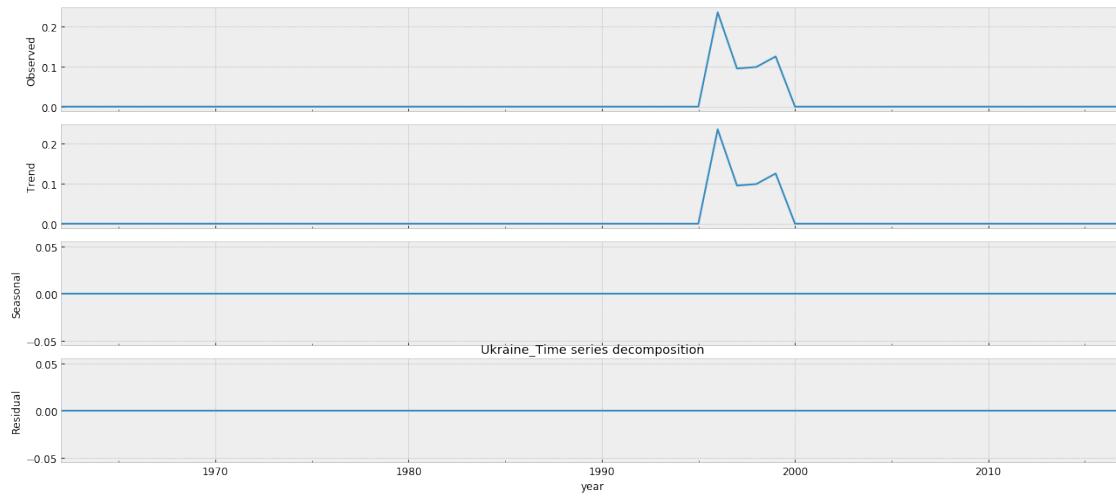
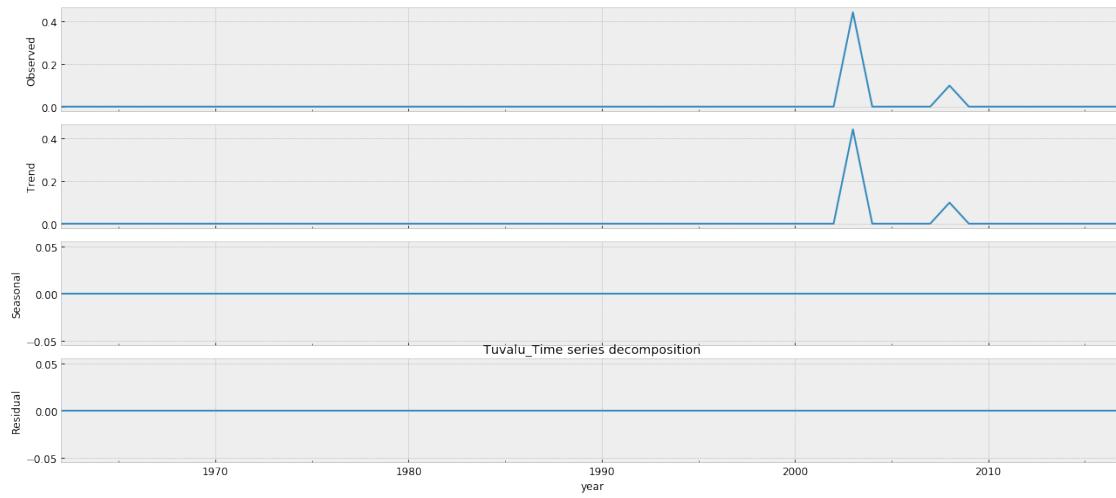


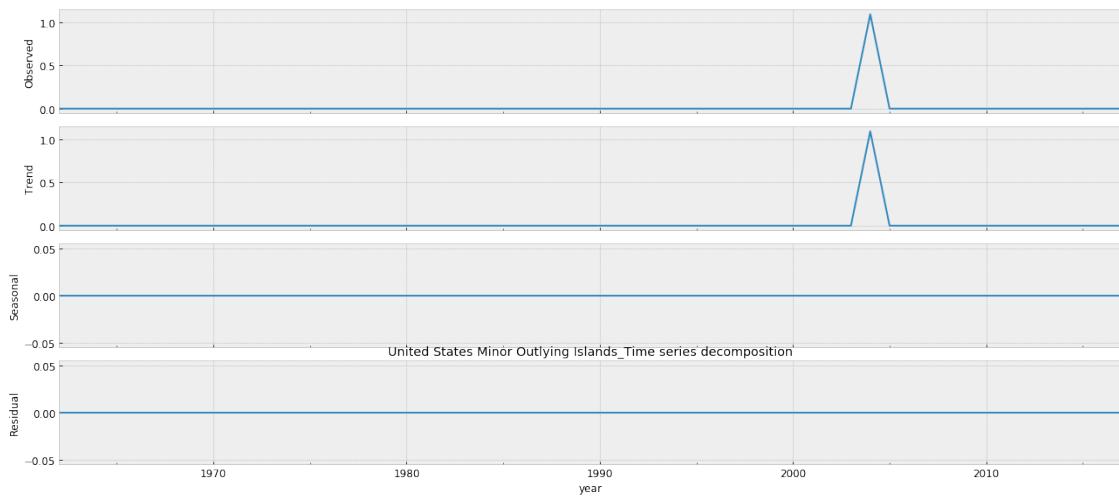
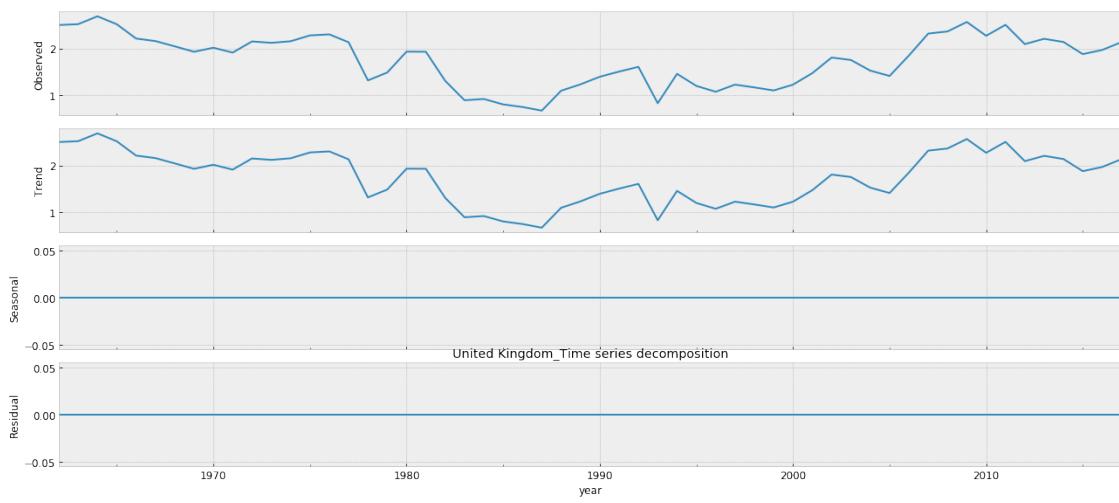


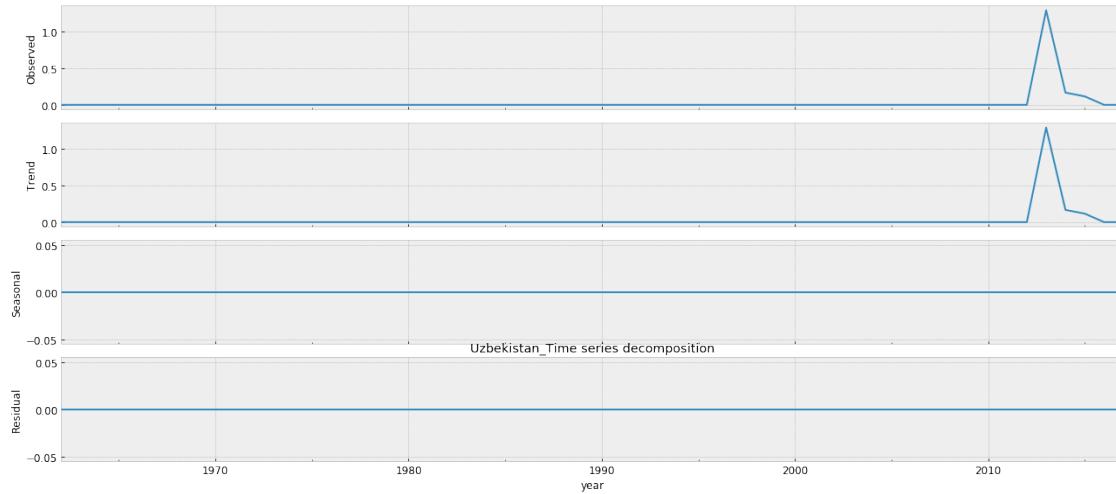
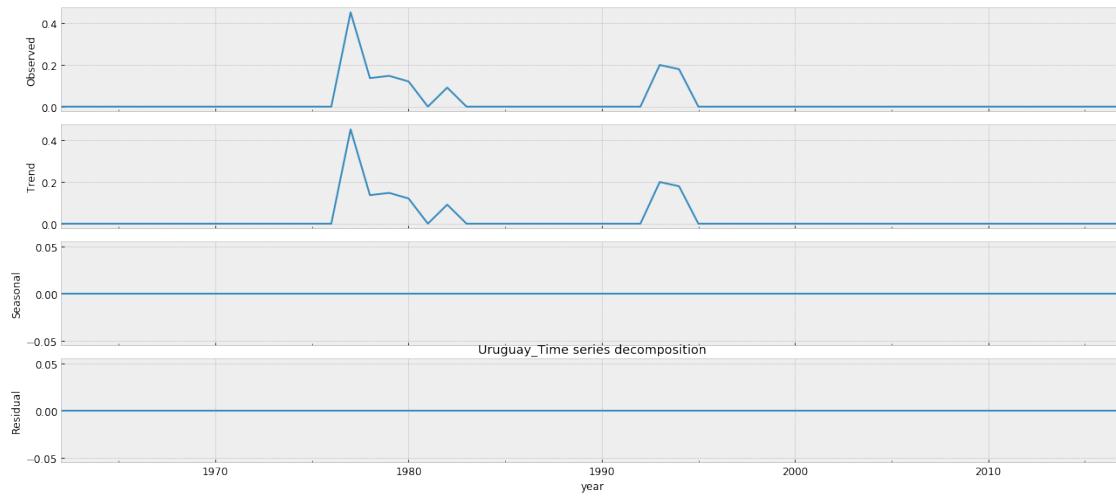
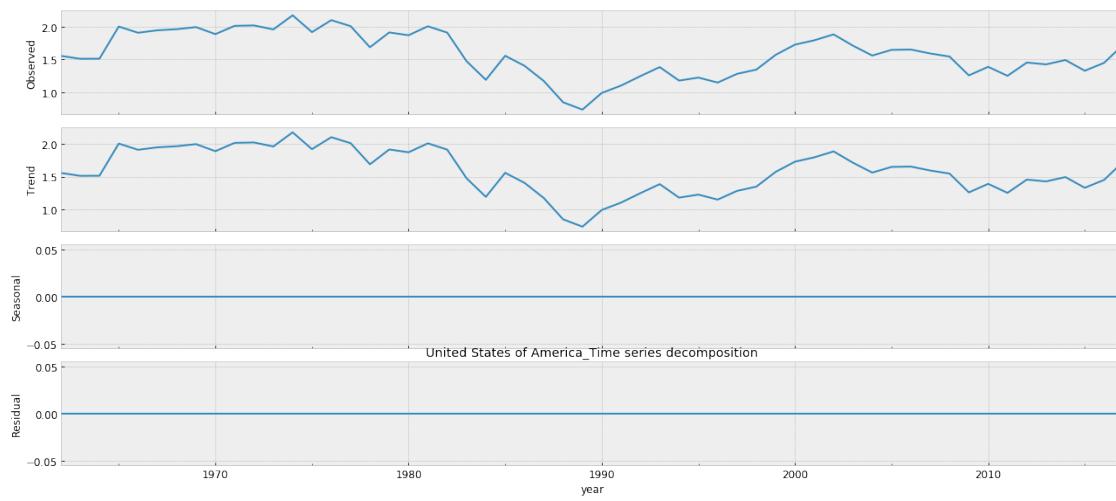


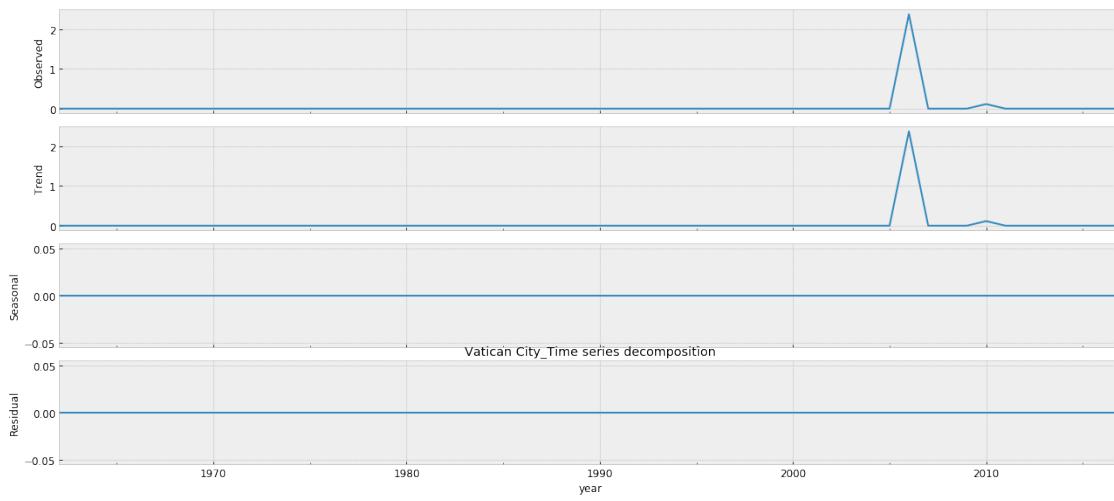
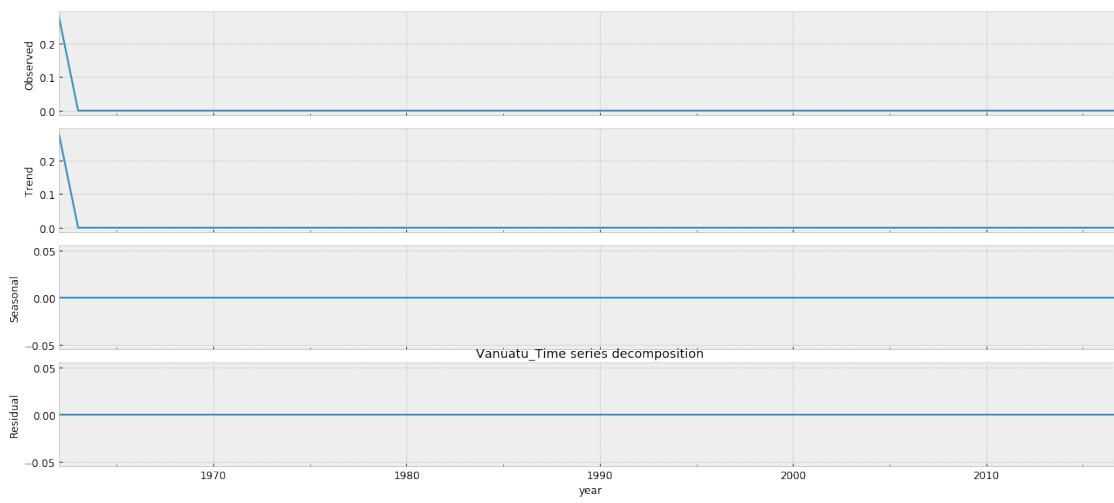


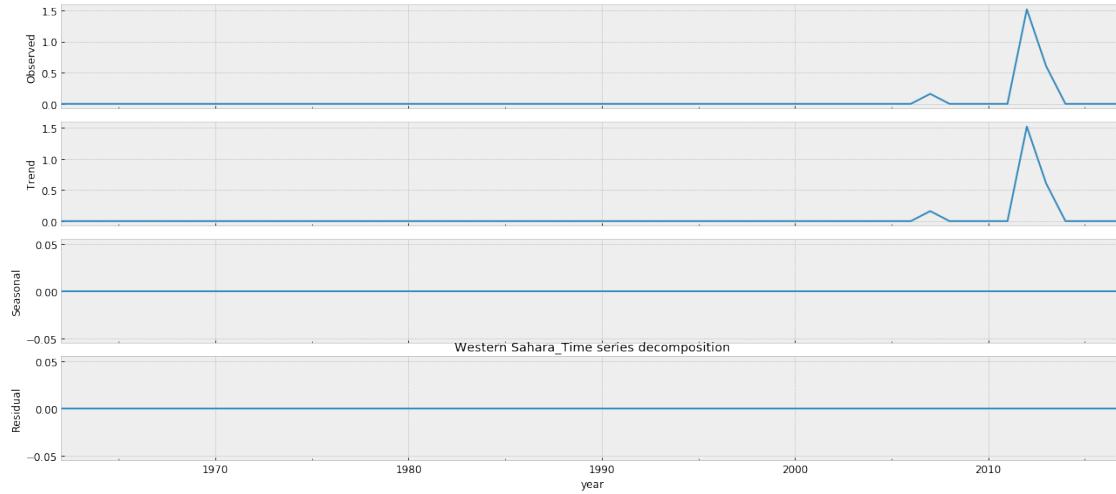
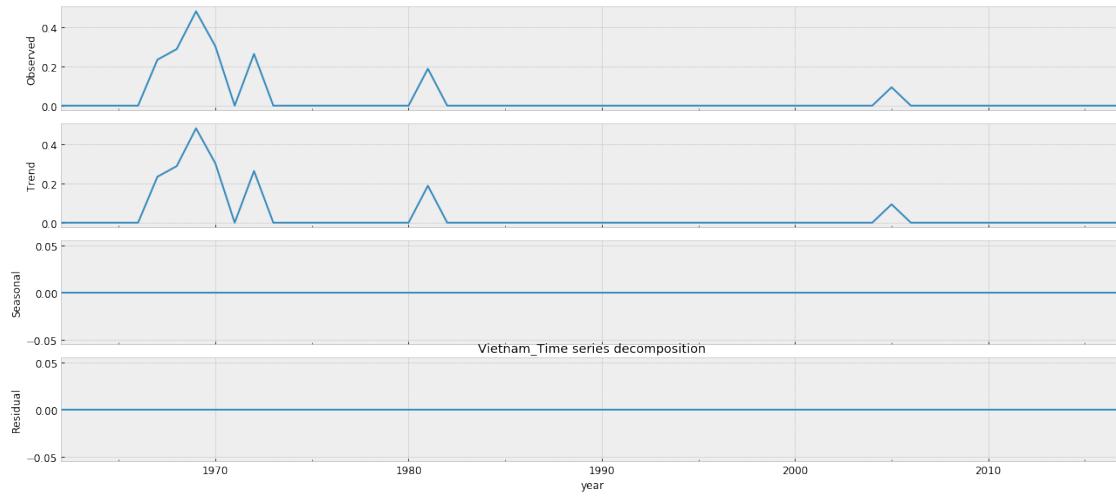
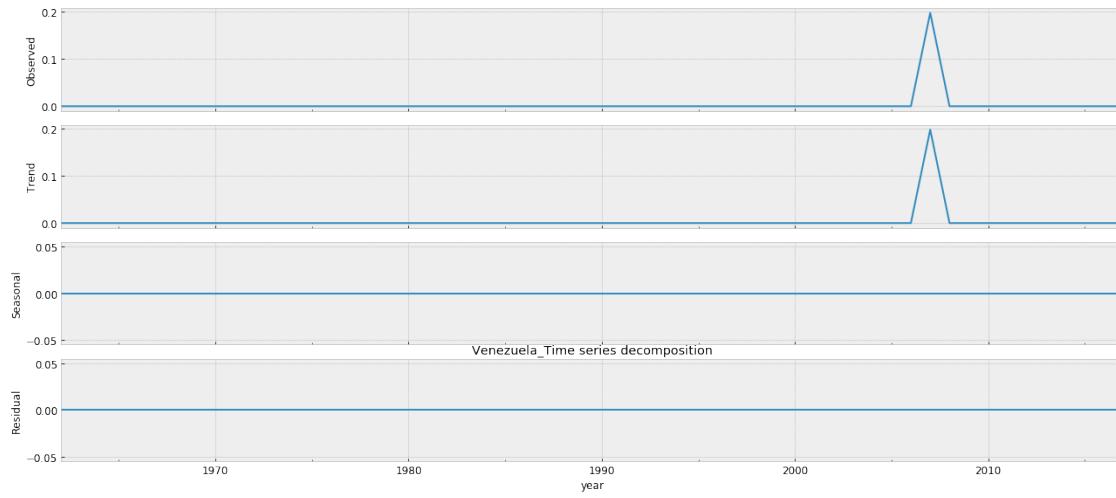


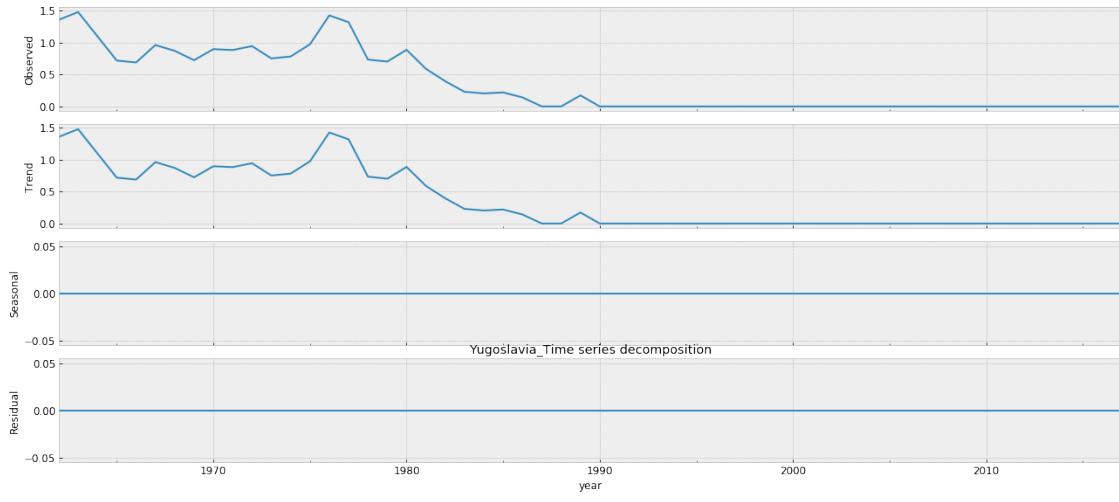
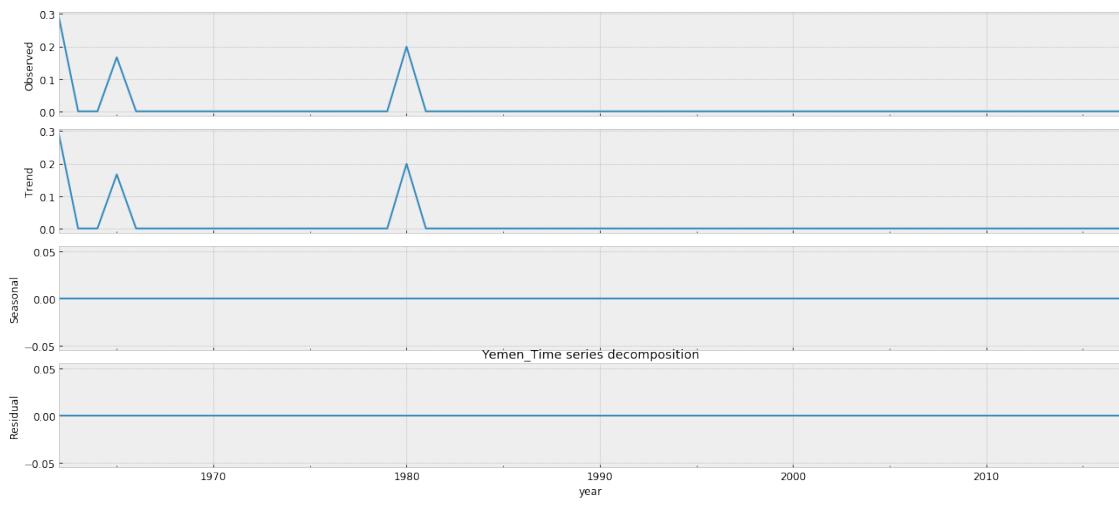


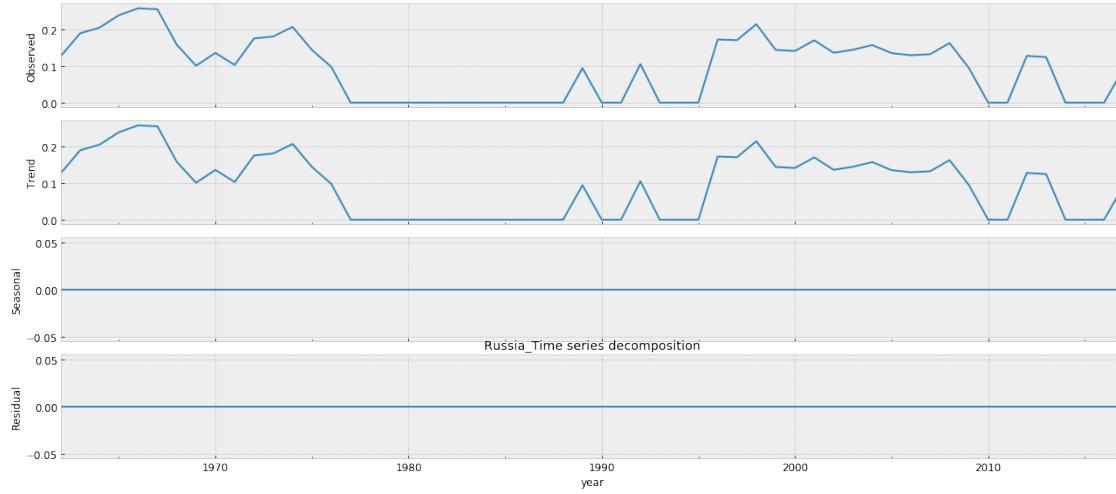
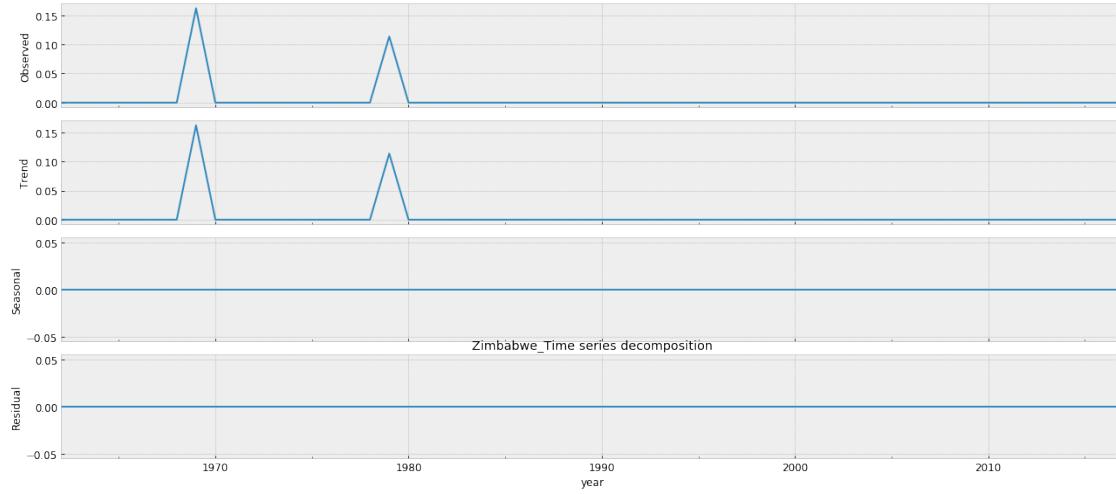












3.1 Stationarity

- Most of the Time Series models work on the assumption that the TS is stationary.
- Major reason for this is that there are many ways in which a series can be non-stationary, but only one way for stationarity.
- Intuitively, we can say that if a Time Series has a particular behaviour over time, there is a very high probability that it will follow the same behaviour in the future.
- Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

Notes from Alkaline ML

- tests of stationarity for testing a null hypothesis that an observable univariate time series is stationary around a deterministic trend (i.e. trend-stationary).

- A time series is **stationary** when its mean, variance and auto-correlation, etc., are **constant over time**.
- Many time-series methods may perform better when a time-series is stationary, since forecasting values becomes a far easier task for a stationary time series (high probability of behaving the same way)
- ARIMAs that include differencing (i.e., $d > 0$) assume that the data becomes stationary after differencing.
- This is called **difference-stationary**
- Auto-correlation plots are an easy way to determine whether your time series is sufficiently stationary for modeling.
- If the plot does not appear relatively stationary, your model will likely need a differencing term.
- These can be determined by using an Augmented Dickey-Fuller test, or various other statistical testing methods.
- Note that `auto_arima` will automatically determine the appropriate differencing term for you by default.

In order to quantitatively determine whether we need to difference our data in order to make it stationary, we can conduct a test of stationarity

We can check stationarity using the following:

1. **ACF and PACF plots:**
 - If the time series is stationary, the ACF/PACF plots will show a **quick drop-off in correlation** after a small amount of lag between points.
2. **Plotting Rolling Statistics:**
 - We can plot the moving average or moving variance and see if it varies with time.
 - **Moving average/variance** is for any instant ‘t’, the average/variance of the last period (e.g year or last 12 months)
3. **Augmented Dickey-Fuller Test [*]:**
 - This is one of the statistical tests for checking stationarity.
 - Here the **null hypothesis is that the TS is non-stationary**.
 - The null hypothesis of the Augmented Dickey-Fuller is that there is a unit root, with the alternative that there is no unit root.
 - If the pvalue is above a critical size, then we cannot reject that there is a unit root.
 - The p-values are obtained through regression surface approximation from MacKinnon 1994, but using the updated 2010 tables
 - If the p-value is close to significant, then the critical values should be used to judge whether to reject the null.
 - The autolag option and maxlag for it are described in Greene
 - The test results comprise of a Test Statistic and some Critical Values for different confidence levels.
 - If the ‘Test Statistic’ is less than the ‘Critical Value’, we can reject the null hypothesis and say that the series is stationary.

References:

1. W. Green. “Econometric Analysis,” 5th ed., Pearson, 2003.
2. Hamilton, J.D. “Time Series Analysis”. Princeton, 1994.

3. MacKinnon, J.G. 1994. "Approximate asymptotic distribution functions for unit-root and cointegration tests. *Journal of Business and Economic Statistics* 12, 167-76.
4. MacKinnon, J.G. 2010. "Critical Values for Cointegration Tests." Queen's University, Dept of Economics, Working Papers. Available at <http://ideas.repec.org/p/qed/wpaper/1227.html>

3.1.1 Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots

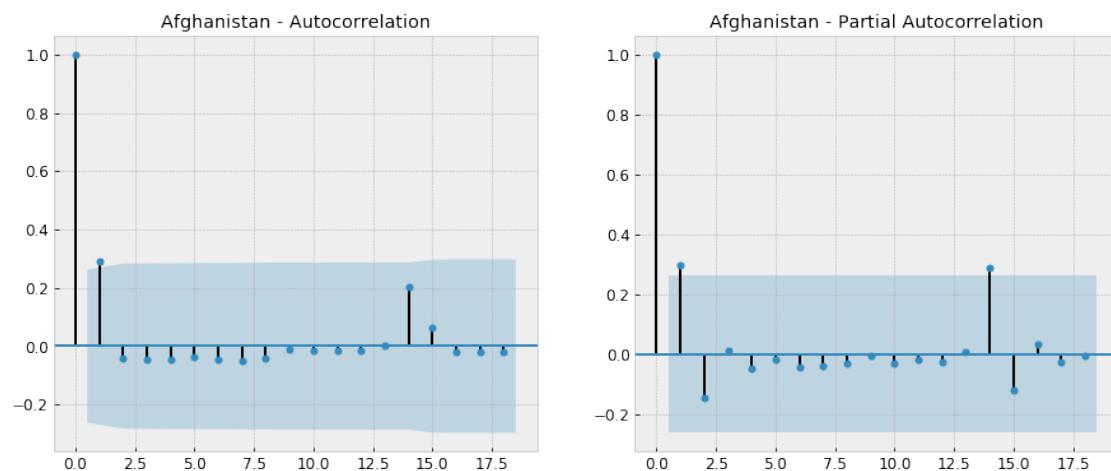
- If the time series is stationary, the ACF/PACF plots will show a **quick drop-off in correlation** after a small amount of lag between points.
- Most of this data is stationary as a high number of previous observations are not correlated with future values.
- Confidence intervals are drawn as a cone.
- By default, this is set to a 95% confidence interval, suggesting that correlation values outside of this cone are very likely a correlation and not a statistical fluke.
- The partial autocorrelation at lag k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags.

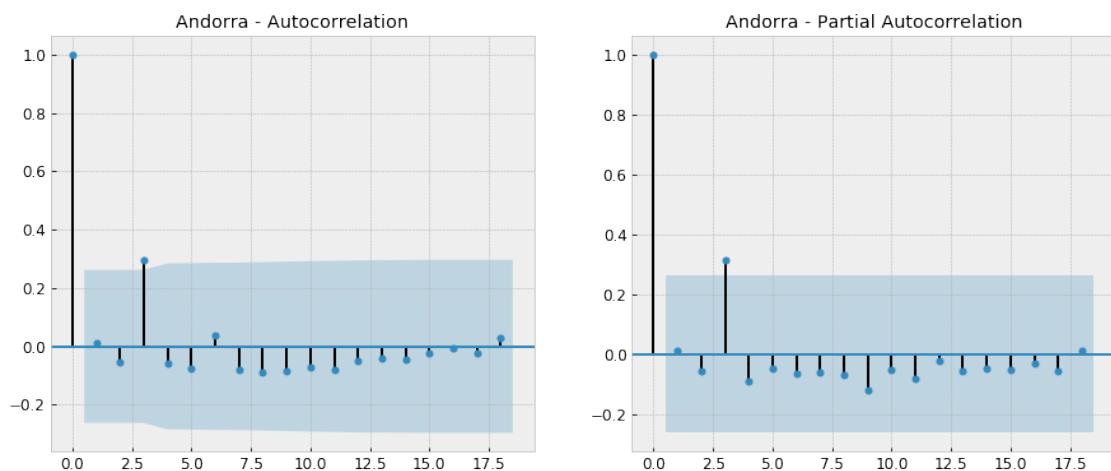
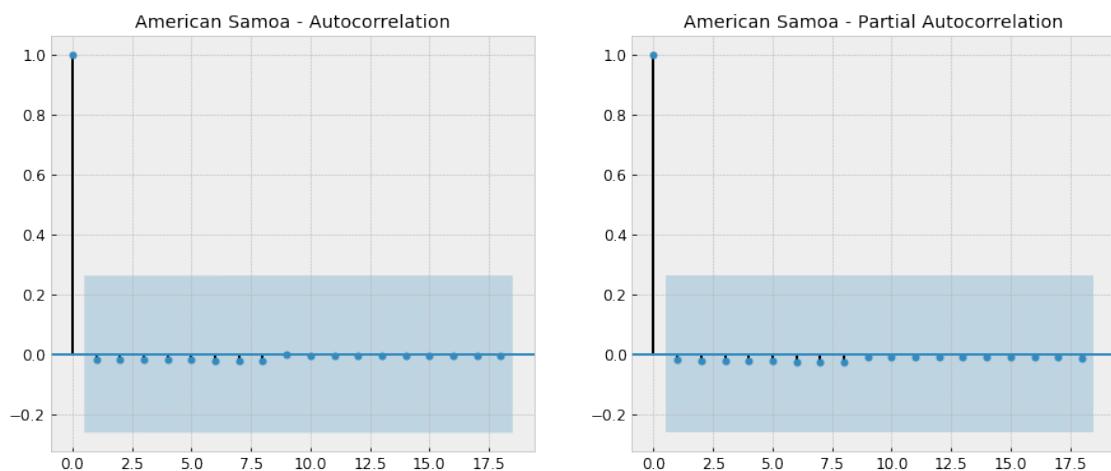
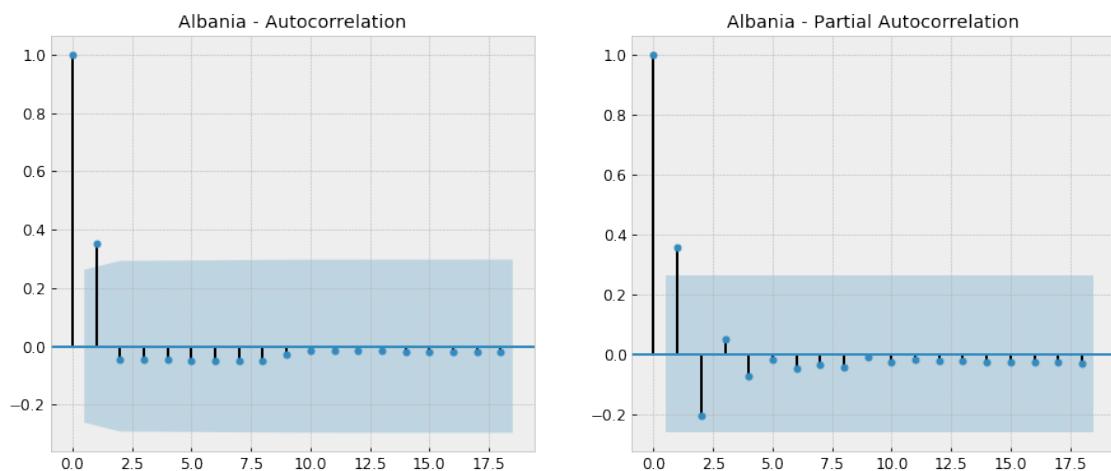
```
[145]: for country in countries:
    savefile = save_images(df_grps, 'acf_pacf')

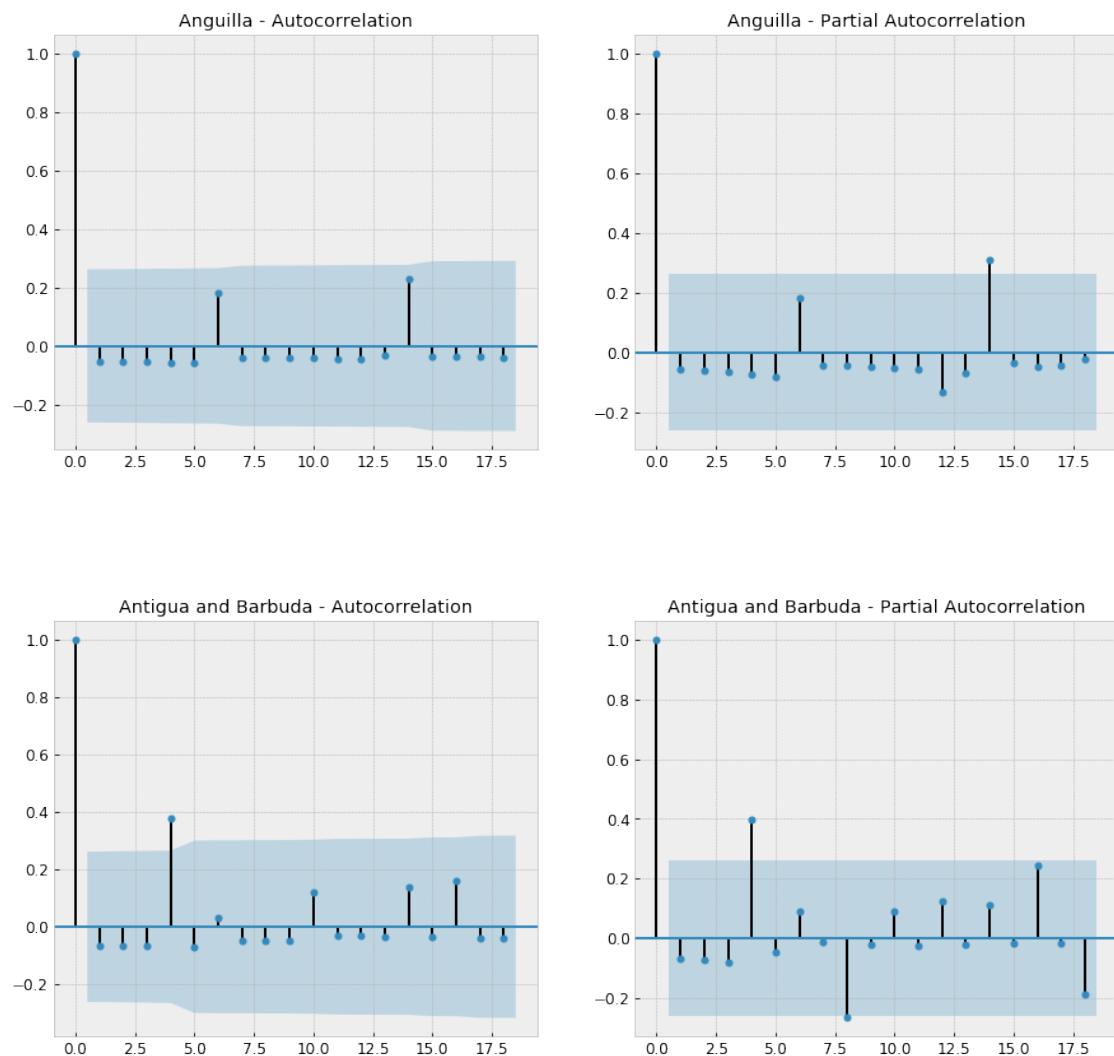
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    plot_acf(df_grps[country], ax=plt.gca(), title=f'{df_grps[country].name} - Autocorrelation')

    plt.subplot(122);
    plot_pacf(df_grps[country], ax=plt.gca(), title=f'{df_grps[country].name} - Partial Autocorrelation')

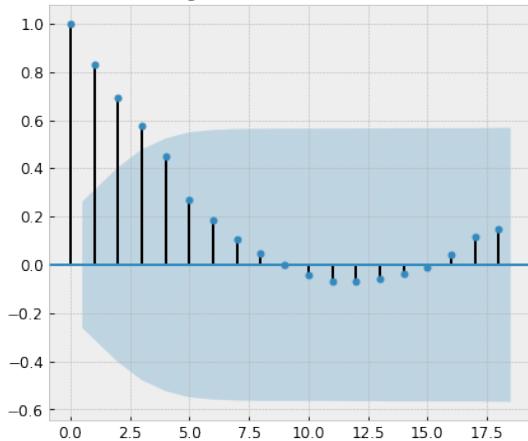
    plt.savefig(str('{}.ACF_PACF'.format(savefile)))
    plt.show()
```



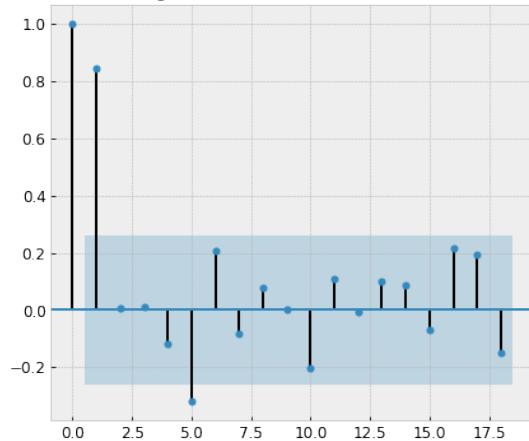




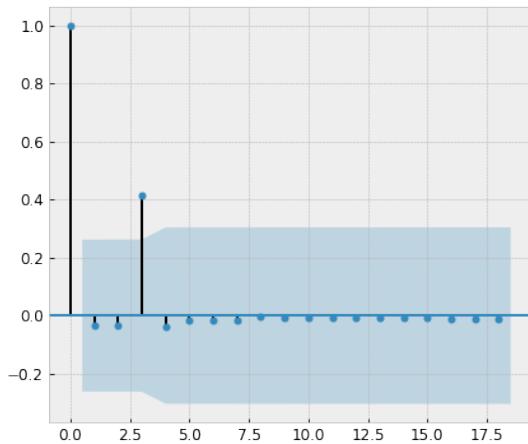
Argentina - Autocorrelation



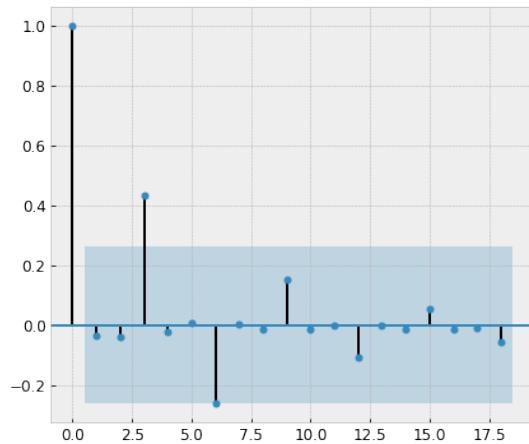
Argentina - Partial Autocorrelation



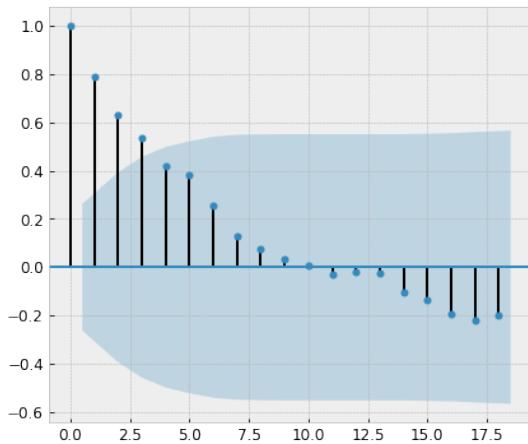
Aruba - Autocorrelation



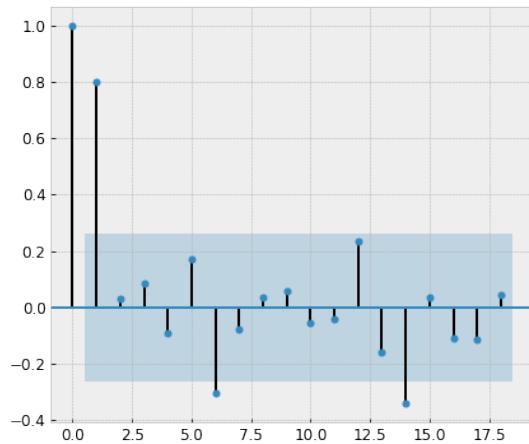
Aruba - Partial Autocorrelation

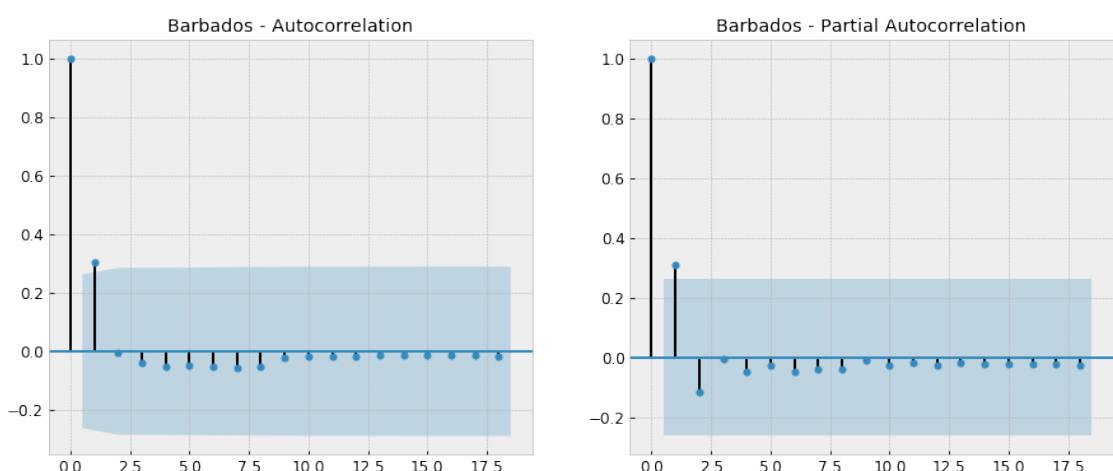
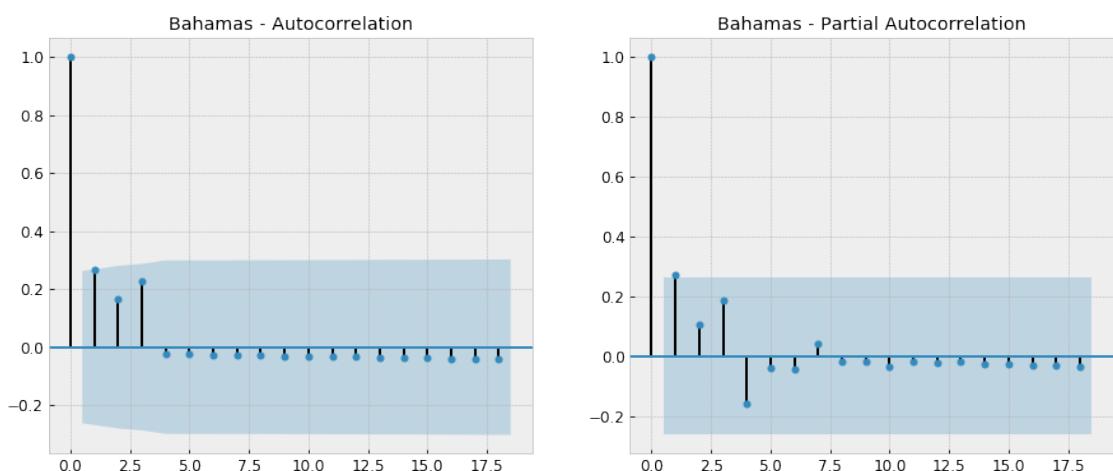
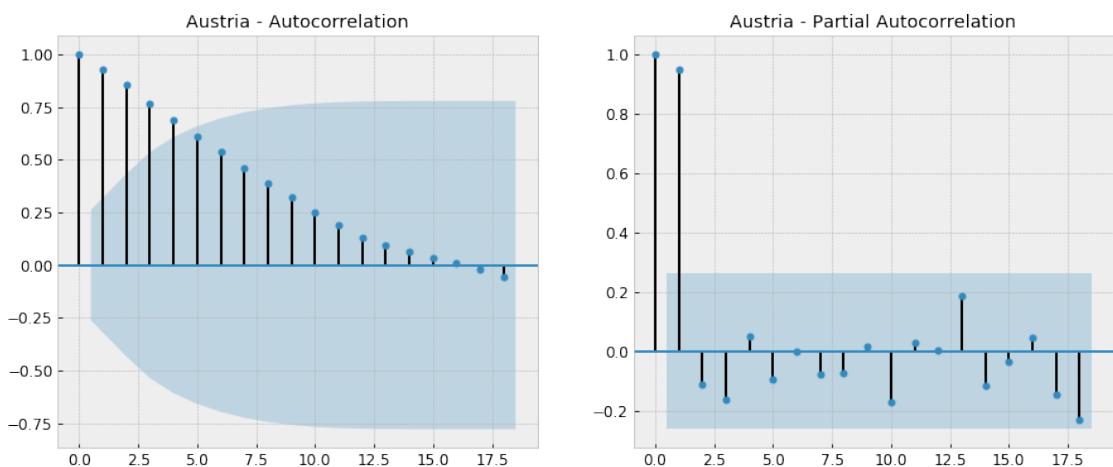


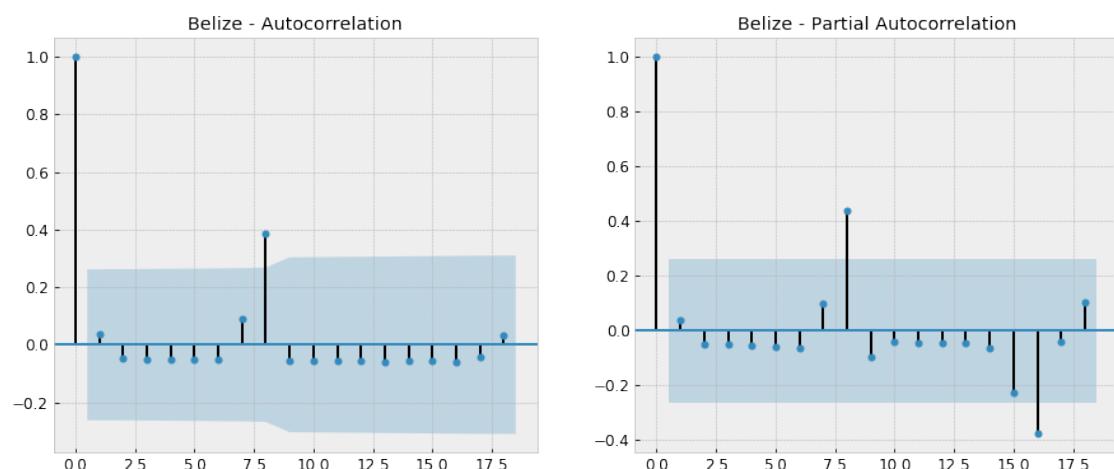
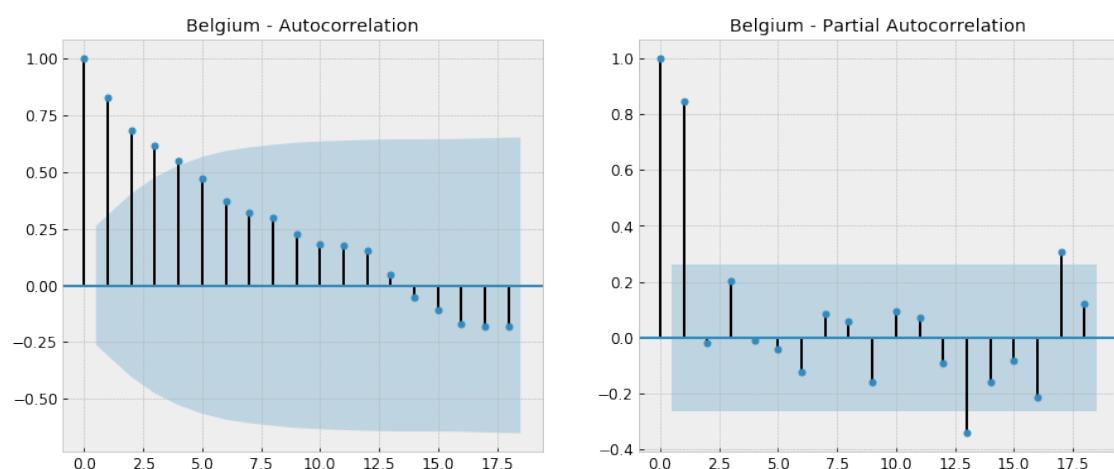
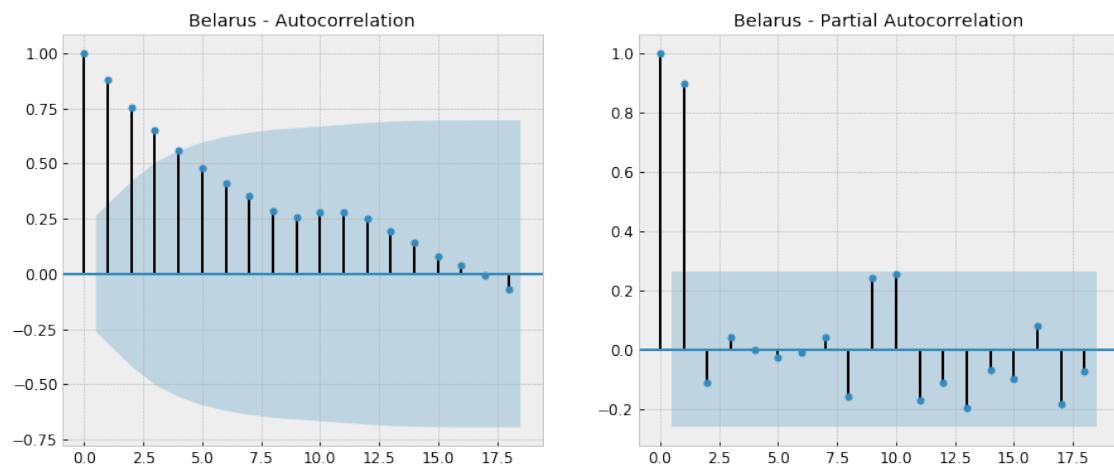
Australia - Autocorrelation

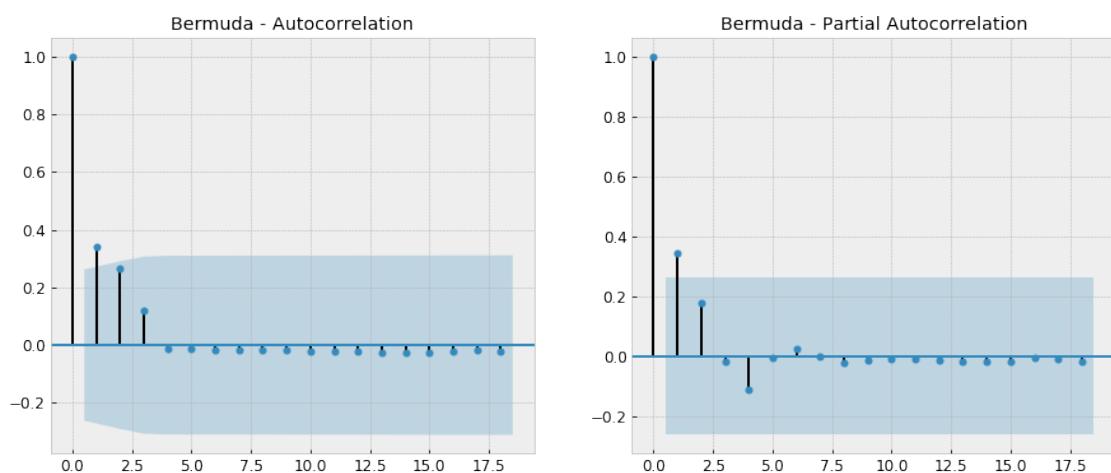
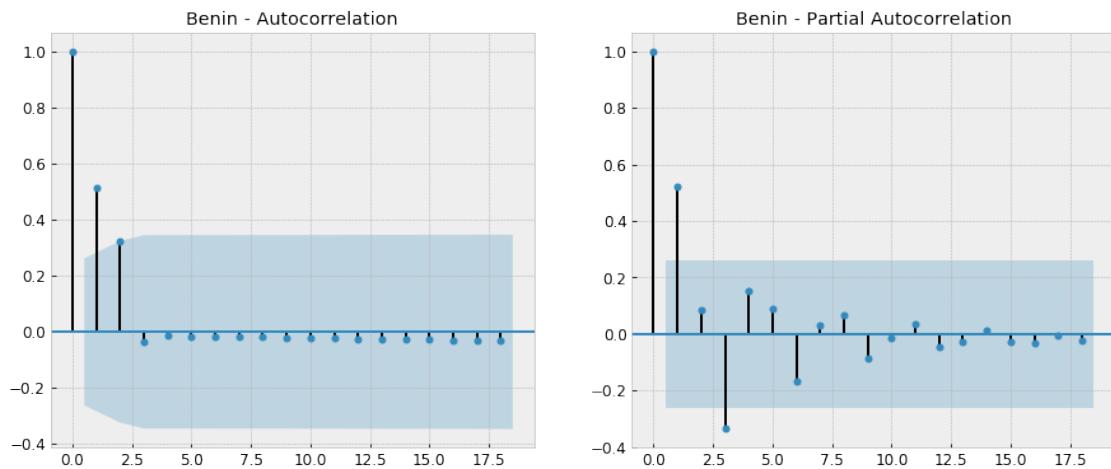


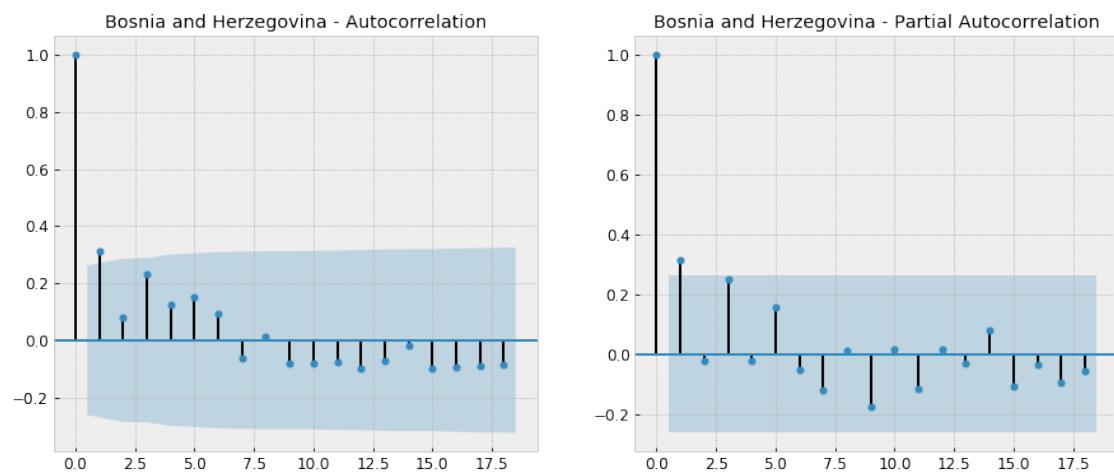
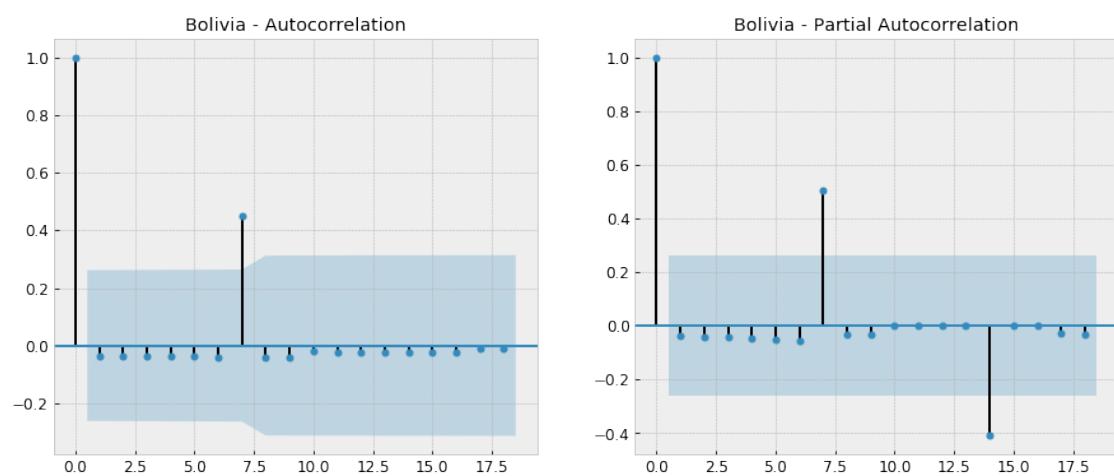
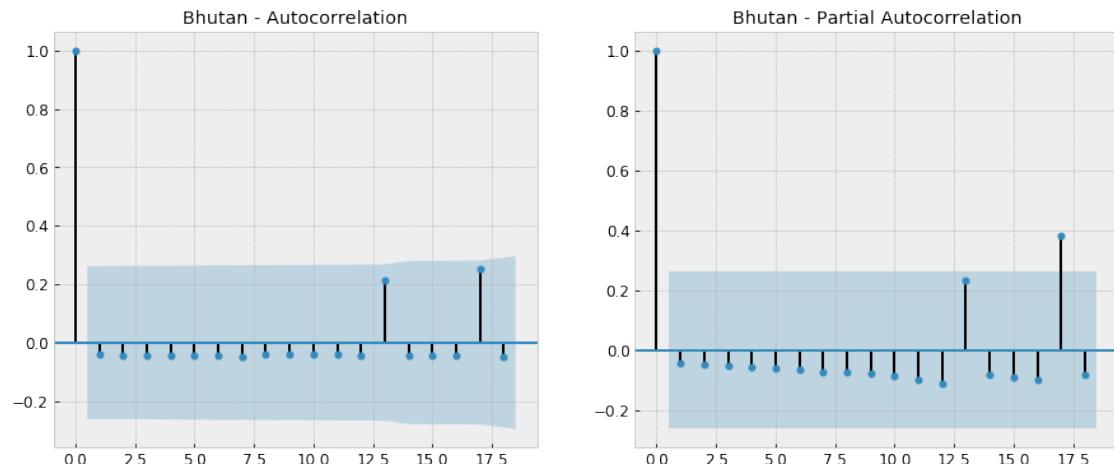
Australia - Partial Autocorrelation

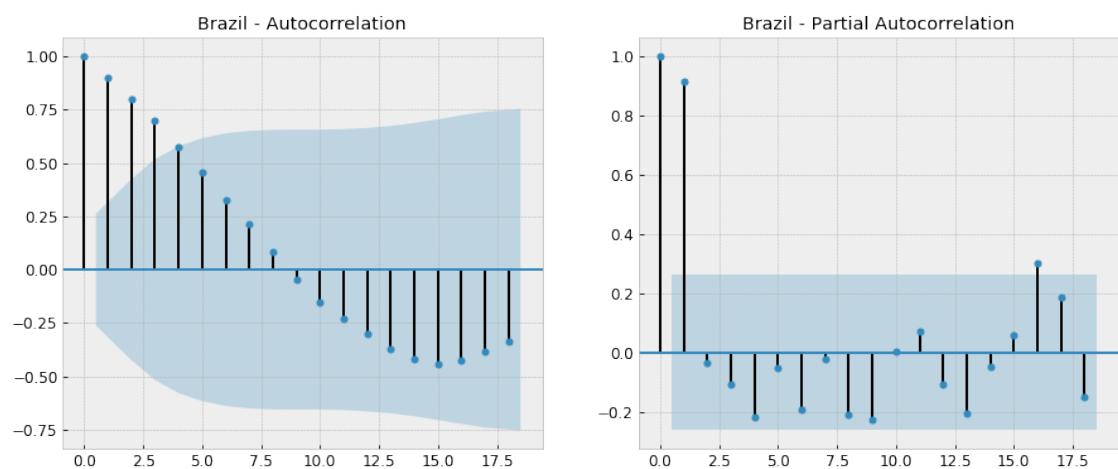
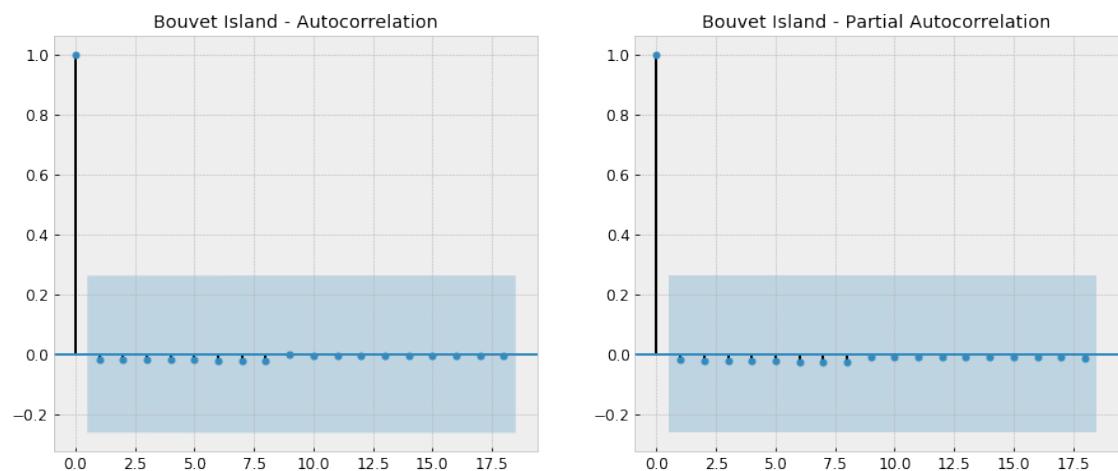
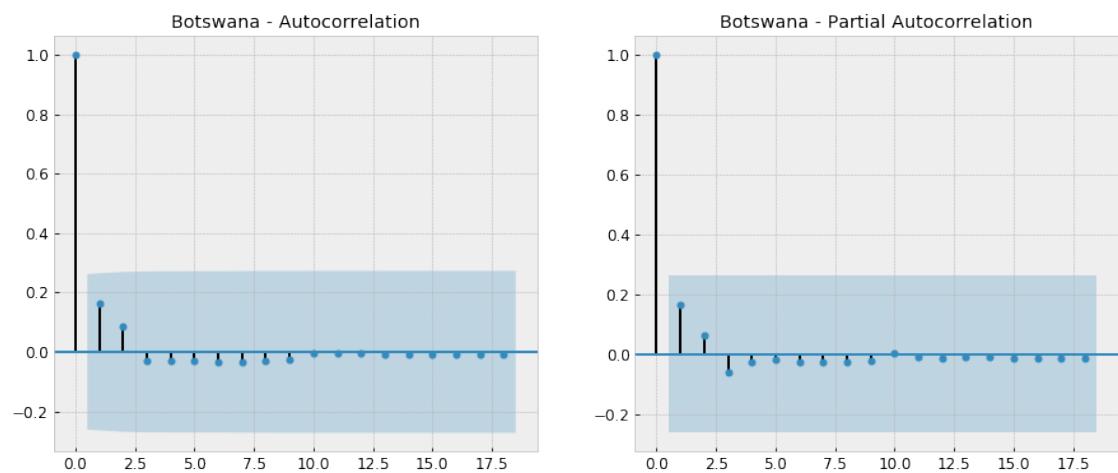


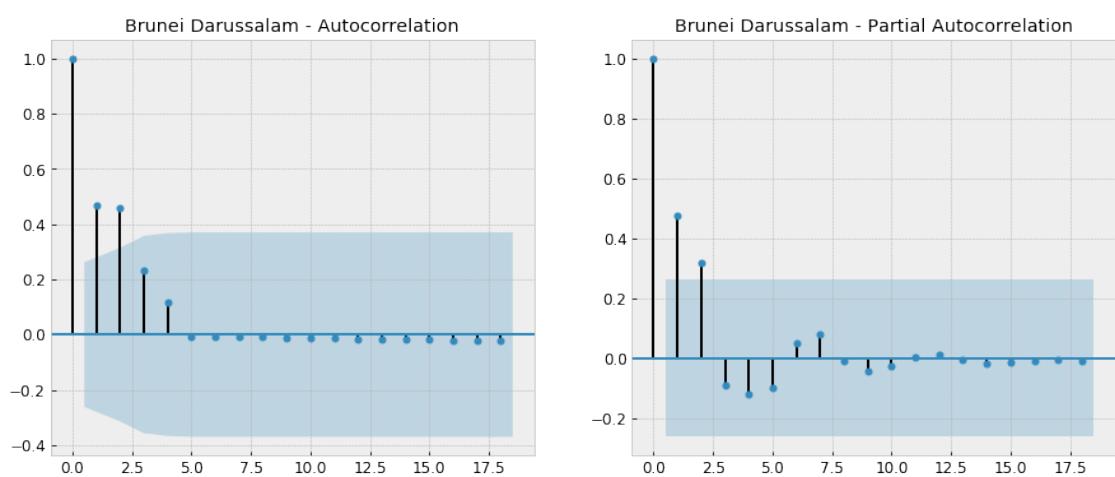
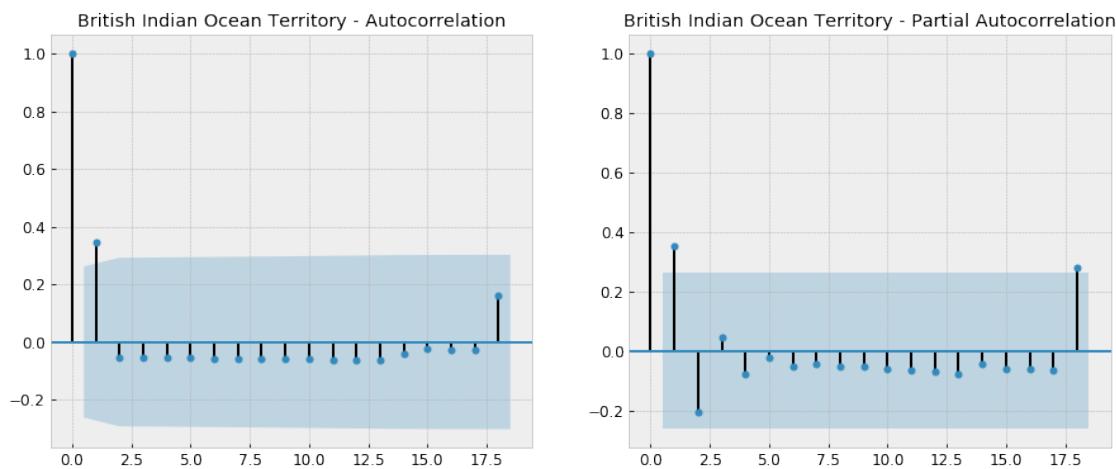


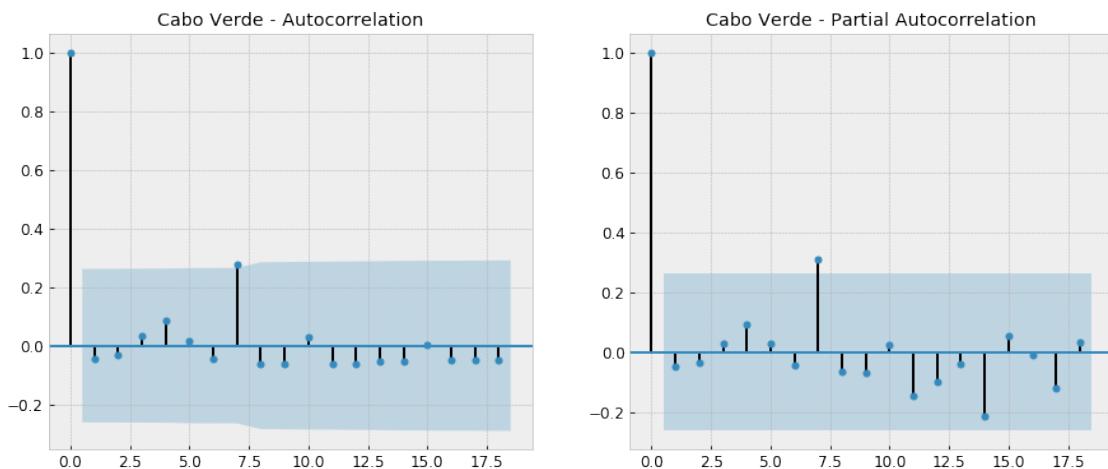
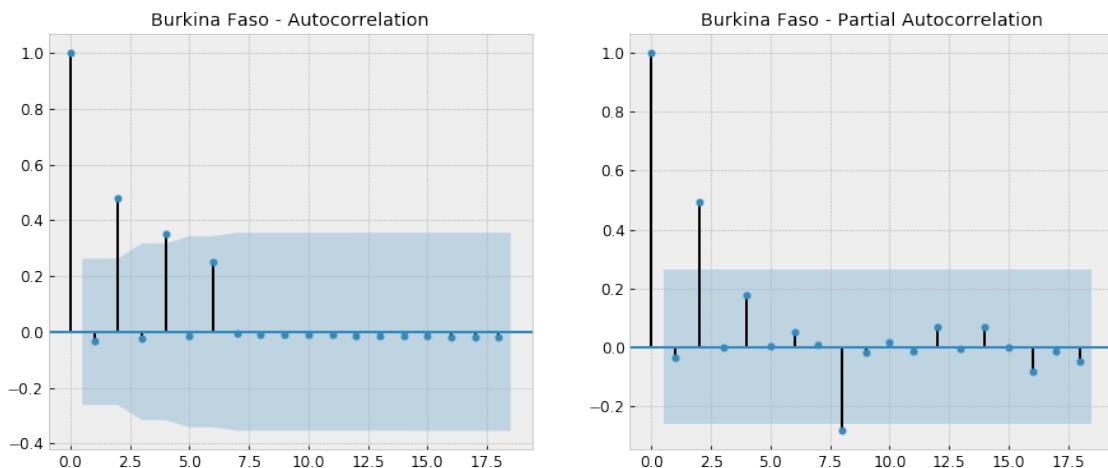
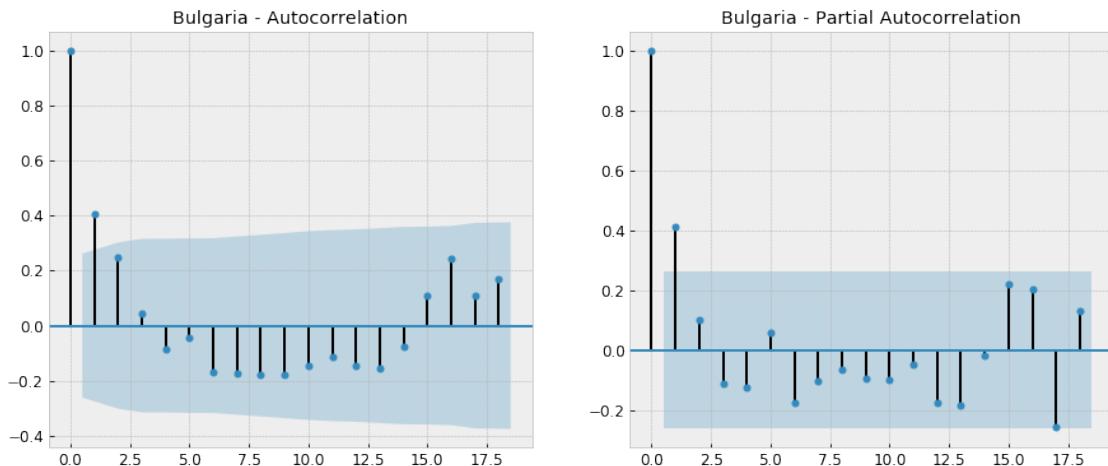


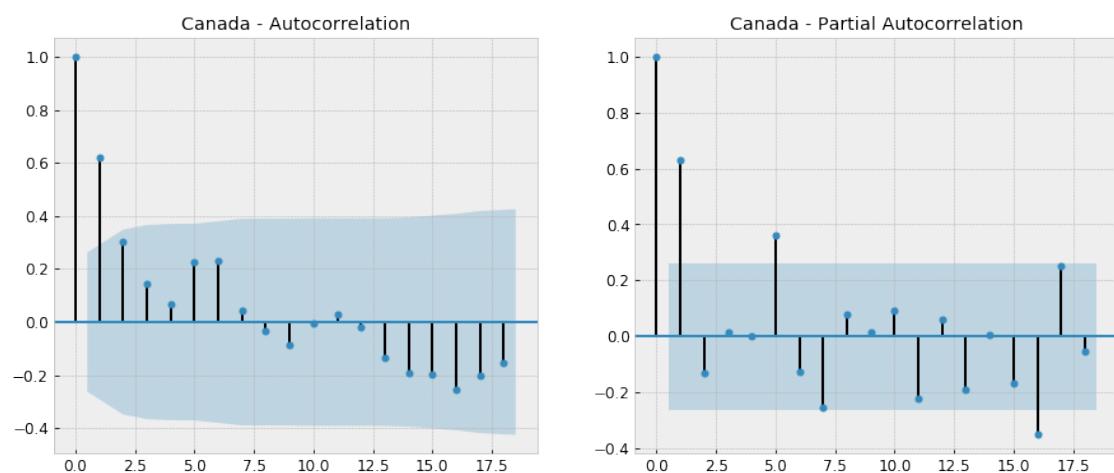
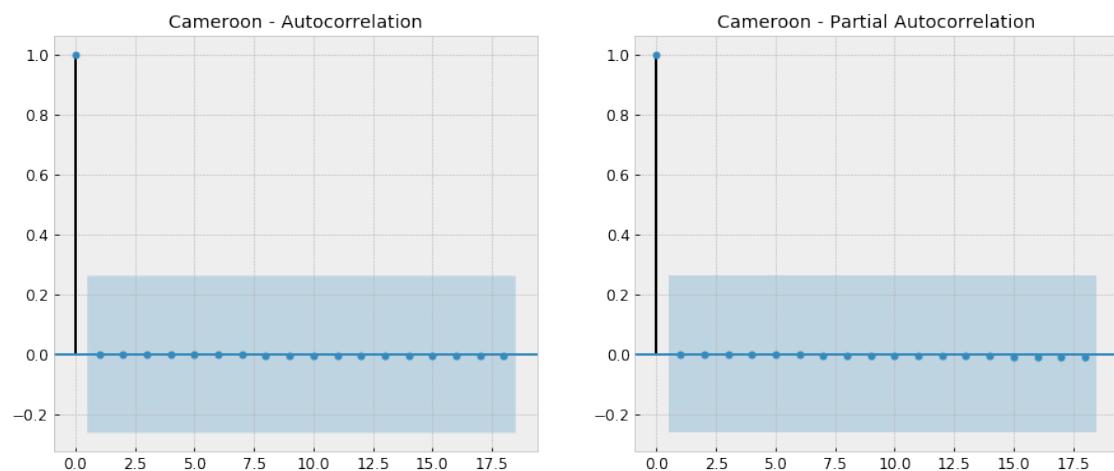
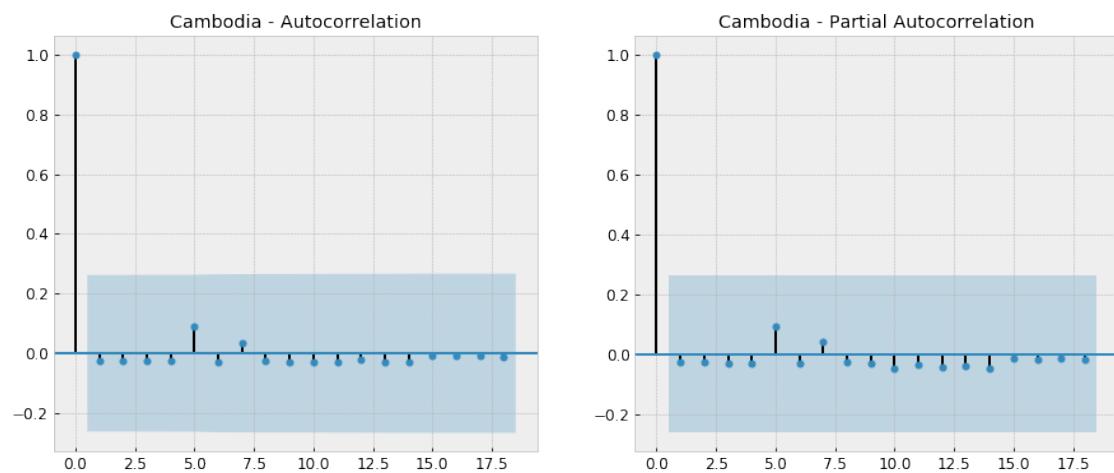


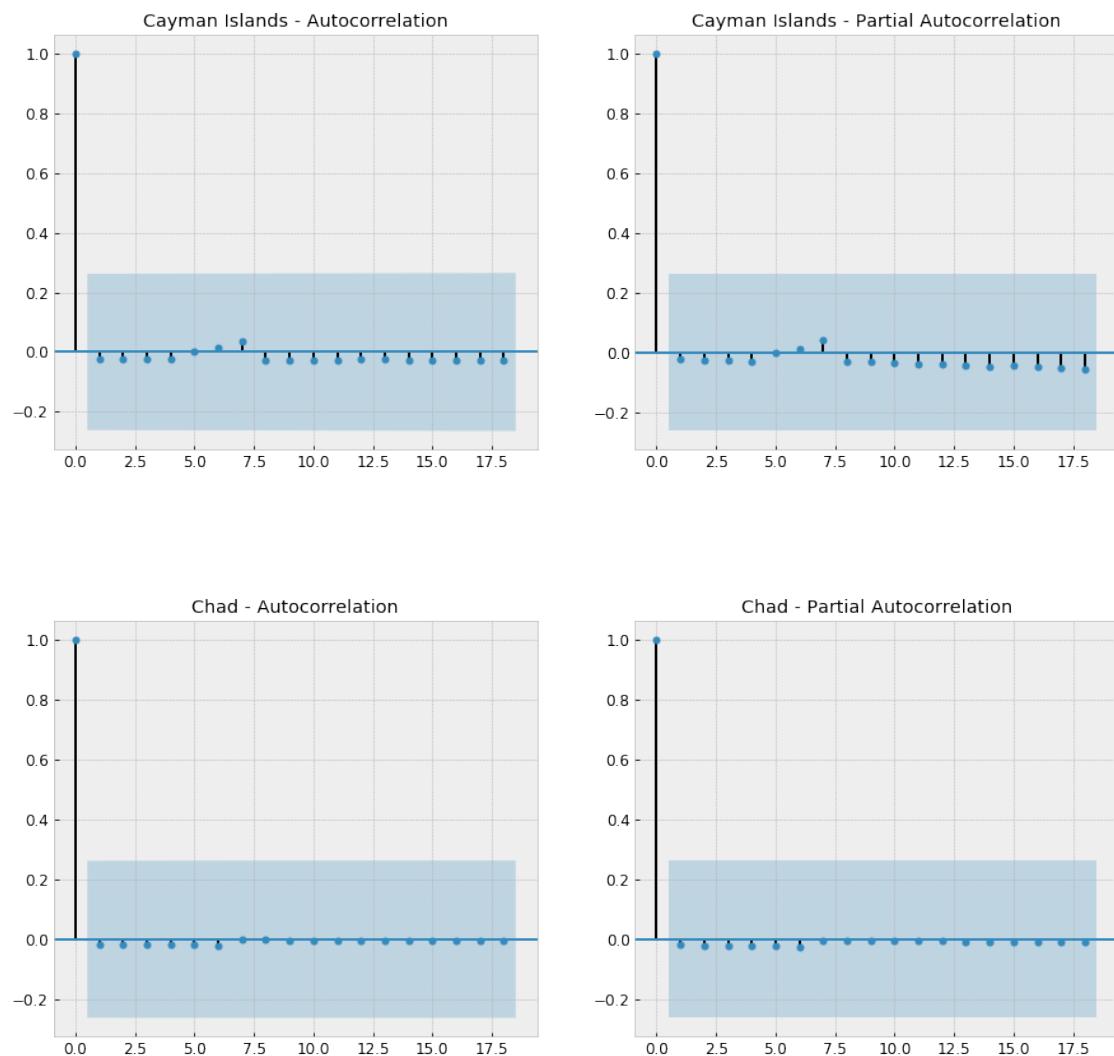




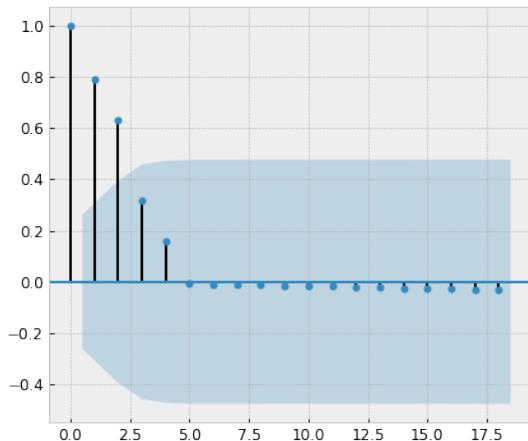




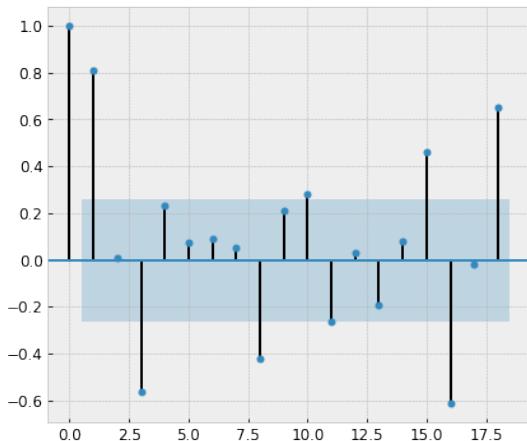




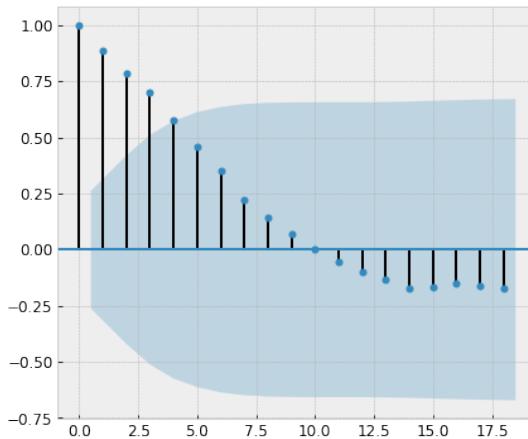
Chile - Autocorrelation



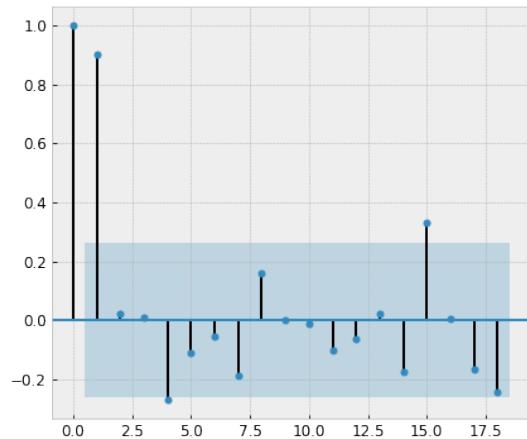
Chile - Partial Autocorrelation



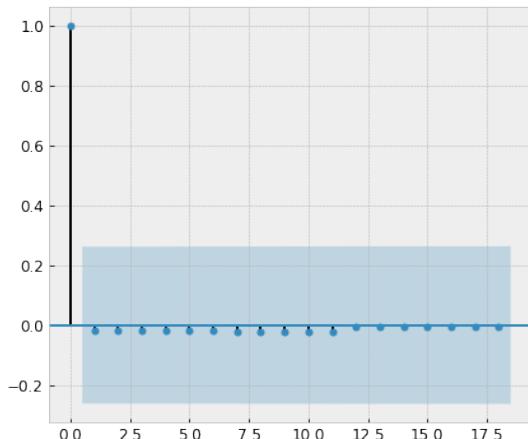
China - Autocorrelation



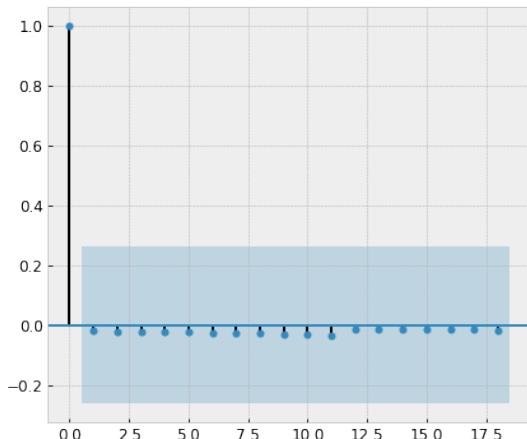
China - Partial Autocorrelation

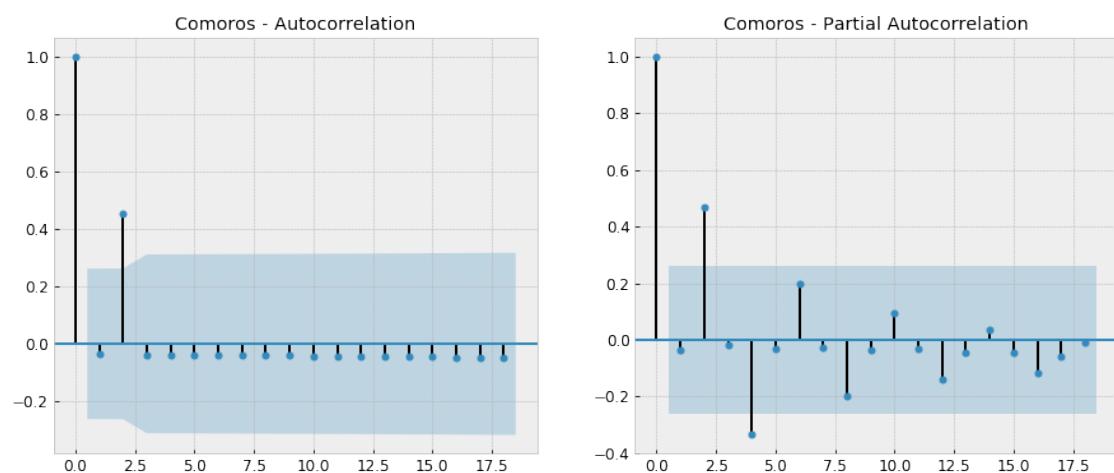
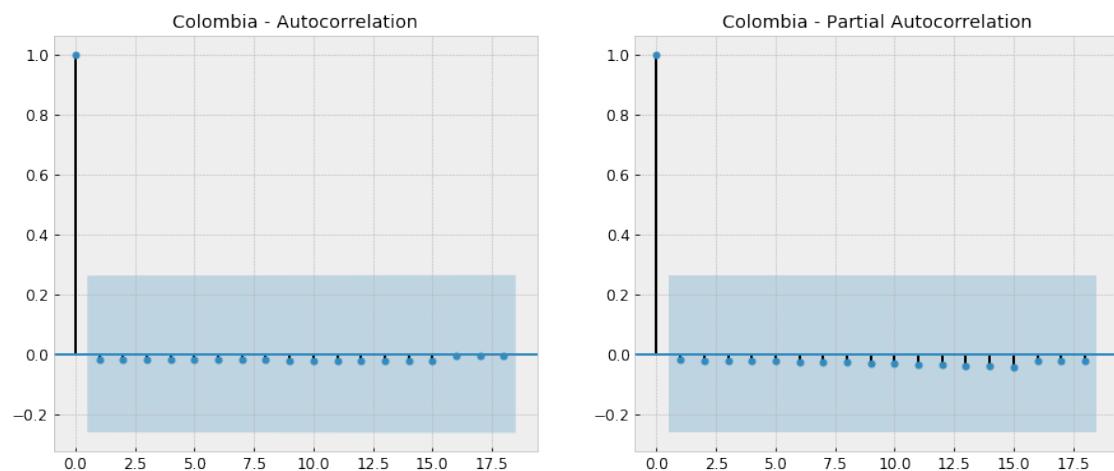
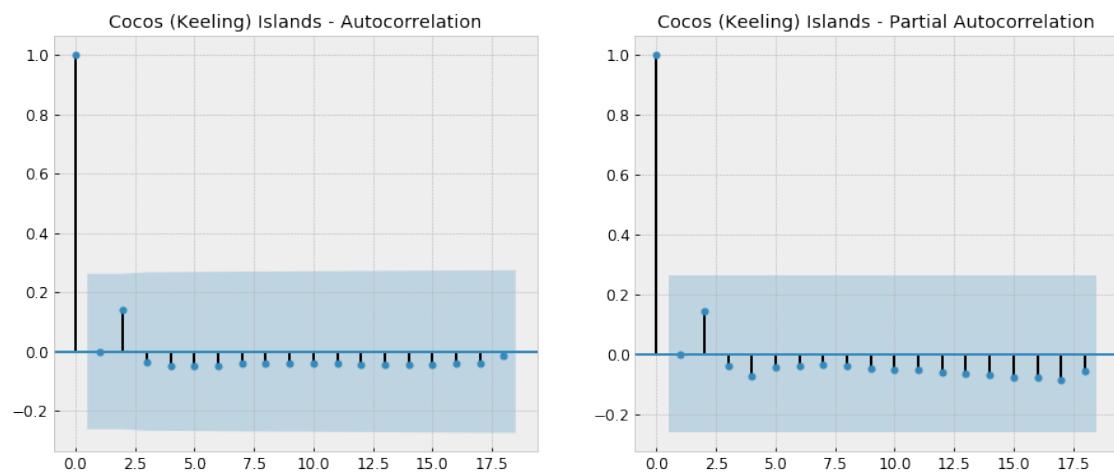


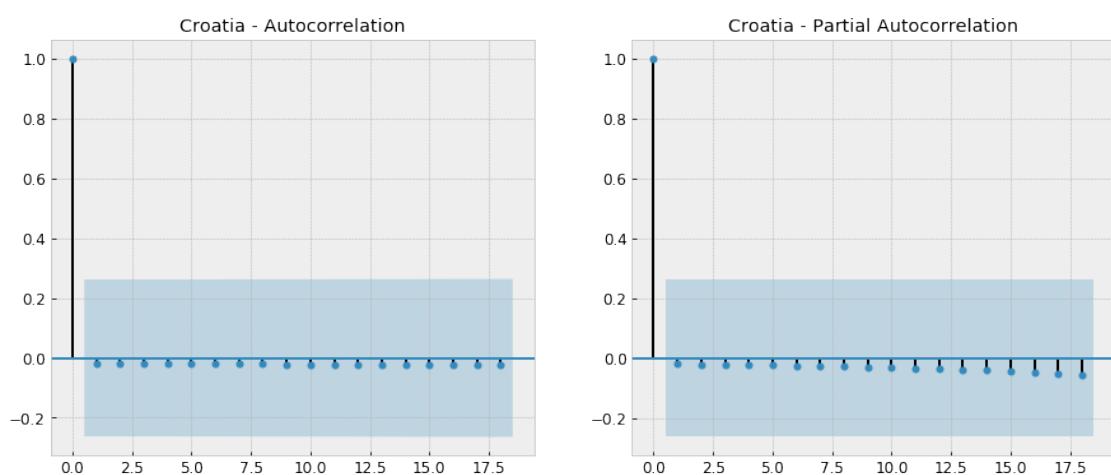
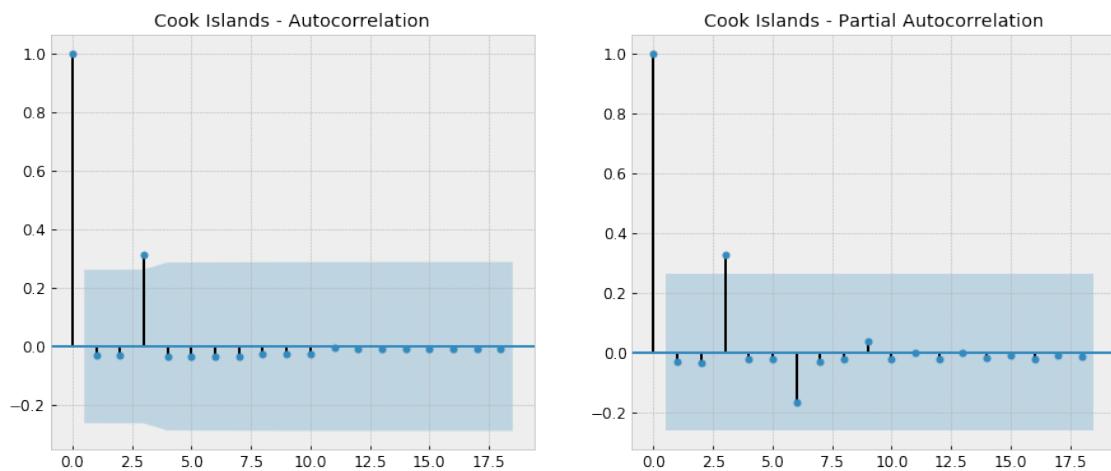
Christmas Island - Autocorrelation



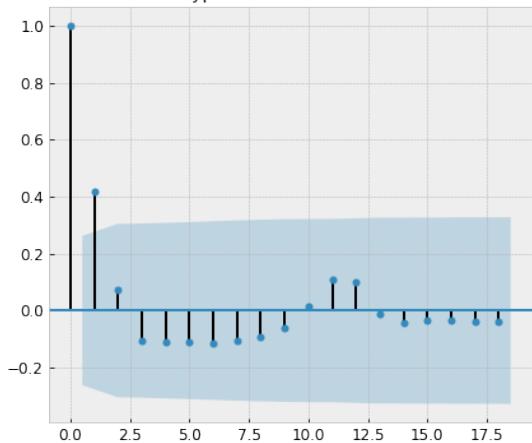
Christmas Island - Partial Autocorrelation



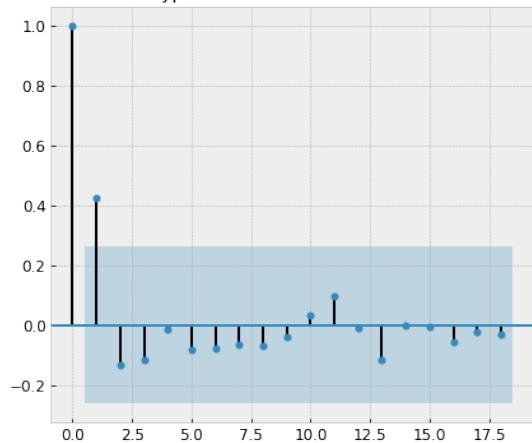




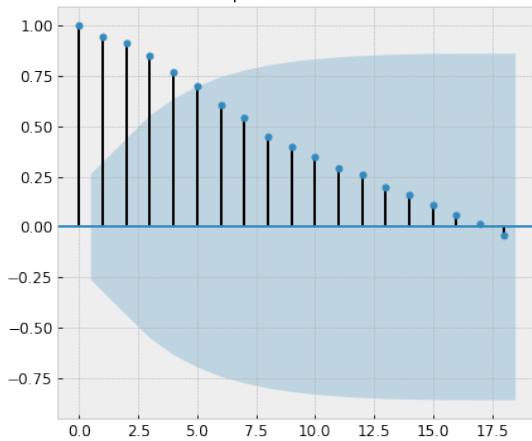
Cyprus - Autocorrelation



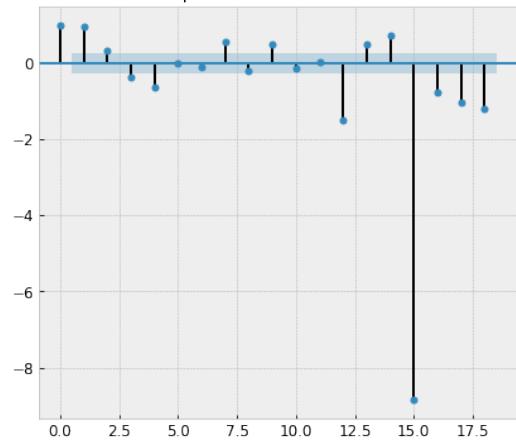
Cyprus - Partial Autocorrelation



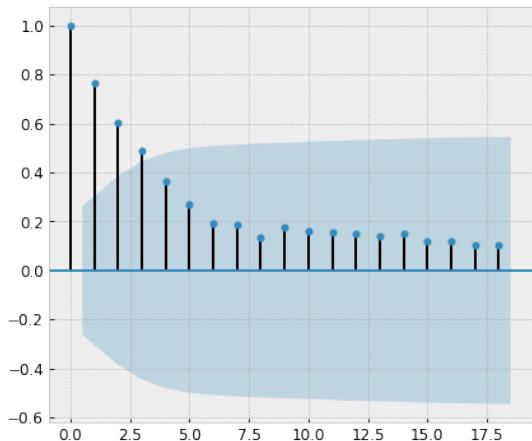
Czech Republic - Autocorrelation



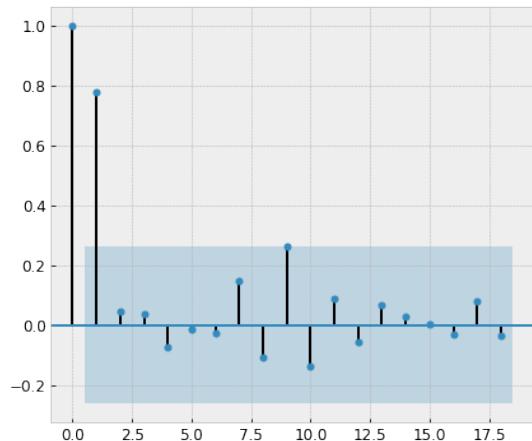
Czech Republic - Partial Autocorrelation

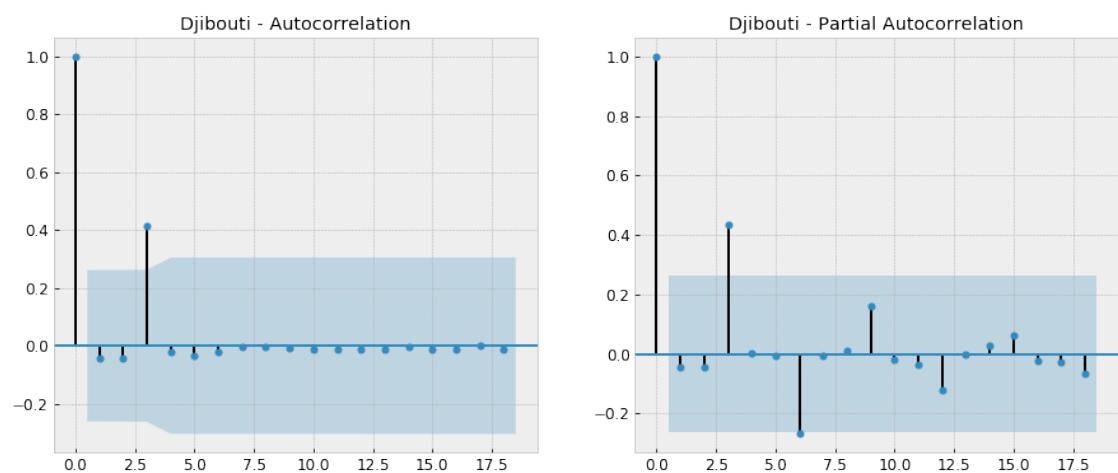
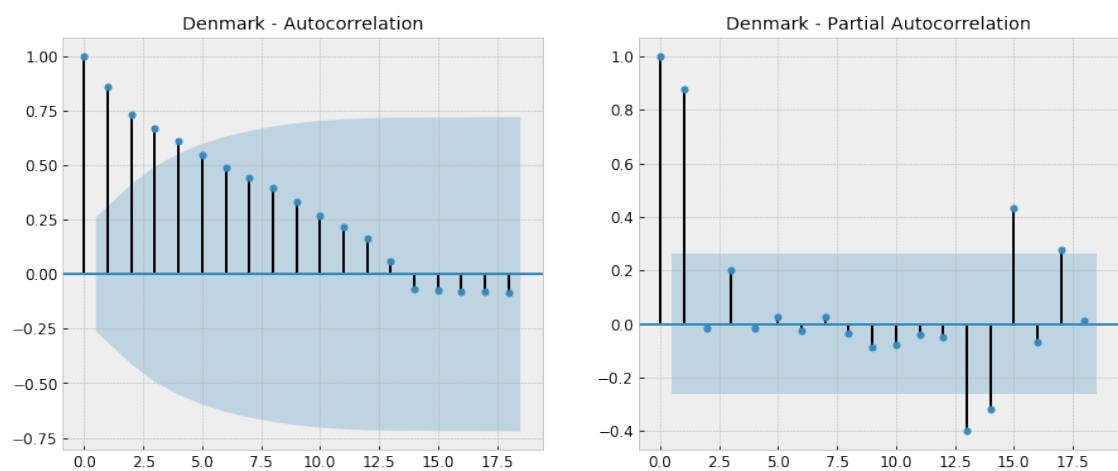
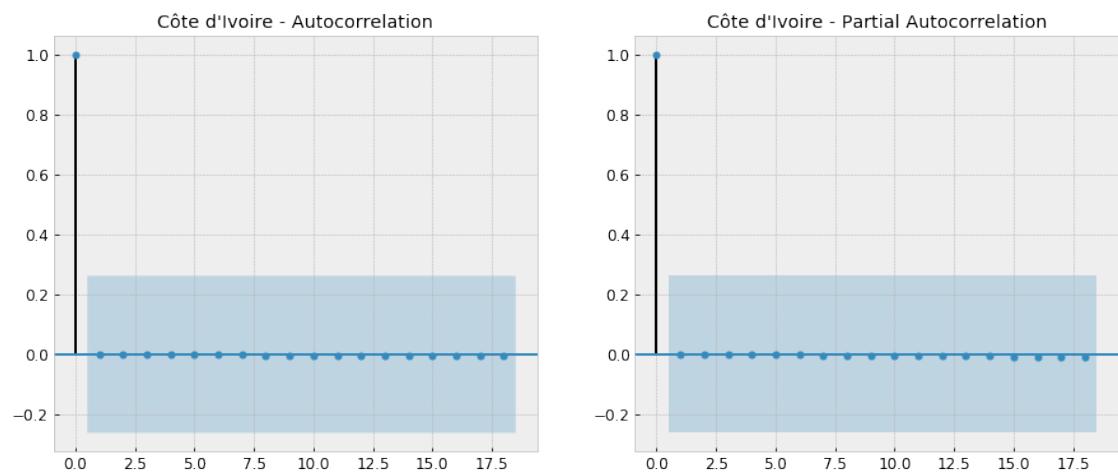


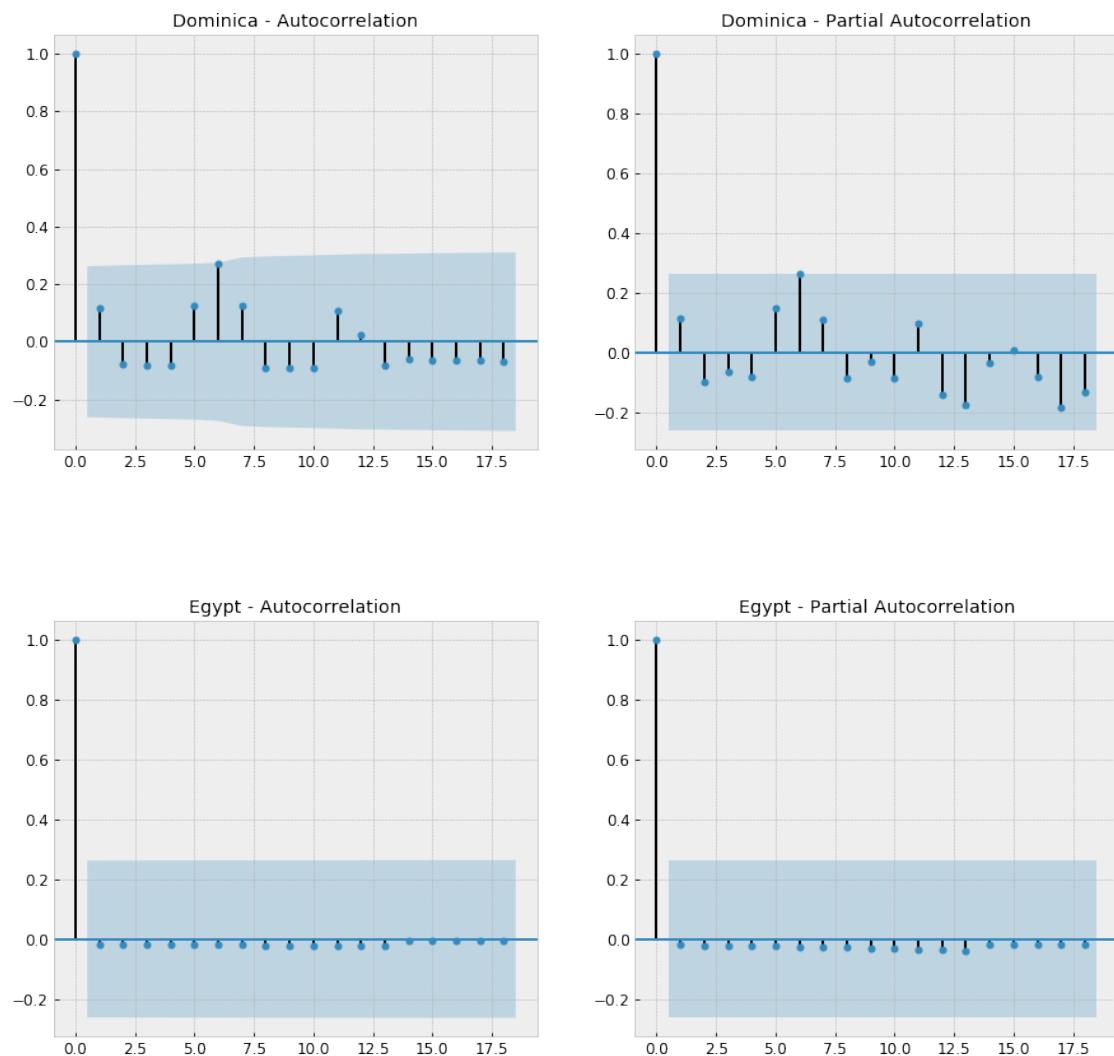
Czechoslovakia - Autocorrelation

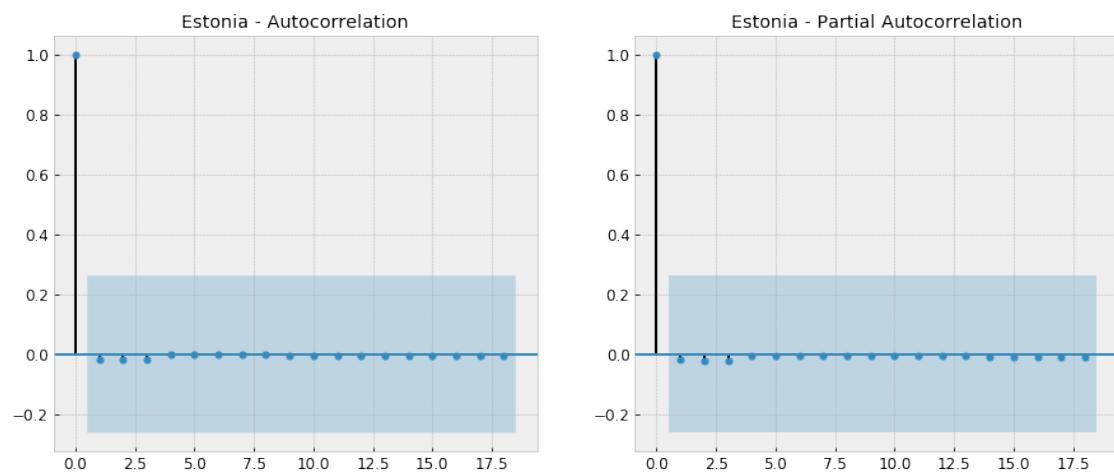
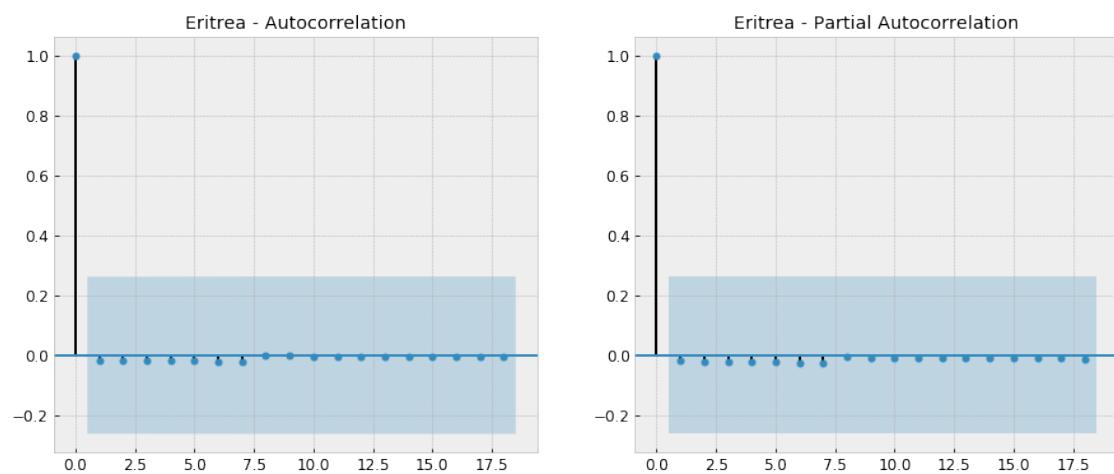
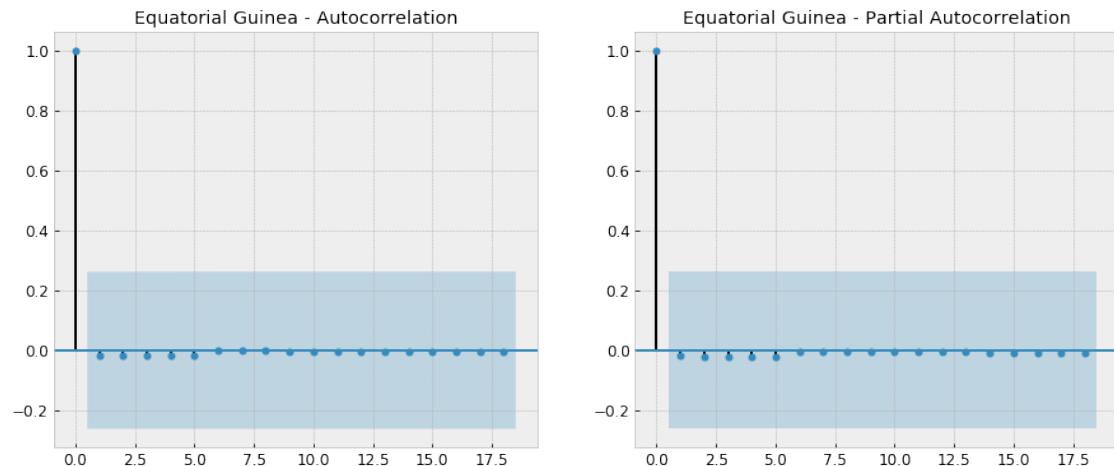


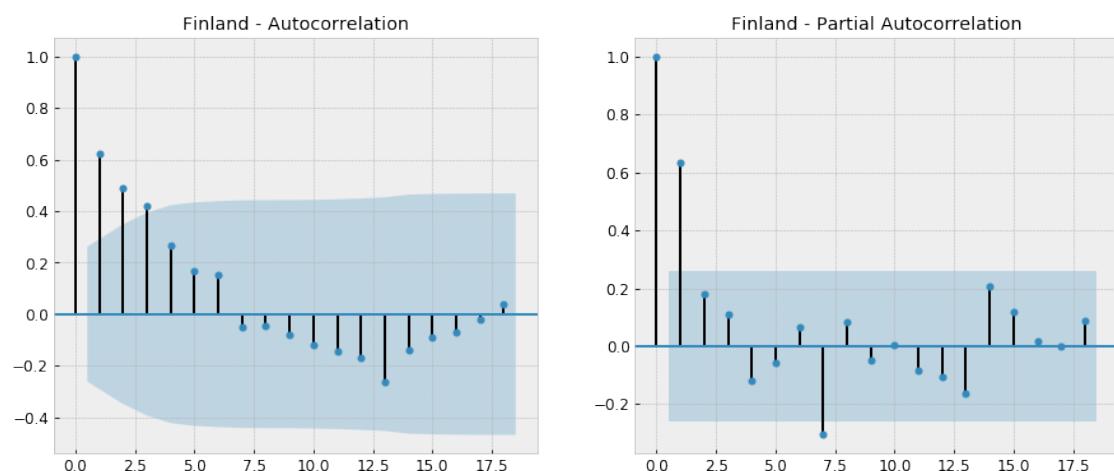
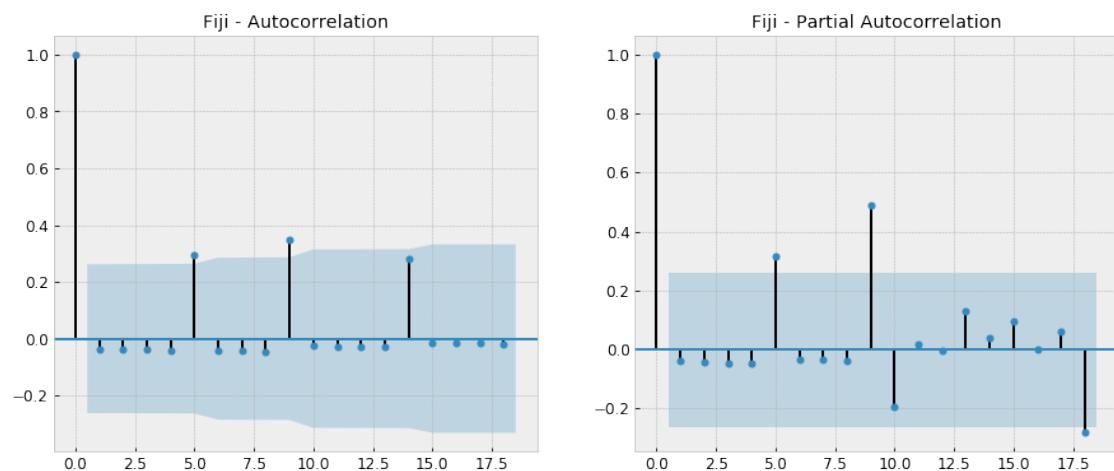
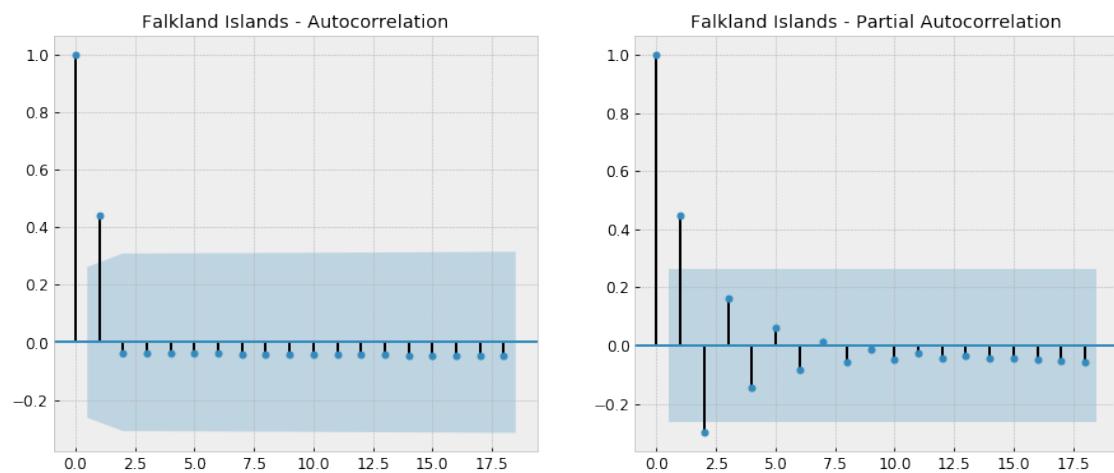
Czechoslovakia - Partial Autocorrelation

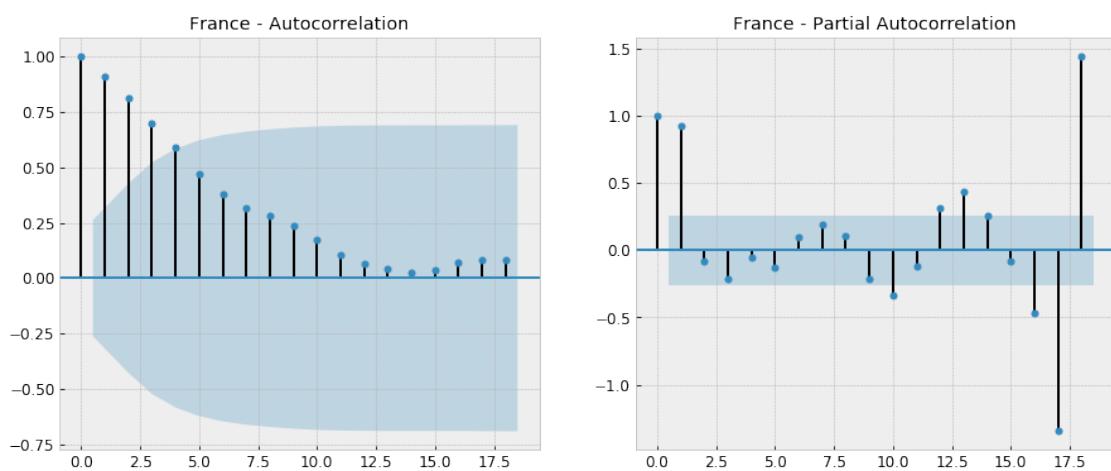
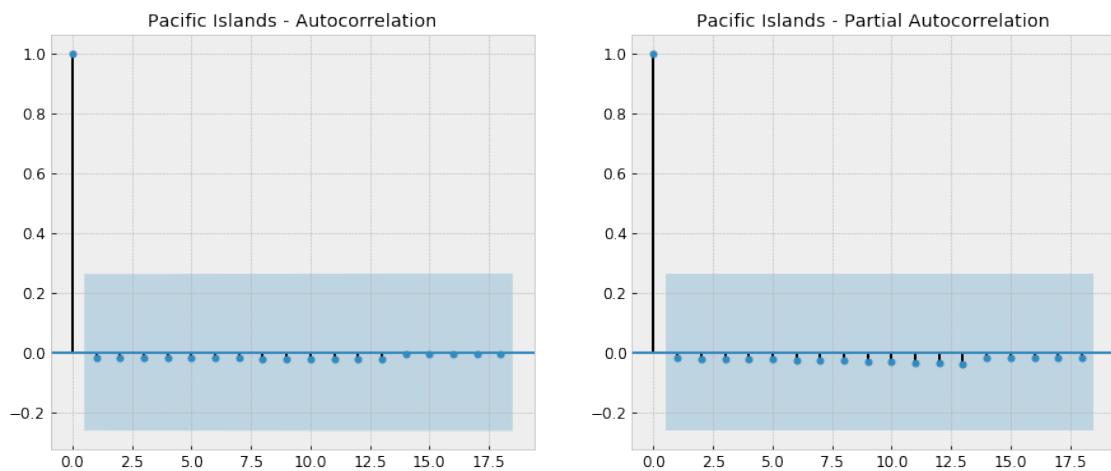


