A Comparison of Time Series Methods to Predict COVID-19 Cases

Table of Contents

Introduction

As of November 29th, 2020, COVID-19 has taken the lives of 1,464,533 people around the world with 63,030,592 reported cases. It has been less than a year, but the virus has completely changed the way people interact with each other. For this project, we wanted to use supervised machine learning regression with different time series models to see which one could accurately predict the number of COVID-19 cases for the top 10 most populated countries.

The data comes from the *European Center for Disease Prevention and Control* (ECDC) which publishes daily worldwide statistics about the COVID-19 pandemic. The dataset was accessed from the site, Our World In Data. It contains data from 218 countries. Each country has the following attributes:

- Continent
- Location
- Population
- Population Density
- Median Age
- % of Population Over 65
- % of Population Over 70
- GDP per capita
- Diabetes Prevalence
- Life Expectancy
- Cardiovascular Death Rate
- Handwashing Facilities
- Hospital Beds Per Thousand
- Human Development Index

- Extreme Poverty
- Female Smokers
- Male Smokers
- COVID-19 Cases
    - This data was JSON formatted with the follow attributes:
        - Date
        - New Cases
        - New Cases Per Million
        - New Cases Smoothed
        - New Cases Smoothed Per Million
        - New Deaths
        - New Deaths Per Million
        - New Deaths Smoothed
        - New Deaths Smoothed Per Million
        - Total Cases
        - Total Cases Per Million
        - Total Deaths
        - Total Deaths Per Million

Citations

Our World In Data
https://ourworldindata.org/coronavirus-source-data

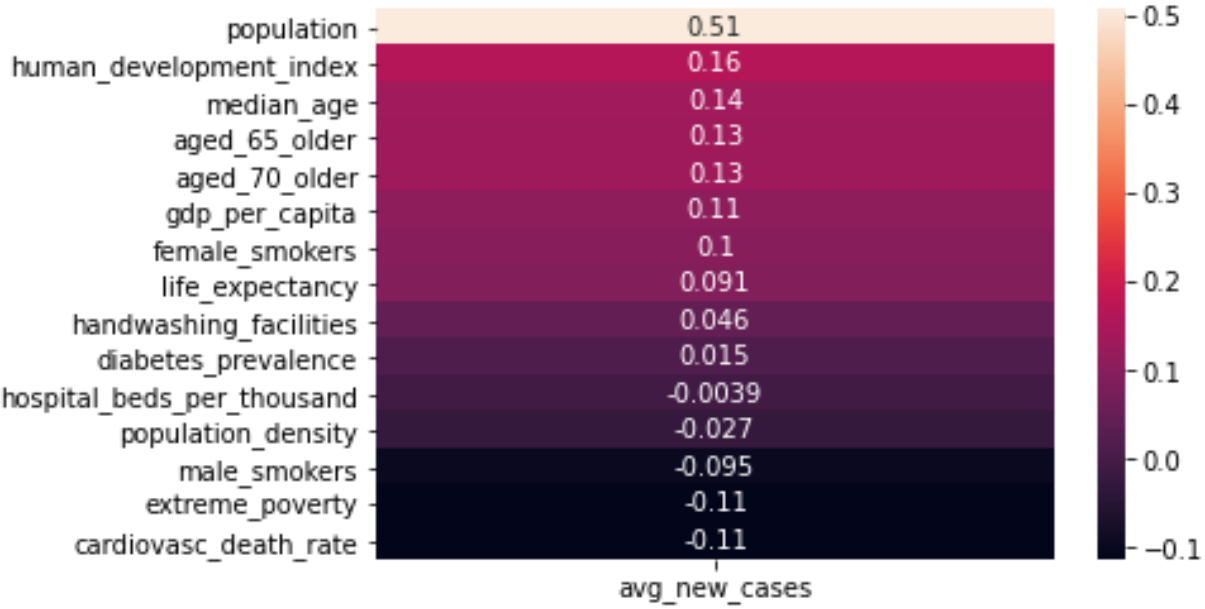
What We First Tried

We first wanted to see if it was possible to predict the amount of new cases for each country depending on the attributes using a regression supervised learning model. To do this we had to clean up the data and find if there were high positive correlation of the attributes with the average new cases for each country. If there was a correlation in the dataset, we could apply a machine learning model to the dataset.

Since there was a lot of null data in some attributes, we replaced it with the average value for the column. After doing this, we found the average new cases each day for each country, and visualized the correlation between all attributes and the average new cases per day per country.

	avg_new_cases
ABW	19.164000
AFG	140.563467
AGO	59.047809
AIA	0.016260
ALB	134.848485
...	...
YEM	9.211207
ZAF	2938.003774
ZMB	69.106299
ZWE	38.186508
OWID_WRL	183420.540541

215 rows × 1 columns



The correlation heatmap results showed some surprising results. We learned that there was low correlation between any of the attributes with the average new cases per day per country except population. This meant that there was not enough correlation in the data for us to apply a machine learning model and extract more data. This meant that we had to pivot and try a different method to predict the average number of new cases for each country.

What We Ended Up Doing

Since we learned that the attributes for each country does not have a correlation with the average new cases, we decided to try to find different time series models that would best predict the changes in average new cases for the 10 most populated countries in the world. These 10 countries were: China, India, United States, Indonesia, Pakistan, Brazil, Nigeria, Bangladesh, Russia, and Mexico. To start, we needed to make a dataframe with the daily new cases for each 10 countries.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
CHN	27.0	0.0	0.0	17.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	17.0	136.0	19.0	151.0	140.0	97
IND	1.0	0.0	0.0	1.0	0.0	1.0	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	0.0	0.0	0.0	0
USA	1.0	0.0	0.0	0.0	1.0	0.0	3.0	0.0	0.0	0.0	1.0	1.0	1.0	3.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1
IDN	2.0	0.0	0.0	0.0	0.0	2.0	0.0	2.0	0.0	13.0	15.0	0.0	35.0	27.0	21.0	17.0	38.0	0.0	55.0	82.0	141.0	64.0	65.0	107
PAK	2.0	0.0	0.0	2.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	10.0	0.0	4.0	1.0	0.0	9.0	1.0	156.0	0.0	115.0	176.0	17
BRA	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	5.0	5.0	0.0	12.0	0.0	9.0	18.0	25.0	21.0	23.0	79.0	34.0	57.0	137.0	193
NGA	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	5.0	0.0	4.0	10
BGD	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	3.0	2.0	0.0	7.0	7.0	3.0	6.0	6.0	0.0	9.0	0.0	0.0	1.0	0.0	2
RUS	2.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
MEX	1.0	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	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0

10 rows × 332 columns

The 0th column is the start date, December 31st, 2019. Each column after that is the next day until the 331st day, November 26th, 2020. This dataframe had some null data that was the result of countries not reporting new cases because they had managed to control the virus and no longer felt necessary to report. These null data points were dropped for each model.

