Recent COVID-19 events have placed an emphasis on "flattening the curve" so that the number of cases does not exceed the capacity to handle those cases

The purpose of this is to provide a time-series forecast model to predict if the number of positives a country/territory will receive within the next number of days will increase or decrease

Models used:

> AutoRegressive Integrated Moving Average (ARIMA) used as a baseline and for feature selection, referenced a guide from Jason Brownlee: https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/ (https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/)

> Multiple classification machine learning algorithms from scikit learn to determine if the number will go up or down

Initialize some things

In [1]:
```
import pandas as pd
from datetime import date
from matplotlib import pyplot as plt
from statsmodels.tsa.vector_ar.var_model import VAR
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.arima_model import ARIMA
from math import sqrt
from math import log
from tabulate import tabulate
import time
import pickle
import warnings
import seaborn as sns
import numpy as np
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree.export import export_text
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import cross_val_score
from statistics import stdev as std
from IPython.display import display, clear_output
import random
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
class Timer:
    def __init__(self):
        self.start = time.time()

    def restart(self):
        self.start = time.time()

    def get_new_time(self):
        value = time.time() - self.start
        self.restart()
        return value
```

In [3]:
```
days_missing = 2 # Threshold for the maximum consequitive days a country can be r
days_total = 20 # Threshold for the minimum number of days required from a countr
cases_max = 20 # Threshold for maximum number of cases recorded in a day to accep

run_autocorrelation = True # runs autocorrelation to look at correlated number of

run_ARIMA = False # runs ARIMA to explore data, this takes 2-5 hours but can be p
outpath_ARIMA = 'ARIMA_results' # path to output pickle file of results from ARI
read_ARIMA_results = True # read ARIMA results from previous runs
input_ARIMA = ['ARIMA_results', 'ARIMA_results_par'] # paths to input pickle fil

test_split = .10 # percent to pull out for testing at end
resample = True # Resamples train and test data, if false will read from file oth
output_sample = 'MLA_sample'
input_sample = 'MLA_sample'

run_class = True # runs MLA classification tests
output_class = 'MLA_classification_results' # path to otuput pickle file with res
read_class_results = False # reads MLA classification results from previous runs
input_class = ['MLA_classification_results'] # paths to input pickle files to rea

run_regerss = True # runs MLA regression tests
output_regress = 'MLA_regression_results' # path to otuput pickle file with resu
read_regress_results = False # reads MLA regression results from previous runs
input_regress = ['MLA_regression_results'] # paths to input pickle files to read
```

Read data

In [4]:
```
# read raw covid19 cases data
# https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic
covid19_cases = pd.read_csv('COVID-19_cases_20200320.csv')

# remove all attributes except country and number of cases by date
covid19_cases = covid19_cases[['DateRep', 'Cases', 'Countries and territories']]
covid19_cases.columns = ['Date', 'Cases', 'Country']
covid19_cases['Date'] = pd.to_datetime(covid19_cases.Date)
covid19_cases.head()

# organize by day number
data = covid19_cases.sort_values(by=['Country','Date'])

# fix something so the cases an population .csv file names match up by country na
data['Country'] = data['Country'].str.replace('_', ' ')
data['Country'] = data['Country'].str.replace('CANADA', 'Canada') # Canada was en
data = data.loc[data['Country'] != 'Cote dIvoire']
data = data.loc[data['Country'] != 'Kosovo']
data = data.loc[data['Country'] != 'Cases on an international conveyance Japan']
data.head()
```

Out[4]:

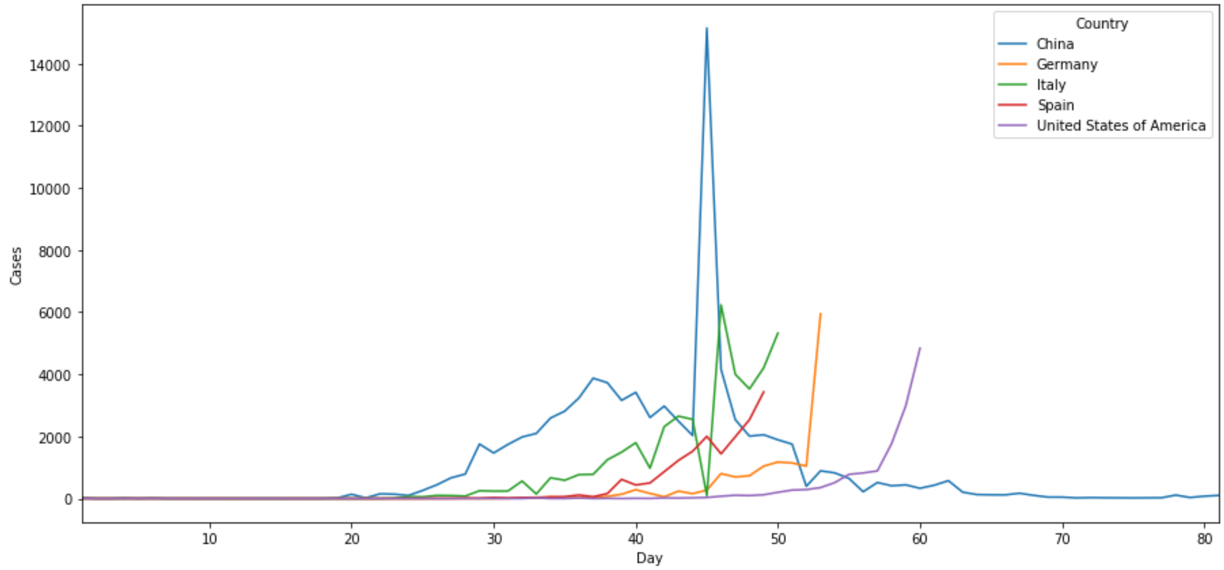
	Date	Cases	Country
70	2019-12-31	0	Afghanistan
69	2020-01-01	0	Afghanistan
68	2020-01-02	0	Afghanistan
67	2020-01-03	0	Afghanistan
66	2020-01-04	0	Afghanistan

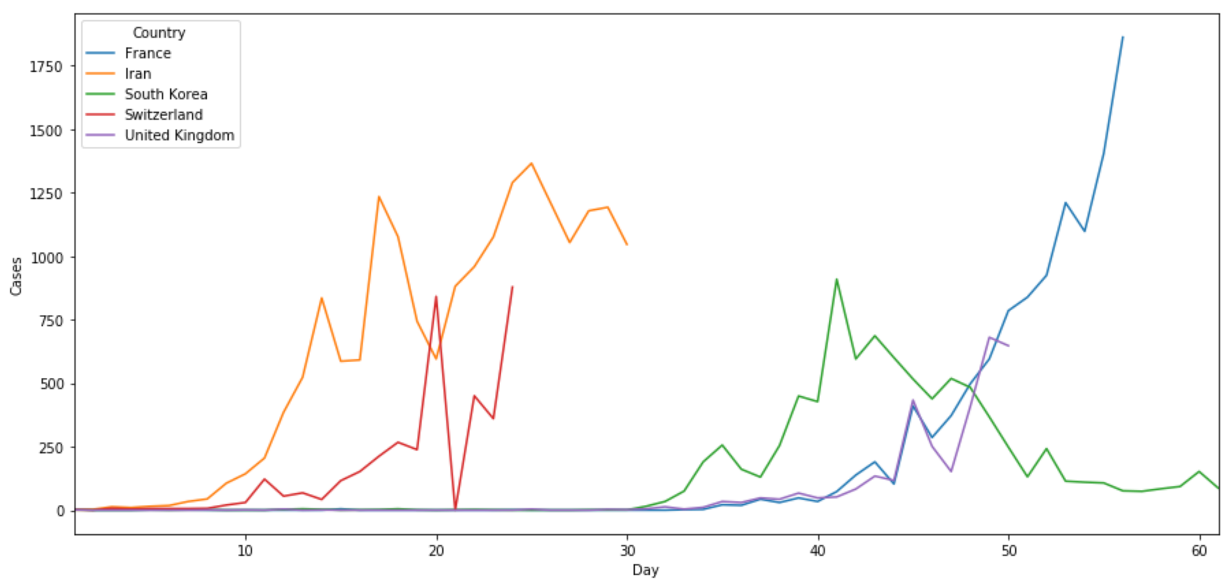
Mungle and Visualize Data

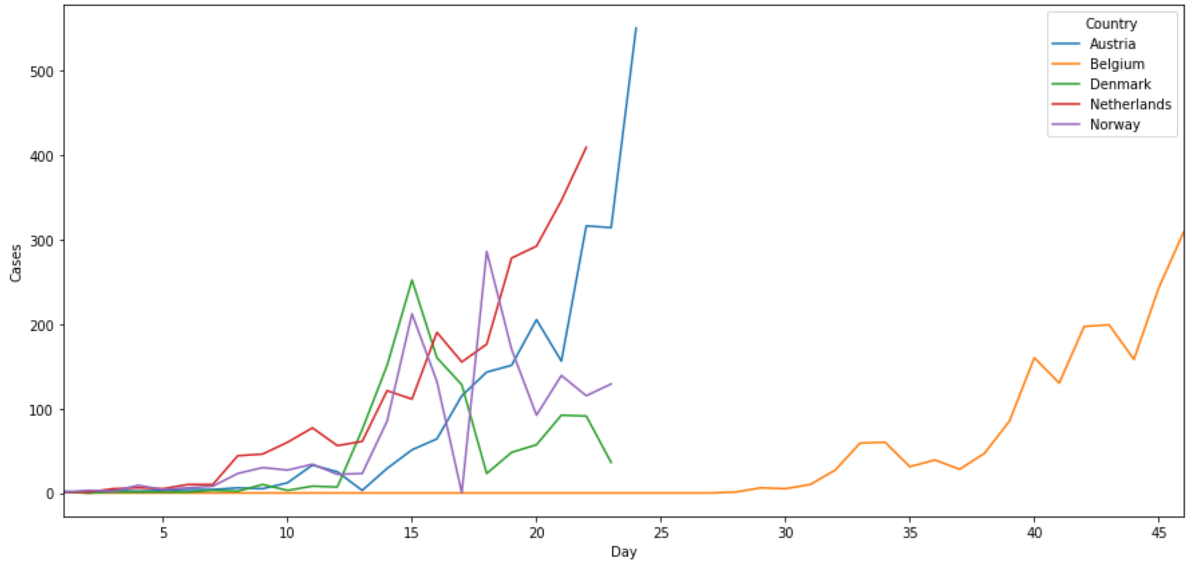
In [5]:
```
def plot5(data, countries, xlabel, ylabel):
    lastCut = 0
    for i in range(0, len(countries), 5):
        lastCountry = ''
        nCountries = 0
        thisCut = lastCut
        for i in range(lastCut, len(data), 1):
            thisCountry = str(data.at[i,'Country'])
            if thisCountry not in lastCountry or i == len(data) - 1:
                nCountries += 1
            lastCountry = thisCountry
            if nCountries > 5:
                break
            thisCut += 1
        fig, ax = plt.subplots(figsize=(15,7))
        data.iloc[lastCut:thisCut].groupby(['Day', 'Country']).sum()['Cases'].uns
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        lastCut = thisCut
```

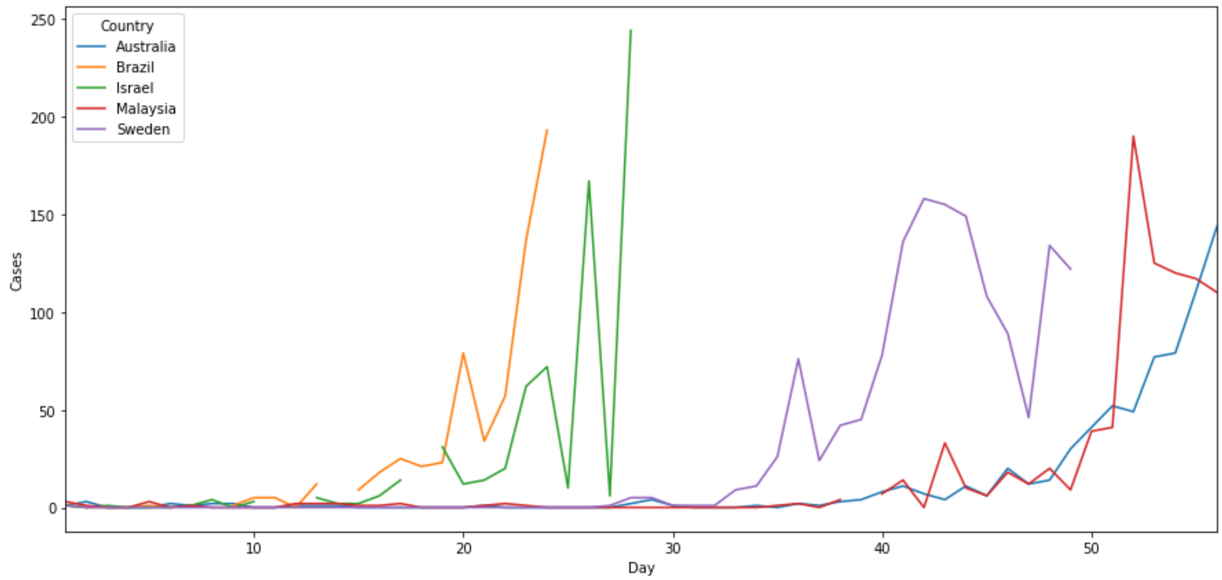
In [6]:
```
# re-rank date by day# starting with first case
    # remove country if more than days_missing in a row day is missing
    # only count if 10 days exist
    # sort by max number of cases
reranked = []
tosort = {}
lastCountry = ''
lastDate = ''
dayNum = 1
firstCase = False
temp = []
skip = False
maxCases = 0
maxes = []
for index, row in data.iterrows():
    thisDate = row[0]
    cases = row[1]
    country = row[2]

    if country not in lastCountry:
        if not skip and len(temp) >= days_total and maxCases > cases_max:
            tosort[lastCountry] = [temp, maxCases]
            maxes.append(maxCases)
        temp = []
        skip = False
        firstCase = False
        dayNum = 1
        maxCases = 0

    if cases > 0 and not firstCase:
        lastDate = thisDate
        firstCase = True

    if firstCase:
        if (thisDate - lastDate).days > days_missing + 1:
            skip = True
        maxCases = max(cases, maxCases)
        dayNum += (thisDate - lastDate).days
        if dayNum < 0:
            break
        temp.append([dayNum, cases, country])

    lastDate = thisDate
    lastCountry = country
if not skip and len(temp) >= days_total and maxCases > cases_max:
    tosort[lastCountry] = [temp, maxCases]
    maxes.append(maxCases)

# sort
while len(maxes) > 0:
    thisMax = max(maxes)
    pops = []
    for country in tosort:
        rows = tosort[country][0]
        maxCases = tosort[country][1]
        if thisMax == maxCases:
```

