SPECIALISED MODELS:

TIME SERIES & SURVIVAL ANALYSIS:

COURSE PROJECT:

ANALYSE AND MODEL TIME SERIES DATA TO MAKE PREDICTIONS:

FORECASTING VIRUS OUTBREAKS GLOBALLY USING COVID-19 DATA

IBM / COURSERA

IDDI ABDUL AZIZ

Introduction

Time series is a sequence of data points collected and arranged successively in equal spaced time intervals. *Time series analysis* involves methods for analysing given time series data to gain meaningful information and insights that are vital for various business problems. *Time series modelling* involves creating relevant models based on passed collected data to encapsulate the inherent structure and characteristics of the given time series. These models are used to *predict/forecast* the future values of a series. This project discusses various methods to analyse single and multiple time series. It also provides different methods that can be used to forecast a single time series and multiple time series.

Import Datasets

The dataset used in this project is *Daily confirmed cases of COVID-19 globally*. This dataset has been made available freely as a github repository by Center of Systems Science and Engineering, Johns Hopkins University (JHU CSSE). The part of dataset used in this notebook is updated daily with the total COVID-19 cases for various countries and provinces. The goal is to accurately predict the global confirmed COVID-19 cases one week into the future.

Importing libraries and Dataset

```
# Importing Libraries
import numpy as np
np.random.seed(0)
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import seaborn as sns
# Hide unnecessary warnings
import warnings
warnings.filterwarnings('ignore')
# Statsmodels
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
# Linear Regression
from sklearn.linear_model import LinearRegression
# pmdarima
import pmdarima as pm
# Prophet
from fbprophet import Prophet
# Clustering
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
# Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```
# Cloning JHU CSSE's COVID-19 repository
!git clone https://github.com/CSSEGISandData/COVID-19.git
```

```
Cloning into 'COVID-19'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 23558 (delta 0), reused 5 (delta 0), pack-reused 23548
Receiving objects: 100% (23558/23558), 105.03 MiB | 11.09 MiB/s, done.
Resolving deltas: 100% (12889/12889), done.
```

```
# Reading data
data = pd.read_csv("COVID-19/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv"
data = data.iloc[:,:102] # Selecting data from various countries from January 22, 2020 to April 28, 2020
data.head()
```

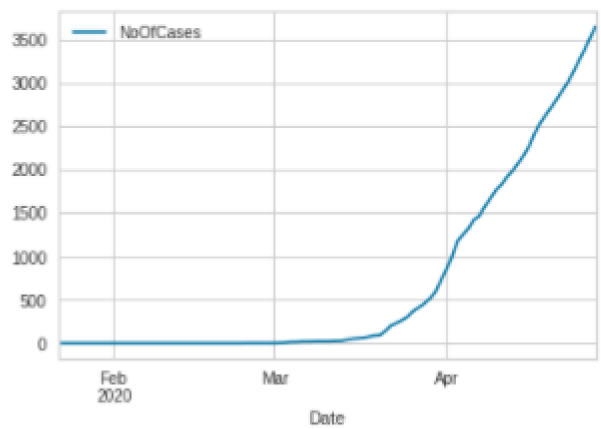
	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/
0	NaN	Afghanistan	33.0000	65.0000	0	0	0	0	0	0	0	0	
1	NaN	Albania	41.1533	20.1683	0	0	0	0	0	0	0	0	
2	NaN	Algeria	28.0339	1.6596	0	0	0	0	0	0	0	0	
3	NaN	Andorra	42.5063	1.5218	0	0	0	0	0	0	0	0	
4	NaN	Angola	-11.2027	17.8739	0	0	0	0	0	0	0	0	

5 rows × 102 columns

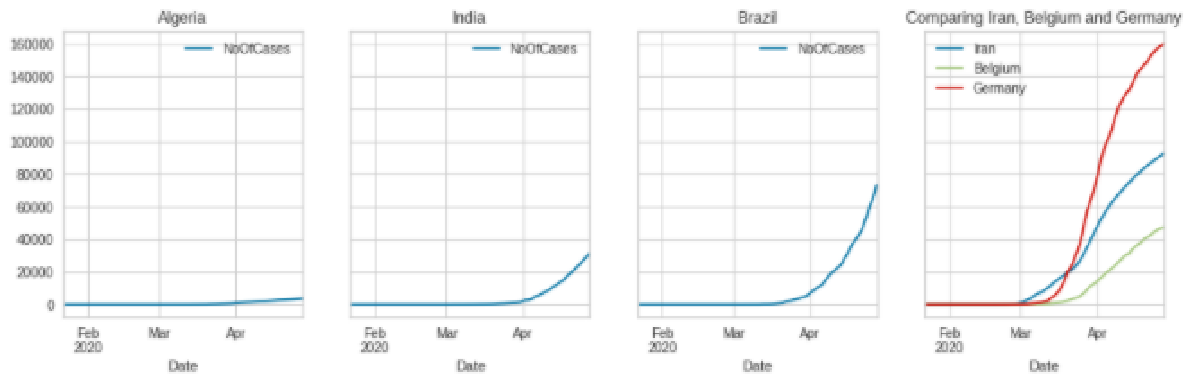
From the above illustration each row represents either an entire country or a Province/State of a country. Columns contain the cumulative increase in COVID-19 cases from January 22, 2020 to April 28, 2020. Each row can be converted to a time series of an individual country or one of its states.

Plotting a country's time series

The plot shown below shows the time series of the row number passed to it. As it is a pandas dataframe, I easily plotted it with the plot() method.

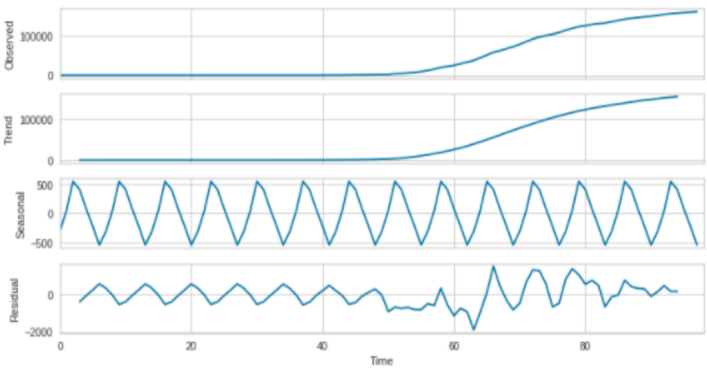


The visualisation below is a comparison of different time series I plotted side-by-side and together using the COVID-19 data.



