RSI Strategy

September 29, 2024

Functions (IGNORE)

[ ]:

```python
# import packages that will be used for analysis
import random
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
```

Collect Stock Data

[ ]:

```python
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
```

[ ]:

```python
# 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
```

Relative Strength Index

[ ]:

```
def calculate_rsi(data, window=14):
    """
    Calculate the Relative Strength Index (RSI) for a given stock data series.

    Parameters:
    data (pd.Series): A pandas series of adjusted close prices.
    window (int): The lookback period for RSI calculation, default is 14.

    Returns:
    pd.Series: RSI values.
    """
    delta = data.diff()  # Difference in price from previous price
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()  # Average␣
 ↪gain
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()  #␣
 ↪Average loss

    # Avoid division by zero, especially at the beginning of the dataset
    rs = gain / loss.replace(0, np.nan)

    # RSI formula
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

[ ]:

```
# create rsi value in sector etf dataframe

def rsi_value(nested_dict,periods,tickers):
    for period in periods:
        for ticker in tickers:
            nested_dict[period][ticker]['RSI'] =␣
 ↪calculate_rsi(nested_dict[period][ticker]['Adj Close'])
```

[ ]:

```
import numpy as np

def create_rsi_signal(nested_dict, periods, tickers):
    """
    Adds a 'Signal' column to the nested dictionary based on RSI values.

    Parameters:
    - nested_dict: A nested dictionary where each period contains dataframes␣
 ↪for tickers.
                   Each dataframe should have an 'RSI' column.
    - periods: A list of periods to iterate over.
    - tickers: A list of tickers to process within each period.

    Returns:
    - The modified nested dictionary with new 'Signal' columns.
    """

    for period in periods:
        for ticker in tickers:
            # Create the 'Signal' column using np.where
            nested_dict[period][ticker]['Signal'] = np.where(
                nested_dict[period][ticker]['RSI'] < 30, 'Buy',
                np.where(nested_dict[period][ticker]['RSI'] > 70, 'Sell',␣
 ↪'Hold')
            )

    return nested_dict
```

[ ]:

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

    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
        rsi_signal_df[period] = signals_period

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

1 Chapter 3: Relative Strength Index

Relative Strength Index is another popular component of technical analysis. Similar to bollinger bands it looks to identify when there is an opportunity to enter the market when equities have been overbought or oversold. The RSI is a moving oscillator and falls between a value of 0 and 100. It is typically plotted below the line of an equity to get an overview of the movement of the stock. An asset is overbought when the value is greater than 70 which implies a sell signal and an asset is oversold when the value is less than 30 which implies a buy signal.

- RSI (between 0 and 100): Calculated over a 14 day period where RS is identified by

1.1 Relative Strength Index Strategy

The goal of the RSI is to create a dataframe of signals based on the thresholds as explained above. This can then be used to analyze the performance of incoprorating RSI signals in comparison to a passive buy and hold strategy.

1.2 Sector ETF and Time Period Setup

[ ]:

```
# 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"),
    'all_data': ('2005-01-01','2024-06-01')
}

economic_cycle_periods_list =␣
 ↪['trough','expansion','peak','contraction','all_data']
```

[ ]:

```
# 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
    ]
```

[ ]:
```
# 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
[*********************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
```

1.3 Relative Strength Index (RSI)

```
[ ]: # create rsi signal and look at 'XLV" in trough
     create_rsi_signal(sector_etf_data,economic_cycle_periods_list,sector_etf_tickers)['trough']['X
      ↪head(50)
```

