TECHNICAL ANALYSIS

SIMPLE MOVING AVERAGE:-

A **simple moving average** (SMA) is an arithmetic **moving average** calculated by adding the closing price of the security for a number of time periods and then dividing this total by the number of time periods.

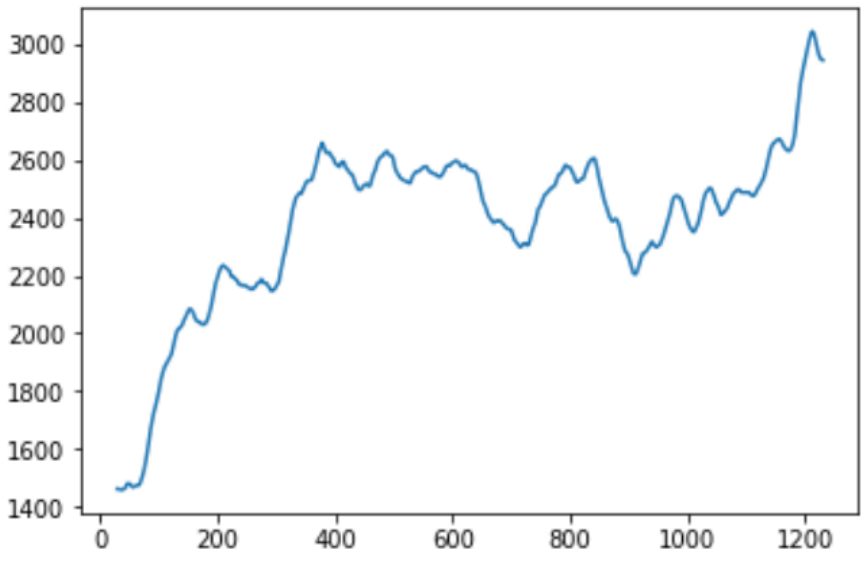
Formula:-

$$SMA = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n} = \frac{1}{n}\sum_{i=0}^{n-1} p_{M-i}$$

Code:-

```
#technical indicators

#Moving Average -20 days,-30 days
def MA(df, n):
    MA = pd.Series(pd.rolling_mean(df['Close'], n), name = 'MA_' + str(n))
    df = df.join(MA)
    return df


```

Graph:-



EXPONENTIAL MOVING AVERAGE:-

An **exponential moving average** (EMA) is a type of **moving average** that is similar to a simple **moving average**, except that more weight is given to the latest data. It's also known as the **exponentially weighted moving average**. This type of **moving average** reacts faster to recent price changes than a simple **moving average**.

Formula:-

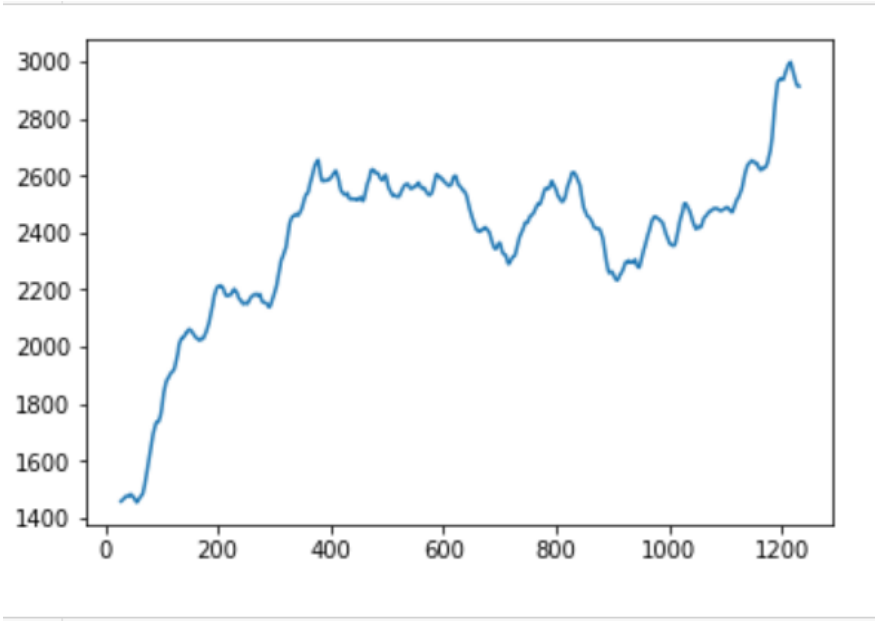
$$EMA = \frac{p_1 + (1-\alpha)p_2 + (1-\alpha)^2 p_3 + (1-\alpha)^3 p_4 + \cdots}{1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \cdots}$$

$$\text{Where} \quad \alpha = \frac{2}{N+1}$$

Code:-

```
1
2  #Exponential Moving Average
3  def EMA(df, n):
4      EMA = pd.Series(pd.ewma(df['Close'], span = n, min_periods = n - 1), name = 'EMA_' + str(n))
5      df = df.join(EMA)
6      return df
```

Graph:-



Stochastic oscillator %K :-

The stochastic oscillator is a momentum indicator comparing the **closing price** of a security to the range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a **moving average** of the result.

Formula:-

BREAKING DOWN 'Stochastic Oscillator'

The stochastic oscillator is calculated using the following formula:

%K = 100(C - L14)/(H14 - L14)

Where:

C = the most recent closing price

L14 = the low of the 14 previous trading sessions

H14 = the highest price traded during the same 14-day period

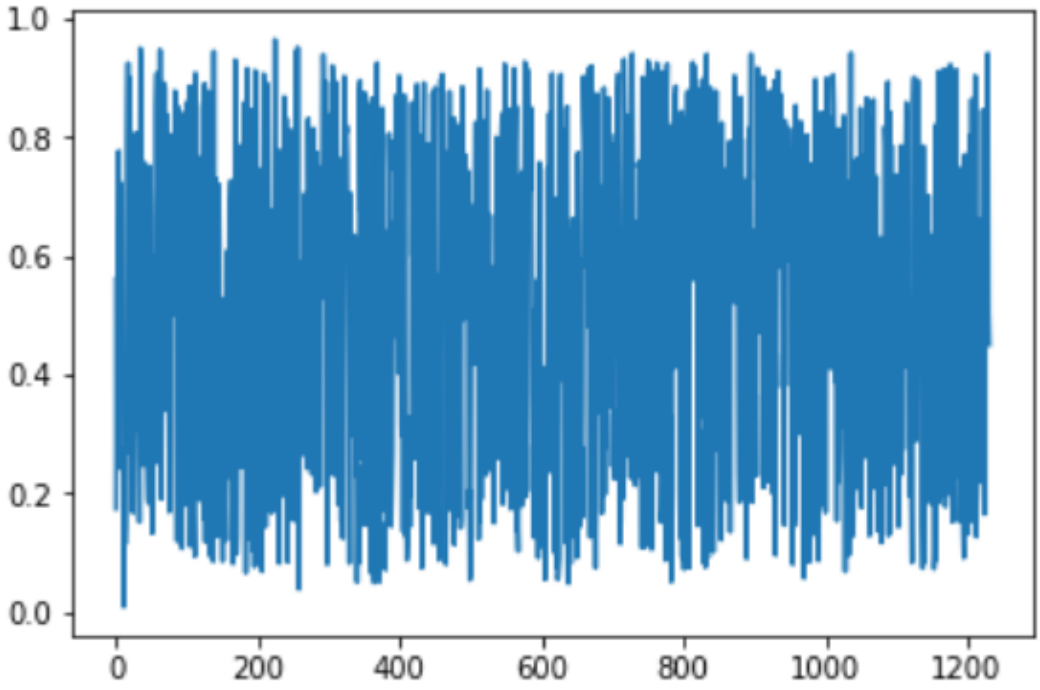
%K= the current market rate for the currency pair

%D = 3-period moving average of %K

Code:-

```
#Stochastic oscillator %K
def STOK(df):
    SOk = pd.Series((df['Close'] - df['Low']) / (df['High'] - df['Low']), name = 'SO%k')
    df = df.join(SOk)
    return df
```

Graph:-



Average Directional Movement Index:-

Average Directional Movement Index or ADX is used to quantify trend strength. ADX calculations are based on a **moving average** of price range expansion over a given period of time. The default setting is 14 bars, although other time periods can be used. ADX can be used on any trading vehicle such as stocks, mutual funds, **exchange-traded funds** and futures.

Formula:-

ADX FORMULA

$$ADX_{(t)} = \frac{(ADX_{(t-1)} \times (n-1)) + DX_{(t)})}{n}$$

where

n = smoothing period

$$DX_{(t)} = \frac{100 \times (PDI_{(t)} - MDI_{(t)})}{PDI_{(t)} + MDI_{(t)}}$$

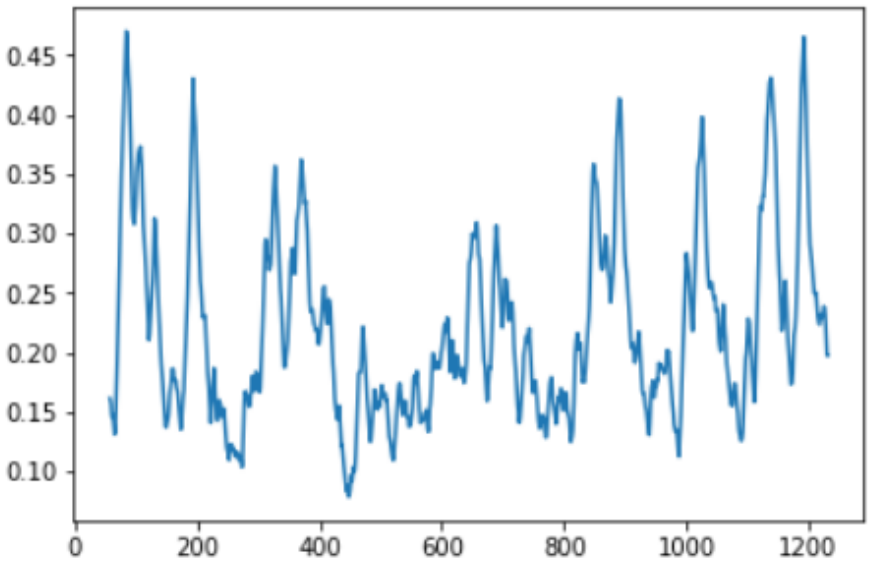
$PDI_{(t)}$ = Positive Directional Movement = $MAX(H_{(t)} - H_{(t-1)}, 0)$

$MDI_{(t)}$ = Negative Directional Movement = $MAX(L_{(t)} - L_{(t-1)}, 0)$

CODE:-

```
#Average Directional Movement Index
def ADX(df, n, n_ADX):
    i = 0
    UpI = []
    DoI = []
    while i + 1 <= df.index[-1]:
        UpMove = df.get_value(i + 1, 'High') - df.get_value(i, 'High')
        DoMove = df.get_value(i, 'Low') - df.get_value(i + 1, 'Low')
        if UpMove > DoMove and UpMove > 0:
            UpD = UpMove
        else: UpD = 0
        UpI.append(UpD)
        if DoMove > UpMove and DoMove > 0:
            DoD = DoMove
        else: DoD = 0
        DoI.append(DoD)
        i = i + 1
    i = 0
    TR_l = [0]
    while i < df.index[-1]:
        TR = max(df.get_value(i + 1, 'High'), df.get_value(i, 'Close')) - min(df.get_value(i + 1, 'Low'), df.get_
        TR_l.append(TR)
        i = i + 1
    TR_s = pd.Series(TR_l)
    ATR = pd.Series(pd.ewma(TR_s, span = n, min_periods = n))
    UpI = pd.Series(UpI)
    DoI = pd.Series(DoI)
    PosDI = pd.Series(pd.ewma(UpI, span = n, min_periods = n - 1) / ATR)
    NegDI = pd.Series(pd.ewma(DoI, span = n, min_periods = n - 1) / ATR)
    ADX = pd.Series(pd.ewma(abs(PosDI - NegDI) / (PosDI + NegDI), span = n_ADX, min_periods = n_ADX - 1), name =
    df = df.join(ADX)
    return df
```

Graph:-



Chaikin Oscillator:-

The **Chaikin Oscillator** is the difference between the **3-day EMA** of the Accumulation Distribution Line and the **10-day EMA** of the Accumulation Distribution Line. Like other momentum indicators, this indicator is designed to anticipate **directional changes** in the Accumulation Distribution Line by measuring the momentum behind the movements. A momentum change is the first step to a trend change.

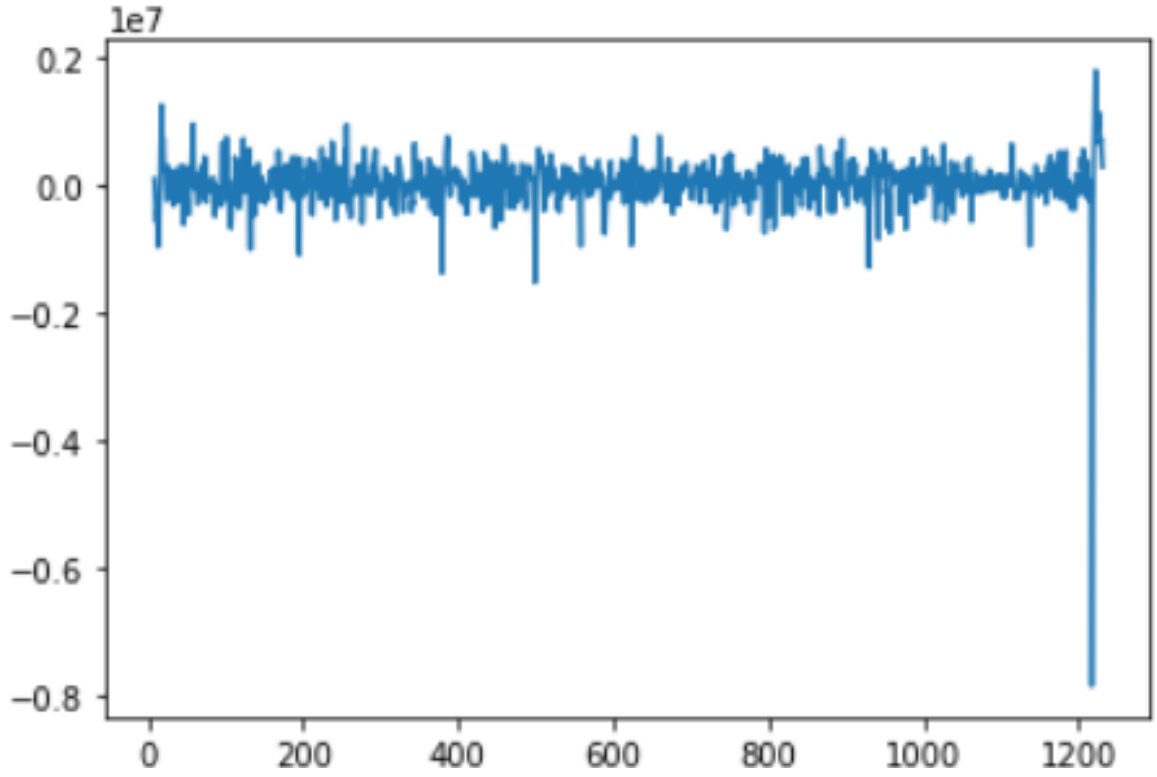
Formula:-

```
There are four steps in calculating The Chaikin Oscillator (This example is for a (3,10) Period):

1. Find the Money Flow Multiplier
   [(Close  -  Low) - (High - Close)] /(High - Low) = Money Flow Multiplier

2. Calculate Money Flow Volume
   Money Flow Multiplier x Volume for the Period = Money Flow Volume

3. Determine ADL
Previous ADL + Current Period Money Flow Volume = ADL

4. Apply EMA (user defined periods) to the ADL to generate the Chaikin Oscillator
  (3-day EMA of ADL)  -  (10-day EMA of ADL) = Chaikin Oscillator
```

Code:-

```
#Chaikin Oscillator
def Chaikin(df):
    ad = (2 * df['Close'] - df['High'] - df['Low']) / (df['High'] - df['Low']) * df['Volume']
    Chaikin = pd.Series(pd.ewma(ad, span = 3, min_periods = 2) - pd.ewma(ad, span = 10, min_periods = 9), name
    df = df.join(Chaikin)
    return df
```

Graph:-



Commodity Channel Index:-

Commodity Channel Index (CCI) is a versatile indicator that can be used to identify a new trend or warn of extreme conditions.CCI measures the current price level relative to an **average price level** over a given period of time. CCI is relatively high when prices are far above their average. CCI is relatively low when prices are far below their average. In this manner, CCI can be used to identify **overbought** and **oversold levels.**

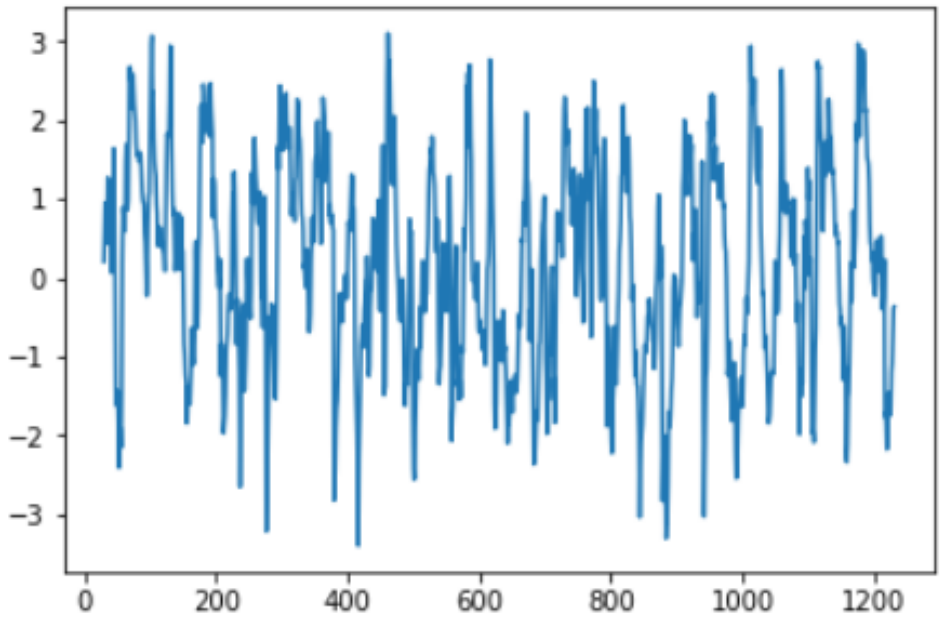
Formula:-

```
There are several steps involved in calculating the Commodity Channel Index.
The following example is for a typical 20 Period CCI:

CCI = (Typical Price  -  20 Period SMA of TP) / (.015 x Mean Deviation)

Typical Price (TP) = (High + Low + Close)/3

Constant = .015
The Constant is set at .015 for scaling purposes.
By including the constant, the majority of CCI values will fall within the 100 to -100 range.

There are three steps to calculating the Mean Deviation.
1. Subtract the most recent 20 Period Simple Moving from each typical price (TP) for the Period.
2. Sum these numbers strictly using absolute values.
3. Divide the value generated in step 3 by the total number of Periods (20 in this case).
```

Code:-

```
#Commodity Channel Index
def CCI(df, n):
    PP = (df['High'] + df['Low'] + df['Close']) / 3
    CCI = pd.Series((PP - pd.rolling_mean(PP, n)) / pd.rolling_std(PP, n), name = 'CCI_' + str(n))
    df = df.join(CCI)
    return df
```

Graph:-



Momentum:-

The **Momentum indicator** is a speed of movement **indicator** designed to identify the speed (or strength) of price movement. The **momentum indicator** compares the most recent closing price to a previous closing price (can be the closing price of any time frame).

Formula:-

- **Calculation:**

  M = CP - CPn

  Or

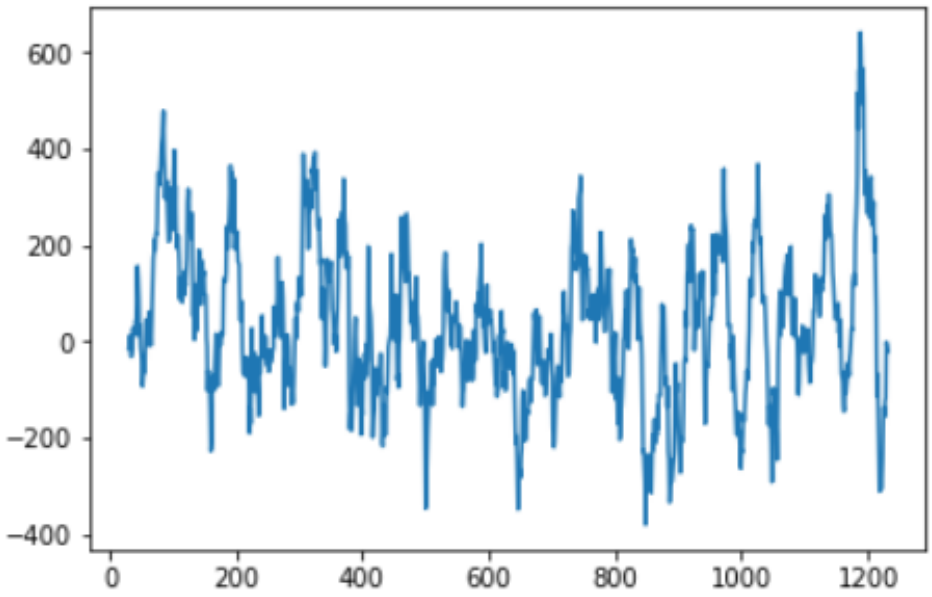
  M = (CP / CPn) * 100

The first calculation just takes the difference between the two closing prices and plots it. The second version of the indicator shows the price difference between the current price and the price n periods ago as a percentage.

Code:-

```
#Momentum
def MOM(df, n):
    M = pd.Series(df['Close'].diff(n), name = 'Momentum_' + str(n))
    df = df.join(M)
    return df

```

Graph:-



Rate of Change:-

The **Rate-of-Change** (ROC) **indicator** is a pure momentum oscillator that measures the percent **change** in price from one period to the next. The ROC calculation compares the current price with the price “n” periods ago.

Formula:-

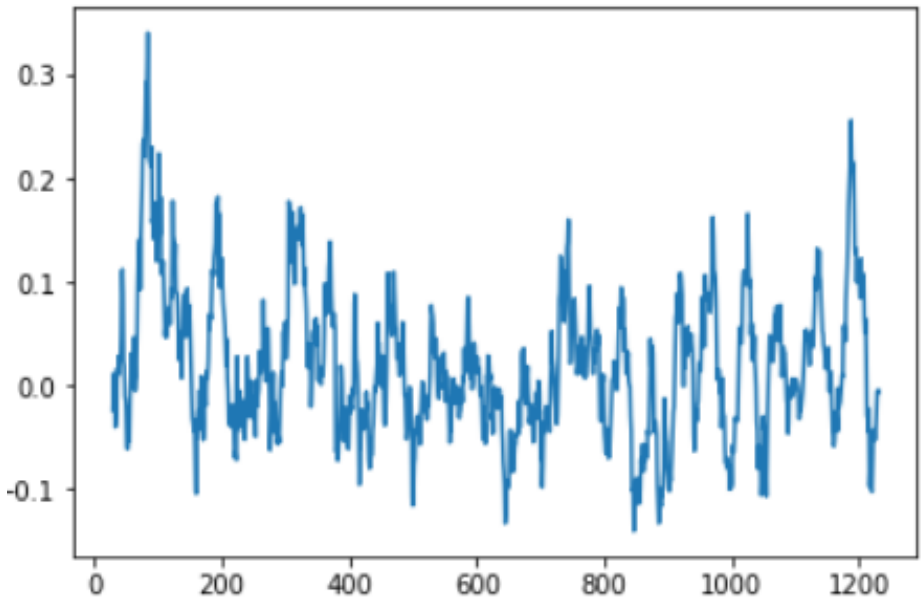
Calculation

```
ROC = [(Close - Close n periods ago) / (Close n periods ago)] * 100
```

Code:-

```
#Rate of Change
def ROC(df, n):
    M = df['Close'].diff(n - 1)
    N = df['Close'].shift(n - 1)
    ROC = pd.Series(M / N, name = 'ROC_' + str(n))
    df = df.join(ROC)
    return df
```

Graph:-



Average True Range :-

The average true range (ATR) is a measure of volatility. The true range indicator is the greatest of the following: current high less the current low, the **absolute value** of the current high less the **previous close** and the absolute value of the current low less the previous close. The average true range is a **moving average**, generally 14 days, of **the true ranges.**

Formula:-

```
Current ATR = [(Prior ATR x 13) + Current TR] / 14

  - Multiply the previous 14-day ATR by 13.
  - Add the most recent day's TR value.
  - Divide the total by 14
```

Code:-

```
#Average True Range
def ATR(df, n):
    i = 0
    TR_l = [0]
    while i < df.index[-1]:
        TR = max(df.get_value(i + 1, 'High'), df.get_value(i, 'Close')) - min(df.get_value(i + 1, 'Low'), df.get_
        TR_l.append(TR)
        i = i + 1
    TR_s = pd.Series(TR_l)
    ATR = pd.Series(pd.ewma(TR_s, span = n, min_periods = n), name = 'ATR_' + str(n))
    df = df.join(ATR)
    return df
```

Graph:-