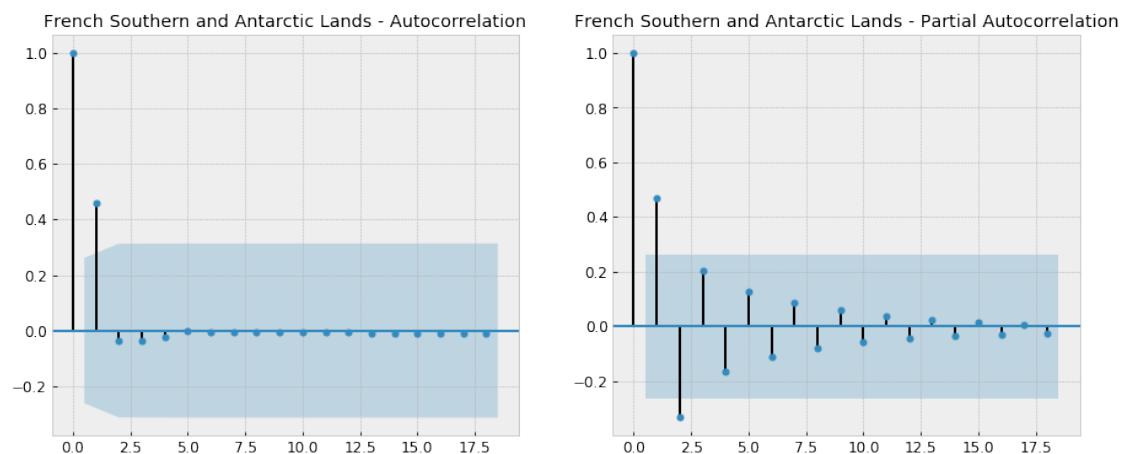
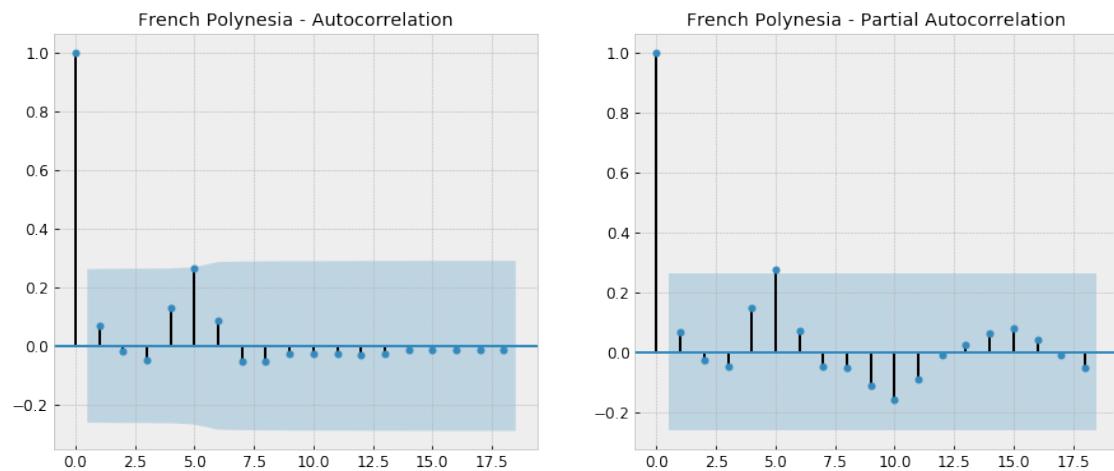
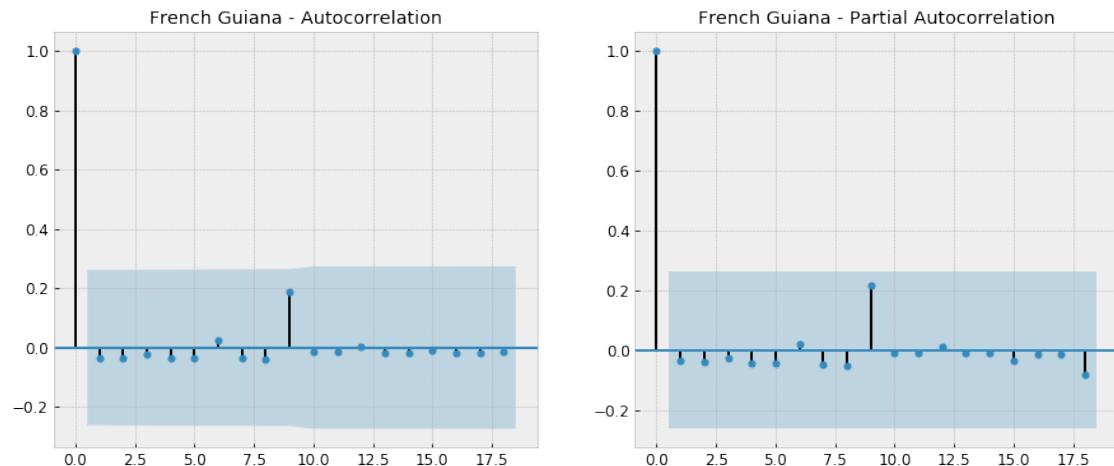


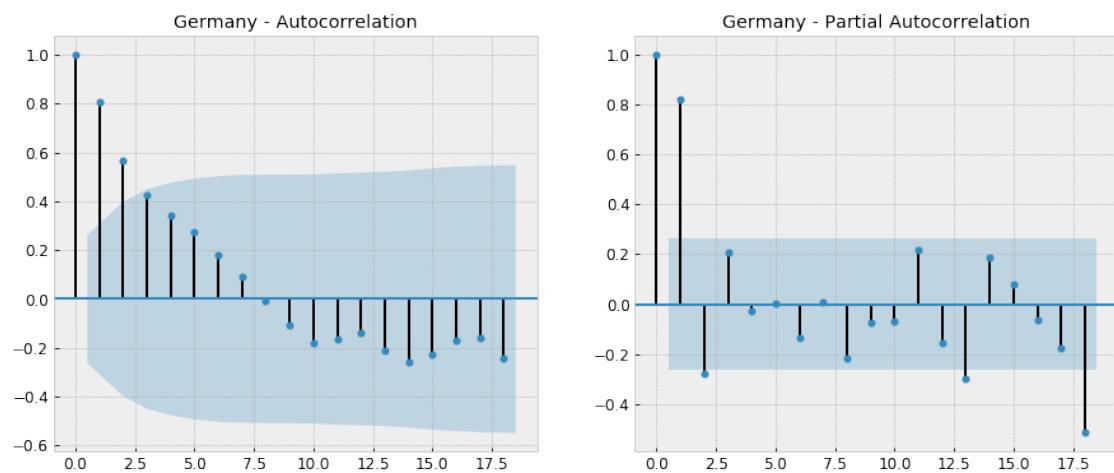
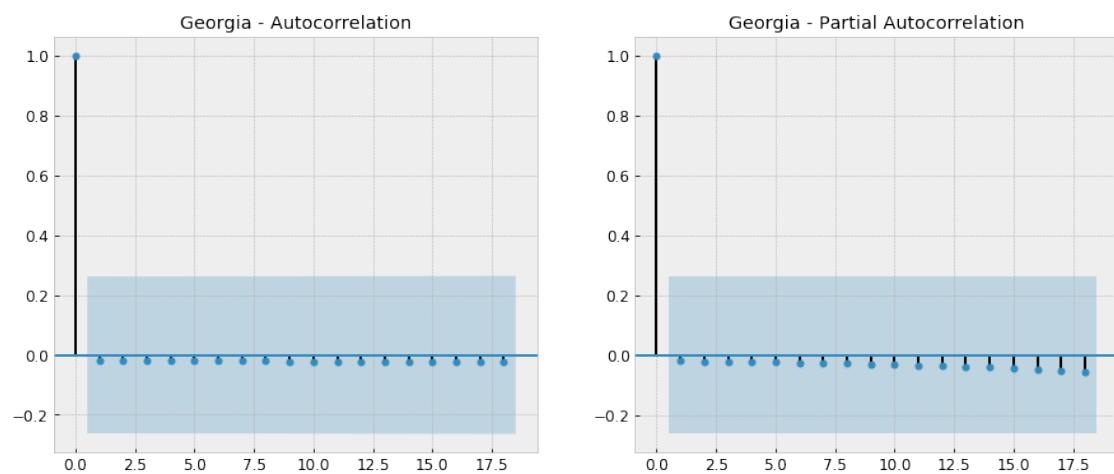
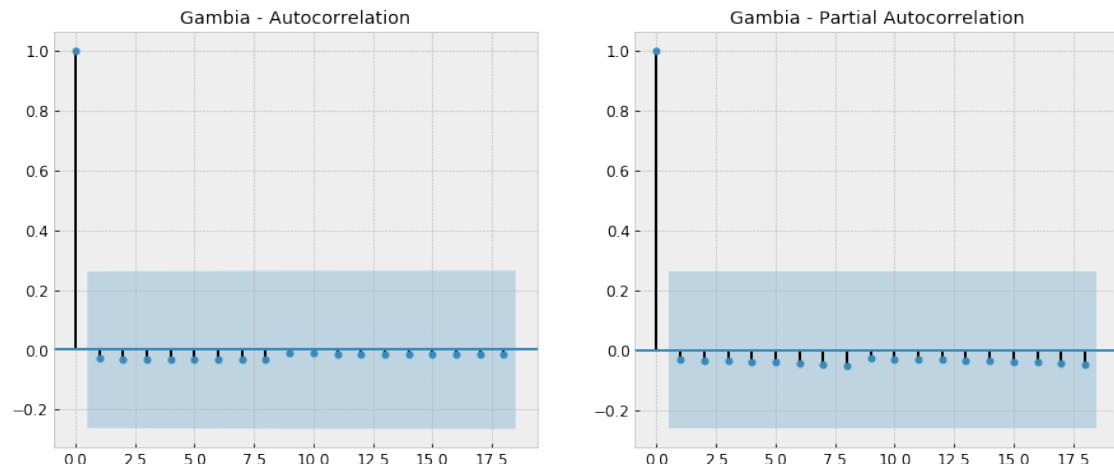


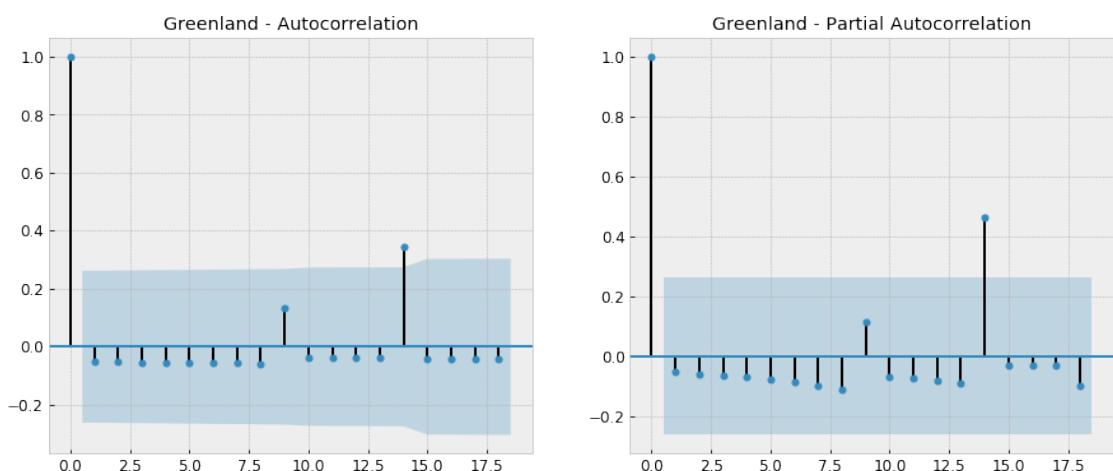
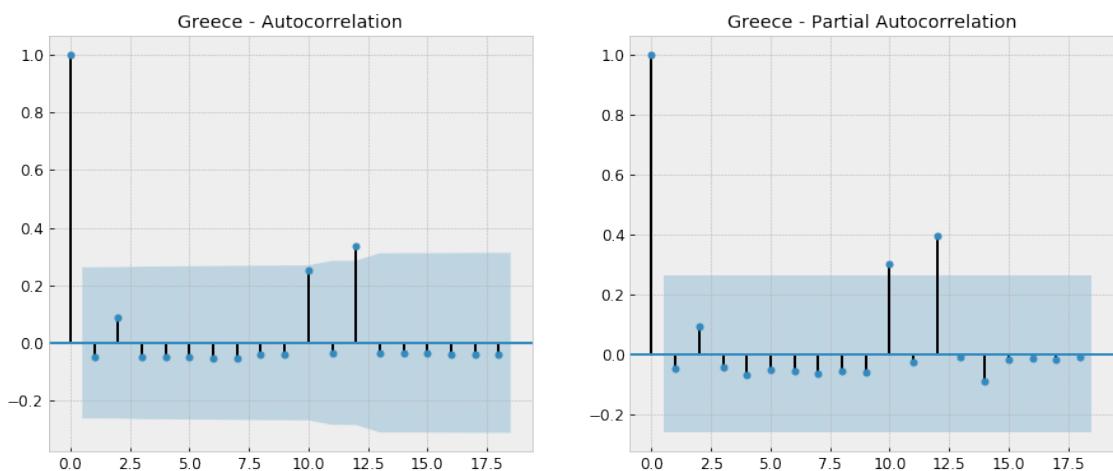
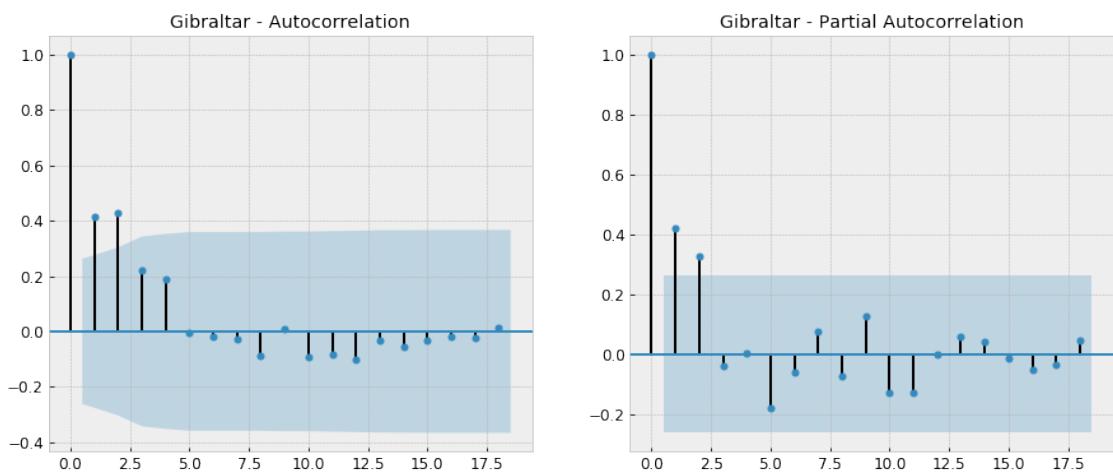


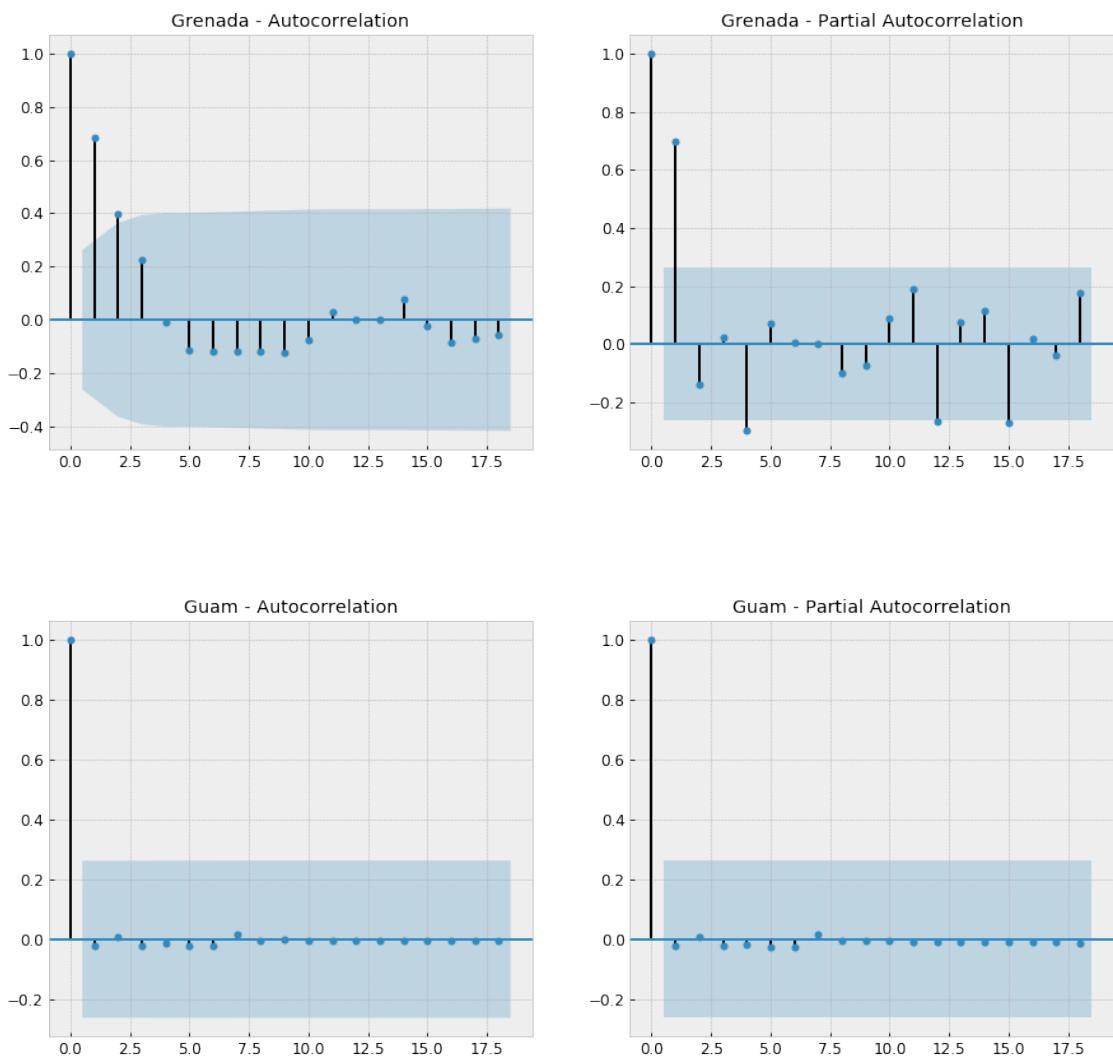


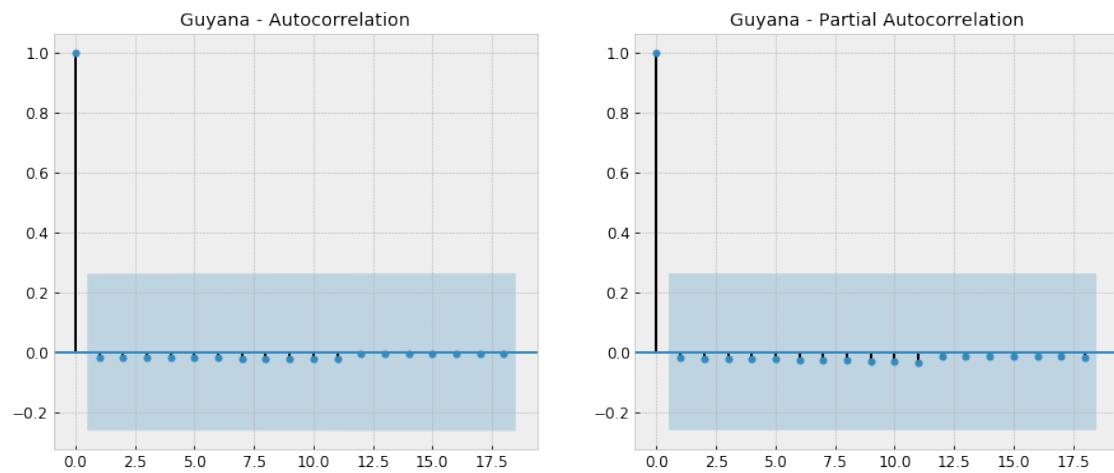
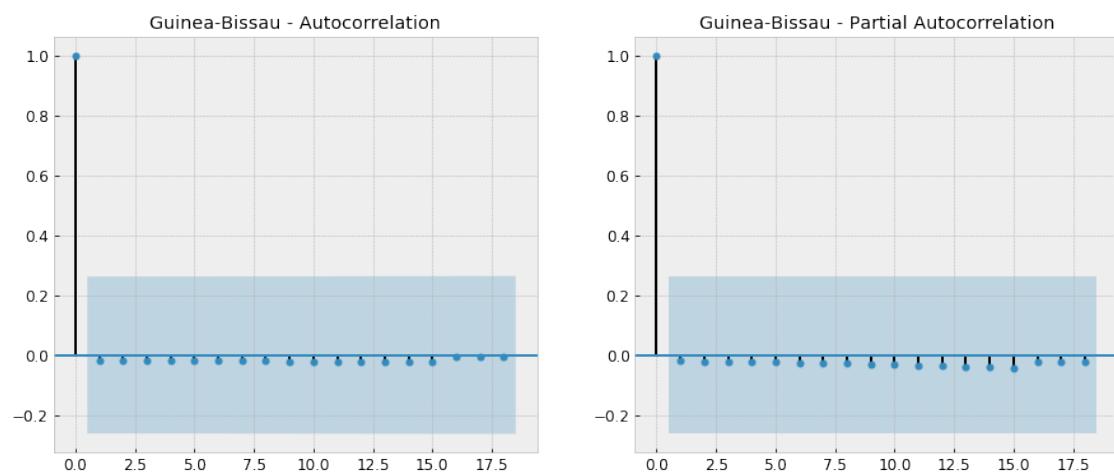
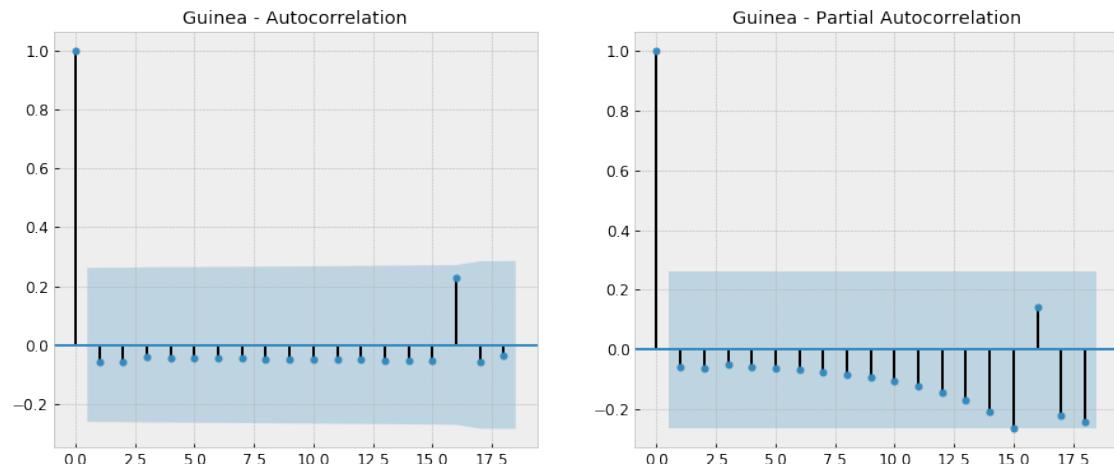


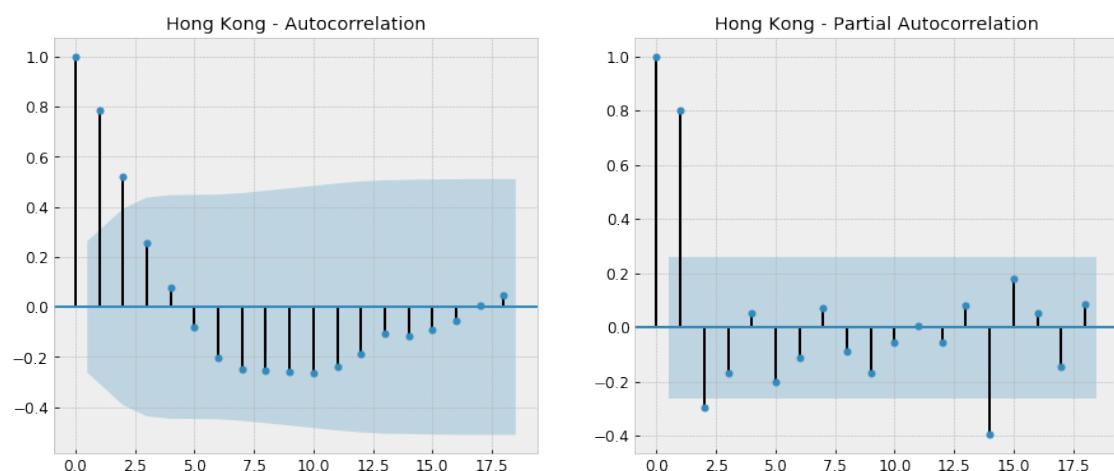
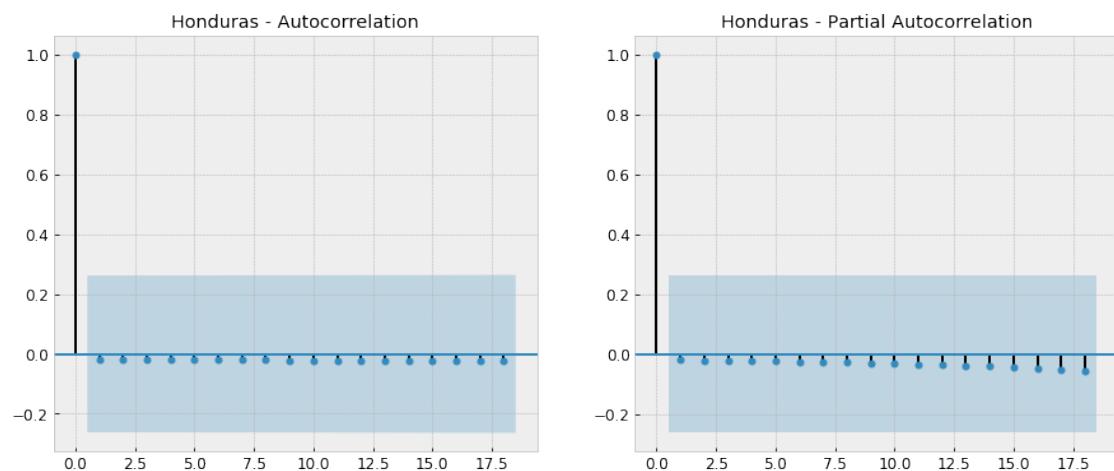
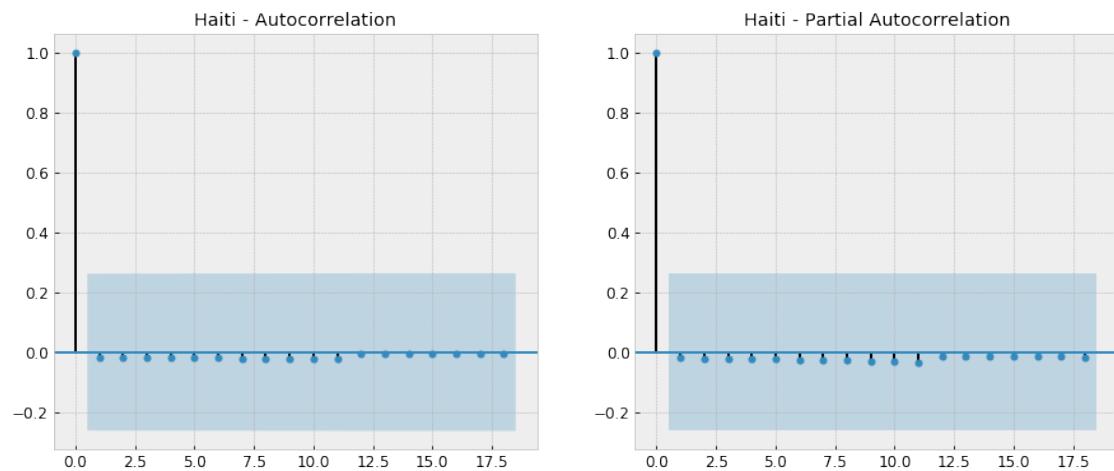


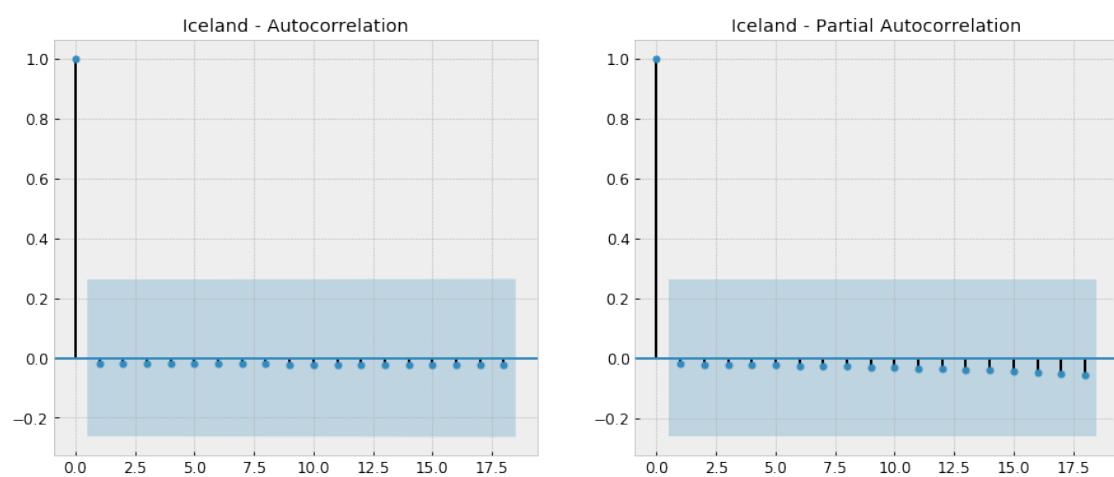
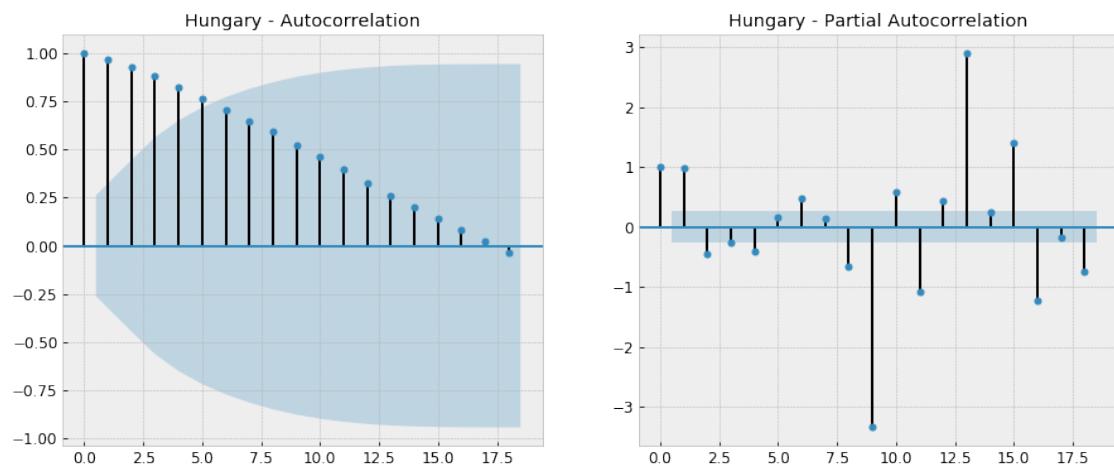




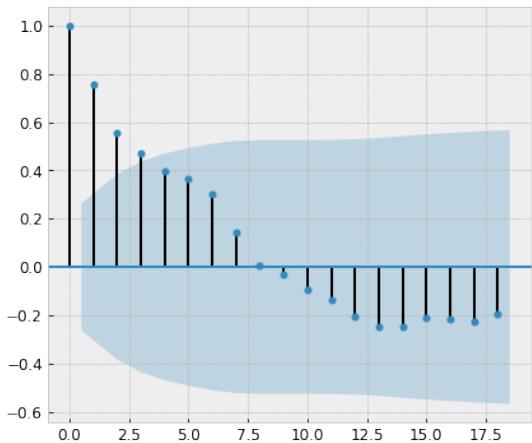




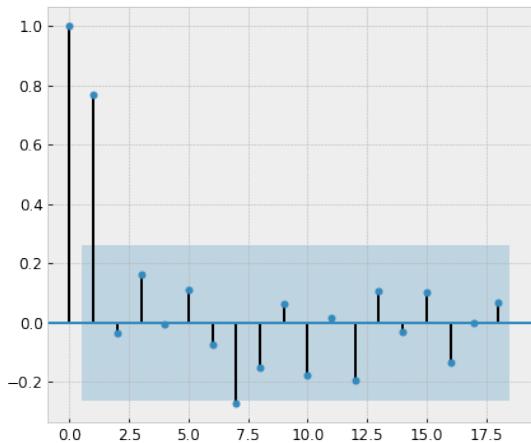




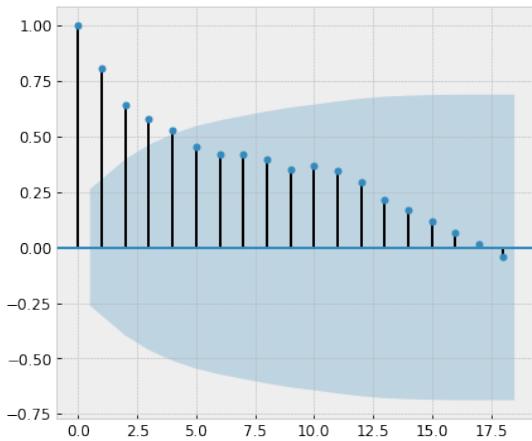
India - Autocorrelation



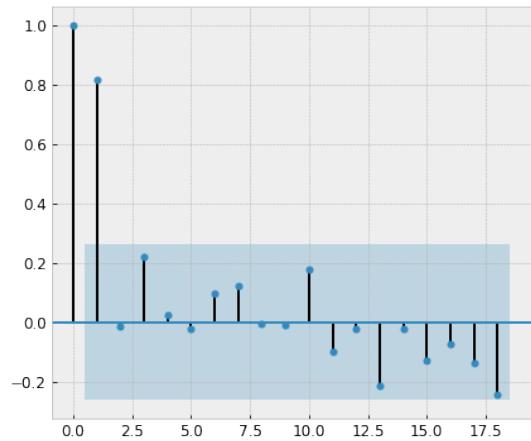
India - Partial Autocorrelation



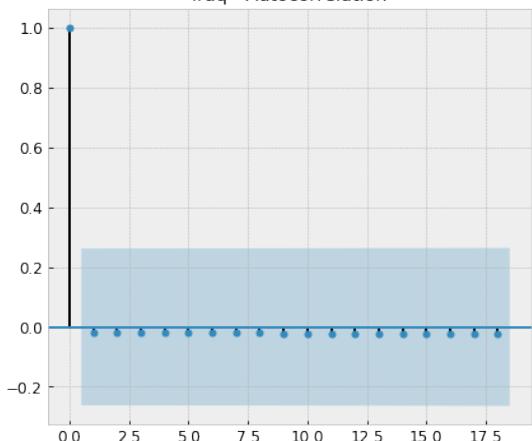
Indonesia - Autocorrelation



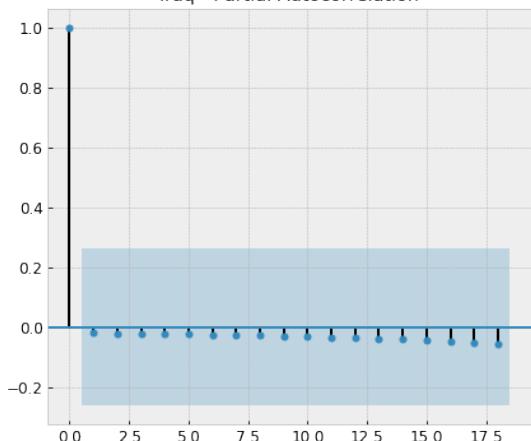
Indonesia - Partial Autocorrelation

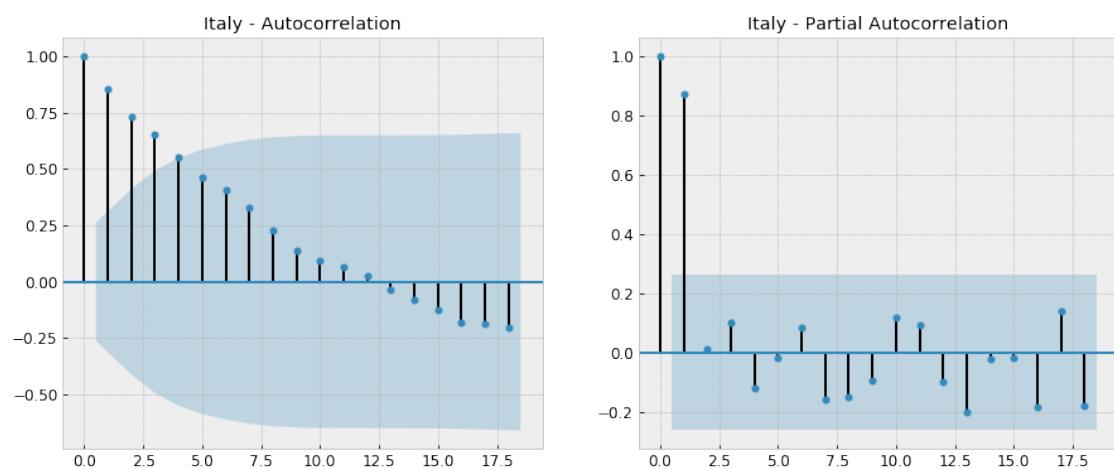
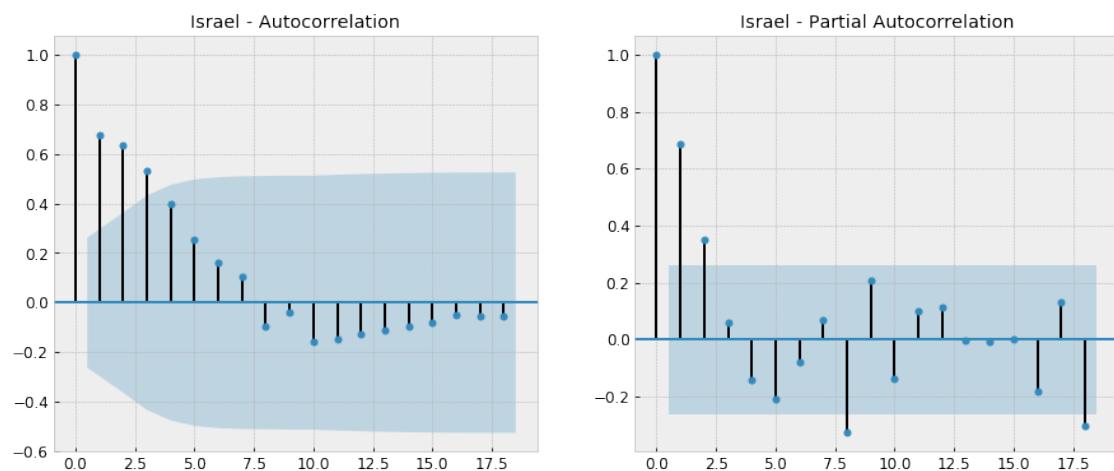
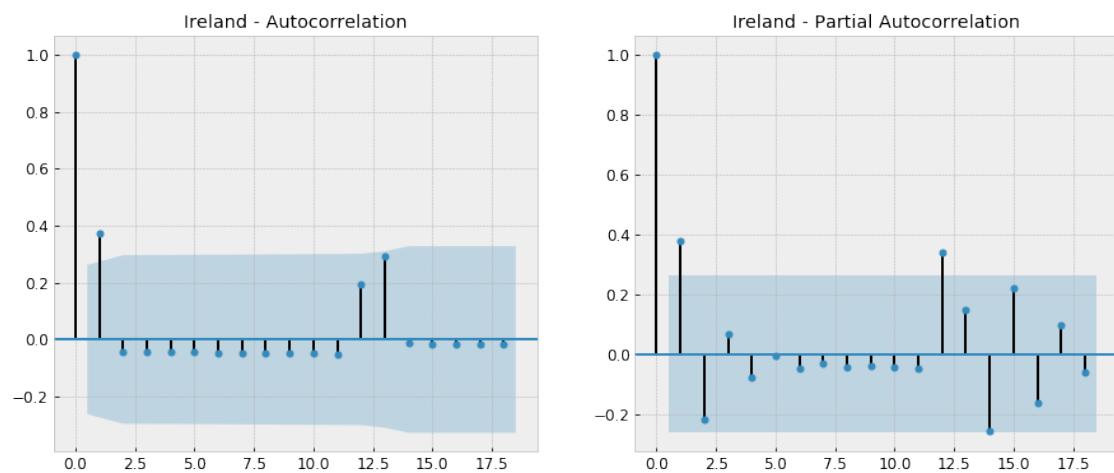


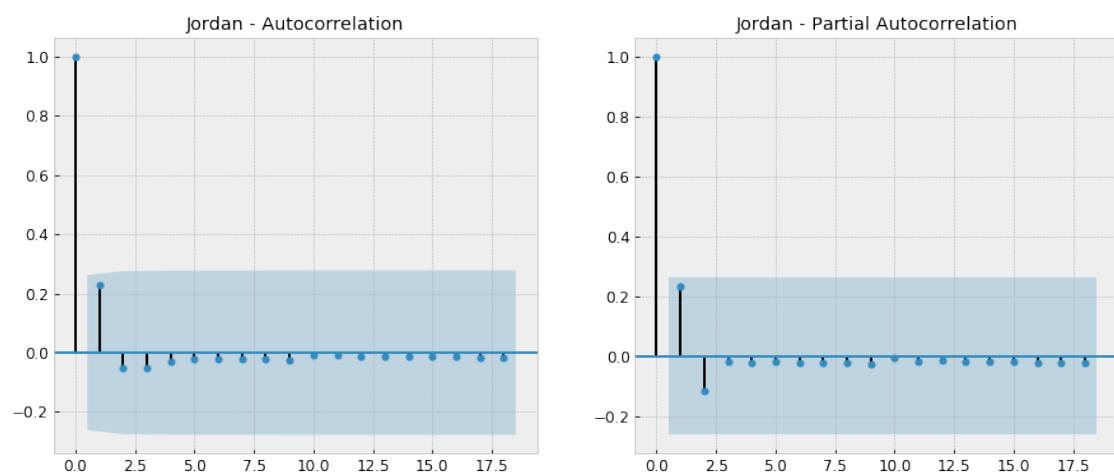
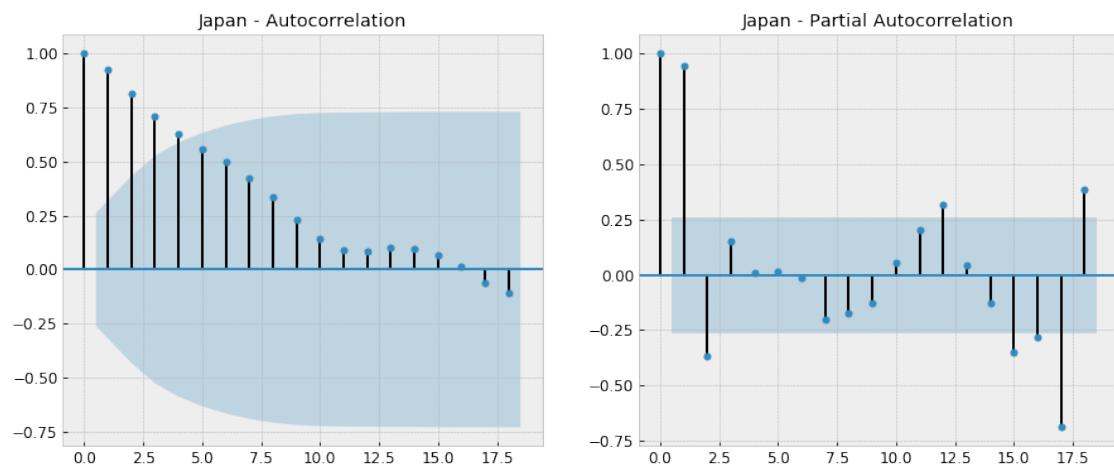
Iraq - Autocorrelation

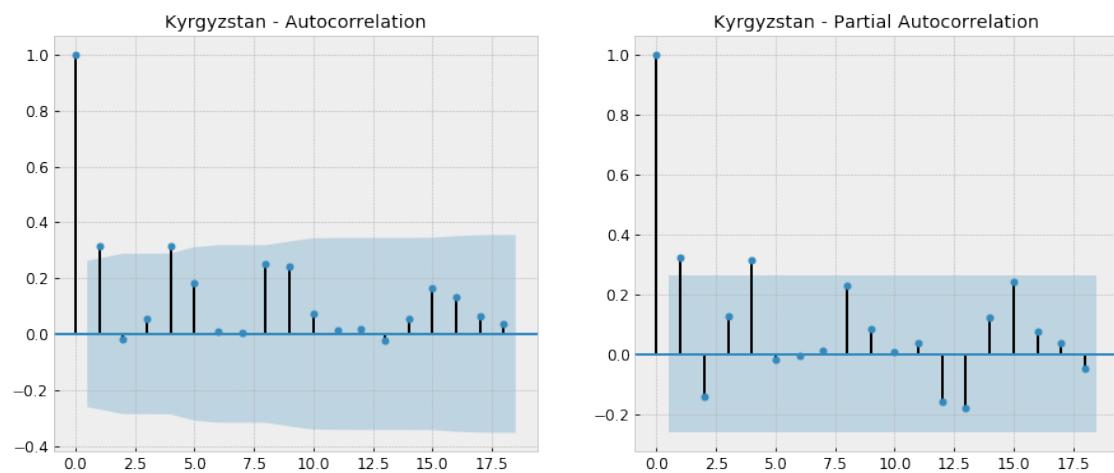
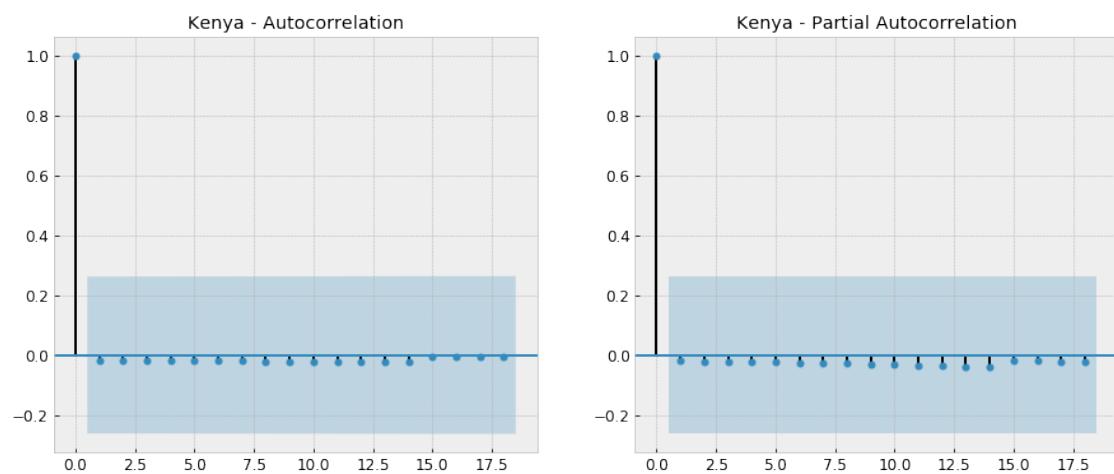
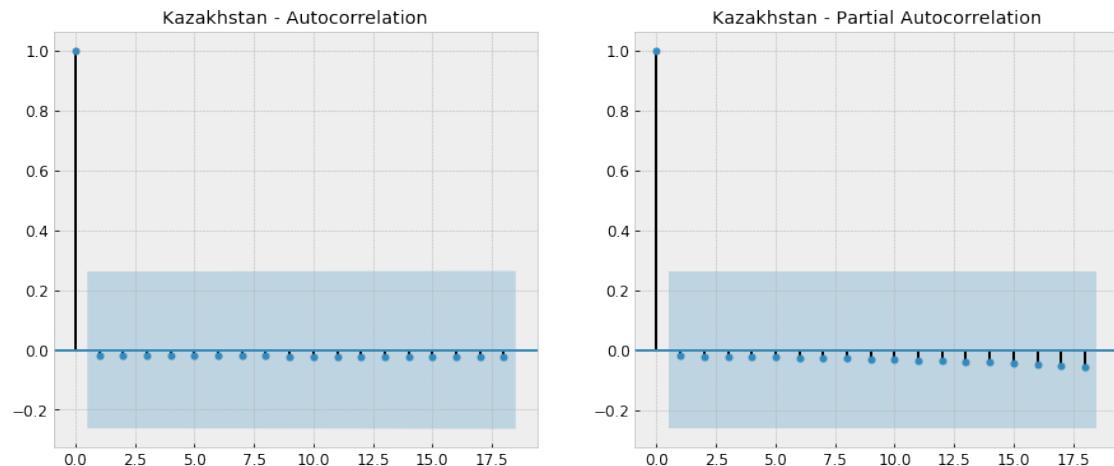


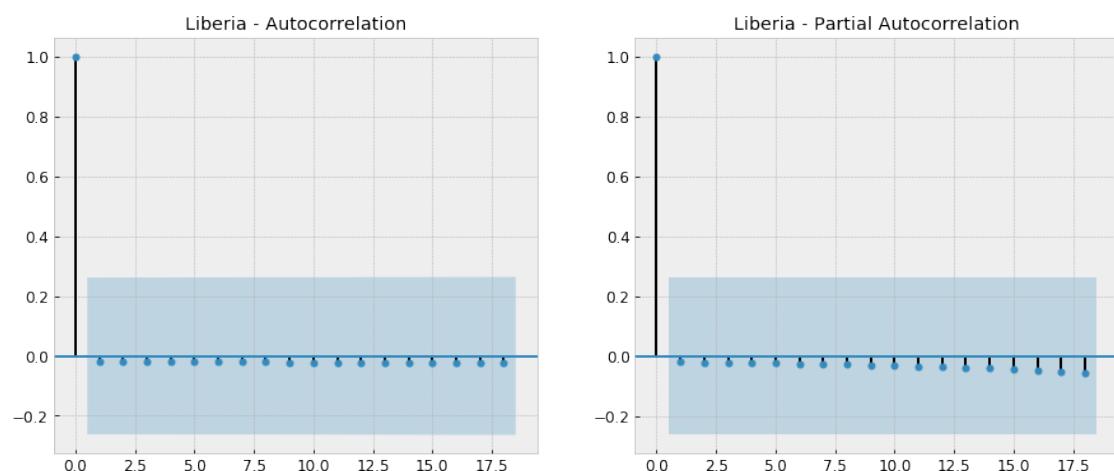
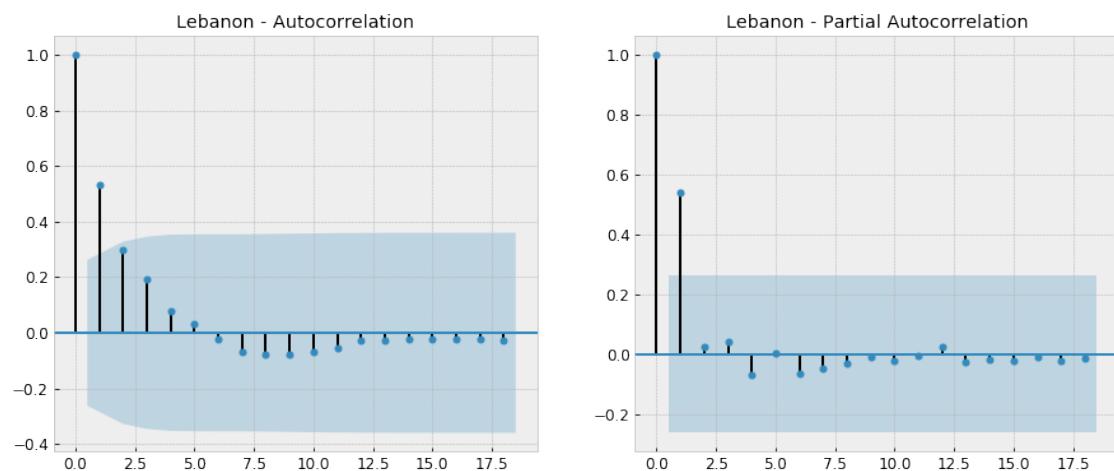
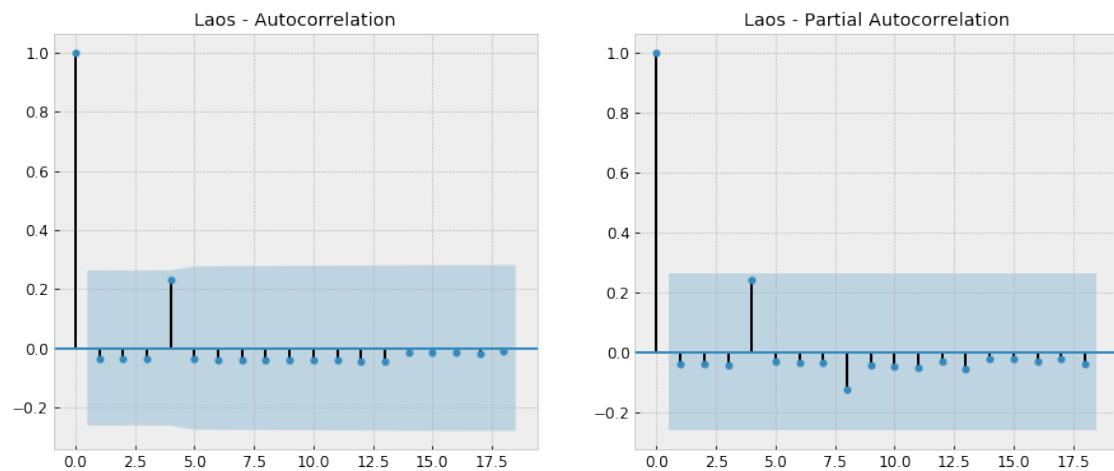
Iraq - Partial Autocorrelation

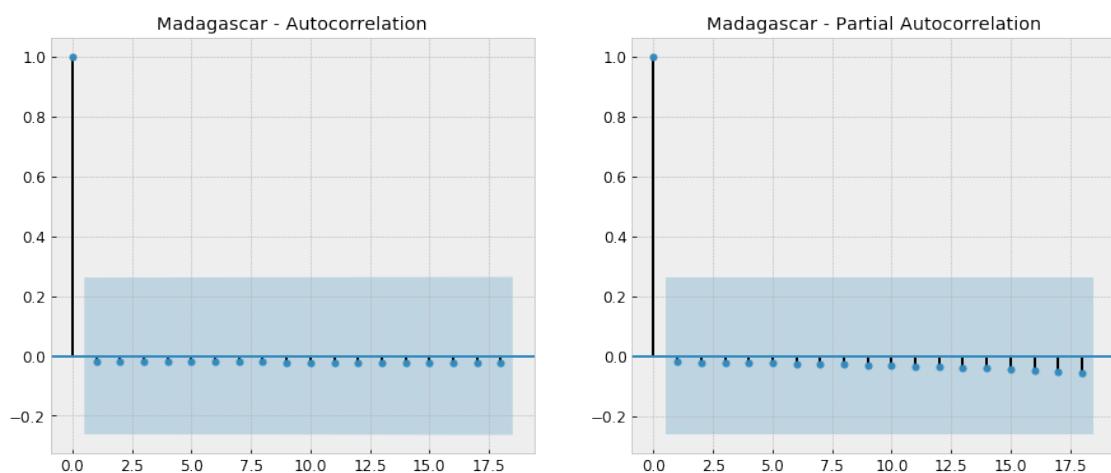
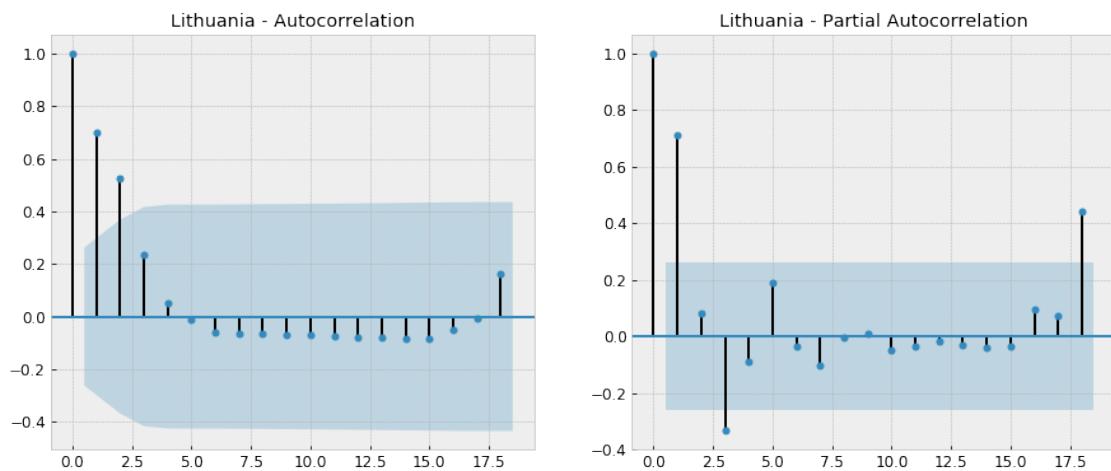


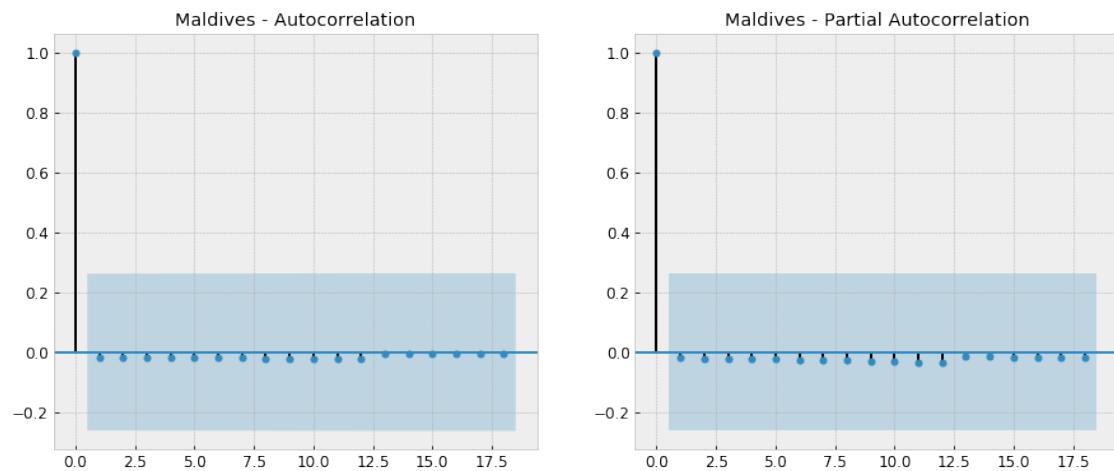
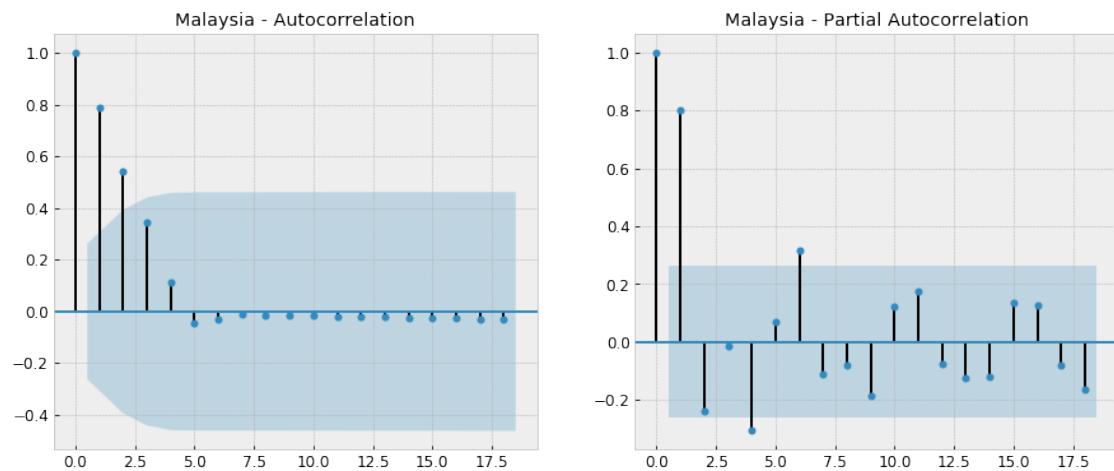
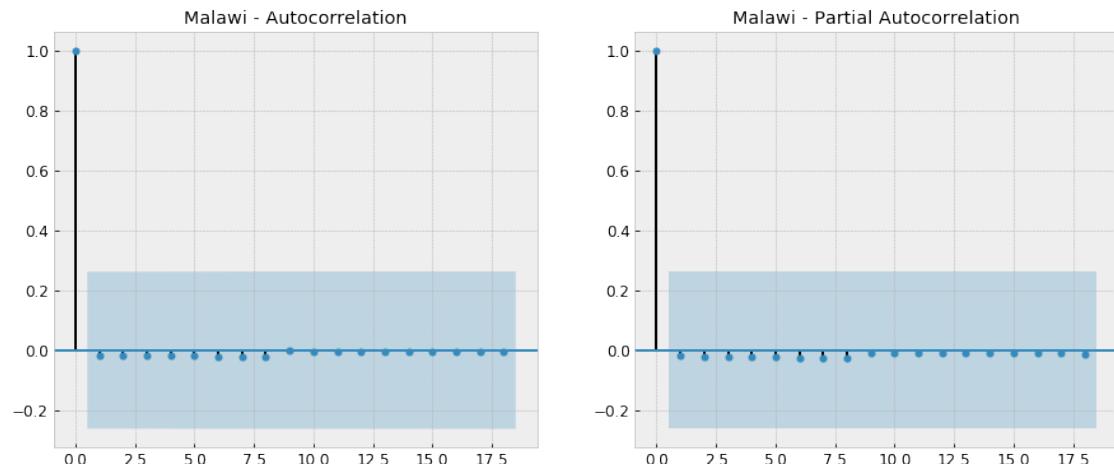


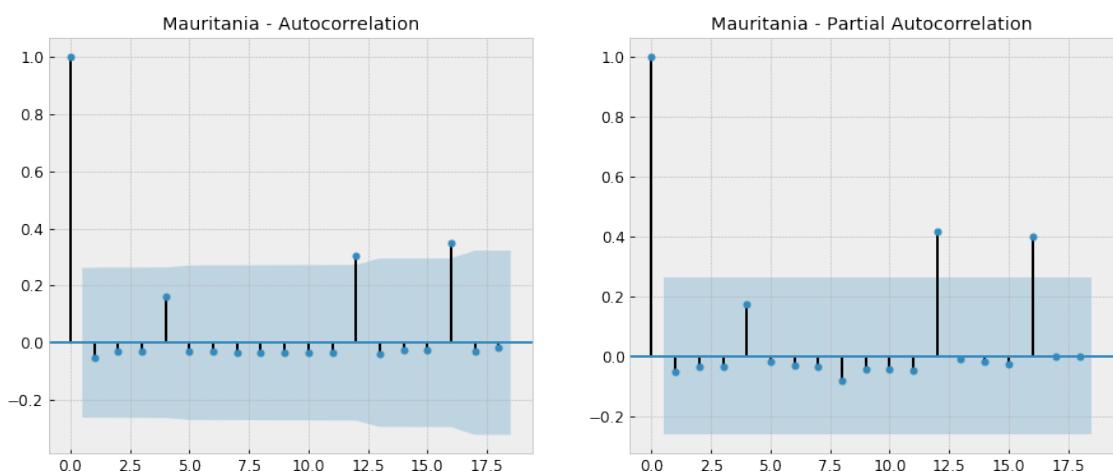
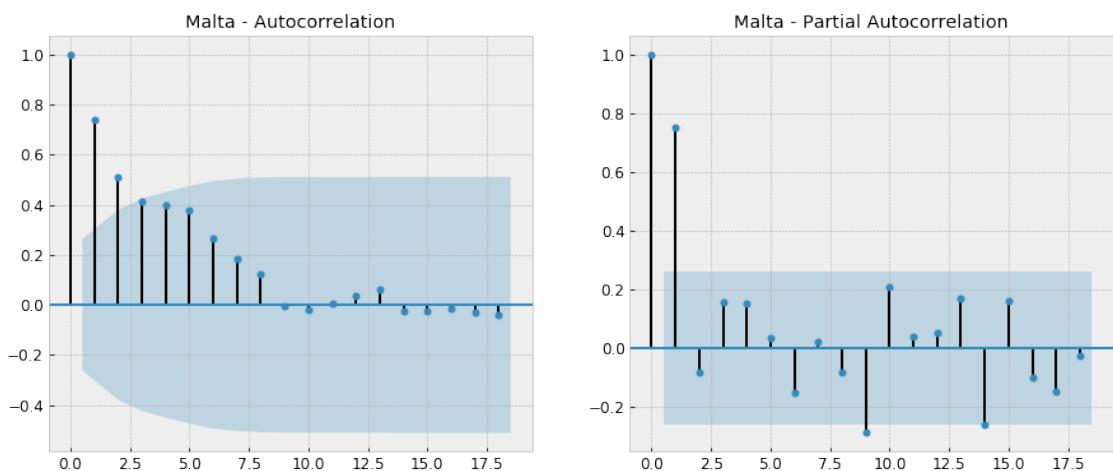
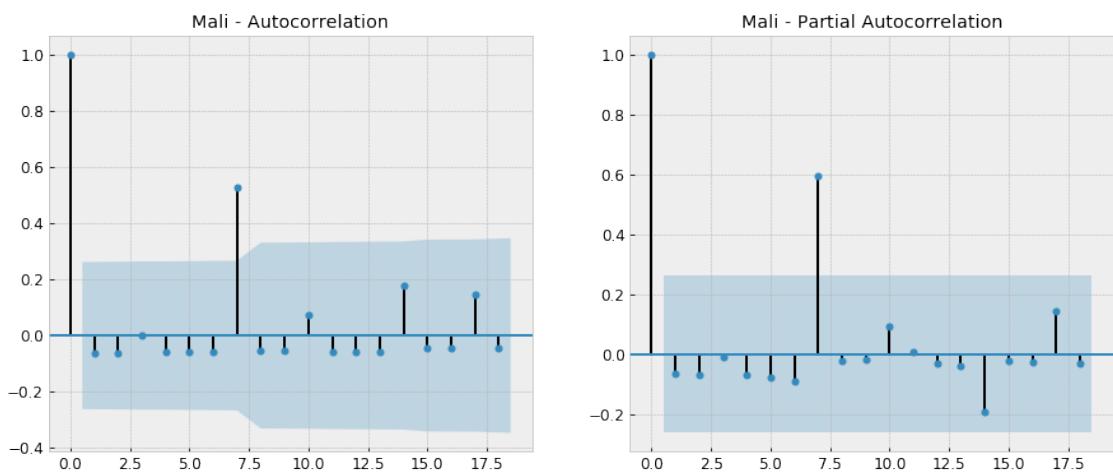


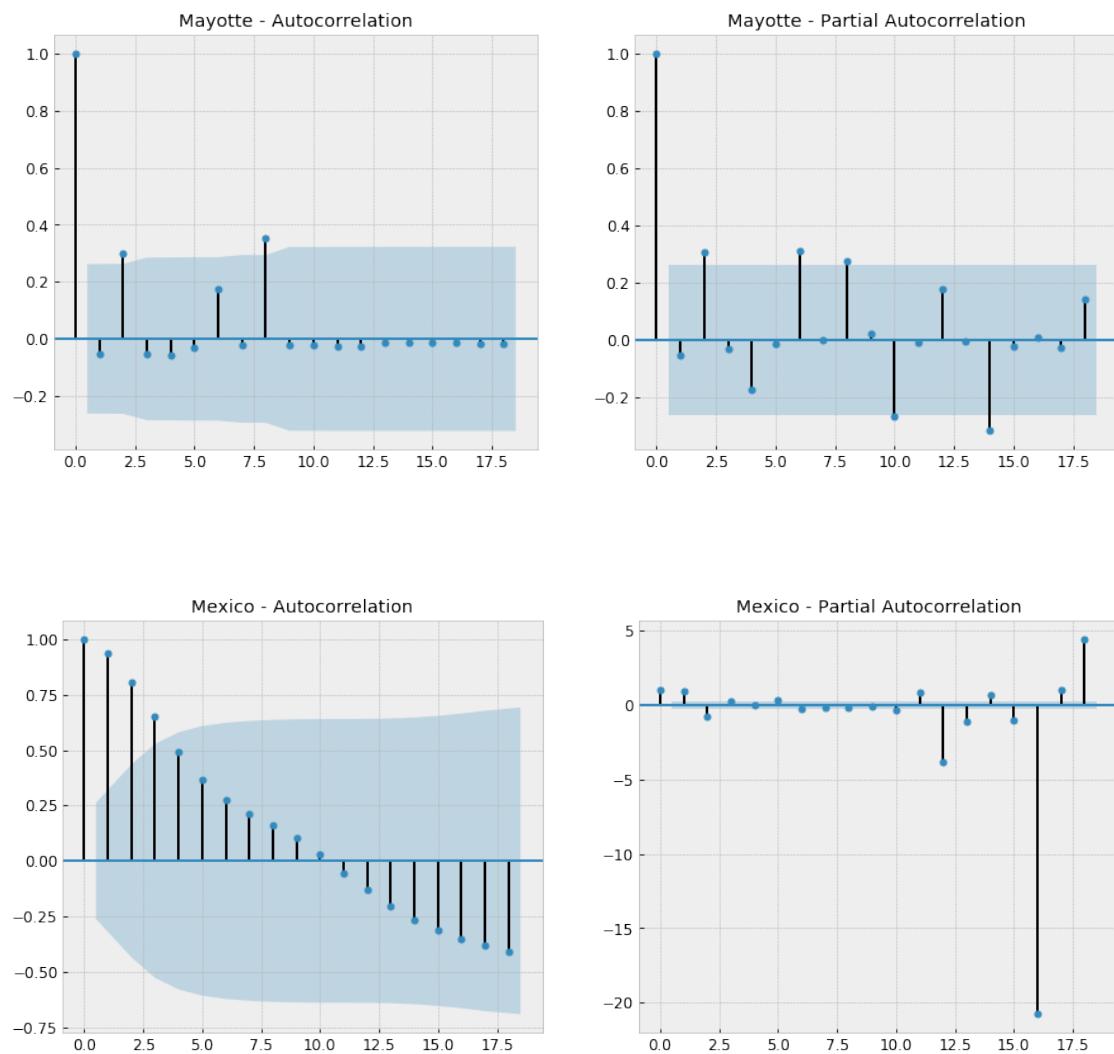


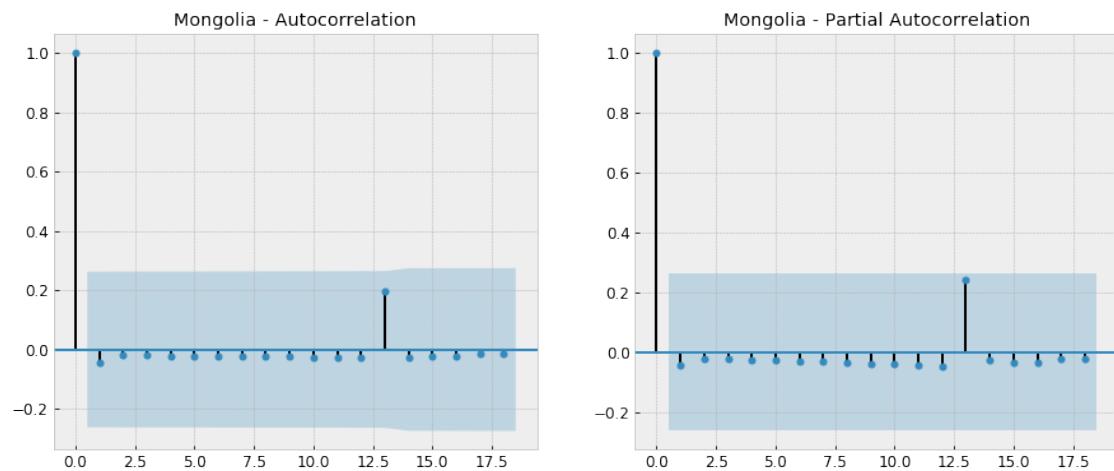
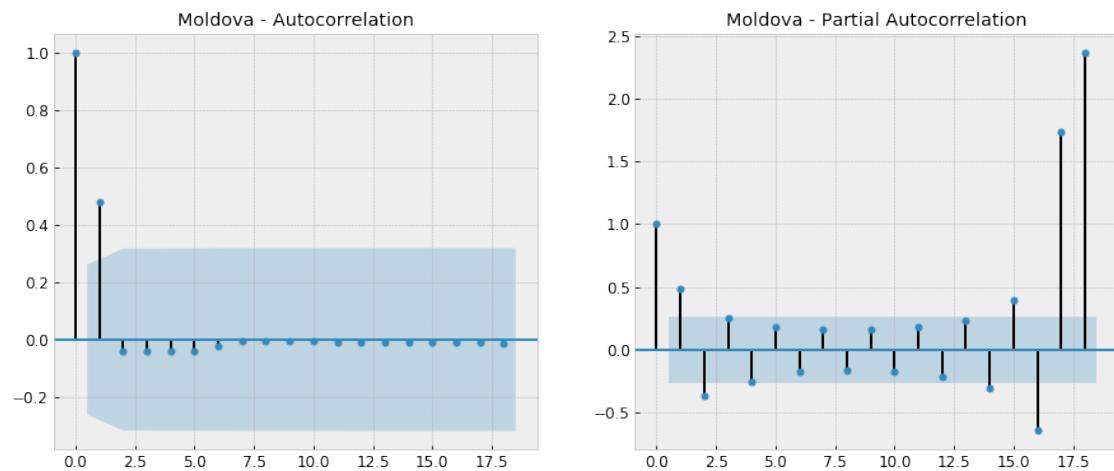
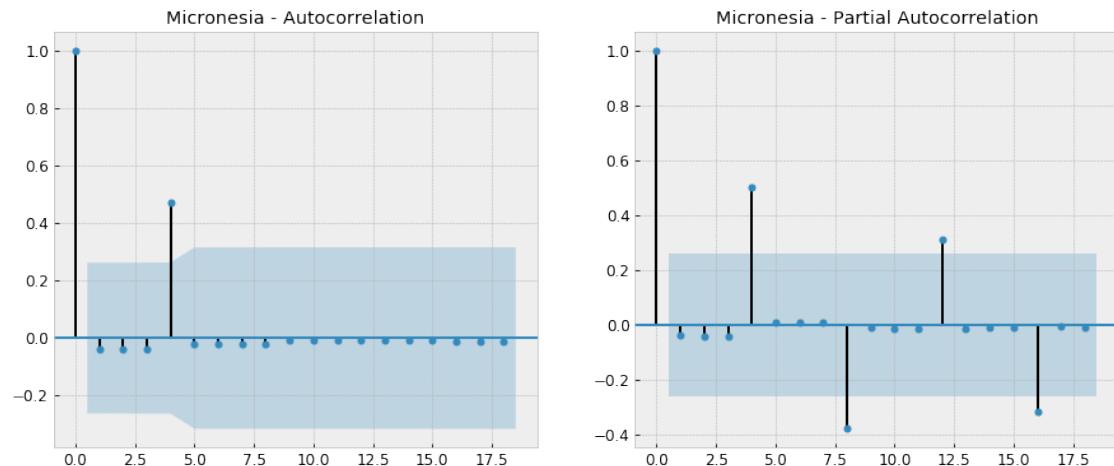


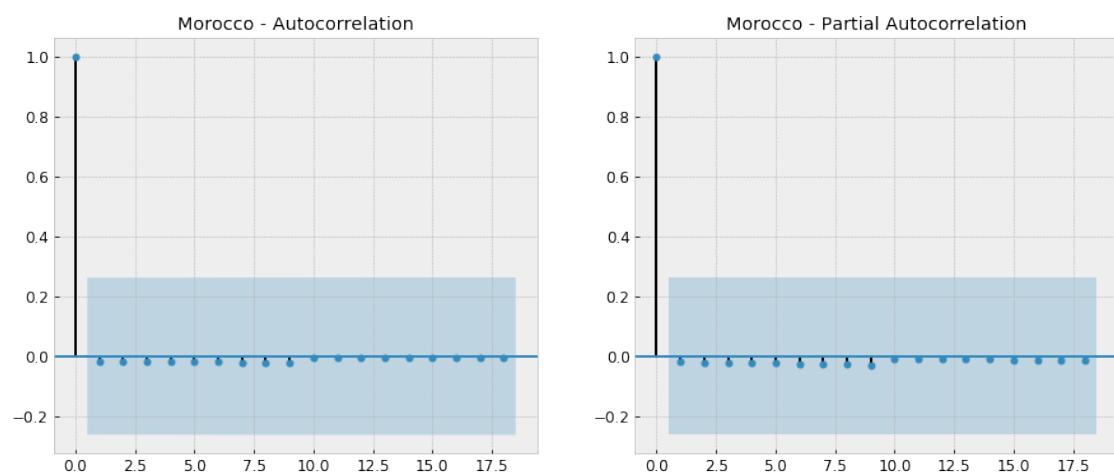
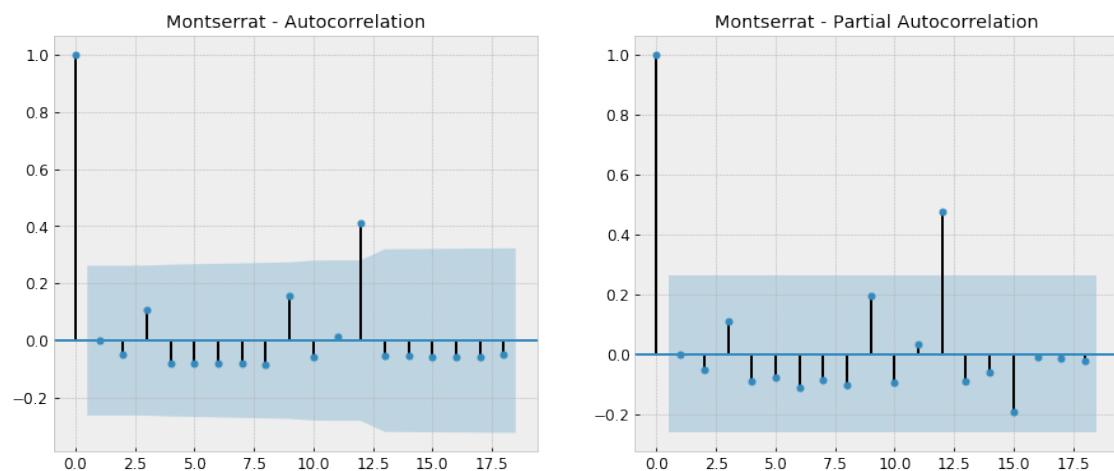
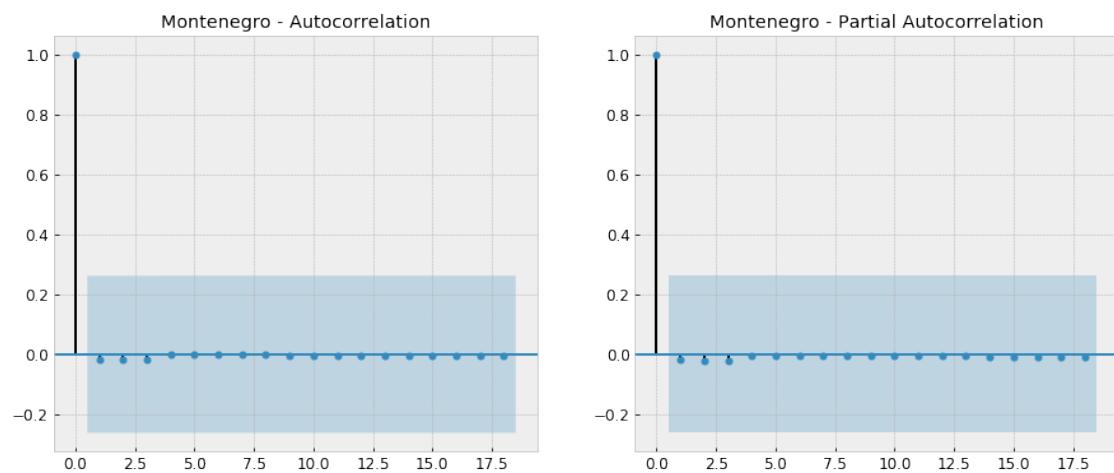


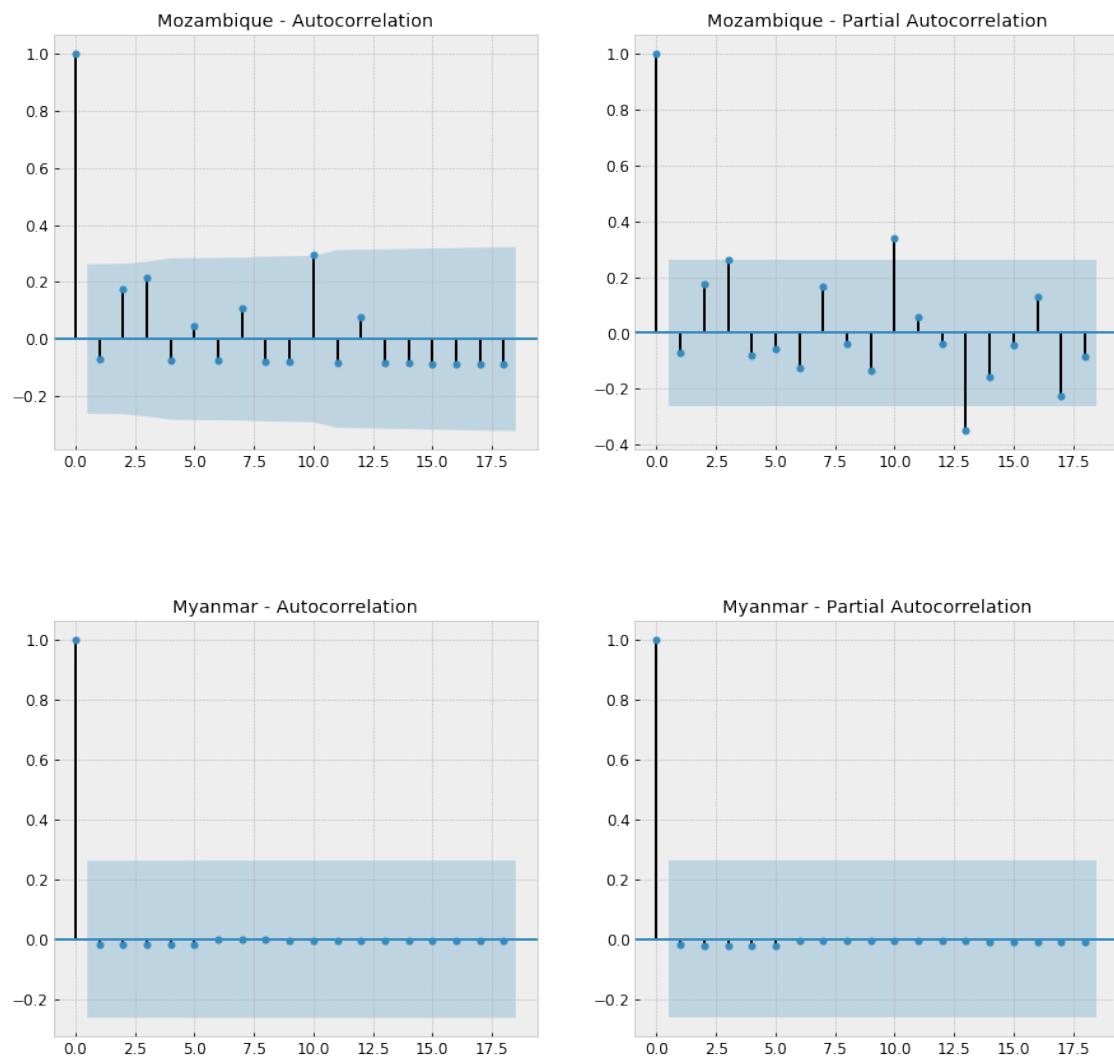


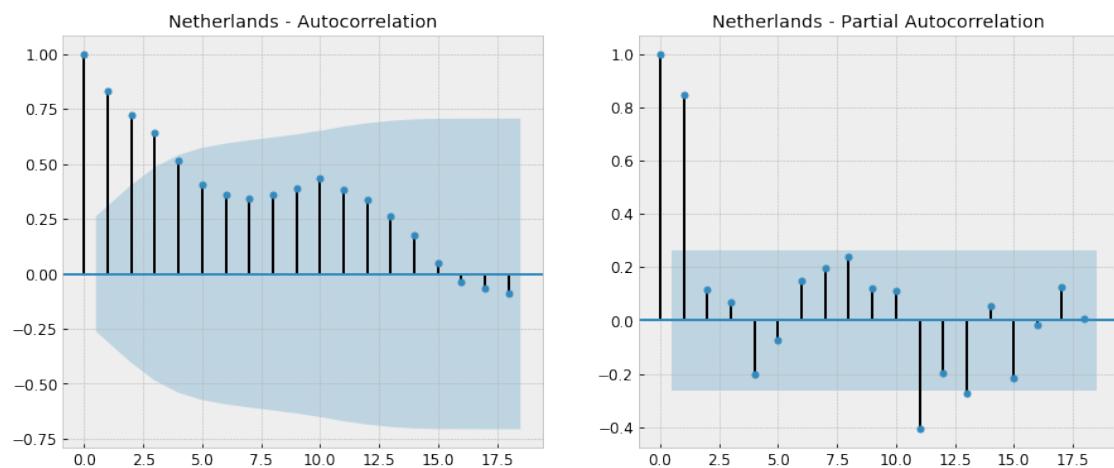
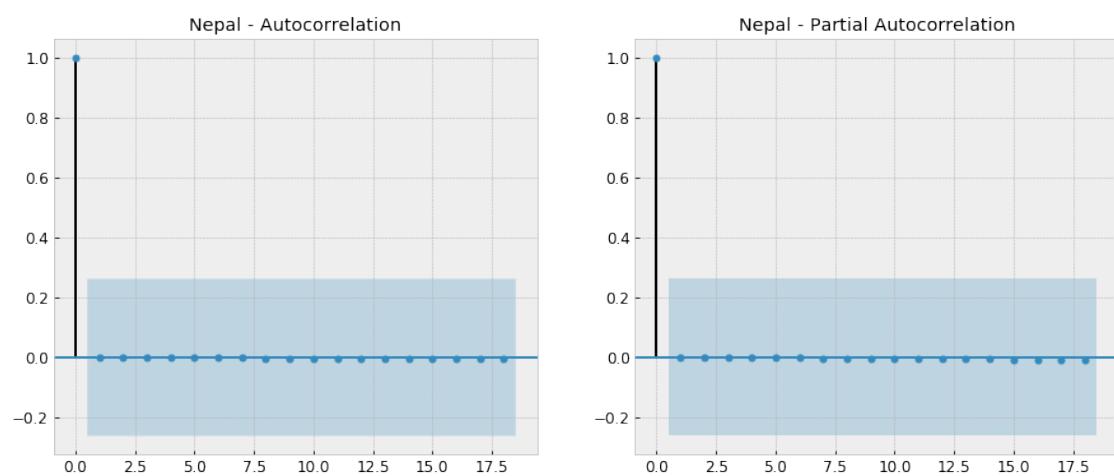
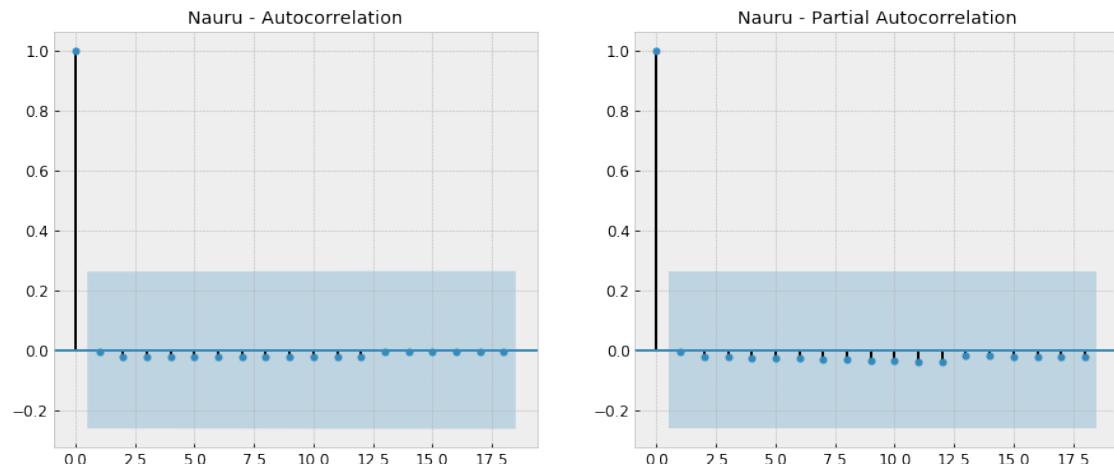


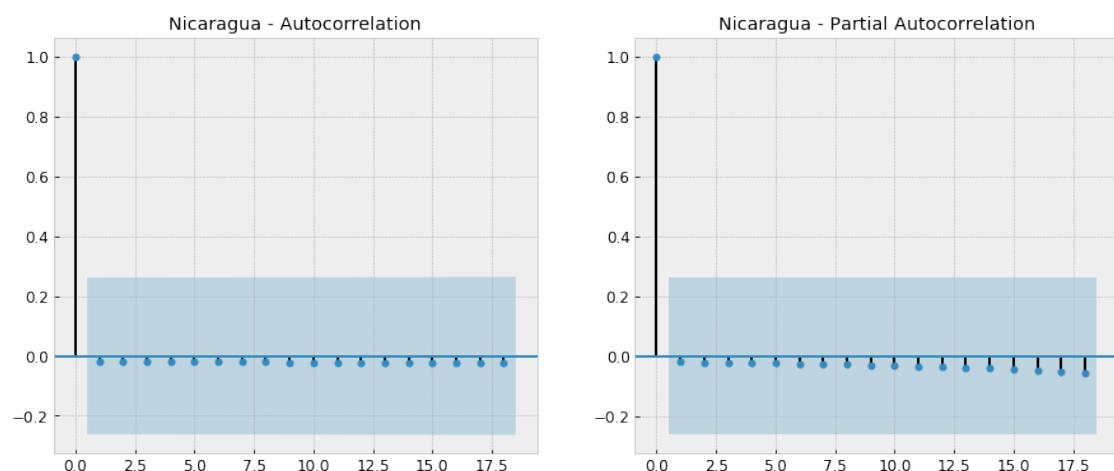
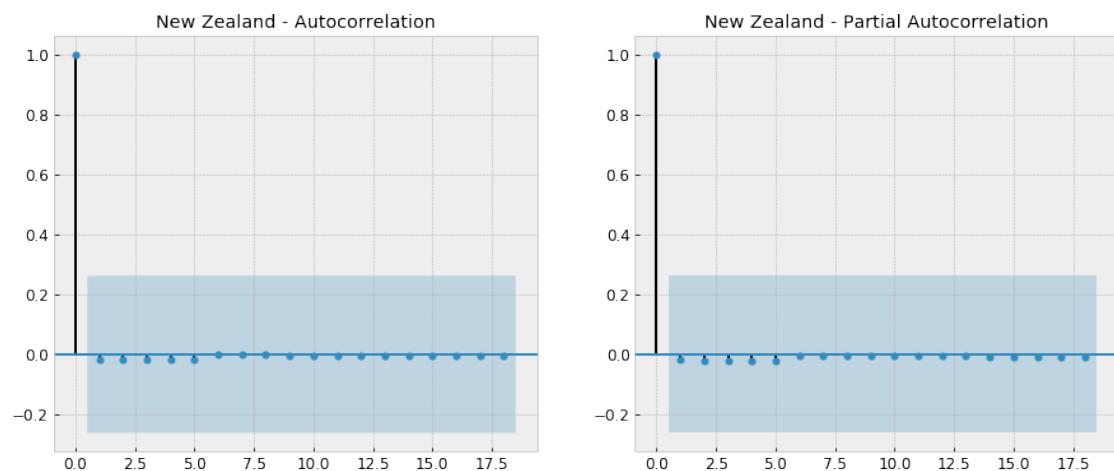
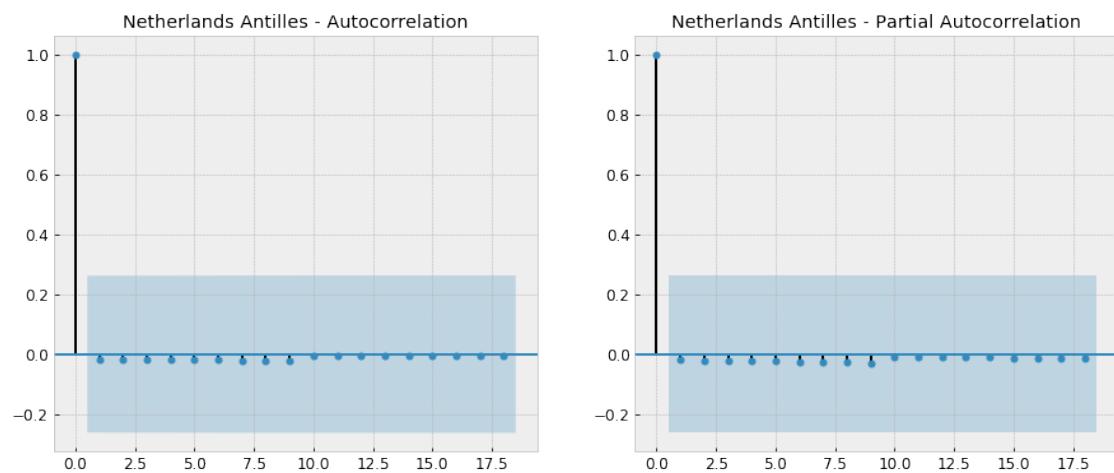


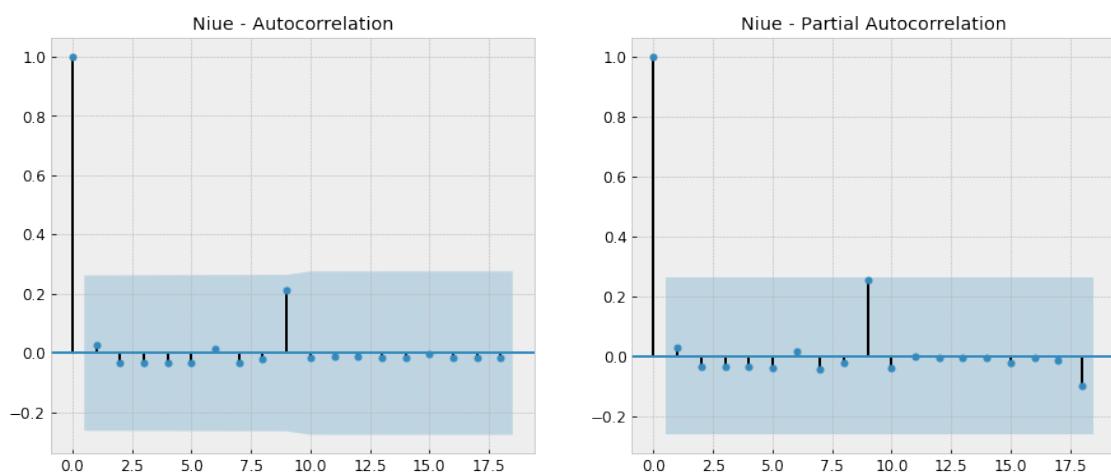
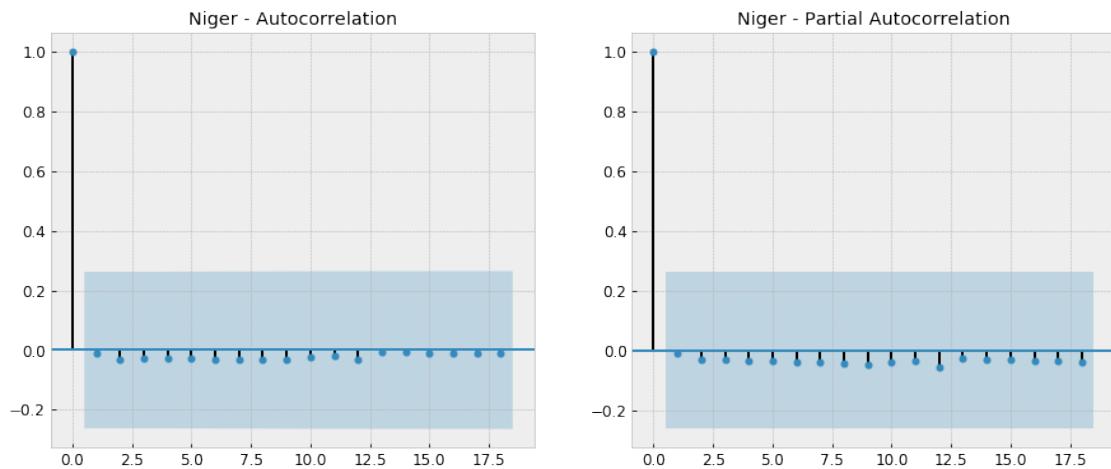


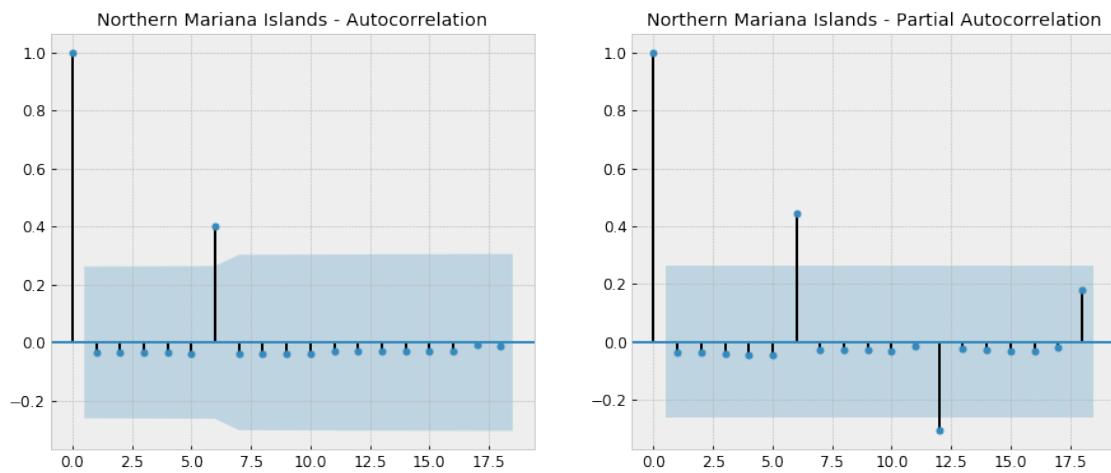
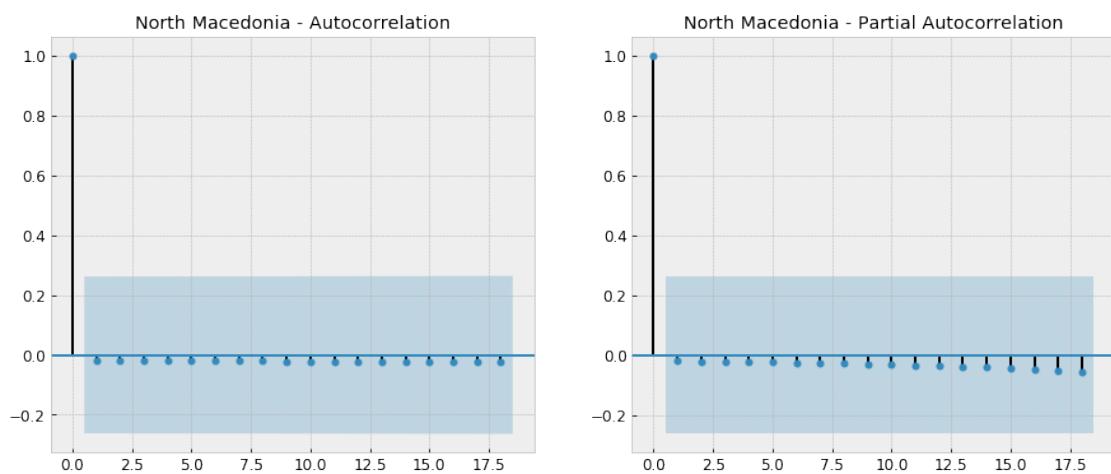
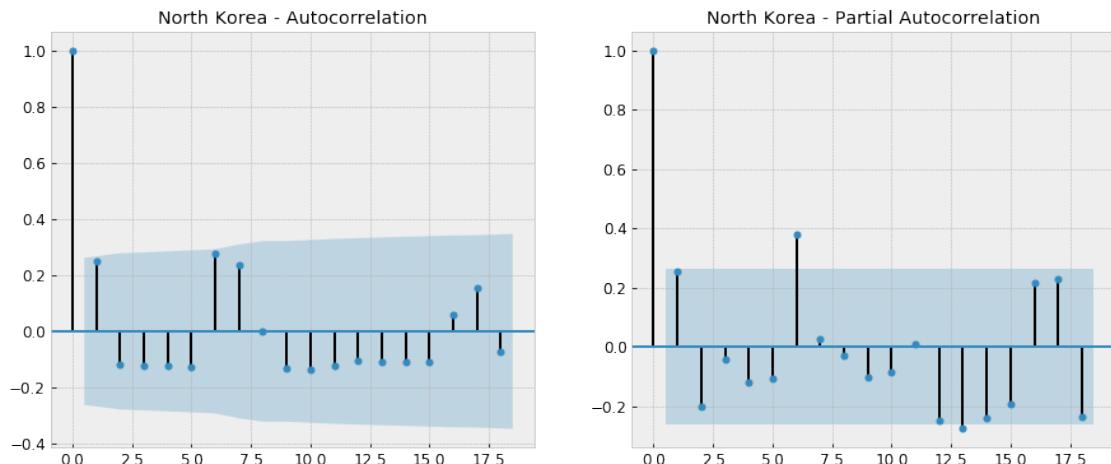


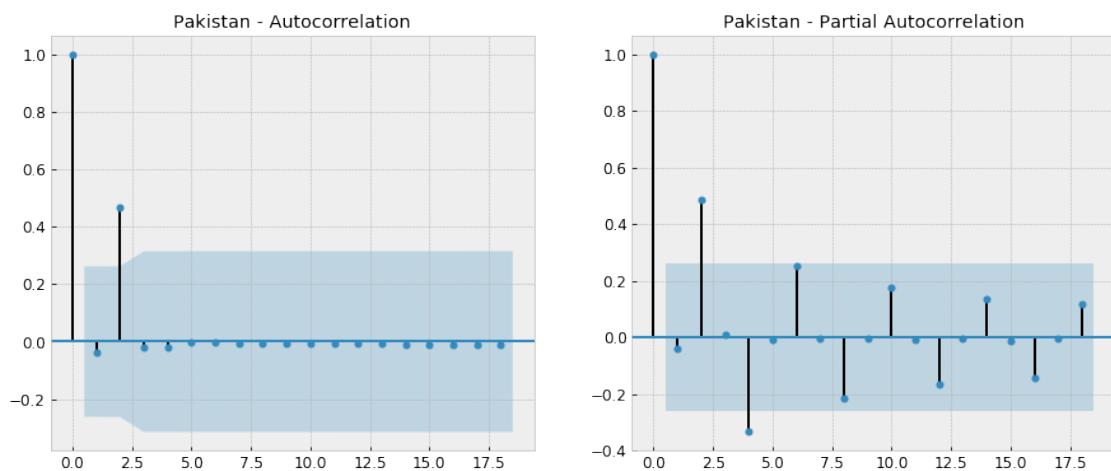
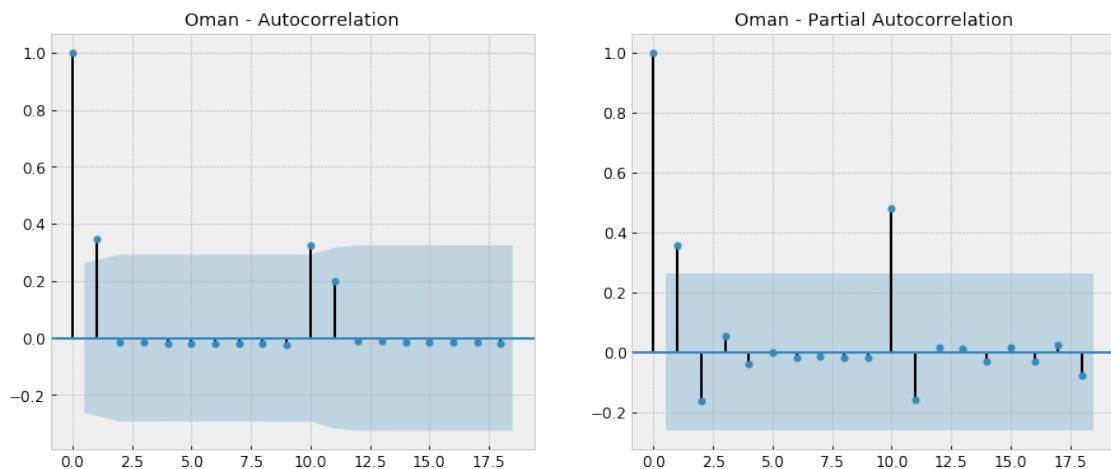
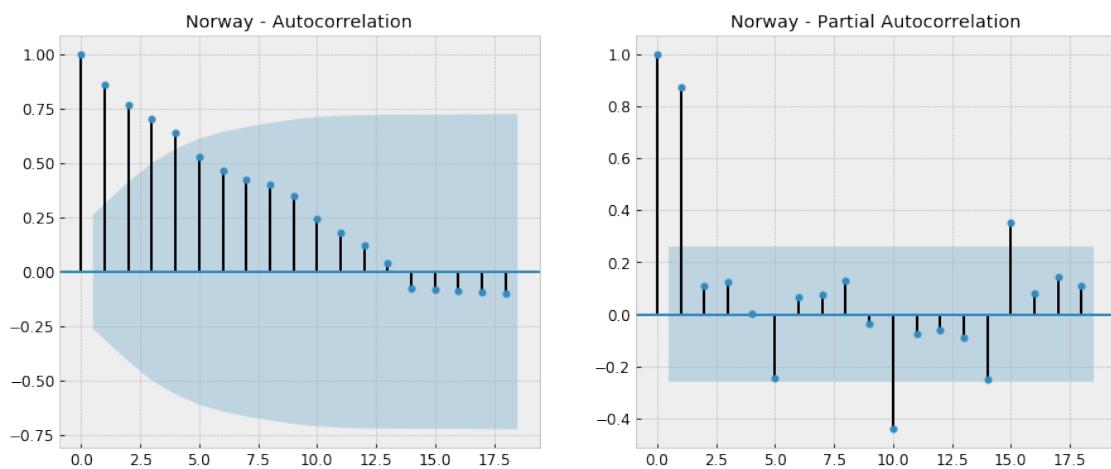


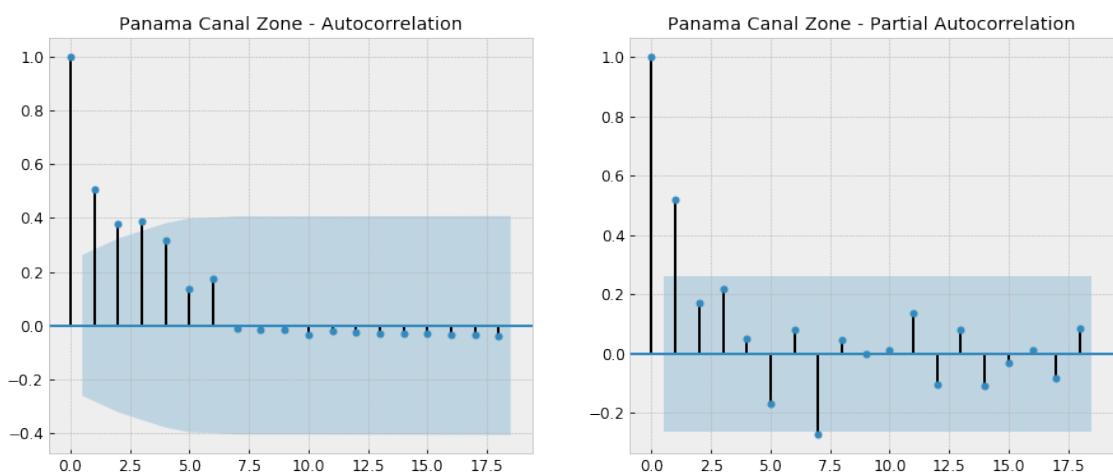
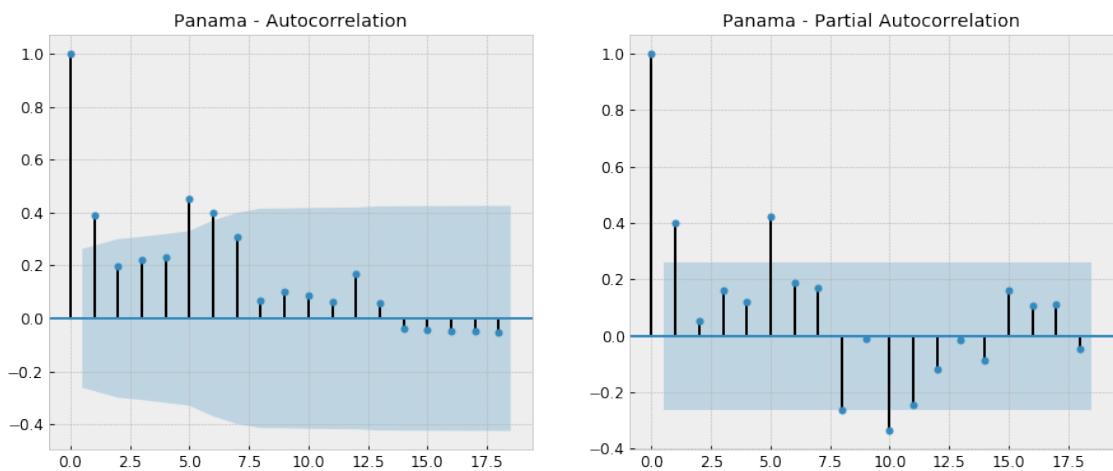


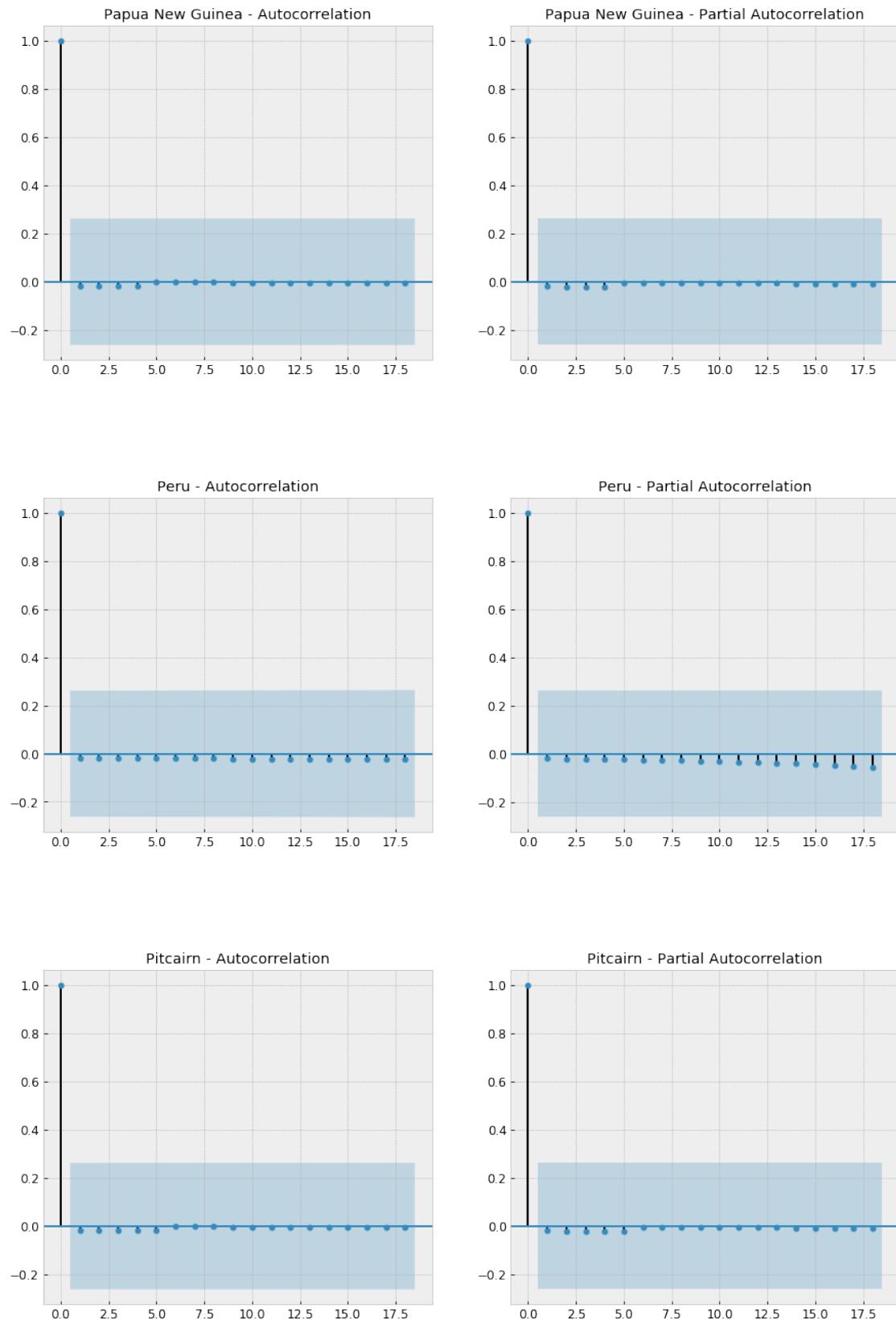


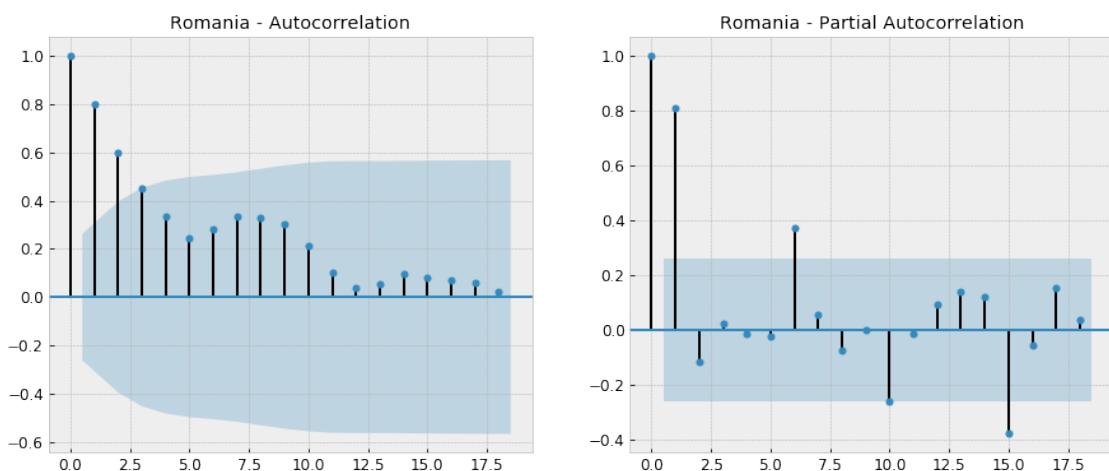
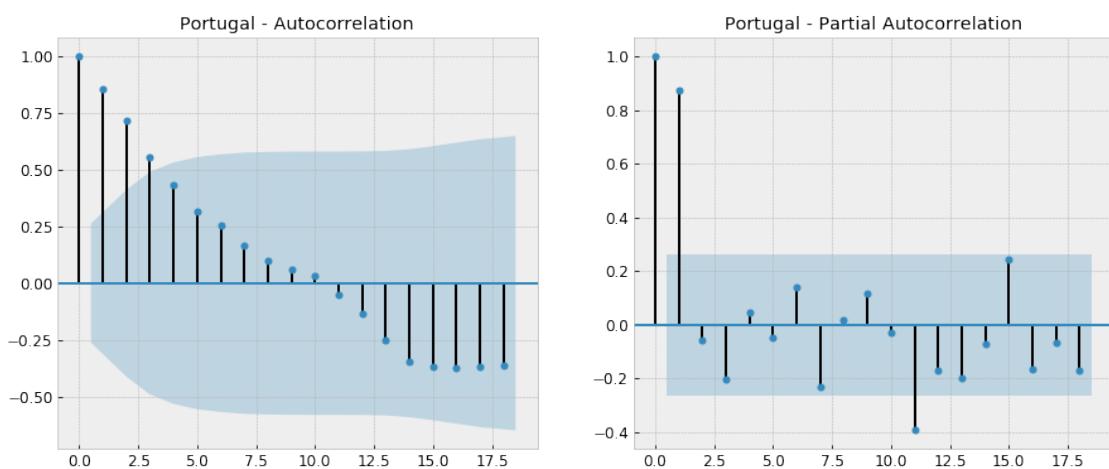
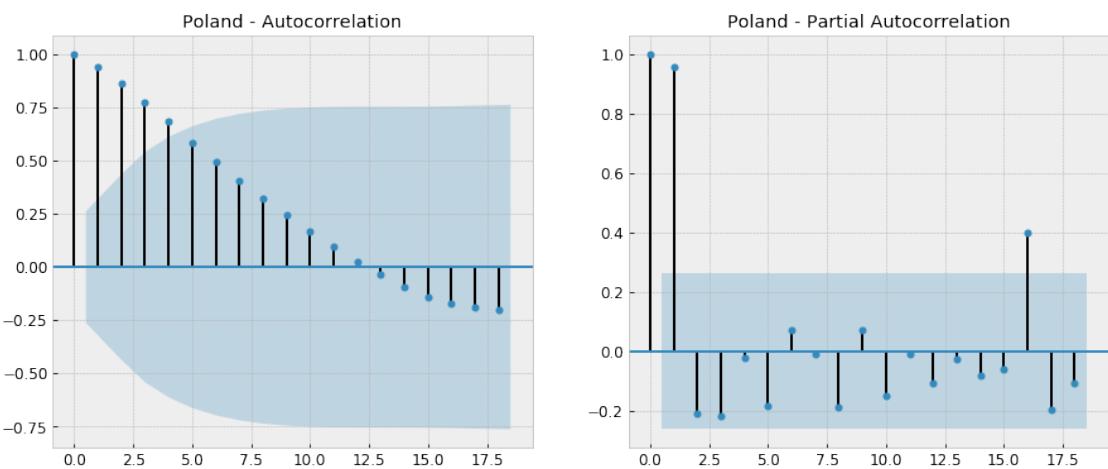


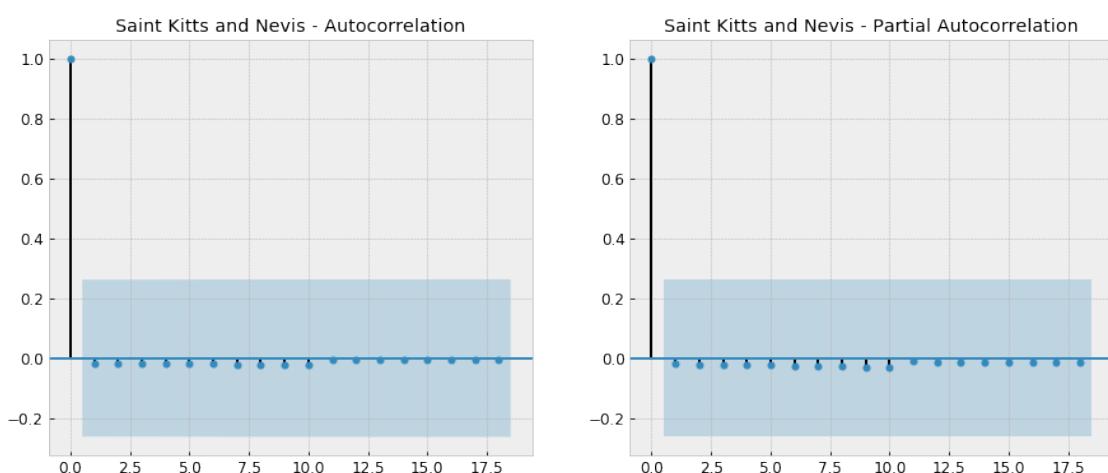
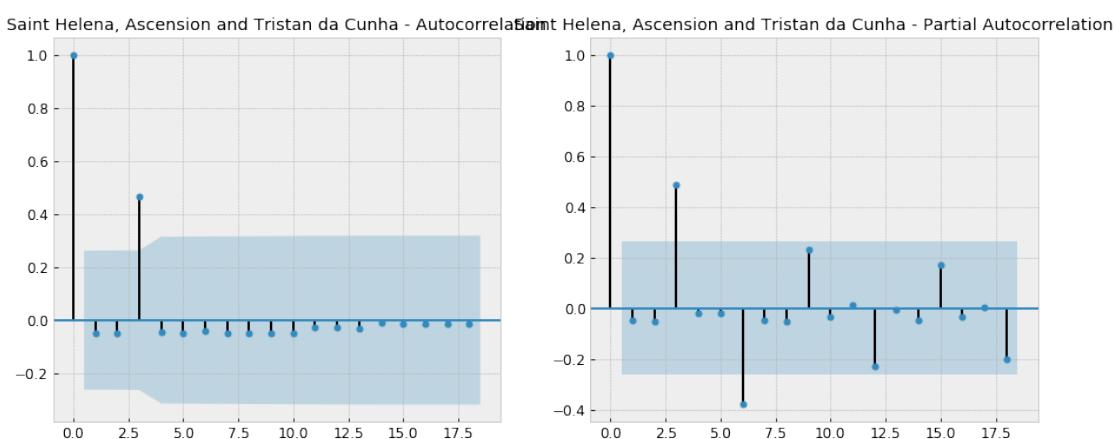
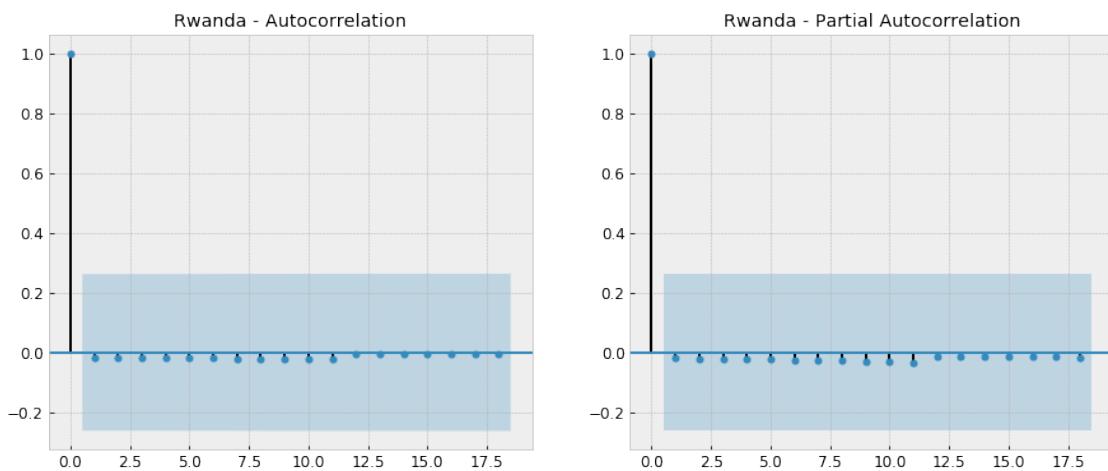


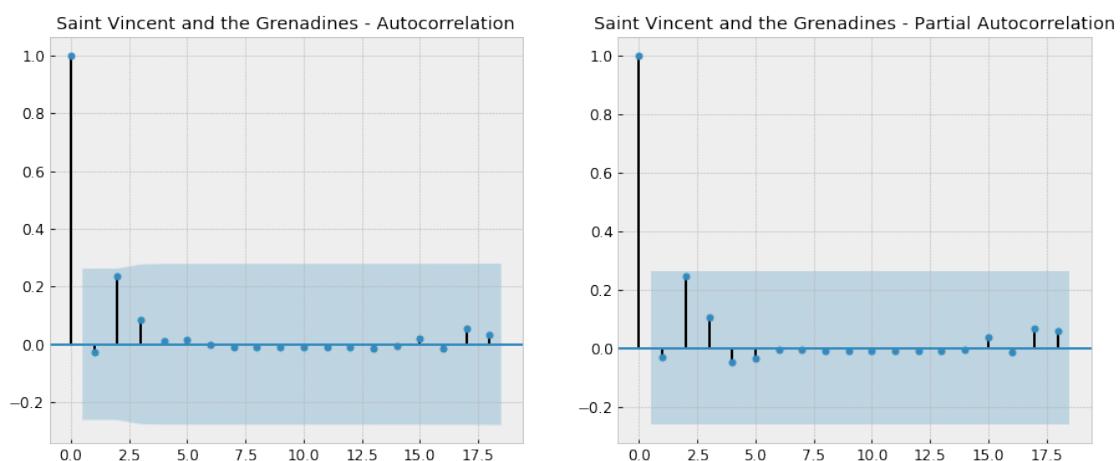
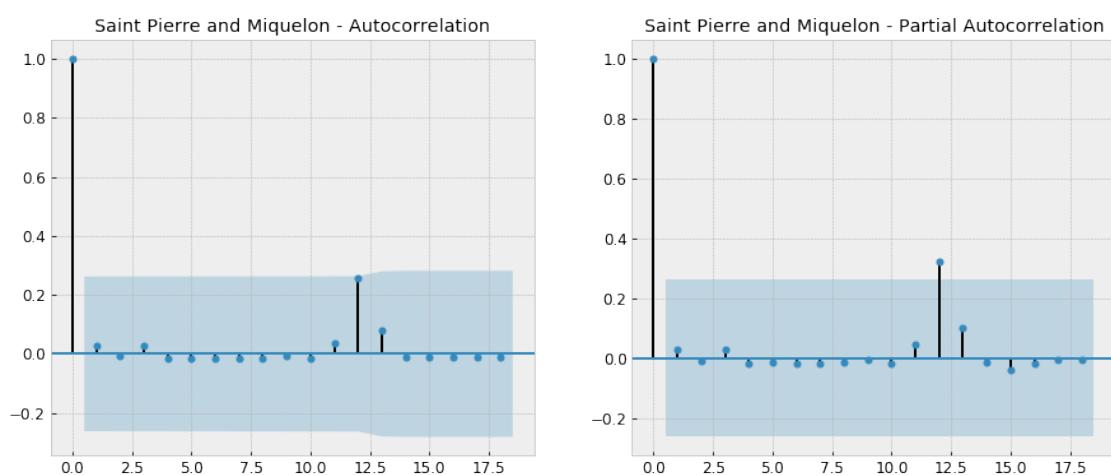
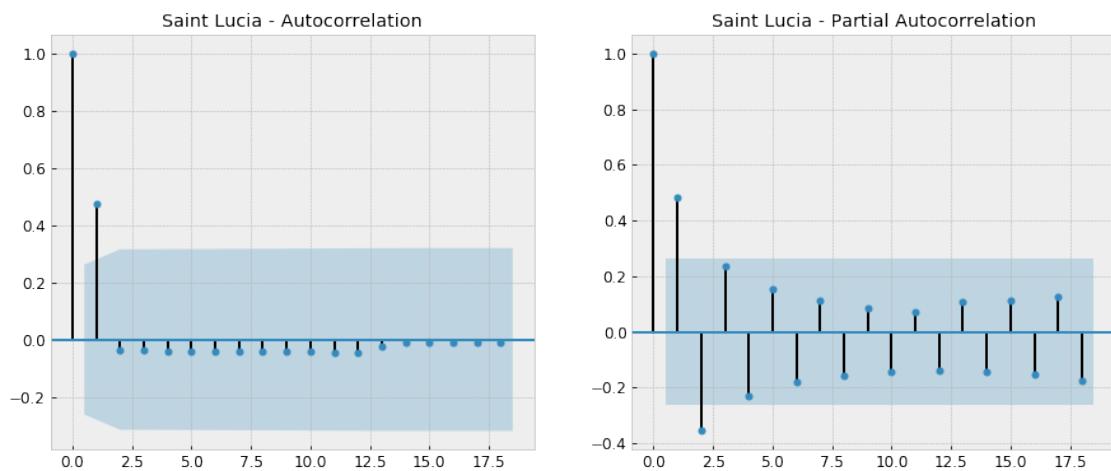


