Markov Chain based Stochastic Processes for Portfolio Optimization

MATH 171

Peter Yim, Arjun Parikh, Takao Oba

I. Introduction

Stochastic processes are mathematical models that are used to describe random events or behavior over time. In finance, stochastic processes are particularly useful for modeling the random fluctuations in financial markets. One popular type of stochastic process used in finance is Markov Chains, which are a mathematical framework that allows us to model the transition probabilities between different states of a system.

The objective of our research paper is to explore the applications of stochastic processes, particularly Markov Chains, in portfolio optimization. Portfolio optimization is the process of constructing a portfolio of assets that maximizes expected returns while minimizing the risk of the investor. To achieve this objective, we plan to build a model using Markov Chains to quantify three states (bullish, neutral, or bearish) and ultimately determine which stocks have the highest probability of positive returns.

II. Research Review

The research article, "Portfolio Optimization by Applying Markov Chains" by Nina Petkovic, Milan Bozinovic, and Sanja Stojanovi explores the use of the Markov chains method in portfolio optimization in the Belgrade Stock Exchange. Petkovic et al. (2018) highlights the lack of sufficient research done in this area and how the Markov chains method has been widely used in financial market analyses worldwide, despite the level of their development. The Markov chains method is non-parametric, less complex, and has been found to produce similar results to the Harry Markowitz model but faster and easier to obtain. This research is unique as it is the first to employ the Markov chains method in the returns analysis on the Belgrade Stock Exchange. The study provides insight into the use of the Markov chains method in portfolio optimization, especially in emerging markets, and its potential benefits over traditional methods.

III. Proposal of Stochastic Process

Stochastic processes have been widely used in finance to model the randomness of financial markets. We introduce Markov Chains, which is a type of a stochastic process that is characterized by the Markov property, which states that the probability of transitioning from one state to another only depends on the current state of the system.

We propose to use Markov Chains to model the long term behavior of stock prices which can provide invaluable information when optimizing one's portfolio. Markov Chains involve state and transitions. We characterize the states to be the following: 1. The stock is most likely going to rise in price, 2. The stock is most likely going to remain at its current price, and 3. The stock is most likely going to decline in price. These three states will be unique to all stocks that we will be examining. However, the transition probabilities will be unique for all stocks and will be calculated through assessing past values. In other words, by using Markov Chains, we will consider historical price changes to determine the probabilities of going from one state to another, which will help us in building a probability transition matrix. Through manipulation of this chain, we can then find long-term probabilities through calculating the stationary distribution of each state.

IV. Computational Analysis of the Model

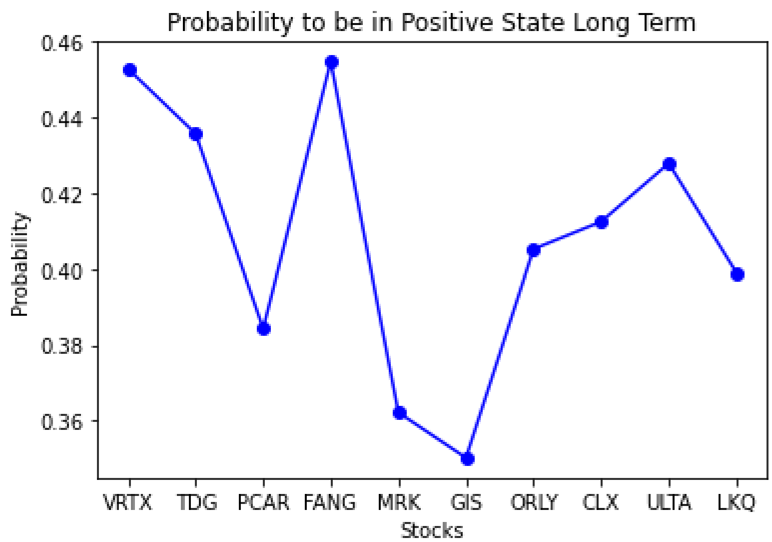
We pulled data on daily returns for the 500 stocks making up the S&P500 from Yahoo Finance. We then computed the Probability Transition matrix for each stock based on a three state model, where the states were a daily return less than -0.005, a daily return in the range -0.005 to 0.005, and a daily return greater than 0.005. We built the Probability Transition Matrix by finding the number of transitions from one state to another, then dividing that value by the total number of days that the stock was in each state (e.g. To find p_{1,2}, we took the number of transitions from a return less than -0.005 to a return in the range -0.005 to 0.005, and divided this by the total number of days the stock had a return less than -0.005). We restricted our computations of these matrices to the ten stocks we found to have the most number of positive trading days over the past year. These stocks have tickers VRTX, TDG, PCAR, FANG, MRK, GIS, ORLY, CLX, ULTA, LKQ. We then calculated the stationary distributions for each of these stocks to see their long term probabilities of being negative, neutral, and positive. The results were as follows:

```
stationary distribution of  VRTX [0.34278249 0.2043344  0.45288311]
stationary distribution of  TDG [0.30977301 0.25448453 0.43574246]
stationary distribution of  PCAR [0.31993319 0.29569762 0.38436919]
stationary distribution of  FANG [0.2843297  0.26087662 0.45479368]
stationary distribution of  MRK [0.31542873 0.32238054 0.36219073]
stationary distribution of  GIS [0.32446809 0.32532187 0.35021004]
stationary distribution of  ORLY [0.29874676 0.29599459 0.40525865]
stationary distribution of  CLX [0.30036404 0.2871515  0.41248446]
stationary distribution of  ULTA [0.28810679 0.28400384 0.42788937]
stationary distribution of  LKQ [0.33465502 0.26629158 0.3990534 ]
```

