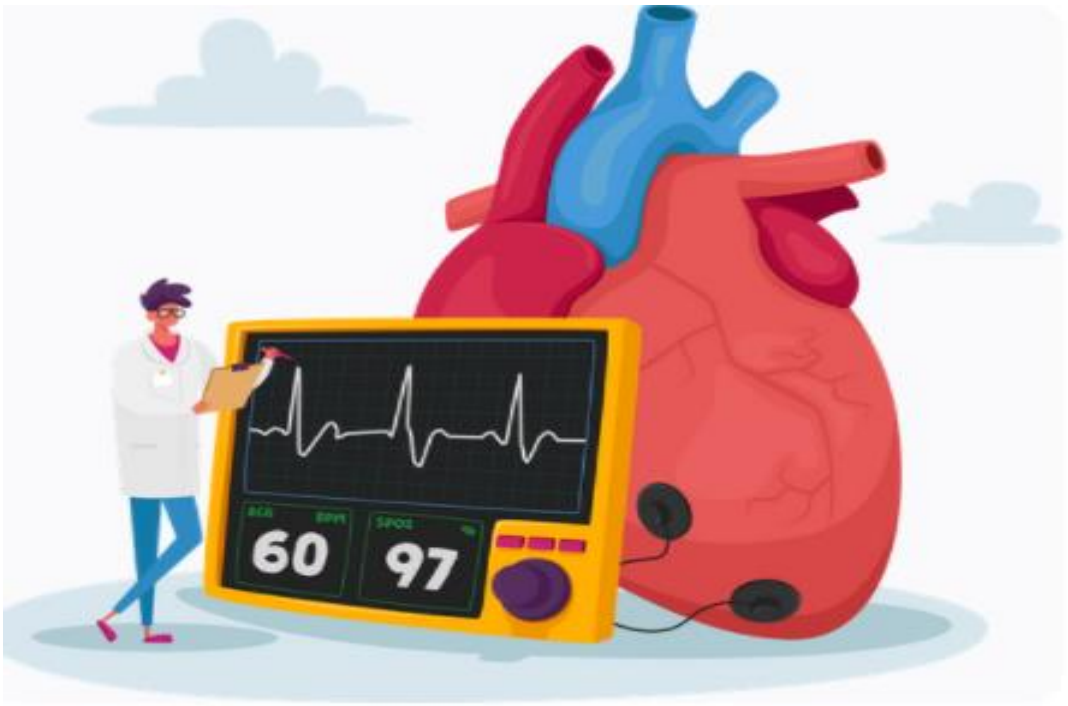
Predicting Heart Rate using Time Series Forecasting with Python

The Ultimate Guide for Time Series Analysis and Forecasting on Heart rate Data



Blog By: Chaithanya Vamshi Sai

Student ID: 21152797

Introduction

Sudden variations in heart rate could lead to the risks of stroke, heart failure, cardiac arrest, and also death. Preventing heart disease is important and with advancements in AI, predicting heart disease can have more accurate future predictions allowing healthcare sectors to detect and treat such situations at early stages making sure that more people can live healthy lives.

In this blog, I will demonstrate and show how we can harness the power of Machine Learning and apply it in healthcare. I will walk you through the entire process of how to predict Heart rate using Time Series Forecasting with Python.

What is Time Series Forecasting?

A time series is a sequence where a metric is recorded over regular time intervals. Depending on the frequency of observations, a time series may typically be hourly, daily, weekly, monthly, quarterly and annual.

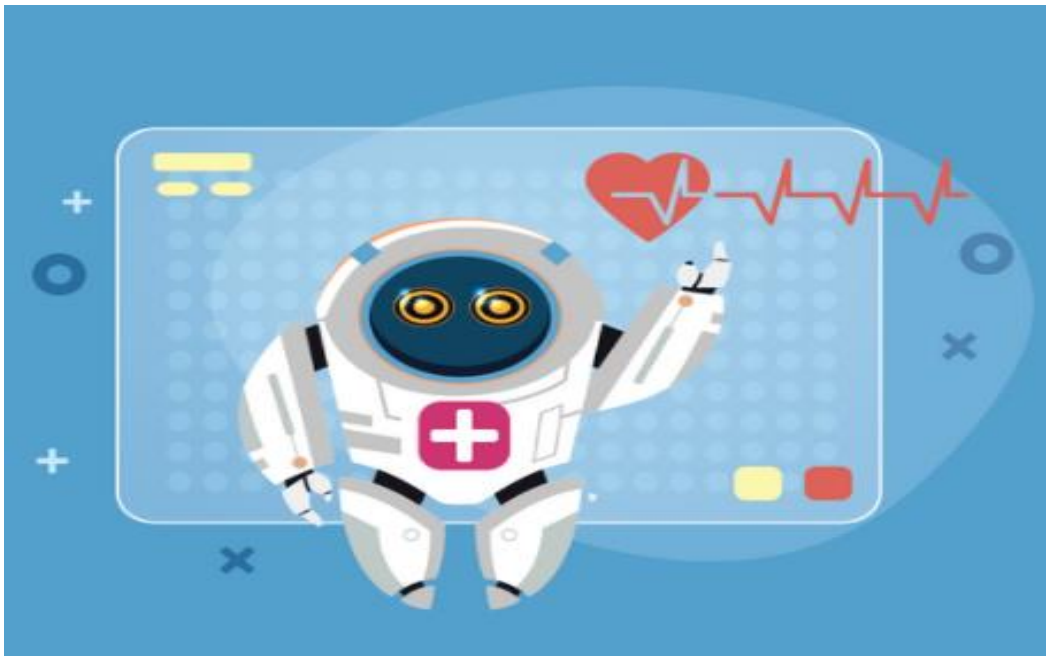
Forecasting involves taking models to fit historical data and making predictions about the future of time-series data.

