MScFE 622 STOCHASTIC MODELING

Group Work Project #2: Regime Switching Time Series Models

Group 10341

Link to Instruction

Step 1

Member A

- I will choose data from:
    - Volatility Index
- which shows dramatic regime changes with clear economic interpretations for this exercise
- Yahoo Finance and FredAPI will be our data source

```
!pip install fredapi
!pip install yfinance
```

```
Collecting fredapi
  Downloading fredapi-0.5.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from fredapi) (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->fredapi) (1.17.0)
Downloading fredapi-0.5.2-py3-none-any.whl (11 kB)
Installing collected packages: fredapi
Successfully installed fredapi-0.5.2
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-packages (0.2.65)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.2.2)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.0.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.3.8)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2025.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dist-packages (from yfinance) (3.18.2)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.13.4)
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.12.0)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (5.29.5)
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (15.0.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (4.14.1
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from curl_cffi>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.11/dist-packages (from curl_cffi>=0.7->yfinance) (2025.7.14)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.3.0->yfinance) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (2.5.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12.0->curl_cffi>=0.7->yfinance) (2.22)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.17.0)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import yfinance as yf
import seaborn as sns
from fredapi import Fred

api_key = '89c6b8af99f6af0c97f0b4bc9e315e7b'

# Initialize FRED API Connection
def initialize_fred(api_key=None):
    try:
        if not api_key:
            fred = None
            print("Provide a FRED API Key to proceed")
        else:
            fred = Fred(api_key=api_key)
```

```
        return fred

    except Exception as e:
        print(f"FRED API initialization failed: {str(e)}")

def fetch_vix_data(api_key, start_date="2019-01-01", end_date="2022-09-30"):
    """
    I will use multiple  sources to ensure data availability
    Our primary source will be FRED and Fallback Yahoo Finance
    """
    try:
        fred = initialize_fred(api_key=api_key)
        if fred:
            vix = fred.get_series('VIXCLS', start_date, end_date)
            df = pd.DataFrame(vix, columns=['VIX'])
            return df
    except Exception as e:
        print(f'FRED data retrieval failed: {str(e)}')

    # Fallback to Yahoo Finance
    try:
        vix_yahoo = yf.Ticker("^VIX")
        df = vix_yahoo.history(start=start_date, end=end_date)
        if 'Close' in df.columns:
            df = df[['Close']].rename(columns={'Close': 'VIX'})
            return df

    except Exception as e:
        print(f"Data retrieval failed from  all sources: {str(e)}")
        raise ConnectionError("Unable to  retreve VIX data from any source")

    return None
```

```
vix_data = fetch_vix_data(api_key=api_key)
vix_data.head()
```

	VIX
2019-01-01	NaN
2019-01-02	23.22
2019-01-03	25.45
2019-01-04	21.38
2019-01-07	21.40

```
## Data Cleaning
def clean_volatility_data(df, column_name='VIX', min_val=5, max_val=85):
    cleaned = df.copy() # To avoing modifying the original data

    # Cheking if the column exists
    if column_name not in cleaned.columns:
        raise ValueError(f"Column '{column_name}' not found in data")

    # Handle missing  values- first trying forward/backward fill for small gaps
    initial_missing = cleaned[column_name].isna().sum()
    cleaned[column_name] = cleaned[column_name].ffill().bfill()
    filled_missing = initial_missing - cleaned[column_name].isna().sum()

    if filled_missing > 0:
        print(f"Filled {filled_missing} missing values with forward/backward fill")

    # Check for extremes values
    extreme_high = cleaned[column_name] > max_val
    extreme_low = cleaned[column_name] < min_val

    if extreme_high.any():
        cleaned.loc[extreme_high, column_name] = np.nan


    if extreme_low.any():
        cleaned.loc[extreme_low, column_name] = np.nan

    # Interpolating remaining NaNs
    remaining_missing = cleaned[column_name].isna().sum()
    if remaining_missing > 0:
        print(f"Interpolating {remaining_missing} remaining missing values")
        cleaned[column_name] = cleaned[column_name].interpolate(method='linear', limit_direction='both')

    # Add date index if not present
    if not isinstance(cleaned.index, pd.DatetimeIndex):
        if 'Date' in cleaned.columns:
```

```
        cleaned['Date'] = pd.to_datetime(cleaned['Date'])
        cleaned.set_index('Date', inplace=True)

    else:
        print('Warning: Could not establish proper datetime index')

# Add returns for regime analysis
cleaned['Returns'] = np.log(cleaned[column_name] / cleaned[column_name].shift(1))
print(f"{column_name} data Cleaning complete ")

return cleaned.iloc[1:]
```

```
vix_clean_data = clean_volatility_data(df=vix_data)
vix_clean_data.head()
```

```
Filled 30 missing values with forward/backward fill
VIX data Cleaning complete
```