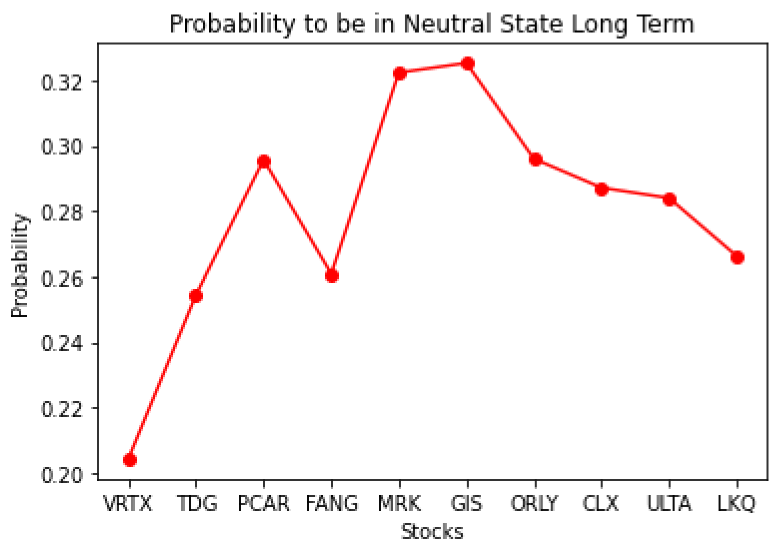
V. Conclusion

In this research paper, we have explored the applications of stochastic processes, particularly Markov Chains, in the realm of finance and specifically to optimize one’s portfolio. We have built a model using Markov Chains to determine the probabilities of stocks moving between negative, neutral, and positive states. Through this model, we were able to select the ten most profitable stocks to invest in based on previous data as a way of “optimizing a portfolio”.

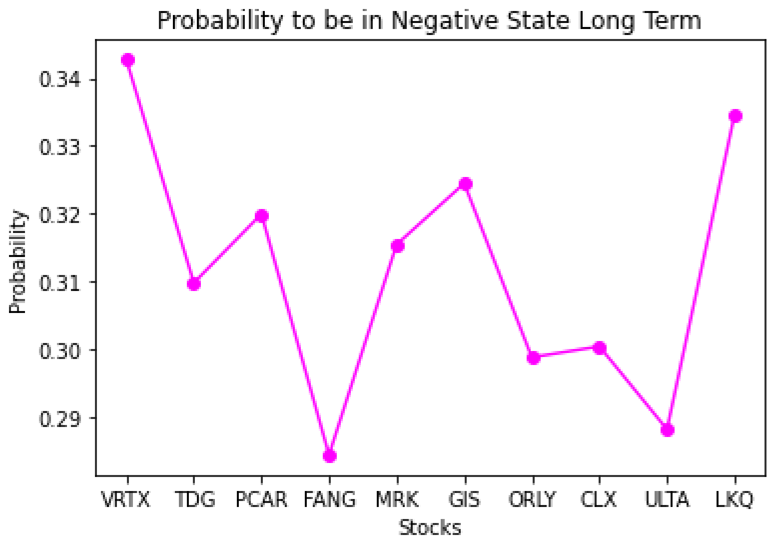
The following are charts representing the stationary distribution, or the long term probabilities for each of the stocks to be in our specified state space. Our model suggested that the stocks VRTX, FANG, and TDG have the highest probability of staying in a positive state over the long term, that is, these are the stocks that have the highest probability of returning daily returns greater than 0.005.



