BIO823-Final-Project

November 22, 2021

1 COVID-19 Forecasting

1.0.1 BIO823 Final Project

Shannon Murphy, Jennie Sun, Malcolm Smith Fraser

1.0.2 Project description

https://github.com/jenniesun/covid_forcasting/blob/main/README.md

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
%cd "/content/drive/MyDrive/bios823-final"
```

```
/content/drive/MyDrive/bios823-final
```

[3]:
```
!ls "/content/drive/MyDrive/bios823-final"
```

```
 AWS_casesDHPC.csv
 AWS_casesdurham.csv
 AWS_casesNC.csv
 AWS_casesUSA.csv
'DHHS Cases & Deaths By County.csv'
 DHHS_HOSPITAL_BEDS_VENTILATORS_REGION.xls
 DHHS_HOSPITAL_BEDS_VENTILATORS_STATE.csv
 DHHS_HOSPITAL_METRICS_REGION.csv
 DHHS_HOSPITAL_METRICS_REGION.xls
 DHHS_HOSPITAL_METRICS_STATE.csv
'DHHS: Positive by County-All.csv'
 logs.log
```

[4]:
```
!pwd
```

```
/content/drive/MyDrive/bios823-final
```

1.0.3 Data Processing

[22]:
```
import pandas as pd
```

[23]:
```
cases_deaths = pd.read_csv('AWS_casesDHPC.csv')
cases_deaths = cases_deaths.drop(columns= [ "Confirmed (window average)␣
↪(CUSTOM)", "Deaths (window average) (CUSTOM)"])
cases_deaths['Date'] = pd.to_datetime(cases_deaths['Date']).dt.date
cases_deaths['Date'] = cases_deaths['Date'].astype('str')
```

[24]:
```
beds_vents = pd.read_excel('DHHS_HOSPITAL_BEDS_VENTILATORS_REGION.xls')
beds_ventsDHPC = beds_vents[beds_vents["Coalition"] == "DHPC"]
hosp = pd.read_excel('DHHS_HOSPITAL_METRICS_REGION.xls')
hospDHPC = hosp[hosp["Coalition"] == "DHPC"]
hospall = pd.merge(hospDHPC, beds_ventsDHPC.drop(columns = ["Coalition"]), how␣
↪= 'outer', on = 'Date')
hospall['Date'] = hospall['Date'].astype('str')
```

[25]:
```
df = pd.merge(hospall, cases_deaths, how = 'outer', on = 'Date')
df.head()
```

[25]:
```
   Index_x  ... Deaths (daily growth) (CUSTOM)
0      2.0  ...                            0.0
1     10.0  ...                            2.0
2     18.0  ...                            0.0
3     26.0  ...                            0.0
4     34.0  ...                            6.0

[5 rows x 16 columns]
```

1.0.4 EDA

[19]:
```
#!pip install https://github.com/pandas-profiling/pandas-profiling/archive/
↪master.zip --quiet
```

[20]:
```
#import pandas_profiling as pp
```

[21]:
```
#pp.ProfileReport(df)
```

[26]:
```
df_new = df[:518].copy()
```

[27]:
```
df_new.isna().any()
```

[27]:
```
Index_x                                          False
Date                                             False
Coalition                                        False
Hospitalizations                                 False
Adult ICU COVID-19 Patients                      False
Confirmed Patient Admitted - Last 24 Hours       False
Hospitalized and Ventilated COVID Inpatient Count  True
Index_y                                          False
```

