RY = NOM – BE

BE = NOM – RY=

**def** MovingAverage(df, OHLC, MA\_Length, Type=**'Simple'**):  
 **if** Type == **'Exponential'**:  
 df[**'Simple\_Moving\_Average\_'** + OHLC + **'\_'** + str(MA\_Length)] = pd.ewma(df[OHLC], span=MA\_Length).fillna(0)  
 **elif** Type == **'Simple'**:  
 df[**'Simple\_Moving\_Average\_'** + OHLC + **'\_'** + str(MA\_Length)] = pd.rolling\_mean(df[OHLC], window=MA\_Length).fillna(0)  
  
  
**def** MACD(df, OHLC, sma, fma, MACDA\_Length, Z\_Score\_Toggle=**'Off'**, Z\_Score\_Length=208):  
  
 df[**'slow\_moving\_average'**] = pd.ewma(df[OHLC], span=sma)  
 df[**'fast\_moving\_average'**] = pd.ewma(df[OHLC], span=fma)  
 df[**'MACD'**] = df.fast\_moving\_average - df.slow\_moving\_average  
 df[**'MACDA'**] = pd.ewma(df.MACD, span=MACDA\_Length)  
  
 **if** Z\_Score\_Toggle == **'On'**:  
  
 df[**'MACD\_ZScore'**] = (df[**'MACD'**] - pd.rolling\_mean(df[**'MACD'**], window = Z\_Score\_Length))/pd.rolling\_std(df[**'MACD'**], window = Z\_Score\_Length)  
 df[**'MACD\_ZScore'**] = df[**'MACD\_ZScore'**].fillna(0)  
  
  
**def** Stochastics(df, OHLC, Stochastics, Stochastics\_PCT\_D, HighLowAvailable=**'Yes'**):  
  
 **if** HighLowAvailable == **'Yes'**:  
 df[**'K\_Fast\_Stochastic'**] = ((df[OHLC] - pd.rolling\_min(df.PX\_LOW, Stochastics)) / (pd.rolling\_max(df.PX\_HIGH, Stochastics) - pd.rolling\_min(df.PX\_LOW, Stochastics))) \* 100  
 df[**'K\_Fast\_Stochastic'**] = df[**'K\_Fast\_Stochastic'**].fillna(0)  
 **else**:  
 df[**'K\_Fast\_Stochastic'**] = ((df[OHLC] - pd.rolling\_min(df[OHLC], Stochastics)) / (pd.rolling\_max(df[OHLC], Stochastics) - pd.rolling\_min(df[OHLC], Stochastics))) \* 100  
 df[**'K\_Fast\_Stochastic'**] = df[**'K\_Fast\_Stochastic'**].fillna(0)  
  
  
 new\_col = **'SStochastic'  
  
 def** apply\_func\_decorator(func):  
 prev\_row = {}  
  
 **def** wrapper(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col] = val  
 **return** val  
 **return** wrapper  
  
 @apply\_func\_decorator  
 **def** running\_total(curr\_row, prev\_row):  
 **return** (curr\_row[**'K\_Fast\_Stochastic'**]) / Stochastics\_PCT\_D + (prev\_row.get(**'SStochastic'**, 0) \* (Stochastics\_PCT\_D - 1)) / Stochastics\_PCT\_D  
  
 df[new\_col] = df.apply(running\_total, axis=1)  
  
  
 new\_col1 = **'SStochasticD'  
  
 def** apply\_func\_decorator1(func):  
 prev\_row = {}  
  
 **def** wrapper1(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col1] = val  
 **return** val  
 **return** wrapper1  
  
 @apply\_func\_decorator1  
 **def** running\_total1(curr\_row, prev\_row):  
 **return** (curr\_row[**'SStochastic'**]) / Stochastics\_PCT\_D + (prev\_row.get(**'SStochasticD'**, 0) \* (Stochastics\_PCT\_D - 1)) / Stochastics\_PCT\_D  
  
 df[new\_col1] = df.apply(running\_total1, axis=1)  
  
  
**def** BollingerBands(df, OHLC, Std\_Dev, MA\_Length):  
 df[**'BMA\_'** + OHLC + **'\_'** + str(Std\_Dev)] = pd.rolling\_mean(df[OHLC], window = MA\_Length)  
 df[**'BHI\_'** + OHLC + **'\_'** + str(Std\_Dev)] = df[**'BMA\_'** + OHLC + **'\_'** + str(Std\_Dev)] + Std\_Dev\*pd.rolling\_std(df[OHLC], window = MA\_Length)  
 df[**'BLI\_'** + OHLC + **'\_'** + str(Std\_Dev)] = df[**'BMA\_'** + OHLC + **'\_'** + str(Std\_Dev)] - Std\_Dev\*pd.rolling\_std(df[OHLC], window = MA\_Length)  
  
  
**def** RSI(df, OHLC, RSI\_Period):  
 df[**'Up\_Days'**] = np.where(df[OHLC] / df[OHLC].shift(1) > 1, 1, 0)  
 df[**'Down\_Days'**] = np.where(df[OHLC] / df[OHLC].shift(1) < 1, 1, 0)  
 df[**'1\_Day\_Change'**] = df[OHLC] - df[OHLC].shift(1)  
 df[**'1\_Day\_Change'**] = df[**'1\_Day\_Change'**].fillna(0)  
  