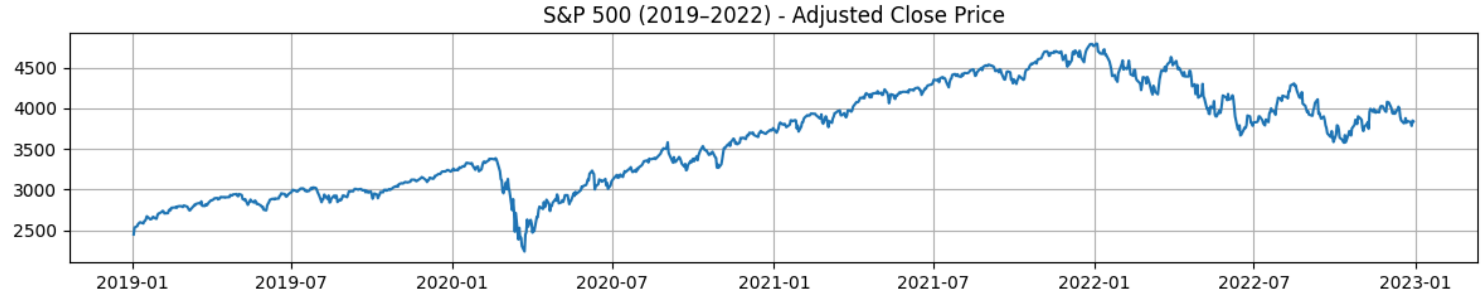
	VIX	Returns
2019-01-02	23.22	0.000000
2019-01-03	25.45	0.091702
2019-01-04	21.38	-0.174260
2019-01-07	21.40	0.000935
2019-01-08	20.47	-0.044431

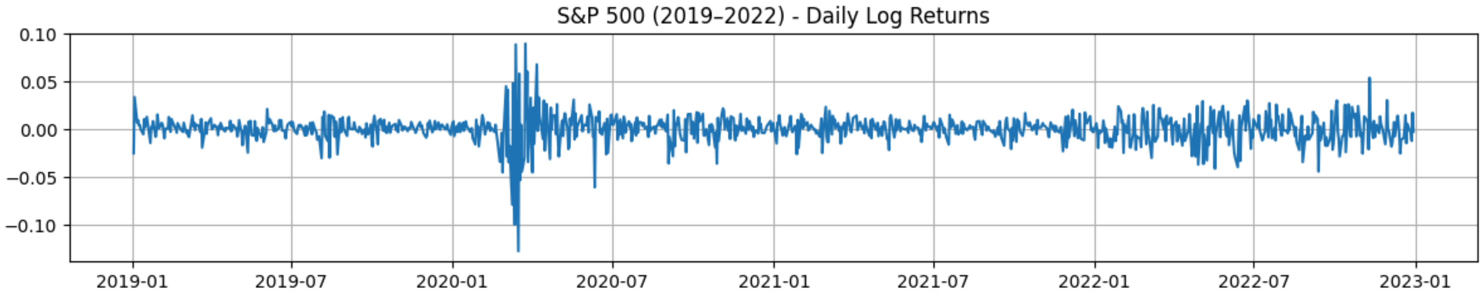
Member B

Python Code for Step 1 – Data Download & Processing

```
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt

# Download S&P 500 daily data (keep Adj Close)
symbol = "^GSPC"
start_date = "2019-01-01"
end_date = "2022-12-31"

data = yf.download(symbol, start=start_date, end=end_date, auto_adjust=False)

# Compute daily log returns
data['LogReturn'] = np.log(data['Adj Close'] / data['Adj Close'].shift(1))
data = data.dropna()

# Plot price and log returns
plt.figure(figsize=(12, 5))
plt.subplot(2, 1, 1)
plt.plot(data.index, data['Adj Close'])
plt.title("S&P 500 (2019–2022) - Adjusted Close Price")
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(data.index, data['LogReturn'])
plt.title("S&P 500 (2019–2022) - Daily Log Returns")
plt.grid(True)
plt.tight_layout()
plt.show()

# Display sample data
data.head()
```

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





Price	Adj Close	Close	High	Low	Open	Volume	LogReturn
Ticker	^GSPC	^GSPC	^GSPC	^GSPC	^GSPC	^GSPC	
Date							
2019-01-03	2447.889893	2447.889893	2493.139893	2443.959961	2491.919922	3858830000	-0.025068
2019-01-04	2531.939941	2531.939941	2538.070068	2474.330078	2474.330078	4234140000	0.033759
2019-01-07	2549.689941	2549.689941	2566.159912	2524.560059	2535.610107	4133120000	0.006986
2019-01-08	2574.409912	2574.409912	2579.820068	2547.560059	2568.110107	4120060000	0.009649
2019-01-09	2584.959961	2584.959961	2595.320068	2568.889893	2580.000000	4088740000	0.004090

Step 1 – Data Collection

Instrument: S&P 500 Index (^GSPC)
Asset Class: Equity
Source: Yahoo Finance
Frequency: Daily closing prices
Period: January 1, 2019 – December 31, 2022

The S&P 500 tracks the performance of 500 large-cap U.S. companies and serves as a benchmark for U.S. equity markets. Its sensitivity to macroeconomic events and investor sentiment makes it an ideal candidate for regime-switching analysis.