Our model suggested that the stocks MRK, GRS, ORLY, and PCAR have the highest probability of staying in the neutral state over the long term, that is, these are the stocks that have the highest probability of returning daily returns in the range -0.005 to 0.005.



Our model suggested that the stocks VRTX, GIS, and LKQ have the highest probability of staying in a negative state over the long term, that is, these are the stocks that have the highest probability of returning daily returns less than -0.005.



It was interesting to see VRTX as the highest in likelihood to be both in the positive and negative states, while having the lowest likelihood of being in the neutral state, thus suggesting heightened risk with investing in this stock. This model can be helpful in informing investing decisions based on risk tolerance. Those investors looking for lower risk investments would likely invest more in the stocks with higher probability of being in a neutral state, while those with more of a risk appetite may consider investing in stocks like VRTX.

While our model has shown promising results, there can be limitations to our approach. It is important to note that for the purpose of this project we chose to focus on the stocks from the S&P500 that had the most number of positive trading days over the past trading year. This said, this process can be used to analyze any stock with sufficient available data to aid in investing decisions and risk tolerance. In addition to historic price, there could be various factors involved in the price movement of a stock and unexpected world-wide events may occur such as the COVID-19 pandemic. Furthermore, the chosen stocks that had the most days with positive returns can be correlated with each other in terms of company-specific factors and sector-specific factors and thus we may have to be careful when investing collectively in a portfolio.

In future research, we plan to expand our model to include other factors that may affect the behavior of financial markets, such as macroeconomic indicators and geopolitical events. It may also be worth testing models based on various time intervals of data. In addition, we plan to test the robustness of our model by applying it to different markets and time periods.

```
!pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.2.12-py2.py3-none-any.whl (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.2/59.2 KB 1.6 MB/s eta 0:00:00
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.9/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.9/dist-packages (from yfinance) (1.4.4)
Collecting frozendict>=2.3.4
  Downloading frozendict-2.3.5-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (112 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 112.8/112.8 KB 3.5 MB/s eta 0:00:00
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.9/dist-packages (from yfinance) (4.11.2)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.9/dist-packages (from yfinance) (4.9.2)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.9/dist-packages (from yfinance) (2.27.1)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.9/dist-packages (from yfinance) (1.1)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.9/dist-packages (from yfinance) (1.22.4)
Collecting appdirs>=1.4.4
  Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.9/dist-packages (from yfinance) (2022.7.1)
Collecting cryptography>=3.3.2
  Downloading cryptography-39.0.2-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 8.4 MB/s eta 0:00:00
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.9/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.9/dist-packages (from cryptography>=3.3.2->yfinance) (1.15.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.9/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.9/dist-packages (from html5lib>=1.1->yfinance) (1.15.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests>=2.26->yfinance) (1.26.15)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests>=2.26->yfinance) (2.0.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests>=2.26->yfinance) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests>=2.26->yfinance) (2022.12.7)
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packages (from cffi>=1.12->cryptography>=3.3.2->yfinance) (2.2
Installing collected packages: appdirs, frozendict, cryptography, yfinance
Successfully installed appdirs-1.4.4 cryptography-39.0.2 frozendict-2.3.5 yfinance-0.2.12
```

```
import numpy as np
import yfinance as yf
import pandas as pd
from pandas_datareader import data as pdr
import datetime as dt
from dateutil.relativedelta import relativedelta

yf.pdr_override()
```

```
#pull data from yahoofinance
end = '2023-03-13'
SPY = pdr.get_data_yahoo('SPY', start='2022-03-13', end=end, progress=False)
SPY
```

```
tickers = pd.read_csv('SPtickers.csv')['Symbol']
tickers.replace({'BRK.B': 'BRK-B', 'BF.B': 'BF-B'}, inplace=True)
tickers = tickers[tickers != 'FBHS']
data = pdr.get_data_yahoo(list(tickers), start='2022-03-13', end='2023-03-13', progress=False)
data.to_csv('SP500-prices.csv')
```

```
data = pd.read_csv('SP500-prices.csv',header=[0,1],index_col=0)
data['Adj Close']
```

```
#compute daily returns
daily_returns = data['Adj Close'].pct_change(1)
daily_returns = daily_returns.iloc[1:]
daily_returns.dropna(axis=1, inplace=True)
daily_returns
```