```
Ventilators In Use                                      False
Ventilators Available                                   False
ICU Beds In Use                                         False
ICU Empty Staffed Beds                                  False
Inpatient Beds In Use                                   False
Inpatient Empty Staffed Beds                            False
Confirmed (daily growth) (CUSTOM)                       False
Deaths (daily growth) (CUSTOM)                          False
dtype: bool
```

[28]:
```
df_new.info()
```

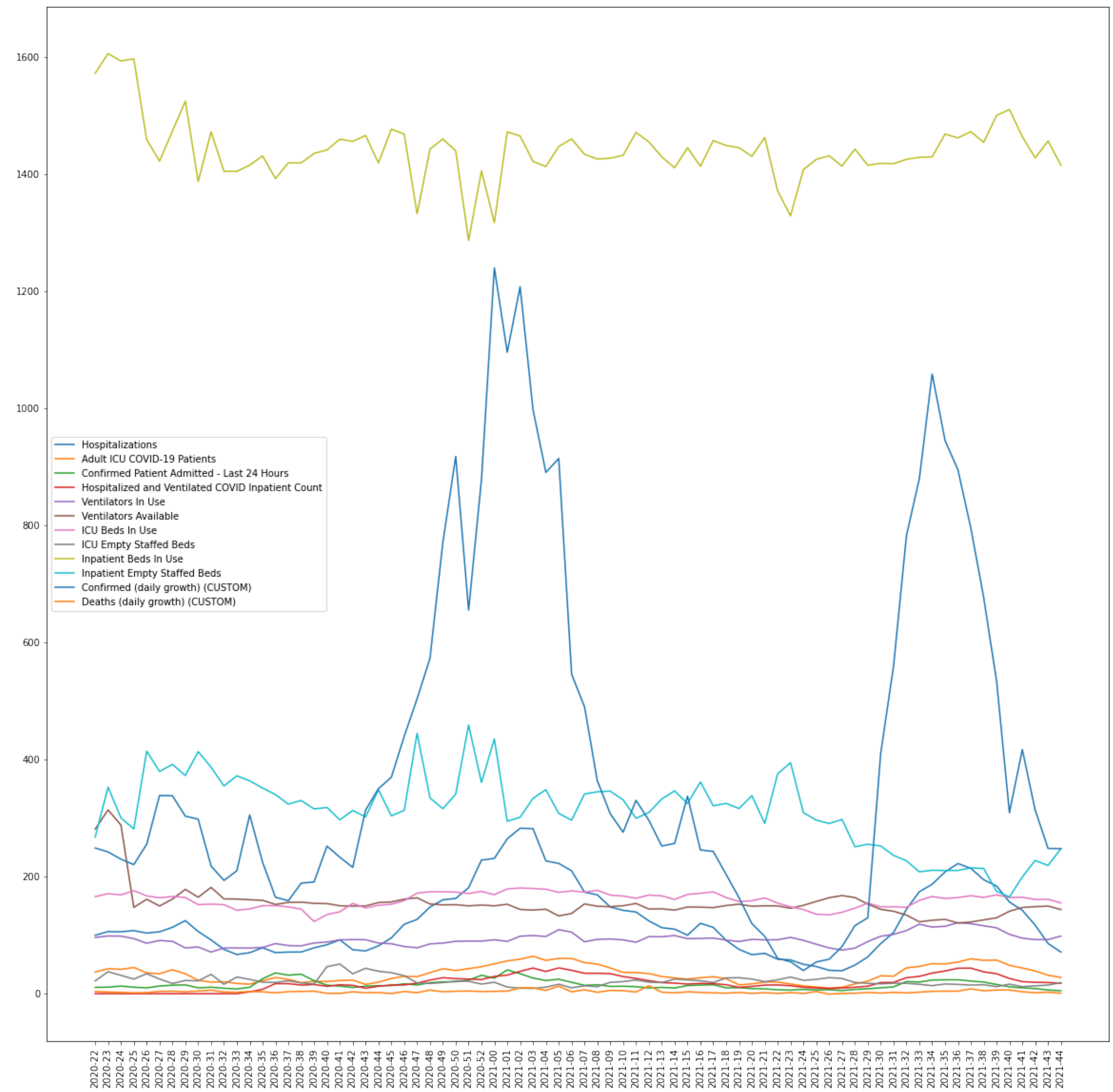
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 518 entries, 0 to 517
Data columns (total 16 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   Index_x                                       518 non-null    float64
 1   Date                                          518 non-null    object
 2   Coalition                                     518 non-null    object
 3   Hospitalizations                              518 non-null    float64
 4   Adult ICU COVID-19 Patients                   518 non-null    float64
 5   Confirmed Patient Admitted - Last 24 Hours    518 non-null    float64
 6   Hospitalized and Ventilated COVID Inpatient Count 434 non-null    float64
 7   Index_y                                       518 non-null    float64
 8   Ventilators In Use                            518 non-null    float64
 9   Ventilators Available                         518 non-null    float64
 10  ICU Beds In Use                               518 non-null    float64
 11  ICU Empty Staffed Beds                        518 non-null    float64
 12  Inpatient Beds In Use                         518 non-null    float64
 13  Inpatient Empty Staffed Beds                  518 non-null    float64
 14  Confirmed (daily growth) (CUSTOM)             518 non-null    float64
 15  Deaths (daily growth) (CUSTOM)                518 non-null    float64
dtypes: float64(14), object(2)
memory usage: 68.8+ KB
```

[29]:
```
df_new.loc[:,'Date'] = pd.to_datetime(df_new['Date'],
 ↪infer_datetime_format=True)
df_new['Year-Week'] = df_new['Date'].dt.strftime('%Y-%U')
df_new = df_new.sort_values('Date',ascending=False).fillna(0).
 ↪drop(['Index_x','Index_y'],axis=1)
df_week = df_new.groupby('Year-Week').mean()
df_week.shape, df_week.columns
```

[29]:
```
((76, 12), Index(['Hospitalizations', 'Adult ICU COVID-19 Patients',
        'Confirmed Patient Admitted - Last 24 Hours',
        'Hospitalized and Ventilated COVID Inpatient Count',
        'Ventilators In Use', 'Ventilators Available', 'ICU Beds In Use',
```

```
'ICU Empty Staffed Beds', 'Inpatient Beds In Use',
 'Inpatient Empty Staffed Beds', 'Confirmed (daily growth) (CUSTOM)',
 'Deaths (daily growth) (CUSTOM)'],
dtype='object'))
```