```
[ ]:                  Open       High        Low      Close  Adj Close    Volume  \
     Date
     2008-10-01  30.100000  30.480000  30.100000  30.250000  22.927488   6053600
     2008-10-02  30.250000  30.590000  29.930000  30.299999  22.965387   6353400
     2008-10-03  30.600000  30.600000  29.650000  29.650000  22.472729   6814400
     2008-10-06  29.400000  29.879999  27.410000  28.540001  21.631420   8545000
     2008-10-07  28.719999  28.780001  27.389999  27.850000  21.108444   5060200
     2008-10-08  27.030001  27.820000  26.549999  27.350000  20.729475   9958100
     2008-10-09  27.639999  27.639999  25.000000  25.250000  19.137815  10773200
     2008-10-10  24.100000  25.540001  22.889999  24.139999  18.296513  15960800
     2008-10-13  25.680000  27.830000  25.219999  27.049999  20.502104   7756200
     2008-10-14  28.450001  28.559999  26.840000  27.600000  20.918964  12535900
     2008-10-15  27.370001  27.370001  24.900000  24.900000  18.872545   8958600
     2008-10-16  25.629999  26.700001  24.389999  26.200001  19.857859  11304400
     2008-10-17  25.620001  27.510000  25.620001  26.100000  19.782055   7743000
     2008-10-20  26.450001  27.480000  26.420000  27.150000  20.577892   8094200
     2008-10-21  27.100000  27.709999  27.010000  27.070000  20.517263   4888900
     2008-10-22  26.750000  26.750000  25.370001  26.400000  20.009443  10562300
     2008-10-23  25.809999  26.590000  24.950001  25.790001  19.547108   7834800
     2008-10-24  24.230000  26.020000  24.230000  25.700001  19.478888  10636200
     2008-10-27  25.209999  25.750000  24.120001  24.600000  18.645163   5929000
```

```
2008-10-28  25.430000  27.500000  24.580000  26.150000  19.819965   7073300
2008-10-29  26.299999  26.850000  25.639999  25.969999  19.683529   7502500
2008-10-30  26.280001  26.860001  25.990000  26.580000  20.145868   4827900
2008-10-31  26.629999  27.209999  26.420000  26.600000  20.161022   6100100
2008-11-03  26.670000  27.129999  26.670000  26.799999  20.312611   3951600
2008-11-04  27.400000  27.570000  27.070000  27.150000  20.577892   3884000
2008-11-05  27.110001  27.420000  26.440001  26.750000  20.274725   5999500
2008-11-06  26.500000  26.629999  25.559999  26.100000  19.782055   8637100
2008-11-07  26.059999  26.590000  25.799999  26.430000  20.032188   5676300
2008-11-10  27.120001  27.120001  26.049999  26.240000  19.888168   4835800
2008-11-11  26.219999  26.330000  25.600000  25.770000  19.531939  10628000
2008-11-12  25.420000  25.730000  24.969999  25.200001  19.099924   4237700
2008-11-13  25.020000  26.549999  24.530001  26.549999  20.123127   7599700
2008-11-14  25.879999  26.709999  25.540001  25.540001  19.357622   5996900
2008-11-17  25.080000  25.830000  25.010000  25.100000  19.024134   6151400
2008-11-18  24.910000  25.450001  24.410000  25.270000  19.152977   5928200
2008-11-19  25.410000  25.610001  24.100000  24.250000  18.379889   6997800
2008-11-20  23.980000  24.389999  22.360001  23.559999  17.856913  12781400
2008-11-21  23.270000  23.670000  21.990000  23.590000  17.879641   9508400
2008-11-24  23.700001  24.670000  23.610001  24.059999  18.235878   6284000
2008-11-25  24.690001  24.840000  24.020000  24.219999  18.357149   6164700
2008-11-26  23.760000  24.719999  23.760000  24.600000  18.645163   6608700
2008-11-28  24.740000  25.139999  24.540001  24.920000  18.887701   1595100
2008-12-01  24.799999  24.799999  23.440001  23.639999  17.917551   5771700
2008-12-02  23.910000  24.290001  23.580000  24.250000  18.379889   7442500
2008-12-03  23.959999  25.030001  23.770000  24.830000  18.819494  10260400
2008-12-04  24.540001  24.959999  24.110001  24.360001  18.463263   8388100
2008-12-05  23.879999  25.340000  23.740000  25.190001  19.092348   8018200
2008-12-08  25.940001  25.940001  25.129999  25.389999  19.243935   9152900
2008-12-09  25.260000  25.459999  24.740000  24.990000  18.940750   8474900
2008-12-10  25.209999  25.290001  24.760000  24.920000  18.887701   7121300

