Assignment: 3 Time Series

Team Members: Jash Patel (M22CS061)
Asmita Rani (M22CS059)

Note: Only numpy is used for Part 1 and Part 2. For Part 3 all others libraries are used.

Part 1: ARIMA Model

- ARIMA has two part one is AR (Average Regression) and other is MA (Moving Average)
- ARIMA has also three parameters: p, d, q
- Known as
- p:The amount of prior time points used to forecast future values of the time series is known as the autoregressive (AR) order, or p. The dependence of the time series' present value on its earlier values is captured by this parameter.

- d: the integrated (I) order, which indicates how many times the time series must be differentiated before they become stationary. This parameter eliminates the non-stationarity of the data and preserves the trend in the time series.

- q: The amount of past error terms used to forecast future values of the time series is referred to as the moving average (MA) order. This parameter encapsulates how the residuals or errors from previous forecasts affect the time series' present value.

- Together, p, d, and q define the ARIMA model and the way it relates to the underlying time series data. The ARIMA model is denoted by ARIMA(p, d, q), where p, d, and q are integers representing the values of the respective parameters.

AR in ARIMA:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

- For an AR(1) model: $-1 < \phi_1 < 1$.
- For an AR(2) model: $-1 < \phi_2 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1$.

- where ε is white noise. This is like a multiple regression but with *lagged values* of y_t as predictors. We refer to this as an **AR(p) model**, an autoregressive model of order p.

MA in ARIMA:

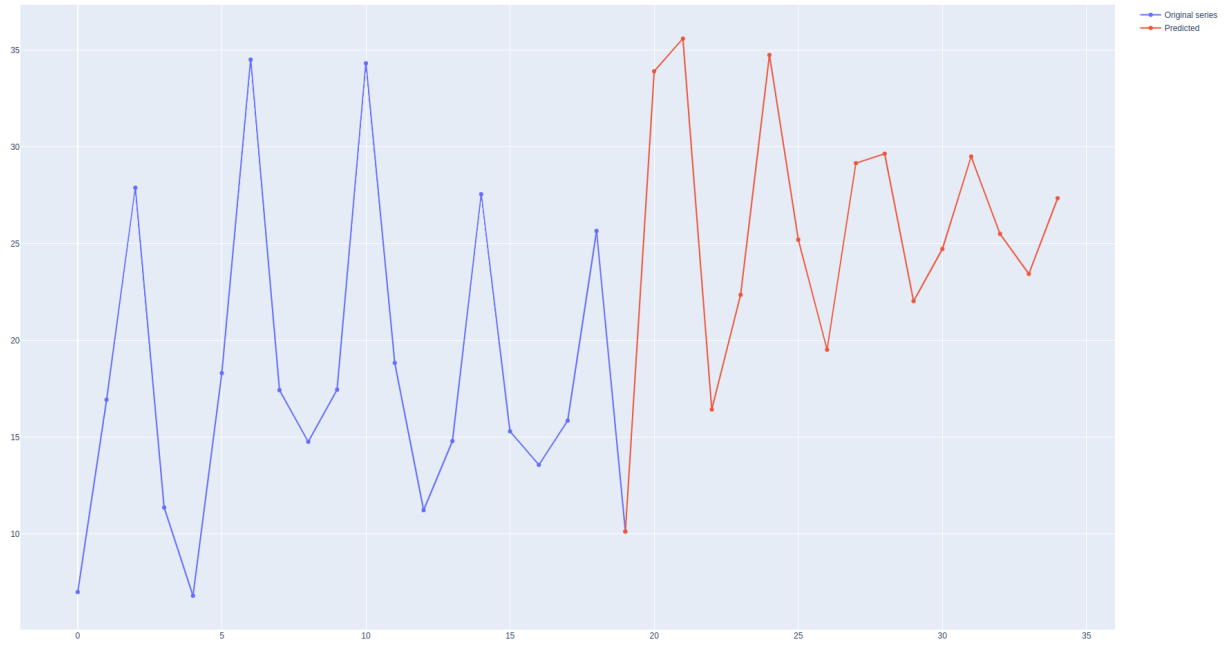
$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

- For an MA(1) model: $-1 < \theta_1 < 1$.
- For an MA(2) model: $-1 < \theta_2 < 1,\ \ \theta_2 + \theta_1 > -1,\ \ \theta_1 - \theta_2 < 1$.

$$\begin{aligned} y_t &= \phi_1 y_{t-1} + \varepsilon_t \\ &= \phi_1 (\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\ &= \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\ &= \phi_1^3 y_{t-3} + \phi_1^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\ &\text{etc.} \end{aligned}$$

- where ε is white noise. We refer to this as an MA(q) model, a moving average model of order q. Of course, we do not *observe* the values of ε, so it is not really a regression in the usual sense.