  
  
 new\_col = **'Average\_Of\_Up\_Days'  
  
 def** apply\_func\_decorator(func):  
 prev\_row = {}  
  
 **def** wrapper(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col] = val  
 **return** val  
  
 **return** wrapper  
  
 @apply\_func\_decorator  
 **def** running\_total(curr\_row, prev\_row):  
 **return** ((RSI\_Period - 1)\*prev\_row.get(**'Average\_Of\_Up\_Days'**, 0) + curr\_row[**'Up\_Days'**]\*curr\_row[**'1\_Day\_Change'**])/RSI\_Period  
  
 df[new\_col] = df.apply(running\_total, axis=1)  
  
  
 new\_col1 = **'Average\_Of\_Down\_Days'  
  
 def** apply\_func\_decorator1(func):  
 prev\_row = {}  
  
 **def** wrapper1(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col1] = val  
 **return** val  
  
 **return** wrapper1  
  
 @apply\_func\_decorator1  
 **def** running\_total1(curr\_row, prev\_row):  
 **return** ((RSI\_Period - 1)\*prev\_row.get(**'Average\_Of\_Down\_Days'**, 0) + abs(curr\_row[**'Down\_Days'**]\*curr\_row[**'1\_Day\_Change'**]))/RSI\_Period  
  
 df[new\_col1] = df.apply(running\_total1, axis=1)  
  
 df[**'Relative\_Strength'**] = df[**'Average\_Of\_Up\_Days'**] / df[**'Average\_Of\_Down\_Days'**]  
 df[**'Relative\_Strength\_Index'**] = 100 - (100/(df[**'Relative\_Strength'**]+1))  
  
  
**def** Divergence(df, OHLC, Indicator, Indicator\_Filter=**'Off'**, Sensitivity = 3, Indicator\_Lower\_Limit = 0, Indicator\_Upper\_Limit = 0):  
   
  
 *# create zig zag* df[**'Percentage\_returns'**] = (((df[OHLC]/df[OHLC].shift(1))-1)\*100).fillna(0)  
 df[**'Price\_Returns'**] = np.log(df[OHLC]/df[OHLC].shift(1)).fillna(0)  
 df[**'Volatility'**] = (pd.rolling\_std(df[**'Price\_Returns'**], window = 2016)\*(252\*\*0.5)).fillna(0)  
 df[**'Zig\_Zag\_Limit'**] = df[**'Volatility'**]\*Sensitivity  
 df[**'PX\_LAST\_Shift'**] = df[OHLC].shift(1).fillna(0)  
  
  
 new\_col2 = **'Zig\_Zag'  
  
 def** apply\_func\_decorator2(func):  
 prev\_row = {}  
  
 **def** wrapper2(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col2] = val  
 **return** val  
  
 **return** wrapper2  
  
 @apply\_func\_decorator2  
 **def** running\_total2(curr\_row, prev\_row):  
 **if** curr\_row[**'Percentage\_returns'**] > curr\_row[**'Zig\_Zag\_Limit'**]:  
 **return** curr\_row[OHLC]  
 **elif** curr\_row[**'Percentage\_returns'**] < -curr\_row[**'Zig\_Zag\_Limit'**]:  
 **return** curr\_row[OHLC]  
 **else**:  
 **return** prev\_row.get(**'Zig\_Zag'**)  
  
 df[new\_col2] = df.apply(running\_total2, axis=1)  
 df[**'Zig\_Zag\_Shift'**] = df[**'Zig\_Zag'**].shift(1)  
  
  
 new\_col3 = **'Full\_Zig\_Zag'  
  
 def** apply\_func\_decorator3(func):  
 prev\_row = {}  
  
 **def** wrapper3(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col3] = val  
 **return** val  
  
 **return** wrapper3  
  
 @apply\_func\_decorator3  
 **def** running\_total3(curr\_row, prev\_row):  
 **if** curr\_row[**'Zig\_Zag'**] > curr\_row[**'Zig\_Zag\_Shift'**]:  
 **return** curr\_row[**'Zig\_Zag'**]  
 **elif** curr\_row[**'Zig\_Zag'**] < curr\_row[**'Zig\_Zag\_Shift'**]:  
 **return** curr\_row[**'Zig\_Zag'**]  
 **else**:  
 **return None** df[new\_col3] = df.apply(running\_total3, axis=1)  
 df[**'Full\_Zig\_Zag'**] = df[**'Full\_Zig\_Zag'**].interpolate()  
  
  
 *# fix last peak / trough* df[**'Previous\_Close'**] = df[**'Full\_Zig\_Zag'**]  
 df[**'Direction\_Change'**] = np.where(df[**'Full\_Zig\_Zag'**] < df[**'Full\_Zig\_Zag'**].shift(1), np.where(df[**'Full\_Zig\_Zag'**].shift(1) > df[**'Full\_Zig\_Zag'**].shift(2), **'Peak'**, **'Downtrend'**), np.where(df[**'Full\_Zig\_Zag'**].shift(1) < df[**'Full\_Zig\_Zag'**].shift(2), **'Trough'**, **'Uptrend'**))  
 df[**'Direction\_Change1'**] = np.where(df[**'Direction\_Change'**].shift(-1) == **'Peak'**, 1, np.where(df[**'Direction\_Change'**].shift(-1) == **'Trough'**, -1, 0))  
  
  
 new\_col = **'Last\_Trough'  
  