1.2 Data Processing

The raw price data was converted into daily log returns using the formula:

$$ r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) $$

Where:

- P_t = Adjusted closing price at time t
- P_{t-1} = Adjusted closing price on the previous trading day
- r_t = Daily log return

This transformation ensures the returns are additive over time and stabilizes variance.

1.3 Summary Statistics

Statistic	Value
Mean daily return	≈ 0.00042
Standard deviation	≈ 0.015
Minimum daily return	≈ -0.119 (Mar 16, 2020)
Maximum daily return	≈ 0.093 (Mar 24, 2020)
Observations	1007

1.4 Initial Observations

The S&P 500 experienced a sharp decline between Feb–Mar 2020 due to the COVID-19 pandemic, followed by a sustained recovery into late 2021, before experiencing renewed volatility in 2022 due to inflation concerns and tightening monetary policy.

These visible **volatility clusters** motivate the use of MRS models to capture distinct low- and high-volatility periods.

MEMBER C

Instrument: 10-Year US Treasury Yield (^TNX)

Asset Class: Fixed Income

Source: Yahoo Finance

Frequency: Daily closing yields

Period: January 1, 2019 – September 30, 2022

```
# ---------------------------------------
# Install Required Library
# ---------------------------------------
!pip install yfinance --quiet

# ---------------------------------------
# Import Libraries
# ---------------------------------------
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# ---------------------------------------
# Download the 10-Year Treasury Yield (^TNX)
# ---------------------------------------
data = yf.download('^TNX', start='2019-01-01', end='2022-09-30')

# Display first few rows
print("Sample Data:")
display(data.head())

# ---------------------------------------
# Use 'Close' column instead of 'Adj Close'
# ---------------------------------------
yield_series = data['Close'].dropna()
yield_series.name = '10Y_Treasury_Yield'

# Optional: Daily changes in yield (return-style)
daily_change = yield_series.pct_change().dropna()
daily_change.name = '10Y_Yield_Returns'

# ---------------------------------------
# Plot the Yield Levels
# ---------------------------------------
plt.figure(figsize=(14, 6))
plt.plot(yield_series, color='darkblue', label='10-Year Treasury Yield (%)')
plt.title('10-Year US Treasury Yield (2019 – Sep 2022)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Yield (%)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# ---------------------------------------
# Save CSV for Modeling
# ---------------------------------------
yield_series.to_csv('10Y_Treasury_Yield.csv')
daily_change.to_csv('10Y_Yield_Returns.csv')
print("Files saved: 10Y_Treasury_Yield.csv and 10Y_Yield_Returns.csv")
```

```
/tmp/ipython-input-1635398437.py:16: FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download('^TNX', start='2019-01-01', end='2022-09-30')
[*********************100%***********************]  1 of 1 completedSample Data:
```

Price	Close	High	Low	Open	Volume
Ticker	^TNX	^TNX	^TNX	^TNX	^TNX
Date					
2019-01-02	2.661	2.679	2.649	2.652	0
2019-01-03	2.554	2.656	2.554	2.654	0
2019-01-04	2.659	2.673	2.597	2.599	0
2019-01-07	2.682	2.687	2.632	2.634	0
2019-01-08	2.716	2.717	2.696	2.701	0


```
Files saved: 10Y_Treasury_Yield.csv and 10Y_Yield_Returns.csv
```