Additive time series decomposition

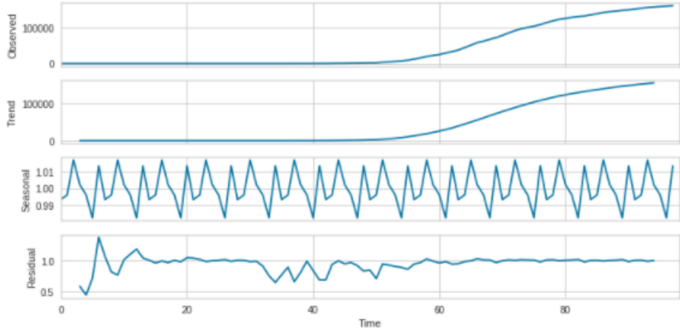
Additive method assumes the time series follows a nearly linear trend. Here, observed value is the sum of its components (trend, seasonality, noise). Below is the additive time series decomposition for Germany.



Multiplicative time series decomposition

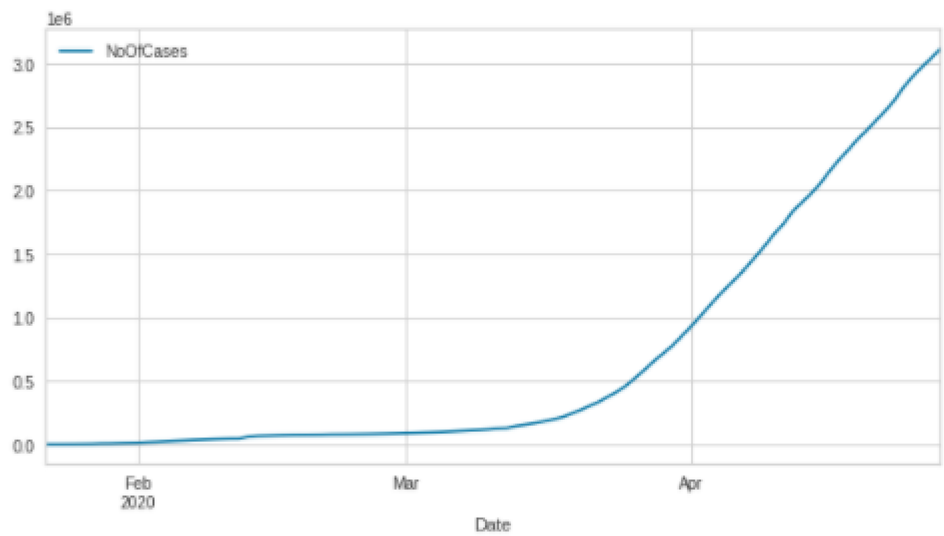
Multiplicative method assumes that the time series is non-linear or exponential. Here, observed value is the product of its components. This is more suited to countries having exponential rise in COVID-19 cases.

Below is the multiplicative time series decomposition for Germany.



A single time series for global confirmed cases

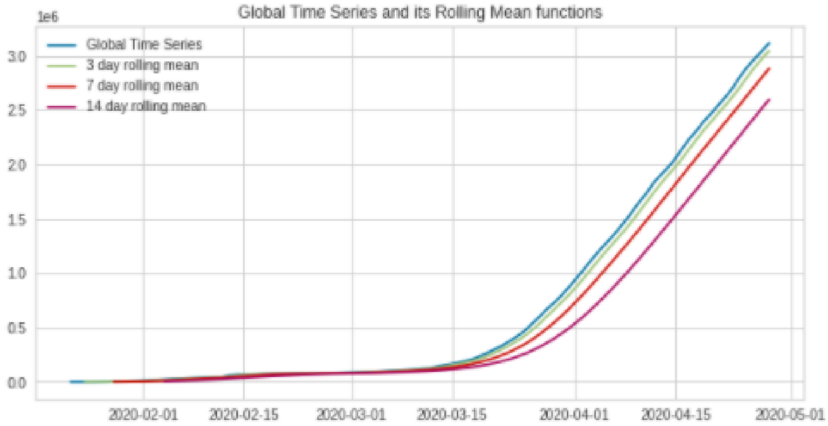
Below is a time series for global confirmed cases, by taking the sum of confirmed cases of each country on each day. I will be using this data to train the models.



Rolling and expanding window functions

Rolling Mean (Moving Average)

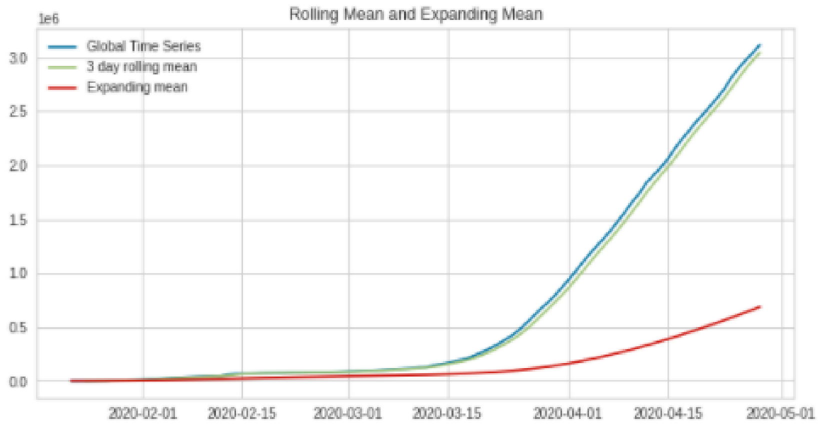
Rolling mean for a given point is the arithmetic mean over a specified number of previous observations. It can be used to forecast future points assuming that the future data points must be very close to mean of past data points. It can also be used to deduce the trend of a time series.



From the plot illustrated, it can be observed that rolling mean deviates from the actual data more and more as the window size increases.

Expanding Mean

Unlike rolling mean, *expanding mean* doesn't have a fixed window size. For a given point, its window includes all the previous points in the time series.



