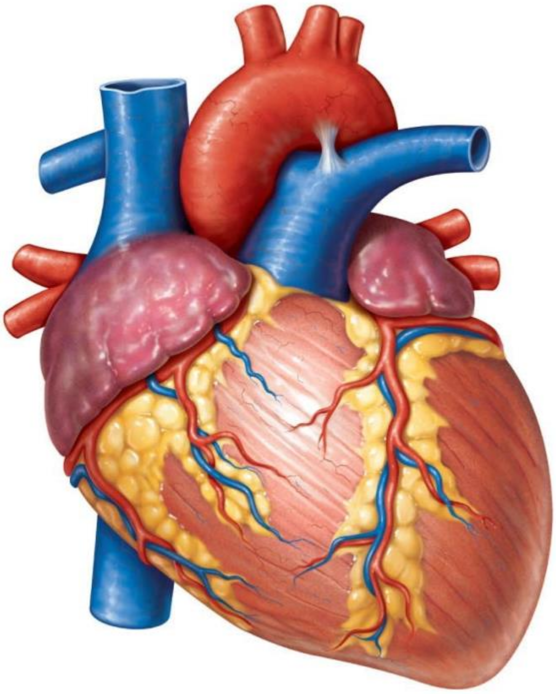
A Guide to forecasting Heart Rates

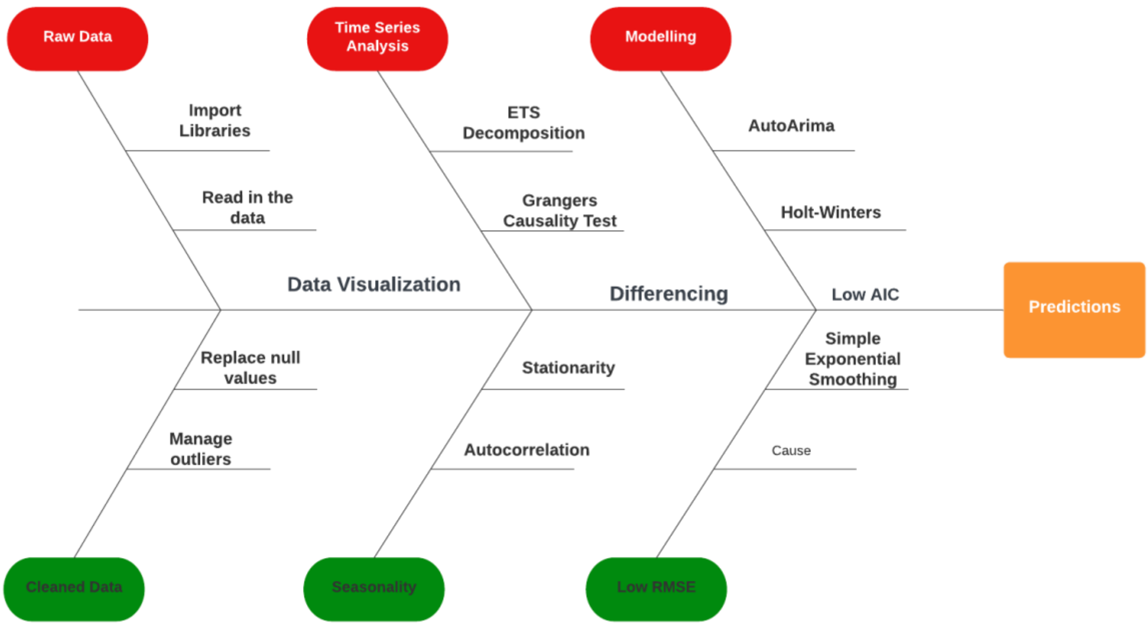


Oluwafunmito Adeyemi 22165053
Colab: Click Here

Matters of the heart can't be overemphasized. We would be going through the process of forecasting, using well known tools in machine learning.

