GROUP WORK PROJECT 2 MScFE 660: RISK MANAGEMENT
GROUP NUMBER: 3986

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Harshil Sumra	India	harshilsumra19972gmail.com	
Gaurav Srivastava	India	gauravsriitk@gmail.com	
Sunil Kumar Sharma	USA	sunilksh+mscfe@outlook.com	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).

Team member 1	Harshil Sumra
Team member 2	Gaurav Srivastava
Team member 3	Sunil Kumar Sharma

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.
Note: You may be required to provide proof of your outreach to non-contributing members upon request.

Section 1 – Introduction

This is the second part of the three-part mini capstone project based on Danish A. Alvi's paper "Application of Probabilistic Graphical Models in Forecasting Crude Oil Price" [1]. Here, our focus will be on the development of the methodology and the model using which we can do the same in our case.
The paper is oriented as follows:

a. Section 1 – Introduction
b. Section 2 - Theory of Hidden Markov Models (answer to steps 1 and 3)
c. Section 3 - Regime Detection (Step 2)
d. Section 4 – Comprehensive Approach (answer to steps 4, 5, 6, 7, 8 and 9)
e. Section 5 - References

Section 2 - Theory of Hidden Markov Models (HMMs)

To understand HMM, we first need to understand "markov process". A "markov process" depicts a sequence of events where the outcome of each event depends only on the current state or condition and is independent of past events, also known as Memorylessness property. Common application is where the focus is on modeling the dynamics of hidden states without generating observable data.

HMMs are a class of statistical models like markov processes but here the focus is on modelling sequences of observable data with hidden states. A Hidden Markov Model (HMM) enables us to engage in a dual discourse, one involving the observed elements, such as input words, and the other concerning concealed elements, such as part-of-speech tags, which are believed to play a pivotal role in determining causality within its probabilistic framework. The HMM is defined by the following constituent elements:

Component [16]	Mathematical Notation [15]	Description [15]
Number of states in the model	$Q = q_1, q_2, \dots q_n$	a set of N states
Number of distinct observations	$A = a_{11} \dots a_{ij} \dots a_{NN}$	a transition probability matrix A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \ \forall i$
State transition model	$O = o_1, o_2, \dots o_T$	a sequence of T observations, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
Observation model	$B = b_i(o_t)$	a sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation o_t being generated from a state i
Initial state distribution	$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Table 1 - Characteristic Elements of a Hidden Markov Model

So, in conclusion, the above five constituent elements help us define an HMM. According to our reference paper [1], there are several issues with utilizing HMMs properly, which are:

a. The Likelihood Challenge: we seek to ascertain the likelihood, represented as $P(O \mid \lambda)$, assuming the presence of a Hidden Markov Model denoted as $\lambda = (A, B)$, and a sequence of observations labeled as O.
The issue is resolved via implementation of forward and backward iterative algorithms.

b. The Deciphering Challenge: In a decoding context, the objective is to deduce the optimal sequence of hidden states, denoted as Q, given the existence of a Hidden Markov Model represented as $\lambda = (A, B)$, and a sequence of observations designated as O.
The issue is resolved via implementation of Viterbi algorithm.

c. The Learning Endeavor: In a learning context, the goal is to unveil the parameters of the Hidden Markov Model, specifically denoted as A and B, under the premise of having a sequence of observations labeled as O, and a set of Hidden Markov Model states.
The issue is resolved via implementation of Baum-Welch algorithm.

Forward algorithm:

The forward algorithm is a fundamental algorithm used in Hidden Markov Models (HMMs) to calculate the probability of observing a particular sequence of observations given the model. It's used for various applications, including speech recognition, bioinformatics, and natural language processing. The forward algorithm essentially computes the likelihood of a sequence of observations and is a key component of many HMM-based tasks, including training and decoding.

Three steps of Forward algorithm
Initialization: Initialize the forward probabilities, often denoted as $\alpha_{(i, t)}$, where i represents the hidden state and t represents the time step (observation sequence index).
Set $\alpha_{(i, 1)}$ to the initial state probabilities times the emission probabilities for the first observation:

$$\alpha_{(i, 1)} = \pi_{(i)} * B_{(i, O1)}$$

where $\pi_{(i)}$ is the initial state probability for state i, and $B_{(i, O1)}$ is the emission probability for state i emitting the first observation, O_1.
Recursion: For each time step t from 2 to T (the length of the observation sequence):
For each hidden state i:
Compute $\alpha_{(i, t)}$ by summing over all possible transitions from the previous time step j and multiplying by the emission probability for state i emitting the observation at time t:

$$\alpha_{(i, t)} = \sum_{[j=1 \text{ to } N]} \alpha_{(j, t-1)} * A_{(j, i)} * B_{(i, Ot)}$$

where $A_{(j, i)}$ is the transition probability from state j to state i, and Ot is the observation at time t.

