Queensland University of Technology
QUTTIC

Python for Finance

A Gentle Introduction

Andrew Collison collison@qut.edu.au

0.1 Indicators

```
"""
QUTTIC Crash Course Python

This file will contain the classes and functions
used to calculate the indicators.

Andrew Collison 07-02-18
"""
# Import the modules we need
import pandas as pd
import numpy as np

# Load the data into a pandas data frame
data = pd.read_csv("pair_data2.csv", parse_dates = True)
print()
## Create the class to hold the functions
class indicators:
    # Define the functions for the desired indicators
    # Moving Average
    def moving_average(data, window):
        # Make the moving average calculation
        MA = data['Close'].rolling(center=False, window = window).mean()
        # Name the indicator
        name = 'MA_'+str(window)
        # Append it to the origional dataset
        data[name] = MA
        # Return the data frame
        return data

    # Keltner Channel
    def keltner(data):
        ### Calculate ATR
        H_minus_L = data.High - data.Low
        H_minus_Cp = data.High - data.Close
        L_minus_Cp = data.Low - data.Close
        # Create a data frame of daily volatility
        ATR_calc = pd.DataFrame({'H-L': H_minus_L, 'H-CP': H_minus_Cp, 'L-CP': L_minus_Cp
    })
        #Calculate the moving average of the ATR
        ATR = ATR_calc.max(axis=1)
        ATR = ATR.rolling(center=False, window=10).mean()
        # Append ATR to the data frame
        data['ATR'] = ATR

        ### Calculate the EXP MA
        data['ExpMA'] = data['Close'].ewm(span=20,min_periods=0,adjust=False,ignore_na=
    False).mean()

        ### Calculate the Keltner Channel
        data['kelt_upper'] = data['ExpMA']+(1.5*data['ATR'])
        data['kelt_lower'] = data['ExpMA']-(1.5*data['ATR'])

        return data

    # MACD
    def MACD(data):
        ewm26 = data.Close.ewm(span=26, adjust=True, min_periods=20).mean()
        ewm12 = data.Close.ewm(span=20, adjust=True, min_periods=20).mean()
        ewm9 = data.Close.ewm(span=9, adjust=True, min_periods=20).mean()
```

```python

        MACD = ewm12 - ewm26
        MACD_signal = ewm = MACD.ewm(span=20, adjust=True, min_periods=20).mean()
        MACD_hist = MACD - MACD_signal


        data['MACD'] = MACD
        data['MACD_signal'] = MACD_signal
        data['MACD_hist'] = MACD_hist

        return data

    # RSI
    def rsi(data):
        window_length =14
        close = data['Close']
        # Get the difference in price from previous step
        delta = close.diff()
        # Get rid of the first row, which is NaN since it did not have a previous
        # row to calculate the differences
        delta = delta[1:]
        # Make the positive gains (up) and negative gains (down) Series
        up, down = delta.copy(), delta.copy()
        up[up < 0] = 0
        down[down > 0] = 0
        # Calculate the EWMA
        roll_up1 = up.ewm(min_periods=14,span=14,adjust=False).mean()
        roll_down1 = down.abs().ewm(min_periods=14,span=14,adjust=False).mean()
        # Calculate the RSI based on EWMA
        RS1 = roll_up1 / roll_down1
        RSI1 = 100.0 - (100.0 / (1.0 + RS1))
        data['RSI'] = RS1
        return data

    # Parabolic Sar
    def psar(data):
        iaf = 0.02
        maxaf = 0.2
        length = len(data)
        dates = list(data.index)
        high = list(data['High'])
        low = list(data['Low'])
        close = list(data['Close'])
        psar = close[0:len(close)]
        psarbull = [None] * length
        psarbear = [None] * length
        bull = True
        af = iaf
        ep = low[0]
        hp = high[0]
        lp = low[0]
        for i in range(2,length):
            if bull:
                psar[i] = psar[i - 1] + af * (hp - psar[i - 1])
            else:
                psar[i] = psar[i - 1] + af * (lp - psar[i - 1])
            reverse = False
            if bull:
                if low[i] < psar[i]:
                    bull = False
                    reverse = True
                    psar[i] = hp
                    lp = low[i]
                    af = iaf
            else:
                if high[i] > psar[i]:
                    bull = True
                    reverse = True
                    psar[i] = lp
                    hp = high[i]
                    af = iaf
            if not reverse:
                if bull:
                    if high[i] > hp:
                        hp = high[i]
                        af = min(af + iaf, maxaf)
                    if low[i - 1] < psar[i]:
                        psar[i] = low[i - 1]
                    if low[i - 2] < psar[i]:
                        psar[i] = low[i - 2]
                else:
                    if low[i] < lp:
                        lp = low[i]
                        af = min(af + iaf, maxaf)
                    if high[i - 1] > psar[i]:
                        psar[i] = high[i - 1]
                    if high[i - 2] > psar[i]:
                        psar[i] = high[i - 2]
            if bull:
                psarbull[i] = psar[i]
            else:
                psarbear[i] = psar[i]


        data['psar'] = psar
        data['psar_bull'] = psarbull
        data['psar_bear'] = psarbear

        return data

# data = indicators.moving_average(data, 20)
# data = indicators.moving_average(data, 200)
# data = indicators.keltner(data, 200)
# data = indicators.psar(data)
# data = indicators.MACD(data)
# data = indicators.rsi(data)
# print(data)

### We will save this file for later
# data.to_csv('indicator_data.csv')
```