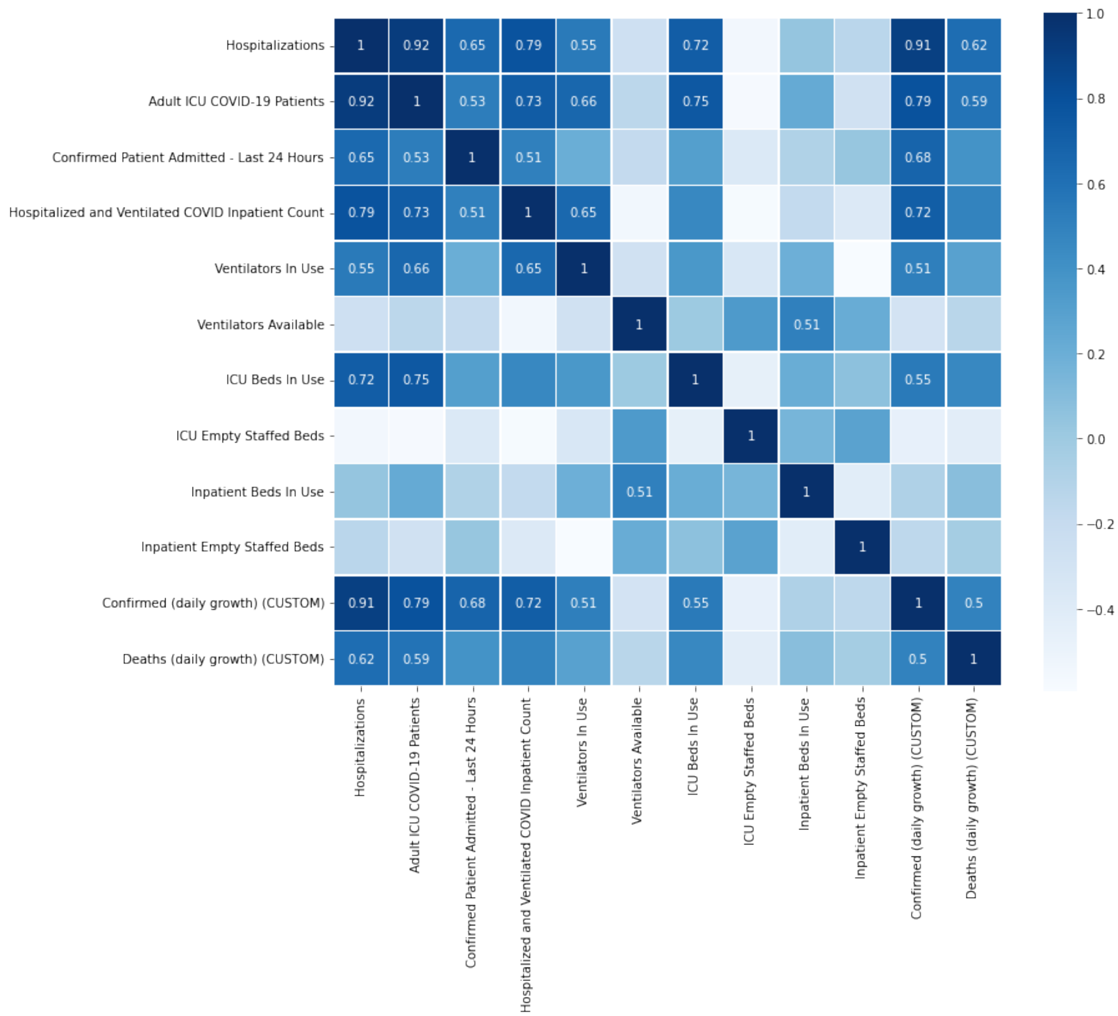
[31]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(20,20))
p = plt.plot(df_week)
plt.legend(df_week.columns)
t = plt.xticks(rotation=90)
```



Correlation Heatmap

[32]:
```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

[33]:
```
f, ax = plt.subplots(figsize=(12, 10))

# calculate the correlation matrix
corr = df_week.corr()

# plot the heatmap
sns.heatmap(corr, cmap="Blues", annot=True,
            square=True,
                linewidth=.5, ax=ax,
        xticklabels=corr.columns,
        yticklabels=corr.columns)

for t in ax.texts:
    if float(t.get_text())>=0.5:
        t.set_text(t.get_text()) #if the value is greater than 0.4 then I set␣
 ↪the text
    else:
        t.set_text("") # if not it sets an empty text
```



1.0.5 Model building and forecasting

Univariate: 1. Deaths (daily growths)

Multivariate: * Ability to forecast future deaths will be compared to the univariate model

1.0.6 Univariate Analysis on the Growth of COVID-19 Death Count

Reference: * https://www.kaggle.com/prashant111/complete-guide-on-time-series-analysis-in-python * https://medium.com/swlh/temperature-forecasting-with-arima-model-in-python-427b2d3bcb53

In this section, we start our analysis on the growth of covid death count with a univariate time series analysis. Since it's a univariate time-series forecasting, we are only using two variables in which one is time and the other is the field to forecaste. In this case, it is the `Deaths (daily growth) (CUSTOM)'` variable in the dataset.

We start our analysis by doing EDA to detect if there's any trend and seasonality pattern of changes with regards to time. These are shown in the the autocorrelation, seasonality, and lag plots below.

We then use an ARIMA (Auto-Regressive Integrated Moving Average) model to conduct the forecasting task. ARIMA is a class of models that based on its own lags and the lagged forecast errors. Any non-seasonal time series that exhibits patterns and is not a random white noise can be modelled with ARIMA models.

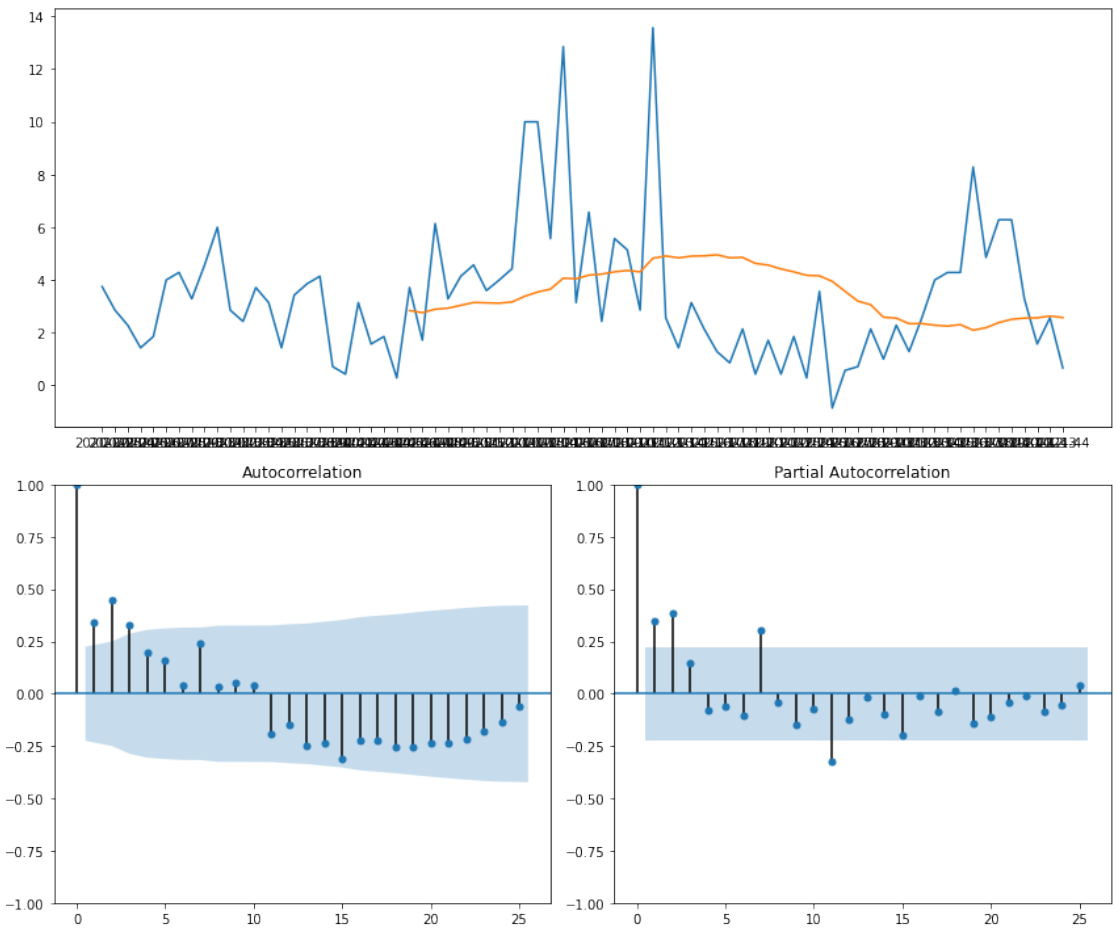
Using the best order returned by the `auto_arima` package for model training, we were able to achieve a RMSE of 2.18 as a result.