Termination: Calculate the likelihood of the observation sequence as the sum of the forward probabilities for all states at the final time step T:
$P(O|\lambda) = \sum_{[i=1 \text{ to } N]} \alpha(i, T)$, where N is the number of hidden states in the HMM.

The result, $P(O|\lambda)$, represents the probability of observing the given sequence of observations O, given the HMM model λ (which includes the initial state probabilities, transition probabilities, and emission probabilities).

Python pseudocode

```
# Define HMM parameters
initial_probabilities = [...]  # Initial state probabilities
transition_matrix = [...]      # Transition probabilities
emission_matrix = [...]        # Emission probabilities
observation_sequence = [...]    # The sequence of observations

# Initialize the forward probabilities matrix
T = len(observation_sequence)   # Length of the observation sequence
N = len(initial_probabilities)  # Number of hidden states
forward_probabilities = [[0.0] * N for _ in range(T)]

# Initialization step
for i in range(N):
    forward_probabilities[0][i] = initial_probabilities[i] * emission_matrix[i][observation_sequence[0]]

# Recursion step
for t in range(1, T):
    for i in range(N):
        forward_probabilities[t][i] = 0.0
        for j in range(N):
            forward_probabilities[t][i] += forward_probabilities[t-1][j] * transition_matrix[j][i]
        forward_probabilities[t][i] *= emission_matrix[i][observation_sequence[t]]

# Termination step
probability_of_observation = sum(forward_probabilities[T-1])
```

Backward algorithm:

The backward algorithm is another fundamental algorithm used in Hidden Markov Models (HMMs), just like the forward algorithm. While the forward algorithm computes the probability of observing a sequence from the initial state to a particular point in time, the backward algorithm computes the probability of observing the remaining part of the sequence from a particular state at a specific point in time.

Three steps of Backward algorithm

Initialization: Initialize the backward probabilities, often denoted as $\beta_{(i,\,t)}$, where i represents the hidden state and t represents the time step (observation sequence index). Set $\beta_{(i,\,T)}$ to 1 for all states i at the final time step T (the length of the observation sequence).

Recursion: For each time step t from T-1 down to 1:

For each hidden state i:

Compute $\beta_{(i,\,t)}$ by summing over all possible transitions to the next time step j and multiplying by the emission probability for state j emitting the observation at time t+1:

$$\beta_{(i,\,t)} = \sum_{[j=1 \text{ to } N]} (A_{(i,\,j)} * B_{(j,\,Ot+1)} * \beta_{(j,\,t+1)})$$

where A(i, j) is the transition probability from state i to state j, $B_{(j,\,Ot+1)}$ is the emission probability for state j emitting the observation at time t+1, and $\beta_{(j,\,t+1)}$ is the backward probability for state j at the next time step t+1.

Termination: Calculate the likelihood of the observation sequence as the sum of the product of the initial state probabilities, emission probabilities for the initial observation, and the corresponding backward probabilities for each state at the initial time step 1:

$$P(O|\lambda) = \sum_{[i=1 \text{ to } N]} (\pi_{(i)} * B_{(i,\,O1)} * \beta_{(i,\,1)})$$

where $\pi(i)$ is the initial state probability for state i, $B_{(i, O1)}$ is the emission probability for state i emitting the first observation, and $\beta_{(i, 1)}$ is the backward probability for state i at time step 1.
The backward algorithm is crucial in various HMM-based tasks, such as parameter estimation (Baum-Welch algorithm), where it's used to compute the expected count of transitions from state i to state j given the observed sequence. It's also used in certain decoding algorithms, such as the Forward-Backward algorithm, to calculate posterior probabilities of states at each time step.
Python pseudocode

```
# Define HMM parameters
initial_probabilities = [...]  # Initial state probabilities
transition_matrix = [...]      # Transition probabilities
emission_matrix = [...]        # Emission probabilities
observation_sequence = [...]    # The sequence of observations

# Initialize the backward probabilities matrix
T = len(observation_sequence)   # Length of the observation sequence
N = len(initial_probabilities)  # Number of hidden states
backward_probabilities = [[0.0] * N for _ in range(T)]

# Initialization step
for i in range(N):
    backward_probabilities[T-1][i] = 1.0

# Recursion step
for t in range(T-2, -1, -1):
    for i in range(N):
        backward_probabilities[t][i] = 0.0
        for j in range(N):
            backward_probabilities[t][i] += transition_matrix[i][j] * emission_matrix[j][observation_sequence[t+1]] * backward_probabilities[t+1][j]

# Termination step
probability_of_observation = sum(initial_probabilities[i] * emission_matrix[i][observation_sequence[0]] * backward_probabilities[0][i] for i in range(N))

# The variable 'probability_of_observation' now contains the likelihood of the observation sequence
```