But why forecast?

Because forecasting a time series is often of tremendous commercial value. In the healthcare domain, it drives the fundamental business planning, procurement, disease predictions and production activities. So, it's important to get the forecasts accurate to save on costs and is critical to success.

Problem Statement

The objective of this task is we are given a data set of time series collected using medical sensors, approximately four hours of data for a patient. Using these data, we will build an effective time series model to predict the next 20 observations (minutes) of heart rate data (Lifetouch Heart Rate).



Importing Libraries and Reading Data

```
# Import libraries

# Data Manipulation
import numpy as np
import pandas as pd
from    pandas import DataFrame

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl

# Statistics
import pmdarima as pm
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from dateutil.parser import parse
import pickle

# Set the options
pd.set_option('display.max_rows', 800)
pd.set_option('display.max_columns', 500)
%matplotlib inline
plt.rcParams.update({'figure.figsize': (12, 8), 'figure.dpi': 120})
```

```
# Input file name with path
input_file_name = '/content/PT_Train.csv'

# Target class name
input_target_variable = 'Heart_Rate'

# Date column name
input_date_variable = 'Timestamp'

#Exogenous variable
input_exogenous_variable = 'Oximeter SpO2'

# Handle missing value
input_treat_missing_value = 'bfill' # choose how to handle missing values from 'ffill','bfill'

# Box-cox transformation flag
input_transform_flag = 'Yes' # choose if you wish to transform the data - 'Yes' or 'No'

# Seasonality
input_seasonality = 12
input_order = (0, 1 , 2)
input_seasonal_order = (2, 1, 0, input_seasonality)

# Forecasting algorithm
# choose the forecasting algorithm from 'auto_arima', 'auto_sarima', 'holt_winters'
'simple_exponential_smoothing'
input_ts_algo = 'auto_arima'
```

Overview of the Dataset

Before attempting to solve the problem, it's very important to have a good understanding of the data.

- Get the descriptive statistics of the data
- Get the information about missing values in the data

```
# First 5 rows of the dataset
df.head()

# Dimension of the data
df.shape

#Info about data types
df.info()

# Summary of the dataset
df.describe()

# Missing values for every column
df.isna().sum()
```

Exploratory Data Analysis

Exploratory data analysis is an approach to analyse or investigate data sets to find out patterns in the data. Primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing tasks.

Handling Missing Values

Time Series algorithms don't work if the data is missing. So, we need to handle the missing values to get the predictions. There are various ways to handle missing values for time series data.

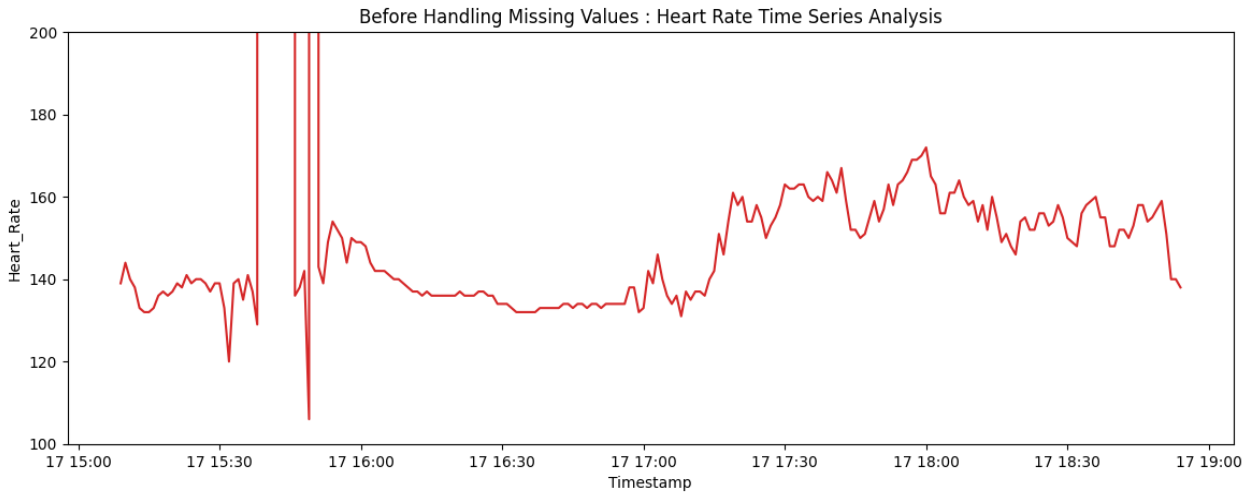
- Forward Fill
- Backward Fill
- Replace