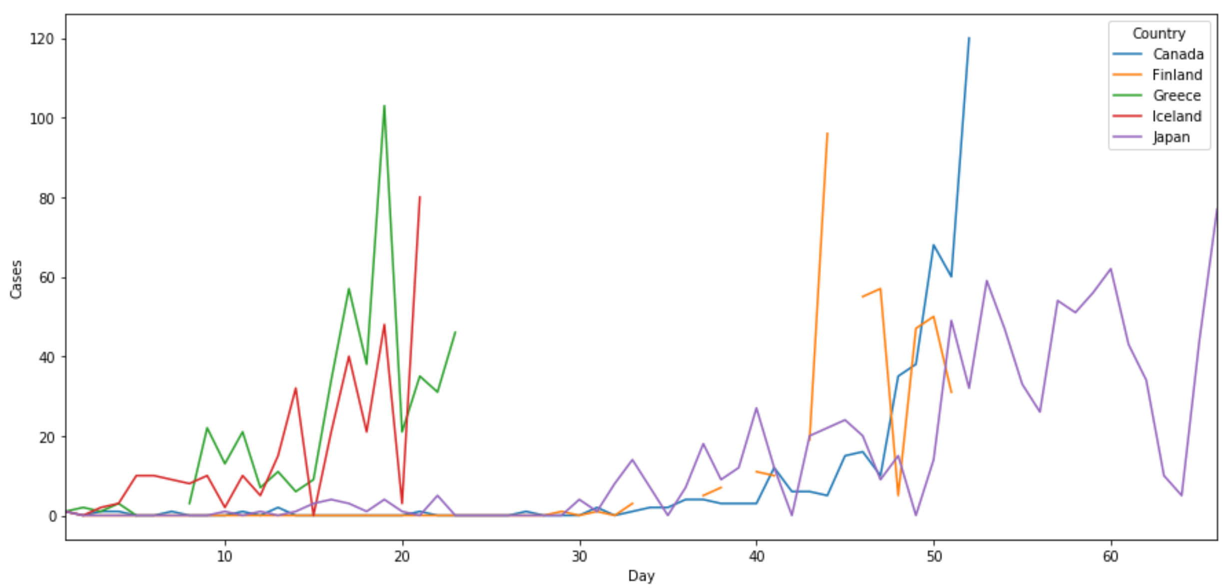
```
            reranked = reranked + rows
            pops.append(country)
    for country in pops:
        tosort.pop(country)
    maxes = [x for x in maxes if x < thisMax]

data = pd.DataFrame(reranked, columns=['Day', 'Cases', 'Country'])
countries = data['Country'].unique()

# plot 5 at a time
plot5(data, countries, 'Day', 'Cases')
```

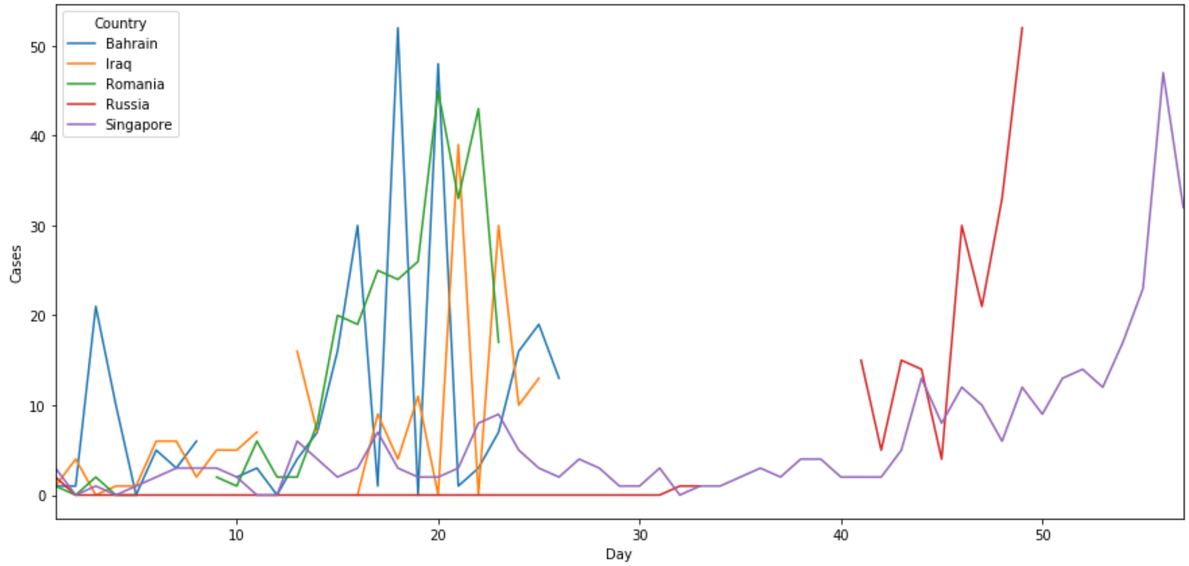


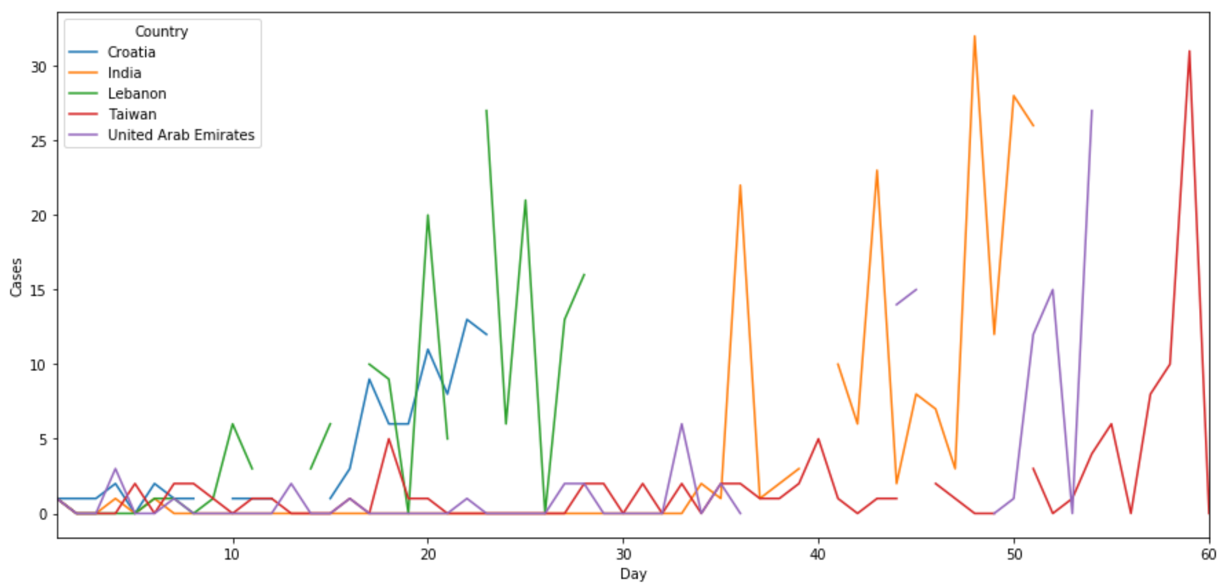



Country
Austria
Belgium
Denmark
Netherlands
Norway
Cases
Day


Country
Australia
Brazil
Israel
Malaysia
Sweden
Cases
Day


Country
Canada
Finland
Greece
Iceland
Japan
Cases
Day

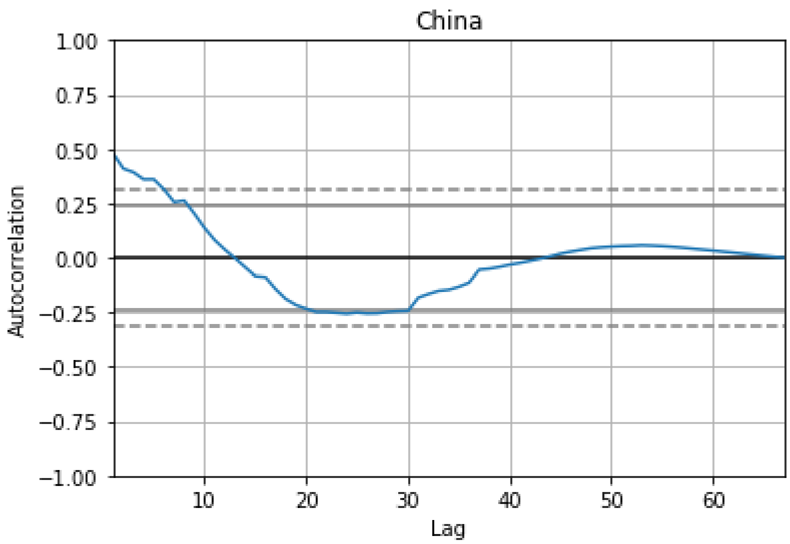


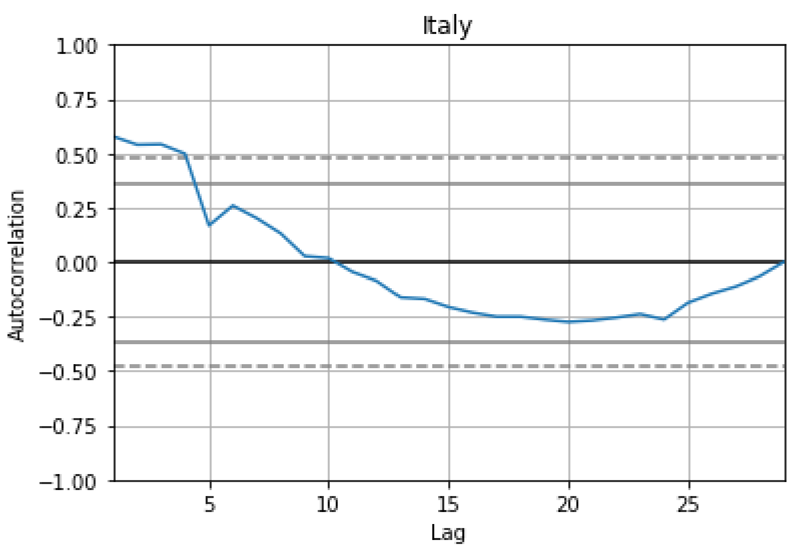


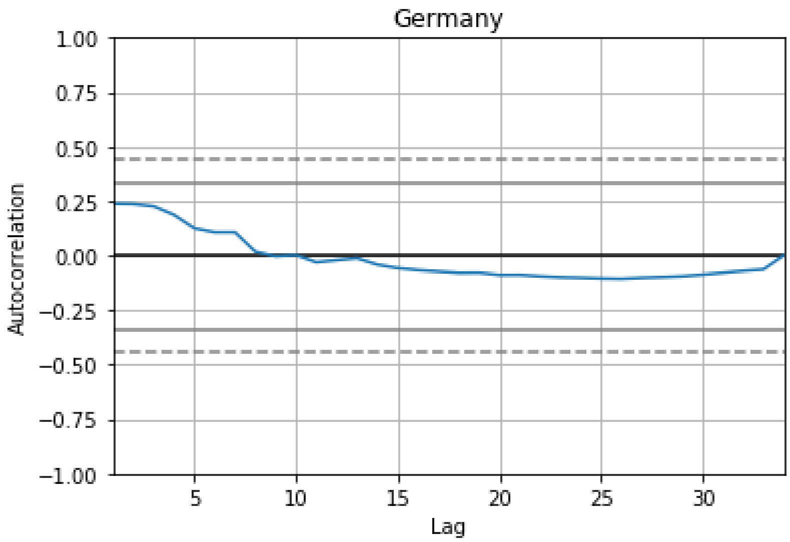
In [7]:
```
# split into dictionary and remove 0s
covid19 = {}
for i in range(len(data)):
    country = data.at[i,'Country']
    if country not in covid19:
        covid19[country] = []
    if data.at[i,'Cases'] > 0:
        covid19[country].append(data.at[i,'Cases'])
```

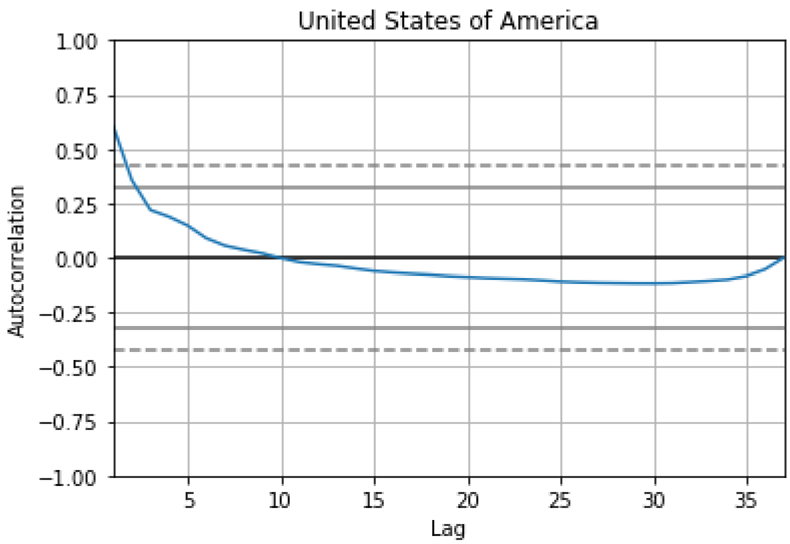
FEATURE SELECTION

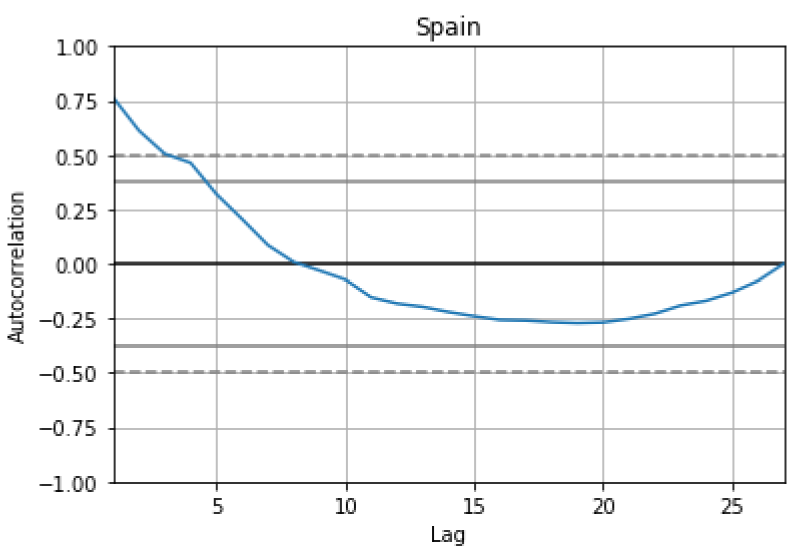
In [8]:
```
# plot autocorrelation for number of lags in time-series
if run_autocorrelation:
    for country in countries:
        autocorrelation_plot(covid19[country])
        plt.title(country)
        plt.show()
```