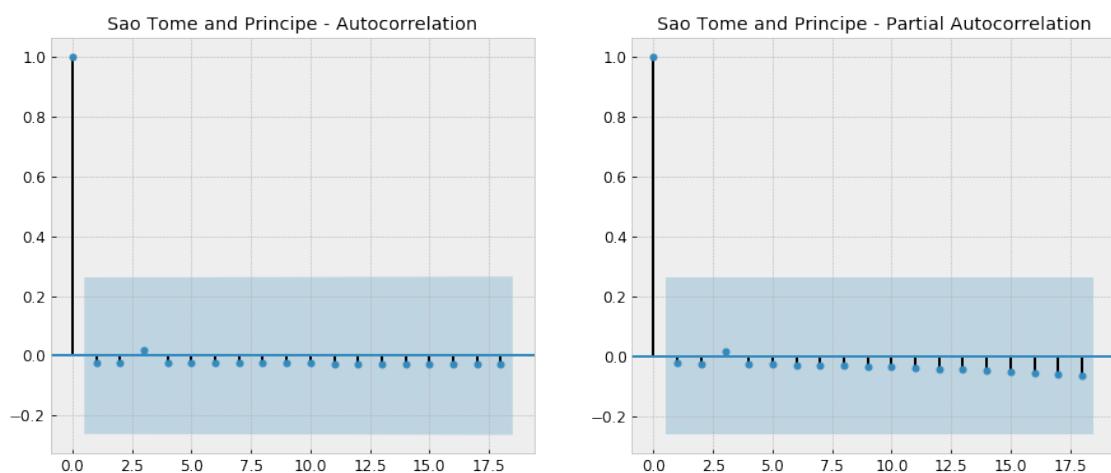
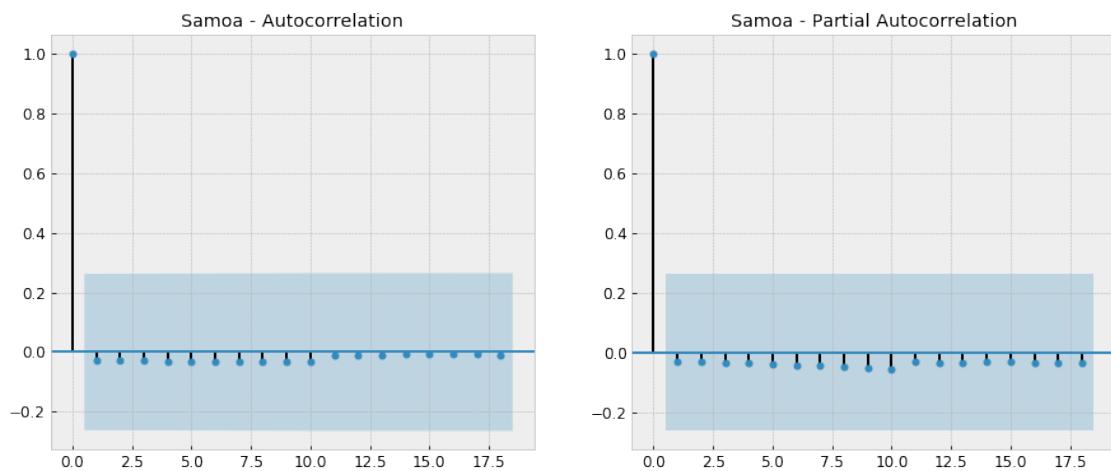


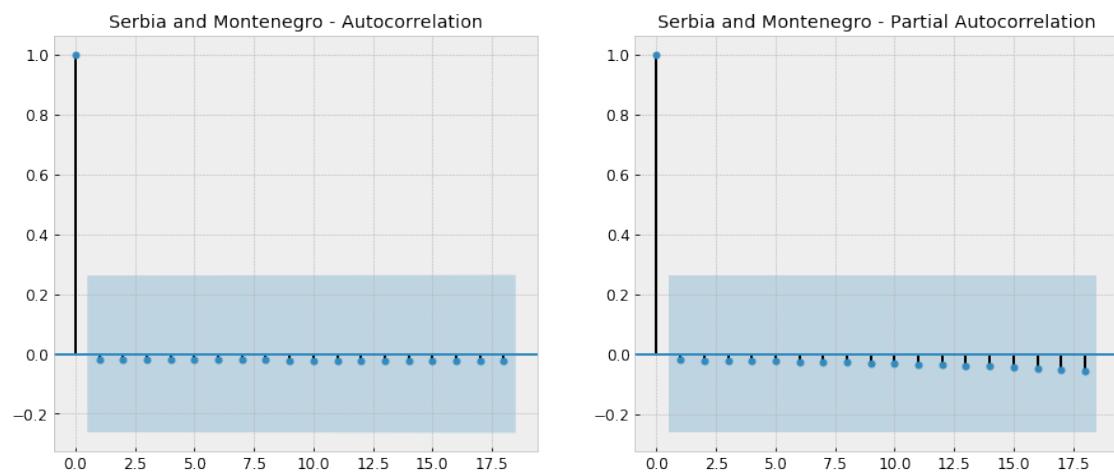
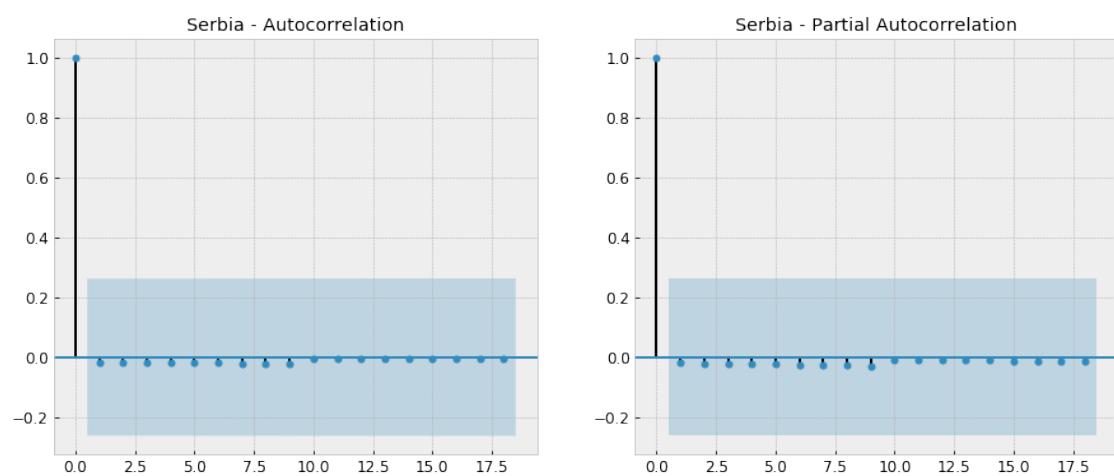
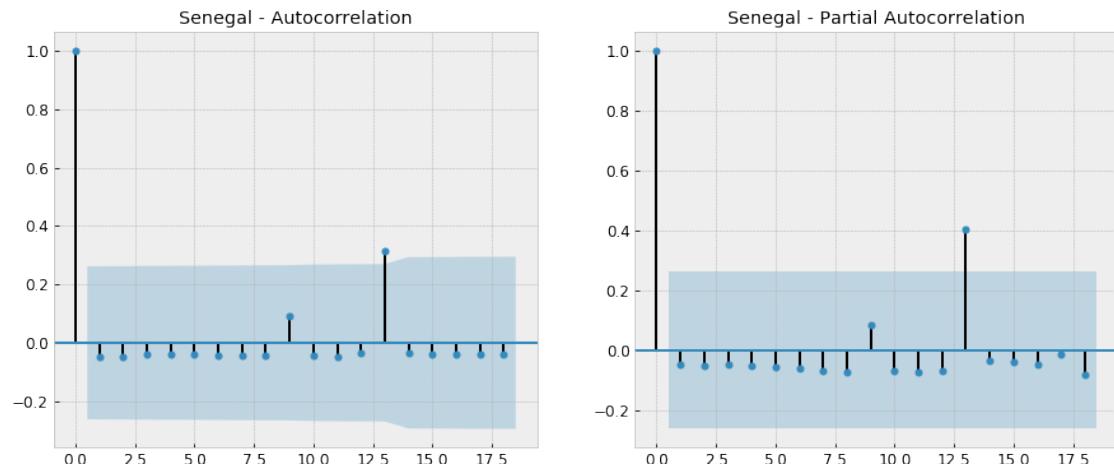


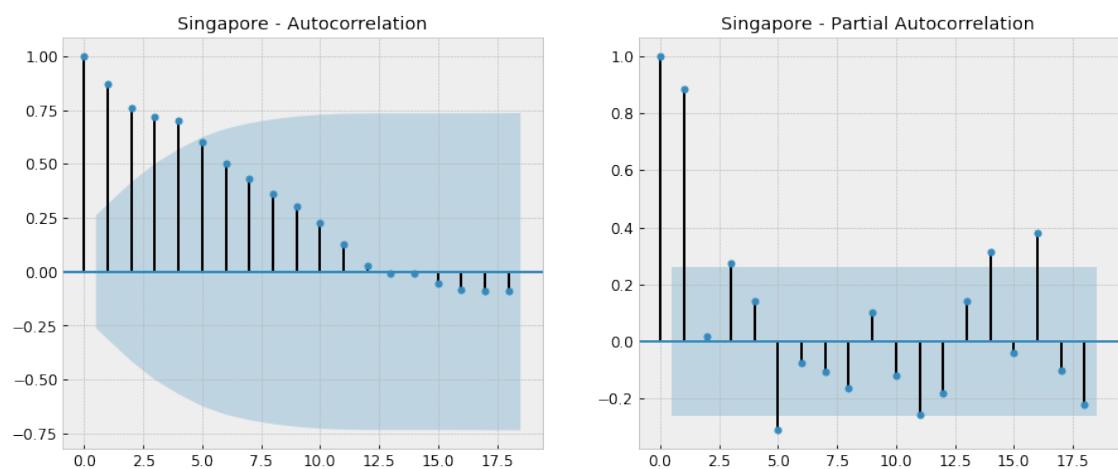
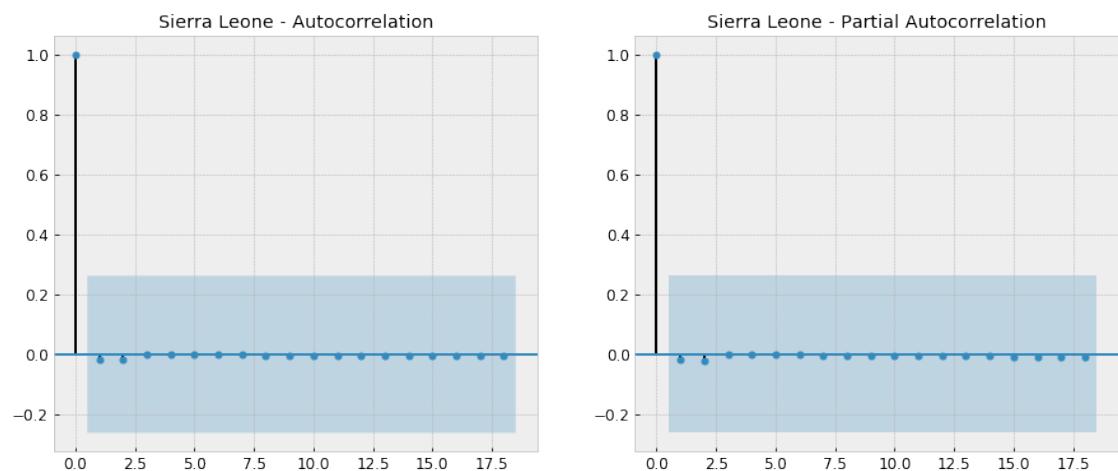
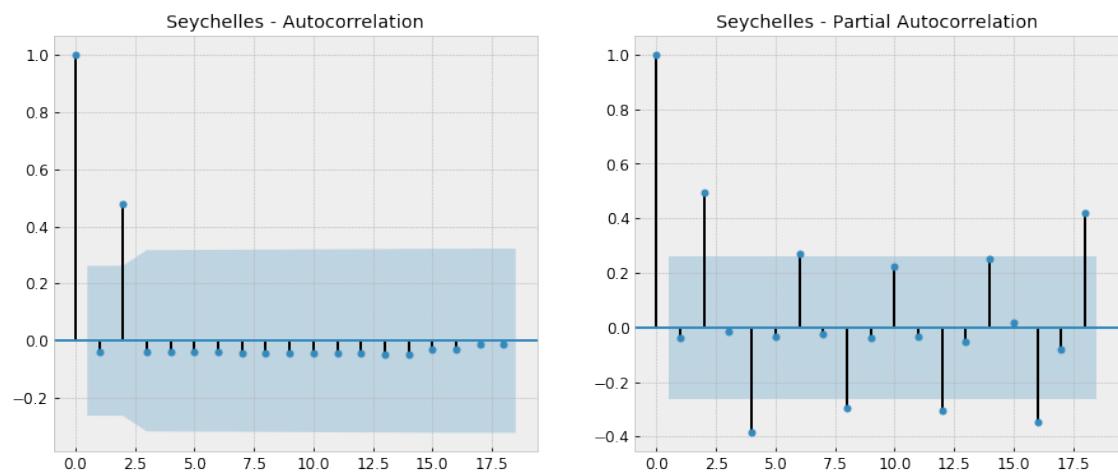


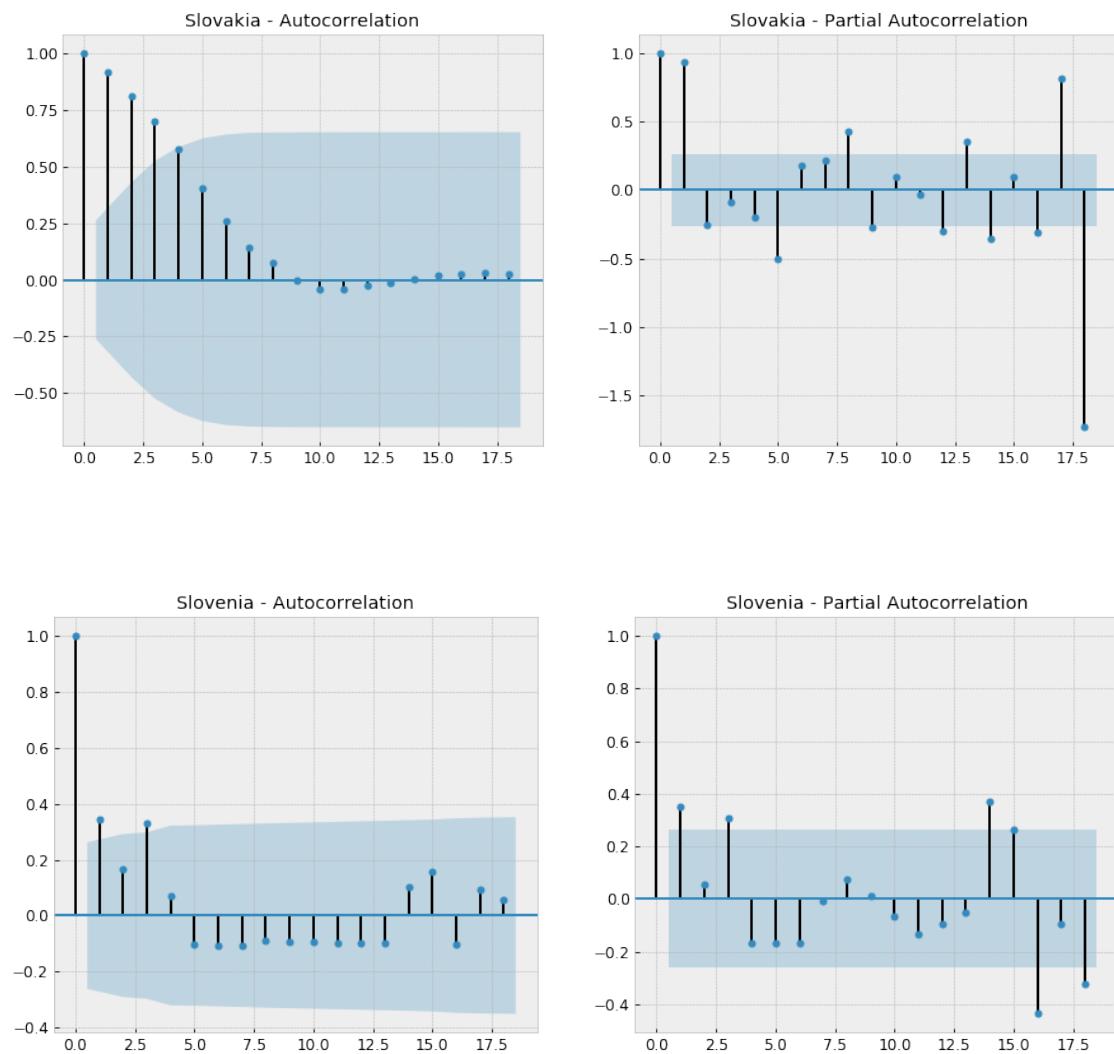


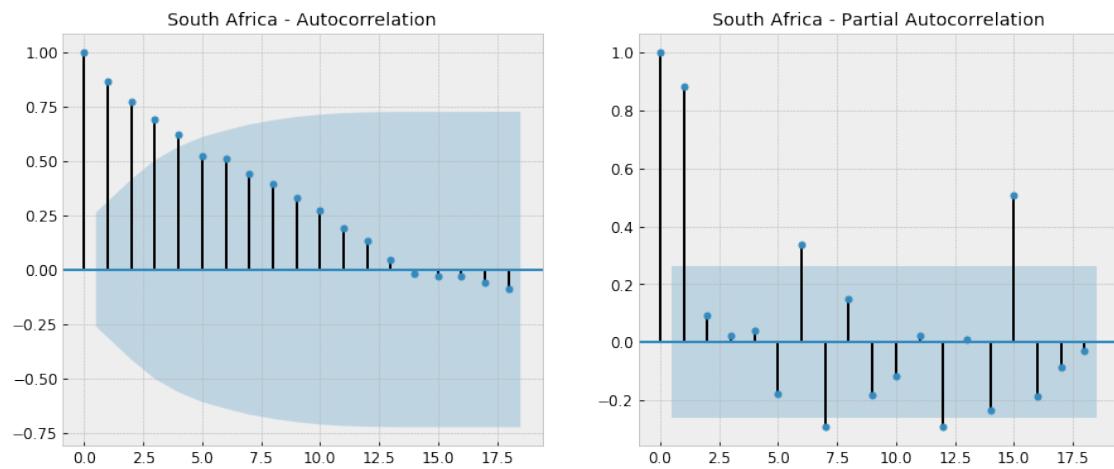
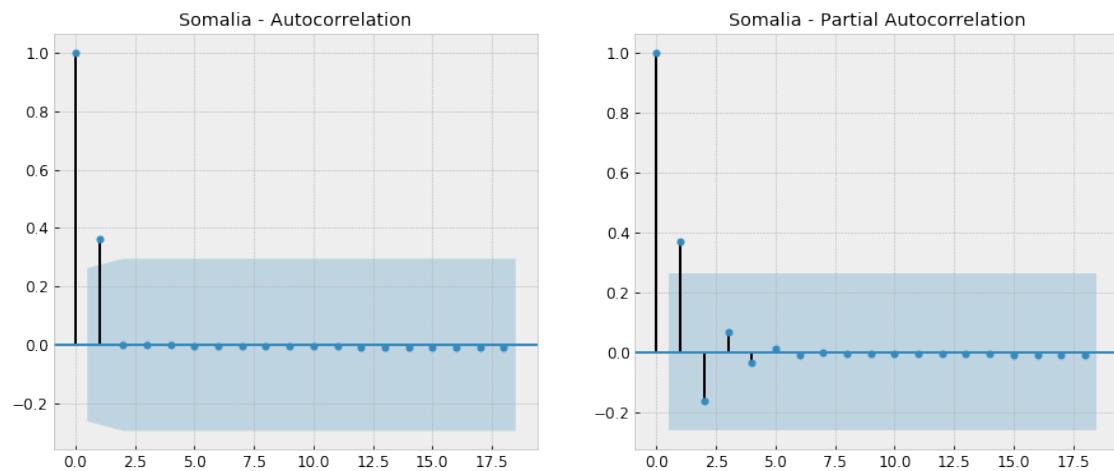
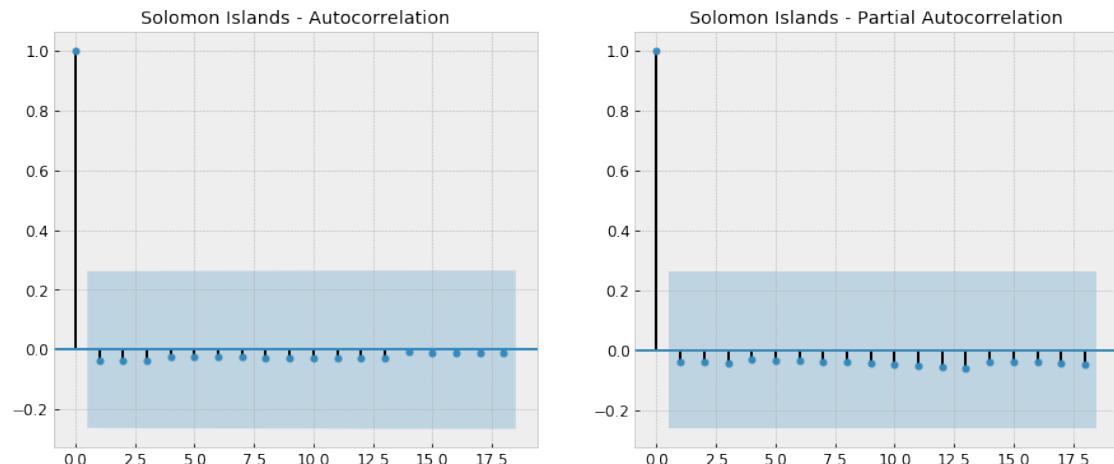


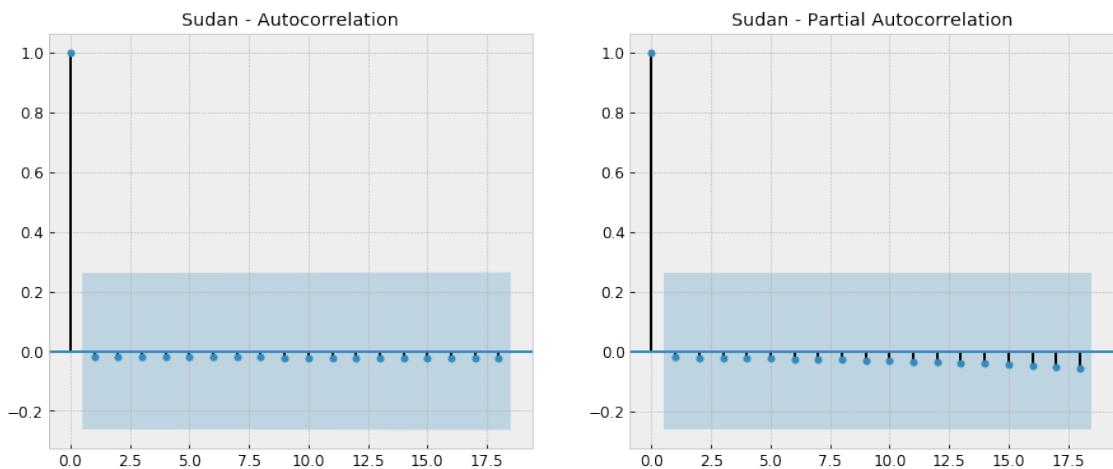
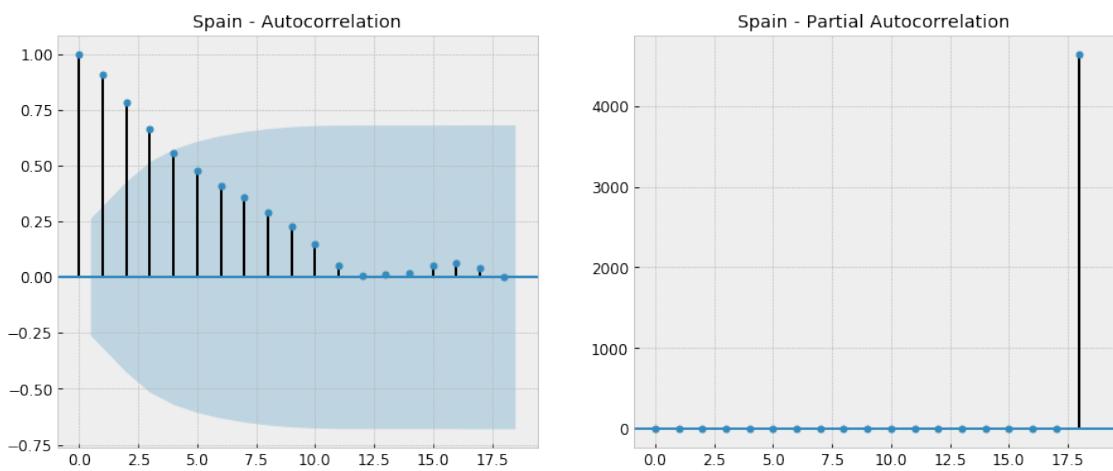
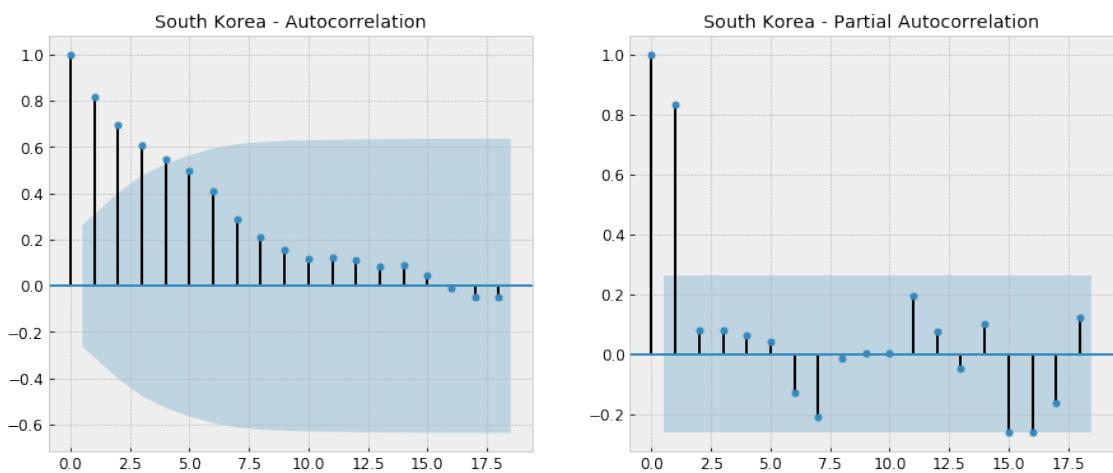


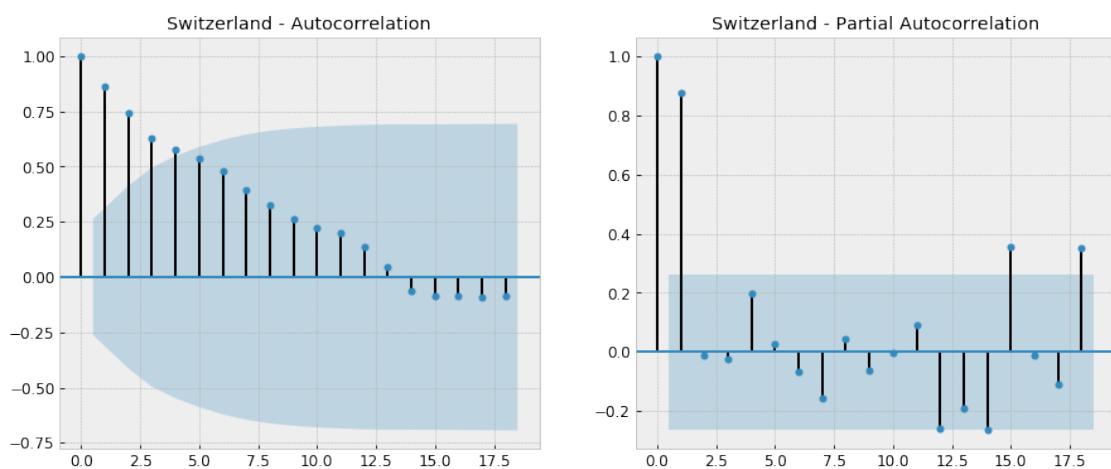
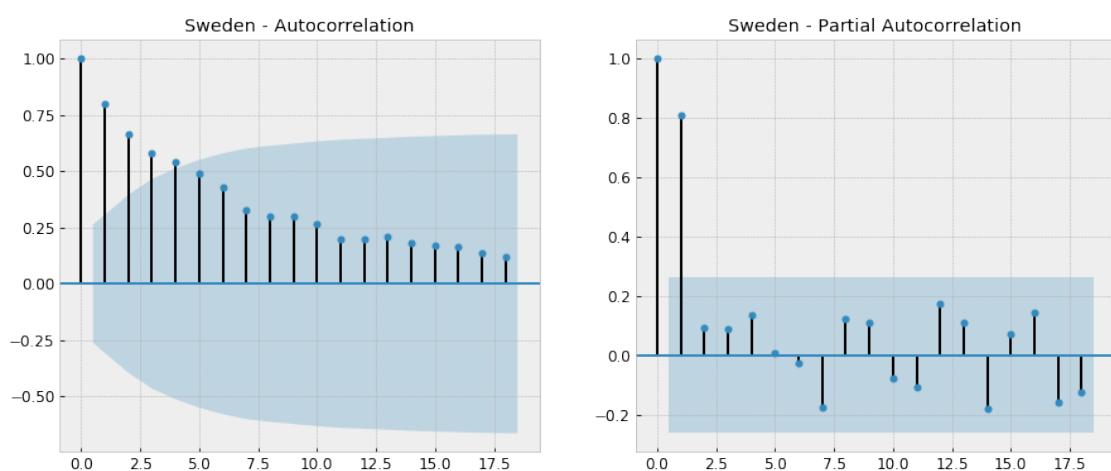
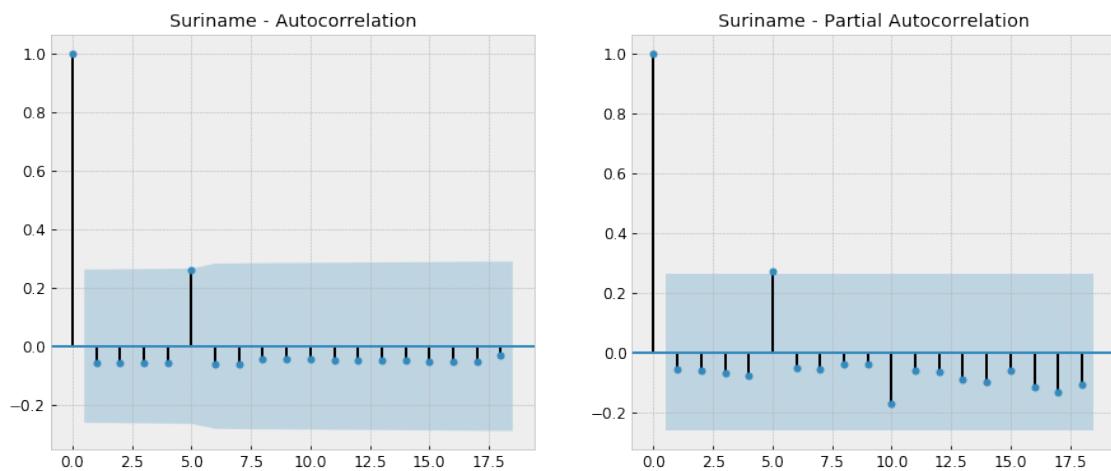


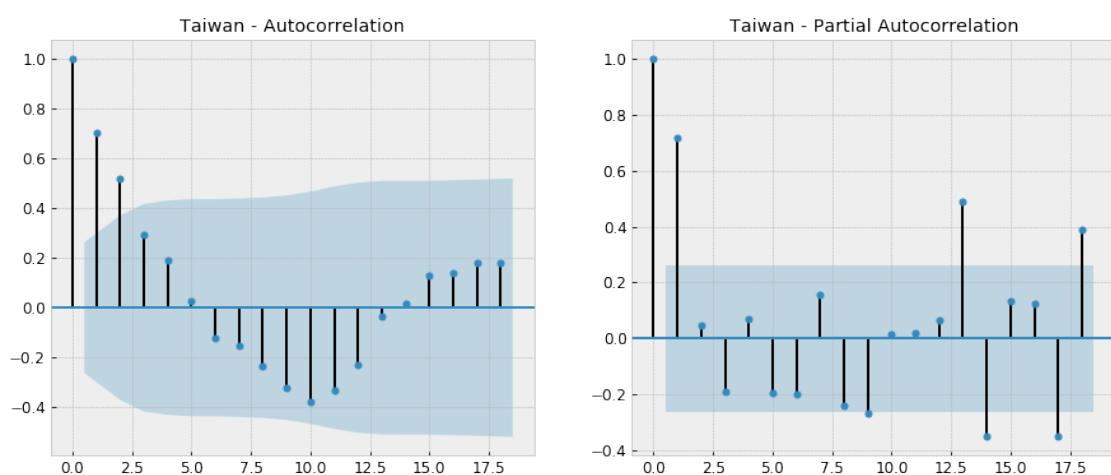
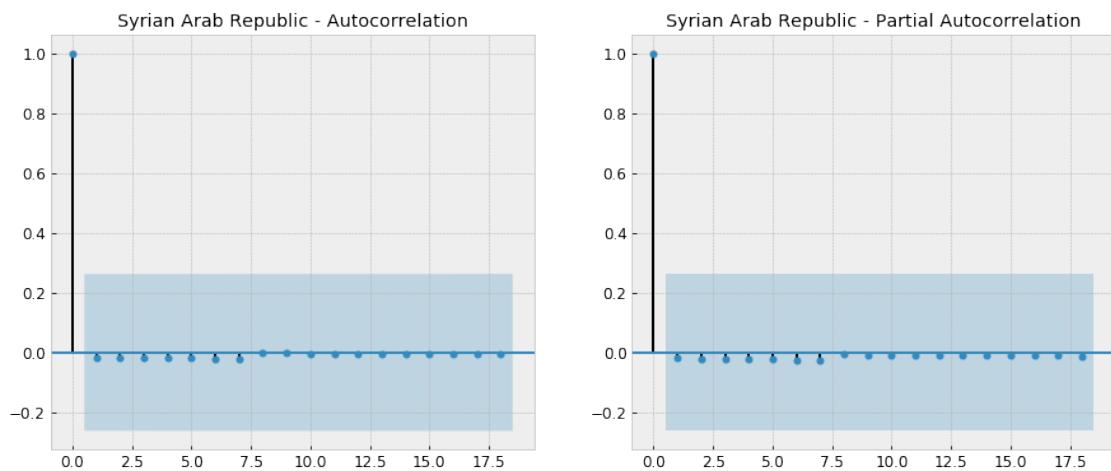


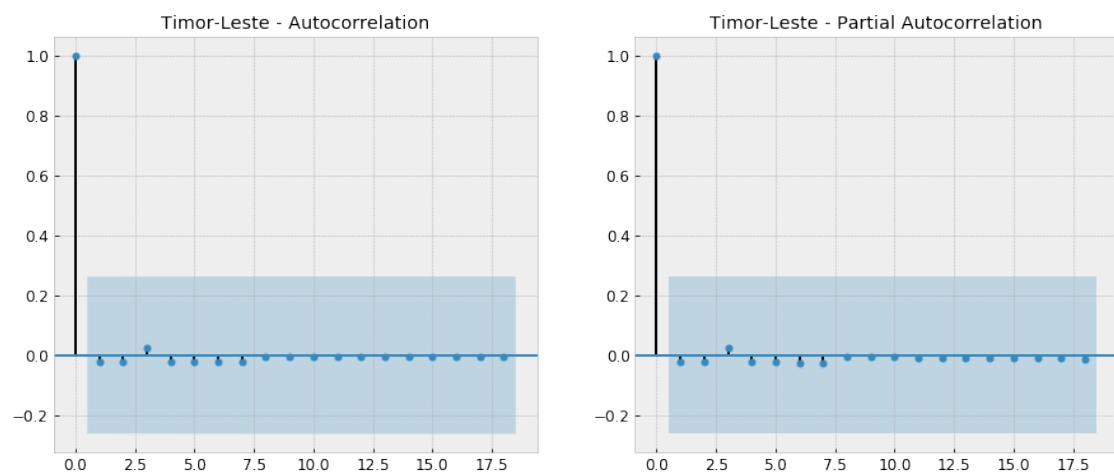
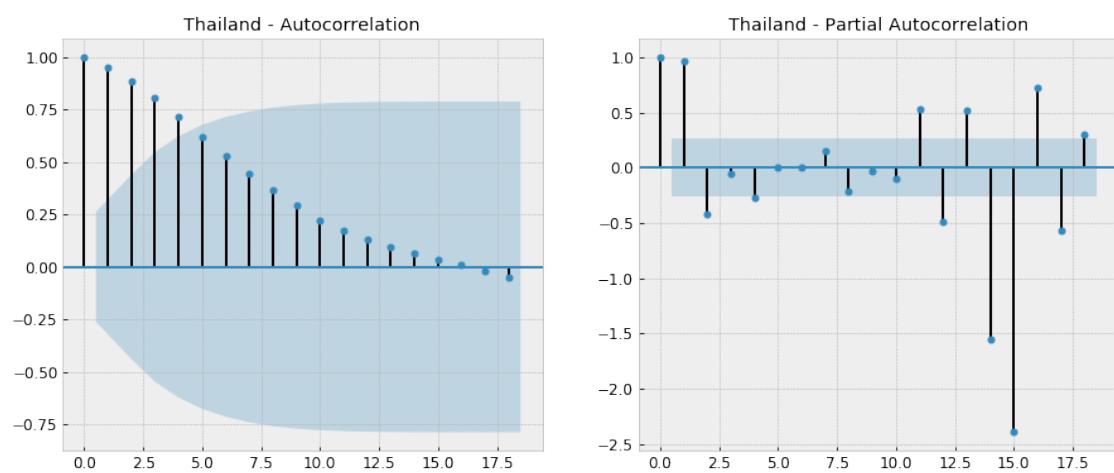
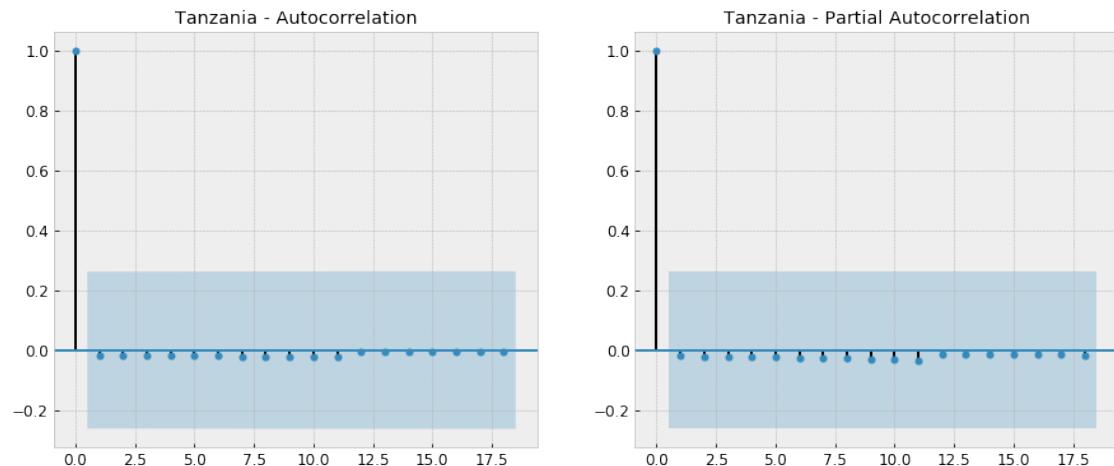


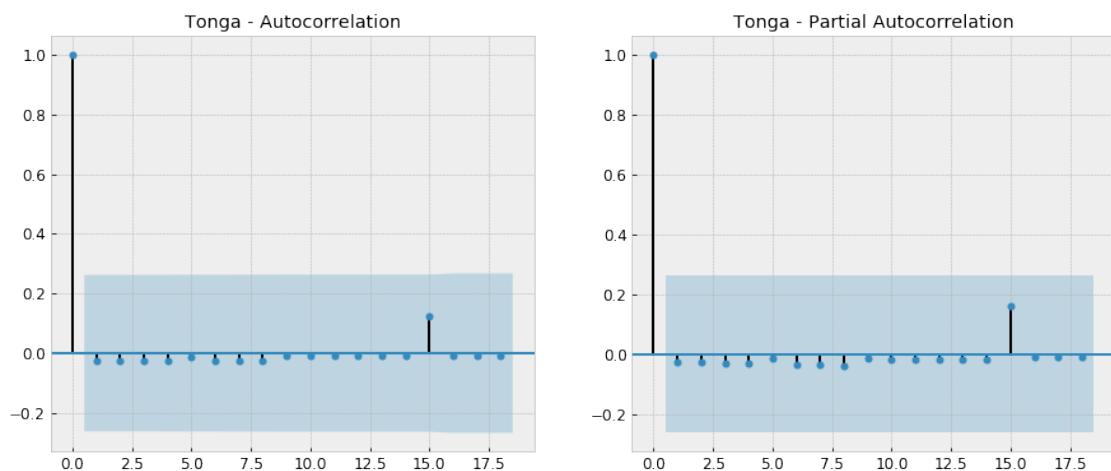
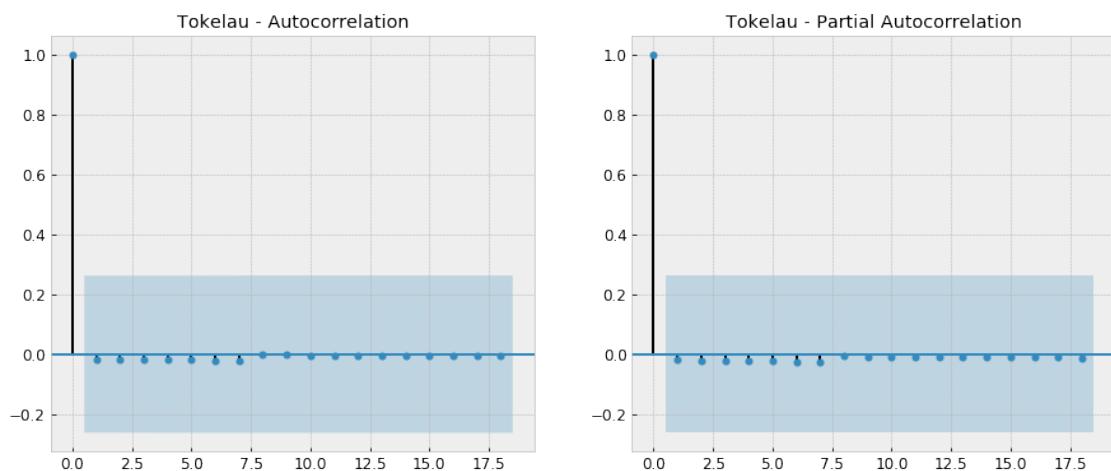
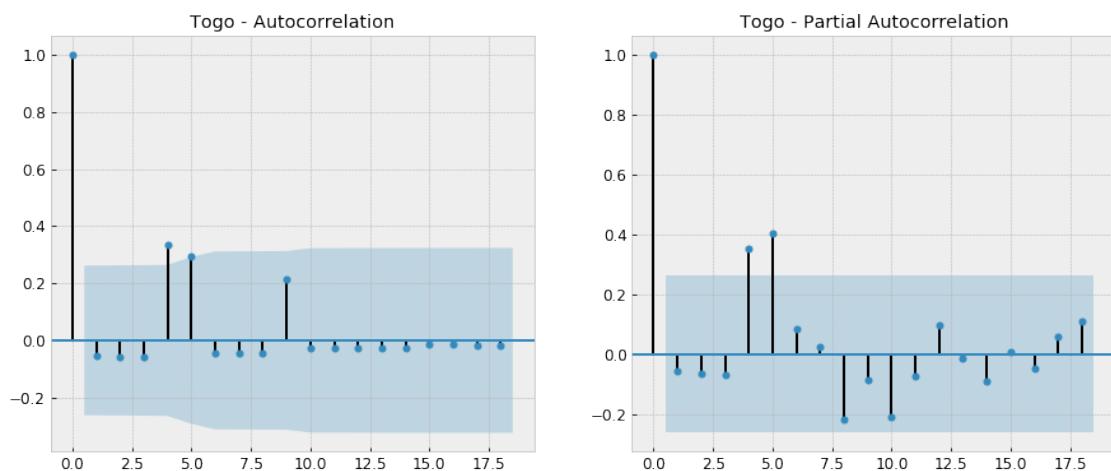


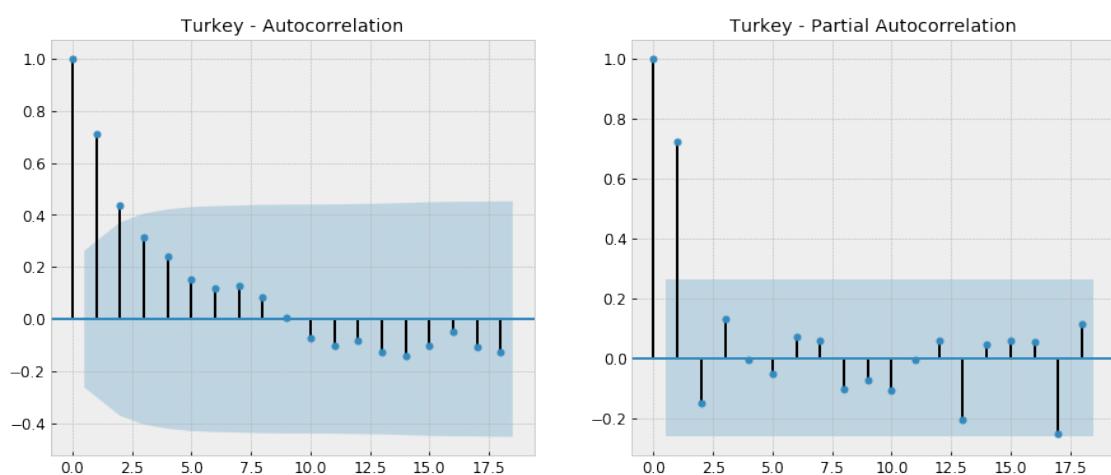
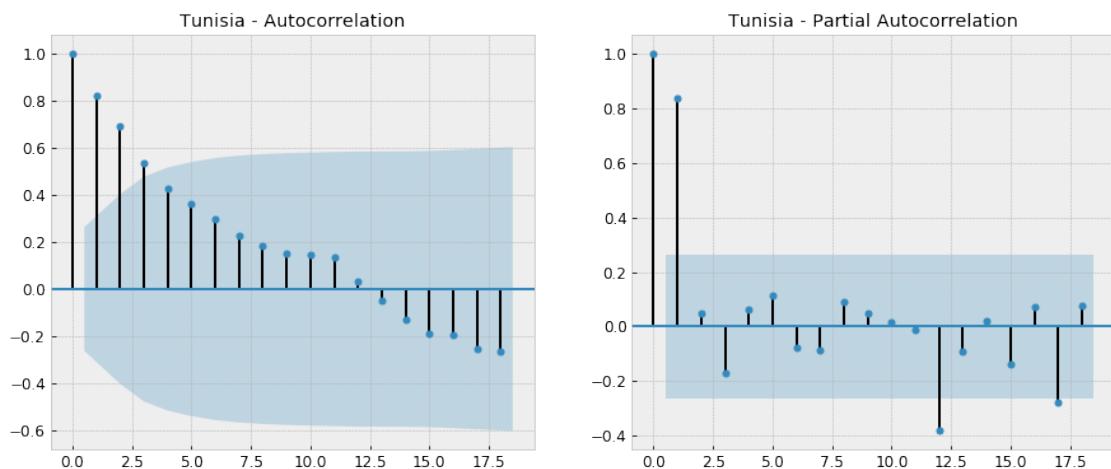


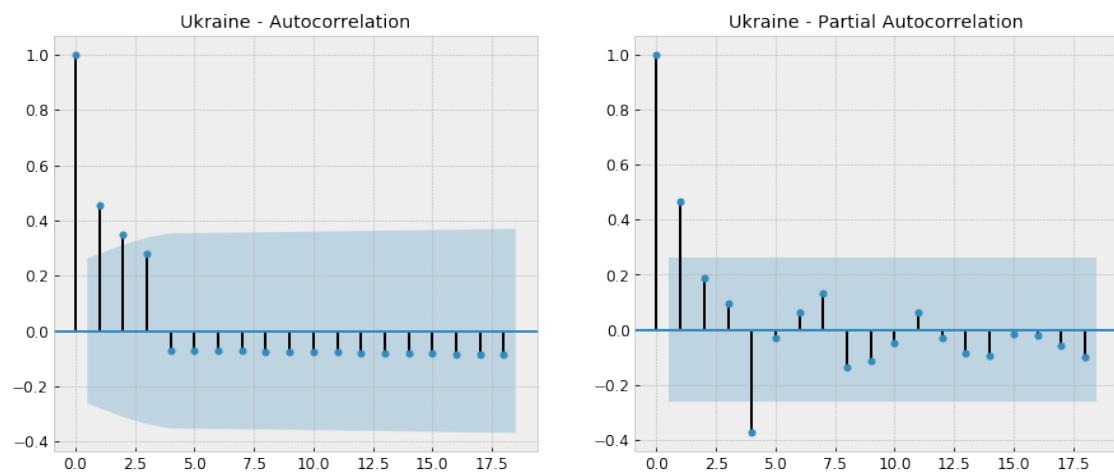
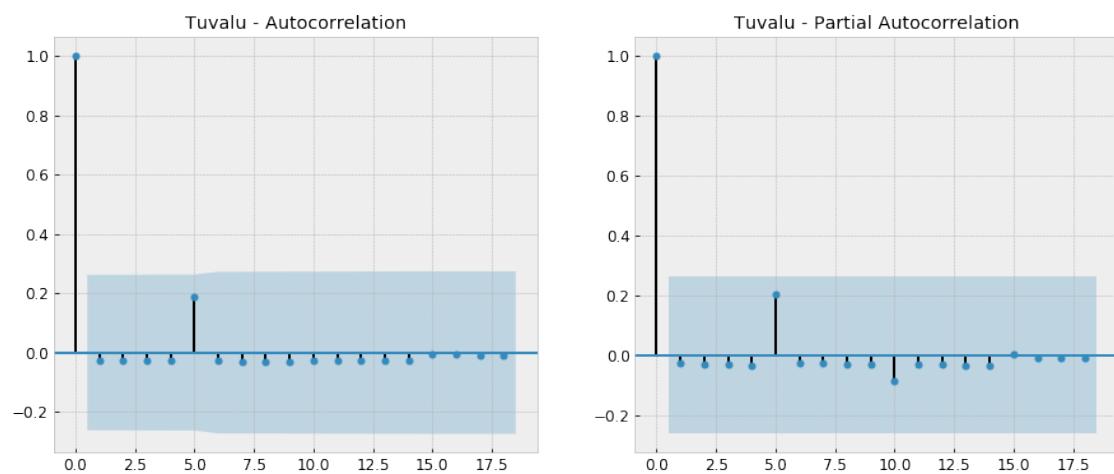
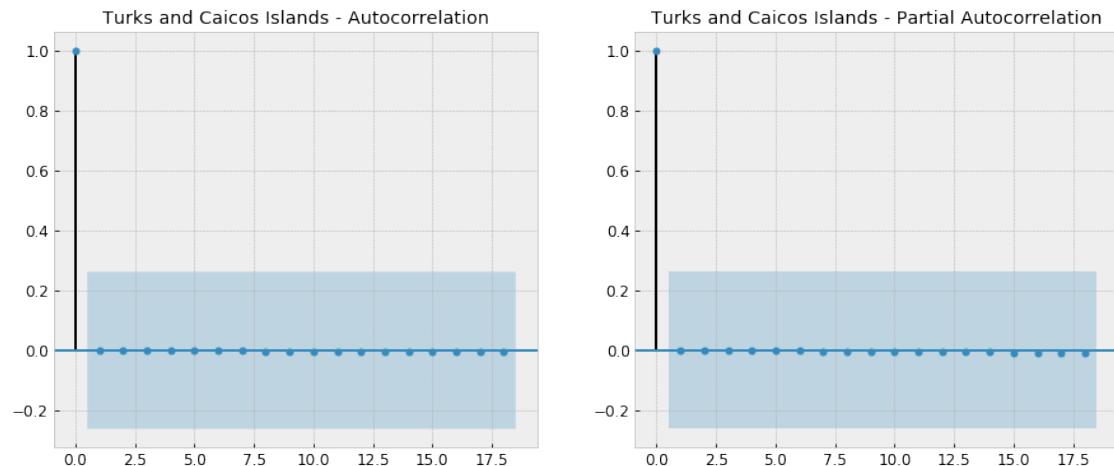


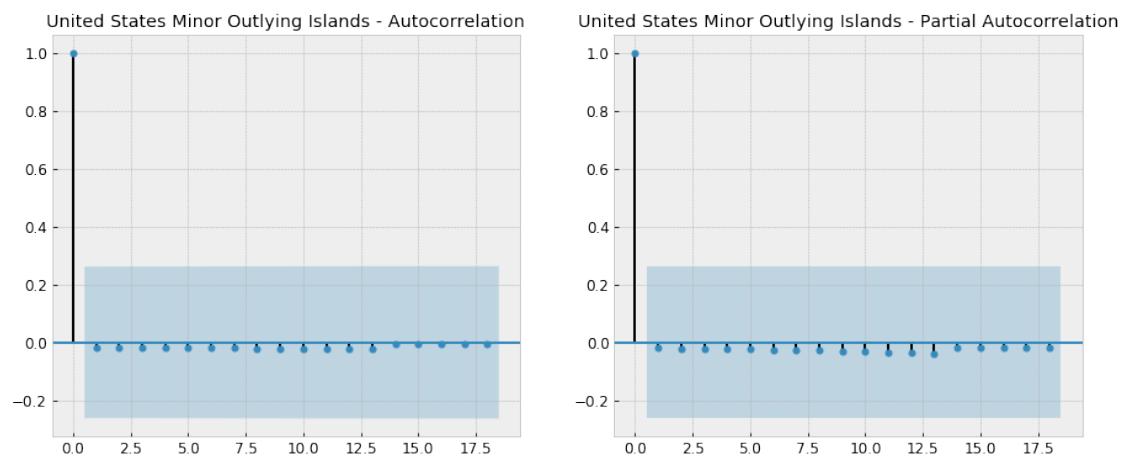
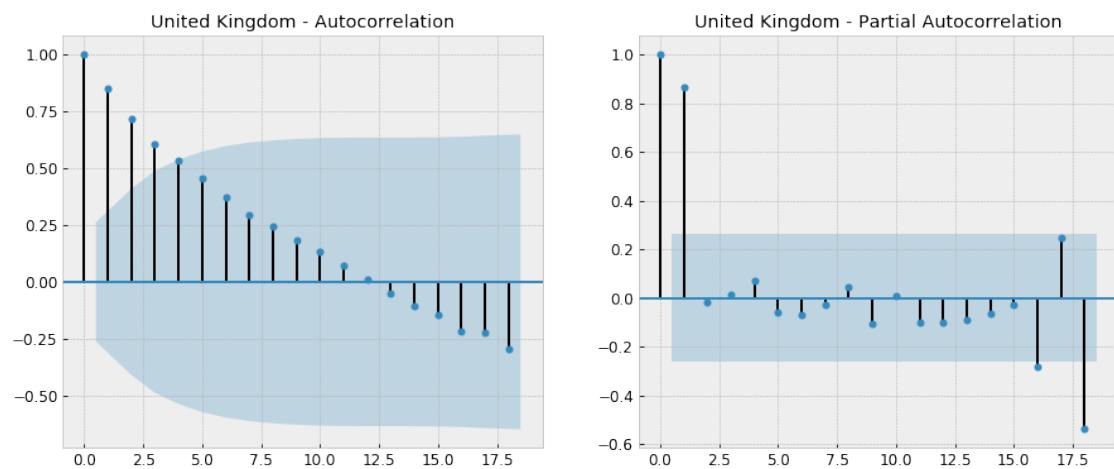
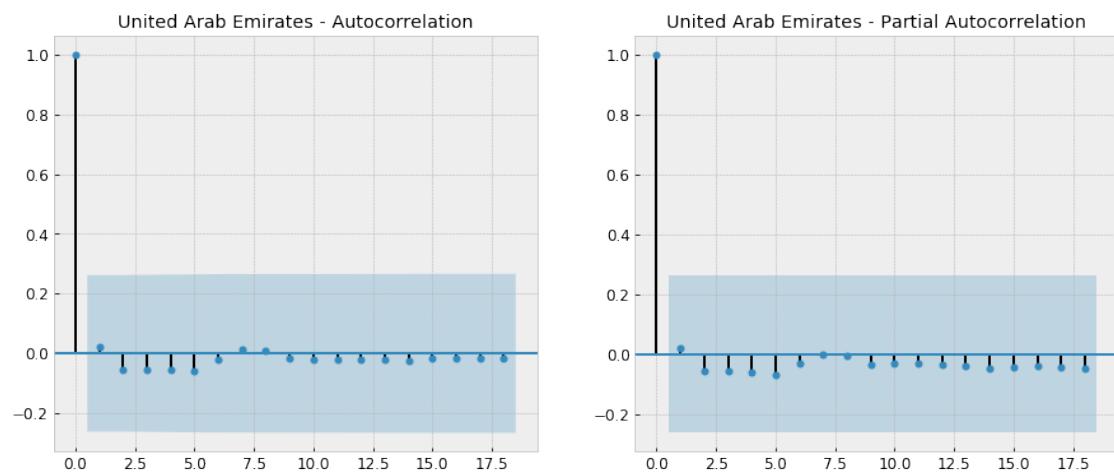


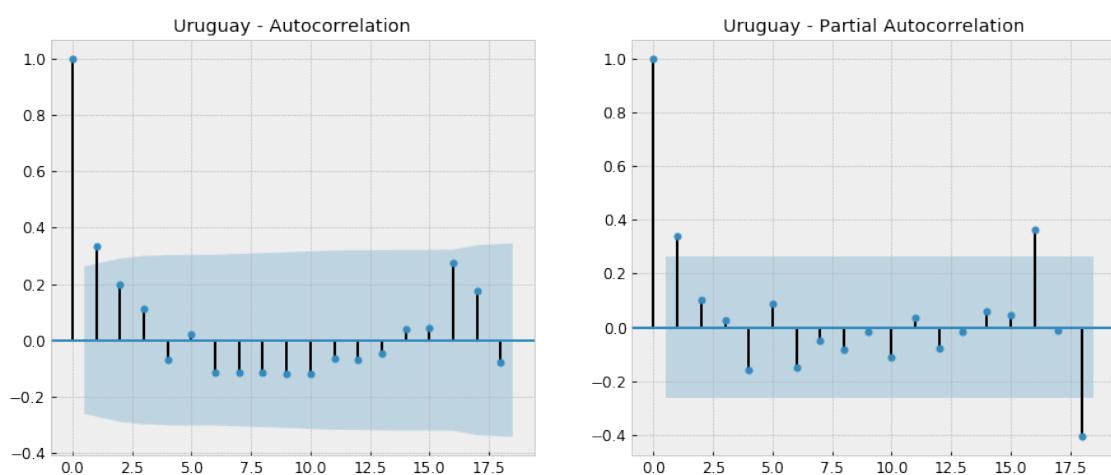
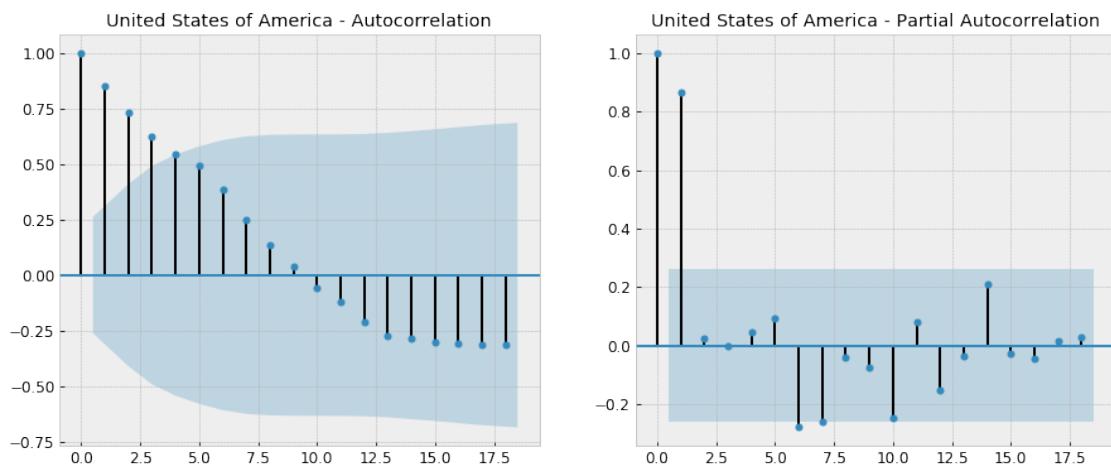


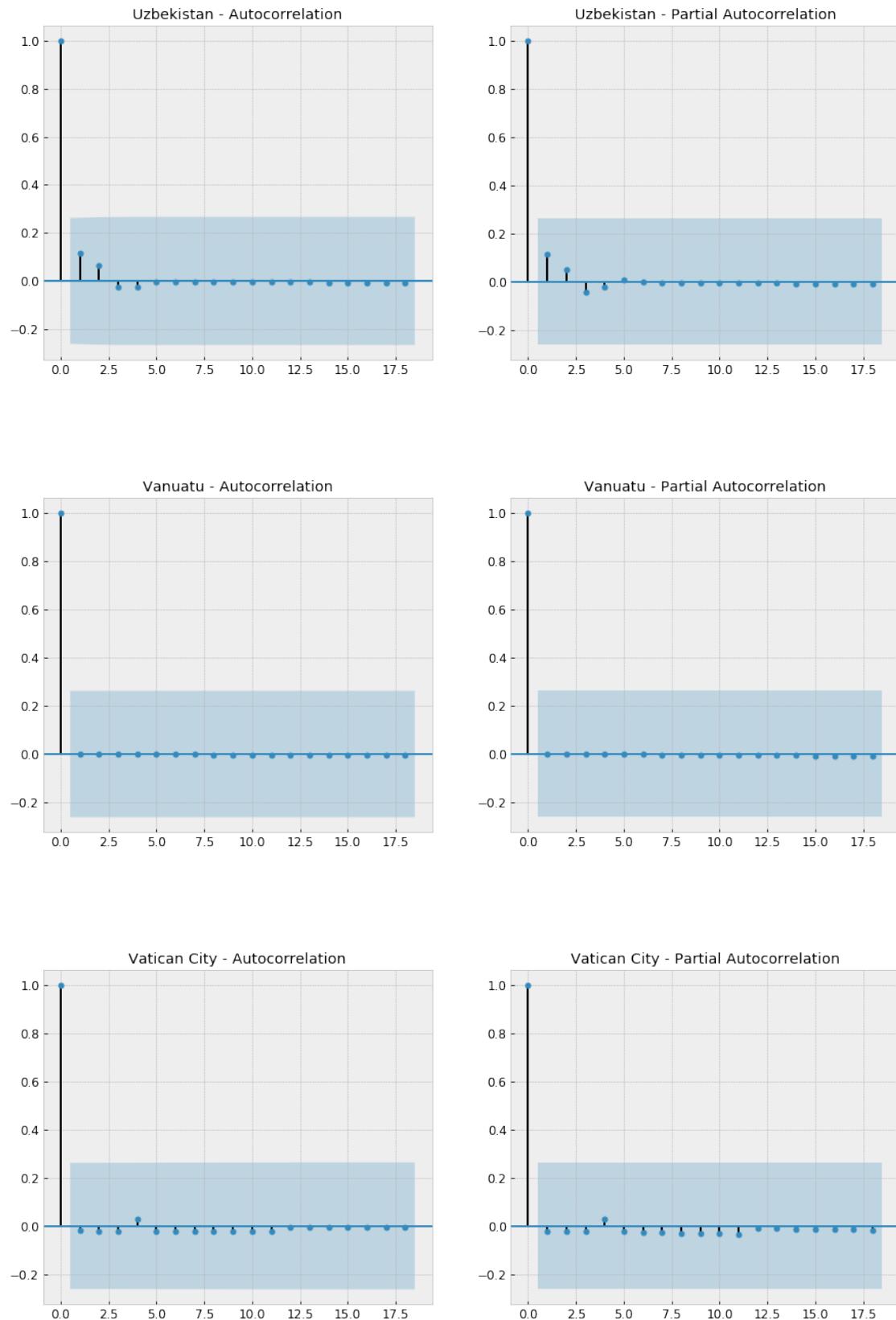


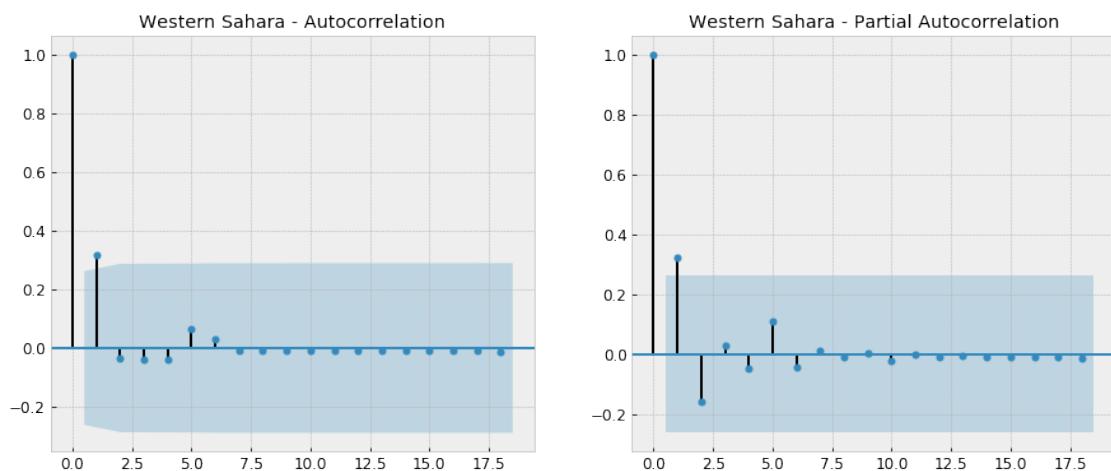
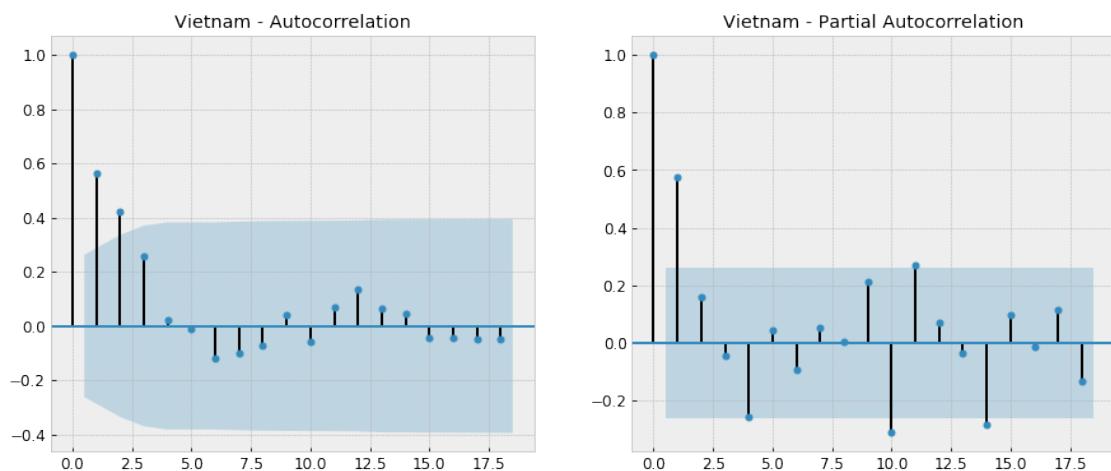
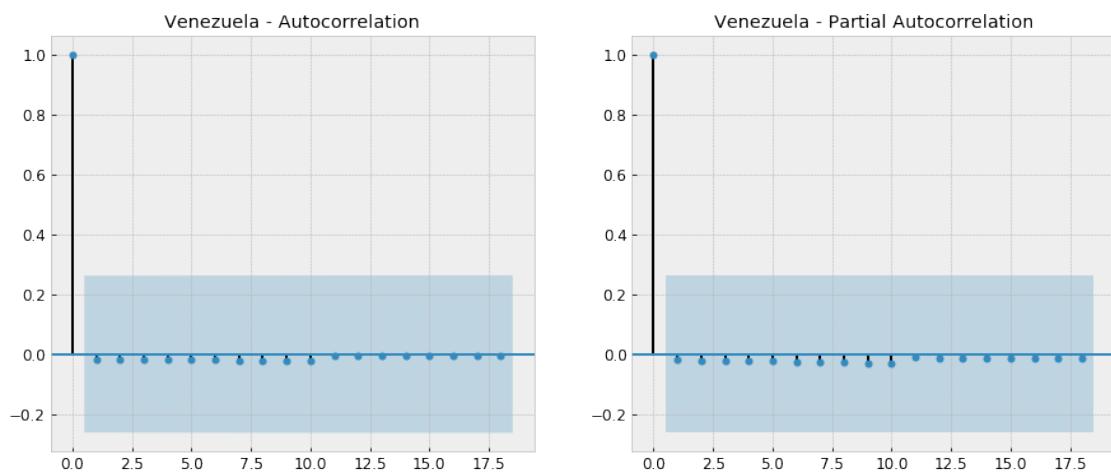


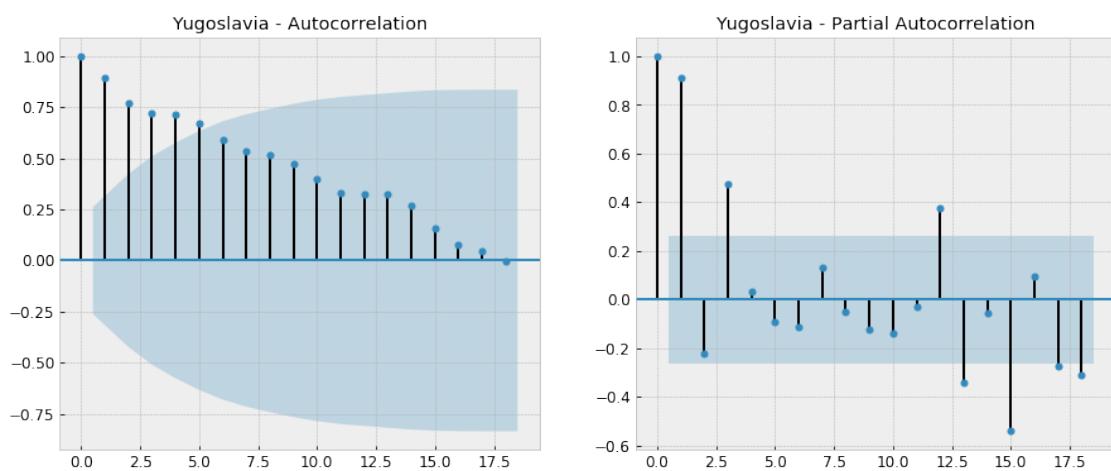
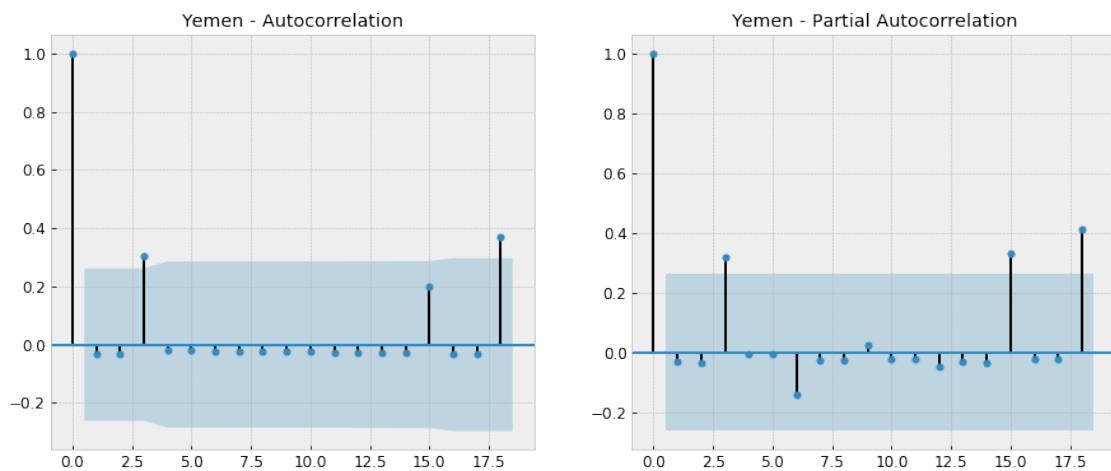


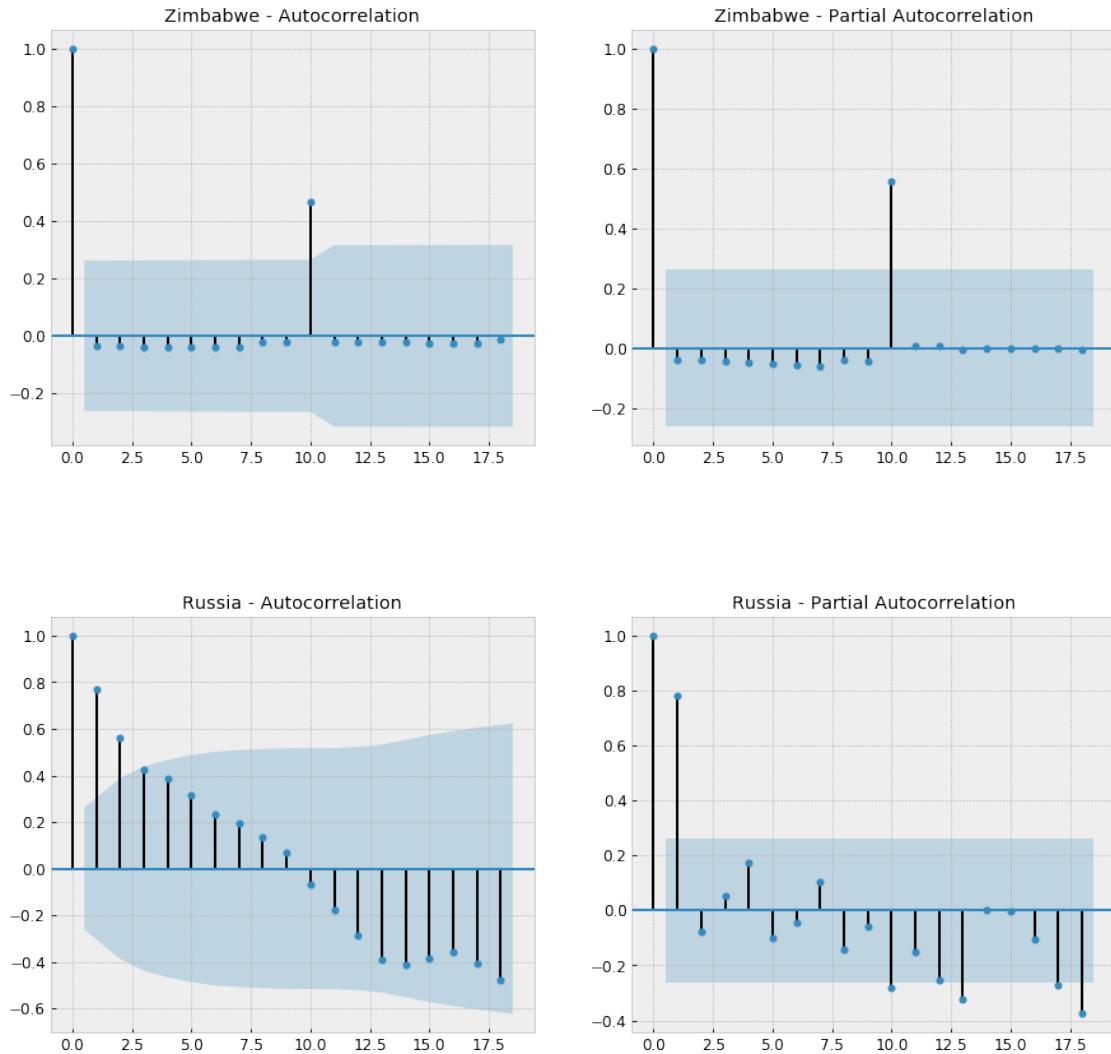












4 TODO

4.1 Interpreting ACF plots

ACF Shape	Indicated Model	Countries fitting description
Exponential, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to identify the order of the autoregressive model	
Alternating positive and negative, decaying to zero Autoregressive model.	Use the partial autocorrelation plot to help identify the order.	

ACF Shape	Indicated Model	Countries fitting description
One or more spikes, rest are essentially zero	Moving average model, order identified by where plot becomes zero.	
Decay, starting after a few lags	Mixed autoregressive and moving average (ARMA) model.	
All zero or close to zero	Data are essentially random.	
High values at fixed intervals	Include seasonal autoregressive term.	
No decay to zero	Series is not stationary	

4.1.1 Plotting Rolling Statistics

- The rolling mean and standard deviation are not constant with respect to time (increasing trend)
- The time series is hence not stationary

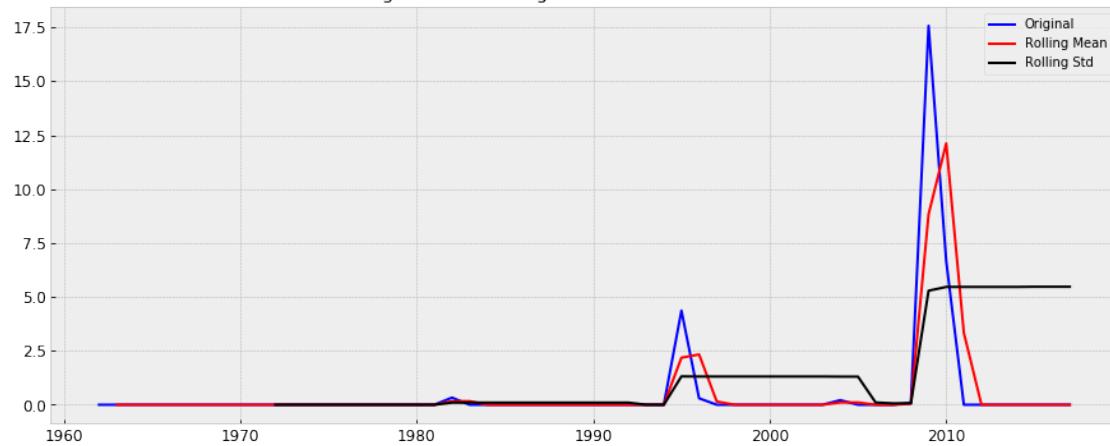
```
[149]: for country in countries:
    savefile = save_images(df_grps, 'rolling_stats')
    cycles = int((len(df_grps[country]))/5)
    plt.figure(figsize=(15,6))

    rolmean = df_grps[country].rolling(2).mean()
    rolstd = df_grps[country].rolling(cycles).std()

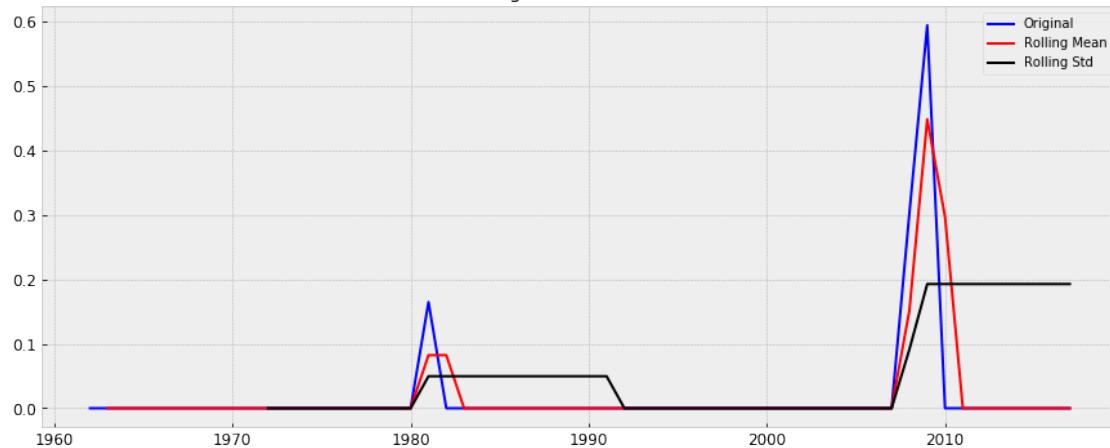
    #Plot rolling statistics:
    orig = plt.plot(df_grps[country], color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')

    plt.legend(loc='best')
    plt.title('{}: Rolling Mean & Standard Deviation'.format(df_grps[country].
    ↪name))
    plt.savefig(str('{}_Rolling_Stats'.format(savefile)))
    plt.show(block=False)
```

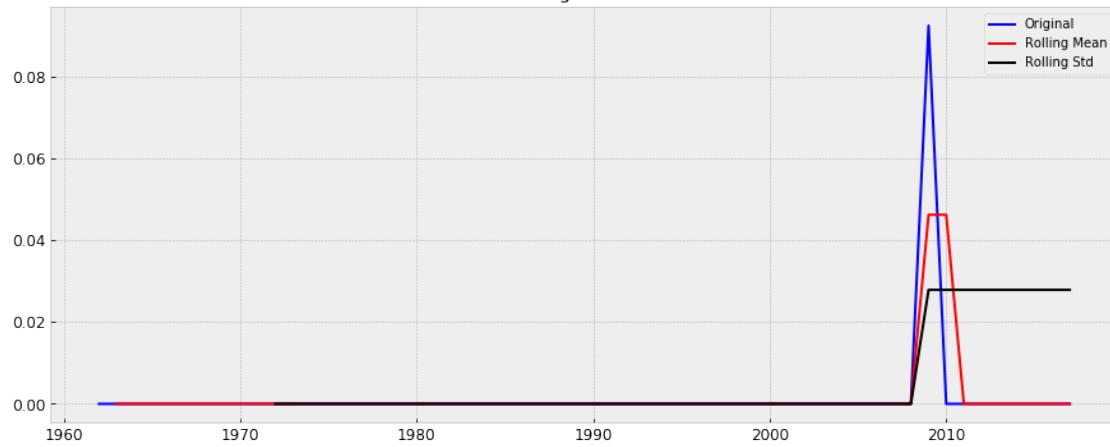
Afghanistan: Rolling Mean & Standard Deviation

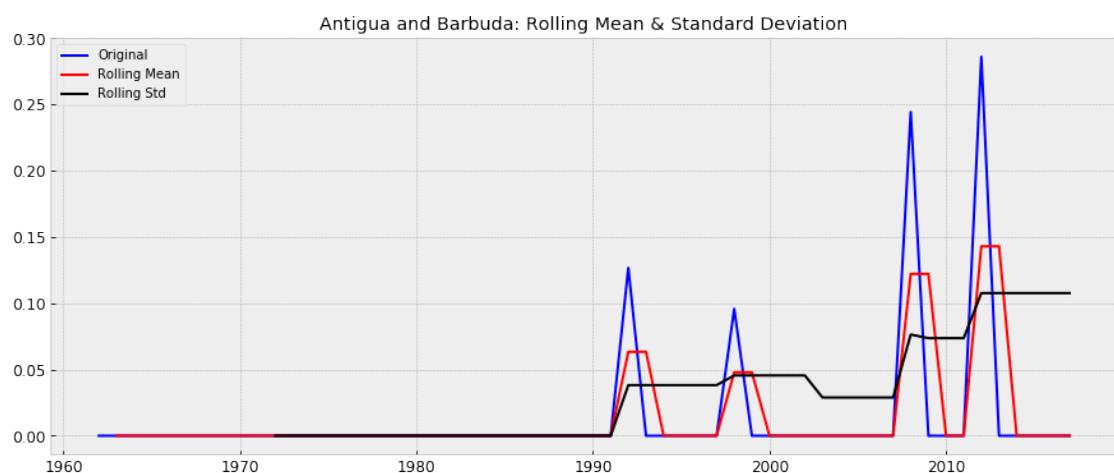
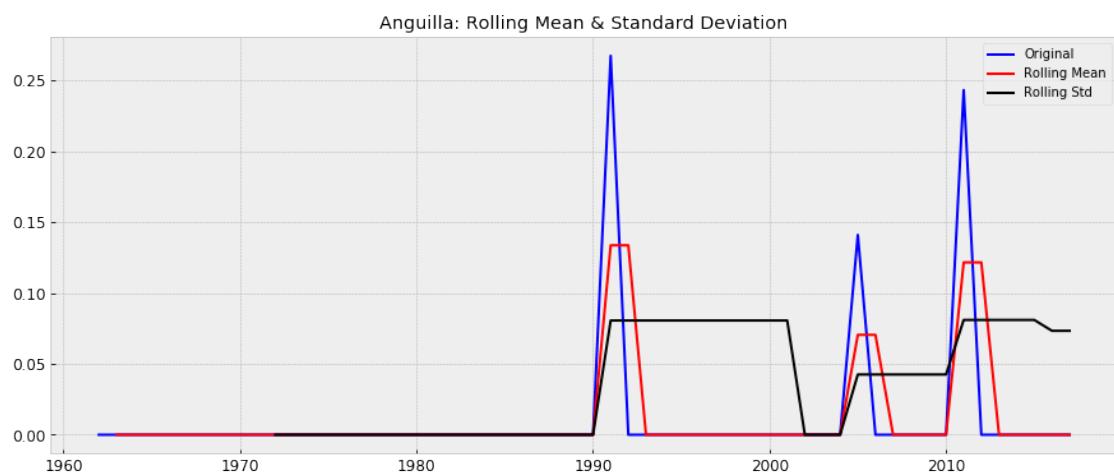
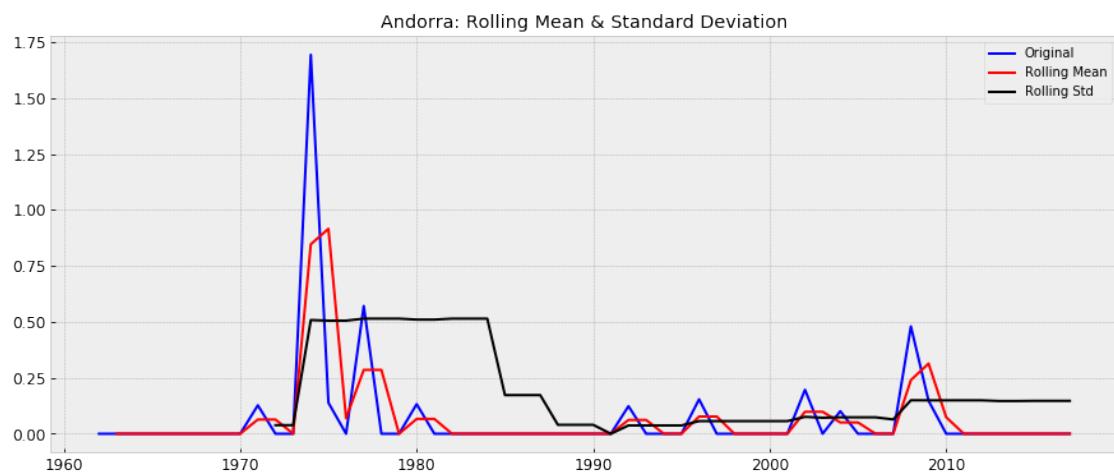


Albania: Rolling Mean & Standard Deviation

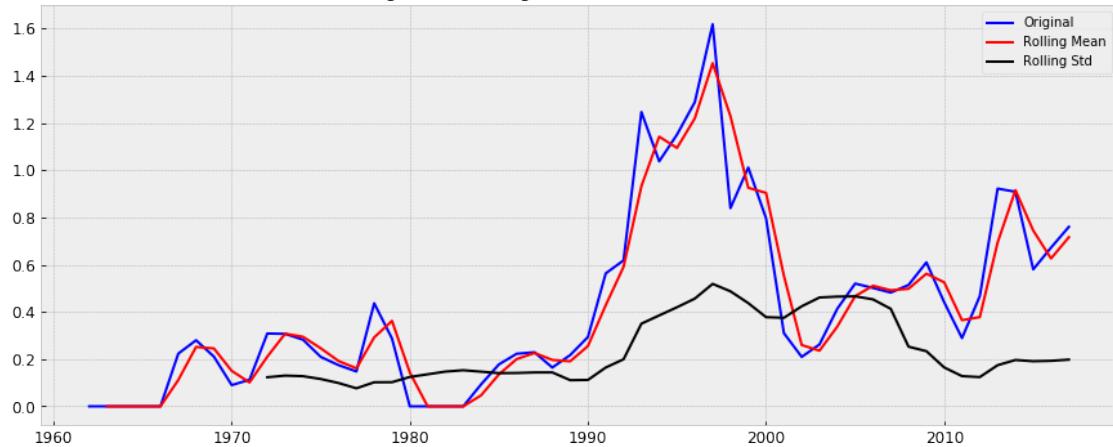


American Samoa: Rolling Mean & Standard Deviation

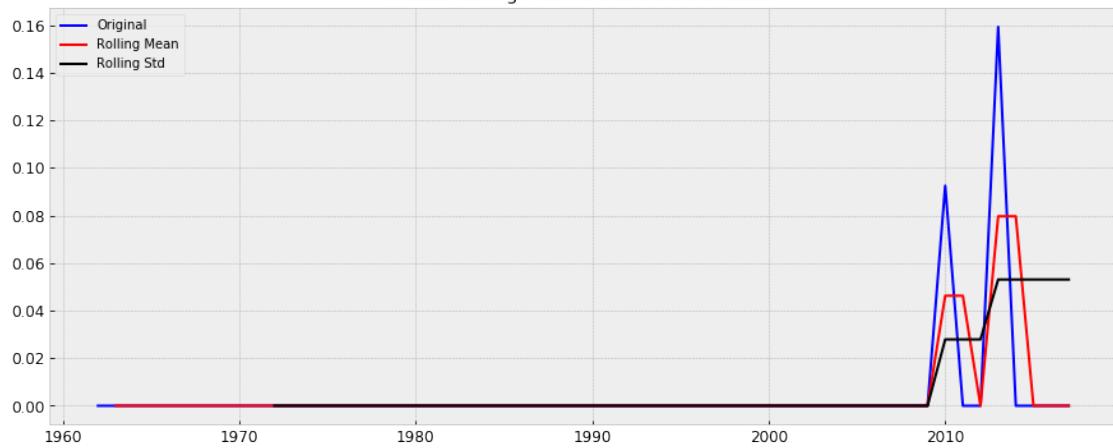




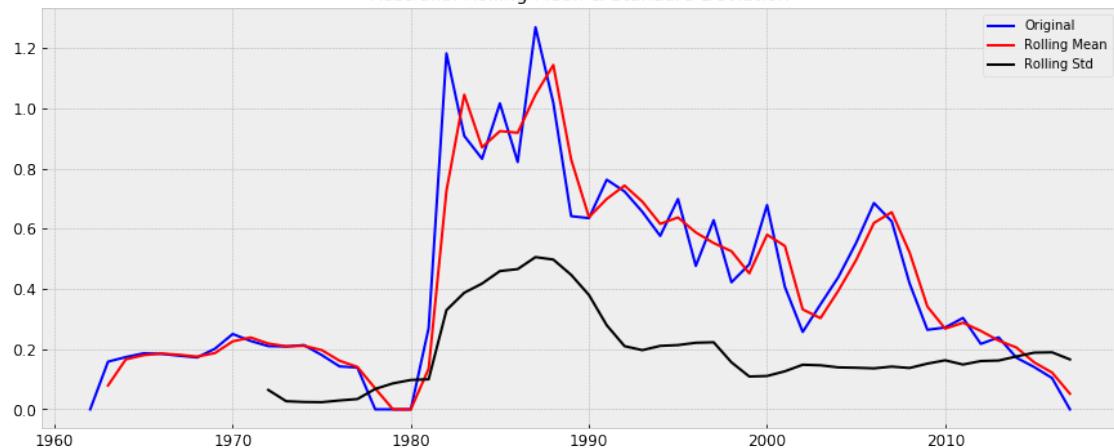
Argentina: Rolling Mean & Standard Deviation



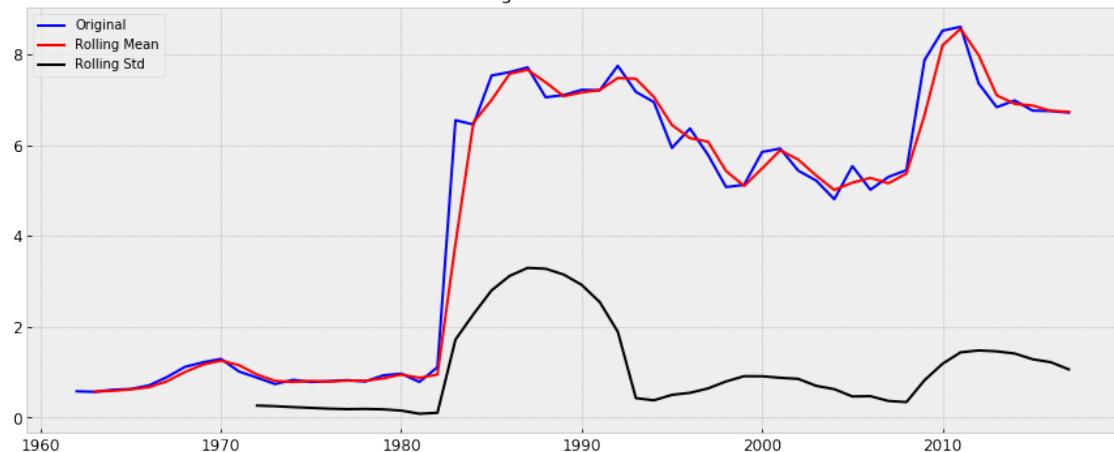
Aruba: Rolling Mean & Standard Deviation



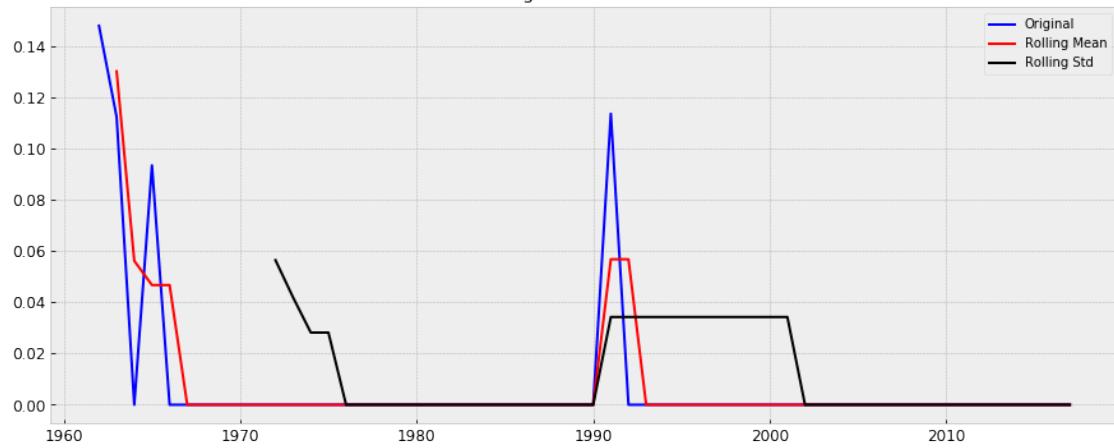
Australia: Rolling Mean & Standard Deviation



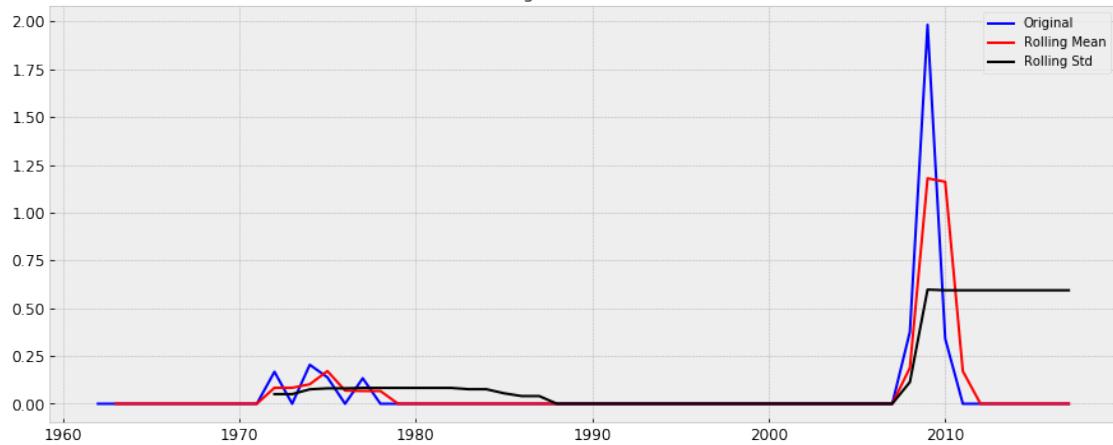
Austria: Rolling Mean & Standard Deviation



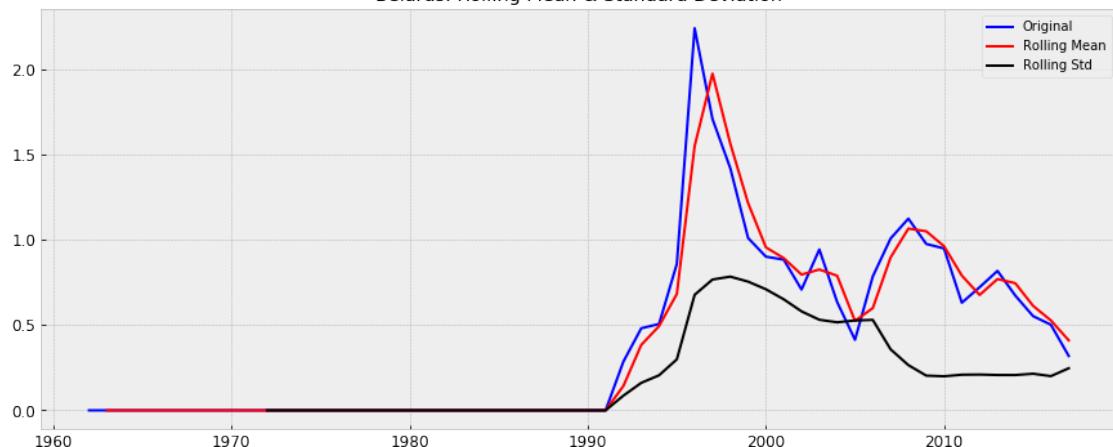
Bahamas: Rolling Mean & Standard Deviation



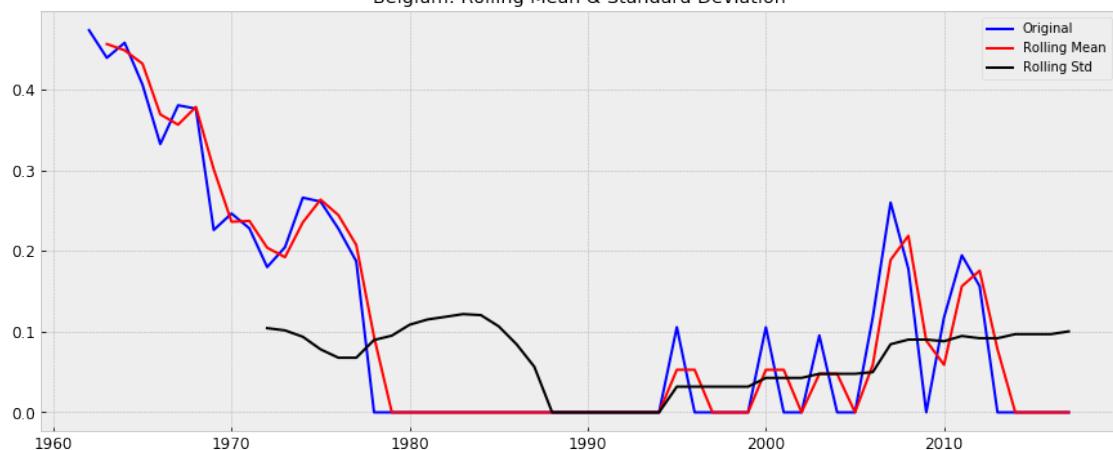
Barbados: Rolling Mean & Standard Deviation

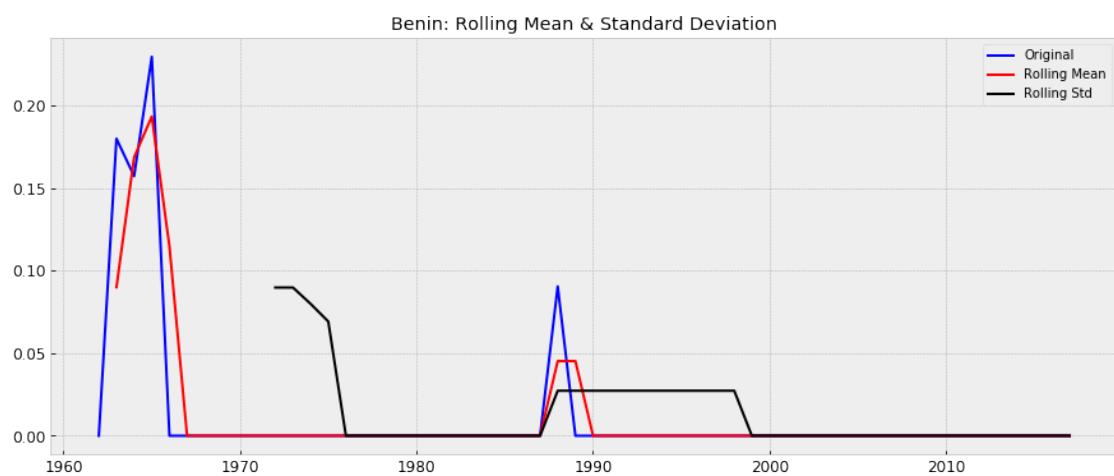
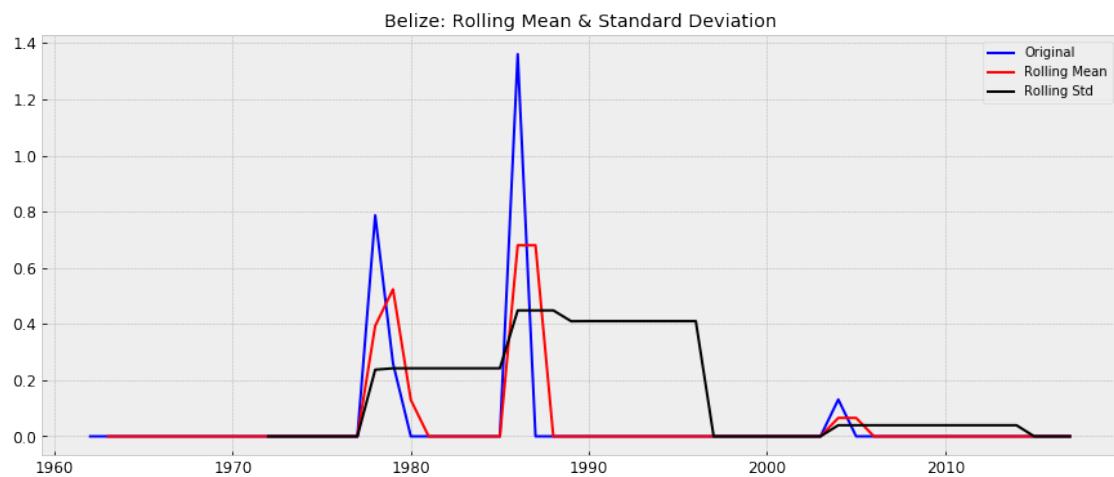


Belarus: Rolling Mean & Standard Deviation

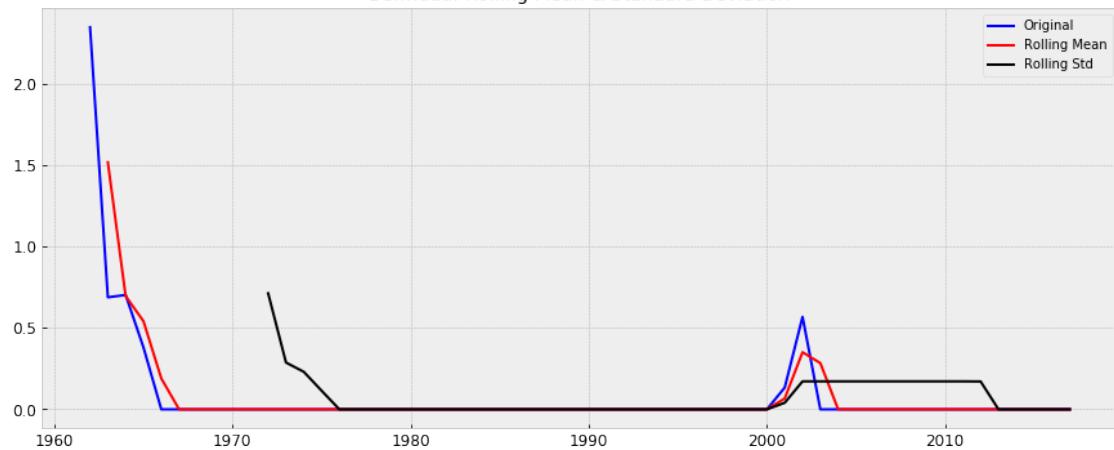


Belgium: Rolling Mean & Standard Deviation

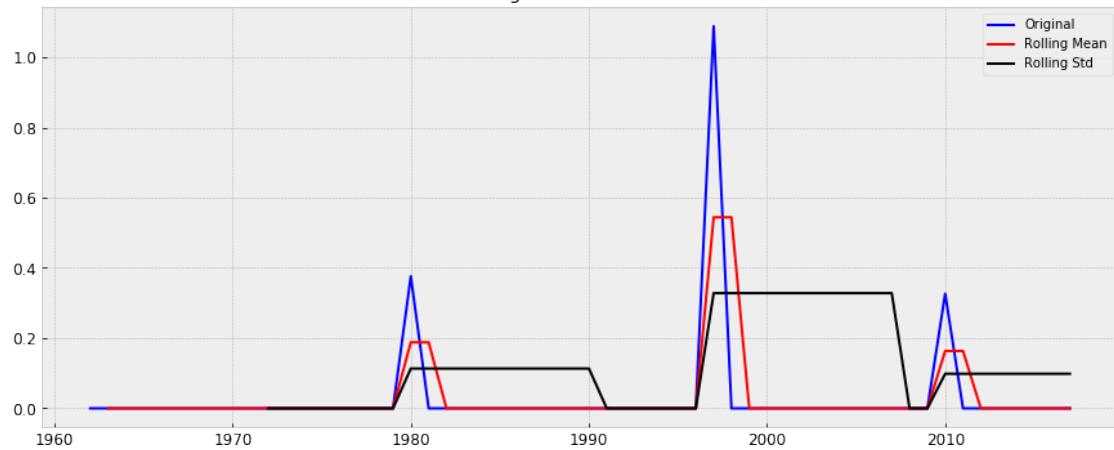




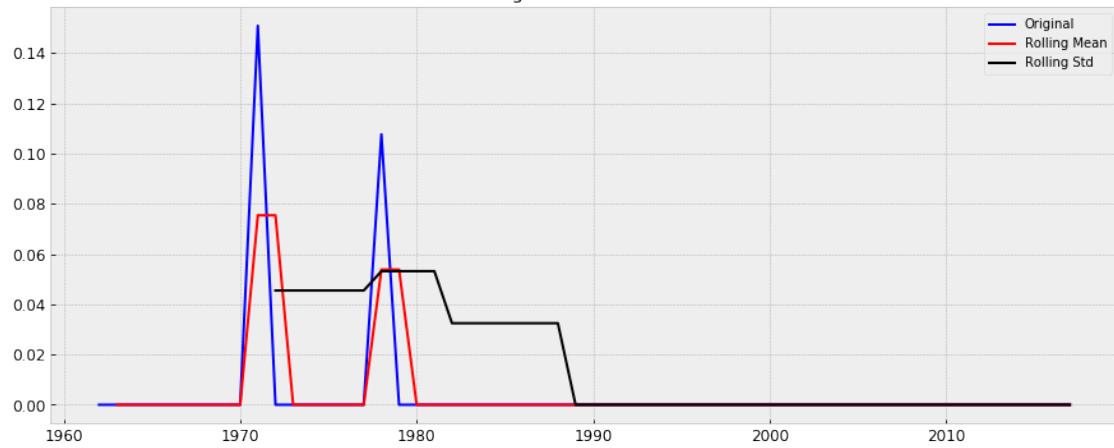
Bermuda: Rolling Mean & Standard Deviation



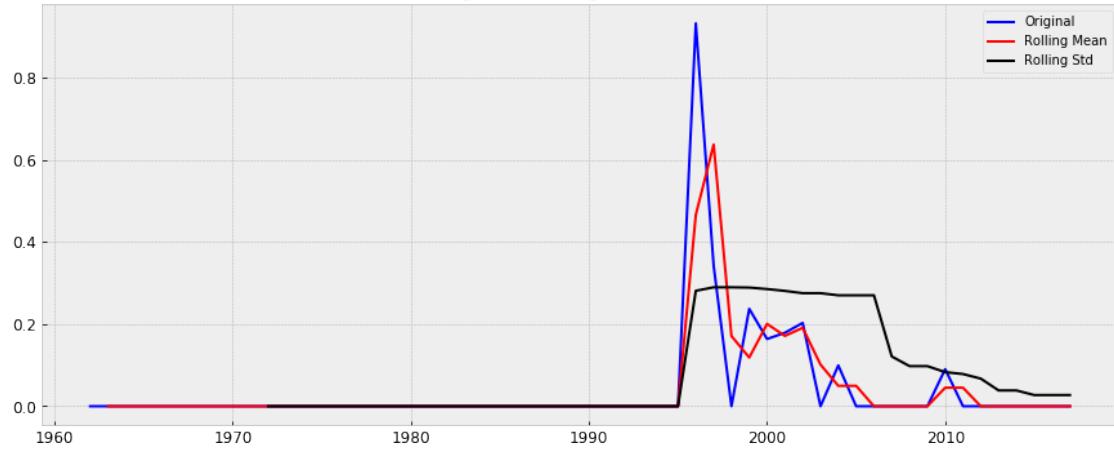
Bhutan: Rolling Mean & Standard Deviation



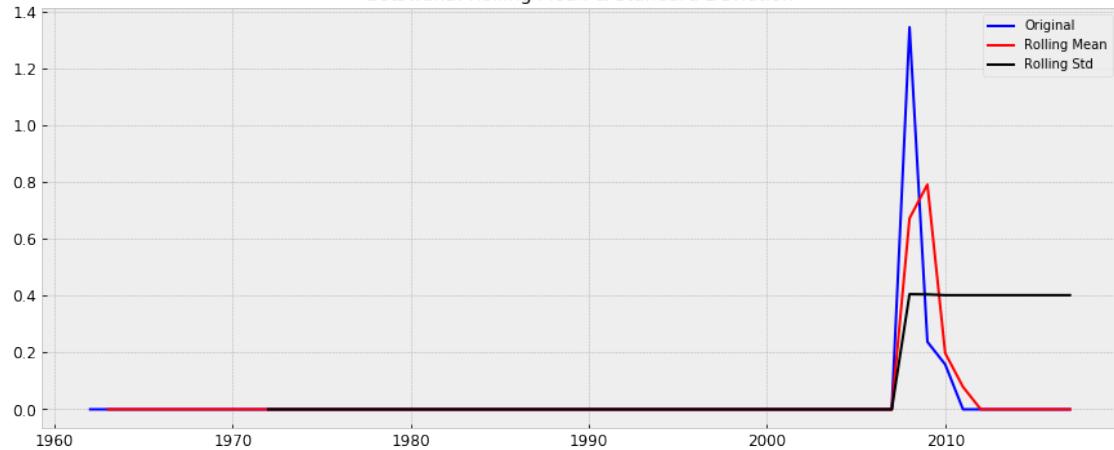
Bolivia: Rolling Mean & Standard Deviation



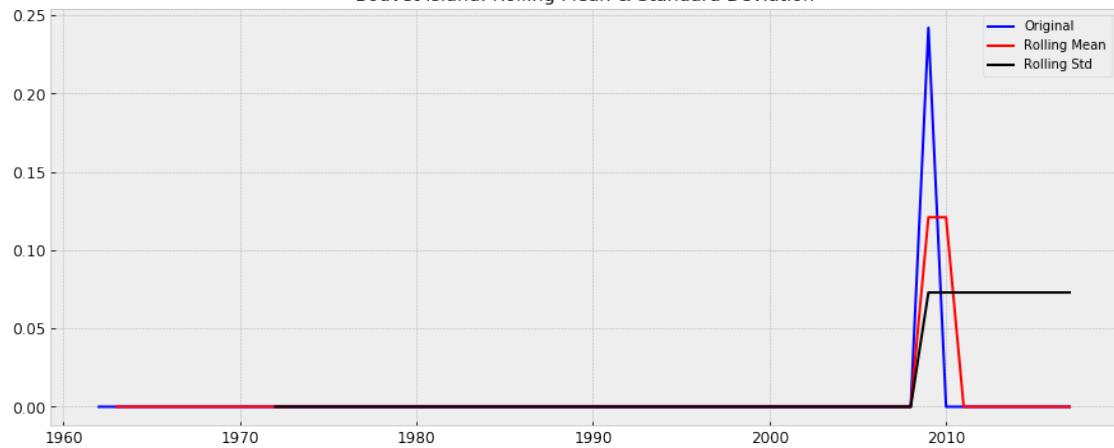
Bosnia and Herzegovina: Rolling Mean & Standard Deviation



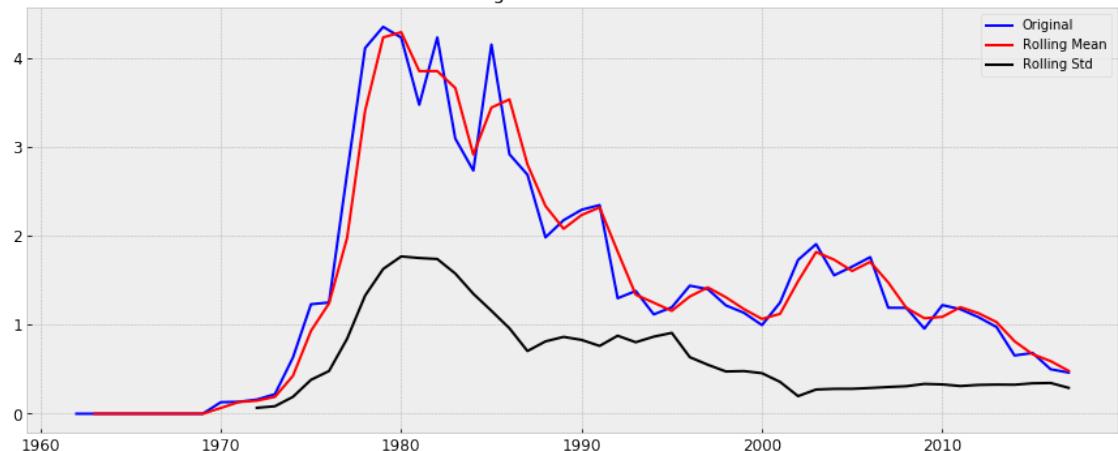
Botswana: Rolling Mean & Standard Deviation



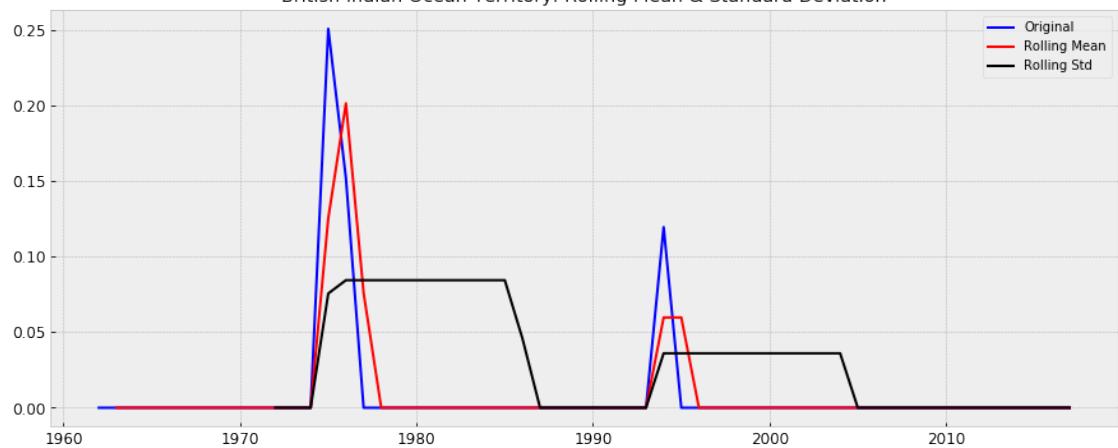
Bouvet Island: Rolling Mean & Standard Deviation



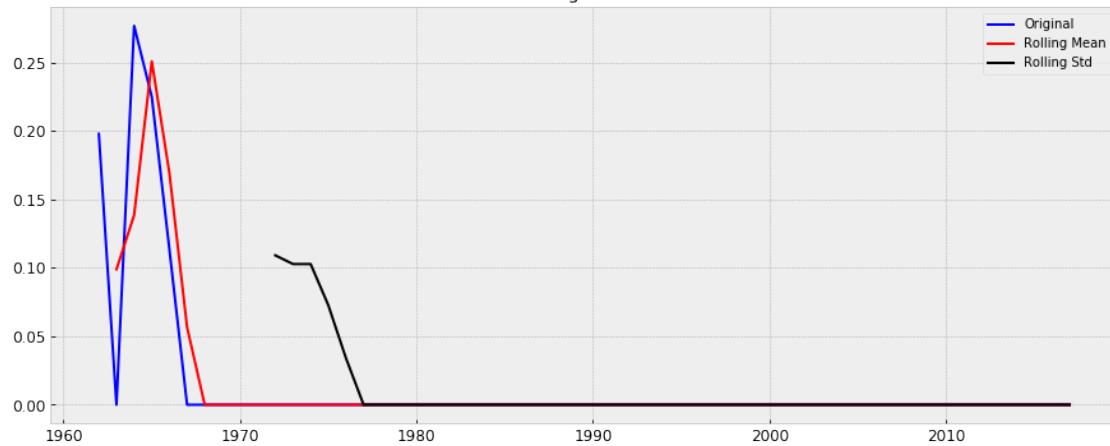
Brazil: Rolling Mean & Standard Deviation



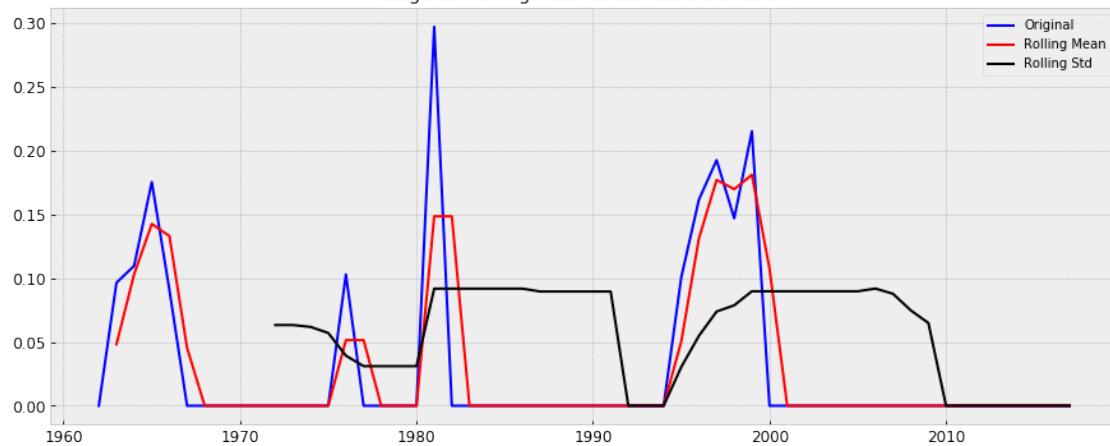
British Indian Ocean Territory: Rolling Mean & Standard Deviation



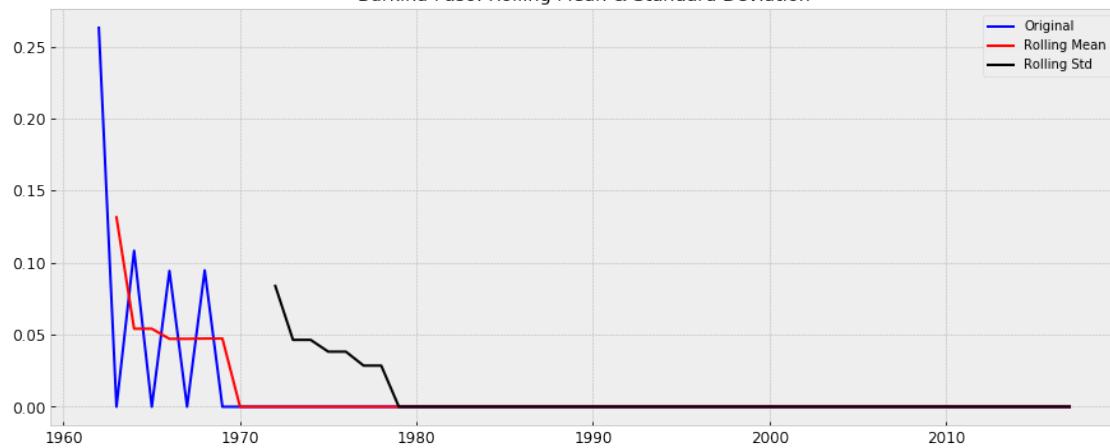
Brunei Darussalam: Rolling Mean & Standard Deviation



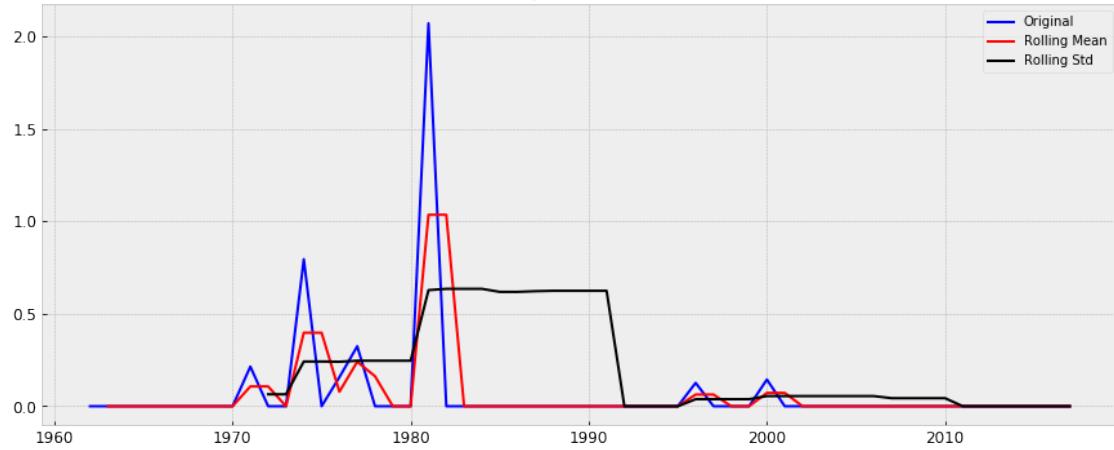
Bulgaria: Rolling Mean & Standard Deviation



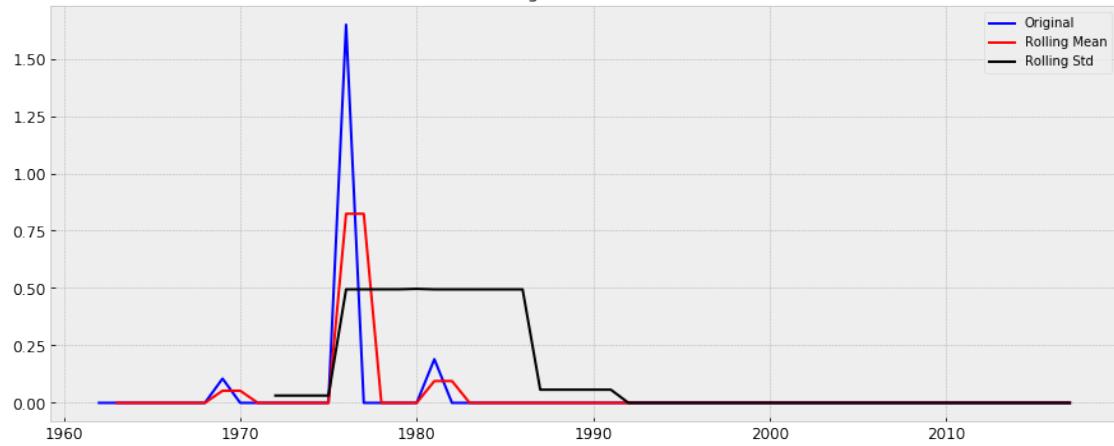
Burkina Faso: Rolling Mean & Standard Deviation



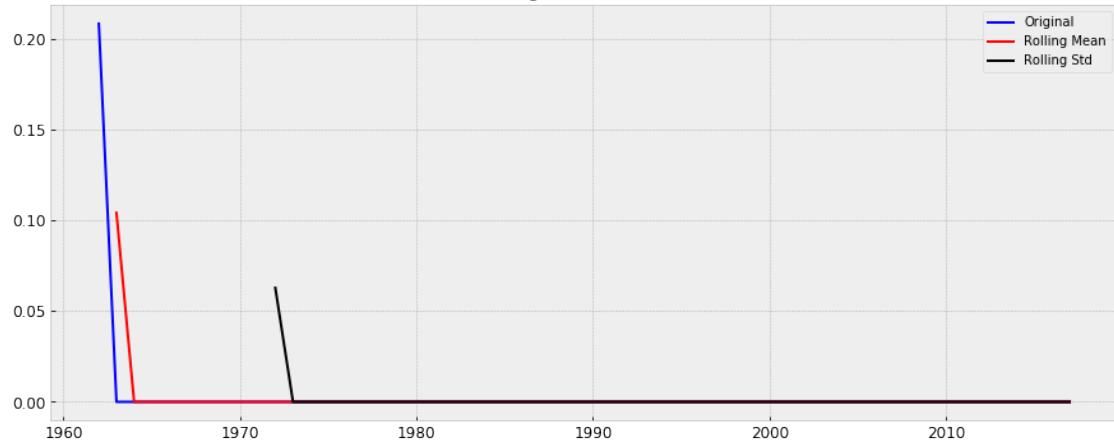
Cabo Verde: Rolling Mean & Standard Deviation



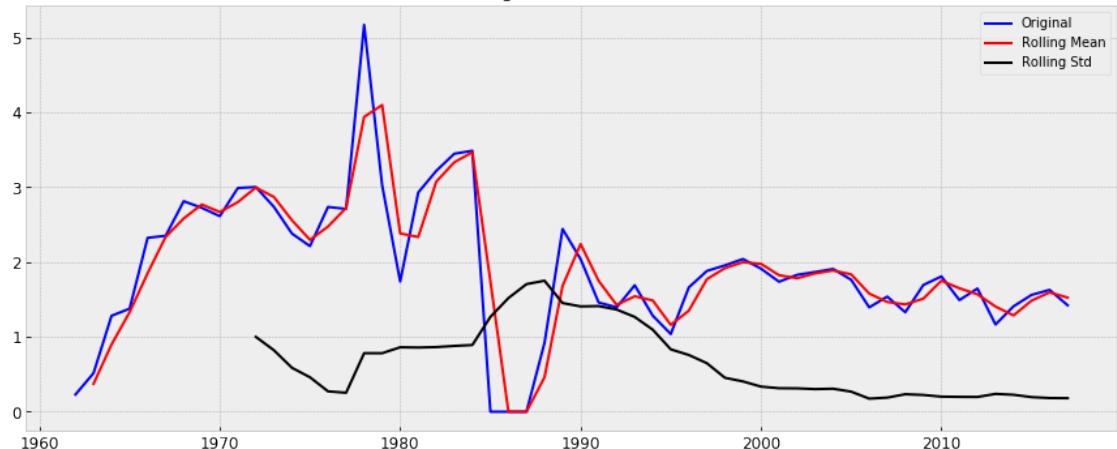
Cambodia: Rolling Mean & Standard Deviation