0.2 Vis Data

```
"""
QUTTIC Crash Course Python

This file will be the main document
    - Here we will call our indicator functions
    - Load data and build our data frame
    - Graph the data

Andrew Collison 09-02-18
"""

# Import the modules we need
from indicators import indicators # we just wrote this module
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

## Load the data into a dataframe object named "df" using pd.read_csv()
df = pd.read_csv('pair_data2.csv')

## Call our indicator functions
# Calculate 50 day moving average
df = indicators.moving_average(df, 50)
df = indicators.moving_average(df, 200)
print(df)

## Display the data
df.plot(x = 0, y = ['Close', 'MA_50', 'MA_200'])
plt.title('Currency Pair')
plt.xlabel('Days')
plt.ylabel('Price')
plt.show()



#### Time pending: implement a simple trading algo using control structure
```

0.3 Trading Strategy

```python
"""
QUTTIC Crash Course Python

This will demonstrate a simple back testing
algo that can be used for evaluation of possible
trading stratergies.

Andrew Collison: 13/09/18

"""

### Import the packages we need
import pandas as pd
import numpy as np
from indicators import indicators
import matplotlib.pyplot as plt
plt.style.use('ggplot')

### Import our pair data
data = pd.read_csv("pair_data2.csv")

### Set the index to the time vector
data = data.set_index(data['time'])

### Calculate the indicators
# 50 and 200 day moving average
data = indicators.moving_average(data, 50)
data = indicators.moving_average(data, 200)
# Drop any NAN values
data = data.dropna(axis=0, how='any')
print(data)

### Starting portfolio param
data["Regime"] = 0
data["Profit"] = 0
### Define our stratergy
class stratergy(object):
    """return 1 for long position
       return -1 for short position
    """
    def Regime(data):
        for index, row in data.iterrows():
            if row["MA_50"] > row["MA_200"]:
                row["Regime"] = 1
                data.loc[index, "Regime"] = 1
                # print("Buy", index, row["Close"], row["Regime"])
            elif row["MA_50"] < row["MA_200"]:
                # row["Regime"] = -1
                data.loc[index, "Regime"] = -1
                # print("Sell", index, row["Close"], row["Regime"])
        print("These are the stratergy results: \n", data["Regime"].value_counts())
        return data


### Calculate Profits for trades
class test_strat(object):
    """docstring for test_strat
    Take the data generated above in the regime
    and place by and sell positions.
    """
    def long_trades(data):
        # Convert into lists
        date = list(data["time"])
        close = list(data["Close"])
        regime = list(data["Regime"])
        profit = list(data["Profit"])
        # Strating param
        open_idx = 0
        close_idx = 0
        p_open = []
        p_close = []

        # If final data point is open trade
        # force close
        if regime[-1] == 1:
            regime[-1] = -1

        # Strart evaluating positions
        for i in range(len(date)):
            if regime[i] == 1 and regime[i-1] == -1:
                open_idx = i

            if regime[i] == -1 and regime[i-1] == 1:
                profit[i] = close[i]-close[open_idx]

        cp = np.cumsum(profit)
        print("Long Profit:", cp[-1])


    def short_trades(data):
        date = list(data["time"])
        close = list(data["Close"])
        regime = list(data["Regime"])
        profit = list(data["Profit"])
        open_idx = 0
        close_idx = 0
        p_open = []
        p_close = []

        # If final data point results in open trade
        # force the trade to close at the closing price
        if regime[-1] == -1:
            regime[-1] = -1
            # print(regime)

        for i in range(len(date)):
            if regime[i] == -1 and regime[i-1] == 1:
                open_idx = i
                # print("Short: Price Open:", close[i] , "Date:", date[i] )

            if regime[i] == 1 and regime[i-1] == -1:
                profit[i] = close[open_idx] - close[i]
                # print("Short: Price Close:",close[i], "Profit:", profit[i], "Date:", date[i] )

        cp = np.cumsum(profit)
        # print(cum_p[-1])
        print("Short Profit:", cp[-1])


```

```
### Function to show data
def vis_results(data):
    fig, axes = plt.subplots(nrows = 2, ncols = 1, sharex = True)
    data[['Close', 'MA_200', 'MA_50']].plot(ax = axes[0])
    data["Regime"].plot(ax = axes[1])
    plt.show()


# stratergy.Regime(data)
data = stratergy.Regime(data)
# print(data)
test_strat.long_trades(data)
test_strat.short_trades(data)

vis_results(data)
```