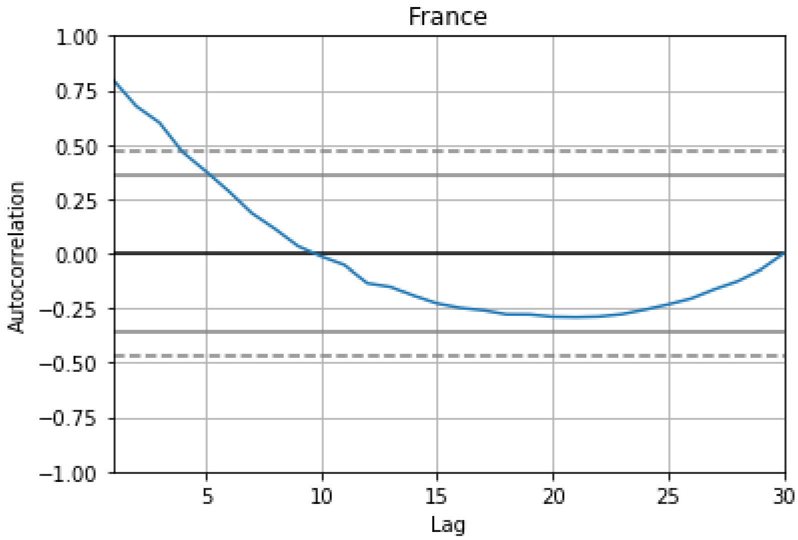


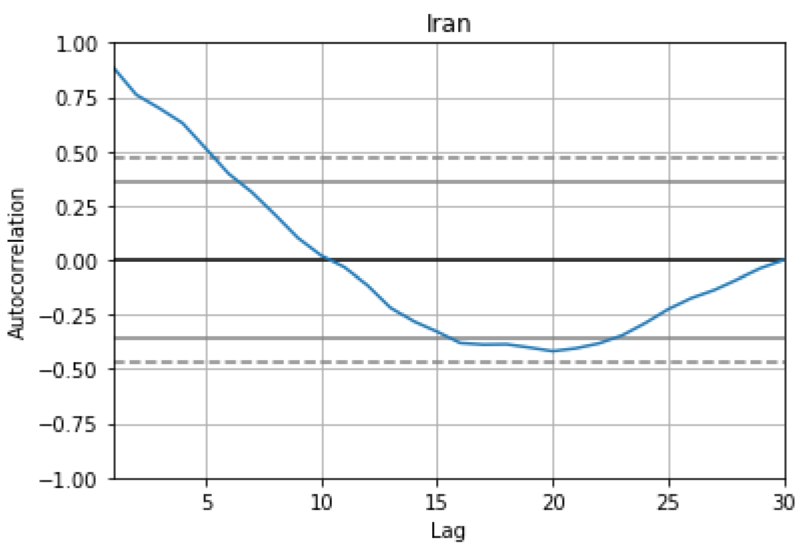


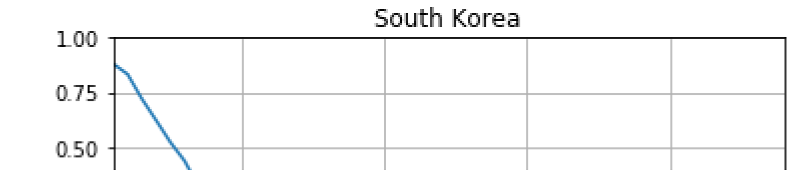

Germany
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25
30

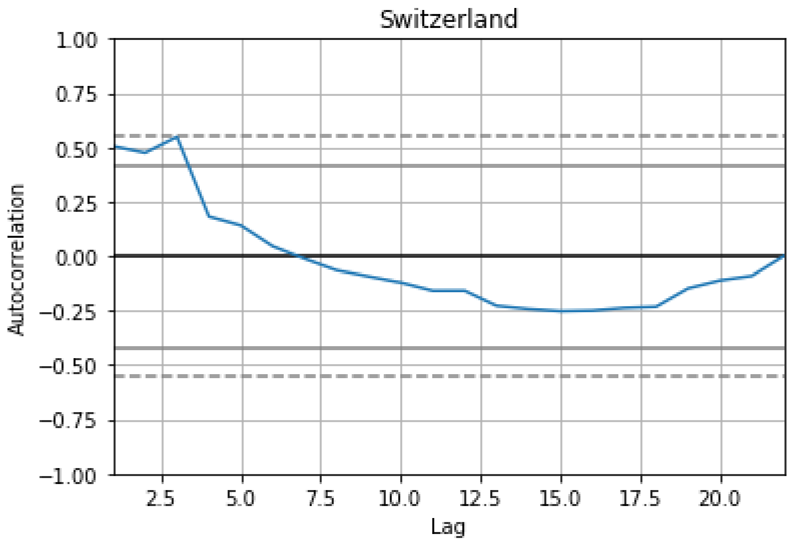

United States of America
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25
30
35

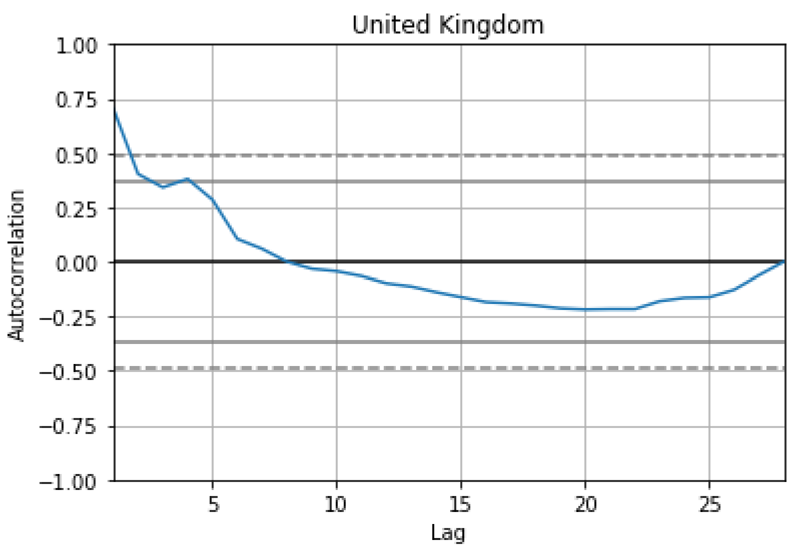

Spain
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25


France
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
5
10
15
20
25
30
Lag