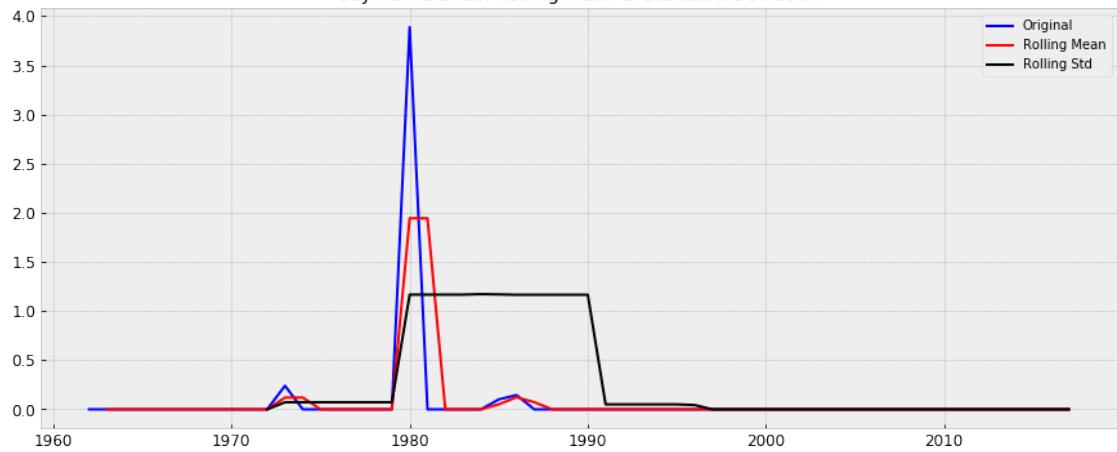
Cameroon: Rolling Mean & Standard Deviation



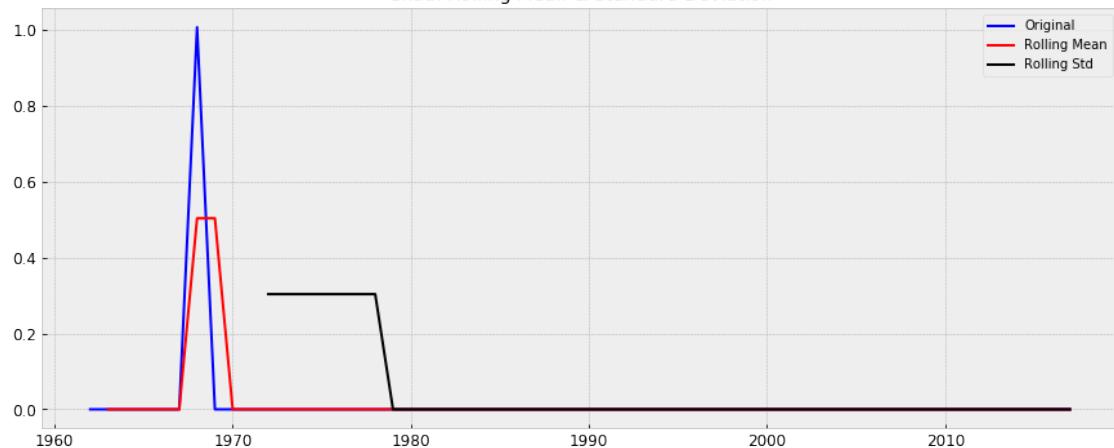
Canada: Rolling Mean & Standard Deviation



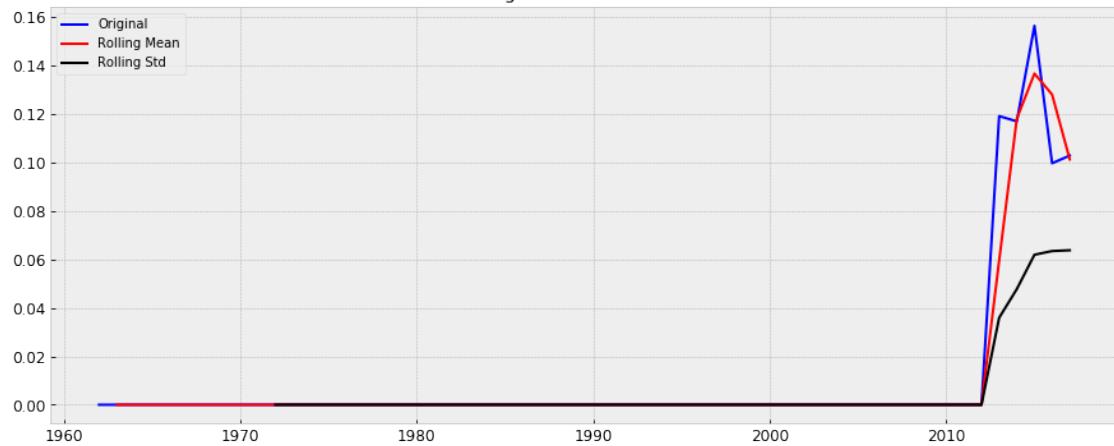
Cayman Islands: Rolling Mean & Standard Deviation



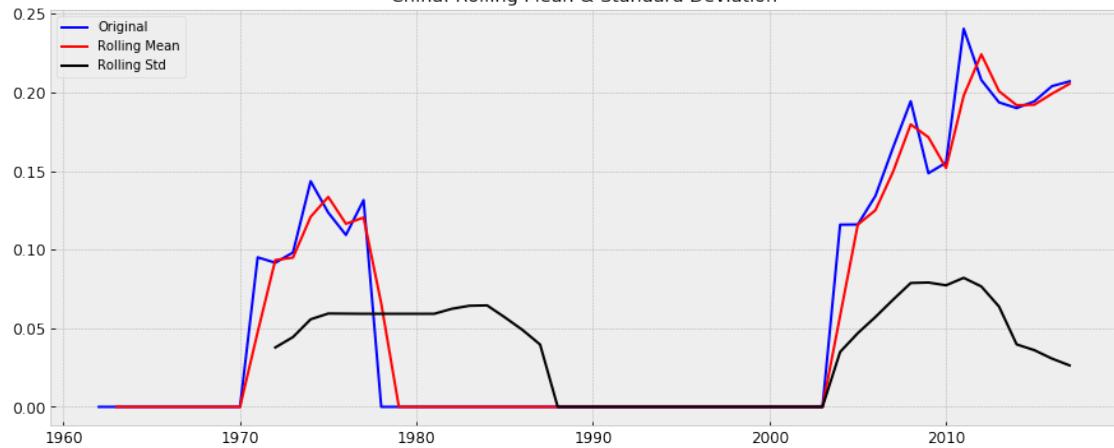
Chad: Rolling Mean & Standard Deviation



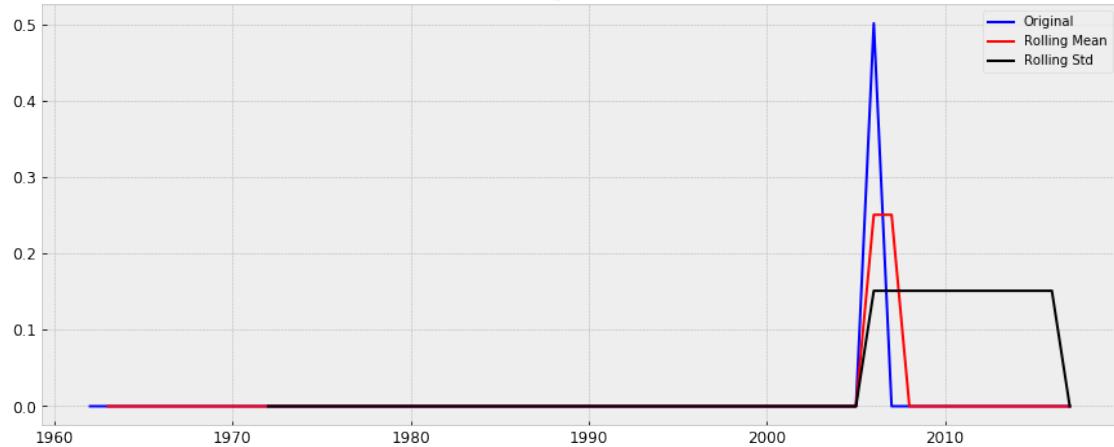
Chile: Rolling Mean & Standard Deviation



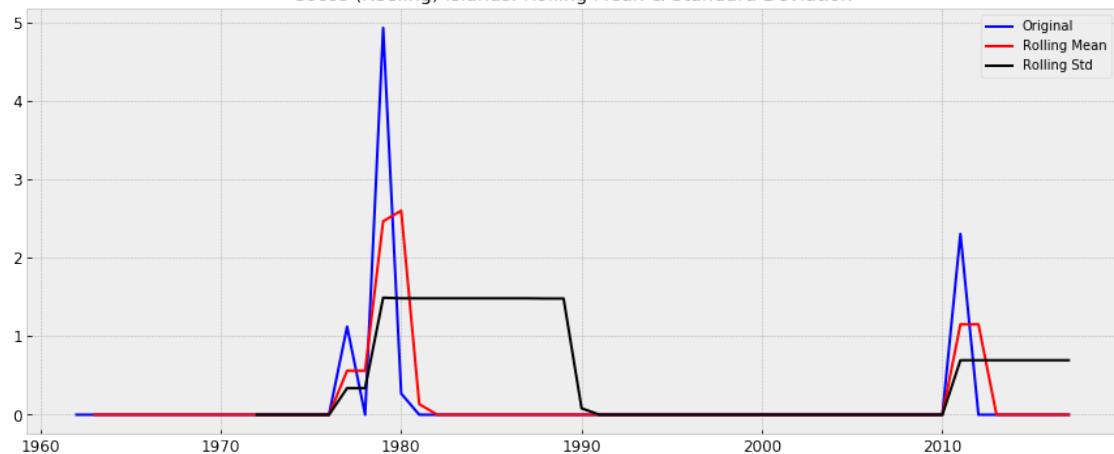
China: Rolling Mean & Standard Deviation



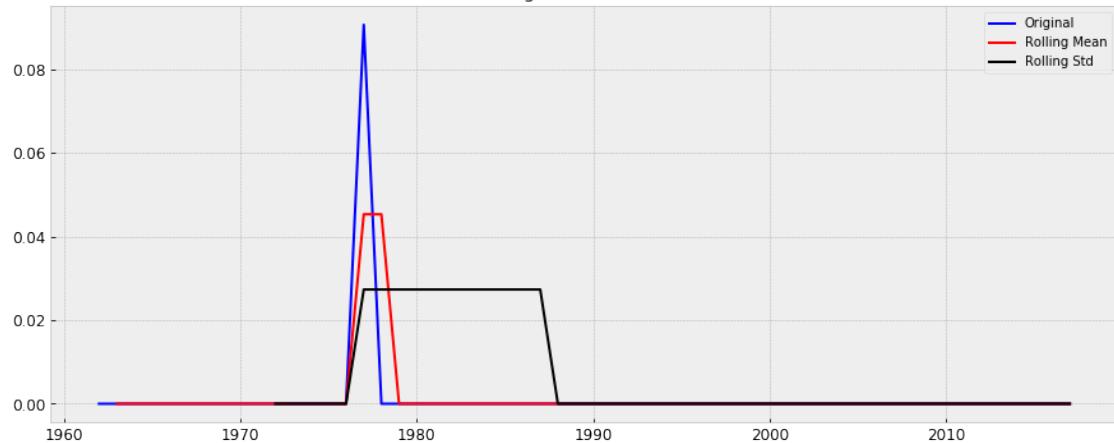
Christmas Island: Rolling Mean & Standard Deviation

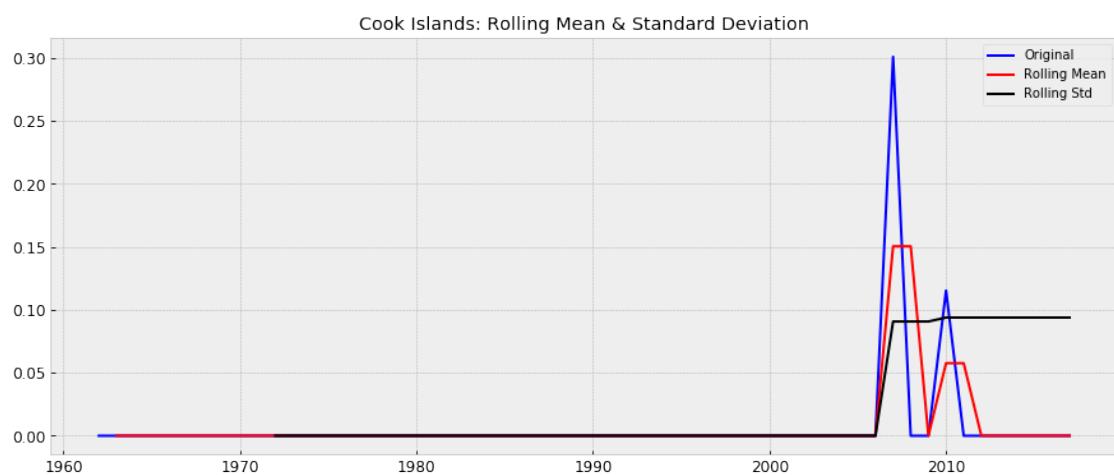
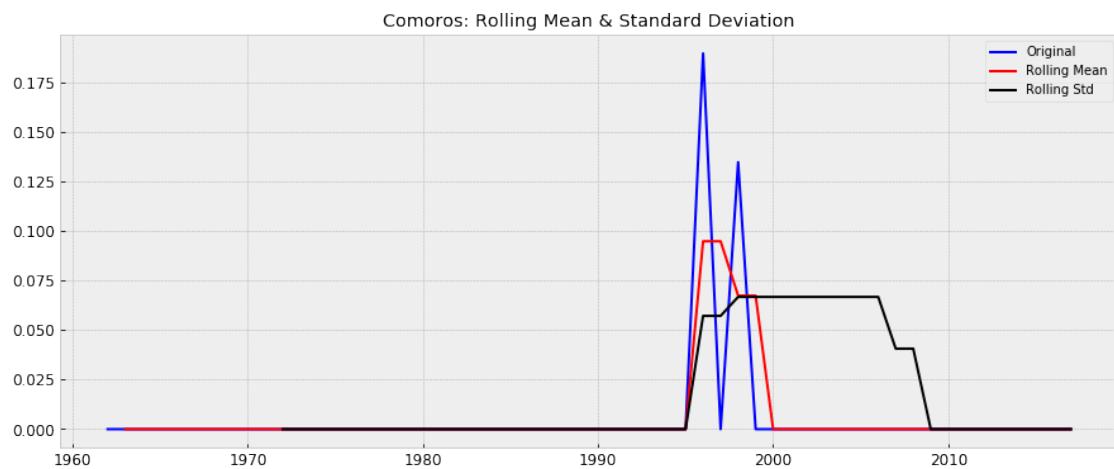


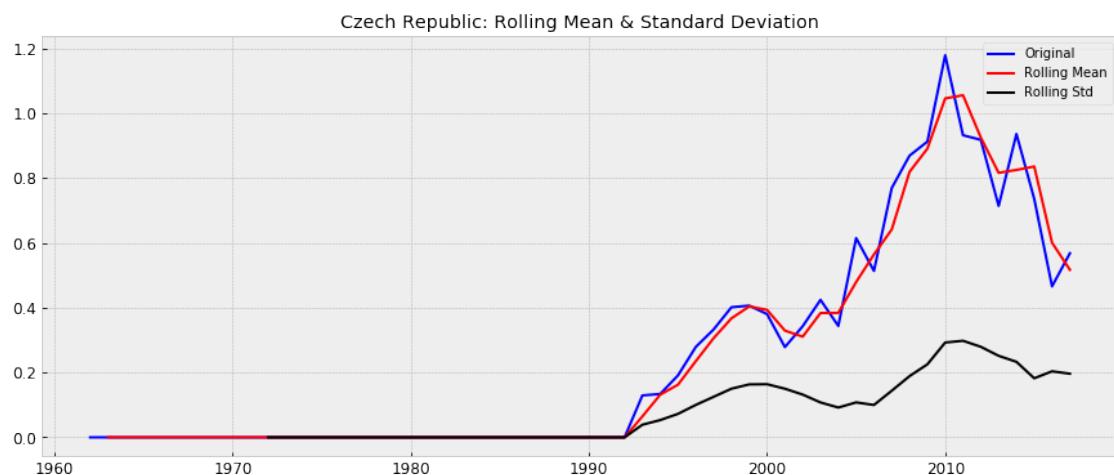
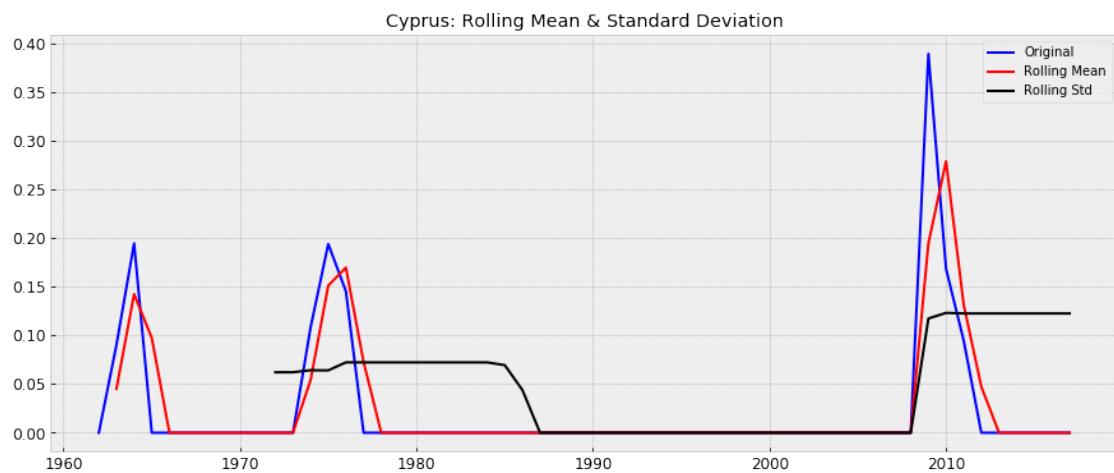
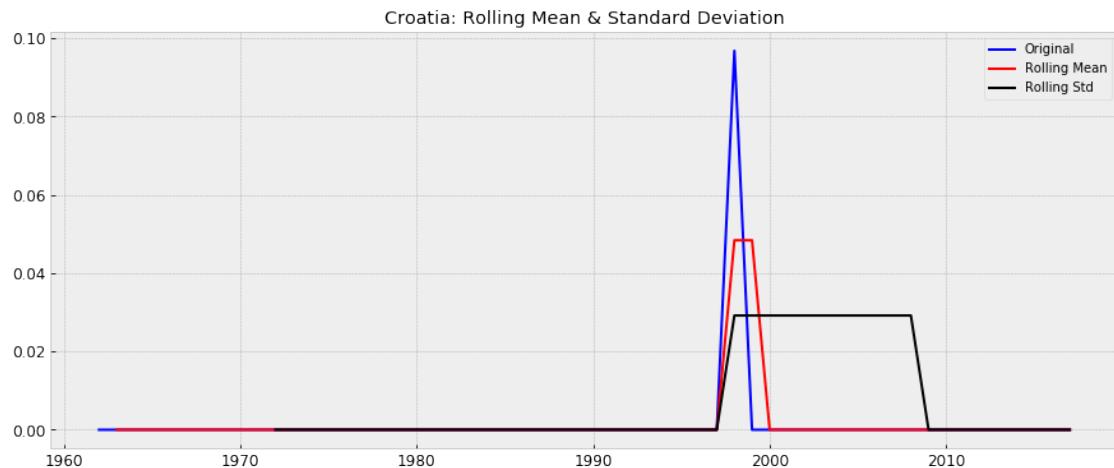
Cocos (Keeling) Islands: Rolling Mean & Standard Deviation



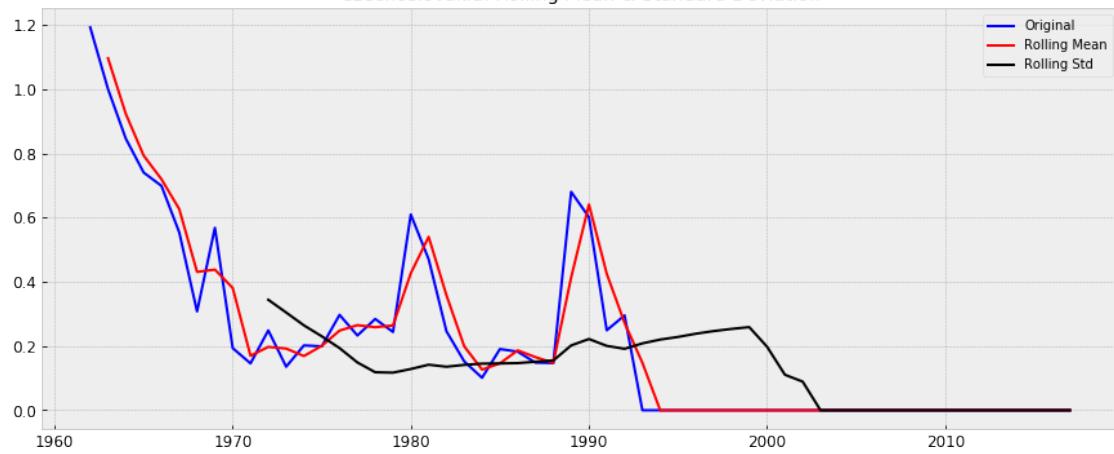
Colombia: Rolling Mean & Standard Deviation



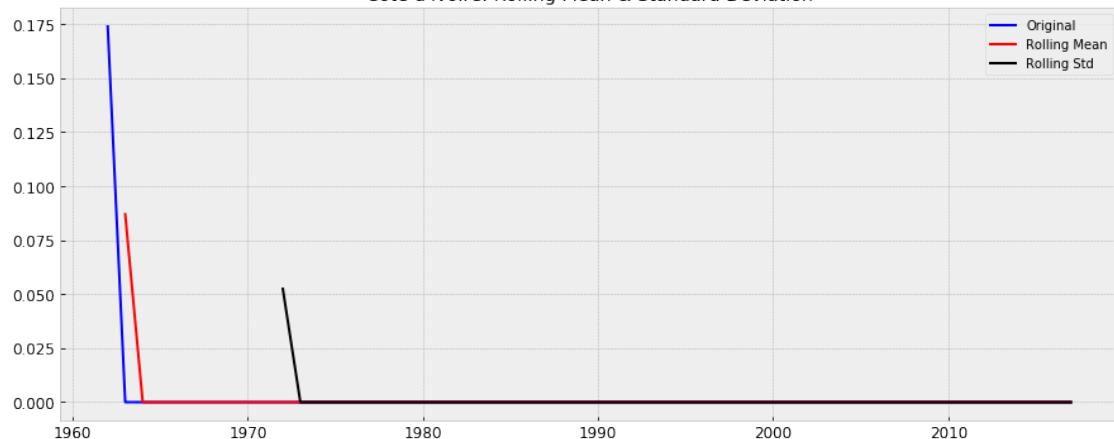




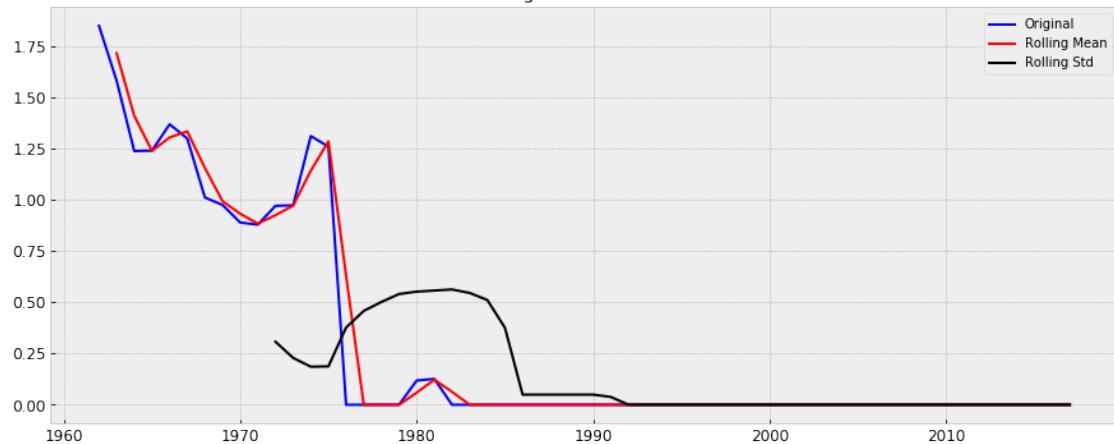
Czechoslovakia: Rolling Mean & Standard Deviation



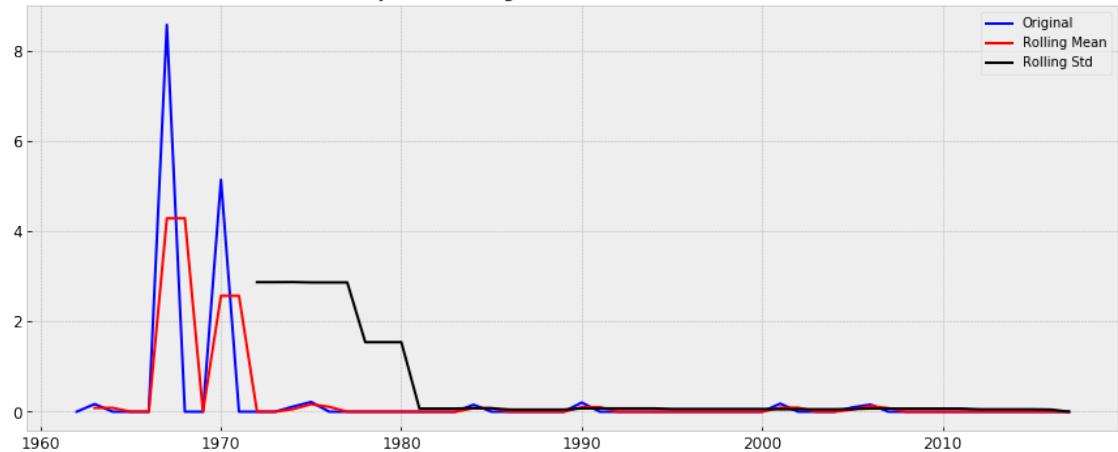
Côte d'Ivoire: Rolling Mean & Standard Deviation



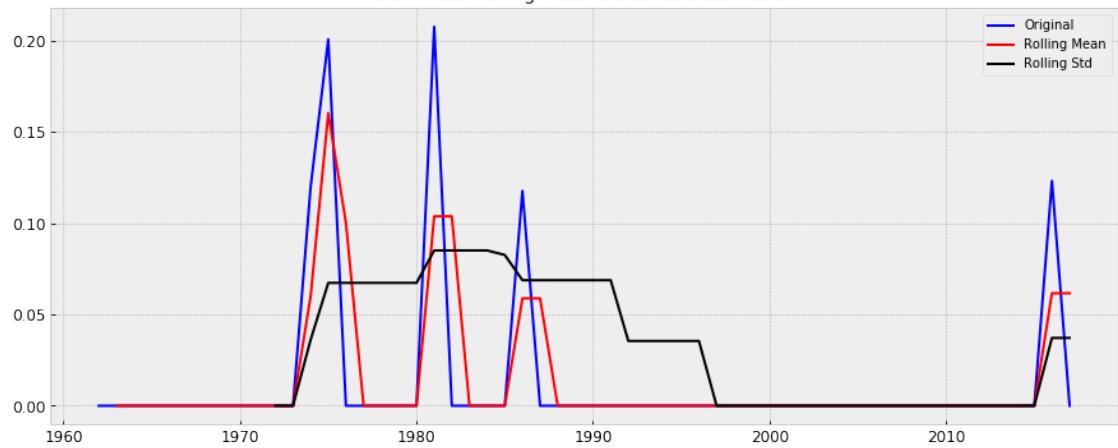
Denmark: Rolling Mean & Standard Deviation



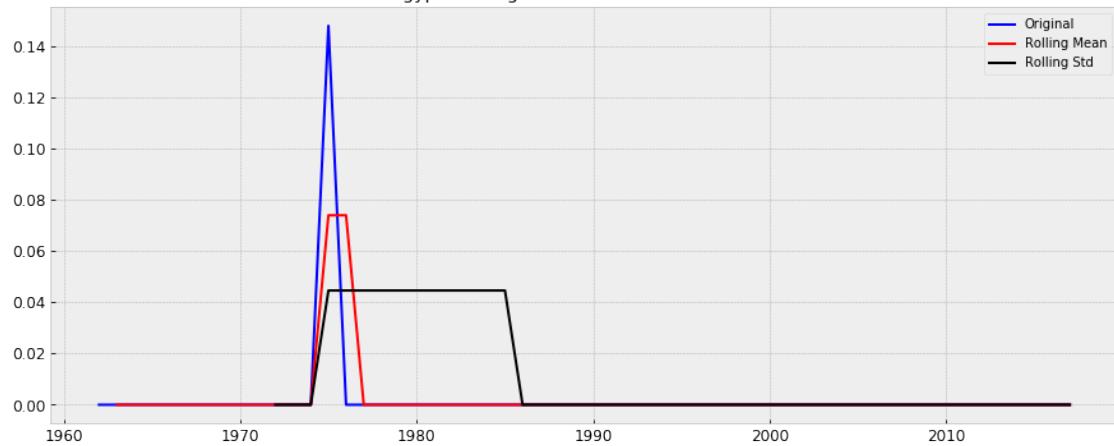
Djibouti: Rolling Mean & Standard Deviation



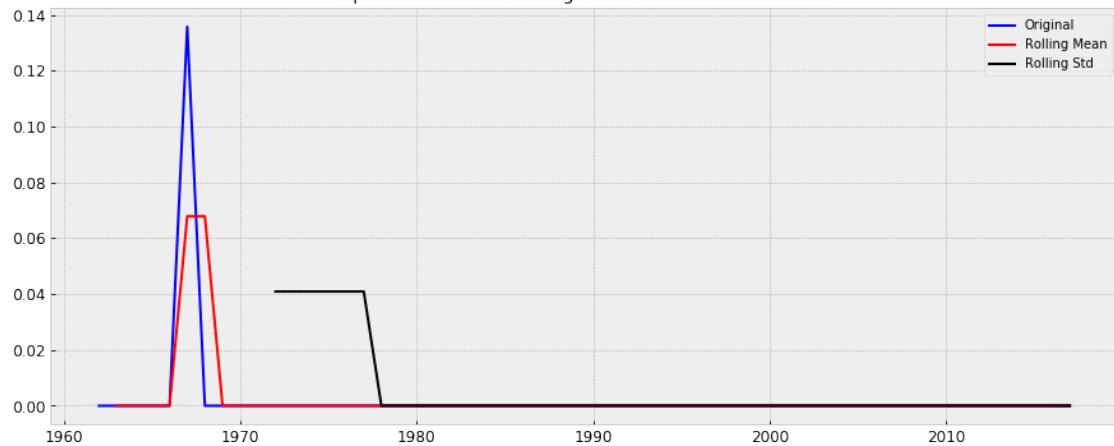
Dominica: Rolling Mean & Standard Deviation



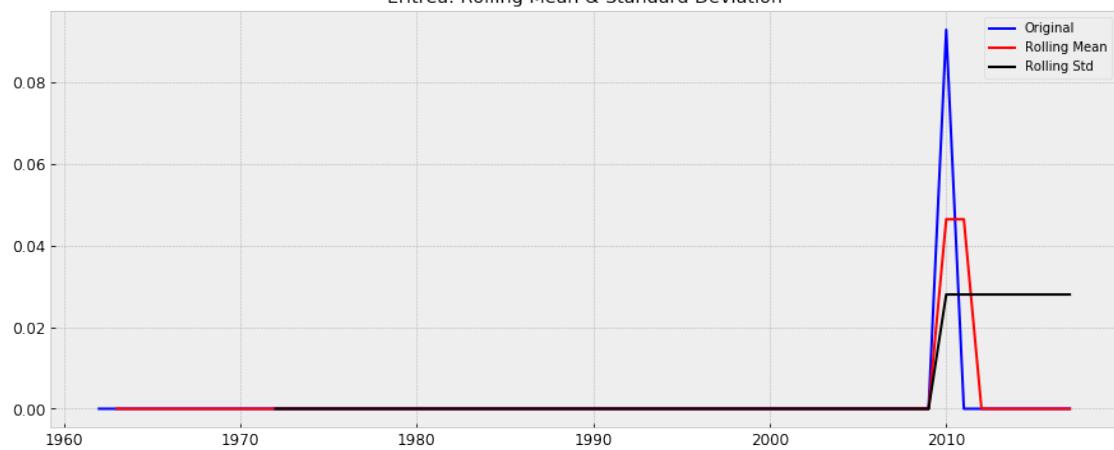
Egypt: Rolling Mean & Standard Deviation



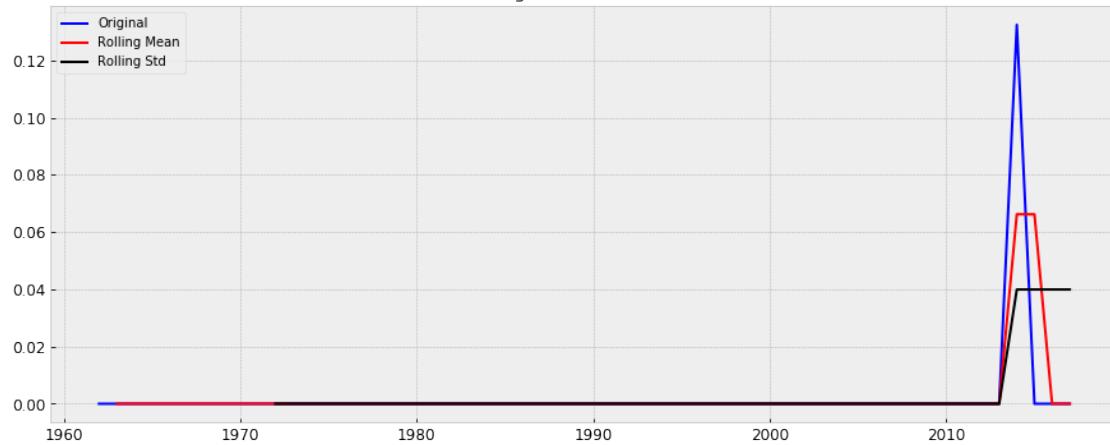
Equatorial Guinea: Rolling Mean & Standard Deviation



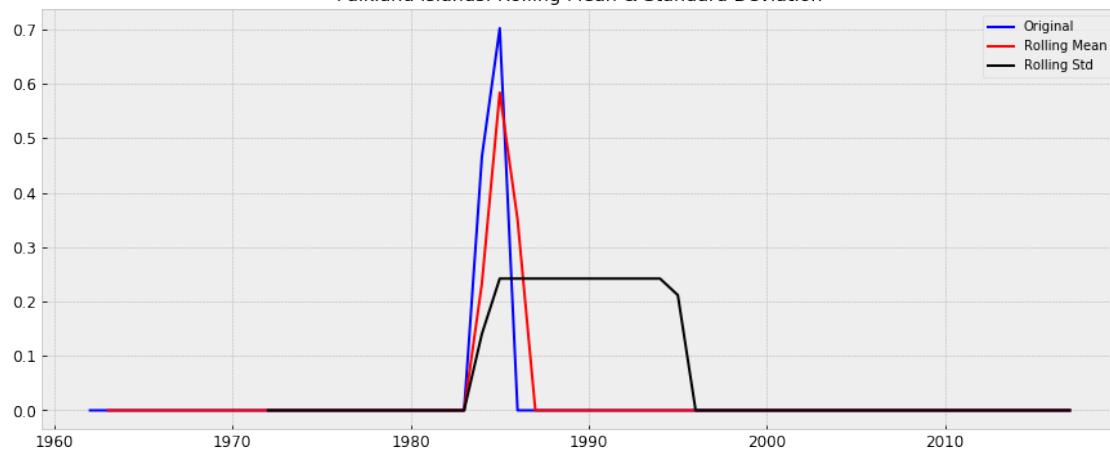
Eritrea: Rolling Mean & Standard Deviation



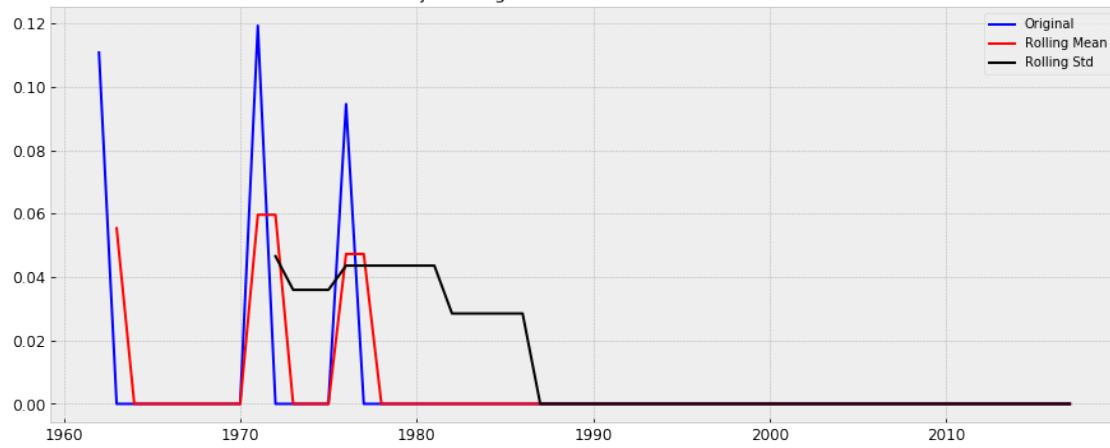
Estonia: Rolling Mean & Standard Deviation



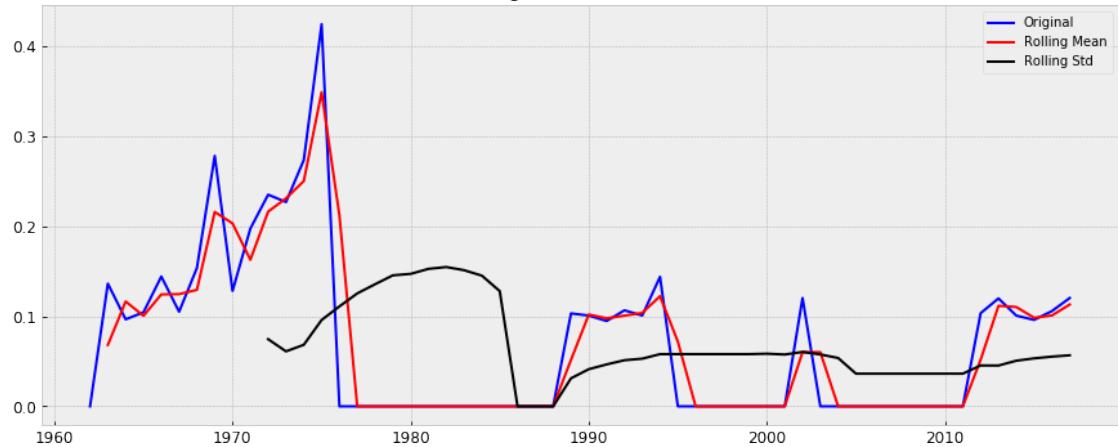
Falkland Islands: Rolling Mean & Standard Deviation



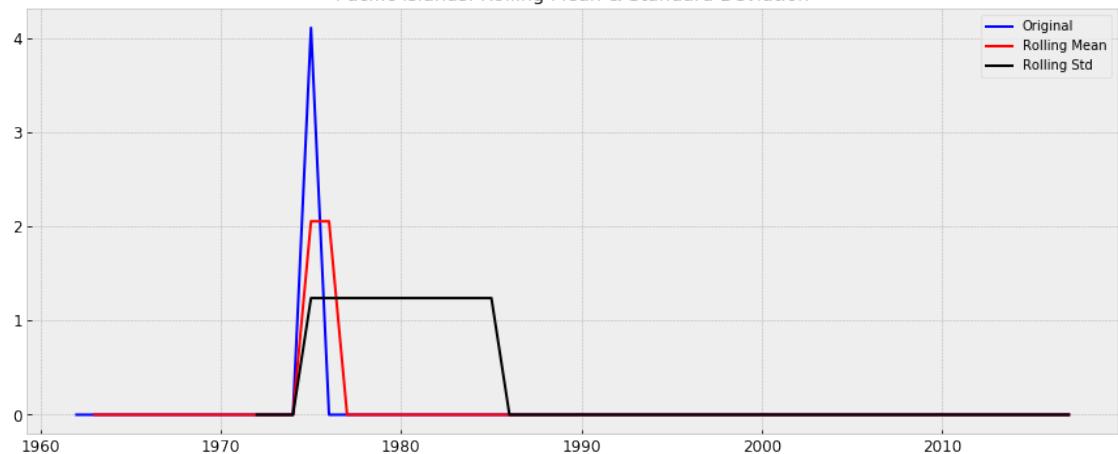
Fiji: Rolling Mean & Standard Deviation



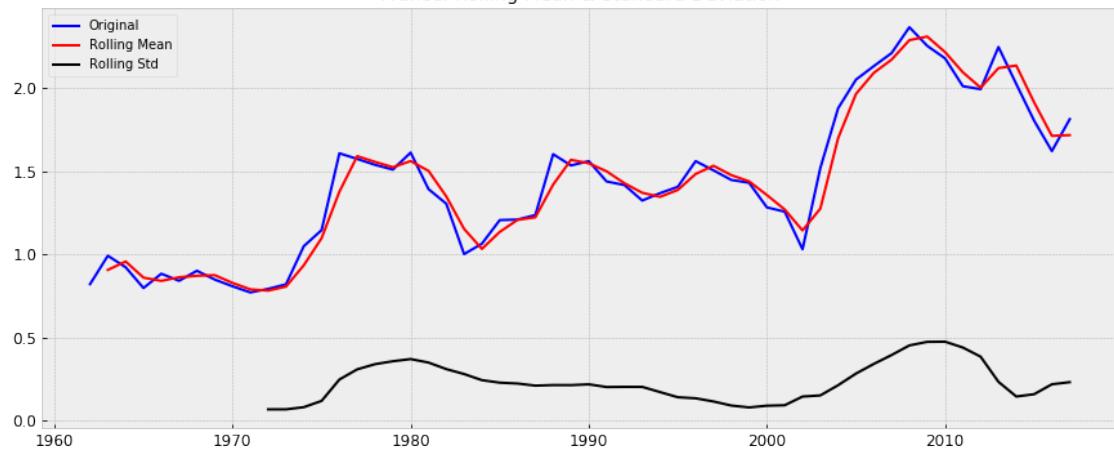
Finland: Rolling Mean & Standard Deviation



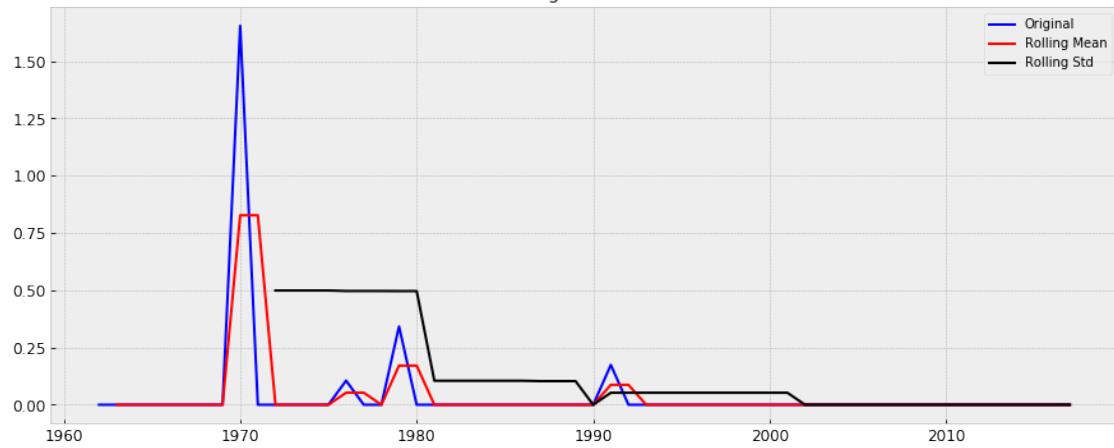
Pacific Islands: Rolling Mean & Standard Deviation



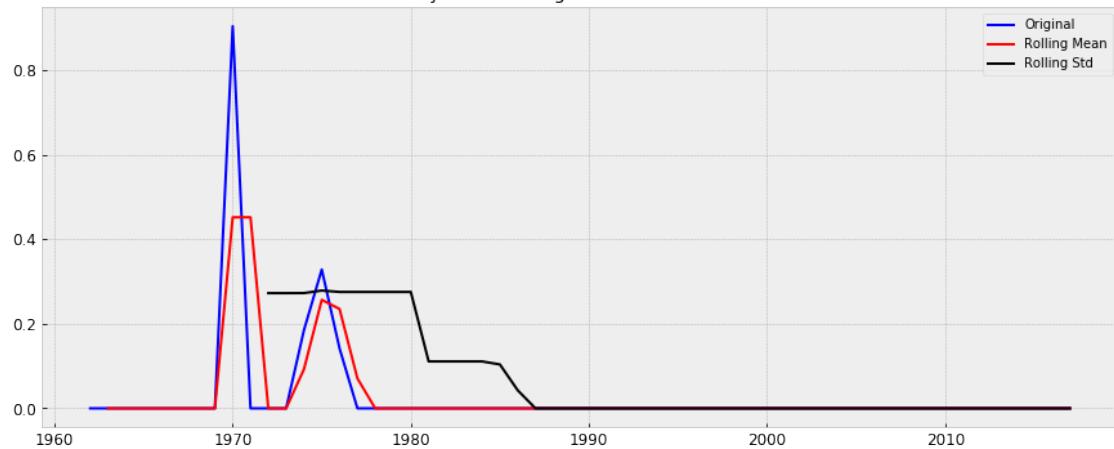
France: Rolling Mean & Standard Deviation



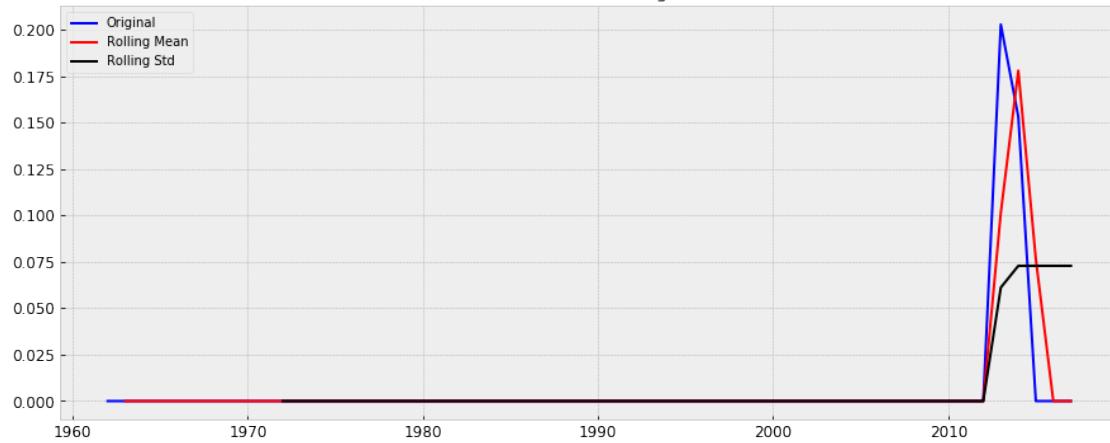
French Guiana: Rolling Mean & Standard Deviation



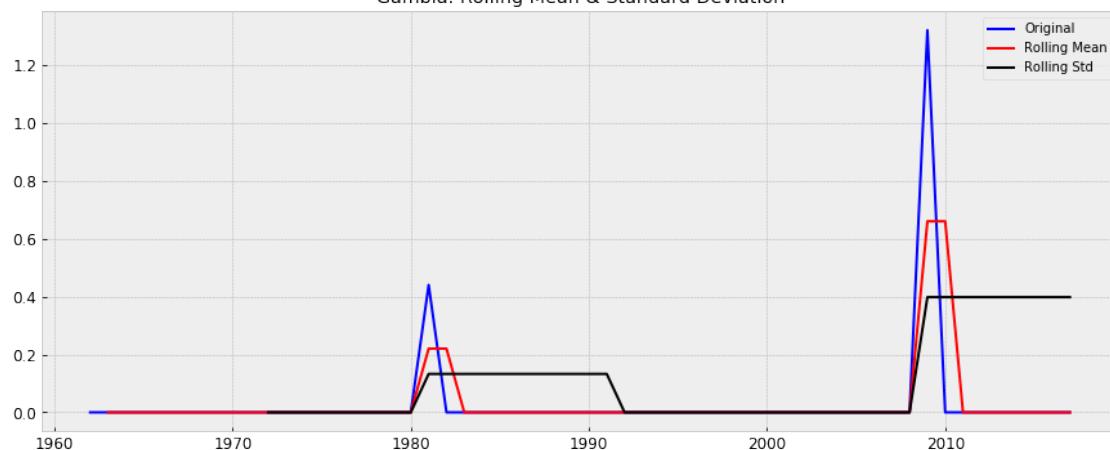
French Polynesia: Rolling Mean & Standard Deviation



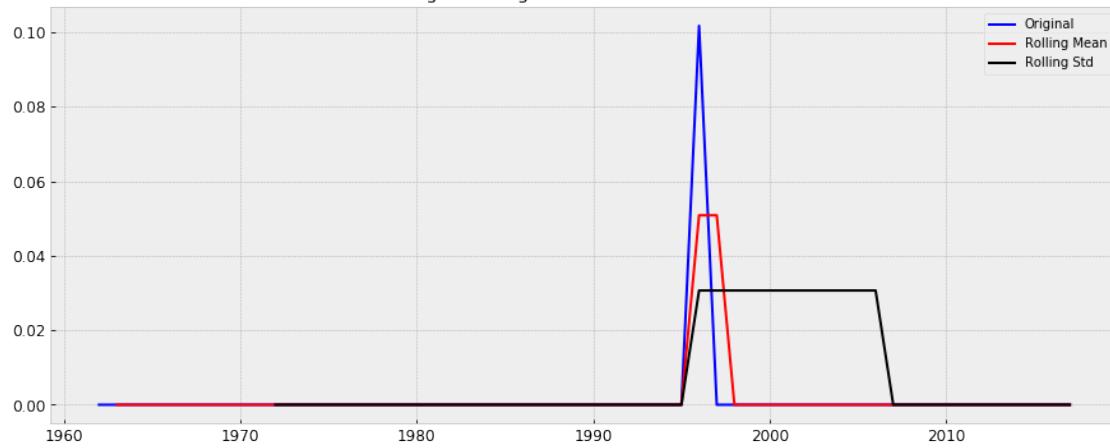
French Southern and Antarctic Lands: Rolling Mean & Standard Deviation



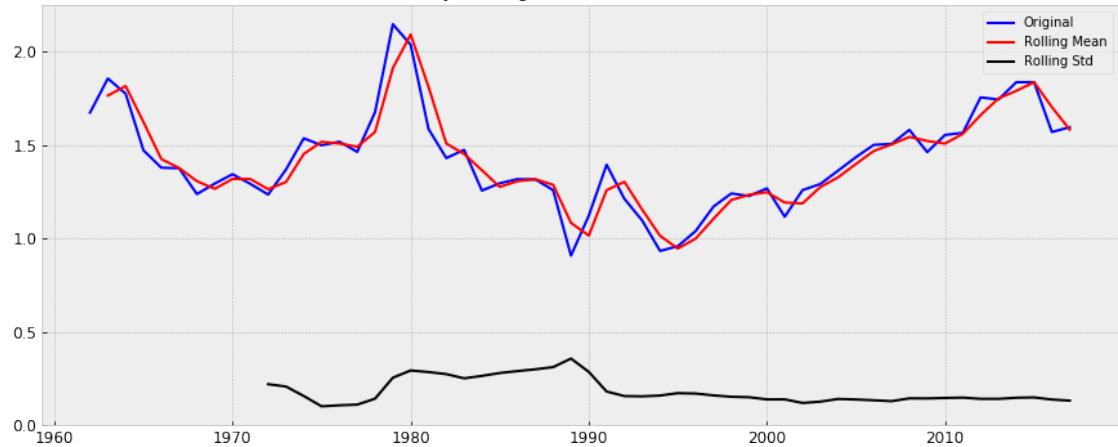
Gambia: Rolling Mean & Standard Deviation



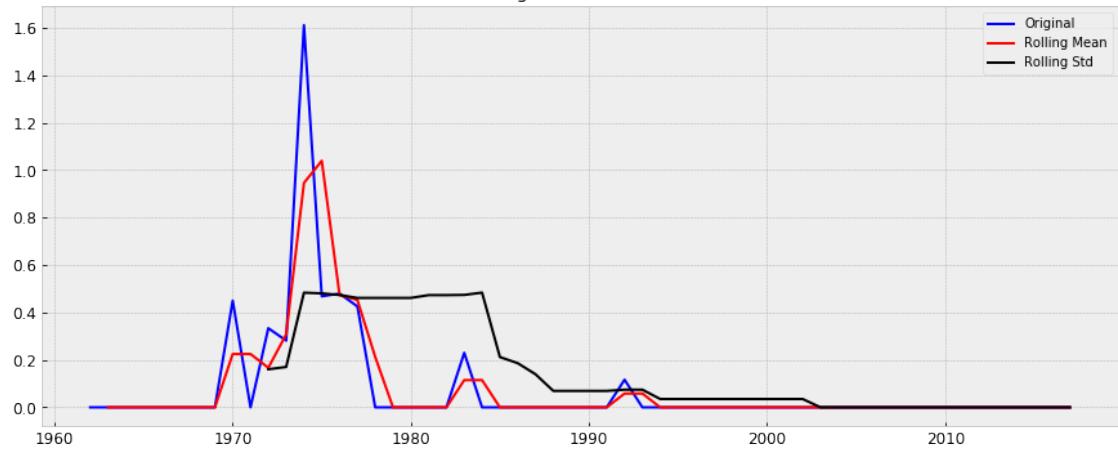
Georgia: Rolling Mean & Standard Deviation



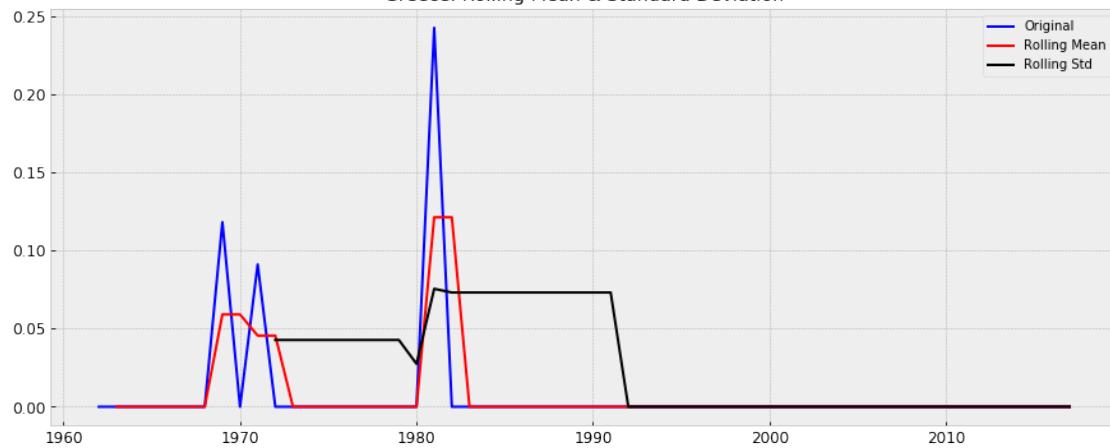
Germany: Rolling Mean & Standard Deviation



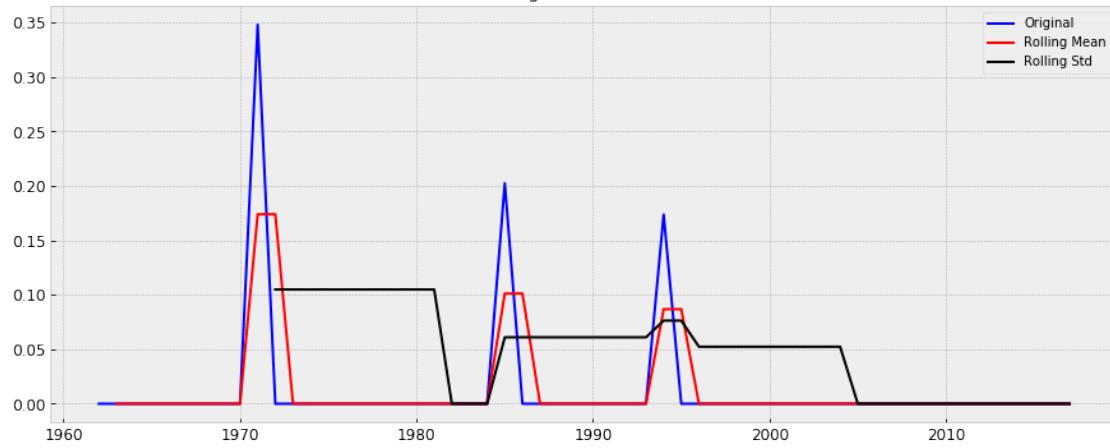
Gibraltar: Rolling Mean & Standard Deviation



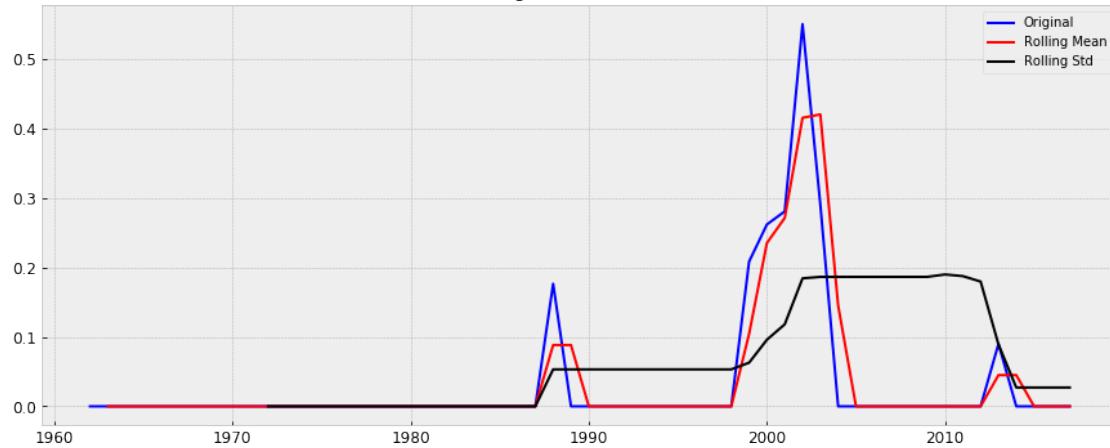
Greece: Rolling Mean & Standard Deviation



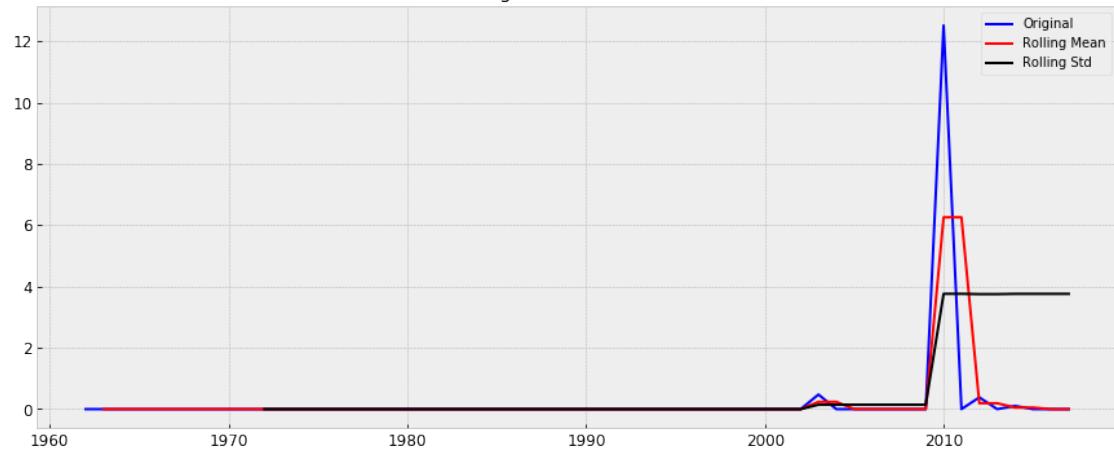
Greenland: Rolling Mean & Standard Deviation



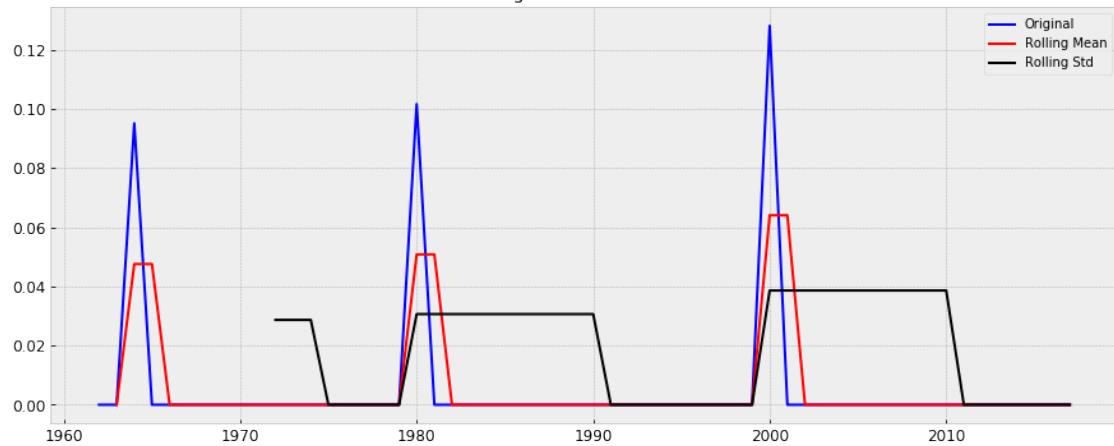
Grenada: Rolling Mean & Standard Deviation



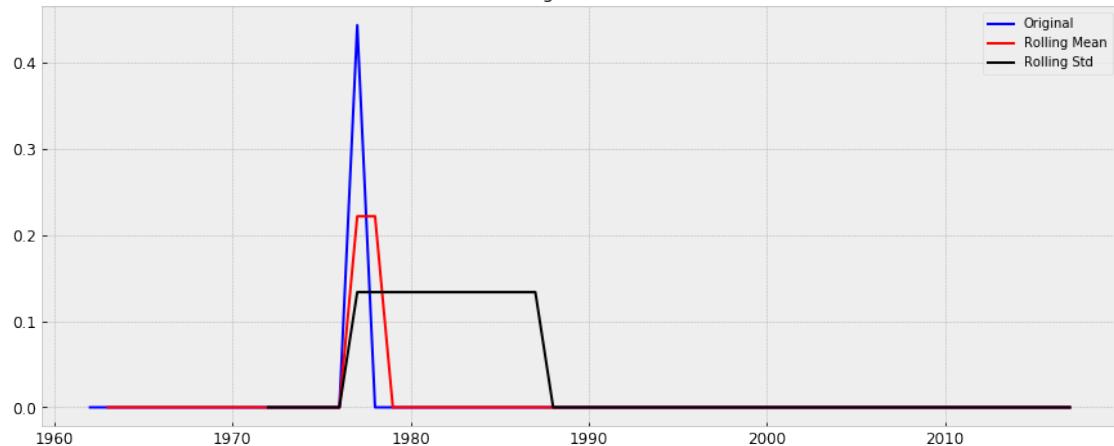
Guam: Rolling Mean & Standard Deviation



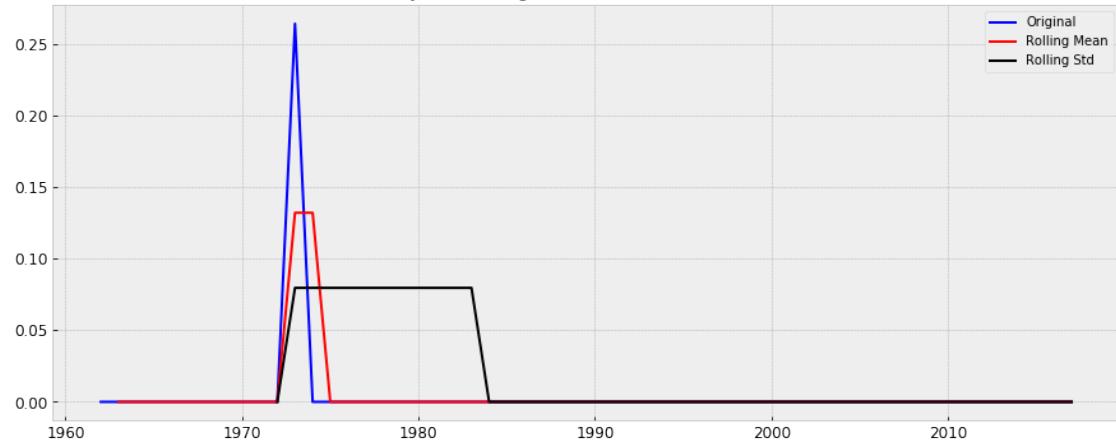
Guinea: Rolling Mean & Standard Deviation



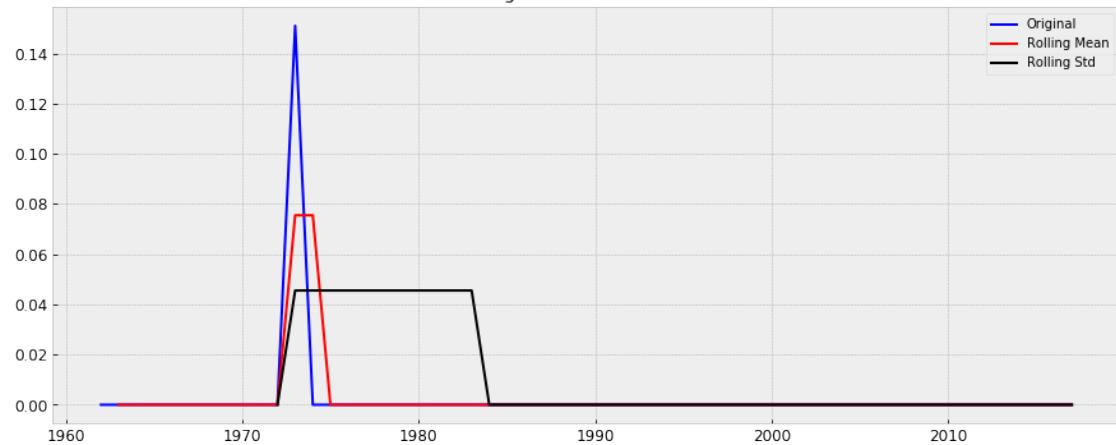
Guinea-Bissau: Rolling Mean & Standard Deviation



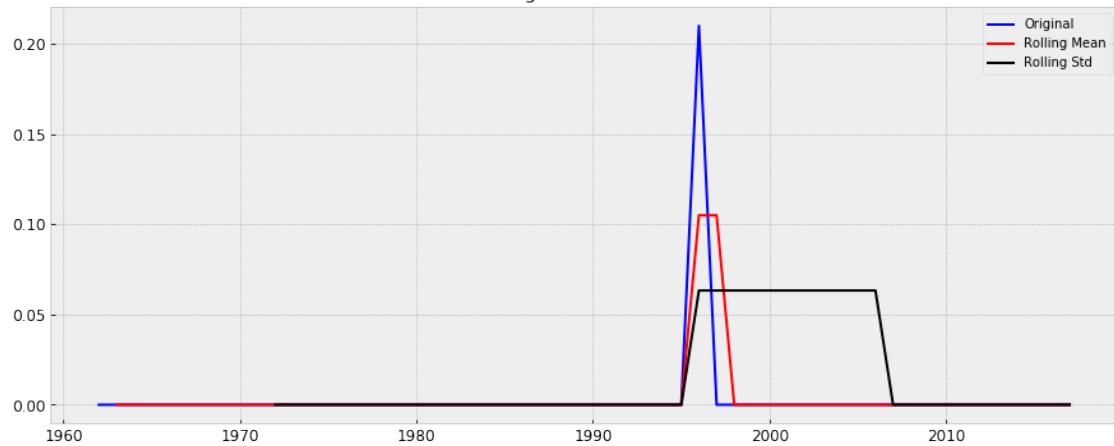
Guyana: Rolling Mean & Standard Deviation



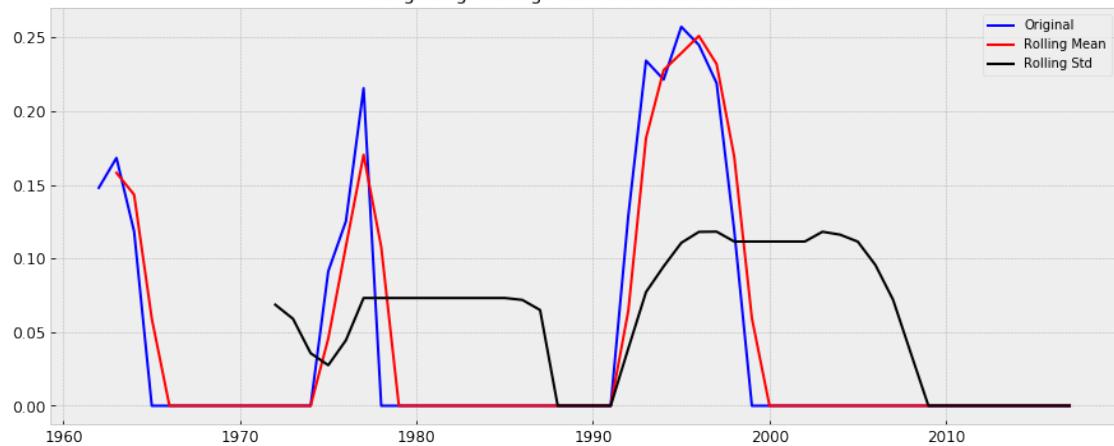
Haiti: Rolling Mean & Standard Deviation



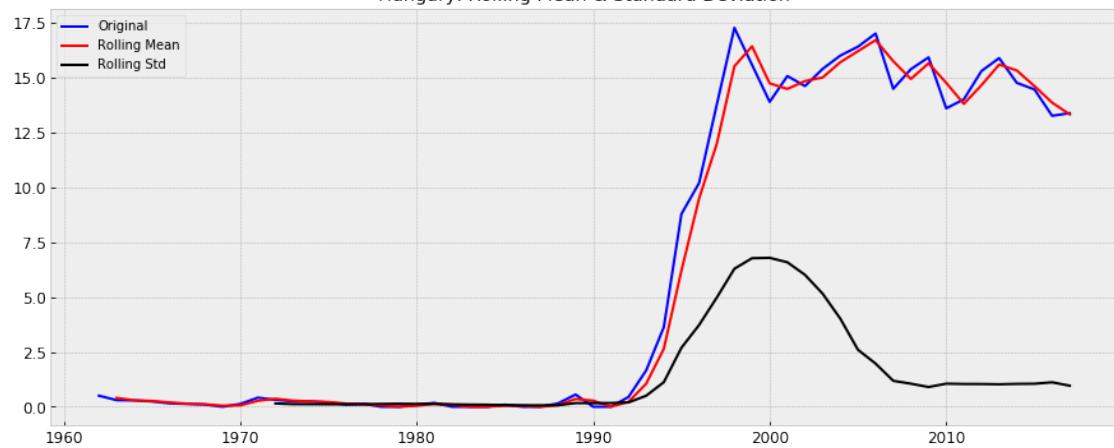
Honduras: Rolling Mean & Standard Deviation



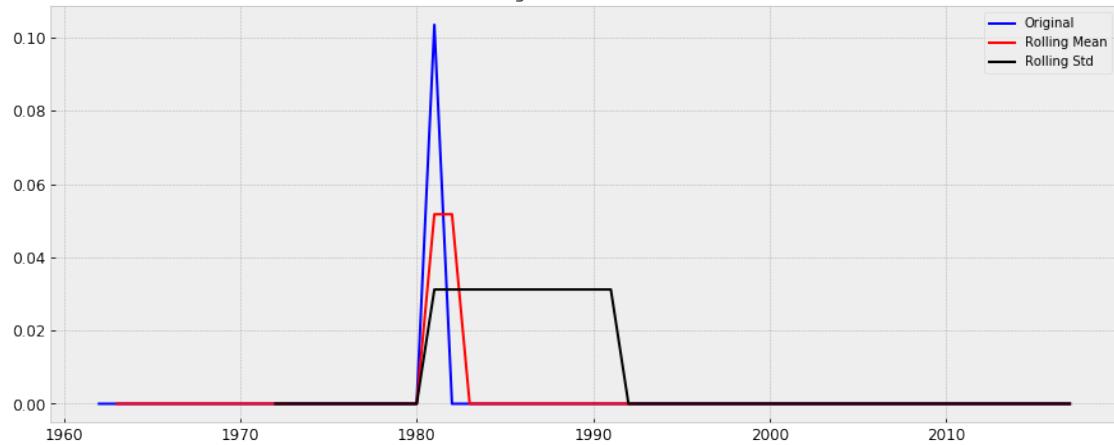
Hong Kong: Rolling Mean & Standard Deviation



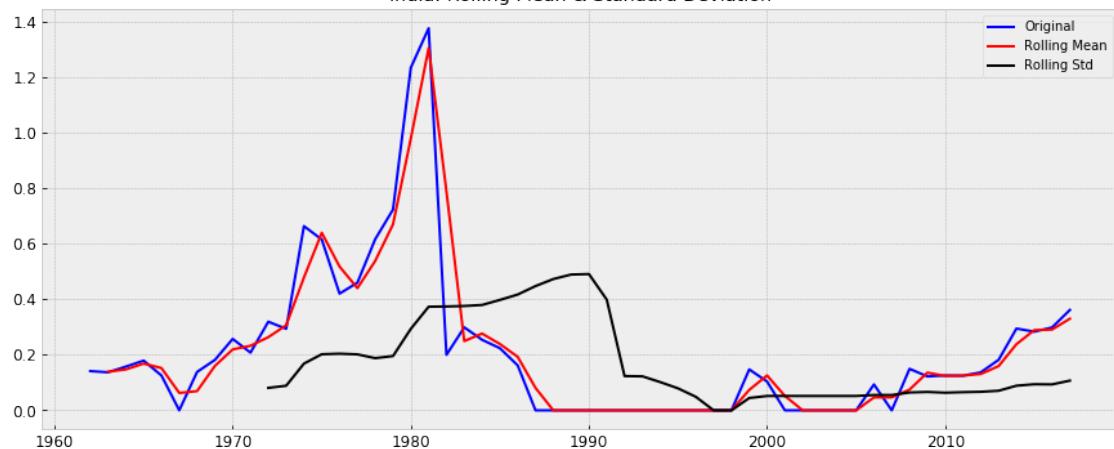
Hungary: Rolling Mean & Standard Deviation



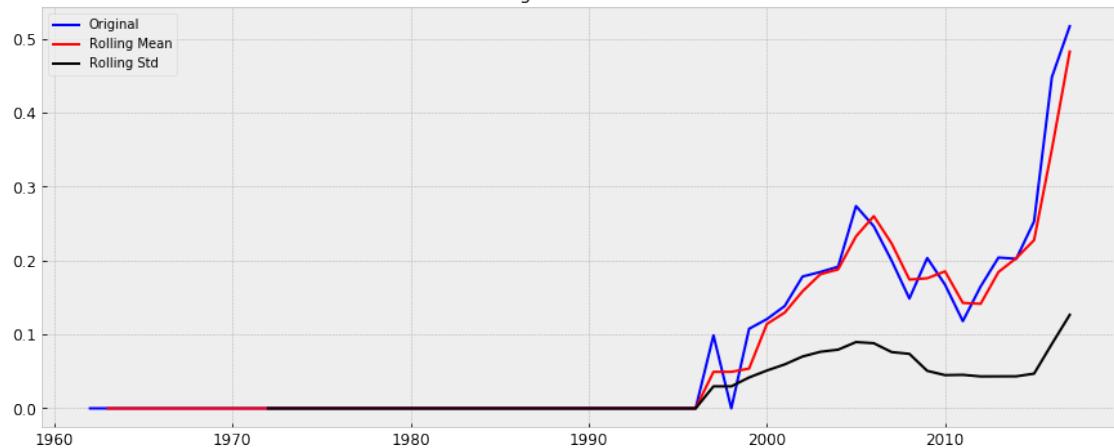
Iceland: Rolling Mean & Standard Deviation



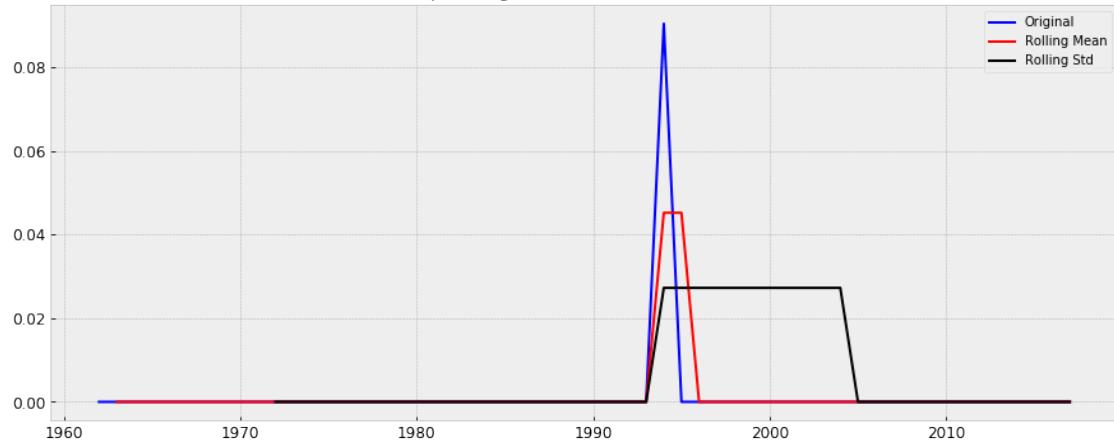
India: Rolling Mean & Standard Deviation



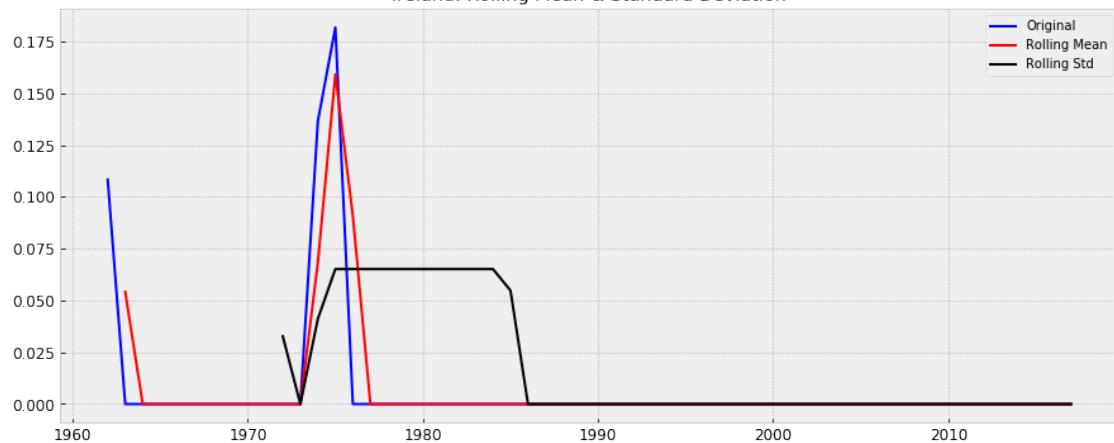
Indonesia: Rolling Mean & Standard Deviation



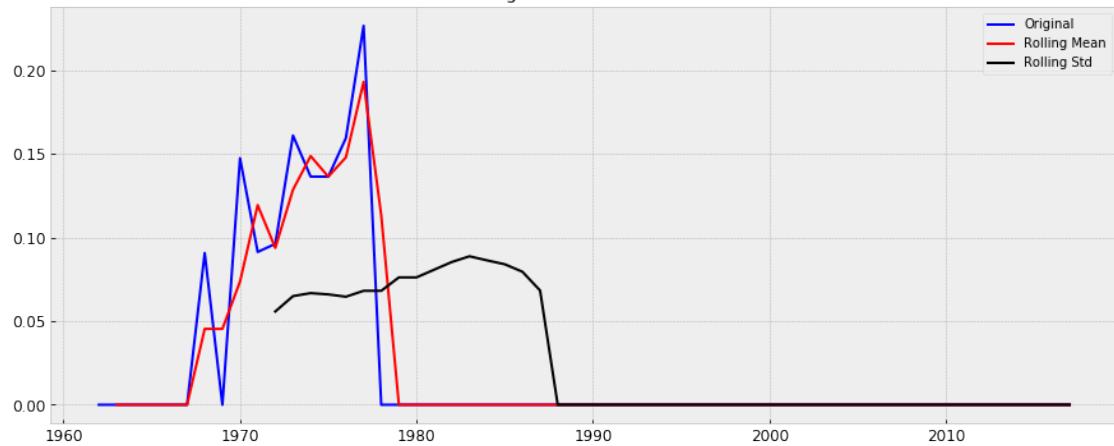
Iraq: Rolling Mean & Standard Deviation



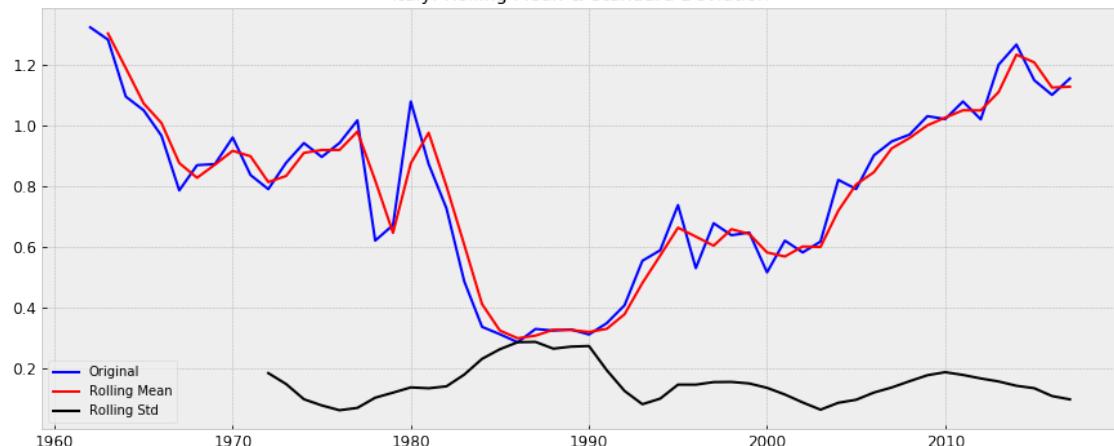
Ireland: Rolling Mean & Standard Deviation



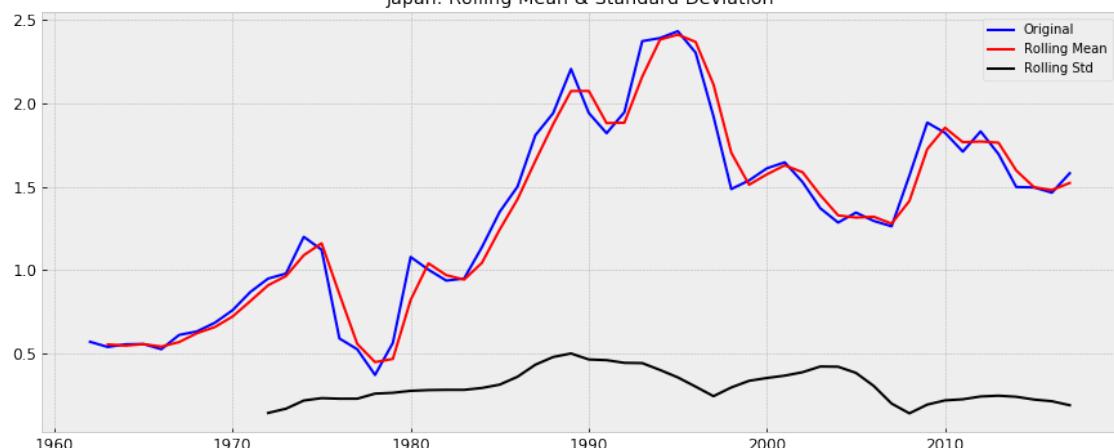
Israel: Rolling Mean & Standard Deviation



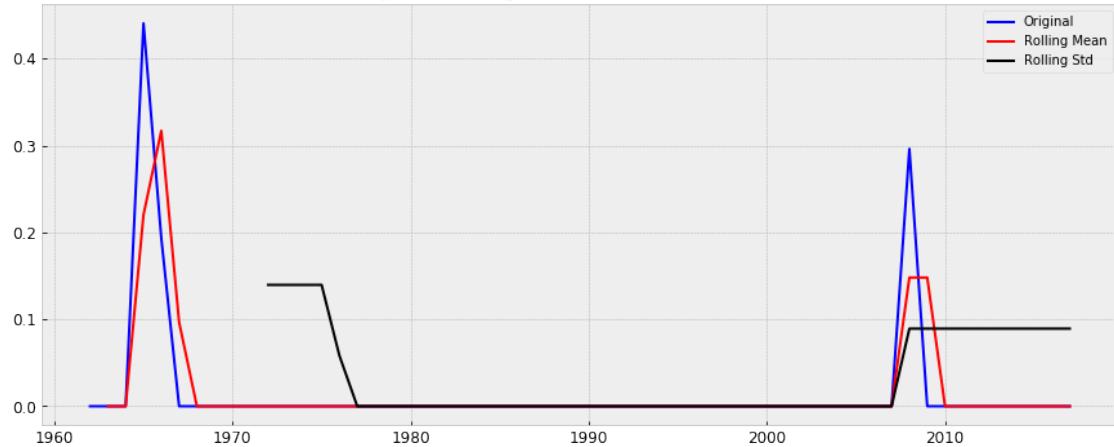
Italy: Rolling Mean & Standard Deviation



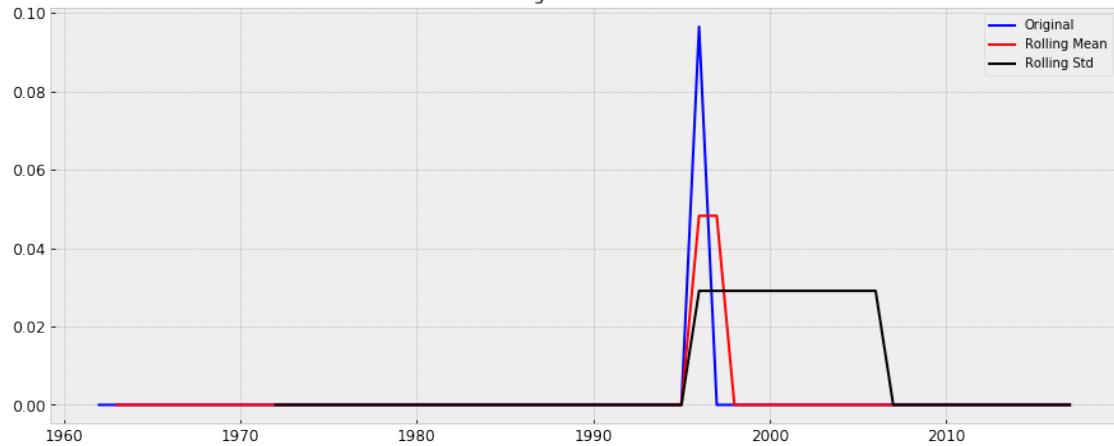
Japan: Rolling Mean & Standard Deviation



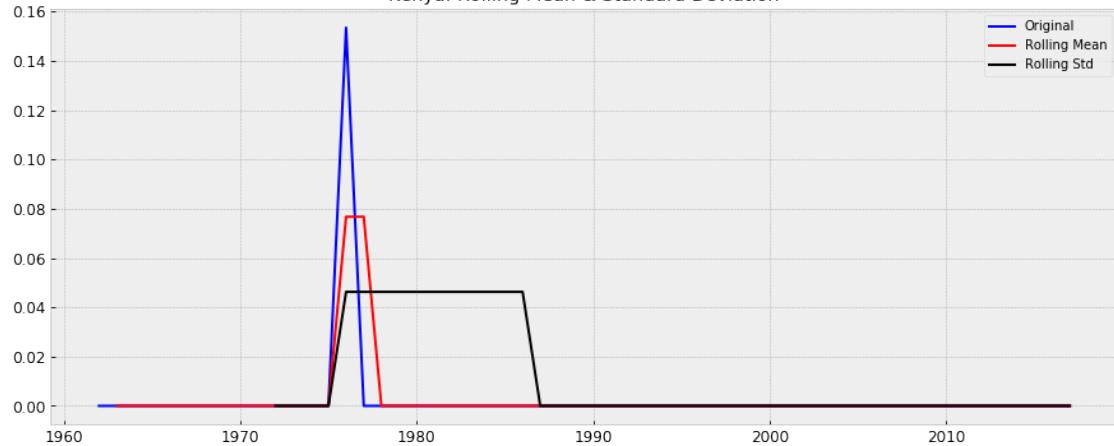
Jordan: Rolling Mean & Standard Deviation



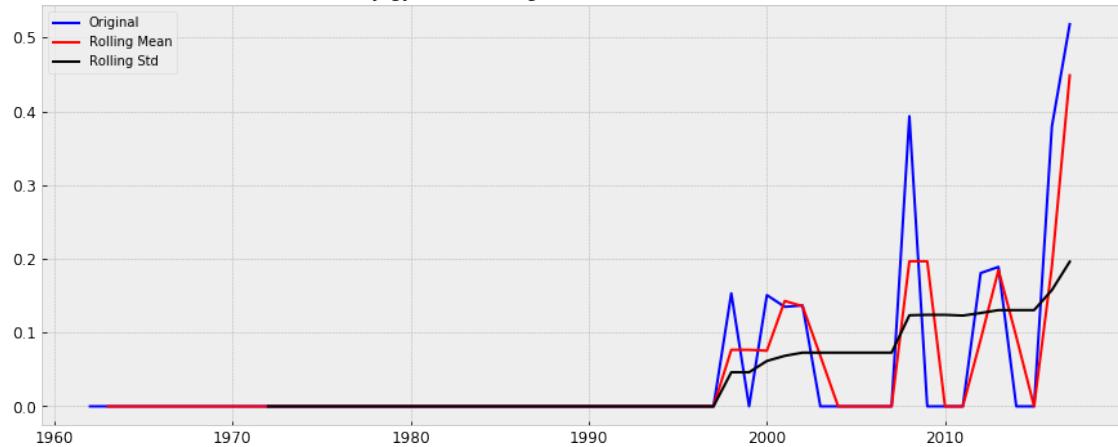
Kazakhstan: Rolling Mean & Standard Deviation



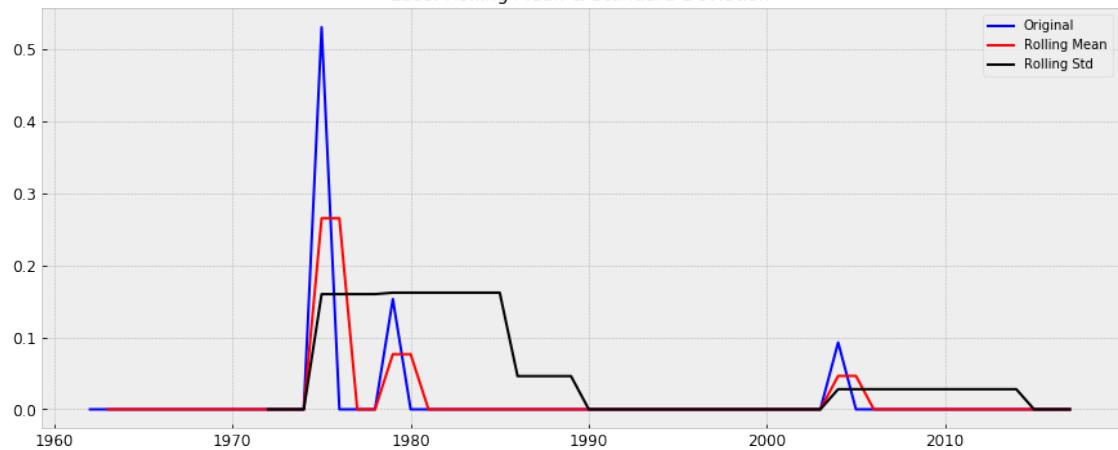
Kenya: Rolling Mean & Standard Deviation



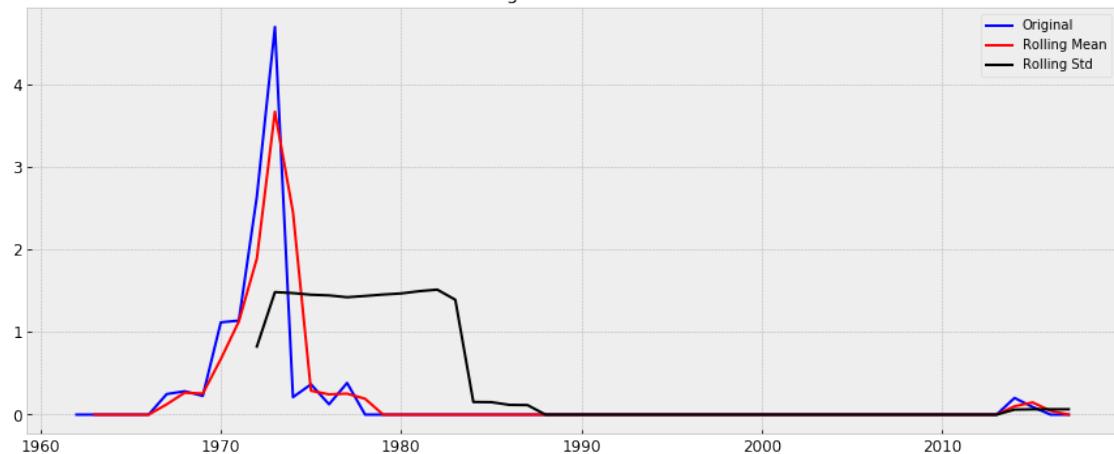
Kyrgyzstan: Rolling Mean & Standard Deviation



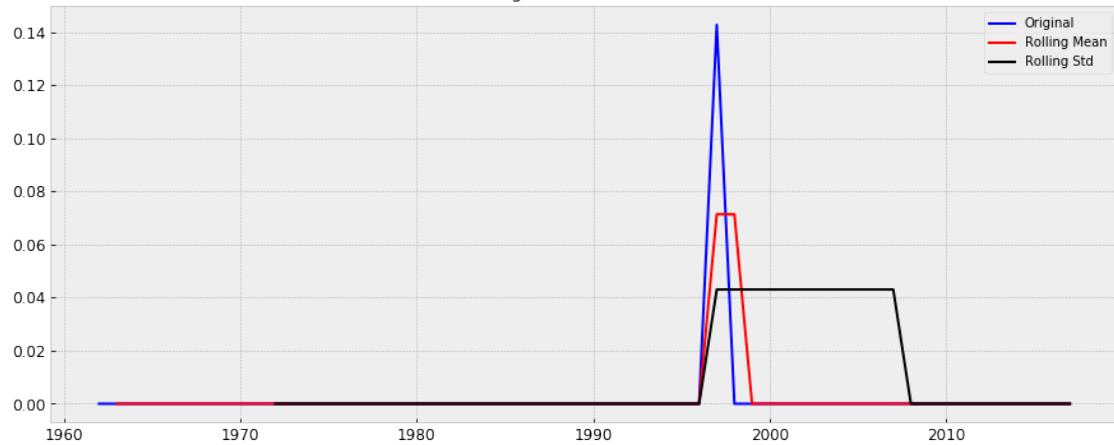
Laos: Rolling Mean & Standard Deviation



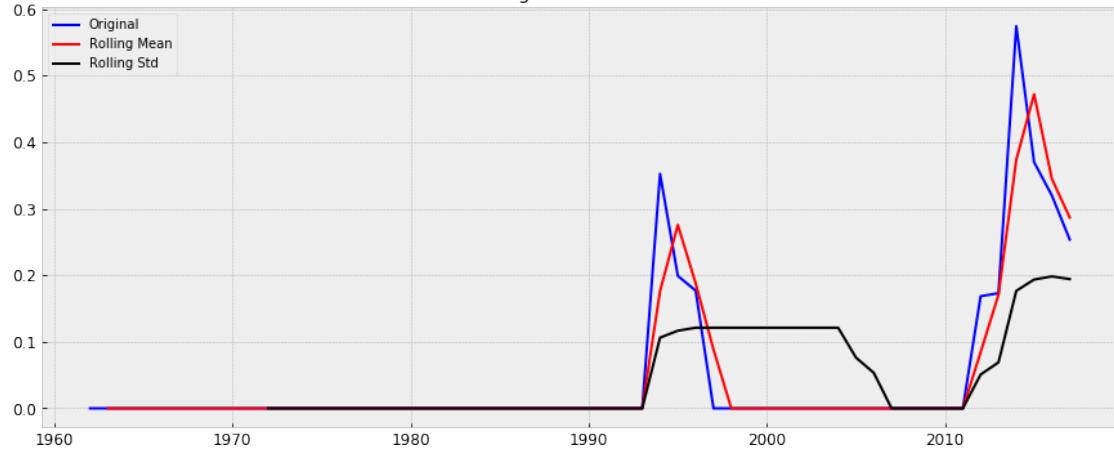
Lebanon: Rolling Mean & Standard Deviation

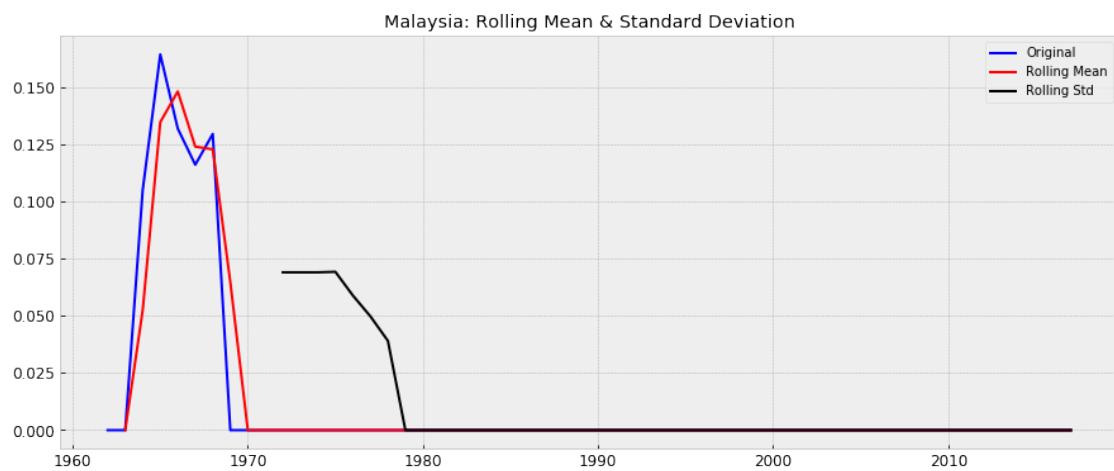
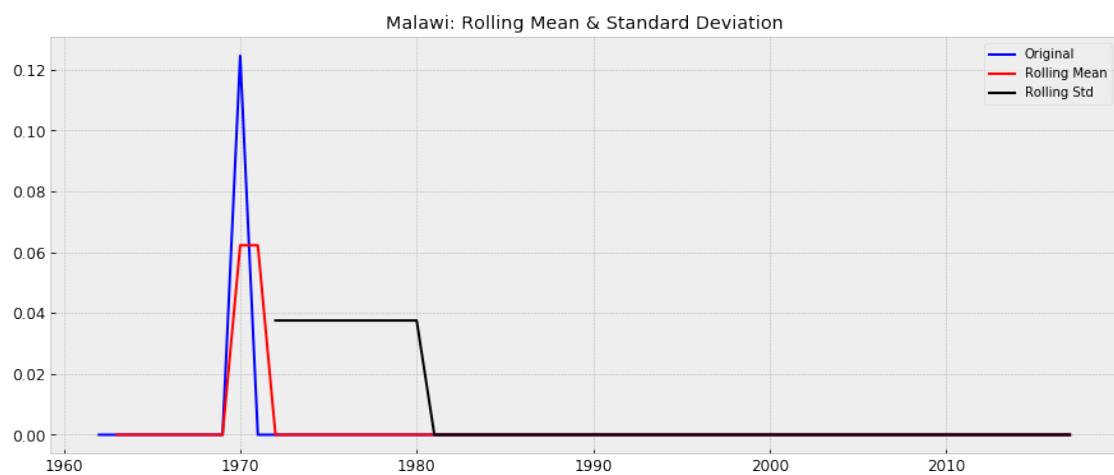
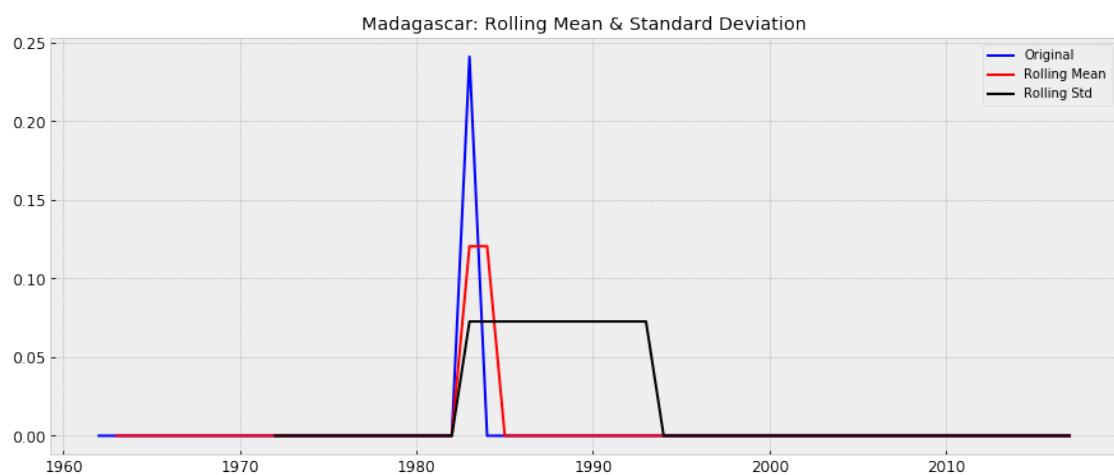


Liberia: Rolling Mean & Standard Deviation

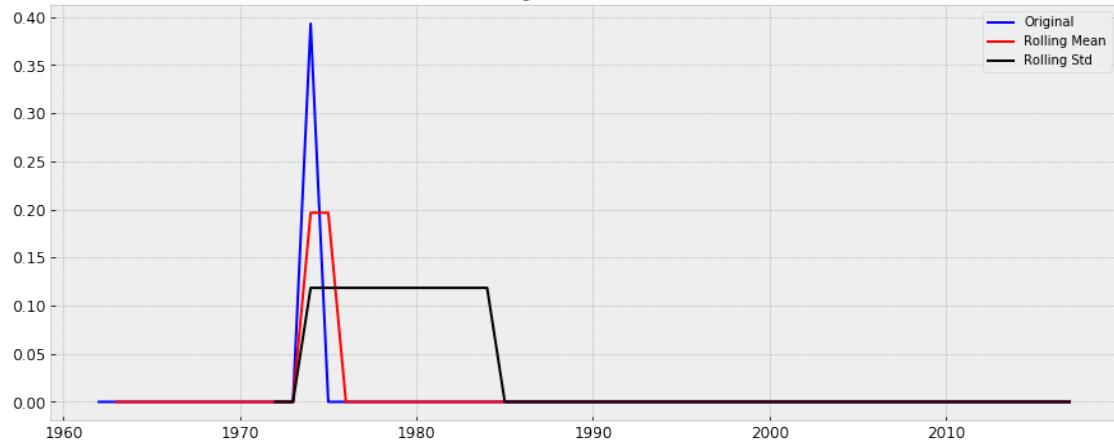


Lithuania: Rolling Mean & Standard Deviation

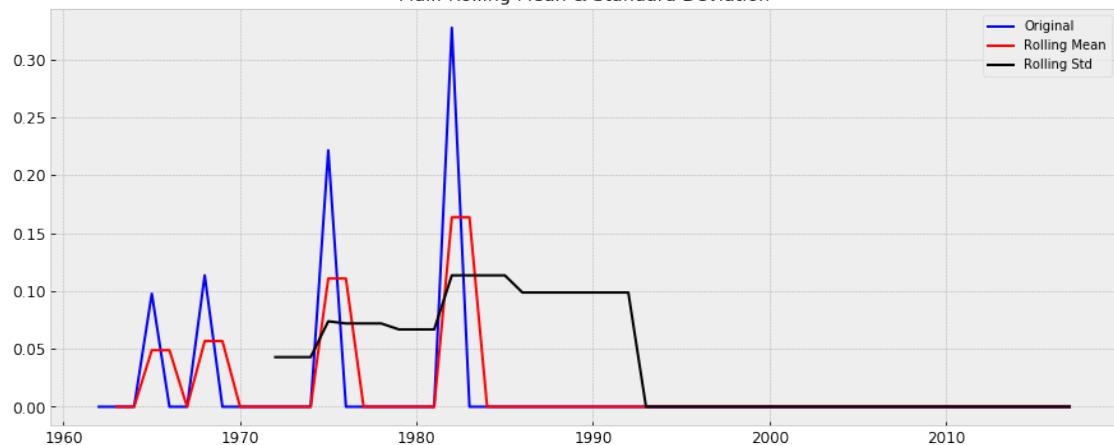




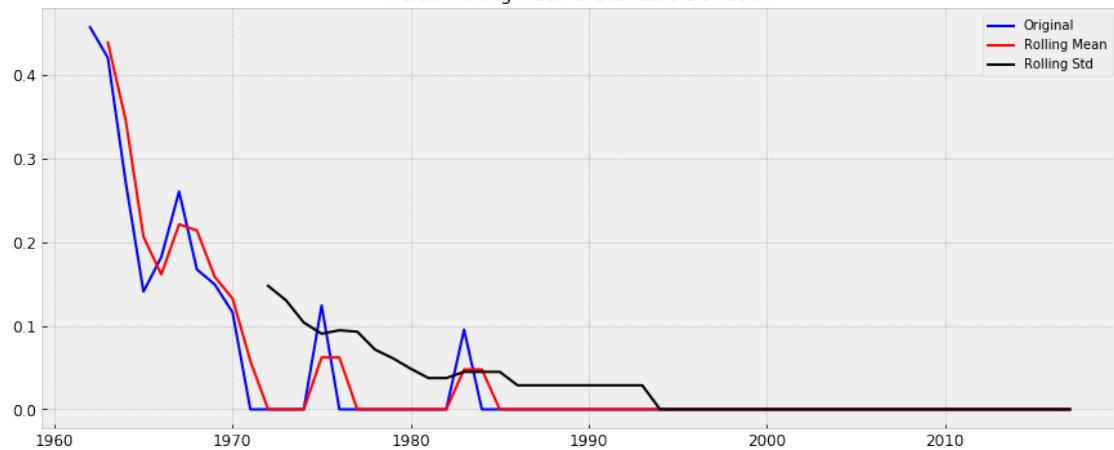
Maldives: Rolling Mean & Standard Deviation



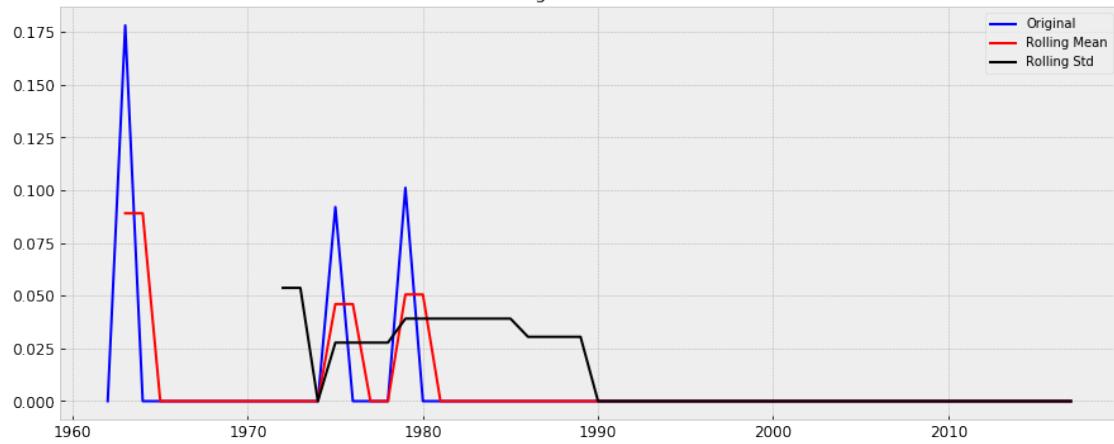
Mali: Rolling Mean & Standard Deviation



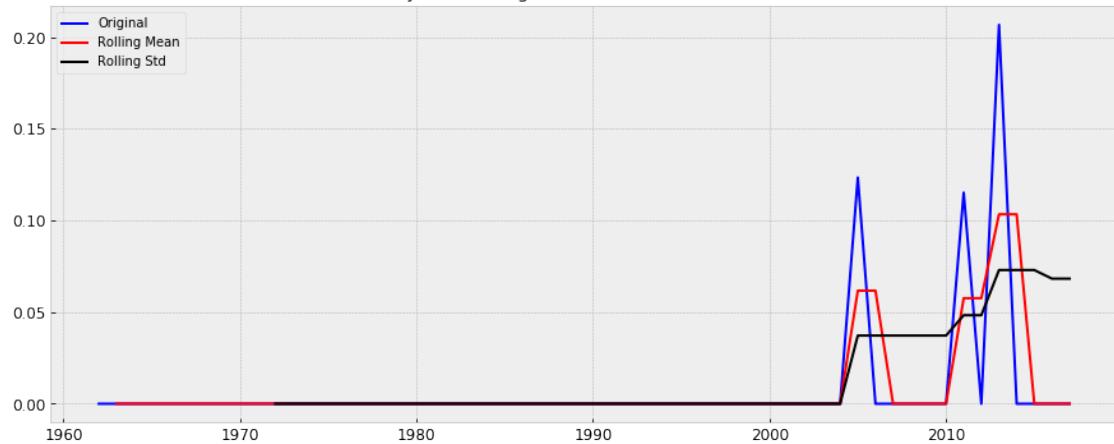
Malta: Rolling Mean & Standard Deviation

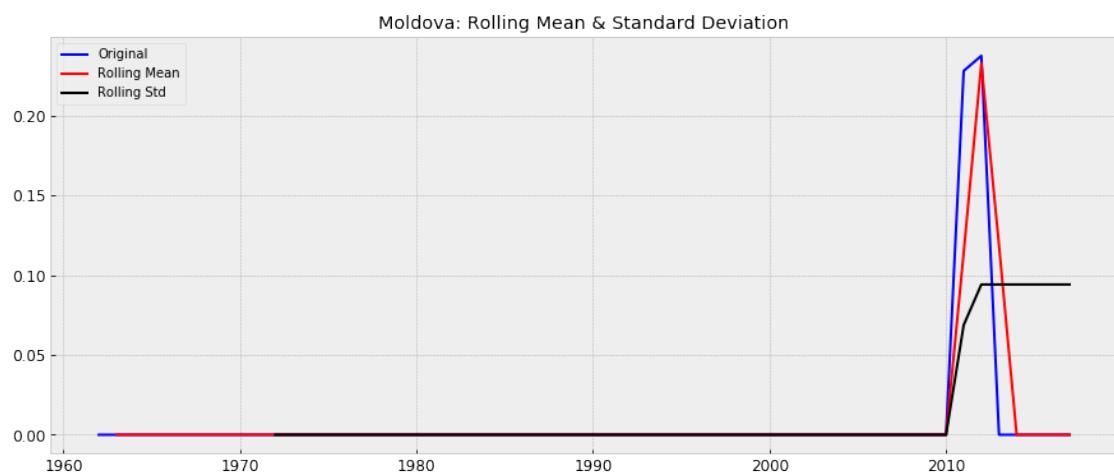
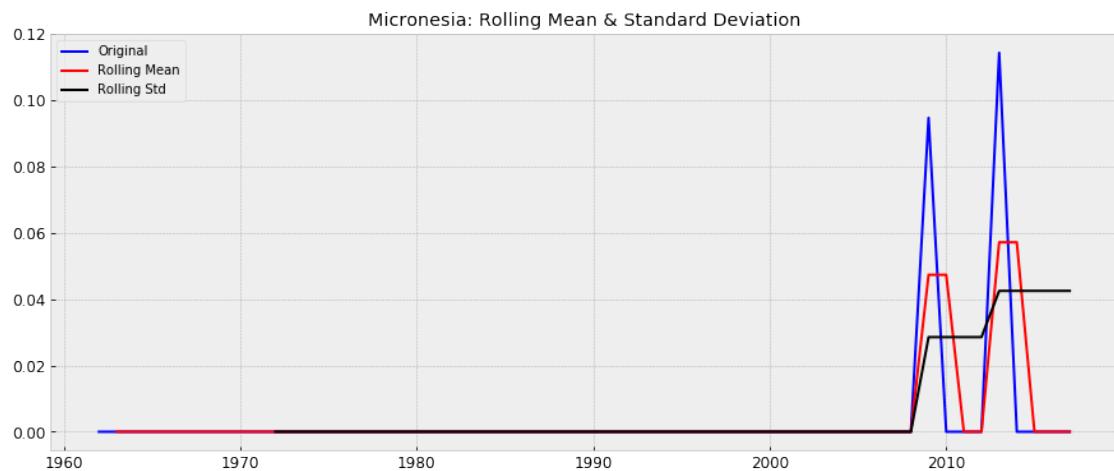
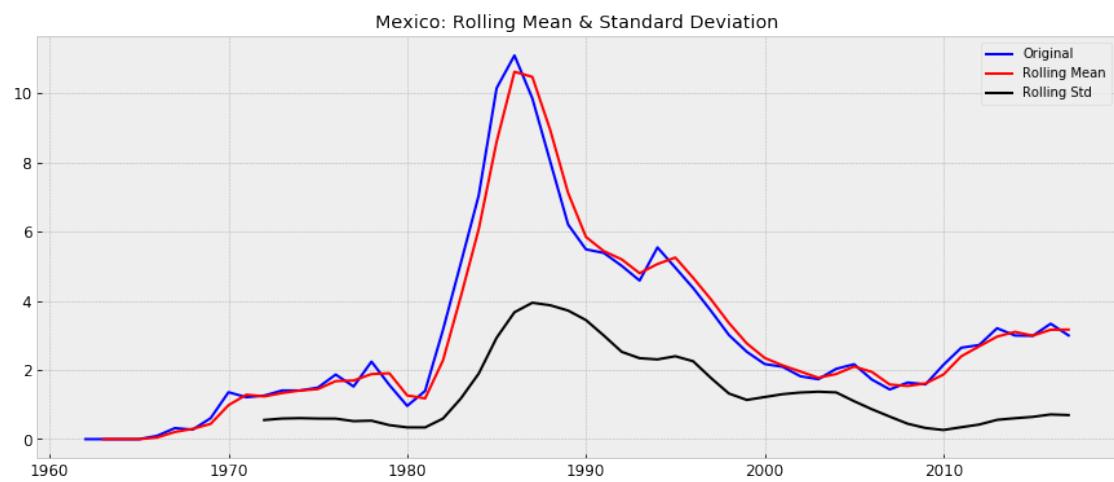


Mauritania: Rolling Mean & Standard Deviation

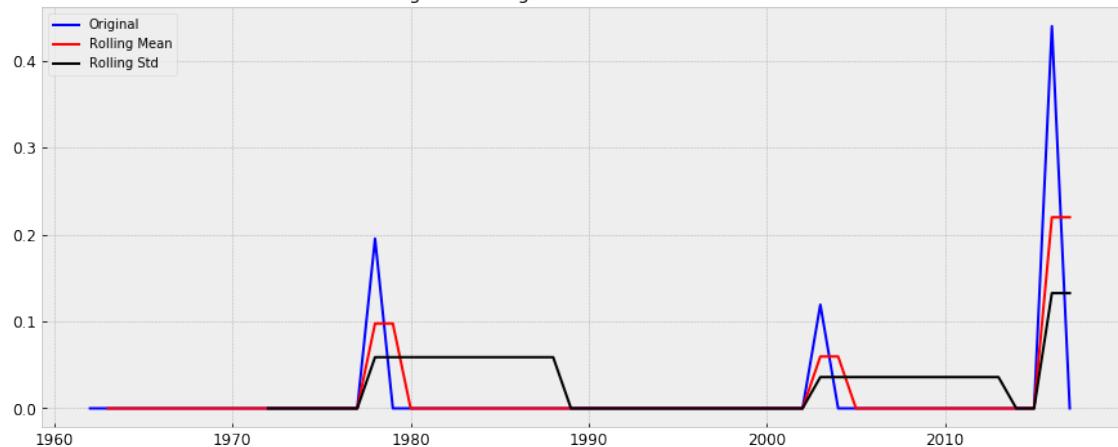


Mayotte: Rolling Mean & Standard Deviation

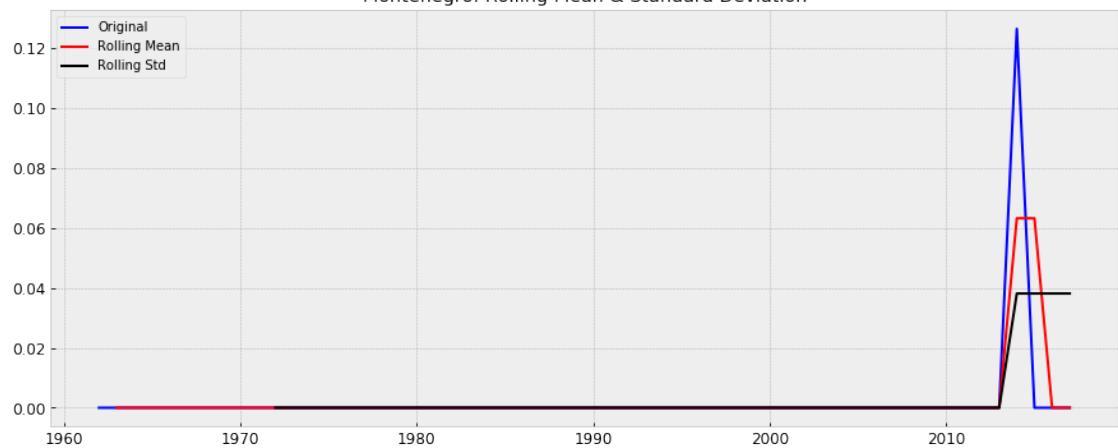




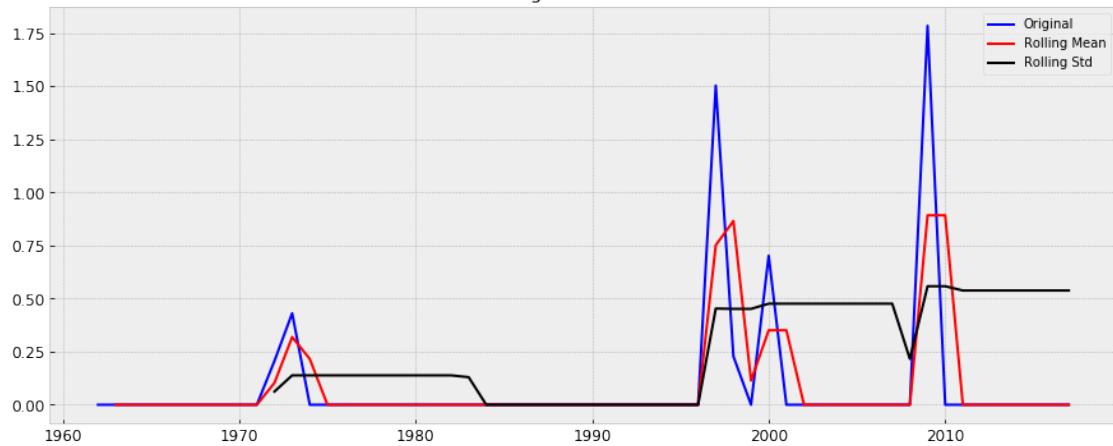
Mongolia: Rolling Mean & Standard Deviation



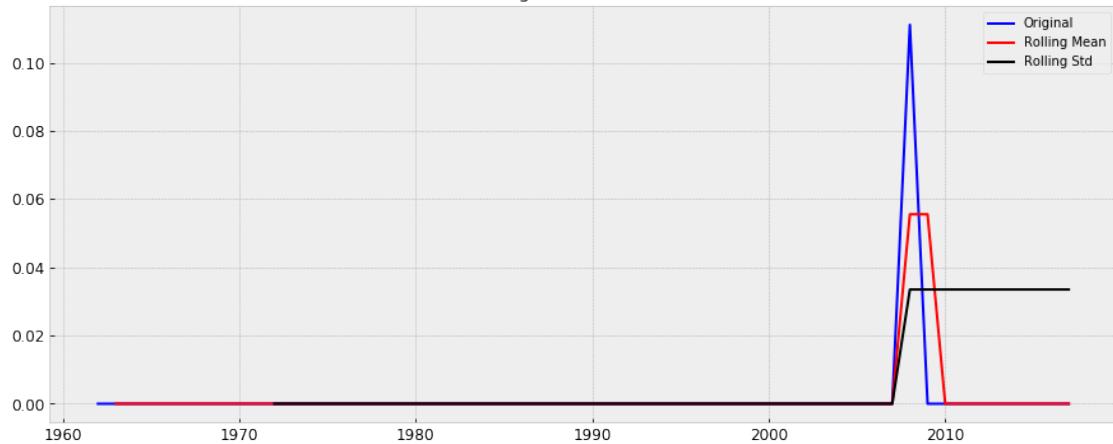
Montenegro: Rolling Mean & Standard Deviation



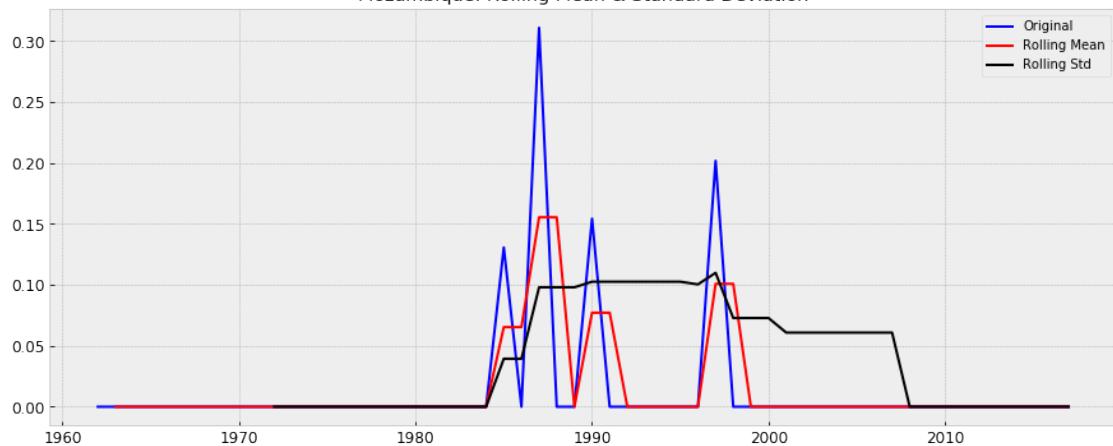
Montserrat: Rolling Mean & Standard Deviation

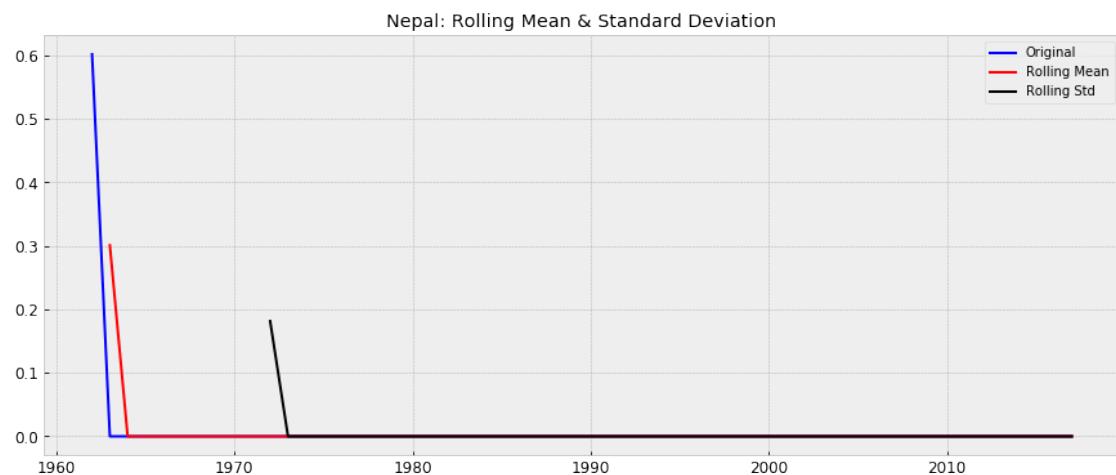
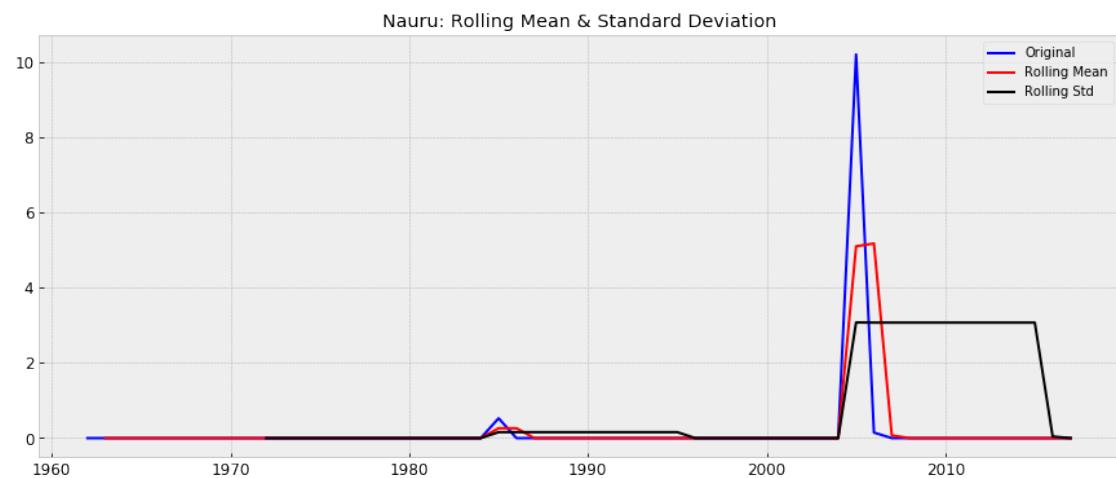
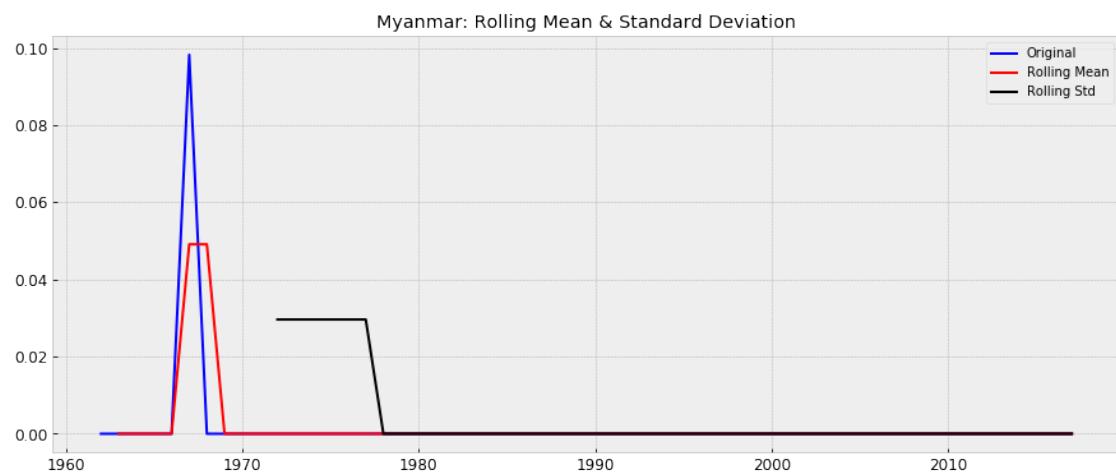


