# Heartrate predication

The goal of this project is to analyse heart rate time series data, develop a time series model, which can forecast future values. Time series analysis recognises that data points collected over time may have an underlying structure (such as autocorrelation, trend, or seasonal fluctuation) that must be taken into consideration.

We have a file**("PT_train")** with 5 columns, however we'll be focusing on Lifetouch Heart Rate. After analysing the data, we discovered that there were some outliers, so we eliminated them and saved the data in a new data frame ("**p_df").**

```
In [52]: fig, ax = plt.subplots()
         ax.plot(df["Lifetouch Heart Rate"], label='original')

Out[52]: [<matplotlib.lines.Line2D at 0x19ba972fe50>]
```

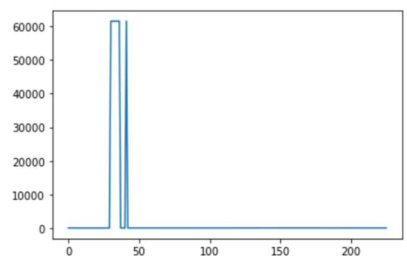



*Figure 1 outliners*

```
In [14]: fig, ax = plt.subplots(figsize=(30, 10))
         ax.plot(p_df.index,p_df["Lifetouch Heart Rate"], label='original')

Out[14]: [<matplotlib.lines.Line2D at 0x19ba812f3d0>]
```

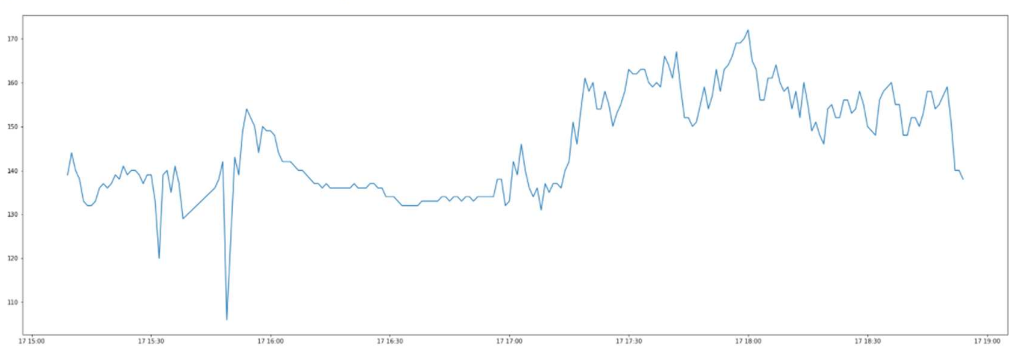


*Figure 2 clean data*

To make time series prediction easier, we used the panadas library **pd.to datetime** to convert the **Timestamp (GMT**) column to date and time and then converted that column to index **pd.set _index** for easy data prediction. We discovered that some of the data was not stationary after plotting the graph, therefore we used the **adfuller test** to establish that the data was stationary and used the differencing approach.

```
In [15]: 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)
```

```
In [16]: X = p_df["Lifetouch Heart Rate"].values
result = adfuller(X)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' %result[1])
print('Critical Values:')
for key, value in result[4].items():
  print('\t%s: %3f' % (key, value))

if result[0] < result[4]["5%"]:
  print ("Reject ")
```

```
ADF Statistic: -2.360427
p-value: 0.153219
Critical Values:
        1%: -3.460992
        5%: -2.875016
        10%: -2.573952
```

*Figure 3 adfuller test*

We imported auto_arima and constructed the auto arima model after making the data stationary, and we used model. summary () to find the value of **p d q**.

```
In [30]: from pmdarima import auto_arima
# ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")
```

```
In [31]: stepwise_fit = auto_arima(p_df["Lifetouch Heart Rate"], trace=True,
                          supress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1306.664, Time=0.24 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1335.460, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1307.913, Time=0.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1304.325, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1333.460, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1306.307, Time=0.08 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1306.315, Time=0.13 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1308.286, Time=0.19 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=1302.325, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=1304.307, Time=0.05 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=1304.315, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=1305.915, Time=0.02 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=1306.286, Time=0.17 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0]
Total fit time: 1.083 seconds
```

*Figure 4 auto_arima*

We discovered that the auto arima model was not the greatest predictor, therefore we tried the **SARIMAX** model has we saw some seasonality in the data. The **SARIMAX** model was created by dividing the data into two parts: testing and training.