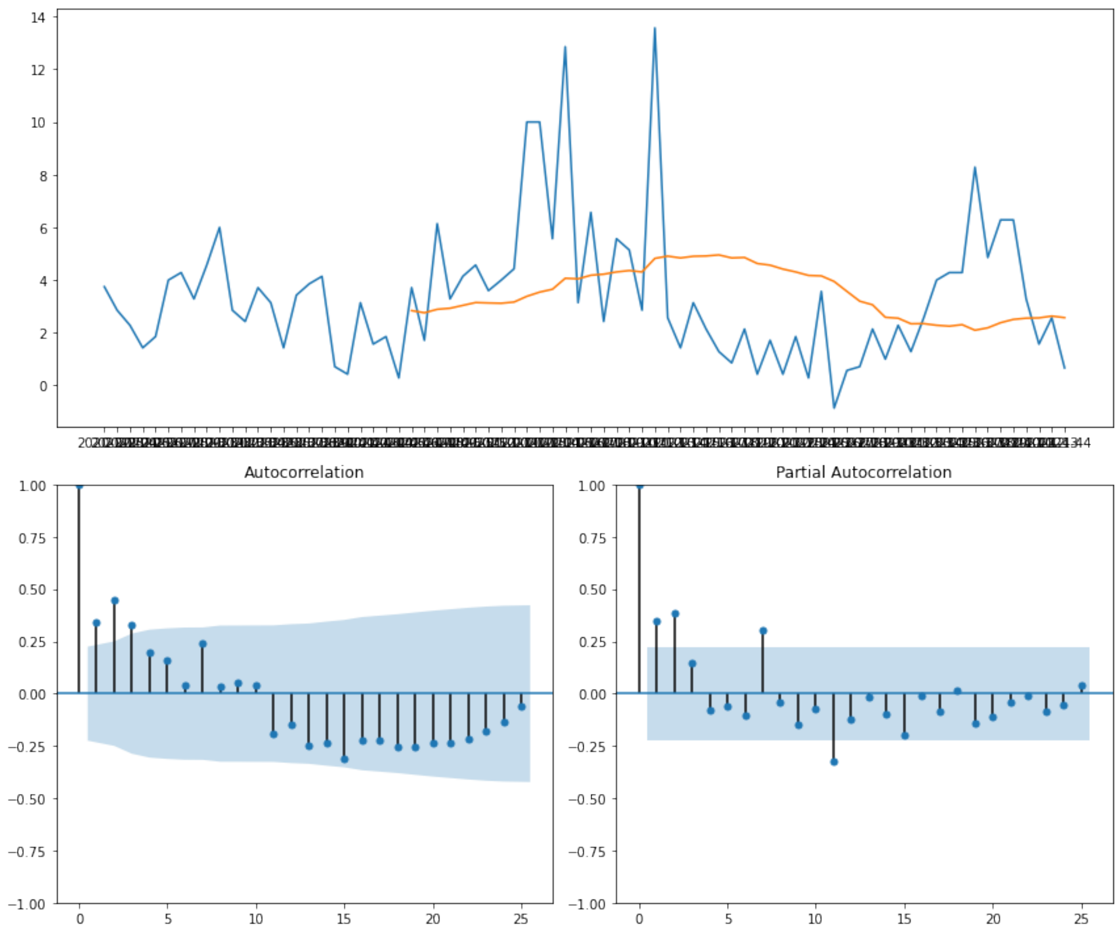
Autocorrelation and Partial Autocorrelation Functions Autocorrelation is simply the correlation of a series with its own lags. If a series is significantly autocorrelated, that means, the previous values of the series (lags) may be helpful in predicting the current value. Partial Autocorrelation also conveys similar information but it conveys the pure correlation of a series and its lag, excluding the correlation contributions from the intermediate lags.