The process flow:

FLOW CHART FOR
TIME SERIES
ANALYSIS



DATASET DESCRIPTION

Column Name	Data Type	Description
Timestamp (GMT)	TimeStamp	Time when each record is taken
Lifetouch Heart Rate	Numerical	heartbeats/minute
Lifetouch Respiration Rate	Numerical	breaths/minute
Oximeter SpO2	Numerical	Oxygen level
Oximeter Pulse	Numerical	Pulse Rate

LIBRARY IMPORTATION

```
import numpy as np
import pandas as pd

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


# Statistics
from statsmodels.tsa.arima.model import ARIMA, ARIMAResults, SARIMAXSpecification
import pmdarima as pm
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing
from statsmodels.tsa.stattools import adfuller,kpss,coint,bds,q_stat,grangercausalitytests,levinson_durbin
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.stats.stattools import durbin_watson
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.special import inv_boxcox
from scipy.stats import boxcox
from scipy.interpolate import interp1d
from sklearn.metrics import mean_absolute_error, mean_squared_error
from dateutil.parser import parse
from statsmodels.tools.eval_measures import rmse

%matplotlib inline
plt.rcParams.update({'figure.figsize': (12, 8), 'figure.dpi': 100})
import warnings
warnings.filterwarnings("ignore")
```

✓ 0.8s Python

DATA IMPORTATION

```
#Reading in the data
df = pd.read_csv('PT_Train.csv', index_col=0, parse_dates=True)

#Casting the heart rate column to float
df['Lifetouch Heart Rate'] = df['Lifetouch Heart Rate'].astype(float)

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

✓ 0.7s

	Lifetouch Heart Rate	Lifetouch Respiration Rate	Oximeter SpO2	Oximeter Pulse
Timestamp (GMT)				
2015-08-17 15:09:00	139.0	41	NaN	NaN
2015-08-17 15:10:00	144.0	40	92.0	140.0
2015-08-17 15:11:00	140.0	42	89.0	144.0
2015-08-17 15:12:00	138.0	45	93.0	141.0
2015-08-17 15:13:00	133.0	42	94.0	134.0

DATA WRANGLING

The ffill and bfill were used because they replace null values by mirroring similarities

```
#checking for null values
df.isna().sum()
```

✓ 0.2s

```
Lifetouch Heart Rate          0
Lifetouch Respiration Rate    0
Oximeter SpO2                35
Oximeter Pulse               35
dtype: int64
```

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

✓ 0.1s

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

✓ 0.7s

