buy and hold - bollinger bands invest

September 23, 2024

Functions (IGNORE)

[ ]:
```
import yfinance as yf
missing_data_tickers = [] # use this as a list of tickers with missing data

def get_data_from_start_to_end(ticker, start_date, end_date):
    global missing_data_tickers  # Use the global list to accumulate missing␣
 ↪tickers
    try:
        stock_data = yf.download(ticker, start=start_date, end=end_date)
        if stock_data.empty:
            missing_data_tickers.append(ticker)
            raise ValueError(f"Stock data for ticker {ticker} during the period␣
 ↪from {start_date} to {end_date} was not found.")
        return stock_data
    except Exception as e:
        print(f"An error occurred for ticker {ticker}: {e}")
        missing_data_tickers.append(ticker)
        return None
```

[ ]:
```
# for a variety of periods load in different list of tickers
def download_stock_data_for_periods(tickers, periods):
    all_data = {}

    for period, (start_date, end_date) in periods.items():
        period_data = {}
        for ticker in tickers:
            data = get_data_from_start_to_end(ticker, start_date, end_date)
            if data is not None:
                period_data[ticker] = data
        all_data[period] = period_data

    return all_data
```

[ ]:
```
import pandas as pd

# Get the adjusted close prices
adj_close_sector_etf = {}
```

```
# Create adjusted close price only listing of sector ETFs
def get_adjusted_closed_price(nested_dict, tickers, periods):
    for period in periods:
        stock_price_df = pd.DataFrame()  # Create a new DataFrame for each␣
 ↪period
        for ticker in tickers:
            stock_price_df[ticker] = nested_dict[period][ticker]['Adj Close']

        adj_close_sector_etf[period] = stock_price_df  # Store the complete␣
 ↪DataFrame for the period

    return adj_close_sector_etf
```

[ ]:
```
import random

def stochastic_modeling(nested_dict, tickers, periods,num_samples):
    # Store the returns in a nested dictionary
    nested_dict_returns = {period: {ticker: [] for ticker in tickers} for␣
 ↪period in periods}

    # Go through each economic time period
    for period in periods:
        max_index = len(nested_dict[period]) - 30  # Ensure there's enough data␣
 ↪to calculate ROI

        # Generate random samples from the valid range
        random_dates = random.choices(range(max_index), k=num_samples)

        for ticker in tickers:
            for date_idx in random_dates:
                start_price = nested_dict[period][ticker].iloc[date_idx]
                end_price = nested_dict[period][ticker].iloc[date_idx + 30]

                # Get the return by the Holding Period Return
                roi = (((end_price - start_price) / start_price) * 100)

                nested_dict_returns[period][ticker].append(roi)

    return nested_dict_returns  # Return the nested dictionary with returns
```

[ ]:
```
def stochastic_roi(tickers,periods,return_rates_list,analysis_type):
    df = pd.DataFrame(index=tickers,columns=periods)
    for period in periods:
        for ticker in tickers:
            data = pd.Series(return_rates_list[period][ticker])
            if analysis_type=='Mean':
                df.at[ticker,period] = data.mean()
```

```
            elif analysis_type=='Median':
                df.at[ticker,period] = data.median()
            elif analysis_type=='Std':
                df.at[ticker,period] = data.std()
            elif analysis_type=='Variance':
                df.at[ticker,period] = data.var()

    return df
```

1 Technical Analysis Investment Strategy

[ ]:
```
# import packages
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
# create time periods for where this takes place
economic_cycle_periods = {

    "trough": ("2008-10-01", "2009-06-01"),
    "expansion": ("2012-01-01", "2015-01-01"),
    "peak": ("2019-06-01", "2020-02-01"),
    "contraction": ("2007-12-01", "2008-10-01"),
}

economic_cycle_periods_list = ['trough','expansion','peak','contraction']
```

[ ]:
```
# create etf tickers for sectors
sector_etf_tickers = [
    'XLB', # materials sector
    'XLI', # industrials sector
    'XLF', # financials
    'XLK', # information technology
    'XLY', # consumer discretionary
    'XLP', # consumer staples
    'XLE', # energy
    'XLV', # healthcare
    'VOX', # communication services
    'XLU', # utilities
    'IYR' # real estate
    ]
```

1.1 Buy and Hold Investment Technique

The buy and hold strategy is a passive investing strategy that will be applied to the 11 sector ETFs during different macroeconomic time periods.