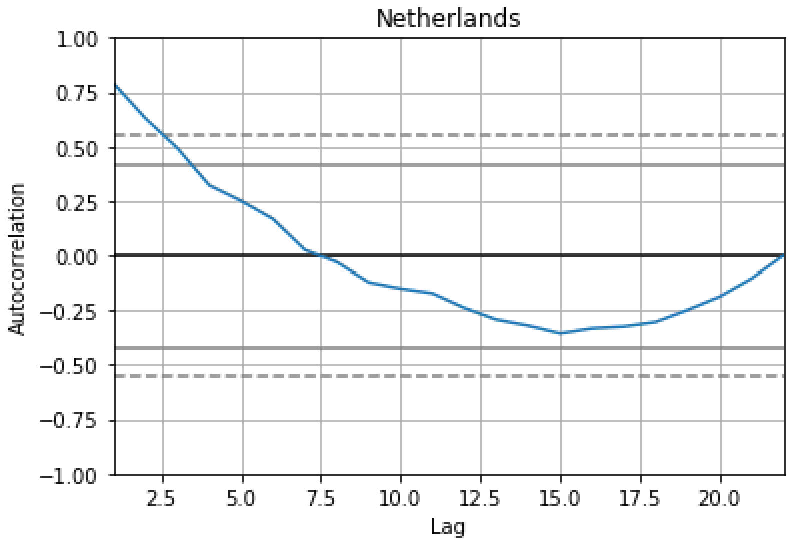

Iran
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
5
10
15
20
25
30
Lag

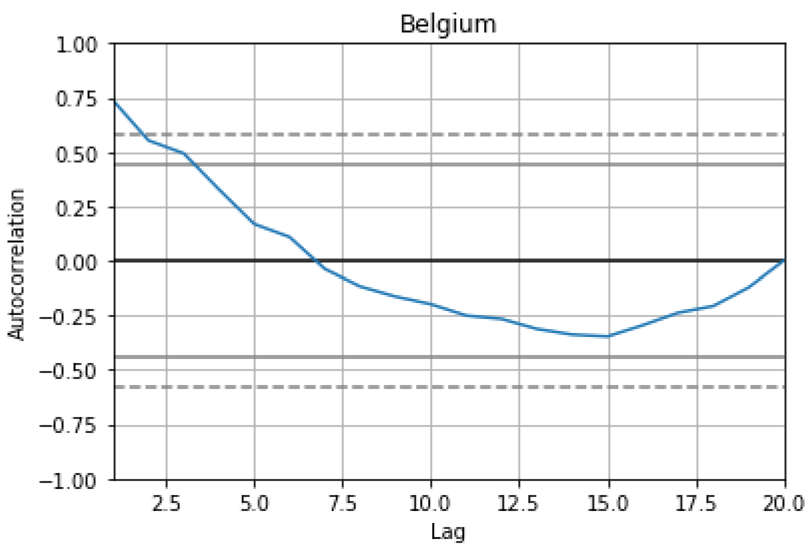

South Korea
1.00
0.75
0.50

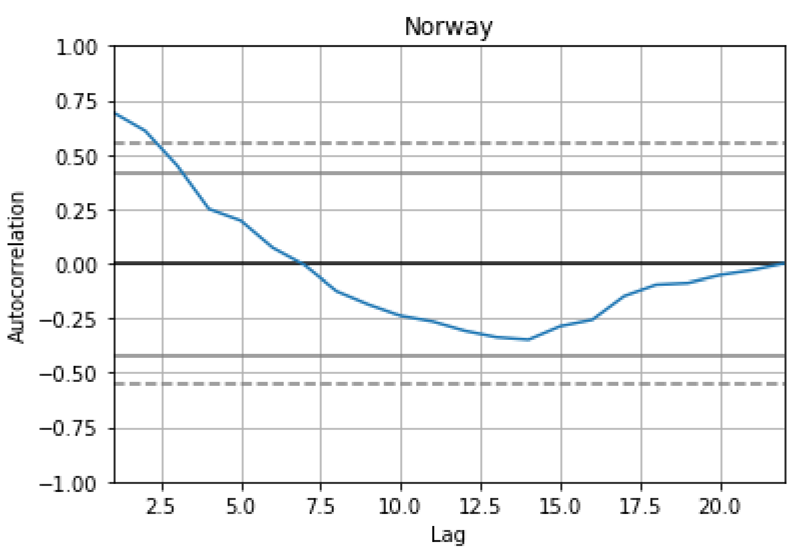

Switzerland
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

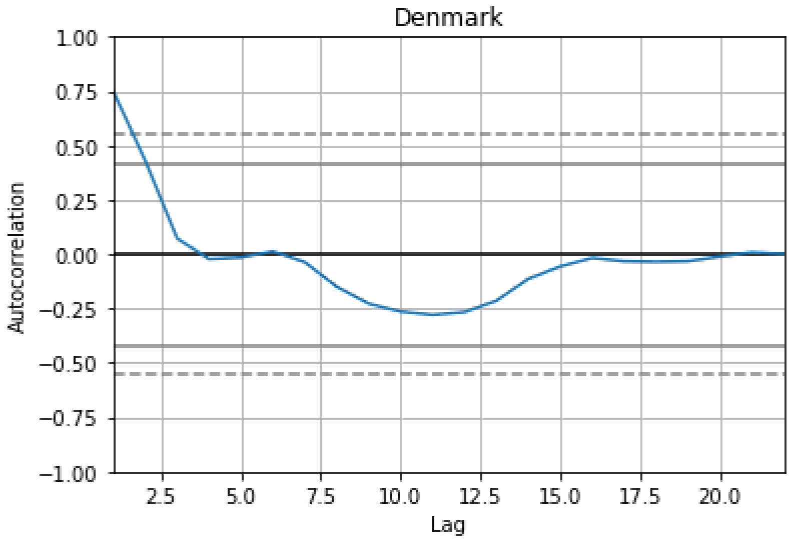

United Kingdom
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25

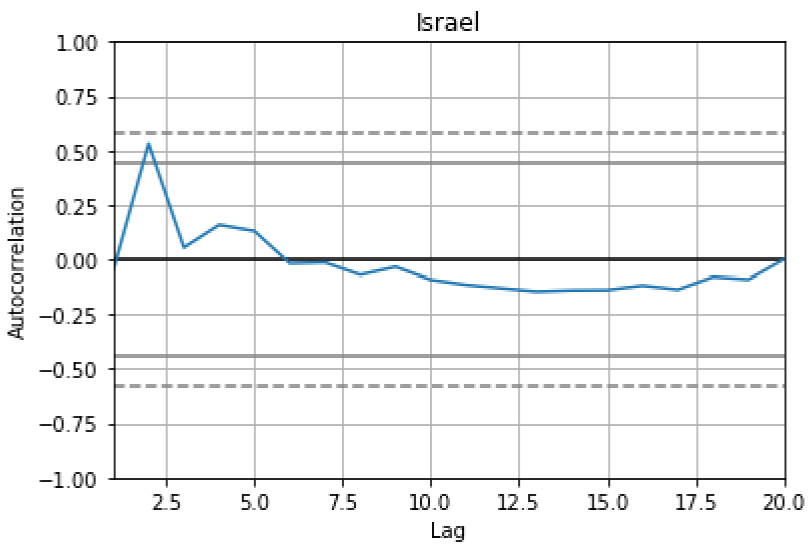

Austria
1.00

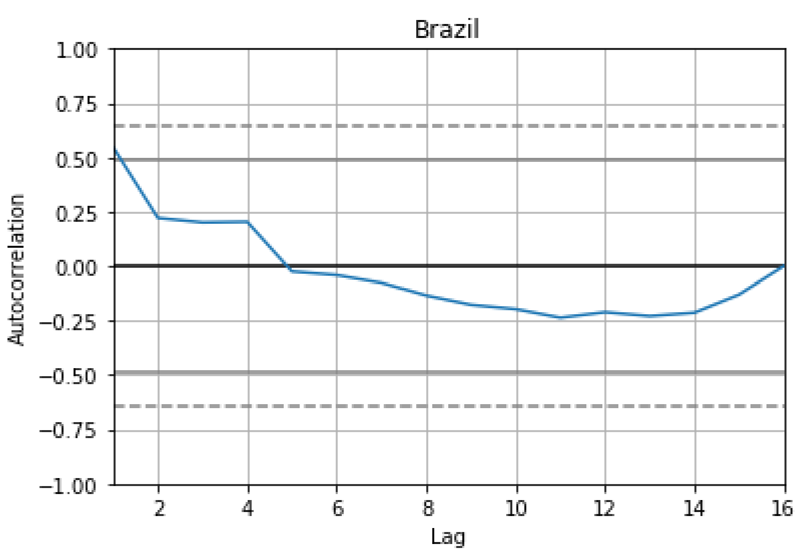

Netherlands
Autocorrelation
Lag

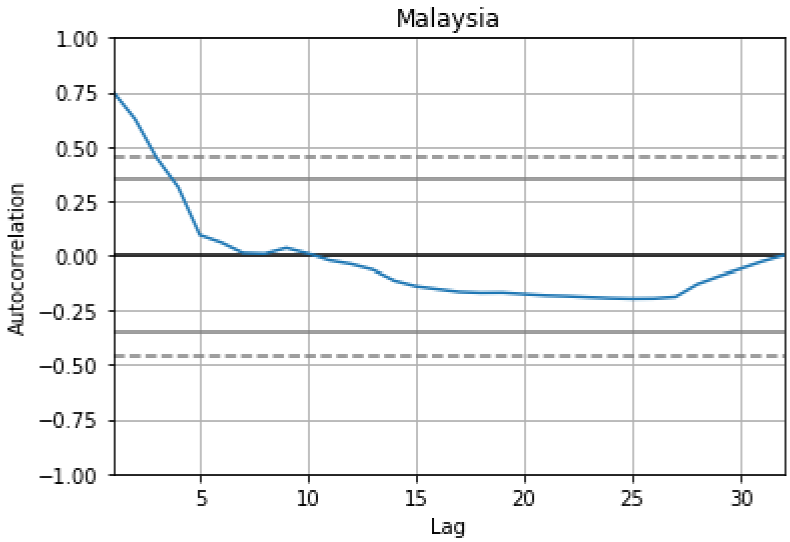

Belgium
Autocorrelation
Lag

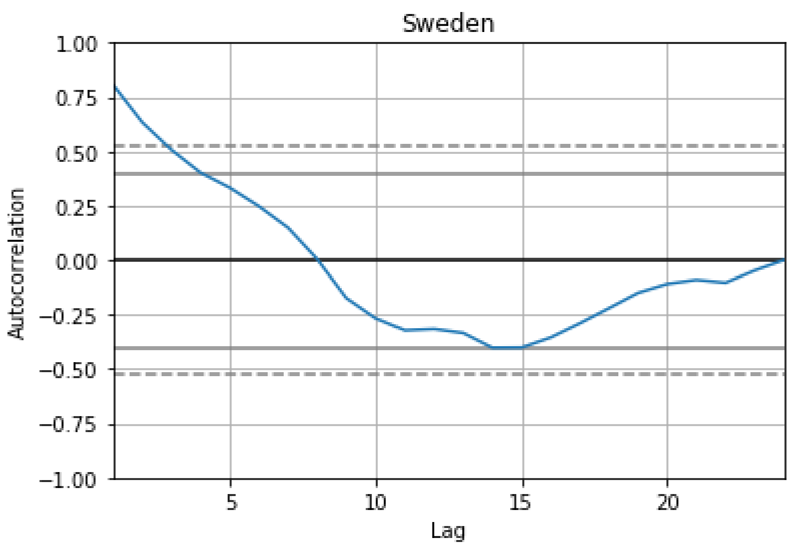

Norway
Autocorrelation
Lag


Denmark
Autocorrelation
Lag

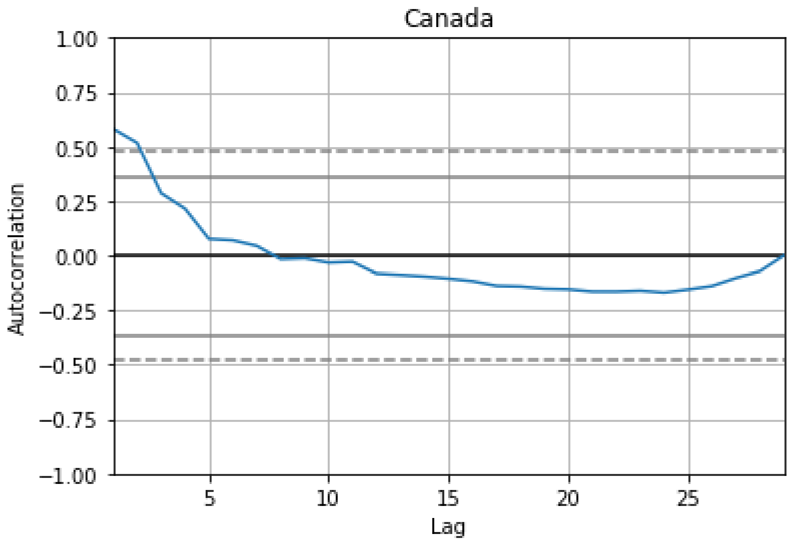

Israel
Autocorrelation
Lag

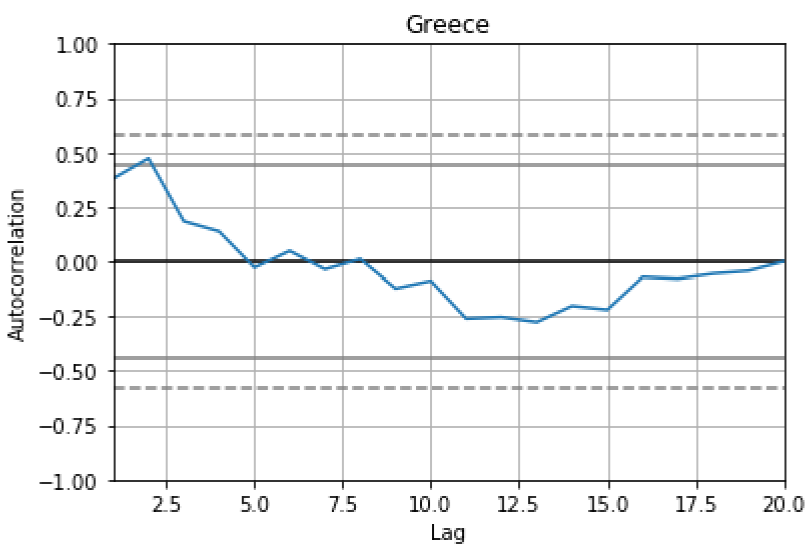

Brazil
Autocorrelation
Lag

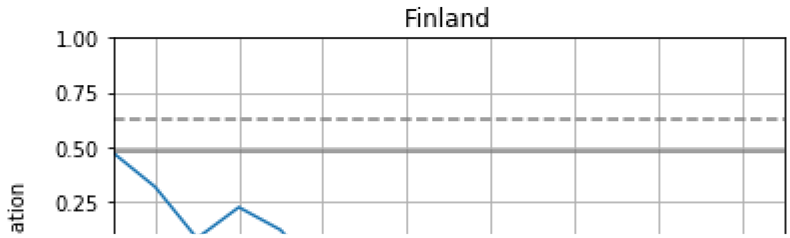

Malaysia
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
5
10
15
20
25
30
Lag

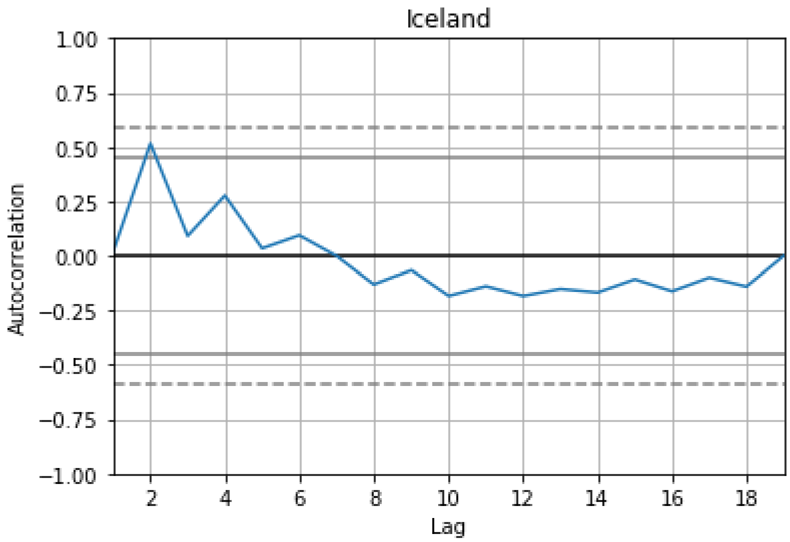

Sweden
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
5
10
15
20
Lag

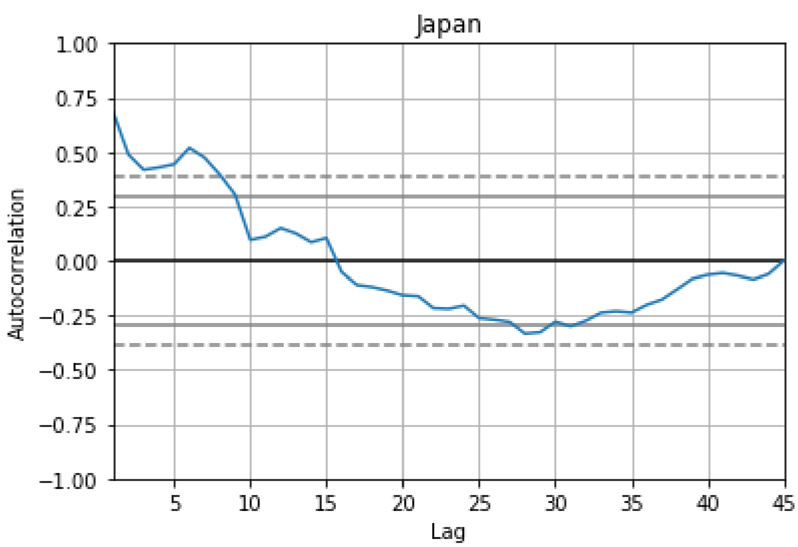

Australia
1.00

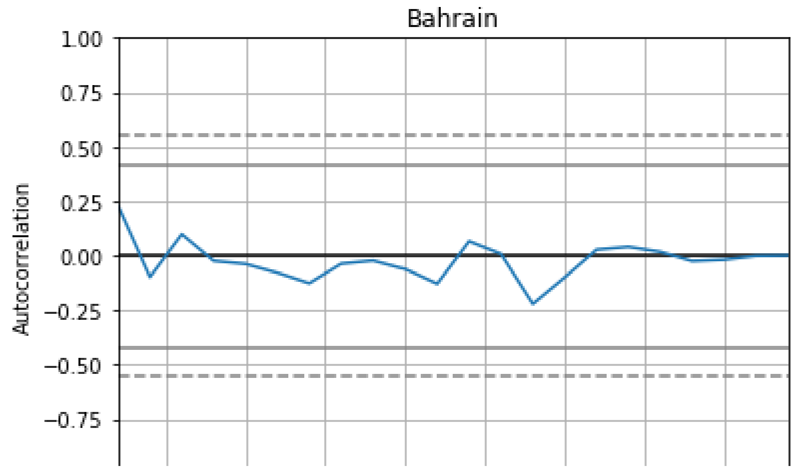

Canada
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25

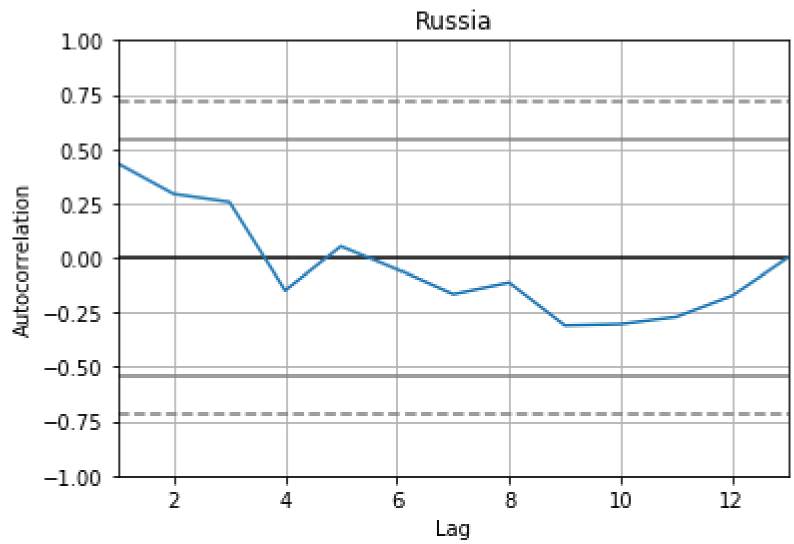

Greece
Autocorrelation
Lag
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

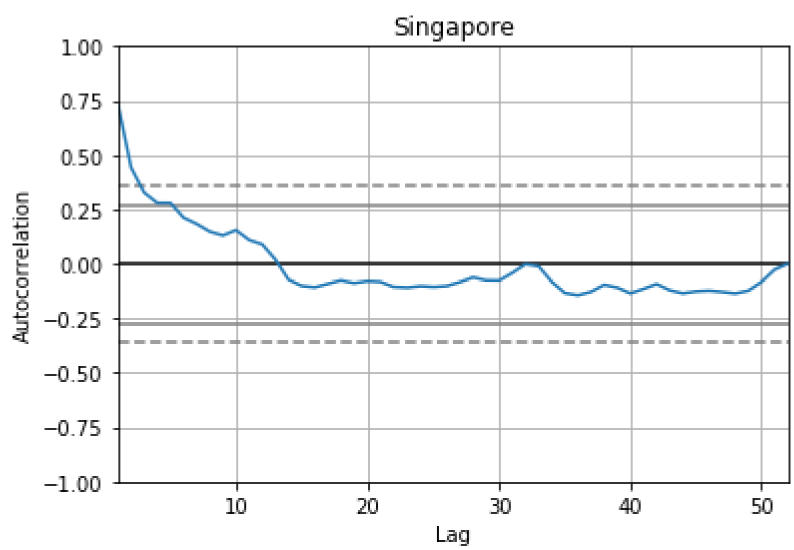

Finland
1.00
0.75
0.50
0.25
ation

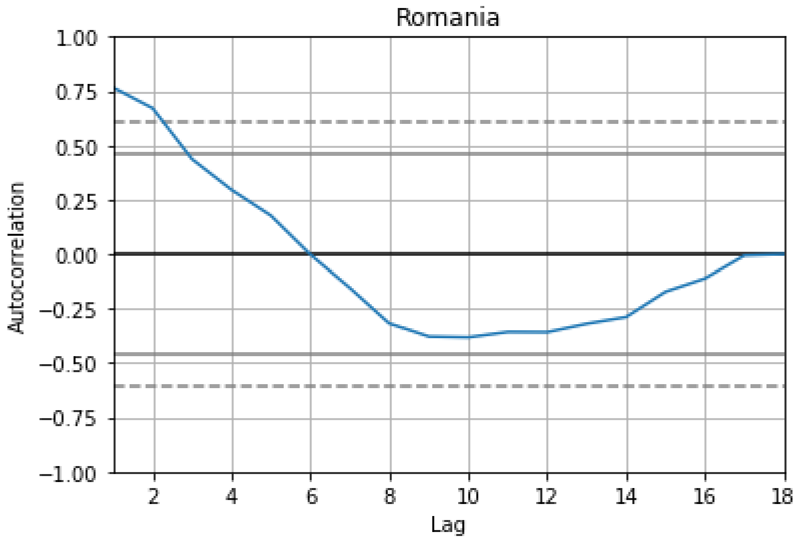

Iceland
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
2
4
6
8
10
12
14
16
18
Lag