	A	AAL	AAP	AAPL	ABBV	ABC	ABMD	ABT	ACGL	ACN	...	WYNN	
Date													
2022-03-15	0.021163	0.092632	0.025606	0.029677	0.024852	0.035854	0.043560	0.016967	0.010360	0.027789	...	0.039474	
2022-03-16	0.036230	0.057803	0.015910	0.029015	0.001091	-0.005879	0.016883	0.017024	-0.003418	0.027448	...	0.081272	-
2022-03-17	0.013481	0.004857	0.010484	0.006454	0.016277	0.011296	0.044546	0.014396	-0.003215	-0.016651	...	0.023148	
2022-03-18	0.016810	0.012085	-0.008838	0.020919	0.003846	-0.001314	-0.010792	0.010313	-0.014194	0.013928	...	0.035626	-
2022-03-21	-0.007044	-0.039403	-0.002859	0.008538	0.005339	0.007763	0.007919	-0.005635	0.028578	0.001698	...	-0.016949	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2023-03-06	-0.004863	-0.014670	-0.024191	0.018539	-0.004998	0.001985	0.000000	-0.016276	0.005610	0.001597	...	-0.003712	-
2023-03-07	-0.020247	0.014888	-0.010656	-0.014496	-0.015134	-0.020772	0.000000	-0.026959	-0.009902	-0.020139	...	-0.019411	-
2023-													

```python
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
days_within_bounds = {}

# Loop through each stock in the dataset
for stock in daily_returns.columns:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Calculate number of days within bounds for current stock
    days_within_bounds[stock] = ((returns >= lower_bound) & (returns <= upper_bound)).sum()

# Print results
print("Number of days within bounds for each stock:")
for stock, days in days_within_bounds.items():
    print(f"{stock}: {days} days")
```

```python
# Define upper bound for returns
upper_bound = 0.005

# Initialize dictionary to hold results
days_within_bounds = {}

# Loop through each stock in the dataset
for stock in daily_returns.columns:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Calculate number of days within bounds for current stock
    days_within_bounds[stock] = (returns > upper_bound).sum()

# Print results
print("Number of days with returns greater than upper bound for each stock:")
for stock, days in days_within_bounds.items():
    print(f"{stock}: {days} days")
```

```python
# Define lower bound for returns
lower_bound = -0.005

# Initialize dictionary to hold results
days_within_bounds = {}

# Loop through each stock in the dataset
for stock in daily_returns.columns:
    # Get daily returns for current stock
    returns = daily_returns[stock]
```

```
    # Calculate number of days within bounds for current stock
    days_within_bounds[stock] = (returns < lower_bound).sum()

# Print results
print("Number of days with returns less than lower bound for each stock:")
for stock, days in days_within_bounds.items():
    print(f"{stock}: {days} days")



# Calculate the number of positive trading days for each stock
positive_days = daily_returns[daily_returns > 0].count()

# Sort the stocks by the number of positive trading days
top_positive = positive_days.sort_values(ascending=False)[:10]

# Print the results
print("Top 10 stocks by number of positive trading days:")
for i, (stock, count) in enumerate(top_positive.items(), start=1):
    print(f"{i}. {stock}: {count} positive trading days")
```

```
Top 10 stocks by number of positive trading days:
1. VRTX: 144 positive trading days
2. TDG: 144 positive trading days
3. PCAR: 142 positive trading days
4. FANG: 141 positive trading days
5. MRK: 140 positive trading days
6. GIS: 140 positive trading days
7. ORLY: 140 positive trading days
8. CLX: 140 positive trading days
9. ULTA: 139 positive trading days
10. LKQ: 138 positive trading days
```

```
# Define threshold return
threshold_return_high = 0.005

# Initialize dictionary to hold results
counts_high = {}

stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Count number of days with return greater than threshold
    count = len(returns[returns > threshold_return_high])

    # Add count to dictionary
    counts_high[stock] = count

# Print results
print("Number of days with return greater than", threshold_return_high, "for each stock:")
for stock, count in counts_high.items():
    if stock in stocks:
        print(f"{stock}: {count} days")
```

```
Number of days with return greater than 0.005 for each stock:
VRTX: 113 days
TDG: 110 days
PCAR: 96 days
FANG: 112 days
MRK: 92 days
GIS: 88 days
ORLY: 102 days
CLX: 104 days
ULTA: 108 days
LKQ: 100 days
```

```
# Define threshold return
threshold_return_low = -0.005

# Initialize dictionary to hold results
counts_low = {}
```

```
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Count number of days with return greater than threshold
    count = len(returns[returns < threshold_return_low])

    # Add count to dictionary
    counts_low[stock] = count

# Print results
print("Number of days with return less than", threshold_return_low, "for each stock:")
for stock, count in counts_low.items():
    if stock in stocks:
        print(f"{stock}: {count} days")
```

```
Number of days with return less than -0.005 for each stock:
VRTX: 83 days
TDG: 94 days
PCAR: 85 days
FANG: 94 days
MRK: 70 days
GIS: 71 days
ORLY: 77 days
CLX: 84 days
ULTA: 83 days
LKQ: 83 days
```

```
# Define threshold returns
lower_threshold_return = -0.005
upper_threshold_return = 0.005

# Initialize dictionary to hold results
counts_mid = {}

stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Count number of days with return between thresholds
    count = len(returns[(returns >= lower_threshold_return) & (returns <= upper_threshold_return)])

    # Add count to dictionary
    counts_mid[stock] = count

# Print results
print("Number of days with return between", lower_threshold_return, "and", upper_threshold_return, "for each stock:")
for stock, count in counts_mid.items():
    if stock in stocks:
        print(f"{stock}: {count} days")
```

```
Number of days with return between -0.005 and 0.005 for each stock:
VRTX: 53 days
TDG: 45 days
PCAR: 68 days
FANG: 43 days
MRK: 87 days
GIS: 90 days
ORLY: 70 days
CLX: 61 days
ULTA: 58 days
LKQ: 66 days
```

```
import pandas as pd

df_days = pd.concat([pd.Series(counts_low), pd.Series(counts_mid), pd.Series(counts_high)], axis=1)
df_days.columns = ['negative', 'neutral', 'positive']
```

```
print(df_days)
```

```
     negative  neutral  positive
VRTX       83       53       113
TDG        94       45       110
PCAR       85       68        96
FANG       94       43       112
MRK        70       87        92
GIS        71       90        88
ORLY       77       70       102
CLX        84       61       104
ULTA       83       58       108
LKQ        83       66       100
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_onetotwo = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from lower to within bounds
    for i in range(1, len(returns)):
        if returns.iloc[i-1] < lower_bound and lower_bound <= returns.iloc[i] <= upper_bound:
            count += 1

    # Add count to dictionary
    counts_onetotwo[stock] = count

# Print results
print("Number of transitions from returns less than lower bound to within bounds for each stock:")
for stock, count in counts_onetotwo.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns less than lower bound to within bounds for each stock:
VRTX: 13 transitions
TDG: 15 transitions
PCAR: 22 transitions
FANG: 17 transitions
MRK: 26 transitions
GIS: 24 transitions
ORLY: 18 transitions
CLX: 26 transitions
ULTA: 15 transitions
LKQ: 21 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_onetothree = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0
```

```
    # Loop through returns and count transitions from lower to upper bounds
    for i in range(1, len(returns)):
        if returns.iloc[i-1] < lower_bound and returns.iloc[i] > upper_bound:
            count += 1

    # Add count to dictionary
    counts_onetothree[stock] = count

# Print results
print("Number of transitions from returns less than lower bound to greater than upper bound for each stock:")
for stock, count in counts_onetothree.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns less than lower bound to greater than upper bound for each stock:
VRTX: 41 transitions
TDG: 42 transitions
PCAR: 31 transitions
FANG: 42 transitions
MRK: 20 transitions
GIS: 25 transitions
ORLY: 36 transitions
CLX: 32 transitions
ULTA: 40 transitions