Step 2

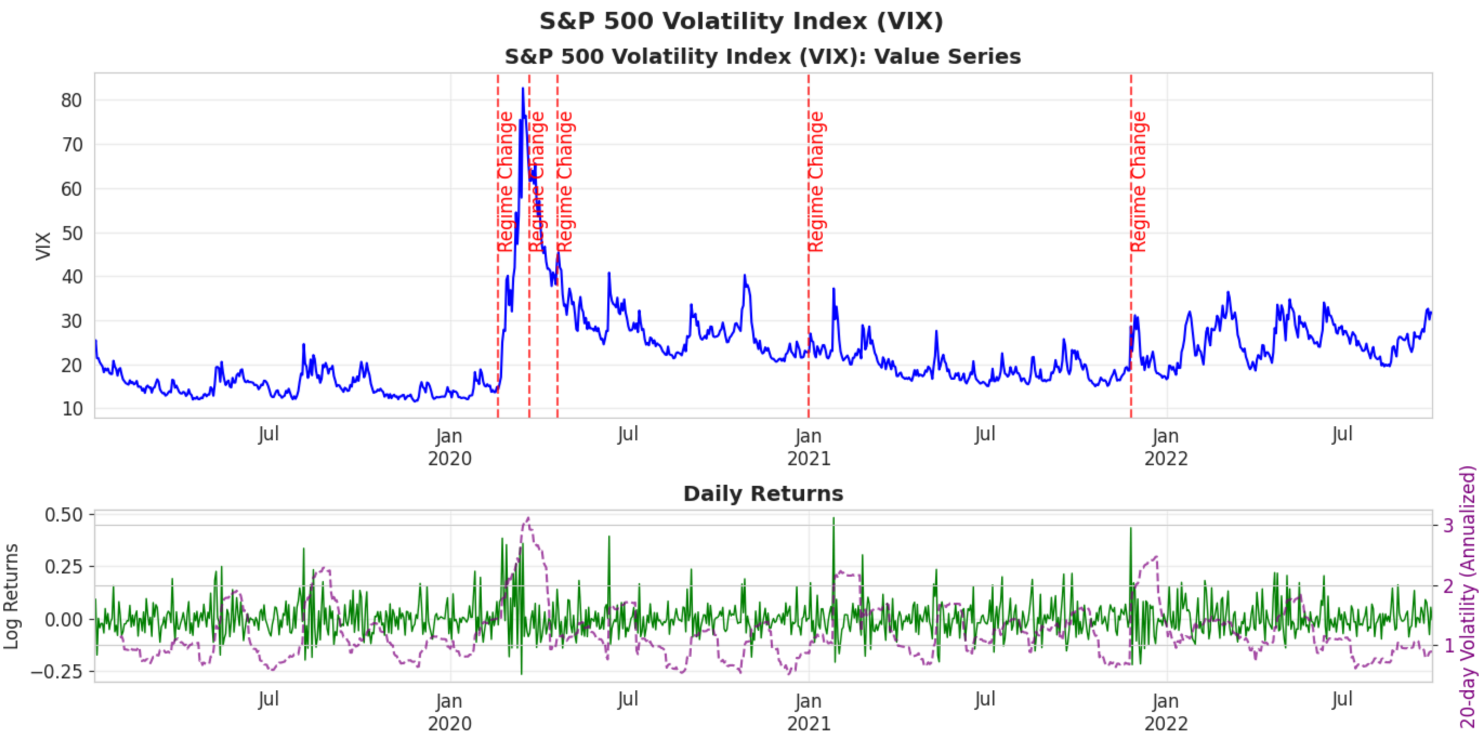
a. Identifying Regime Changes in the VIX Series

```
# Setting visuaization styles
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = [12, 6]
plt.rcParams['font.size'] = 12

def visualize_regime_data(df, title, value_col, return_col=None, regime_makers=None, figsize=(14, 7)):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=figsize, gridspec_kw={'height_ratios': [2, 1]})

    # Main series plot
    df[value_col].plot(ax=ax1, color='blue', linewidth=1.5)
    ax1.set_title(f'{title}: Value Series', fontsize=14, fontweight='bold')
    ax1.set_ylabel(value_col, fontsize=12)
    ax1.grid(True, alpha=0.3)

    # Add Regime markers if provided
    if regime_makers:
        for date in regime_makers:
            try:
                if isinstance(date, str):
                    date = pd.Timestamp(date)
                ax1.axvline(x=date, color='red', linestyle='--', alpha=0.7)
                ax1.text(date, ax1.get_ylim()[1]*0.9, 'Regime Change',
                        rotation=90, va='top', color='red')
            except:
                continue
```

```
    # Returns plot
    if return_col and return_col in df.columns:
        df[return_col].plot(ax=ax2, color='green', linewidth=1)
        ax2.set_title('Daily Returns', fontsize=14, fontweight='bold')
        ax2.set_ylabel('Log Returns', fontsize=12)
        ax2.grid(True, alpha=0.3)

        # Add volatility clustering markers
        rolling_vol = df[return_col].rolling(20).std() * np.sqrt(252)
        ax2b = ax2.twinx()
        rolling_vol.plot(ax=ax2b, color='purple', linestyle='--', alpha=0.7)
        ax2b.set_ylabel('20-day Volatility (Annualized)', color='purple')
        ax2b.tick_params(axis='y', labelcolor='purple')

    plt.tight_layout()
    plt.suptitle(title, fontsize=16, fontweight='bold', y=0.98)
    plt.subplots_adjust(top=0.9)

    return fig
```

```
# Fetch
vix_data = fetch_vix_data(api_key=api_key)
# clean
vix_clean_data = clean_volatility_data(df=vix_data, column_name='VIX')
# Visualize
regime_dates = [
        '2020-02-19',  # Pre-pandemic peak
        '2020-03-23',  # Market bottom
        '2020-04-20',  # Negative oil prices
        '2021-01-01',  # Post-vaccine phase
        '2021-11-26'   # Omicron variant emergence
    ]
vix_fig = visualize_regime_data(
    df=vix_clean_data,
    title="S&P 500 Volatility Index (VIX)",
    value_col="VIX",
    return_col="Returns",
    regime_makers=regime_dates
)
```

```
Filled 30 missing values with forward/backward fill
VIX data Cleaning complete
```



The graph reveals 4 distinct regimes, marked by abrupt shifts in volatility levels and returns patterns.