Final equation like = AR + MA



Part 2 Holt Winters'

- The analysis of time series data that exhibit pattern and seasonality is made possible by the well-known Holt-Winters time series forecasting method. Another name for this method is triple exponential filtering.
- Holt-Winters divides the time series into three categories: level, trend, and seasonality. The level component shows the series' beginning point or average value, the trend component shows the direction and rate of change over time, and the seasonality component shows a cyclical or repeating pattern in the data.
- We must identify the type of the seasonality component in the time series in order to decide whether the Holt-Winters technique should be used in an additive or multiplicative manner.
- Any level of the time series exhibits additive seasonality, where the magnitude of the seasonal fluctuations is constant. For instance, the seasonality is likely to be additive if the time series' seasonal fluctuations remain consistent from year to year.
- The seasonal variations, on the other hand, are proportional to the level of the time series in a multiplicative seasonality. For instance, the seasonality is likely to be multiplicative if the seasonal fluctuations in the time series grow as the level of the time series rises.
- The character of the seasonality component can be determined by plotting the time series or visually analyzing the seasonal pattern. If the seasonal variations appear to be stable over time, seasonality is presumably additive. If the seasonal differences appear to increase or decrease with the level of the time series, the seasonality is likely multiplicative.
- Another way to determine the sort of seasonality component is to run a statistical test. One of these tests, the Box-Cox test, can be used to determine whether the time series should undergo additive or exponential change.

Additive:

$$Overall\ Equation : \hat{y}_{t+h} = l_t + hb_t + s_{t+h-m}$$

$$Level\ Equation : l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$$

$$Trend\ Equation : b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

$$Seaonality\ Equation : s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$$

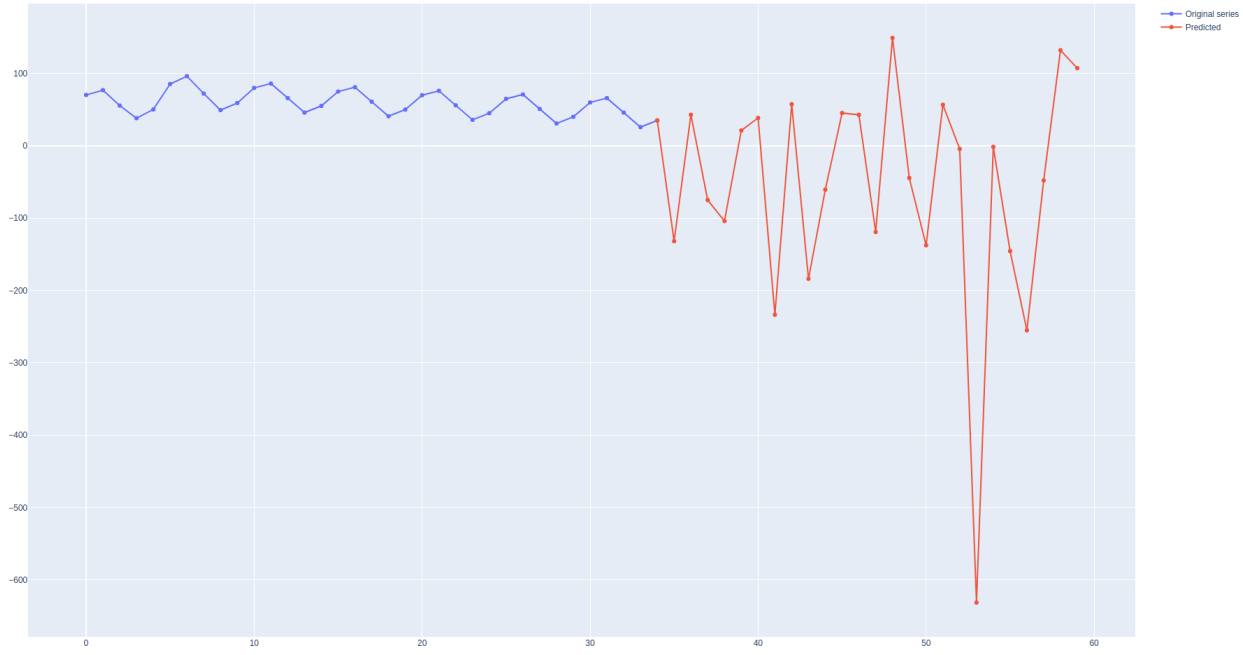
Multiplicative:

$$Overall\ Equation : \hat{y}_{t+h} = (l_t + hb_t)s_{t+h-m}$$

$$Level\ Equation : l_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1})$$

$$Trend\ Equation : b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

$$Seaonality\ Equation : s_t = \gamma\frac{y_t}{l_{t-1} + b_{t-1}} + (1-\gamma)s_{t-m}$$



