Heart Rate Time Series Analysis and Prediction

- **Name:** Nkechi Rosemary Ogbonna
- **Student ID:** 24145555
- **E-mail:** nkechi.ogbonna@mail.bcu.ac.uk

- **Link:** https://colab.research.google.com/drive/1h2TFcF3-y0HlyEn5yU5WYii8_ZFuu4HM?usp=sharing

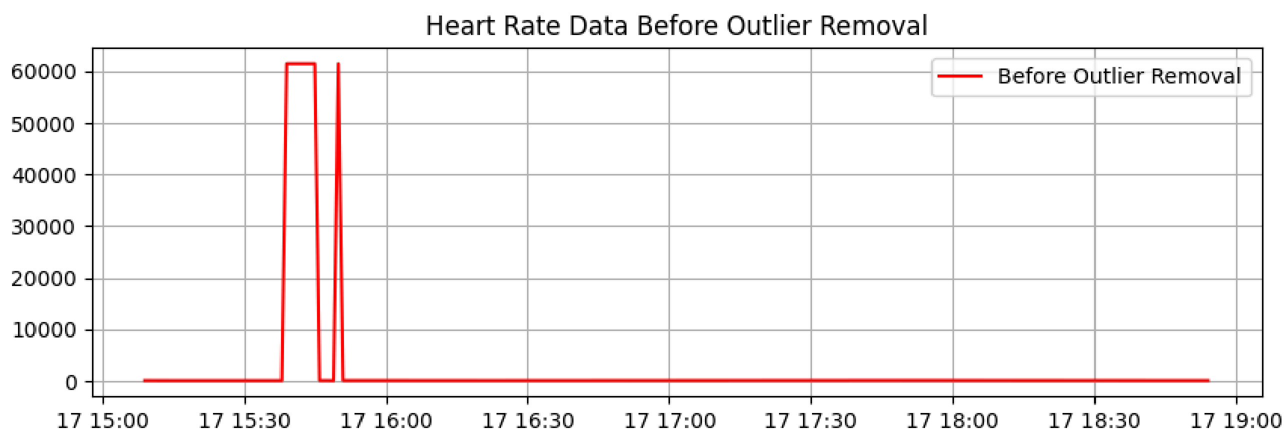
Introduction

Working on heart rate prediction has been an insightful challenge. I aimed to clean, analyze, and forecast heart rate data using time series methods to ensure accurate predictions. The process took raw, irregular data and transformed it into meaningful insights through thorough preprocessing and model testing.

Initial Data Exploration and Cleaning

The dataset contained minute-by-minute heart rate readings over four hours. Initially, I noticed extreme outliers, with some values reaching 61,000 BPM—which is biologically impossible. The visual contrast was remarkable before outliers removal. Outliers dominated the graph, distorting any meaningful analysis.