[34]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
def plot_ts(ts, lags=None):
    fig = plt.figure(figsize = (12, 10))
    ax1 = plt.subplot2grid((2, 2), (0, 0), colspan=2)
    ax2 = plt.subplot2grid((2, 2), (1, 0))
    ax3 = plt.subplot2grid((2, 2), (1, 1))

    ax1.plot(ts)
    ax1.plot(ts.rolling(window=lags).mean())
    plot_acf(ts, ax=ax2, lags=lags)
    plot_pacf(ts, ax=ax3, lags=lags)

    plt.tight_layout()
    return fig

plot_ts(df_week['Deaths (daily growth) (CUSTOM)'], lags=25)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  FutureWarning,
```

[34]:


Autocorrelation
Partial Autocorrelation

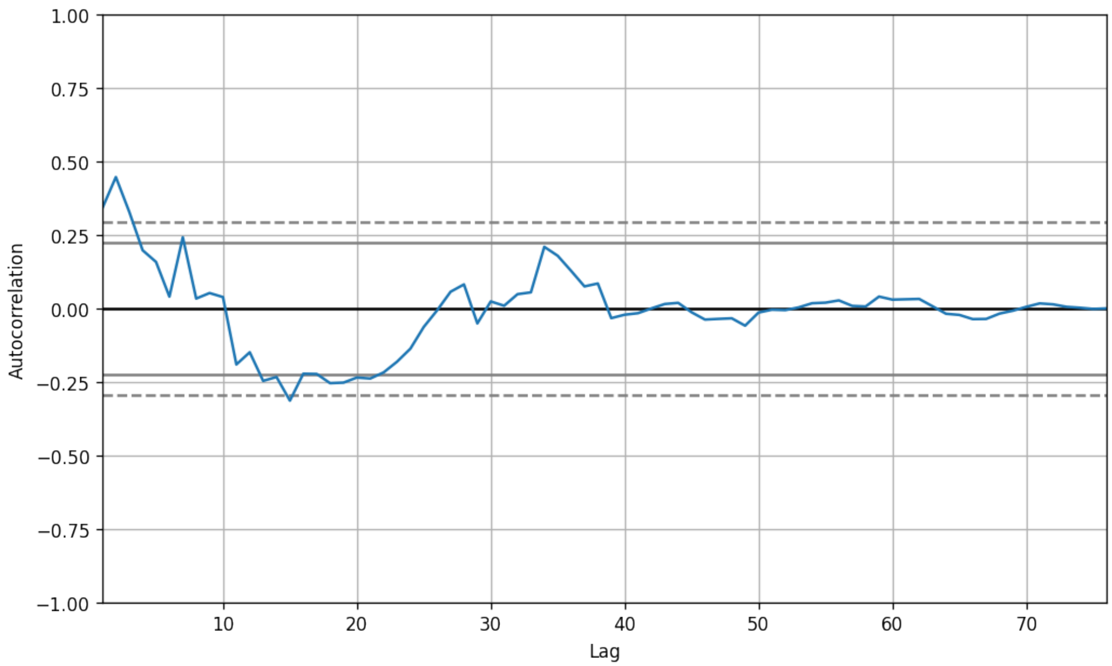


Test for seasonality of a time series? The common way to test for seasonality of a time series is to plot the series and check for repeatable patterns in fixed time intervals. So, the types of seasonality is determined by the clock or the calendar. * Hour of day * Day of month * Weekly * Monthly * Yearly

However, if we want a more definitive inspection of the seasonality, use the Autocorrelation Function (ACF) plot. There is a strong seasonal pattern, the ACF plot usually reveals definitive repeated spikes at the multiples of the seasonal window.

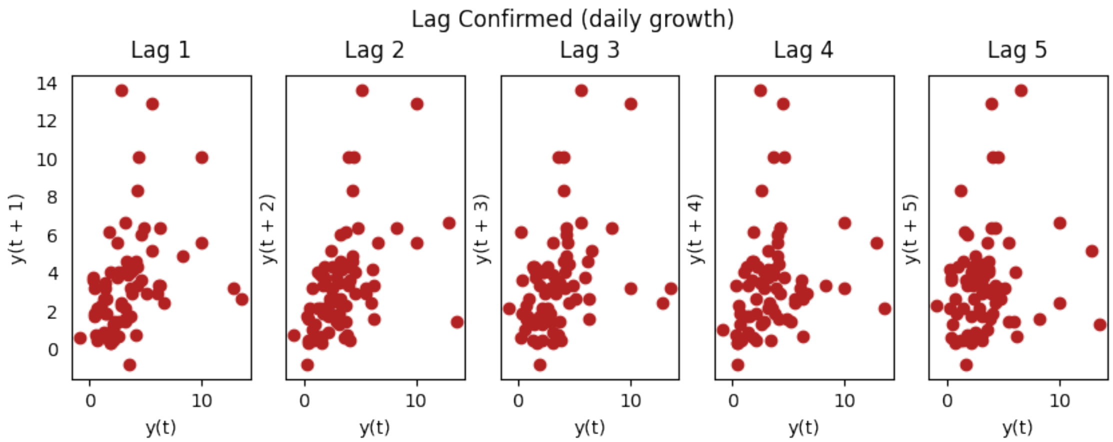
[30]:
```
# Test for seasonality
from pandas.plotting import autocorrelation_plot

# Draw Plot
plt.rcParams.update({'figure.figsize':(10,6), 'figure.dpi':120})
autocorrelation_plot(df_week['Deaths (daily growth) (CUSTOM)'].tolist())
```

[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8bad83de50>
```



Lag Plots A Lag plot is a scatter plot of a time series against a lag of itself. It is normally used to check for autocorrelation. If there is any pattern existing in the series, the series is autocorrelated. If there is no such pattern, the series is likely to be random white noise.