 def** apply\_func\_decorator(func):  
 prev\_row = {}  
  
 **def** wrapper(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col] = val  
 **return** val  
  
 **return** wrapper  
  
 @apply\_func\_decorator  
 **def** running\_total(curr\_row, prev\_row):  
 **if** curr\_row[**'Direction\_Change1'**] == -1:  
 **return** curr\_row[**'Previous\_Close'**]  
 **else**:  
 **return** prev\_row.get(**'Last\_Trough'**)  
  
 df[new\_col] = df.apply(running\_total, axis=1)  
  
 df[**'Previous\_Trough'**] = df[**'Last\_Trough'**].shift(1)  
  
  
 new\_col1 = **'Last\_Peak'  
  
 def** apply\_func\_decorator1(func):  
 prev\_row = {}  
  
 **def** wrapper1(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col1] = val  
 **return** val  
  
 **return** wrapper1  
  
 @apply\_func\_decorator1  
 **def** running\_total1(curr\_row, prev\_row):  
 **if** curr\_row[**'Direction\_Change1'**] == 1:  
 **return** curr\_row[**'Previous\_Close'**]  
 **else**:  
 **return** prev\_row.get(**'Last\_Peak'**)  
  
 df[new\_col1] = df.apply(running\_total1, axis=1)  
  
 df[**'Previous\_Peak'**] = df[**'Last\_Peak'**].shift(1)  
  
  
 new\_col4 = **'Indicator\_Trough'  
  
 def** apply\_func\_decorator4(func):  
 prev\_row = {}  
  
 **def** wrapper4(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col4] = val  
 **return** val  
  
 **return** wrapper4  
  
 @apply\_func\_decorator4  
 **def** running\_total4(curr\_row, prev\_row):  
 **if** curr\_row[**'Direction\_Change1'**] == -1:  
 **return** curr\_row[Indicator]  
 **else**:  
 **return** prev\_row.get(**'Indicator\_Trough'**)  
  
 df[new\_col4] = df.apply(running\_total4, axis=1)  
  
  
 new\_col5 = **'Indicator\_Peak'  
  
 def** apply\_func\_decorator5(func):  
 prev\_row = {}  
  
 **def** wrapper5(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col5] = val  
 **return** val  
  
 **return** wrapper5  
  
 @apply\_func\_decorator5  
 **def** running\_total5(curr\_row, prev\_row):  
 **if** curr\_row[**'Direction\_Change1'**] == 1:  
 **return** curr\_row[Indicator]  
 **else**:  
 **return** prev\_row.get(**'Indicator\_Peak'**)  
  
 df[new\_col5] = df.apply(running\_total5, axis=1)  
  
  
 *# divergence* **if** Indicator\_Filter == **'Off'**:  
  
 df[**'Bearish\_Divergence'**] = np.where(df[**'Direction\_Change1'**] == 1, np.where(df[**'Last\_Peak'**] > df[**'Previous\_Peak'**], np.where(df[**'Indicator\_Peak'**] < df[**'Indicator\_Peak'**].shift(1), 1, 0), 0), 0)  
 df[**'Bullish\_Divergence'**] = np.where(df[**'Direction\_Change1'**] == -1, np.where(df[**'Last\_Trough'**] < df[**'Previous\_Trough'**], np.where(df[**'Indicator\_Trough'**] > df[**'Indicator\_Trough'**].shift(1), 1, 0), 0), 0)  
  
 **else**:  
  
 df[**'Bearish\_Divergence'**] = np.where(df[**'Direction\_Change1'**] == 1, np.where(df[**'Last\_Peak'**] > df[**'Previous\_Peak'**], np.where(df[**'Indicator\_Peak'**] < df[**'Indicator\_Peak'**].shift(1), np.where(df[Indicator] > Indicator\_Upper\_Limit, 1, 0), 0), 0), 0)  
 df[**'Bullish\_Divergence'**] = np.where(df[**'Direction\_Change1'**] == -1, np.where(df[**'Last\_Trough'**] < df[**'Previous\_Trough'**], np.where(df[**'Indicator\_Trough'**] > df[**'Indicator\_Trough'**].shift(1), np.where(df[Indicator] < Indicator\_Lower\_Limit, 1, 0), 0), 0), 0)  
  
 *# df['Currently\_Divergent'] = np.where(df['PX\_LAST'] > df['Last\_Peak'], np.where(df[Indicator] < df['Indicator\_Peak'], 1, 0), np.where(df['PX\_LAST'] < df['Last\_Trough'], np.where(df[Indicator] > df['Indicator\_Trough'], -1, 0), 0))***def** ADX(df, ADX\_Period):  
 df[**'True\_Range'**] = np.where((df[**'PX\_HIGH'**] - df[**'PX\_LOW'**]) > abs(df[**'PX\_HIGH'**] - df[**'PX\_LAST'**]), np.where((df[**'PX\_HIGH'**] - df[**'PX\_LOW'**]) > abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]), df[**'PX\_HIGH'**] - df[**'PX\_LOW'**], np.where(abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]) > abs(df[**'PX\_HIGH'**] - df[**'PX\_LAST'**]), abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]), abs(df[**'PX\_HIGH'**] - df[**'PX\_LAST'**]))), np.where(abs(df[**'PX\_HIGH'**] - df[**'PX\_LAST'**]) > abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]), abs(df[**'PX\_HIGH'**] - df[**'PX\_LAST'**]), np.where(abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]) > (df[**'PX\_HIGH'**] - df[**'PX\_LOW'**]), abs(df[**'PX\_LOW'**] - df[**'PX\_OPEN'**]), (df[**'PX\_HIGH'**] - df[**'PX\_LOW'**]))))  
 df[**'Directional\_Movement\_Positive'**] = np.where((df[**'PX\_HIGH'**] - df[**'PX\_HIGH'**].shift(1)) > (df[**'PX\_LOW'**].shift(1) - df[**'PX\_LOW'**]), np.where(df[**'PX\_HIGH'**] - df[**'PX\_HIGH'**].shift(1) > 0, df[**'PX\_HIGH'**] - df[**'PX\_HIGH'**].shift(1), 0), 0)  
 df[**'Directional\_Movement\_Negative'**] = np.where((df[**'PX\_HIGH'**] - df[**'PX\_HIGH'**].shift(1)) < (df[**'PX\_LOW'**].shift(1) - df[**'PX\_LOW'**]), np.where(df[**'PX\_LOW'**].shift(1) - df[**'PX\_LOW'**] > 0, df[**'PX\_LOW'**].shift(1) - df[**'PX\_LOW'**], 0), 0)  
  