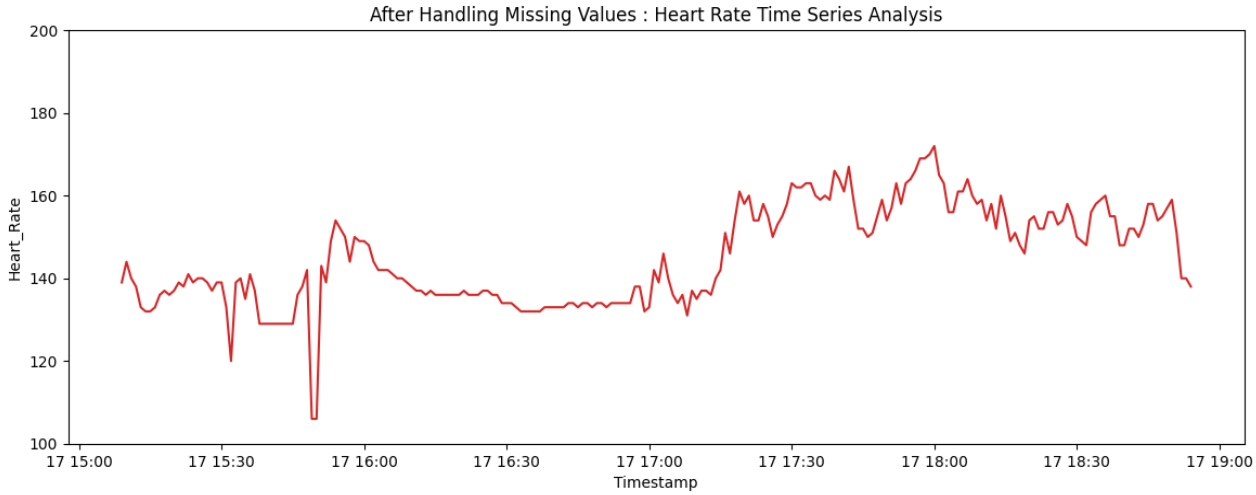
```
# Select how you wish to treat missing values according to the input
df = df.ffill()
df = df.bfill()

#Replacing the extreme values with Nan and forward filling
df = df.replace({'Heart_Rate':{61441:np.nan, 61442:np.nan}, 'Lifetouch Respiration Rate':
{61441:np.nan, 61442:np.nan}})

df = df.ffill()

#Checking  Missing Values
df.isna().sum()

#Storing cleaned dataset
df.to_csv('data.csv')
```

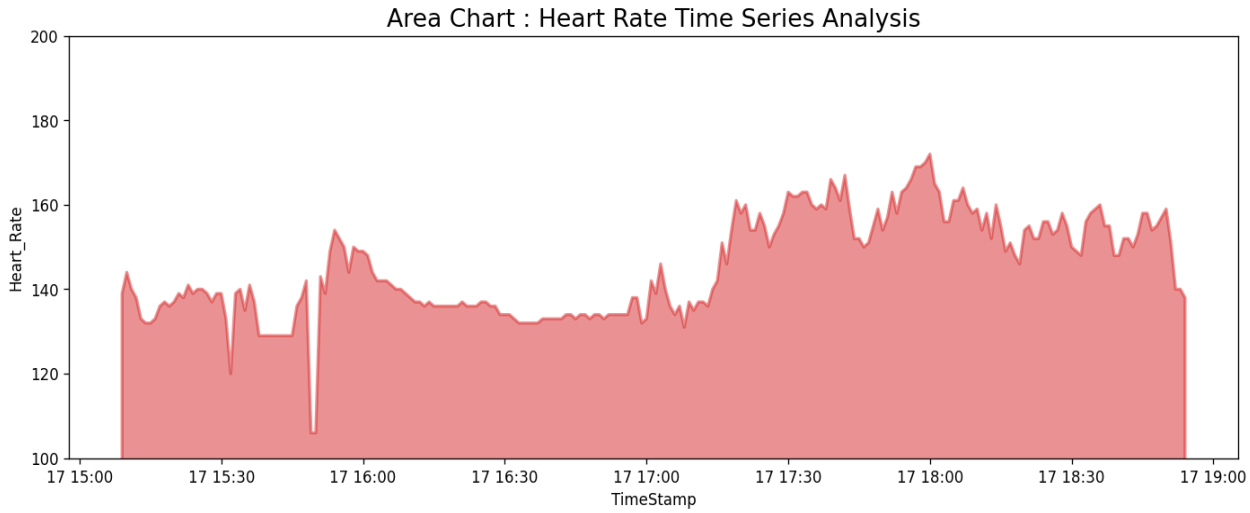
Visualizing the Time series data

A line plot and area chart of a time series can provide a lot of insight into the problem. Some observations from the plot can include:

- Whether the trend appears to be level around the mean
- Whether there appear to be any obvious outliers