[31]:
```
from pandas.plotting import lag_plot
plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})

n = 5
# Plot
fig, axes = plt.subplots(1, n, figsize=(10,3), sharex=True, sharey=True,␣
 ↪dpi=100)
for i, ax in enumerate(axes.flatten()[:n]):
    lag_plot(df_week['Deaths (daily growth) (CUSTOM)'], lag=i+1, ax=ax,␣
 ↪c='firebrick')
    ax.set_title('Lag ' + str(i+1))

fig.suptitle('Lag Confirmed (daily growth)', y=1.05)
plt.show()
```



ARIMA Model - figuring out the best order

[ ]:
```
#!setup.py install
```

[16]:
```
#!pip install pmdarima
```

[17]:
```
#!pip install scipy==1.2.0
```

[18]:
```
#!pip install statsmodels
```

[32]:
```
from pmdarima import auto_arima
stepwise_fit = auto_arima(df_week['Deaths (daily growth) (CUSTOM)'], trace=True,
                          suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=349.393, Time=0.09 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=366.852, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=359.373, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=363.443, Time=0.03 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=439.475, Time=0.01 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=348.800, Time=0.06 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=356.376, Time=0.04 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=352.816, Time=0.05 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=349.156, Time=0.07 sec
 ARIMA(0,0,3)(0,0,0)[0] intercept   : AIC=349.002, Time=0.06 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=351.051, Time=0.05 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=351.041, Time=0.18 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=355.537, Time=0.04 sec

Best model:  ARIMA(1,0,2)(0,0,0)[0] intercept
Total fit time: 0.726 seconds
```

Train the Model

[53]:
```
#### Split The Dataset
df_train, df_test= df_week[0:-5], df_week[-5:]
print(df_train.shape, df_test.shape)
```

```
(71, 12) (5, 12)
```

[51]:
```
import warnings
warnings.filterwarnings("ignore")
```

[52]:
```
from statsmodels.tsa.arima.model import ARIMA
```

[54]:
```
model_deaths=ARIMA(df_train['Deaths (daily growth) (CUSTOM)'],order=(1,0,2))
model_deaths=model_deaths.fit()
model_deaths.summary()
```

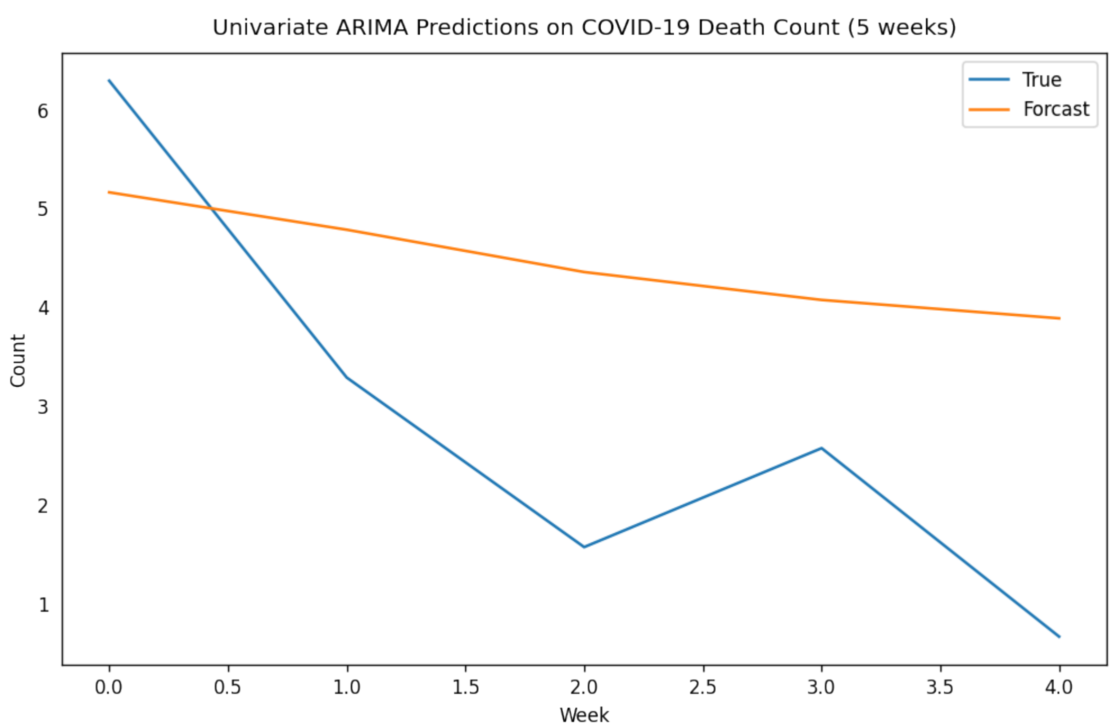
[54]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:     Deaths (daily growth) (CUSTOM)   No. Observations:
71
Model:                               ARIMA(1, 0, 2)   Log Likelihood
-159.231
Date:                              Mon, 22 Nov 2021   AIC
328.463
Time:                                      20:18:58   BIC
339.776
Sample:                                           0   HQIC
332.961
                                               - 71
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          3.5249      0.952      3.702      0.000       1.659       5.391
ar.L1          0.6590      0.179      3.688      0.000       0.309       1.009
ma.L1         -0.5335      0.209     -2.548      0.011      -0.944      -0.123
ma.L2          0.3799      0.105      3.620      0.000       0.174       0.586
sigma2         5.1482      0.613      8.402      0.000       3.947       6.349
================================================================================
===
Ljung-Box (L1) (Q):                  0.03   Jarque-Bera (JB):
47.86
Prob(Q):                             0.87   Prob(JB):
0.00
Heteroskedasticity (H):              0.91   Skew:
1.13
Prob(H) (two-sided):                 0.82   Kurtosis:
```

```
6.33
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

1.0.7 Check How Good The Model Is

[57]:
```python
start=len(df_train)
end=len(df_train)+len(df_test)-1
pred=model_deaths.predict(start=start,end=end).rename('ARIMA Predictions')
pred.reset_index(inplace=True, drop=True)#.plot(legend=True)
test = df_test['Deaths (daily growth) (CUSTOM)']#.plot(legend=True)
test.reset_index(drop=True, inplace=True)
test.plot(legend=True, label = 'True')
pred.plot(legend=True, label = "Forcast")
plt.title('Univariate ARIMA Predictions on COVID-19 Death Count (5 weeks)')
plt.xlabel('Week')
plt.ylabel('Count')
```