Viterbi algorithm

The Viterbi algorithm is a dynamic programming algorithm used in Hidden Markov Models (HMMs) for finding the most likely sequence of hidden states (state path) that generated a given sequence of observations. It is particularly useful in sequence labeling and pattern recognition tasks, such as part-of-speech tagging, speech recognition, and gene prediction in bioinformatics. The algorithm employs a max-product approach during the forward pass, replacing the sum-product method, and in the backward pass, it traces back to determine the most likely sequence from time 1 to t by recursively identifying the most probable states.

Three steps of Viterbi algorithm:

Initialization: Initialize the Viterbi probabilities, often denoted as $\delta_{(i, t)}$, where i represents the hidden state and t represents the time step (observation sequence index).
Set $\delta_{(i, 1)}$ to the product of the initial state probabilities and the emission probabilities for the first observation:

$$\delta_{(i, 1)} = \pi_{(i)} * B_{(i, O1)}$$

where $\pi(i)$ is the initial state probability for state i, and $B_{(i, O1)}$ is the emission probability for state i emitting the first observation, O1.
Initialize a backpointer table, often denoted as $\psi_{(i, t)}$, which will be used to keep track of the best path.

Recursion: For each time step t from 2 to T (the length of the observation sequence):
For each hidden state i:
Compute $\delta_{(i, t)}$ as the maximum over all possible transitions from the previous time step j, multiplied by the emission probability for state i emitting the observation at time t:

$$\delta_{(i, t)} = \max[\delta_{(j, t-1)} * A_{(j, i)} * B_{(i, Ot)}]$$

where $A_{(j, i)}$ is the transition probability from state j to state i, and Ot is the observation at time t.
Update the backpointer $\psi_{(i, t)}$ to store the index j that maximized the above equation.
Termination: Calculate the highest probability for the most likely state sequence:

$$P^* = \max[\delta_{(i, T)}]$$

where i ranges over all possible states.
Initialize the state sequence by backtracking through the backpointer table:
Set the last state in the sequence as the one that maximized

$$\delta_{(i, T)}: S(T) = \text{argmax}[\delta_{(i, T)}].$$

For each previous time step t from T-1 down to 1:
Set

$$S_{(t)} = \psi(S_{(t+1)}, t+1).$$

The result is the most likely sequence of hidden states, S(1), S(2), ..., S(T), that generated the observed sequence O, given the HMM model λ.

Pseudocode

```
# Define HMM parameters
initial_probabilities = [...]  # Initial state probabilities
transition_matrix = [...]       # Transition probabilities
emission_matrix = [...]         # Emission probabilities
observation_sequence = [...]     # The sequence of observations

# Initialize the Viterbi path and Viterbi probabilities matrices
T = len(observation_sequence)   # Length of the observation sequence
N = len(initial_probabilities)  # Number of hidden states
viterbi_path = [-1] * T        # Initialize the Viterbi path
viterbi_probabilities = [[0.0] * N for _ in range(T)]

# Initialization step
for i in range(N):
    viterbi_probabilities[0][i] = initial_probabilities[i] * emission_matrix[i][observation_sequence[0]]
    viterbi_path[0] = -1  # Initialize the first state in the path as -1
```

```
# Recursion step
for t in range(1, T):
    for i in range(N):
        max_prob = 0.0
        max_state = -1
        for j in range(N):
            prob = viterbi_probabilities[t - 1][j] * transition_matrix[j][i]
            if prob > max_prob:
                max_prob = prob
                max_state = j
        viterbi_probabilities[t][i] = max_prob * emission_matrix[i][observation_sequence[t]]
        viterbi_path[t] = max_state

# Termination step
max_final_prob = max(viterbi_probabilities[T - 1])
final_state = viterbi_probabilities[T - 1].index(max_final_prob)

# Backtrack to find the best path
best_path = [final_state]
for t in range(T - 1, 0, -1):
    best_path.insert(0, viterbi_path[t])

# The variable 'best_path' now contains the most likely sequence of hidden states
```

Baum-Welch algorithm

The Baum-Welch algorithm, also known as the Forward-Backward algorithm or the Expectation-Maximization (EM) algorithm for Hidden Markov Models (HMMs), is used for estimating the parameters of an HMM from a given set of observations. These parameters include the initial state probabilities, transition probabilities, and emission probabilities.