For the time series methods, each team member randomly picked a method from the most popular data prediction methods. The time series methods we ended up using were: Auto Regressive Integrated Moving Average (ARIMA), Bayesian Ridge Regression, Holt's Linear Trend, and Support Vector Regressor, and Linear Regression.

Modeling

Bayesian Ridge Regression

Bayesian Ridge Regression is a type of linear regression. It reflects the Bayesian framework which is forming an initial estimate and then improving the estimate as more data is gathered. Bayesian regression is able to deal with insufficient data or poorly distributed data by

formulating linear regression using probability distributors rather than point estimates. The coefficient weights are slightly shifted toward zeros, which stabilises them.

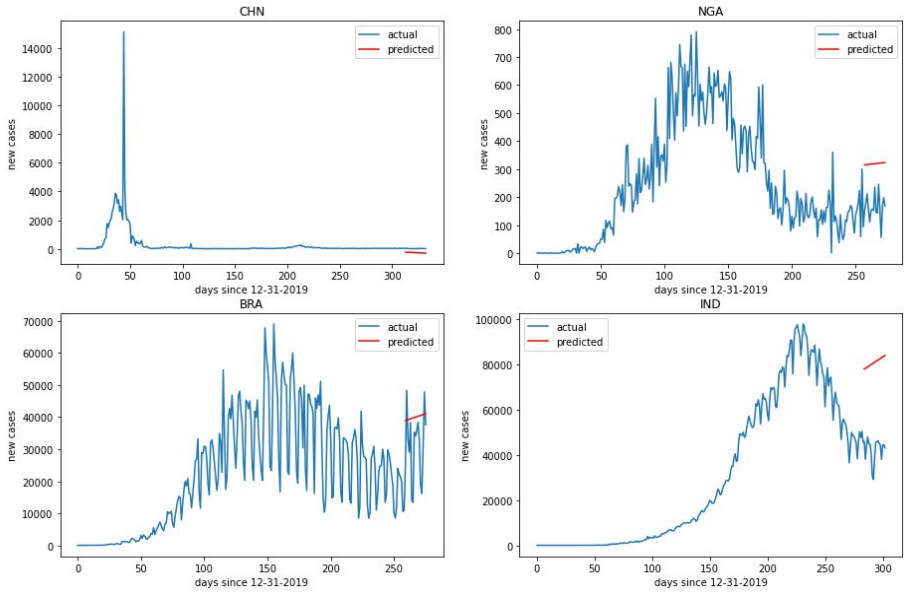
This was the model used to make the predictions:

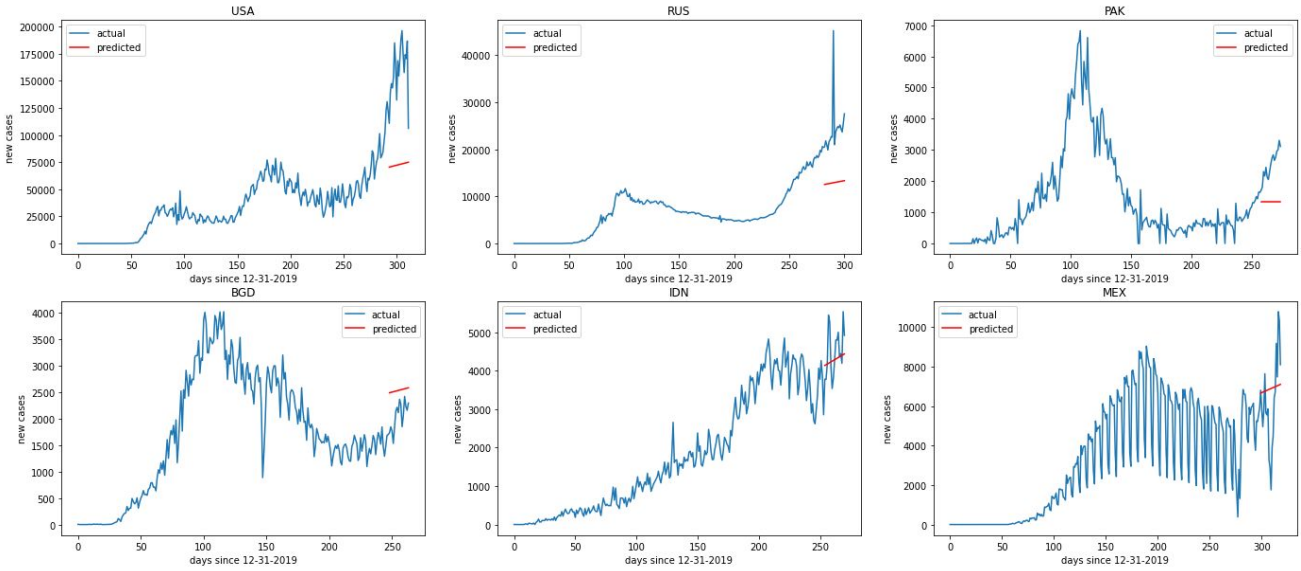
```
def applyBRR(data):
  split = int(len(data.dropna())*0.94)
  train = data.dropna()[:split]
  test = data.dropna()[split:]
  predictions = list()
  xtrain = np.reshape(train.index, (-1, 1))
  xtest = np.reshape(test.index, (-1, 1))
  bay_ridge = BayesianRidge()
  bay_ridge.fit(xtrain, train.values)
  output = bay_ridge.predict(xtest)
  error = mean_squared_error(test, output)
  predictions = pd.Series(output, index=test.index)
  return predictions

predicted = new_cases.apply(applyBRR, axis=1)
predicted
```

The data rows containing null were dropped, and the data was split into 96% training data and 4% testing data. The training data was fitted with Bayesian Ridge Regression which was then used to predict new cases from the testing data set.