                  RSI Signal
Date
2008-10-01        NaN   Hold
2008-10-02        NaN   Hold
2008-10-03        NaN   Hold
2008-10-06        NaN   Hold
2008-10-07        NaN   Hold
2008-10-08        NaN   Hold
2008-10-09        NaN   Hold
2008-10-10        NaN   Hold
2008-10-13        NaN   Hold
2008-10-14        NaN   Hold
2008-10-15        NaN   Hold
2008-10-16        NaN   Hold
2008-10-17        NaN   Hold
```

```
2008-10-20  39.541174   Hold
2008-10-21  39.328893   Hold
2008-10-22  37.435575   Hold
2008-10-23  37.532326   Hold
2008-10-24  40.179817   Hold
2008-10-27  39.071982   Hold
2008-10-28  46.231205   Hold
2008-10-29  52.571428   Hold
2008-10-30  59.036999   Hold
2008-10-31  47.879293   Hold
2008-11-03  46.101333   Hold
2008-11-04  64.222397   Hold
2008-11-05  53.922974   Hold
2008-11-06  50.000000   Hold
2008-11-07  44.736960   Hold
2008-11-10  44.028744   Hold
2008-11-11  45.333317   Hold
2008-11-12  45.603591   Hold
2008-11-13  55.332432   Hold
2008-11-14  55.964455   Hold
2008-11-17  42.245217   Hold
2008-11-18  44.822508   Hold
2008-11-19  33.751804   Hold
2008-11-20  30.612349   Hold
2008-11-21  29.074281    Buy
2008-11-24  30.166891   Hold
2008-11-25  33.245028   Hold
2008-11-26  39.697866   Hold
2008-11-28  39.614747   Hold
2008-12-01  34.449746   Hold
2008-12-02  41.058836   Hold
2008-12-03  47.826120   Hold
2008-12-04  35.648795   Hold
2008-12-05  47.651026   Hold
2008-12-08  52.011101   Hold
2008-12-09  48.118234   Hold
2008-12-10  55.161751   Hold
```

[ ]:

```
# collect the signals as dataframes based on the period
rsi_signals =␣
 ↪collect_signals(sector_etf_data,economic_cycle_periods_list,sector_etf_tickers)
```

[ ]:

```
# adjusted close price dataframe
adj_close_sector_etf␣
 ↪=get_adjusted_closed_price(sector_etf_data,sector_etf_tickers,economic_cycle_periods_list)
```

[ ]:

```
import pandas as pd
import numpy as np
from datetime import timedelta
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

def portfolio_investment(bb_signals_nd, adj_close_nd, periods_date,␣
 ↪periods_list, tickers, n_sample, initial_investment, percent_to_buy,␣
 ↪percent_to_sell):
    # Track actions day by day for the entire portfolio
    portfolio_tracker = {period: pd.DataFrame(columns=['Date', 'Account␣
 ↪Balance', 'Portfolio Value', 'Total Value', 'Profit', 'Sector Allocation'])␣
 ↪for period in periods_list}

    # Portfolio summary - nested dictionary for each period and ticker
    portfolio_summary = {period: {ticker: pd.DataFrame() for ticker in tickers}␣
 ↪for period in periods_list}

    # Set data to be accessed
    adj_close_data = adj_close_nd
    bollinger_band_data = bb_signals_nd

    all_data = {
        'Stock Tracker': portfolio_summary,
        'Portfolio Tracker': portfolio_tracker,
        'Adjusted Close Price': adj_close_nd,
        'Bollinger Band Signal': bollinger_band_data
    }