Steps of Baum-Welch algorithm:

Initialization: Initialize the HMM with arbitrary parameters. This initialization can be done randomly or using some prior knowledge.

Expectation (E-step): Calculate the forward probabilities (α) using the forward algorithm for the given observations and the current model parameters. Then calculate the backward probabilities (β) using the backward algorithm for the same observations and current model parameters.

Estimation: Update the model parameters using the calculated forward and backward probabilities:

Initial state probabilities (π): Estimate the new initial state probabilities as the normalized state occupation probabilities at the first time step:

$$\pi_{(i)} = \alpha_{(i,\,1)} * \beta_{(i,\,1)} / P(O|\lambda)$$

where i represents a hidden state.

Transition probabilities (A): Estimate the new transition probabilities as the normalized expected number of transitions from state i to state j over all time steps:

$$A_{(i,j)} = \Sigma_{[t=1 \text{ to } T-1]} [\alpha_{(i,t)} * A_{(i,j)} * B_{(j,O(t+1))} * \beta_{(j,t+1)}] / \Sigma_{[t=1 \text{ to } T-1]} [\alpha_{(i,t)} * \beta_{(i,t)} / P(O|\lambda)]$$

where i and j represent hidden states.
Emission probabilities (B): Estimate the new emission probabilities as the normalized expected number of times state i emits observation symbol k over all time steps:

$$B_{(i,k)} = \Sigma_{[t=1 \text{ to } T]} [\alpha_{(i,t)} * \beta_{(i,t)} * \delta_{(O(t),k)}] / \Sigma_{[t=1 \text{ to } T]} [\alpha_{(i,t)} * \beta_{(i,t)} / P(O|\lambda)]$$

where i represents a hidden state, k represents an observation symbol, and $\delta(x, y)$ is the Kronecker delta function (1 if x=y, 0 otherwise).

Iteration: Repeat the E-step and Estimation steps until convergence is reached. Convergence can be determined by monitoring the change in the log-likelihood of the data or by a predefined number of iterations.

Final Parameters: The final estimated model parameters (π, A, B) represent the maximum likelihood estimates of the HMM parameters given the observed data.

The Baum-Welch algorithm uses the EM framework to iteratively improve the model parameters until convergence. It's a fundamental tool for training HMMs, especially in cases where the true parameters are unknown.

Pseudocode

```
# Define the HMM model parameters
N = ...  # Number of hidden states
M = ...  # Number of observable symbols
T = ...  # Length of the observation sequence

# Initialize the HMM parameters randomly or with some prior knowledge
pi = [...]  # Initial state probabilities
A = [...]   # Transition probabilities
B = [...]   # Emission probabilities

# Define the observation sequence
O = [...]

# Set convergence criteria (e.g., a tolerance threshold)
tolerance = ...

# Initialize variables for convergence tracking
converged = False
iteration = 0
```

```
while not converged:
    # Initialize variables to store expected counts
    new_pi = [0.0] * N
    new_A = [[0.0] * N for _ in range(N)]
    new_B = [[0.0] * M for _ in range(N)]

    # Perform the E-step: Calculate forward and backward probabilities
    forward_probs = forward_algorithm(pi, A, B, O)
    backward_probs = backward_algorithm(pi, A, B, O)

    # Calculate the scaling factor for numerical stability
    scaling_factors = compute_scaling_factors(forward_probs)

    # Update the expected counts using the forward and backward probabilities
    for t in range(T - 1):
        for i in range(N):
            for j in range(N):
                new_A[i][j] += (forward_probs[t][i] * A[i][j] * B[j][O[t + 1]] * backward_probs[t + 1][j]) / scaling_factors[t]

    for t in range(T):
        for i in range(N):
            if O[t] == observation_symbol:
                new_B[i][O[t]] += (forward_probs[t][i] * backward_probs[t][i]) / scaling_factors[t]

    for i in range(N):
        new_pi[i] += (forward_probs[0][i] * backward_probs[0][i]) / scaling_factors[0]
```