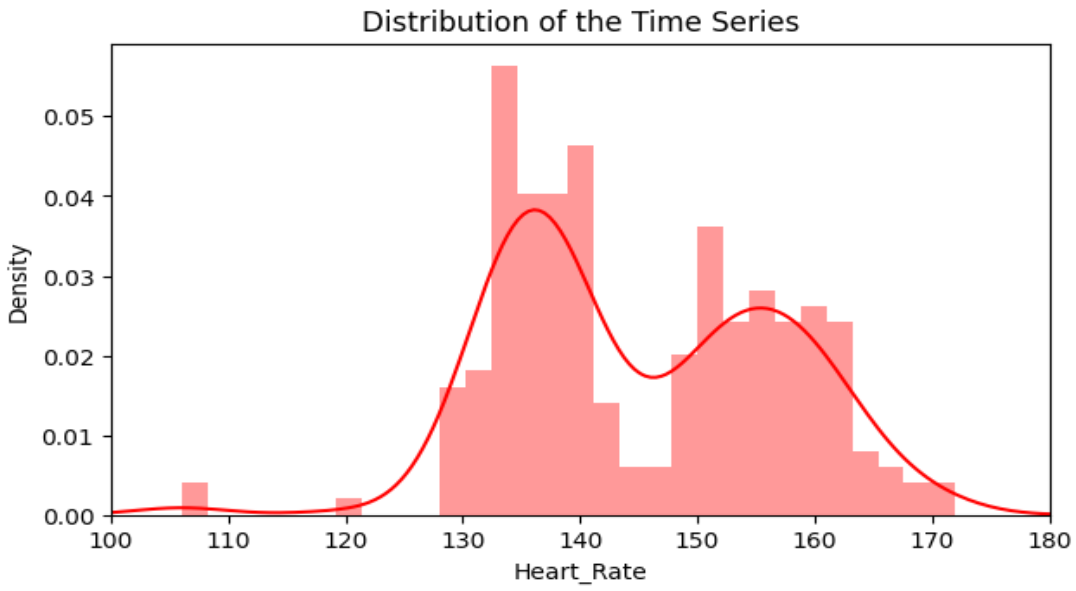


Checking Distribution of the Time series data

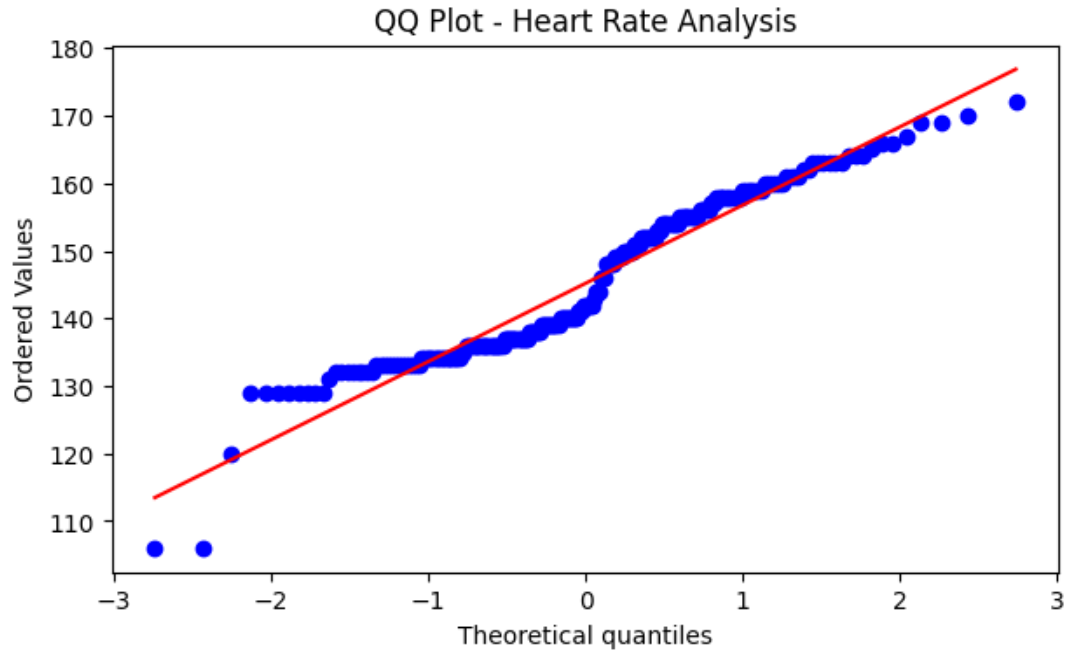
We should check the distribution of the time series so that we can decide if we need to transform the data or can be used as it is.

Additionally, it provides insights into the type of distribution the data follows.