LKQ: 34 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_twotoone = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from within bounds to lower bound
    for i in range(1, len(returns)):
        if lower_bound <= returns.iloc[i-1] <= upper_bound and returns.iloc[i] < lower_bound:
            count += 1

    # Add count to dictionary
    counts_twotoone[stock] = count

# Print results
print("Number of transitions from returns within bounds to lower than lower bound for each stock:")
for stock, count in counts_twotoone.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns within bounds to lower than lower bound for each stock:
VRTX: 16 transitions
TDG: 20 transitions
PCAR: 22 transitions
FANG: 19 transitions
MRK: 20 transitions
GIS: 23 transitions
ORLY: 25 transitions
CLX: 25 transitions
ULTA: 22 transitions
LKQ: 22 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
```

```
counts_twotothree = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from within bounds to upper bound
    for i in range(1, len(returns)):
        if lower_bound <= returns.iloc[i-1] <= upper_bound and returns.iloc[i] > upper_bound:
            count += 1

    # Add count to dictionary
    counts_twotothree[stock] = count

# Print results
print("Number of transitions from returns within bounds to greater than upper bound for each stock:")
for stock, count in counts_twotothree.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns within bounds to greater than upper bound for each stock:
VRTX: 18 transitions
TDG: 18 transitions
PCAR: 29 transitions
FANG: 21 transitions
MRK: 38 transitions
GIS: 33 transitions
ORLY: 22 transitions
CLX: 20 transitions
ULTA: 23 transitions
LKQ: 22 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_threetoone = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from upper bound to outside bounds
    for i in range(1, len(returns)):
        if returns.iloc[i-1] > upper_bound and returns.iloc[i] < lower_bound:
            count += 1

    # Add count to dictionary
    counts_threetoone[stock] = count

# Print results
print("Number of transitions from returns greater than upper bound to less than lower bound for each stock:")
for stock, count in counts_threetoone.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns greater than upper bound to less than lower bound for each stock:
VRTX: 38 transitions
TDG: 38 transitions
PCAR: 32 transitions
```

```
FANG: 40 transitions
MRK: 26 transitions
GIS: 26 transitions
ORLY: 29 transitions
CLX: 33 transitions
ULTA: 33 transitions
LKQ: 34 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_threetotwo = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from upper bound to within bounds
    for i in range(1, len(returns)):
        if returns.iloc[i-1] > upper_bound and lower_bound <= returns.iloc[i] <= upper_bound:
            count += 1

    # Add count to dictionary
    counts_threetotwo[stock] = count

# Print results
print("Number of transitions from returns greater than upper bound to within bounds for each stock:")
for stock, count in counts_threetotwo.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns greater than upper bound to within bounds for each stock:
VRTX: 22 transitions
TDG: 23 transitions
PCAR: 29 transitions
FANG: 23 transitions
MRK: 33 transitions
GIS: 33 transitions
ORLY: 30 transitions
CLX: 19 transitions
ULTA: 31 transitions
LKQ: 23 transitions
```

```
# Define upper bound for returns
upper_bound = 0.005

# Initialize dictionary to hold results
counts_threetothree = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from upper bound to greater than upper bound
    for i in range(1, len(returns)):
        if returns.iloc[i-1] > upper_bound and returns.iloc[i] > upper_bound:
            count += 1

    # Add count to dictionary
    counts_threetothree[stock] = count
```

```
# Print results
print("Number of transitions from returns greater than upper bound to greater than upper bound for each stock:")
for stock, count in counts_threetothree.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns greater than upper bound to greater than upper bound for each stock:
VRTX: 53 transitions
TDG: 49 transitions
PCAR: 35 transitions
FANG: 49 transitions
MRK: 33 transitions
GIS: 29 transitions
ORLY: 43 transitions
CLX: 51 transitions
ULTA: 44 transitions
LKQ: 43 transitions
```

```
# Define upper bound for returns
lower_bound = -0.005

# Initialize dictionary to hold results
counts_onetoone = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]

    # Initialize counter
    count = 0

    # Loop through returns and count transitions from upper bound to greater than upper bound
    for i in range(1, len(returns)):
        if returns.iloc[i-1] < lower_bound and returns.iloc[i] < lower_bound:
            count += 1

    # Add count to dictionary
    counts_onetoone[stock] = count

# Print results
print("Number of transitions from returns less than lower bound to less than lower bound for each stock:")
for stock, count in counts_onetoone.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns less than lower bound to less than lower bound for each stock:
VRTX: 29 transitions
TDG: 36 transitions
PCAR: 31 transitions
FANG: 34 transitions
MRK: 24 transitions
GIS: 22 transitions
ORLY: 23 transitions
CLX: 26 transitions
ULTA: 28 transitions
LKQ: 27 transitions
```

```
# Define lower and upper bounds for returns
lower_bound = -0.005
upper_bound = 0.005

# Initialize dictionary to hold results
counts_twototwo = {}

# Define the list of stocks to loop through
stocks = ['VRTX', 'TDG', 'PCAR', 'FANG', 'MRK', 'GIS', 'ORLY', 'CLX', 'ULTA', 'LKQ']

# Loop through each stock in the list
for stock in stocks:
    # Get daily returns for current stock
    returns = daily_returns[stock]
```

```
    # Initialize counter
    count = 0

    # Loop through returns and count transitions within bounds
    for i in range(1, len(returns)):
        if lower_bound <= returns.iloc[i-1] <= upper_bound and lower_bound <= returns.iloc[i] <= upper_bound:
            count += 1

    # Add count to dictionary
    counts_twototwo[stock] = count

# Print results
print("Number of transitions from returns within bounds to returns within bounds for each stock:")
for stock, count in counts_twototwo.items():
    if stock in stocks:
        print(f"{stock}: {count} transitions")
```

```
Number of transitions from returns within bounds to returns within bounds for each stock:
VRTX: 18 transitions
TDG: 7 transitions
PCAR: 17 transitions
FANG: 3 transitions
MRK: 28 transitions
GIS: 33 transitions
ORLY: 22 transitions
CLX: 16 transitions
ULTA: 12 transitions
LKQ: 22 transitions
```

```
import pandas as pd

df_states = pd.concat([pd.Series(counts_onetoone), pd.Series(counts_onetotwo), pd.Series(counts_onetothree), pd.Series(counts_twototwo), pd.S
df_states.columns = ['onetoone', 'onetotwo', 'onetothree', 'twotoone', 'twototwo', 'twotothree', 'threetoone', 'threetotwo', 'threetothree']

print(df_states)
```

```
      onetoone  onetotwo  onetothree  twotoone  twototwo  twotothree  \
VRTX        29        13          41        18        16          18
TDG         36        15          42         7        20          18
PCAR        31        22          31        17        22          29
FANG        34        17          42         3        19          21
MRK         24        26          20        28        20          38
GIS         22        24          25        33        23          33
ORLY        23        18          36        22        25          22
CLX         26        26          32        16        25          20
ULTA        28        15          40        12        22          23
LKQ         27        21          34        22        22          22

      threetoone  threetotwo  threetothree
VRTX          38          22            53
TDG           38          23            49
PCAR          32          29            35
FANG          40          23            49
MRK           26          33            33
GIS           26          33            29
ORLY          29          30            43
CLX           33          19            51
ULTA          33          31            44
LKQ           34          23            43
```

```
df_days
```