Below is a graph of the original data and predicted data from using Bayesian Ridge Regression.





The mean squared error per country is shown below:

Country	MSE
CHN	8.347194e+04
IND	1.529012e+09
USA	8.044607e+09
IDN	3.605882e+05
PAK	1.584696e+06
BRA	1.846691e+08
NGA	2.593841e+04
BGD	2.870292e+05
RUS	1.562519e+08
MEX	5.521772e+06

The average mean squared error is: `992240352.7305161`

Linear Regression (Polynomial)

Linear Regression is a statistical method that is used for predictive analysis. The algorithm shows a linear relationship between dependent and independent variables. The result of this model is to find how the value of the dependent variable is changing according to the value of the independent variable. However, the data set is not ideally linear, so we apply polynomial regression, which allows us to fit a polynomial line for better accuracy.

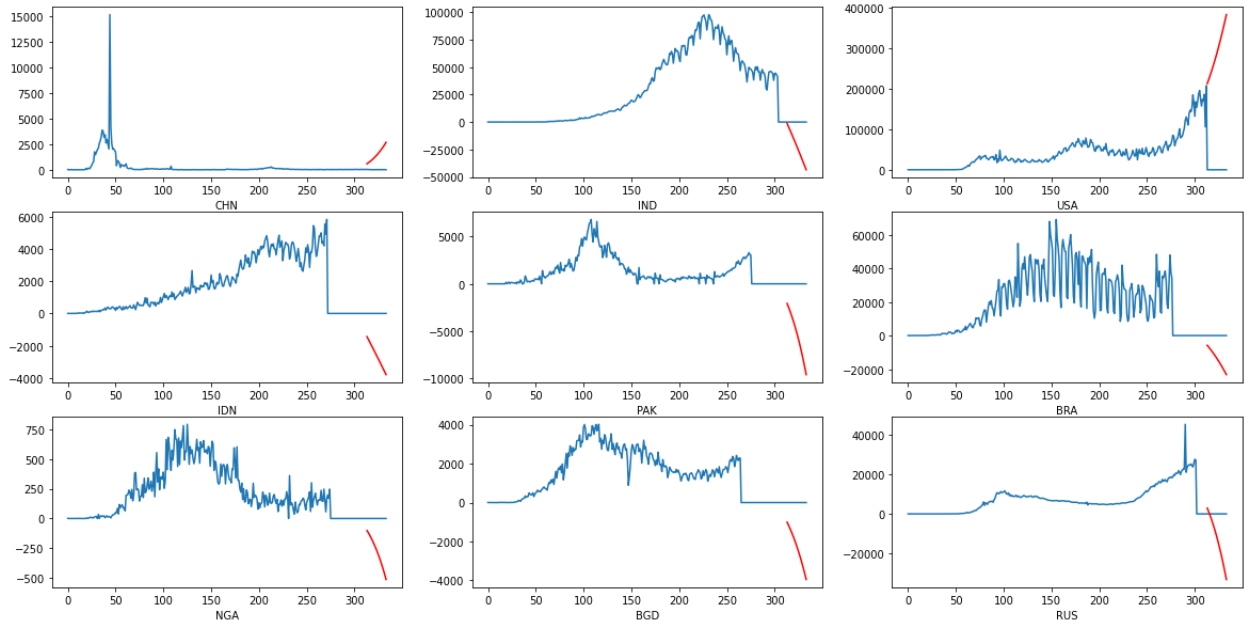
Linear Regression model for this application

```
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
def applyPoly(data):
 split = int(len(data.dropna())*0.94)
 train = data.dropna()[:split]
 test = data.dropna()[split:]
 p = make_pipeline(PolynomialFeatures(5),LinearRegression())
 p.fit(np.arange(train.index[0], train.index[-1]+1).reshape(-1,1), train)
 output = p.predict(np.arange(test.index[0],
test.index[-1]+1).reshape(-1,1))
 polyRMSE.append(sqrt(mean_squared_error(test, output)))
 return pd.Series(output, index=test.index)

polyRMSE = list()
predictedPoly = new_cases.apply(applyPoly, axis=1)
```

Below is the graph result from SVR:



Here is a dataframe with the respective mean squared error (MSE) for each country:

	Country	MSE
0	CHN	2.580023e+06
1	IND	6.482014e+08
2	USA	8.759772e+10
3	IDN	7.299102e+06
4	PAK	3.445861e+07
5	BRA	2.207894e+08
6	NGA	9.349162e+04
7	BGD	6.168226e+06
8	RUS	2.848091e+08
9	MEX	7.292971e+07