```
Lifetouch Heart Rate          0
Lifetouch Respiration Rate    0
Oximeter SpO2                 0
Oximeter Pulse                0
dtype: int64
```

MANAGING OUTLIERS

Heart rates > 300 are and are replaced with moderate values.

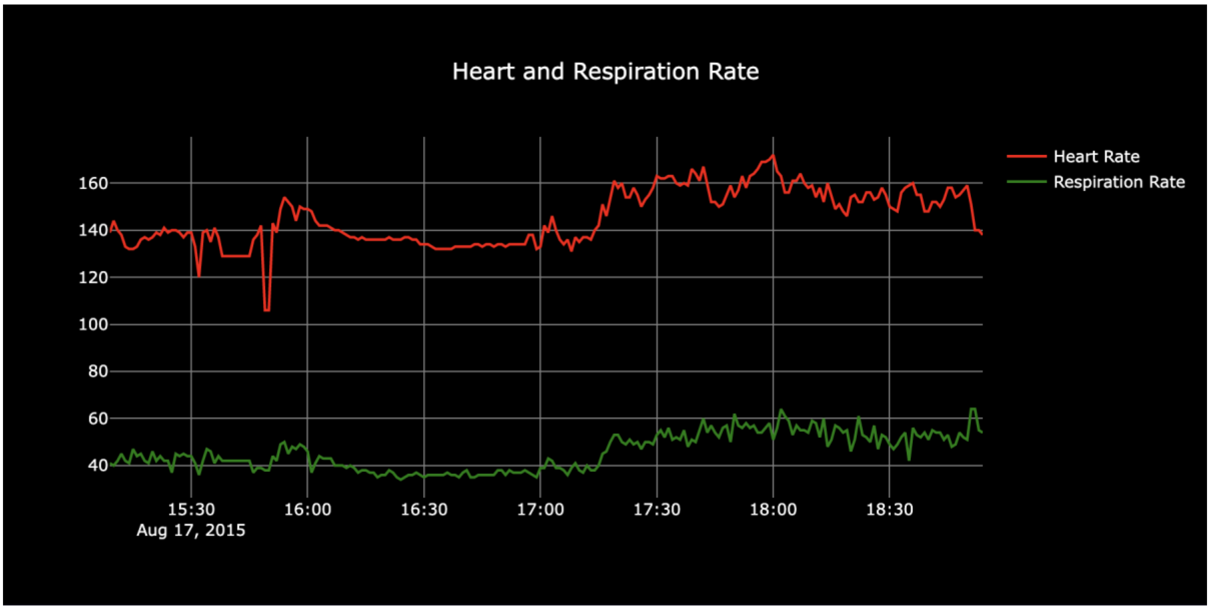
```
df = df.replace({'Lifetouch Heart Rate':{61441:np.nan, 61442:np.nan}, 'Lifetouch Respiration Rate':{61441:np.nan, 61442:np.
```