    # Loop through each economic period
    for period in periods_list:
        # Create the date range for the current period
        date_range = pd.date_range(start=pd.
 ↪to_datetime(periods_date[period][0]), end=pd.
 ↪to_datetime(periods_date[period][1]) - timedelta(days=90))
        # Get random dates for stochastic modeling
        start_dates = np.random.choice(date_range, size=n_sample, replace=False)

        # Loop through sampled start dates
        for start_date in start_dates:
            time_stamp = pd.to_datetime(start_date)

            # Initialize balance for portfolio investment
            account_balance = initial_investment
            shares_number = {ticker: 0 for ticker in tickers}  # Initialize␣
 ↪share count for each ticker
```

```
            # Extract the adjusted close and signal data for time period
            adj_close_period = adj_close_data[period].loc[time_stamp:time_stamp + timedelta(days=90)]
            bb_signals_period = bollinger_band_data[period].loc[time_stamp:time_stamp + timedelta(days=90)]

            # Iterate over each row in the Bollinger Band signals (day by day)
            for row_idx, row in bb_signals_period.iterrows():
                daily_balance_change = 0
                portfolio_value = 0

                # Initialize tracking for each ticker
                for col_idx, signal in enumerate(row):
                    ticker = tickers[col_idx]  # Correctly get ticker for each column
                    adj_close_price = adj_close_period.loc[row_idx, ticker]  # Get corresponding adjusted close price

                    # Initialize stock tracker for current ticker
                    stock_tracker = all_data['Stock Tracker'][period][ticker]

                    # Handle Buy action
                    if signal == 'Buy':
                        amount_to_buy = percent_to_buy * account_balance
                        if account_balance >= amount_to_buy:
                            # Calculate shares to buy
                            shares_to_buy = amount_to_buy / adj_close_price
                            shares_number[ticker] += shares_to_buy

                            # Track investment for the current period
                            stock_tracker = stock_tracker.append({
                                'Date': row_idx,
                                'Share Price': adj_close_price,
                                'Signal': 'Buy',
                                'Buy/Sell Amount ($)': amount_to_buy,
                                'Buy/Sell Number of Shares': shares_to_buy,
                                'Shares ($) Ownership': shares_number[ticker] * adj_close_price,  # Update based on current price
                                'Shares Ownership': shares_number[ticker]
                            }, ignore_index=True)

                            # Update account balance after buying
                            account_balance -= amount_to_buy

                    # Handle Sell action
                    elif signal == 'Sell':
                        if shares_number[ticker] > 0:  # Ensure we have shares␣
↪to sell
                            amount_to_sell = percent_to_sell *␣
↪(shares_number[ticker] * adj_close_price)
                            shares_to_sell = amount_to_sell / adj_close_price
                            if shares_number[ticker] >= shares_to_sell:
                                shares_number[ticker] -= shares_to_sell

                                # Track the sell action
                                stock_tracker = stock_tracker.append({
                                    'Date': row_idx,
                                    'Share Price': adj_close_price,
                                    'Signal': 'Sell',
                                    'Buy/Sell Amount ($)': amount_to_sell,
                                    'Buy/Sell Number of Shares': shares_to_sell,
                                    'Shares ($) Ownership':␣
↪shares_number[ticker] * adj_close_price,  # Update based on current price
                                    'Shares Ownership': shares_number[ticker]
                                }, ignore_index=True)

                                # Update account balance after selling
                                account_balance += amount_to_sell

                    # Handle Hold action (no action taken)
                    else:
                        # Track the hold state
                        stock_tracker = stock_tracker.append({
                            'Date': row_idx,
                            'Share Price': adj_close_price,
                            'Signal': 'Hold',
                            'Buy/Sell Amount ($)': 0,
                            'Buy/Sell Number of Shares': 0,
                            'Shares ($) Ownership': shares_number[ticker] *␣
↪adj_close_price,  # Update based on current price
                            'Shares Ownership': shares_number[ticker]
                        }, ignore_index=True)