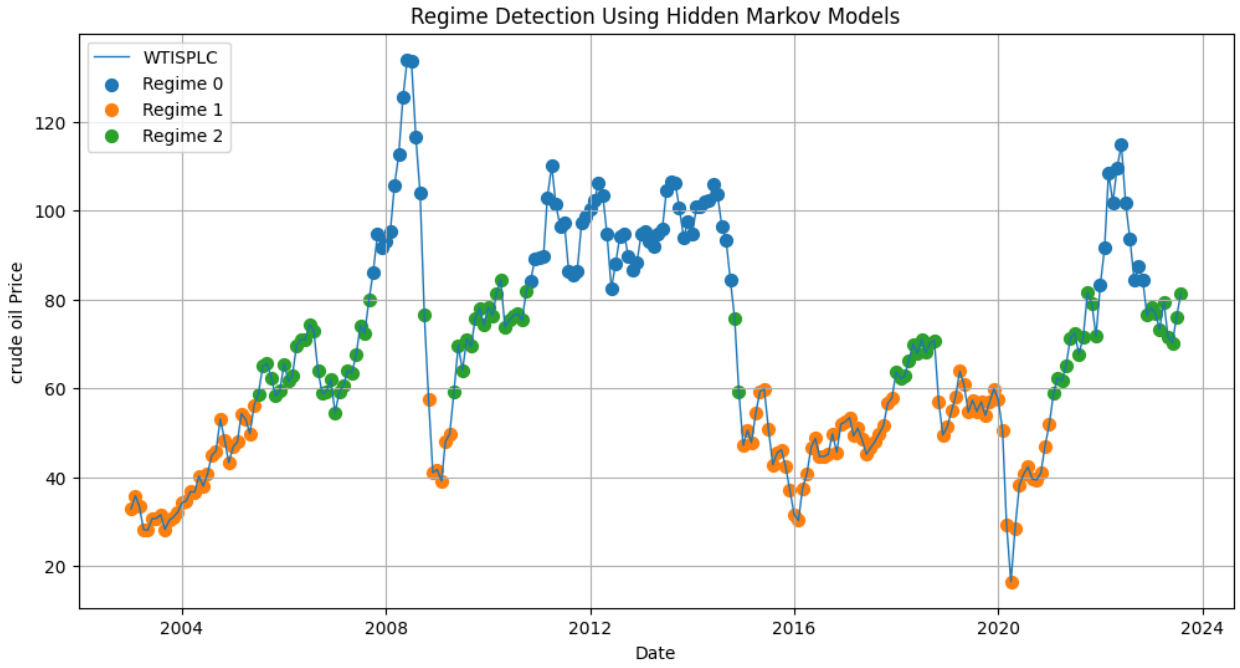
*Note- For continuity, one “for” loop is common between the above and below two snips. Also the toy example and its outputs are shared In the attached code file.

```
    for i in range(N):
        new_pi[i] += (forward_probs[0][i] * backward_probs[0][i]) / scaling_factors[0]

    # Normalize the expected counts to get new parameter estimates
    new_pi = normalize(new_pi)
    new_A = normalize_matrix(new_A)
    new_B = normalize_matrix(new_B)

    # Check for convergence by comparing new and old parameter estimates
    pi_difference = calculate_difference(pi, new_pi)
    A_difference = calculate_matrix_difference(A, new_A)
    B_difference = calculate_matrix_difference(B, new_B)

    if pi_difference < tolerance and A_difference < tolerance and B_difference < tolerance:
        converged = True

    # Update the HMM parameters with the new estimates
    pi = new_pi
    A = new_A
    B = new_B

    # Increment the iteration counter
    iteration += 1

# The variables pi, A, and B now contain the estimated HMM parameters
```