Morocco: Rolling Mean & Standard Deviation

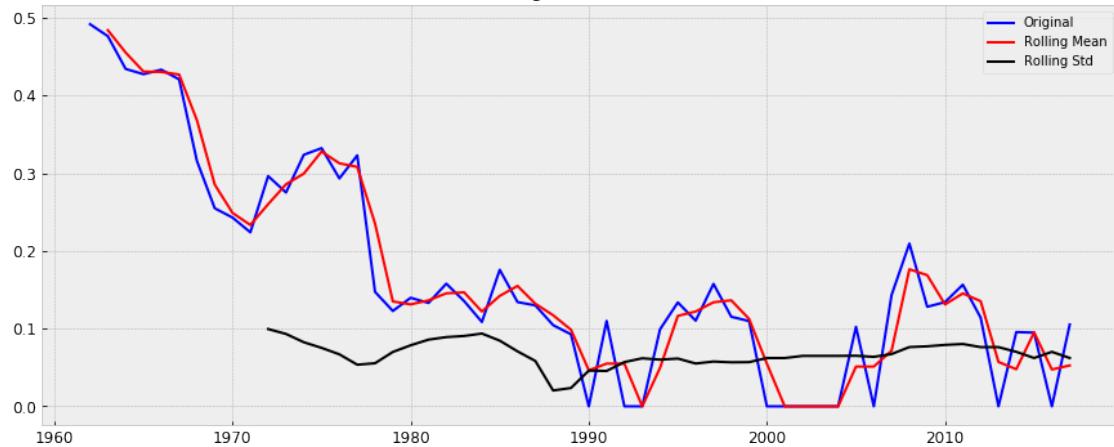


Mozambique: Rolling Mean & Standard Deviation

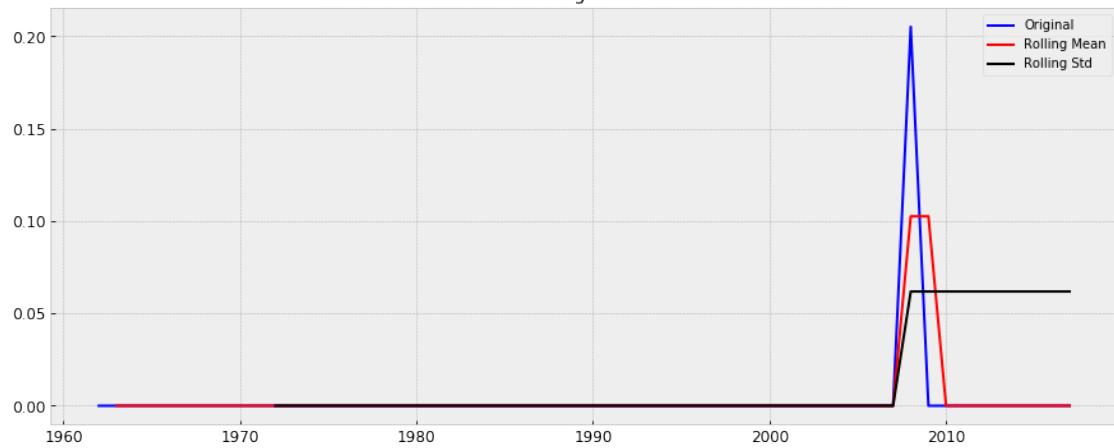




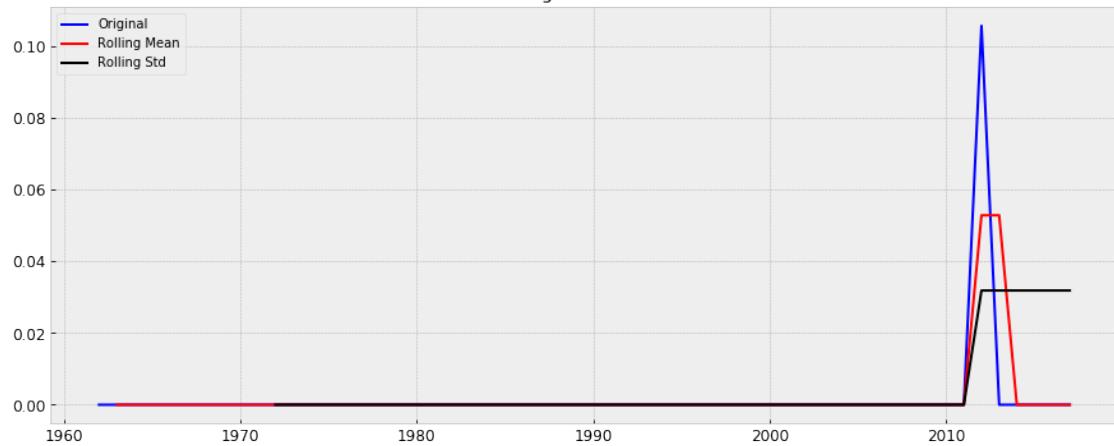
Netherlands: Rolling Mean & Standard Deviation



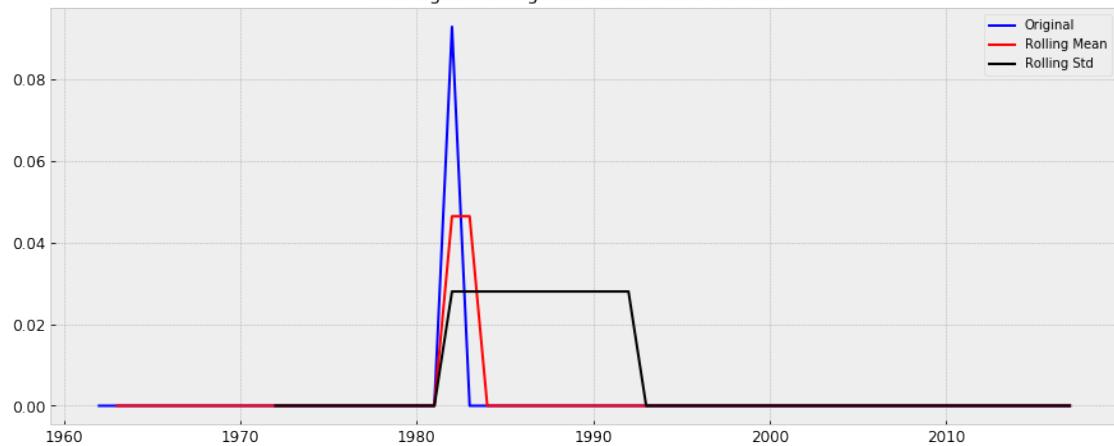
Netherlands Antilles: Rolling Mean & Standard Deviation



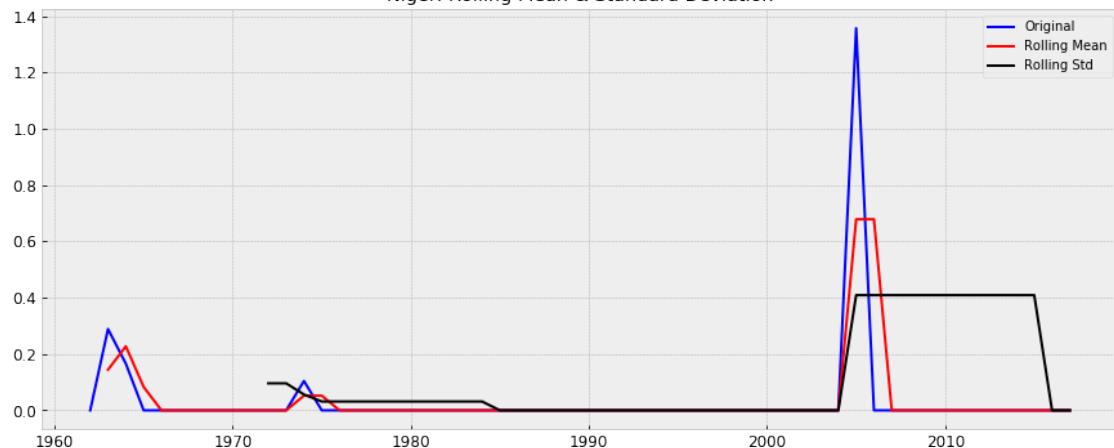
New Zealand: Rolling Mean & Standard Deviation



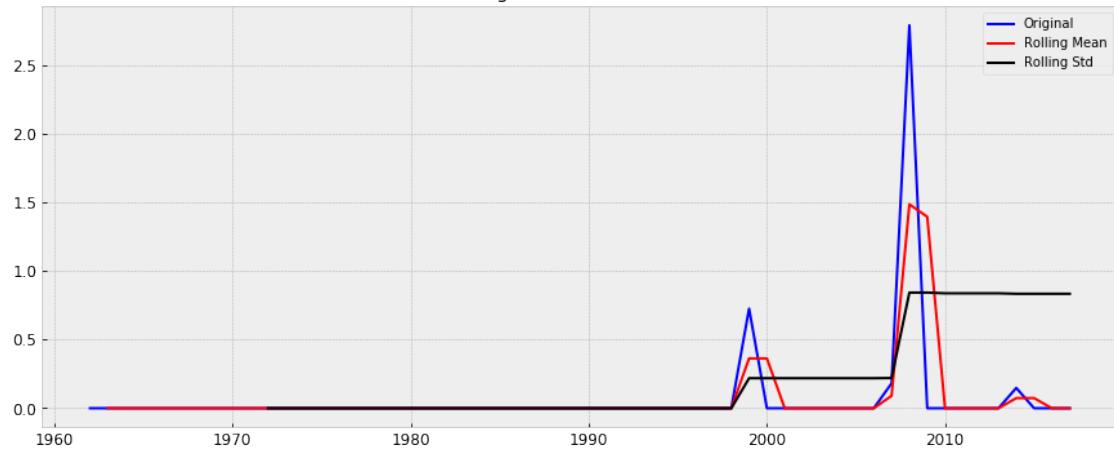
Nicaragua: Rolling Mean & Standard Deviation



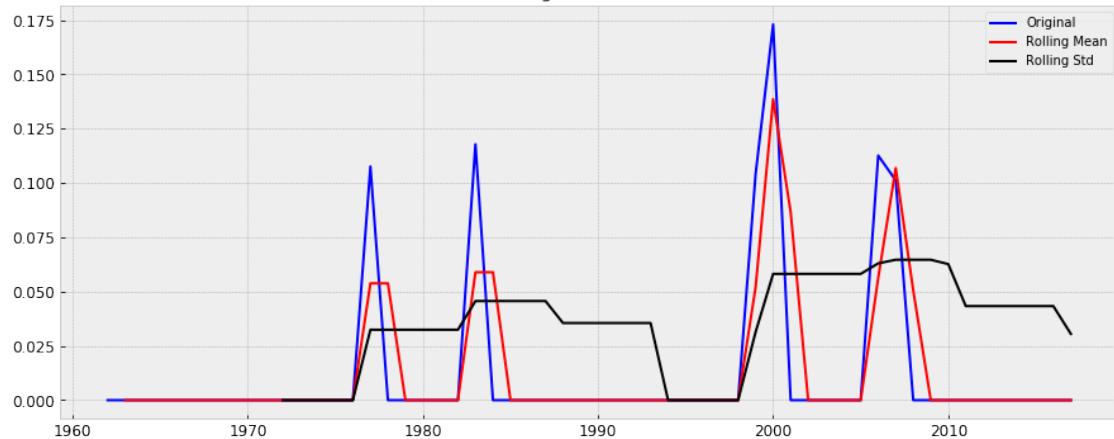
Niger: Rolling Mean & Standard Deviation



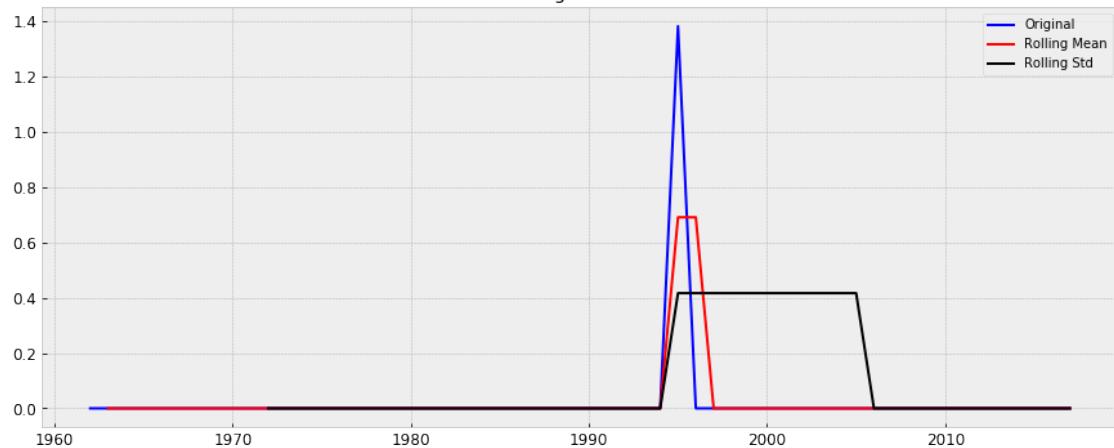
Niue: Rolling Mean & Standard Deviation



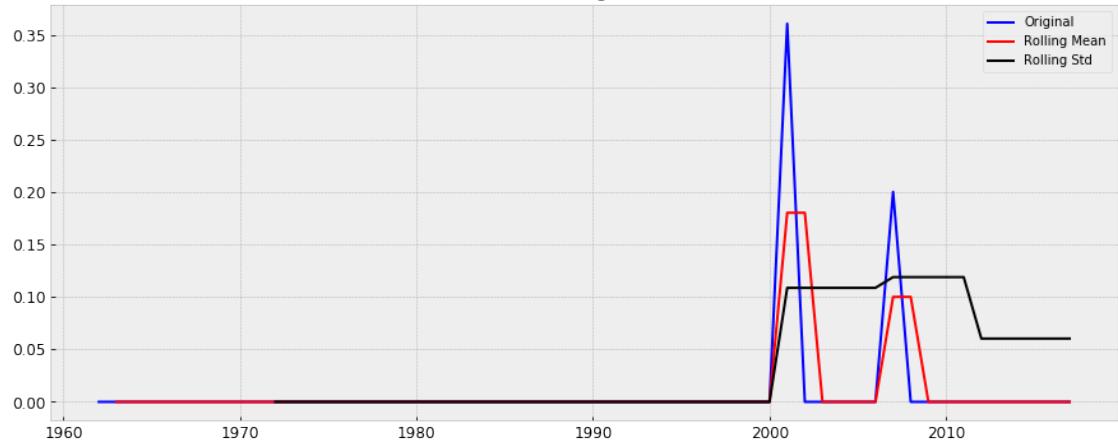
North Korea: Rolling Mean & Standard Deviation



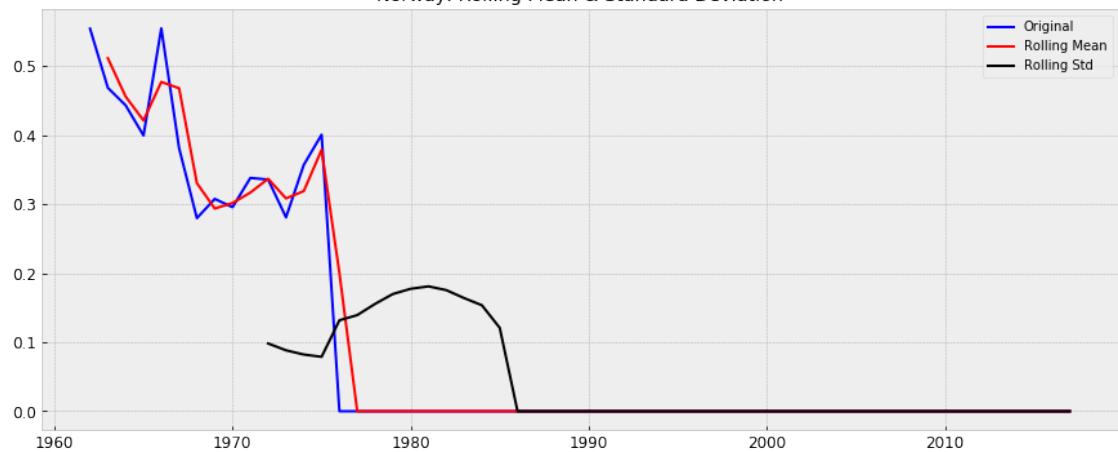
North Macedonia: Rolling Mean & Standard Deviation

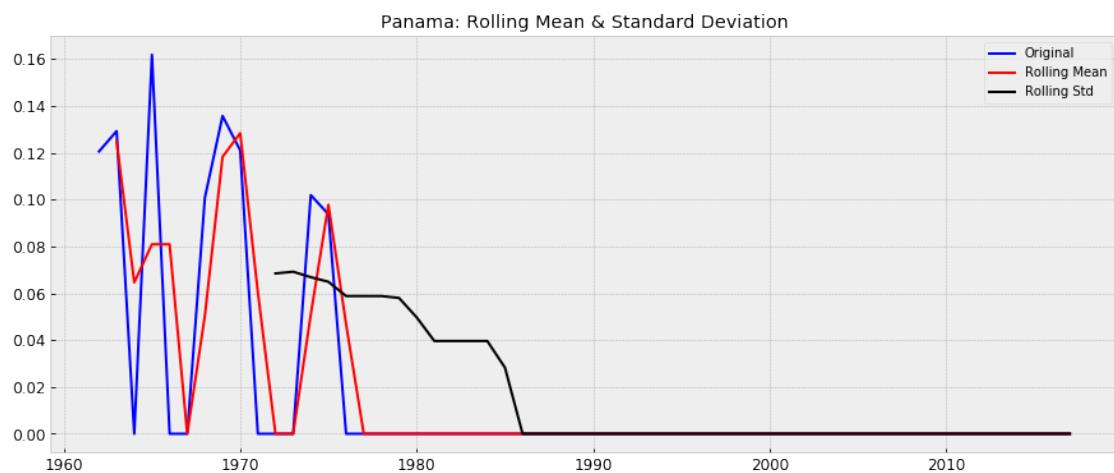
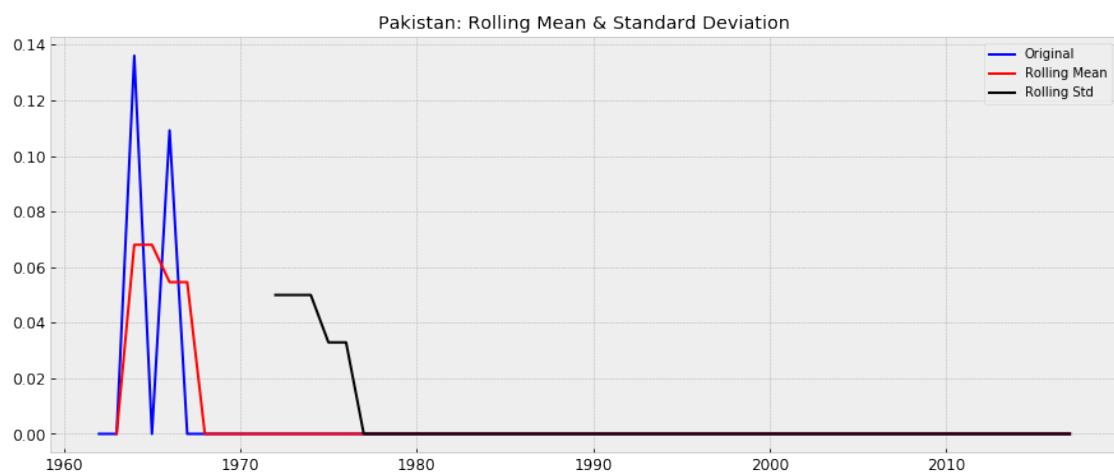
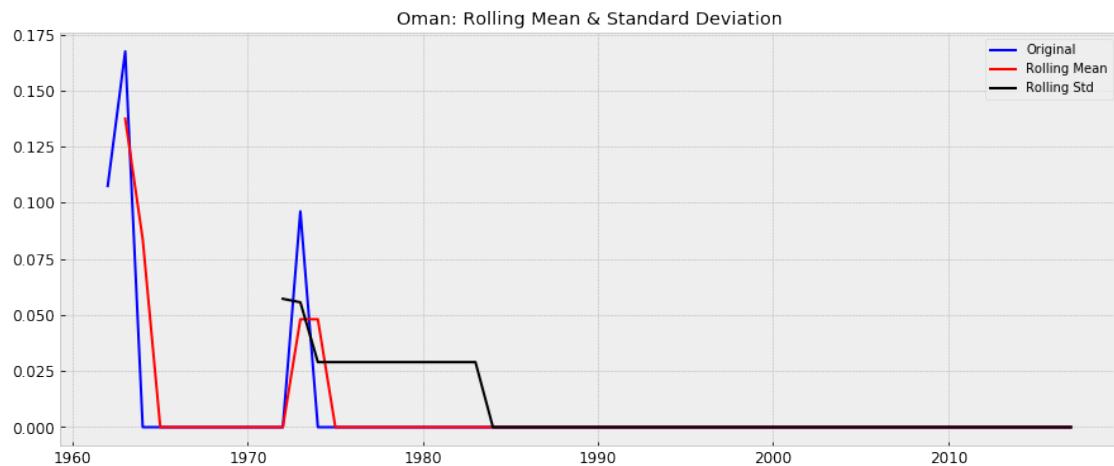


Northern Mariana Islands: Rolling Mean & Standard Deviation

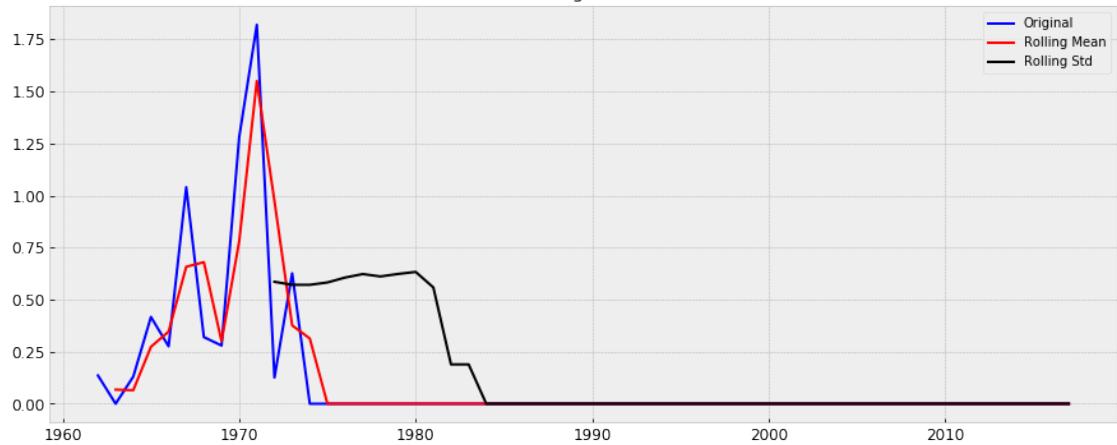


Norway: Rolling Mean & Standard Deviation

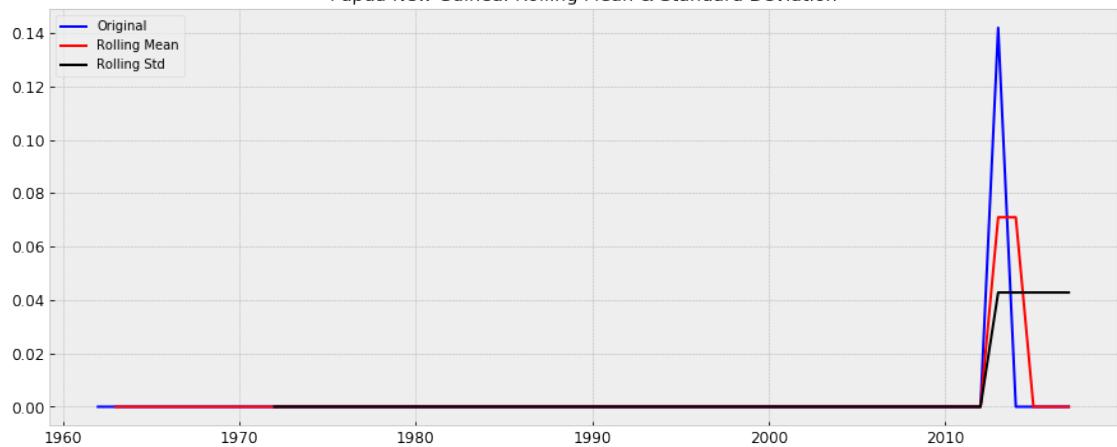




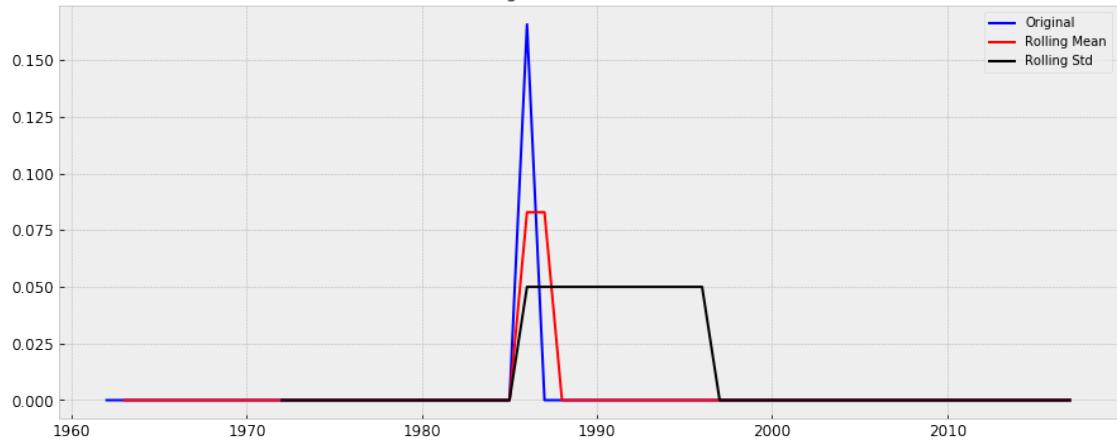
Panama Canal Zone: Rolling Mean & Standard Deviation

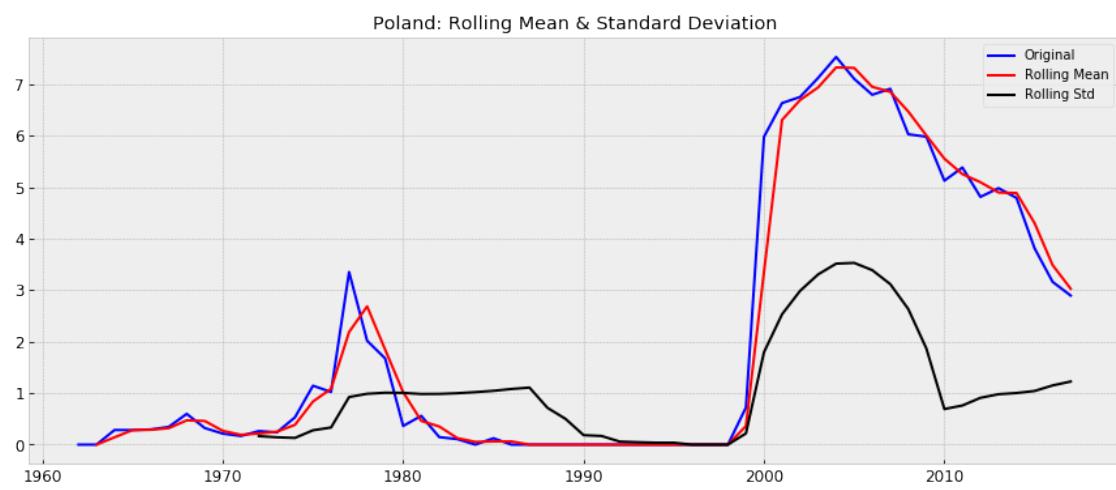
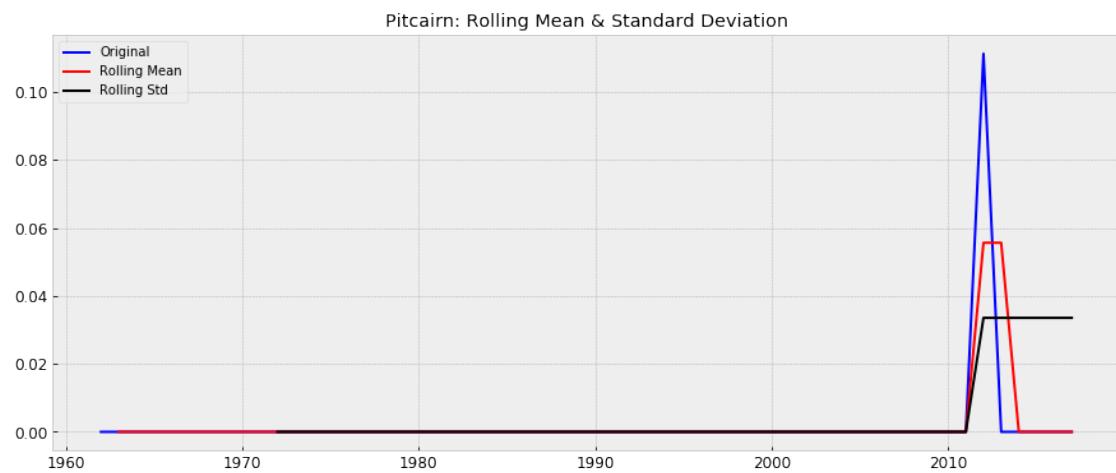


Papua New Guinea: Rolling Mean & Standard Deviation

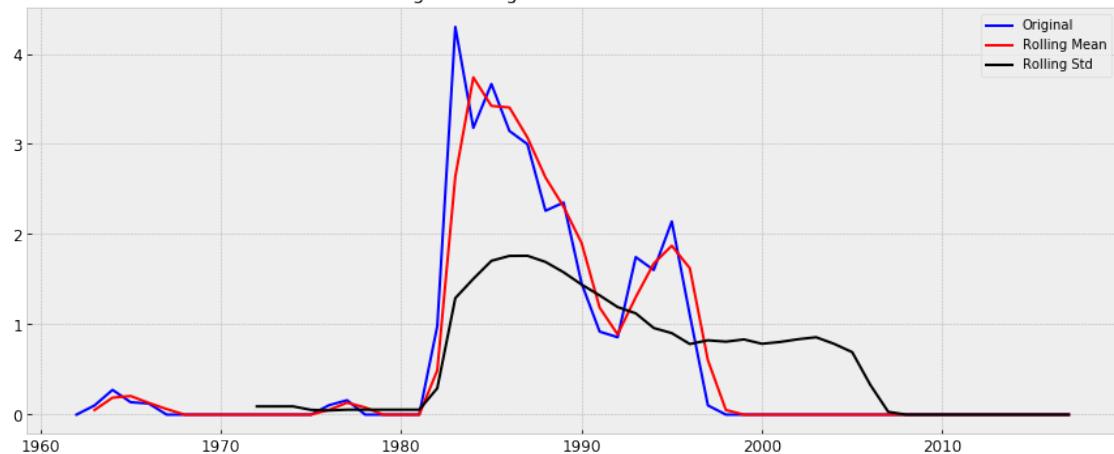


Peru: Rolling Mean & Standard Deviation

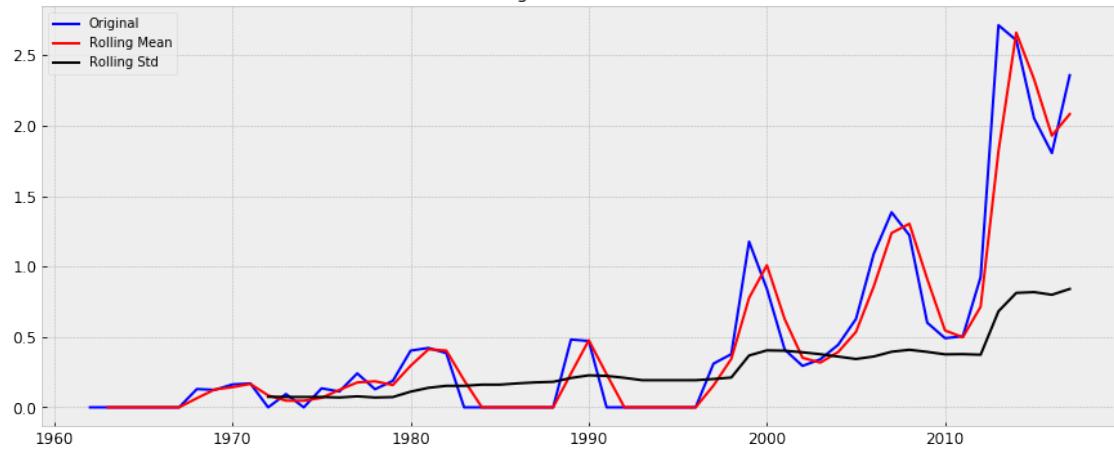




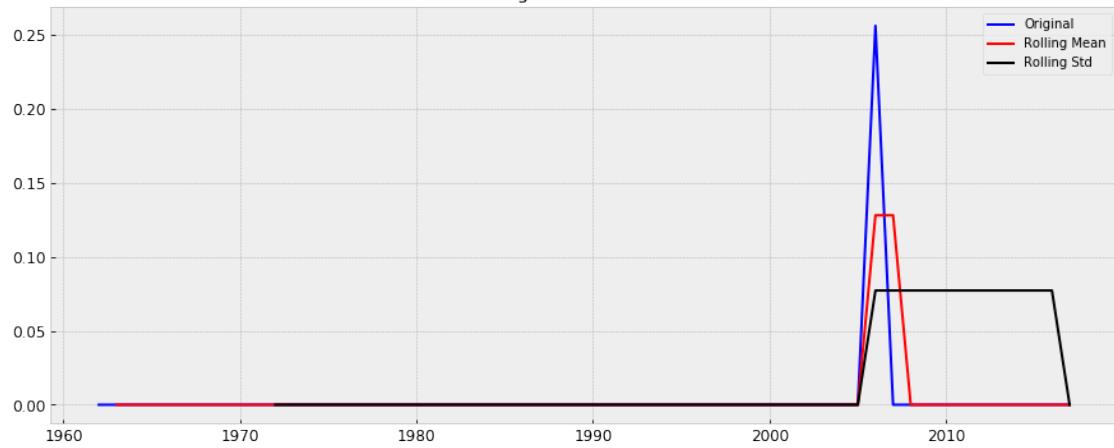
Portugal: Rolling Mean & Standard Deviation

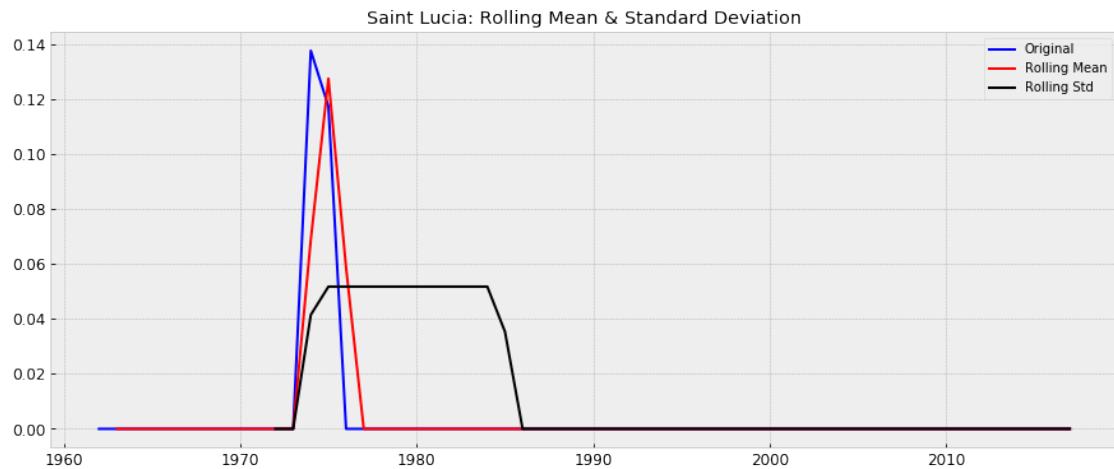
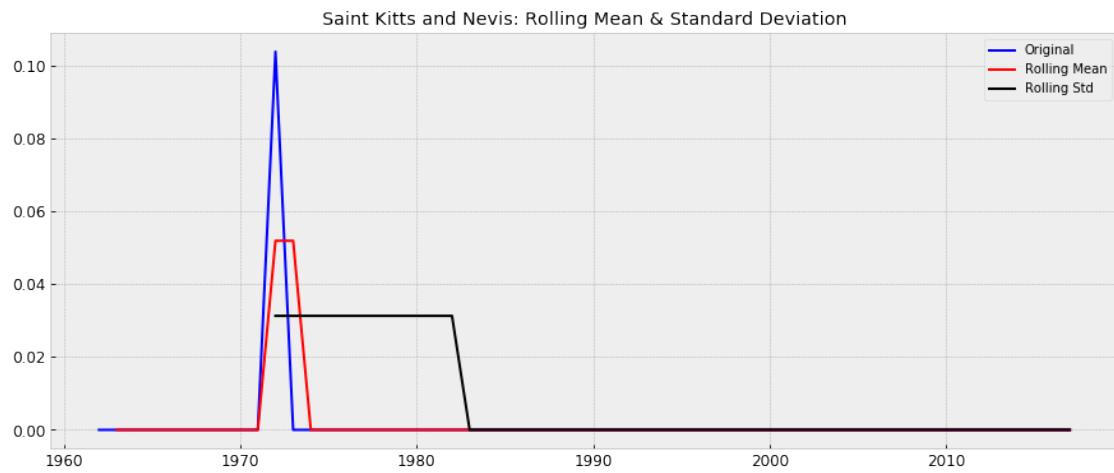
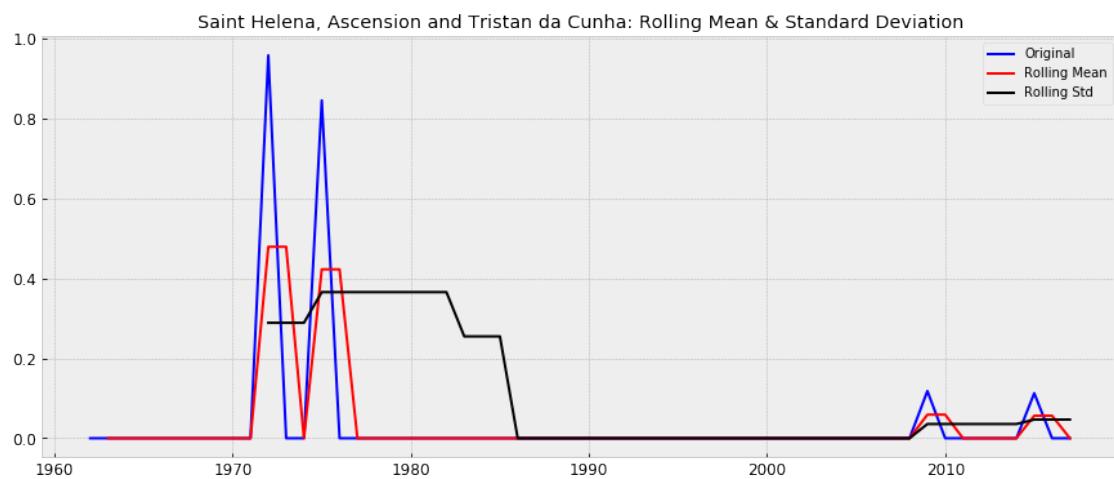


Romania: Rolling Mean & Standard Deviation

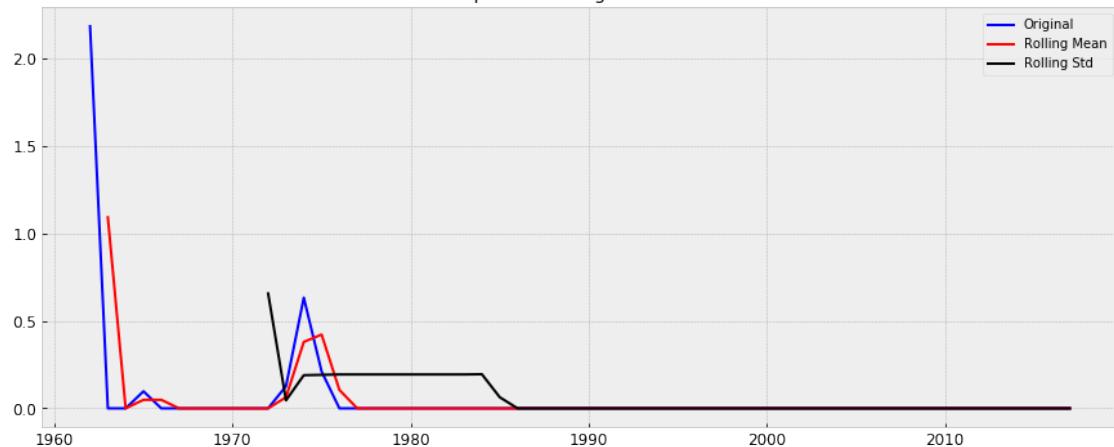


Rwanda: Rolling Mean & Standard Deviation

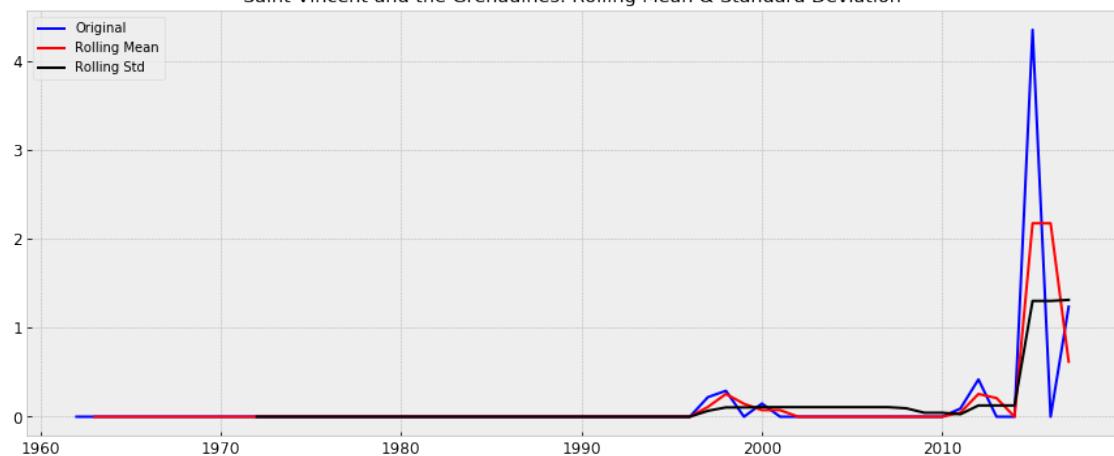




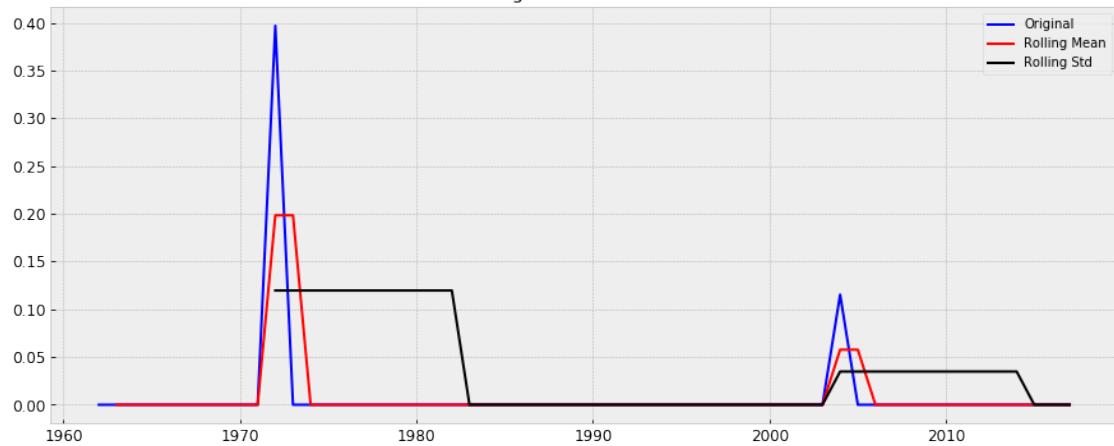
Saint Pierre and Miquelon: Rolling Mean & Standard Deviation



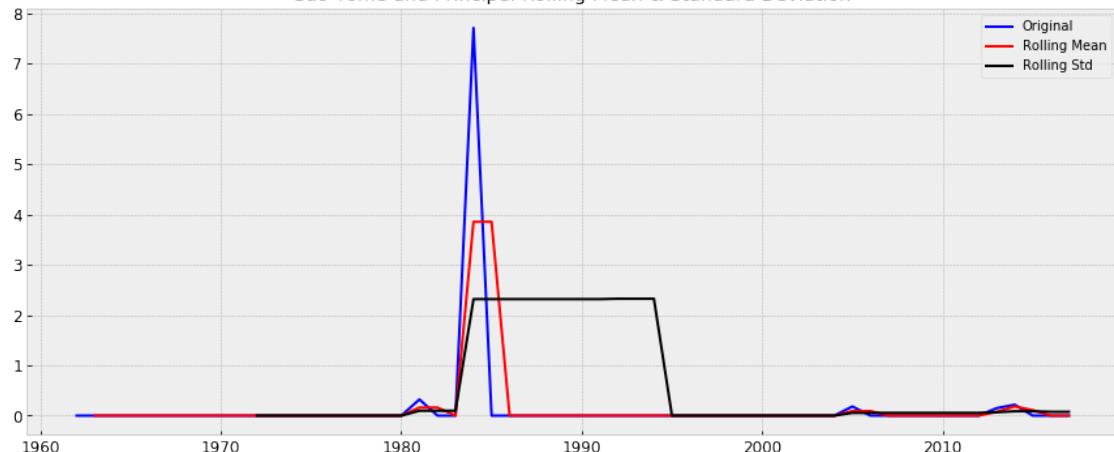
Saint Vincent and the Grenadines: Rolling Mean & Standard Deviation



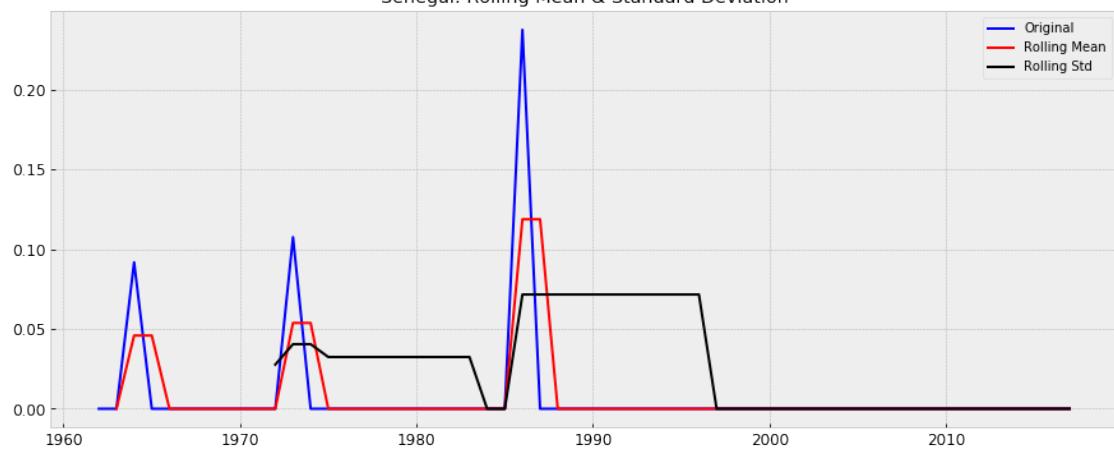
Samoa: Rolling Mean & Standard Deviation

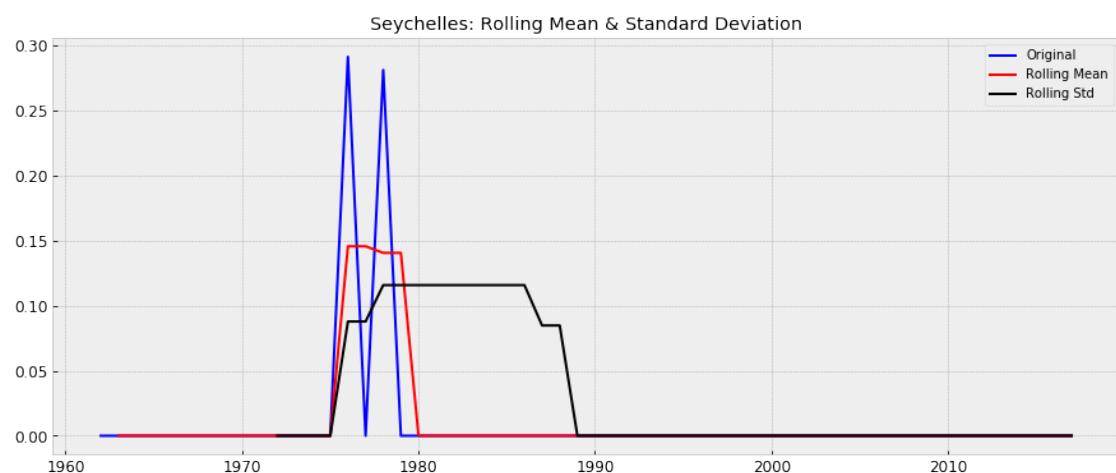
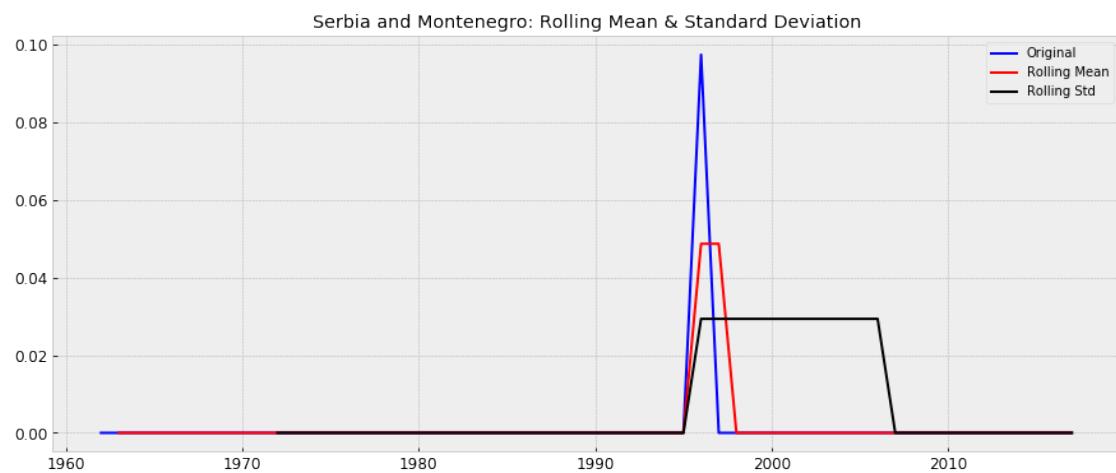
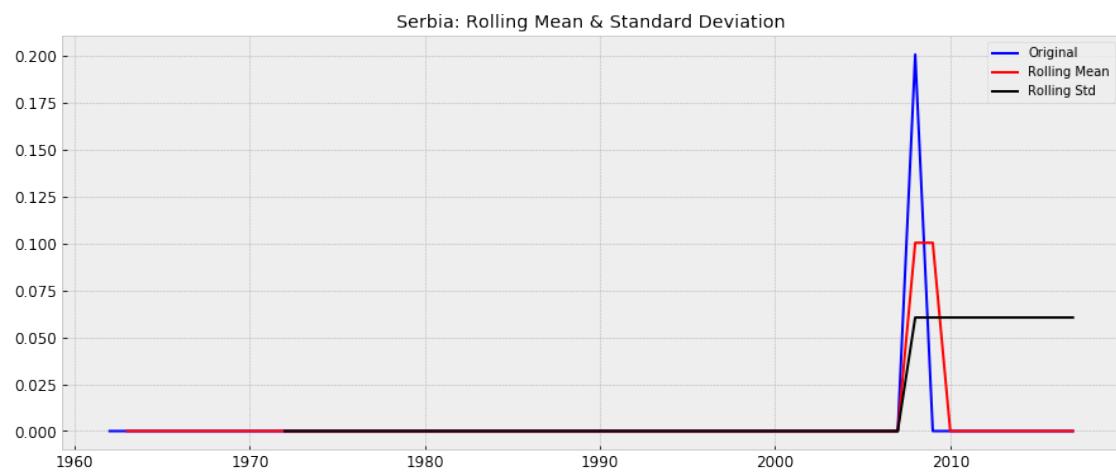


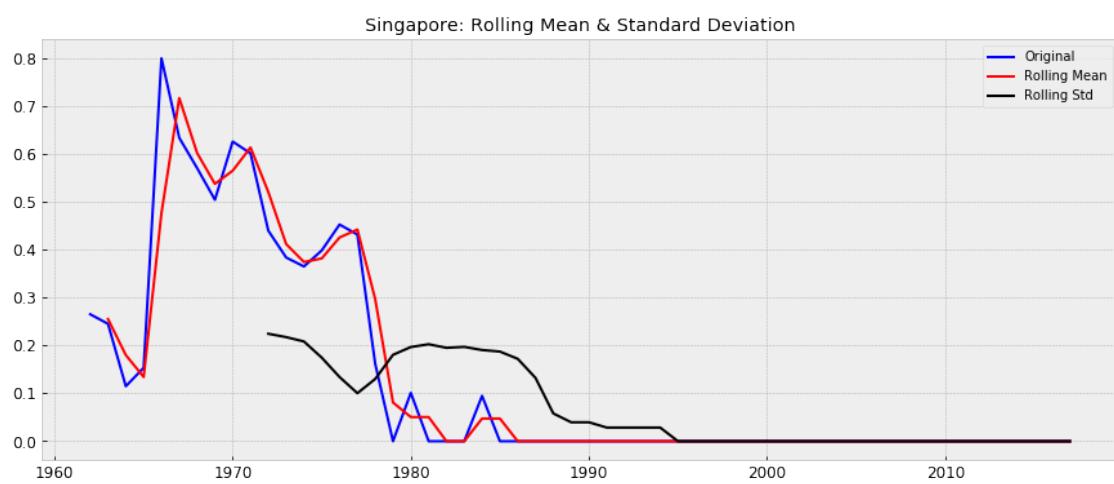
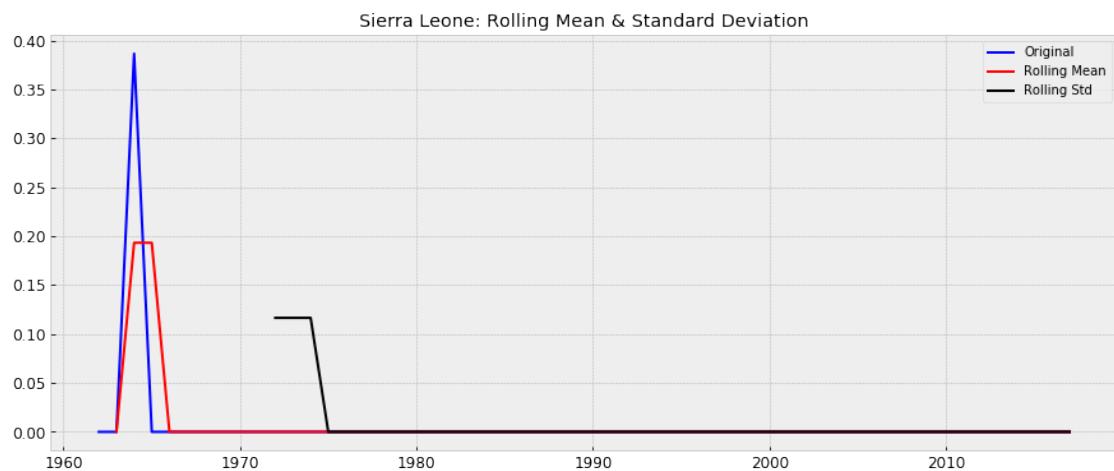
Sao Tome and Principe: Rolling Mean & Standard Deviation



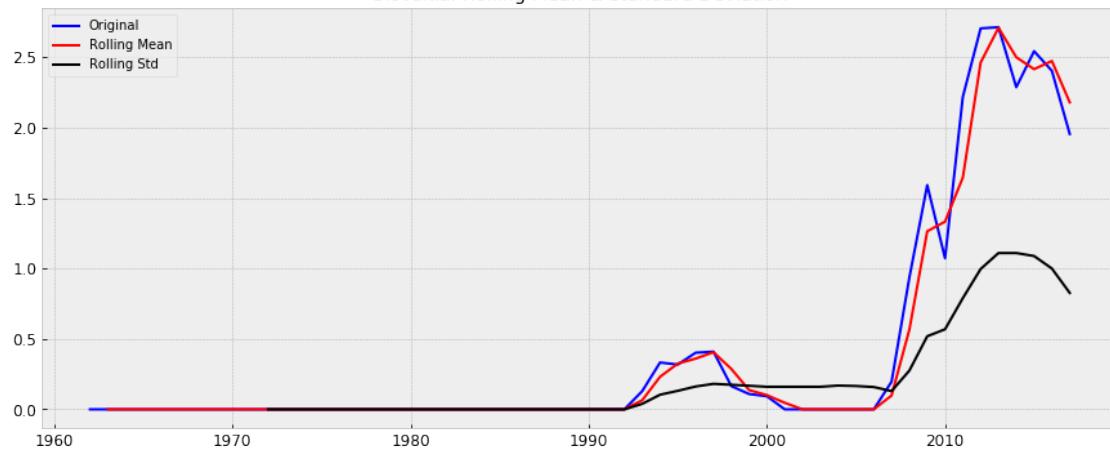
Senegal: Rolling Mean & Standard Deviation



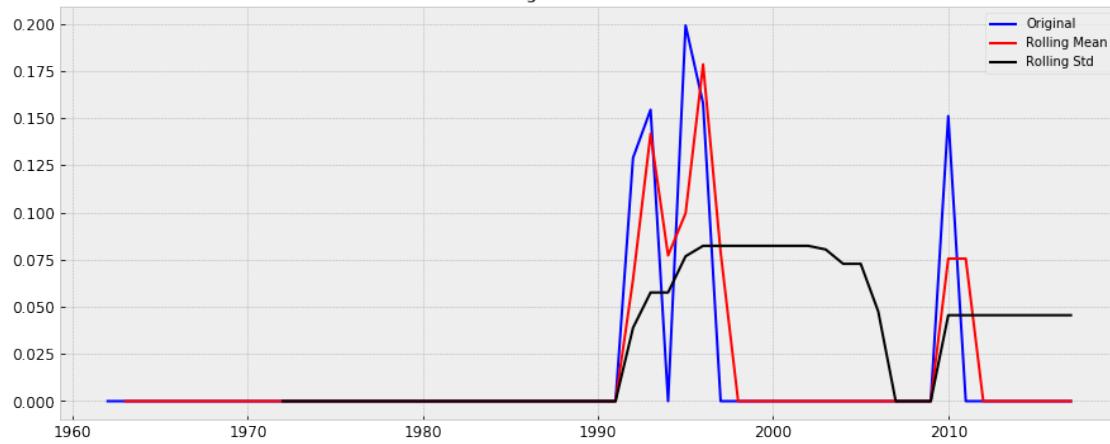




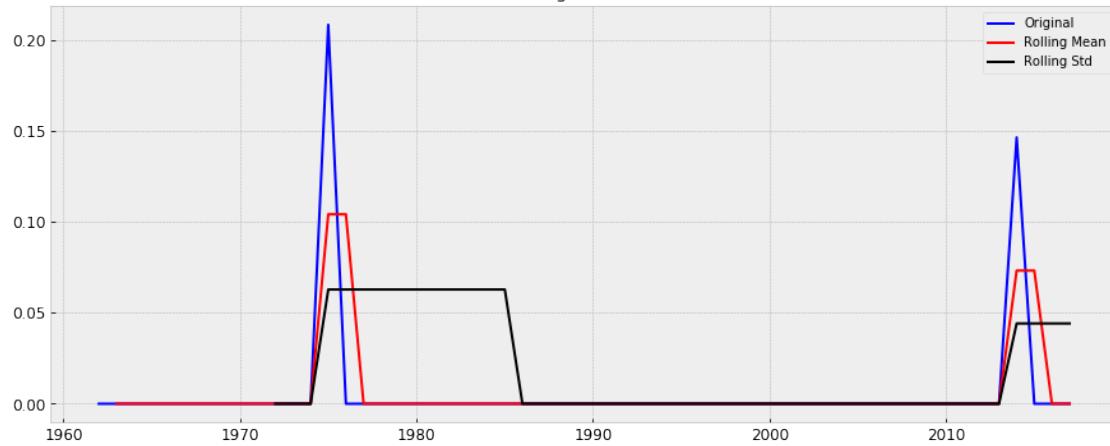
Slovakia: Rolling Mean & Standard Deviation

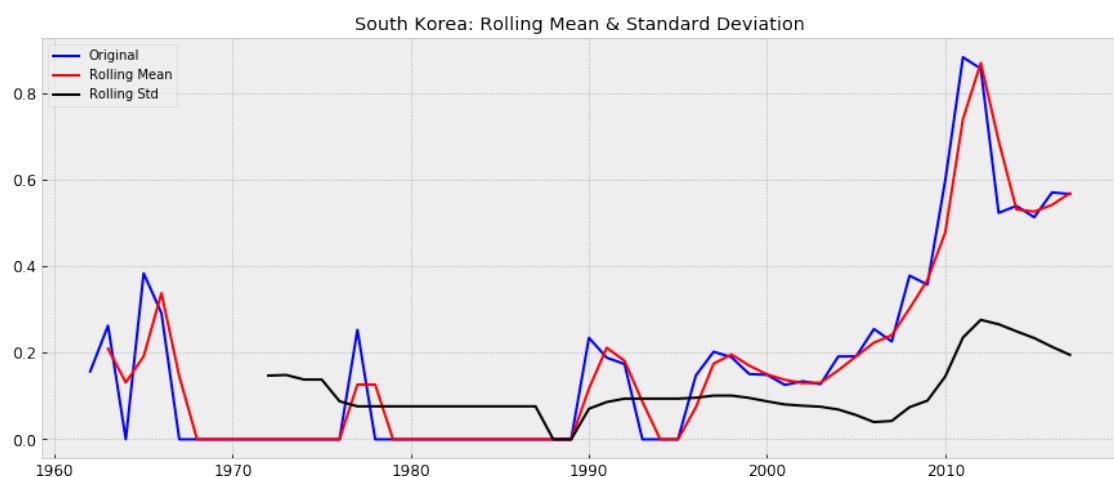
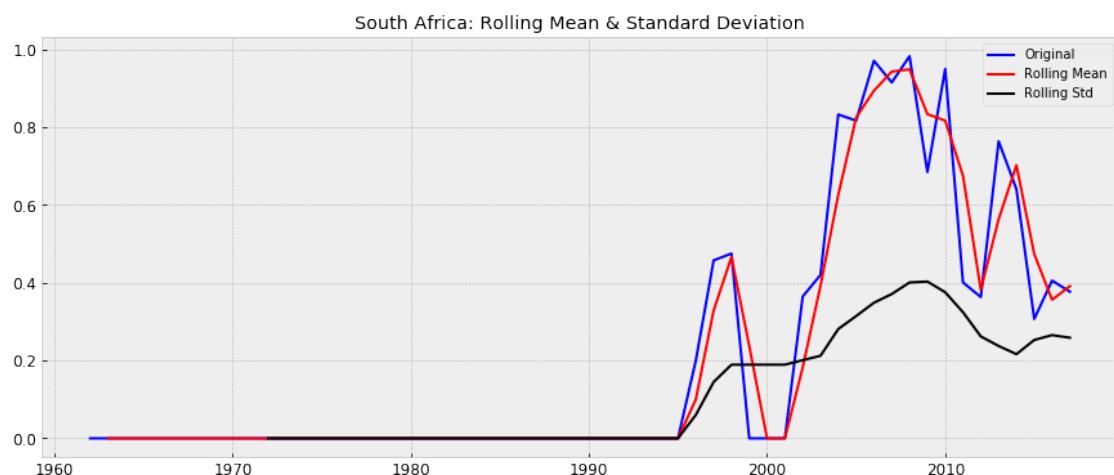
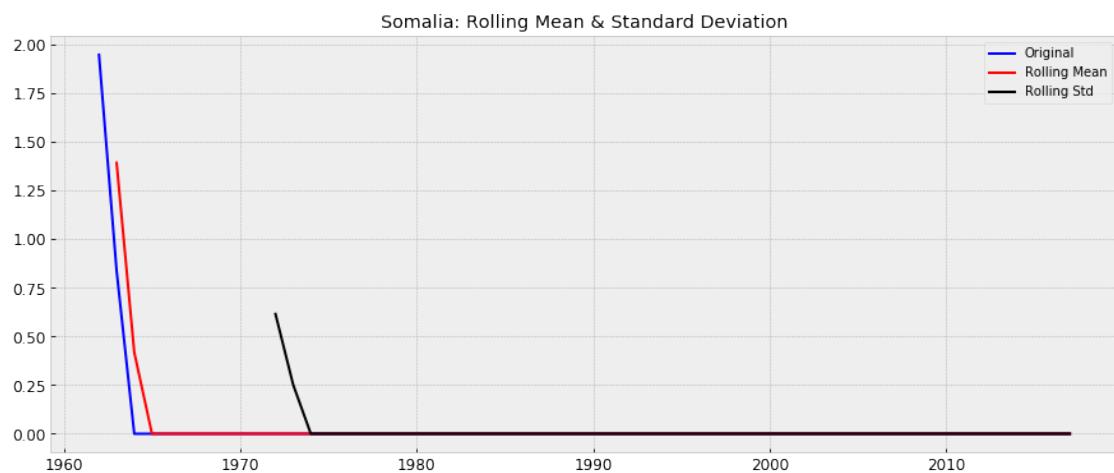


Slovenia: Rolling Mean & Standard Deviation

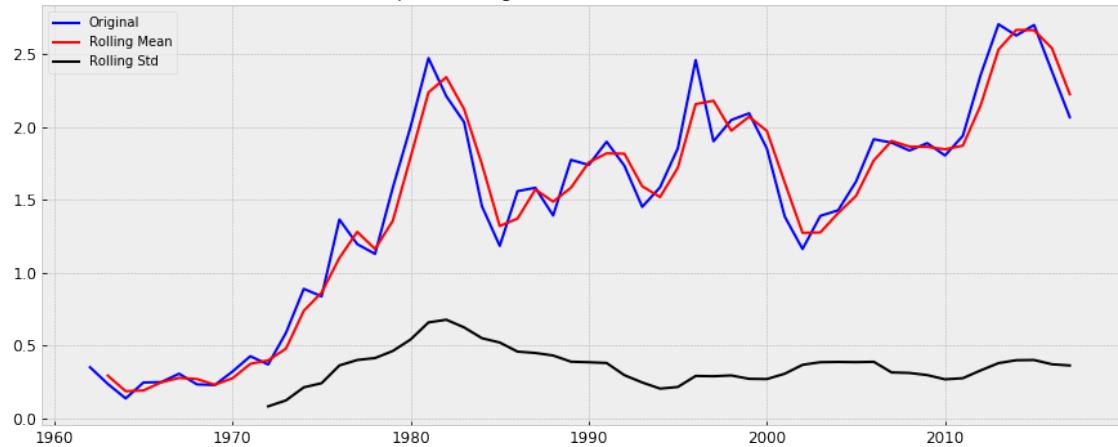


Solomon Islands: Rolling Mean & Standard Deviation

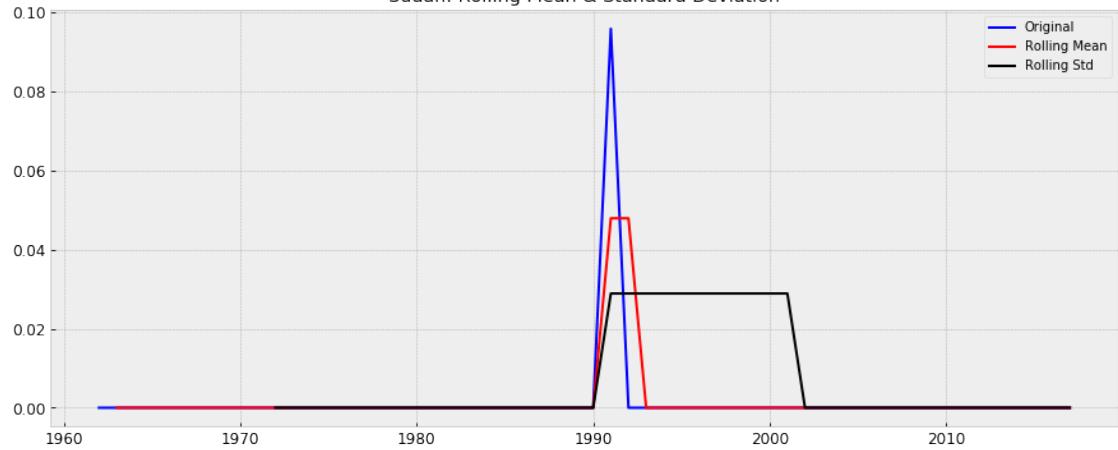




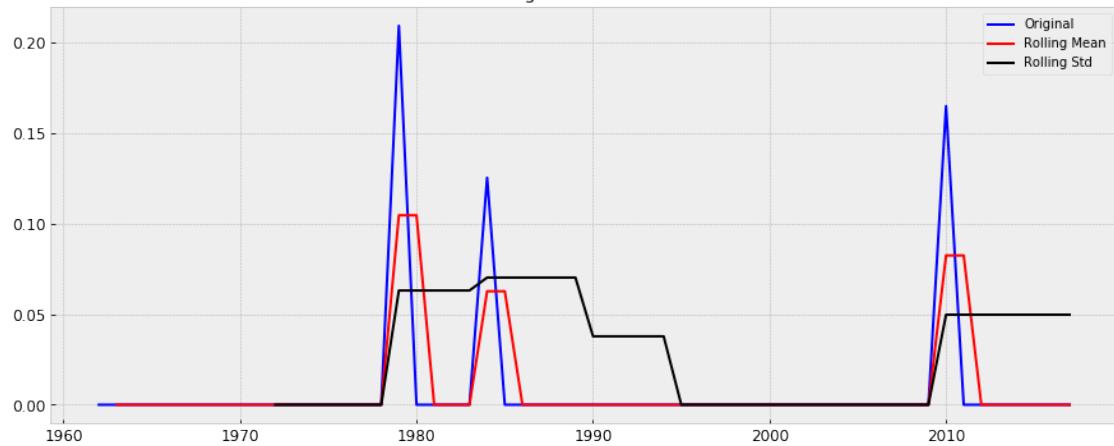
Spain: Rolling Mean & Standard Deviation



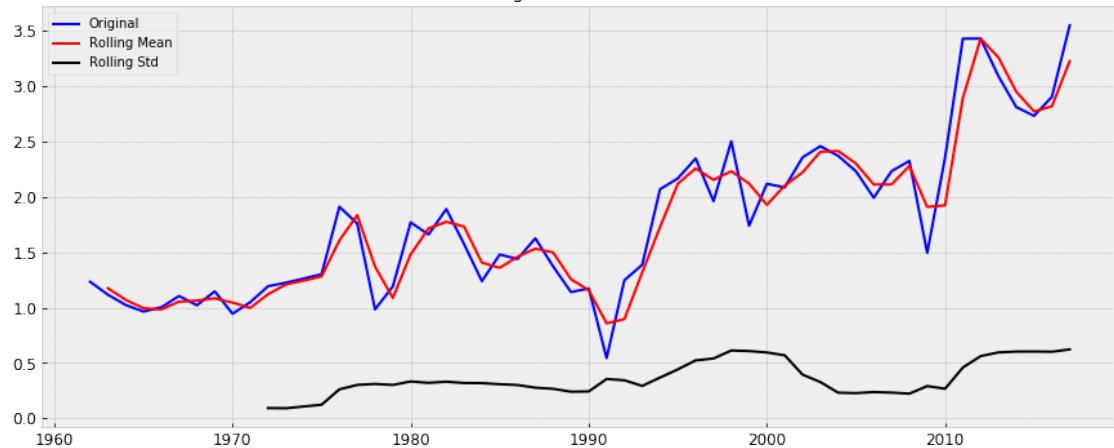
Sudan: Rolling Mean & Standard Deviation



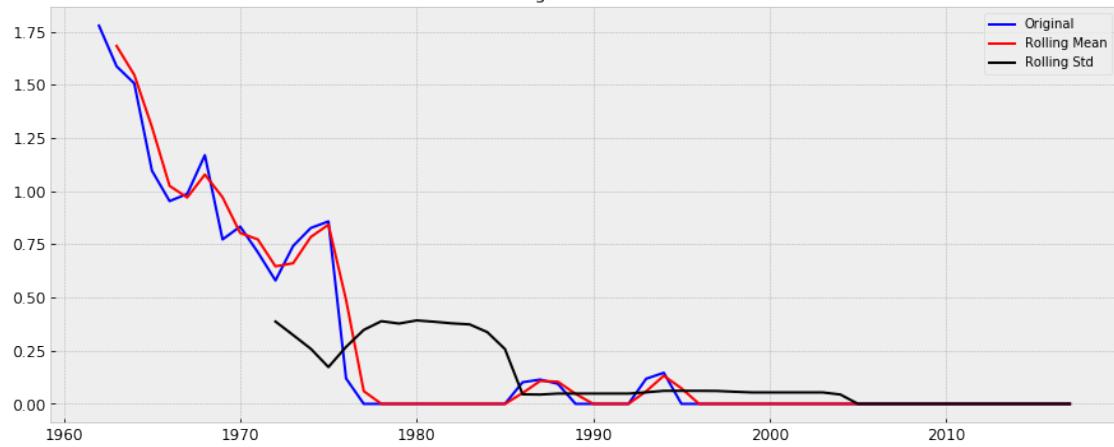
Suriname: Rolling Mean & Standard Deviation

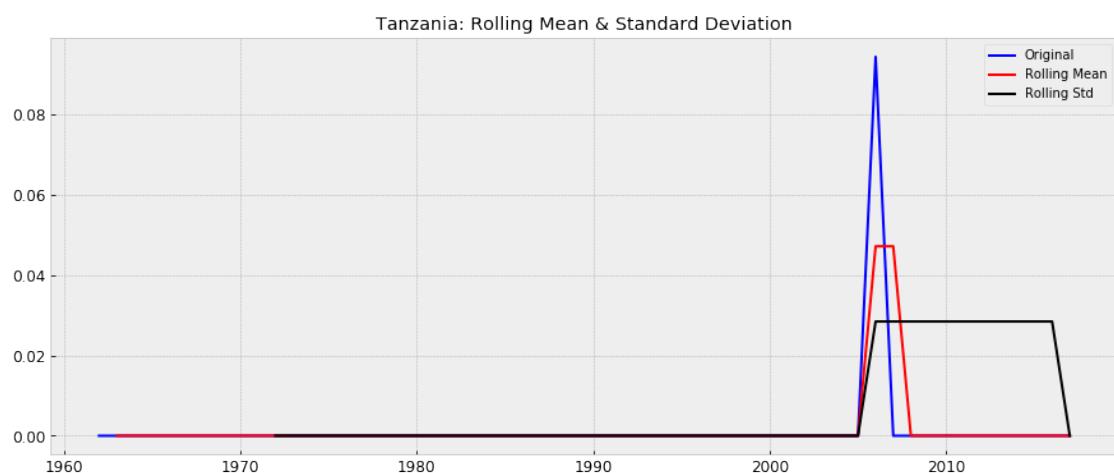
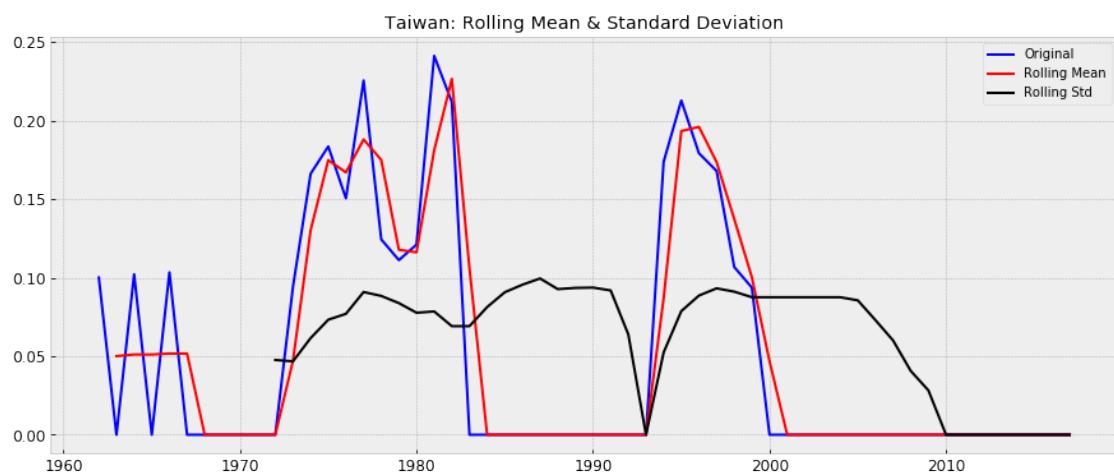
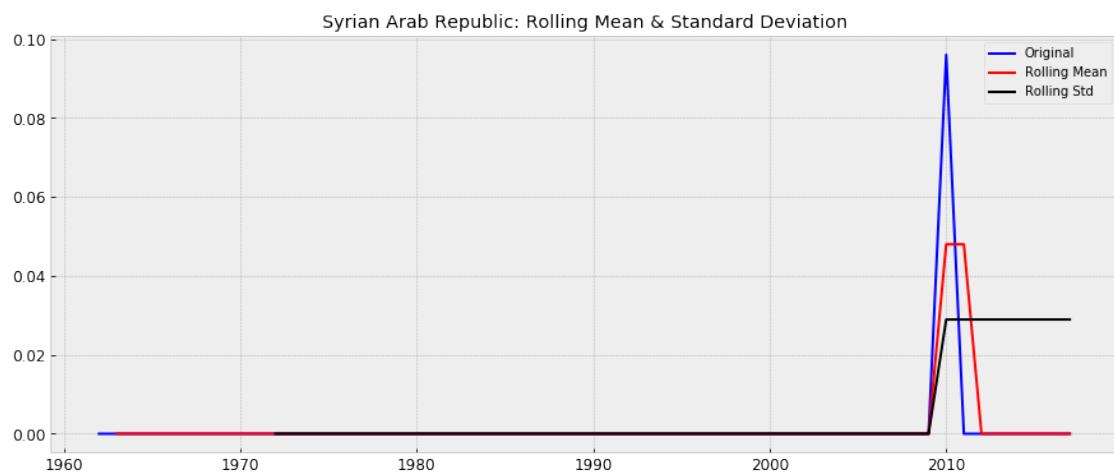


Sweden: Rolling Mean & Standard Deviation

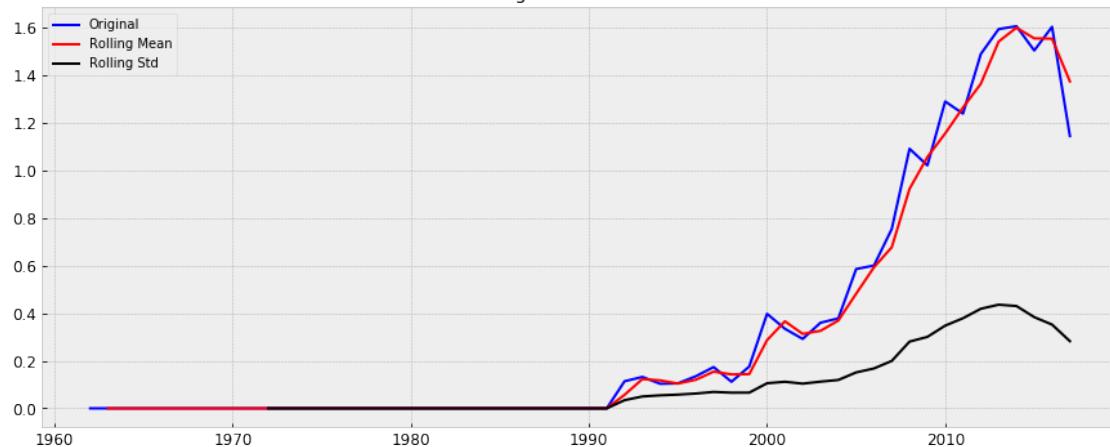


Switzerland: Rolling Mean & Standard Deviation

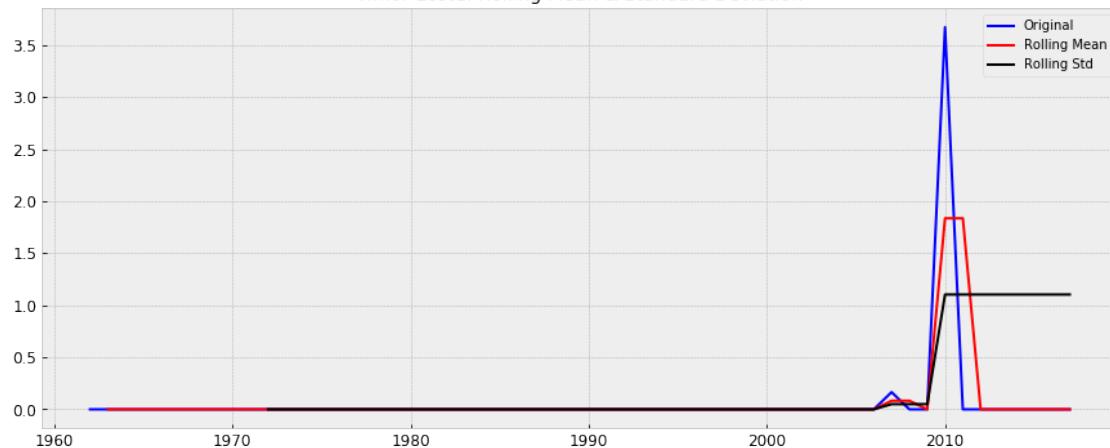




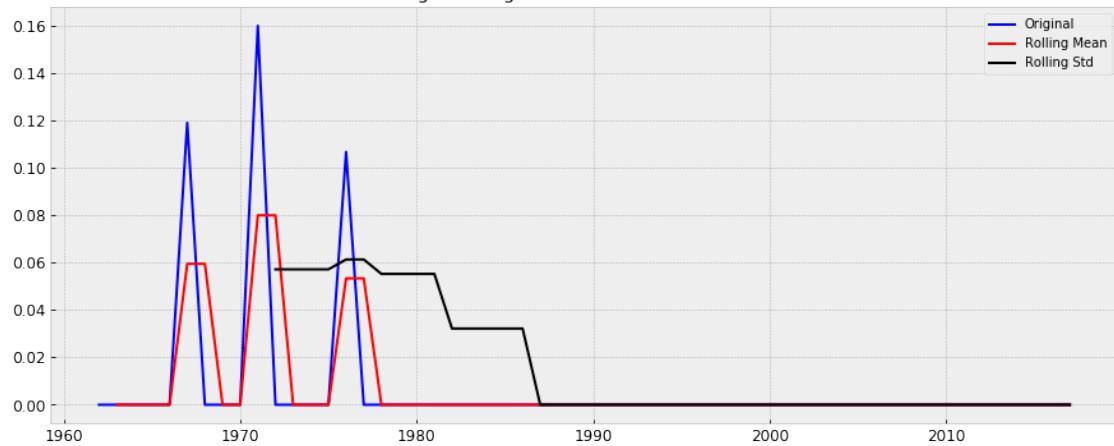
Thailand: Rolling Mean & Standard Deviation



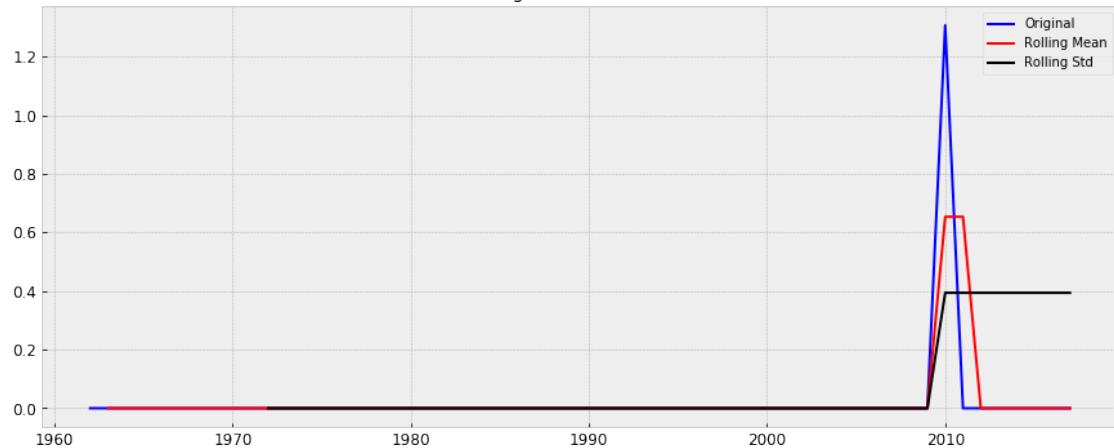
Timor-Leste: Rolling Mean & Standard Deviation



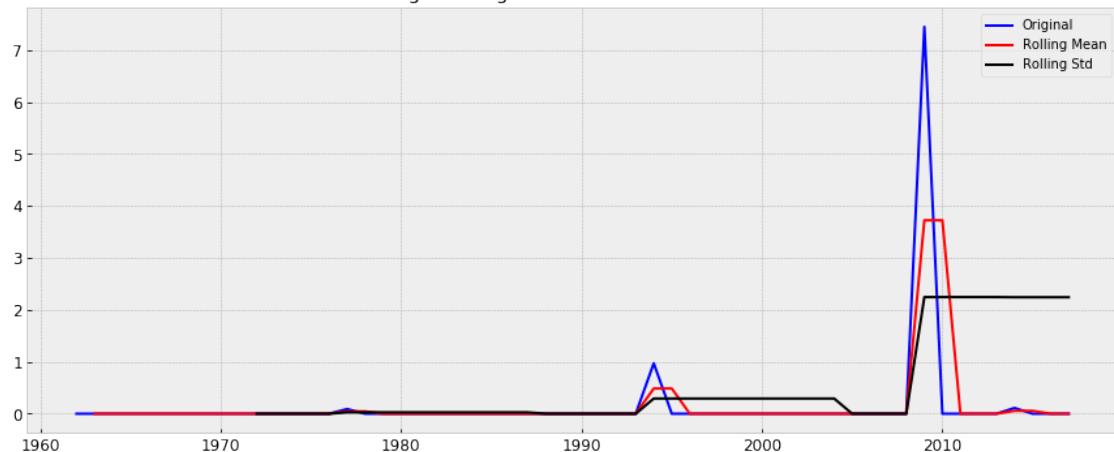
Togo: Rolling Mean & Standard Deviation



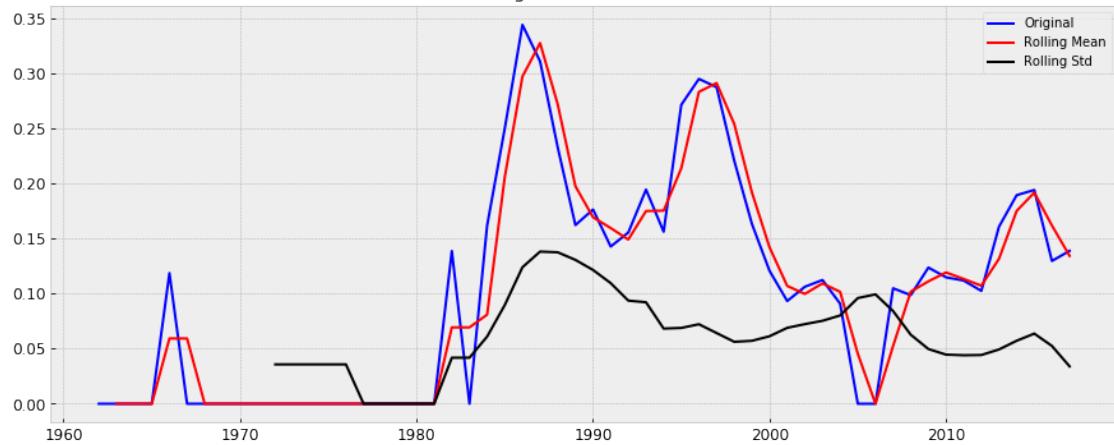
Tokelau: Rolling Mean & Standard Deviation



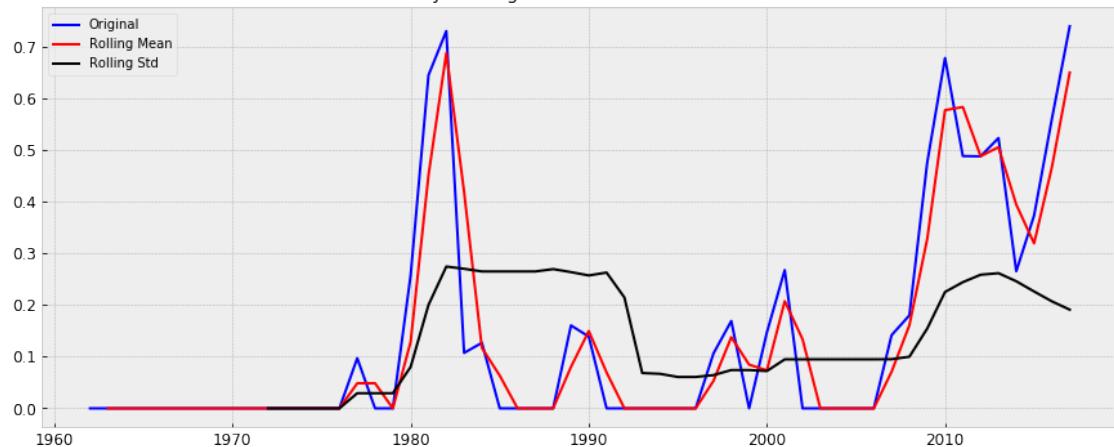
Tonga: Rolling Mean & Standard Deviation



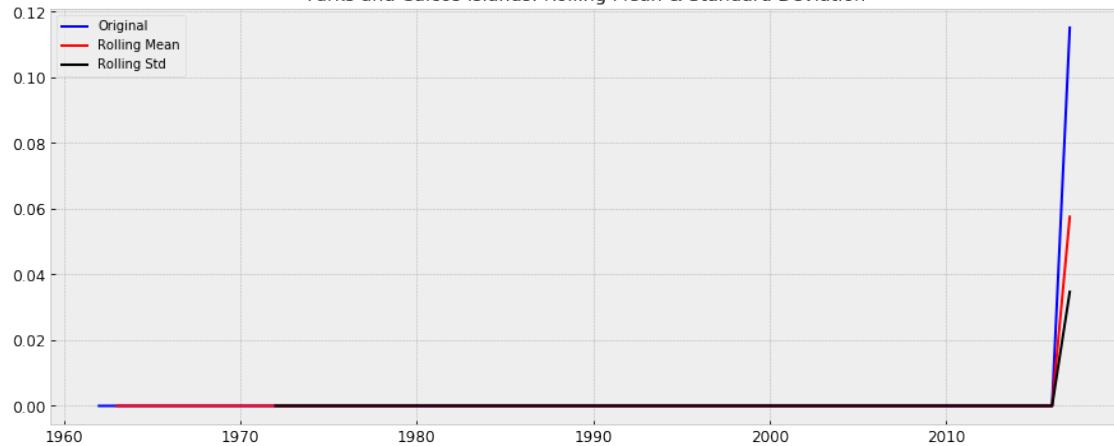
Tunisia: Rolling Mean & Standard Deviation



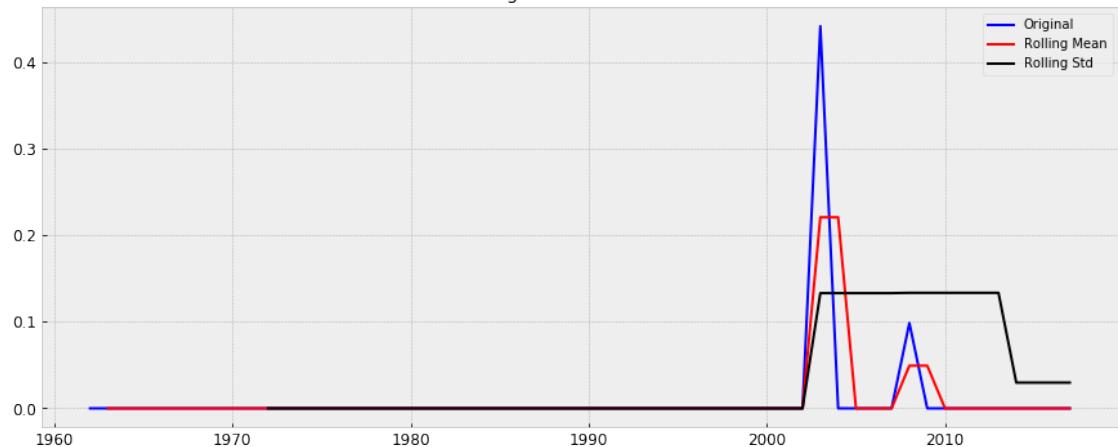
Turkey: Rolling Mean & Standard Deviation



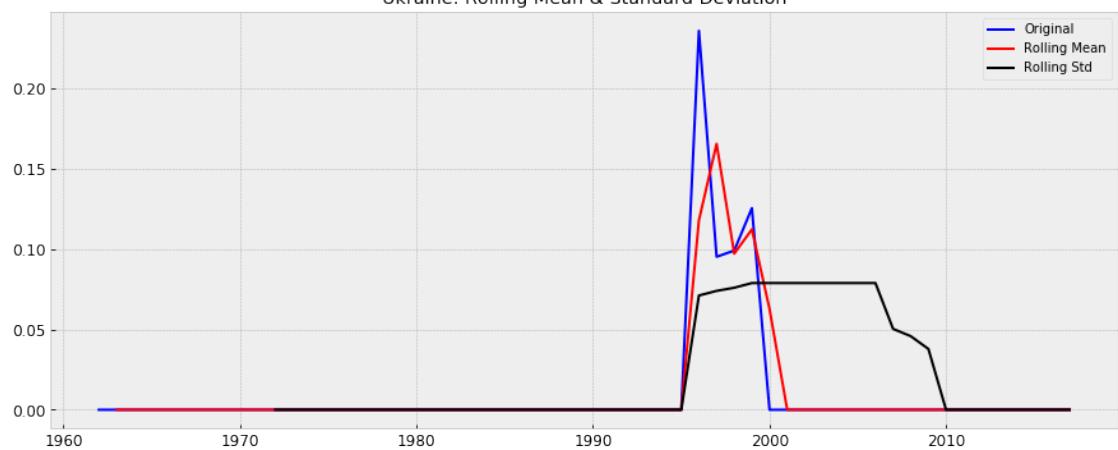
Turks and Caicos Islands: Rolling Mean & Standard Deviation



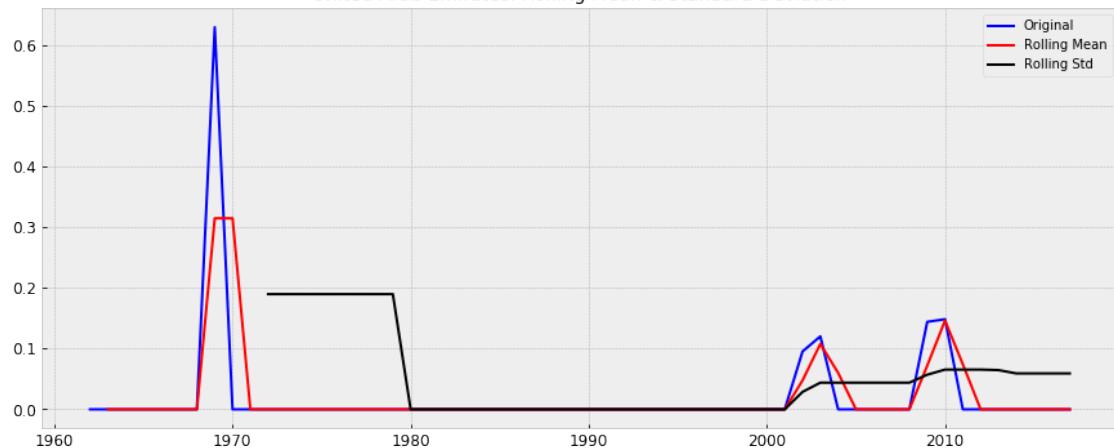
Tuvalu: Rolling Mean & Standard Deviation



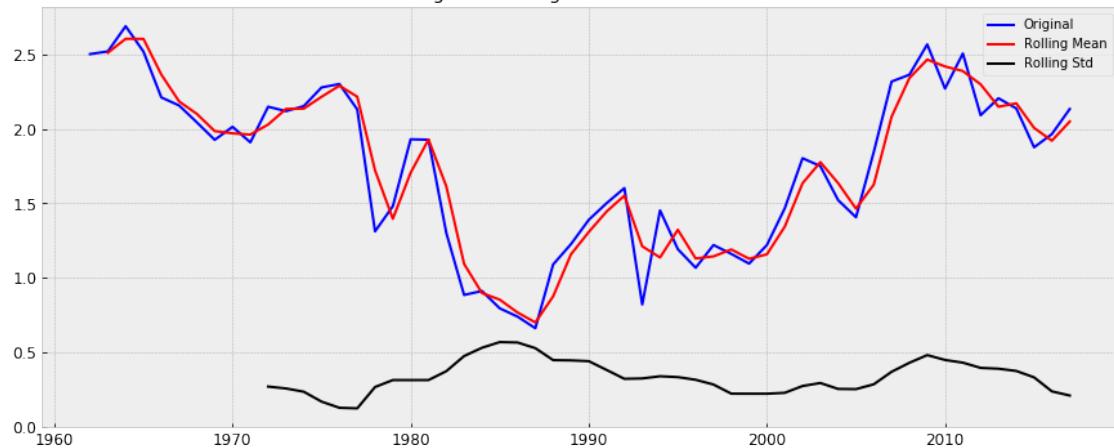
Ukraine: Rolling Mean & Standard Deviation



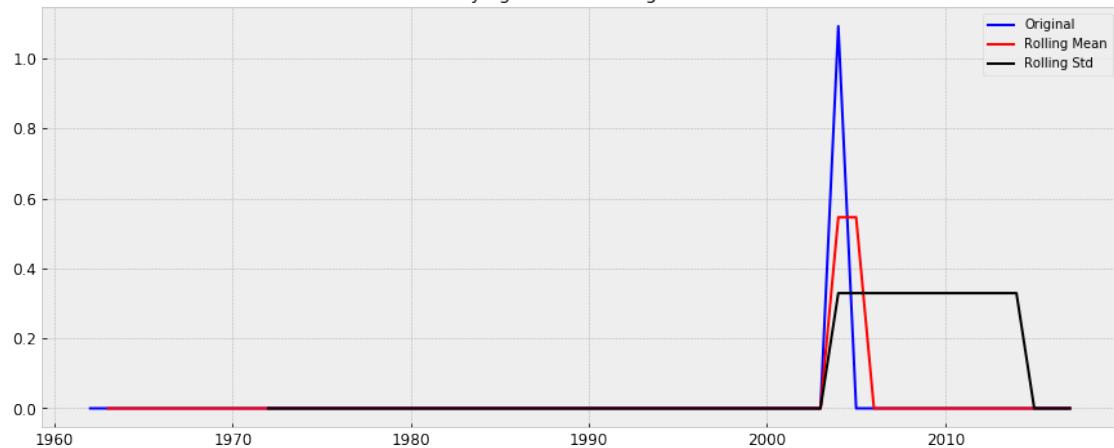
United Arab Emirates: Rolling Mean & Standard Deviation

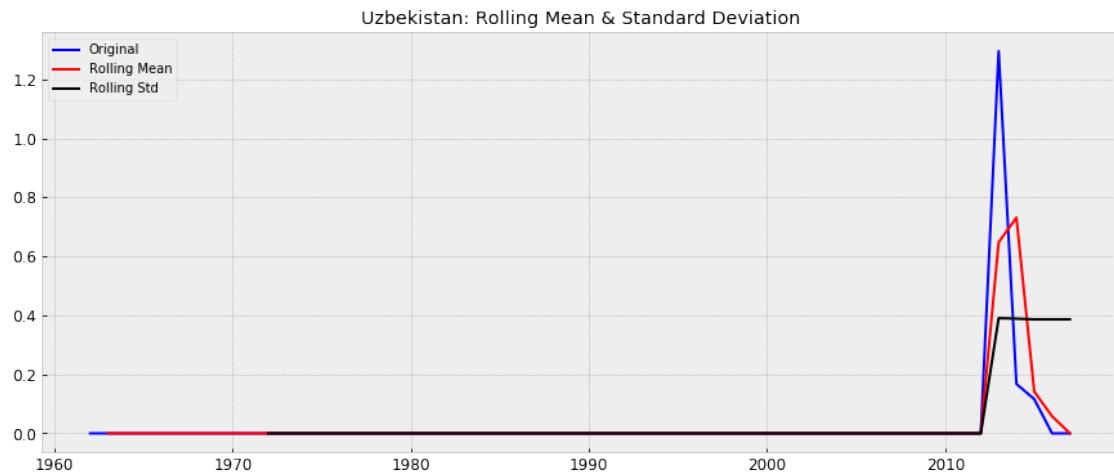
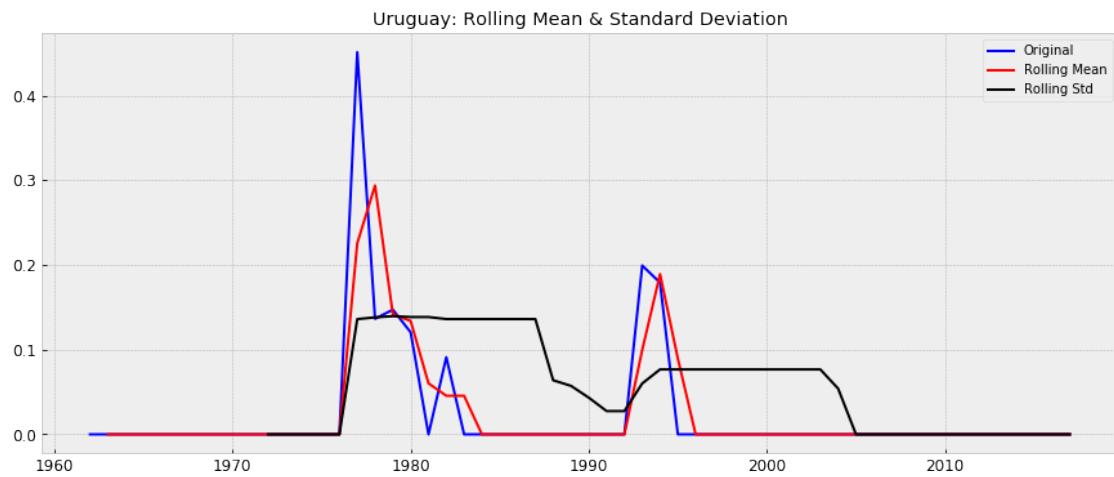
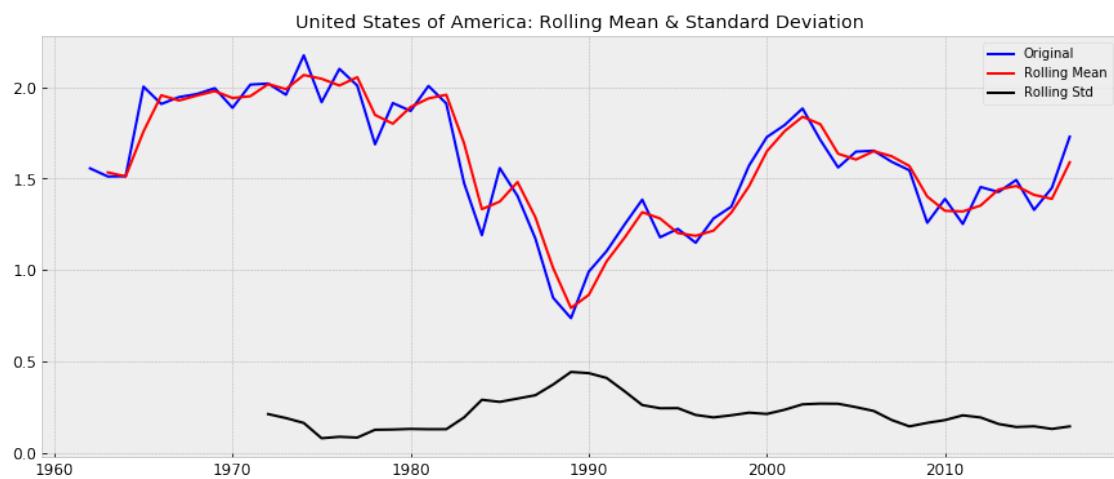


United Kingdom: Rolling Mean & Standard Deviation

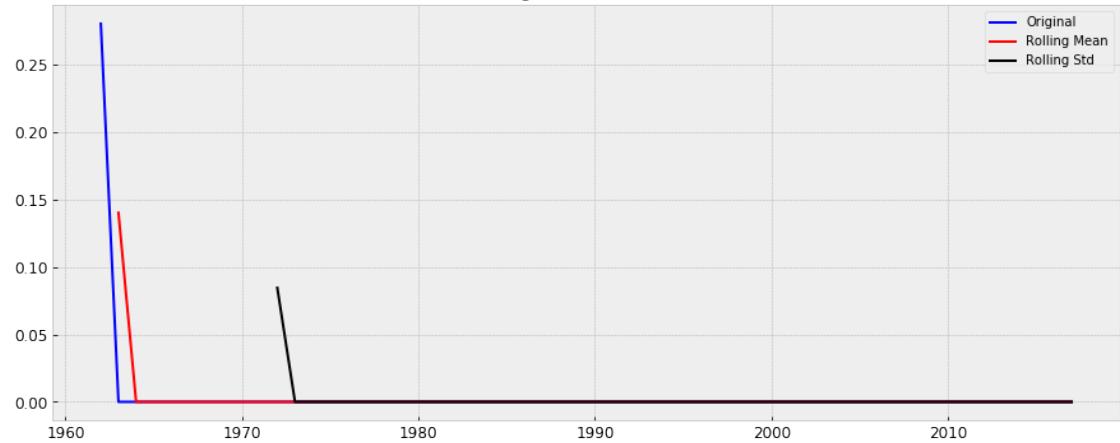


United States Minor Outlying Islands: Rolling Mean & Standard Deviation

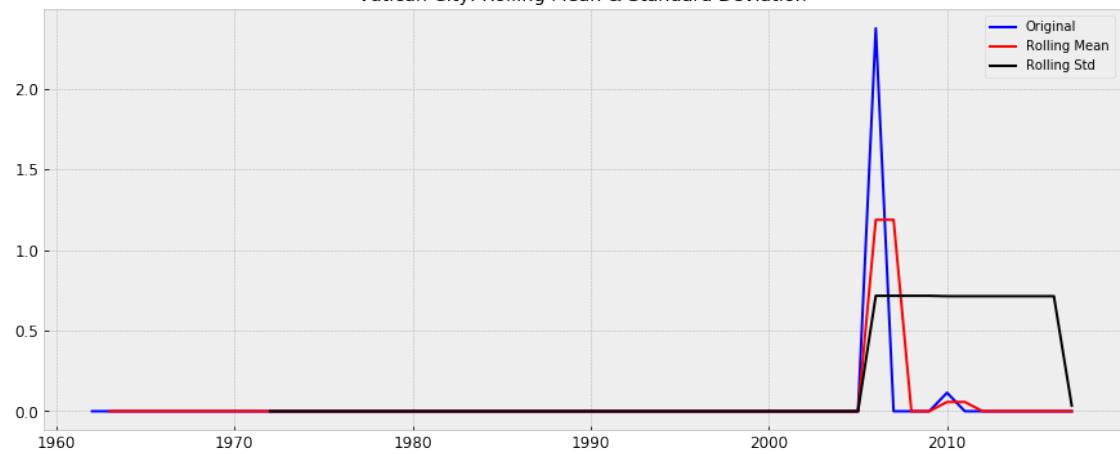




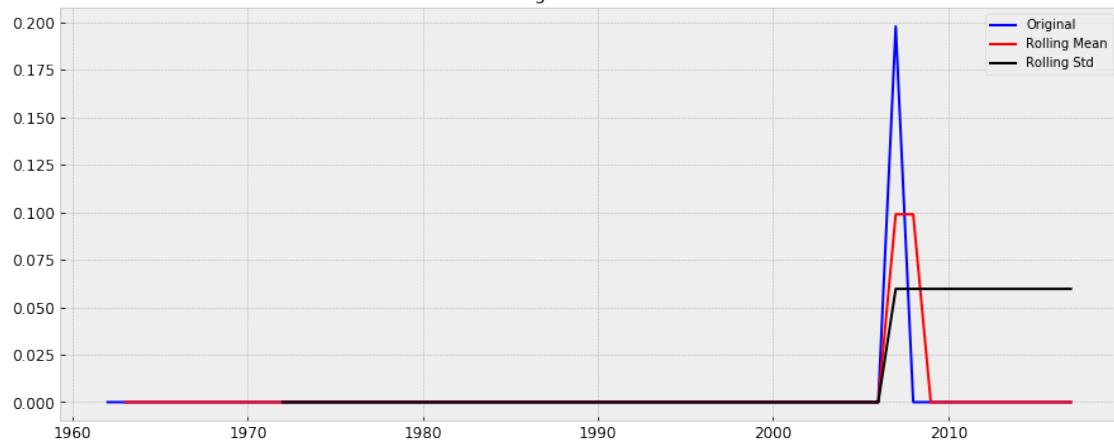
Vanuatu: Rolling Mean & Standard Deviation



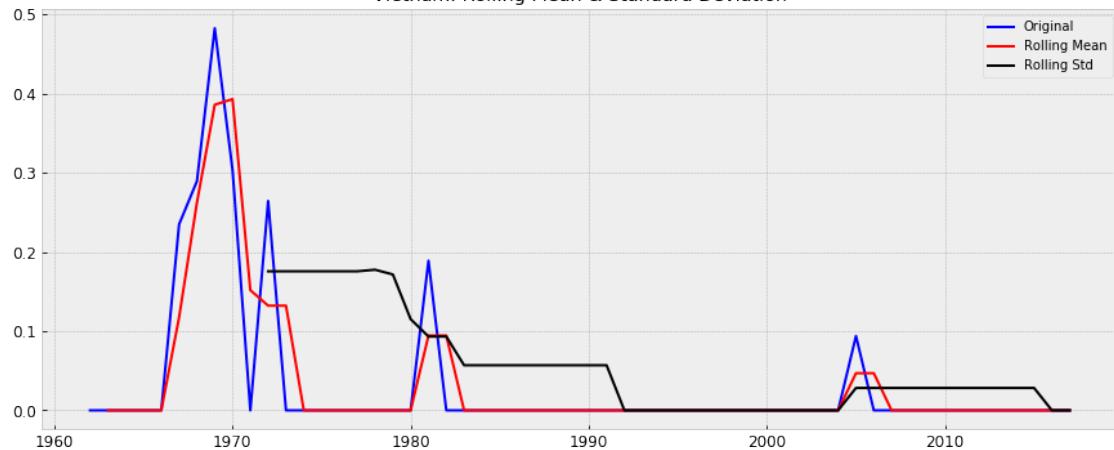
Vatican City: Rolling Mean & Standard Deviation



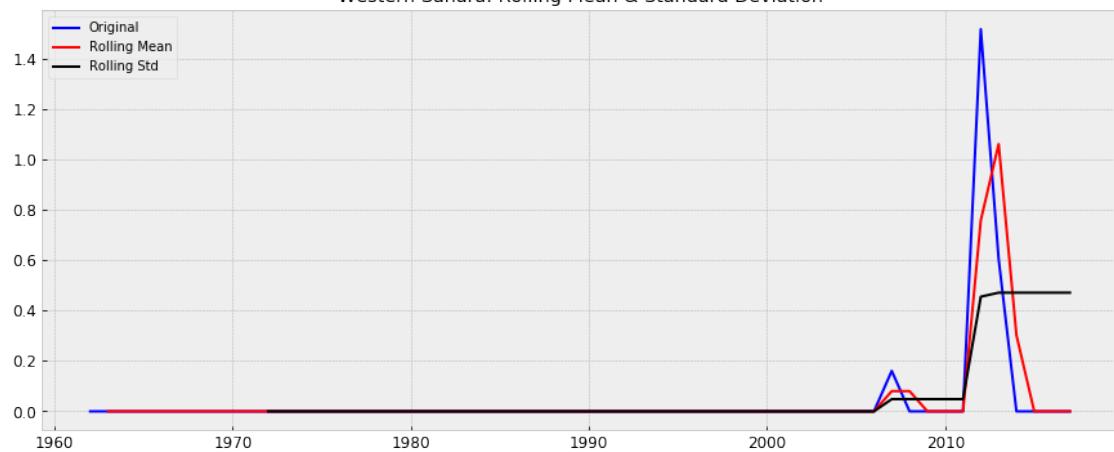
Venezuela: Rolling Mean & Standard Deviation

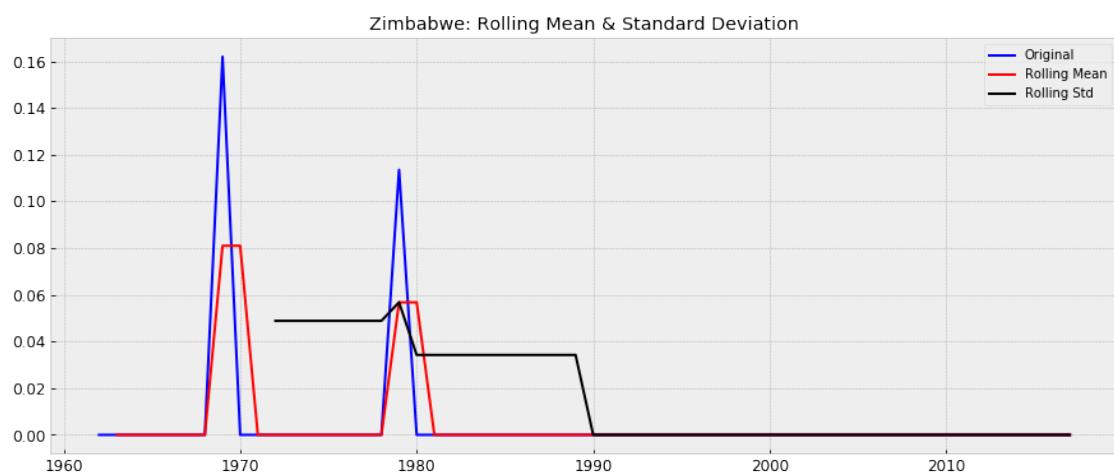
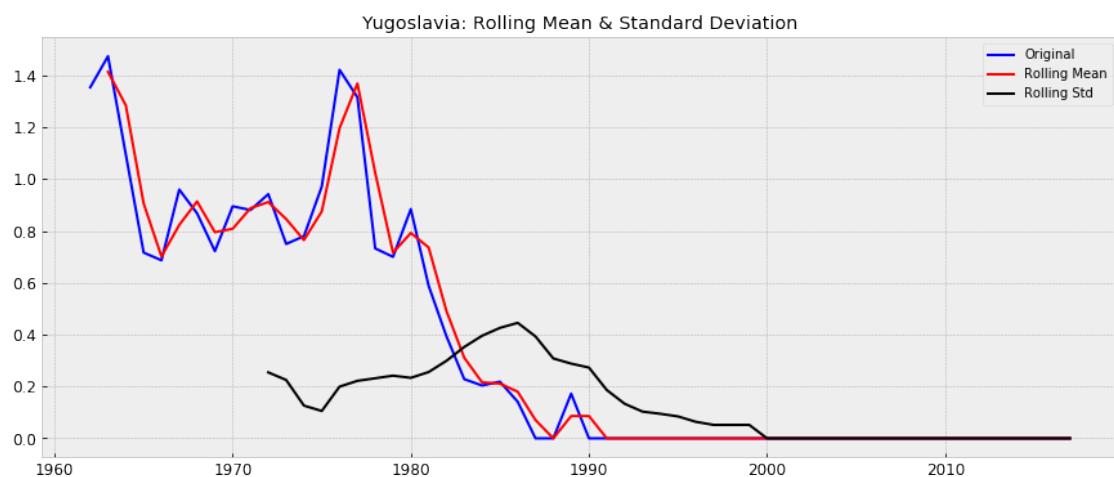
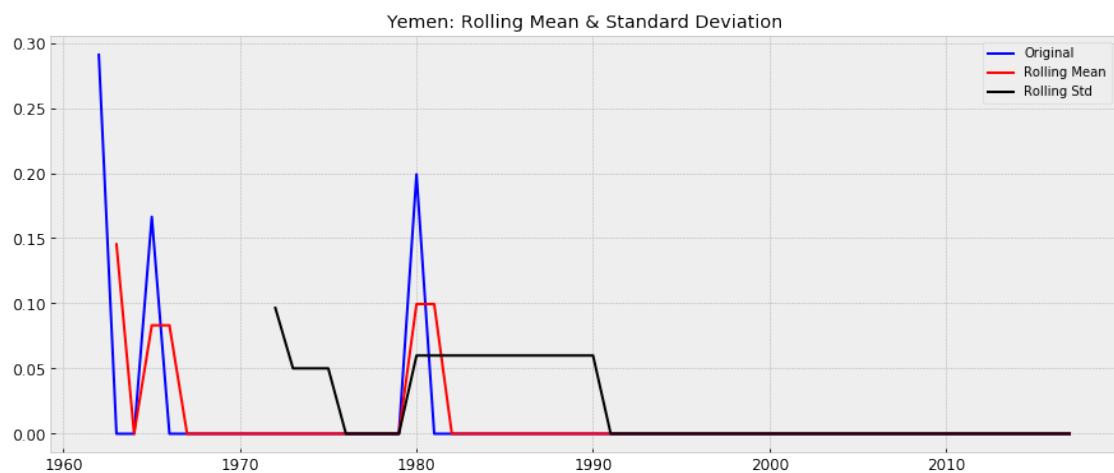


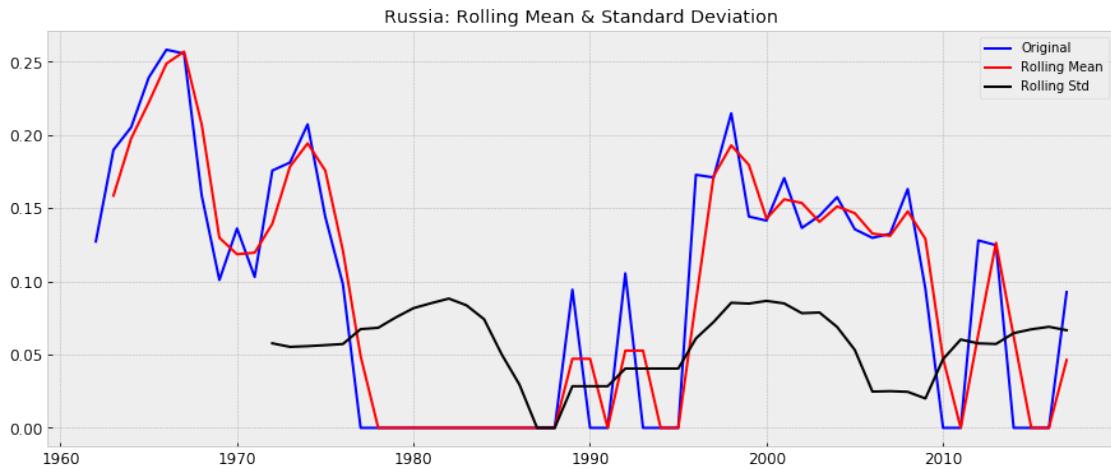
Vietnam: Rolling Mean & Standard Deviation



Western Sahara: Rolling Mean & Standard Deviation







4.1.2 Augmented Dickey-Fuller (ADF) Test

- The intuition behind the test is that if the series is integrated then the lagged level of the series $y(t-1)$ will provide no relevant information in predicting the change in $y(t)$.

