Machine Learning and Forecasting Write-Up

We will be applying two different approaches for the machine learning aspect of this project. For the machine learning part, we will be predicting what the growth rate of Sweden would have been if they applied a certain specific set of policies, while for the time series part, we will be forecasting the next two months of Sweden. We particularly chose Sweden for this project because we were given that Caladan is a country with a population of 3.2Million people and has 2 major cities. The closest country to this, which had all the Covid data was New Zealand and Sweden. We disregarded New Zealand because it is an island nation and is an outlier.

Machine Learning steps:

1. Cleaning and preprocessing the data
2. Finding the best combination of policies, excluding Sweden, with the lowest growth rates
3. Finding the best policies to model on using a correlation matrix
4. Using XGBoost Regressor for training and testing the model, excluding Sweden data
5. Applying the best combination of policies for each month and predicting Sweden data

Forecasting steps:

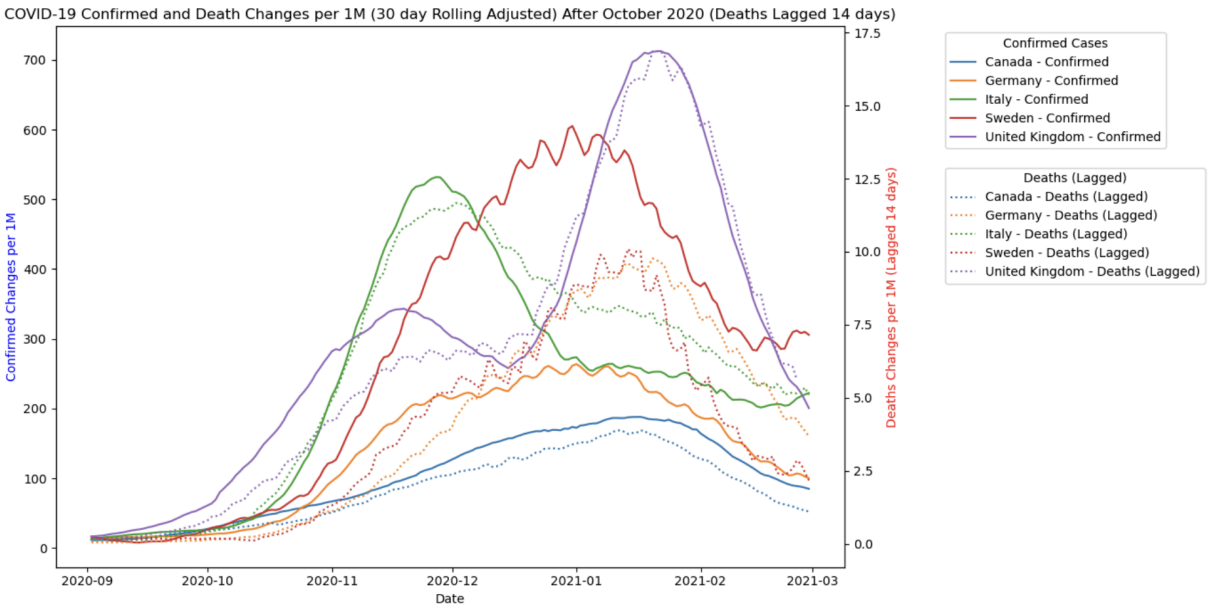
1. Performing statistical analysis test to ensure the applicability of the SARIMA model
2. Applying the SARIMA model
3. Forecasting Sweden data

ML modeling Sweden:

We first merged all the data that we had into one and saved it as final_df, which had information about the 10 countries we have data about. Then we made some new columns on top of all the data that we had, which took population into account.

```
final_df['Confirmed_per1M'] = (final_df['Confirmed']/final_df['Sum of 2020'])*1000000
final_df['Confirmed_Change_per1M'] = (final_df['Confirmed_Change']/final_df['Sum of 2020'])*1000000
final_df['Deaths_per1M'] = (final_df['Deaths']/final_df['Sum of 2020'])*1000000
final_df['Deaths_Change_per1M'] = (final_df['Deaths_Change']/final_df['Sum of 2020'])*1000000
final_df['Recovered_per1M'] = (final_df['Recovered']/final_df['Sum of 2020'])*1000000
final_df['Recovered_Change_per1M'] = (final_df['Recovered_Change']/final_df['Sum of 2020'])*1000000
```

We then calculated the growth rate and the rolling averages for deaths change per 1 million people and confirmed change per 1 million people. After creating all the required columns, we performed some EDA and found that a 14 day lag to the death aligned with the cases for a country and had a higher correlation with the policies as well.