Section 3 – Regime detection

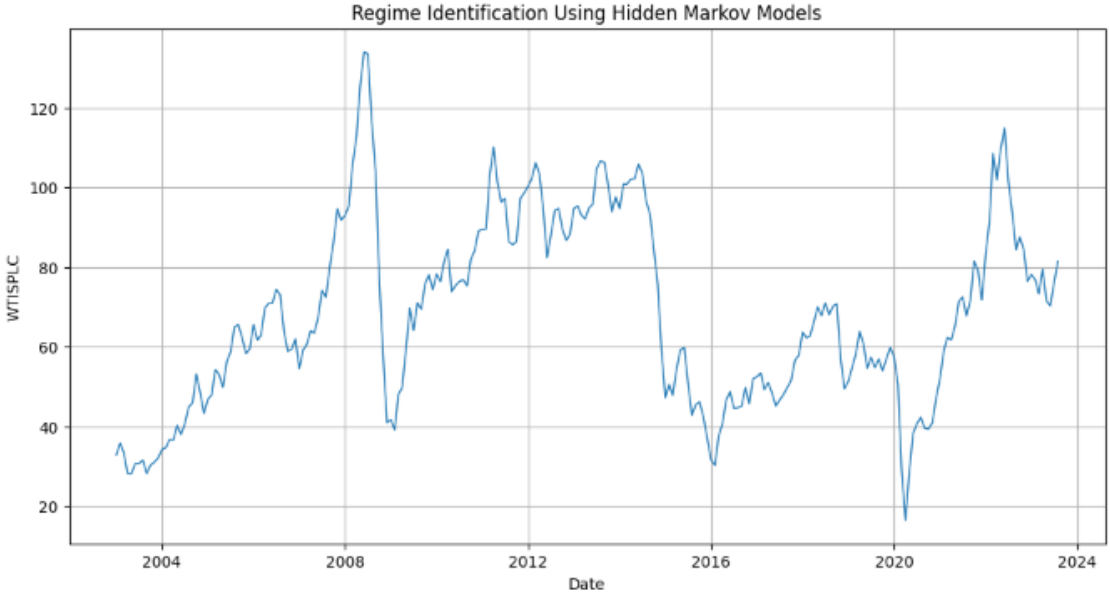
Regime detection involves identifying recurring periods of volatility within a time series, which can encompass hidden states such as bull, bear, or stagnant phases, as well as high or low volatility intervals. These periods are often categorized based on quantitative attributes. Since these volatility patterns are latent and observable only through returns (referred to as emissions), Hidden Markov Models (HMMs) are well-suited for solving the common volatility detection problem. In the context of a discrete-time Hidden Markov Model (HMM), the goal is to detect three primary hidden states: bull, bear, and stagnant states, as described in the subsections below. Utilizing graphical representations of these hidden states enables visualizing and identifying the regimes detected by the HMM.

We used Viterbi algorithm for regime detection and got below conclusion:
Note - The hidden state 0 corresponds to the "Bear" regime, represented in orange, while hidden state 1 represents the "Stagnant" regime in green(rarely seen). Hidden state 2 signifies the "Bull" regime, depicted in blue. In the graph, the green section corresponds to bull regimes, which are periods when crude oil prices consistently rise after a period of decline or stagnation.



We further illustrated how HMMs can be used for regime change identification, the result is as shown below:

```
Regime Change: Date=2005-06-01 00:00:00, Regime=2
Regime Change: Date=2007-09-01 00:00:00, Regime=1
Regime Change: Date=2008-11-01 00:00:00, Regime=0
Regime Change: Date=2009-05-01 00:00:00, Regime=2
Regime Change: Date=2009-10-01 00:00:00, Regime=1
Regime Change: Date=2014-12-01 00:00:00, Regime=0
Regime Change: Date=2017-11-01 00:00:00, Regime=2
Regime Change: Date=2020-02-01 00:00:00, Regime=0
Regime Change: Date=2021-02-01 00:00:00, Regime=2
Regime Change: Date=2021-10-01 00:00:00, Regime=1
```



Section 4 - Comprehensive Approach

The idea here is to utilize Microeconomic and Financial context data along with macroeconomic and geopolitical data to try and develop a more accurate HMM for crude oil price prediction.

Macro Research The first step was conducting macro research to understand the structure and key drivers of the crude oil market. This involved reviewing literature from government agencies like the Energy Information Administration (EIA) and Federal Reserve Bank (FRED) to identify the main macroeconomic factors influencing crude oil prices.

The research found that major drivers include:

- Supply factors like OPEC and non-OPEC oil production
- Demand factors like consumption from OECD and non-OECD countries
- Inventories and stockpiles which act to balance supply and demand
- Economic growth and activity in major economies
- Geopolitics, conflicts, and OPEC policy decisions
- Exchange rates and the value of the US dollar
- Refinery operations, capacity, and utilization

This macro research provided an overview of the crude oil market and key variables to incorporate into the forecasting model.
Indicator Identification Based on the macro research, the next step was identifying specific macroeconomic time series datasets and indicators to include in the model. The main data sources used were EIA and FRED which provide hundreds of free economic indicators.
Key indicator series identified included:
From EIA:
OPEC and non-OPEC oil production
OECD and non-OECD oil consumption
OECD commercial inventories
US refinery inputs and utilization

From FRED:
GDP growth rates for OECD and non-OECD countries
CPI inflation rate
Industrial production indexes
Manufacturing capacity utilization
US dollar foreign exchange rates

In total, 12 indicator time series from EIA and 8 from FRED were identified as useful based on literature review. The macro research guided which specific datasets to retrieve.