  
 new\_col2 = **'True\_Range\_Sum'  
  
 def** apply\_func\_decorator2(func):  
 prev\_row = {}  
  
 **def** wrapper2(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col2] = val  
 **return** val  
  
 **return** wrapper2  
  
 @apply\_func\_decorator2  
 **def** running\_total2(curr\_row, prev\_row):  
 **return** prev\_row.get(**'True\_Range\_Sum'**, 0) - prev\_row.get(**'True\_Range\_Sum'**, 0)/ADX\_Period + curr\_row[**'True\_Range'**]  
  
 df[new\_col2] = df.apply(running\_total2, axis=1)  
  
  
 new\_col3 = **'Positive\_Directional\_Movement\_Sum'  
  
 def** apply\_func\_decorator3(func):  
 prev\_row = {}  
  
 **def** wrapper3(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col3] = val  
 **return** val  
  
 **return** wrapper3  
  
 @apply\_func\_decorator3  
 **def** running\_total3(curr\_row, prev\_row):  
 **return** prev\_row.get(**'Positive\_Directional\_Movement\_Sum'**, 0) - (prev\_row.get(**'Positive\_Directional\_Movement\_Sum'**, 0)/ADX\_Period) + curr\_row[**'Directional\_Movement\_Positive'**]  
  
 df[new\_col3] = df.apply(running\_total3, axis=1)  
  
  
 new\_col4 = **'Negative\_Directional\_Movement\_Sum'  
  
 def** apply\_func\_decorator4(func):  
 prev\_row = {}  
  
 **def** wrapper4(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col4] = val  
 **return** val  
  
 **return** wrapper4  
  
 @apply\_func\_decorator4  
 **def** running\_total4(curr\_row, prev\_row):  
 **return** prev\_row.get(**'Negative\_Directional\_Movement\_Sum'**, 0) - prev\_row.get(**'Negative\_Directional\_Movement\_Sum'**, 0)/ADX\_Period + curr\_row[**'Directional\_Movement\_Negative'**]  
  
 df[new\_col4] = df.apply(running\_total4, axis=1)  
  
  
 df[**'Directional\_Movement\_Indicator\_Positive'**] = (df[**'Positive\_Directional\_Movement\_Sum'**] / df[**'True\_Range\_Sum'**])\*100  
 df[**'Directional\_Movement\_Indicator\_Negative'**] = (df[**'Negative\_Directional\_Movement\_Sum'**] / df[**'True\_Range\_Sum'**])\*100  
 df[**'Directional\_Index'**] = (abs(df[**'Directional\_Movement\_Indicator\_Positive'**] - df[**'Directional\_Movement\_Indicator\_Negative'**])/(df[**'Directional\_Movement\_Indicator\_Positive'**] + df[**'Directional\_Movement\_Indicator\_Negative'**]))\*100  
 df[**'Directional\_Index'**] = df[**'Directional\_Index'**].fillna(0)  
  
  
  
 new\_col5 = **'Average\_Directional\_Index'  
  
 def** apply\_func\_decorator5(func):  
 prev\_row = {}  
  
 **def** wrapper5(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col5] = val  
 **return** val  
  
 **return** wrapper5  
  
 @apply\_func\_decorator5  
 **def** running\_total5(curr\_row, prev\_row):  
 **return** (prev\_row.get(**'Average\_Directional\_Index'**, 0)\*(ADX\_Period-1) + curr\_row[**'Directional\_Index'**])/ADX\_Period  
  
 df[new\_col5] = df.apply(running\_total5, axis=1)  
  
  
**def** AMA(df, AMA\_Fast\_Length, AMA\_Slow\_Length, ER\_Length):  
 df[**'AMA\_Change'**] = abs(df[**'PX\_LAST'**] - df[**'PX\_LAST'**].shift(ER\_Length))  
 df[**'AMA\_Vol'**] = pd.rolling\_sum(abs(df[**'PX\_LAST'**] - df[**'PX\_LAST'**].shift(1)), window = ER\_Length)  
 df[**'Efficency\_Ratio'**] = df[**'AMA\_Change'**]/df[**'AMA\_Vol'**]  
 df[**'AMA\_slow\_moving\_average'**] = pd.ewma(df.PX\_LAST, span=AMA\_Slow\_Length)  
 df[**'AMA\_fast\_moving\_average'**] = pd.ewma(df.PX\_LAST, span=AMA\_Fast\_Length)  
 df[**'Smoothing\_Constant'**] = (df[**'Efficency\_Ratio'**]\*((2/(AMA\_Fast\_Length+1)) - (2/(AMA\_Slow\_Length+1))) + (2/(AMA\_Slow\_Length+1)))\*\*2  
 df[**'Smoothing\_Constant'**] = df[**'Smoothing\_Constant'**].fillna(0)  
  