We then chose the three policies we believe were the most important for COVID-19 spread, which were 'C1_School_closing', 'C5_Close_public_transport', 'C7_Restrictions_on_internal_movement'. These policies also performed very well on the logistic regression with 1% and 3% as the required threshold values. We found what the best levels of these policies were for each month by aggregating data and made sure that these sets of policies were in place for at least 40 days to be considered.

```
policy_columns = ['C1_School_closing', 'C5_Close_public_transport', 'C7_Restrictions_on_internal_movement', 'Yea

grouped_data = data.groupby(policy_columns).agg({
    'Deaths_Change_per1M': lambda x: x[x >= 0].mean(),
    'Confirmed_Change_per1M': lambda x: x[x >= 0].mean()
}).reset_index()

policy_days_data = data.groupby(policy_columns).size().reset_index(name='Days_Active')


full_grouped_data = pd.merge(grouped_data, policy_days_data, on=policy_columns, how='left')

active_policies = full_grouped_data[full_grouped_data['Days_Active'] >= 40]

optimal_active_policies = active_policies.sort_values(by=['YearMonth', 'Deaths_Change_per1M', 'Confirmed_Change_

optimal_active_policies.head(20)
```

YearMonth	C1_School_closing	C5_Close_public_transport	C7_Restrictions_on_internal_movement	Deaths_Change_per1M	Confirmed_Change_per1M	Days_Active
2020-02	0	0	0	0.000122	0.025068	186
2020-03	0	0	0	0.026226	2.297413	66
2020-04	3	0	2	2.460449	38.864766	59
2020-05	3	0	1	1.188643	5.876332	43
2020-06	3	0	2	0.827703	9.918807	43
2020-07	1	0	1	0.159498	10.150895	40
2020-08	2	0	2	0.033615	8.703368	44
2020-09	1	0	0	0.154231	22.131199	44
2020-10	1	0	1	0.370558	75.870154	56
2020-11	2	0	2	4.655588	270.054045	57
2020-12	1	1	1	2.438502	146.019347	51
2021-01	3	1	2	10.554950	350.181468	78
2021-02	3	1	2	5.476095	141.949812	95
2021-03	3	1	2	3.193362	178.309668	73

After these preliminary steps, we started building the ML Model.

We trained on the policies we selected and got rid of Sweden to avoid leakage and also got rid of Great Britain because it had an error in reporting where the deaths were reported negative which changed the rolling average for an entire month and had a significant effect on the model.

```
params = {
    'objective': 'reg:squarederror',
    'max_depth': 6,
    'learning_rate': 0.1,
    'colsample_bytree': 0.8,
    'subsample': 0.8,
    'n_estimators': 1000
}

X_confirmed_filtered = data_filtered[features_confirmed]
X_deaths_filtered = data_filtered[features_deaths]
y_confirmed_filtered = data_filtered['Confirmed_Change_per1M']
y_deaths_filtered = data_filtered['Deaths_Change_per1M']

X_confirmed_filtered['Month_Year'] = encoder.fit_transform(X_confirmed_filtered['Month_Year'])
X_deaths_filtered['Month_Year'] = encoder.fit_transform(X_deaths_filtered['Month_Year'])

X_train_confirmed_f, X_test_confirmed_f, y_train_confirmed_f, y_test_confirmed_f = train_test_split(X_confirmed_
X_train_deaths_f, X_test_deaths_f, y_train_deaths_f, y_test_deaths_f = train_test_split(X_deaths_filtered, y_dea


model_confirmed_f = xgb.XGBRegressor(**params)
model_confirmed_f.fit(X_train_confirmed_f, y_train_confirmed_f)
y_pred_confirmed_f = model_confirmed_f.predict(X_test_confirmed_f)

model_deaths_f = xgb.XGBRegressor(**params)
model_deaths_f.fit(X_train_deaths_f, y_train_deaths_f)
y_pred_deaths_f = model_deaths_f.predict(X_test_deaths_f)

rmse_confirmed_f = mean_squared_error(y_test_confirmed_f, y_pred_confirmed_f, squared=False)
mae_confirmed_f = mean_absolute_error(y_test_confirmed_f, y_pred_confirmed_f)

rmse_deaths_f = mean_squared_error(y_test_deaths_f, y_pred_deaths_f, squared=False)
mae_deaths_f = mean_absolute_error(y_test_deaths_f, y_pred_deaths_f)

(rmse_confirmed_f, mae_confirmed_f, rmse_deaths_f, mae_deaths_f)
```