[57]:
```
Text(0, 0.5, 'Count')
```



Check Accuracy Metric

[38]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
df_test['Deaths (daily growth) (CUSTOM)'].mean()
rmse=sqrt(mean_squared_error(pred,df_test['Deaths (daily growth) (CUSTOM)']))
print(rmse)
```

```
2.1833080251452355
```

1.0.8 Multivariate Modeling

In this section we compare the effectiveness of the univariate model to that of a multivariate model at forecasting deaths. We explores three approaches to the multivariate model. * Model with only the features highly correlated with deaths * Model with only the features with low correlation to deaths * Model with all features

Since the best performace of the univariate model used a lag of 1 week, we evaluate all our multvariate models with a lag of one week - though an exploratory analysis also showed that lag=1 was the optimal choice for forecasting deaths with the multivariate models.

As with the univariate modeling, we reserved the last 5 weeks of the data for testing and trained on the rest.

Results The multivariate model with only the low correlated features was the best at forecasting future deaths - outperforming the univariate model (rmse 2.183) and all other multivariate models with an rmse of 1.576 across the 5 forecasted weeks. The multivariate models with all features and the highly correlated features both underperformed as compared to the univariate model with rmse values of 8.993 and 3.379 respectively.

Intuitively, this result is rather interesting. One would expect that the multivaraite model with features uncorrelated with deaths would yeild results similar to the multivariate model. Another interesting finding with the multivariate model with the highly correlated features is that although more of the variance in the distribution of Deaths is accounted for by other features - these relationships make the model worse at forecasting future values.

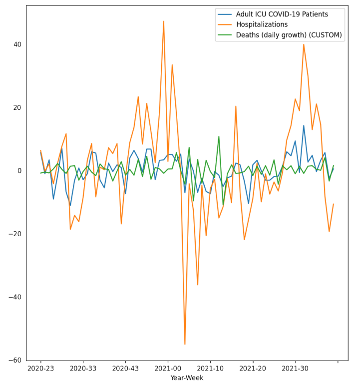
[39]:
```
## Features with low correlation to Deaths
df_multivar = df_week[['Ventilators Available',
                       "ICU Empty Staffed Beds",
                       "Inpatient Beds In Use",
                       "Inpatient Empty Staffed Beds",
                       "Confirmed (daily growth) (CUSTOM)",
                       "Deaths (daily growth) (CUSTOM)"]]

## Features with high correlation to Deaths
df_multivar_deaths = df_week[['Adult ICU COVID-19 Patients',
                              'Hospitalizations',
                              "Deaths (daily growth) (CUSTOM)"]]
```

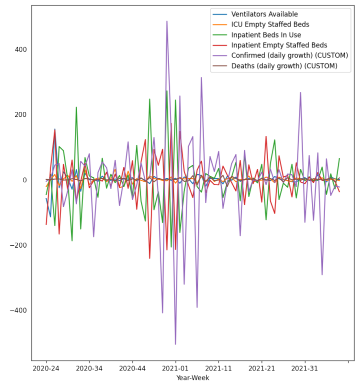
[40]:
```
## Check for stationarity

from statsmodels.tsa.stattools import adfuller
```

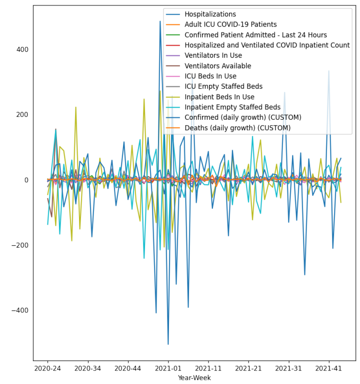
```
def adf_test(ts, signif=0.05):
    dftest = adfuller(ts, autolag='AIC')
    adf = pd.Series(dftest[0:4], index=['Test Statistic','p-value','# Lags','#␣
↪Observations'])
    for key,value in dftest[4].items():
       adf['Critical Value (%s)'%key] = value
    print (adf)

    p = adf['p-value']
    if p <= signif:
        print(f" Series is Stationary")
    else:
        print(f" Series is Non-Stationary")

df_train_multivar_deaths, df_test_multivar_deaths= df_multivar_deaths[0:-5],␣
↪df_multivar_deaths[-5:]
df_train_multivar, df_test_multivar= df_multivar[0:-5], df_multivar[-5:]
df_train_week, df_test_week= df_week[0:-5], df_week[-5:]

#apply adf test on the series
# for col in df_train_multivar_deaths.columns:
#   print(col)
#   adf_test(df_train[col])
#   print('\n')
```

[41]:
```
## Differencing to stationize data