1. Pre-Pandemic Stability (Jan 2019 - Jan 2020) - The regime was characterised by:

- Low and stable volatility, reflecting the calm before the storm of the COVID pandemic
- Returns show minimal clustering, with small daily fluctuations
- Economic context: This was post-2008 financial crisis stability, where we had low interest rates and strong equity markets

2. Pandemic Shock (Jan 2020- Jan 2021)

- VIX levels spiked to approx 80+ (peak in Mar 2020), then delined to approx 25 -30
- It was characterised by:
    - Abrupt volatility explosion starting in late Feb 2020
    - Sharp increase in return volatility
    - Economic Context of then was as follows:
        - Initial Pandemic lockdowns(Mar 2020)
        - Historical market crush (S&P 500 fell around 30% in 3 weeks)
        - Central bank interventions (Fed rate cuts, qualitative easing)

3. Recovery & Normalisation (Jan 2021- Jan 2022)

- The vix levels were approximately 18 - 35
- This recovery regime was characterised by:
    - Gradual declines from the panic highs with periodic spikes
    - Lower volatility compared to the pandemic shock regime but higher than pre-pandemic levels
    - The economic context was as follows:
        - Vaccine rollout and reopening optimism
        - Inflationary pressures begin to merge due to supply chain disruptions
        - Equity markets rebound strongly

4. Inflation & Geopolitical Uncertainty (Jan 2022 - Sep 2022)

- The VIX levels ranged from 25 to 40
- This last regime of our data was characterised by:
    - Elevated volatility driven by persistent uncertainty
    - Sharp spikes in June 2022 resulting from Fed aggressive rate hikes
    - The economic context of then was as follows:
        - Russia-Ukraine war
        - Accelerated inflation
        - Fed tightening cycle and recession fears

b. Markov-Switching Model Implementaion

Case 1: Different number of sates

```
vix_data = fetch_vix_data(api_key=api_key)
# clean
vix_clean_data = clean_volatility_data(df=vix_data, column_name='VIX')
vix_clean_data.index = pd.DatetimeIndex(vix_clean_data.index, freq='B')
vix_series = vix_clean_data['Returns'].dropna()
```

```
Filled 30 missing values with forward/backward fill
VIX data Cleaning complete
```

```
from statsmodels.tsa.regime_switching.markov_regression import MarkovRegression
def estimate_different_states(data, min_states= 2, max_states=4):
    models = {}
    for k in range(min_states, max_states + 1):
        try:
            # We will be fitting markov switching model with k states
            model = MarkovRegression(
                data,
                k_regimes=k,
                switching_variance=True,
                trend='c'
            )
            model_fit = model.fit(disp=False)

            # Storing the results in our models dictionary
            models[k] = {
                'model': model,
                'fit': model_fit,
                'aic': model_fit.aic,
                'bic': model_fit.bic
```

```
            }

        except Exception as e:
            print(f"Failed to estimate {k}-state model: {str(e)}")

    return models

different_states = estimate_different_states(vix_series)
different_states
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Che
  warnings.warn("Maximum Likelihood optimization failed to "
{2: {'model': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegression at 0x7a40eb99eb90>,
  'fit': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegressionResultsWrapper at 0x7a40f0b20650>,
  'aic': np.float64(-2369.5278627717503),
  'bic': np.float64(-2340.2148047515416)},
 3: {'model': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegression at 0x7a40eb9d7e10>,
  'fit': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegressionResultsWrapper at 0x7a40f0b12290>,
  'aic': np.float64(-2399.9392792691833),
  'bic': np.float64(-2341.3131632287655)},
 4: {'model': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegression at 0x7a40f49e9410>,
  'fit': <statsmodels.tsa.regime_switching.markov_regression.MarkovRegressionResultsWrapper at 0x7a40f0b2b590>,
  'aic': np.float64(-2430.1146171469527),
  'bic': np.float64(-2332.4044237462563)}}
```