- **Histogram to check the distribution of the attribute**



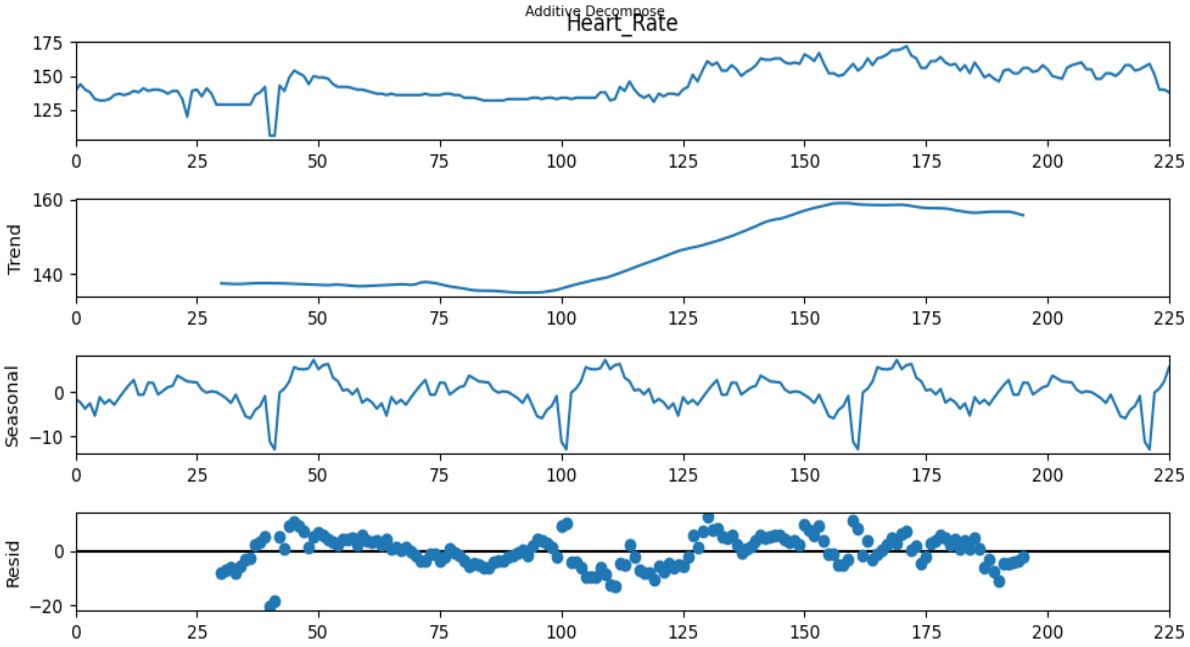
- **Q-Q Plot to check Normal Distribution of attribute**



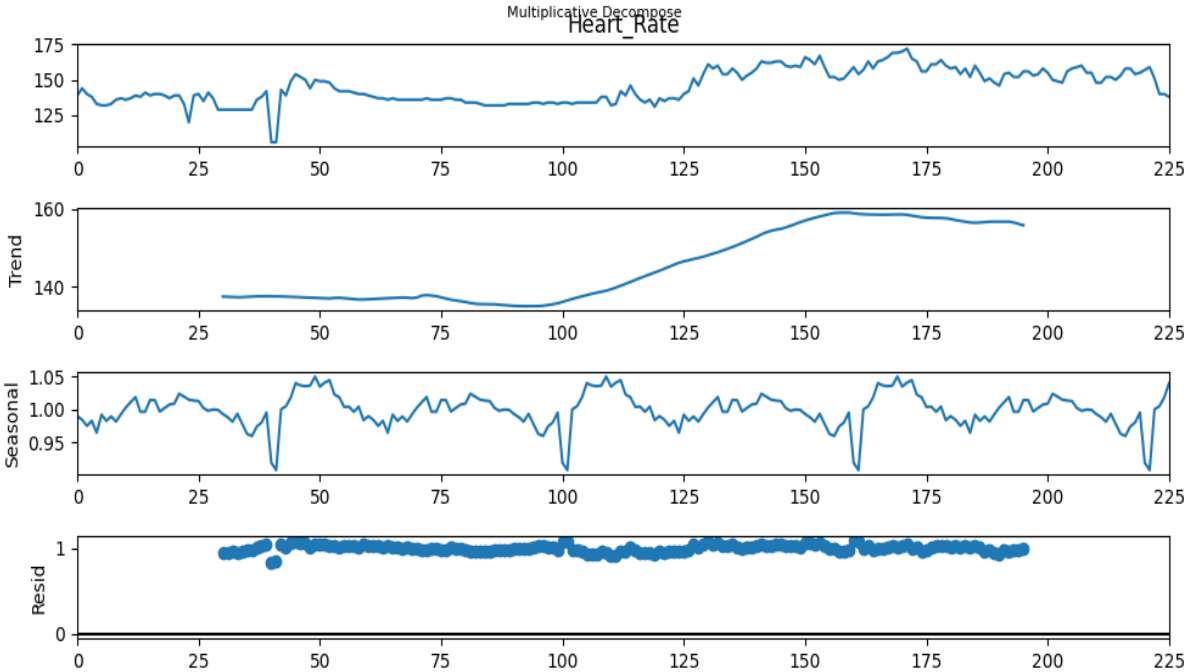
Time Series Decomposition

Depending on the nature of the trend and seasonality, a time series can be modelled as an additive or multiplicative, wherein, each observation in the series can be expressed as either a sum or a product of the components.

- **Additive time series:** Value = Base Level + Trend + Seasonality + Error



- **Multiplicative Time Series:** Value = Base Level x Trend x Seasonality x Error



Stationarity test - Augmented Dickey-Fuller test (ADH Test)

There are multiple tests to test if a time series is stationary or not. The most commonly used is the ADF test, where the null hypothesis is the time series possesses a unit root and is non-stationary. So, if the P-Value in the ADH test is less than the significance level (0.05), we reject the null hypothesis.