The average mean squared error is: `8887505112.246666`

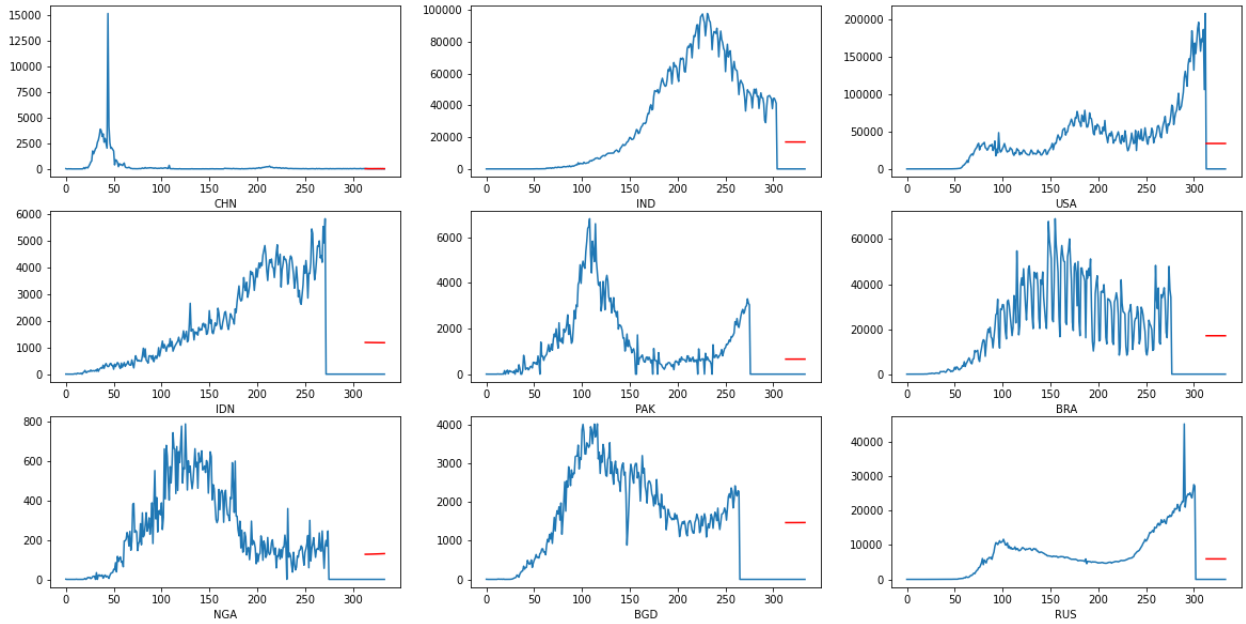
Support Vector Regression

Support Vector Regression is developed from Support Vector Machine (SVM). The differences is that SVR is used for regression applications. Assuming that our datas are moving on with SVR. The algorithm applies two lines around our datas, which is called 'Decision Boundary'. Between two the boundary lines we have what is called a Hyperplane which is our best fit line. Thus, the algorithm will take only datas that are within the decision boundary and have the least error rate. This gives us a better fitting model.

SVR model for this application

```
from sklearn.svm import SVR
def applySVR(data):
 split = int(len(data.dropna())*0.94)
 train = data.dropna()[:split]
 test = data.dropna()[split:]
 predictions = list()
 xtrain = np.reshape(train.index, (-1, 1))
 xtest = np.reshape(test.index, (-1, 1))
 for t in range(len(test)):
   svr = SVR()
   svr.fit(xtrain, train.values)
   output = svr.predict(xtest)
   svrRMSE.append(sqrt(mean_squared_error(test, output)))
   predictions = pd.Series(output, index=test.index)
 return predictions
```

Below is the graph result from SVR:



Here is a dataframe with the respective mean squared error (MSE) for each country:

	Country	MSE
0	CHN	2.211692e+02
1	IND	2.880866e+08
2	USA	1.159858e+09
3	IDN	1.399385e+06
4	PAK	4.354731e+05
5	BRA	2.912670e+08
6	NGA	1.662548e+04
7	BGD	2.154277e+06
8	RUS	3.532840e+07
9	MEX	1.650976e+07

The average mean squared error is: `179505529.09110424`

Holt's Linear Trend

Also known as linear exponential smoothing, it is a popular model used to forecast data with a trend. It has 3 separate equations that work together to generate the final forecast. The first is a smoothing equation that adjusts the last smoothed value for the last trend period. The second equation will update the trend over time where the trend is expressed as a difference between the last 2 smoothed values. The model has 2 parameters, one for overall smoothing, and one for trend smoothing equation.

The Holt model was imported from the statsmodel library.