	negative	neutral	positive
VRTX	83	53	113

```
dfs = []
for i in range(10):
    df = [[df_states['onetoone'][i]/df_days['negative'][i], df_states['onetotwo'][i]/df_days['negative'][i], df_states['onetothree'][i]/df_da
          [df_states['twotoone'][i]/df_days['neutral'][i], df_states['twototwo'][i]/df_days['neutral'][i], df_states['twotothree'][i]/df_days
          [df_states['threetoone'][i]/df_days['positive'][i], df_states['threetotwo'][i]/df_days['positive'][i], df_states['threetothree'][i]
    dfs.append(pd.DataFrame(df))
```

```
for i in range(len(dfs)):
  print("Probability Transition Matrix for stock ", df_days.iloc[i].name, "\n", dfs[i])
```

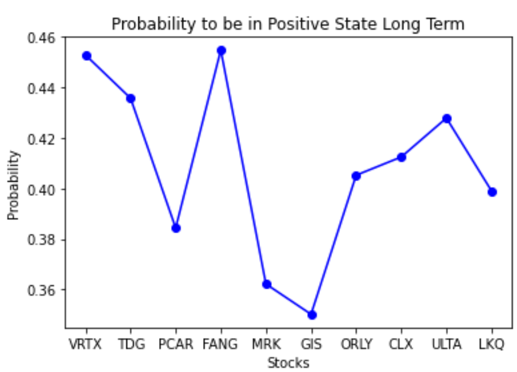
```
    Probability Transition Matrix for stock  VRTX
            0         1         2
0  0.349398  0.156627  0.493976
1  0.339623  0.301887  0.339623
2  0.336283  0.194690  0.469027
    Probability Transition Matrix for stock  TDG
            0         1         2
0  0.382979  0.159574  0.446809
1  0.155556  0.444444  0.400000
2  0.345455  0.209091  0.445455
    Probability Transition Matrix for stock  PCAR
            0         1         2
0  0.364706  0.258824  0.364706
1  0.250000  0.323529  0.426471
2  0.333333  0.302083  0.364583
    Probability Transition Matrix for stock  FANG
            0         1         2
0  0.361702  0.180851  0.446809
1  0.069767  0.441860  0.488372
2  0.357143  0.205357  0.437500
    Probability Transition Matrix for stock  MRK
            0         1         2
0  0.342857  0.371429  0.285714
1  0.321839  0.229885  0.436782
2  0.282609  0.358696  0.358696
    Probability Transition Matrix for stock  GIS
            0         1         2
0  0.309859  0.338028  0.352113
1  0.366667  0.255556  0.366667
2  0.295455  0.375000  0.329545
    Probability Transition Matrix for stock  ORLY
            0         1         2
0  0.298701  0.233766  0.467532
1  0.314286  0.357143  0.314286
2  0.284314  0.294118  0.421569
    Probability Transition Matrix for stock  CLX
            0         1         2
0  0.309524  0.309524  0.380952
1  0.262295  0.409836  0.327869
2  0.317308  0.182692  0.490385
    Probability Transition Matrix for stock  ULTA
            0         1         2
0  0.337349  0.180723  0.481928
1  0.206897  0.379310  0.396552
2  0.305556  0.287037  0.407407
    Probability Transition Matrix for stock  LKQ
            0         1         2
0  0.325301  0.253012  0.409639
1  0.333333  0.333333  0.333333
2  0.340000  0.230000  0.430000
```

```
import numpy as np
stationaries=[]
# Define the probability transition matrix P
for i in range(len(dfs)):
  P = dfs[i]

  # Compute the transpose of P
  PT = P.T

  # Compute the eigenvalues and eigenvectors of PT
  eig_vals, eig_vecs = np.linalg.eig(PT)

  # Find the index of the eigenvalue with a value of 1
  index = np.argmin(np.abs(eig_vals - 1))
```