Dataset Retrieval With the list of required indicators identified, the next step was retrieving the time series data from the EIA and FRED APIs. Both providers offer Python client libraries that allow querying the indicators and returning Pandas DataFrames.

The EIA API required registering for an API key while the FRED API just needed the indicator series codes. Loops and functions were written to download each identified dataset into a Pandas DataFrame with the column name set to the indicator series code.

This process compiled all the relevant macroeconomic time series data into Python DataFrames for cleansing and analysis.

Data Cleaning We didn't require much clearning for both FRED and EIA data, as it was clean and ready for use. The EIA and FRED DataFrames were combined using Pandas concat into a composite dataset with consistent datetime indexing across all columns.

Finally, the data was split into train, validation, and test sets for use in modeling. The validation and test sets acted as holdout samples for evaluating model performance.

In summary, thorough macro research guided the indicator selection process. Public APIs provided access to the time series data which was cleaned and compiled into a master dataset for analysis. The data retrieval and cleansing were key steps in preparing the inputs for the forecasting models.

Time Series Transformation Process

The datasets obtained from the Energy Information Administration (EIA) and the Federal Reserve Economic Data (FRED) are essentially time series data, spanning from January 1, 2003, to December 31, 2023. To make these time series datasets conducive for analysis, they undergo a meticulous transformation process. This transformation involves dividing the data into three distinct subsets: the training dataset, the validation dataset, and the testing dataset, each of which serves specific purposes.

Dataset Integration: Initially, the data series from both sources are consolidated into a single dataset by merging the corresponding data frames. Notably, the crude oil prices are incorporated into this unified dataset. Two distinct columns are added: one representing the current oil price, and the other indicating the forecasted crude oil price for the subsequent month. The inclusion of the forecasted price serves as a valuable reference for decision-making in trading activities.

Data Slicing: Subsequently, the integrated dataset is divided row-wise according to a predetermined ratio. In this study, the widely accepted ratio of 80:10:10 is employed, where 80% of the data is allocated to the training dataset, 10% is reserved for the validation dataset, and the remaining 10% is designated for the testing dataset.

Data Nature Consideration: It is essential to acknowledge that the collected data is numerical, non-categorical, continuous, and untagged. In contrast, Bayesian belief networks (BBNs) or Bayesian networks (BNs) typically operate with variables characterized by discrete states. Consequently, the data undergoes a transformation process to map it into a more manageable form with a limited number of states. These states primarily encompass market conditions, such as bull, bear, and stagnant markets. This transformation involves the application of Hidden Markov Models (HMMs), particularly suited for detecting hidden states within time series datasets, a process commonly referred to as regime detection.

Regime Detection Process: The 'WTISPL' column, denoting the Spot Crude Oil Price for West Texas Intermediate, is employed in the pursuit of identifying market regimes within the time series data. Initially, this column is used to classify market conditions as either a rise (1) or a decline (0) in oil prices relative to the preceding period (month). The sequence of observed returns, referred to as emissions, is generated as the output of this step.

Parameter Learning with Baum-Welch Algorithm: The subsequent phase involves the application of the Baum-Welch algorithm to facilitate the learning of parameters governing the Hidden Markov Models (HMMs) responsible for generating the data. This intricate process is elaborated upon in the following section, detailing the specifics of parameter learning and its significance in the analysis.

Parameter Learning

In our project, the crucial process of parameter learning is executed through the employment of the Baum-Welch Algorithm. This algorithm, integral to our analysis, aims to enhance the estimation of model probabilities iteratively. However, it is important to recognize that the challenge of local optima may arise during the iterative process, potentially deviating from the desired global optima. Moreover, financial time-series data presents a dynamic characteristic, with parameters subject to variation. Consequently, our approach necessitates training Hidden Markov Models (HMMs) each time new data is introduced, adapting to the evolving nature of financial markets.

Parameter Estimation: Initially, we acquire a comprehensive understanding of the parameters governing the generation of the emission sequence. This exploration into the model's parameters is instrumental in our analysis, providing insights into the inner workings of the Hidden Markov Models (HMMs) involved. This process is achieved through the hmms.print_parameters(dhmm_r) command.

HMM Training and Dataset Construction: Subsequently, we proceed with the training of Hidden Markov Models (HMMs) designed to recognize latent state sequences within the data. As a pivotal step in the analysis, we construct a panda DataFrame encompassing all the variables of interest. This DataFrame, serving as our training dataset, encapsulates the essential components required for further analysis. This approach is systematically applied to the entire training dataset, facilitating the discretization process and ensuring that the model is trained effectively.