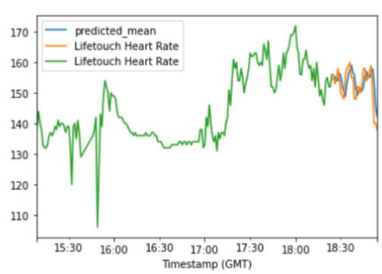
```
In [35]: model = sm.tsa.statespace.SARIMAX(p_df['Lifetouch Heart Rate'],
                                          order=(0, 1, 1),
                                          seasonal_order=(0, 0, 0, 2),
                                          enforce_stationarity=False,
                                          enforce_invertibility=False)
         #train model
         model_fit = model.fit()
         print(model_fit.summary())

                               SARIMAX Results
==========================================================================================
Dep. Variable:     Lifetouch Heart Rate   No. Observations:                  218
Model:                 SARIMAX(0, 1, 1)   Log Likelihood                -643.651
Date:                  Sat, 19 Feb 2022   AIC                           1291.302
Time:                          04:35:23   BIC                           1298.043
Sample:                               0   HQIC                          1294.026
                                  - 218
Covariance Type:                    opg
========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------
ma.L1         -0.4055      0.032    -12.819      0.000      -0.468      -0.344
sigma2        23.3230      0.965     24.171      0.000      21.432      25.214
==========================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1522.83
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.49   Skew:                            -1.09
Prob(H) (two-sided):                  0.00   Kurtosis:                        15.86
==========================================================================================

Warnings:
```

We found the p q d value as (0,1,1). We projected the last 10 values of the train data using model fit.predict, and then plotted the mean data prediction, test data prediction, and train data prediction.

Make prediction on Test set

```
In [36]: start=len(train)
         end=len(train)+len(test)-1
         pred=model_fit.predict(start=start,end=end,typ='levels')

         pred.index=p_df.index[start:end+1]

         print(pred)

Timestamp (GMT)
2015-08-17 18:25:00    154.478149
2015-08-17 18:26:00    155.382862
2015-08-17 18:27:00    153.966293
2015-08-17 18:28:00    153.986331
2015-08-17 18:29:00    156.372386
2015-08-17 18:30:00    155.556527
2015-08-17 18:31:00    152.253271
2015-08-17 18:32:00    150.319259
2015-08-17 18:33:00    148.940501
2015-08-17 18:34:00    153.137247
2015-08-17 18:35:00    156.028067
2015-08-17 18:36:00    157.794828
2015-08-17 18:37:00    159.105763
2015-08-17 18:38:00    156.664960
2015-08-17 18:39:00    155.675171
2015-08-17 18:40:00    151.112419
2015-08-17 18:41:00    149.262142
2015-08-17 18:42:00    150.889749
2015-08-17 18:43:00    151.549774
```

```
In [40]: pred.plot(legend=True)
         test['Lifetouch Heart Rate'].plot(legend=True)
         train['Lifetouch Heart Rate'].plot(legend=True)

Out[40]: <AxesSubplot:xlabel='Timestamp (GMT)'>
```




We imported Date Offset from pandas.tseries.offsets to predict future values. We created a second model for future prediction, with the start value equal to the data's end value and the end value equal to 20 minutes.

In [62]:
```
index_future_dates=pd.date_range(start='17/08/2015 18:54:00',end='2015-08-17 19:04:00')
pred=model2.predict(start=len(p_df),end=len(p_df),typ='levels').rename('ARIMA Productions')
pred.index=index_future_dates
print(pred)
#pred= model2.predict(start=pd.to_datetime('2015-08-17 18:54'), end=pd.to_datetime('2015-08-17 19:04'), dynamic=False)
```

```
2015-08-17 18:54:00    139.730489
Freq: D, Name: ARIMA Productions, dtype: float64
```

In [63]:
```
forecasts = model_fit.forecast(steps=20)
print(forecasts)
```

```
218    139.730489
219    139.730489
220    139.730489
221    139.730489
222    139.730489
223    139.730489
224    139.730489
225    139.730489
226    139.730489
227    139.730489
228    139.730489
229    139.730489
230    139.730489
231    139.730489
232    139.730489
233    139.730489
234    139.730489
235    139.730489
236    139.730489
237    139.730489
Name: predicted_mean, dtype: float64
```

**Conclusion:** In this project, I learned and implemented heart rate prediction using time series analysis, and I discovered that the best model to predict and forecast values is SARIMAX.