Senior Capstone Project: Feature Engineering

Feature engineering is the next step following an exploratory data analysis (EDA). This is because the EDA has given knowledge to the data analyst/data scientist who now should better understand what data is important and requires deeper attention. Ensuring detail in this part of the project is fundamental to reduce the possibility of errors to come up in the future.

In my capstone project, I will be using feature engineering in the following ways:

Macroeconomic cycle creation: By utilizing backward reflection on different time periods, you can label the time period of investment with different economic cycles. These cycles reflect different aggregate demand and overall economic activity. For example, during a trough, there is less activity and lower returns or even negative returns are expected.

These periods are created using the following code

```
# 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
```

This utilizes a function 'get_data_from_start_to_end' which simplest get the ticker candlestick data. Using a nested dictionary you are able to track different time periods with different equities so that a greater insight can be made on the performance of equities during different macroeconomic time periods.

Technical Analysis Indicators

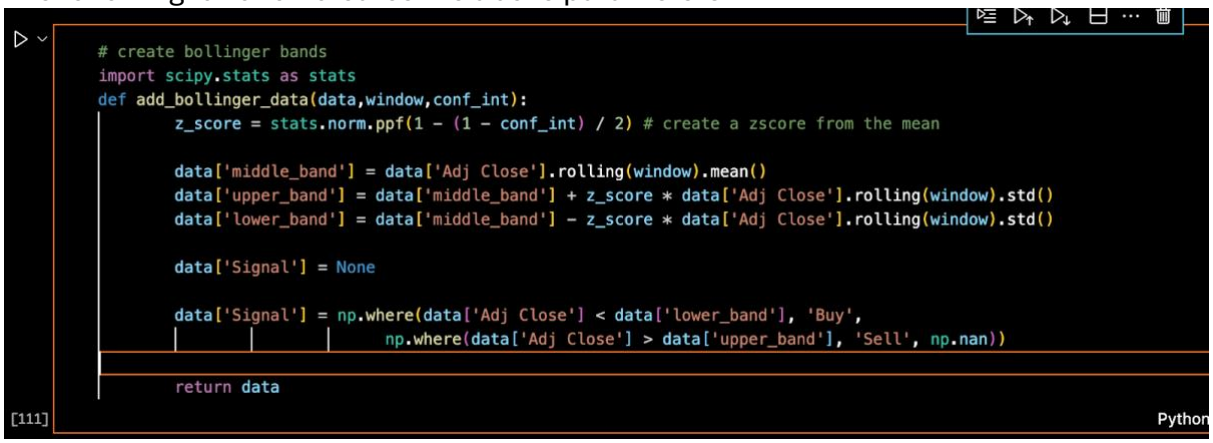
This project utilizes technical analysis which is meant to identify different components of candlestick stock data such as momentum, support and resistance which lead to different actions. The following are the different features that will be created from the nested dictionary data.

Bollinger Bands

Bollinger bands created a channel where 2 standard deviations above and below the rolling mean average suggests where you would expect the movement of the data to lie in. If it goes below the lower band it suggests a buy signal as it is being oversold and if it goes above the band it will represent a sell signal.

- Lower_band
- Upper_band
- Middle_band
- Signal

The following function creates the above parameters



```python
# create bollinger bands
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
```

[111] Python

Relative Strength Index

The relative strength index is a technical analysis technique which identifies momentum to identify if a stock is over or undervalued. It is on a scale of 100 and can be used as another way of indicating the stock to buy or sell.

RSI is calculated

α

$$RS = \frac{Avg.Gain}{Avg.Loss}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

by

If the RSI falls below an RSI of 30 it suggests a buy signals, if it goes above 70 then it suggests a sell signal.

- RSI
- Signal

The following function can create the above functions

```python
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
```

✓ 0.0s Python

```python
# 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'])

    return nested_dict
```

✓ 0.0s Python

Moving Average Conversion Diversion

The Moving Average Conversion Diversion (MACD) is another popular technique which concentrates on the momentum of a stock. It has two lines which oscillate between one another.

The first is the MACD line which is a long term EMA (26 days) / a short term EMA (12 days).
The second line is the signal line where a 9 day EMA of the MACD line is observed.

It serves to find bullish and bearish signals.

When the MACD line crosses the above signal line it suggests upward momentum (buy signal)
When the MACD line crosses below the signal line it suggests a downward momentum (sell signal)
Divergence is when the price action diverging from the MACD can signal potential reverses.