                    # Save the updated tracker back to portfolio summary
                    all_data['Stock Tracker'][period][ticker] = stock_tracker.
↪copy()

                # Calculate total portfolio value for all tickers for the day
                portfolio_value = sum(shares_number[ticker] * adj_close_period.
↪loc[row_idx, ticker] for ticker in tickers)

                # Total value (account balance + portfolio value)
                total_value = account_balance + portfolio_value

                # Calculate profit (difference from initial investment)
                profit = total_value - initial_investment

                # Calculate percentage allocation of each ticker to total␣
 ↪portfolio value
                sector_allocation = {ticker: (shares_number[ticker] *␣
 ↪adj_close_period.loc[row_idx, ticker]) / portfolio_value * 100 if␣
 ↪portfolio_value > 0 else 0 for ticker in tickers}

                # Track portfolio changes for the current day
                portfolio_tracker[period] = portfolio_tracker[period].append({
                    'Date': row_idx,
                    'Account Balance': account_balance,
                    'Portfolio Value': portfolio_value,
                    'Total Value': total_value,
                    'Profit': profit,
                    'Sector Allocation': sector_allocation
                }, ignore_index=True)

            # Update the portfolio tracker for the period
            all_data['Portfolio Tracker'][period] = portfolio_tracker[period]

    # Return the complete portfolio summary for all periods and tickers
    return all_data
```

```
[ ]: rsi_investment =␣
      ↪portfolio_investment(rsi_signals,adj_close_sector_etf,economic_cycle_periods,economic_cycle
      ↪05,0.20)
```

```
[ ]: rsi_investment['Portfolio Tracker']['peak']
```

```
[ ]:           Date  Account Balance  Portfolio Value    Total Value       Profit  \
     0   2019-08-07     90250.000000      9750.000000  100000.000000     0.000000
     1   2019-08-08     90250.000000      9983.013220  100233.013220   233.013220
     2   2019-08-09     85737.500000     14393.743675  100131.243675   131.243675
     3   2019-08-12     77378.093750     22569.054648   99947.148398   -52.851602
     4   2019-08-13     73509.189062     26694.241455  100203.430517   203.430517
     ..         …              …               …              …            …
     59  2019-10-30     65717.663822     40984.365851  106702.029673  6702.029673
     60  2019-10-31     66461.240933     40044.940264  106506.181198  6506.181198
     61  2019-11-01     69537.126187     37577.844840  107114.971027  7114.971027
     62  2019-11-04     72020.361454     35619.971176  107640.332630  7640.332630
     63  2019-11-05     76293.807572     31382.264723  107676.072295  7676.072295

                                        Sector Allocation
```

```
0   {'XLB': 0.0, 'XLI': 0.0, 'XLF': 0.0, 'XLK': 0…
1   {'XLB': 0.0, 'XLI': 0.0, 'XLF': 0.0, 'XLK': 0…
2   {'XLB': 0.0, 'XLI': 0.0, 'XLF': 0.0, 'XLK': 0…
3   {'XLB': 0.0, 'XLI': 18.99448189986576, 'XLF': …
4   {'XLB': 0.0, 'XLI': 16.251333064864813, 'XLF':…
..                                                …
59  {'XLB': 11.282917435244947, 'XLI': 18.67610187…
60  {'XLB': 11.418102414953877, 'XLI': 18.90949782…
61  {'XLB': 9.871360526590884, 'XLI': 16.471894153…
62  {'XLB': 8.396099731183936, 'XLI': 14.062989270…
63  {'XLB': 7.636714216574952, 'XLI': 12.796340417…

[64 rows x 6 columns]
```

```
[ ]:
```