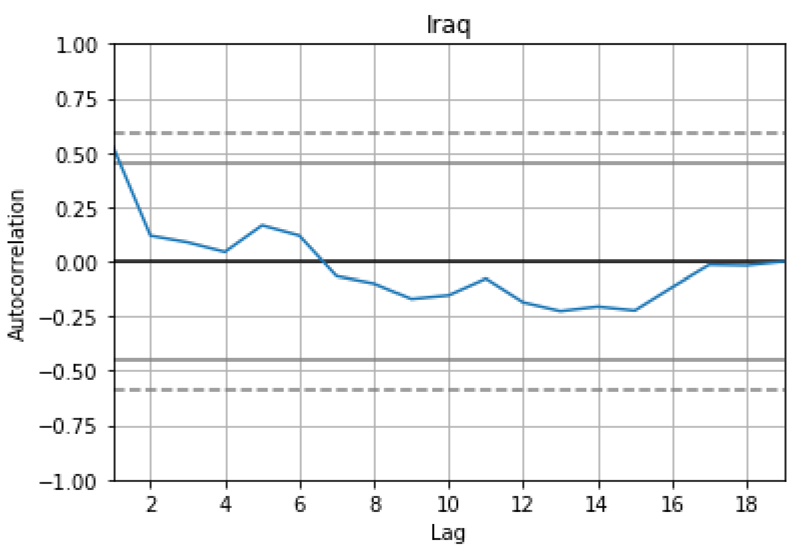

Japan
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
5
10
15
20
25
30
35
40
45
Lag

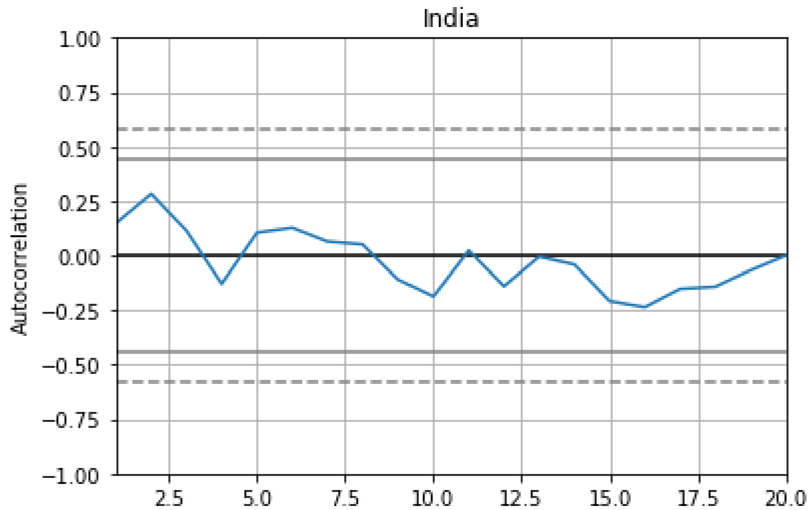

Bahrain
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

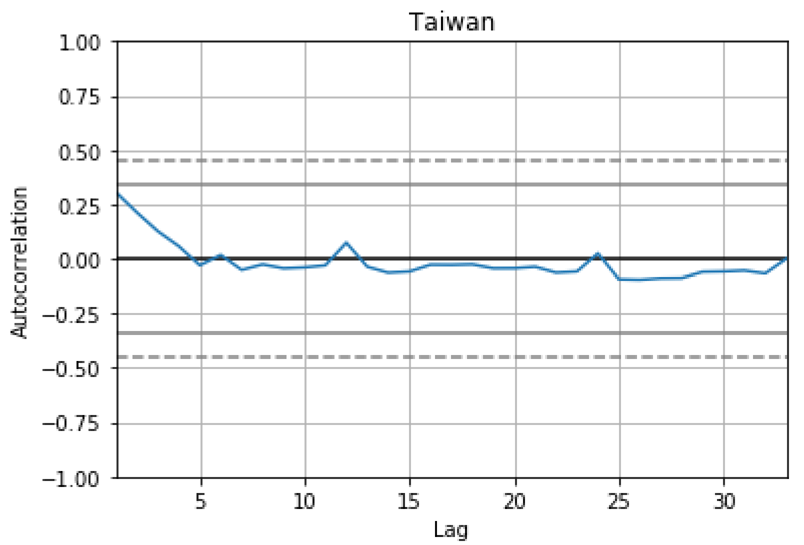

Russia
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2
4
6
8
10
12
Lag

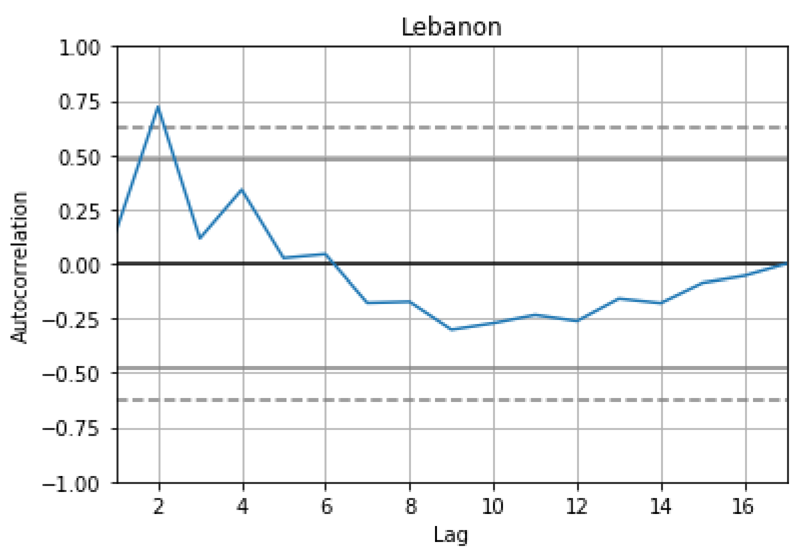

Singapore
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
10
20
30
40
50
Lag

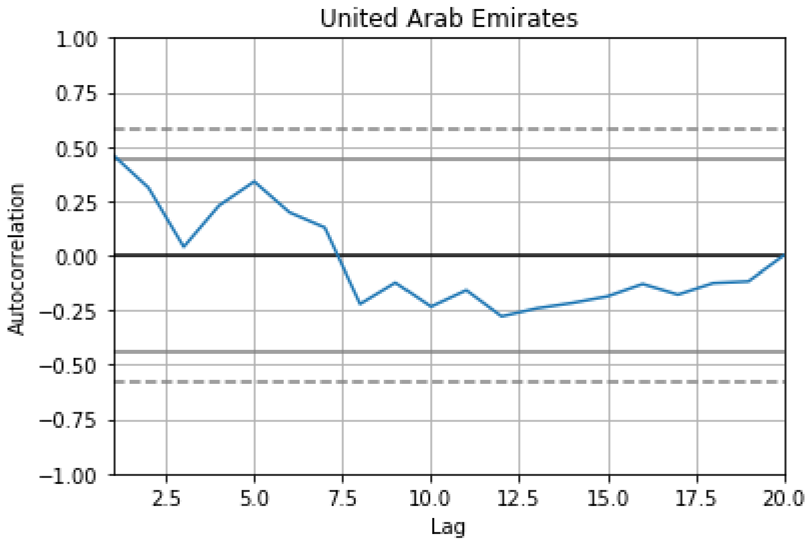

Romania
Autocorrelation
Lag

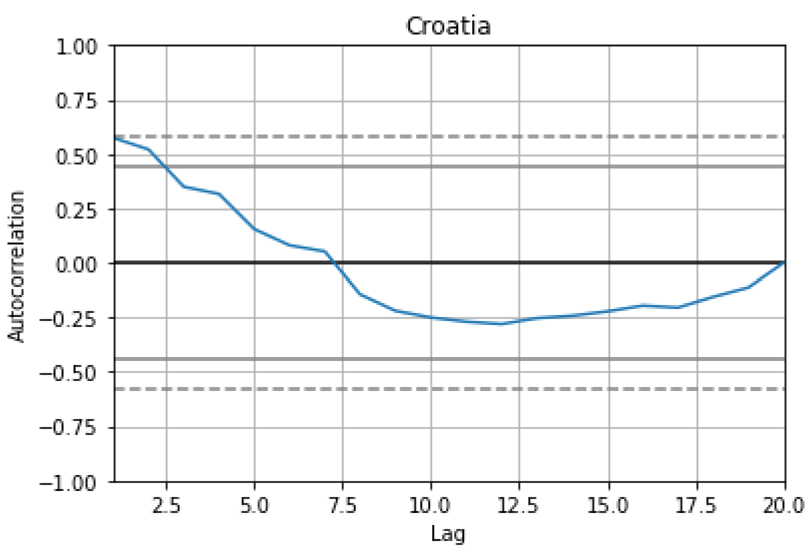

Iraq
Autocorrelation
Lag


India
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Taiwan
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
5
10
15
20
25
30
Lag


Lebanon
Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2
4
6
8
10
12
14
16
Lag





In [9]:
```
# looks like between ~(1-10) lags are correlated (some countries can use more, mo
# lets use ARIMA to get a baseline, an idea of how many lags should be used, any
```

In [10]:
```
def root_mean_squared_error(test, predictions):
    if len(test) == len(predictions):
        error = 0.
        for i in range(len(test)):
            error += (test[i] - predictions[i]) ** 2
        error /= float(len(test))
        return sqrt(error)

def root_mean_squared_percent_error(test, predictions):
    if len(test) == len(predictions):
        error = 0.
        for i in range(len(test)):
            error += ((test[i] - predictions[i]) / test[i]) ** 2
        error /= float(len(test))
        return 100 * sqrt(error)
```

In [11]:
```
# ARIMA testing over a range of lags, derivatives, and moving average days
header = ['Lags', 'Derv', 'Days', 'RMSE %', 'RMSE', '% Converged', 'Exec Time (s
if run_ARIMA:
    warnings.filterwarnings('ignore')
    rmspe = {}
    rmse = {}
    params = {}
    allTests = {}
    allPredictions = {}
    rmspe['all'] = []
    rmse['all'] = []
    params['all'] = []
    allTests['all'] = []
    allPredictions['all'] = []
    table = []
    times = []
    nIters = 1
    timer = Timer()
    for nDays in range(0, 8, 1):
        for derivative in range(0, 2, 1):
            for lag in range(nDays + 1, nDays + 8, 1):
                theseTests = []
                thesePredictions = []
                nConverged = 0
                nTried = 0
                for country in countries:
                    X = covid19[country]
                    if len(X) <= lag:
                        continue
                    size = 1 + lag + derivative
                    train, test = X[0:size], X[size:len(X)]
                    test_conv = []
                    history = [x for x in train]
                    predictions = list()
                    nSuccess = len(test)
                    for t in range(len(test)):
                        nTried += 1
                        try:
                            hist_temp = history
                            history.append(test[t])
                            model = ARIMA(hist_temp, order=(lag, derivative, nDay
                            model_fit = model.fit(maxiter=1000)
                            output = model_fit.forecast()
                            test_conv.append(test[t])
                            predictions.append(output[0])
                        except Exception as e:
                            nSuccess -= 1
                    if nSuccess > 0:
                        nConverged += nSuccess
                        theseTests = theseTests + test_conv
                        thesePredictions = thesePredictions + predictions
                        error1 = root_mean_squared_error(test_conv, predictions)
                        error2 = root_mean_squared_percent_error(test_conv, pred
                        info = f'{country} rmse={error1:.2f} rmspe={error2:.2f}'
                        #print(info)
```

```
                        if country not in rmse:
                            rmse[country] = []
                            rmspe[country] = []
                            params[country] = []
                            allTests[country] = []
                            allPredictions[country] = []
                        rmse[country].append(error1)
                        rmspe[country].append(error2)
                        params[country].append([lag,derivative,nDays])
                        allTests[country].append(test_conv)
                        allPredictions[country].append(predictions)

                # log results
                this_time = timer.get_new_time()
                times.append(this_time)
                if nConverged > 0:
                    error1 = root_mean_squared_error(theseTests, thesePrediction
                    error2 = root_mean_squared_percent_error(theseTests, thesePr
                    params['all'].append([lag,derivative,nDays])
                    allTests['all'].append(theseTests)
                    allPredictions['all'].append(thesePredictions)
                    rmse['all'].append(error1)
                    rmspe['all'].append(error2)
                    table.append([ lag, derivative, nDays, round(error2, 2), rou
                                  , round(100 * nConverged / nTried, 2), this_ti
                    clear_output()
                    print(tabulate(table, headers=header))
                print(f'estimated time left = {((64-nIters) * sum(times) / len(t
                nIters += 1

    #write pickle file
    ARIMA_results = [rmspe,rmse,params,allTests,allPredictions,table]
    with open(output_ARIMA, 'wb') as outfile:
        pickle.dump(ARIMA_results, outfile)
```

In [12]:
```python
# read pickle files (from previous ran ARIMAs)
ARIMA_results = {}
if read_ARIMA_results:
    for arima in input_ARIMA:
        with open(arima, 'rb' ) as infile:
            ARIMA_results[arima] = pickle.load(infile)
            print(tabulate(ARIMA_results[arima][-1], headers=header))
```