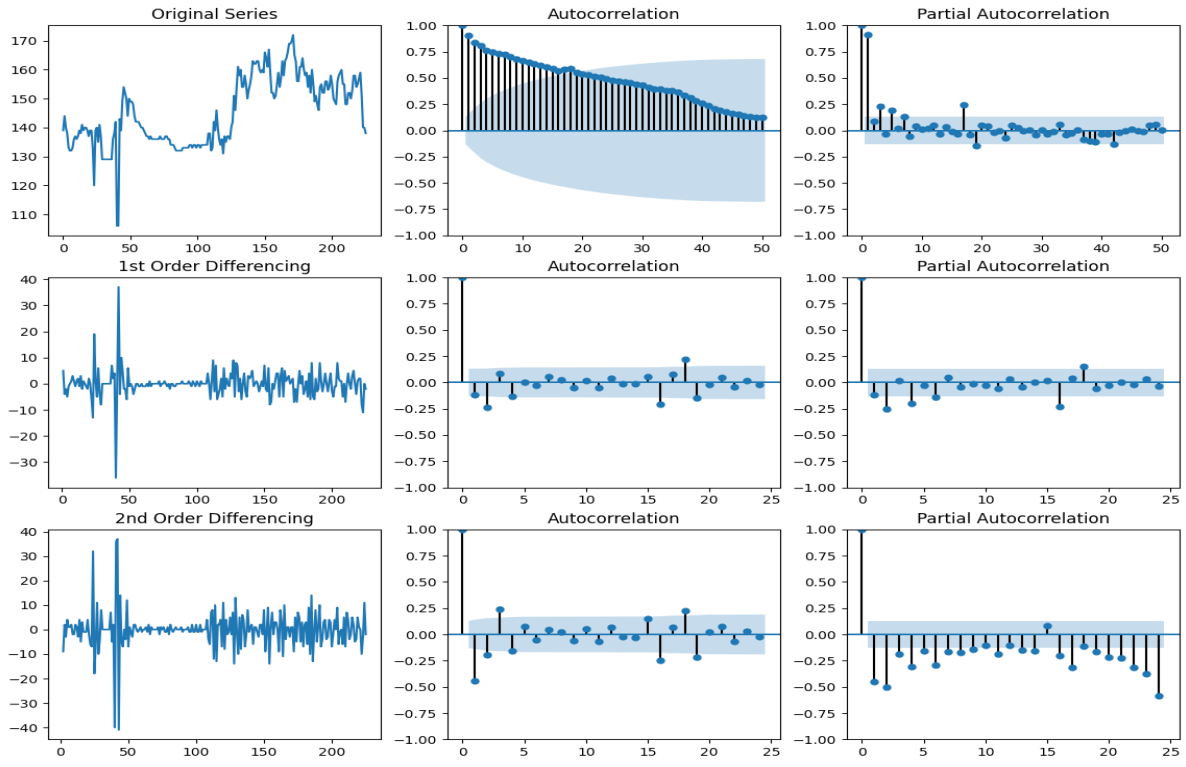
```
# ADF Test
result = adfuller(df[input_target_variable].values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')

Output :

ADF Statistic: -2.0062983264332157
p-value: 0.28381834919001564
Critial Values: 1%, -3.4602906385073884
Critial Values: 5%, -2.874708679520702
Critial Values: 10%, -2.573788599127782
```

Since the p-value obtained 0.28 is greater than significance level 0.05, we accept the null hypothesis i.e., time series is not stationary.

ACF-PACF plots and Order of Differencing

The right order of differencing is the minimum difference required to get a near-stationary series that roams around a defined mean and the ACF plot reaches zero fairly quick.

If the autocorrelations are positive for many numbers of lags (10 or more), then the series needs further differencing. On the other hand, if the lag 1 autocorrelation itself is too negative, then the series is probably over-differenced.



Model Building – Time Series Forecasting

Train – Test Split

```
#train and split the dataset
train_pct = 0.80
train_size = int(len(df) * train_pct)
test_size = len(df) - train_size
train, test = df[0:train_size], df[train_size:]
```

Model 1 - ARIMA

ARIMA, short for 'Auto-Regressive Integrated Moving Average' is a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modelled with ARIMA models.

The initial order of the ARIMA model can be determined using the ACF and PACF plots. The initial starting point for 'p' is the number of significant lags from the PACF plot and 'q' comes from the number of significant lags from the ACF plot.

Model 2 – Auto ARIMA

When implementing an ARIMA model, it is particularly common to automate the selection of the p, d, q coordinates using a library such as pmdarima in Python.

ARIMA model is designed to work with non-stationary data. It does this by specifying a value for the d parameter, or the number of differences that are necessary to make the model stationary.

The only difference is one can either elect to manually select these parameters by inspecting the data.

Alternatively, numerous packages have been designed to automate the selection of these parameters.

Model 3 – Simple Exponential Smoothing

Single Exponential Smoothing is a time series forecasting algorithm for univariate data which doesn't have a trend or seasonality. Only one parameter is required for SES - alpha (a). It is the smoothing coefficient.

Model 4 – Holt-Winters

Holt-Winters is a time series forecasting algorithm. It models three aspects of the time series: a typical value (average), a slope (trend) over time, and a cyclical repeating pattern (seasonality). We need to pass on the parameter seasonality to it.

Model 5 – FB Prophet

Time series modelling released by Facebook in 2017, forecasting tool Prophet is designed for analysing time-series that display patterns on different time scales such as yearly, weekly, daily and hourly.

It also has advanced capabilities for modelling on a minute time-series and implementing custom changepoints. Therefore, we are using FB Prophet to get a model up and running.

Summary of Time Series Model Evaluation

Below is the summary table showing model evaluation metrics such as Root Mean Square Error (RMSE) score, Mean Absolute Error (MAE) Score on Training Data set for different time series models.