The RMSE values that we got were 45.75 for confirmed and 1.54 for deaths, and these values were within our acceptable threshold of >10% of the range. The range of the confirmed change per

million was around 800 and the range of the deaths was around slightly above 16. Therefore, we decided to go ahead with this model.

We used this model on the best policy level for each month that we had calculated earlier for each month, and applied this model for Sweden.

```
df = pd.read_csv('optimal.csv')
df = df.drop(columns = 'Unnamed: 0')
```

```
swe_data = data[data["CountryName"] == 'Sweden']
swe_data = swe_data.reset_index().drop(columns = 'index')
swe_data
```

...

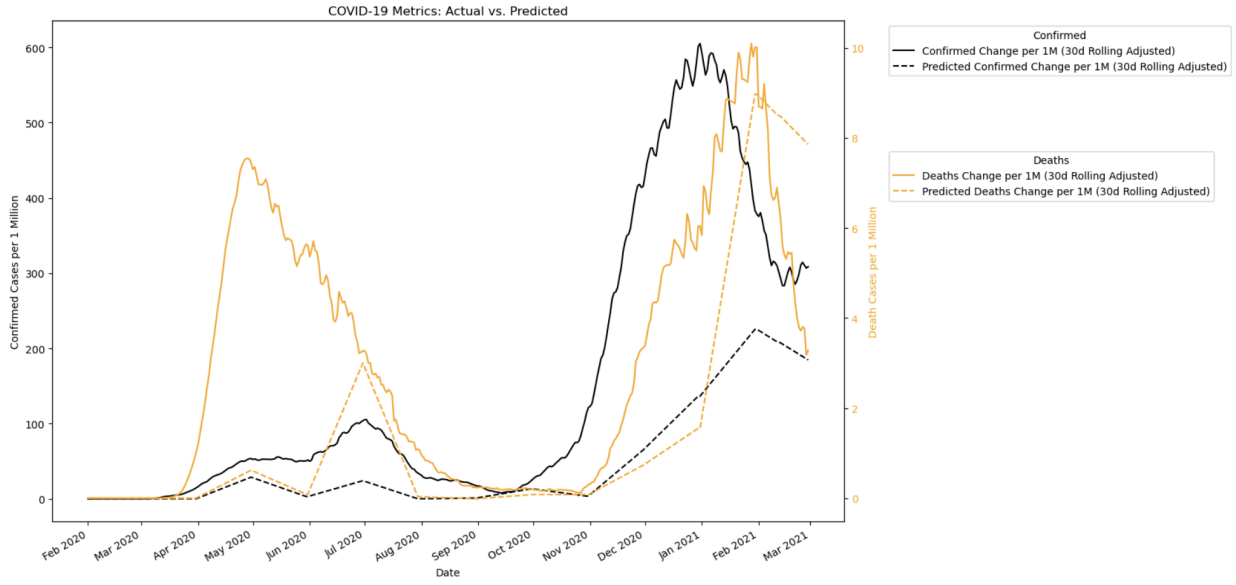
```
df['YearMonth'] = pd.to_datetime(df['YearMonth'], format='%Y-%m')

daily_dates = pd.date_range(start='2020-02-01', end='2021-03-25')

expanded_df = df.set_index('YearMonth').reindex(daily_dates, method='ffill').reset_index()
expanded_df.rename(columns={'index': 'Date'}, inplace=True)

expanded_df.head()
```

...

```
expanded_df = expanded_df[expanded_df['Date'].isin(swe_data['Date'])]
expanded_df['Deaths_Change_per1M'] = swe_data['Deaths_Change_per1M']
expanded_df['Confirmed_Change_per1M'] = swe_data['Confirmed_Change_per1M']
```

```
data_sweden = expanded_df

data_sweden['Month_Year'] = pd.to_datetime(data_sweden['Date']).dt.to_period('M')

encoder = LabelEncoder()
data_sweden['Month_Year'] = encoder.fit_transform(data_sweden['Month_Year'])

X_confirmed_sweden = data_sweden[features_confirmed]
X_deaths_sweden = data_sweden[features_deaths]

y_pred_confirmed_sweden = model_confirmed_f.predict(X_confirmed_sweden)
y_pred_deaths_sweden = model_deaths_f.predict(X_deaths_sweden)

results_sweden = data_sweden[['Date', 'Confirmed_Change_per1M', 'Deaths_Change_per1M']]
results_sweden['Predicted_Confirmed_Change_per1M'] = y_pred_confirmed_sweden
results_sweden['Predicted_Deaths_Change_per1M'] = y_pred_deaths_sweden
```

We added rolling averages for the predictions to our dataset and plotted the results. The predictions showed a significant decrease in the cases and the deaths, with a different set of policies for each month.



Time-Series Sweden:

First, we loaded the data for just sweden.

```
swe_data = df[df['ISO3'] == 'SWE']
swe_data
```

Then we performed, Kwiatkowski-Phillips-Schmidt-Shin test, to see if the time series is stationary or not.

```
def kpss_test(series, **kw):
    statistic, p_value, lags, crit = kpss(series, **kw)
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print('Critical Values:', crit)

kpss_test(swe_data['Confirmed_Change'])
```

```
KPSS Statistic: 2.3666035256147016
p-value: 0.01
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
```

These were the results from this test, which suggests that the time series is not stationary and instead contains a unit root.

Next, we performed an augmented Dickey-Fuller test (ADF Test), which serves a similar purpose but has the opposite hypothesis from the KPSS test.

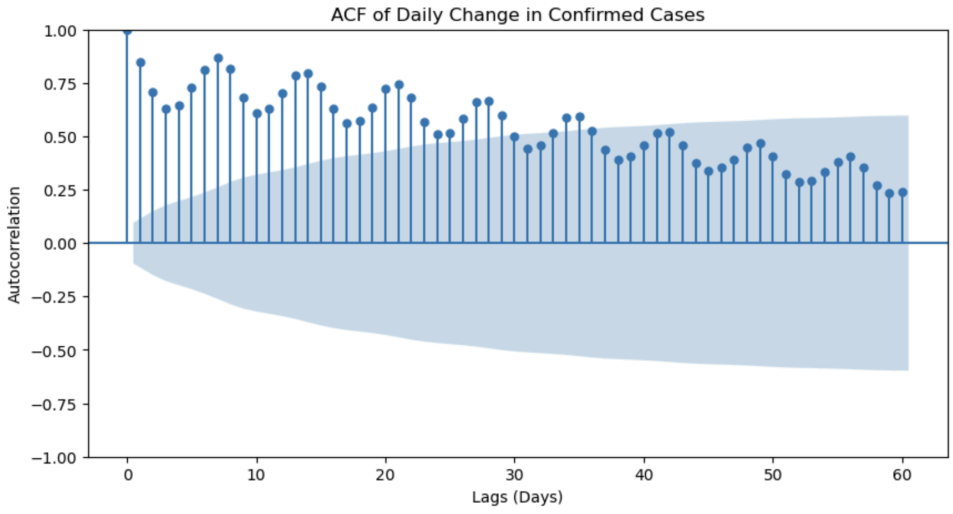
```python
from statsmodels.tsa.stattools import adfuller

def adf_test(series):
    result = adfuller(series.dropna())
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')
    print('Critical Values:', result[4])

adf_test(swe_data['Confirmed_Change'])
```

This test gave the same results that we got from the KPSS test.

```
ADF Statistic: -1.8960403091897706
p-value: 0.3339261610543488
Critical Values: {'1%': -3.4467631030732506, '5%': -2.868774682311516, '10%': -2.570623877774392}
```

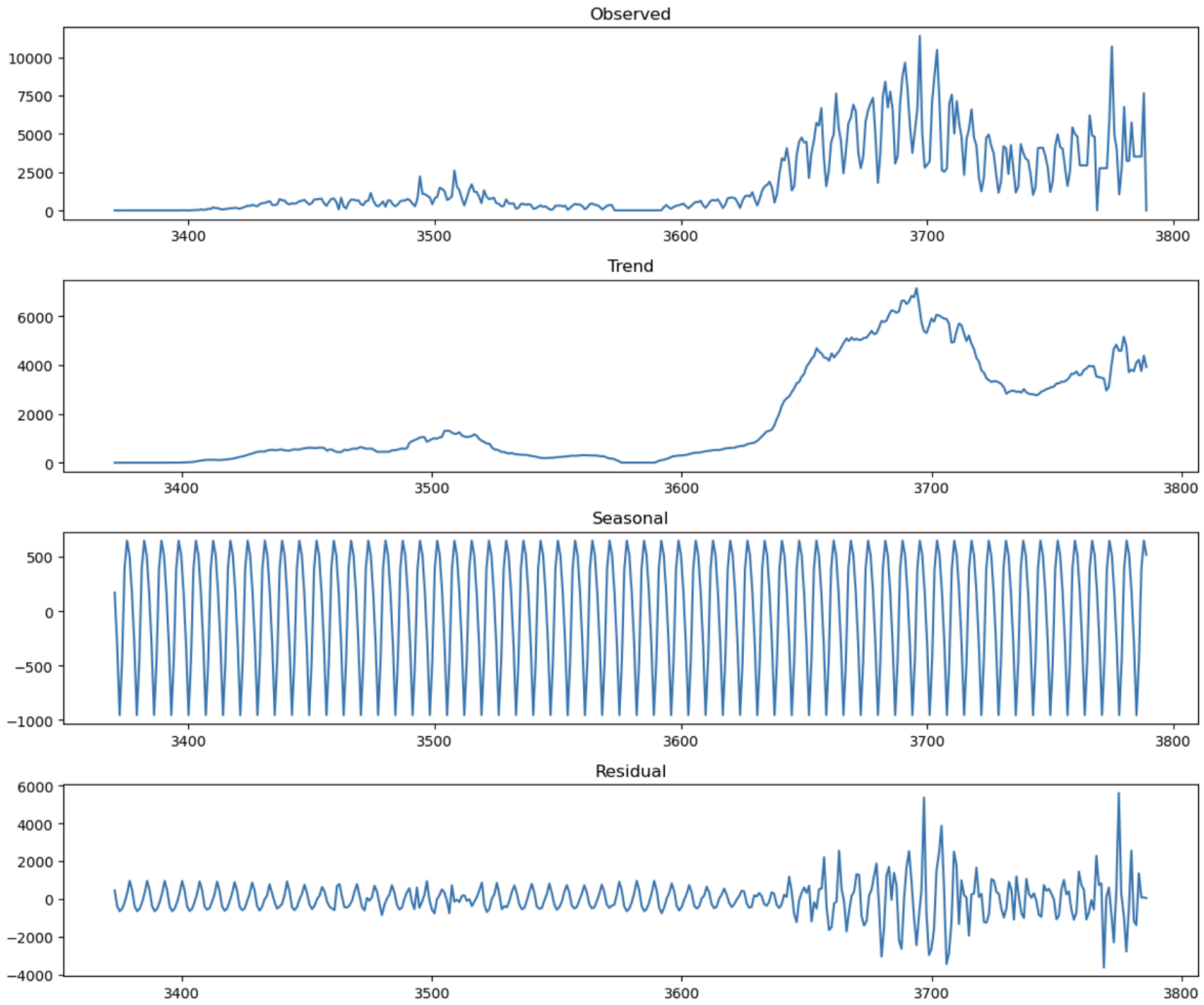
Next, we plotted the ACF of daily cases and saw that peaks usually occurred in intervals of 7 and therefore, decided to use period = 7.

```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
fig, ax = plt.subplots(figsize=(10, 5))
plot_acf(swe_data['Confirmed_Change'].dropna(), ax=ax, lags=60, title='ACF of Daily Change in Confirmed Cases')
plt.xlabel('Lags (Days)')
plt.ylabel('Autocorrelation')
plt.show()
```



To address the non-stationarity, we decided to decompose this seasonally and apply a Seasonal ARIMA model for forecasting.

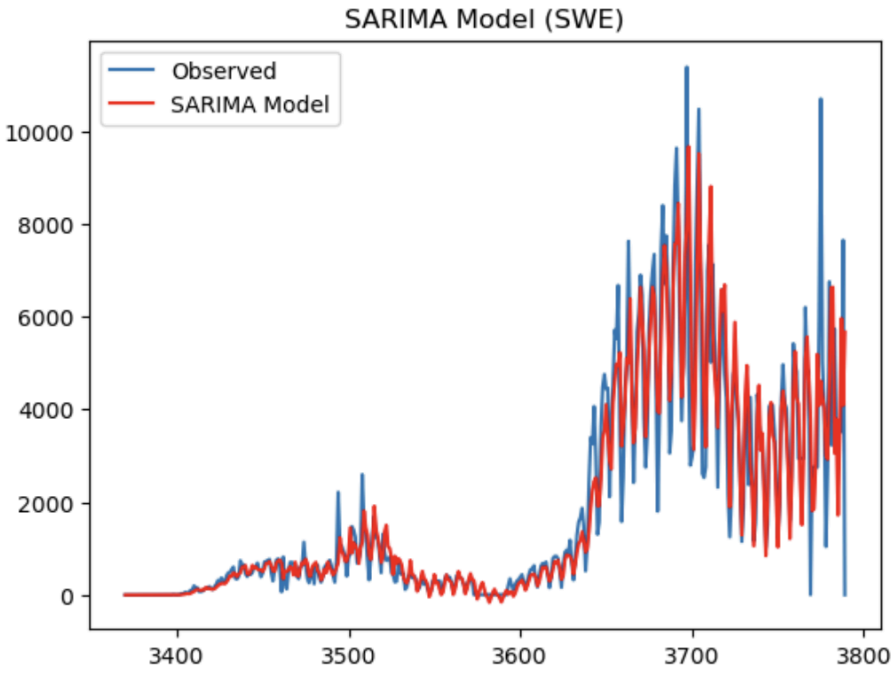
```python
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(swe_data['Confirmed_Change'], model='additive', period=7)

fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10))
result.observed.plot(ax=ax1, title='Observed')
result.trend.plot(ax=ax2, title='Trend')
result.seasonal.plot(ax=ax3, title='Seasonal')
result.resid.plot(ax=ax4, title='Residual')
plt.tight_layout()
plt.show()
```



We fit a SARIMA model with a period of 7.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(swe_data['Confirmed_Change'],
                order=(1, 1, 1),
                seasonal_order=(1, 1, 1, 7),
                enforce_stationarity=False,
                enforce_invertibility=False)

sarima_model = model.fit(disp=False)

sarima_model.summary()
```

This is how our SARIMA Model performed and calculated the fitted values:



Lastly, we plotted the 30 day rolling average of the forecast for the next 60 days in Sweden.

```
last_n = 60
cutoff_index = len(df) - last_n

plt.figure(figsize=(10, 6))
plt.plot(df['Date'][:cutoff_index], df['Rolling_Adjusted'][:cutoff_index], label='Actual')

plt.plot(df['Date'][cutoff_index:], df['Rolling_Adjusted'][cutoff_index:], color='orange', label='Predicted')

plt.xlabel('Date')
plt.ylabel('30 Day Rolling Average')
plt.title('Time Series Plot')
plt.legend()
plt.grid(True)

plt.show()
```