 new\_col6 = **'Adaptive\_Moving\_Average'  
  
 def** apply\_func\_decorator6(func):  
 prev\_row = {}  
  
 **def** wrapper6(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col6] = val  
 **return** val  
  
 **return** wrapper6  
  
 @apply\_func\_decorator6  
 **def** running\_total6(curr\_row, prev\_row):  
 **return** prev\_row.get(**'Adaptive\_Moving\_Average'**, 0) + (curr\_row[**'PX\_LAST'**] - prev\_row.get(**'Adaptive\_Moving\_Average'**, 0))\*curr\_row[**'Smoothing\_Constant'**]  
  
 df[new\_col6] = df.apply(running\_total6, axis=1)  
  
  
**def** DeMark(df, OHLC):  
  
 df[**'PX\_LAST\_DM\_Shift1'**] = df[OHLC].shift(1).fillna(0)  
 df[**'PX\_LOW\_DM\_Shift2'**] = df[**'PX\_LOW'**].shift(2).fillna(0)  
 df[**'PX\_HIGH\_DM\_Shift2'**] = df[**'PX\_HIGH'**].shift(2).fillna(0)  
 df[**'PX\_LAST\_DM\_Shift4'**] = df[OHLC].shift(4).fillna(0)  
 df[**'PX\_LAST\_DM\_Shift5'**] = df[OHLC].shift(5).fillna(0)  
  
  
 new\_col12 = **'DeMark\_Setup'  
  
 def** apply\_func\_decorator12(func):  
 prev\_row = {}  
 **def** wrapper12(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col12] = val  
 **return** val  
 **return** wrapper12  
  
 @apply\_func\_decorator12  
 **def** running\_total12(curr\_row, prev\_row):  
 **if** prev\_row.get(**'DeMark\_Setup'**, 0) == **None**:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Setup'**, 0) <= -1:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LAST\_DM\_Shift4'**]:  
 **return** prev\_row.get(**'DeMark\_Setup'**, 0) - 1  
 **elif** curr\_row[**'PX\_LAST\_DM\_Shift1'**] > curr\_row[**'PX\_LAST\_DM\_Shift5'**]:  
 **return** 0  
 **else**:  
 **return** 1  
 **elif** prev\_row.get(**'DeMark\_Setup'**, 0) >= 1:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_LAST\_DM\_Shift4'**]:  
 **return** prev\_row.get(**'DeMark\_Setup'**, 0) + 1  
 **elif** curr\_row[**'PX\_LAST\_DM\_Shift1'**] < curr\_row[**'PX\_LAST\_DM\_Shift5'**]:  
 **return** 0  
 **else**:  
 **return** -1  
 **elif** curr\_row[**'PX\_LAST\_DM\_Shift1'**] > curr\_row[**'PX\_LAST\_DM\_Shift5'**]:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LAST\_DM\_Shift4'**]:  
 **return** -1  
 **else**:  
 **return** 0  
 **elif** curr\_row[**'PX\_LAST\_DM\_Shift1'**] < curr\_row[**'PX\_LAST\_DM\_Shift5'**]:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_LAST\_DM\_Shift4'**]:  
 **return** 1  
 **else**:  
 **return** 0  
 **else**:  
 **return** 0  
  
 df[new\_col12] = df.apply(running\_total12, axis=1)  
  
  
 df[**'DeMark\_PriceFlip'**] = np.where(df[OHLC] < df[**'PX\_LAST\_DM\_Shift4'**], np.where(df[**'PX\_LAST\_DM\_Shift1'**] > df[**'PX\_LAST\_DM\_Shift5'**], -1, 0), np.where(df[**'PX\_LAST\_DM\_Shift1'**] < df[**'PX\_LAST\_DM\_Shift5'**], 1, 0))  
  
  
 new\_col20 = **'TDST\_Resistance'** df[**'RollingMin'**] = pd.rolling\_min(df[**'PX\_LOW'**], window = 9)  
 df[**'RollingMax'**] = pd.rolling\_max(df[**'PX\_HIGH'**], window = 9)  
  
 **def** apply\_func\_decorator20(func):  
 prev\_row = {}  
 **def** wrapper20(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col20] = val  
 **return** val  
 **return** wrapper20  
  
 @apply\_func\_decorator20  
 **def** running\_total20(curr\_row, prev\_row):  
 **if** curr\_row[**'DeMark\_Setup'**] == 9:  
 **return** curr\_row[**'RollingMin'**]  
 **elif** curr\_row[**'DeMark\_Setup'**] == -9:  
 **return** curr\_row[**'RollingMax'**]  
 **else**:  
 **return** prev\_row.get(**'TDST\_Resistance'**, 0)  
  
 df[new\_col20] = df.apply(running\_total20, axis=1)  
  
  
 new\_col14 = **'Most\_Recent\_DeMark\_Setup'  
  
 def** apply\_func\_decorator14(func):  
 prev\_row = {}  
 **def** wrapper14(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col14] = val  
 **return** val  
 **return** wrapper14  
  