From all the models, the ARIMA model achieved the best MAE score around 3.316 implies the model is about 96.684% accurate in predicting the next 20 observations.

Also, the lower value of the RMSE score of 4.203 indicates a better fit.

Time Series Model	RMSE Score	MAE Score
ARIMA	4.203	3.316
Auto ARIMA	9.604	8.114
Simple Exponential Smoothing	9.517	8.027
Holt-Winters	14.472	12.52
FB- Prophet	6.393	4.735

Implementation of Model building using Time series forecasting Algorithms

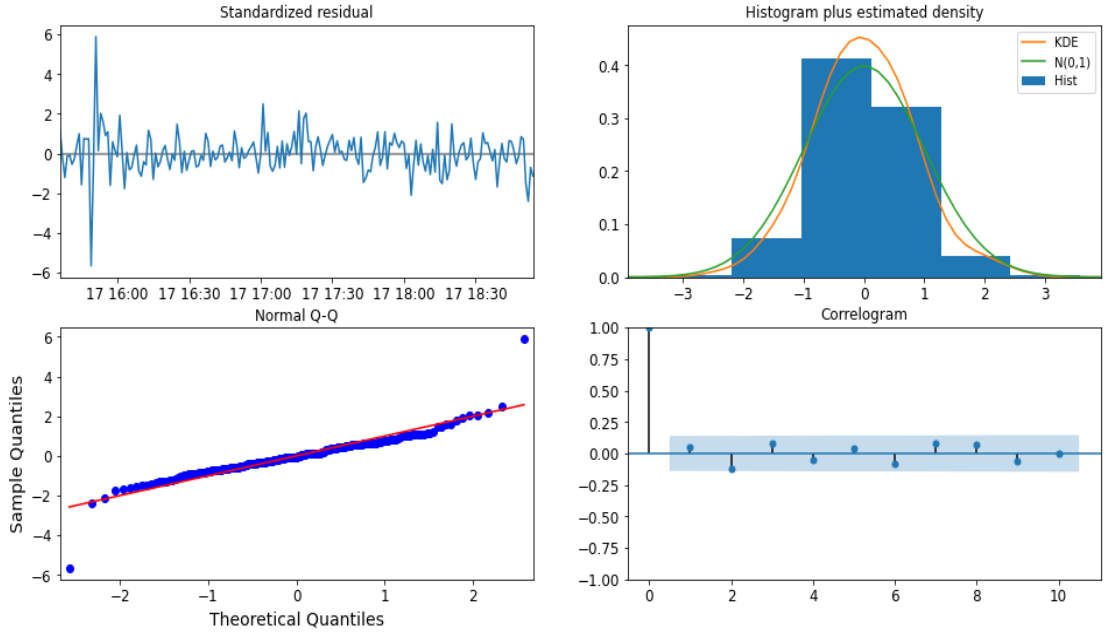
Python Source Code of the Project Link: **GitHub**

Interpreting the Residual plots in the ARIMA model

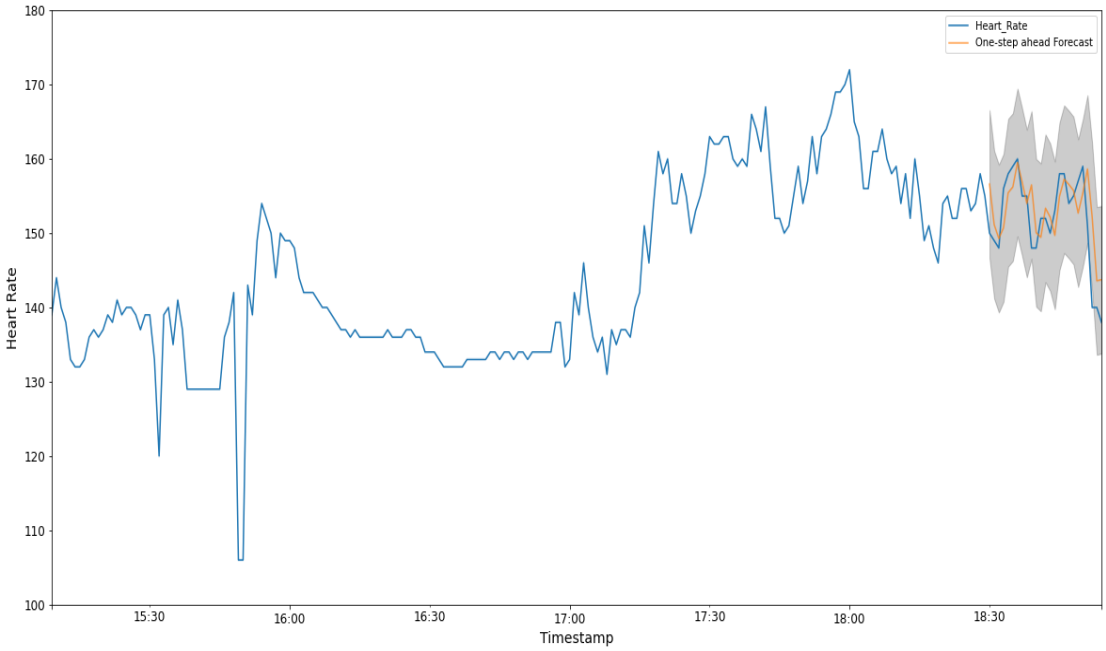
- **Standardized Residual:** The residual errors seem to fluctuate around a mean of zero and have a uniform variance.
- **Density Plot:** The density plot suggests normal distribution with a mean zero.
- **Q-Q Plot:** All the dots should fall perfectly in line with the red line. Any significant deviations would imply the distribution is skewed.