Part 3 Hyperparameter tuning

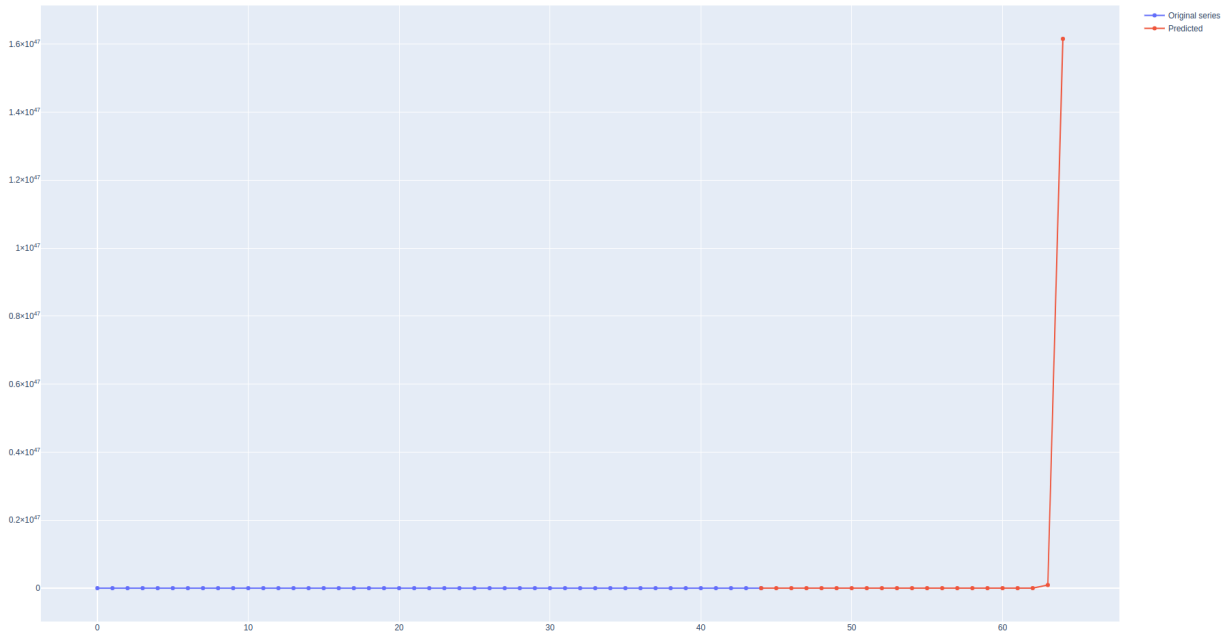
- For hyperparameter tuning we first append all list in single list
- Then with the help of forecasting and parameters function in forecasting.py tune the parameters.
- Use forecasting function as part 1 and part 2
- In hyperparameter functions we used third party package statsmodels
- From statsmodels we use `statsmodels.api.tsa.arima.ARIMA for ARIMA and statsmodels.tsa.holtwinters.ExponentialSmoothing for Holt Winter`
- Take different parameters for ARIMA and Holt Winter and tune them and return

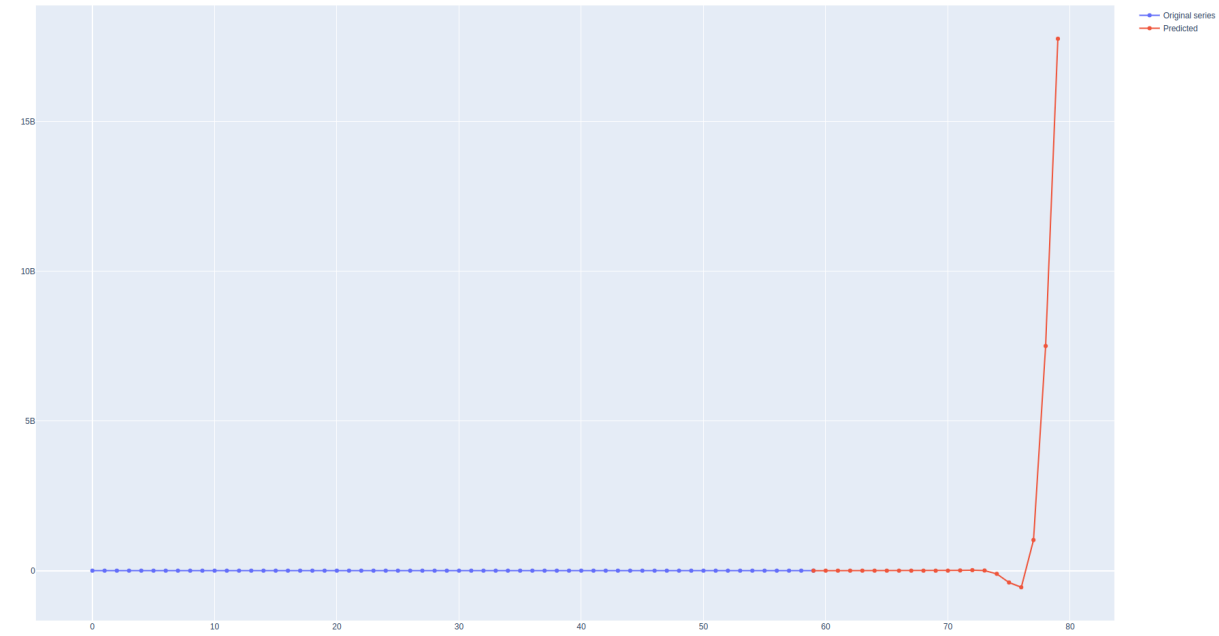
tuple of best hyperparameter

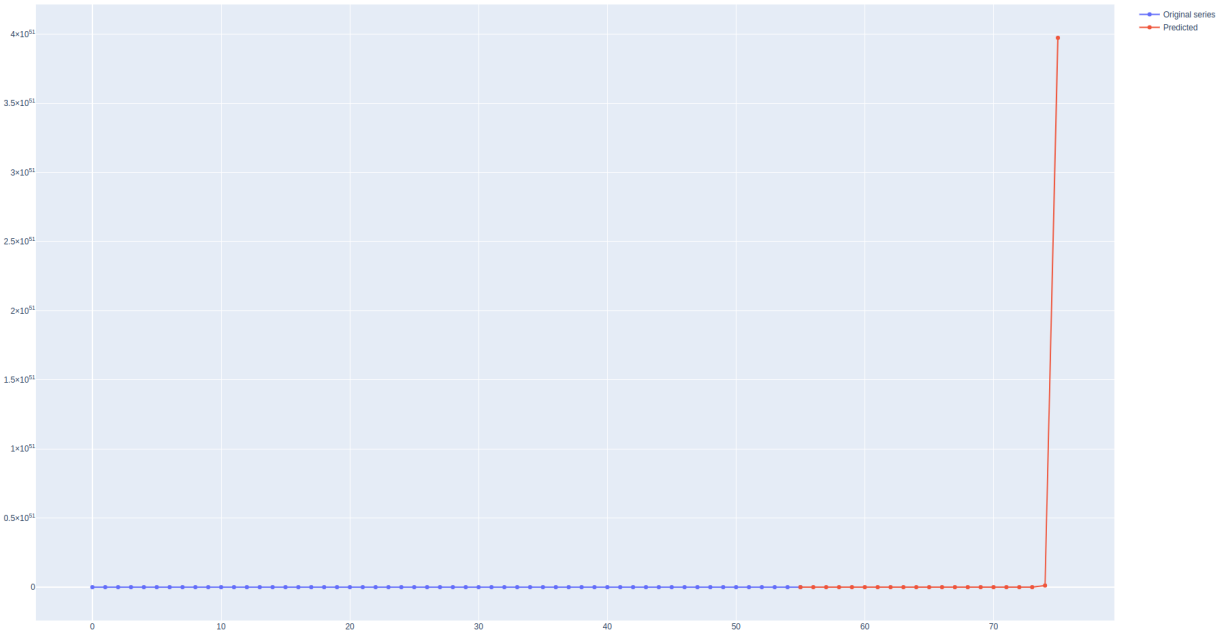
- Hyperparameter is decided by the least mean square error.
- For mean square error used sklearn
- Parameter take in ARIMA – P = 2, D = [0,1], Q = [0, 1]
- Parameter take in Holt Winter: alpha = [0.3, 0.7, 0.5], beta = [0.2, 0.6, 0.8], gamma = [0.25, 0.35, 0.65], seasonality = [2, 3, 5]

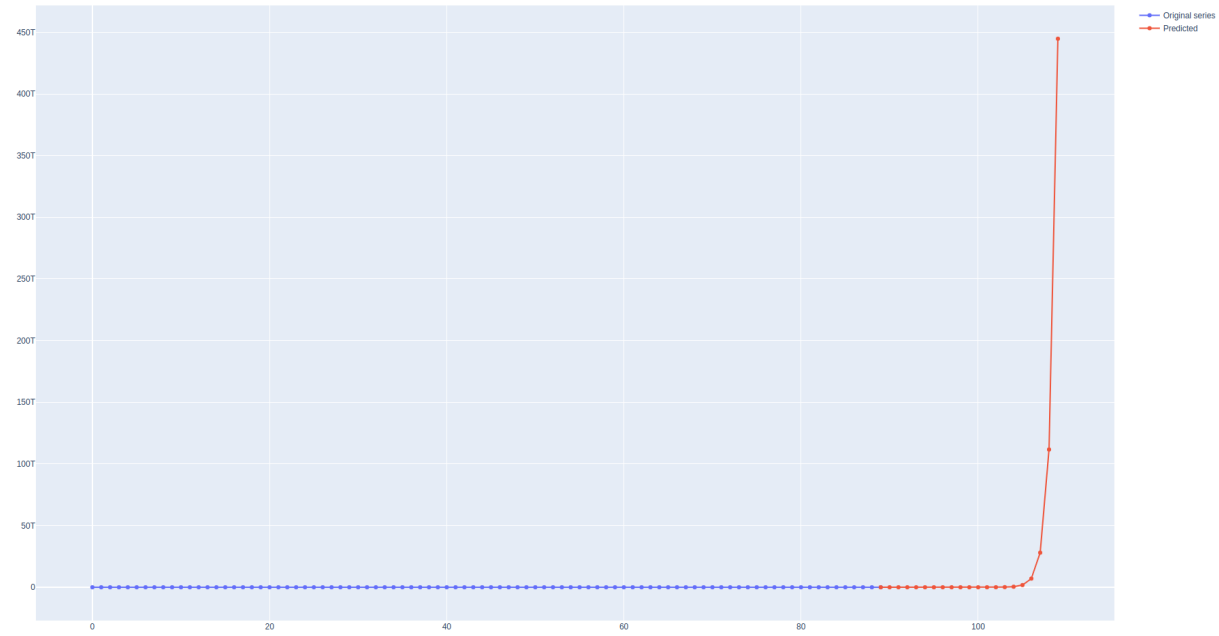
ARIMA Hyperparameters tuning output:

```
ARIMA (2, 1, 1) MSE=1797.601
ARIMA (2, 0, 1) MSE=53.430
ARIMA (2, 0, 1) MSE=108.959
ARIMA (2, 1, 0) MSE=2717.159
/home/jash/miniconda3/lib/python3.9/site-packages

Maximum Likelihood optimization failed to converg

ARIMA (2, 0, 1) MSE=34075.195
```




Original series
Predicted
15B
10B
5B
0
0
10
20
30
40
50
60
70
80

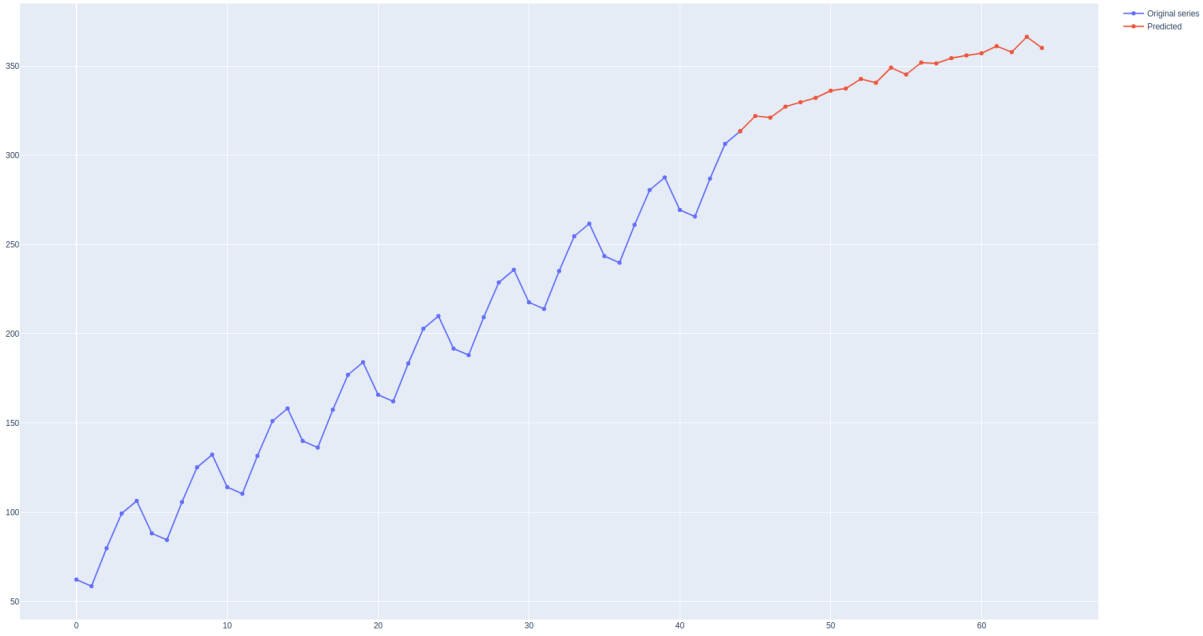

Original series
Predicted
4×10^51
3.5×10^51
3×10^51
2.5×10^51
2×10^51
1.5×10^51
1×10^51
0.5×10^51
0
0
10
20
30
40
50
60
70

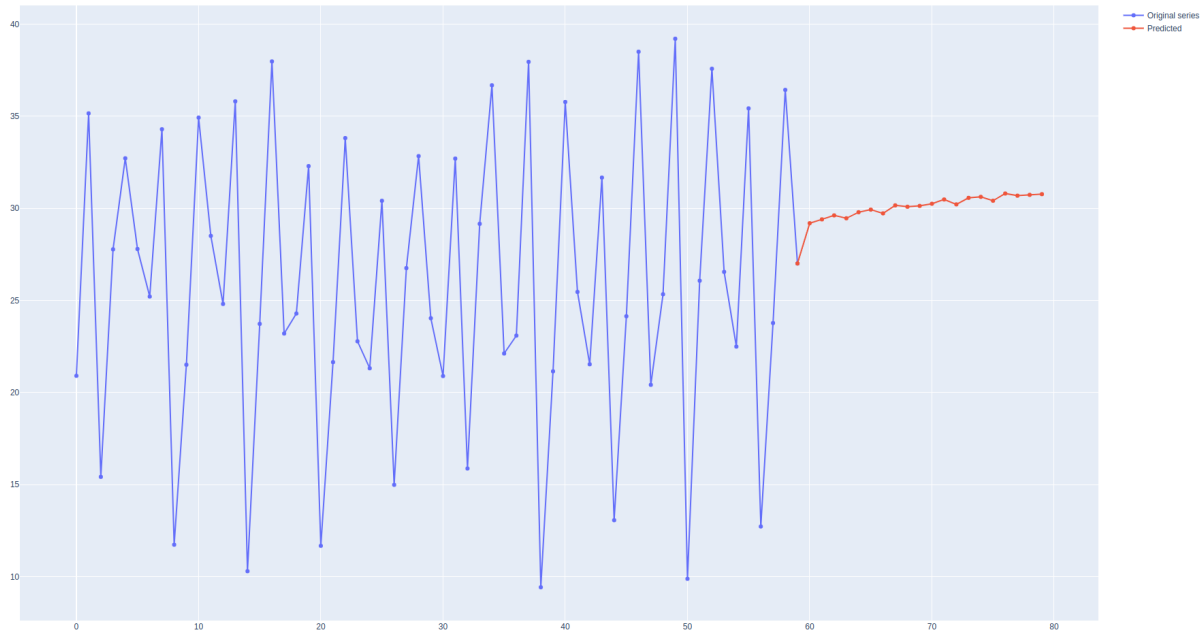

Original series
Predicted
450T
400T
350T
300T
250T
200T
150T
100T
50T
0
0
20
40
60
80
100

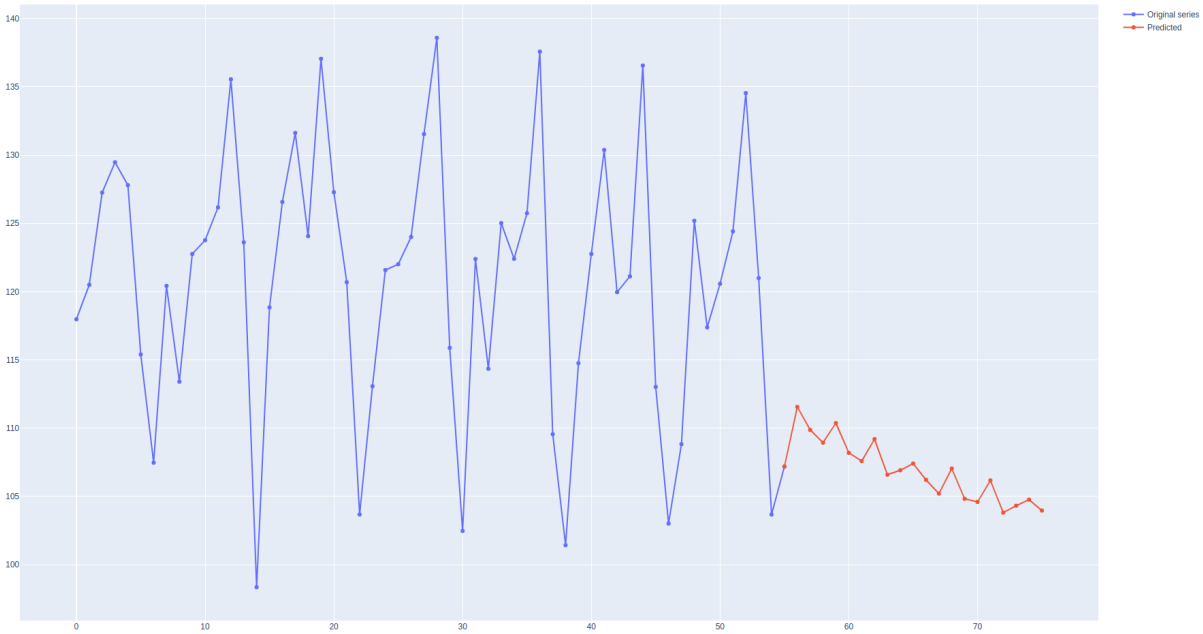

Original series
Predicted
8×10^31
7×10^31
6×10^31
5×10^31
4×10^31
3×10^31
2×10^31
1×10^31
0
0
10
20
30
40
50
60
70

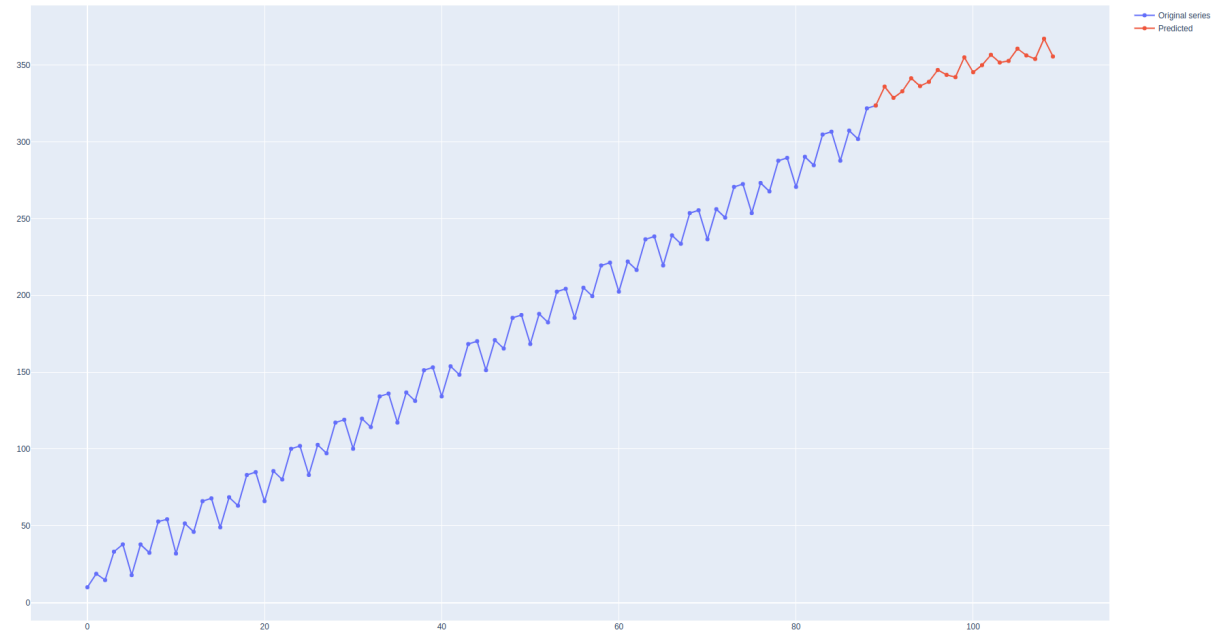
Holt Winters' Hyperparameter tuning output:

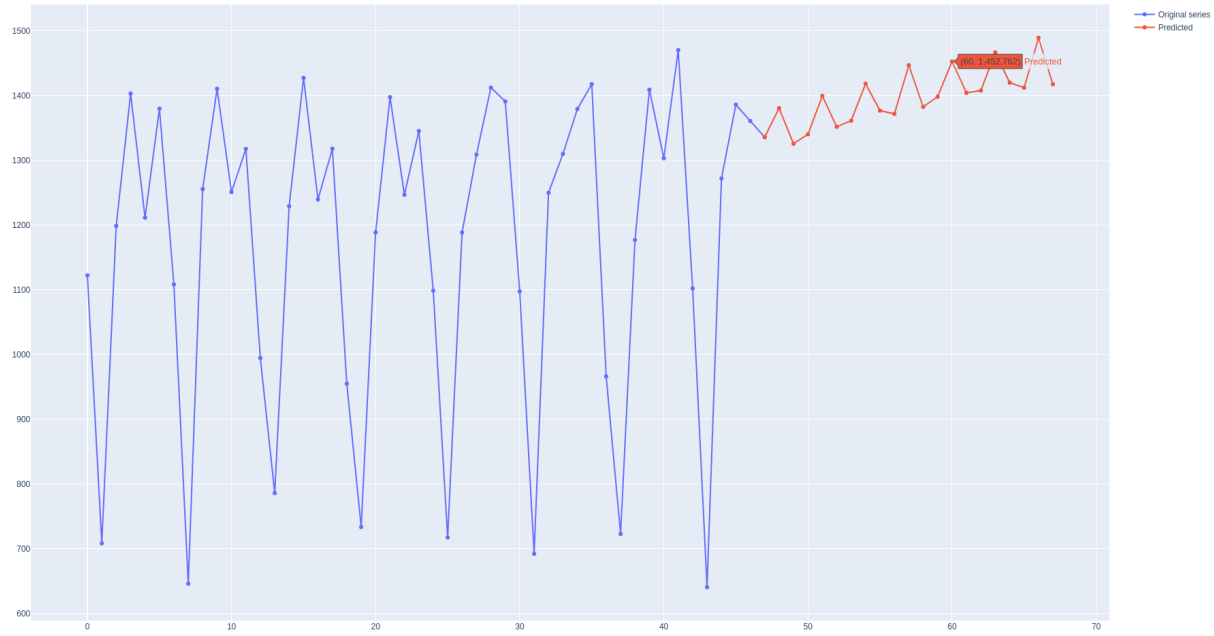
```
HoltWinter (0.7, 0.2, 0.25, 2) MSE=2345.210
HoltWinter (0.3, 0.2, 0.25, 2) MSE=94.194
HoltWinter (0.3, 0.2, 0.25, 2) MSE=109.095
HoltWinter (0.7, 0.2, 0.25, 2) MSE=2521.950
HoltWinter (0.3, 0.2, 0.25, 2) MSE=80485.623
(base) jash@jash-Iaptop:~/WorkSpace/MTech/sem2/AA
/home/jash/miniconda3/lib/python3.9/site-packages
```




Original series
Predicted
40
35
30
25
20
15
10
0
10
20
30
40
50
60
70
80


Original series
Predicted
140
135
130
125
120
115
110
105
100
0
10
20
30
40
50
60
70


Original series
Predicted
0
50
100
150
200
250
300
350
0
20
40
60
80
100


Original series
Predicted
(60, 1,452.762)
Predicted
600
700
800
900
1000
1100
1200
1300
1400
1500
0
10
20
30
40
50
60
70

Code that need to end of the tests.py file in main function

```
    S = []
    S.append(S1)
    S.append(S2)
    S.append(S3)
    S.append(S4)
    S.append(S5)

    for i in range(0,5):
        P, D, Q = forecasting.ARIMA_Paramters(S[i])
        ARIMA_S = forecasting.ARIMA_Forecast(S[i], P, D, Q, 20)
        Plot(S[i], ARIMA_S)

    for i in range(0,5):
        Alpha, Beta, Gamma, Seasonality = forecasting.HoltWinter_Parameters(S[i])
            HoltWinters_S = forecasting.HoltWinter_Forecast(S[i], Alpha, Beta, Gamma,
Seasonality, 20)
        Plot(S[i], HoltWinters_S)
```