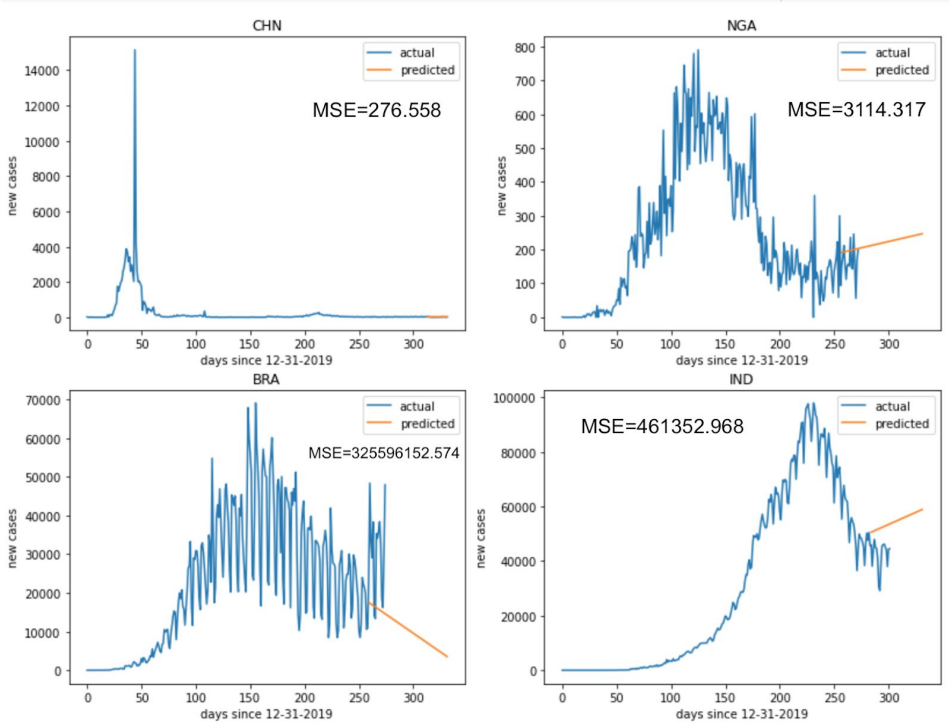
```
def applyHolt(df):
```

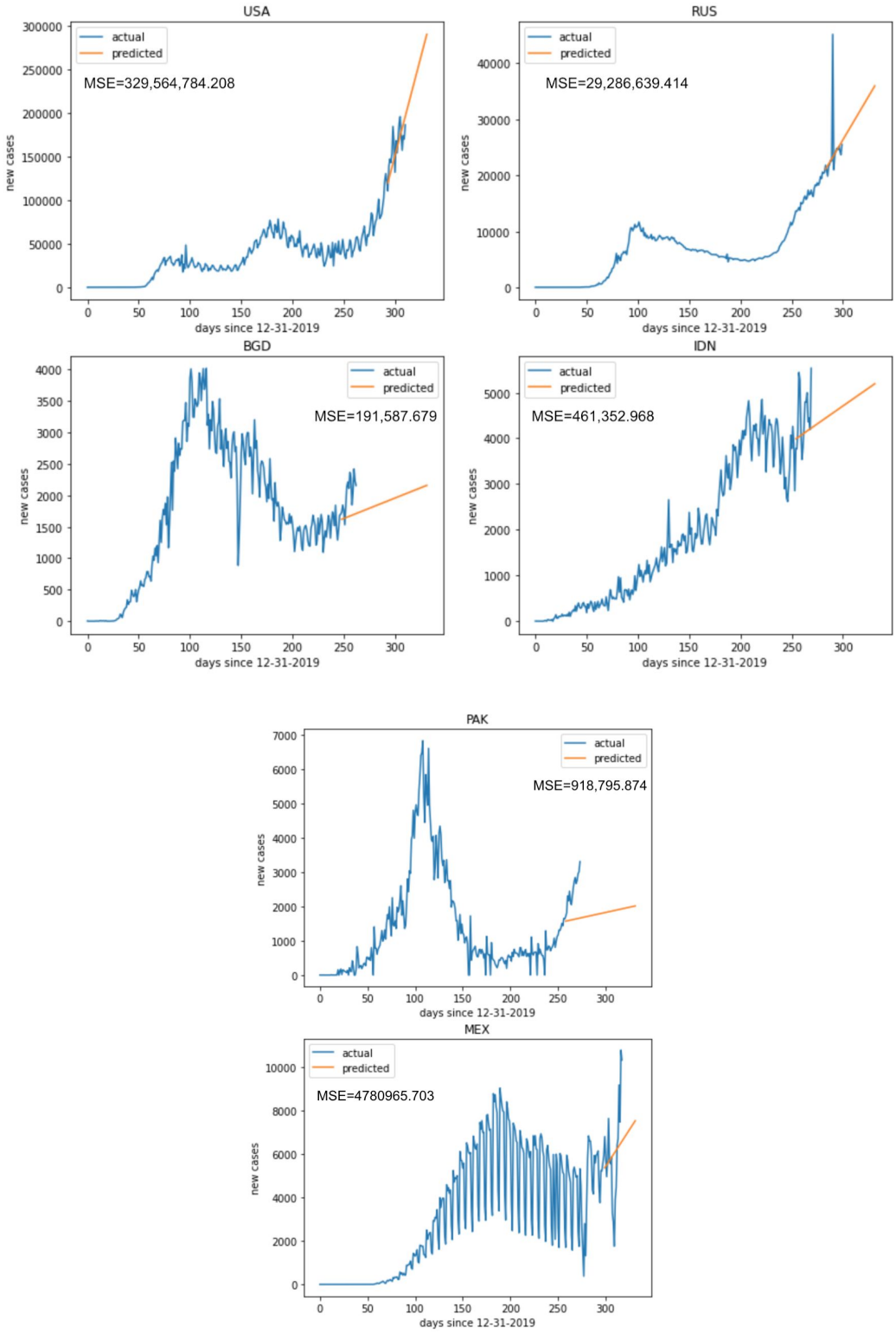
```
    data = df.dropna()
    train = df.iloc[:int(len(data)*.94)]
    test = df.iloc[int(len(data)*.94)+1:]
    model = Holt(train).fit()
    pred = model.predict(start=test.index[0], end=test.index[-1])
    return pd.Series(pred)


predicted_cases = new_cases.apply(applyHolt, axis=1)
predicted_cases
```

First, the null data were dropped before splitting the training and testing data 96% and 4% respectively. The training data was then fitted with Holt's forecasting model before it predicted the new cases for the corresponding days in the testing dataset. A new dataframe was made with the predicted data from the model.

Below are the graphs of the actual data and the predicted data from the time series model:





Here is a dataframe with the respective mean squared error (MSE) for each country:

CHN	2.765578e+02
IND	1.248312e+08
USA	3.295648e+08
IDN	4.613530e+05
PAK	9.187959e+05
BRA	3.255962e+08
NGA	3.114317e+03
BGD	1.915877e+05
RUS	2.928664e+07
MEX	4.780966e+06

The average MSE for all 10 countries was `81563485.911`

Autoregressive Integrated Moving Average

When we looked at the graphs of new cases per day of each country, we found out that there are often a lot of irregular spikes in the cases. As regression models usually only work with trends, we needed a supervised machine learning model that works well with irregular spikes.

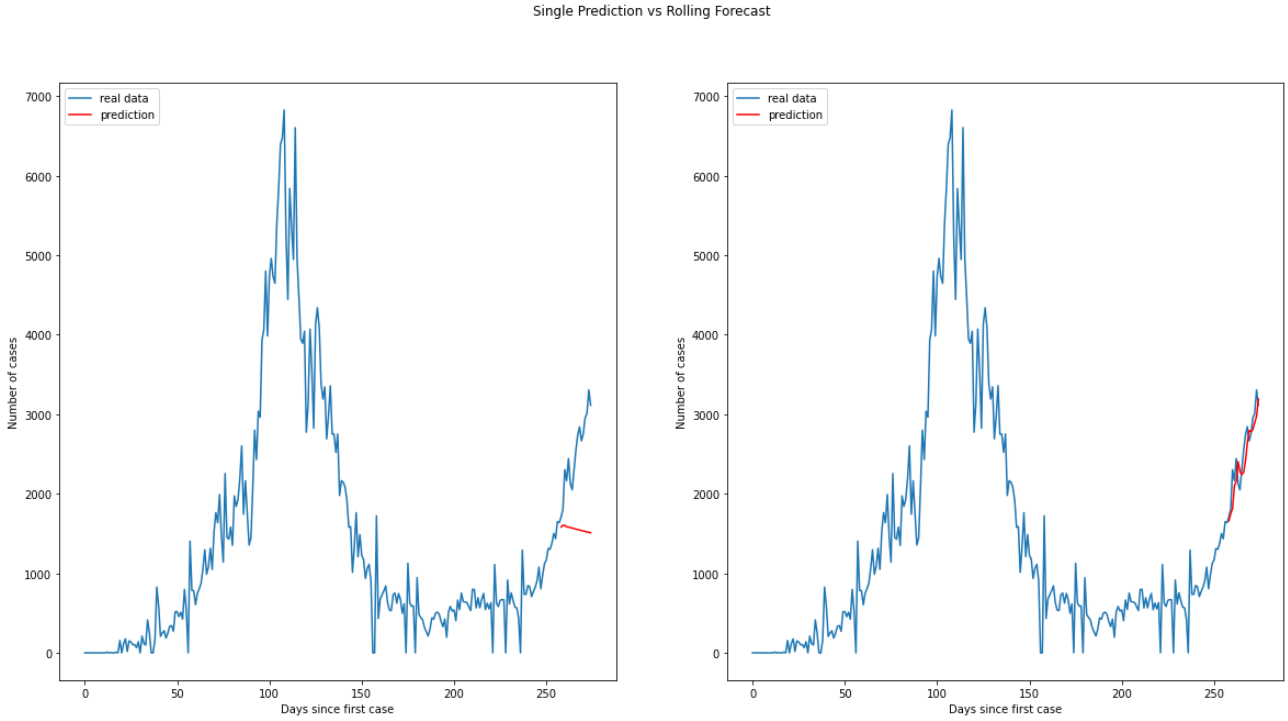
Upon researching on the internet, we found out that Autoregressive Integrated Moving Average (ARIMA) is usually used by stock traders to predict the price of securities. Due to its nature of predicting future times series based on the weights of past values, it is more tolerant to irregular spikes and can return an overall good estimation.