- **Correlogram:** ACF plot shows the residual errors are not autocorrelated. Any autocorrelation would imply that there is some pattern in the residual errors which are not explained in the model.

  Overall better model and let's predict on test data and forecast.

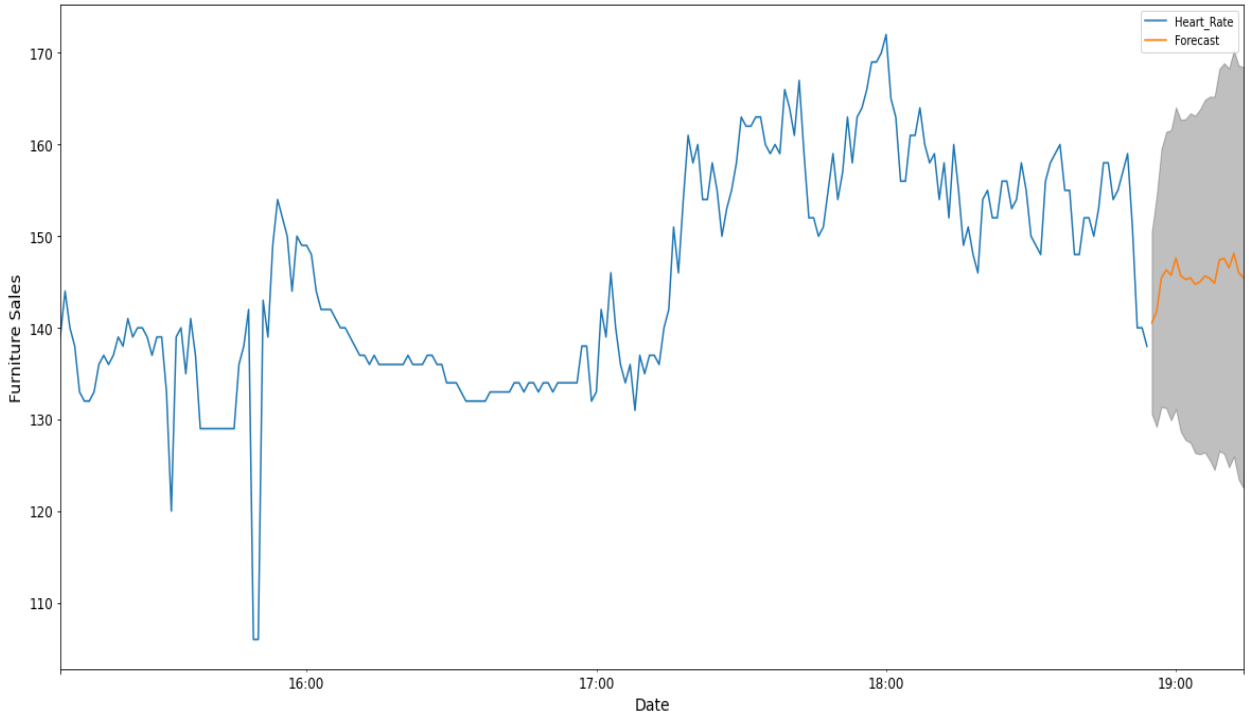


Predictions on Test Data



Forecast on Next 20 Heart Rate Observations

Using the best time series model ARIMA, we will make forecasts (Future Predictions) and save the predictions to a .csv file named predictions.csv.



Forecasted (Future Predictions) in .csv file

```
  forecasts = results.forecast(steps=20)
  print(forecasts)

--INSERT--
```

```
2015-08-17 18:55:00    140.548966
2015-08-17 18:56:00    141.779846
2015-08-17 18:57:00    145.461597
2015-08-17 18:58:00    146.316806
2015-08-17 18:59:00    145.743794
2015-08-17 19:00:00    147.609589
2015-08-17 19:01:00    145.680301
2015-08-17 19:02:00    145.270993
2015-08-17 19:03:00    145.448008
2015-08-17 19:04:00    144.742557
2015-08-17 19:05:00    145.040112
2015-08-17 19:06:00    145.648534
2015-08-17 19:07:00    145.400936
2015-08-17 19:08:00    144.861930
2015-08-17 19:09:00    147.423558
2015-08-17 19:10:00    147.569866
2015-08-17 19:11:00    146.548205
2015-08-17 19:12:00    148.130060
2015-08-17 19:13:00    146.021073
2015-08-17 19:14:00    145.498037
```

Conclusion

In this blog, we have learnt how to approach a Time Series forecasting problem. We started with an understanding problem, exploration of data. Then we handled missing values, stationary tests and visualised time series data.

Finally, we were able to apply different Time series algorithms and make forecasts (future predictions) on the data set using the best Time series model ARIMA which outperformed the other models.

References

https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/

https://machinelearningmastery.com/work-time-series-forecast-project/

https://pythondata.com/forecasting-time-series-data-prophet-part-4/