Comprehensive Parameter Learning: It is imperative to emphasize that our parameter learning process encompasses the entirety of the Hidden Markov Models (HMMs) and their associated variables. All parameters are meticulously examined and documented to provide a comprehensive understanding of the underlying data generation processes.

Regime Identification Validation: To validate the effectiveness of our Hidden Markov Models (HMMs) in correctly identifying market regimes across all datasets, we employ a regime-switching model approach. By plotting regime-switching models for each dataset, we observe the latent states, namely "bull," "bear," and "stagnant." This visual examination allows us to assess the accuracy of regime identification. While it is important to note that perfect accuracy is not always attainable, our models excel in identifying major regimes with a high degree of accuracy. This capability empowers risk managers to proactively mitigate risks associated with regime changes influenced by external factors, a crucial aspect of financial market analysis.

The paper [1] has four phases for regime detection as follows:

1. Transforming: In this phase, the time series data is transformed into a sequence of emission using the said time series first-order integration with the focus to replace the emissions (returns) with the emission's (return's) parity.
2. Learning: In this phase, the value of π is obtained through the presumed HMM parameters that is producing the time series by means of applying the Baum-Welch algorithm.
3. Finding: In this phase, the Viterbi algorithm is used to find the utmost probable hidden states sequence as the fundamental regimes that is conceivably assumed to generate the observed emissions (returns) sequence.
4. Identifying: In this phase, the hidden connotation surrounding each latent state through indexing by arithmetic mean.

"pgmpy" for model training, testing and validation.

In the context of our analysis using the 'pgmpy' library, we will follow a structured approach involving training, validation, and testing processes.

- Training and Model Parameter Estimation: We will initiate the process by employing the 'Hillclimb Search' Algorithm to estimate model parameters and construct a Bayesian Network. In this phase, we will utilize simulated data in a simplified scenario, employing basic nodes, and utilize the hill climb algorithm as a local search strategy. Subsequently, we will examine the network's edges to determine the best model configuration.
- Data Splitting: After constructing the model, we will partition our dataset into three distinct segments: training, validation, and testing. This segregation will facilitate the assessment of the model's performance and its generalization capabilities.
- Model Training: The Bayesian Network will undergo training using the training dataset. This phase is essential for the model to learn patterns and relationships within the data.
- Performance Evaluation - Validation: We will assess the model's performance by making predictions and quantifying the prediction errors. Initially, our validation results indicate an error rate of 82.86%, signifying room for improvement. Consequently, we will explore parameter adjustments to enhance the model's accuracy.
- Performance Evaluation - Testing: Following the adjustments, we will evaluate the model's performance using the test dataset. Additionally, we will employ data discretization techniques to potentially enhance the model's predictive capabilities.
- Final Performance Assessment: The evaluation process will be repeated on all three datasets: training, validation, and testing. Our analysis will reveal notable improvements, with the error rate decreasing from 82% to 42%, indicating enhanced model performance and predictive accuracy.

We found that the below nodes as important ones in the HMM post validation:

In [14]:
```
# Create a HillClimbSearch object using the df_macro DataFrame
hc = HillClimbSearch(df_macro)

# Perform the Hill Climbing search for the best model with a maximum of 10 iterations
best_model = hc.estimate(max_iter=10)

# Print the edges of the best model found
print(best_model.edges())
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

[('CPIENGSL', 'CUSR0000SEHE'), ('CUSR0000SEHE', 'CAPUTLG211S'), ('PCU3241913241915', 'CPIENGSL'), ('CAPUTLG211S', 'IPG211S'),
('IPG211S', 'INDPRO'), ('INDPRO', 'IPN213111N'), ('INDPRO', 'IPMINE'), ('INDPRO', 'IPG211111CN'), ('INDPRO', 'WTISPLC'), ('IPN2
13111N', 'CAPG211S')]
```

Section 5 - References

1. Alvi, Danish A. Application of Probabilistic Graphical Models in Forecasting Crude Oil Price. 2018. University College London, Dissertation. Link - https://arxiv.org/abs/1804.10869
2. FRED. "Federal Reserve Economic Data | FRED | St. Louis Fed", 14 Jan. 2023, https://fred.stlouisfed.org/
3. Chen, Chin-Ling,Chang, Qingqing,Hu, Jincheng,"Application of Hidden Markov Model in Financial Time Series Data"[2022], https://doi.org/10.1155/2022/1465216
4. Tuyen, Luc. (2013). Markov Financial Model Using Hidden Markov Model. International Journal of Applied Mathematics and Statistics. 40. 72-83.