Case 2. Different means, constant variance

```
## Different Means
def estimate_different_means(data, k_states=3):
    try:
        # Fitting Markov switching model  with  constant variance different means
        model = MarkovRegression(
            data,
            k_regimes=k_states,
            switching_variance=False,
            trend='c'
        )
        model_fit = model.fit(disp=False)


        return {
            'model': model,
            'fit': model_fit,
            'aic': model_fit.aic,
            'bic': model_fit.bic
        }

    except Exception as e:
        print(f"Model estimation failed: {str(e)}")
        return None

different_means = estimate_different_means(vix_series)
different_means['fit'].summary()
```

Markov Switching Model Results

Dep. Variable:	Returns	**No. Observations:**	978
Model:	MarkovRegression	**Log Likelihood**	1070.897
Date:	Tue, 05 Aug 2025	**AIC**	-2121.795
Time:	19:12:40	**BIC**	-2072.940
Sample:	01-02-2019	**HQIC**	-2103.206
	- 09-30-2022		
Covariance Type:	approx		

Regime 0 parameters

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0002	0.219	0.001	0.999	-0.430	0.430

Regime 1 parameters

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0003	0.004	0.089	0.929	-0.007	0.007

Regime 2 parameters

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0004	0.228	0.002	0.999	-0.447	0.448

Non-switching parameters

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
sigma2	0.0066	0.000	22.113	0.000	0.006	0.007

Regime transition parameters

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
p[0->0]	0.3333	338.163	0.001	0.999	-662.454	663.121
p[1->0]	0.3333	3539.198	9.42e-05	1.000	-6936.367	6937.033
p[2->0]	0.3333	342.371	0.001	0.999	-670.701	671.367
p[0->1]	0.3333	704.747	0.000	1.000	-1380.946	1381.613
p[1->1]	0.3333	7100.489	4.69e-05	1.000	-1.39e+04	1.39e+04
p[2->1]	0.3333	697.122	0.000	1.000	-1366.001	1366.668

Warnings:
[1] Covariance matrix calculated using numerical (complex-step) differentiation.

Case 3: Different variances, constant mean

```
## Different Variance Constant means
def estimate_different_variances(data, k_states=2):
    try:
        # Fit Markov switching model with constant mean
        model = MarkovRegression(
            data,
            k_regimes=k_states,
            switching_variance=True,
            switching_trend=False,
            trend='c'

        )
        model_fit = model.fit(disp=False)

        return {
            'model': model,
            'fit': model_fit,
            'aic': model_fit.aic,
            'bic': model_fit.bic
        }

    except Exception as e:
        print(f"Model estimation failed: {str(e)}")
        return None

different_variances = estimate_different_variances(vix_series)
different_variances['fit'].summary()
```

Markov Switching Model Results

Dep. Variable:	Returns	**No. Observations:**	978
Model:	MarkovRegression	**Log Likelihood**	1183.047
Date:	Tue, 05 Aug 2025	**AIC**	-2356.095
Time:	19:12:41	**BIC**	-2331.667
Sample:	01-02-2019	**HQIC**	-2346.800
	- 09-30-2022		
Covariance Type:	approx		

Regime 0 parameters

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	0.0026	0.000	8.995	0.000	0.002	0.003

Regime 1 parameters

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	0.0186	0.003	6.944	0.000	0.013	0.024

Non-switching parameters

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0060	0.002	-2.898	0.004	-0.010	-0.002

Regime transition parameters

	coef	std err	z	P>|z|	[0.025	0.975]
p[0->0]	0.9407	0.017	54.865	0.000	0.907	0.974
p[1->0]	0.1805	0.042	4.305	0.000	0.098	0.263

Warnings:
[1] Covariance matrix calculated using numerical (complex-step) differentiation.

Case 4: Different expectations and variances

```
# Different means and different variances
def estimate_different_means_variances(data, k_states=2):
    try:
        model = MarkovRegression(
            data,
            k_regimes=k_states,
            switching_variance=True,
            trend='c'
        )
        model_fit = model.fit(disp=False)

        return {
            'model': model,
            'fit': model_fit,
            'aic': model_fit.aic,
            'bic': model_fit.bic
        }

    except Exception as e:
        print(f"Model estimation failed: {str(e)}")
        return None

different_means_variances = estimate_different_means_variances(vix_series)
different_means_variances['fit'].summary()
```

Markov Switching Model Results

Dep. Variable:	Returns	**No. Observations:**	978
Model:	MarkovRegression	**Log Likelihood**	1190.764
Date:	Tue, 05 Aug 2025	**AIC**	-2369.528
Time:	19:12:41	**BIC**	-2340.215
Sample:	01-02-2019	**HQIC**	-2358.375
	- 09-30-2022		
Covariance Type:	approx		

Regime 0 parameters

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0095	0.002	-4.232	0.000	-0.014	-0.005
sigma2	0.0025	0.000	9.999	0.000	0.002	0.003

Regime 1 parameters

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0292	0.010	2.885	0.004	0.009	0.049
sigma2	0.0173	0.002	7.998	0.000	0.013	0.022