ARIMA uses Autoregression (AR) and Moving Average (MA) models and needs inputs of p, d, and q.

p: the number of lag observations in the model
d: the number of times that the raw observations are differenced
q: the size of the moving average window

We decided to use 5 as the value of p because we had an assumption that any data earlier than 5 days will be less relevant. We also used q value as 0 because there are very frequent spikes in the cases in most countries that we are observing.

Single Prediction vs Rolling Forecast



Upon extensive research, we found out that the ARIMA model can better predict the times series if we predict on a rolling time step basis. After predicting on each time step, the actual data is appended to the list called history to keep track of the observations. Since the ARIMA model puts weight on recent data when forecasting the future, it generates a better overall estimation. The graph above shows the comparison of results generated by single prediction using simple model.fit/model.predict and rolling forecast.

Below is the complete code that we used:

```
from statsmodels.tsa.arima_model import ARIMA
def applyARIMA(data):
 split = int(len(data.dropna())*0.94)
 train = data.dropna()[:split]
 test = data.dropna()[split:]
 history = [x for x in train]
 predictions = list()
 for t in range(len(test)):
   try:
     model = ARIMA(history, order=(5,1,0))
     model_fit = model.fit(disp=0)
     output = model_fit.forecast()
```

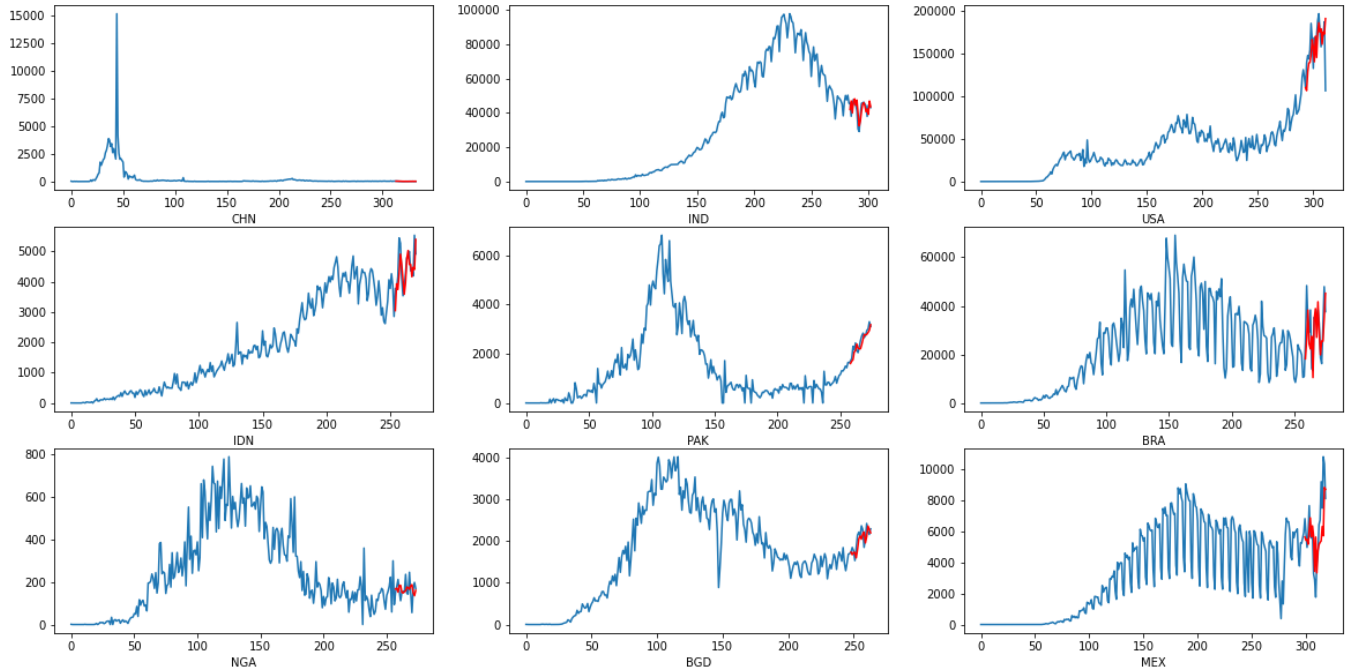
```
      yhat = output[0]
      predictions.append(yhat)
      obs = test[t+test.index[0]]
      history.append(obs)
    except:
      arimaRMSE.append(np.nan)
      return np.nan
  arimaRMSE.append(sqrt(mean_squared_error(test, predictions)))
  return pd.Series(predictions, index=test.index)

arimaRMSE = list()
predictedARIMA = new_cases.apply(applyARMA, axis=1)
```

Problem

Although the ARIMA model works well across different countries, we found out that it doesn't work on Russia dataset at all. Instead, it returns `LinAlgError: SVD did not converge` error. Upon looking at the actual dataset, we found out that Russia saw a jump from over 20,000 cases to about 40,000 cases in one day followed by 20,000 cases the next day. Due to this highly irregular spike, ARIMA couldn't estimate the new cases for Russia.