Lags	Derv	Days	RMSE %	RMSE	% Converged	Exec Time (s)
1	0	0	310.89	212.1	97.39	41.6357
2	0	0	259.17	592.11	91.27	83.7639
3	0	0	270.63	405.47	81.3	137.791
4	0	0	259.55	1303.2	61.14	162.8
5	0	0	267.84	593.19	49.68	231.219
6	0	0	262.77	985.45	38.56	276.92
7	0	0	327.78	1015.42	29.7	436.478
2	0	1	20135.7	7857.26	25.62	39.1062
3	0	1	299.75	981.6	20.59	48.201
4	0	1	302.97	620.75	21.04	75.3245
5	0	1	321.75	1126.41	19.33	100.52
6	0	1	373.69	164.35	15.53	132.456
7	0	1	528.01	214.9	12.77	167.973
8	0	1	414.4	166.3	12.27	337.194
3	0	2	85.01	1382.57	11.01	50.0767
4	0	2	75.66	73.34	10.89	82.5539
5	0	2	502.06	316.46	10.12	112.661
6	0	2	88.44	55.23	7.36	129.306
7	0	2	97.22	166.9	4.3	118.095
8	0	2	1817.37	391.01	8.64	384.478
9	0	2	1027	424.24	4.33	364.123
4	0	3	91.85	23.63	8.42	82.362
5	0	3	427.82	178.19	8.43	113.973
6	0	3	42362.5	15932.2	9.54	176.965
7	0	3	494.29	243.11	4.45	229.478
8	0	3	81.14	40.03	3.64	291.546
9	0	3	162.21	98.22	4.82	729.857
10	0	3	158.61	11.84	4.61	1076.19
5	0	4	131.29	74.73	7.26	195.439
6	0	4	132.92	242.52	4.63	223.49
7	0	4	11904.6	157.18	4.16	282.873
8	0	4	79.87	43.6	4.09	699.521
9	0	4	103.9	21.74	5.46	960.822
10	0	4	110.6	26.43	4.27	888.042
11	0	4	462.8	1181.3	3.83	1197.95
6	0	5	238.39	307.66	3.13	247.353
7	0	5	3995.65	290.12	3.16	342.821
8	0	5	12945.4	130.25	3.48	467.233
9	0	5	758.57	239.54	6.1	1264.3
10	0	5	390.43	182.32	4.1	845.045
11	0	5	146.63	13.84	2.37	885.355
12	0	5	203.47	83.37	1.56	901.756
7	0	6	9177.21	203.16	5.6	578.773
8	0	6	50196.4	560.09	3.33	442.067
9	0	6	668.04	202.98	4.82	1011.15
10	0	6	927.21	426.5	3.07	773.593
11	0	6	150.11	87.33	1.09	597.398

```
    12       0       6    145.05     18.56            1.95          705.106
    13       0       6  19924.9     286.64            2.73         1776.99
     8       0       7  55451.9     567.83            2.73          455.772
     9       0       7   4676.46     79.11            2.89          578.379
    10       0       7   1936.88    888.35            2.56          614.749
    11       0       7   1377.5      15.62            2.37         1083.14
    12       0       7    616.18    270.17            2.73         1278.65
    13       0       7  32061.7     450.2             2.73         1876.06
    14       0       7   7723.01    114.8             1.59          442.039
  Lags    Derv    Days         RMSE %            RMSE    % Converged    Exe
c Time (s)
------  ------  ------  --------------  ---------------  -------------  -----
----------
     1       1       0   172.23           590.53                 98.53
27.1333
     2       1       0   245.17           560.3                  90.65
52.2685
     3       1       0   282.23           555.78                 80.69
90.9832
     4       1       0   314.75           134.19                 67.06
123.879
     5       1       0   214.22           210.39                 54.9
180.486
     6       1       0   231.58           149.27                 45.34
262.697
     7       1       0   164.46           141.72                 34.09
339.975
     2       1       1  2101.27         39261.4                  34.44
39.65
     3       1       1    81.69           575.61                 31.56
51.7229
     4       1       1    92.34           485.63                 27.5
55.6726
     5       1       1     5.66714e+06      1.13343e+06          22.89
82.0398
     6       1       1   109.76           356.58                 16.79
104.187
     7       1       1    90.37           278.63                 15
169.4
     8       1       1   199.54           341.15                 13.8
237.423
     3       1       2    93.15           357.01                 20.67
65.9385
     4       1       2    75.4            605.49                 19.58
81.2387
     5       1       2    88.49           572.72                 14.99
92.3929
     6       1       2    86.04           221.28                 12.05
120.61
     7       1       2    75.83           110.83                  9.7
142.449
     8       1       2    92.93            66.94                 12.2
437.178
     9       1       2    87.94           139.92                  9.39
520.718
     4       1       3    87.48           261.72                 14.53
91.1747
```

```
     5       1       3   128.53            219.37                   10.9
105.479
     6       1       3   124.84            177.35                    9.9
184.789
     7       1       3   562.06            687.33                    8.03
263.042
     8       1       3    64.89            113.33                    9.31
600.738
     9       1       3   242.55             95.8                     7
710.402
    10       1       3   732.99            208.19                    7.29
1163.04
     5       1       4    61.21            126.37                    8.99
119.701
     6       1       4   127.35             78.4                     6.46
181.267
     7       1       4  6402.33           9267.17                    7.73
341.698
     8       1       4   136.45            223.21                    6.58
724.954
     9       1       4   131.28             65.53                    6.83
909.05
    10       1       4   231.25            137.33                    5.65
1581.46
    11       1       4   465.12           1873.73                    6.25
1663.4
     6       1       5    48.46            115.74                    4.88
141.878
     7       1       5  1297.59            166.42                    4.09
281.678
     8       1       5   383.1             497.39                    5.3
582.496
     9       1       5  1020.17            227.85                    5.63
977.072
    10       1       5   662.15            295.78                    4.55
1186.54
    11       1       5   116.38             18.11                    2.54
1102.27
    12       1       5  1318.35             52.32                    3.57
1901.8
     7       1       6   130.22            108.61                    3.79
315.064
     8       1       6   336.74            118.69                    3.05
433.337
     9       1       6   691.17            235.54                    5.63
1010.74
    10       1       6  1990.3             913.01                    2.91
920.957
    11       1       6   281.11            321.75                    2.34
981.193
    12       1       6    34.91              2.71                    1.05
666.902
    13       1       6   524.37            533.31                    4.32
2573.13
     8       1       7   570.74            255.41                    4.82
660.807
     9       1       7   617               333.46                    3.92
```

```
747.733
    10       1       7  3772.39          1736.22                  2.91
720.848
    11       1       7   167.43           486.35                  2.34
777.897
    12       1       7    57.86           109.51                  1.47
419.282
    13       1       7  8913.62           719.49                  2.95
1717.91
    14       1       7  3354.35           432.41                  3.22
1830.21
```

In [13]:

```
# from the two tables I chose the cases which have the best RMSE and convergence,
# A) 12 lags, 1 derivative, 6 day moving window >>>> 34.91% RMSE, 3 cases RMSE, 1
# B) 1 lag, 1 derivative, 0 day moving window >>>> 172.23% RMSE, 591 cases RMSE,
# C) 4 lag, 0 derivative, 2 day moving window >>>> 75.66% RMSE, 73 cases RMSE, 16
```

Prepare Data for Machine Learning

In [14]:

```
# takes the moving sum of data
# @data is pandas data frame split by ('Day', 'Cases', 'Country')
    # assumed to be grouped by country name and sorted by day number in ascending
# @days_moving is window size of moving sum
# returns dataFrame similar to @data except days are moving sums of those days
    # there will be days_moving-1 less rows for each country
def movingSum(data, days_moving):
    # helper
    def addTo(window, idx, val):
        idx += 1
        if idx >= days_moving:
            idx = 0
        window[idx] = val
        return window, idx

    # take moving sum
    newRows = []
    lastCountry = ''
    counter = 0
    idx = 0
    window = [None] * days_moving
    for i in range(len(data)):
        country = str(data.at[i, 'Country'])
        if country not in lastCountry:
            counter = 0
        window, idx = addTo(window, idx, data.at[i, 'Cases'])
        counter += 1
        if counter >= days_moving:
            newRows.append([counter - days_moving, sum(window), country])
        lastCountry = country
    data2 = pd.DataFrame(newRows, columns=['Day', 'Cases', 'Country'])
    return data2
```

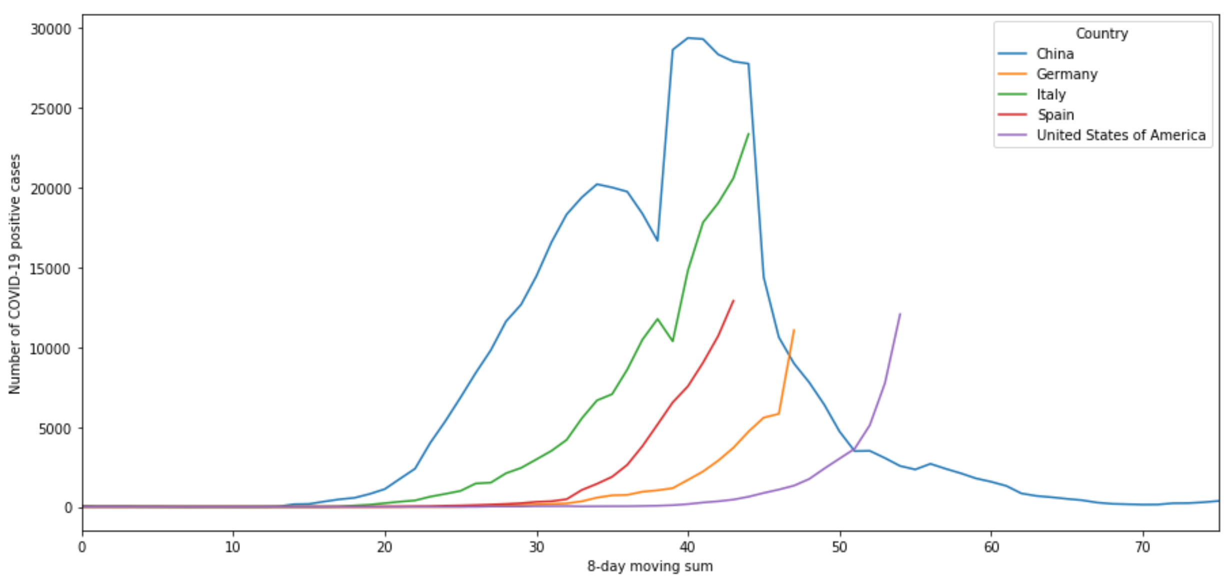
In [15]:
```
datas = {}

Xs = {} # fill chunks of feature vectors
Ys_class = {} # fill with labels of if next day will be higher or lower
Ys_regress = {} # fill with labels of value of next day
linked_countries = {} # fill with corresponding index of country used for data cl
linked_days = {} # fill with corresponding index of label number

Xs_train = {} # fill chunks of feature vectors
Ys_class_train = {} # fill with labels of if next day will be higher or lower
Ys_regress_train = {} # fill with labels of value of next day
linked_countries_train = {} # fill with corresponding index of country used for d
linked_days_train = {} # fill with corresponding index of label number

Xs_test = {} # fill chunks of feature vectors
Ys_class_test = {} # fill with labels of if next day will be higher or lower
Ys_regress_test = {} # fill with labels of value of next day
linked_countries_test = {} # fill with corresponding index of country used for da
linked_days_test = {} # fill with corresponding index of label number
```

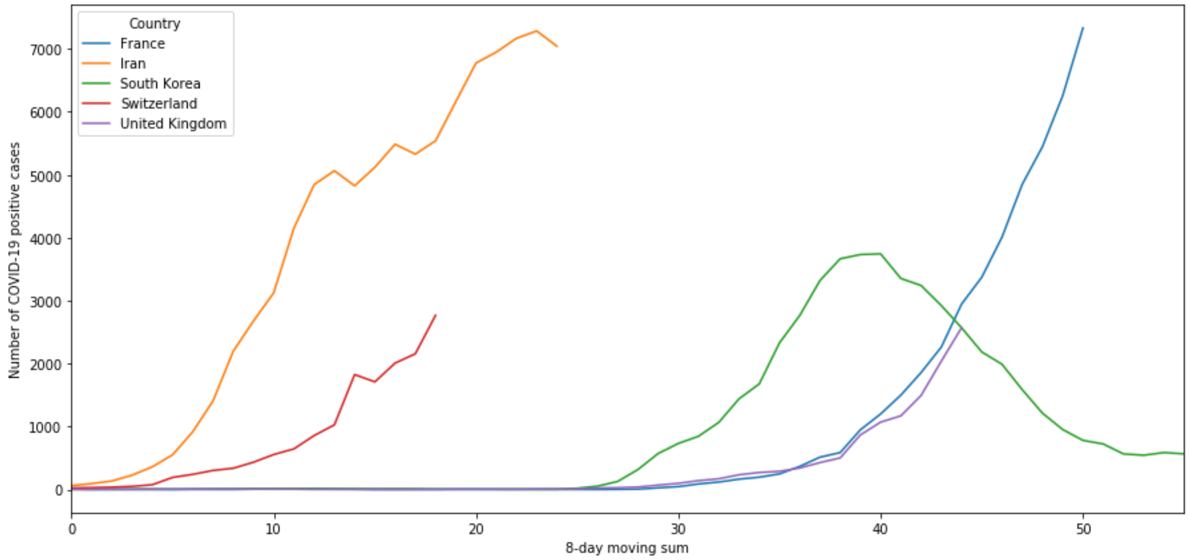
In [16]:
```
# map lags
lags = {'A' : 12, 'B' : 1, 'C' : 4}

# resample data
if resample:
    # make data for the 3 models which chose best RMSE, convergence, and balance
    datas['A'] = movingSum(data, 6)
    datas['B'] = data
    datas['C'] = movingSum(data, 2)

    # map derivatives (this looks at change in cases rather than absolute number
    dervs = {'A' : True, 'B' : True, 'C' : False}

    # make chunks of data based on lag size for each feature vector, and next va
    for model in datas:

        # grab data for this model
        this_data = datas[model]

        # create empty matrix to be filled with data chunks
        Xs[model] = [] # feature vectors
        Ys_class[model] = [] # class labels
        Ys_regress[model] = [] # regression labels
        linked_countries[model] = [] # country names
        linked_days[model] = [] # day numbers of label
        Xs_test[model] = [] # feature vectors
        Ys_class_test[model] = [] # class labels
        Ys_regress_test[model] = [] # regression labels
        linked_countries_test[model] = [] # country names
        linked_days_test[model] = [] # day numbers of label
        Xs_train[model] = [] # feature vectors
        Ys_class_train[model] = [] # class labels
        Ys_regress_train[model] = [] # regression labels
        linked_countries_train[model] = [] # country names
        linked_days_train[model] = [] # day numbers of label

        # move through all rows in data
        for i in range(len(this_data)):

            # get country for this row
            country = str(this_data.at[i, 'Country'])

            # make empty feature vector
            x = []

            # check out of bounds
            if i + lags[model] >= len(this_data):
                break

            # get number of lags
            for j in range(i, i + lags[model], 1):

                # check if not the same country
                if this_data.at[j, 'Country'] not in country:
                    break
```