# df_train_multivar_deaths 1st difference
df_differenced_deaths = df_train_multivar_deaths.diff().dropna()

# df_train_multivar 2nd difference
df_differenced_multivar = df_train_multivar.diff().dropna().diff().dropna()

# df_train_week 2nd differnce
df_differenced_week = df_week.diff().dropna().diff().dropna()


fig, ax = plt.subplots(1,3,figsize=(30,10))
df_differenced_deaths.plot(ax=ax[0])
df_differenced_multivar.plot(ax=ax[1])
df_differenced_week.plot(ax=ax[2])
```

[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8bb7910e90>







[42]:
```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.api import VAR
```

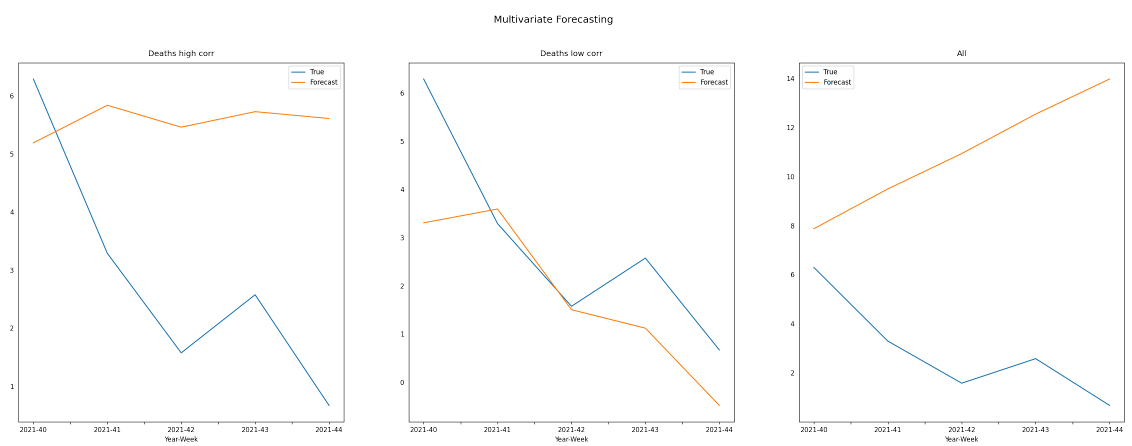
[43]:
```python
model_deaths = VAR(df_differenced_deaths)
results_deaths = model_deaths.fit(maxlags=1, ic='aic')

model_multivar = VAR(df_differenced_multivar)
results_multivar = model_multivar.fit(maxlags=1, ic='aic')

model_week = VAR(df_differenced_week)
results_week = model_week.fit(maxlags=1, ic='aic')
```

[45]:
```python
# forecasting
def forecast(results, df_test):
  pred = results.forecast(results.fittedvalues.values, steps=5) # forcast 5␣
↪steps out
  df_forecast = pd.DataFrame(pred, index=df_test.index[:], columns=df_test.
↪columns + '_1d')
  return df_forecast

df_forecast_deaths   = forecast(results_deaths, df_test_multivar_deaths)
df_forecast_multivar = forecast(results_multivar,df_test_multivar)
df_forecast_week     = forecast(results_week, df_test_week)


# inverting the difference transformations for the forcasted values
def invert_transformation(df_train, df_forecast, second_diff=False):
    """Revert back the differencing to get the forecast to original scale."""
    df_fc = df_forecast.copy()
    columns = df_train.columns
    for col in columns:
        # Roll back 2nd Diff
        if second_diff:
```

```
            df_fc[str(col)+'_1d'] = (df_train[col].iloc[-1]-df_train[col].
 ↪iloc[-2]) + df_fc[str(col)+'_1d'].cumsum()
        # Roll back 1st Diff
        df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] +␣
 ↪df_fc[str(col)+'_1d'].cumsum()
    return df_fc

# get inverted results in a dataframe
df_results_deaths = invert_transformation(df_train_multivar_deaths,␣
 ↪df_forecast_deaths, second_diff=False)
df_results_multivar = invert_transformation(df_train_multivar,␣
 ↪df_forecast_multivar, second_diff=True)
df_results_weeks = invert_transformation(df_train_week, df_forecast_week,␣
 ↪second_diff=True)
```

[49]:
```
fig, ax = plt.subplots(1,3,figsize=(30,10))
plt.suptitle("Multivariate Forecasting", size=15)
df_test_multivar_deaths['Deaths (daily growth) (CUSTOM)'].
 ↪plot(ax=ax[0],label='True',legend=True, title='Deaths high corr')
df_results_deaths.filter(like='forecast')['Deaths (daily growth)␣
 ↪(CUSTOM)_forecast'].plot(ax=ax[0],label='Forecast',legend=True)

df_test_multivar['Deaths (daily growth) (CUSTOM)'].
 ↪plot(ax=ax[1],label='True',legend=True, title='Deaths low corr')
df_results_multivar.filter(like='forecast')['Deaths (daily growth)␣
 ↪(CUSTOM)_forecast'].plot(ax=ax[1],label='Forecast',legend=True)

df_test_week['Deaths (daily growth) (CUSTOM)'].
 ↪plot(ax=ax[2],label='True',legend=True, title="All")
df_results_weeks.filter(like='forecast')['Deaths (daily growth)␣
 ↪(CUSTOM)_forecast'].plot(ax=ax[2],label='Forecast',legend=True)
```

[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b99f18450>
```



[47]:
```
#####
# RMSE between forcasted and real values for deaths (and overall) across the␣
↪last 5 weeks
#####

def get_error(results, df_test):
    p_error = (results.values - df_test[:].values)/df_test[:].values
    mse = mean_squared_error(df_test, results,multioutput='raw_values',␣
↪squared=False)
    return mse


error_deaths   = get_error(df_results_deaths.filter(like    = 'forecast'),␣
↪df_test_multivar_deaths)
error_multivar = get_error(df_results_multivar.filter(like = 'forecast'),␣
↪df_test_multivar)
error_weeks    = get_error(df_results_weeks.filter(like     = 'forecast'),␣
↪df_test_week)

print(f"==>Death high corr RMSE\n Overall: {abs(error_deaths.mean()).mean():.
↪4f}\n{list(zip(df_test_multivar_deaths,error_deaths))[-1]}\n")
print(f"==>Death low corr RMSE\n Overall: {abs(error_multivar.mean()).mean():.
↪4f}\n{list(zip(df_test_multivar,error_multivar))[-1]}\n")
print(f"==>Death all RMSE\n Overall: {abs(error_weeks.mean()).mean():.
↪4f}\n{list(zip(df_test_week,error_weeks))[-1]}\n")
```

```
==>Death high corr RMSE
 Overall: 31.9465
('Deaths (daily growth) (CUSTOM)', 3.3799641450971647)

==>Death low corr RMSE
 Overall: 118.0806
('Deaths (daily growth) (CUSTOM)', 1.5768054964607212)

==>Death all RMSE
 Overall: 69.8256
('Deaths (daily growth) (CUSTOM)', 8.993726124217844)
```

[47]: