Heart Rate Prediction Using Time series analysis

The goal of this study is to use the time series method to analyze heart rate predictions. A time series is a collection of observations spanning a period of time. How we come across heart rate variations per minute is a basic example of time series. We'll look at what a time series is, what methods are used to forecast them, and what makes time series data so unique, a complicated issue in data science.

To achieve a better prediction and forecast for this project, I implemented the SARIMA Model. First, we must clean the data after importing all of the essential libraries (Figure 1)

```
[1]  !pip install pmdarima
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import matplotlib.ticker as ticker
     import statsmodels.api as sm
     import itertools
     from sklearn.metrics import mean_squared_error
     from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
     from scipy.spatial.distance import cdist
```

Figure 1. Importing the libraries

Reading the CSV File using Pandas

```
data_f=pd.read_csv('PT_Train.csv')
print('shape of data',data_f.shape)
data_f.head()
```

shape of data (226, 5)

	Timestamp (GMT)	Lifetouch Heart Rate	Lifetouch Respiration Rate	Oximeter SpO2	Oximeter Pulse
0	17/08/2015 15:09	139	41	NaN	NaN
1	17/08/2015 15:10	144	40	92.0	140.0
2	17/08/2015 15:11	140	42	89.0	144.0
3	17/08/2015 15:12	138	45	93.0	141.0
4	17/08/2015 15:13	133	42	94.0	134.0

```
[3]  ev_df = data_f.loc[data_f['Lifetouch Heart Rate']>= 1000]
     print(ev_df)
     print(data_f)
```

```
     Timestamp (GMT)  Lifetouch Heart Rate  ...  Oximeter SpO2  Oximeter Pulse
30  17/08/2015 15:39                 61441  ...           94.0           152.0
31  17/08/2015 15:40                 61442  ...           92.0           146.0
32  17/08/2015 15:41                 61441  ...           -1.0            -1.0
33  17/08/2015 15:42                 61441  ...           98.0           143.0
34  17/08/2015 15:43                 61441  ...           97.0           142.0
35  17/08/2015 15:44                 61441  ...           96.0           140.0
36  17/08/2015 15:45                 61442  ...           96.0           144.0
41  17/08/2015 15:50                 61442  ...           94.0           148.0

[8 rows x 5 columns]
      Timestamp (GMT)  Lifetouch Heart Rate  ...  Oximeter SpO2  Oximeter Pulse
0    17/08/2015 15:09                   139  ...            NaN             NaN
1    17/08/2015 15:10                   144  ...           92.0           140.0
2    17/08/2015 15:11                   140  ...           89.0           144.0
3    17/08/2015 15:12                   138  ...           93.0           141.0
4    17/08/2015 15:13                   133  ...           94.0           134.0
..                ...                   ...  ...            ...             ...
221  17/08/2015 18:50                   159  ...            NaN             NaN
222  17/08/2015 18:51                   151  ...            NaN             NaN
223  17/08/2015 18:52                   140  ...            NaN             NaN
224  17/08/2015 18:53                   140  ...            NaN             NaN
225  17/08/2015 18:54                   138  ...            NaN             NaN

[226 rows x 5 columns]
```

Figure 2 Reading the CSV file

Removing the Outliers from the dataset

```
[4] og_data=data_f.loc[data_f['Lifetouch Heart Rate']<= 500 ]
    print(og_data)

         Timestamp (GMT)  Lifetouch Heart Rate  ...  Oximeter SpO2  Oximeter Pulse
0    17/08/2015 15:09                   139  ...            NaN             NaN
1    17/08/2015 15:10                   144  ...           92.0           140.0
2    17/08/2015 15:11                   140  ...           89.0           144.0
3    17/08/2015 15:12                   138  ...           93.0           141.0
4    17/08/2015 15:13                   133  ...           94.0           134.0
..                ...                   ...  ...            ...             ...
221  17/08/2015 18:50                   159  ...            NaN             NaN
222  17/08/2015 18:51                   151  ...            NaN             NaN
223  17/08/2015 18:52                   140  ...            NaN             NaN
224  17/08/2015 18:53                   140  ...            NaN             NaN
225  17/08/2015 18:54                   138  ...            NaN             NaN

[218 rows x 5 columns]
```

Figure 3 Dropping the Unnecessary Data

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(og_data.index, og_data["Lifetouch Heart Rate"], label='original')

[<matplotlib.lines.Line2D at 0x7fa322ac9110>]
```

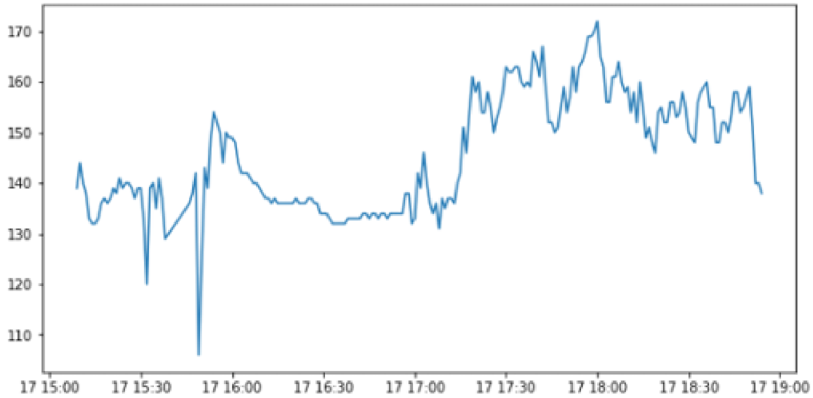


Figure 4 Plotting the Lifetouch Heart Rate

Applying the dickey fuller test to check whether the data is stationary or not

Dickey Fuller Test

```
[13] from statsmodels.tsa.stattools import adfuller
     def adfuller_test(database):
       dftest = adfuller(database, autolag = 'AIC')
       print("1. ADF : ",dftest[0])
       print("2. P-Value : ", dftest[1])
       print("3. Num Of Lags : ", dftest[2])
       print("4. Num Of Observations Used For ADF Regression and Critical Values Calculation :", dftest[3])
       print("5. Critical Values :",dftest[4])
       for key, val in dftest[4].items():
         print("\t",key, ": ", val)
```

```
[14] adfuller_test(og_data['Lifetouch Heart Rate'])

     1. ADF :  -2.3604265607058403
     2. P-Value :  0.1532185464358677
     3. Num Of Lags :  1
     4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 216
     5. Critical Values : {'1%': -3.4609922013325267, '5%': -2.875015740963014, '10%': -2.5739524288408777}
             1% :  -3.4609922013325267
             5% :  -2.875015740963014
             10% :  -2.5739524288408777
```

Figure 5 Checking the Stationarity using Dickey-Fuller Test

As we can see the P-value is greater than 0.05 so it is not stationary

Applying the Differencing to the Data

Differencing method

```
og_data['Minutes Difference'] = og_data['Lifetouch Heart Rate']-og_data['Lifetouch Heart Rate'].shift(4)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
  """Entry point for launching an IPython kernel.
```

```
[16] og_data['Minutes Difference'].dropna().plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7fa32892e310>
```

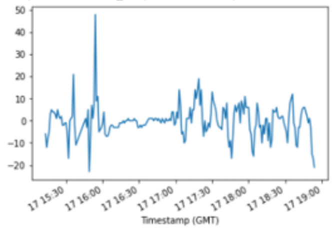


Figure 6 Differencing method to make the data stationary

```
[18] adfuller_test(og_data['Minutes Difference'].dropna())

1. ADF :  -3.9353979424262286
2. P-Value :  0.0017892434622948385
3. Num Of Lags :  15
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 198
5. Critical Values : {'1%': -3.4638151713286316, '5%': -2.876250632135043, '10%': -2.574611347821651}
        1% :  -3.4638151713286316
        5% :  -2.876250632135043
        10% :  -2.574611347821651

[19] og_data['Lifetouch Heart Rate_diff_4'] = og_data['Lifetouch Heart Rate'] - og_data['Lifetouch Heart Rate'].shift(4)
     og_data['Lifetouch Heart Rate_diff_4'].dropna().plot()

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  """Entry point for launching an IPython kernel.
<matplotlib.axes._subplots.AxesSubplot at 0x7fa3288c9710>
```

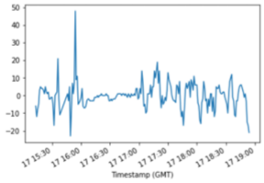


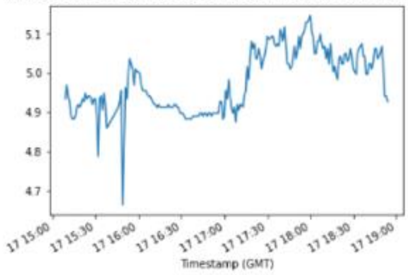
Figure 7 Checking again the data is stationary or not after differencing method

Applying the Transformation

Tranforming the data

```
[20] # Create transformation columns
     import numpy as np

     # Calculate the log
     og_data['adj_log'] = np.log(og_data['Lifetouch Heart Rate'])

     # Calculate the square root
     og_data['adj_sqrt'] = np.sqrt(og_data['Lifetouch Heart Rate'])

     # Calculate the cubed root
     og_data['adj_cbrt'] = np.cbrt(og_data['Lifetouch Heart Rate'])
```

```
og_data['adj_log'].dropna().plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7fa32882ed50>
```




```
[22] og_data['adj_sqrt'].dropna().plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7fa32ab70fd0>
```

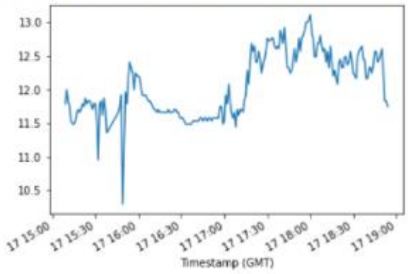
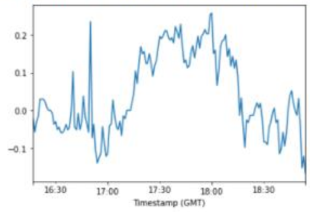


Figure 8 Transforming the data

```
[23] og_data['Lifetouch Heart Rate_log_diff'] = og_data['adj_log'] - og_data['adj_log'].shift(60)
     og_data['Lifetouch Heart Rate_log_diff'].dropna().plot()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inc
  """Entry point for launching an IPython kernel.
<matplotlib.axes._subplots.AxesSubplot at 0x7fa32872e350>
```




```
[24] og_data['Lifetouch Heart Rate_sqrt_diff'] = og_data['adj_sqrt'] - og_data['adj_sqrt'].shift(1)
     og_data['Lifetouch Heart Rate_sqrt_diff'].dropna().plot()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inc
  """Entry point for launching an IPython kernel.
<matplotlib.axes._subplots.AxesSubplot at 0x7fa3288add50>
```

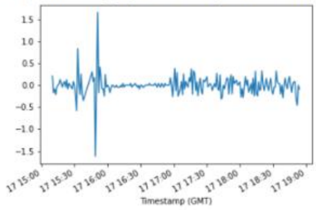


Figure 8.1

After differencing and transforming, Now Lets's run once again the dickey fuller test.

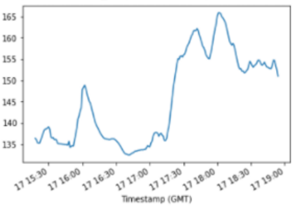
Rolling Statistics

```
[25] og_data['#MA10'] = og_data['Lifetouch Heart Rate'].rolling(window = 10 ).mean()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  """Entry point for launching an IPython kernel.
```

```
[26] og_data['#MA10'].dropna().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa32868d690>
```




```
[27] adfuller_test(og_data['Minutes Difference'].dropna())
```

```
1. ADF :  -3.9353979424262286
2. P-Value :  0.0017892434622948385
3. Num Of Lags :  15
4. Num Of Observations Used For ADF Regression and Critical Values Calculation : 198
5. Critical Values : {'1%': -3.4638151713286316, '5%': -2.876250632135043, '10%': -2.574611347821651}
        1% :  -3.4638151713286316
        5% :  -2.876250632135043
        10% :  -2.574611347821651
```

Figure 9 Rolling statistics

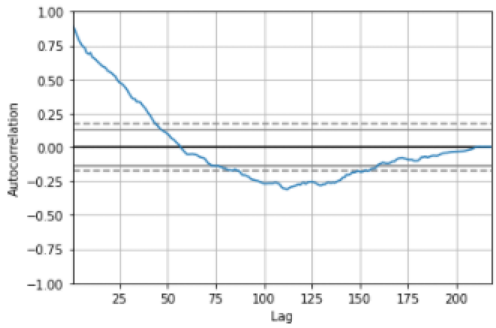
As we can see the data is stationary after running the dickey fuller test.

Building the Auto-Regressive Model

Auto Regressive Model

```
[28] import pydot
     from pandas.plotting import autocorrelation_plot
     autocorrelation_plot(og_data['Lifetouch Heart Rate'])
     plt.show
```

```
<function matplotlib.pyplot.show>
```




```
[29] from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

Figure 10 Applying the Auto Regressive Model

```
[30] fig = plt.figure(figsize=(12,8))
     ax1 = fig.add_subplot(211)
     fig = sm.graphics.tsa.plot_acf(og_data['Minutes Difference'].iloc[4:],lags=40,ax=ax1)
     ax2 = fig.add_subplot(212)
     fig = sm.graphics.tsa.plot_pacf(og_data['Minutes Difference'].iloc[4:],lags=40,ax=ax2)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/graphics/tsaplots.py:353: FutureWarning: The default r
  FutureWarning,
```

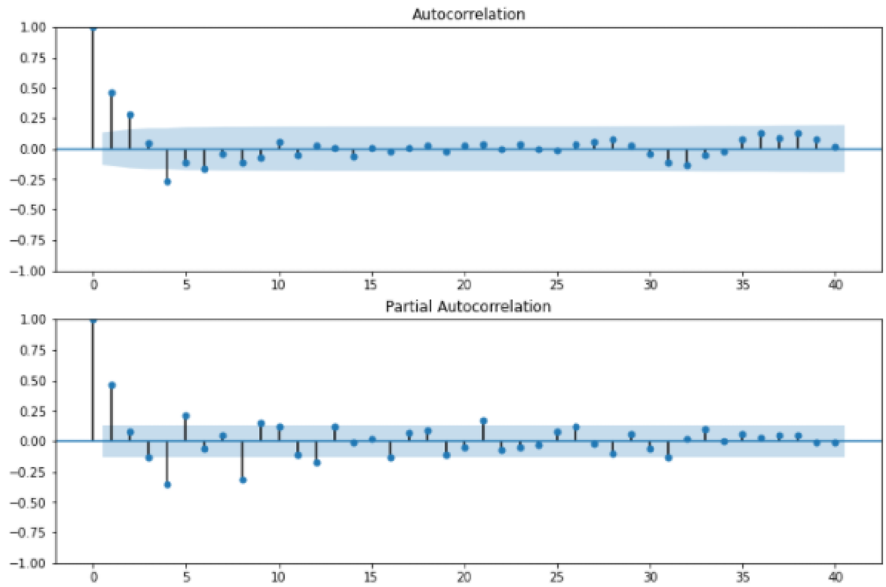


Figure 11 To find the P Q Value

Considering the above graph Autocorrelation is 0(Q) and Partial autocorrelation is 1(P)

Building the SARIMA model

SARIMA Model

```
[31] from pmdarima import auto_arima
     # ignore harmless warnings
     import warnings
     warnings.filterwarnings("ignore")
```

```
[32] stepwise_fit = auto_arima(og_data["Lifetouch Heart Rate"], trace=True,
                               supress_warnings=True)
     stepwise_fit.summary()
```

```
[33] from statsmodels.tsa.arima_model import ARIMA
```

Split Data into Training and Testing

```
[34] print(og_data.shape)
     train=og_data.iloc[:-30]
     test=og_data.iloc[-30:]
     print(train.shape,test.shape)

     (218, 12)
     (188, 12) (30, 12)
```

```
model = sm.tsa.statespace.SARIMAX(og_data['Lifetouch Heart Rate'],
                                  order=(0, 1, 2),
                                  seasonal_order=(0, 0, 0, 10),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)

#train model
model_fit = model.fit()
print(model_fit.summary())
```

Figure 12 Building SARIMA Model

We got the better Prediction using the SARIMAX Model as we can see in Figure 13

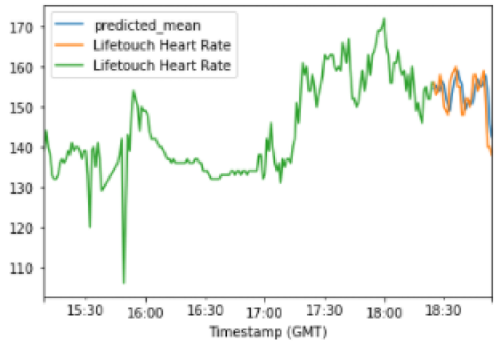
```
start=len(train)
end=len(train)+len(test)-1
pred=model_fit.predict(start=start,end=end,typ='levels')
pred.index=og_data.index[start:end+1]
print(pred)
```

```
Timestamp (GMT)
2015-08-17 18:25:00    154.459915
2015-08-17 18:26:00    155.330284
2015-08-17 18:27:00    153.943099
2015-08-17 18:28:00    153.999252
2015-08-17 18:29:00    156.354592
2015-08-17 18:30:00    155.518042
2015-08-17 18:31:00    152.281835
2015-08-17 18:32:00    150.402909
2015-08-17 18:33:00    149.019818
2015-08-17 18:34:00    153.153542
2015-08-17 18:35:00    155.939607
2015-08-17 18:36:00    157.694661
2015-08-17 18:37:00    159.022448
2015-08-17 18:38:00    156.631373
2015-08-17 18:39:00    155.709809
2015-08-17 18:40:00    151.185641
2015-08-17 18:41:00    149.384660
2015-08-17 18:42:00    150.955696
2015-08-17 18:43:00    151.545232
2015-08-17 18:44:00    150.625153
2015-08-17 18:45:00    152.038630
2015-08-17 18:46:00    155.525982
2015-08-17 18:47:00    156.924906
2015-08-17 18:48:00    155.178493
2015-08-17 18:49:00    155.101810
2015-08-17 18:50:00    156.221319
2015-08-17 18:51:00    157.839135
2015-08-17 18:52:00    153.784816
2015-08-17 18:53:00    145.733910
2015-08-17 18:54:00    142.491386
Name: predicted_mean, dtype: float64
```

Figure 13 Predictions

```
[37] pred.plot(legend=True)
     test['Lifetouch Heart Rate'].plot(legend=True)
     train['Lifetouch Heart Rate'].plot(legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa325064910>
```




```
[38] test['Lifetouch Heart Rate'].mean()
```

```
152.63333333333333
```

```
[39] from sklearn.metrics import mean_squared_error
     from math import sqrt
     rmse=sqrt(mean_squared_error(pred,test['Lifetouch Heart Rate']))
     print(rmse)
```

Figure 14 Finding the Mean Squared Error

Forecasting

```
[38] import statsmodels.api as sm
     model=sm.tsa.statespace.SARIMAX(og_data['Lifetouch Heart Rate'],order=(0,1,2),seasonal_order=(1,1,1,12))
     results=model.fit()
```

```
og_data['forecast']=results.predict(start=90,end=216,dynamic=True)
og_data[['Lifetouch Heart Rate','forecast']].plot(figsize=(12,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4a3336b710>
```

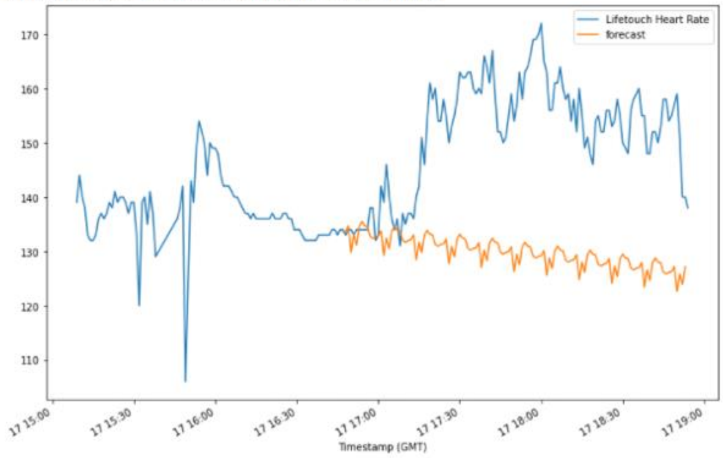


Figure 15 Forecasting using SARIMA

```
[40] from pandas.tseries.offsets import DateOffset
     future_dates=[og_data.index[-1]+ DateOffset(minutes=x)for x in range(0,20)]
```

```
[41] future_dataset_og_data=pd.DataFrame(index=future_dates[1:],columns=data_f.columns)
```

```
[42] future_dataset_og_data.tail(5)
```

	Timestamp (GMT)	Lifetouch Heart Rate	Lifetouch Respiration Rate	Oximeter SpO2	Oximeter Pulse
2015-08-17 19:09:00	NaN	NaN	NaN	NaN	NaN
2015-08-17 19:10:00	NaN	NaN	NaN	NaN	NaN
2015-08-17 19:11:00	NaN	NaN	NaN	NaN	NaN
2015-08-17 19:12:00	NaN	NaN	NaN	NaN	NaN
2015-08-17 19:13:00	NaN	NaN	NaN	NaN	NaN

```
[43] future_data=pd.concat([og_data,future_dataset_og_data])
```

```
[44] # Out-of-sample forecasts
     forecasts = model_fit.forecast(steps=20)
     print(forecasts)
```

```
218    139.902298
219    139.945947
220    139.945947
221    139.945947
222    139.945947
223    139.945947
224    139.945947
225    139.945947
226    139.945947
227    139.945947
228    139.945947
229    139.945947
230    139.945947
231    139.945947
232    139.945947
233    139.945947
234    139.945947
235    139.945947
236    139.945947
237    139.945947
Name: predicted_mean, dtype: float64
```

Figure 16 Forecasting values

Conclusion

In this project, I have learned and implemented the Heart rate Prediction using Time series analysis. A major topic of this project is how SARIMAX can be useful when exogenous seasonality factors appear in the time series and you can get the most reliable forecasts and predictions.

Google Colab Link: https://colab.research.google.com/drive/1NHOfP54roJEa1L8MlcpBbMMJtrJ6luMe?usp=sharing