 @apply\_func\_decorator14  
 **def** running\_total14(curr\_row, prev\_row):  
 **if** curr\_row[**'DeMark\_Setup'**] == 9:  
 **return 'Completed\_Sell\_Setup'  
 elif** curr\_row[**'DeMark\_Setup'**] == -9:  
 **return 'Completed\_Buy\_Setup'  
 elif** prev\_row.get(**'Most\_Recent\_DeMark\_Setup'**, 0) == **'Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'Most\_Recent\_DeMark\_Setup'**, 0)  
 **elif** prev\_row.get(**'Most\_Recent\_DeMark\_Setup'**, 0) == **'Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'Most\_Recent\_DeMark\_Setup'**, 0)  
 **else**:  
 **return** prev\_row.get(**'Most\_Recent\_DeMark\_Setup'**, 0)  
  
 df[new\_col14] = df.apply(running\_total14, axis=1)  
  
  
 new\_col13 = **'DeMark\_Countdown'  
  
 def** apply\_func\_decorator13(func):  
 prev\_row = {}  
 **def** wrapper13(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col13] = val  
 **return** val  
 **return** wrapper13  
  
 @apply\_func\_decorator13  
 **def** running\_total13(curr\_row, prev\_row):  
 **if** curr\_row[**'Most\_Recent\_DeMark\_Setup'**] == **'Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown'**, 0) > -1:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** -1  
 **else**:  
 **return** 0  
 **elif** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'DeMark\_Countdown'**, 0) - 1  
 **else**:  
 **return** prev\_row.get(**'DeMark\_Countdown'**, 0)  
 **elif** curr\_row[**'Most\_Recent\_DeMark\_Setup'**] == **'Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown'**, 0) < 1:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** 1  
 **else**:  
 **return** 0  
 **elif** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'DeMark\_Countdown'**, 0) + 1  
 **else**:  
 **return** prev\_row.get(**'DeMark\_Countdown'**, 0)  
 **else**:  
 **return** 0  
  
 df[new\_col13] = df.apply(running\_total13, axis=1)  
  
  
 new\_col18 = **'8th\_Countdown\_Qualifier'** df[**'DeMark\_Countdown\_Shift'**] = df[**'DeMark\_Countdown'**].shift(1).fillna(0)  
  
  
 **def** apply\_func\_decorator18(func):  
 prev\_row = {}  
 **def** wrapper18(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col18] = val  
 **return** val  
 **return** wrapper18  
  
 @apply\_func\_decorator18  
 **def** running\_total18(curr\_row, prev\_row):  
 **if** curr\_row[**'DeMark\_Countdown'**] == 8:  
 **if** curr\_row[**'DeMark\_Countdown\_Shift'**] == 7:  
 **return** curr\_row[OHLC]  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier'**,0)  
 **elif** curr\_row[**'DeMark\_Countdown'**] == -8:  
 **if** curr\_row[**'DeMark\_Countdown\_Shift'**] == -7:  
 **return** curr\_row[OHLC]  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier'**,0)  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier'**,0)  
  
 df[new\_col18] = df.apply(running\_total18, axis=1)  
  
  
 new\_col16 = **'DeMark\_Countdown\_With\_Qualifier'  
  
 def** apply\_func\_decorator16(func):  
 prev\_row = {}  
 **def** wrapper16(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col16] = val  
 **return** val  
 **return** wrapper16  
  
 @apply\_func\_decorator16  
 **def** running\_total16(curr\_row, prev\_row):  
 **if** curr\_row[**'Most\_Recent\_DeMark\_Setup'**] == **'Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) > -1:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** -1  
 **else**:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) == -12:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **if** curr\_row[**'PX\_LOW'**] < curr\_row[**'8th\_Countdown\_Qualifier'**]:  
 **return** -13  
 **else**:  
 **return** -12  
 **else**:  
 **return** -12  
 **elif** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) - 1  
 **else**:  
 **return** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0)  
 **elif** curr\_row[**'Most\_Recent\_DeMark\_Setup'**] == **'Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) < 1:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** 1  
 **else**:  
 **return** 0  
 **elif** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) == 12:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **if** curr\_row[**'PX\_HIGH'**] > curr\_row[**'8th\_Countdown\_Qualifier'**]:  
 **return** 13  
 **else**:  
 **return** 12  
 **else**:  
 **return** 12  
 **elif** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0) + 1  
 **else**:  
 **return** prev\_row.get(**'DeMark\_Countdown\_With\_Qualifier'**, 0)  
 **else**:  
 **return** 0  
  
 df[new\_col16] = df.apply(running\_total16, axis=1)  
  
  
 *# 2nd DeMark Counts* new\_col19 = **'2nd\_Completed\_Setup'** df[**'Most\_Recent\_DeMark\_Setup\_Shift'**] = df[**'Most\_Recent\_DeMark\_Setup'**].shift(1).fillna(0)  
  
 **def** apply\_func\_decorator19(func):  
 prev\_row = {}  
 **def** wrapper19(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col19] = val  
 **return** val  
 **return** wrapper19  
  