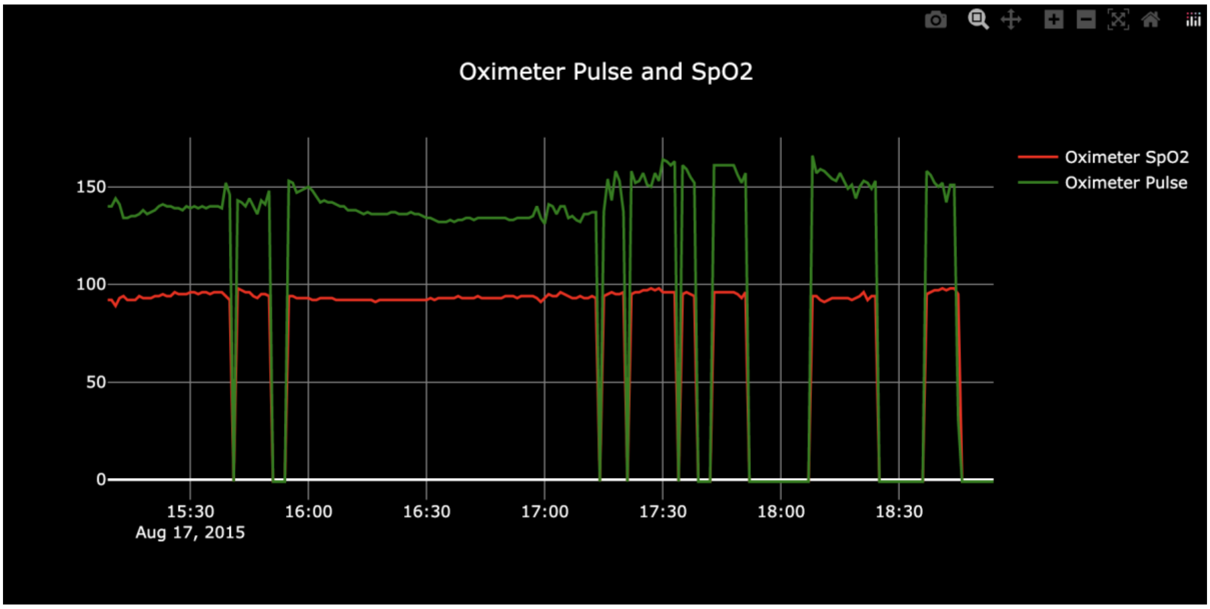
✓ 0.5s Python

DATA VISUALIZATION

Similarities are observed in the following:

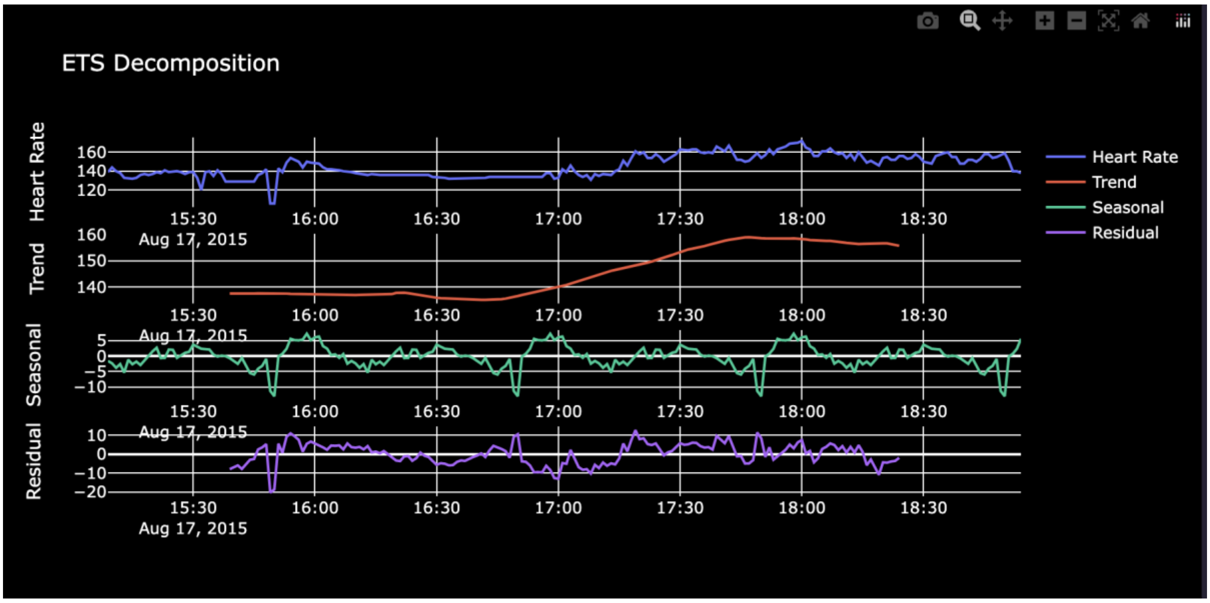
a. The Heart Rate and Respiration Rate
b. The OximeterSpO2 and OximeterPulse





ETS DECOMPOSITION

Trends, seasonality and errors are best accessed using an ETS Decomposition diagram:



GRANGER CAUSALITY TESTS

Below shows causality between Heart rate and Respiration Rate,