Below are the graphs of the actual data and the predicted data from the time series model:



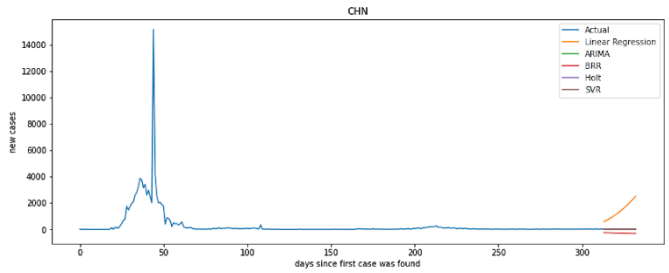
Here is a dataframe with the respective mean squared error (MSE) for each country:

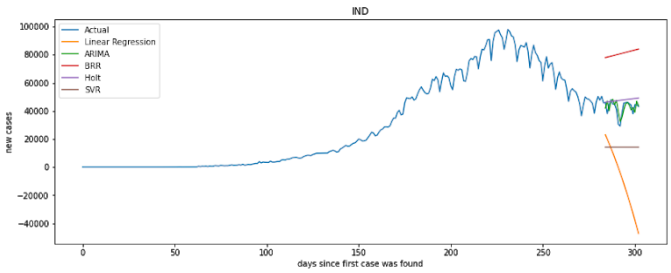
	Country	MSE
0	CHN	7.070722e+01
1	IND	1.875731e+07
2	USA	7.133535e+08
3	IDN	2.864086e+05
4	PAK	6.002071e+04
5	BRA	8.820598e+07
6	NGA	2.752891e+03
7	BGD	4.210501e+04
8	RUS	NaN
9	MEX	3.813403e+06

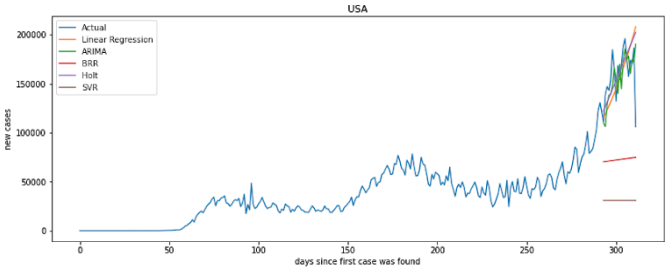
The average MSE for all 0 countries was `91613505.66`

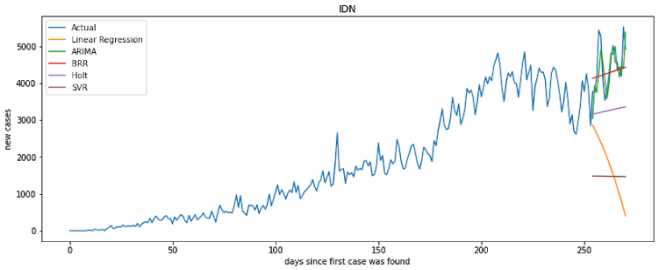
Results

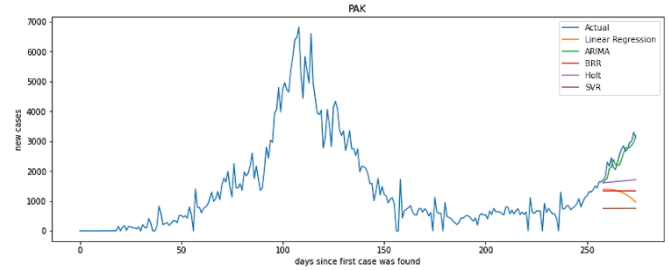
The results after using different models show how different models perform differently in each of the countries. The graph below shows the comparison of estimation by the different models that we used.