 @apply\_func\_decorator19  
 **def** running\_total19(curr\_row, prev\_row):  
 **if** curr\_row[**'DeMark\_Setup'**] == 9:  
 **if** curr\_row[**'Most\_Recent\_DeMark\_Setup\_Shift'**] == **'Completed\_Sell\_Setup'**:  
 **return '2nd\_Completed\_Sell\_Setup'  
 else**:  
 **return 'None'  
 elif** curr\_row[**'DeMark\_Setup'**] == -9:  
 **if** curr\_row[**'Most\_Recent\_DeMark\_Setup\_Shift'**] == **'Completed\_Buy\_Setup'**:  
 **return '2nd\_Completed\_Buy\_Setup'  
 else**:  
 **return 'None'  
 elif** prev\_row.get(**'2nd\_Completed\_Setup'**, 0) == **'2nd\_Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'2nd\_Completed\_Setup'**, 0)  
 **elif** prev\_row.get(**'2nd\_Completed\_Setup'**, 0) == **'2nd\_Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'2nd\_Completed\_Setup'**, 0)  
 **else**:  
 **return** prev\_row.get(**'2nd\_Completed\_Setup'**, 0)  
  
 df[new\_col19] = df.apply(running\_total19, axis=1)  
  
  
 new\_col17 = **'2nd\_DeMark\_Countdown1'  
  
 def** apply\_func\_decorator17(func):  
 prev\_row = {}  
 **def** wrapper17(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col17] = val  
 **return** val  
 **return** wrapper17  
  
 @apply\_func\_decorator17  
 **def** running\_total17(curr\_row, prev\_row):  
 **if** curr\_row[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0) > -1:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** -1  
 **else**:  
 **return** 0  
 **elif** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0) - 1  
 **else**:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0)  
 **elif** curr\_row[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0) < 1:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** 1  
 **else**:  
 **return** 0  
 **elif** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0) + 1  
 **else**:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown1'**, 0)  
 **else**:  
 **return** 0  
  
 df[new\_col17] = df.apply(running\_total17, axis=1)  
  
  
 new\_col21 = **'8th\_Countdown\_Qualifier1'** df[**'2nd\_DeMark\_Countdown1\_Shift'**] = df[**'2nd\_DeMark\_Countdown1'**].shift(1).fillna(0)  
  
 **def** apply\_func\_decorator21(func):  
 prev\_row = {}  
 **def** wrapper21(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col21] = val  
 **return** val  
 **return** wrapper21  
  
 @apply\_func\_decorator21  
 **def** running\_total21(curr\_row, prev\_row):  
 **if** curr\_row[**'2nd\_DeMark\_Countdown1'**] == 8:  
 **if** curr\_row[**'2nd\_DeMark\_Countdown1\_Shift'**] == 7:  
 **return** curr\_row[OHLC]  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier'**,0)  
 **return** curr\_row[OHLC]  
 **elif** curr\_row[**'2nd\_DeMark\_Countdown1'**] == -8:  
 **if** curr\_row[**'2nd\_DeMark\_Countdown1\_Shift'**] == -7:  
 **return** curr\_row[OHLC]  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier'**,0)  
 **else**:  
 **return** prev\_row.get(**'8th\_Countdown\_Qualifier1'**,0)  
  
 df[new\_col21] = df.apply(running\_total21, axis=1)  
  
  
 new\_col22 = **'2nd\_DeMark\_Countdown'  
  
 def** apply\_func\_decorator22(func):  
 prev\_row = {}  
 **def** wrapper22(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col22] = val  
 **return** val  
 **return** wrapper22  
  
 @apply\_func\_decorator22  
 **def** running\_total22(curr\_row, prev\_row):  
 **if** curr\_row[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) > -1:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** -1  
 **else**:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) == -12:  
 **if** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **if** curr\_row[**'PX\_LOW'**] < curr\_row[**'8th\_Countdown\_Qualifier1'**]:  
 **return** -13  
 **else**:  
 **return** -12  
 **else**:  
 **return** -12  
 **elif** curr\_row[OHLC] < curr\_row[**'PX\_LOW\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) - 1  
 **else**:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0)  
 **elif** curr\_row[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) < 1:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** 1  
 **else**:  
 **return** 0  
 **elif** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) == 12:  
 **if** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **if** curr\_row[**'PX\_HIGH'**] > curr\_row[**'8th\_Countdown\_Qualifier1'**]:  
 **return** 13  
 **else**:  
 **return** 12  
 **else**:  
 **return** 12  
 **elif** curr\_row[OHLC] > curr\_row[**'PX\_HIGH\_DM\_Shift2'**]:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0) + 1  
 **else**:  
 **return** prev\_row.get(**'2nd\_DeMark\_Countdown'**, 0)  
 **else**:  
 **return** 0  
  
 df[new\_col22] = df.apply(running\_total22, axis=1)  
  
  
 new\_col23 = **'3rd\_Completed\_Setup'** df[**'2nd\_DeMark\_Setup\_Shift'**] = df[**'2nd\_Completed\_Setup'**].shift(1).fillna(0)  
  
  
 **def** apply\_func\_decorator23(func):  
 prev\_row = {}  
 **def** wrapper23(curr\_row, \*\*kwargs):  
 val = func(curr\_row, prev\_row)  
 prev\_row.update(curr\_row)  
 prev\_row[new\_col23] = val  
 **return** val  
 **return** wrapper23  
  