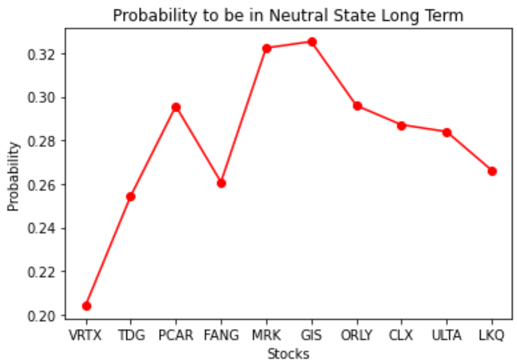
```
  # Extract the corresponding eigenvector
  stationary = eig_vecs[:, index].real

  # Normalize the eigenvector
  stationary = stationary / np.sum(stationary)
  stationaries.append(stationary)
  print("stationary distribution of ",df_days.iloc[i].name, stationaries[i])
```

```
stationary distribution of  VRTX [0.34278249 0.2043344  0.45288311]
stationary distribution of  TDG [0.30977301 0.25448453 0.43574246]
stationary distribution of  PCAR [0.31993319 0.29569762 0.38436919]
stationary distribution of  FANG [0.2843297  0.26087662 0.45479368]
stationary distribution of  MRK [0.31542873 0.32238054 0.36219073]
stationary distribution of  GIS [0.32446809 0.32532187 0.35021004]
stationary distribution of  ORLY [0.29874676 0.29599459 0.40525865]
stationary distribution of  CLX [0.30036404 0.2871515  0.41248446]
stationary distribution of  ULTA [0.28810679 0.28400384 0.42788937]
stationary distribution of  LKQ [0.33465502 0.26629158 0.3990534 ]
```