```
grangercausalitytests(df[['Lifetouch Heart Rate','Lifetouch Respiration Rate']],maxlag=3);
```

✓ 0.3s

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=11.4368 , p=0.0009  , df_denom=222, df_num=1
ssr based chi2 test:   chi2=11.5913 , p=0.0007  , df=1
likelihood ratio test: chi2=11.3026 , p=0.0008  , df=1
parameter F test:         F=11.4368 , p=0.0009  , df_denom=222, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=5.9751  , p=0.0030  , df_denom=219, df_num=2
ssr based chi2 test:   chi2=12.2231 , p=0.0022  , df=2
likelihood ratio test: chi2=11.9013 , p=0.0026  , df=2
parameter F test:         F=5.9751  , p=0.0030  , df_denom=219, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.9717  , p=0.1192  , df_denom=216, df_num=3
ssr based chi2 test:   chi2=6.1067  , p=0.1065  , df=3
likelihood ratio test: chi2=6.0246  , p=0.1104  , df=3
parameter F test:         F=1.9717  , p=0.1192  , df_denom=216, df_num=3
```

STATIONARITY

ADF(Augmented-Dickey-Fuller) test is most popular. If p-value<0.05, the time-series is stationary.

Function for ADF-Testing

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

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype: float64"

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```
✓ 0.2s

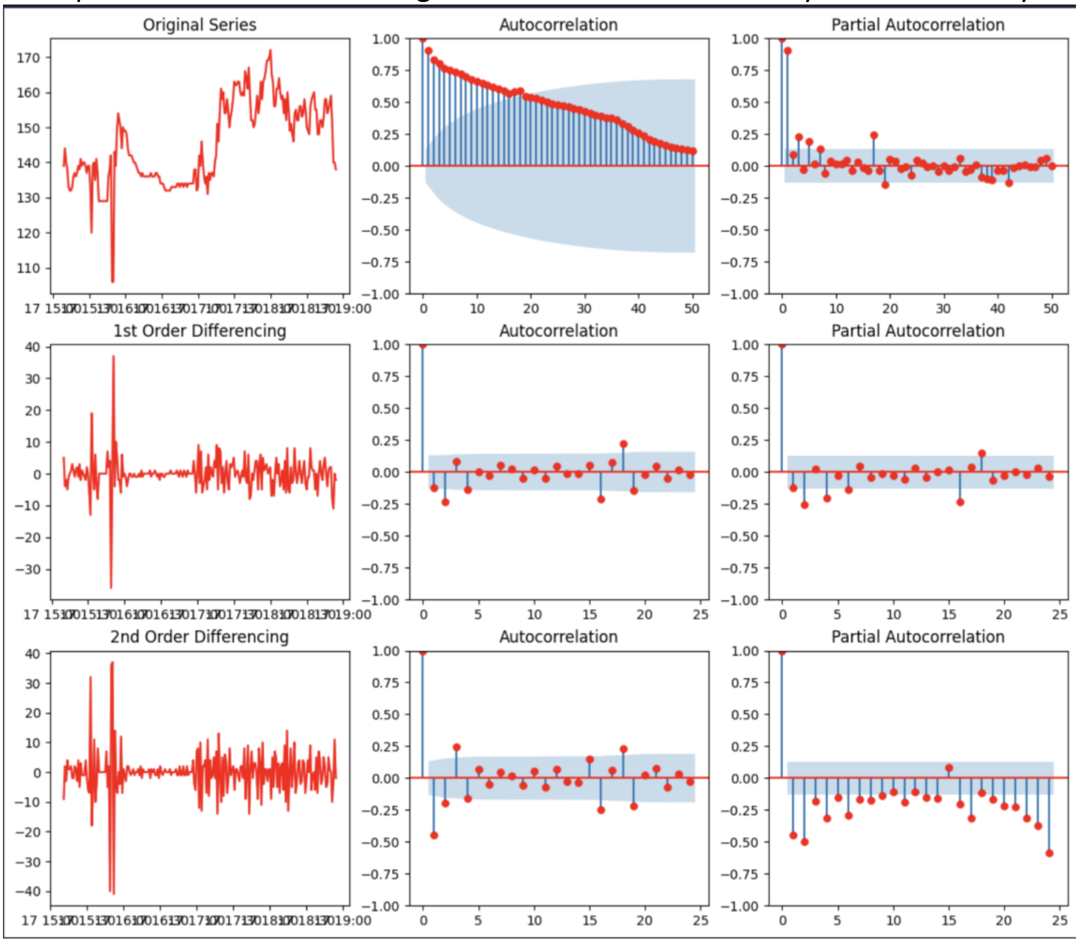
Heart Rate and Respiration rate indicate non-stationarity:

```
Stationarity test for the Heart Rate Column
Augmented Dickey-Fuller Test:
ADF test statistic       -2.006298
p-value                   0.283818
# lags used               4.000000
# observations          221.000000
critical value (1%)      -3.460291
critical value (5%)      -2.874709
critical value (10%)     -2.573789
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary

Stationarity test for the Respiration Rate Column
Augmented Dickey-Fuller Test:
ADF test statistic       -0.955644
p-value                   0.769097
# lags used               5.000000
# observations          220.000000
critical value (1%)      -3.460428
critical value (5%)      -2.874769
critical value (10%)     -2.573821
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