Null hypothesis (H₀): The time series is not stationary, it presents heterocedasticity. In other words, the time series depends on itself (i.e.: y_t depends on y_{t-1} , y_{t-1} depends on y_{t-2} - Rejecting the null hypothesis (i.e. a very low p-value) will indicate stationarity. Put simply, the t-test result is less than all critical values (1%, 5%, 10%)

Alternative hypothesis(H_a): the series is stationary

Reference ADF results description

```
[150]: from statsmodels.tsa.stattools import adfuller, kpss

[151]: def ADF_Stationarity_Test(ts_groups):
    #countries = list(ts_groups.columns)
    adf_results = {}
    col_names=['Test Statistic','p_value','Lags used','Observations Used','1% Critical value','5% Critical value','10% Critical value','IC_Best']
    data = pd.DataFrame(columns=col_names,index=countries)

    for country in countries:
        #Dickey-Fuller test
        adf_results[country] = tsa.adfuller(ts_groups[country],autolag='AIC')

    for country in adf_results:
        data.loc[country] = pd.Series({'Test Statistic':adf_results[country][0],
```

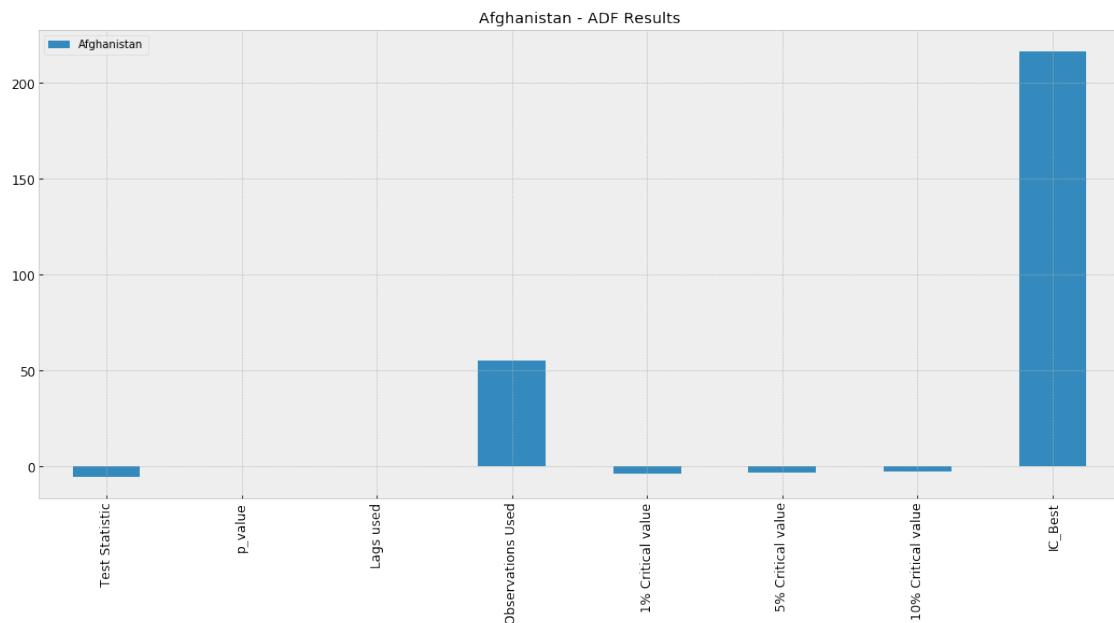
```

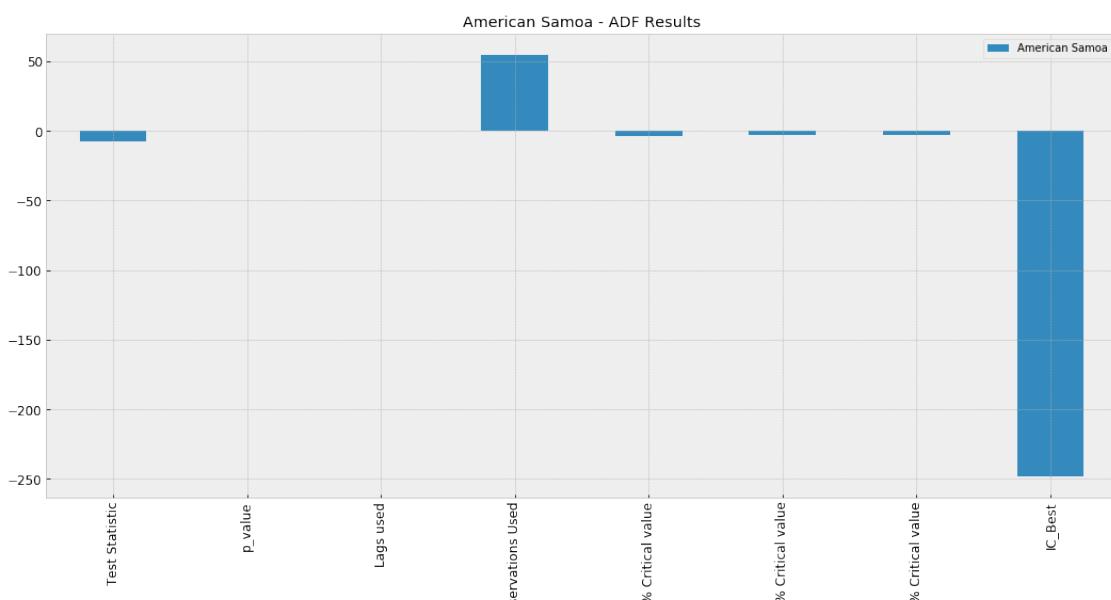
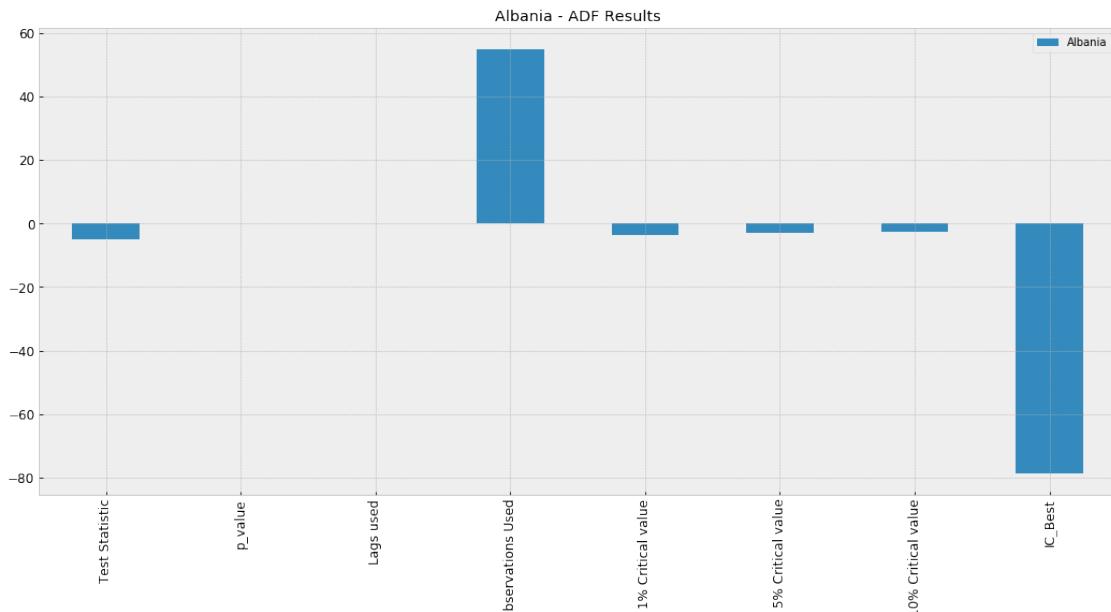
        'p_value':adf_results[country][1],
        'Lags used':adf_results[country][2],
        'Observations Used':
    ↪adf_results[country][3],
        '1% Critical value': ↪
    ↪adf_results[country][4]['1%'],
        '5% Critical value': ↪
    ↪adf_results[country][4]['5%'],
        '10% Critical value': ↪
    ↪adf_results[country][4]['10%'],
        'IC_Best': adf_results[country][5]
    })
return data.transpose()

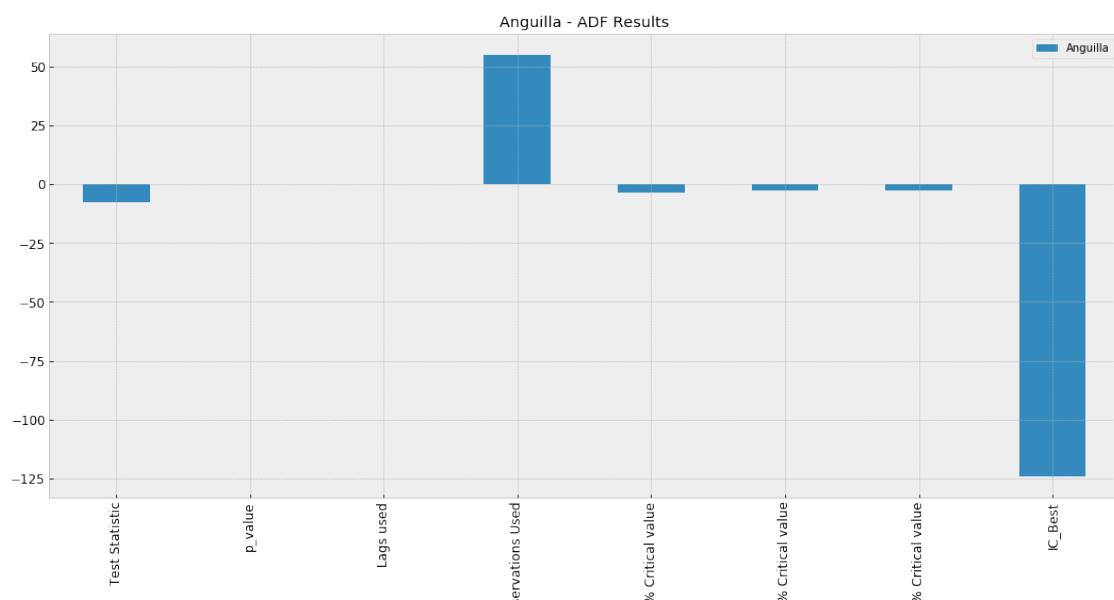
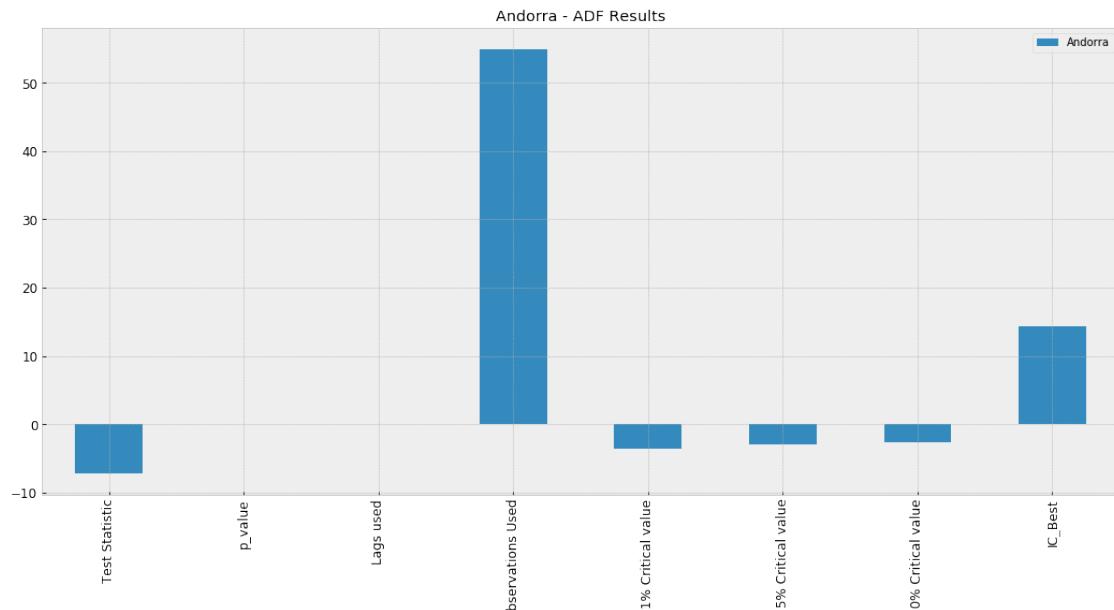
```

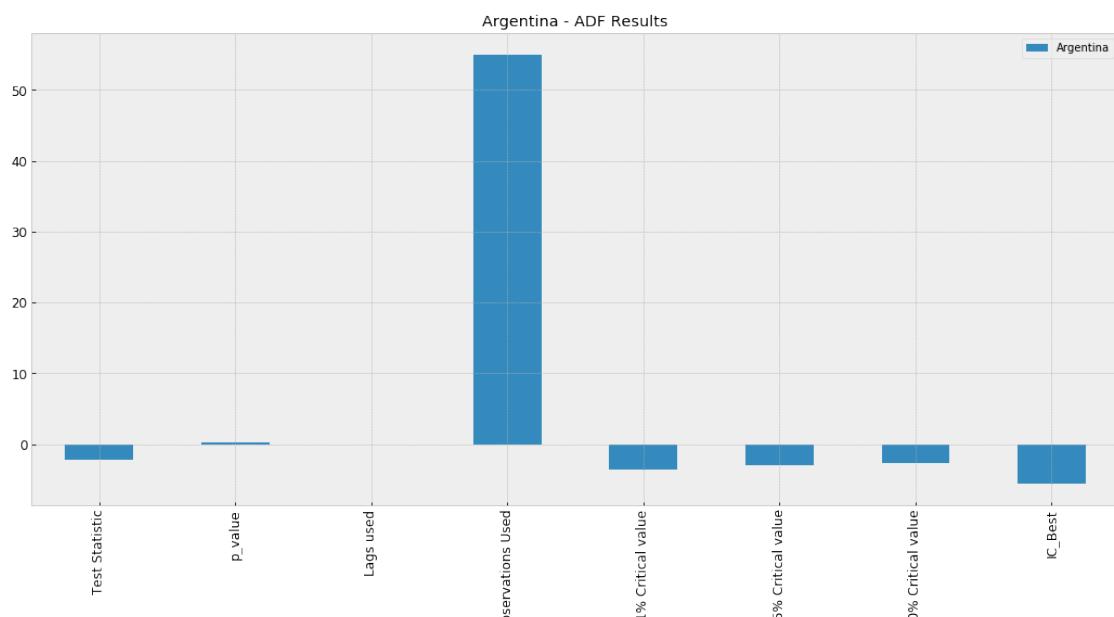
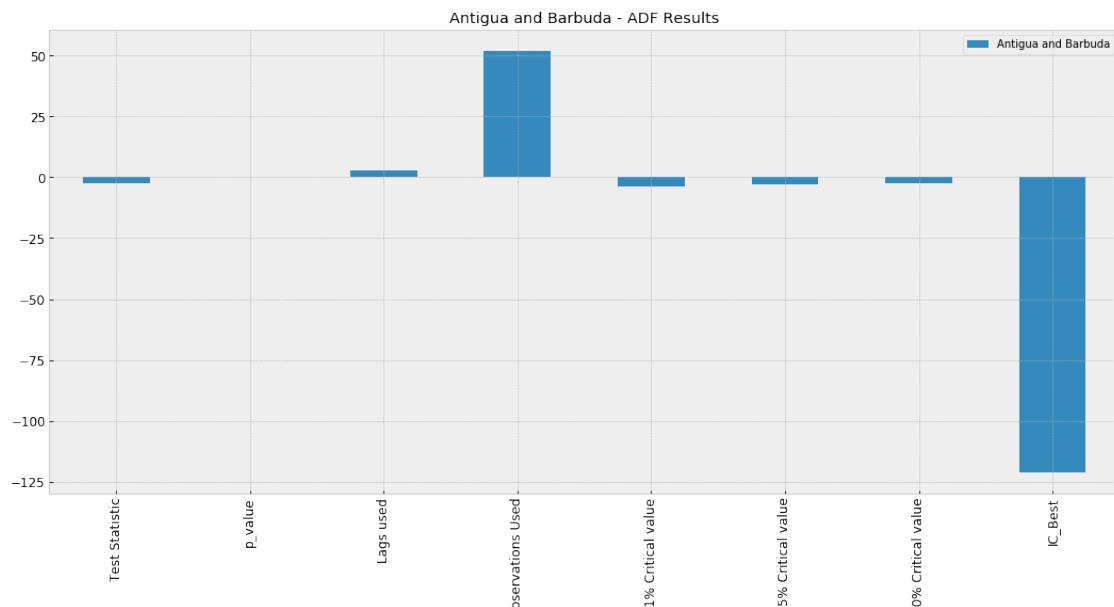
```
[152]: adf_test = ADF_Stationarity_Test(df_grps)
adf_countries = adf_test.columns
```

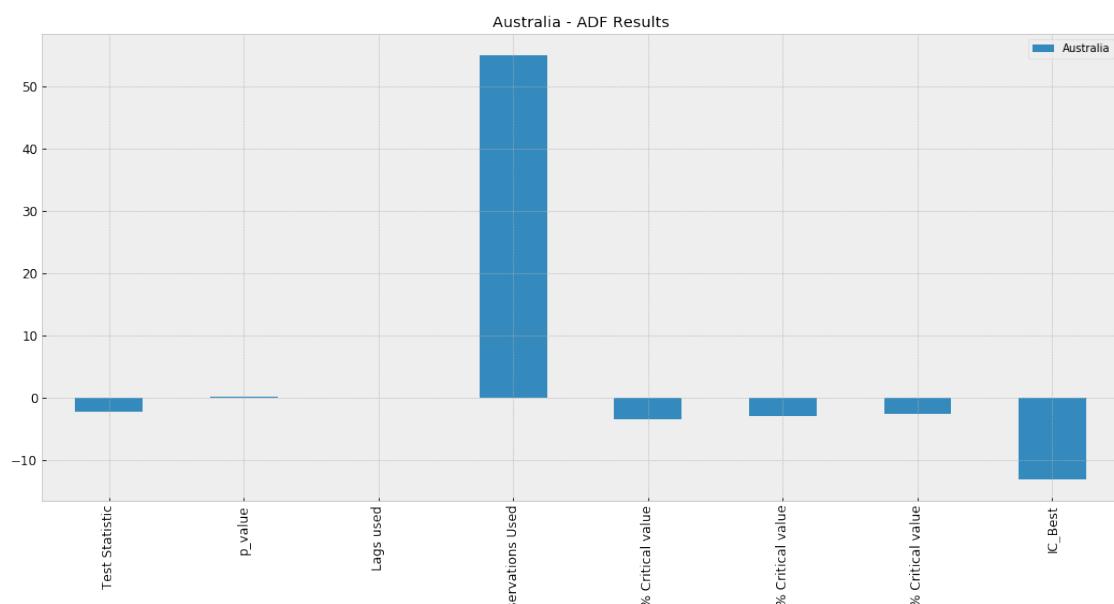
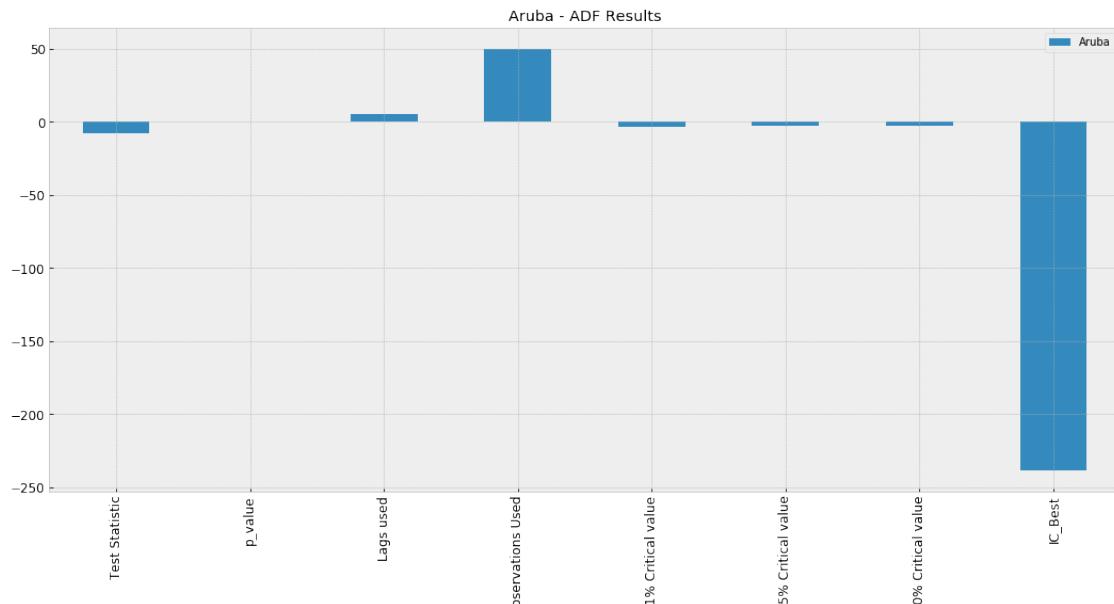
```
[154]: for country in adf_test:
    savefile = save_images(adf_test,'adf_test')
    adf_test[country].plot.bar()
    plt.legend(loc='best')
    plt.title('{} - ADF Results'.format(adf_test[country].name))
    plt.savefig(str('{}_ADF.png'.format(savefile)))
    plt.show()
```

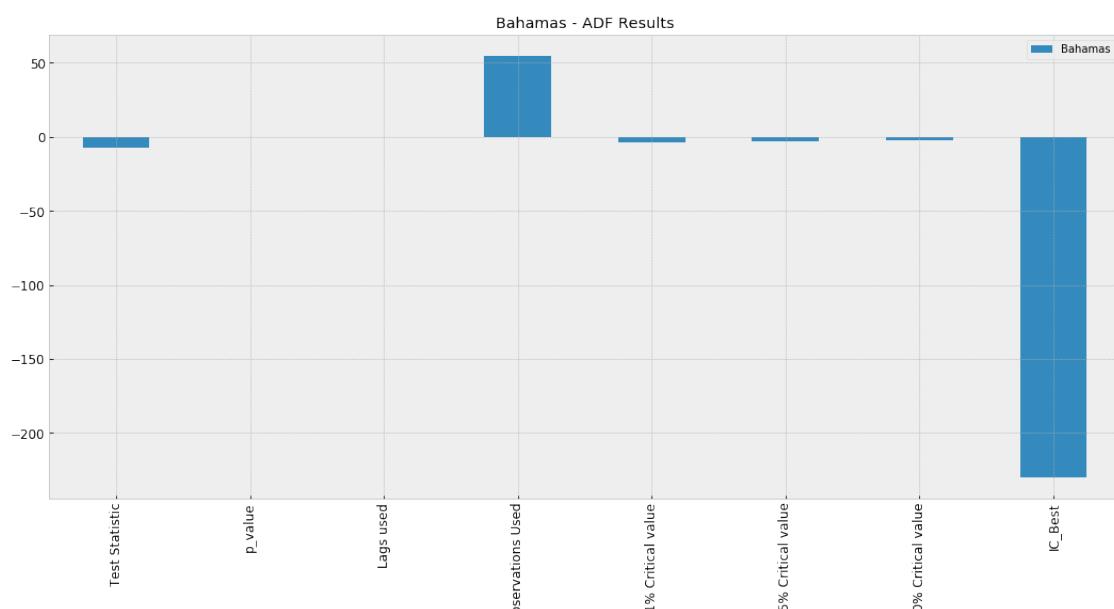
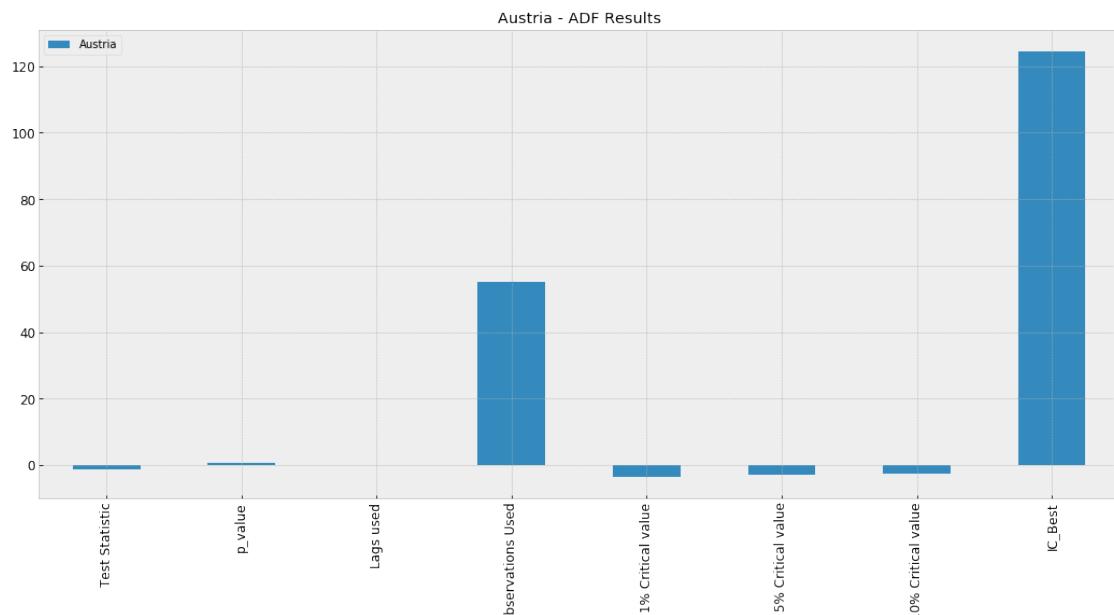


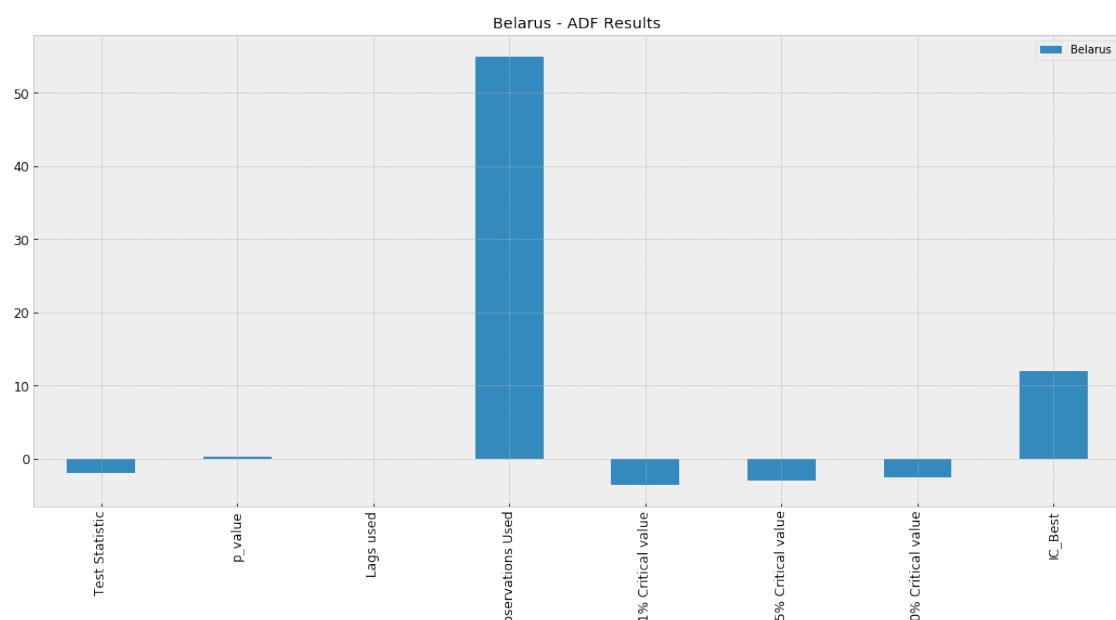
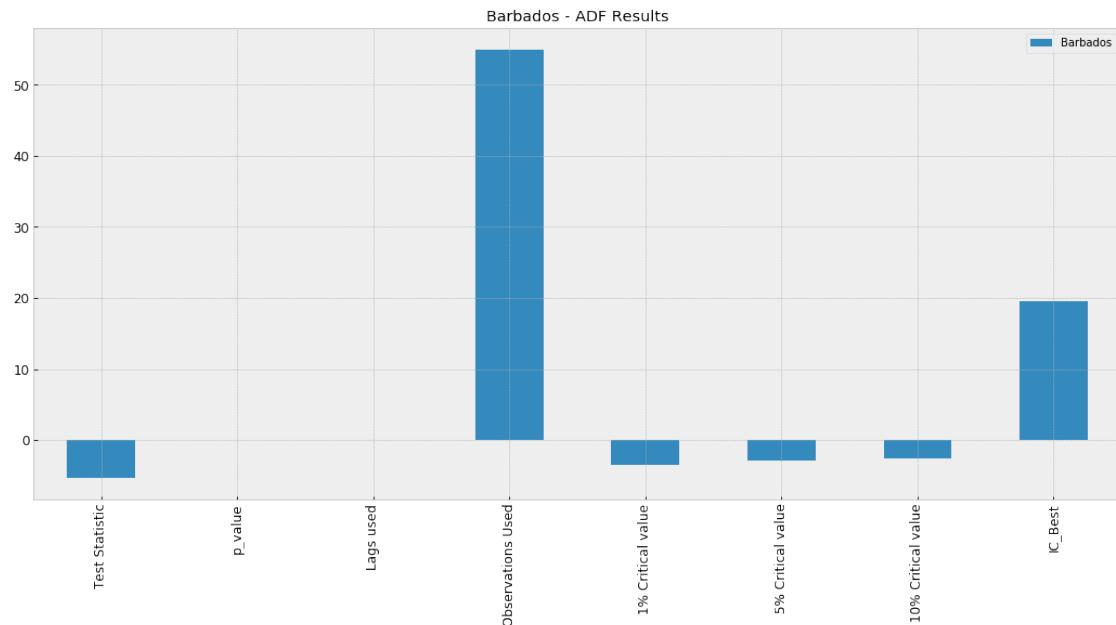


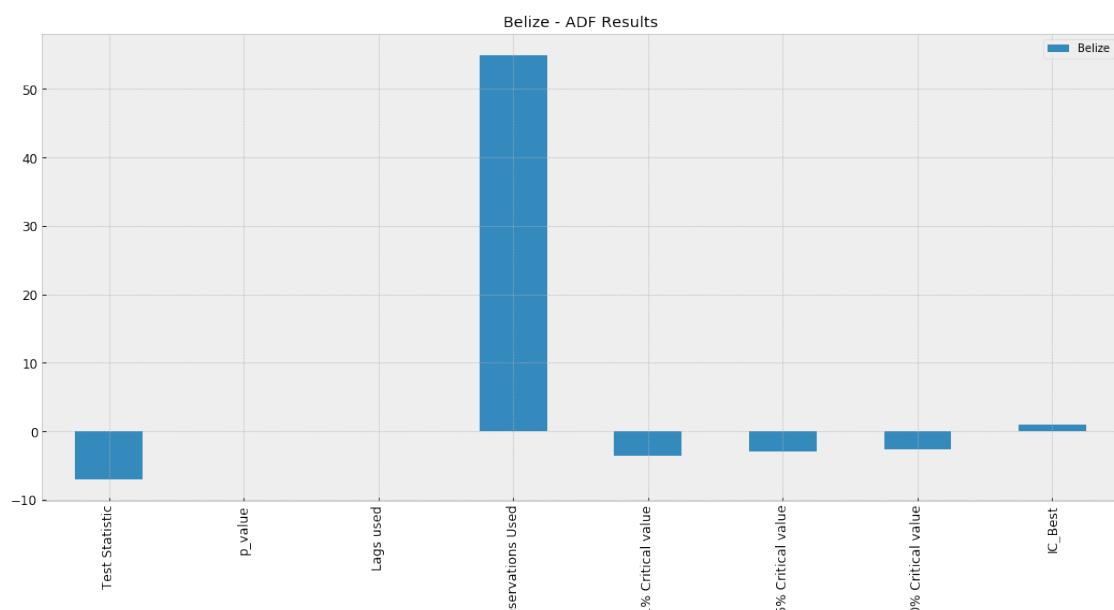
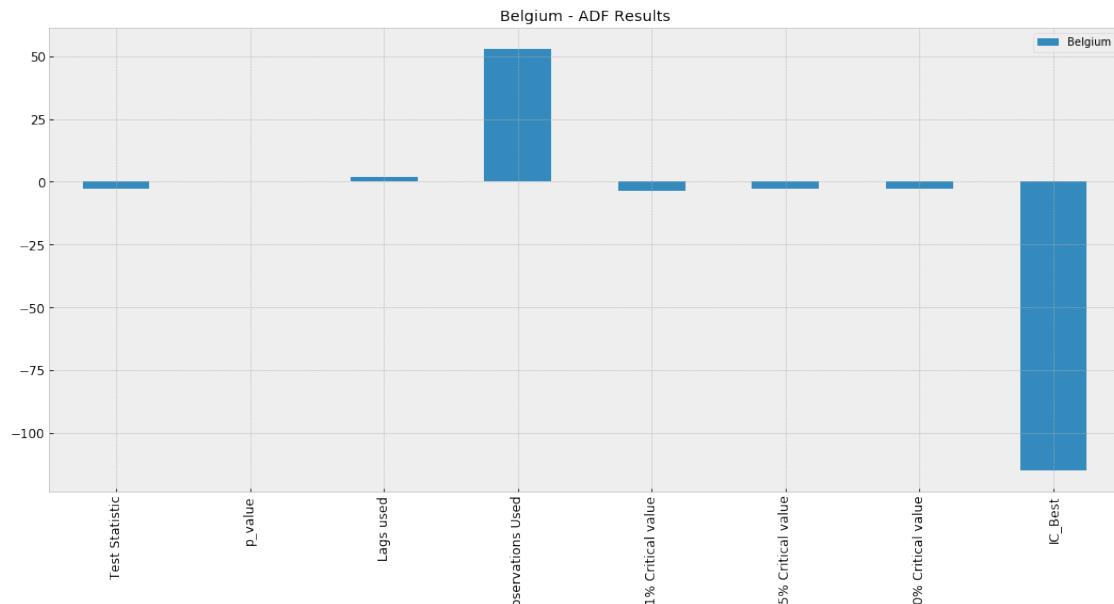


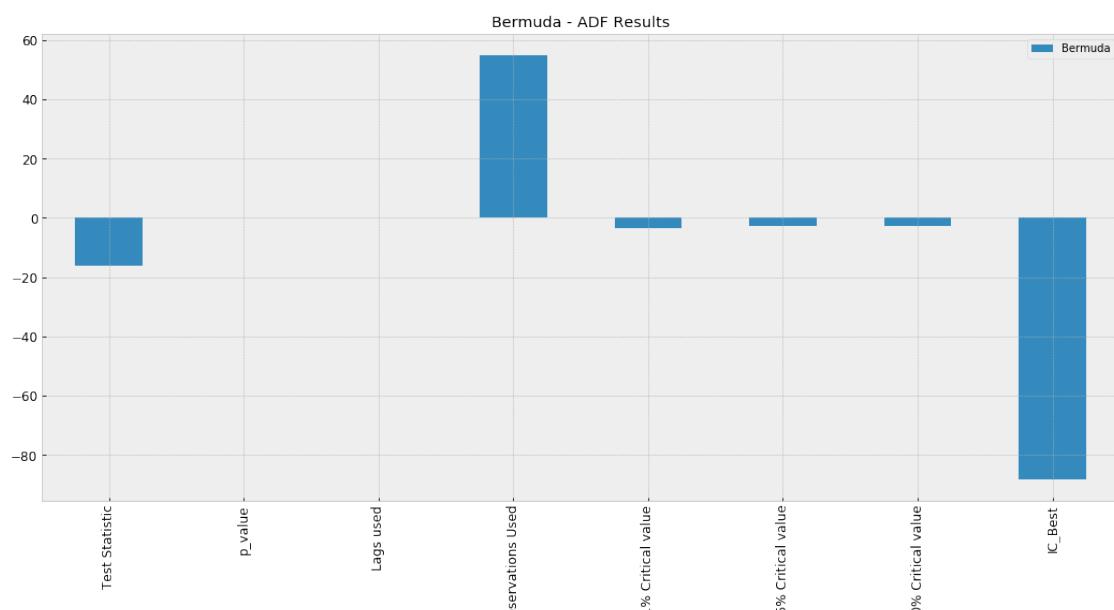
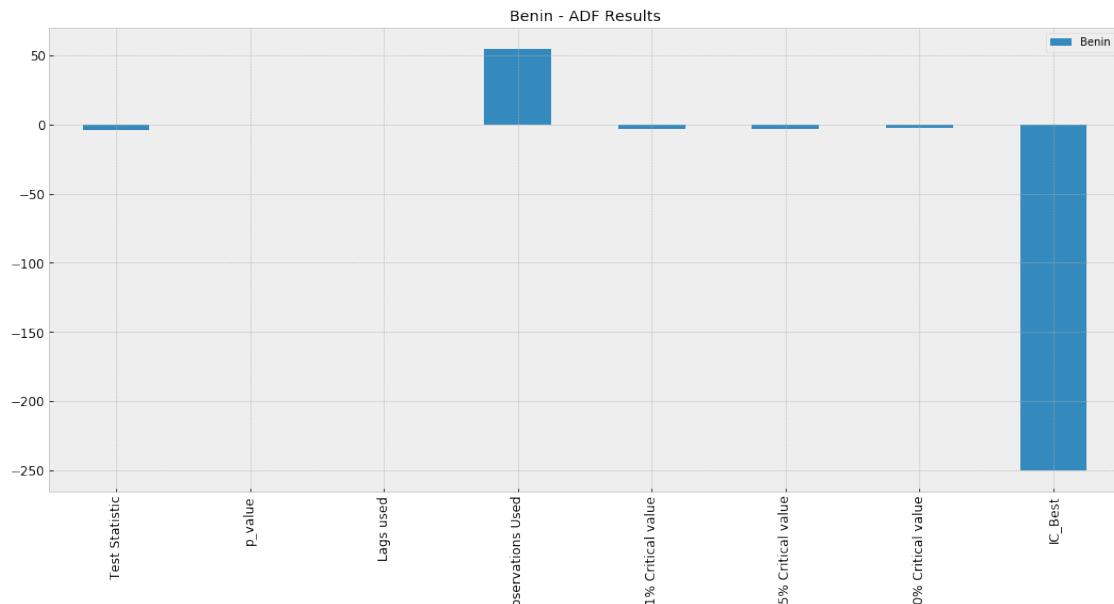


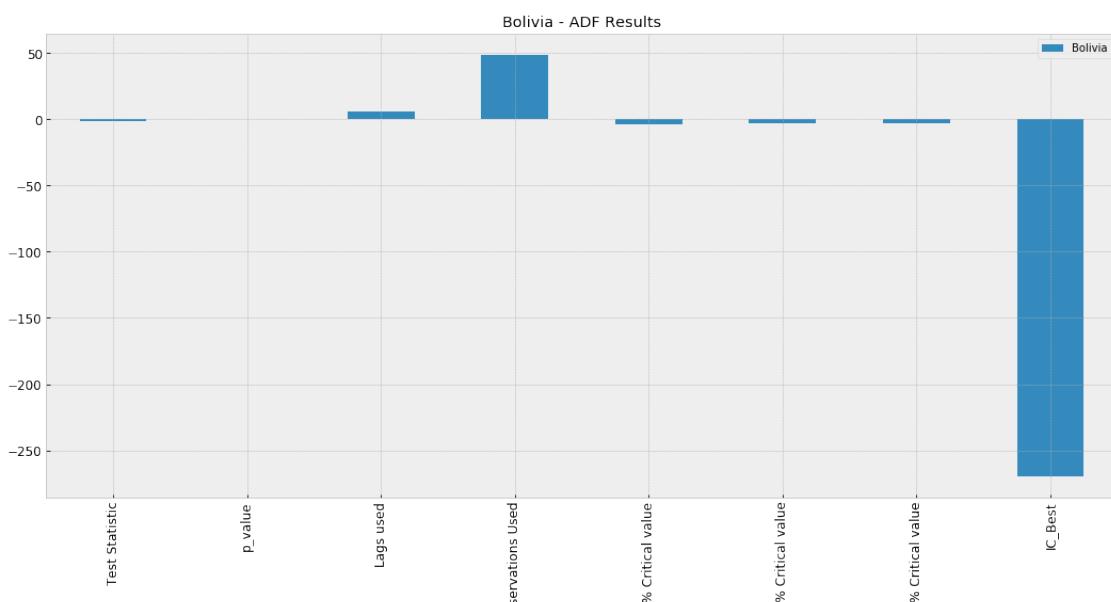
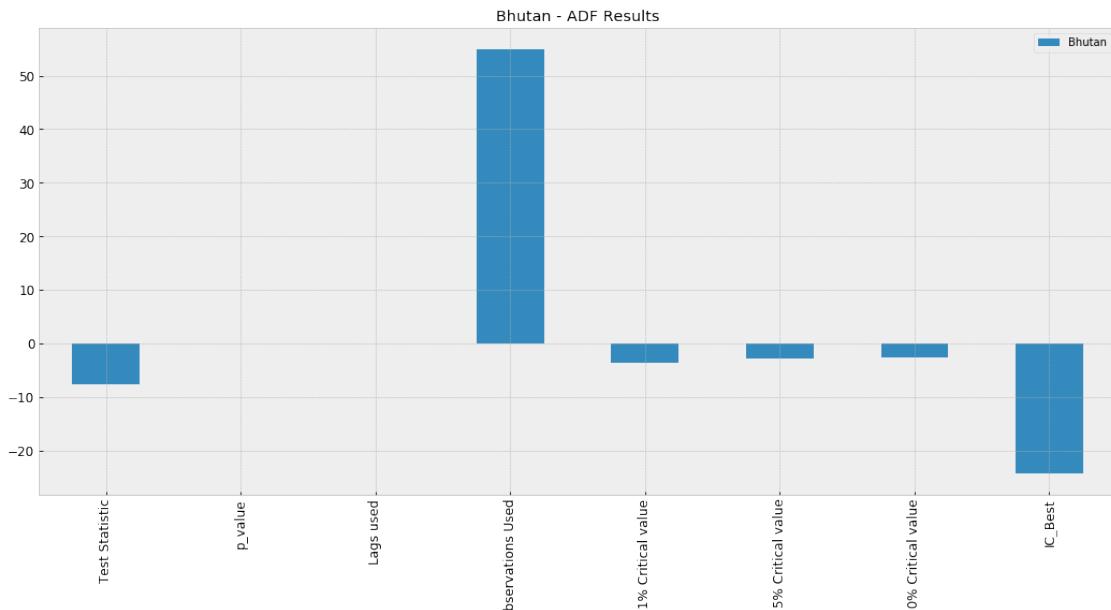


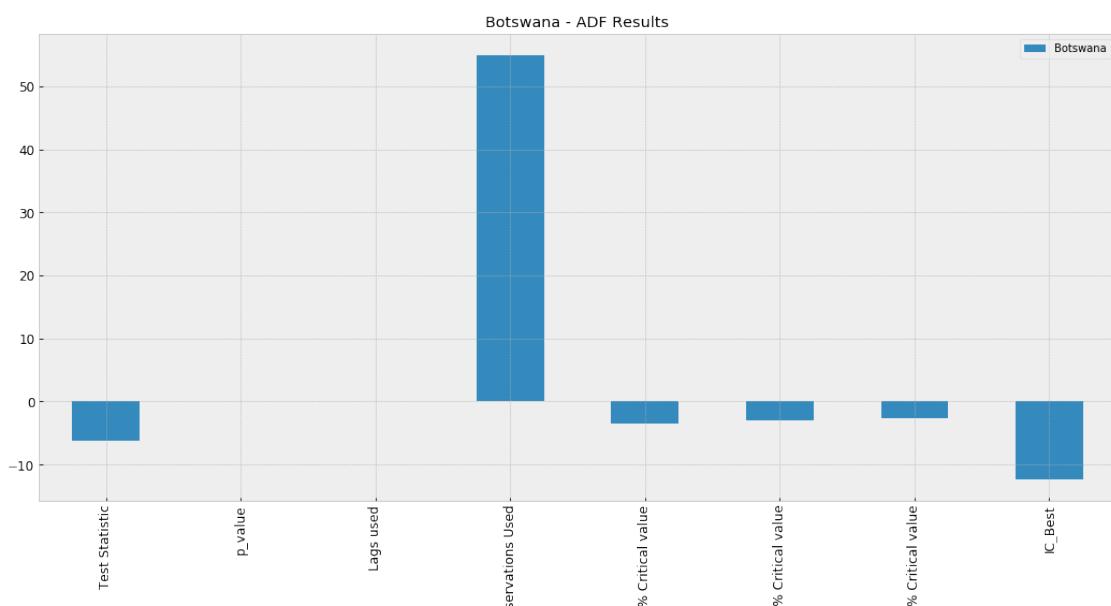
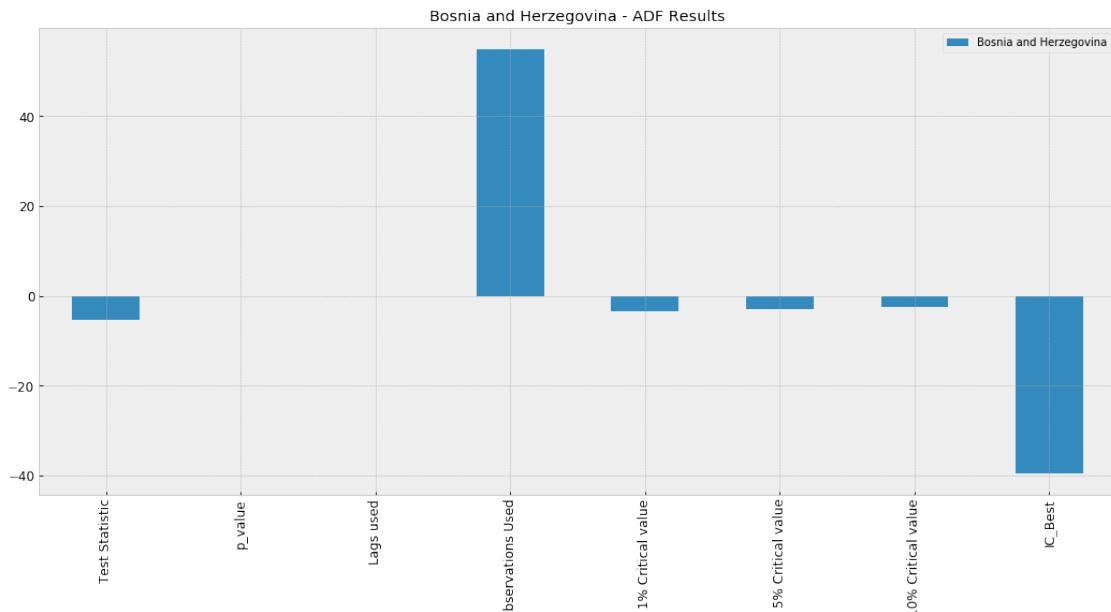


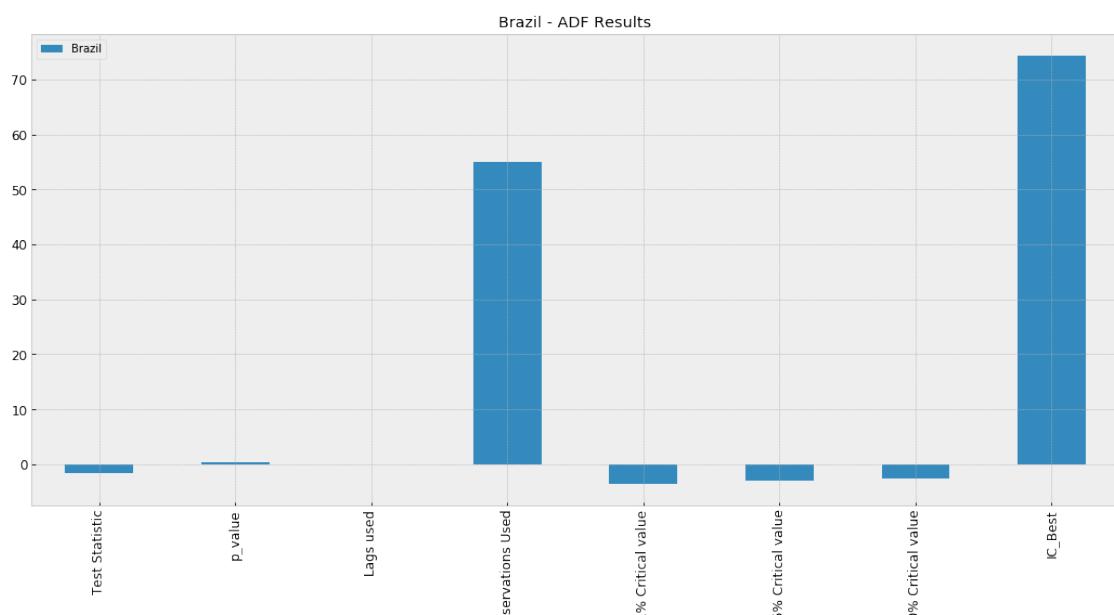
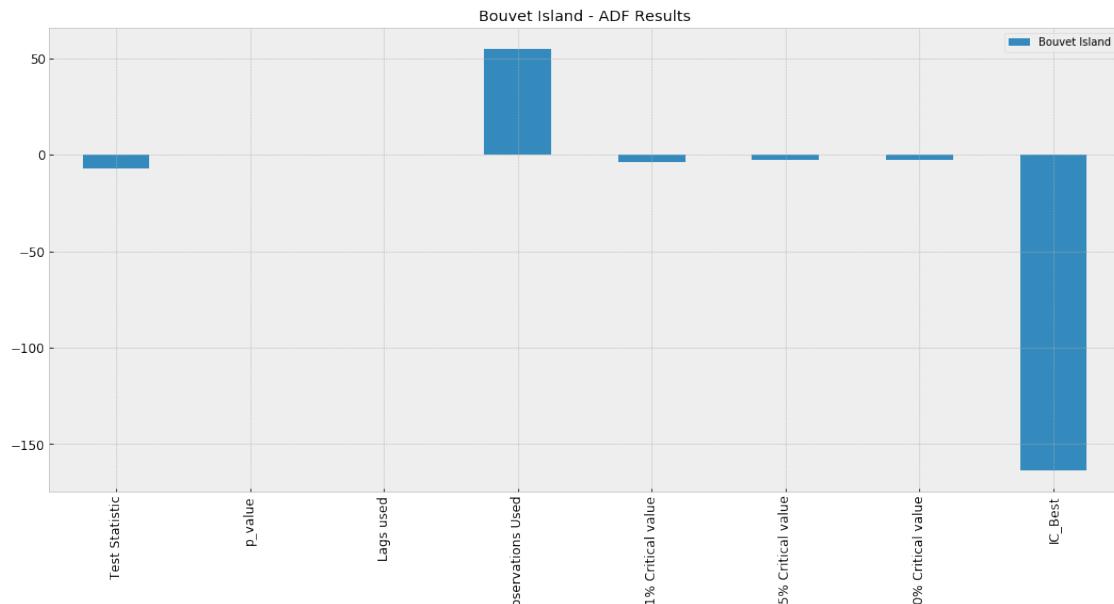


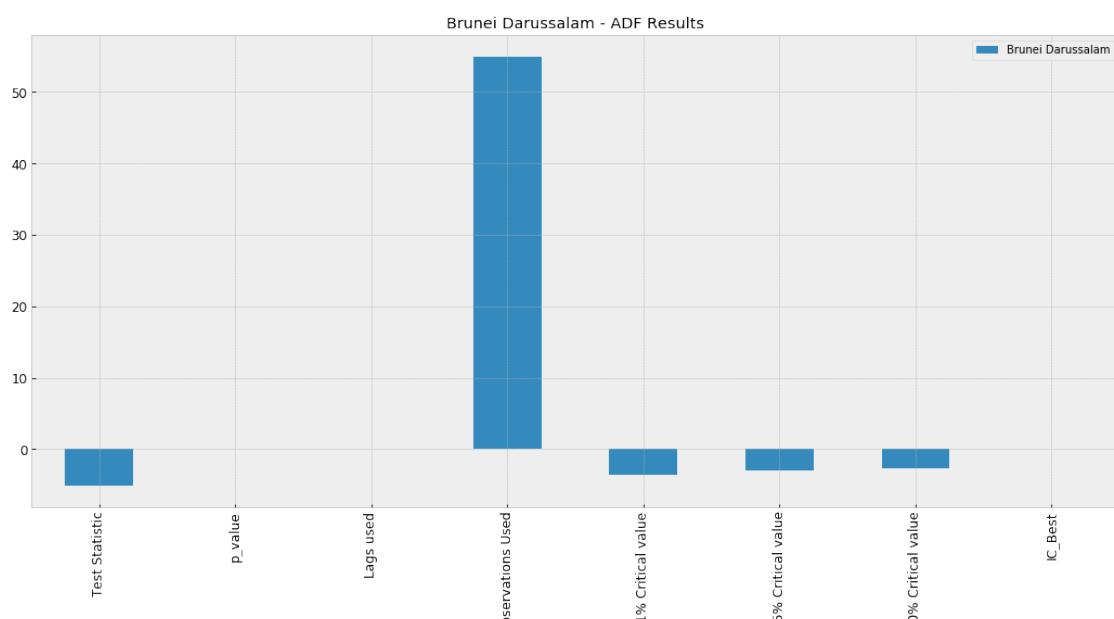
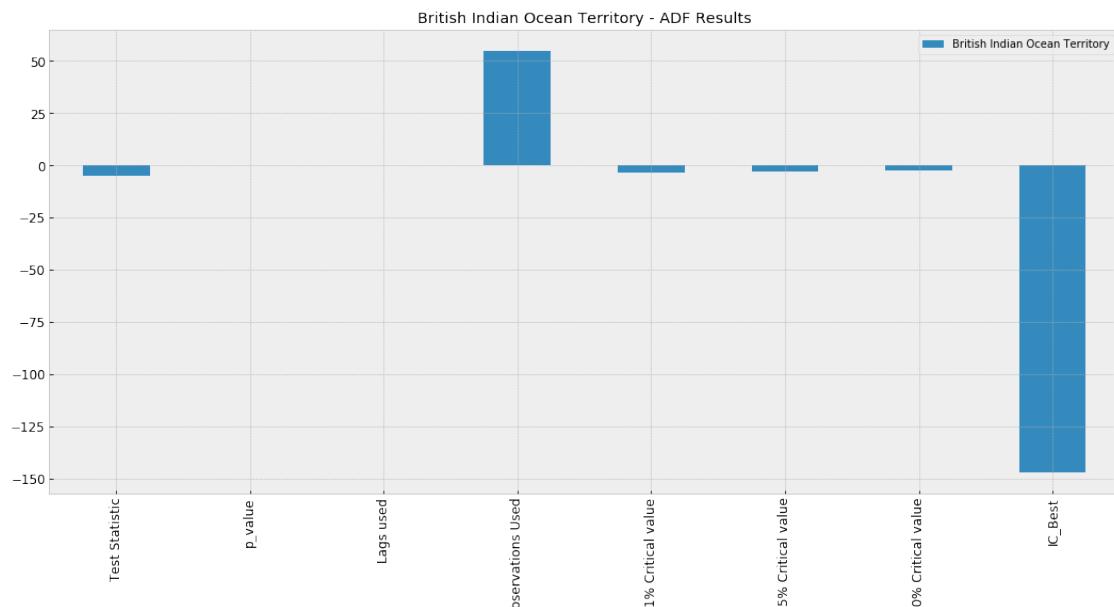


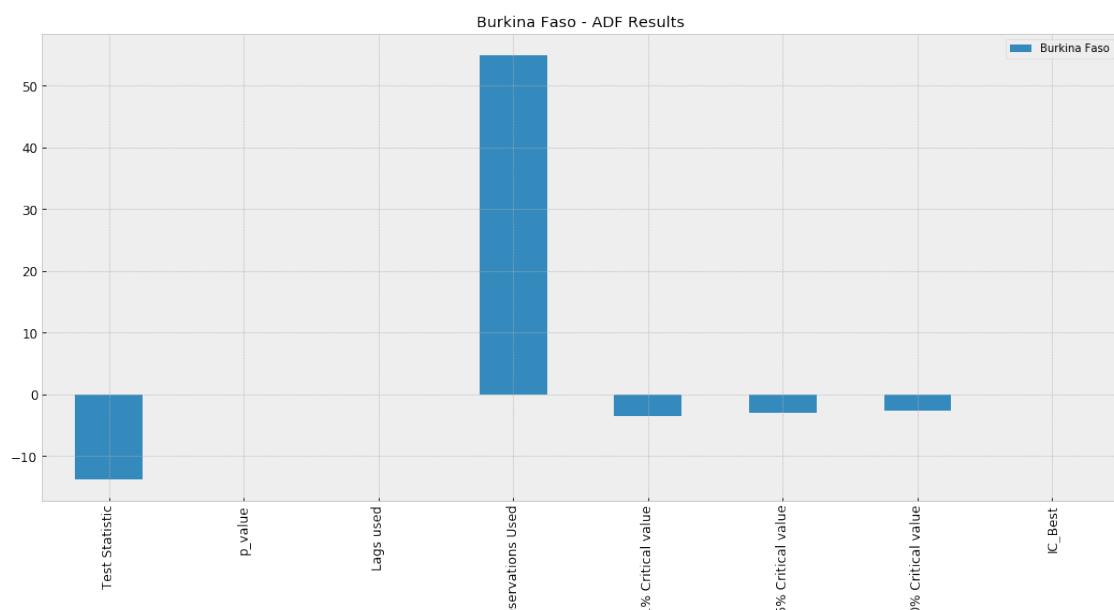
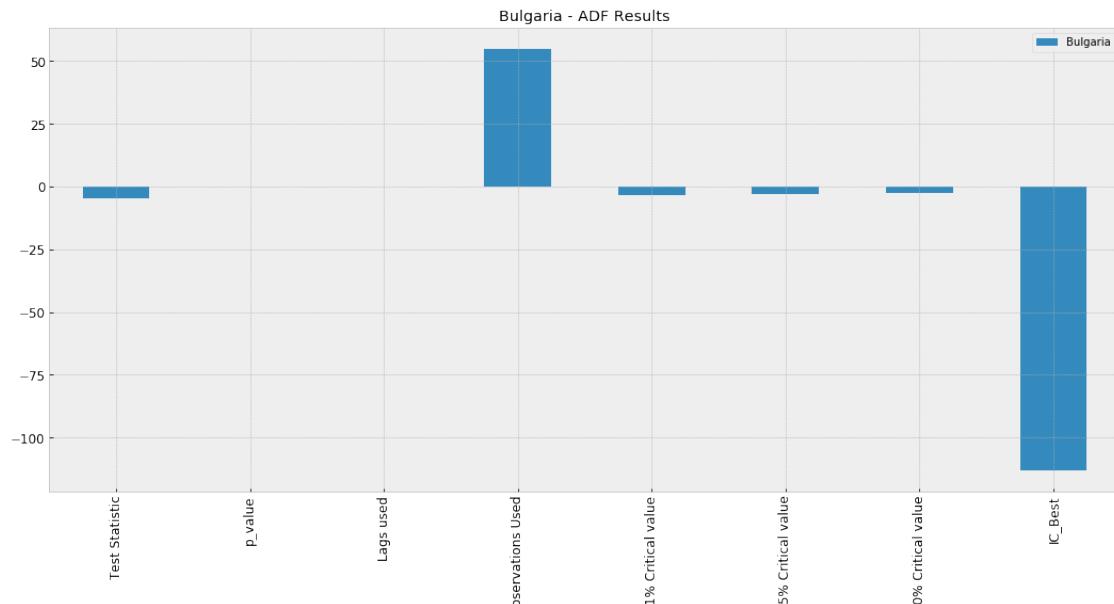


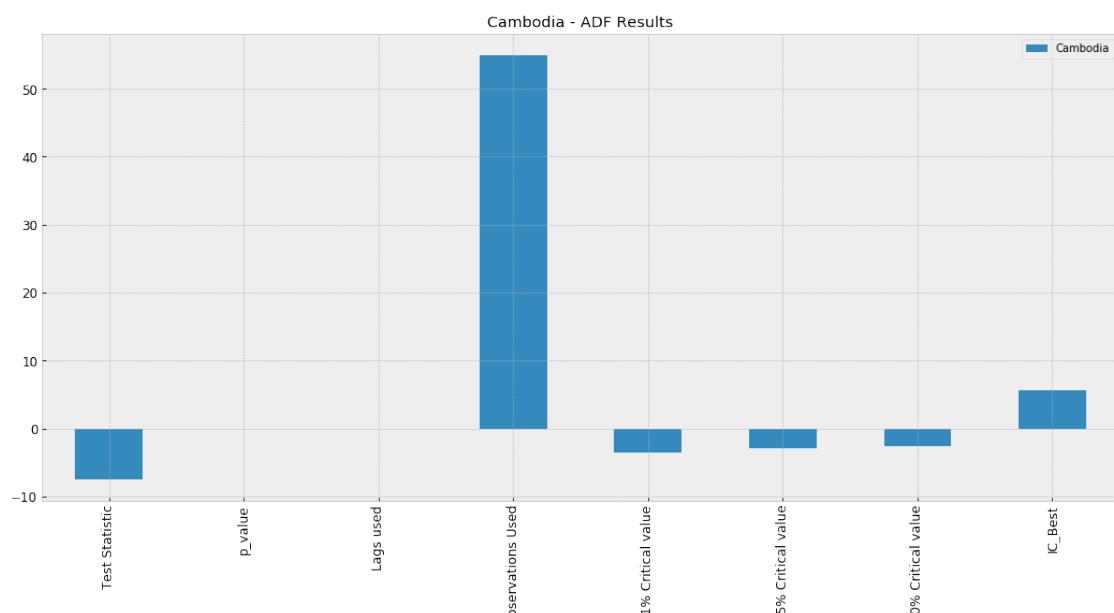
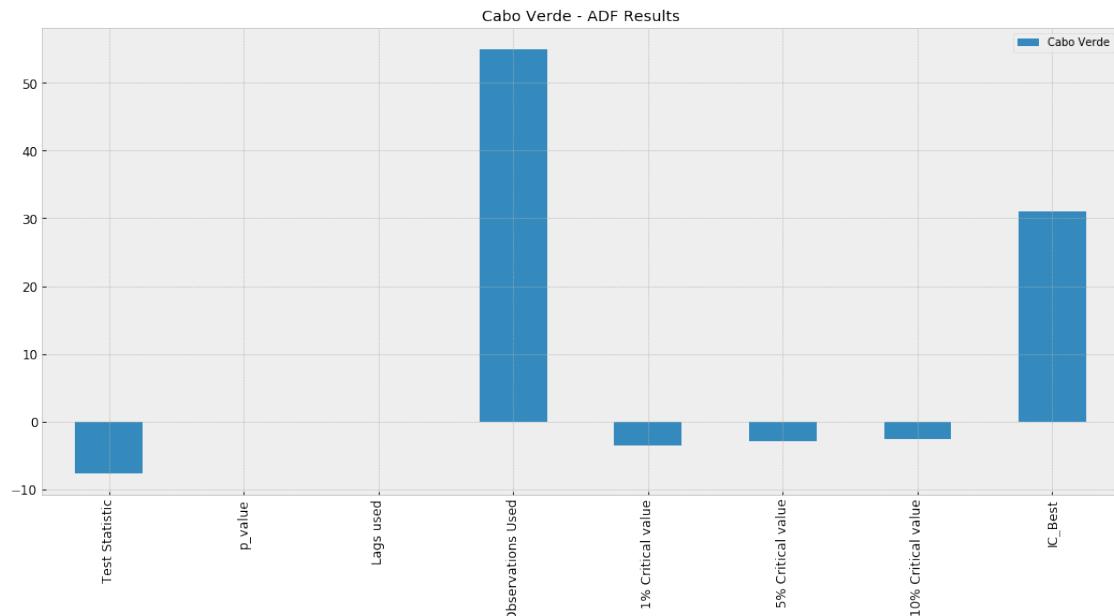


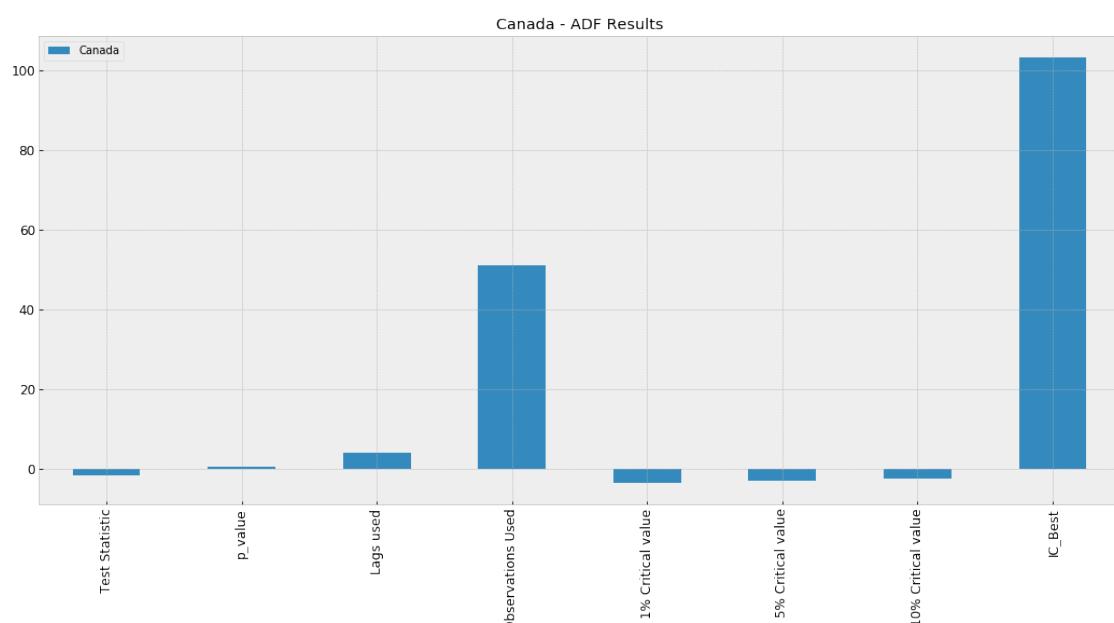
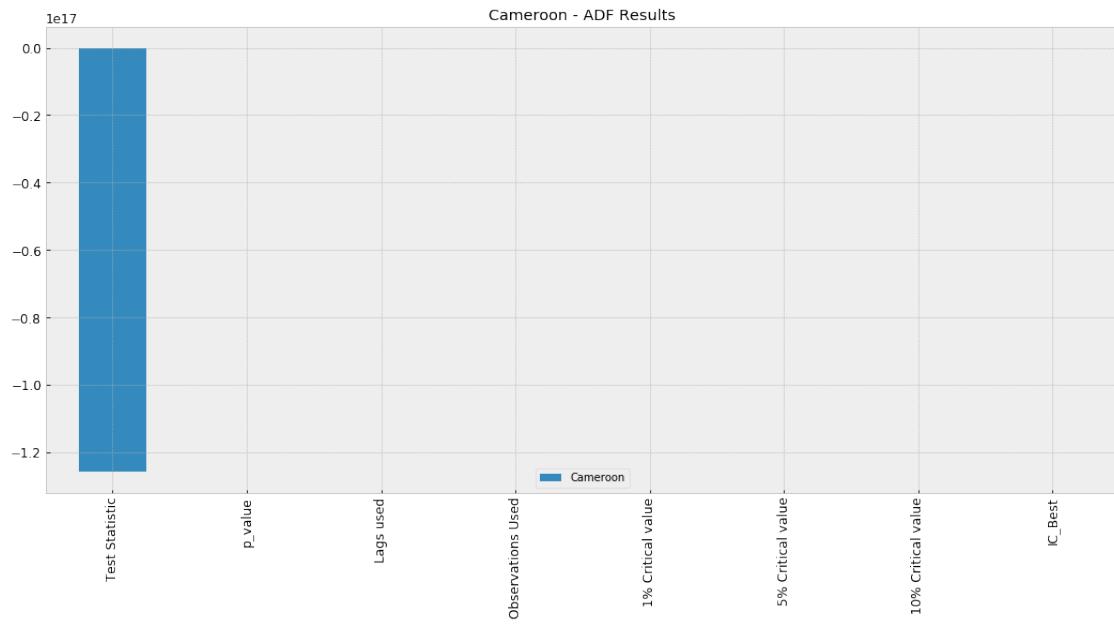


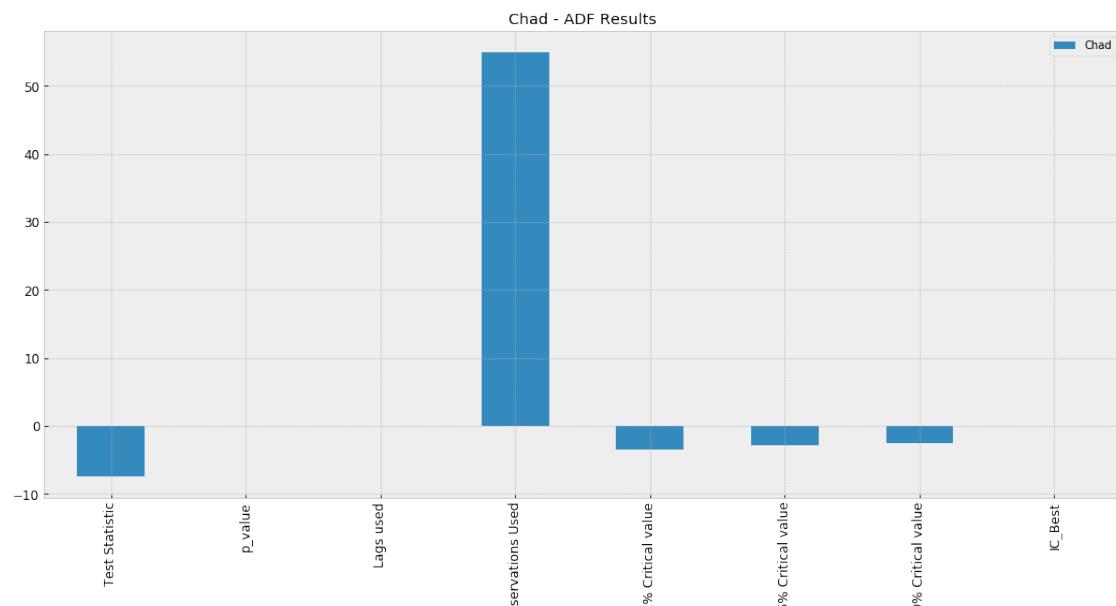
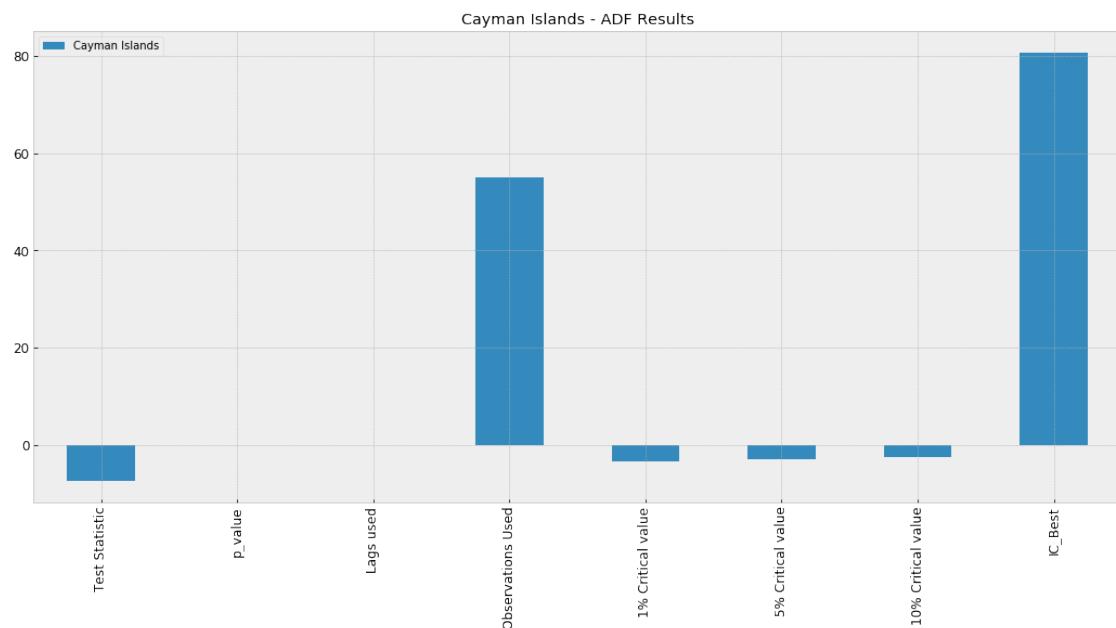


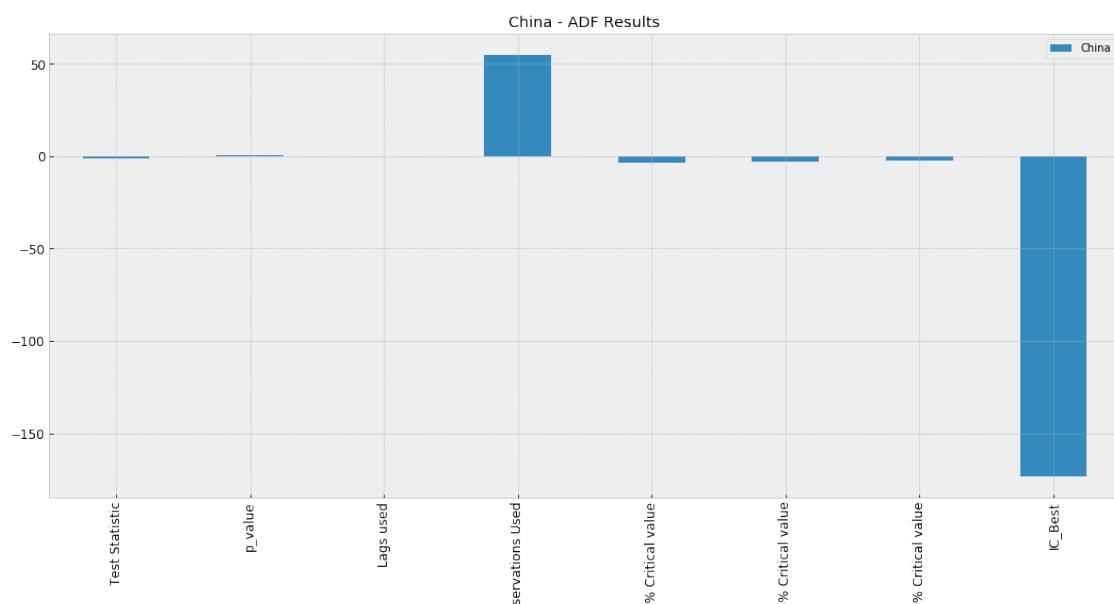
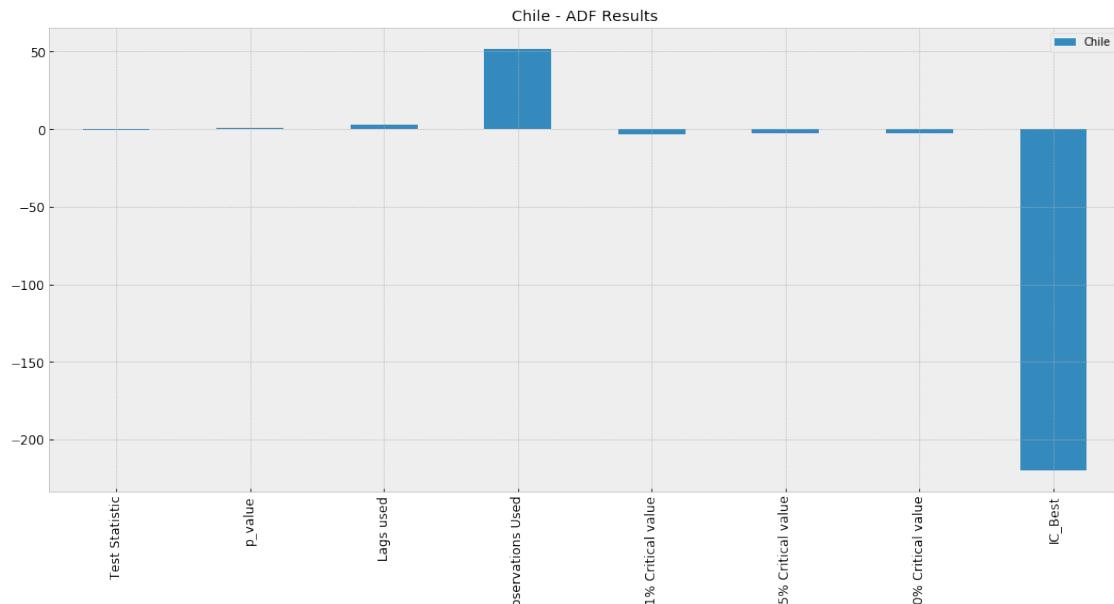


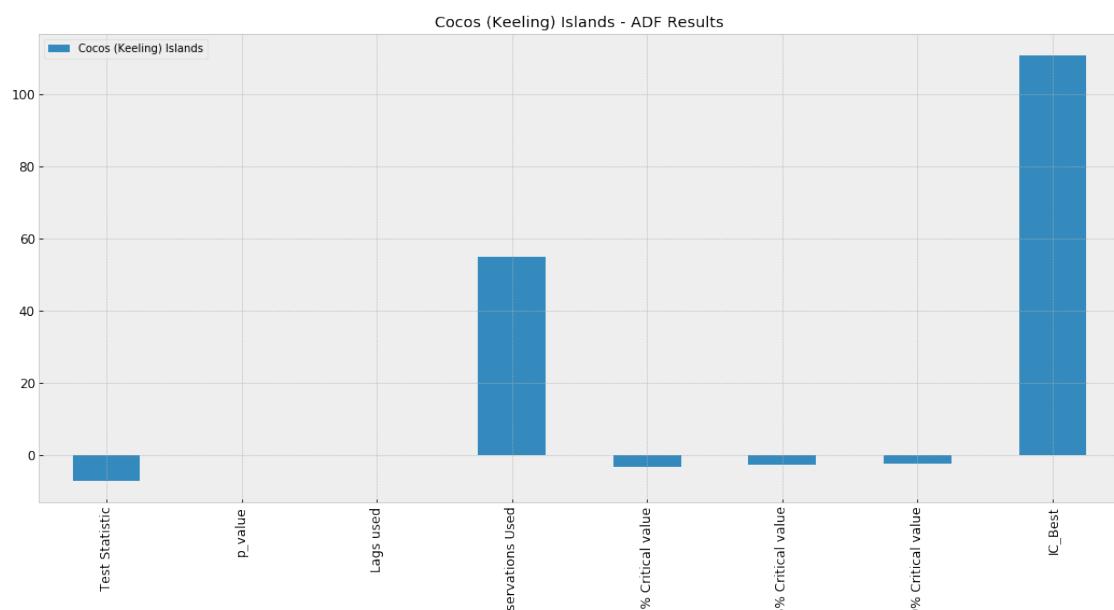
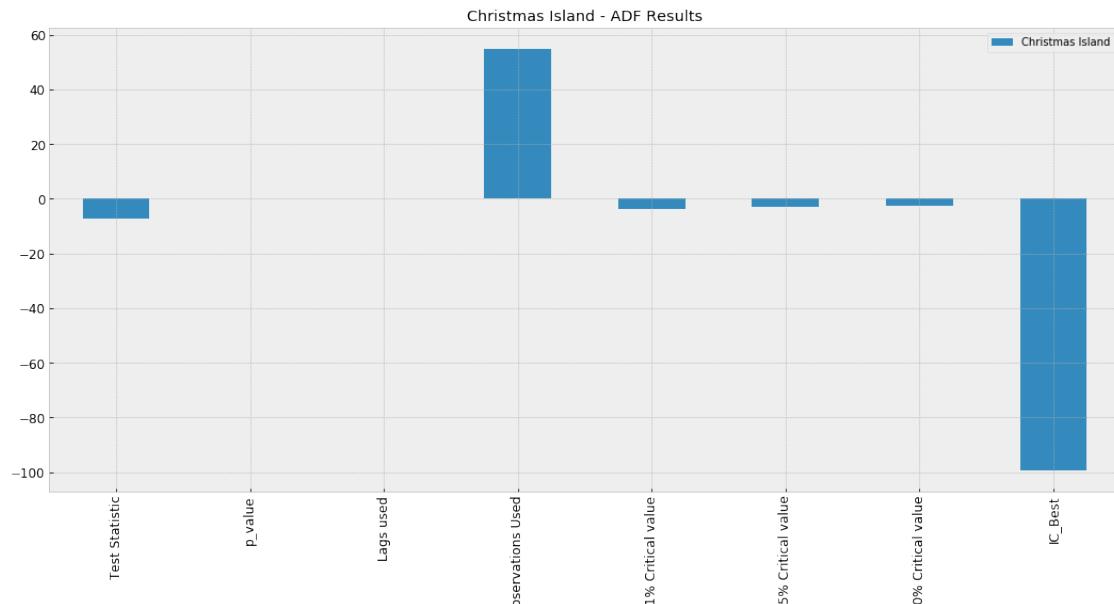


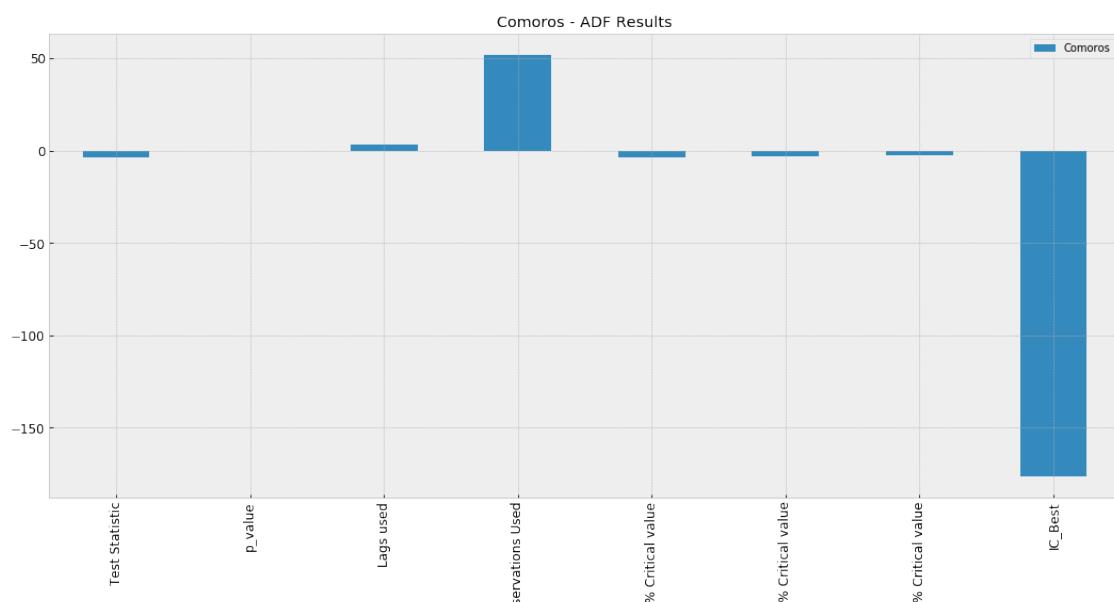
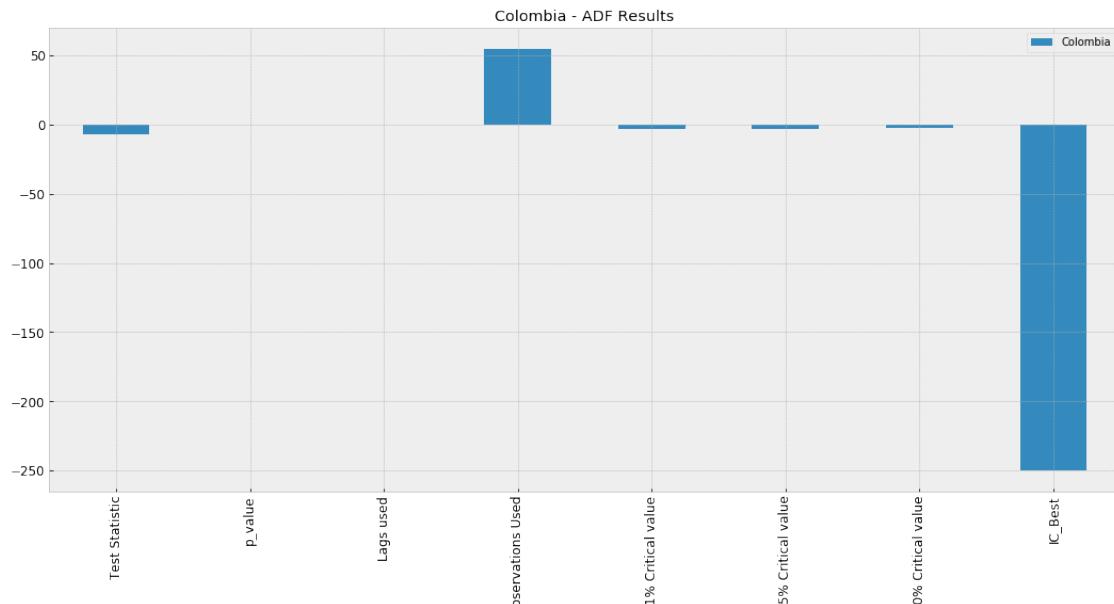


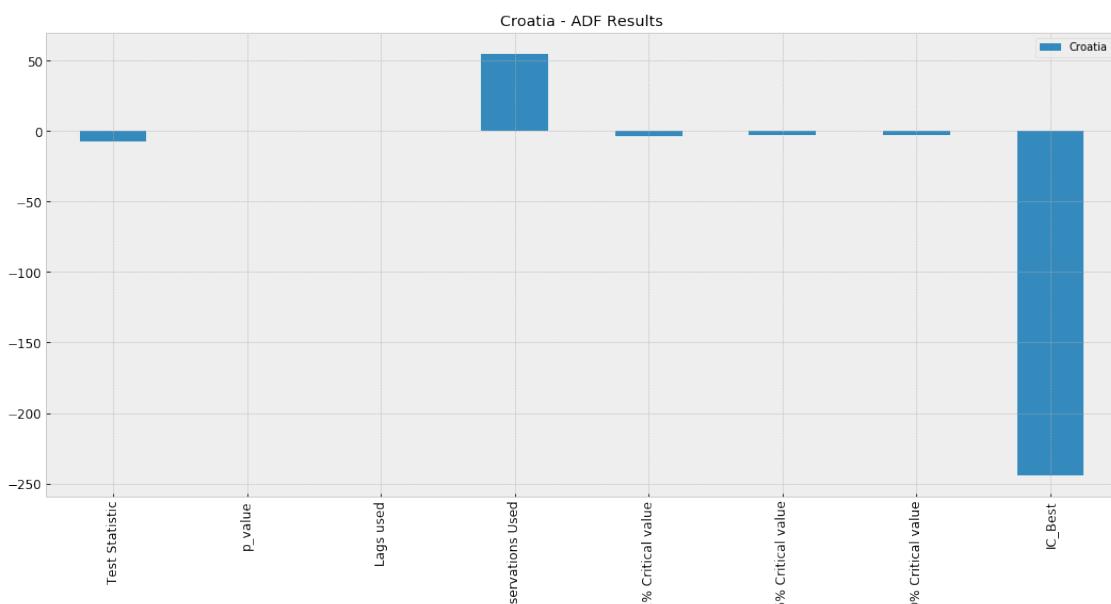
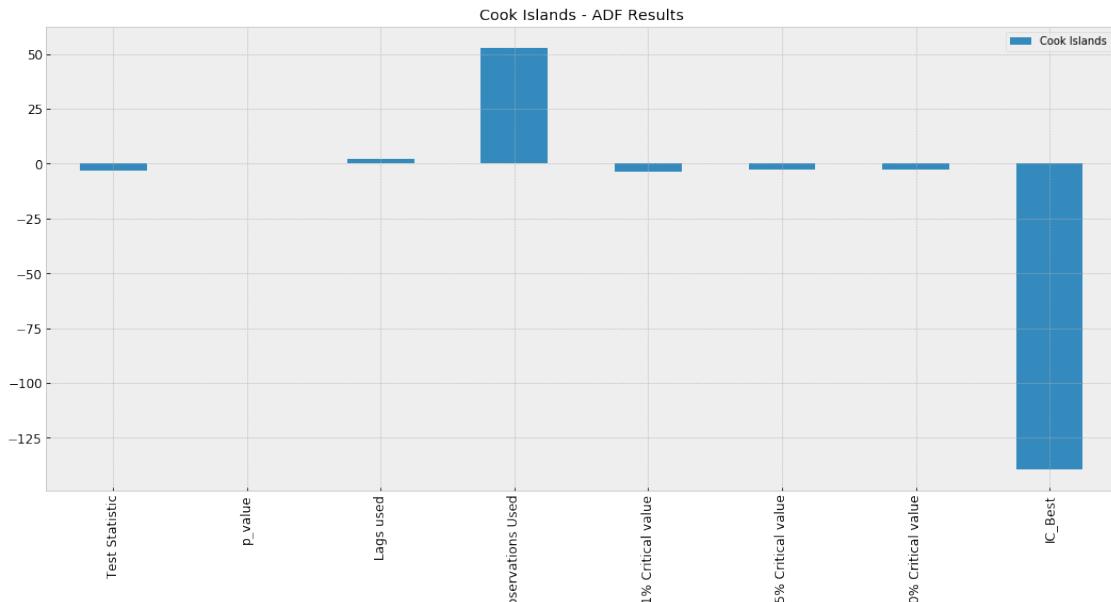


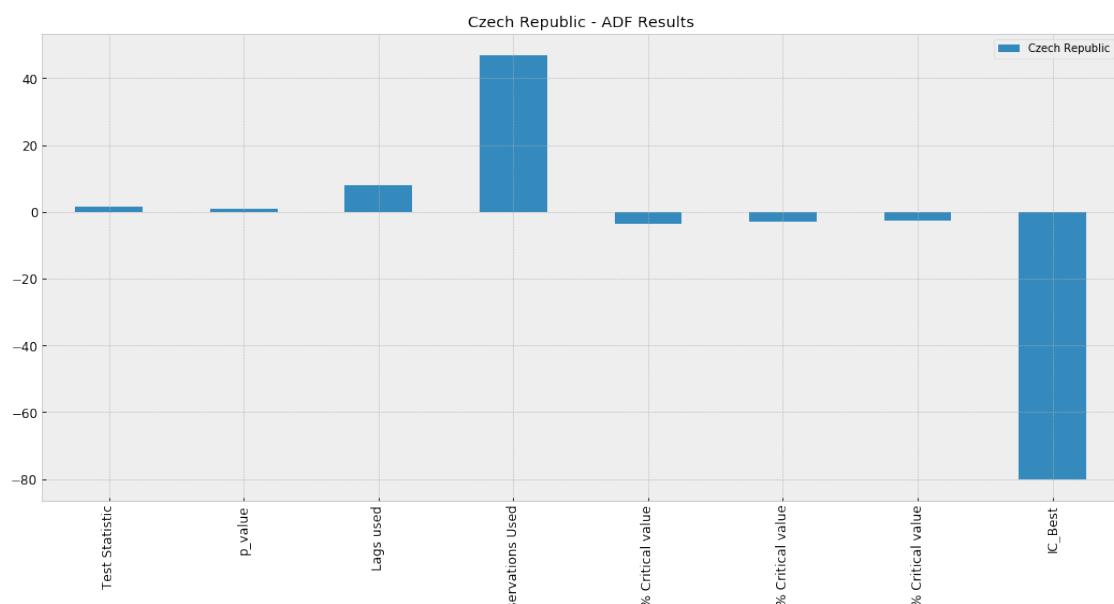
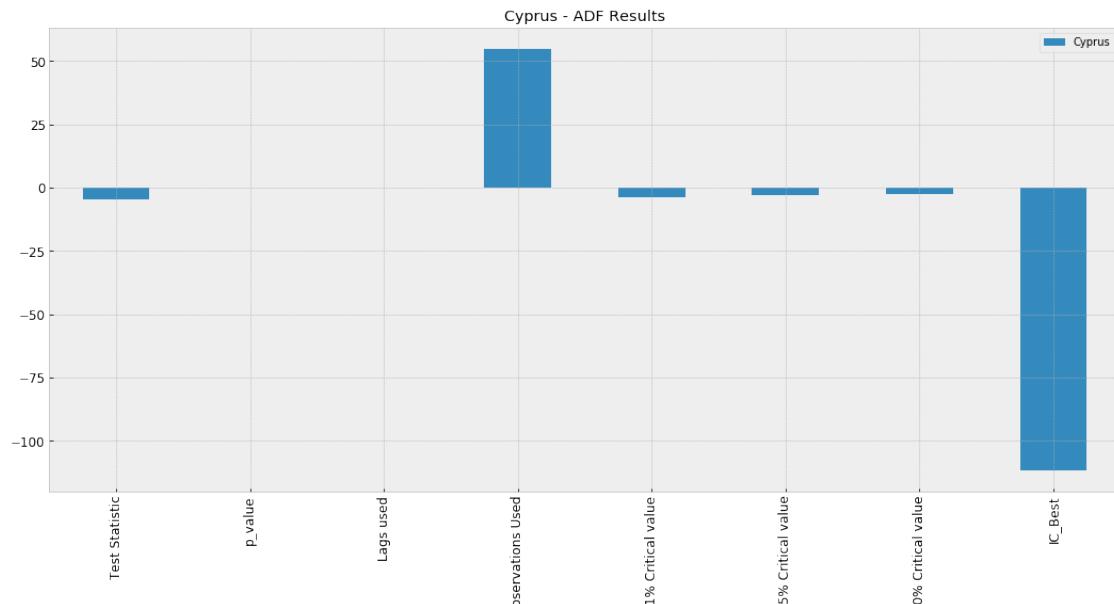


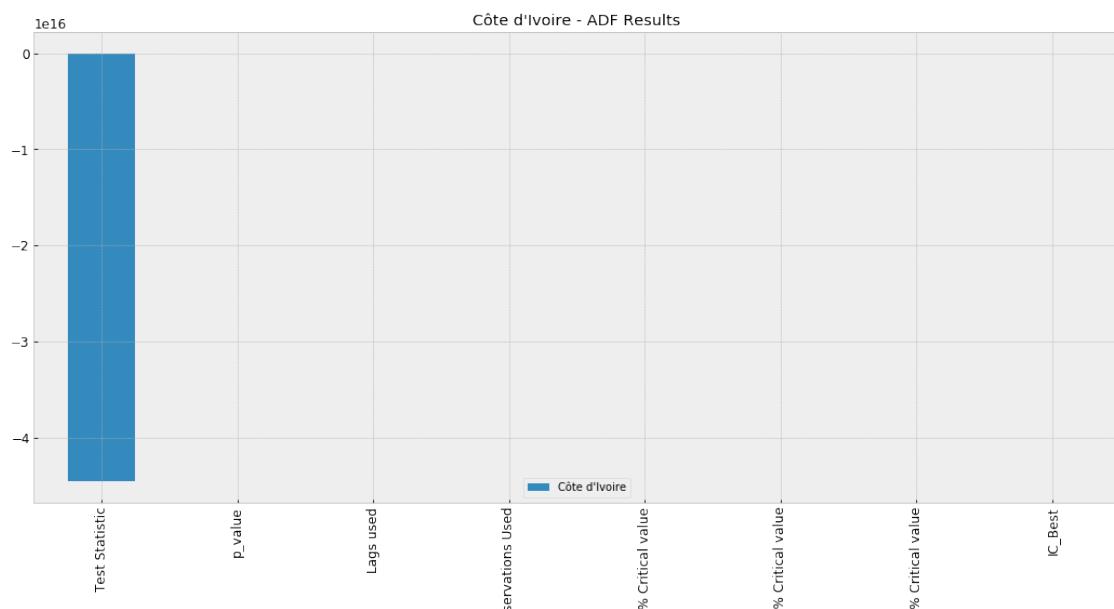
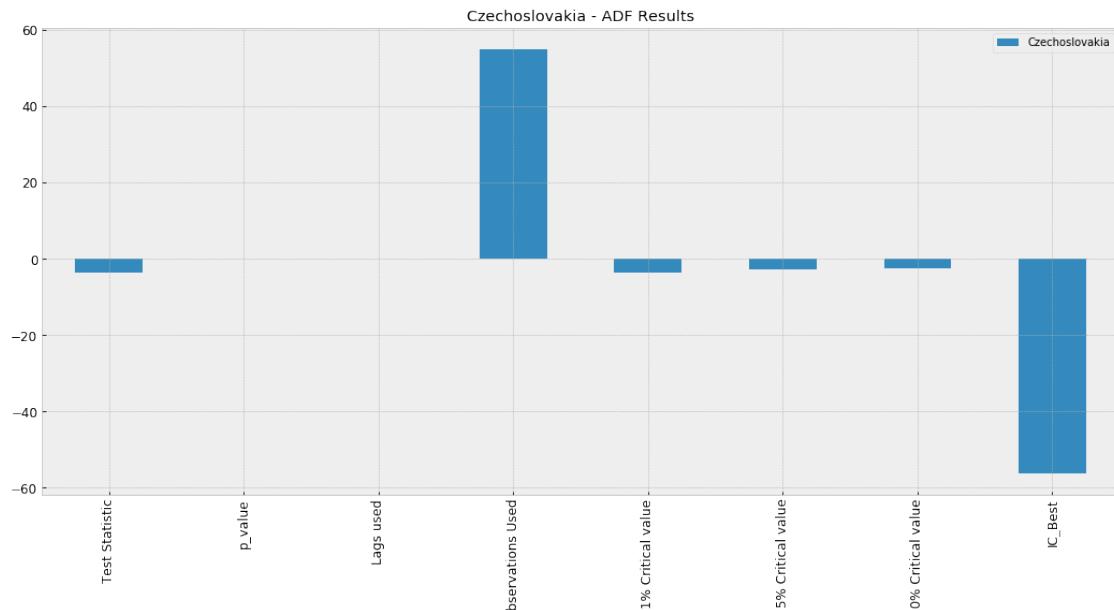


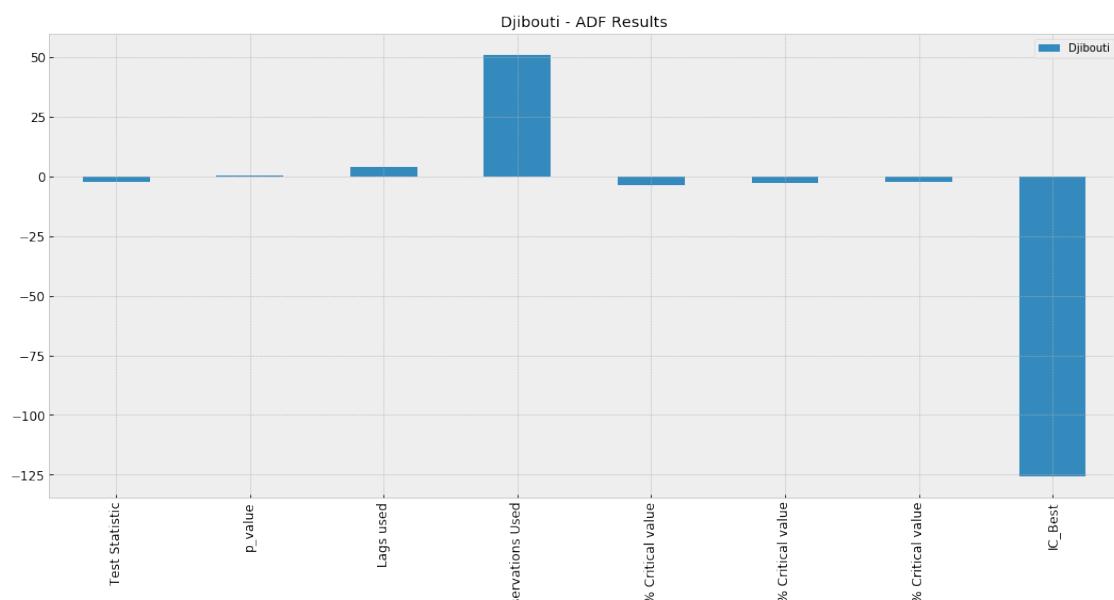
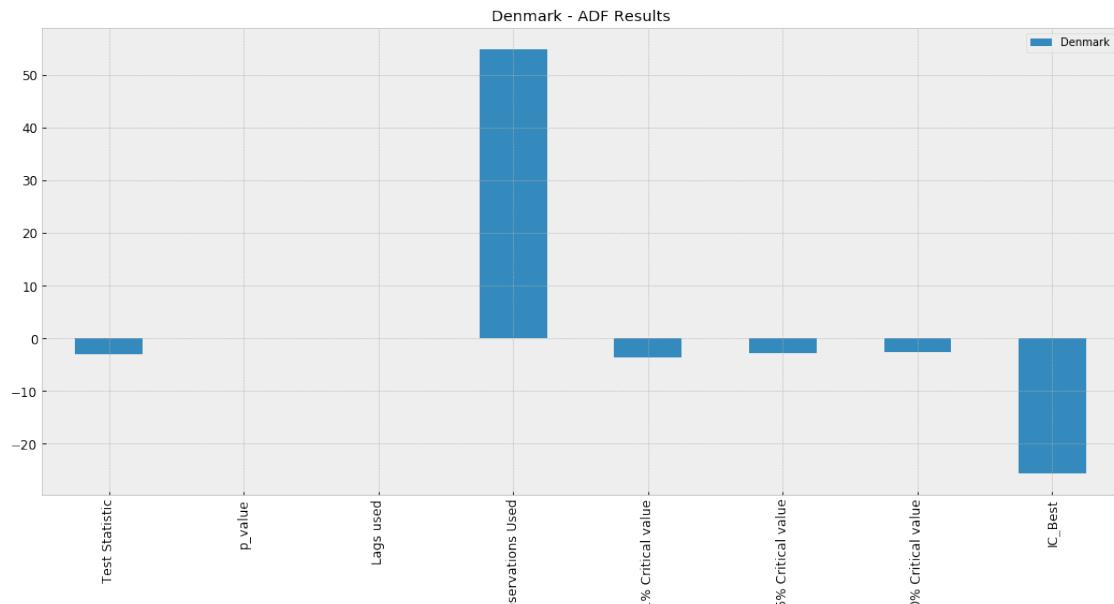


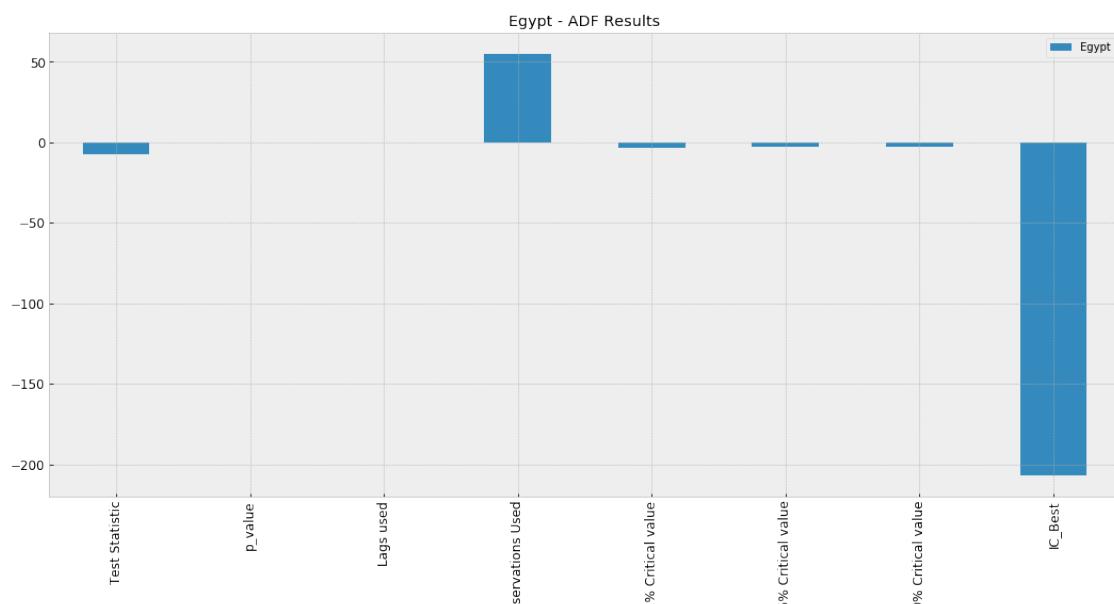
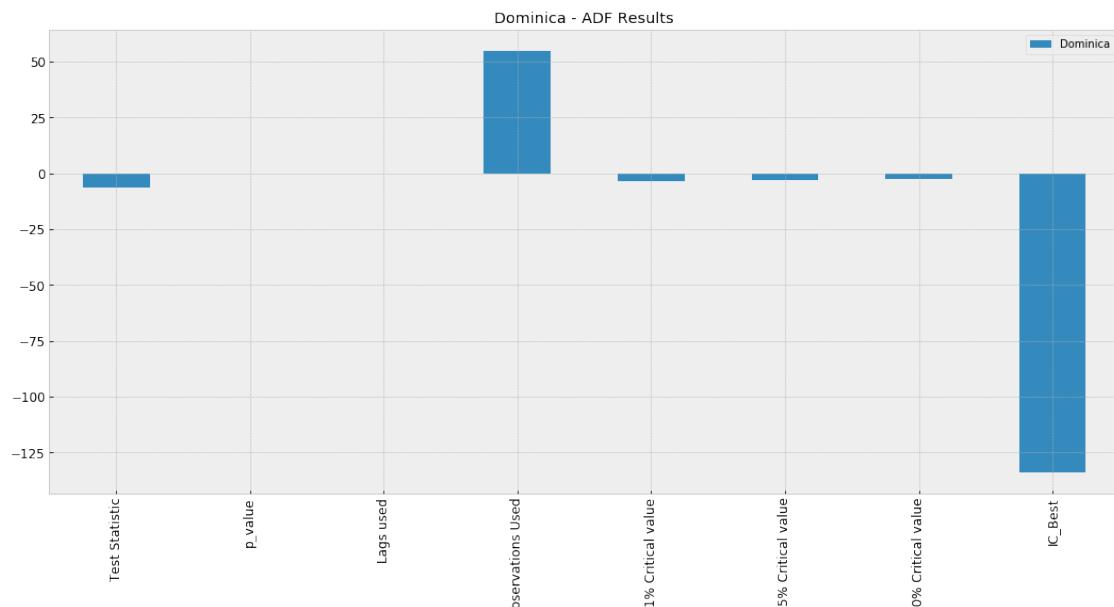


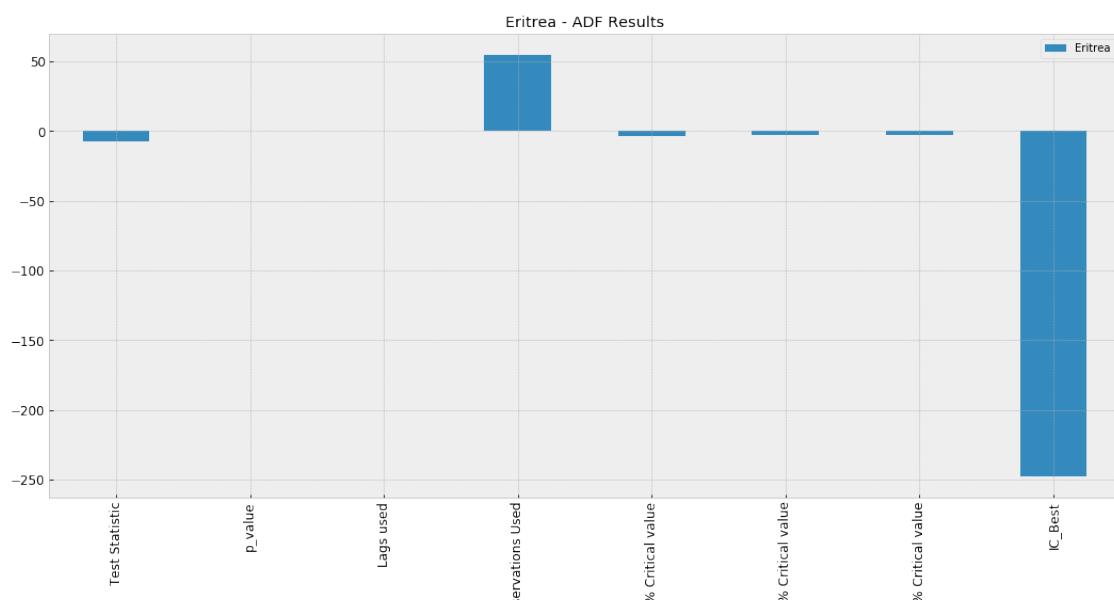
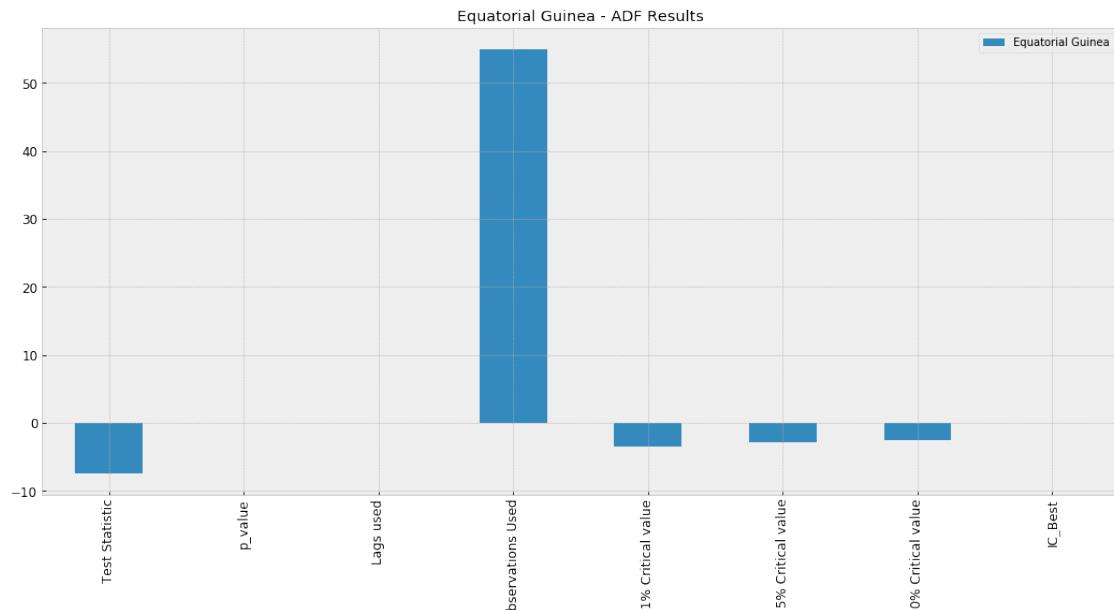


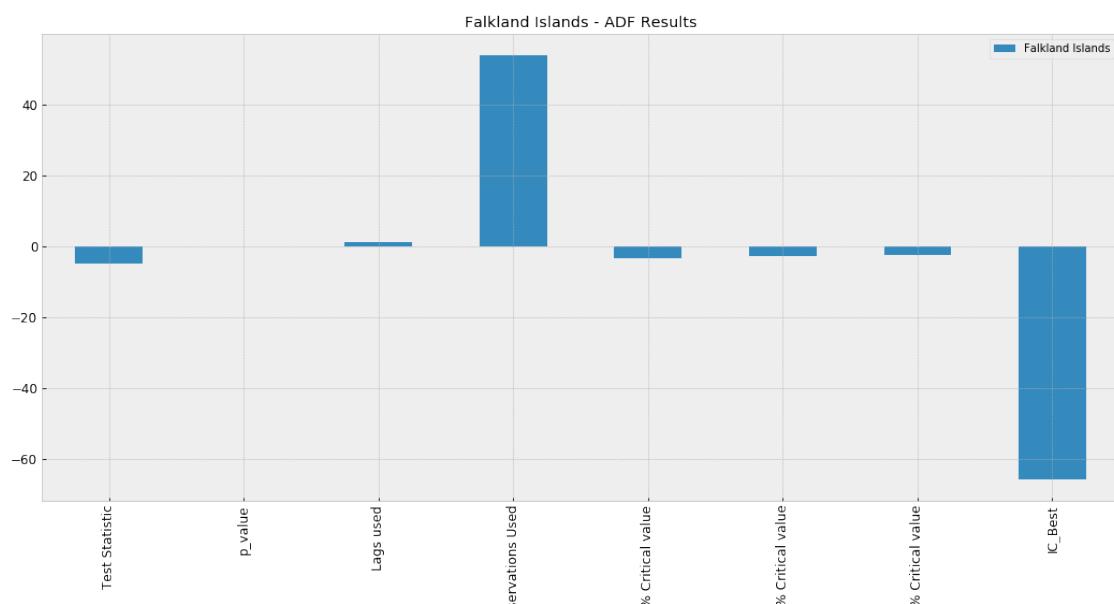
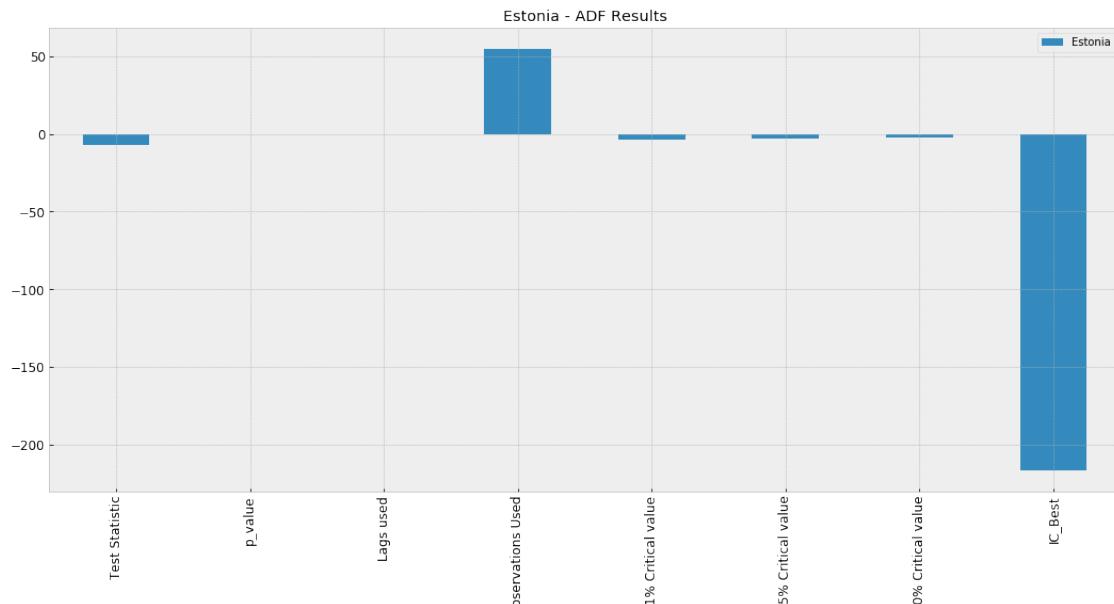


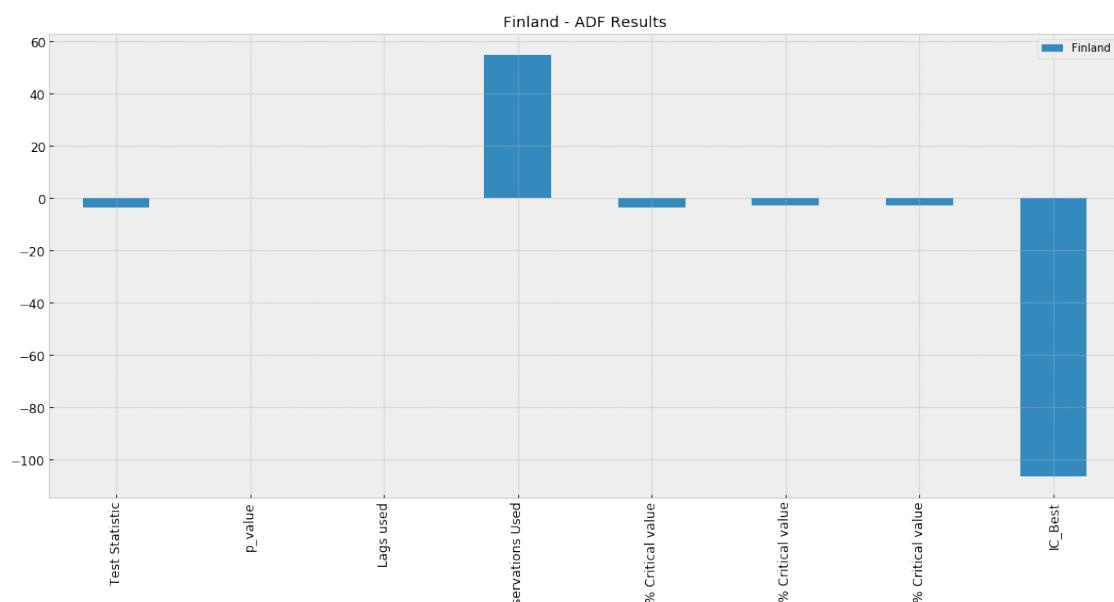
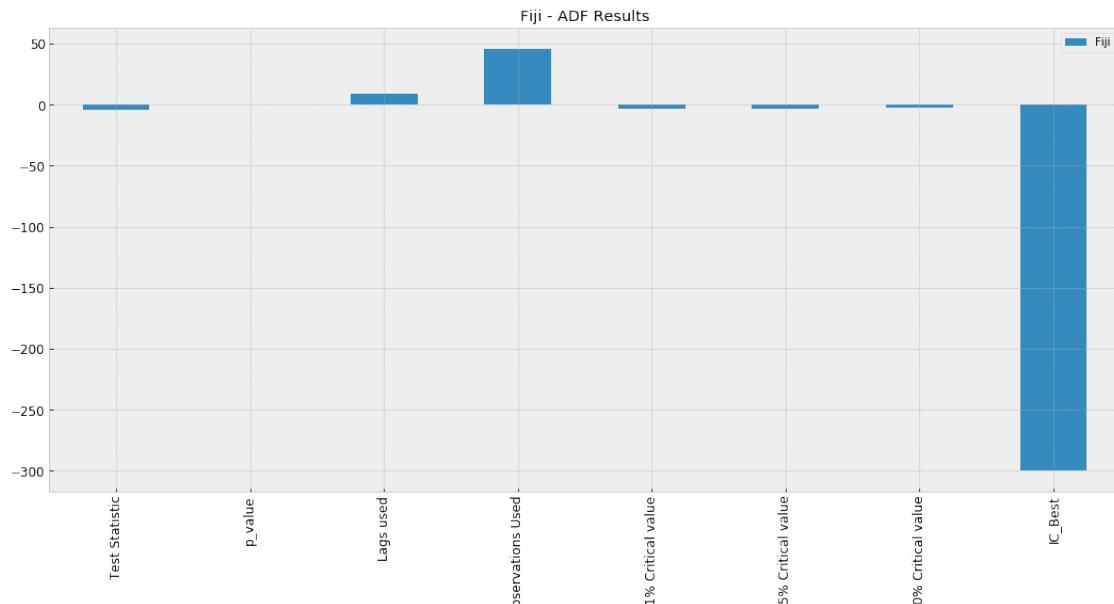


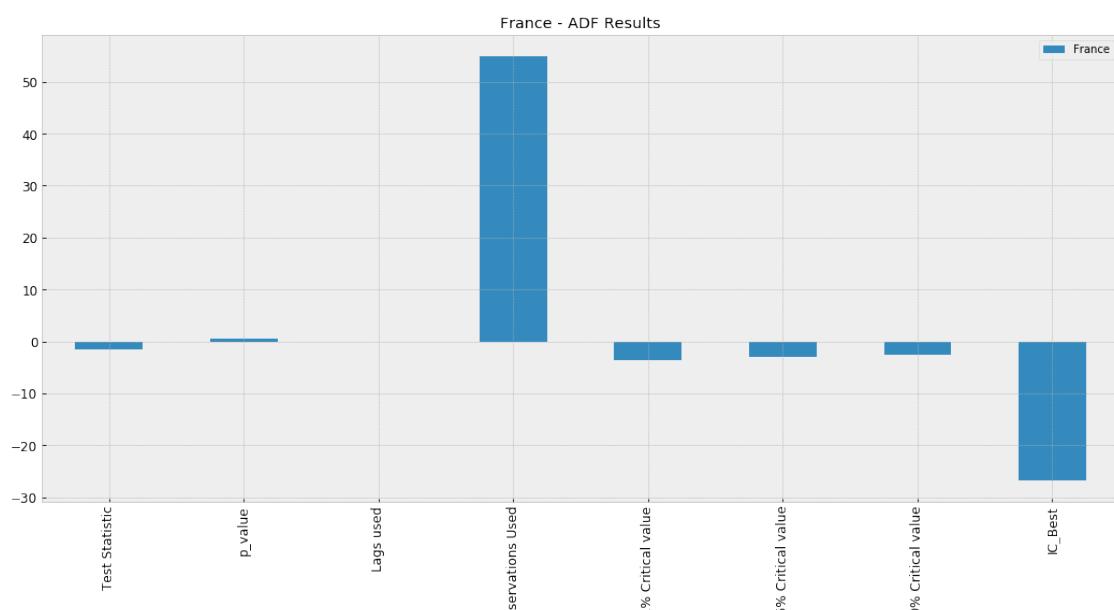
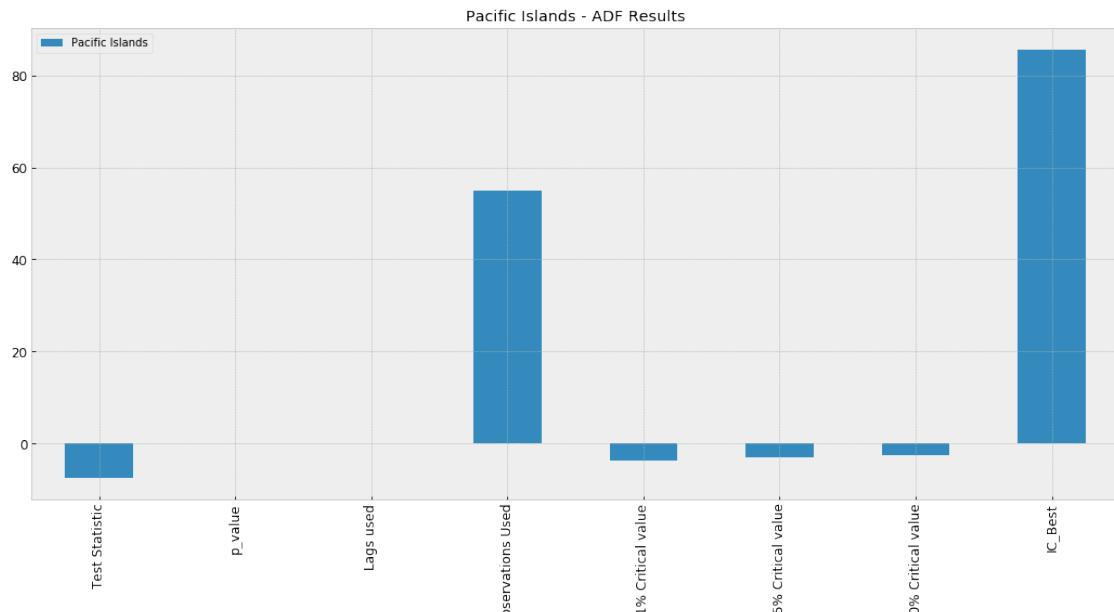


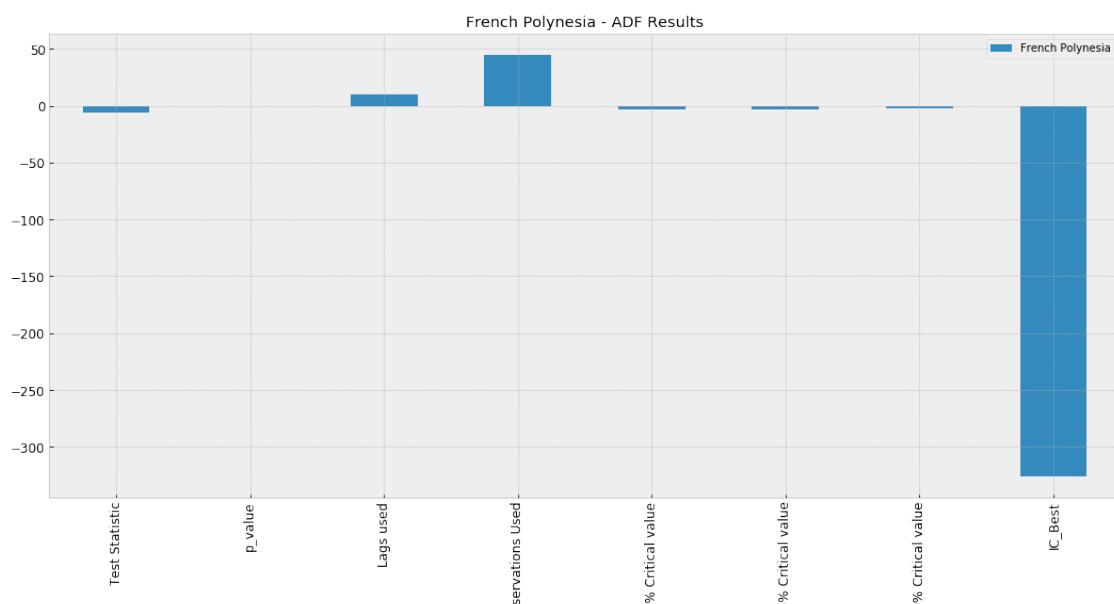
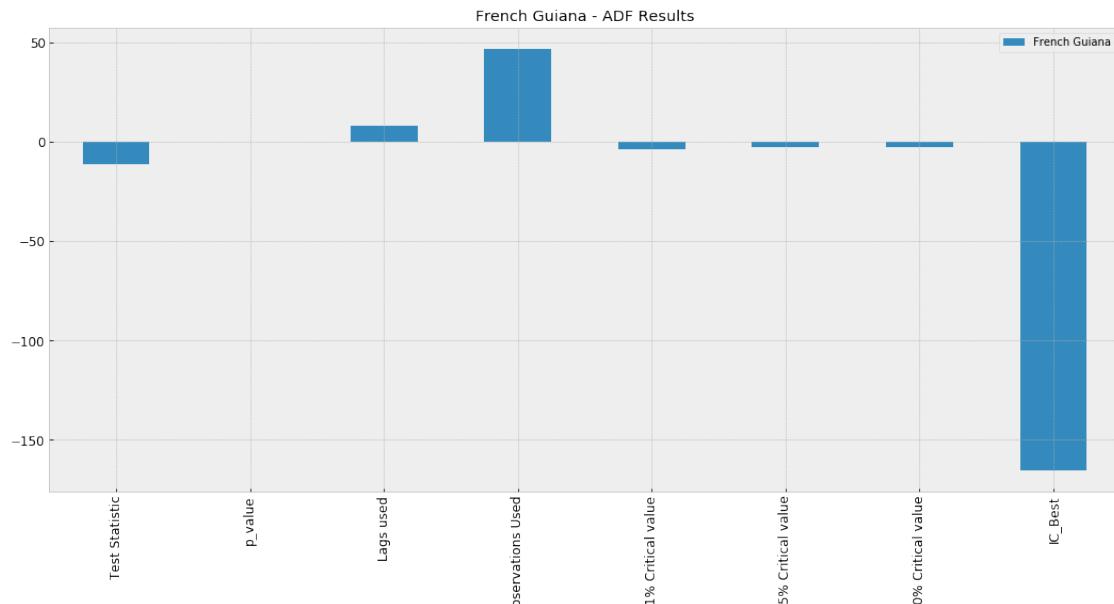


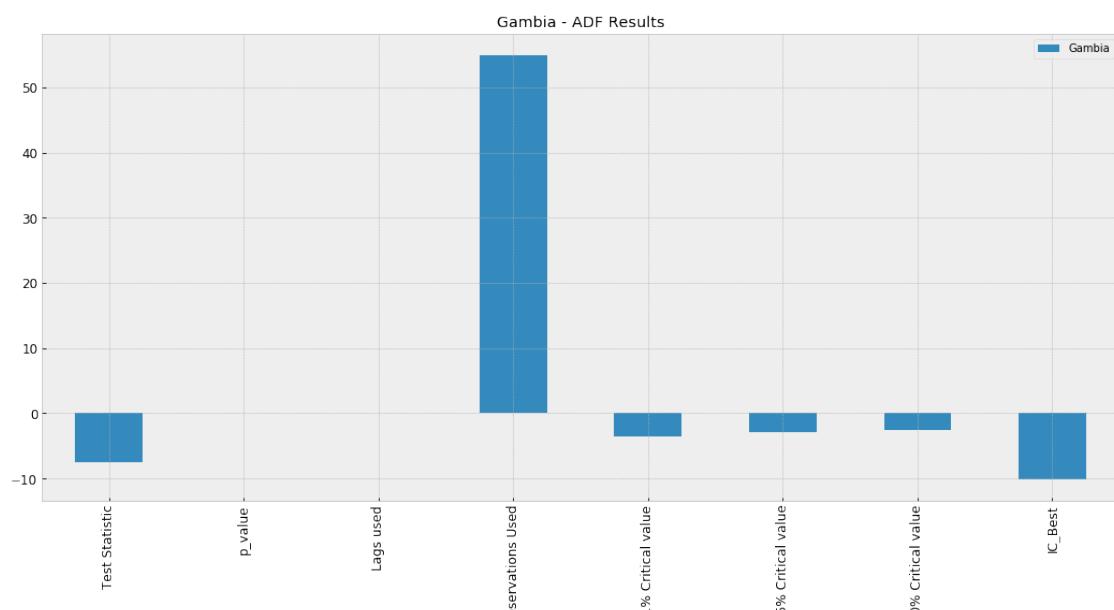
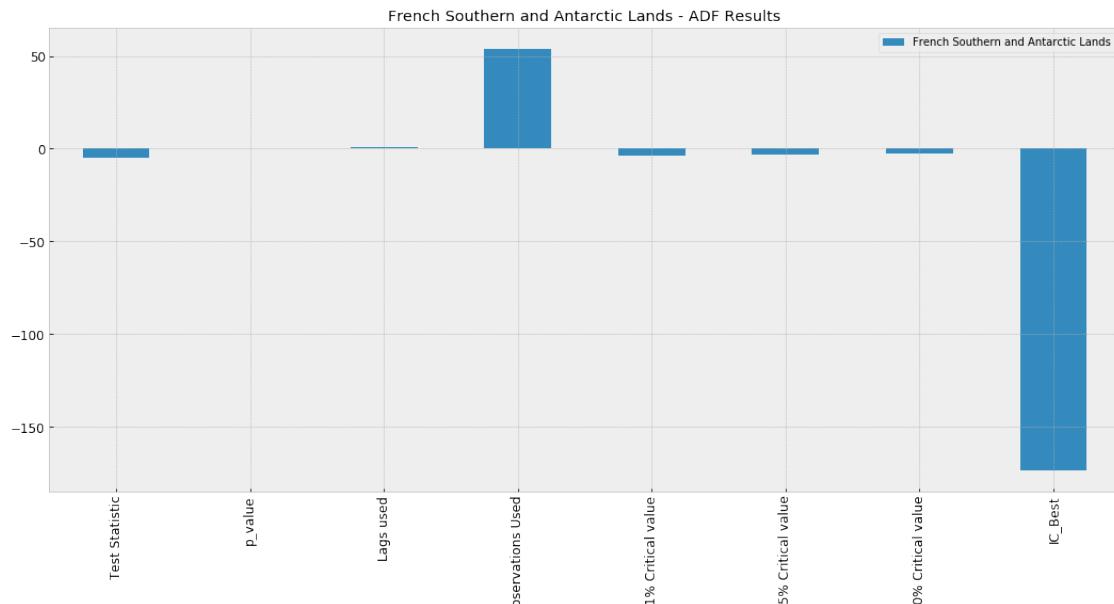


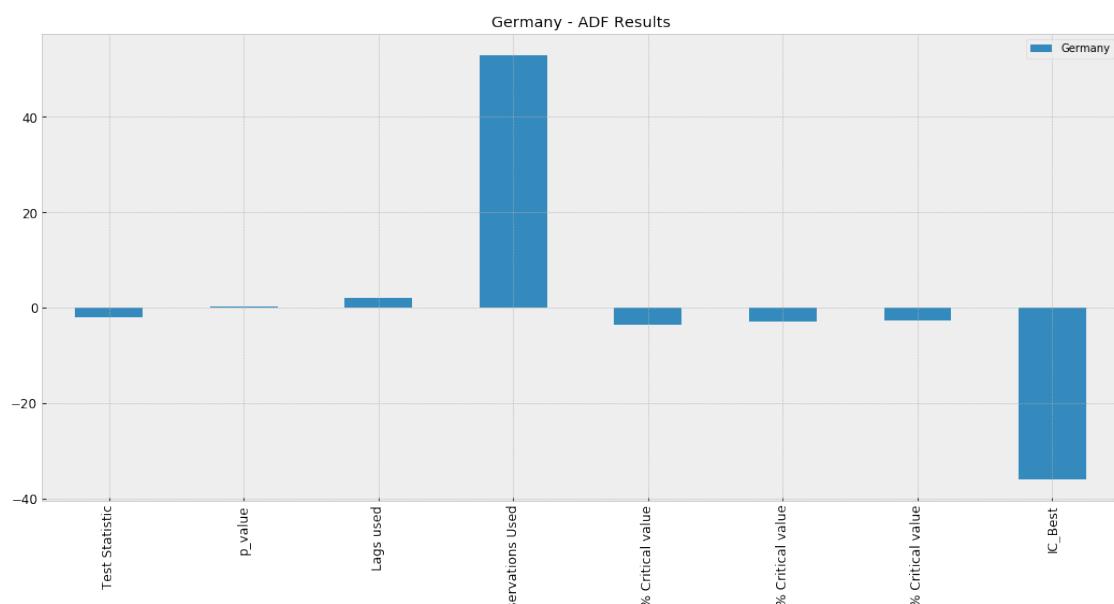
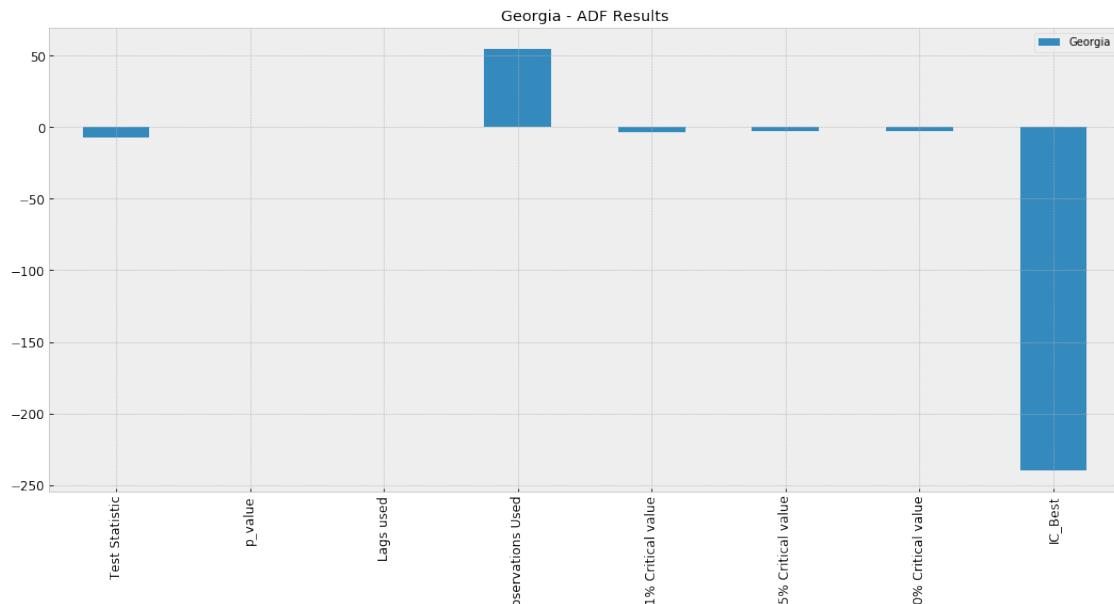