Show code

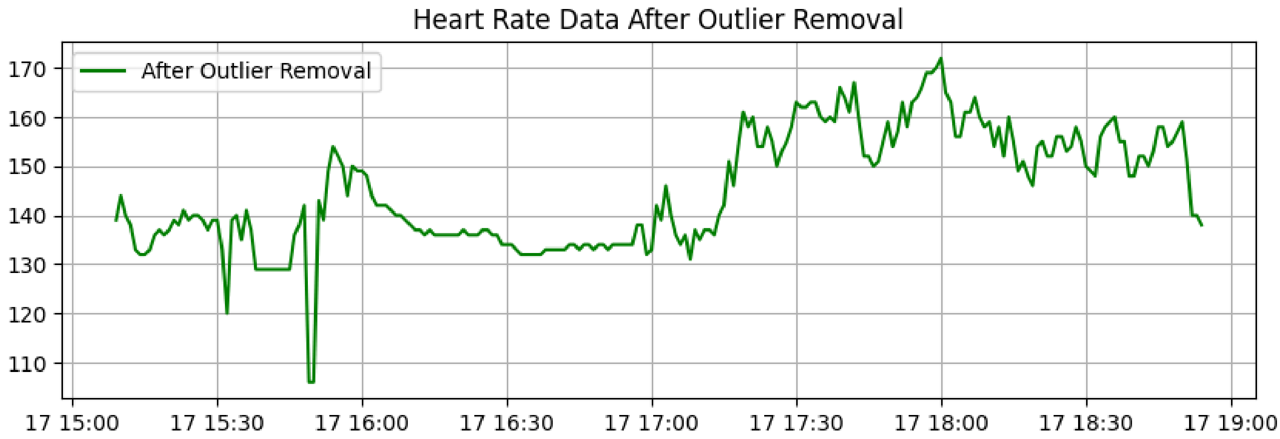


Outlier Detection and Removal

Using the Z-score method with a threshold of 3 standard deviations, I removed these unrealistic spikes. Missing values were then handled with forward and backward filling.

After cleaning, the heart rate data normalized, settling within a realistic range of 120-170 BPM.

Show code



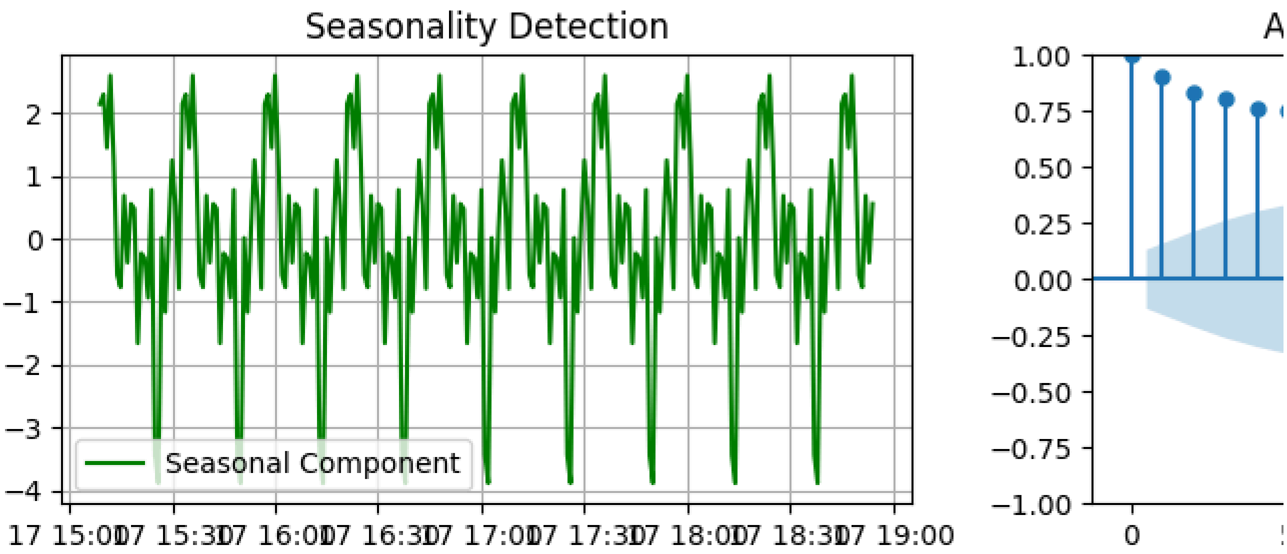
Understanding Time Series Data Patterns

Seasonality Stationary Detection

I performed Seasonal Decomposition and found clear hourly seasonality patterns. Autocorrelation analysis confirmed a strong correlation at regular intervals, reflecting a likely hourly seasonal pattern.

```
# Seasonal Decomposition
decomposition = seasonal_decompose(series, model='additive', period=period)
# Autocorrelation Plot (ACF) to Verify Seasonality
sm.graphics.tsa.plot_acf(series.dropna(), lags=24)  # Check up to 24 lags
```