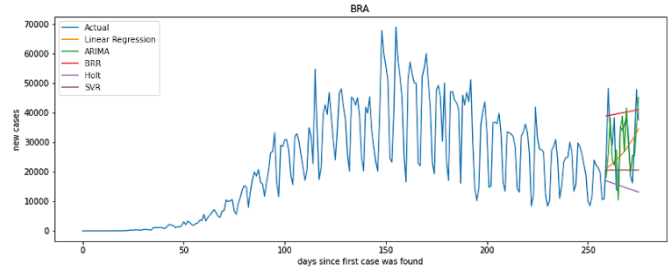


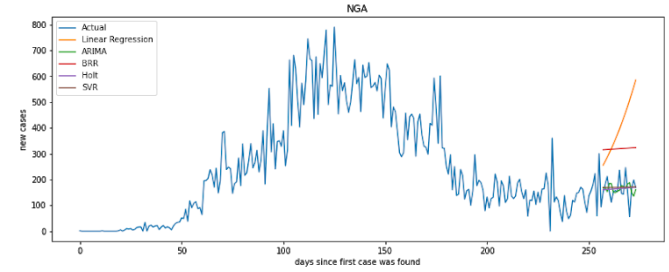


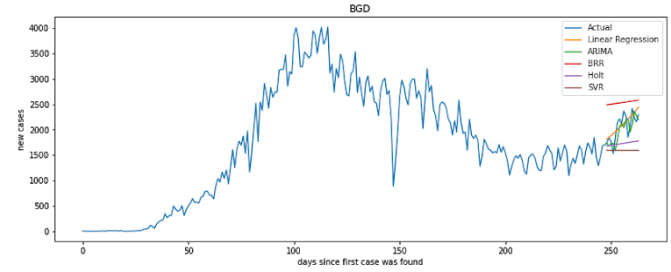


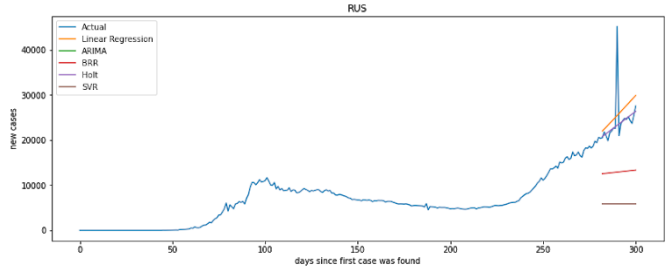


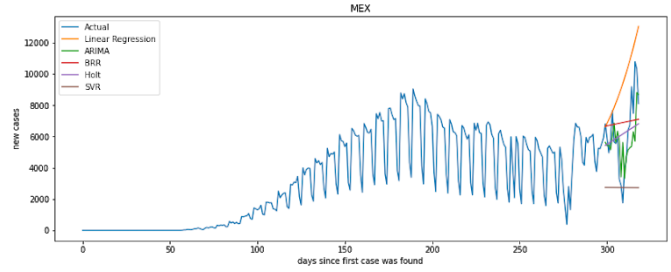










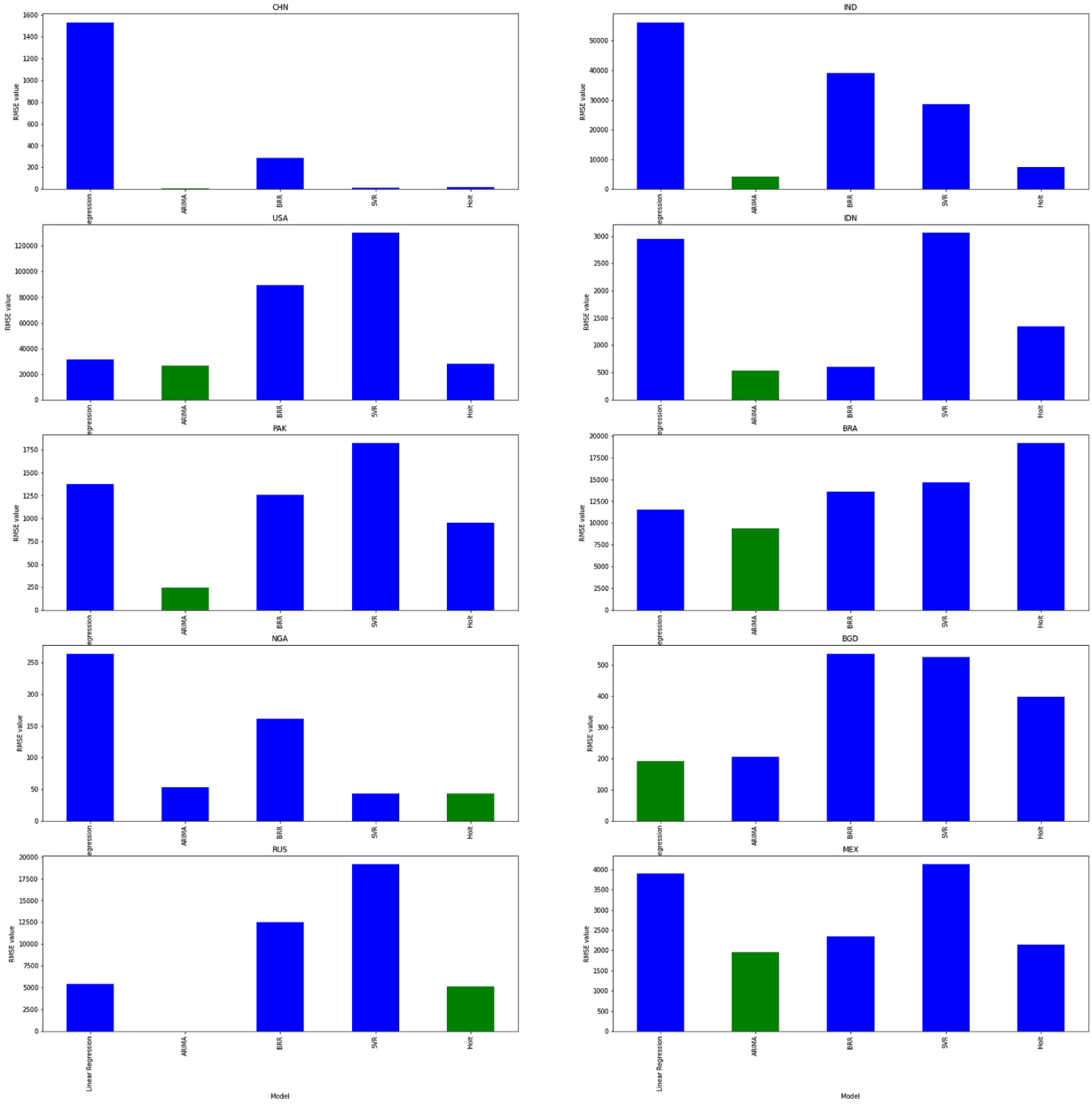


We can see that most models perform fairly poorly for each country, with the exception of ARIMA. From the graph, we can see that ARIMA performs fairly well and is able to predict the dips and peaks of cases. The other models are only able to predict the general trend of the graph for some countries. To get a more objective answer about each model, we have to look at the results of the MSE. Below is a table of the average MSE for all the models:

Time Series Model	Average MSE
Bayesian Ridge Regression	992,240,352.7305161
Linear Regression (Polynomial)	8,887,505,112.246666
Support Vector Regression	179,505,529.09110424
Holt's Linear Trend	81,563,485.911
Autoregressive Integrated Moving Average	91,613,505.66

We can see that BRR, Linear Regression, and SVR have on average much higher errors compared to Holt's Linear Trend and ARIMA.

To get the metrics to objectively compare the overall performance of each model, we calculated Root Mean Squared Error (RMSE) of each model for each country. The bar plot below shows the comparison of RMSE value.



The bar plot shows the RMSE values for each model for each country with the lowest RMSE model highlighted in green. From the bar plot, we can conclude that although ARIMA has the

lowest error value in most of the countries, Holts performs better than ARIMA on the Nigeria dataset. Moreover, we can also see that the ARIMA model doesn't work at all on the Russian dataset.

COVID-19 brought a mortifying impact on the health of the global population. Amid the global fight against the coronavirus, there has been a continued rise in the number of people getting infected. From this project that we conducted, we found out that the number of COVID-19 cases of a country does not strongly correlate to any objective metrics of that country such as GDP, population density, etc. As the rise in coronavirus cases depends on the guidelines and regulations set by the authorities, the problem of estimating the number of new cases per day is highly complex.

Although we couldn't find one single machine learning model that can accurately predict the number of coronavirus cases, we concluded that several different models should be tested out to obtain the best model which can fit the dataset. If we were to choose a model to use it would be ARIMA or Holt's Linear Trend because of their low MSE, but much more rigorous testing would be needed before a decision is made. Nevertheless, generating high-accuracy prediction could result in optimized usage of national resources as well as speedy recovery of global health and economy as we together fight against the coronavirus pandemic.