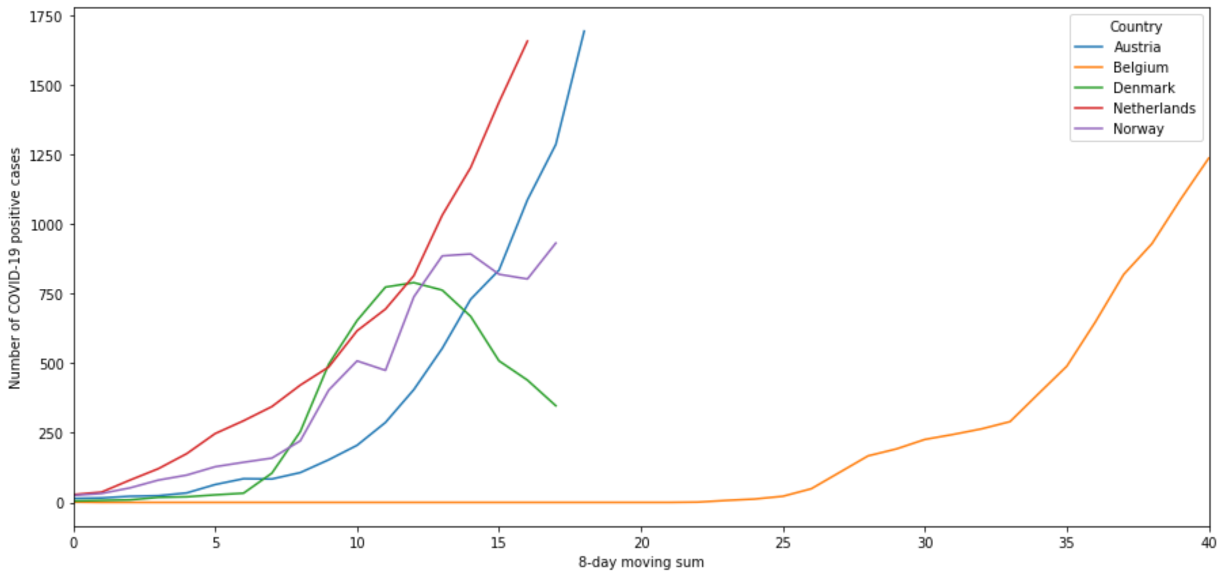
```
                # add cases to data chunk
                x.append(this_data.at[j, 'Cases'])

            # get label
            y_class = -1
            y_regress = -1
            day = -1
            if this_data.at[i + lags[model], 'Country'] in country:
                y_regress = int(this_data.at[i + lags[model], 'Cases'])
                if dervs[model]:
                    y_regress = int(this_data.at[i + lags[model], 'Cases']) - int
                y_class = 1 if int(this_data.at[i+lags[model], 'Cases']) > int(tl
                day = int(this_data.at[i + lags[model], 'Day'])

            # add to matrix if feature vector is right size and label is valid
            if len(x) == lags[model] and y_regress > -1:
                Xs[model].append(x)
                Ys_regress[model].append(y_regress)
                Ys_class[model].append(y_class)
                linked_countries[model].append(country)
                linked_days[model].append(day)

                # add to train or test
                if random.random() > test_split:
                    Xs_test[model].append(x)
                    Ys_regress_test[model].append(y_regress)
                    Ys_class_test[model].append(y_class)
                    linked_countries_test[model].append(country)
                    linked_days_test[model].append(day)
                else:
                    Xs_train[model].append(x)
                    Ys_regress_train[model].append(y_regress)
                    Ys_class_train[model].append(y_class)
                    linked_countries_train[model].append(country)
                    linked_days_train[model].append(day)


    #write pickle files
    pickle_out = [datas
                  ,Xs,Ys_class,Ys_regress,linked_countries,linked_days
                 ,Xs_train,Ys_class_train,Ys_regress_train,linked_countries_trai
                 ,Xs_test,Ys_class_test,Ys_regress_test,linked_countries_test,li
    with open(output_sample, 'wb') as outfile:
        pickle.dump(pickle_out, outfile)
else:
    # read pickle files (from previous sampling)
    with open(input_sample, 'rb' ) as infile:
        pickle_in = pickle.load(infile)

        idx = 0
        datas = pickle_in[idx]

        idx += 1
        Xs = pickle_in[idx]
        idx += 1
        Ys_class = pickle_in[idx]
        idx += 1
```

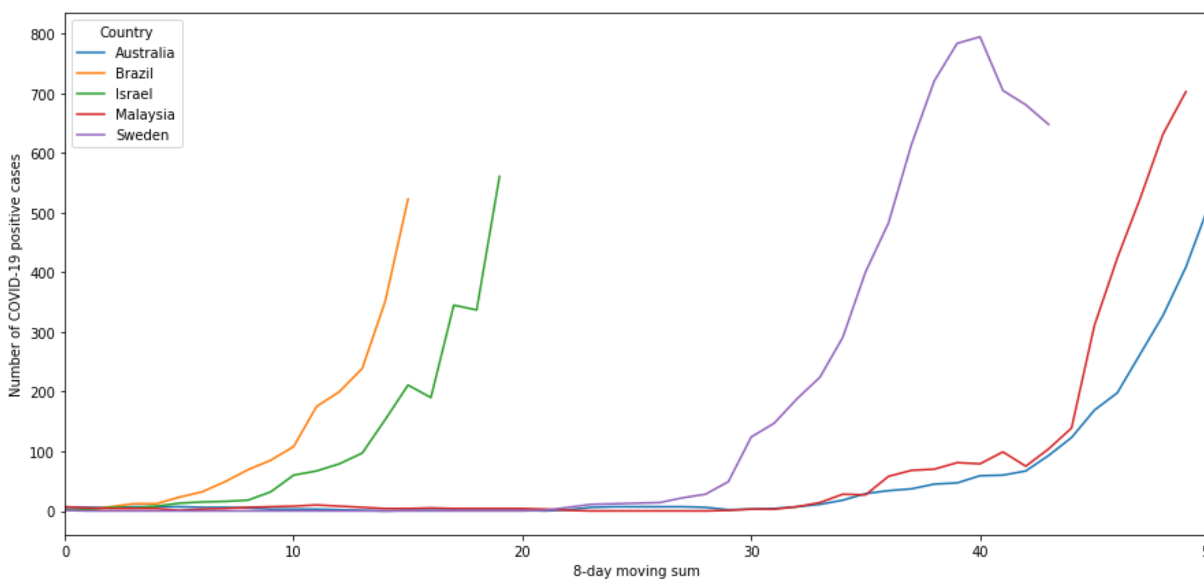
```
        Ys_regress = pickle_in[idx]
        idx += 1
        linked_countries = pickle_in[idx]
        idx += 1
        linked_days = pickle_in[idx]
        idx += 1

        Xs_train = pickle_in[idx]
        idx += 1
        Ys_class_train = pickle_in[idx]
        idx += 1
        Ys_regress_train = pickle_in[idx]
        idx += 1
        linked_countries_train = pickle_in[idx]
        idx += 1
        linked_days_train = pickle_in[idx]
        idx += 1

        Xs_test = pickle_in[idx]
        idx += 1
        Ys_class_test = pickle_in[idx]
        idx += 1
        Ys_regress_test = pickle_in[idx]
        idx += 1
        linked_countries_test = pickle_in[idx]
        idx += 1
        linked_days_test = pickle_in[idx]
        idx += 1
```

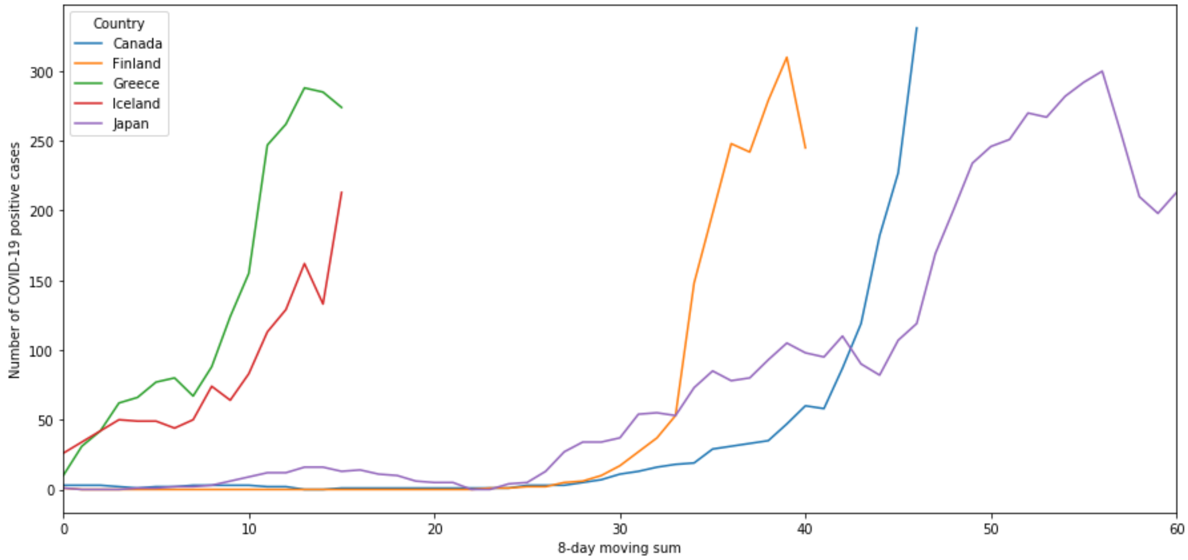
```
# view the 8-day moving window (looks much much smoother now)
plot5(datas['A'], countries, '8-day moving sum', 'Number of COVID-19 positive ca
```

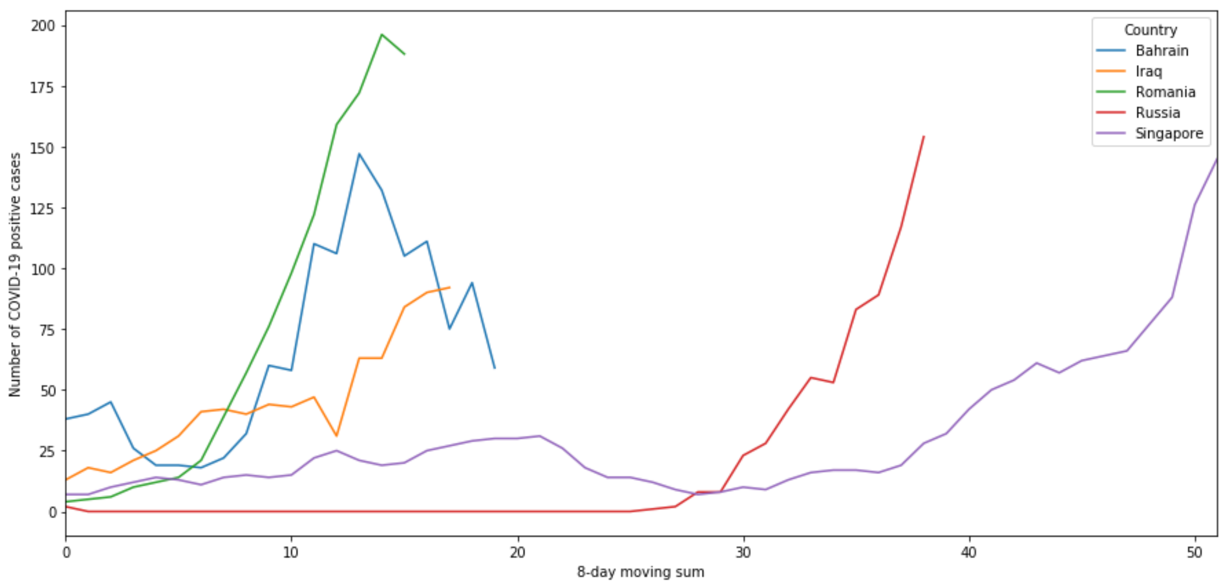



Country
France
Iran
South Korea
Switzerland
United Kingdom
Number of COVID-19 positive cases
8-day moving sum


Country
Austria
Belgium
Denmark
Netherlands
Norway
Number of COVID-19 positive cases
8-day moving sum


Country
Australia
Brazil
Israel
Malaysia
Sweden
Number of COVID-19 positive cases
8-day moving sum


Country
Canada
Finland
Greece
Iceland
Japan
Number of COVID-19 positive cases
8-day moving sum


Country
Bahrain
Iraq
Romania
Russia
Singapore
Number of COVID-19 positive cases
8-day moving sum

Cross-validate for model selection

In [17]:
```
def crossValidation_class(name, X, y, nIters, nFolds, layers):
    dis = display(f'running MLA {name}',display_id=True)

    # create dummy Machine Learning Algorithm (MLA) object
    mla = tree.DecisionTreeClassifier()

    # keep track of results
    means = [None] * nIters
    #stds = [None] * nIters

    # get results
    for i in range(nIters):
         # create MLA with new random seed
        if 'DT' in name:
            mla = DecisionTreeClassifier(random_state = i)
        elif 'NB' in name:
            mla = GaussianNB()
        elif 'RF' in name:
            mla = RandomForestClassifier(random_state = i, n_estimators=100)
        elif 'ET' in name:
            mla = ExtraTreesClassifier(random_state = i, n_estimators=100)
        elif 'SV' in name:
            mla = SVC(random_state = i, gamma='auto', probability=True)
        elif 'KN' in name:
            mla = KNeighborsClassifier(n_neighbors=2)
        elif 'ML' in name:
            mla = MLPClassifier(random_state = i, hidden_layer_sizes=layers, max_

        scores = cross_val_score(mla, X, y, cv=nFolds)
        means[i] = scores.mean()
        #stds[i] = std(scores)
        dis.update(f'iteration {i+1} of {nIters} done')
    dis.update(f'Average classification accuracy = {100. * sum(means) / len(mean

    return means
```