[ ]:

```python
# save nested dictionary data as a variable to be accessed.
sector_etf_data = download_stock_data_for_periods(sector_etf_tickers,economic_cycle_periods)
```

```
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

[ ]:
```
# get adjusted close price
sector_etf_adjusted_close =␣
 ↪get_adjusted_closed_price(sector_etf_data,sector_etf_tickers,economic_cycle_periods_list)
```

1.1.1 Perform stochastic modeling using buy and hold strategy

Use a different day where the stock begins investing then hold for a month and see the return.

[ ]:
```
# perform stochastic modeling on the buy and
stochastic_buy_hold =␣
 ↪stochastic_modeling(sector_etf_adjusted_close,sector_etf_tickers,economic_cycle_periods_lis
```

[ ]:
```
# this can be repeated for mean, median, std and var
stochastic_roi(sector_etf_tickers,economic_cycle_periods_list,stochastic_buy_hold,'Mean')
```

[ ]:

	trough	expansion	peak	contraction
XLB	1.028555	1.565582	1.256081	-1.295992
XLI	-1.070246	2.174193	2.097345	-1.996158
XLF	-2.580278	2.586072	2.814295	-4.721434
XLK	1.412234	2.125499	4.573979	-2.724051
XLY	2.100069	2.48151	1.525487	-0.95973
XLP	-1.503632	2.049964	1.964145	-0.034513
XLE	0.037745	0.814569	-0.063137	-1.244337
XLV	-1.099252	3.038329	2.802807	-1.592813
VOX	2.531715	1.718887	2.530143	-3.423181
XLU	-1.213921	1.726195	2.344213	-2.718045
IYR	-2.883691	1.578877	1.516647	-0.034839

1.2 Bollinger Bands Investment Technique

Using John Bollinger's techniques 'Bollinger Bands' to create buy and sell signals to observe the roi for investing for a month.

[ ]:
```
# add bollinger data
import scipy.stats as stats
def add_bollinger_data(data,window,conf_int):
        z_score = stats.norm.ppf(1 - (1 - conf_int) / 2) # create a zscore from␣
 ↪the mean

        data['middle_band'] = data['Adj Close'].rolling(window).mean()
        data['upper_band'] = data['middle_band'] + z_score * data['Adj Close'].
 ↪rolling(window).std()
        data['lower_band'] = data['middle_band'] - z_score * data['Adj Close'].
 ↪rolling(window).std()

        data['Signal'] = None

        data['Signal'] = np.where(data['Adj Close'] < data['lower_band'], 'Buy',
```

```
                                np.where(data['Adj Close'] > data['upper_band'],␣
↪'Sell', np.nan))

        return data
```

[ ]:
```
# create bollinger data for multiple time period and multiple tickers
def␣
↪bollinger_data_multiple_periods_tickers(periods,tickers,data,window,confidence_period):
↪
    # for each ticker in economic time periods
    for period in periods:
            for ticker in tickers:
                    try:
                        ␣
↪add_bollinger_data(data[period][ticker],window,confidence_period)
                    except KeyError:
                        print(f'Data for {ticker} does not exist during␣
↪{period}')
```

[ ]:
```
# create bollinger bands in stock data
bollinger_data_multiple_periods_tickers(economic_cycle_periods_list,sector_etf_tickers,sector_
↪95)
sector_etf_data['trough']['XLB']
```

[ ]:
```
                 Open       High        Low      Close  Adj Close    Volume  \
Date
2008-10-01  32.759998  33.189999  32.130001  32.849998  23.222450  14639500
2008-10-02  31.540001  31.860001  29.930000  30.490000  21.554117  12581300
2008-10-03  30.190001  31.690001  29.780001  30.190001  21.342041  16770600
2008-10-06  29.510000  29.510000  26.889999  28.700001  20.288715  22512700
2008-10-07  29.160000  29.530001  27.049999  27.219999  19.242476  16004900
…                 …          …          …          …          …         …
2009-05-22  26.530001  26.660000  26.110001  26.299999  18.932842   8421500
2009-05-26  26.170000  26.969999  25.830000  26.930000  19.386368   7886900
2009-05-27  26.790001  26.850000  25.860001  25.920000  18.659294   7216600
2009-05-28  26.190001  26.440001  25.760000  26.379999  18.990433   8773400
2009-05-29  26.670000  27.200001  26.500000  27.170000  19.559143   7792200

            middle_band  upper_band  lower_band Signal
Date
2008-10-01          NaN         NaN         NaN    nan
2008-10-02          NaN         NaN         NaN    nan
2008-10-03          NaN         NaN         NaN    nan
2008-10-06          NaN         NaN         NaN    nan
2008-10-07          NaN         NaN         NaN    nan
…                   …           …           …      …
2009-05-22    18.825941   19.996749   17.655134    nan
```