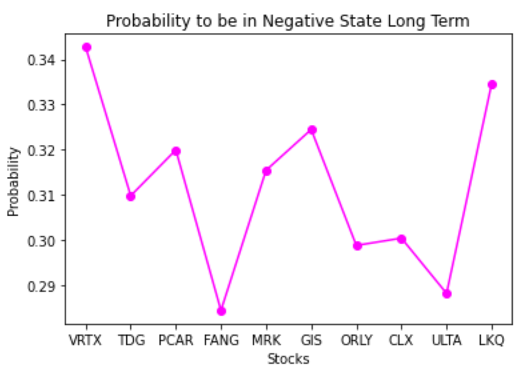
```
import matplotlib.pyplot as plt

# Define the x-axis values
x = df_days.index.values

# Define the y-axis values (the third entry of each stationary distribution)
y = [stationary[2] for stationary in stationaries]

# Plot the line graph
plt.plot(x, y, color='blue', marker='o')

# Set the title and axis labels
plt.title('Probability to be in Positive State Long Term')
plt.xlabel('Stocks')
plt.ylabel('Probability')

# Display the plot
plt.show()
```




```
import matplotlib.pyplot as plt

# Define the x-axis values
x = df_days.index.values

# Define the y-axis values (the third entry of each stationary distribution)
y = [stationary[1] for stationary in stationaries]

# Plot the line graph
plt.plot(x, y, color='red', marker='o')

# Set the title and axis labels
plt.title('Probability to be in Neutral State Long Term')
plt.xlabel('Stocks')
plt.ylabel('Probability')

# Display the plot
plt.show()
```




```
import matplotlib.pyplot as plt

# Define the x-axis values
x = df_days.index.values

# Define the y-axis values (the third entry of each stationary distribution)
y = [stationary[0] for stationary in stationaries]

# Plot the line graph
plt.plot(x, y, color='magenta', marker='o')

# Set the title and axis labels
plt.title('Probability to be in Negative State Long Term')
plt.xlabel('Stocks')
plt.ylabel('Probability')

# Display the plot
plt.show()
```



End