As can be observed, in the global time series, there has been an exponential rise in cases. The effect of past points which have much less magnitude compared to the more recent points leads to the large deviation of expanding mean from the original time series.

Time Series Clustering

The COVID-19 dataset contains data from almost all the countries in the world. It will be interesting to know in which countries the outbreak occurred in a similar way. An effective method to do is *KMeans Clustering*.

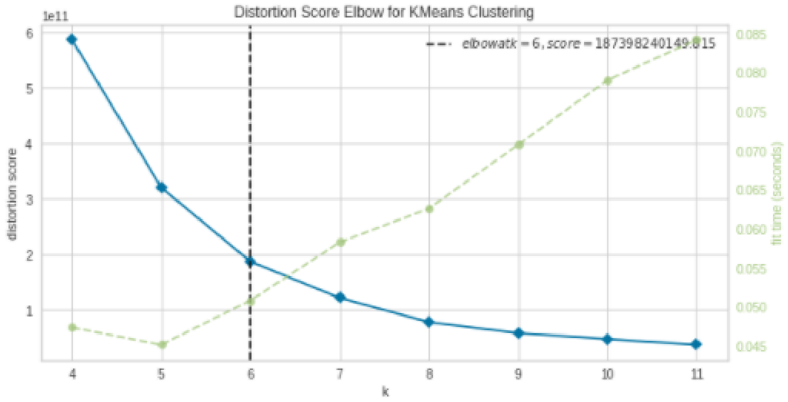
KMeans Clustering

KMeans clustering is an unsupervised learning algorithm that divides into *k* clusters. Over successive iterations, *k* clusters are computed and adjusted so that they are surrounded by similar points until their positions don't change anymore.

	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	1/31/20	2/
0	Afghanistan	33.0000	65.0000	0	0	0	0	0	0	0	0	0	0	
1	Albania	41.1533	20.1683	0	0	0	0	0	0	0	0	0	0	
2	Algeria	28.0339	1.6596	0	0	0	0	0	0	0	0	0	0	
3	Andorra	42.5063	1.5218	0	0	0	0	0	0	0	0	0	0	
4	Angola	-11.2027	17.8739	0	0	0	0	0	0	0	0	0	0	

5 rows × 101 columns

In KMeans clustering, number of clusters to be formed is specified before calculations. The elbow method is used to calculate this parameter. *KElbowVisualizer* class of *yellowbrick* package provides this feature. The figure below shows six clusters are required to be formed.




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcedb0734e0>
```

I will then cluster with 6 clusters, in defining the model, estimating clusters, and then assigning cluster to individual countries. Shown below are the clustered countries.

```
[13]: 0    168
      5      8
      2      4
      4      3
      3      1
      1      1
      Name: Cluster, dtype: int64
```

From the illustration above, one can observe that 168 countries belong to Cluster 0 while clusters 1 and 3 have one country each. Now, Let's see the country names to get a better idea.

```
for i in range(1,6):
  print("Countries in cluster {} are {}".format(i,list(data_by_country[data_by_country.Cluster==i]["Country/Region"
```

```
Countries in cluster 1 are ['US']
Countries in cluster 2 are ['France', 'Germany', 'Italy', 'Spain']
Countries in cluster 3 are ['China']
Countries in cluster 4 are ['Iran', 'Turkey', 'United Kingdom']
Countries in cluster 5 are ['Belgium', 'Brazil', 'Canada', 'India', 'Netherlands', 'Portugal', 'Russia', 'Switzerla
nd']
```

- Unsurprisingly, the U.S. and China's data is so different from others that they don't belong to any other clusters.
- China was the first country where COVID-19 cases were confirmed. So, for initial days where every other country has zero or few cases, cases in China were rapidly increasing, hence a separate cluster for China.
- The U.S. has now become the country with the most cases and this number is increasing rapidly each day, hence a separate cluster just for the U.S.
- Cluster 2 has France, Germany, Italy and Spain. These countries have also been severely affected by COVID-19.
- Cluster 4 has Iran, Turkey and United Kingdom has lesser cases than countries in Cluster 2 but still a large number of cases have been confirmed in these countries.
- Cluster 5 has countries where daily cases started increasing later than most of the highly-affected countries. Now, these numbers are significantly high.
- Cluster 0 has all the other countries present in the dataset.

Clustering results can vary based on random_state values. Also, much better results can be achieved from clustering by normalizing data with countries' population.

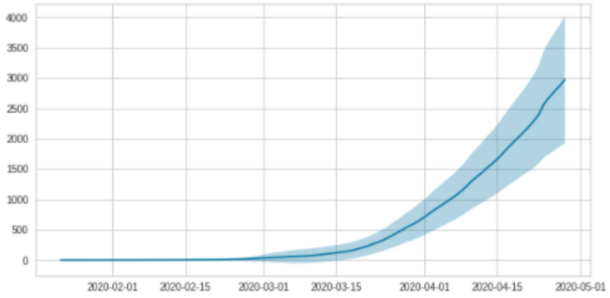
A good way to visualize these clusters is to plot a time series aggregrated with average number of cases of all countries in the cluster daily with a 95% confidence interval.

Confidence interval for population is computed as the sample mean ± a margin of error.

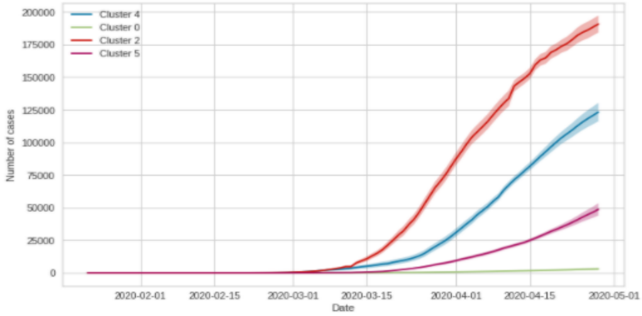
$$\bar{x} \pm (z \times \frac{s}{\sqrt{n}})$$

where $\bar{x}$ is the sample mean, s is sample standard deviation, n is the number of observations and z=1.96 for 95% confidence interval.

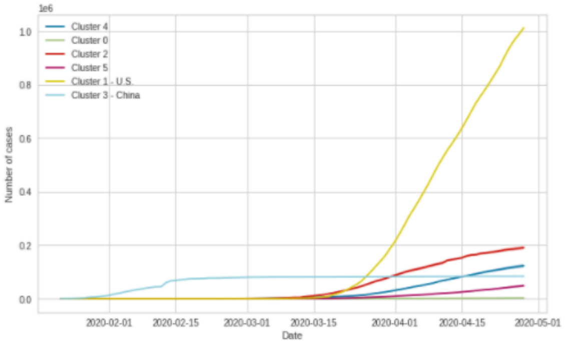
The following plot shows average cases for Cluster '0':



The following plot shows all clusters except the U.S. and China with 95% confidence intervals. Differences in the trends of the four clusters is very significant.



The following plot shows all six clusters. China has a unique plot where cases become nearly constant from around February 15. The U.S. clearly has a separate cluster due to high number of cumulative cases.



Time Series Modelling

AR models

The autoregressive (AR) model specifies that the output variable depends linearly on its own previous values and on a *stochastic* term (an imperfectly predictable term); thus, the model is in the form of a stochastic difference equation.

MA models

The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

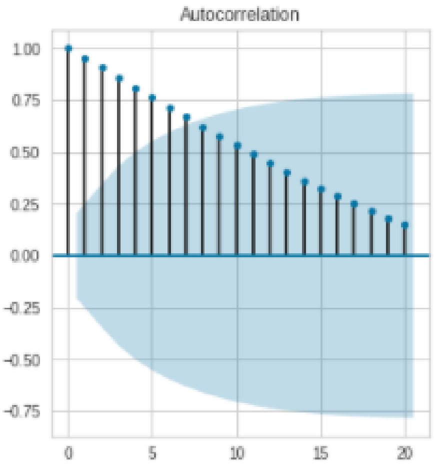
ARIMA model

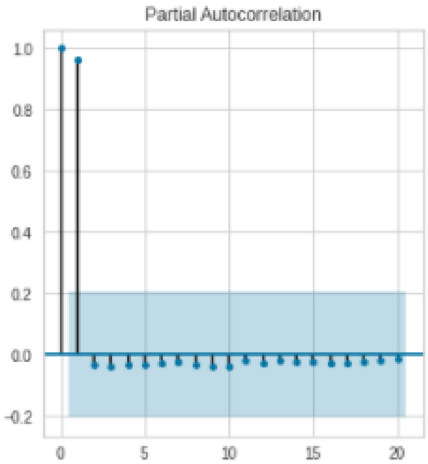
An ARIMA model combines the AR models and MA models with an initial differencing *parameter(I).* These models are effective in forecasting future data points for a stationary series. ARIMA model has three parameters:

- *p*: number of time lags of the AR model.
- *d*: degree of differencing (number of times past data had to be subtracted from the data to make it stationary). It can be 0, 1 or 2.
- *q*: number of time lags of the MA model.

Autocorrelation and Partial Autocorrelation

Autocorrelation can be defined as the correlation of time series observations with their lagged forms. It describes how much a data point depends on its previous data points.
Partial Autocorrelation can be defined as the correlation of time series observations with their lagged forms without the effect of intervening observations.





For a series with zero order differencing, Autocorrelation plot shows significant lags till 7th lag which can be the *q(MA)* parameter of the ARIMA model. Partial autocorrelation plot shows significant lags till the first lag, which can be the *p(AR)* parameter of the ARIMA model.

Augment Dickey-Fuller Test

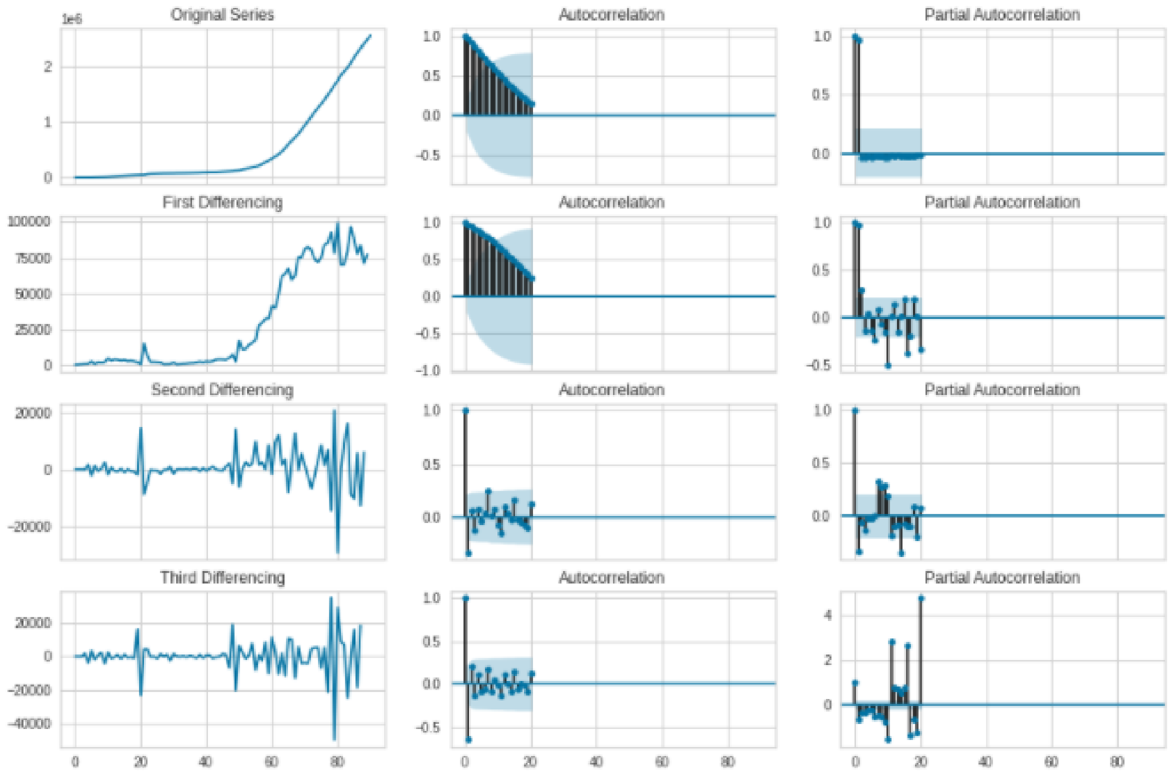
An *augmented Dickey–Fuller* test (ADF) tests the null hypothesis that a unit root is present in a time series sample. Unit root being present means time series is not stationary. Unit root is present when $\alpha = 1$.

For a series to be stationary, $\alpha \neq 1$. To achieve this, we subtract previous values from series again and again, and perform ADF tests to get a *p-value* less than 0.05 so we can reject the null hypothesis. Number of times differencing was done determines the *d* parameter of an ARIMA model:

```
p-value after differencing 0 times: 0.9984784076669422
p-value after differencing 1 times: 0.5959120852641959
p-value after differencing 2 times: 0.573589245311854
p-value after differencing 3 times: 0.0055082296135170045
```

Differencing parameter *d* should be 3 but I chose a maximum value of 2 for the given package.

Following is a plot that shows autocorrelation and partial autocorrelation plots of 0th, 1st, 2nd and 3rd order differencing:

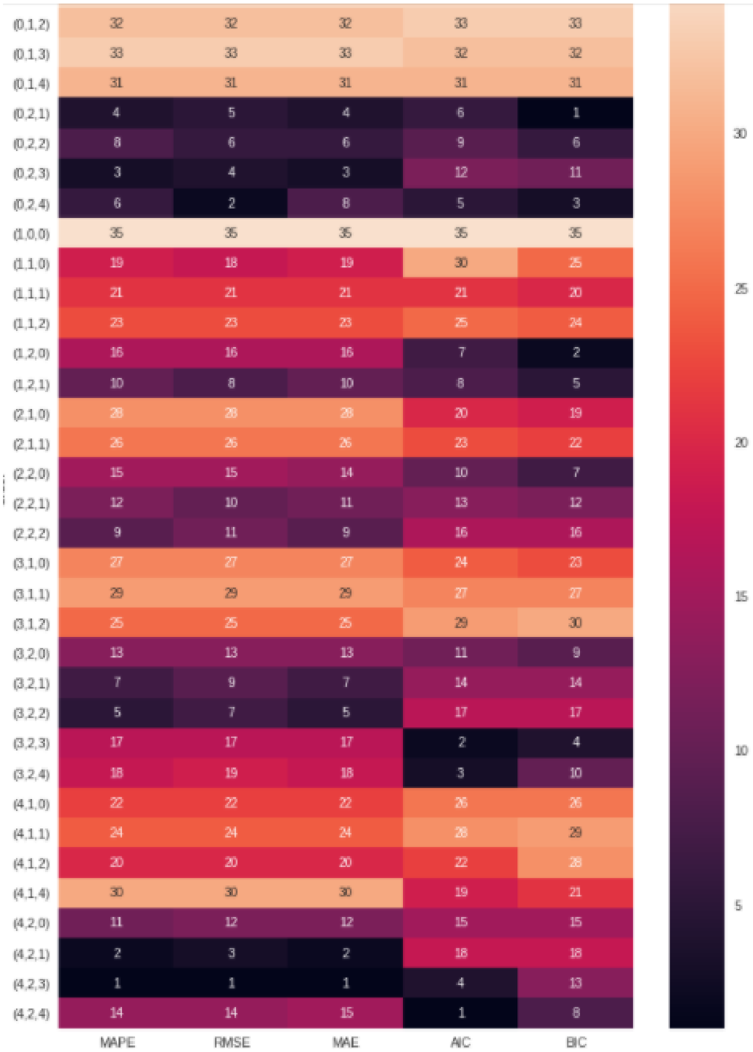


From the above plot, the best model made is ARIMA (1,2,1).

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                   D2.y   No. Observations:                   89
Model:                 ARIMA(1, 2, 1)   Log Likelihood                -902.118
Method:                       css-mle   S.D. of innovations           6102.551
Date:                Thu, 30 Apr 2020   AIC                           1812.235
Time:                        16:38:57   BIC                           1822.190
Sample:                             2   HQIC                          1816.248

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        865.5786    426.230      2.031      0.045      30.182    1700.975
ar.L1.D2.y    -0.0368      0.330     -0.112      0.911      -0.683       0.610
ma.L1.D2.y    -0.3214      0.317     -1.014      0.313      -0.942       0.300
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1          -27.1748           +0.0000j           27.1748            0.5000
MA.1            3.1113           +0.0000j            3.1113            0.0000
-----------------------------------------------------------------------------
```

Intuition aside, it is best to try all possible models to know the best parameters.

I tried all possible models to know the best parameters.




```python
print("Best ranked model on the basis of metrics MAE, RMSE and MAPE")
arima_results[arima_results["Order"]=="(4,2,3)"]
```

Best ranked model on the basis of metrics MAE, RMSE and MAPE

	Order	RMSE	MAE	MAPE	AIC	BIC
34	(4,2,3)	15406.543539	11868.741453	0.400241	1920.594373	1943.579265

```python
print("Best ranked model on the basis of AIC")
arima_results[arima_results["Order"]=="(4,2,4)"]
```

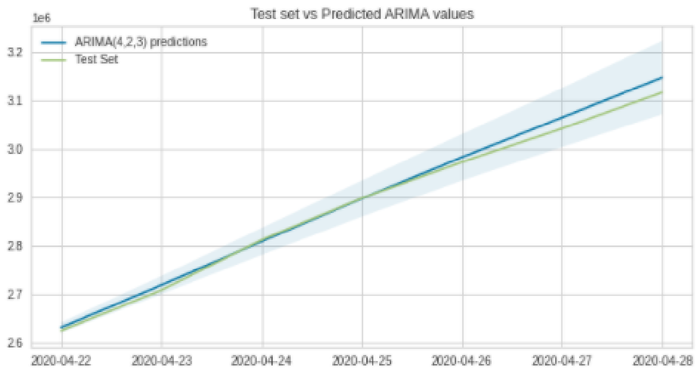
Best ranked model on the basis of AIC

	Order	RMSE	MAE	MAPE	AIC	BIC
35	(4,2,4)	22781.12522	19079.003875	0.649689	1914.963854	1940.502623

```python
print("Best ranked model on the basis of BIC")
arima_results[arima_results["Order"]=="(0,2,1)"]
```

Best ranked model on the basis of BIC

	Order	RMSE	MAE	MAPE	AIC	BIC
5	(0,2,1)	19747.533953	15767.277179	0.542033	1925.676276	1933.337907



For now, the best model is of the order `(4,2,3)` as it is ranked first for three out of five metrics

For now, the best model is of the order (4,2,3) as it is ranked for three out of five metrics.

Auto Arima

pmdarima package provides the *auto_arima* model that automatically returns the best model parameters.

```
Performing stepwise search to minimize aic
Fit ARIMA: (1, 2, 1)x(0, 0, 1, 7) (constant=True); AIC=1802.015, BIC=1814.458, Time=0.133 seconds
Fit ARIMA: (0, 2, 0)x(0, 0, 0, 7) (constant=True); AIC=1819.080, BIC=1824.058, Time=0.010 seconds
Fit ARIMA: (1, 2, 0)x(1, 0, 0, 7) (constant=True); AIC=1800.108, BIC=1810.062, Time=0.196 seconds
Fit ARIMA: (0, 2, 1)x(0, 0, 1, 7) (constant=True); AIC=1800.883, BIC=1810.837, Time=0.060 seconds
Fit ARIMA: (0, 2, 0)x(0, 0, 0, 7) (constant=False); AIC=1818.668, BIC=1821.157, Time=0.008 seconds
Fit ARIMA: (1, 2, 0)x(0, 0, 0, 7) (constant=True); AIC=1810.687, BIC=1818.153, Time=0.017 seconds
Fit ARIMA: (1, 2, 0)x(2, 0, 0, 7) (constant=True); AIC=1803.382, BIC=1815.825, Time=0.139 seconds
Fit ARIMA: (1, 2, 0)x(1, 0, 1, 7) (constant=True); AIC=1802.779, BIC=1815.222, Time=0.109 seconds
Fit ARIMA: (1, 2, 0)x(0, 0, 1, 7) (constant=True); AIC=1800.814, BIC=1810.768, Time=0.060 seconds
Fit ARIMA: (1, 2, 0)x(2, 0, 1, 7) (constant=True); AIC=1802.468, BIC=1817.399, Time=0.836 seconds
Near non-invertible roots for order (1, 2, 0)(2, 0, 1, 7); setting score to inf (at least one inverse root too clos
e to the border of the unit circle: 0.999)
Fit ARIMA: (0, 2, 0)x(1, 0, 0, 7) (constant=True); AIC=1815.060, BIC=1822.526, Time=0.042 seconds
Fit ARIMA: (2, 2, 0)x(1, 0, 0, 7) (constant=True); AIC=1802.571, BIC=1815.014, Time=0.073 seconds
Fit ARIMA: (1, 2, 1)x(1, 0, 0, 7) (constant=True); AIC=1803.470, BIC=1815.914, Time=0.082 seconds
Fit ARIMA: (0, 2, 1)x(1, 0, 0, 7) (constant=True); AIC=1797.719, BIC=1807.673, Time=0.219 seconds
Fit ARIMA: (0, 2, 1)x(0, 0, 0, 7) (constant=True); AIC=1811.254, BIC=1818.720, Time=0.029 seconds
Fit ARIMA: (0, 2, 1)x(2, 0, 0, 7) (constant=True); AIC=1805.357, BIC=1817.801, Time=0.121 seconds
Fit ARIMA: (0, 2, 1)x(1, 0, 1, 7) (constant=True); AIC=1802.880, BIC=1815.323, Time=0.116 seconds
Fit ARIMA: (0, 2, 1)x(2, 0, 1, 7) (constant=True); AIC=1803.571, BIC=1818.503, Time=0.350 seconds
Near non-invertible roots for order (0, 2, 1)(2, 0, 1, 7); setting score to inf (at least one inverse root too clos
e to the border of the unit circle: 0.992)
Fit ARIMA: (0, 2, 2)x(1, 0, 0, 7) (constant=True); AIC=1806.829, BIC=1819.272, Time=0.069 seconds
Fit ARIMA: (1, 2, 2)x(1, 0, 0, 7) (constant=True); AIC=1802.321, BIC=1817.253, Time=0.230 seconds
Total fit time: 2.949 seconds
```

```
                           Statespace Model Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:                   91
Model:             SARIMAX(0, 2, 1)x(1, 0, 0, 7)   Log Likelihood                -894.859
Date:                            Thu, 30 Apr 2020   AIC                           1797.719
Time:                                    16:40:53   BIC                           1807.673
Sample:                                         0   HQIC                          1801.731
                                             - 91
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept    383.1208    340.785      1.124      0.261    -284.805    1051.047
ma.L1         -0.4909      0.072     -6.784      0.000      -0.633      -0.349
ar.S.L7        0.4513      0.104      4.345      0.000       0.248       0.655
sigma2      3.456e+07      0.009   3.91e+09      0.000    3.46e+07    3.46e+07
===================================================================================
Ljung-Box (Q):                       37.48   Jarque-Bera (JB):                27.93
Prob(Q):                              0.58   Prob(JB):                         0.00
Heteroskedasticity (H):               6.06   Skew:                             0.47
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.58
===================================================================================
```

Warnings:

[1] Covariance matrix calculated using t he outer product of gradients (complex -step).

[2] Covariance matrix is singular or near -singular, with condition number 1.23e +25. Standard errors may be unstable.

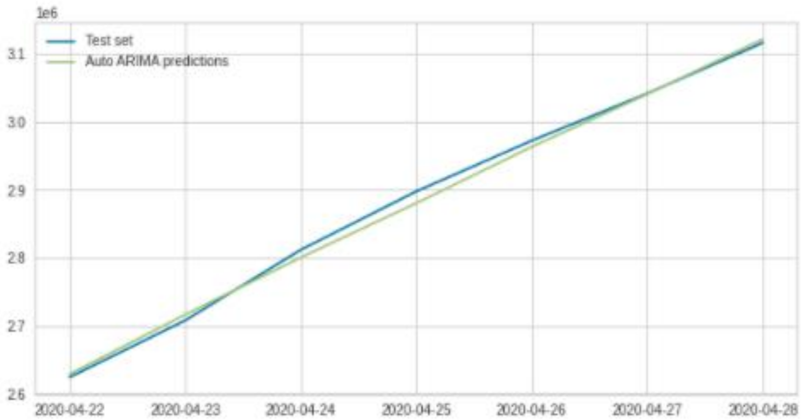
Root Mean Squared Error for best mod el: 9493.93046728813

Mean Absolute Error for best model: 8 122.663539528648

Mean Absolute Percentage Error for b est model: 6.590550809381285.

RMSE, MAE, AIC and BIC have significantly decreased, but MAPE has increased substantially when compared to previous best model.

The best model above is actually a SARIMA model, which is an ARIMA model with a seasonal component. SARIMA models have seasonal parameters (*P*,*D*,*Q*,*M*) where *M* is the frequency and *P*, *D*, *Q* are AR, differencing and MA terms for the seasonal component of the time series, respectively.



Linear Regression

While *ARIMA* and *SARIMA* give reasonably good results, a linear regression model with lag and moving average based features is also an effective way for forecasting data points.

	Date	NoOfCases	lag1	lag2	lag3	lag4	lag5	lag6	lag7	lag8	lag9	lag10
93	2020-04-24	2811603	2707742.0	2624107.0	2548821.0	2471759.0	2400843.0	2317339.0	2239723.0	2151872.0	2055506.0	1975581.0
94	2020-04-25	2897624	2811603.0	2707742.0	2624107.0	2548821.0	2471759.0	2400843.0	2317339.0	2239723.0	2151872.0	2055506.0
95	2020-04-26	2972363	2897624.0	2811603.0	2707742.0	2624107.0	2548821.0	2471759.0	2400843.0	2317339.0	2239723.0	2151872.0
96	2020-04-27	3041764	2972363.0	2897624.0	2811603.0	2707742.0	2624107.0	2548821.0	2471759.0	2400843.0	2317339.0	2239723.0
97	2020-04-28	3116398	3041764.0	2972363.0	2897624.0	2811603.0	2707742.0	2624107.0	2548821.0	2471759.0	2400843.0	2317339.0

After training and test sets for Linear model, starting from 15th observation as first 14 rows have null values due to lag and moving average features. Now, fitting the linear regression model, I've found that the linear model, all errors are negligible, and for that matter, its clearly the best model, below is the results.

```
Root Mean Squared Error for best model: 0.000020
Mean Absolute Error for best model: 0.000020
Mean Absolute Percentage Error for best model: 0.00000000069
```

Time Series Cross Validation

To be more confident about the results, I used time series cross validation techniques. The performance of a model can be tested by model training on variable training data sizes. Here, the *auto_arima* model has been trained on variable training sizes for the same test set.

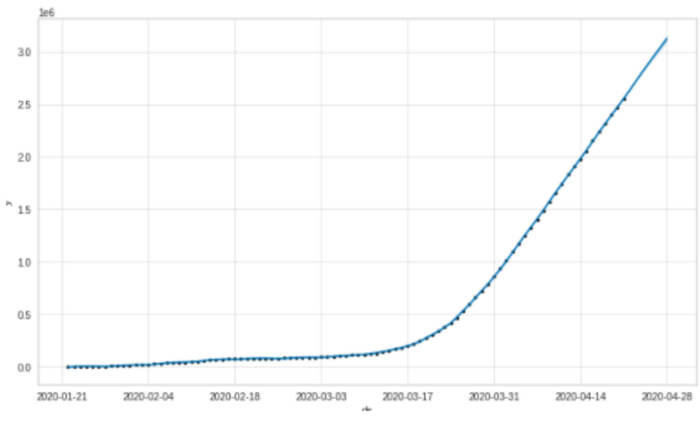
```
 20%|██         | 1/5 [00:03<00:12,  3.14s/it]
 Root Mean Squared Error by training on 13 weeks: 9493.93046728813
 Mean Absolute Error by training on 13 weeks: 8122.663539528648
 Mean Absolute Percentage Error by training on 13 weeks: 6.590550809381285
 40%|████       | 2/5 [00:06<00:09,  3.08s/it]
 Root Mean Squared Error by training on 11 weeks: 13954.883923230984
 Mean Absolute Error by training on 11 weeks: 11769.118267871972
 Mean Absolute Percentage Error by training on 11 weeks: 6.759070960347902
 60%|██████     | 3/5 [00:09<00:06,  3.04s/it]
 Root Mean Squared Error by training on 9 weeks: 15340.633687747173
 Mean Absolute Error by training on 9 weeks: 12787.226278849013
 Mean Absolute Percentage Error by training on 9 weeks: 6.776176592058174
 80%|████████   | 4/5 [00:11<00:03,  3.02s/it]
 Root Mean Squared Error by training on 7 weeks: 20113.02199390957
 Mean Absolute Error by training on 7 weeks: 14984.063318533024
 Mean Absolute Percentage Error by training on 7 weeks: 6.890685119219773
100%|██████████| 5/5 [00:15<00:00,  3.03s/it]
 Root Mean Squared Error by training on 5 weeks: 16726.082749871384
 Mean Absolute Error by training on 5 weeks: 13598.101015875514
 Mean Absolute Percentage Error by training on 5 weeks: 6.806585619579921

Average mean_absolute_percentage_error: 15125.710564409446
mean_absolute_percentage_error after averaging predictions: 14358.56807396486
Average mean_absolute_error: 12252.234484131634
mean_absolute_error after averaging predictions: 12102.975187067608
Average mean_absolute_percentage_error: 6.764613820117411
mean_absolute_percentage_error after averaging predictions: 6.7639018642874555
```

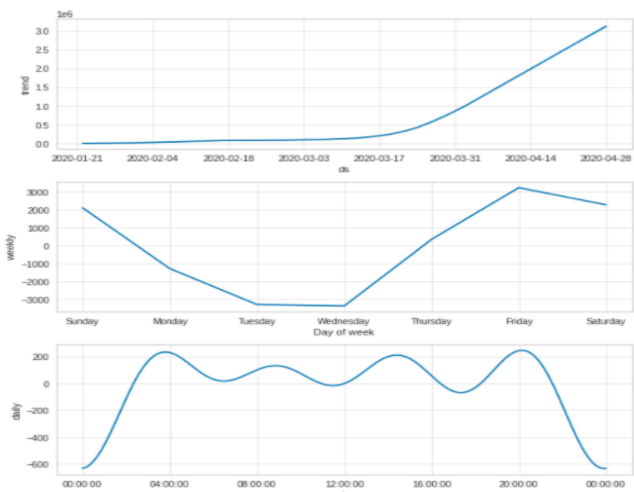
As I decreased training data, errors increase by a significant margin, therefore *auto_arima* models will be more effective with more reliable performance with a larger dataset.