 @apply\_func\_decorator23  
 **def** running\_total23(curr\_row, prev\_row):  
 **if** curr\_row[**'DeMark\_Setup'**] == 9:  
 **if** curr\_row[**'2nd\_DeMark\_Setup\_Shift'**] == **'2nd\_Completed\_Sell\_Setup'**:  
 **return '3rd\_Completed\_Sell\_Setup'  
 else**:  
 **return 'None'  
 elif** curr\_row[**'DeMark\_Setup'**] == -9:  
 **if** curr\_row[**'2nd\_DeMark\_Setup\_Shift'**] == **'2nd\_Completed\_Buy\_Setup'**:  
 **return '3rd\_Completed\_Buy\_Setup'  
 else**:  
 **return 'None'  
 elif** prev\_row.get(**'3rd\_Completed\_Setup'**, 0) == **'3rd\_Completed\_Sell\_Setup'**:  
 **if** curr\_row[**'PX\_HIGH'**] < curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'3rd\_Completed\_Setup'**, 0)  
 **elif** prev\_row.get(**'3rd\_Completed\_Setup'**, 0) == **'3rd\_Completed\_Buy\_Setup'**:  
 **if** curr\_row[**'PX\_LOW'**] > curr\_row[**'TDST\_Resistance'**]:  
 **return 'None'  
 else**:  
 **return** prev\_row.get(**'3rd\_Completed\_Setup'**, 0)  
 **else**:  
 **return** prev\_row.get(**'3rd\_Completed\_Setup'**, 0)  
  
 df[new\_col23] = df.apply(running\_total23, axis=1)  
  
  
 df[**'Current\_DeMark\_Countdown'**] = np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**] > 0, np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**] <= 13, np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**].shift(1) == 13, df[**'2nd\_DeMark\_Countdown'**], df[**'DeMark\_Countdown\_With\_Qualifier'**]), df[**'2nd\_DeMark\_Countdown'**]),  
 np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**] >= -13, np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**].shift(1) == -13, df[**'2nd\_DeMark\_Countdown'**], df[**'DeMark\_Countdown\_With\_Qualifier'**]), df[**'2nd\_DeMark\_Countdown'**]))  
  
 df[**'Clean\_or\_Dirty?'**] = np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**] == 13, np.where(df[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Sell\_Setup'**, **'Dirty'**, **'Clean'**), np.where(df[**'DeMark\_Countdown\_With\_Qualifier'**] == -13, np.where(df[**'2nd\_Completed\_Setup'**] == **'2nd\_Completed\_Buy\_Setup'**, **'Dirty'**, **'Clean'**),  
 np.where(df[**'2nd\_DeMark\_Countdown'**] == 13, np.where(df[**'3rd\_Completed\_Setup'**] == **'3rd\_Completed\_Sell\_Setup'**, **'Dirty'**, **'Clean'**), np.where(df[**'2nd\_DeMark\_Countdown'**] == -13, np.where(df[**'3rd\_Completed\_Setup'**] == **'3rd\_Completed\_Buy\_Setup'**, **'Dirty'**, **'Clean'**), 0))))  
  
  
**def** Create\_Baskets(basket, basket\_df):  
  
 eurusd\_df1 = pd.DataFrame()  
  
 y = basket  
  
 **for** x **in** y:  
  
 **for** Enter\_Asset **in** x:  
  
 **try**:  
 px\_last\_ts = bh.bhTsRead(**"ts.test"**, **"BBG"**, **"Curncy"**, **"\*"**, **"\*"**, Enter\_Asset, **"PX\_LAST"**, today\_10Y, yesterday)  
 PX\_LAST = bhc.ts\_to\_series(px\_last\_ts)  
  
 **except** TypeError:  
  
 **try**:  
  
 Enter\_Asset = Enter\_Asset[3:] + Enter\_Asset[:3]  
 px\_last\_ts = bh.bhTsRead(**"ts.test"**, **"BBG"**, **"Curncy"**, **"\*"**, **"\*"**, Enter\_Asset, **"PX\_LAST"**, today\_10Y, yesterday)  
 PX\_LAST = bhc.ts\_to\_series(px\_last\_ts)  
 PX\_LAST = 1/PX\_LAST  
  
 Enter\_Asset = Enter\_Asset[3:] + Enter\_Asset[:3]  
  
 **except** TypeError:  
  
 Enter\_Asset = Enter\_Asset[3:] + Enter\_Asset[:3]  
 Enter\_Asset1 = **'USD'** + Enter\_Asset[3:]  
 px\_last\_ts = bh.bhTsRead(**"ts.test"**, **"BBG"**, **"Curncy"**, **"\*"**, **"\*"**, Enter\_Asset1, **"PX\_LAST"**, today\_10Y, yesterday)  
 PX\_LAST1 = bhc.ts\_to\_series(px\_last\_ts)  
  
 Enter\_Asset2 = **'USD'** + Enter\_Asset[:3]  
 px\_last\_ts = bh.bhTsRead(**"ts.test"**, **"BBG"**, **"Curncy"**, **"\*"**, **"\*"**, Enter\_Asset2, **"PX\_LAST"**, today\_10Y, yesterday)  
 PX\_LAST2 = bhc.ts\_to\_series(px\_last\_ts)  
  
 PX\_LAST = PX\_LAST1/PX\_LAST2  
  
  
 eurusd\_df1[Enter\_Asset] = PX\_LAST  
 eurusd\_df1.index = eurusd\_df1.index.to\_datetime()  
 eurusd\_df1[Enter\_Asset + **' Rebased'**] = ((eurusd\_df1[Enter\_Asset]/eurusd\_df1[Enter\_Asset].shift(1)).fillna(1)).cumprod()  
  
 Enter\_Asset\_Base = [s + **' Rebased' for** s **in** x]  
 eurusd\_df1[Enter\_Asset[:3] + **' vs Basket'**] = eurusd\_df1[Enter\_Asset\_Base].mean(axis=**'columns'**)  
  
 basket\_df[Enter\_Asset[:3] + **' vs Basket'**] = eurusd\_df1[Enter\_Asset[:3] + **' vs Basket'**]