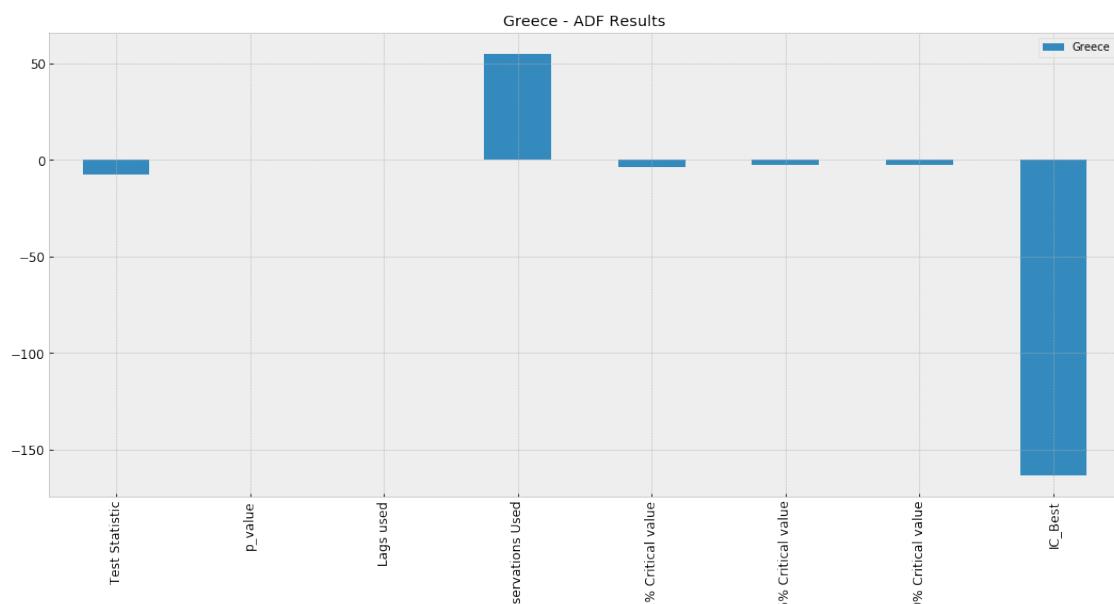
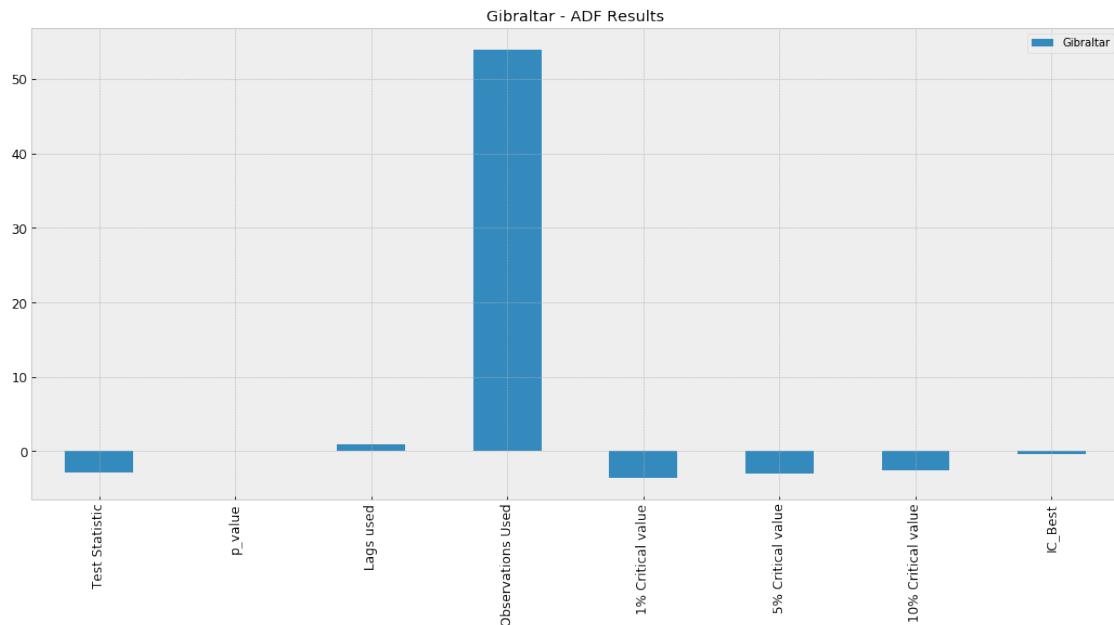


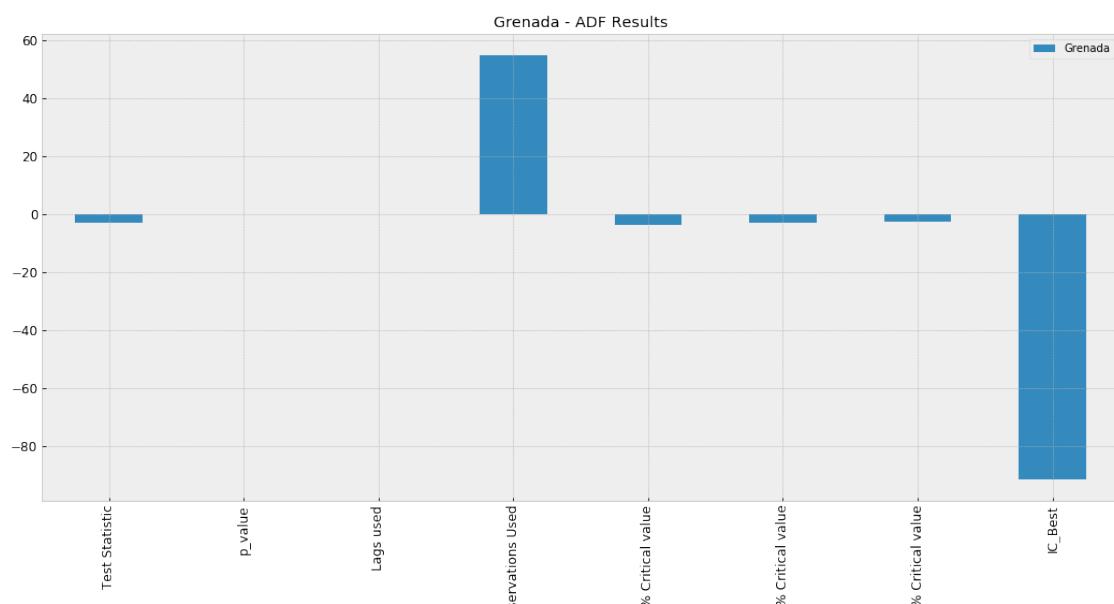
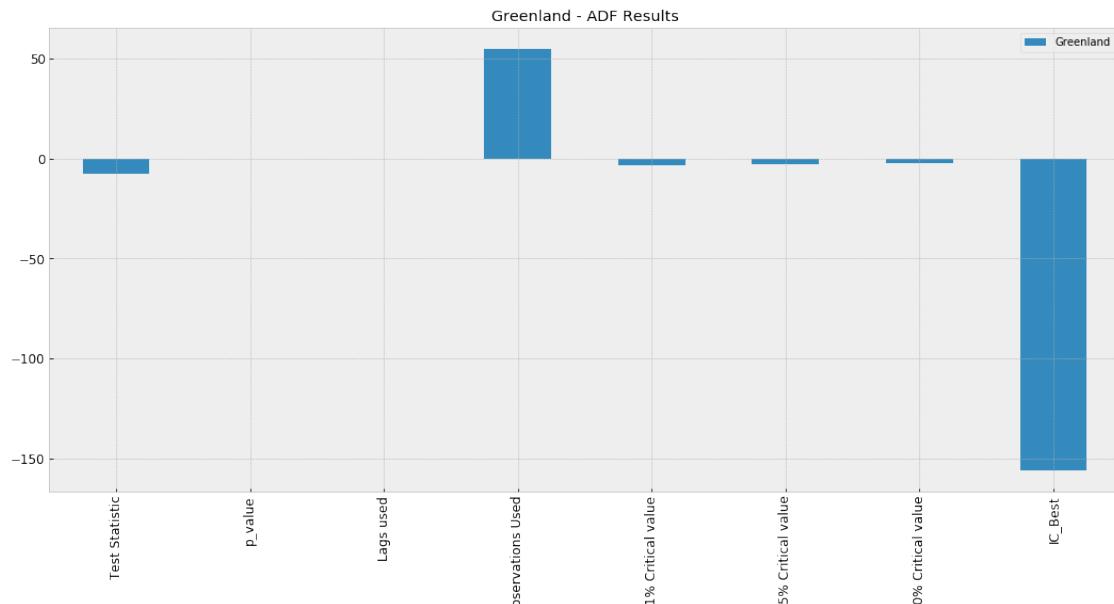


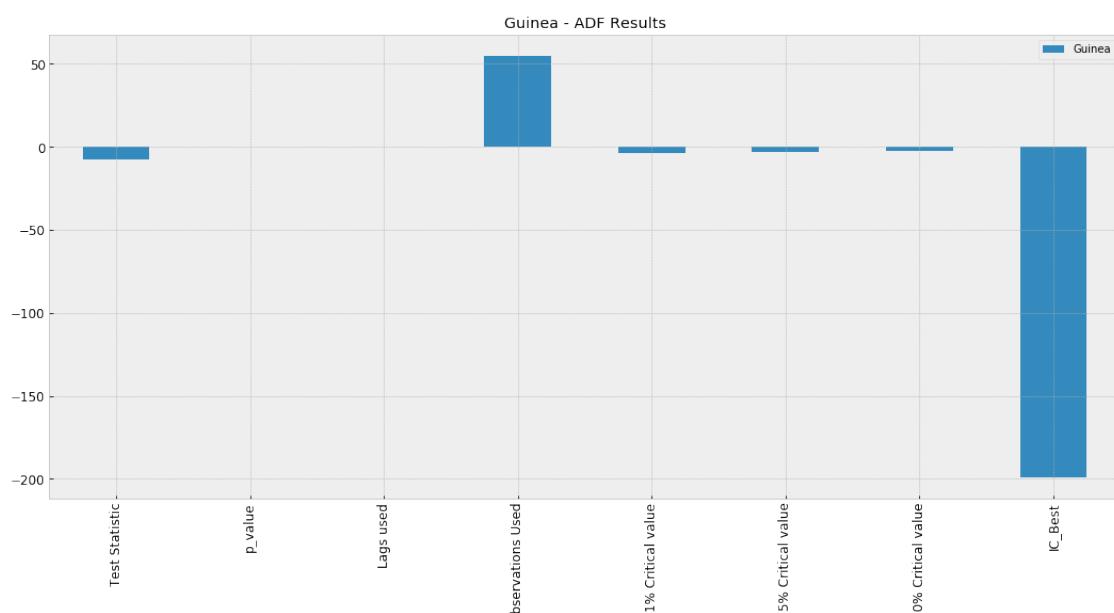
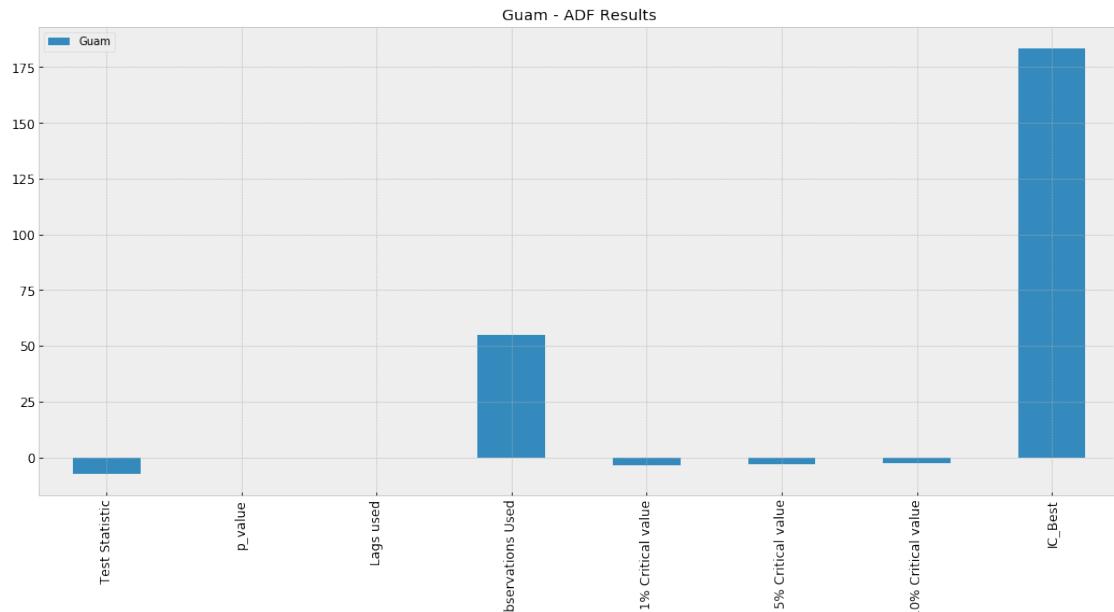


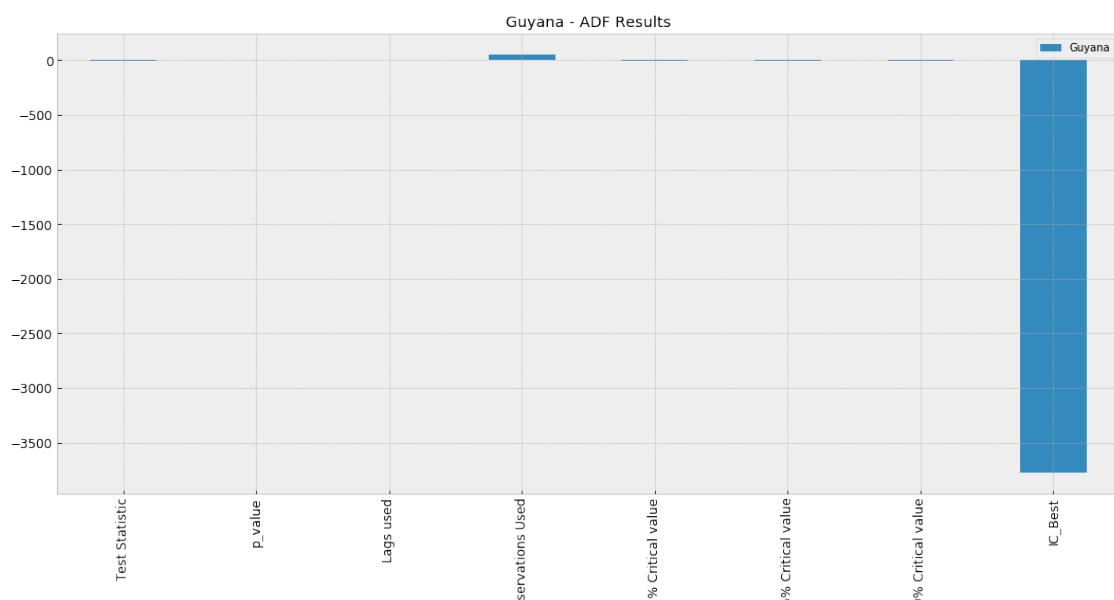
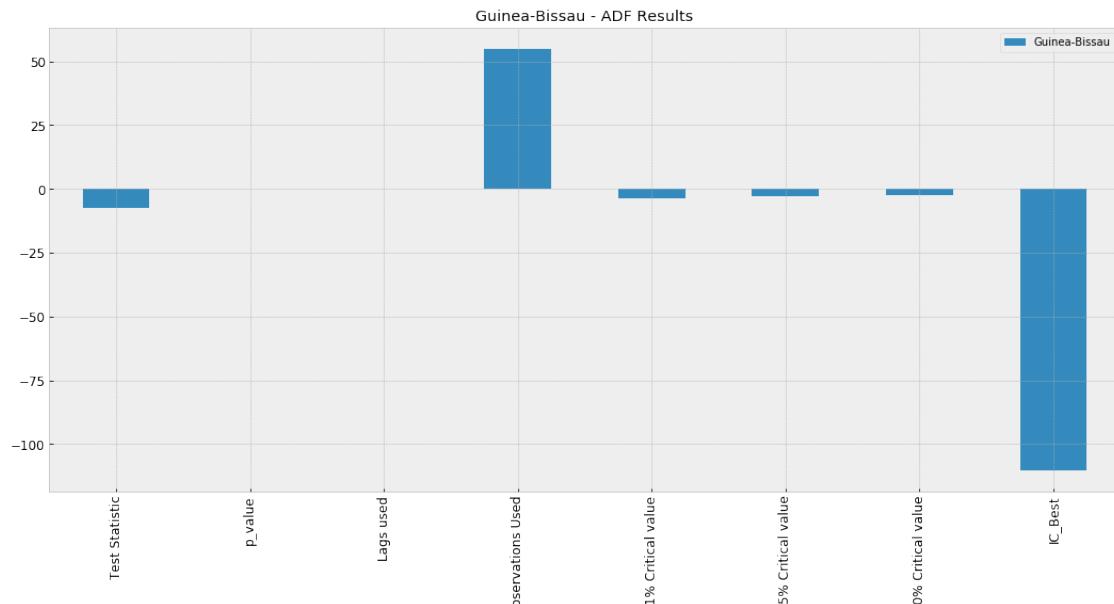


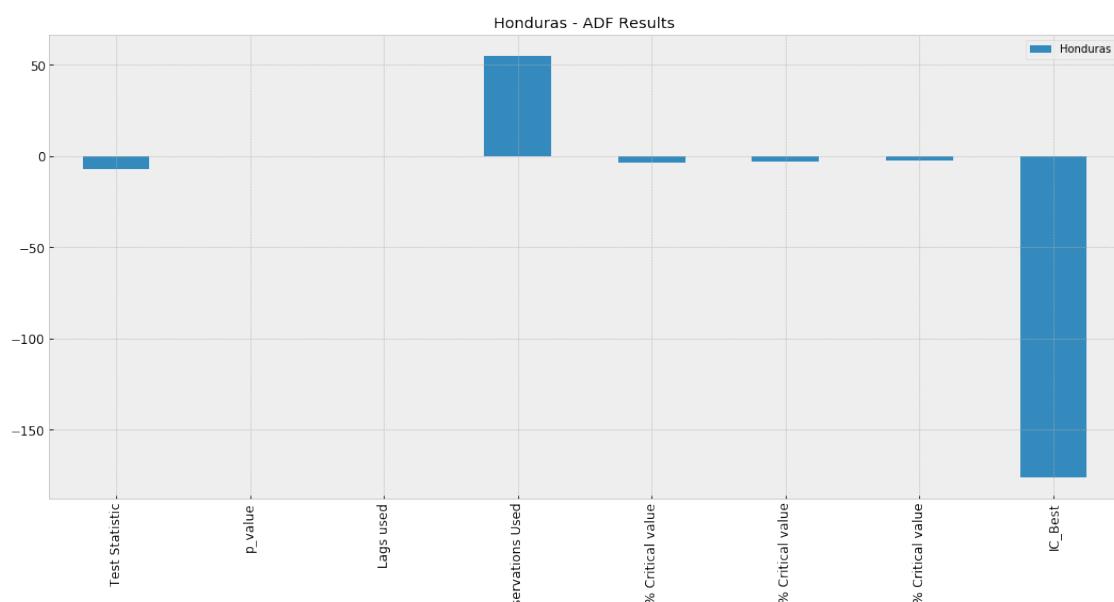
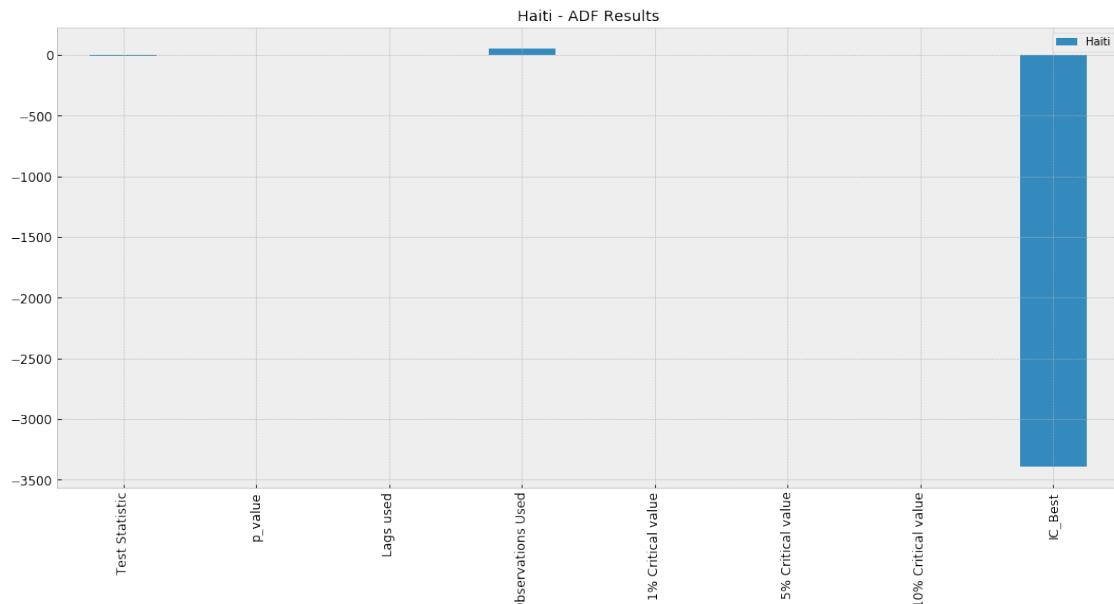


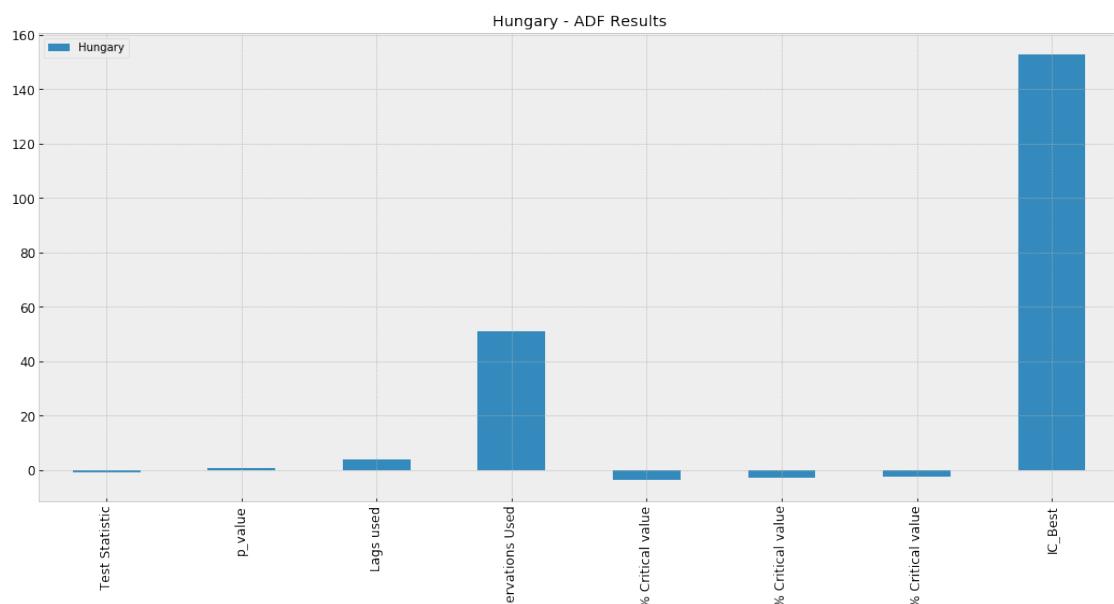
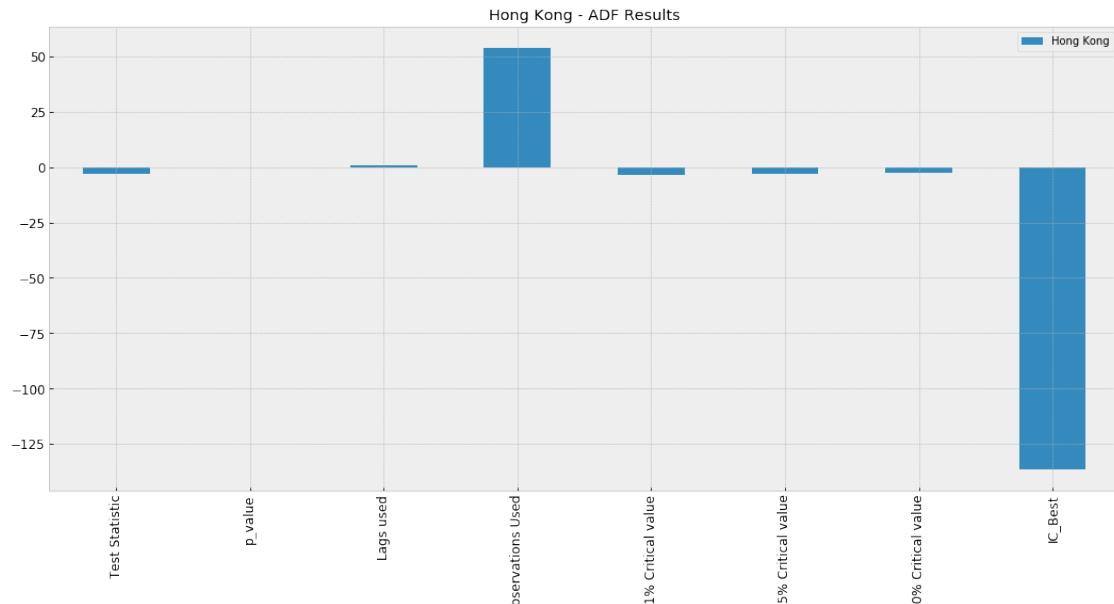


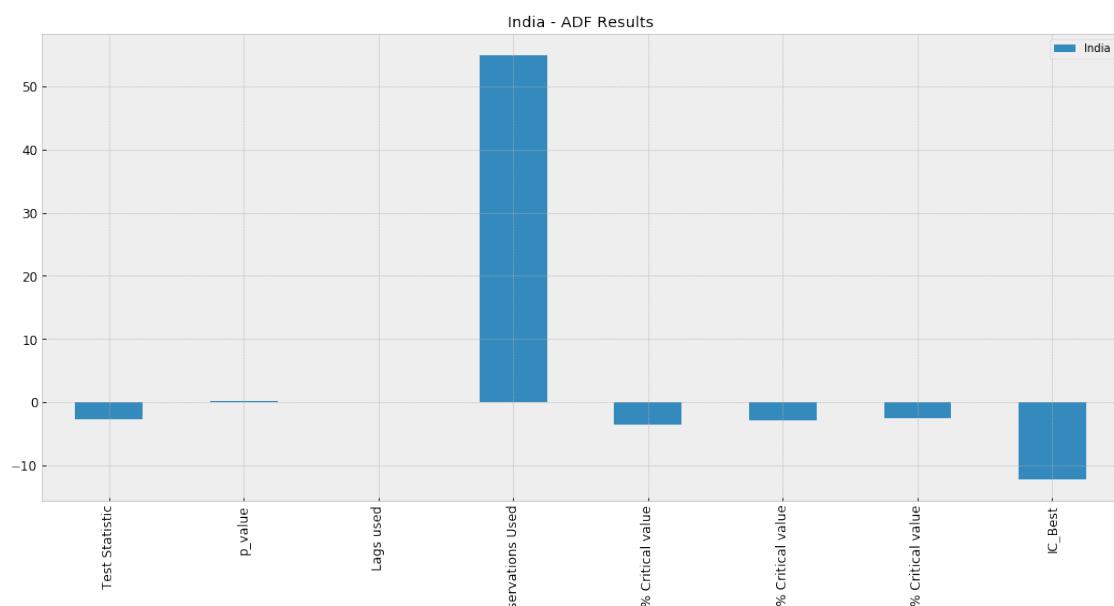
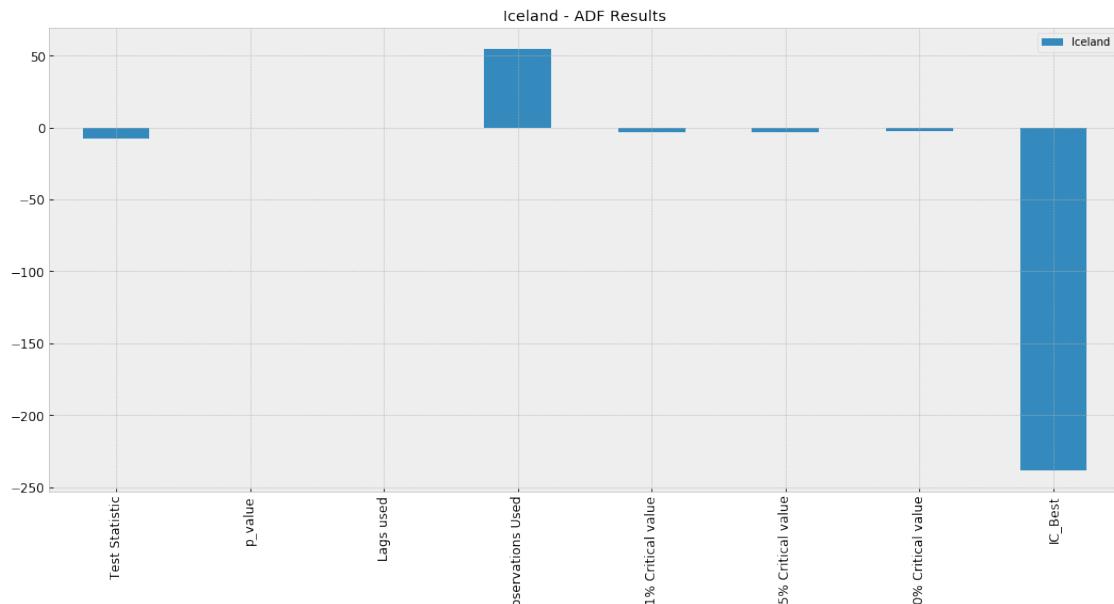


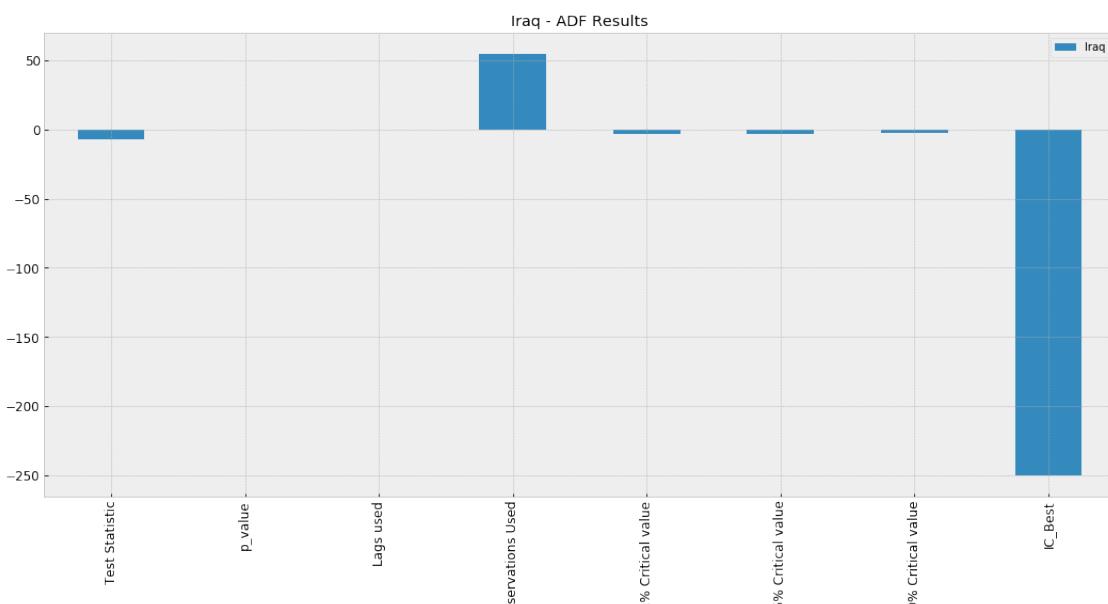
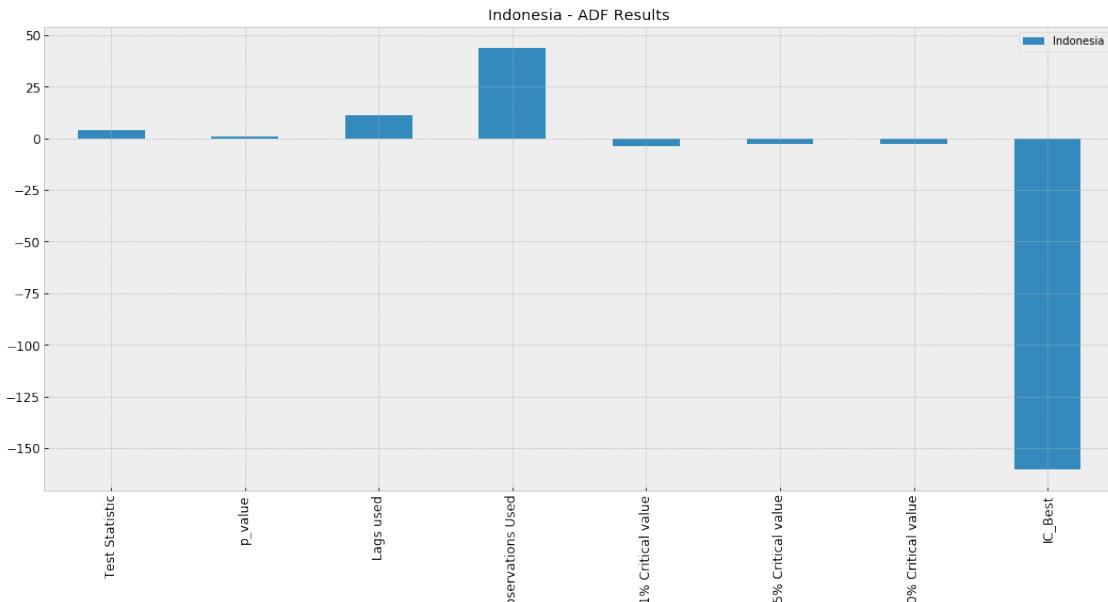


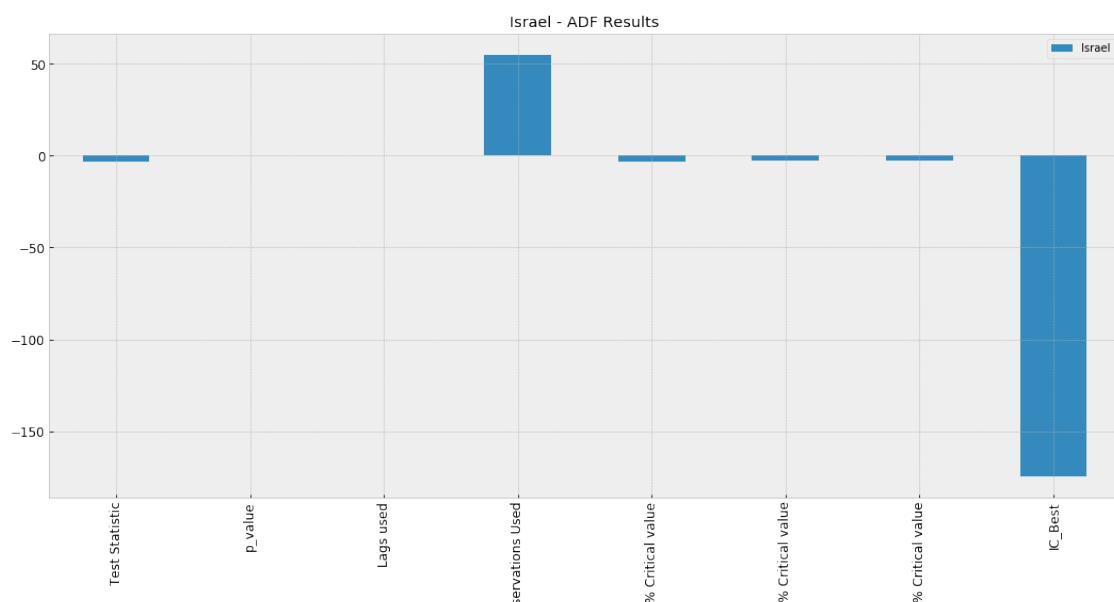
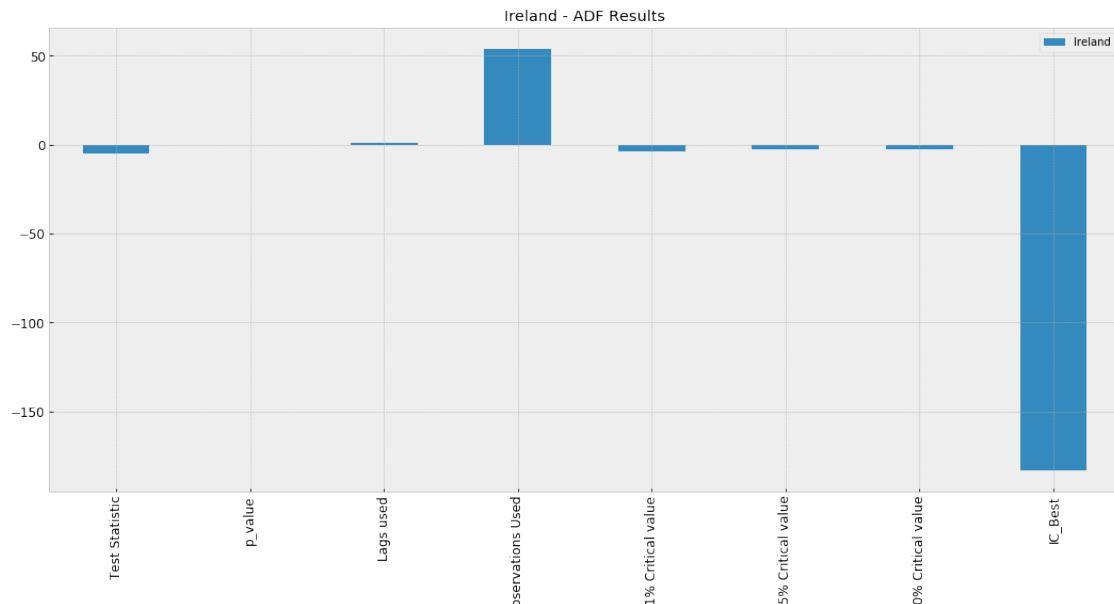


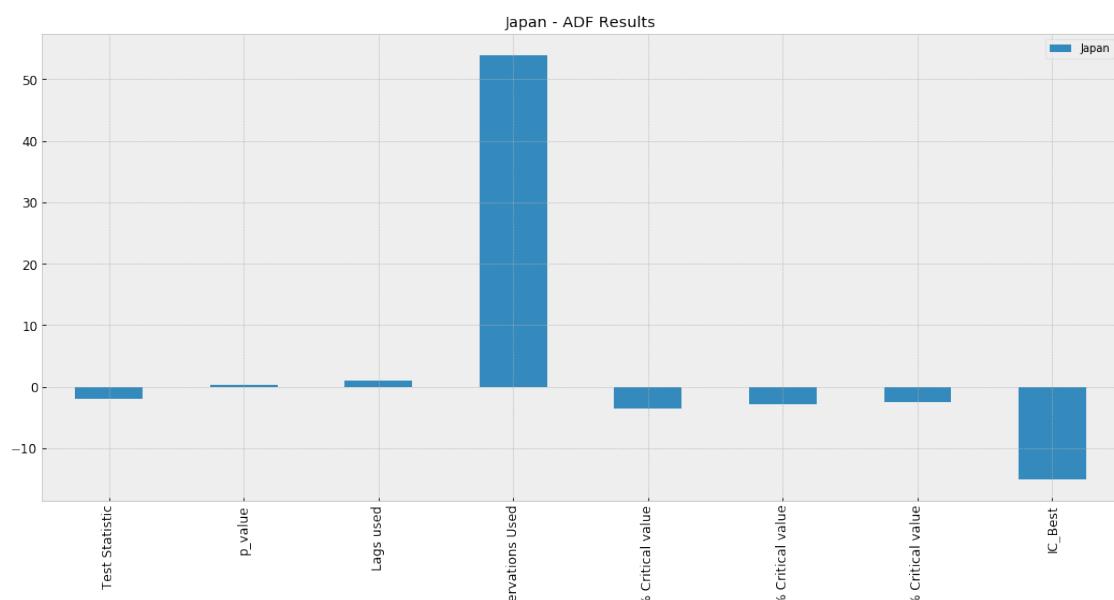
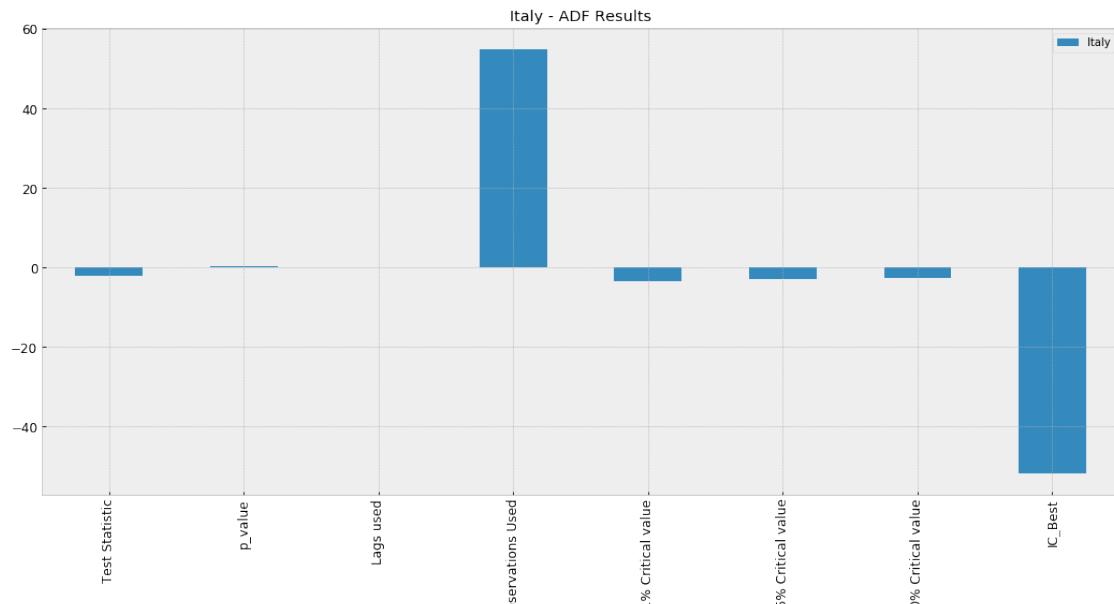


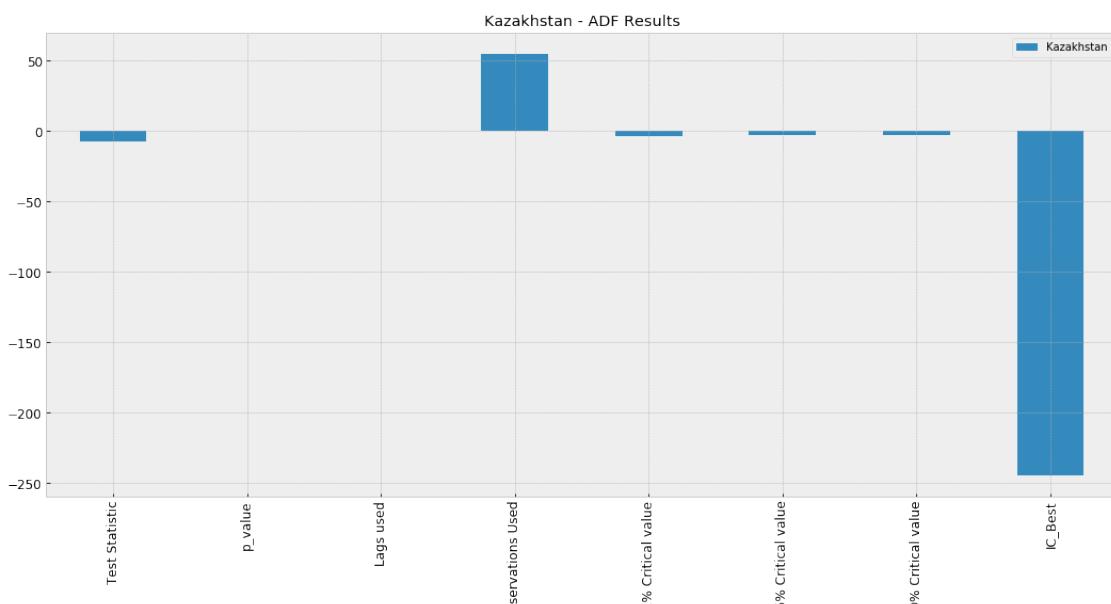
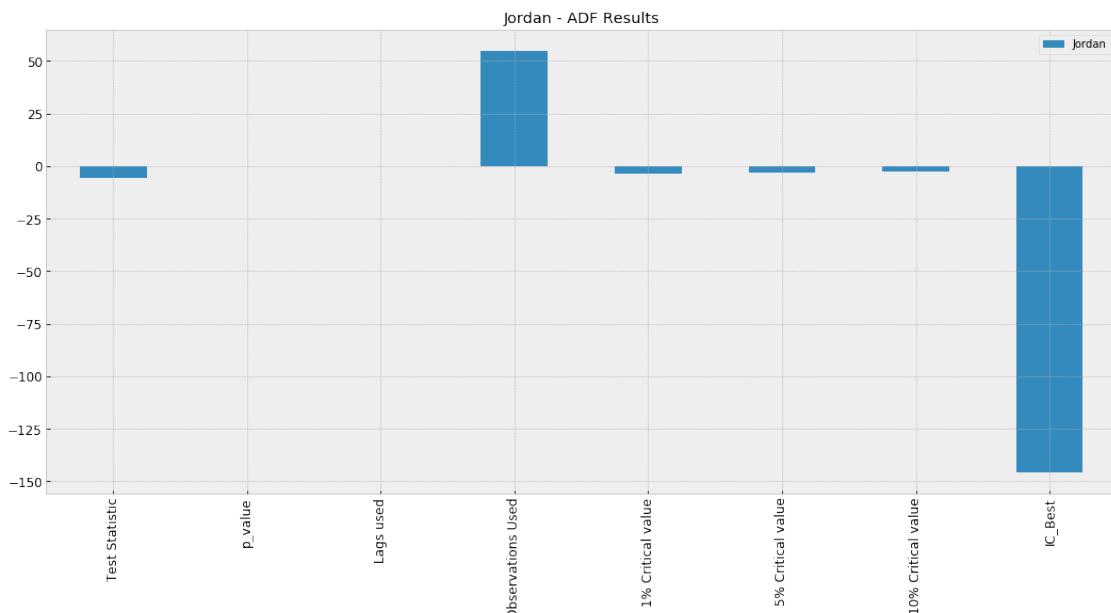


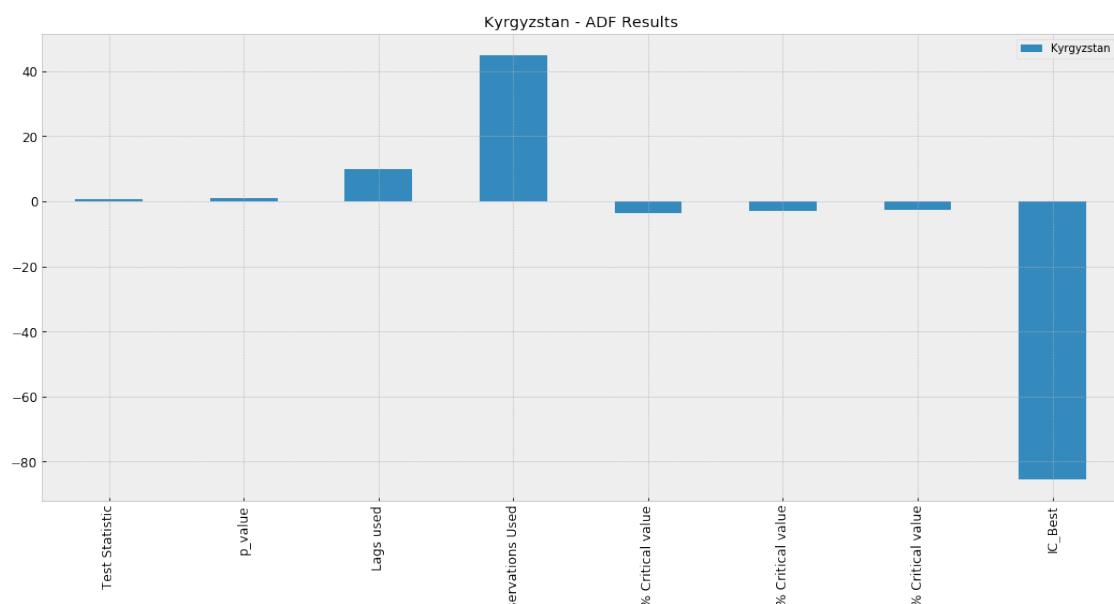
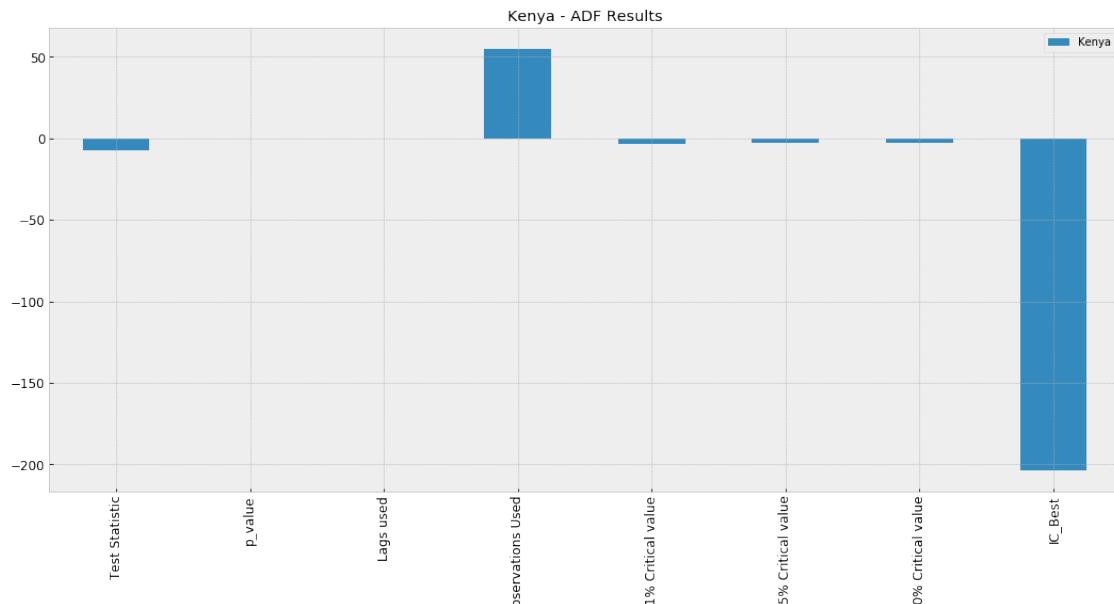


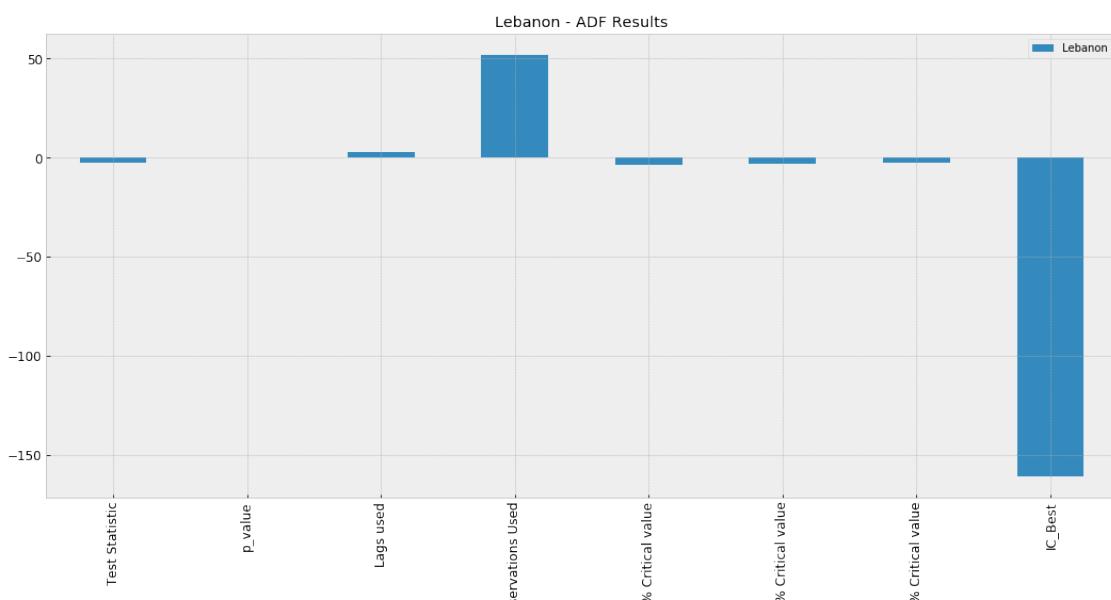
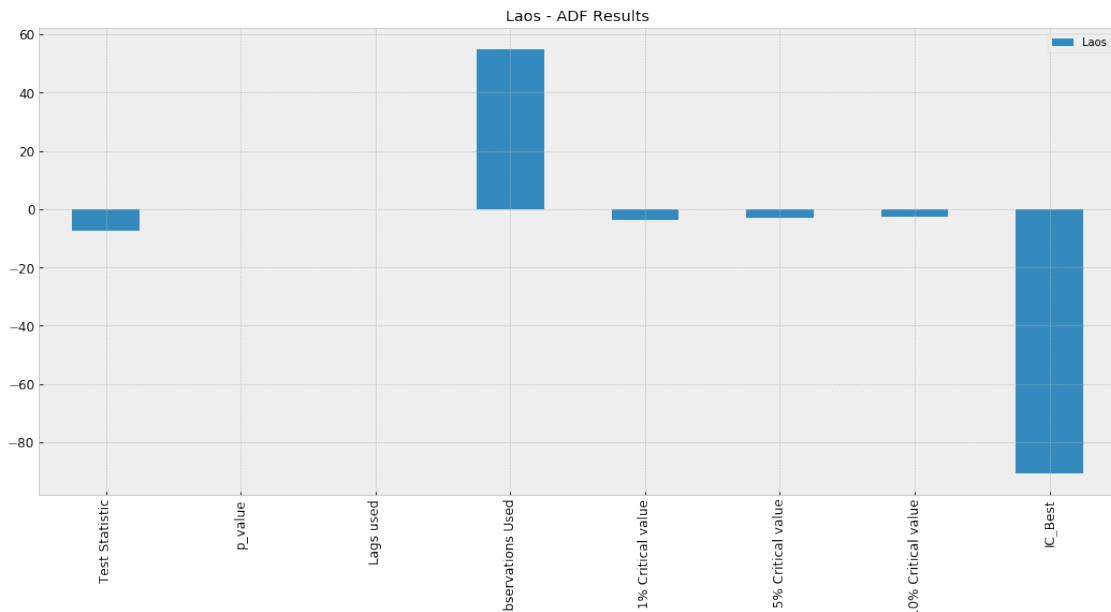


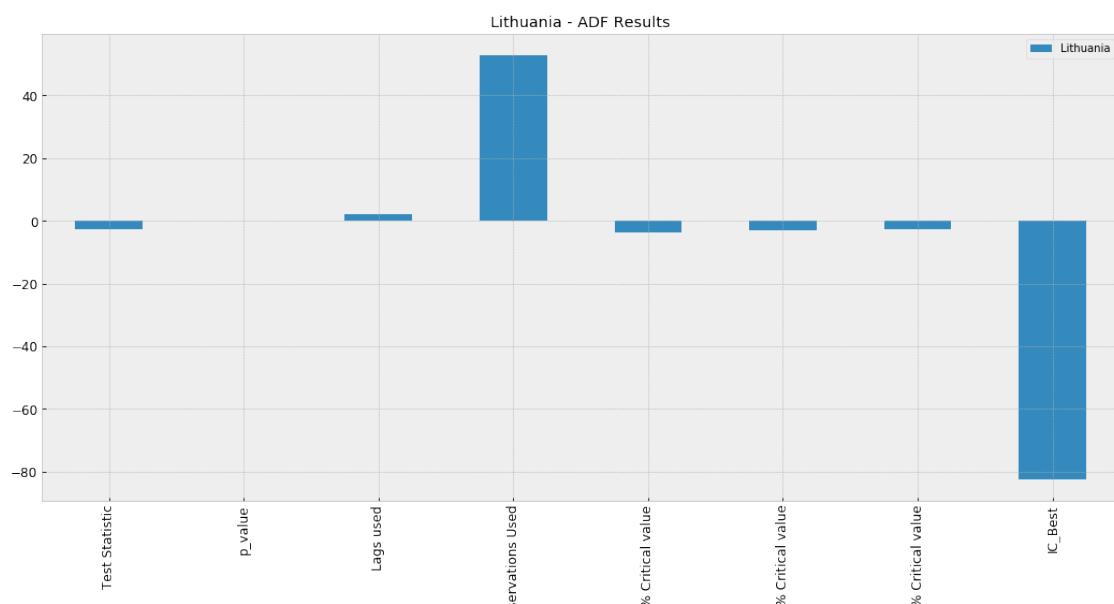
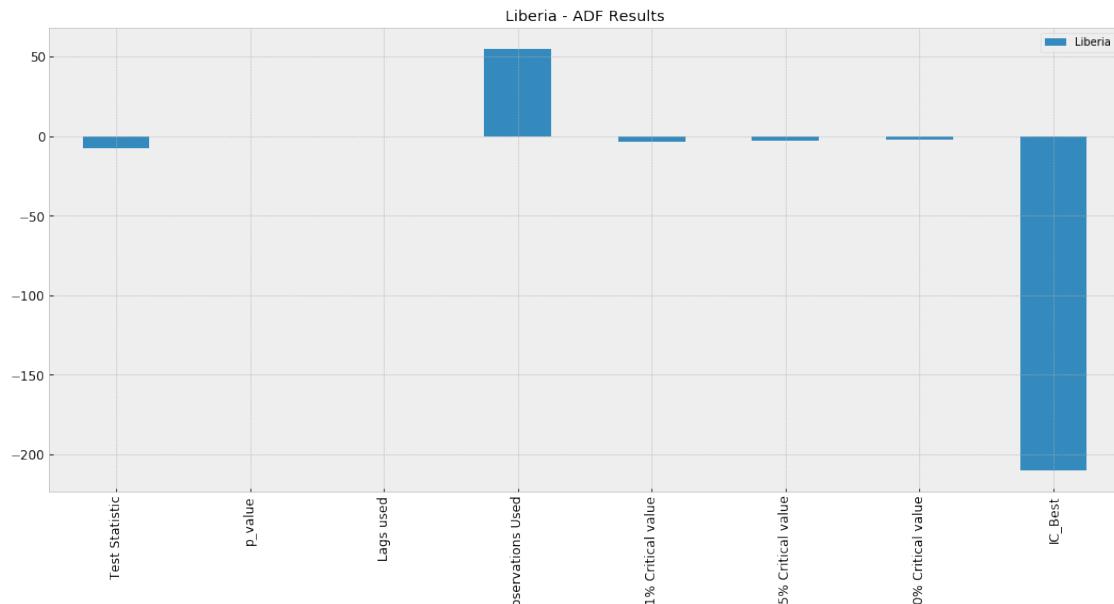


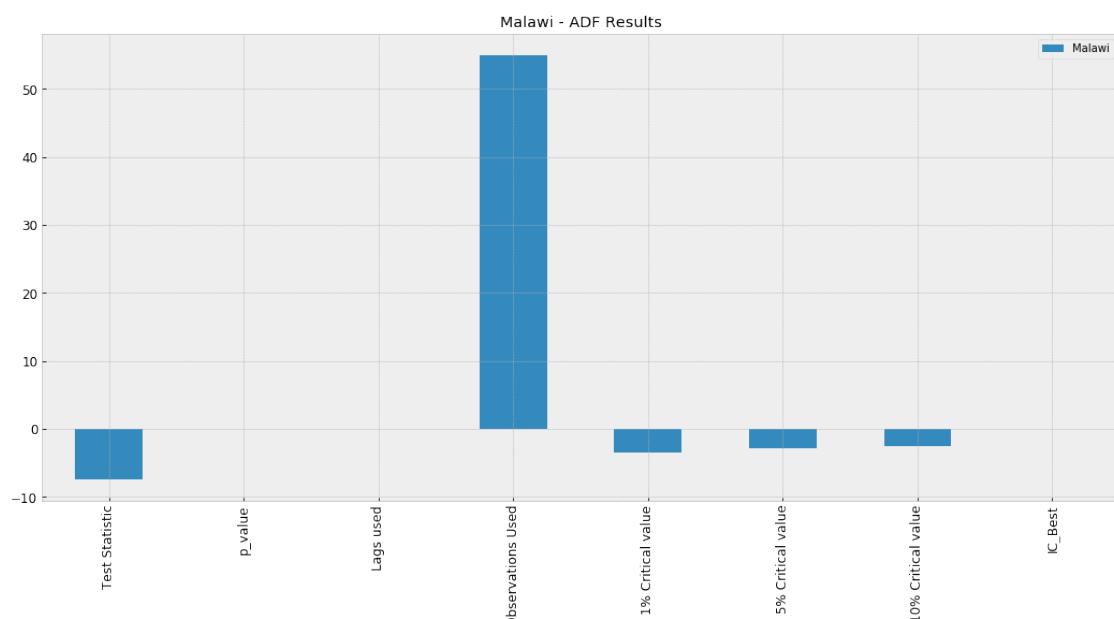
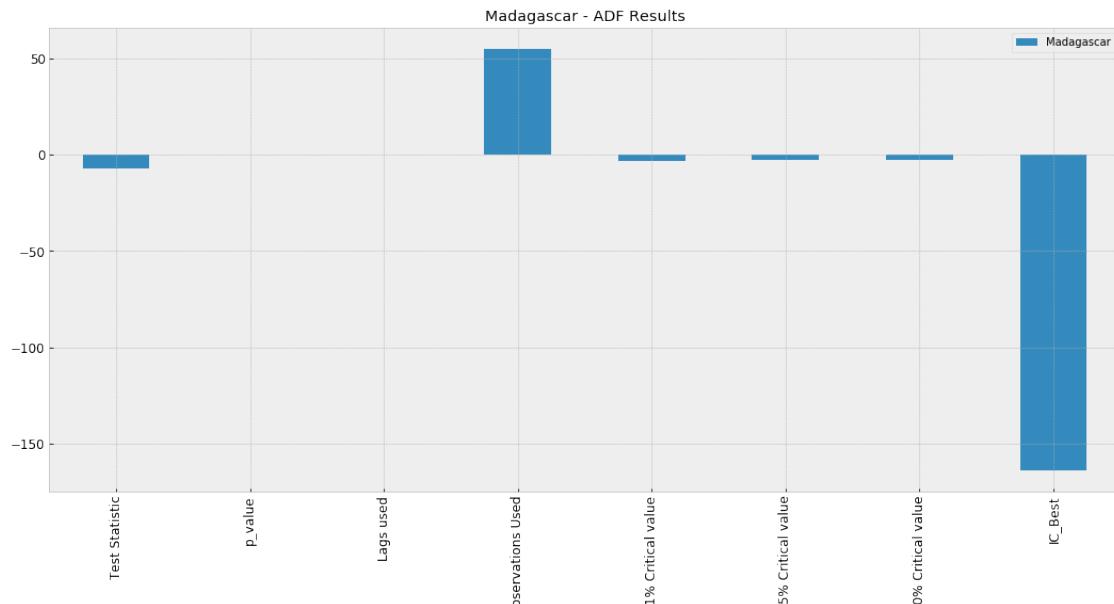


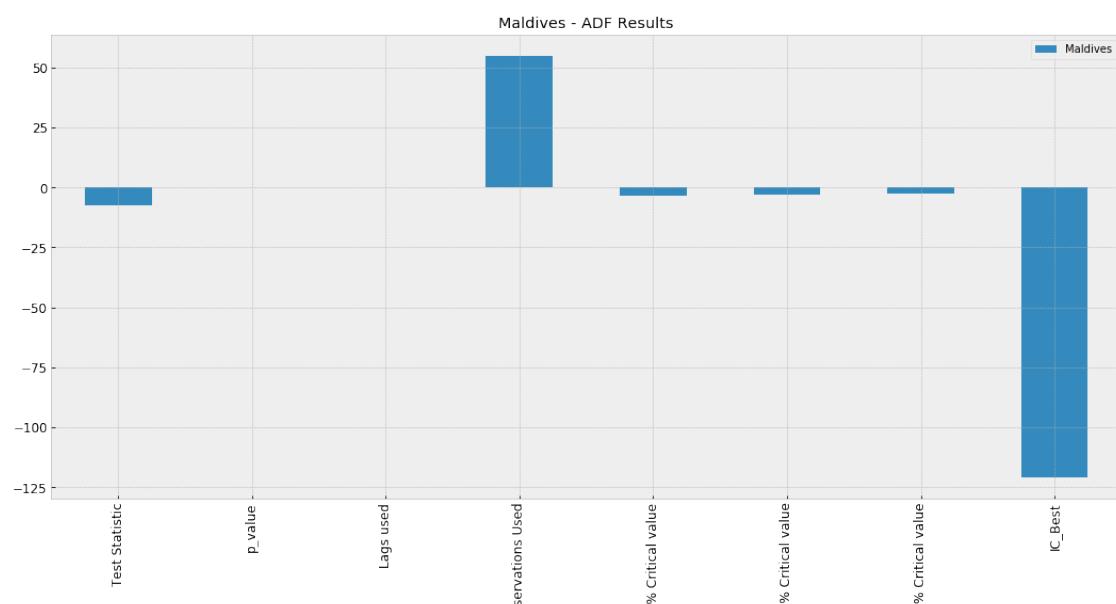
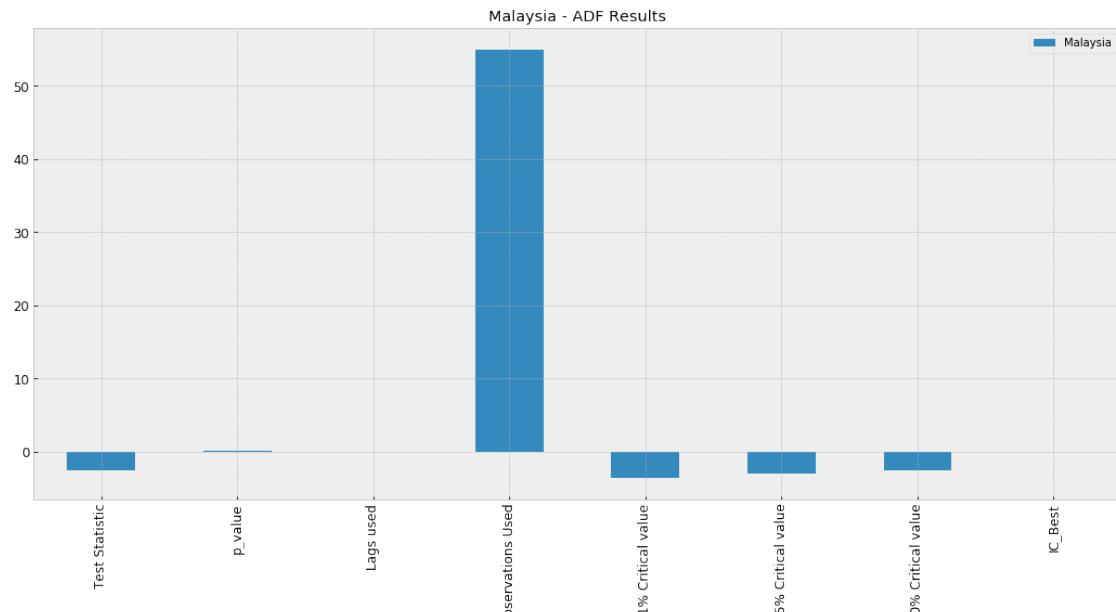


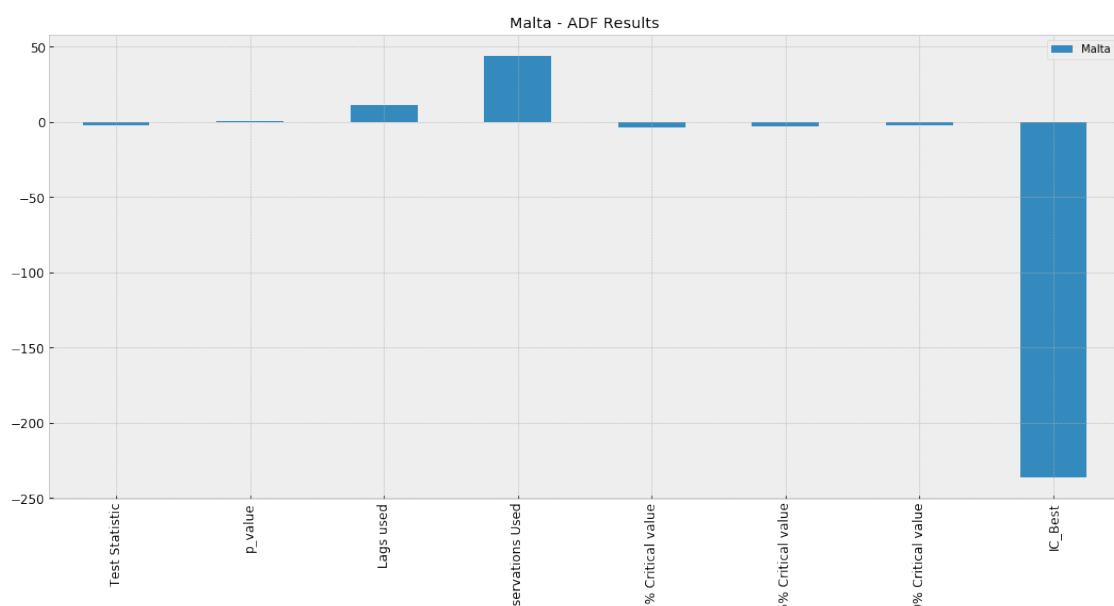
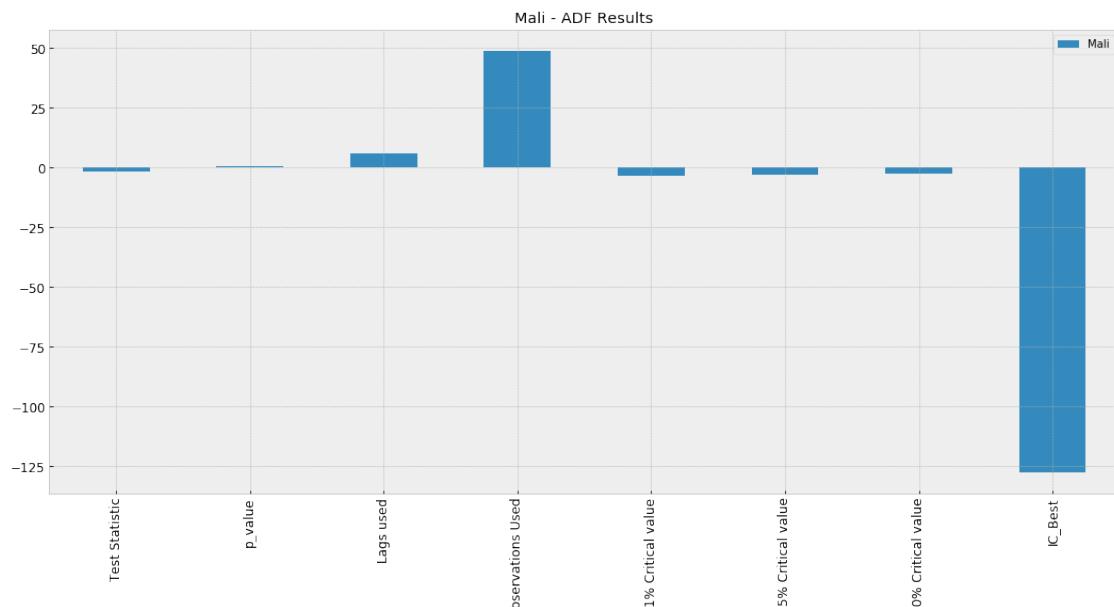


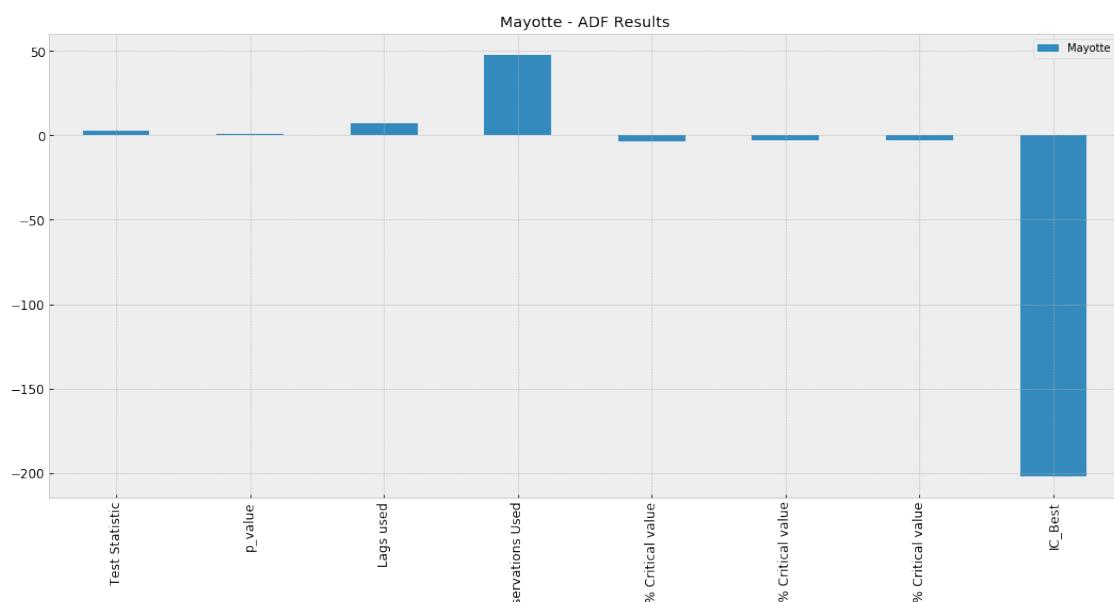
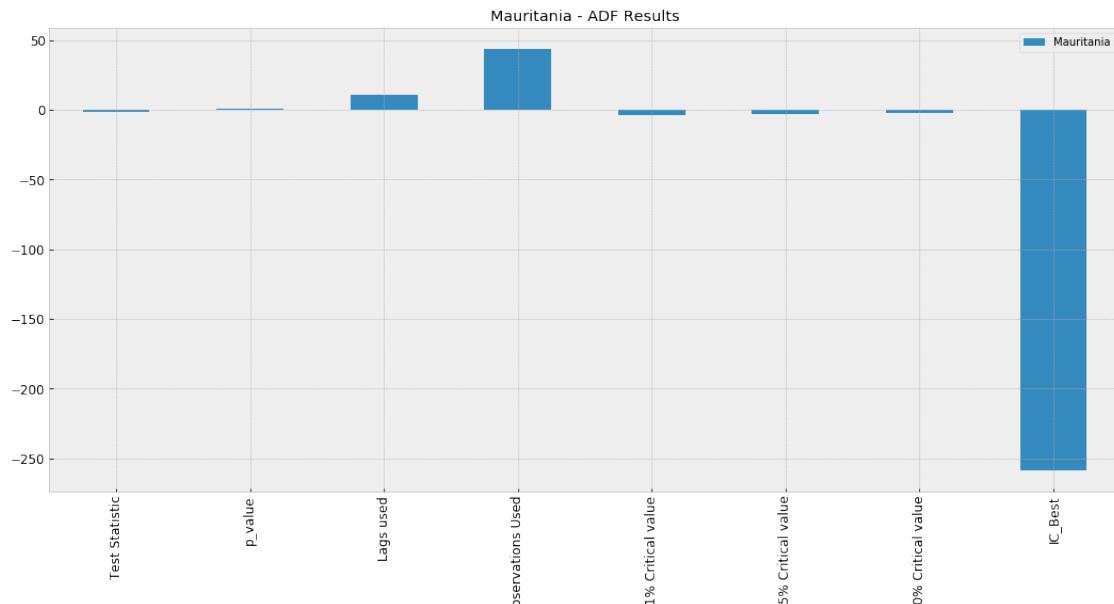


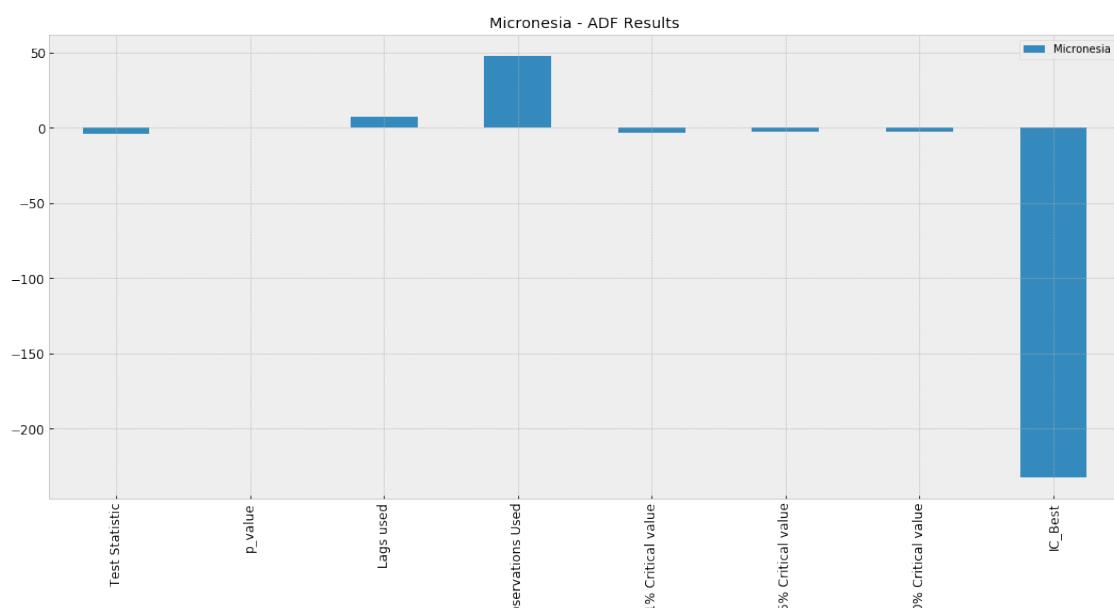
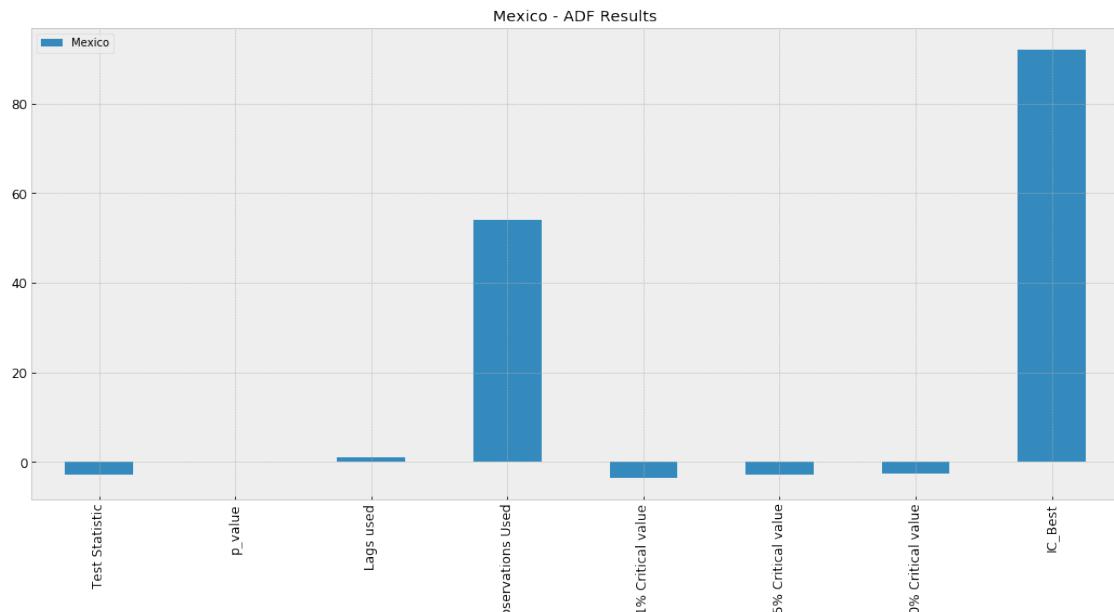


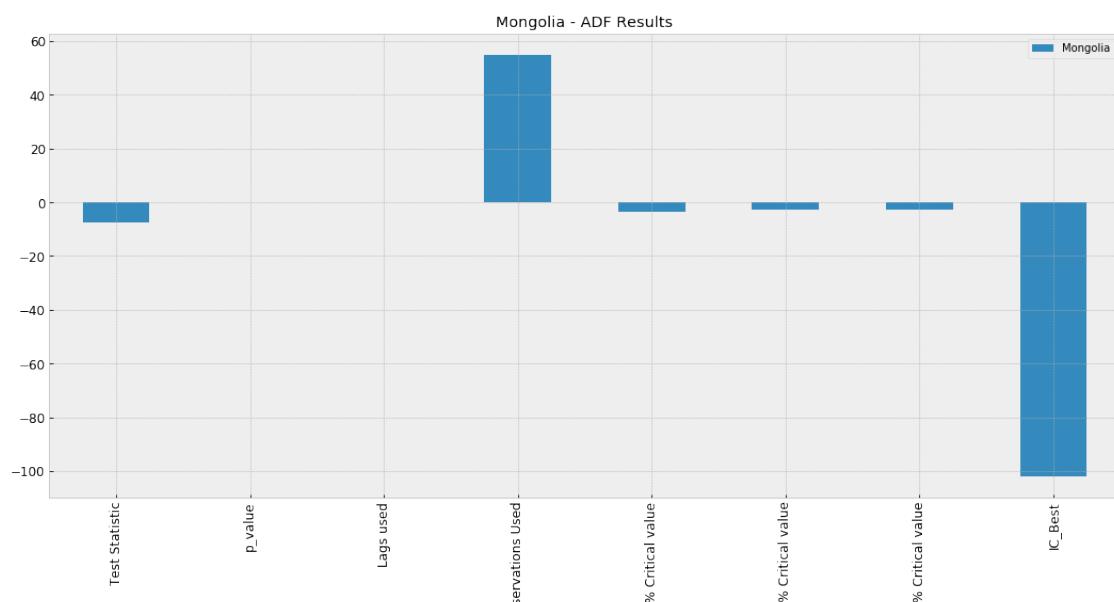
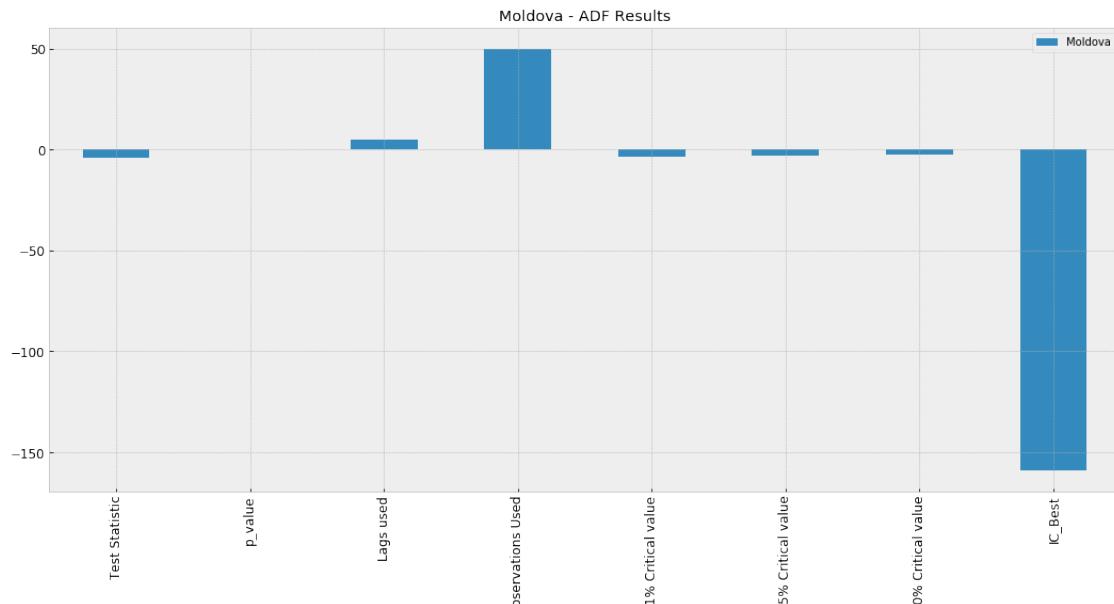


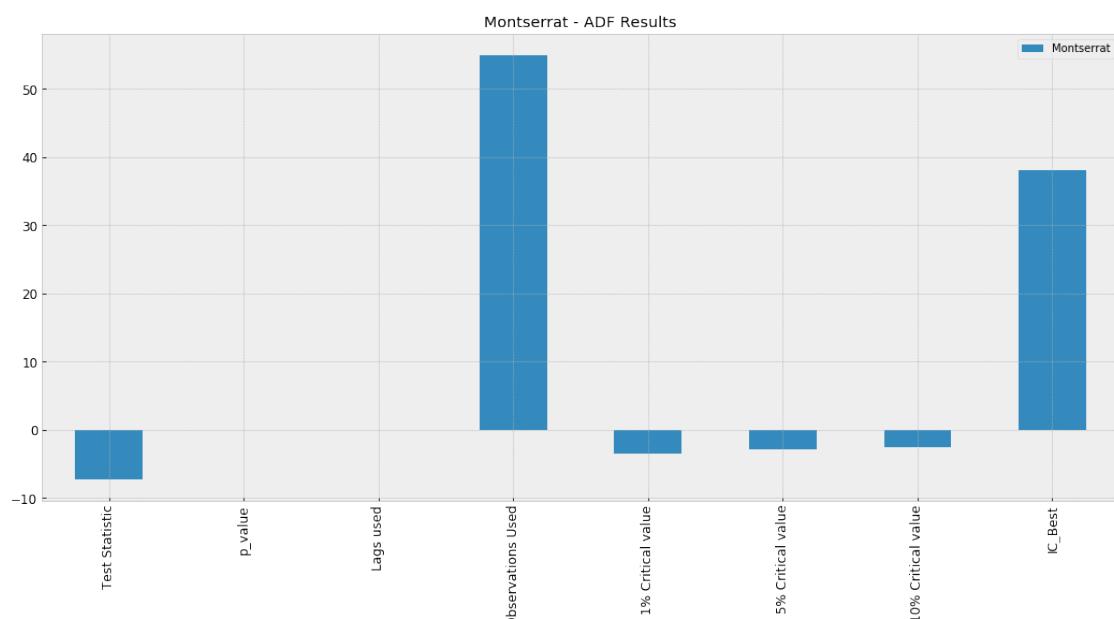
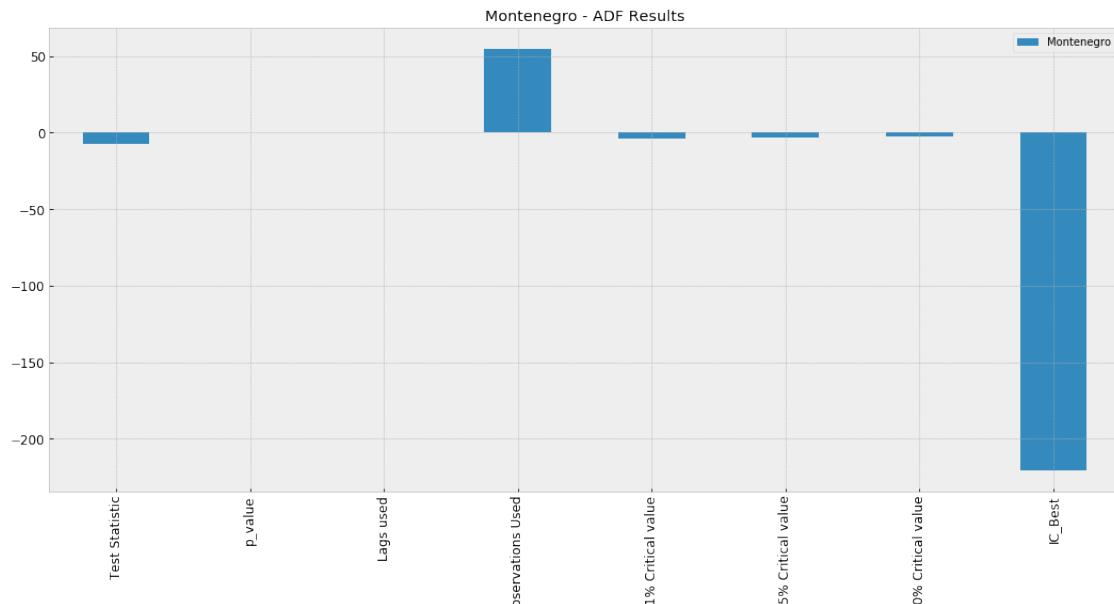


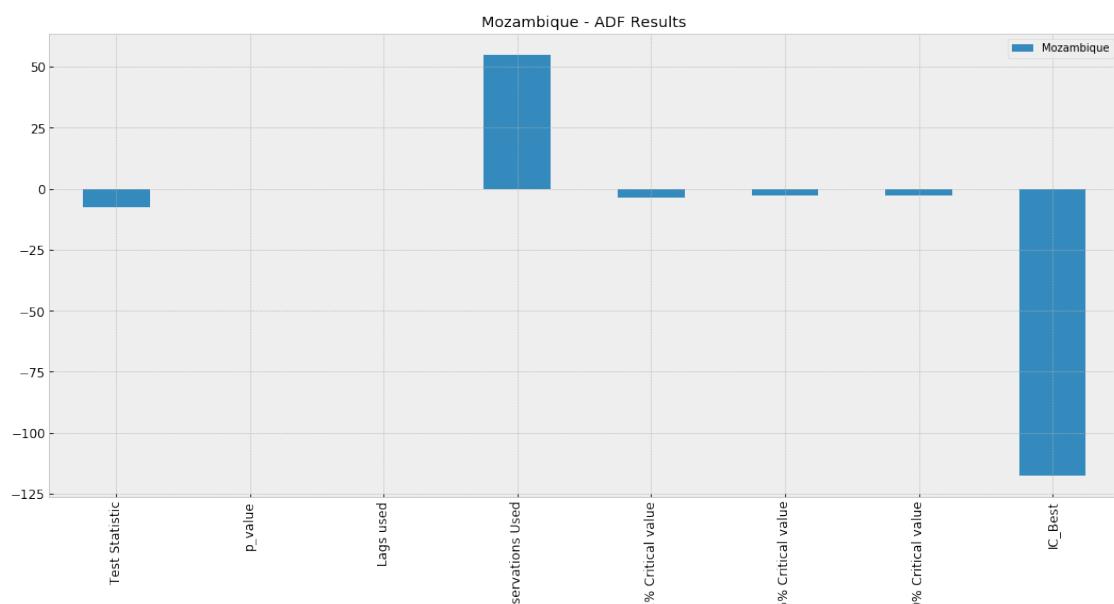
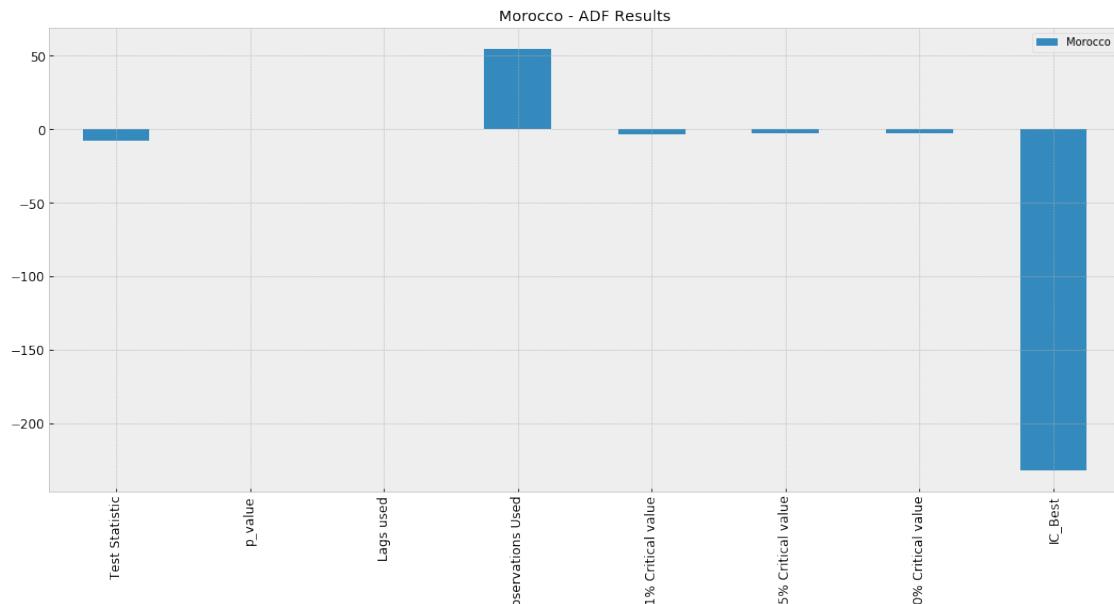


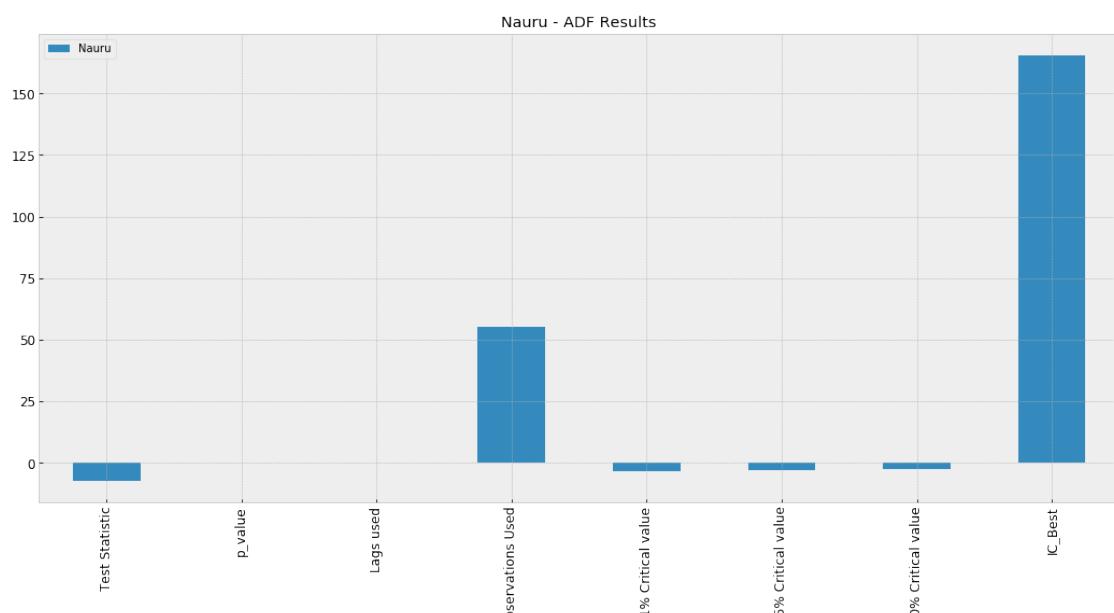
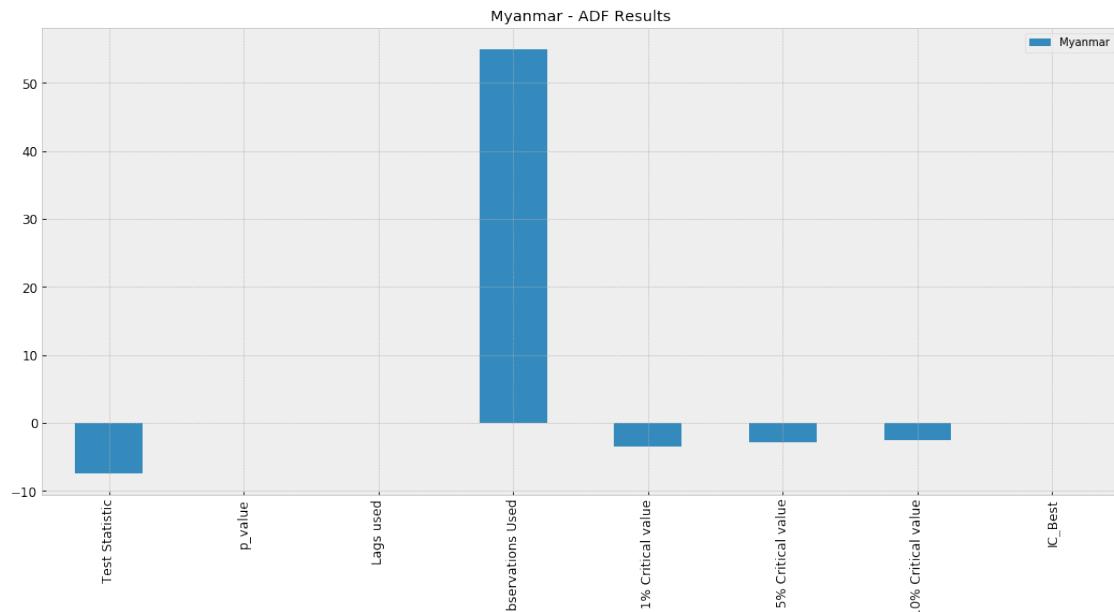


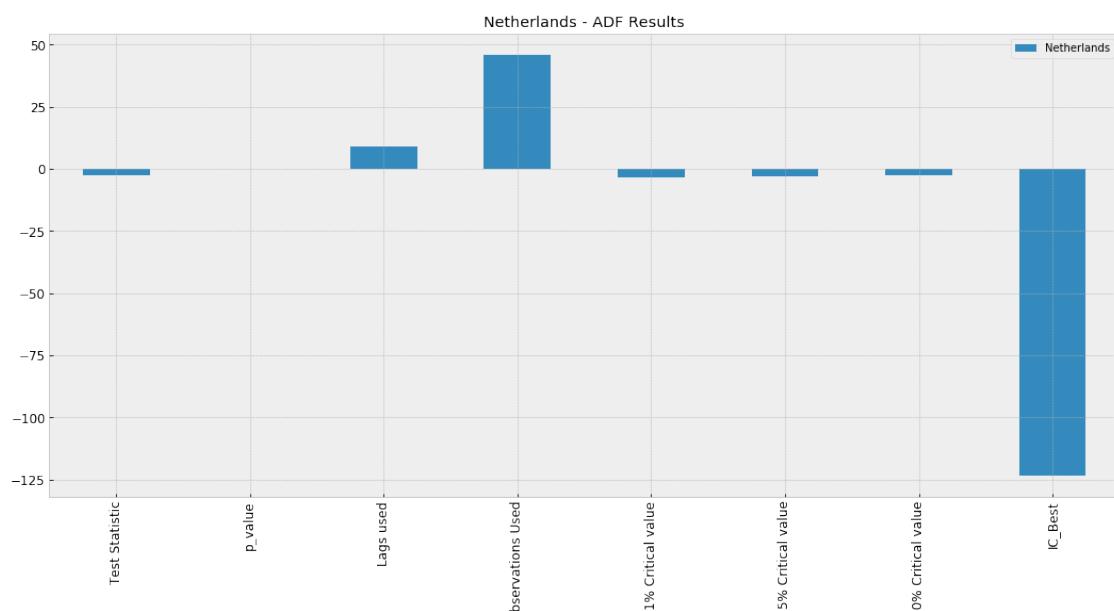
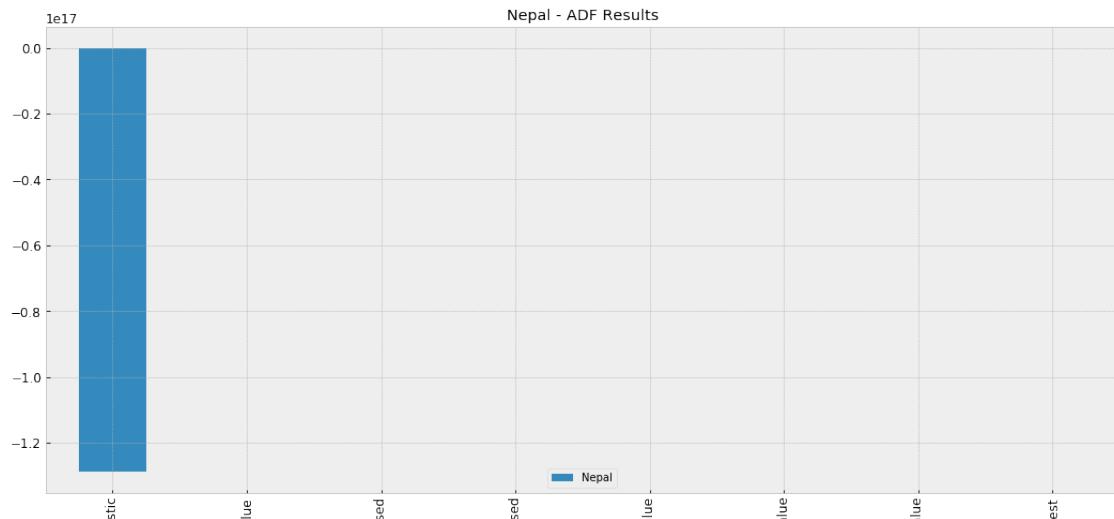


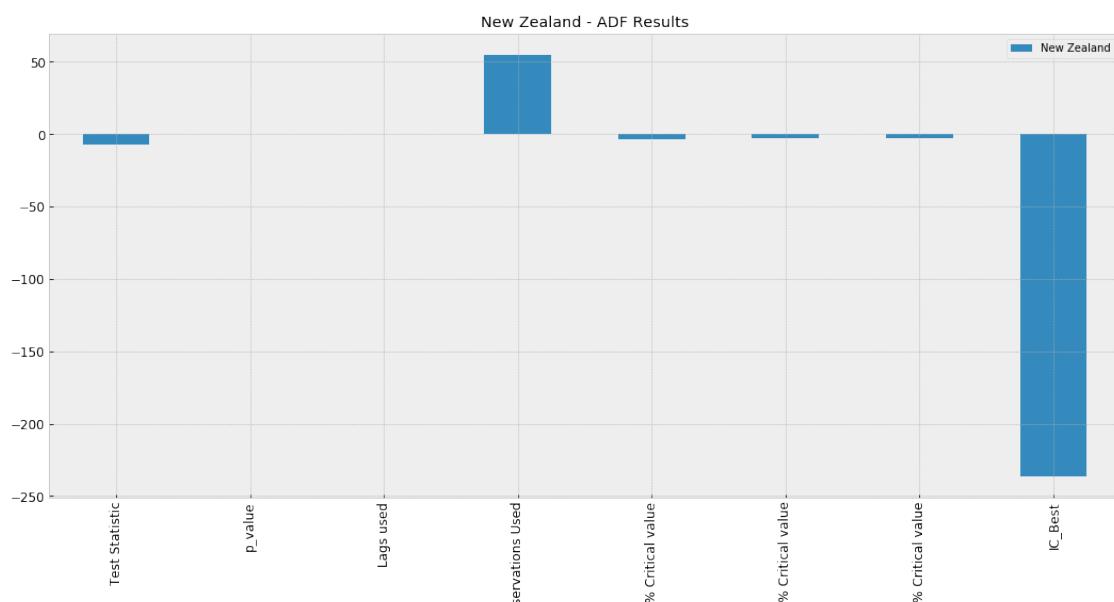
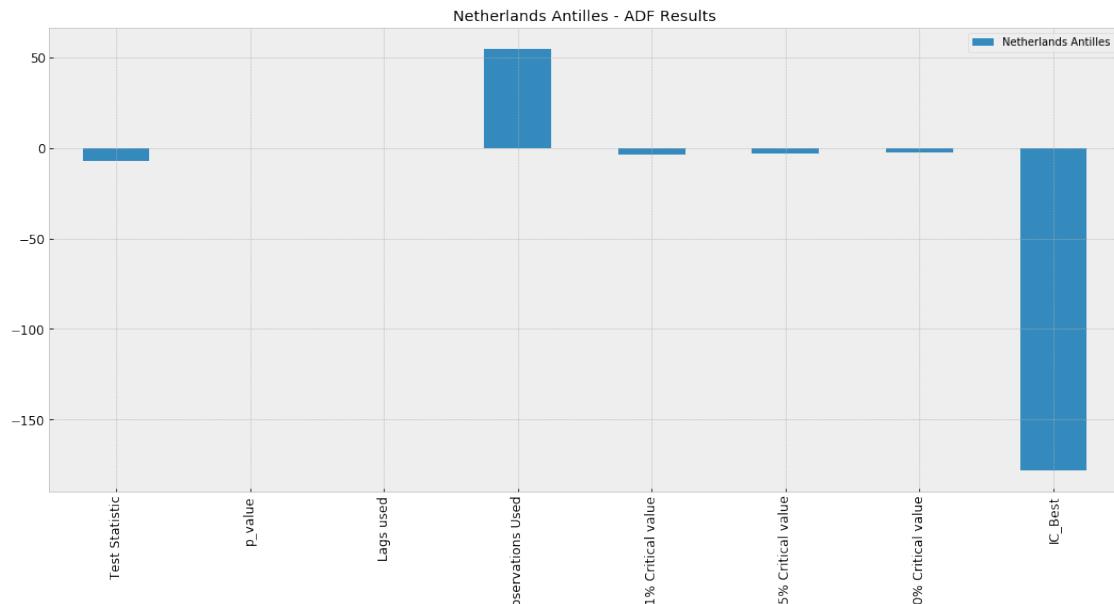


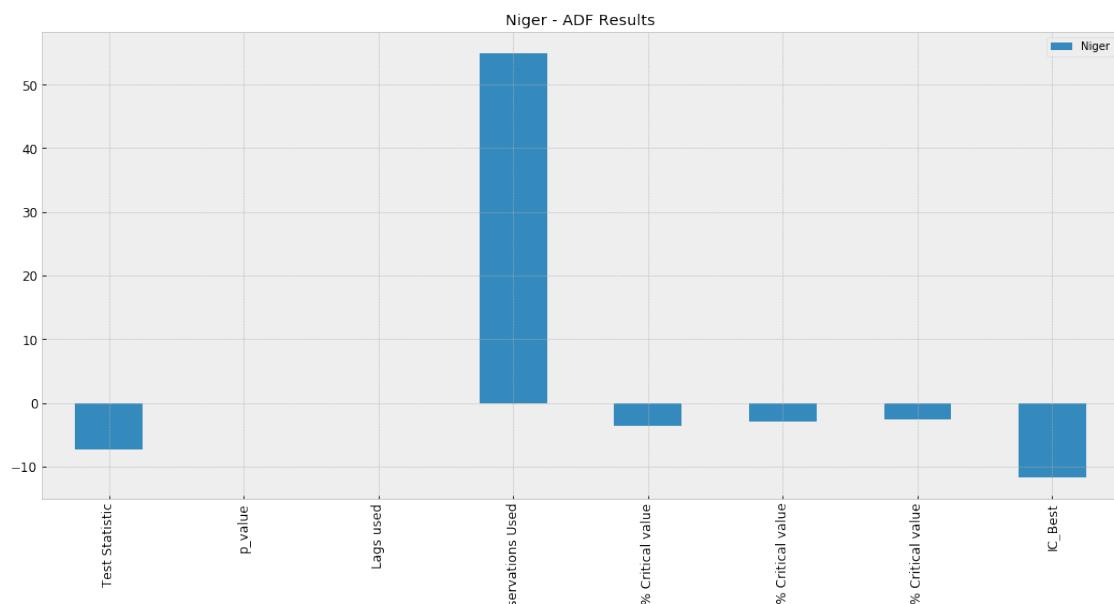
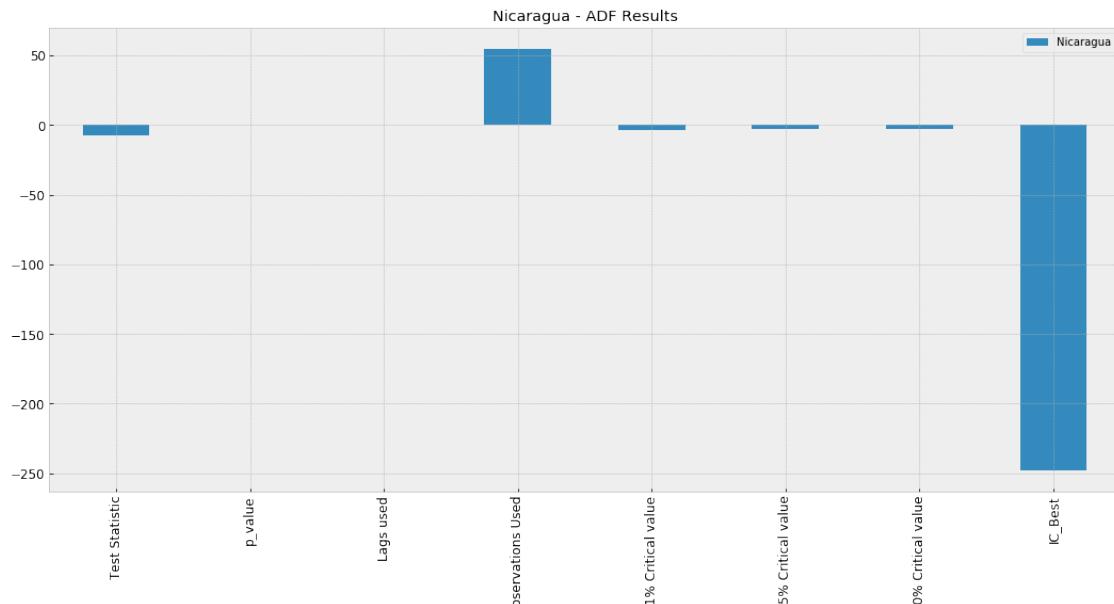


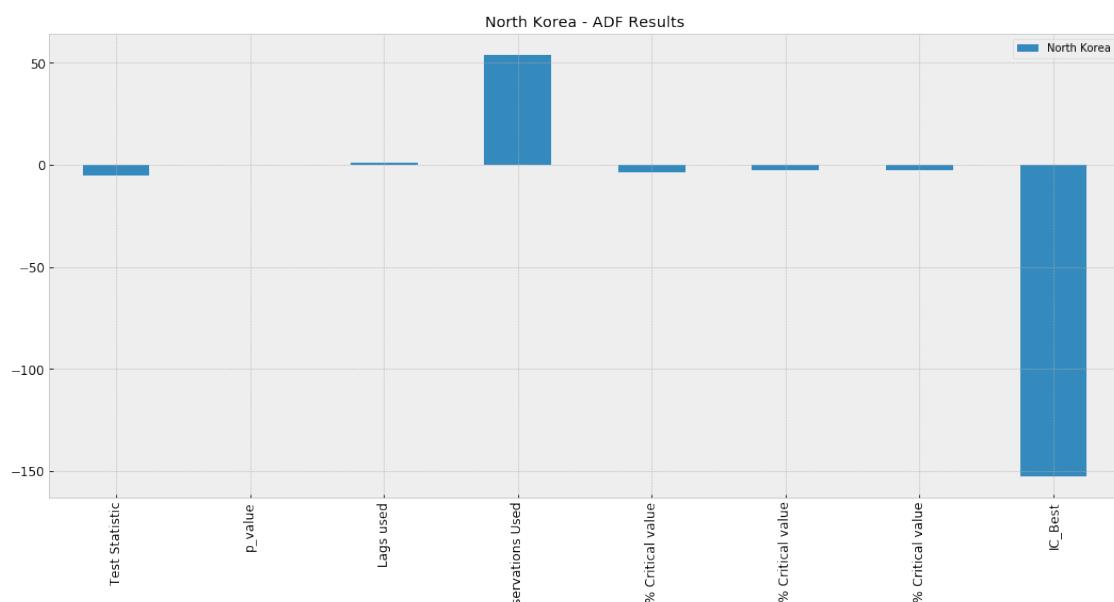
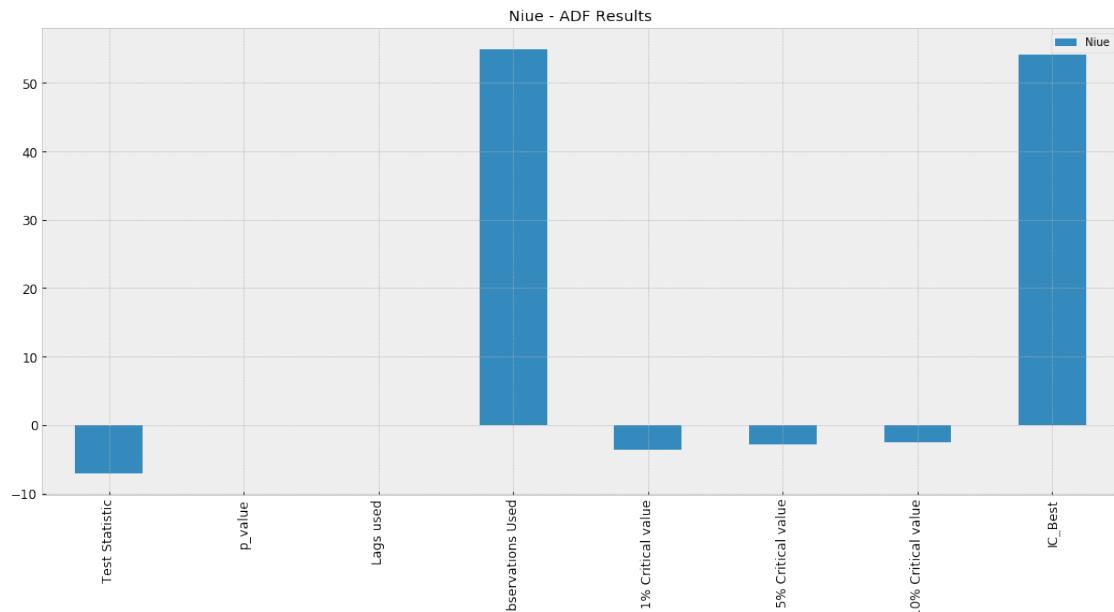


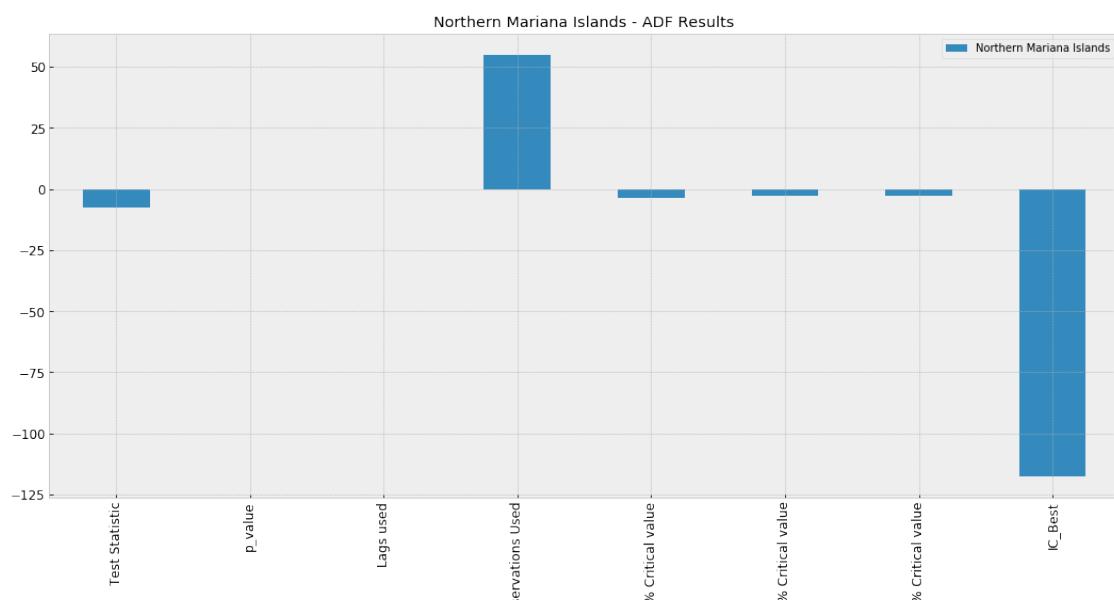
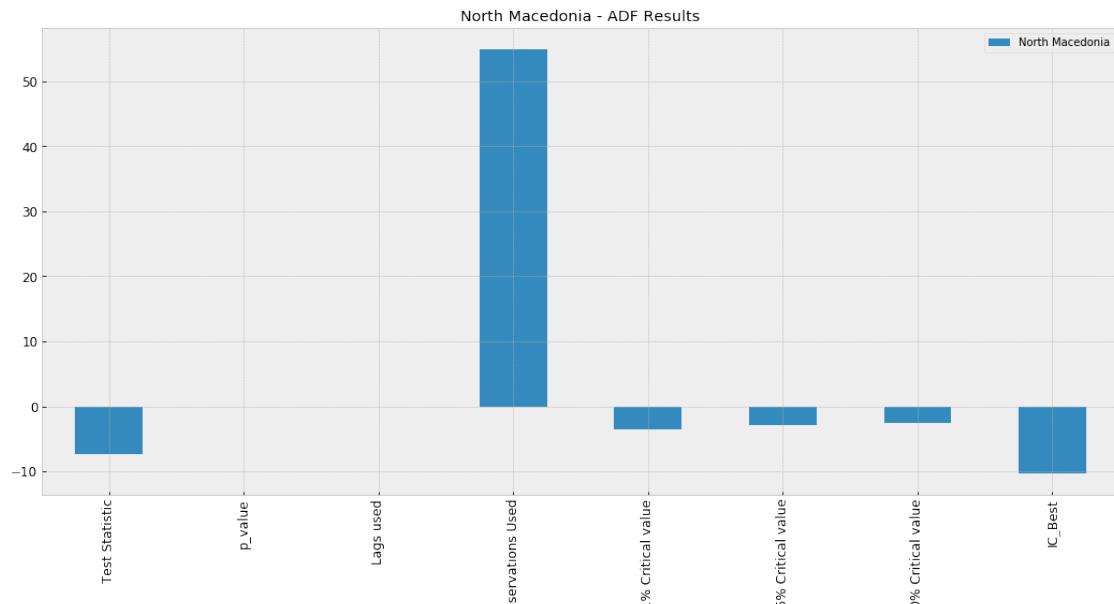


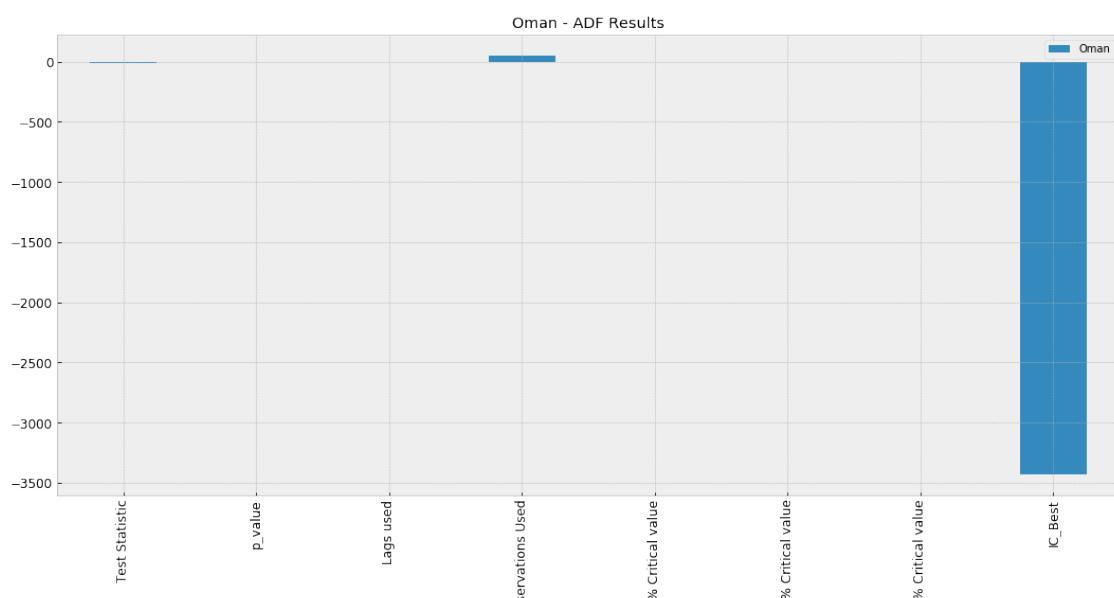
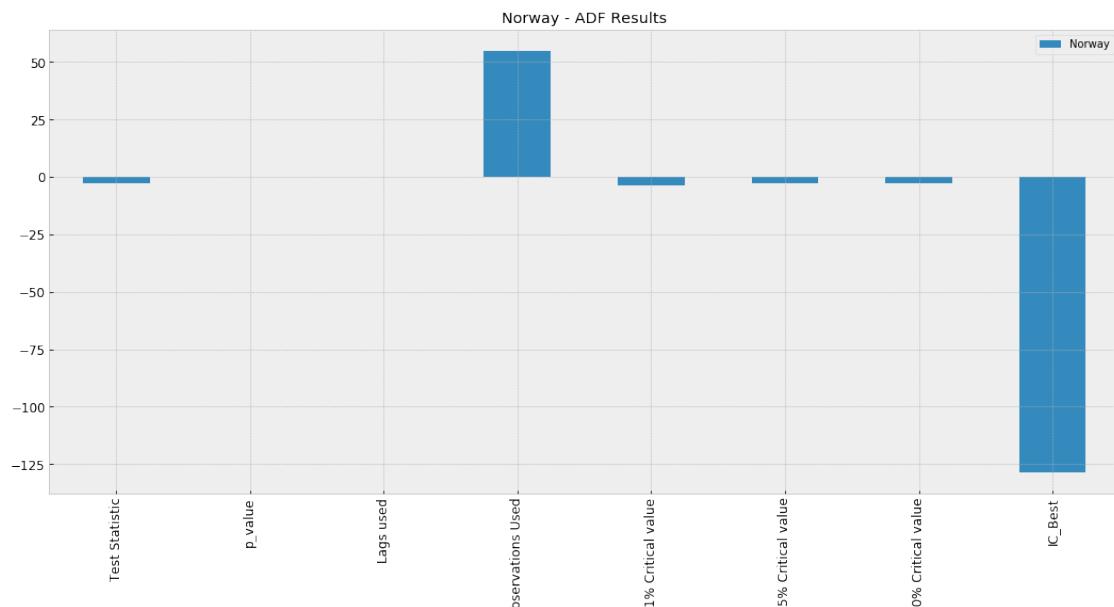


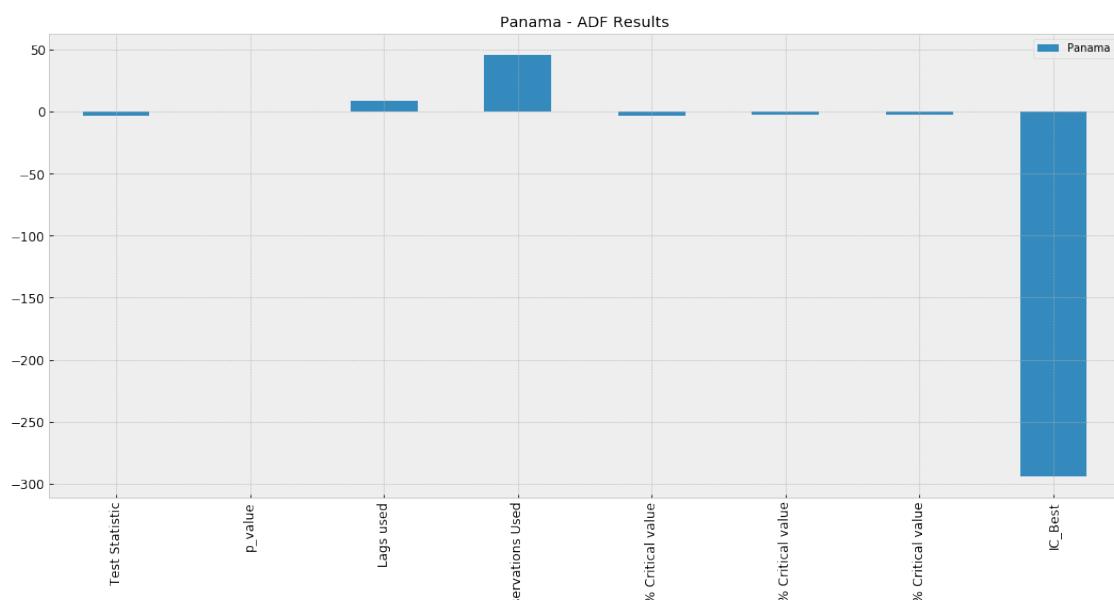
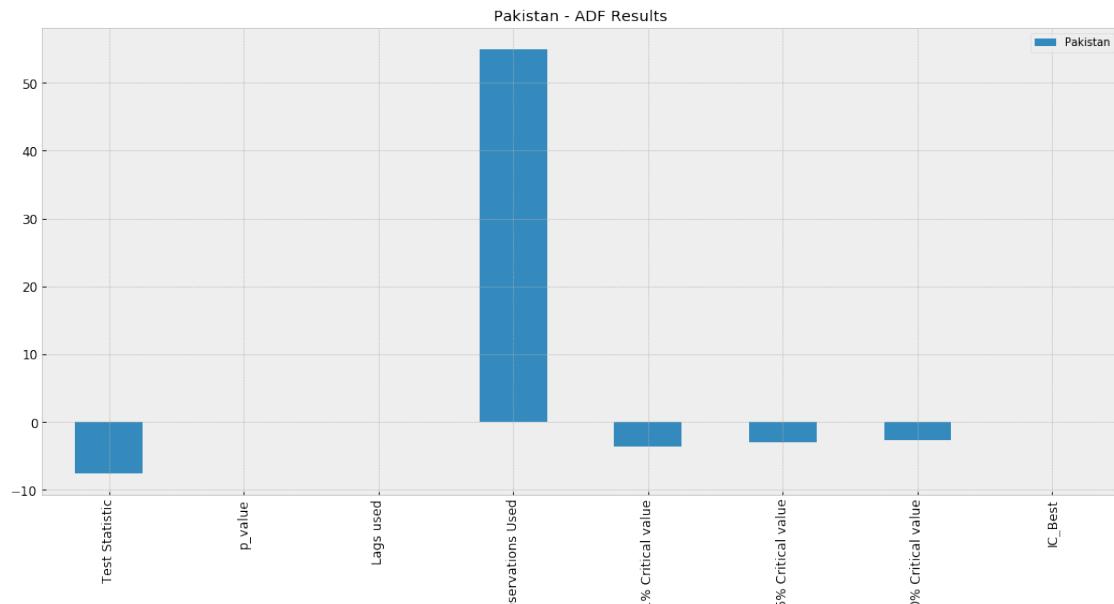