Show code



KPSS Test for Stationarity: Identifying Trend in the Dataset

The **KPSS** test confirmed that the dataset was non-stationary, meaning that statistical properties changed over time due to an underlying trend. This suggests that transformations such as differencing or detrending may be necessary before applying time series forecasting models.

```
def check_stationarity(series):
  statistic, p_value, _, _ = kpss(series.dropna(), regression='c')
        result = "The series has a trend (Not Stationary)" if p_value < 0.05 else "The se
```

Show code

	Statistic	p-value	Stationarity
0	1.490202	0.01	The series has a trend (Not Stationary)

Evaluating Transformations for Different Models

Initially, I experimented with different transformations based on the stationarity and seasonality analysis. The goal was to apply the most suitable transformation for each model to achieve the best predictive accuracy:

Exponential Smoothing & ARIMA: Used `python df['Deseasonalised_Heart_Rate']` to manage both trend and seasonality.

SARIMA: Required `python df['Heart_Rate_diff']` to capture seasonal and non-seasonal trends.

TBATS: Paired with `pythondf['Log_Heart_Rate']`, as TBATS is designed to handle complex seasonal patterns.

Discovering the Best Transformation: Log Transformation

After extensive experimentation, I found that using only the Log Transformation (`pythonnp.log(df['Heart_Rate'] + 1)`) produced the most consistent results across all models. Initially, I applied deseasonalization and differencing, but later realized these transformations were unnecessary as most models could inherently handle seasonality and trends.

The log transformation alone effectively stabilized variance, preserving natural patterns while reducing extreme fluctuations.

Key realization: Removing unnecessary transformations improved model interpretability and generalization.

Code Implementation of Different Transformations

(A) Initial Transformations Attempted

```
# Log Transformation (Handling Zero Values)
df['Log_Heart_Rate'] = np.log(df['Heart_Rate'] + 1)

# Seasonal Differencing (Removing Seasonality)
df['Deseasonalised_Heart_Rate'] = df['Log_Heart_Rate'].diff(periods=60)
df['Deseasonalised_Heart_Rate'].fillna(method='bfill', inplace=True)

# First-Order Differencing (Removing Trend)
df['Heart_Rate_diff'] = df['Deseasonalised_Heart_Rate'].diff(periods=1)
df['Heart_Rate_diff'].fillna(method='bfill', inplace=True)
```

(B) Checking Stationarity After Transformation

```
check_stationarity(df['Heart_Rate_diff'])
```

Final Decision: Using Only Log Transformation

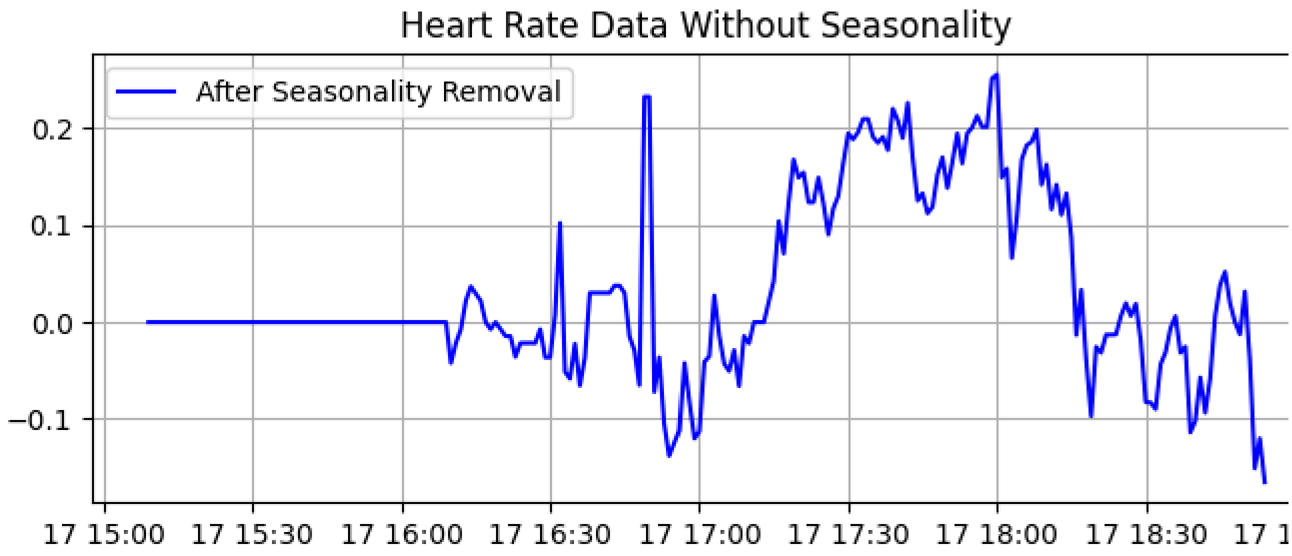
Given that most models inherently handle seasonality and trends, the final transformation applied was:

`python #Final Transformation: Log Only df['Log_Heart_Rate'] = np.log(df['Heart_Rate'] + 1)`

This approach ensured minimal data manipulation while allowing each model to operate optimally with stable variance.

(C) Visualizing the Impact of Transformations

Show code



📊 Model Performance Summary

The table below shows how different models performed. TBATS had the best accuracy with the lowest errors.

Model	Order / Configuration	MSE	MAE	RMSE	MAPE (%)	Predicted Range (min - max)
Exponential Smoothing	Seasonal Periods: 60	0.00463	0.04851	0.06804	0.97	141.40 - 163.20
SARIMA	(1, 1, 1, 60)	0.00427	0.04966	0.06538	0.99	141.71 - 161.05
ARIMA	(1, 1, 1)	0.00206	0.03171	0.04536	0.64	154.59 - 156.45
TBATS	Seasonal Periods: 60	0.00200	0.03198	0.04474	0.64	154.27 - 155.09

🏆 Best Model: TBATS

TBATS had the lowest errors and handled seasonality and trends best using log-transformed data.

```
# TBATS Configuration
tbats_model = TBATS(seasonal_periods=[60]).fit(train_df["Log_Heart_Rate"])
TBATS Model Evaluation Metrics - MSE: 0.0020017226589508328, MAE: 0.03197649047511693, RM

array([5.05045808, 5.04892995, 5.04576773, 5.04588514, 5.0451865 ,
       5.04531059, 5.04514236, 5.0451929 , 5.04514947, 5.04516651,
```

```
       5.04515474, 5.04516008, 5.04515678, 5.0451584 , 5.04515746,
       5.04515794, 5.04515767, 5.04515782, 5.04515774, 5.04515778])
Range of the predicted result min_value=154.26640121444584 min_value=155.09395218906292
```

📊 Forecasted Heart Rate Inverse Results

Timestamp	Predicted Heart Rate
2015-08-17 18:35:00	155.093952
2015-08-17 18:36:00	154.855602
2015-08-17 18:37:00	154.363530
2015-08-17 18:38:00	154.381774
2015-08-17 18:39:00	154.273255
2015-08-17 18:40:00	154.292524
2015-08-17 18:41:00	154.266401
2015-08-17 18:42:00	154.274249
2015-08-17 18:43:00	154.267506
2015-08-17 18:44:00	154.270152
2015-08-17 18:45:00	154.268323
2015-08-17 18:46:00	154.269153
2015-08-17 18:47:00	154.268641
2015-08-17 18:48:00	154.268892
2015-08-17 18:49:00	154.268746
2015-08-17 18:50:00	154.268821
2015-08-17 18:51:00	154.268779
2015-08-17 18:52:00	154.268801
2015-08-17 18:53:00	154.268789
2015-08-17 18:54:00	154.268795