OPTIMAL DIFFERENCING ORDER WITH ACF AND PACF PLOTS

The optimal order of differencing is the smallest number that yields stationarity.



MODEL BUILDING

Data split:

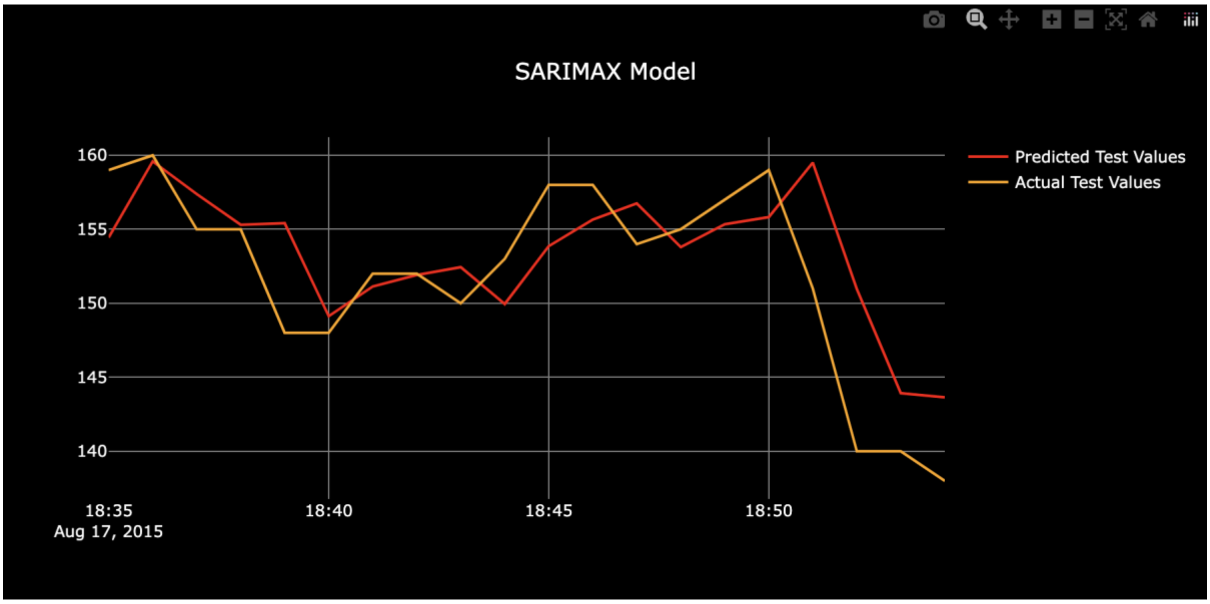
```
print(df['Lifetouch Heart Rate'].shape)
train=df['Lifetouch Heart Rate'].iloc[:-20]
test=df['Lifetouch Heart Rate'].iloc[-20:]
print(train.shape,test.shape)
```

✓ 0.3s

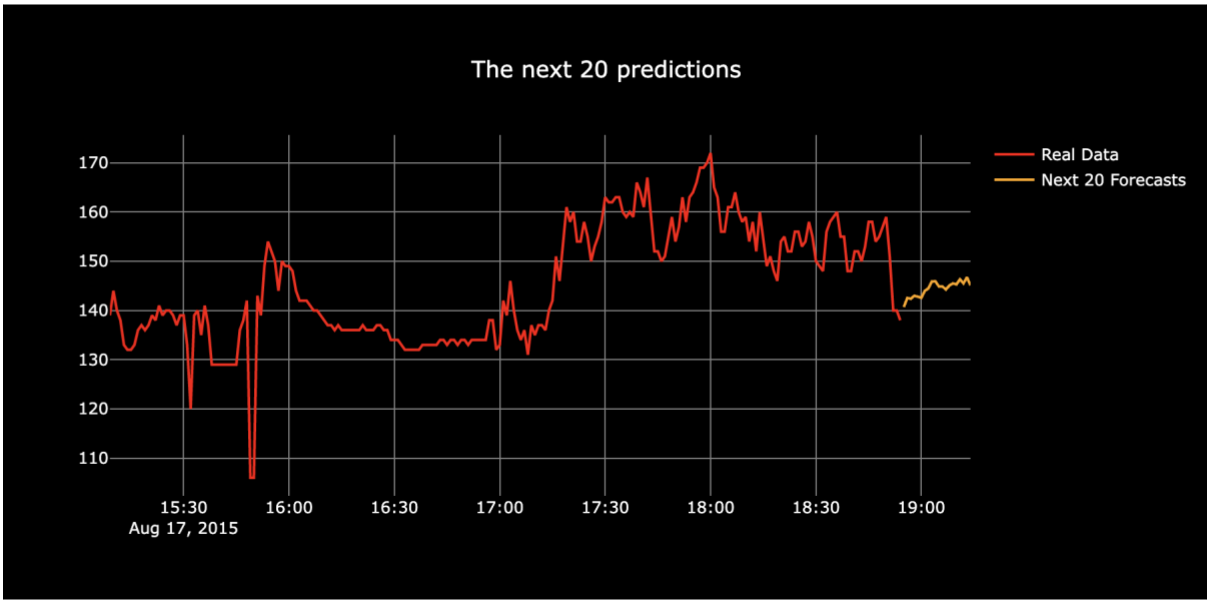
```
(226,)
(206,) (20,)
```

Auto-Arima:

Description of the Auto-Arima method can be found Here



Final 20:

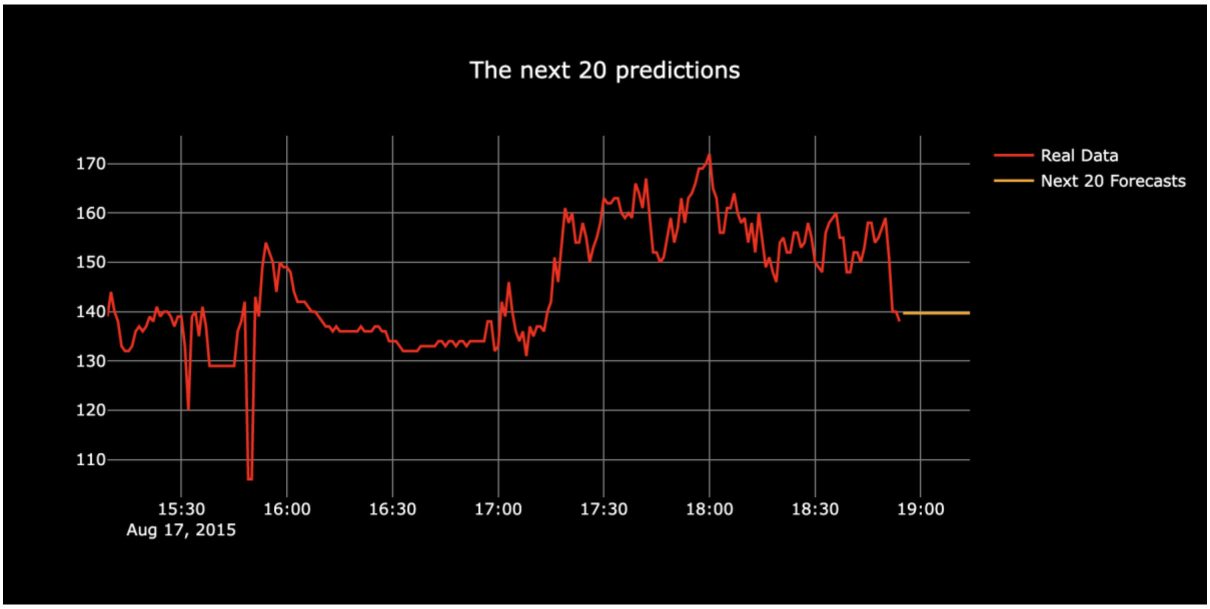


Holt-Winters

A detailed description of the Holt-Winters method can be found Here

Test vs Actual

Final 20

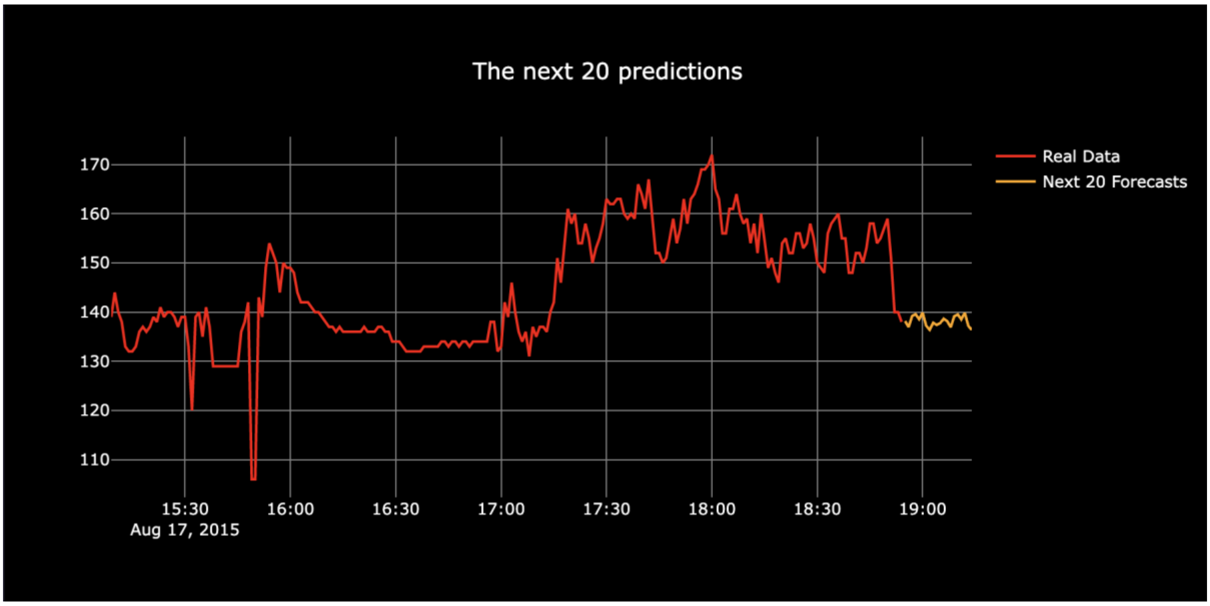


Simple Exponential Smoothing

A description of SES method can be found [Here]

Test vs Actual

Final 20



EVALUATION

MODEL	RMSE
SARIMA	6.644806887320173
HOLT-WINTERS	7.504877455841614
SIMPLE EXPONENTIAL SMOOTHING	8.322931587398356

OBSERVATIONS

1. The SARIMA model is performing better.

2. Holt-Winters may not be suitable for capturing more complex seasonal.
3. The SES is too simple for complex situations

FORECASTING

```
2015-08-17 18:35:00    155.816460
2015-08-17 18:36:00    155.511127
2015-08-17 18:37:00    154.828208
2015-08-17 18:38:00    155.001133
2015-08-17 18:39:00    153.816095
2015-08-17 18:40:00    154.156036
2015-08-17 18:41:00    154.560916
2015-08-17 18:42:00    153.684077
2015-08-17 18:43:00    154.865213
2015-08-17 18:44:00    155.922294
2015-08-17 18:45:00    155.915910
2015-08-17 18:46:00    156.219187
2015-08-17 18:47:00    156.066942
2015-08-17 18:48:00    155.469862
2015-08-17 18:49:00    154.870118
2015-08-17 18:50:00    155.619157
2015-08-17 18:51:00    155.169706
2015-08-17 18:52:00    154.271928
2015-08-17 18:53:00    155.020129
2015-08-17 18:54:00    154.271865
Freq: T, Name: predicted_mean, dtype: float64
```

In this blog, we we dealt with the tenets of solving a Time Series forecasting problem. As we saw, understanding the nature of the data is key to implementing right strategies and forecasting optimal results.