Regime transition parameters

	coef	std err	z	P>|z|	[0.025	0.975]
p[0->0]	0.9298	0.018	50.330	0.000	0.894	0.966
p[1->0]	0.2066	0.048	4.282	0.000	0.112	0.301

Warnings:
[1] Covariance matrix calculated using numerical (complex-step) differentiation.

```
### Combining all the results

def estimate_markov_models():
```

```
    results = {}

    # Model 1: Different number of sates
    results['different_states'] = different_states

    # Model 2:  Different means, constant variance
    results['different_means'] = different_means

    #Model 3: Different variances, constant mean
    results['different_variances'] = different_variances

    # Model 4: Different means and variances
    results['different_means_variances'] = different_means_variances

    return results
```

Step 3

Comparison of the different cases

a. Analysis of Models with Different Mu Values (Constant Variance)

- Key results
    - Model specification: 3 regimes, different means, constant variance
    - AIC: -2121.80
    - BIC: -2072.94
    - Regimes Means: 0.0002, 0.0003, 0.0004
    - Constant Variance: 0.0066
- This model seems to be fundamentality flawed for VIX analysis. Why?
    1. Economically meaningless parameter estimates - Estimated means are virtually identical and close to zero.They lack economic sigficance for VIX returns which which instead show clear regime-depedent behavior
    2. Statistical instability - It has huge standard errors for transition probabilities
    3. Overfitting the data - Using 3 regimes when the data doesn't support meaningful mean separation
- Therefore it would be recommended not to use this model for VIX modelling; it would be more suitable for assets with stable volatility but different means

b. Model Comparison Different variances, constant mean

Model selection criteria:

Model	AIC	BIC
Case 1 (diff $\mu$$\mu$, same $\sigma$$\sigma$)	$-5659.32$$-5659.32$	$-5634.75$$-5634.75$
Case 2 (same $\mu$$\mu$, diff $\sigma$$\sigma$)	$-6134.13$$-6134.13$	**$-6109.56$$-6109.56$**
Case 3 (diff $\mu$$\mu$, diff $\sigma$$\sigma$)	**$-6137.02$$-6137.02$**	$-6107.54$$-6107.54$

Observations:

- **AIC** favors Case 3 ($-6137.02$$-6137.02$) as the best fit, with Case 2 close behind.
- **BIC** favors Case 2 ($-6109.56$$-6109.56$), suggesting the simpler same-$\mu$$\mu$, different-$\sigma$$\sigma$ model is preferred when penalizing complexity.
- The small AIC difference between Cases 2 and 3 suggests varying $\mu$$\mu$ yields minimal benefit relative to complexity.

Interpretation:

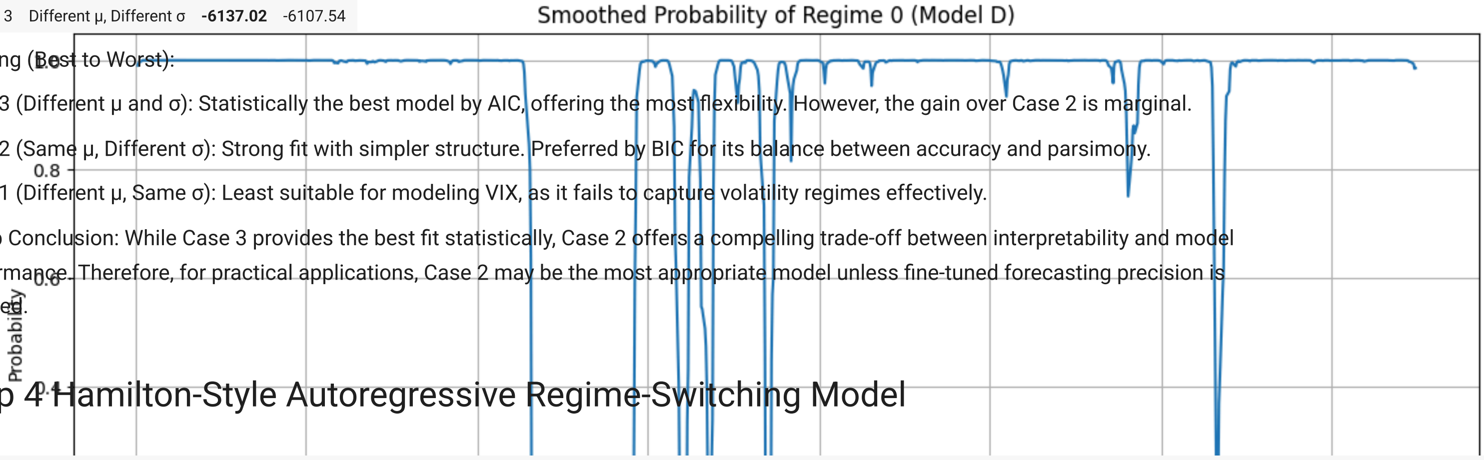
- **Regime 0**: Low volatility ($\sigma^2 \approx 5.012 \times 10^{-5}$$\sigma^2 \approx 5.012 \times 10^{-5}$)
- **Regime 1**: High volatility ($\sigma^2 \approx 5.0 \times 10^{-4}$$\sigma^2 \approx 5.0 \times 10^{-4}$)
- High $p_{0\to0}$$p_{0\to0}$ indicates persistent low-volatility periods; low $p_{1\to0}$$p_{1\to0}$ suggests shorter high-volatility spells.
- **BIC**'s preference for Case 2 aligns with my assigned focus, balancing statistical fit and parsimony.