```
2009-05-26    18.911607   19.977388   17.845826    nan
2009-05-27    18.965598   19.847745   18.083451    nan
2009-05-28    19.016350   19.771105   18.261594    nan
2009-05-29    19.070341   19.818088   18.322593    nan

[166 rows x 10 columns]
```

[ ]:
```
# example case of bollinger bands in stock data
sector_etf_data['expansion']['XLB']['Signal'].value_counts()
```

[ ]:
```
nan     670
Buy      45
Sell     39
Name: Signal, dtype: int64
```

[ ]:
```
def collect_signals(nested_dict, periods, tickers):
    # Initialize an empty dictionary to hold DataFrames for each period
    bb_nested_dict = {}

    for period in periods:
        # Create a DataFrame for each period with the tickers as columns
        signals_period = pd.DataFrame(columns=tickers)

        # Loop through each ticker and extract the 'Signal'
        for ticker in tickers:
            signals_period[ticker] = nested_dict[period][ticker]['Signal']

        # Store the DataFrame in the dictionary using the period as the key
        bb_nested_dict[period] = signals_period

    # Return the dictionary containing DataFrames for each period
    return bb_nested_dict
```

[ ]:
```
bb_bands_signals = collect_signals(sector_etf_data,economic_cycle_periods_list,sector_etf_tickers)
bb_bands_signals['trough']
```

[ ]:
```
            XLB  XLI  XLF  XLK  XLY  XLP  XLE  XLV  VOX  XLU  IYR
Date
2008-10-01  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2008-10-02  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2008-10-03  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2008-10-06  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2008-10-07  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
…           …    …    …    …    …    …    …    …    …    …    …
2009-05-22  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2009-05-26  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
```

```
2009-05-27  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2009-05-28  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan
2009-05-29  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan  nan

[166 rows x 11 columns]
```

[ ]:
```
# make sure that the length of the two dataframes are same
# this is so that you can treat them as two panes stacked on top of one another␣
 ↪and each index is relevant to the date and ticker
len(sector_etf_adjusted_close) == len(bb_bands_signals)
```

[ ]:
```
True
```

[ ]:
```
# goal is to create a function that uses the bb signal df and adj close signal␣
 ↪df
# you run through the signal day by day ticker by ticker
# get the buy/sell/hold action
# dependent on action get the adj close price
# invest/sell a certain amount based on how much cash is available
# perform stocastic modeling similar to buy and hold to see how the investment␣
 ↪changes for each different sectors in different time periods

# create function for signals
def bb_band_roi(bb_signals_nd, adj_close_nd,periods,tickers,n_sample):
    # create a nested dictionary with technical analysis signals and adj close␣
 ↪price as pages
    all_data = {
        'Adj Close': adj_close_nd,
        'Bollinger Band': bb_signals_nd
    }

    # create start index dates
    random_dates = random.choices(range(max_index),k=n_sample)

    # go through each period
    for period in periods:
        # go through each day in the signals then collect the location
        for row_idx, row in all_data['Bollinger Band'][period].iterrows():
            for col_idx,value in enumerate(row):
                for ticker in tickers:
                    if row[value]=='Buy':
                        adj_close_price = adj_close_nd.iloc[col_idx,row_idx]
                        print(f'buy ticker {col_idx} at day {row_idx} ')
                        print(f'at price {adj_close_price}')
                    elif row[ticker] == 'Sell':
                        print(f'sell ticker {col_idx} at day {row_idx}')
                        adj_close_price = adj_close_nd.iloc[col_idx,row_idx]
```

```
                    print(f'at price {adj_close_price}')
                else: continue




# get the max index so that there is a month of investing time
max_index = len(adj_close_nd[period]-30)

# this is your own keep working
```