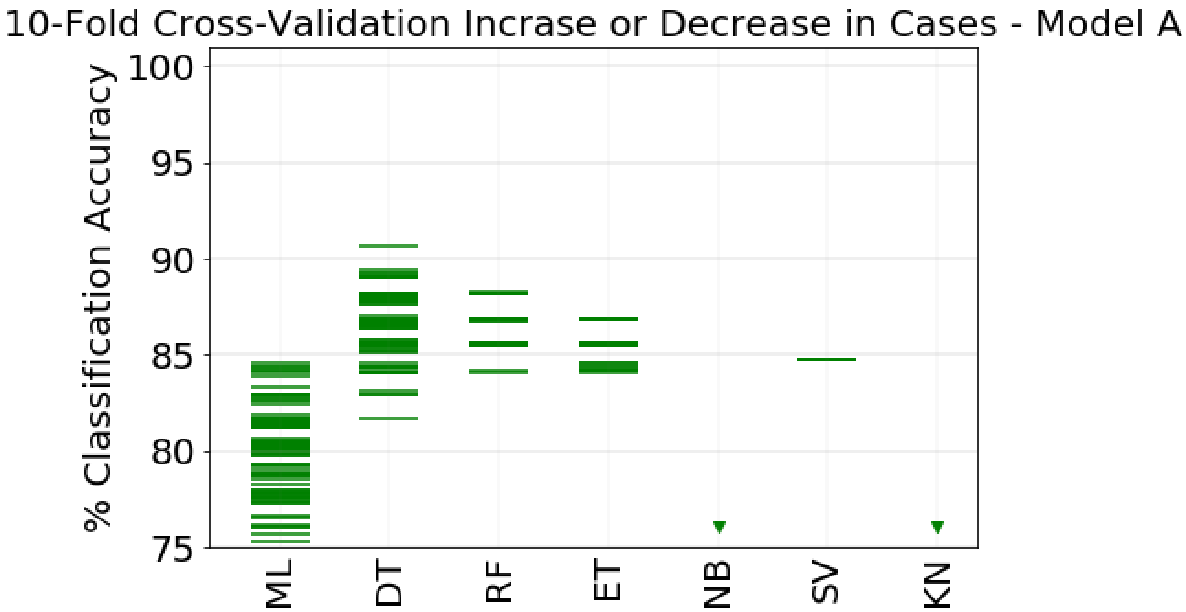
In [21]:
```
def plotResults_class(results, mlas, model):
    x_labels = []
    for mla in mlas:
        x_labels.append(mla)
        x_labels.append(' ')
        x_labels.append(' ')
        x_labels.append(' ')
    x_nums = [x for x in range(5*len(mlas))]
    fig = plt.figure(figsize=(7.5,5))
    plt.xticks(x_nums, x_labels, rotation='vertical', fontsize=20)
    plt.yticks([x for x in range(75, 101, 5)], fontsize=20)
    plt.tick_params(axis='x', length=0)
    plt.ylabel('% Classification Accuracy', fontsize=20)
    plt.title('10-Fold Cross-Validation Incrase or Decrease in Cases - Model ' +
    plt.grid(axis='y', linewidth=0.4)
    plt.ylim([75, 101])
    offset = 0
    for mla in results:
        x = [offset for _ in range(len(results[mla]))]
        y = 100. * np.array(results[mla])
        if int(max(y)) <= 75:
            plt.scatter(offset+0, 76, color='green', marker='v')
        else:
            plt.scatter(x, y, color='green', marker='_', s=1000)
        plt.axvline(x=offset, color='grey', linewidth=0.3, alpha=0.3)
        offset += 4
    fig.text(0, -.20, 'Illustrates distribution of accuracies for each MLA.\nResu
             , fontsize=20)
```

In [19]:
```
# run classification results of checking if number of cases increases or decrease
class_results = {}
mlas = ['ML', 'DT', 'RF', 'ET', 'NB', 'SV', 'KN']
if run_class:
    for model in datas:
        class_results[model] = {}
        print('model ' + model + ' ...')
        for mla in mlas:
            print('MLA ' + mla + ' ...')
            class_results[model][mla] = crossValidation_class(mla, Xs_train[model
                                                                , (max(int(lags[mod

    #write pickle file with results
    with open(output_class, 'wb') as outfile:
        pickle.dump(class_results, outfile)
```

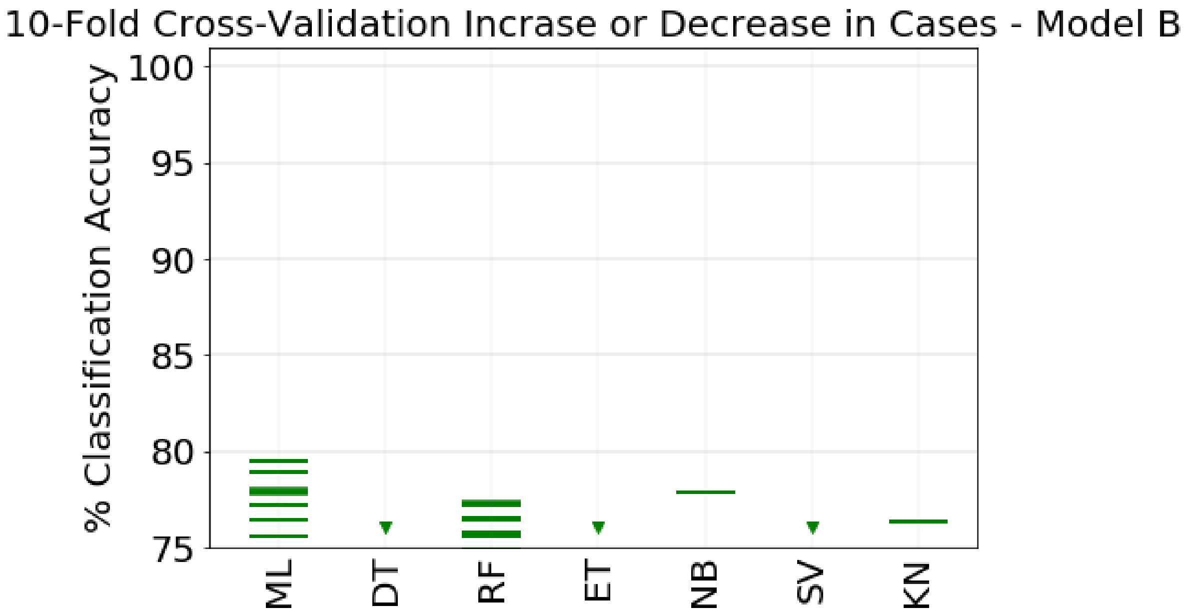
```
model A ...
MLA ML ...

'Average classification accuracy = 79.98%'

MLA DT ...

'Average classification accuracy = 86.39%'

MLA RF ...

'Average classification accuracy = 86.89%'

MLA ET ...

'Average classification accuracy = 85.69%'

MLA NB ...

'Average classification accuracy = 75.14%'

MLA SV ...

'Average classification accuracy = 84.70%'

MLA KN ...

'Average classification accuracy = 74.56%'

model B ...
MLA ML ...

'Average classification accuracy = 62.71%'

MLA DT ...

'Average classification accuracy = 74.06%'

MLA RF ...

'Average classification accuracy = 74.93%'

MLA ET ...

'Average classification accuracy = 74.06%'
```

```
MLA NB ...

'Average classification accuracy = 77.87%'

MLA SV ...

'Average classification accuracy = 75.54%'

MLA KN ...

'Average classification accuracy = 76.32%'

model C ...
MLA ML ...

'Average classification accuracy = 55.46%'

MLA DT ...

'Average classification accuracy = 66.48%'

MLA RF ...

'Average classification accuracy = 67.19%'

MLA ET ...

'Average classification accuracy = 65.98%'

MLA NB ...

'Average classification accuracy = 40.61%'

MLA SV ...

'Average classification accuracy = 67.20%'

MLA KN ...

'Average classification accuracy = 56.89%'
```

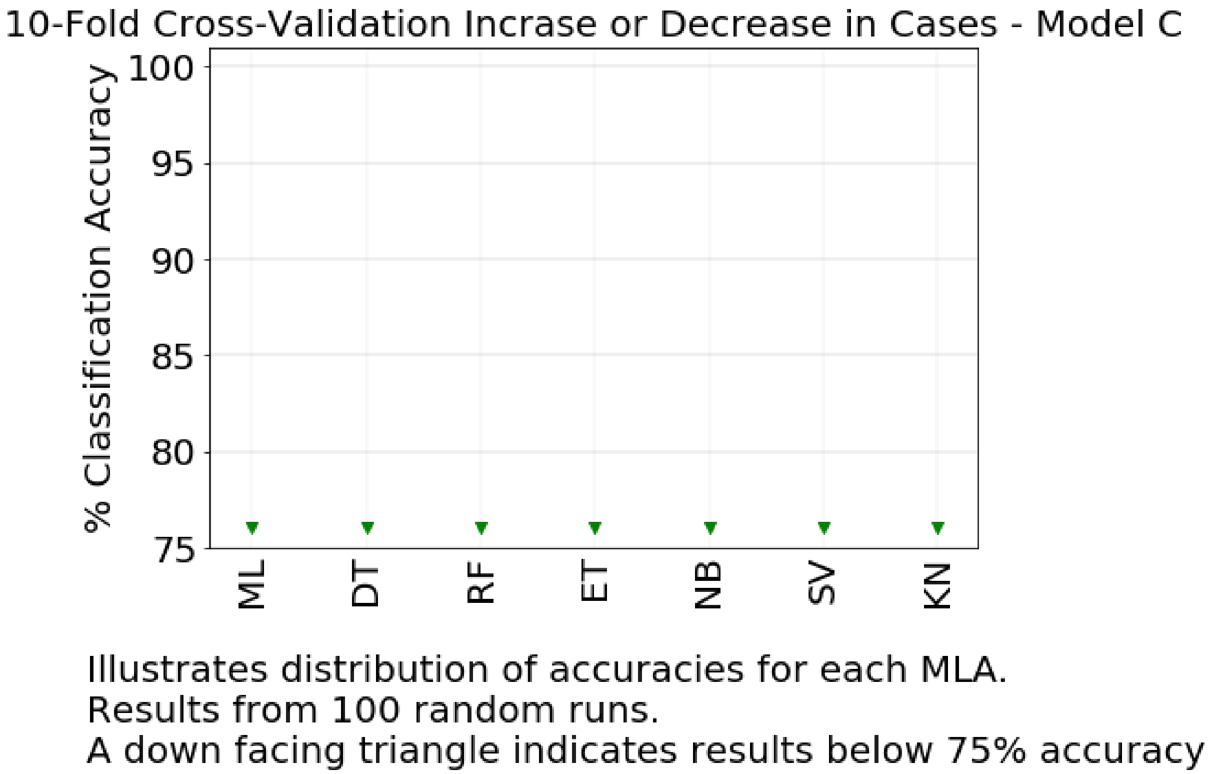
In [25]:
```
# plot results
class_results_read = {}
if read_class_results:
    for file in input_class:
        with open(file, 'rb' ) as infile:
            class_results_read[file] = pickle.load(infile)
            for model in class_results_read[file]:
                plotResults_class(class_results_read[file][model], mlas, model)
else:
    for model in class_results:
        plotResults_class(class_results[model], mlas, model)
```



Illustrates distribution of accuracies for each MLA.
Results from 100 random runs.
A down facing triangle indicates results below 75% accuracy



Illustrates distribution of accuracies for each MLA.
Results from 100 random runs.
A down facing triangle indicates results below 75% accuracy



In [28]:
```
# It is very clear from the above plots that model A is the most robust
    # Model A = 12 lags, 1 derivative, 6 day moving window
# The most robust MLA's were Decision Trees, Random Forests, Extra Trees, and Su
# I chose an SVM since they are indepedendent of randomness and performed on-par
```

Machine Learning Algorithm Optimization

In [29]:
```
# So our final selection is Model A with an SVM
# the task left is to optimize the parameters of the SVM using a grid search and
```

In [60]:
```
# optimize SVM params
warnings.filterwarnings('ignore')
parameters = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'], 'C':[1, 10, 100, 10(
              , 'tol':[0.1, 0.01, 0.001, 0.0001], 'shrinking':[True, False]
             ,'coef0':[0,1,2,3], 'gamma' : ['auto', 'scale']}
svc = SVC(probability=True)
clf = GridSearchCV(svc, parameters)
clf.fit(Xs_train['A'], Ys_class_train['A'])
cv_results = clf.cv_results_['mean_test_score']
cv_params = clf.cv_results_['params']
max_acc = max(cv_results)
opt_results = []
opt_params = []
for i in range(len(cv_results)):
    if abs(cv_results[i]-max_acc) <= 0.01:
        opt_results.append(cv_results[i])
        opt_params.append(cv_params[i])
print('optimizied classification accuracy =', max_acc)
print('These parameters obtained optimized acc:')
for params in opt_params:
    print(params)
```

```
optimizied classification accuracy = 0.8533333333333334
These parameters obtained optimized acc:
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.1}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.01}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.001}
{'C': 10, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.0001}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.1}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.01}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.001}
{'C': 10, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.0001}
```

```
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.1}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.01}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.001}
{'C': 10, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.0001}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.1}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.01}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.001}
{'C': 10, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 'to
l': 0.0001}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.1}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.01}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.001}
{'C': 100, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.0001}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
```

```
ol': 0.1}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.01}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.001}
{'C': 100, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.0001}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.1}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.01}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.001}
{'C': 100, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.0001}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.1}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.01}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.001}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'to
l': 0.0001}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.1}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.01}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.001}
{'C': 100, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False, 't
ol': 0.0001}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.1}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.01}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.001}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.0001}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.1}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.01}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.001}
{'C': 1000, 'coef0': 0, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.0001}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.1}
```

```
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.01}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.001}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.0001}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.1}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.01}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.001}
{'C': 1000, 'coef0': 1, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.0001}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.1}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.01}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.001}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.0001}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.1}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.01}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.001}
{'C': 1000, 'coef0': 2, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.0001}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.1}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.01}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.001}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 't
ol': 0.0001}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.1}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.01}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.001}
{'C': 1000, 'coef0': 3, 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': False,
'tol': 0.0001}
```

In [61]:

```
# kernel 'rbf' is the most robust, gamma 'auto' is the most robust, C 10-1000 are
# coef0 makes no difference, shrkining makes no difference, tol makes no differen
# thus the final svm used kernel 'rbg', gamma 'auto', C 10, and default values fo
svc = SVC(kernel='rbf', gamma='auto', C=10, probability=True)
```

Final Results

In [69]:
```
# now we train svs on data and test on the test data
svc.fit(Xs_train['A'], Ys_class_train['A'])
test_class = svc.predict(Xs_test['A'])
print(f'final test accuracy is:', round(100. * accuracy_score(test_class, Ys_cla
```

```
final test accuracy is: 83.33 %
```

In [ ]: