```
import numpy as np
import time
from binance import Client, ThreadedWebsocketManager, ThreadedDepthCacheManager
import pandas as pd
pd.set_option('display.max_rows', 3000)
pd.set_option('display.max_columns', 3000)
pd.set_option('display.width', 1000)
import mplfinance as mpf

def EMA(list_prices, period):
    # Calculating the SMA for the EMA first value

    SMA_period_1 = []

    x = 0
    while x < period:
        SMA_period_1.append( list_prices[x] )
        x += 1

    SMAs = sum( SMA_period_1 ) / period

    # Calculating the EMA for the same period

    EMAs = []
    EMAs.append( SMAs )

    x = period
    while x < len( list_prices ):
        EMAs_prices = (list_prices[x] - EMAs[x - period]) * (2 / (period + 1)) +
EMAs[x - period]
        EMAs.append( EMAs_prices )
        x += 1

    return EMAs

def TR (th, tl, yc):

    TR = []
    u = 0

    while u < len (th)-1:

        if abs( th[u + 1] - yc[u] ) > abs( tl[u + 1] - yc[u] ) and abs( th[u + 1] -
yc[u] ) > (
                th[u + 1] - tl[u + 1]) and not abs( th[u + 1] - yc[u] ) == abs(
tl[u + 1] - yc[u] ) and not abs(
                th[u + 1] - yc[u] ) == (th[u + 1] - tl[u + 1]) and not abs( tl[u +
1] - yc[u] ) == (
                th[u + 1] - tl[u + 1]):
            TR.append( abs( th[u + 1] - yc[u] ) )
        if abs( tl[u + 1] - yc[u] ) > (th[u + 1] - yc[u]) and abs( tl[u + 1] -
yc[u] ) > (
                th[u + 1] - tl[u + 1]) and not abs( th[u + 1] - yc[u] ) == abs(
tl[u + 1] - yc[u] ) and not abs(
                th[u + 1] - yc[u] ) == (th[u + 1] - tl[u + 1]) and not abs( tl[u +
1] - yc[u] ) == (
                th[u + 1] - tl[u + 1]):
            TR.append( abs( tl[u + 1] - yc[u] ) )

        if (th[u + 1] - tl[u + 1]) > (th[u + 1] - yc[u]) and abs( th[u + 1] - tl[u
+ 1] ) > abs(
                tl[u + 1] - yc[u] ) and not abs( th[u + 1] - yc[u] ) == abs( tl[u +
1] - yc[u] ) and not abs(
                th[u + 1] - yc[u] ) == (th[u + 1] - tl[u + 1]) and not abs( tl[u +
1] - yc[u] ) == (
                th[u + 1] - tl[u + 1]):
            TR.append( th[u + 1] - tl[u + 1] )

        if abs( th[u + 1] - yc[u] ) == abs( tl[u + 1] - yc[u] ):
            TR.append( abs( th[u + 1] - yc[u] ) )

        if abs( th[u + 1] - yc[u] ) == (th[u + 1] - tl[u + 1]):
            TR.append( th[u + 1] - tl[u + 1] )

        if abs( tl[u + 1] - yc[u] ) == (th[u + 1] - tl[u + 1]):
            TR.append( abs( tl[u + 1] - yc[u] ) )

        u+=1

    return TR

def ATR (list_prices, period):

    ATR_list = []
    ATR_list.append (list_prices [0])

    w = 1
    while w < len (list_prices):
        ATR_prices = (ATR_list [w-1] * (period-1) + (list_prices [w]))/ period
        ATR_list.append(ATR_prices)
        w += 1

    return ATR_list

def PDM (highp_list, lowp_list):

    highp_values = []

    for i in highp_list:
        highp_values.append (i)

    lowp_values = []
    for i in lowp_list:
        lowp_values.append (i)

    PDM_values = []

    q = 1
    while q < len (highp_values):

        if (lowp_values[q] == lowp_values[q - 1]) and (highp_values[q] >
highp_values[q - 1]) and (
                highp_values[q - 1] > lowp_values[q]):
            PDM_daily_value_1 = highp_values[q] - highp_values[q - 1]
            PDM_values.append( PDM_daily_value_1 )
            print( q, "1" )

        if (lowp_values[q] < lowp_values[q - 1]) and (highp_values[q] >
highp_values[q - 1]) and (
                highp_values[q] - highp_values[q - 1]) > (lowp_values[q - 1] -
lowp_values[q]):
            PDM_daily_value_2 = highp_values[q] - highp_values[q - 1]
            PDM_values.append( PDM_daily_value_2 )
            print( q, "2" )

        if (lowp_values[q] < lowp_values[q - 1]) and (highp_values[q] >
highp_values[q - 1]) and (
                highp_values[q] - highp_values[q - 1]) == (lowp_values[q - 1] -
lowp_values[q]):
            PDM_daily_value_2_1 = 0
            PDM_values.append( PDM_daily_value_2_1 )
            print( q, "2_1" )

        if (lowp_values[q] > lowp_values[q - 1]) and (highp_values[q - 1] ==
lowp_values[q]) and (
                highp_values[q] > highp_values[q - 1]):
            PDM_daily_value_3 = highp_values[q] - highp_values[q - 1]
            PDM_values.append( PDM_daily_value_3 )
            print( q, "3" )

        if (lowp_values[q - 1] < lowp_values[q]) and (highp_values[q - 1] >
lowp_values[q]) and (
                highp_values[q] > highp_values[q - 1]):
            PDM_daily_value_4 = highp_values[q] - highp_values[q - 1]
            PDM_values.append( PDM_daily_value_4 )
            print( q, "4" )

        if (lowp_values[q - 1] < lowp_values[q]) and (highp_values[q] <
highp_values[q - 1]):
```

```
            PDM_daily_value_5 = 0
            PDM_values.append( PDM_daily_value_5 )
            print( q, "5" )

        if (lowp_values[q - 1] < lowp_values[q]) and (highp_values[q] ==
highp_values[q - 1]):
            PDM_daily_value_5_1 = 0
            PDM_values.append( PDM_daily_value_5_1 )
            print( q, "5_1" )

        if (lowp_values[q - 1] == lowp_values[q]) and (highp_values[q] ==
highp_values[q - 1]):
            PDM_daily_value_6 = 0
            PDM_values.append( PDM_daily_value_6 )
            print( q, "6" )

        if (highp_values[q - 1] == highp_values[q]) and (lowp_values[q - 1] >
lowp_values[q]):
            PDM_daily_value_7 = 0
            PDM_values.append( PDM_daily_value_7 )
            print( q, "7" )

        if (lowp_values[q - 1] > lowp_values[q]) and (highp_values[q - 1] >
lowp_values[q]) and (
                highp_values[q] > highp_values[q - 1]) and (
                lowp_values[q - 1] - lowp_values[q] > highp_values[q] -
highp_values[q - 1]):
            PDM_daily_value_8 = 0
            PDM_values.append( PDM_daily_value_8 )
            print( q, "8" )

        if (lowp_values[q - 1] > lowp_values[q]) and (highp_values[q - 1] >
lowp_values[q]) and (
                highp_values[q] > highp_values[q - 1]) and (
                lowp_values[q - 1] - lowp_values[q] == highp_values[q] -
highp_values[q - 1]):
            PDM_daily_value_8_1 = 0
            PDM_values.append( PDM_daily_value_8_1 )
            print( q, "8_1" )

        if (highp_values[q - 1] > highp_values[q]) and (lowp_values[q - 1] ==
highp_values[q]) and (
                lowp_values[q] < lowp_values[q - 1]):
            PDM_daily_value_9 = 0
            PDM_values.append( PDM_daily_value_9 )
            print( q, "9" )

        if (highp_values[q - 1] > highp_values[q]) and (highp_values[q] >
lowp_values[q - 1]) and (
                lowp_values[q] < lowp_values[q - 1]):
```

```
            PDM_daily_value_10 = 0
            PDM_values.append( PDM_daily_value_10 )
            print( q, "10" )

        if (lowp_values[q - 1] == lowp_values[q]) and (highp_values[q - 1] >
highp_values[q]):
            PDM_daily_value_11 = 0
            PDM_values.append( PDM_daily_value_11 )
            print( q, "11" )

        if (lowp_values[q - 1] == lowp_values[q] == highp_values[q - 1] ==
highp_values[q]):
            PDM_daily_value_12 = 0
            PDM_values.append( PDM_daily_value_12 )
            print( q, "12" )

        if (lowp_values[q - 1] < lowp_values[q] and highp_values[q - 1] <
lowp_values[q] and highp_values[q] >
                lowp_values[q]):
            PDM_daily_value_13 = highp_values[q] - highp_values[q - 1]
            PDM_values.append( PDM_daily_value_13 )

        if highp_values[q - 1] > highp_values[q] and lowp_values[q - 1] >
highp_values[q]:
            PDM_daily_value_14 = 0
            PDM_values.append( PDM_daily_value_14 )

        q+=1

    return PDM_values

def NDM (lowp_list, highp_list):

    highp_values = []

    for i in highp_list:
        highp_values.append (i)

    lowp_values = []
    for i in lowp_list:
        lowp_values.append (i)

    NDM_values = []

    w = 1
    while w < len (lowp_values):

        if (highp_values[w - 1] == highp_values[w]) and (lowp_values[w - 1] >
lowp_values[w]):
            NDM_daily_value_1 = lowp_values[w - 1] - lowp_values[w]
            NDM_values.append( NDM_daily_value_1 )
            print( w, "1" )

        if (lowp_values[w - 1] > lowp_values[w]) and (highp_values[w - 1] >
lowp_values[w]) and (
                highp_values[w] > highp_values[w - 1]) and (
                lowp_values[w - 1] - lowp_values[w] > highp_values[w] -
highp_values[w - 1]):
            NDM_daily_value_2 = lowp_values[w - 1] - lowp_values[w]
            NDM_values.append( NDM_daily_value_2 )
            print( w, "2" )

        if (lowp_values[w - 1] > lowp_values[w]) and (highp_values[w - 1] >
lowp_values[w]) and (
                highp_values[w] > highp_values[w - 1]) and (
                lowp_values[w - 1] - lowp_values[w] == highp_values[w] -
highp_values[w - 1]):
            NDM_daily_value_2_1 = 0
            NDM_values.append( NDM_daily_value_2_1 )
            print( w, "2_1" )

        if (highp_values[w - 1] > highp_values[w]) and (lowp_values[w - 1] ==
highp_values[w]) and (
                lowp_values[w] < lowp_values[w - 1]):
            NDM_daily_value_3 = lowp_values[w - 1] - lowp_values[w]
            NDM_values.append( NDM_daily_value_3 )
            print( w, "3" )

        if (highp_values[w - 1] > highp_values[w]) and (highp_values[w] >
lowp_values[w - 1]) and (
                lowp_values[w] < lowp_values[w - 1]):
            NDM_daily_value_4 = lowp_values[w - 1] - lowp_values[w]
            NDM_values.append( NDM_daily_value_4 )
            print( w, "4" )

        if (lowp_values[w - 1] < lowp_values[w]) and (highp_values[w] <
highp_values[w - 1]):
            NDM_daily_value_5 = 0
            NDM_values.append( NDM_daily_value_5 )
            print( w, "5" )

        if (lowp_values[w - 1] == lowp_values[w]) and (highp_values[w] ==
highp_values[w - 1]):
            NDM_daily_value_6 = 0
            NDM_values.append( NDM_daily_value_6 )
            print( w, "6" )

        if (lowp_values[w] == lowp_values[w - 1]) and (highp_values[w] >
highp_values[w - 1]) and (
                highp_values[w - 1] > lowp_values[w]):
```

```
            NDM_daily_value_7 = 0
            NDM_values.append( NDM_daily_value_7 )
            print( w, "7" )

        if (lowp_values[w] < lowp_values[w - 1]) and (highp_values[w] >
highp_values[w - 1]) and (
                highp_values[w] - highp_values[w - 1]) > (lowp_values[w - 1] -
lowp_values[w]):
            NDM_daily_value_8 = 0
            NDM_values.append( NDM_daily_value_8 )
            print( w, "8" )

        if (lowp_values[w] < lowp_values[w - 1]) and (highp_values[w] >
highp_values[w - 1]) and (
                highp_values[w] - highp_values[w - 1]) == (lowp_values[w - 1] -
lowp_values[w]):
            NDM_daily_value_8_1 = 0
            NDM_values.append( NDM_daily_value_8_1 )
            print( w, "8_2" )

        if (lowp_values[w] > lowp_values[w - 1]) and (highp_values[w - 1] ==
lowp_values[w]) and (
                highp_values[w] > highp_values[w - 1]):
            NDM_daily_value_9 = 0
            NDM_values.append( NDM_daily_value_9 )
            print( w, "9" )

        if (lowp_values[w - 1] < lowp_values[w]) and (highp_values[w - 1] >
lowp_values[w]) and (
                highp_values[w] > highp_values[w - 1]):
            NDM_daily_value_10 = 0
            NDM_values.append( NDM_daily_value_10 )
            print( w, "10" )

        if (lowp_values[w - 1] < lowp_values[w]) and (highp_values[w] ==
highp_values[w - 1]):
            NDM_daily_value_10_1 = 0
            NDM_values.append( NDM_daily_value_10_1 )
            print( w, "10_1" )

        if (lowp_values[w - 1] == lowp_values[w]) and (highp_values[w - 1] >
highp_values[w]):
            NDM_daily_value_11 = 0
            NDM_values.append( NDM_daily_value_11 )
            print( w, "11" )

        if (lowp_values[w - 1] == lowp_values[w] == highp_values[w - 1] ==
highp_values[w]):
            NDM_daily_value_12 = 0
            NDM_values.append( NDM_daily_value_12 )
            print( w, "12" )

        if (lowp_values[w - 1] < lowp_values[w] and highp_values[w - 1] <
lowp_values[w] and highp_values[w] >
                lowp_values[w]):
            NDM_daily_value_13 = 0
            NDM_values.append( NDM_daily_value_13 )

        if highp_values[w - 1] > highp_values[w] and lowp_values[w - 1] >
highp_values[w]:
            NDM_daily_value_14 = lowp_values[w - 1] - lowp_values[w]
            NDM_values.append( NDM_daily_value_14 )

        w+=1

    return NDM_values

def PDI (PDM_list, TR_list, period):

    PDM_values = []
    for i in PDM_list:
        PDM_values.append (i)

    TR_values = []
    for i in TR_list:
        TR_values.append (i)

    SUM_14_PDM_values = sum (PDM_values [0:period])
    SUM_14_TR_values = sum (TR_values [0:period])

    PDM14_values = []
    PDM14_values.append (SUM_14_PDM_values)

    r= 1
    while r < len (PDM_values)-period+1:

        Today_PDM_14= PDM14_values [r-1] - (PDM14_values [r-1] / period) +
PDM_values [r+period-1]
        PDM14_values.append (Today_PDM_14)
        r+=1

    TR14_values = []
    TR14_values.append (SUM_14_TR_values)

    y = 1
    while y < len (TR_values)-period+1:
        Today_TR_14 = TR14_values [y-1] - (TR14_values [y-1] / period) + TR_values
[y+period-1]
        TR14_values.append (Today_TR_14)
        y+=1
    PDI_values = []
    PDI_first_value = SUM_14_PDM_values / SUM_14_TR_values
    PDI_values.append (PDI_first_value * 100)

    i = 0
    while i < len (TR14_values):
        Today_PDI_14 = PDM14_values [i] / TR14_values [i]
        PDI_values.append (Today_PDI_14 * 100)
        i+=1

    return PDI_values

def NDI (NDM_list, TR_list, period):

    NDM_values = []
    for i in NDM_list:
        NDM_values.append (i)

    TR_values = []
    for i in TR_list:
        TR_values.append (i)

    SUM_14_NDM_values = sum (NDM_values [0:period])
    SUM_14_TR_values = sum (TR_values [0:period])

    NDM14_values = []
    NDM14_values.append (SUM_14_NDM_values)

    t = 1
    while t < len (NDM_values)-period+1:
        Today_NDM_14 = NDM14_values [t-1] - (NDM14_values [t-1] / period) +
NDM_values[t+period-1]
        NDM14_values.append (Today_NDM_14)
        t+= 1

    TR14_values = []
    TR14_values.append (SUM_14_TR_values)

    u = 1
    while u < len (TR_values)-period+1:
        Today_TR_14 = TR14_values [u-1] - (TR14_values [u-1] / period) + TR_values
[u+period-1]
        TR14_values.append (Today_TR_14)
        u+=1

    NDI_values = []
    NDI_first_value = SUM_14_NDM_values / SUM_14_TR_values
    NDI_values.append (NDI_first_value * 100)
    o = 0
    while o < len (TR14_values):
        Today_NDI_14 = NDM14_values [o] / TR14_values [o]
        NDI_values.append (Today_NDI_14 * 100)
        o+=1

    return NDI_values

def Directionality_percentage (PDI_list, NDI_list): #---> indica cuántas veces de
los últimos 14 días hubo direccionalidad

    PDI_values_TRDM = []
    NDI_values_TRDM = []

    for i in PDI_list:
        PDI_values_TRDM.append (i)
    for i in NDI_list:
        NDI_values_TRDM.append (i)

    DirectionalPorcentage_values = []

    p = 0
    while p < len (PDI_values_TRDM):
        DirectionalPorcentage_value_today = PDI_values_TRDM [p] + NDI_values_TRDM
[p]
        DirectionalPorcentage_values.append (DirectionalPorcentage_value_today)
        p+=1

    return DirectionalPorcentage_values

def TrueDirectionalMovement (PDI_list, NDI_list): #---> nos dice si la tendencia de
Directional_percentage fue más + o -

    PDI_values_TRDM = []
    NDI_values_TRDM = []

    for i in PDI_list:
        PDI_values_TRDM.append (i)
    for i in NDI_list:
        NDI_values_TRDM.append (i)

    TDRM_values = []

    p = 0
    while p < len (PDI_values_TRDM):
        TDRM_value_today = PDI_values_TRDM [p] - NDI_values_TRDM [p]
        TDRM_values.append (TDRM_value_today)
        p+=1

    return TDRM_values
```

```
def DX (PDI_list, NDI_list):

    PDI_values_DX = []
    NDI_values_DX =[]

    for i in PDI_list:
        PDI_values_DX.append (i)
    for i in NDI_list:
        NDI_values_DX.append (i)

    DX_values = []

    a=0
    while a < len (PDI_values_DX):
        DX_value_today = (PDI_values_DX [a] - NDI_values_DX [a]) / (PDI_values_DX
[a] + NDI_values_DX [a])
        DX_values.append(abs(DX_value_today)*100)
        a+=1

    return DX_values

def ADX (DX_list, period):

    DX_values = []
    for i in DX_list:
        DX_values.append (i)

    ADX_14_values = []
    ADX_14_values.append ((sum (DX_values [0:period]))/period)

    s=1
    while s < len (DX_values)-period+1:
       ADX_values_today = (((ADX_14_values [s-1] * (period -1)) + DX_values
[s+period-1]) / period)
       ADX_14_values.append (ADX_values_today)
       s+=1

    return ADX_14_values

def Trading_ADX (PDI_list, NDI_list, ADX_list):

    ADX_values = []
    for i in ADX_list:
        ADX_values.append (i)

    PDI_values = []
    for i in PDI_list:
        PDI_values.append (i)
    NDI_values = []
    for i in NDI_list:
        NDI_values.append (i)

    Long_short_points = []

    try:
        d = len (ADX_list)-2
        while d < len (ADX_list)-1:
            if (PDI_values [d-1+14] < NDI_values [d-1+14]) and (PDI_values [d+14] >
NDI_values [d+14]):
                Long_short_points.append(PDI_values [d+14-1])
            d+=1

    except:
        print ("Impossible, check it manually")
        pass

    return Long_short_points

#Setup
Apikey= "API Key"
Secret= "Secret Key"

#Authenticate
client = Client (Apikey, Secret)

#Get tickers
tickers = client.get_all_tickers()
tickers_df = pd.DataFrame (tickers, columns = ["symbol", "price"])

#List of symbols
list_of_symbols = []
for i in tickers_df ["symbol"]:
    if "USDT" in i:
       list_of_symbols.append(i)

print (list_of_symbols)

#Getting the ADX of each symbol
Trading_ADX_symbols = []
Trading_ADX_values = []
for i in list_of_symbols:

    try:

        historical = client.get_historical_klines(""+i+"",
Client.KLINE_INTERVAL_1DAY, "01 Oct 2021")
        hist_df = pd.DataFrame (historical)
        hist_df.columns = ["Open_time", "Open", "High", "Low", "Close", "Volume",
"Close time", "Quote asset volume","Number of trades", "Taker buy base asset
volume", "Taker buy quote asset volume", "Can be ignored"]
        hist_df["Open_time"] = pd.to_datetime( hist_df["Open_time"] / 1000,
unit="s" )
        hist_df["Close time"] = pd.to_datetime( hist_df["Close time"] / 1000,
unit="s" )
        numeric_columns = ["Open", "High", "Low", "Close", "Volume", "Quote asset
volume", "Taker buy base asset volume","Taker buy quote asset volume"]
        hist_df[numeric_columns] = hist_df[numeric_columns].apply(pd.to_numeric,
axis=1)
        hist_df_reduced = pd.DataFrame (hist_df, columns = ["Open_time", "Close"])

        Open_time = hist_df["Open_time"]
        highp = hist_df["High"]
        lowp = hist_df["Low"]
        closep = hist_df["Close"]
        openp = hist_df["Open"]

        try:

            TR( highp, lowp, closep )
            TR_values_applied = TR( highp, lowp, closep )

            ATR( TR_values_applied, 14 )
            ATR_TR = ATR( TR_values_applied, 14 )

            PDM( highp, lowp )
            PDM_applied = PDM( highp, lowp )

            NDM( lowp, highp )
            NDM_applied = NDM( lowp, highp )

            PDI( PDM_applied, TR_values_applied, 14 )
            PDI_applied = PDI( PDM_applied, TR_values_applied, 14 )

            NDI( NDM_applied, TR_values_applied, 14 )
            NDI_applied = NDI( NDM_applied, TR_values_applied, 14 )

            Directionality_percentage( PDI_applied, NDI_applied )
            Directionality_percentage_applied = Directionality_percentage(
PDI_applied, NDI_applied )

            TrueDirectionalMovement( PDI_applied, NDI_applied )
            TDRM_applied = TrueDirectionalMovement( PDI_applied, NDI_applied )

            DX( PDI_applied, NDI_applied )
            DX_applied = DX( PDI_applied, NDI_applied )

            ADX( DX_applied, 14 )
            ADX_applied = ADX( DX_applied, 14 )
            print (i, ADX_applied)

            Trading_ADX( PDI_applied, NDI_applied, ADX_applied )
            Trading_ADX_applied = Trading_ADX( PDI_applied, NDI_applied,
ADX_applied )
            Trading_ADX_symbols.append (i)
            try:
                Trading_ADX_values.append (Trading_ADX_applied [0])
            except:
                Trading_ADX_values.append (0)

            print(i, "Trading_ADX", Trading_ADX_applied)

        except KeyError:
            print (i, "not available data for today")

    except ValueError:
        print (i, "N/A")

Trading_ADX_symbols_df = pd.DataFrame (Trading_ADX_symbols)
Trading_ADX_symbols_df.columns = ["Cryptocurrency"]
Trading_ADX_values_df = pd.DataFrame (Trading_ADX_values)
Trading_ADX_values_df.columns = ["Values"]
Trading_ADX_df = pd.concat ([Trading_ADX_symbols_df, Trading_ADX_values_df],
axis=1)
with pd.ExcelWriter(
'C:\\Users\lucia\Desktop\Luciano\Programación\Trading_ADX_strategy.xlsx' ) as
writer:
    Trading_ADX_df.to_excel( writer, sheet_name="Trading ADX Strategy", index=False
)

print (Trading_ADX_symbols)
print ("len TS", len (Trading_ADX_symbols))
print (Trading_ADX_values)
print ("len ADX values", len (Trading_ADX_values))
```