Facebook Prophet

Prophet is a procedure for time series forecasting developed by Facebook. It works best for data with strong seasonal effects. And below are the results:

Root Mean Squared Error for Prophet model: 8984.997092967915
Mean Absolute Error for Prophet model: 8394.739396519892
Mean Absolute Percentage Error for Prophet model: 6.584128076279673





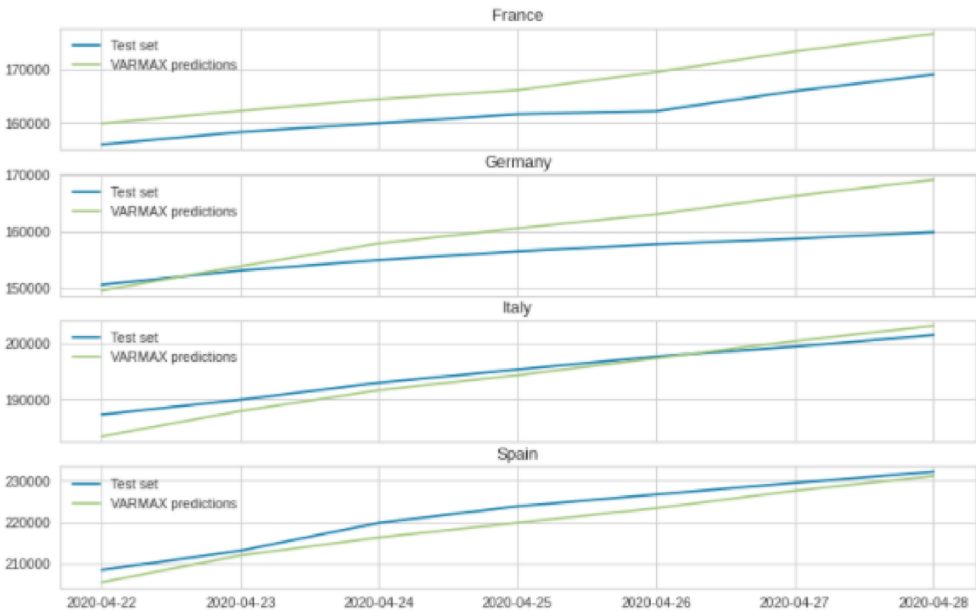
From above plots, its performance is very comparable to the *auto_arima* model but it is a much simp ler implementation. Weekly component shows a drop-in cases on Tuesday and Wednesday before in creasing again during the weekend.

Multiple Time Series Forecasting

VARMAX model

Time series with multiple columns can also be forecasted using the VARMAX model (Vector Autoregressive Moving Average with exogenous regressors model). These are like ARIMA models but without a degree of differencing parameter and can-do forecasting for multiple time series columns. This has been used below to forecast future predictions for countries in Cluster 2, (i.e., France, Germany, Italy and Spain).

```
  0%|          | 0/4 [00:00<?, ?it/s]
{'best_rmse': [31055.611836776377, (1, 1)], 'best_mae': [28376.771206913243, (1, 1)], 'best_mape': [15.176937578688
099, (1, 1)], 'best_aic': [7383.616470094896, (1, 1)], 'best_bic': [7499.116007394671, (1, 1)]}
{'best_rmse': [16539.1084774156, (1, 2)], 'best_mae': [14806.070706736675, (1, 2)], 'best_mape': [7.793877372116833
5, (1, 2)], 'best_aic': [7176.7581532110735, (1, 2)], 'best_bic': [7332.431442615118, (1, 2)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6755.024579927107, (1, 3)], 'best_bic': [6950.871621435421, (1, 3)]}
 25%|██        | 1/4 [01:19<03:59, 79.84s/it]
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6658.0046857890275, (1, 4)], 'best_bic': [6894.025479401611, (1, 4)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6658.0046857890275, (1, 4)], 'best_bic': [6894.025479401611, (1, 4)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6658.0046857890275, (1, 4)], 'best_bic': [6894.025479401611, (1, 4)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6658.0046857890275, (1, 4)], 'best_bic': [6894.025479401611, (1, 4)]}
 50%|█████     | 2/4 [03:26<03:07, 93.91s/it]
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6658.0046857890275, (1, 4)], 'best_bic': [6894.025479401611, (1, 4)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
 75%|███████▌  | 3/4 [06:45<02:05, 125.37s/it]
{'best_rmse': [6807.051271182086, (1, 3)], 'best_mae': [4604.351218417904, (1, 3)], 'best_mape': [2.25645459308279
4, (1, 3)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
{'best_rmse': [4427.769955408083, (4, 1)], 'best_mae': [3981.7856304661204, (4, 1)], 'best_mape': [2.17763638874155
8, (4, 1)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
{'best_rmse': [4427.769955408083, (4, 1)], 'best_mae': [3981.7856304661204, (4, 1)], 'best_mape': [2.17763638874155
8, (4, 1)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
{'best_rmse': [4427.769955408083, (4, 1)], 'best_mae': [3981.7856304661204, (4, 1)], 'best_mape': [2.17763638874155
8, (4, 1)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
100%|██████████| 4/4 [11:13<00:00, 168.49s/it]
{'best_rmse': [4288.736626972232, (4, 4)], 'best_mae': [3542.6477175392306, (4, 4)], 'best_mape': [2.05578685614399
6, (4, 4)], 'best_aic': [6090.534440349377, (3, 1)], 'best_bic': [6286.381481857692, (3, 1)]}
```



Although it has taken a long time to select the best model, the individual forecasts don't deviate much.

Conclusion

Data is extensively collected worldwide in the form of time series. Some examples include daily opening and closing prices of stocks, hourly air temprature, monitoring heart rate continuously, hourly weather data, monthly sales figures and more. In future, I look forward to working on others