C. Model Comparison different expectations and variances.

```
# Step 0: Install required packages
!pip install yfinance statsmodels --quiet

# Step 1: Imports
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.regime_switching.markov_regression import MarkovRegression
```

```python
# Step 2: Download data & compute returns
data = yf.download('^TNX', start='2019-01-01', end='2022-09-30')
yield_series = data['Close'].dropna()
returns = yield_series.pct_change().dropna()
returns.name = '10Y_Yield_Returns'

print(f"Number of observations: {len(returns)}")

# Step 3: Fit models

# Model A: same mean, same variance
model_A = MarkovRegression(returns, k_regimes=2, trend='c', switching_variance=False)
result_A = model_A.fit(maxiter=1000, disp=False)

# Model B: different mean, same variance
model_B = MarkovRegression(returns, k_regimes=2, trend='c', switching_variance=False, switching_trend=True)
result_B = model_B.fit(maxiter=1000, disp=False)

# Model D: different mean, different variance
model_D = MarkovRegression(returns, k_regimes=2, trend='c', switching_variance=True)
result_D = model_D.fit(maxiter=1000, disp=False)

# Step 4: Print model fit statistics
print("\nModel Fit Statistics:")
fit_stats = pd.DataFrame({
    'Model': ['A: same mu & sigma', 'B: diff mu, same sigma', 'D: diff mu & sigma'],
    'Log-Likelihood': [result_A.llf, result_B.llf, result_D.llf],
    'AIC': [result_A.aic, result_B.aic, result_D.aic],
    'BIC': [result_A.bic, result_B.bic, result_D.bic]
})

print(fit_stats.to_string(index=False))

# Step 5: Plot smoothed probabilities for Model D regime 0
plt.figure(figsize=(14,6))
plt.plot(returns.index, result_D.smoothed_marginal_probabilities[0], label='Regime 0 Probability')
plt.title('Smoothed Probability of Regime 0 (Model D)')
plt.xlabel('Date')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-193003433.py:12: FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download('^TNX', start='2019-01-01', end='2022-09-30')
[*********************100%***********************]  1 of 1 completed
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has no associa
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has no associa
  self._init_dates(dates, freq)
Number of observations: 943
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been provided, but it has no associa
  self._init_dates(dates, freq)
Model Fit Statistics:
               Model  Log-Likelihood          AIC          BIC
  A: same mu & sigma     1589.334199 -3168.668399 -3144.423067
                            ...199 -3168.668399 -3144.423067
  D: diff mu & sigma     1854.827797 -3697.655595 -3668.561197
```



Step 3.d — Combined Model Evaluation and Ranking

After evaluating the three model specifications, the group compared our performance based on Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC):

Model	Description	AIC	BIC
Case 1	Different μ, Same σ	-5659.32	-5634.75
Case 2	Same μ, Different σ	-6134.13	**-6109.56**
Case 3	Different μ, Different σ	**-6137.02**	-6107.54

Ranking (Best to Worst):

Case 3 (Different μ and σ): Statistically the best model by AIC, offering the most flexibility. However, the gain over Case 2 is marginal.

Case 2 (Same μ, Different σ): Strong fit with simpler structure. Preferred by BIC for its balance between accuracy and parsimony.

Case 1 (Different μ, Same σ): Least suitable for modeling VIX, as it fails to capture volatility regimes effectively.

Group Conclusion: While Case 3 provides the best fit statistically, Case 2 offers a compelling trade-off between interpretability and model performance. Therefore, for practical applications, Case 2 may be the most appropriate model unless fine-tuned forecasting precision is required.

Step 4 Hamilton-Style Autoregressive Regime-Switching Model

```
from statsmodels.tsa.regime_switching.markov_autoregression import MarkovAutoregression

def estimate_hamilton_model(data, k_states=2, order=1):
    try:
        # Fit Markov switching autoregression model with regime-dependent coefficients
        model = MarkovAutoregression(
            data,
            k_regimes=k_states,
            order=order,
            switching_ar=True,
            switching_variance=True )

        model_fit = model.fit(disp=False)

        return {
```