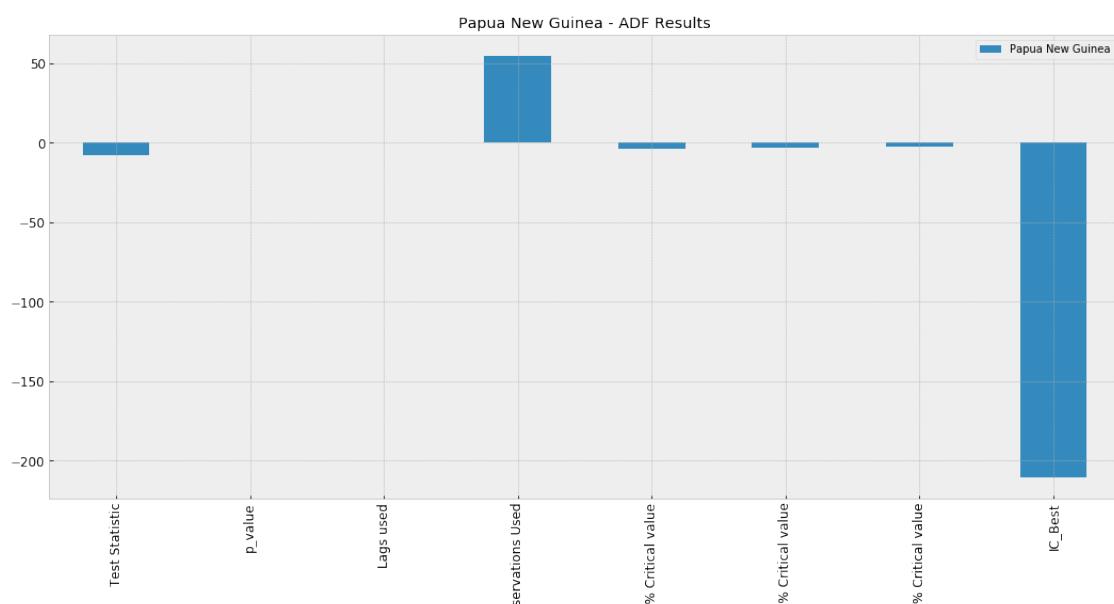
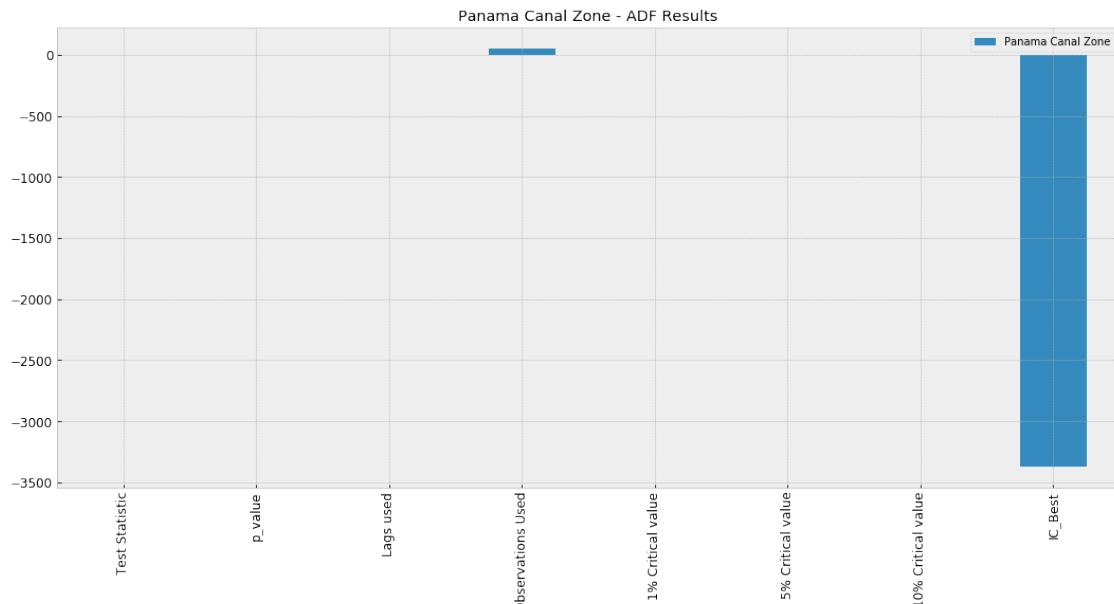


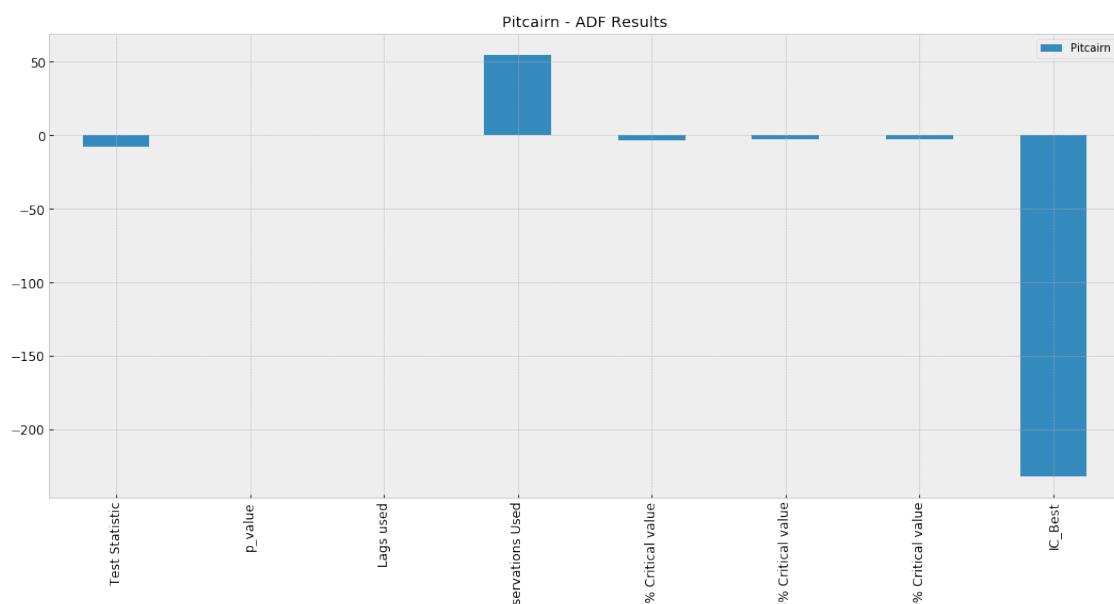
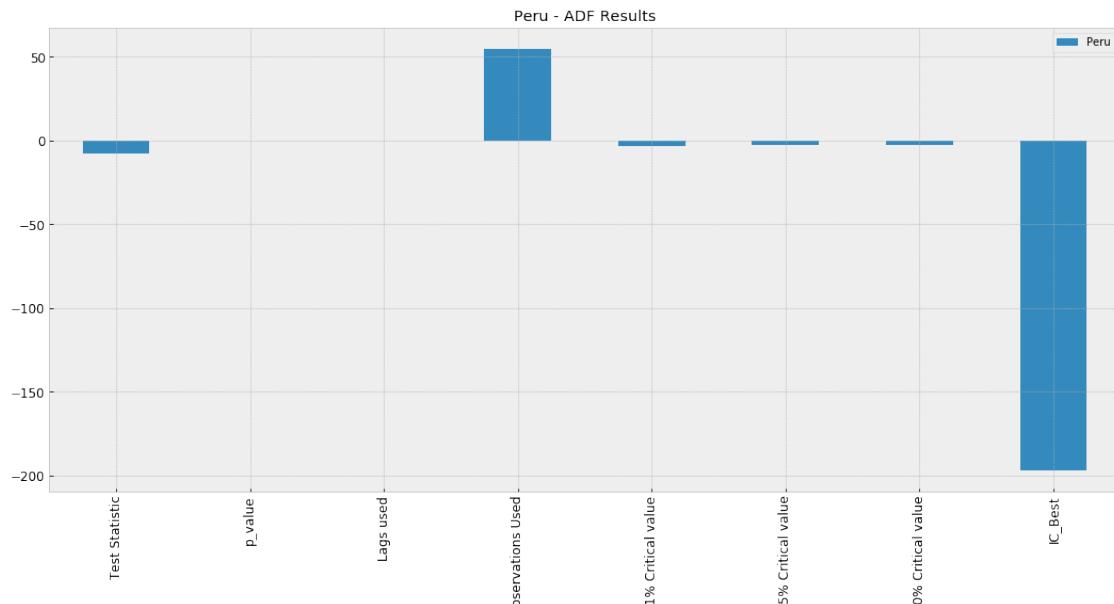


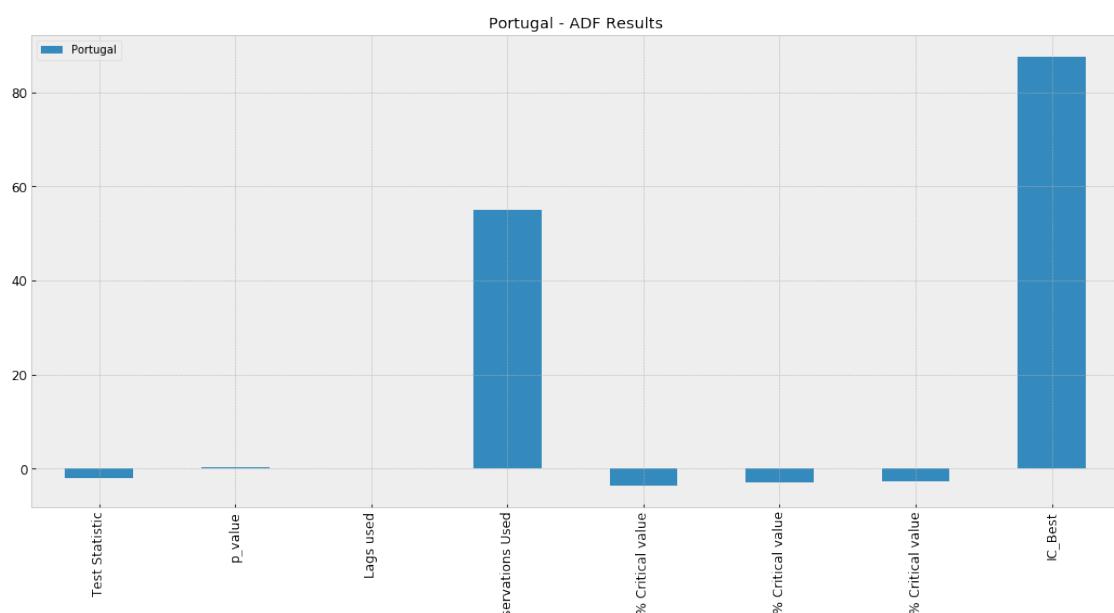
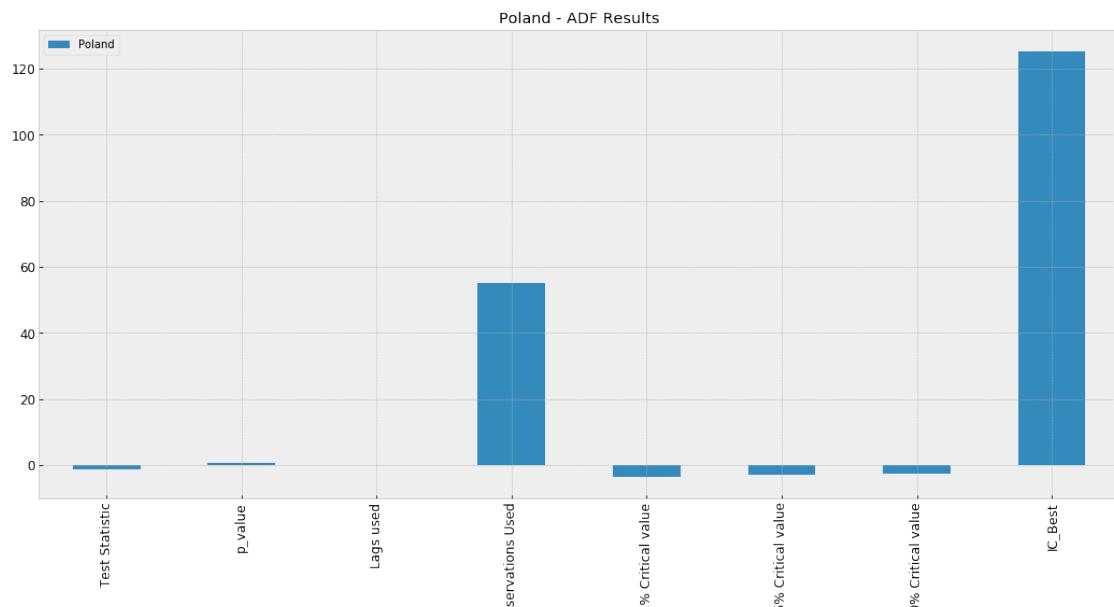


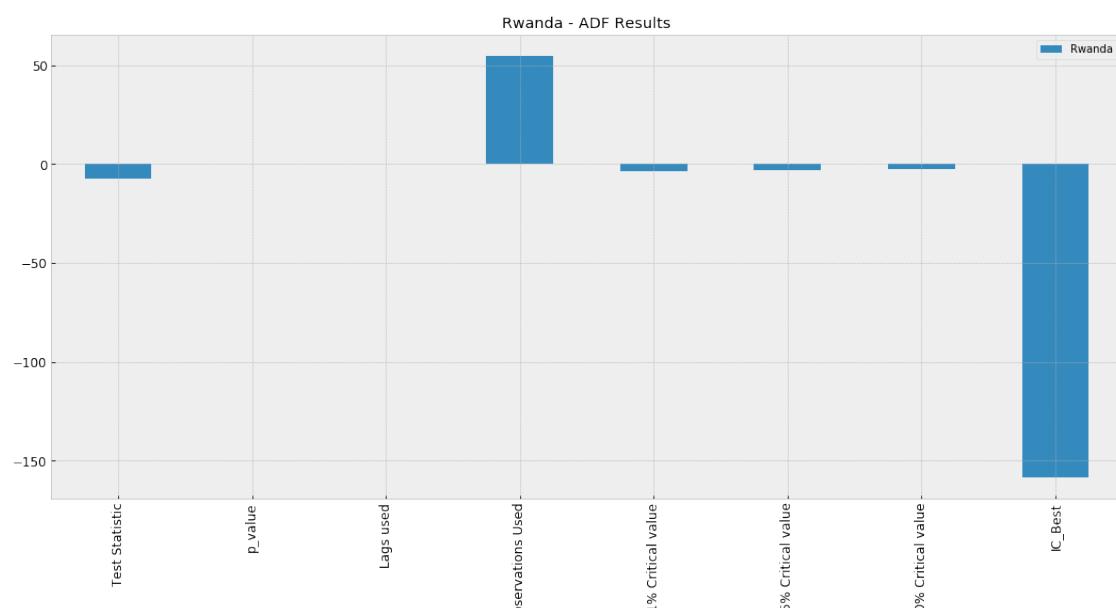
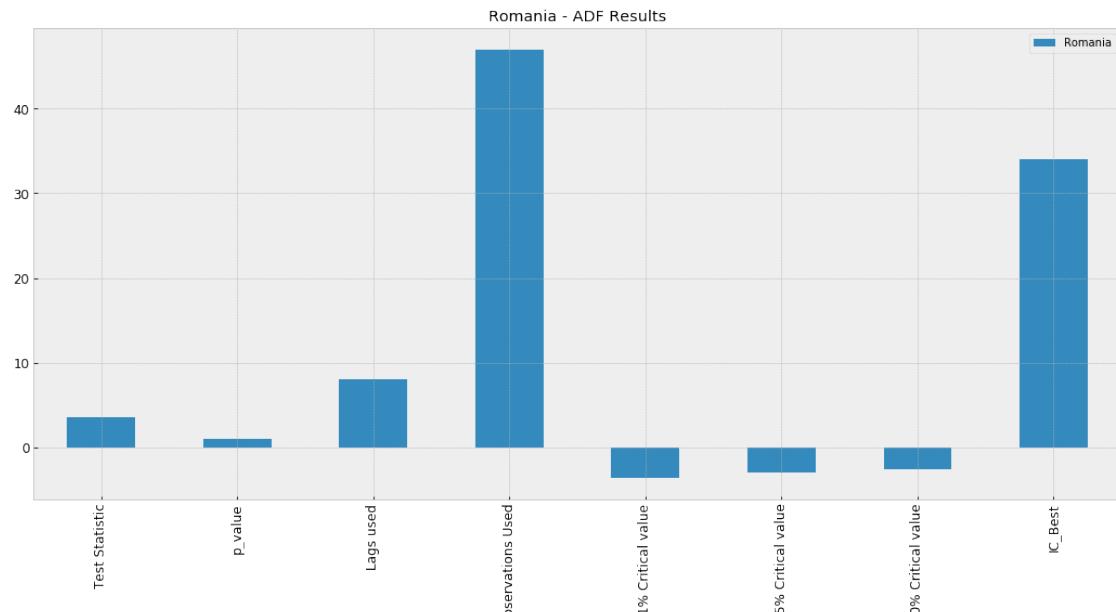


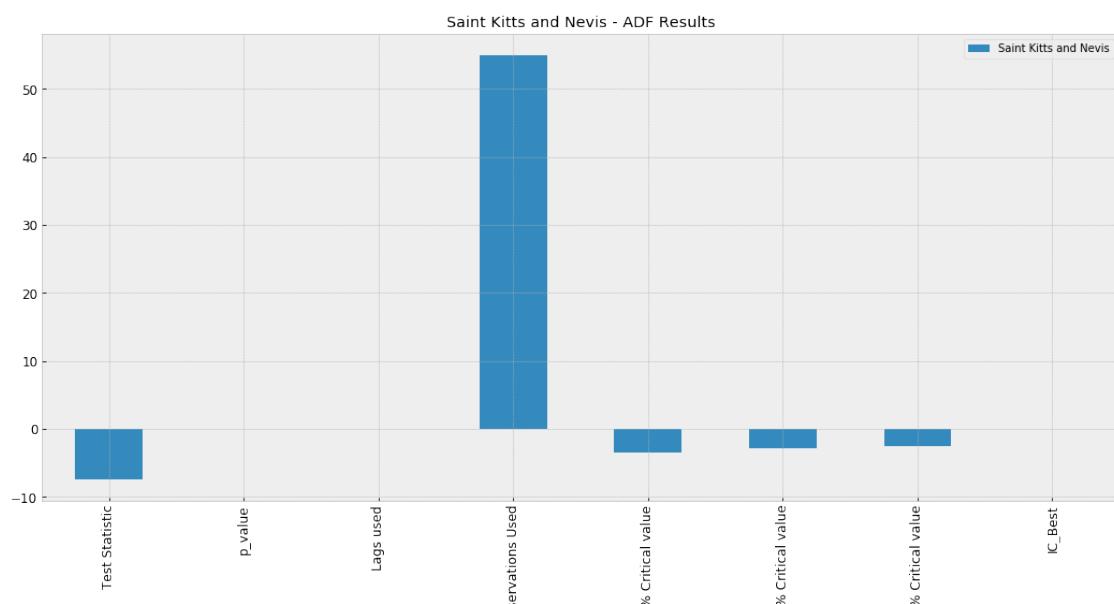
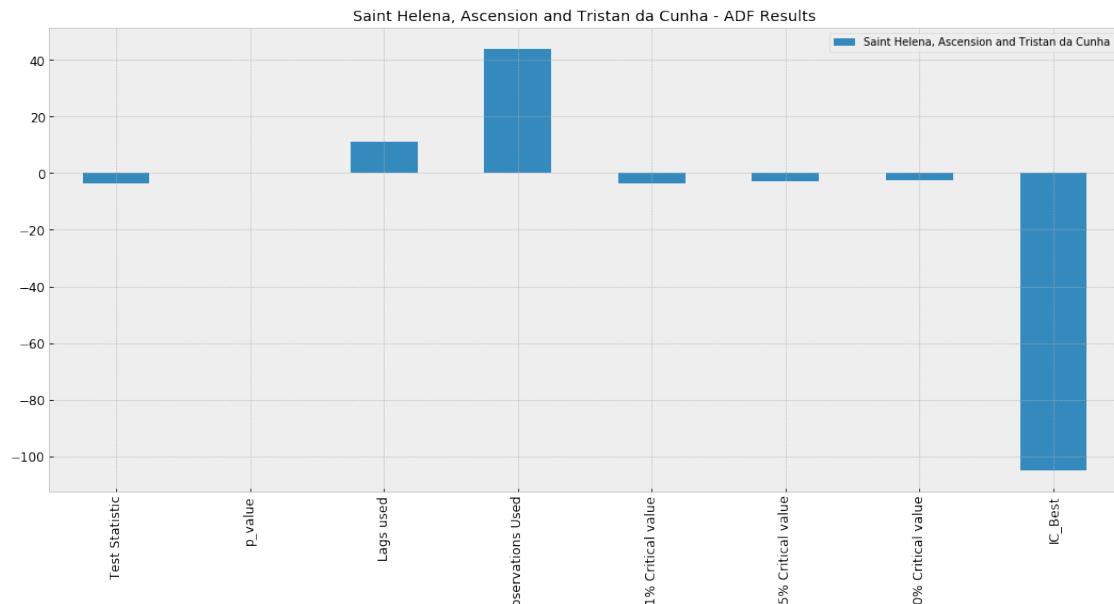


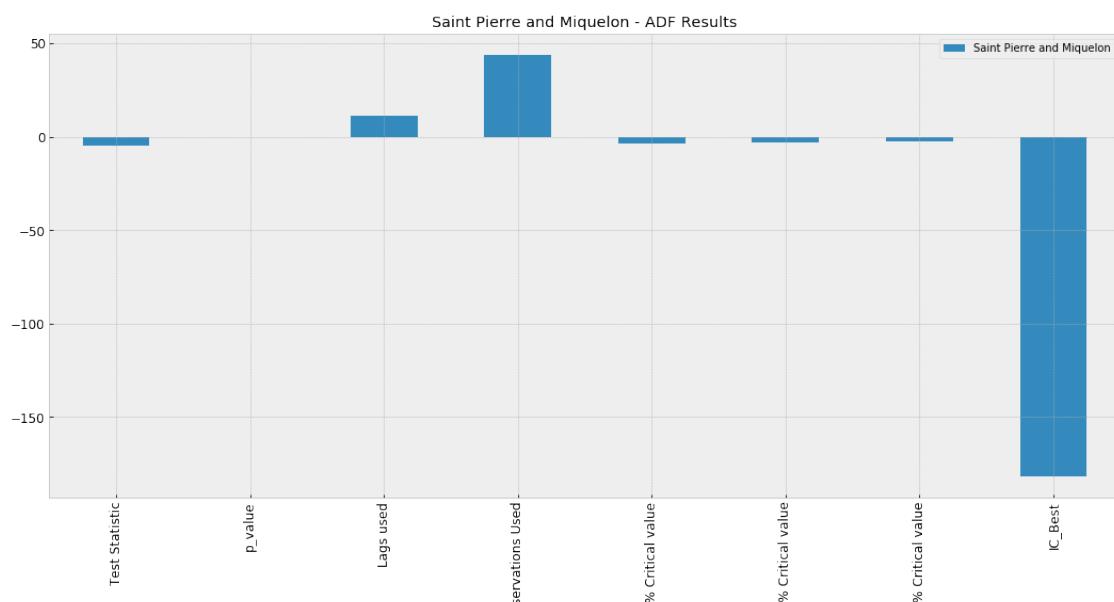
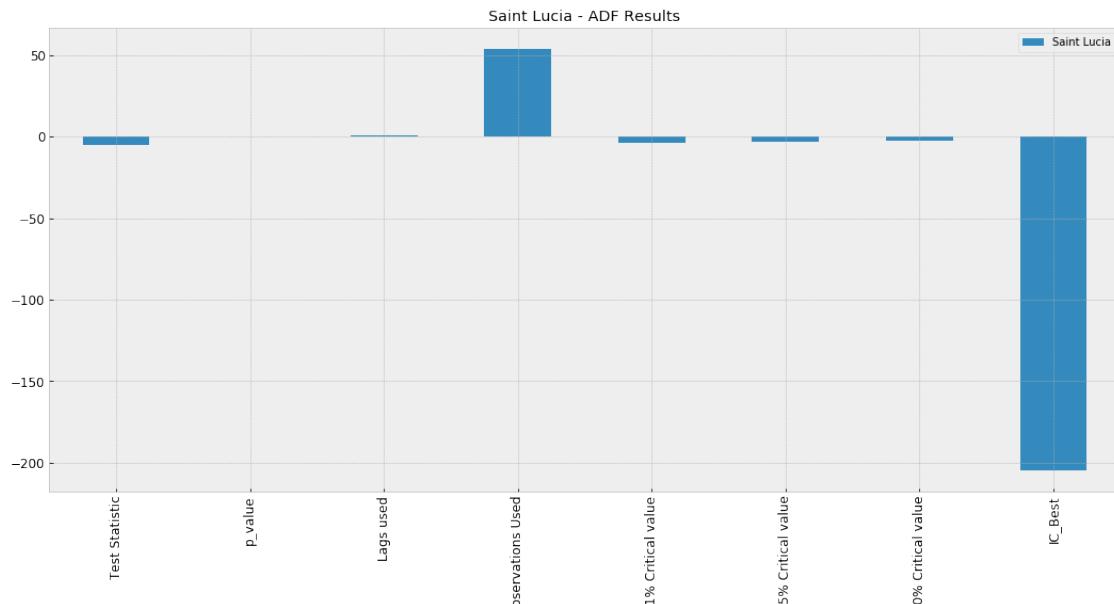




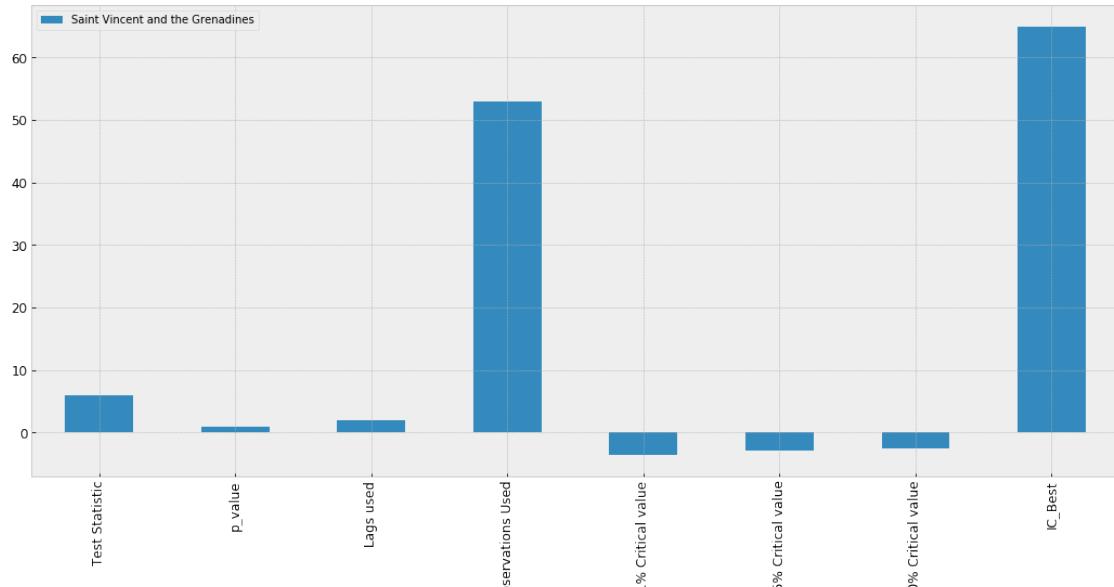




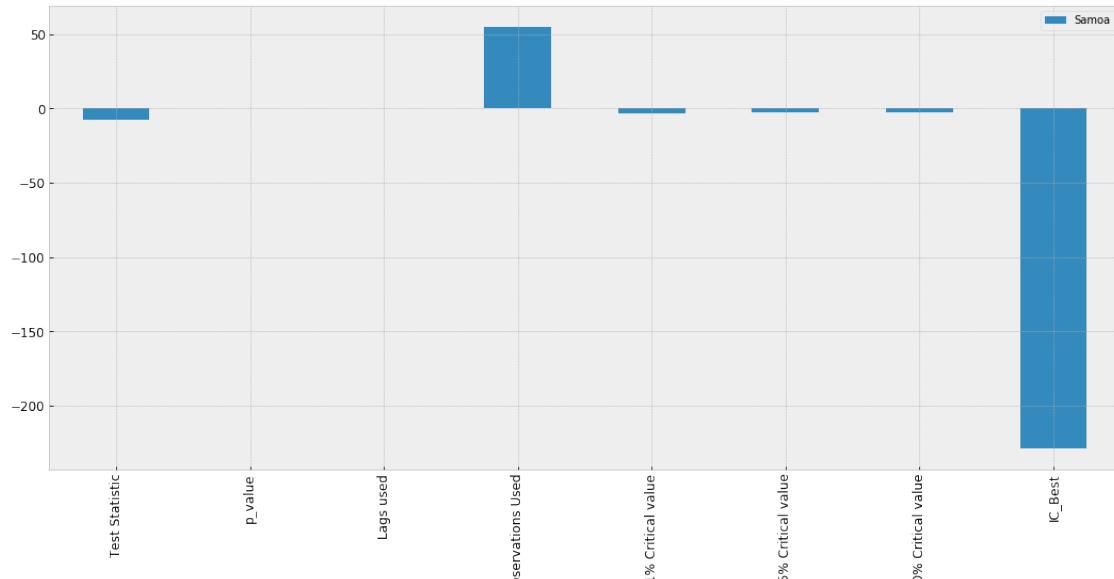


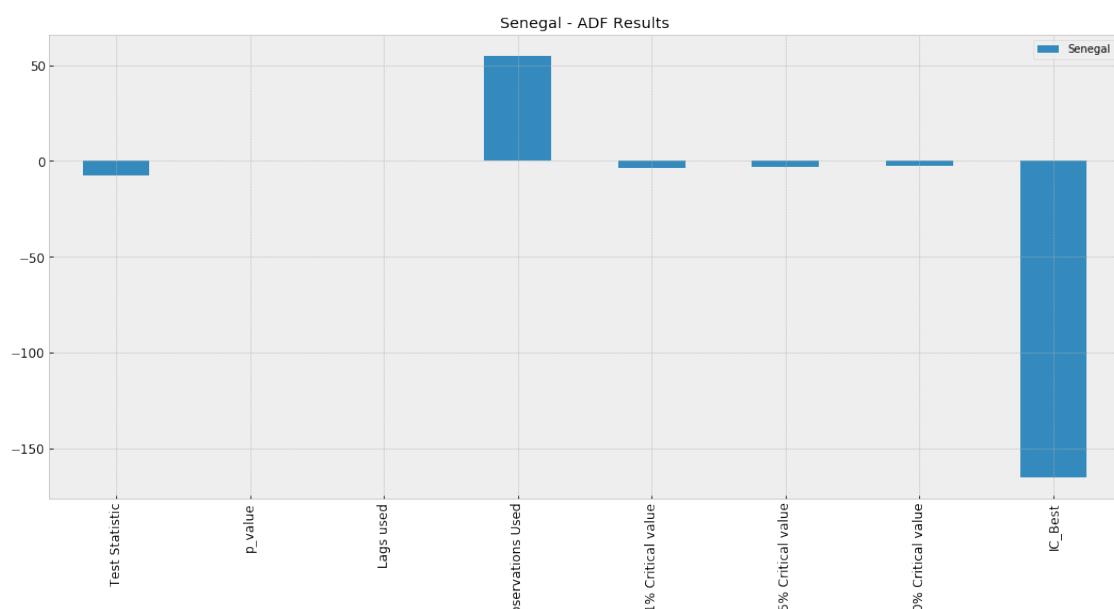
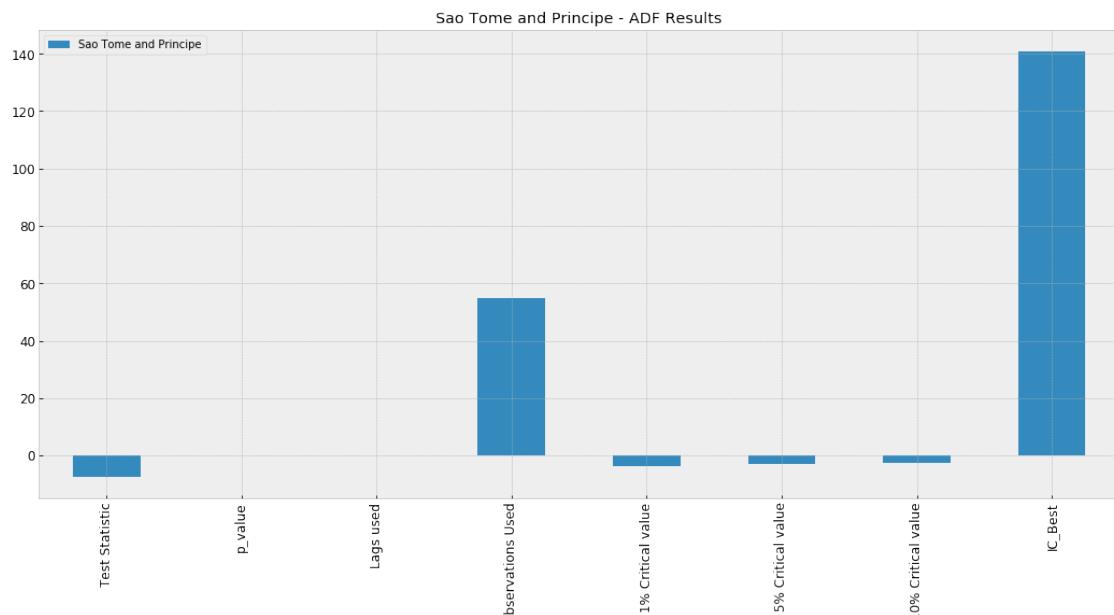


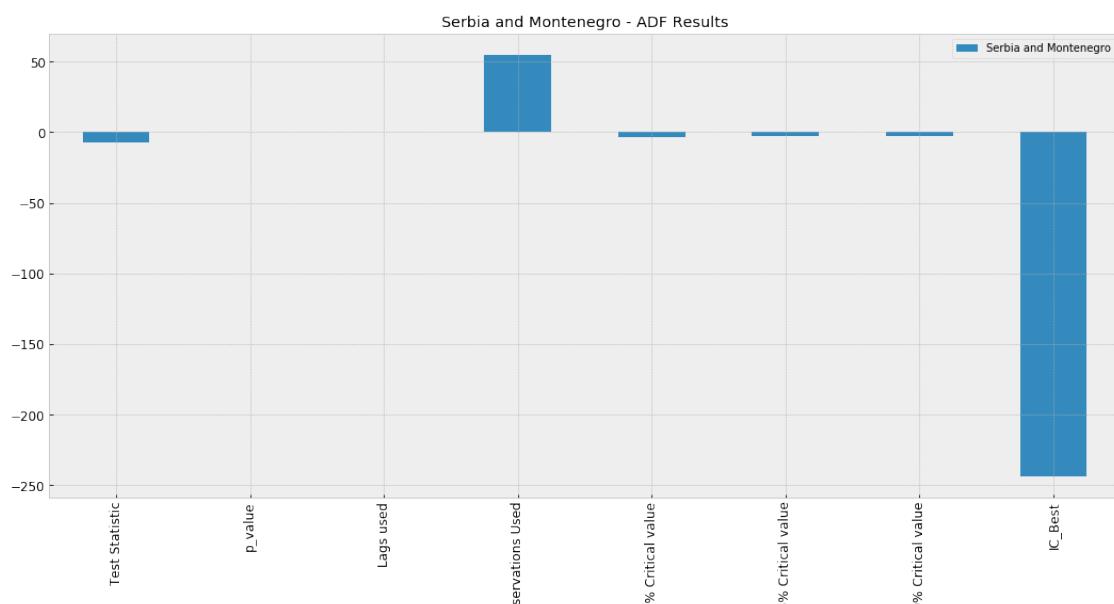
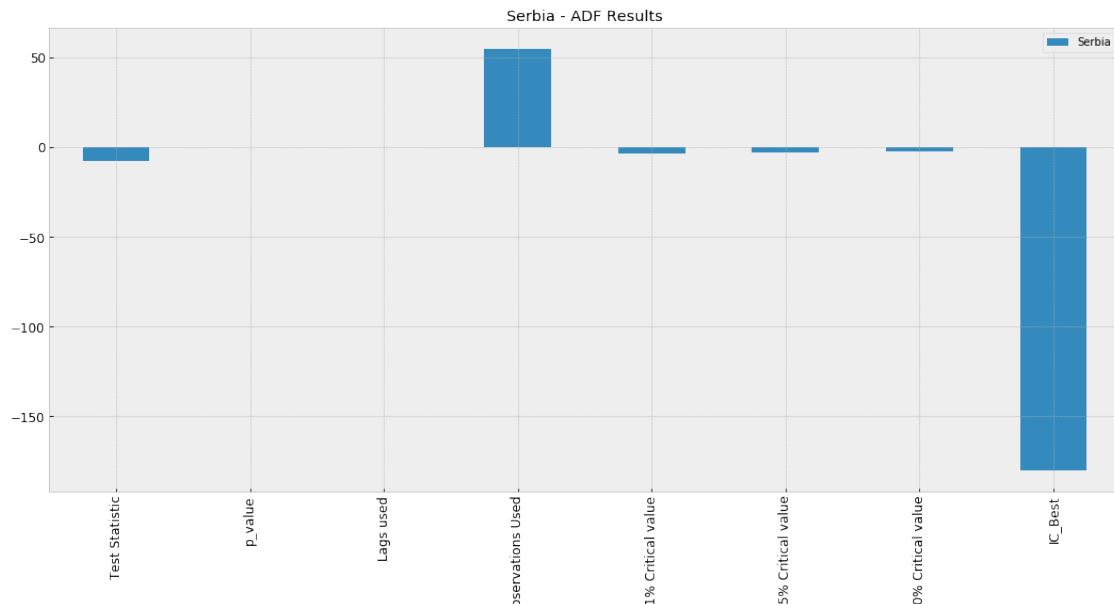
Saint Vincent and the Grenadines - ADF Results

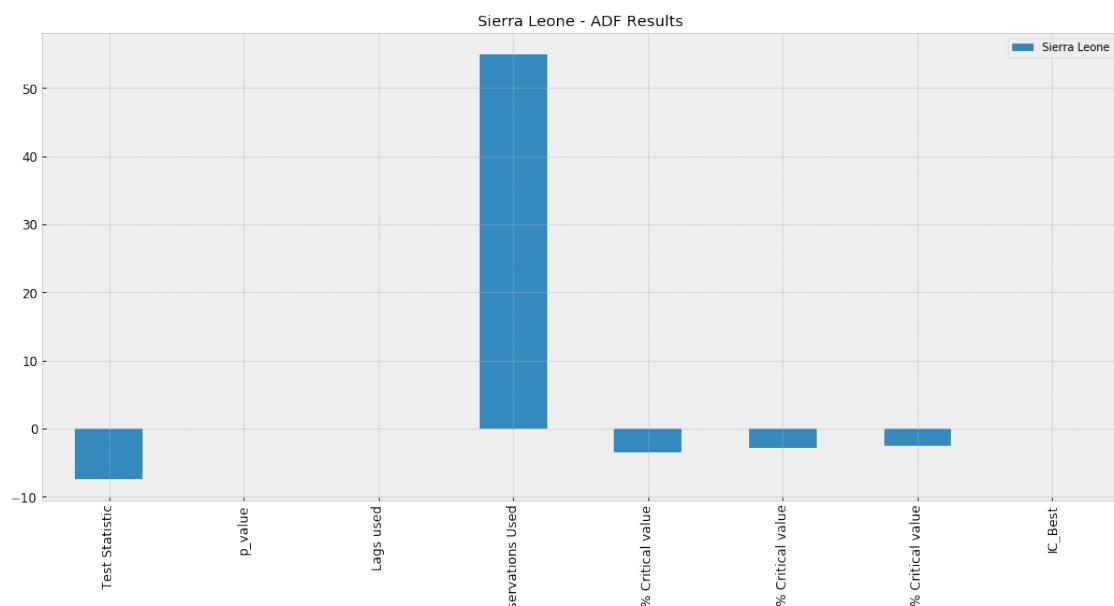
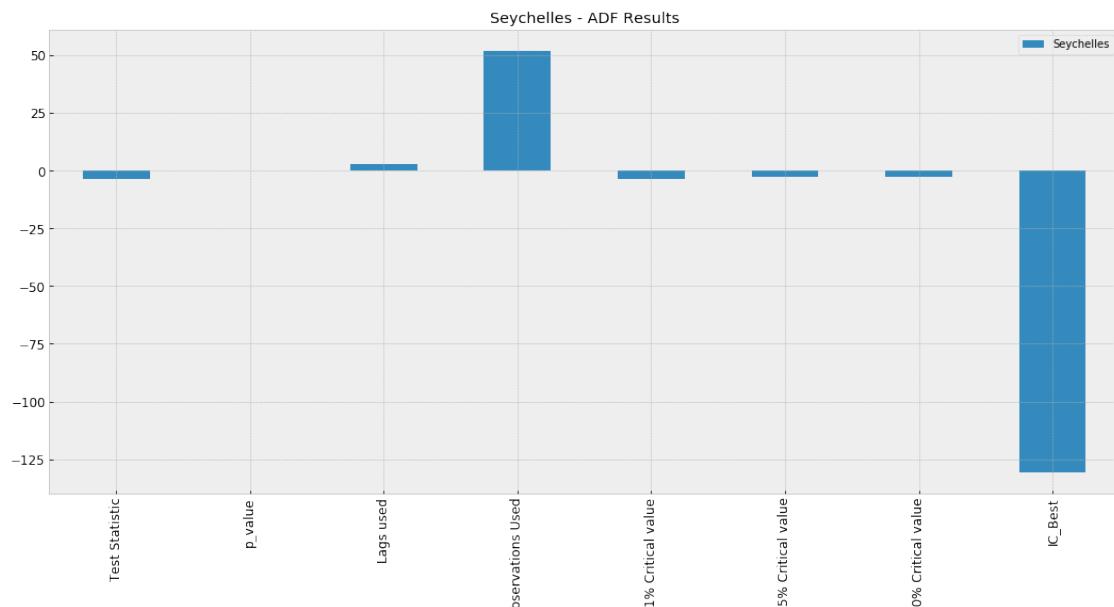


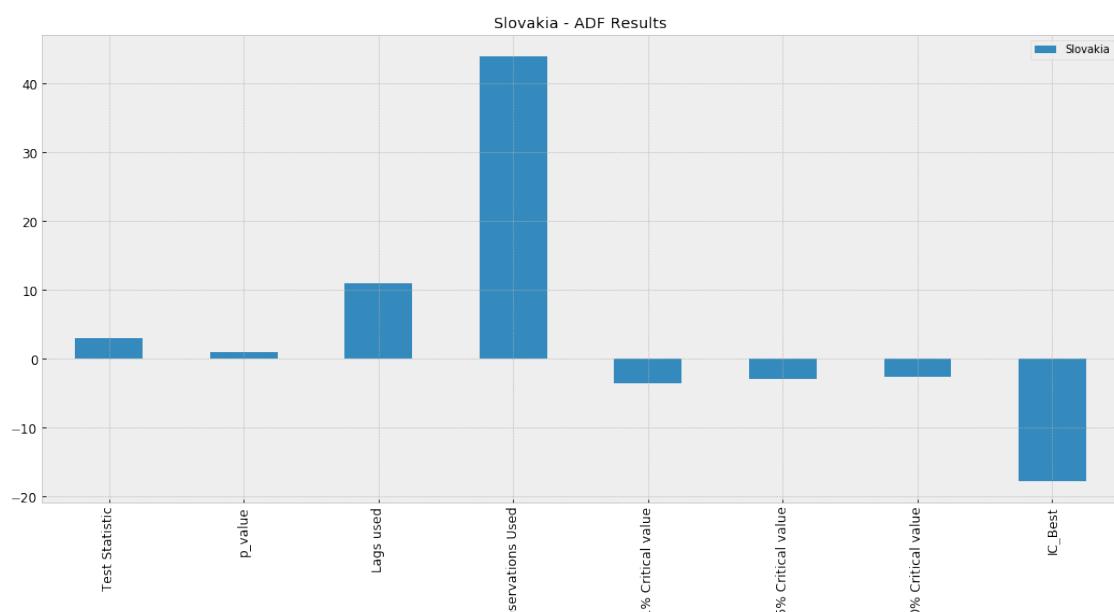
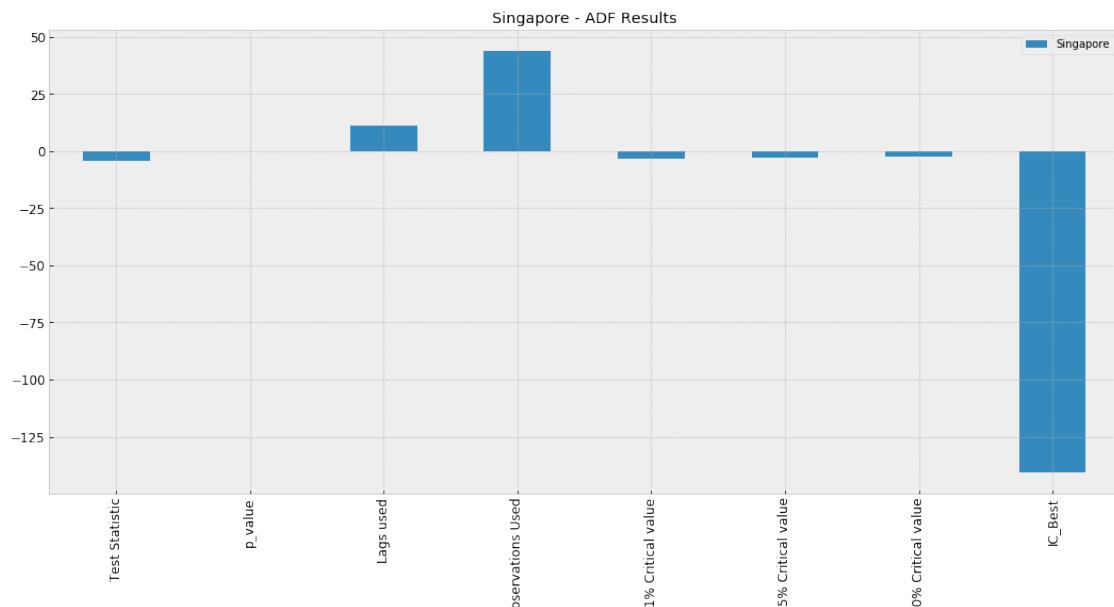
Samoa - ADF Results

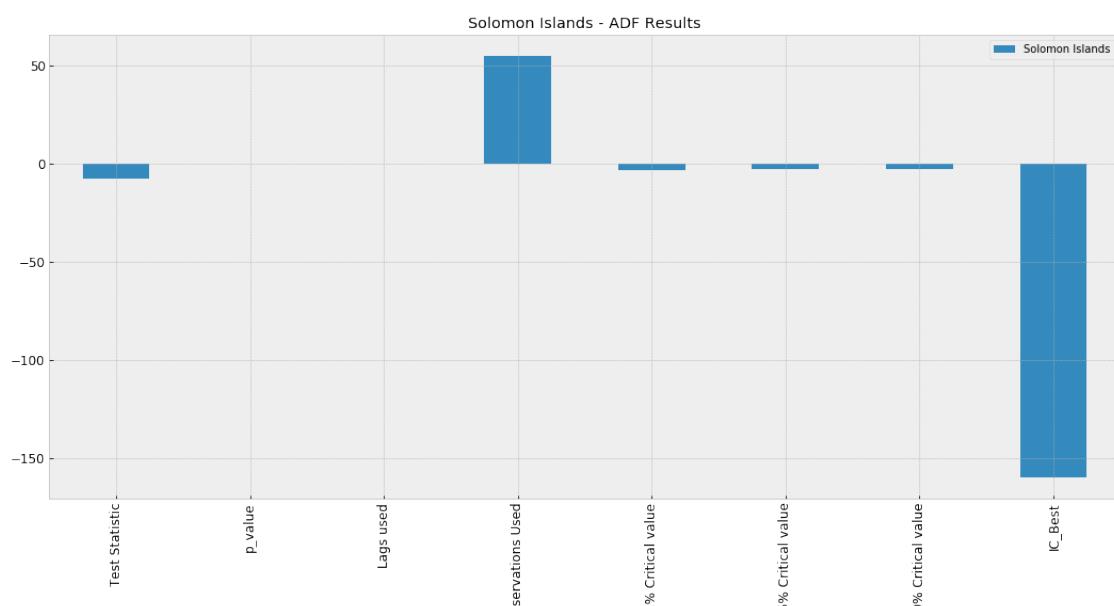
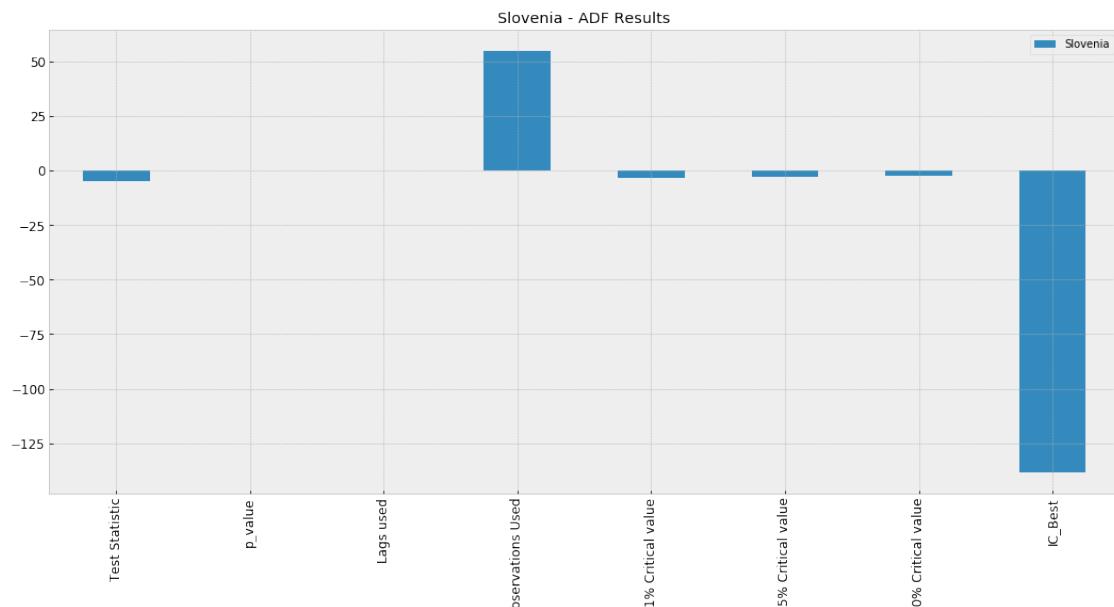


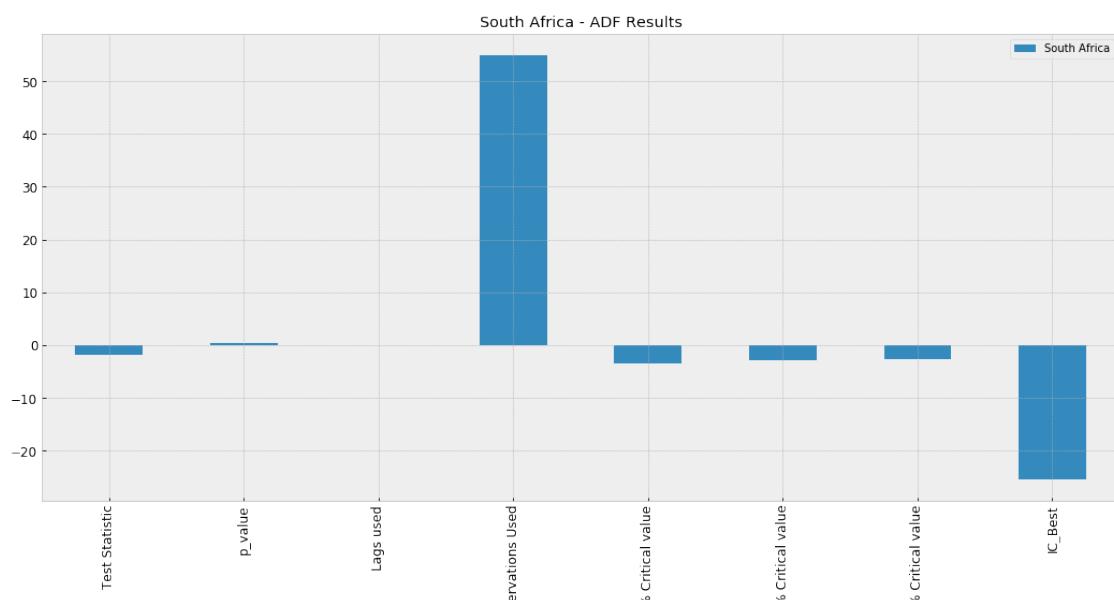
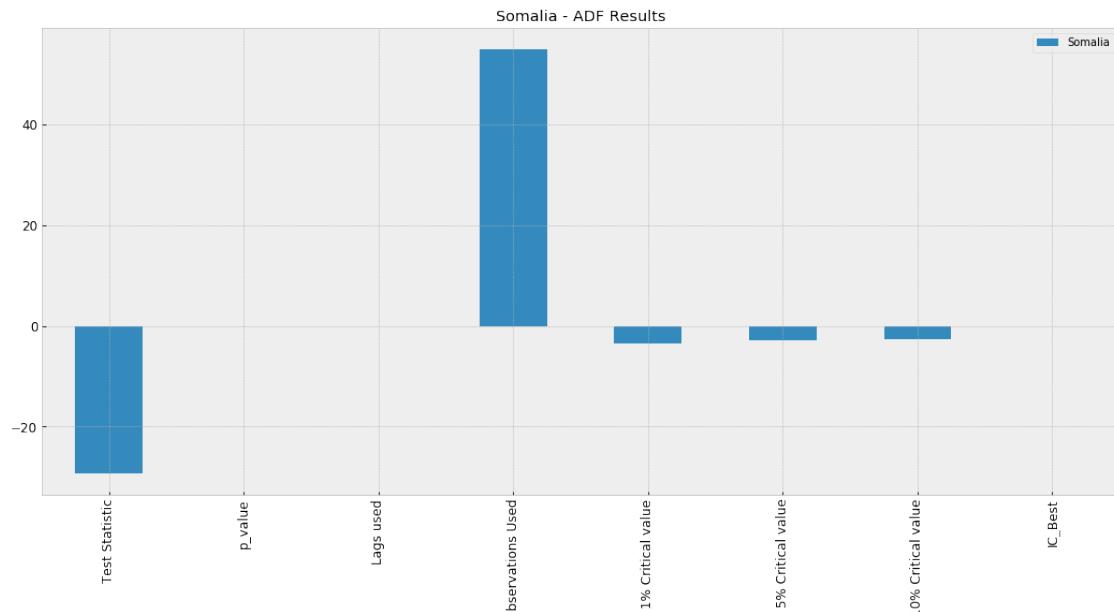


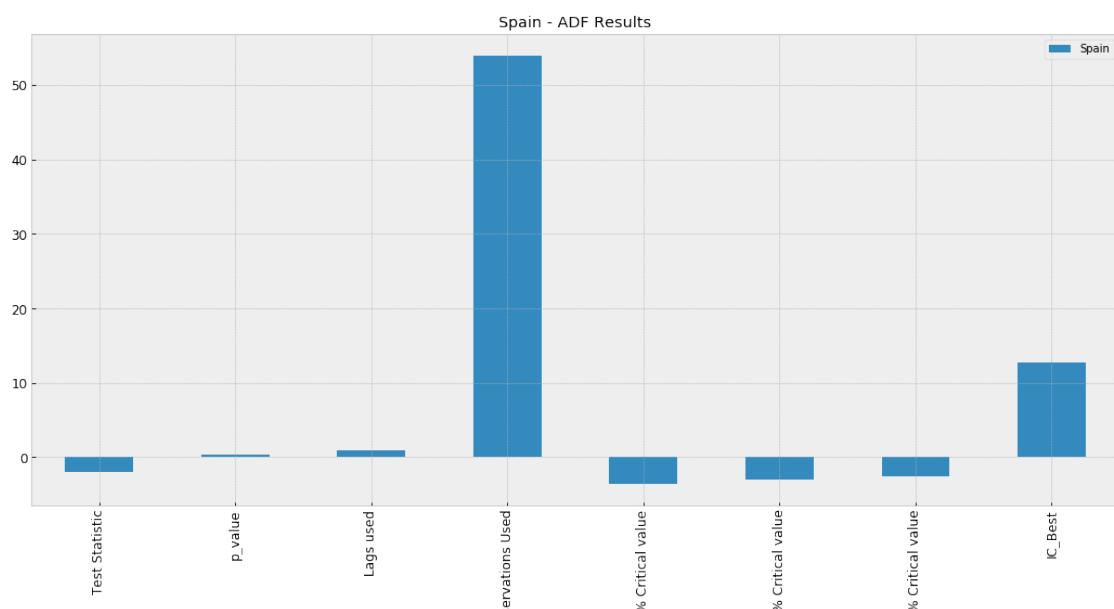
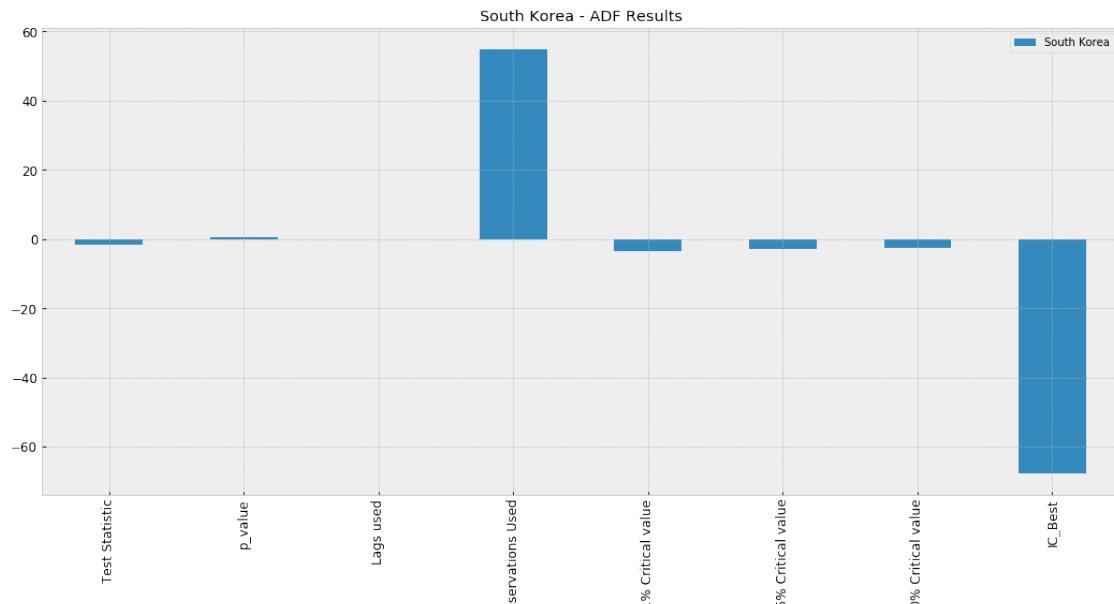


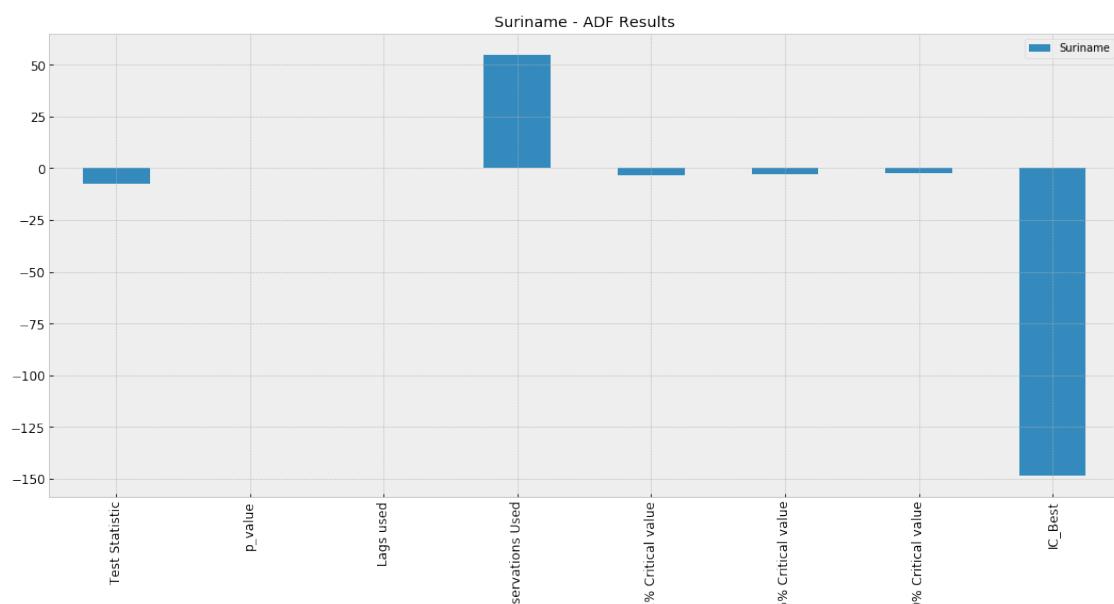
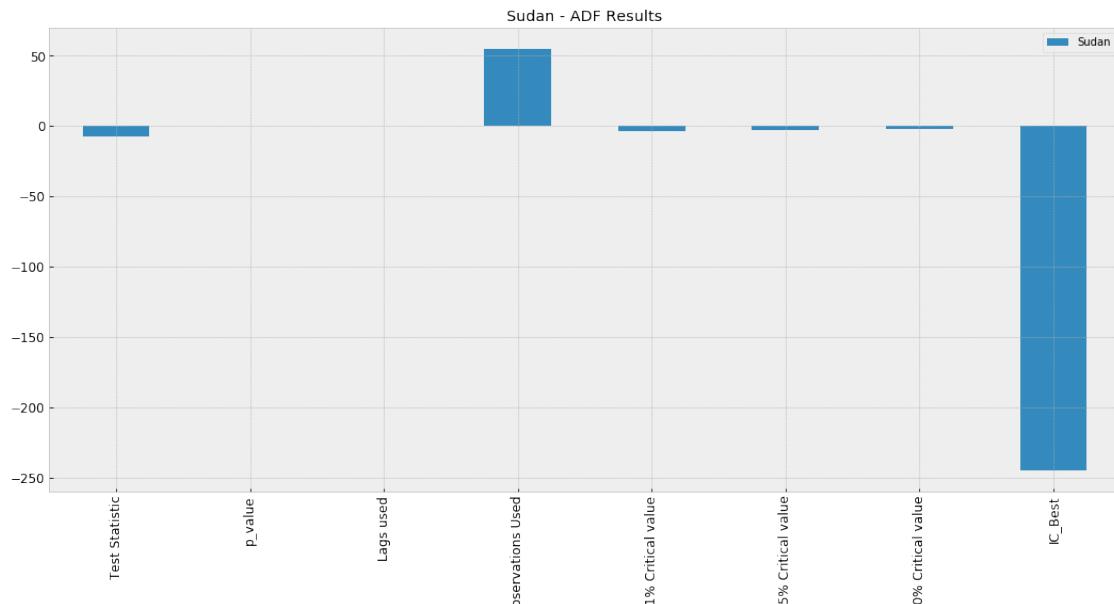


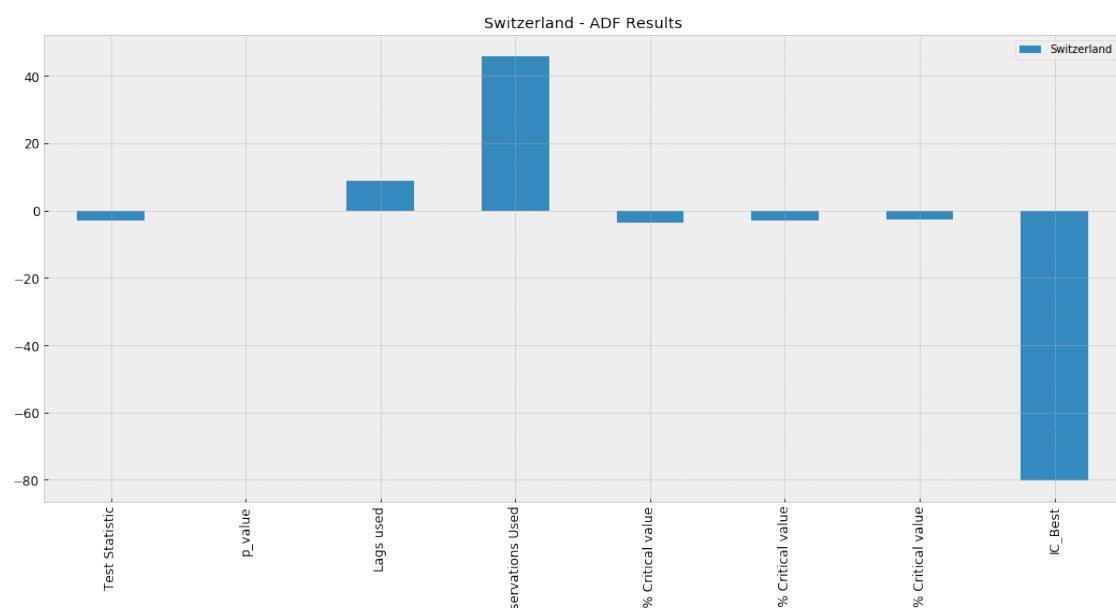
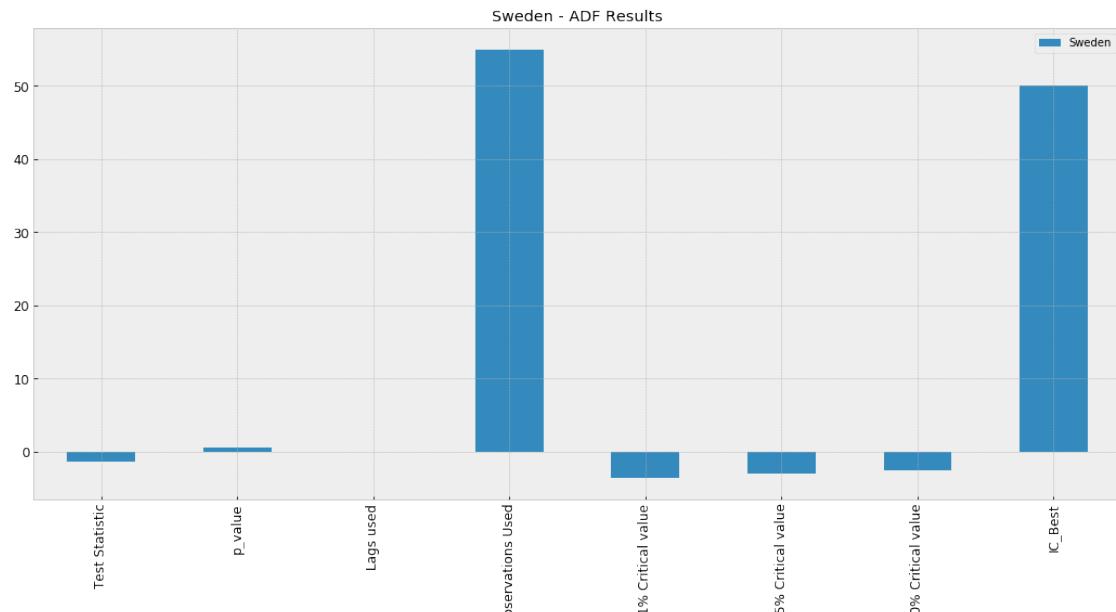


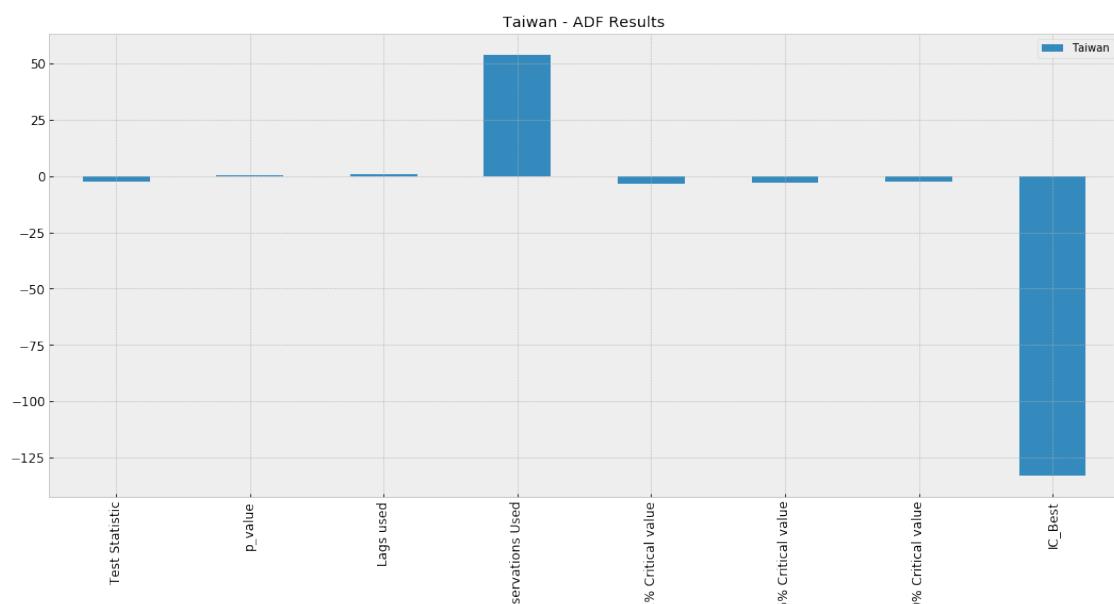
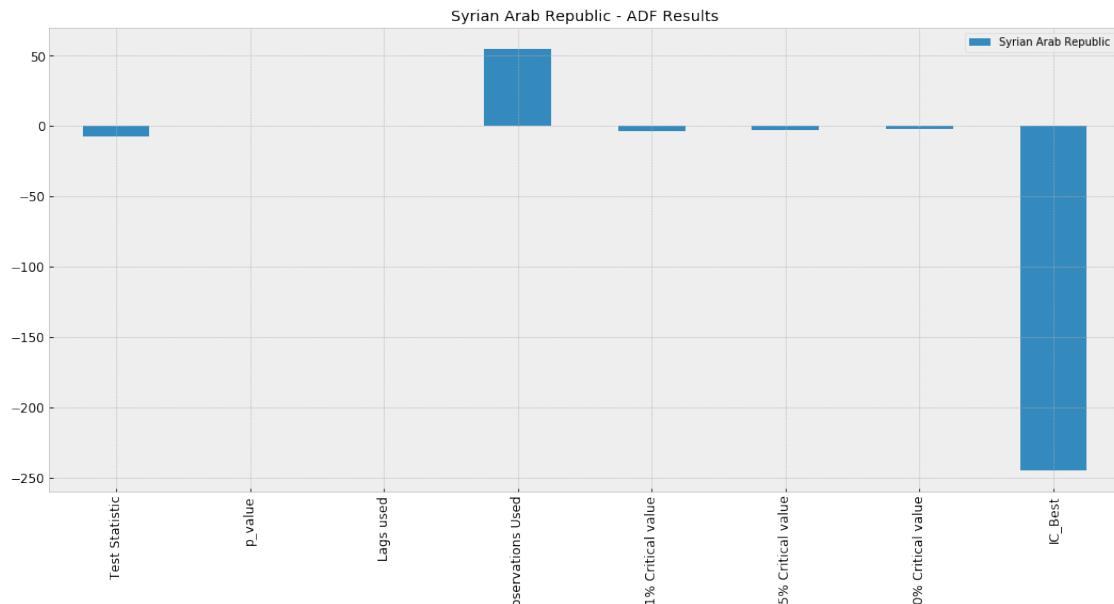


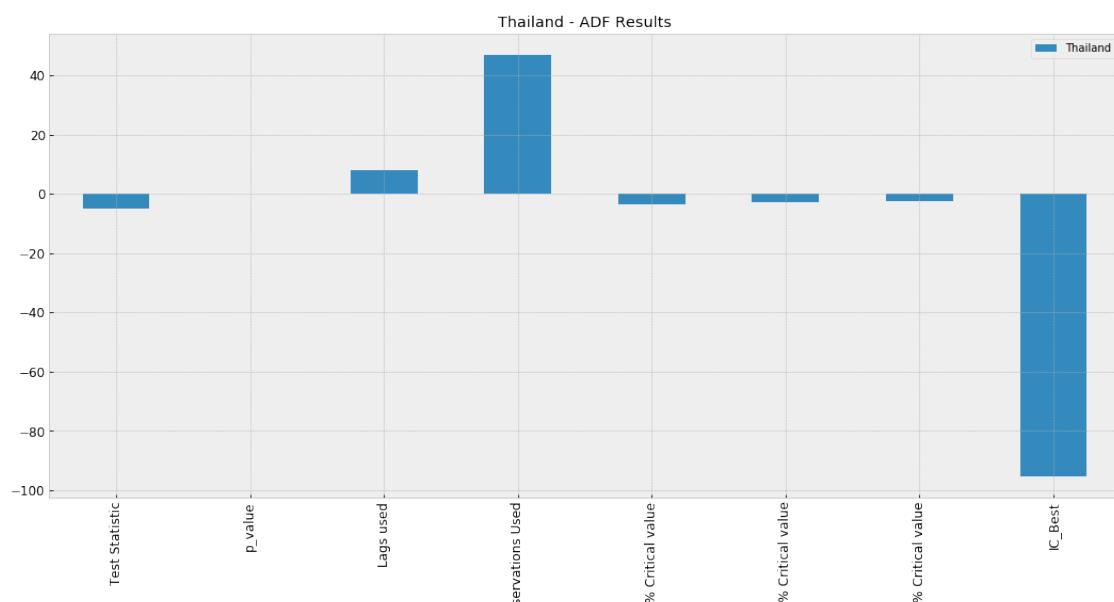
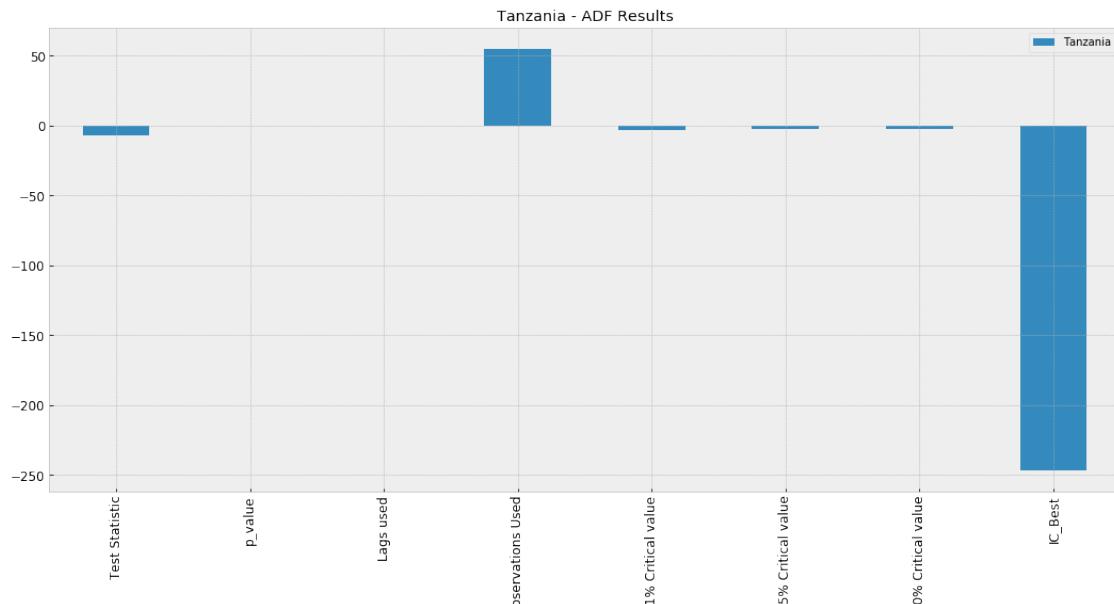


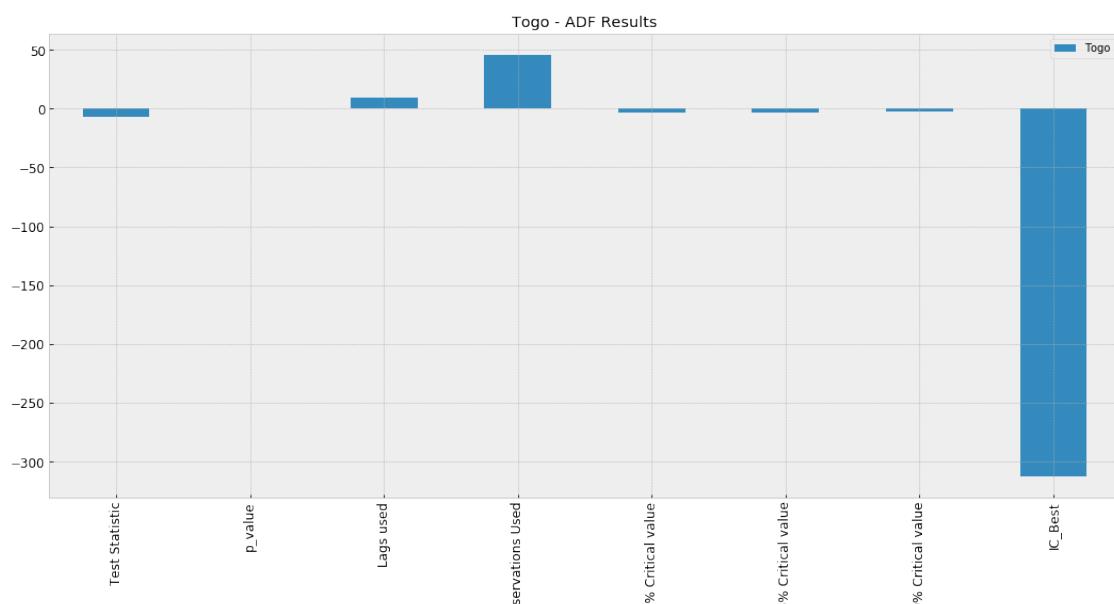
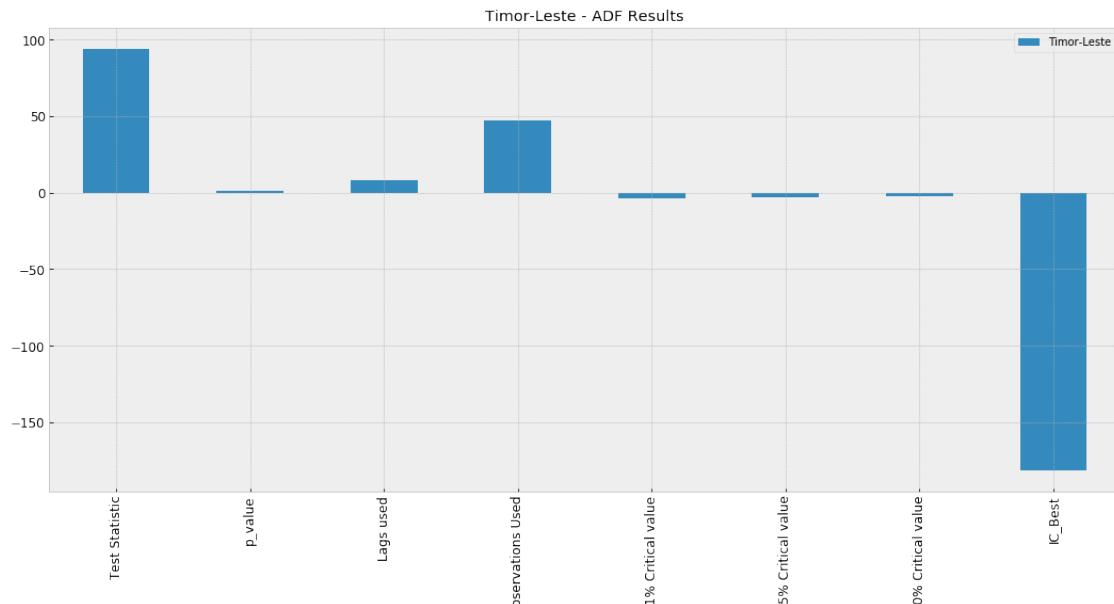


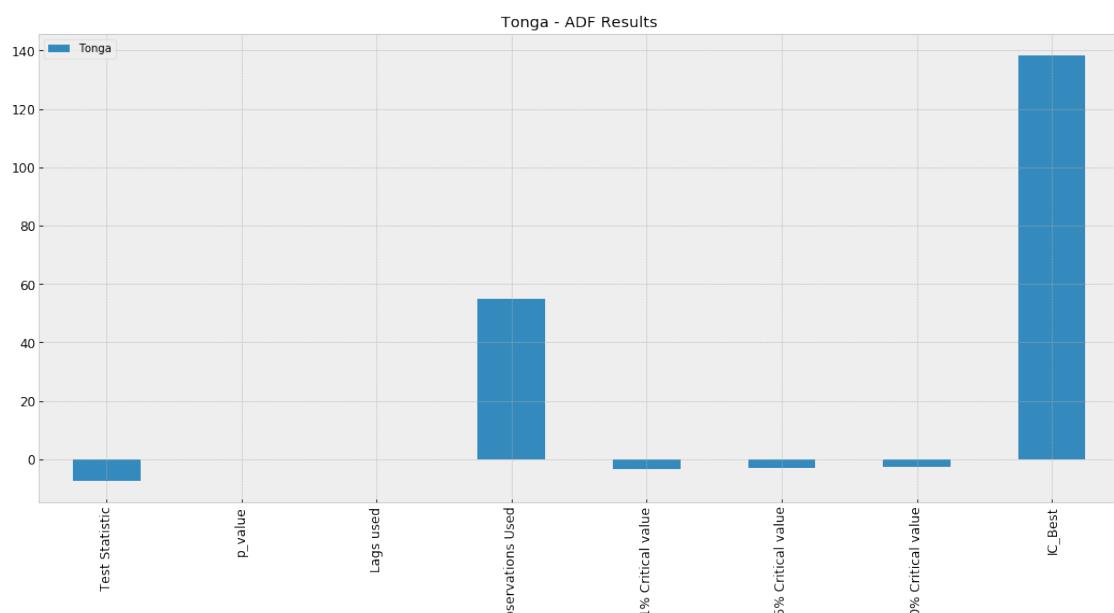
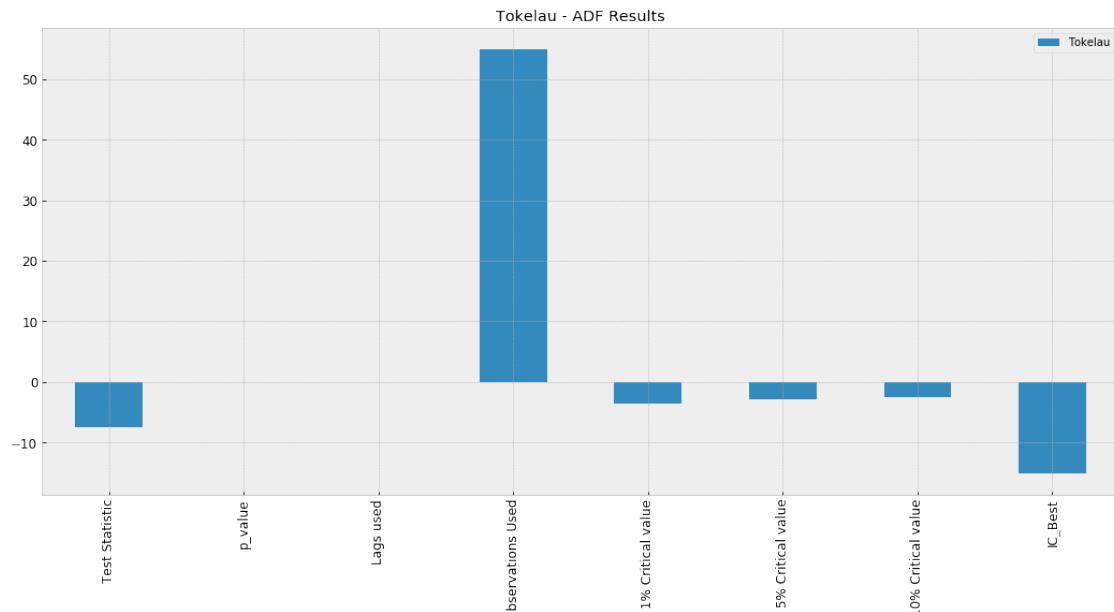


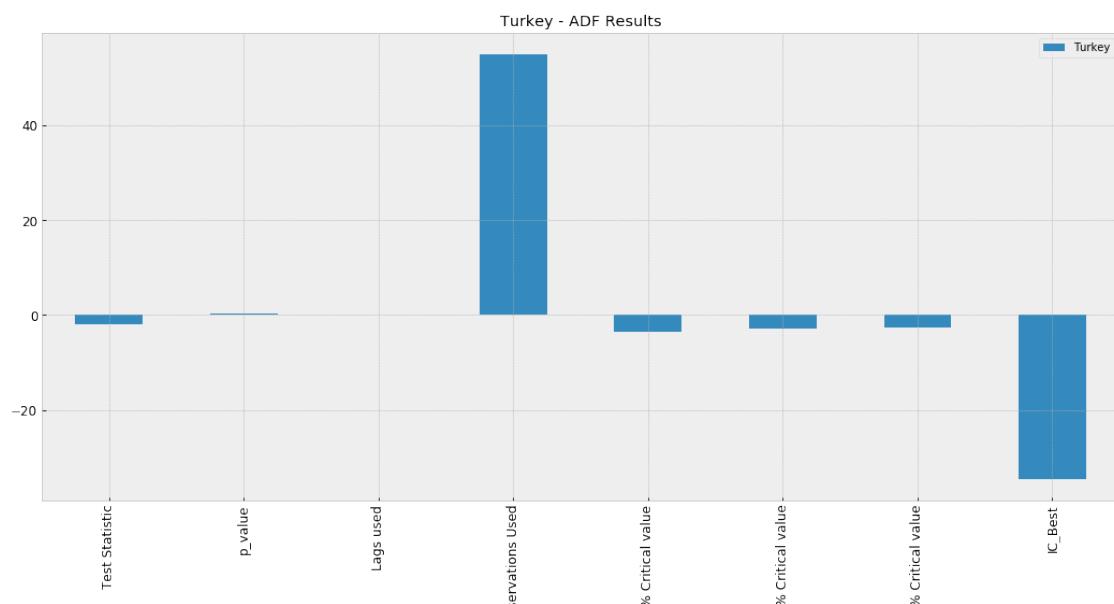
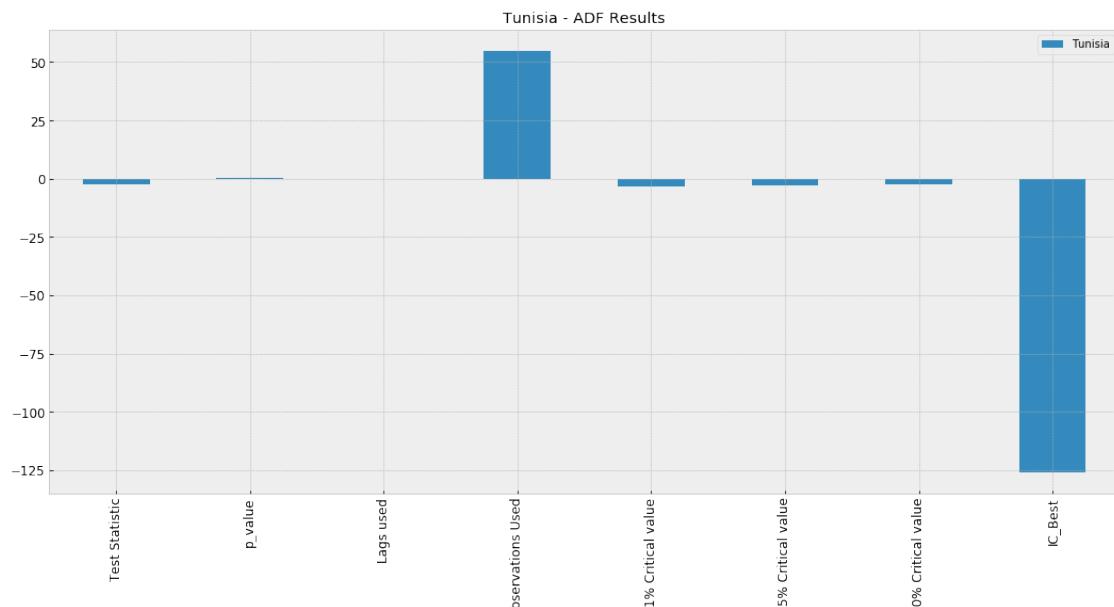


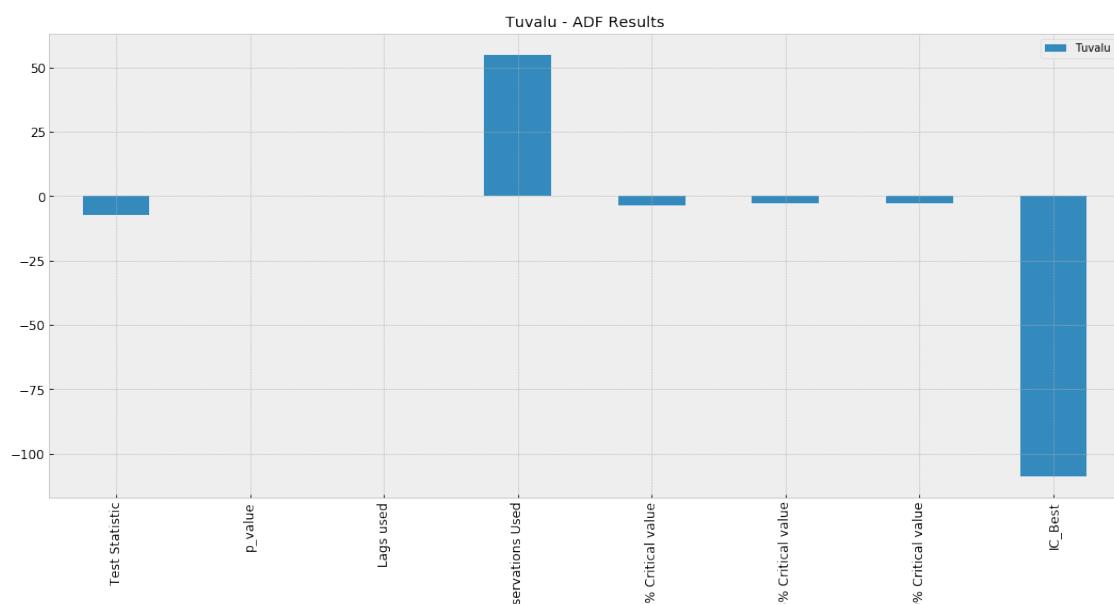
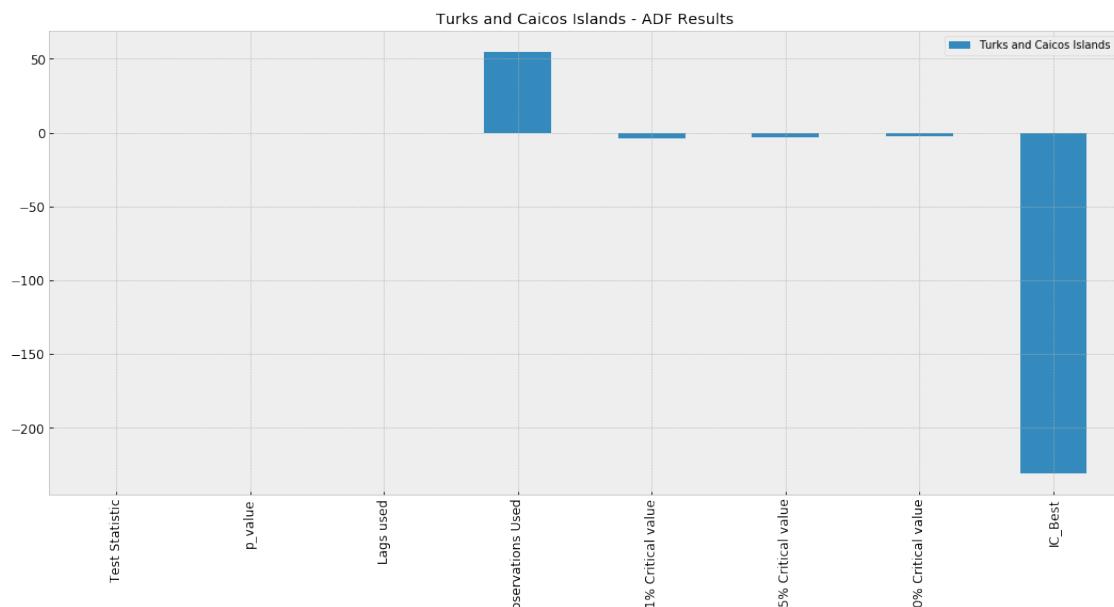


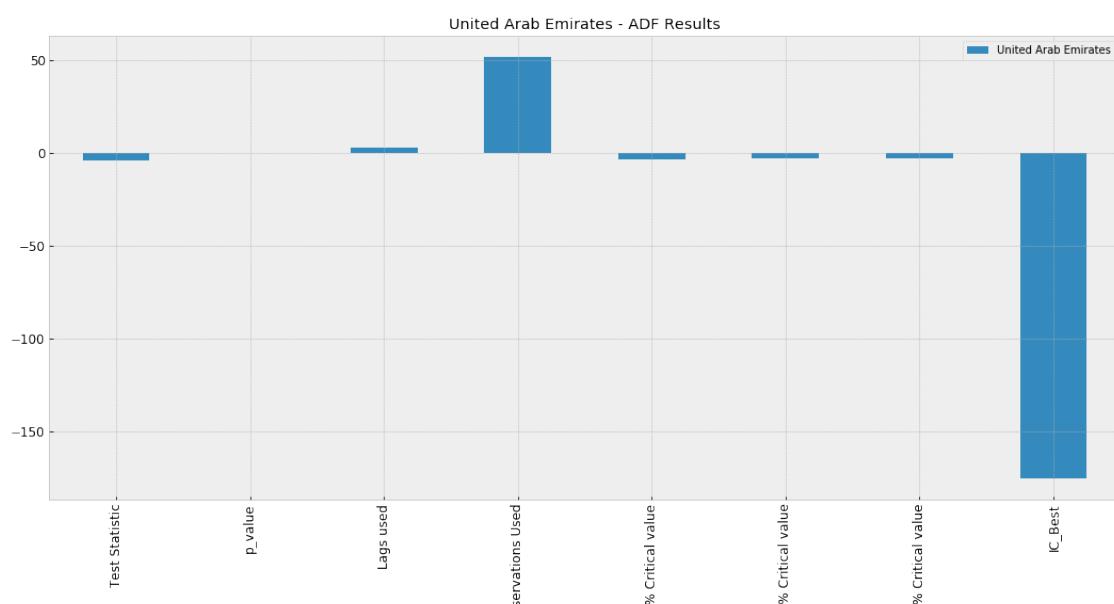
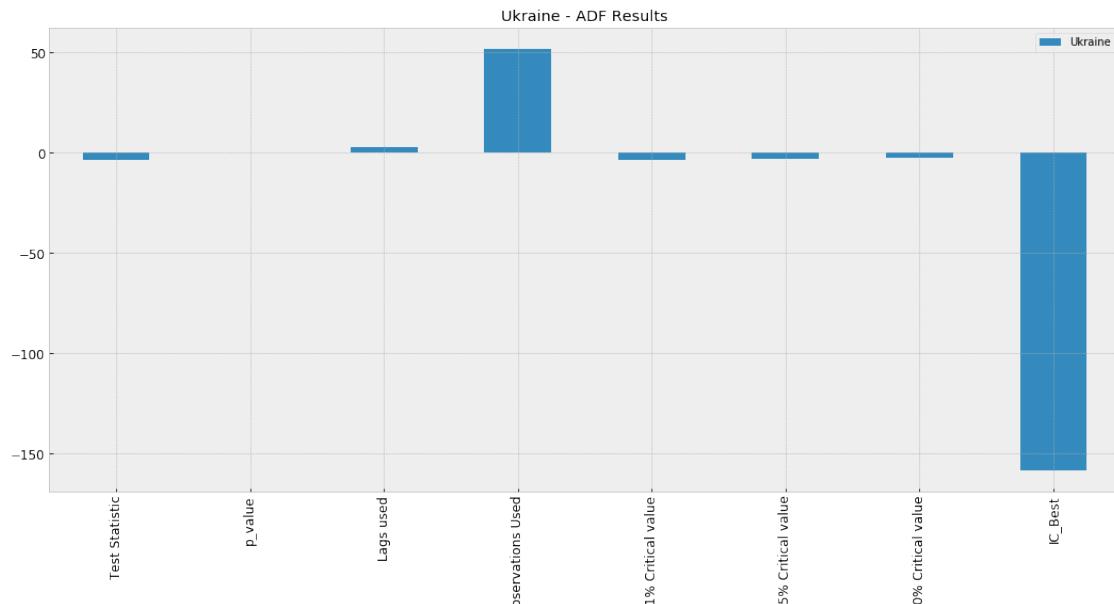


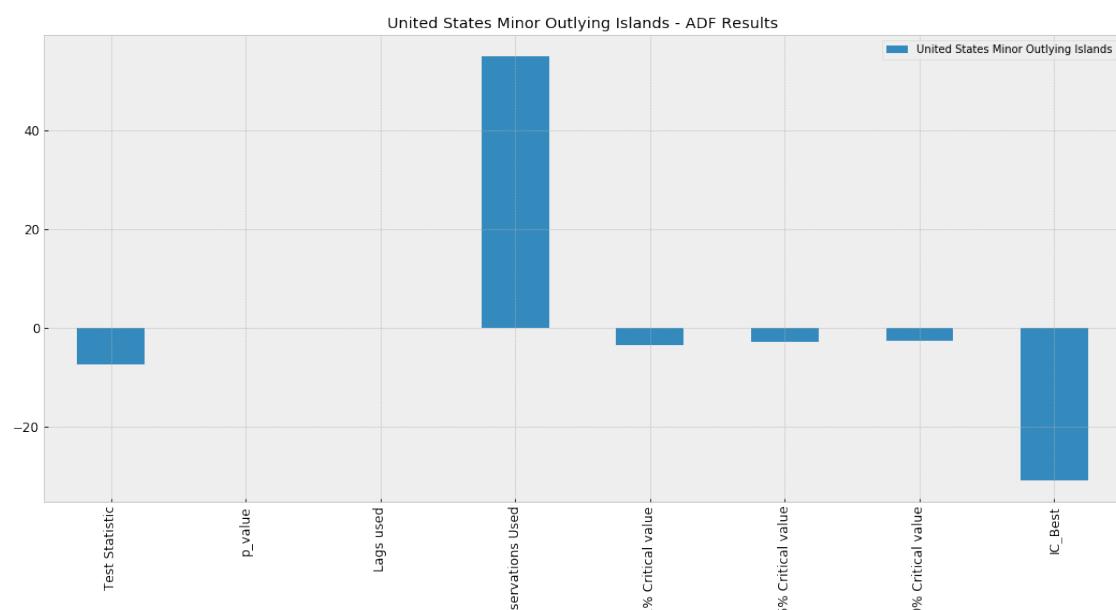
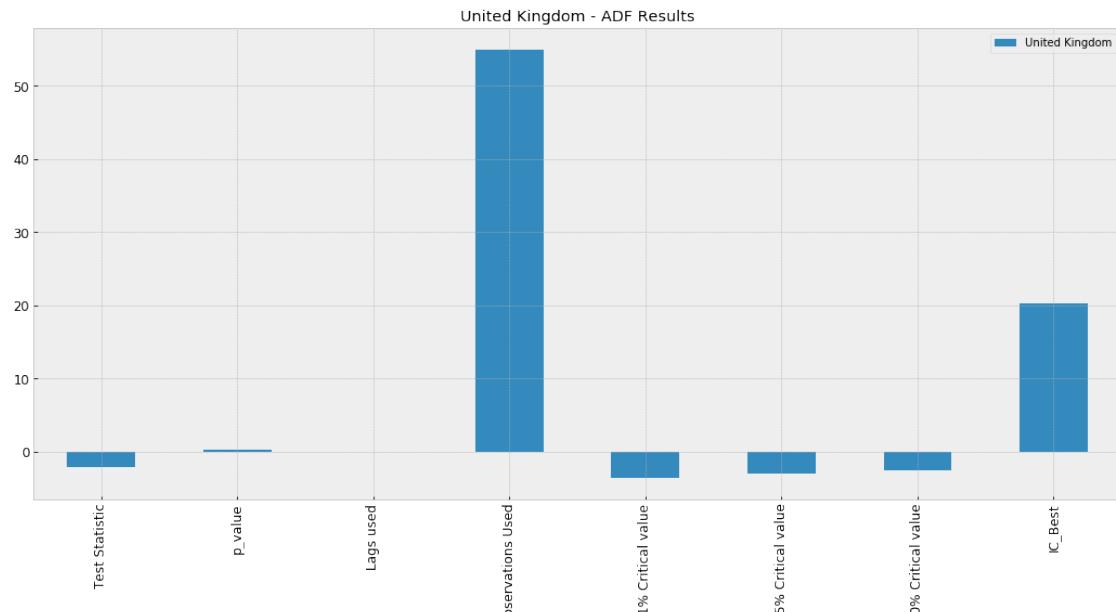


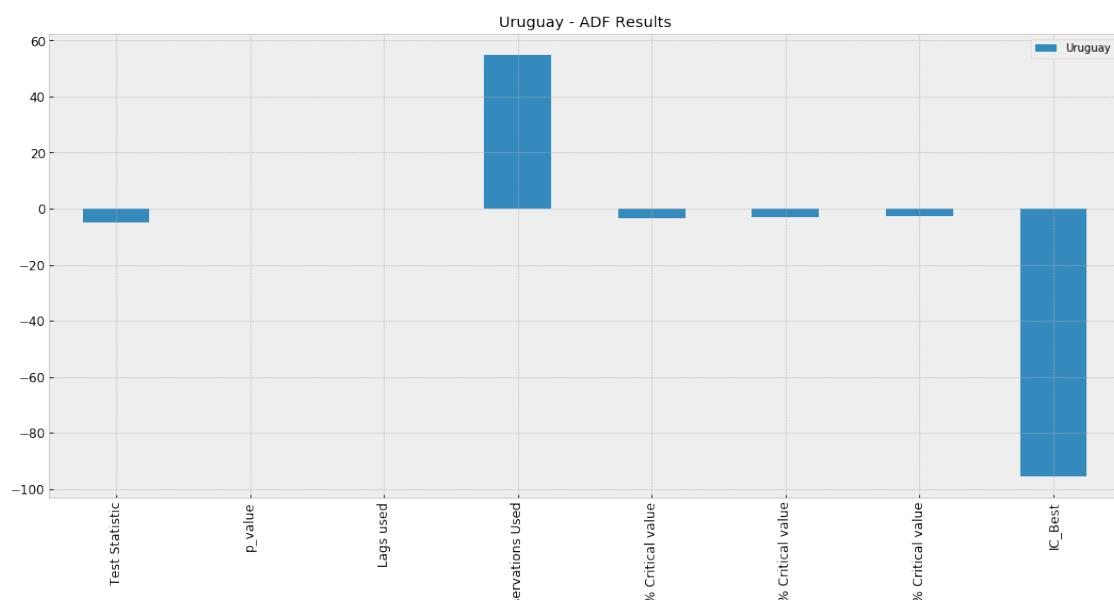
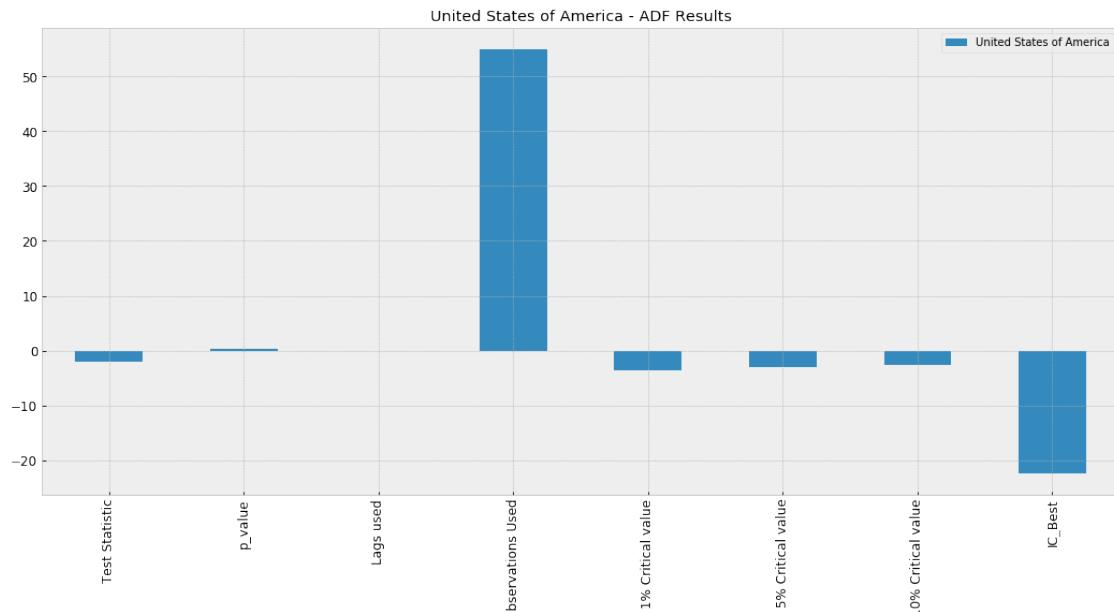


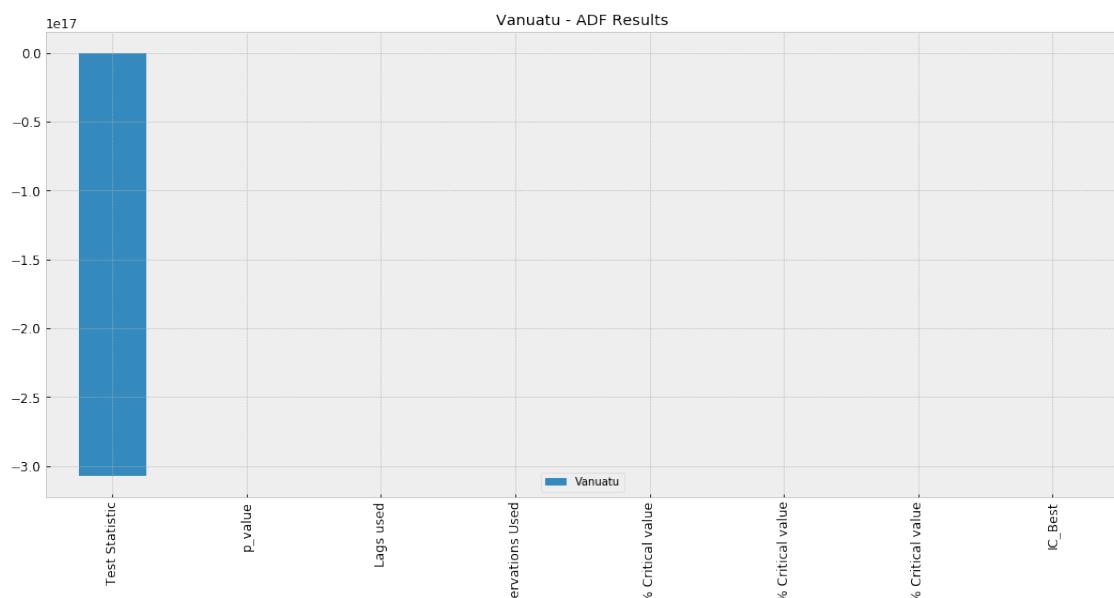
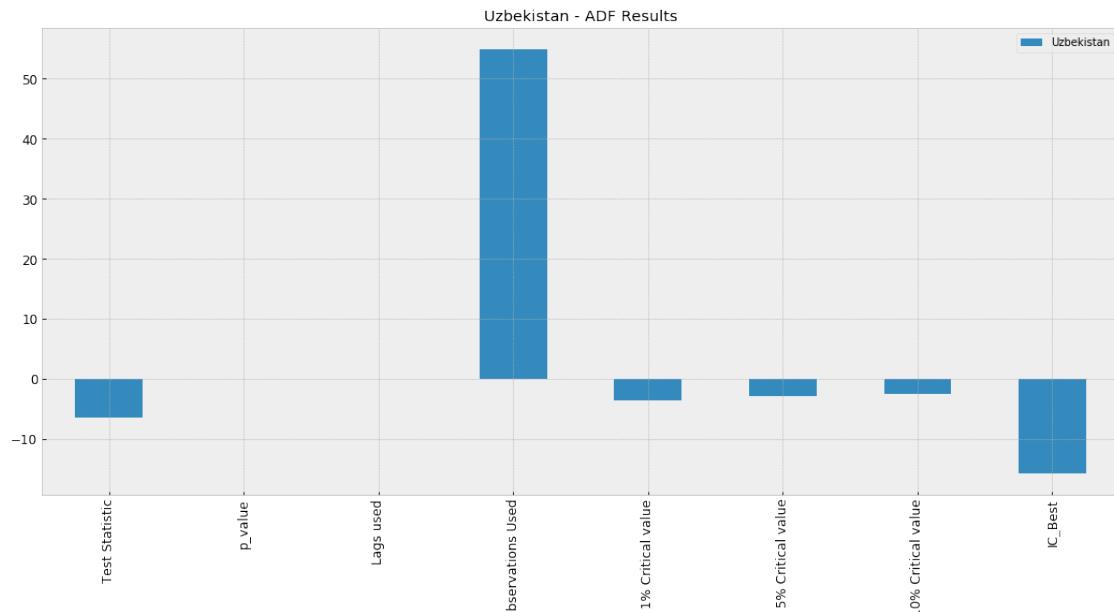


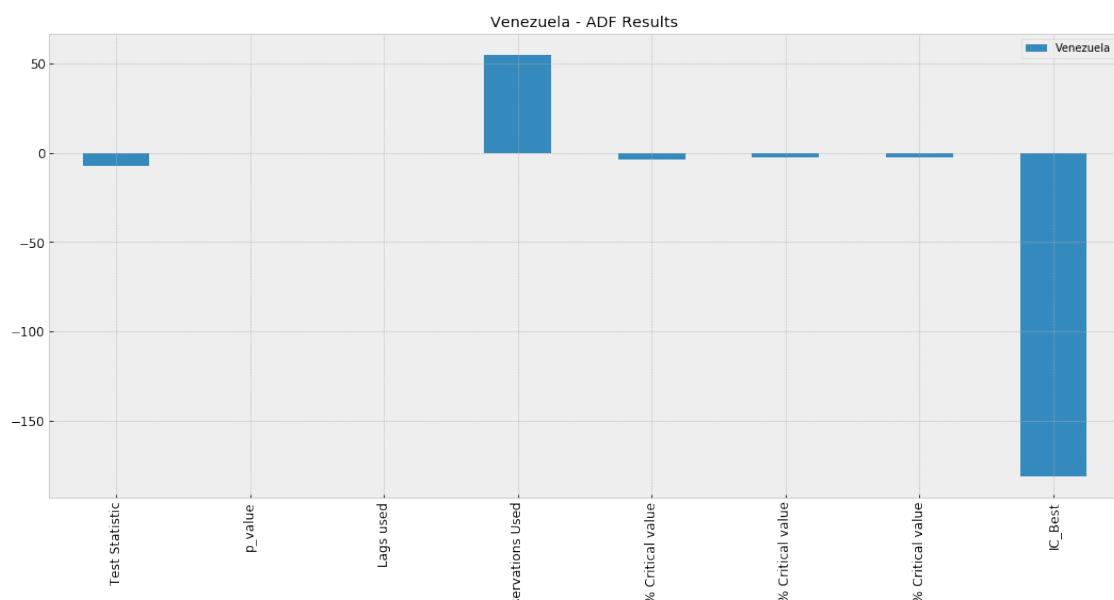
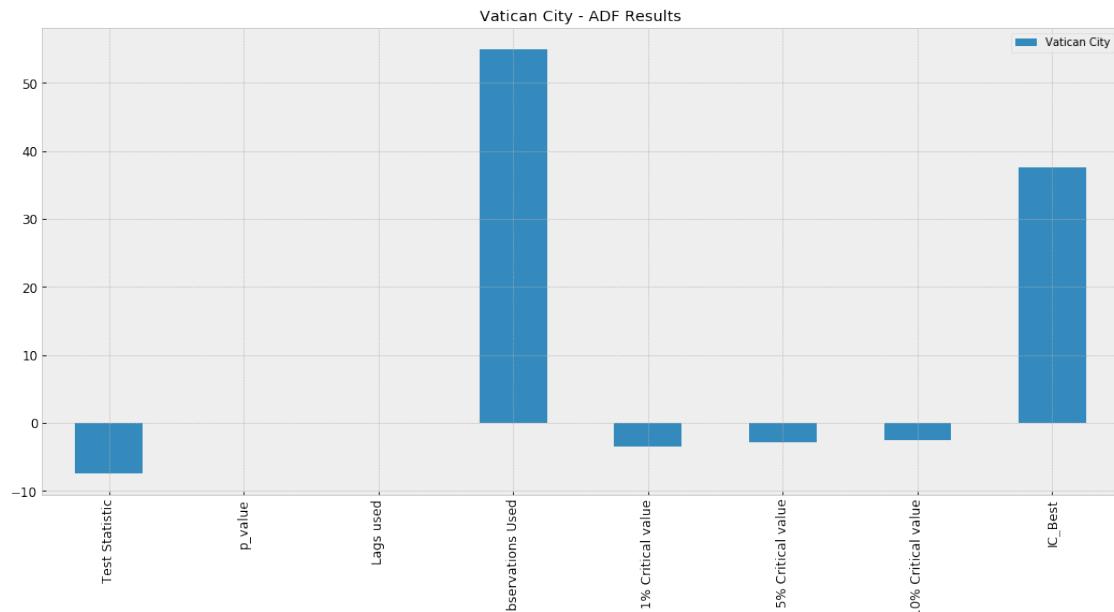


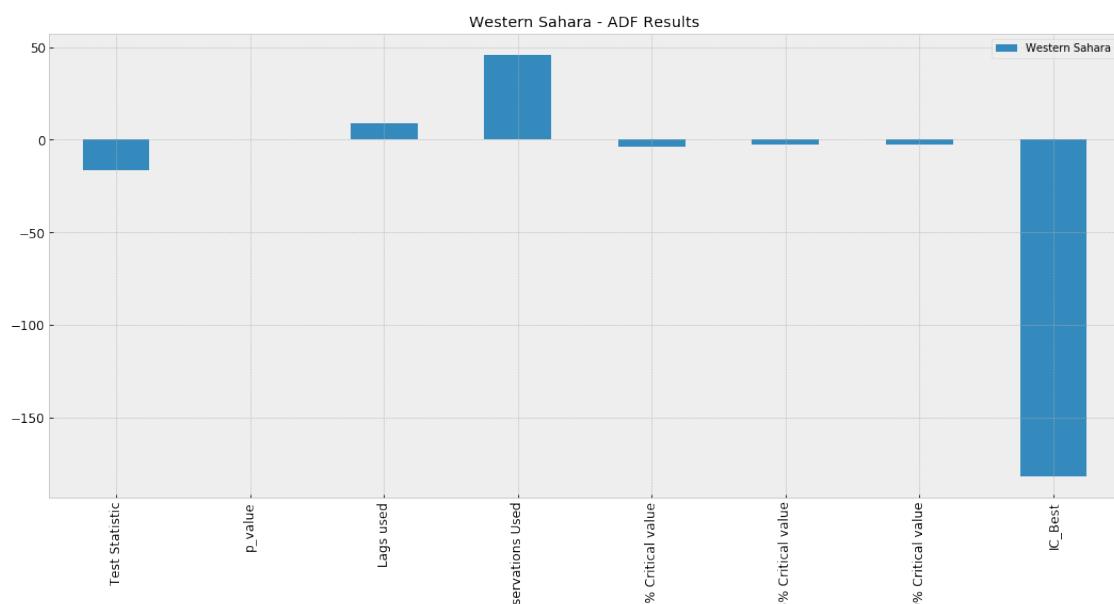
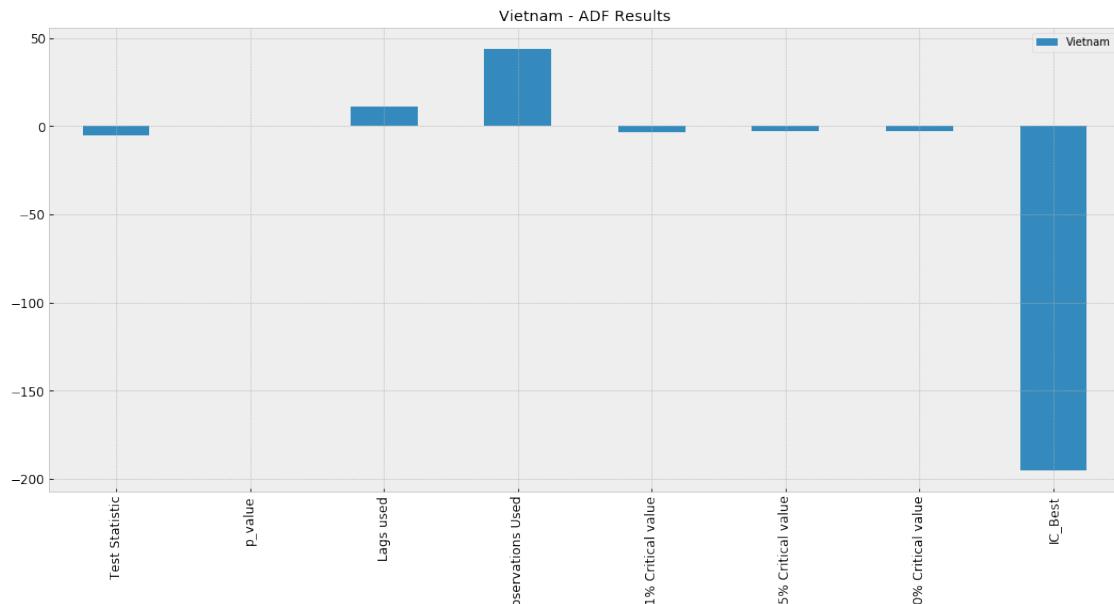


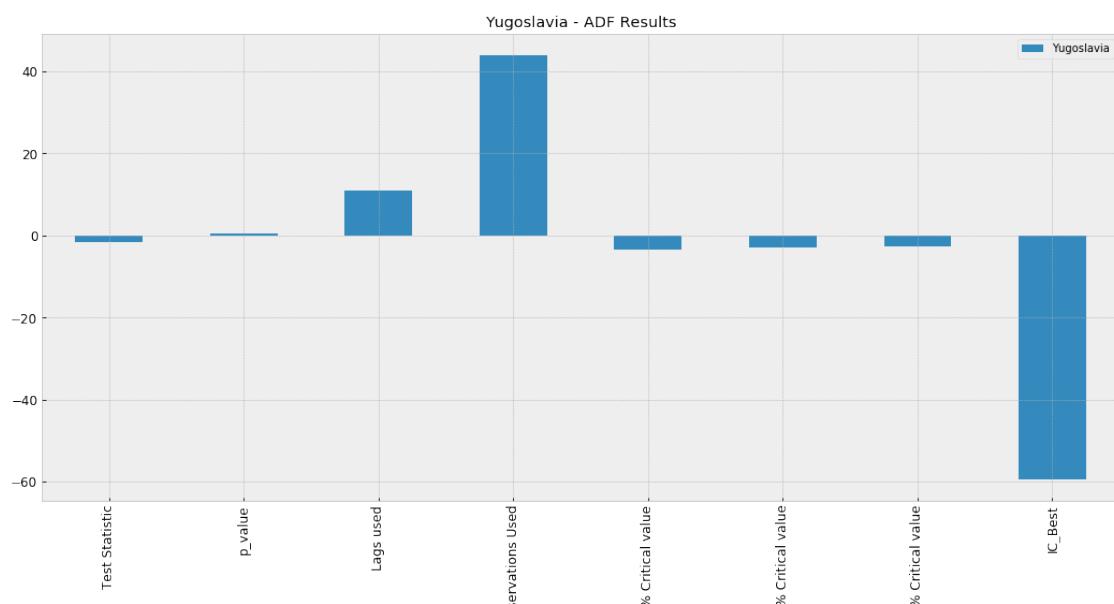
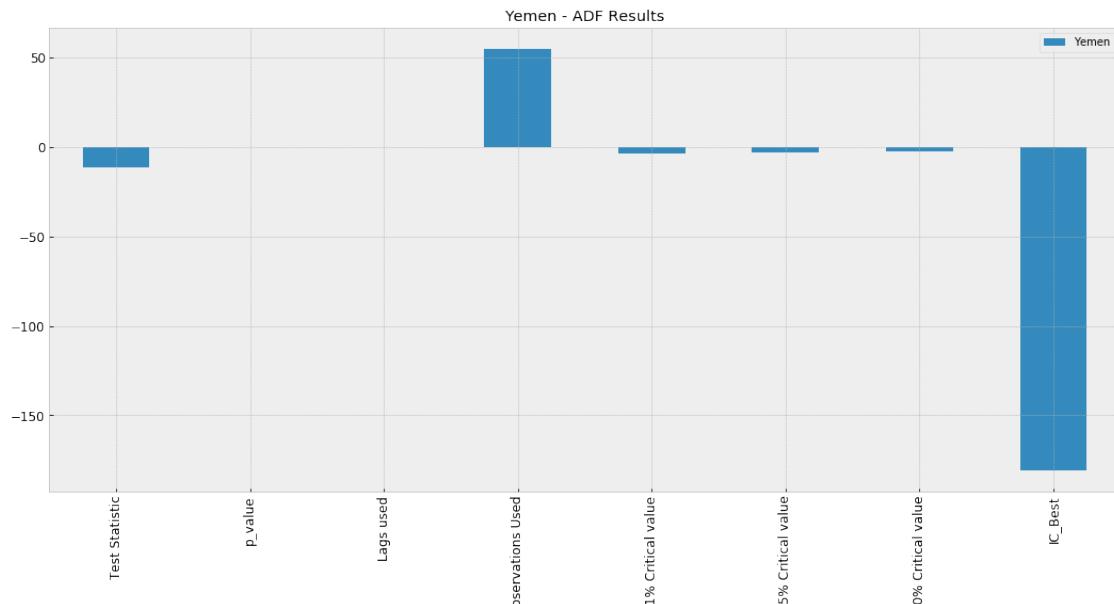


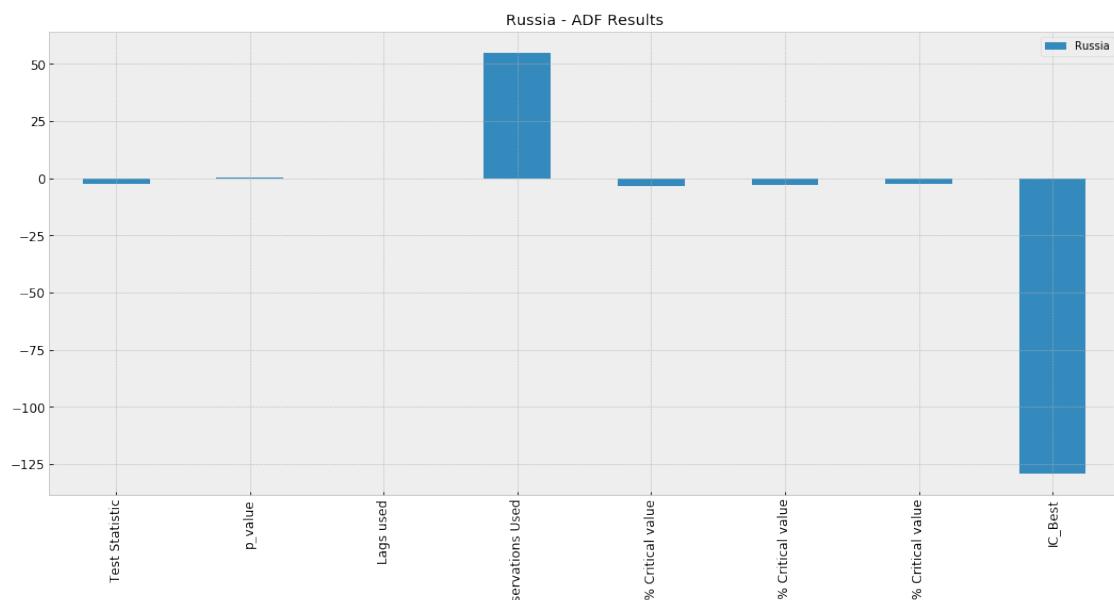
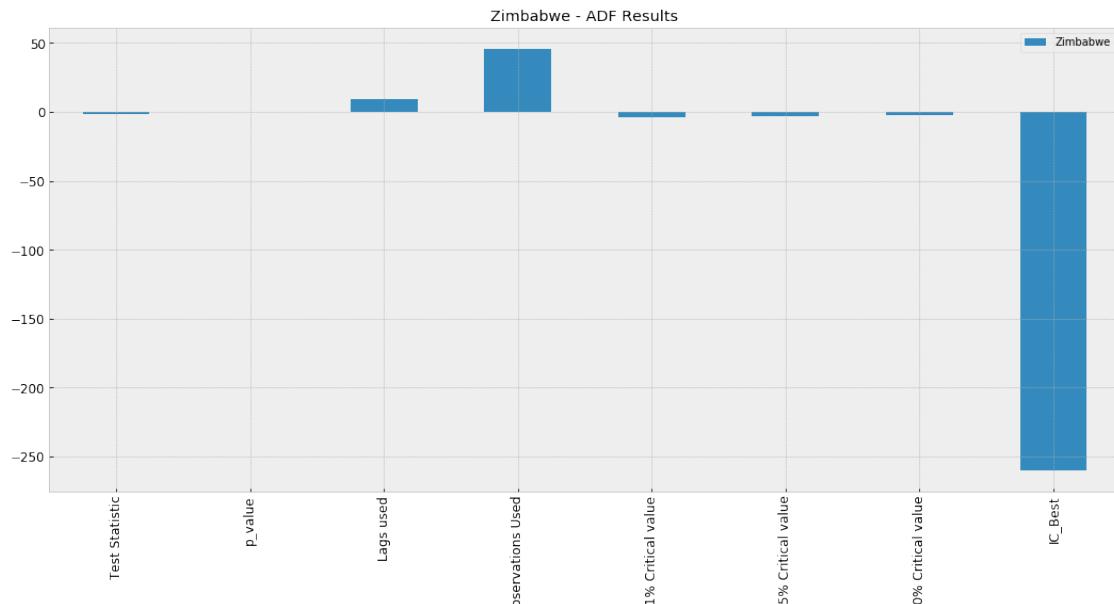












Automate p-value diagnosis: Try this to automate p-value rejection for non-stationary time series

4.2 Making Time Series Stationary

There are 2 major reasons behind non-stationaruty of a TS:

1. **Trend** – varying mean over time

2. **Seasonality** – variations at specific time-frames. eg people might have a tendency to buy cars in a particular month because of pay increment or festivals.

Our dataset had no seasonality, but suffers from a downward and an upward trend, which needs to be corrected in order for forecasting to be possible

[Other sources to check](#)

4.2.1 Transformations

- We can apply transformation which penalize higher values more than smaller values
- These can be taking a log, square root, cube root

Log Scale Transformation