- Short EMA
- Long EMA
- MACD Line
- Signal Line
- Signal

```python
def macd_components(nested_dict,periods,tickers):
    for period in periods:
        for ticker in tickers:
            # get the short and long ema
            nested_dict[period][ticker]['short_ema'] = nested_dict[period][ticker]['Close'].ewm(span=12, adjust=False
            nested_dict[period][ticker]['long_ema'] = nested_dict[period][ticker]['Close'].ewm(span=26, adjust=False)

            # create the MACD line
            nested_dict[period][ticker]['macd_line'] = nested_dict[period][ticker]['short_ema'] - nested_dict[period]
            nested_dict[period][ticker]['signal_line'] = nested_dict[period][ticker]['macd_line'].ewm(span=9, adjust=

    return nested_dict
```

✓ 0.0s Python

```python
def add_macd_signals(nested_dict,periods, tickers, short_window=12, long_window=26, signal_window=9):
    """
    Function to calculate MACD, Signal Line, and generate Buy/Sell/Hold signals based on crossovers.
    """
    for period in periods:
        for ticker in tickers:

            data = nested_dict[period][ticker]

            # Generate 'Buy', 'Sell', 'Hold' signals using np.where() based on crossovers
            data['Signal'] = np.where(data['macd_line'] > data['signal_line'], 'Buy',
                                      np.where(data['macd_line'] < data['signal_line'], 'Sell', 'Hold'))
```

✓ 0.0s Python

Ichimoku Cloud

The ichimoku cloud is an all in one indicator which looks to take into account momentum, support and resistance and is made up of 5 lines which need to be constructed.

- The conversion line is meant to represent short term trends
- The baseline which evaluates the strength of a medium term trend it lags the price
- Leading span A is used to identify the future of support and resistance
- Leading span B is the line used to identify future support and resistance
- Lagging span showsp possible support and resistance
- Cloud space between A and B represents divergence in price evolution

```python
def ichimoku_components(nested_dict, periods, tickers):
    for period in periods:
        for ticker in tickers:
            # Create parameters
            df_ticker = nested_dict[period][ticker]

            # Create conversion line
            hi_val = df_ticker['High'].
            low_val = df_ticker['Low'].rolling(window=9).min()
            df_ticker['Conversion'] = (hi_val + low_val) / 2

            # Create baseline
            hi_val2 = df_ticker['High'].rolling(window=26).max()
            low_val2 = df_ticker['Low'].rolling(window=26).min()
            df_ticker['Baseline'] = (hi_val2 + low_val2) / 2

            # Create Leading Span A (Senkou Span A)
            df_ticker['Leading Span A'] = ((df_ticker['Conversion'] + df_ticker['Baseline']) / 2).shift(26)

            # Create Leading Span B (Senkou Span B)
            hi_val3 = df_ticker['High'].rolling(window=52).max()
            low_val3 = df_ticker['Low'].rolling(window=52).min()
            df_ticker['Leading Span B'] = ((hi_val3 + low_val3) / 2).shift(26)

            # Create Lagging Span (Chikou Span)
            df_ticker['Lagging Span'] = df_ticker['Close'].shift(-26)
```

(function) rolling: Any

Python

```python
def ichimoku_signal_with_labels(nested_dict, periods, tickers):
    for period in periods:
        for ticker in tickers:
            # Create parameters
            df_ticker = nested_dict[period][ticker]

            # Initialize the Signal column with 'Hold'
            df_ticker['Signal'] = 'Hold'

            # Generate buy signals
            buy_condition = (
                (df_ticker['Conversion'].shift(1) < df_ticker['Baseline'].shift(1)) &
                (df_ticker['Conversion'] > df_ticker['Baseline'])
            )

            # Generate sell signals
            sell_condition = (
                (df_ticker['Conversion'].shift(1) > df_ticker['Baseline'].shift(1)) &
                (df_ticker['Conversion'] < df_ticker['Baseline'])
            )

            # Assign 'Buy' and 'Sell' based on the conditions
            df_ticker.loc[buy_condition, 'Signal'] = 'Buy'
            df_ticker.loc[sell_condition, 'Signal'] = 'Sell'
```

Python

Data Splitting has been done through the creation of macroeconomic cycles. This is then going to be used as a parameter when going to test the success of Bollinger Bands as an investment strategy.