**CHAPTER FIVE**

**Summary, Conclusion and Recommendations**

**5.1 Summary of Findings**

The research focused on evaluating the effectiveness of three primary methods for calculating Value-at-Risk (VaR) for Nigerian investment portfolios: the variance-covariance method, historical simulation, and Monte Carlo simulation. The analysis involved assessing the accuracy of each method using a dataset comprising real market data from the Nigerian Stock Exchange. Key performance indicators included the T-statistic and p-value results from the T-tests, comparing predicted VaR values with actual market risk exposure.

The findings indicated that the Monte Carlo simulation method exhibited the most robust performance, closely aligning with the actual VaR values and providing a more comprehensive risk estimate by simulating a wide array of market conditions. In contrast, the variance-covariance method, though easy to implement, often underestimated the risk due to its assumptions of normally distributed returns and linearity, which do not always hold true in the volatile Nigerian market. The historical simulation method, while grounded in real market data, struggled with predictive accuracy when market conditions deviated from historical patterns. Overall, the study highlighted the need for flexible and dynamic risk assessment tools that can adapt to the unique characteristics of the Nigerian investment landscape.

**5.2 Conclusion**

The study concluded that the Monte Carlo simulation is the most effective method for calculating VaR in Nigerian investment portfolios. This method’s strength lies in its ability to model a wide range of possible future outcomes based on random sampling, which captures the complexity and variability of market conditions better than the other methods assessed. The variance-covariance approach, while methodologically simpler, often falls short in markets that experience frequent and significant fluctuations, as seen in Nigeria. Similarly, the historical simulation method’s reliance on past data limits its predictive accuracy when future market conditions diverge from historical norms.

The findings underscore the importance of selecting an appropriate VaR calculation method that not only aligns with the statistical characteristics of the investment market but also accommodates the specific risks present in that market. For Nigerian investors and risk managers, utilizing the Monte Carlo simulation approach can provide a more reliable risk assessment, allowing for better-informed decision-making processes that are critical in managing investment portfolios effectively in a dynamic market environment.

**5.3 Recommendations**

Based on the findings and conclusions of this research, several recommendations are proposed:

1. **Adoption of Monte Carlo Simulation for Risk Management**: Investment firms and risk managers in Nigeria should prioritize the adoption of Monte Carlo simulation techniques for VaR calculations. This approach provides a more nuanced understanding of risk exposure, accounting for the inherent volatility and unpredictability of the Nigerian market.
2. **Regular Model Updates with Current Data:** To enhance the accuracy of VaR estimations, it is crucial to continuously update the risk models with the most recent market data. This will ensure that the risk assessments reflect current market conditions, thereby improving the predictive power of the models.
3. **Integration of Python-Based Tools for Efficiency:** To improve the efficiency and accuracy of risk calculations, it is recommended that investment firms integrate Python-based analytical tools into their risk management frameworks. Python’s powerful data analysis libraries can facilitate faster, more accurate computations and enable the automation of complex risk assessments.
4. **Training and Capacity Building:** It is essential to invest in training programs for financial analysts and risk managers to build their capacity in using advanced risk management tools such as Monte Carlo simulations and Python-based analysis. This will help to ensure that the human resources within investment firms are well-equipped to utilize these sophisticated tools effectively.
5. **Periodic Review and Stress Testing**: To complement VaR calculations, firms should also conduct periodic stress testing and scenario analysis. This will help in understanding the potential impact of extreme market conditions, thus providing a more comprehensive risk management strategy.
6. **Policy Implications for Regulatory Bodies:** Regulatory bodies in Nigeria should consider setting guidelines or recommendations for the adoption of advanced risk assessment methods like Monte Carlo simulation in financial institutions. This will help standardize risk management practices across the industry, leading to more robust financial stability in the investment sector.

By implementing these recommendations, Nigerian investors and financial institutions can significantly improve their risk management practices, leading to better investment decisions and enhanced protection against market volatility.

**Appendices:**

All codes and data are in https://github.com/Theideabased/value\_at\_risk\_ngx.git

# CALCULATING VALUE AT RISK

!git clone https://github.com/Theideabased/value\_at\_risk\_ngx.git

Cloning into 'value\_at\_risk\_ngx'...  
remote: Enumerating objects: 37, done.ote: Counting objects: 100% (37/37), done.ote: Compressing objects: 100% (34/34), done.ote: Total 37 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)

The next code will import al the data that we have downloaded from the investing.com the it will make the Date format Day-Month-Year course in python the default format for date is Month-Day-Year.

import pandas as pd  
import numpy as np  
import datetime as dt  
import matplotlib.pyplot as plt  
import scipy.stats as stats  
import os

directories = os.listdir('/content/value\_at\_risk\_ngx/data')  
directories

['GUINNES Historical Data.csv',  
 'DANGCEM Historical Data.csv',  
 'NB Historical Data.csv',  
 'JBERGER Historical Data.csv',  
 'README.md',  
 'AIICO Historical Data.csv',  
 'NESTLE Historical Data.csv',  
 'ZENITHB Historical Data.csv',  
 'UBA Historical Data.csv',  
 'NSE All Share Historical Data (1).csv',  
 'UNILEVE Historical Data.csv']

aiico\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/AIICO Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
dangcem\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/DANGCEM Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
guiness\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/GUINNES Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
jberger\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/JBERGER Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
nestle\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/NESTLE Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
nse\_all\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/NSE All Share Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
unilever\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/UNILEVE Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
zenith\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/ZENITHB Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
nb\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/NB Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
uba\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/data/UBA Historical Data.csv', parse\_dates=['Date'], dayfirst=True)

aiico\_data.head()

Although for our analysis adjusted close prices is the best to use for variance covariance method as it put into consideration the dividends and stock split but since Nigerian stock does not have adjusted close prices we will have to use the normal close prices for our evaluation

# Using the close price which is what we will use for the variance covariance cal  
# Calculations  
aiico\_data['aiico'] = aiico\_data['Price']  
aiico\_data\_close = aiico\_data[['Date', 'aiico']]  
dangcem\_data['dangcem'] = dangcem\_data['Price']  
guiness\_data['guiness'] = guiness\_data['Price']  
jberger\_data['jberger'] = jberger\_data['Price']  
nestle\_data['nestle'] = nestle\_data['Price']  
nse\_all\_data['nse\_all'] = nse\_all\_data['Price']  
unilever\_data['unilever'] = unilever\_data['Price']  
zenith\_data['zenith'] = zenith\_data['Price']  
nb\_data['nb'] = nb\_data['Price']  
uba\_data['uba'] = uba\_data['Price']  
aiico\_data\_close.head()

For us to start our analysis we need to put all the data into one table so that it will be easy for analysis and we will be able to calculate all the datas in at once to avoid redundancy

stock\_close\_data = pd.concat([aiico\_data\_close, dangcem\_data['dangcem'],\  
 guiness\_data['guiness'], jberger\_data['jberger'],\  
 nestle\_data['nestle'], nse\_all\_data['nse\_all'],\  
 unilever\_data['unilever'], zenith\_data['zenith'],\  
 nb\_data['nb'],uba\_data['uba']], axis=1)  
stock\_close\_data.set\_index('Date', inplace=True)  
stock\_close\_data.head()

some of the data are type string so we have to change it to float so that we will be able to make use it to calculate our value at risk effectively

stock\_close\_data.dtypes[stock\_close\_data.dtypes == 'object']  
stock\_close\_data.dropna()  
stock\_close\_data['nestle'] = stock\_close\_data['nestle'].str.replace(',', '').astype(float)  
stock\_close\_data['nse\_all'] = stock\_close\_data['nse\_all'].str.replace(',', '').astype(float)  
# stock\_close\_data[['nestle, nse\_all']] = stock\_close\_data[['nestle', 'nse\_all']].astype(float)  
stock\_close\_data.dtypes

aiico float64  
dangcem float64  
guiness float64  
jberger float64  
nestle float64  
nse\_all float64  
unilever float64  
zenith float64  
nb float64  
uba float64  
dtype: object

## Getting the Correlation matrix

# Correlation matrix  
# Calculate the correlation matrix  
correlation\_matrix = stock\_close\_data.corr()  
  
# Display the correlation matrix  
correlation\_matrix.to\_csv('correlation\_matrix.csv')  
correlation\_matrix

# Assuming stock\_close\_data is your DataFrame and Date is the index  
# If Date is not set as the index, you can set it using:  
# stock\_close\_data.set\_index('Date', inplace=True)  
  
# Number of columns  
num\_columns = len(stock\_close\_data.columns)  
  
# Creating subplots with a grid that can accommodate all plots  
fig, axes = plt.subplots(nrows=num\_columns, ncols=1, figsize=(14, 5 \* num\_columns), sharex=True)  
  
# Plotting each column against Date in a separate subplot  
for i, column in enumerate(stock\_close\_data.columns):  
 axes[i].plot(stock\_close\_data.index, stock\_close\_data[column], label=column)  
 axes[i].set\_title(f'Time Series of {column} Closing Prices')  
 axes[i].set\_xlabel('Date')  
 axes[i].set\_ylabel('Closing Price')  
 axes[i].legend(loc='upper left')  
 axes[i].grid(True)  
  
# Adjust layout to prevent overlap  
plt.tight\_layout()  
  
# Display the plots  
plt.show()

# Descriptive statistics  
stock\_describe = stock\_close\_data.describe()  
stock\_skew = stock\_close\_data.skew()  
stock\_kurtosis = stock\_close\_data.kurtosis()  
stock\_skew\_df = pd.DataFrame({"skewness" : stock\_skew})  
stock\_kurtosis\_df = pd.DataFrame({"kurtosis" : stock\_kurtosis})  
stock\_describe = pd.concat([stock\_describe, stock\_skew\_df, stock\_kurtosis\_df], axis=1)  
  
stock\_describe.to\_csv('stock\_describe.csv')  
stock\_describe

## Calculate the daily log returns and drop any null value(i.e, columns that have no values or days that has not closing return)

log\_returns = np.log(stock\_close\_data / stock\_close\_data.shift(1))  
log\_returns.dropna(inplace=True)  
log\_returns.head()

## Create an equally weighted portfolio

portfolio\_value = 100000  
weights = np.array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])  
print(weights)

[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]

# USING VARIANCE-COVARIANCE

# Calculate the Historical Portfolio returns

# historical\_returns = log\_returns.dot(weights).sum(axis=1)  
historical\_returns = (log\_returns \* weights).sum(axis=1)  
print(historical\_returns)

days = 5  
historical\_x\_day\_returns = historical\_returns.rolling(window=days).sum()  
# historical\_x\_day\_returns

# Create a covariance matrix for all the securities

There are 252 trading days in a year so to annualize our covariance matrix we have to multiply it by 252

cov\_matrix = log\_returns.cov()\*252

## Calculate portfolio standard deviation

portfolio\_std\_dev = np.sqrt(weights.T @ cov\_matrix @ weights)

## Set different confidence levels to visualize

confidence\_levels = [0.9, 0.95, 0.99]

## Calculate VaR at different confidence levels

from scipy.stats import norm  
  
va\_cov\_VaRs = []  
for cl in confidence\_levels:  
 VaR = portfolio\_value \* portfolio\_std\_dev \* norm.ppf(cl) \* np.sqrt(days/252)  
 va\_cov\_VaRs.append(VaR)

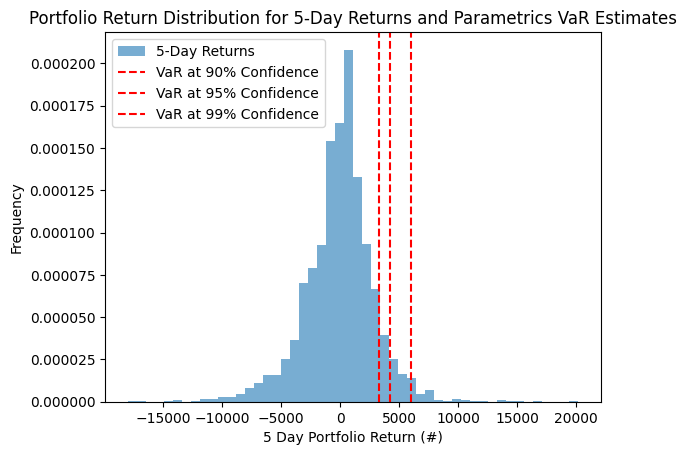
## Print out the VaRs result

print(f'{"Confidence Level":<20} {"Value at Risk":<20}')  
print("-" \* 40)  
# print each confidence interval and it corresponding VaR  
for cl, VaR in zip(confidence\_levels, va\_cov\_VaRs):  
 print(f'{cl \* 100:<20} {VaR:<20.2f}')

Confidence Level Value at Risk   
----------------------------------------  
90.0 3318.40   
95.0 4259.12   
99.0 6023.75

## Set labels, title, and legend

# Conver returns to naira values for the histogram  
historical\_x\_day\_returns\_naira = historical\_x\_day\_returns \* portfolio\_value  
  
# Plot the histogram  
plt.hist(historical\_x\_day\_returns\_naira, bins=50, density=True, alpha=0.6, label=f'{days}-Day Returns')  
  
# Add vertical lines represnting VaR at each confidence level  
for cl, VaR in zip(confidence\_levels, va\_cov\_VaRs):  
 plt.axvline(x=VaR, color='r', linestyle='--', label='VaR at {}% Confidence'.format(int(cl \* 100)))  
  
plt.xlabel(f'{days} Day Portfolio Return (#)')  
plt.ylabel('Frequency')  
plt.title(f'Portfolio Return Distribution for {days}-Day Returns and Parametrics VaR Estimates')  
plt.legend()  
plt.show()



# USING MONTE-CARLO SIMULATION

log\_returns.head()

def expected\_return(weights, log\_returns):  
 return np.sum(log\_returns.mean()\*weights)  
  
def standard\_deviation(weights, cov\_matrix):  
 variance = weights.T @ cov\_matrix @ weights  
 return np.sqrt(variance)

## Doing the covarinace matrix

cov\_matrix = log\_returns.cov()  
print(cov\_matrix)

aiico dangcem guiness jberger nestle nse\_all \  
aiico 0.001471 -0.000006 0.000066 0.000038 0.000014 0.000054   
dangcem -0.000006 0.000464 0.000051 0.000011 0.000058 0.000135   
guiness 0.000066 0.000051 0.000754 0.000046 0.000018 0.000049   
jberger 0.000038 0.000011 0.000046 0.000803 0.000026 0.000027   
nestle 0.000014 0.000058 0.000018 0.000026 0.000401 0.000053   
nse\_all 0.000054 0.000135 0.000049 0.000027 0.000053 0.000097   
unilever 0.000033 0.000025 0.000056 0.000034 0.000033 0.000040   
zenith 0.000189 0.000069 0.000086 0.000059 0.000051 0.000127   
nb 0.000045 0.000070 0.000063 0.000053 0.000057 0.000097   
uba 0.000189 0.000068 0.000088 0.000049 0.000041 0.000121   
  
 unilever zenith nb uba   
aiico 0.000033 0.000189 0.000045 0.000189   
dangcem 0.000025 0.000069 0.000070 0.000068   
guiness 0.000056 0.000086 0.000063 0.000088   
jberger 0.000034 0.000059 0.000053 0.000049   
nestle 0.000033 0.000051 0.000057 0.000041   
nse\_all 0.000040 0.000127 0.000097 0.000121   
unilever 0.000877 0.000102 0.000038 0.000075   
zenith 0.000102 0.000632 0.000152 0.000349   
nb 0.000038 0.000152 0.000656 0.000142   
uba 0.000075 0.000349 0.000142 0.000769

## Create an equally weighted portfolio and find total portfolio expected return

portfolio\_value = 1000000  
weights = np.array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])  
portfolio\_expected\_return = expected\_return(weights, log\_returns)  
portfolio\_standard\_deviation = standard\_deviation(weights, cov\_matrix)

# Create a function that gives a random Z-score based on normal distribution

def random\_z\_score():  
 return np.random.normal(0, 1)

## Create a function to calsulate secnarioGainloss

days = 5  
def scenario\_gain\_loss(portfolio\_value, portfolio\_standard\_deviation, z\_score, days):  
 return portfolio\_value \* portfolio\_expected\_return \* days + portfolio\_value \* portfolio\_standard\_deviation \* z\_score \* np.sqrt(days)

## Run 100,000 simulations

simulations = 100000  
scenerioReturn = []  
for i in range(simulations):  
 z\_score = random\_z\_score()  
 scenerioReturn.append(scenario\_gain\_loss(portfolio\_value, portfolio\_standard\_deviation, z\_score, days))

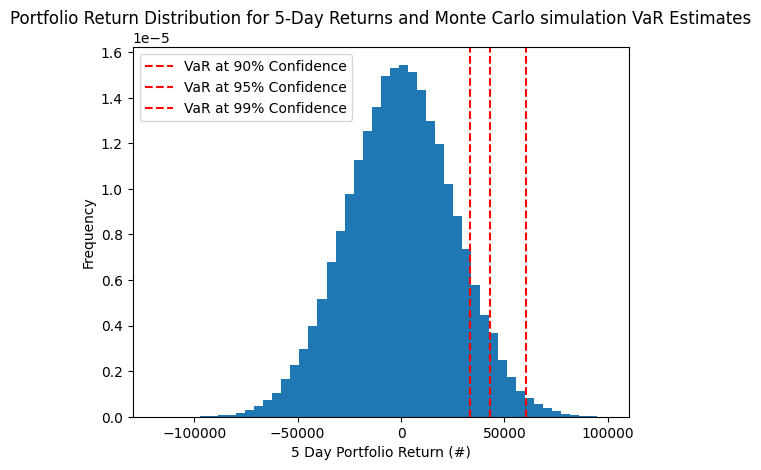
## Specify different confidence interval and calculate the Value at Risk (VaR) the common one is 0.9, 0.95, 0.99

confidence\_levels = [0.9, 0.95, 0.99]  
mc\_VaRs = []  
for cl in confidence\_levels:  
 VaR = -np.percentile(scenerioReturn, (1 - cl) \* 100)  
 mc\_VaRs.append(VaR)  
  
print(f'{"Confidence Level":<20} {"Value at Risk":<20}')  
print("-" \* 40)  
# print each confidence interval and it corresponding VaR  
for cl, VaR in zip(confidence\_levels, mc\_VaRs):  
 print(f'{cl \* 100:<20} {VaR:<20.2f}')

Confidence Level Value at Risk   
----------------------------------------  
90.0 33426.18   
95.0 42857.37   
99.0 60325.89

## Plot the results of all 100000 scenarios

plt.hist(scenerioReturn, bins=50, density=True)  
  
# Add vertical lines represnting VaR at each confidence level  
for cl, VaR in zip(confidence\_levels, mc\_VaRs):  
 plt.axvline(x=VaR, color='r', linestyle='--', label='VaR at {}% Confidence'.format(int(cl \* 100)))  
  
plt.xlabel(f'{days} Day Portfolio Return (#)')  
plt.ylabel('Frequency')  
plt.title(f'Portfolio Return Distribution for {days}-Day Returns and Monte Carlo simulation VaR Estimates')  
plt.legend()  
plt.show()



# USING HISTORICAL METHOD

## calculate the historical portfolio returns

historical\_returns = (log\_returns \* weights).sum(axis=1)  
print(historical\_returns)

Date  
2024-07-16 0.004348  
2024-07-15 -0.002739  
2024-07-12 -0.005773  
2024-07-11 0.017418  
2024-07-10 -0.002695  
 ...   
2015-01-09 0.015111  
2015-01-08 -0.003649  
2015-01-07 0.019395  
2015-01-06 0.056209  
2015-01-05 0.057262  
Length: 2359, dtype: float64

## find the X day historical returns

range\_returns = historical\_returns.rolling(window=days).sum()  
range\_returns = range\_returns.dropna()  
print(range\_returns)

Date  
2024-07-10 0.010558  
2024-07-09 0.008731  
2024-07-08 0.019577  
2024-07-05 0.020226  
2024-07-04 0.010330  
 ...   
2015-01-09 0.048528  
2015-01-08 0.044377  
2015-01-07 0.075130  
2015-01-06 0.099781  
2015-01-05 0.144328  
Length: 2355, dtype: float64

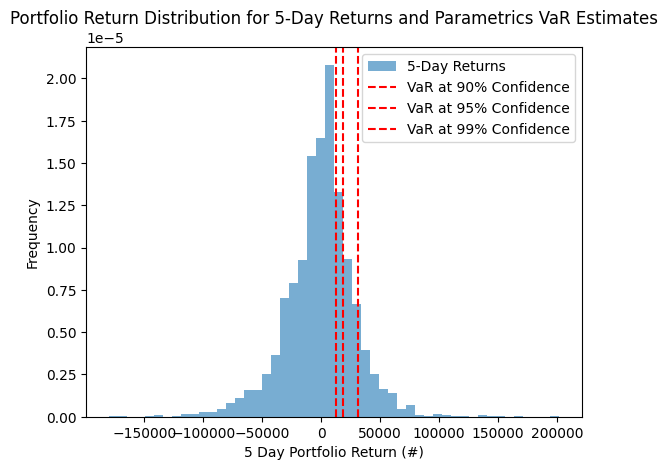
## Specify the confidence interval for the VaR for historical method

confidence\_levels = [0.9, 0.95, 0.99]  
range\_returns = pd.Series(historical\_returns).dropna()  
hist\_VaRs = []  
for cl in confidence\_levels:  
 VaR = -np.percentile(range\_returns, (1 - cl) \* 100)\*portfolio\_value  
 hist\_VaRs.append(VaR)  
  
print(f'{"Confidence Level":<20} {"Value at Risk":<20}')  
print("-" \* 40)  
# print each confidence interval and it corresponding VaR  
for cl, VaR in zip(confidence\_levels, hist\_VaRs):  
 print(f'{cl \* 100:<20} {VaR:<20.2f}')

Confidence Level Value at Risk   
----------------------------------------  
90.0 12442.04   
95.0 18860.06   
99.0 30827.11

## Plot the results of the historical returns

return\_window = days  
range\_returns = historical\_returns.rolling(window=return\_window).sum()  
range = range\_returns.dropna()  
range\_returns\_naira = range\_returns \* portfolio\_value  
  
plt.hist(range\_returns\_naira, bins=50, density=True, alpha=0.6, label=f'{return\_window}-Day Returns')  
for cl, VaR in zip(confidence\_levels, hist\_VaRs):  
 plt.axvline(x=VaR, color='r', linestyle='--', label='VaR at {}% Confidence'.format(int(cl \* 100)))  
  
plt.xlabel(f'{days} Day Portfolio Return (#)')  
plt.ylabel('Frequency')  
plt.title(f'Portfolio Return Distribution for {days}-Day Returns and Parametrics VaR Estimates')  
plt.legend()  
plt.show()



directories = os.listdir('/content/value\_at\_risk\_ngx/test\_data')  
directories

['GUINNES Historical Data (1).csv',  
 'NB Historical Data (2).csv',  
 'ZENITHB Historical Data (1).csv',  
 'UBA Historical Data (1).csv',  
 'UNILEVE Historical Data (1).csv',  
 'NESTLE Historical Data (1).csv',  
 'README.md',  
 'JBERGER Historical Data (1).csv',  
 'NSE All Share Historical Data.csv',  
 'DANGCEM Historical Data (2).csv',  
 'AIICO Historical Data (1).csv']

aiico\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/AIICO Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
dangcem\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/DANGCEM Historical Data (2).csv', parse\_dates=['Date'], dayfirst=True)  
guiness\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/GUINNES Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
jberger\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/JBERGER Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
nestle\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/NESTLE Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
nse\_all\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/NSE All Share Historical Data.csv', parse\_dates=['Date'], dayfirst=True)  
unilever\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/UNILEVE Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
zenith\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/ZENITHB Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)  
nb\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/NB Historical Data (2).csv', parse\_dates=['Date'], dayfirst=True)  
uba\_test\_data = pd.read\_csv('/content/value\_at\_risk\_ngx/test\_data/UBA Historical Data (1).csv', parse\_dates=['Date'], dayfirst=True)

aiico\_test\_data['aiico'] = aiico\_test\_data['Price']  
aiico\_test\_data\_close = aiico\_test\_data[['Date', 'aiico']]  
dangcem\_test\_data['dangcem'] = dangcem\_test\_data['Price']  
guiness\_test\_data['guiness'] = guiness\_test\_data['Price']  
jberger\_test\_data['jberger'] = jberger\_test\_data['Price']  
nestle\_test\_data['nestle'] = nestle\_test\_data['Price']  
nse\_all\_test\_data['nse\_all'] = nse\_all\_test\_data['Price']  
unilever\_test\_data['unilever'] = unilever\_test\_data['Price']  
zenith\_test\_data['zenith'] = zenith\_test\_data['Price']  
nb\_test\_data['nb'] = nb\_test\_data['Price']  
uba\_test\_data['uba'] = uba\_test\_data['Price']  
aiico\_test\_data\_close.head()

stock\_test\_close\_data = pd.concat([aiico\_test\_data\_close, dangcem\_test\_data['dangcem'],\  
 guiness\_test\_data['guiness'], jberger\_test\_data['jberger'],\  
 nestle\_test\_data['nestle'], nse\_all\_test\_data['nse\_all'],\  
 unilever\_test\_data['unilever'], zenith\_test\_data['zenith'],\  
 nb\_test\_data['nb'],uba\_test\_data['uba']], axis=1)  
stock\_test\_close\_data.set\_index('Date', inplace=True)  
stock\_test\_close\_data.head()

stock\_test\_close\_data.dtypes[stock\_test\_close\_data.dtypes == 'object']  
stock\_test\_close\_data.dropna()  
# stock\_test\_close\_data['nestle'] = stock\_test\_close\_data['nestle'].str.replace(',', '').astype(float)  
stock\_test\_close\_data['nse\_all'] = stock\_test\_close\_data['nse\_all'].str.replace(',', '').astype(float)  
# stock\_close\_data[['nestle, nse\_all']] = stock\_close\_data[['nestle', 'nse\_all']].astype(float)  
stock\_test\_close\_data.dtypes

aiico float64  
dangcem float64  
guiness float64  
jberger float64  
nestle float64  
nse\_all float64  
unilever float64  
zenith float64  
nb float64  
uba float64  
dtype: object

Getting the log return for the test data test data and using the historical method to get the value at risk at each time

test\_log\_returns = np.log(stock\_test\_close\_data / stock\_test\_close\_data.shift(1))  
test\_log\_returns.dropna(inplace=True)  
weights = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]  
test\_historical\_returns = (test\_log\_returns \* weights).sum(axis=1)  
print(test\_historical\_returns)

Date  
2024-09-06 0.004545  
2024-09-05 -0.001329  
2024-09-04 0.008398  
2024-09-03 0.001536  
2024-09-02 0.010189  
2024-08-30 -0.006246  
2024-08-29 -0.003376  
2024-08-28 -0.009596  
2024-08-27 -0.015653  
2024-08-26 -0.007942  
2024-08-23 0.005208  
2024-08-22 0.000175  
2024-08-21 -0.004576  
2024-08-20 0.001392  
2024-08-19 -0.001529  
2024-08-16 0.017940  
2024-08-15 0.001597  
2024-08-14 -0.006799  
2024-08-13 0.007303  
2024-08-12 0.000368  
2024-08-09 -0.005731  
2024-08-08 -0.032114  
2024-08-07 -0.027556  
2024-08-06 0.003509  
2024-08-05 -0.004097  
2024-08-02 -0.008137  
2024-08-01 -0.009441  
2024-07-31 -0.007052  
2024-07-30 0.021166  
2024-07-29 0.021773  
2024-07-26 0.006806  
2024-07-25 0.006851  
2024-07-24 0.004996  
2024-07-23 0.006859  
2024-07-22 0.003231  
2024-07-19 -0.004906  
2024-07-18 0.005272  
dtype: float64

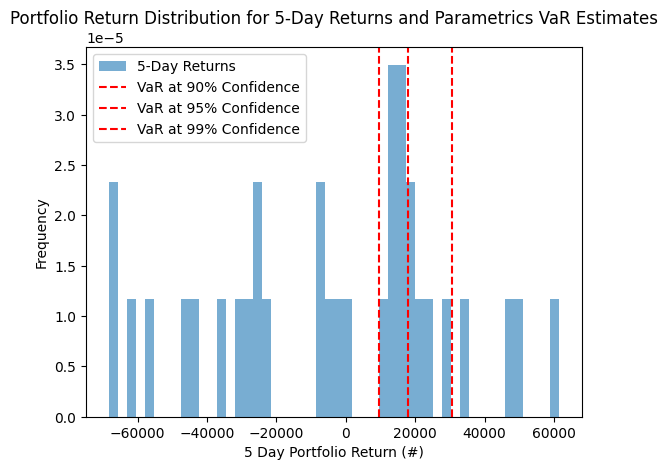
portfolio\_value = 1000000  
days = 5  
test\_range\_returns = test\_historical\_returns.rolling(window=days).sum()  
test\_range\_returns = test\_range\_returns.dropna()  
print(test\_range\_returns)

Date  
2024-09-02 0.023339  
2024-08-30 0.012548  
2024-08-29 0.010502  
2024-08-28 -0.007492  
2024-08-27 -0.024682  
2024-08-26 -0.042813  
2024-08-23 -0.031358  
2024-08-22 -0.027808  
2024-08-21 -0.022787  
2024-08-20 -0.005742  
2024-08-19 0.000671  
2024-08-16 0.013402  
2024-08-15 0.014824  
2024-08-14 0.012601  
2024-08-13 0.018513  
2024-08-12 0.020409  
2024-08-09 -0.003262  
2024-08-08 -0.036973  
2024-08-07 -0.057730  
2024-08-06 -0.061524  
2024-08-05 -0.065989  
2024-08-02 -0.068395  
2024-08-01 -0.045722  
2024-07-31 -0.025218  
2024-07-30 -0.007562  
2024-07-29 0.018308  
2024-07-26 0.033252  
2024-07-25 0.049543  
2024-07-24 0.061591  
2024-07-23 0.047284  
2024-07-22 0.028742  
2024-07-19 0.017030  
2024-07-18 0.015452  
dtype: float64

confidence\_levels = [0.9, 0.95, 0.99]  
test\_range\_returns = pd.Series(test\_historical\_returns).dropna()  
test\_VaRs = []  
for cl in confidence\_levels:  
 VaR = -np.percentile(test\_range\_returns, (1 - cl) \* 100)\*portfolio\_value  
 test\_VaRs.append(VaR)  
  
print(f'{"Confidence Level":<20} {"Value at Risk":<20}')  
print("-" \* 40)  
# print each confidence interval and it corresponding VaR2  
for cl, VaR in zip(confidence\_levels, test\_VaRs):  
 print(f'{cl \* 100:<20} {VaR:<20.2f}')

Confidence Level Value at Risk   
----------------------------------------  
90.0 9503.17   
95.0 18033.46   
99.0 30472.96

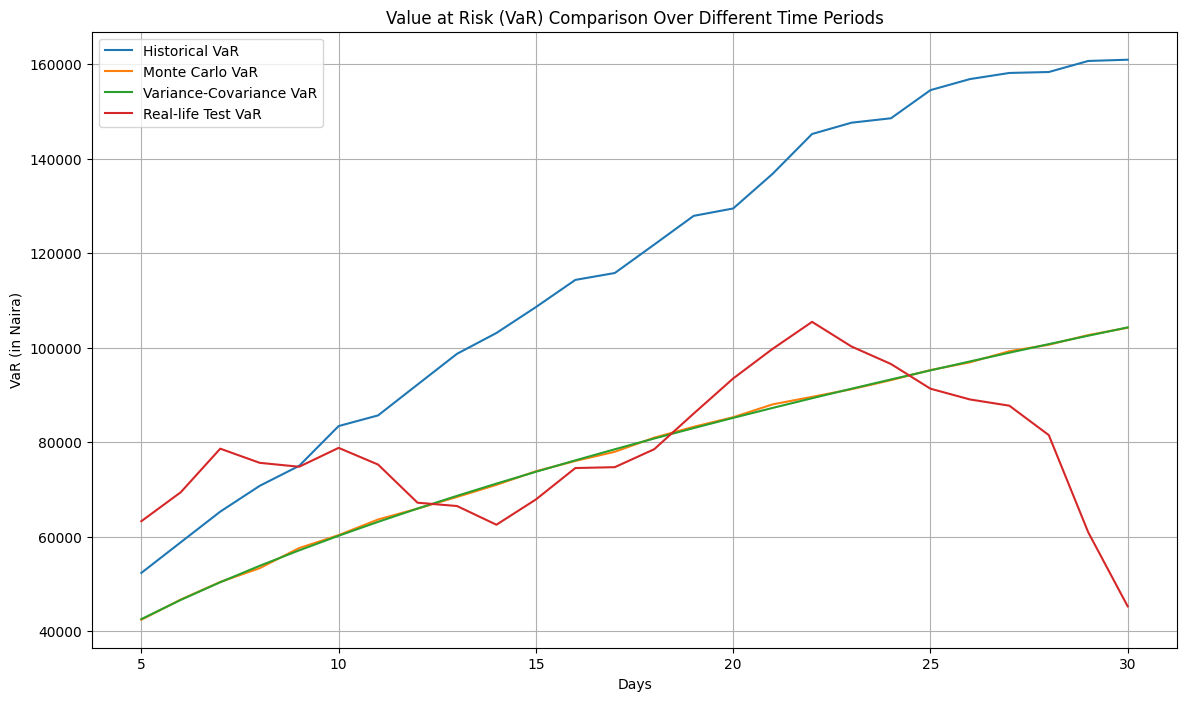
return\_window = days  
test\_range\_returns = test\_historical\_returns.rolling(window=return\_window).sum()  
test\_range = test\_range\_returns.dropna()  
test\_range\_returns\_naira = test\_range\_returns \* portfolio\_value  
  
plt.hist(test\_range\_returns\_naira, bins=50, density=True, alpha=0.6, label=f'{return\_window}-Day Returns')  
for cl, VaR in zip(confidence\_levels, test\_VaRs):  
 plt.axvline(x=VaR, color='r', linestyle='--', label='VaR at {}% Confidence'.format(int(cl \* 100)))  
  
plt.xlabel(f'{days} Day Portfolio Return (#)')  
plt.ylabel('Frequency')  
plt.title(f'Portfolio Return Distribution for {days}-Day Returns and Parametrics VaR Estimates')  
plt.legend()  
plt.show()



portfolio\_value = 1000000 # Example portfolio value  
confidence\_interval = 0.95  
simulations = 100000  
  
# Lists to store VaR results for each method  
hist\_VaRs = []  
va\_cov\_VaRs = []  
mc\_VaRs = []  
test\_VaRs = []  
  
# Define a function to generate random z-scores for Monte Carlo simulation  
def random\_z\_score():  
 return np.random.normal()  
  
# Define a function to simulate gain/loss in a scenario  
def scenario\_gain\_loss(portfolio\_value, portfolio\_std\_dev, z\_score, days):  
 return portfolio\_value \* (z\_score \* portfolio\_std\_dev \* np.sqrt(days / 252))  
day = 5  
# Loop through days 5 to 30 to calculate VaR for each method  
while day <= 30:  
 # Historical method VaR  
 range\_returns = historical\_returns.rolling(window=day).sum().dropna()  
 hist\_VaR = -np.percentile(range\_returns, (1 - confidence\_interval) \* 100) \* portfolio\_value  
 hist\_VaRs.append(hist\_VaR)  
  
 # Monte Carlo Simulation method VaR  
 scenario\_return = []  
 simulation = 0  
 while simulation <= simulations:  
  
 z\_score = random\_z\_score()  
 simulated\_return = scenario\_gain\_loss(portfolio\_value, portfolio\_std\_dev, z\_score, day)  
 scenario\_return.append(simulated\_return)  
 simulation += 1  
 mc\_VaR = -np.percentile(scenario\_return, (1 - confidence\_interval) \* 100)  
 mc\_VaRs.append(mc\_VaR)  
  
 # Variance-Covariance method VaR  
 va\_cov\_VaR = portfolio\_value \* portfolio\_std\_dev \* norm.ppf(confidence\_interval) \* np.sqrt(day / 252)  
 va\_cov\_VaRs.append(va\_cov\_VaR)  
  
 # Real-life dataset VaR (similar to historical)  
 test\_range\_returns = test\_historical\_returns.rolling(window=day).sum().dropna()  
 test\_VaR = -np.percentile(test\_range\_returns, (1 - confidence\_interval) \* 100) \* portfolio\_value  
 test\_VaRs.append(test\_VaR)  
  
 day += 1 # Increment day by 5 for the next iteration  
  
# Convert results to numpy arrays for better handling  
hist\_VaRs = np.array(hist\_VaRs)  
mc\_VaRs = np.array(mc\_VaRs)  
va\_cov\_VaRs = np.array(va\_cov\_VaRs)  
test\_VaRs = np.array(test\_VaRs)  
  
# Print the VaR results  
print("Historical VaRs:", hist\_VaRs)  
print("Monte Carlo VaRs:", mc\_VaRs)  
print("Variance-Covariance VaRs:", va\_cov\_VaRs)  
print("Real-life Test VaRs:", test\_VaRs)

Historical VaRs: [ 52371.13530594 58857.2284848 65336.45335401 70797.60537443  
 75028.25390394 83447.06776617 85703.60532253 92233.40869084  
 98736.19075496 103148.08598768 108626.78750545 114383.43749485  
 115842.25985615 121863.57487566 127939.43320893 129499.07360309  
 136835.17815053 145264.10161635 147656.87265936 148593.10036051  
 154547.08517087 156884.8971446 158182.41531855 158383.37557353  
 160721.9673095 160983.99696193]  
Monte Carlo VaRs: [ 42459.93083174 46732.43005312 50449.68211602 53373.57374499  
 57625.35795589 60363.14323859 63673.73526675 65976.52186284  
 68438.6107395 71007.50189745 73876.53767994 76063.11920198  
 78004.99964458 80996.51725384 83296.64093901 85318.22897913  
 88038.57743634 89629.53045501 91244.15988761 93169.89342727  
 95296.06265987 96942.7062322 99285.24158118 100659.865582  
 102705.60078948 104261.01505809]  
Variance-Covariance VaRs: [ 42591.16221997 46656.28059648 50394.54274966 53874.03232353  
 57142.04037875 60232.99924872 63172.90256388 65981.94478943  
 68676.1855234 71268.64582616 73770.0569184 76189.38717166  
 78534.22271496 80811.0484853 83025.45981289 85182.32443994  
 87285.90846661 89339.97558031 91347.86618493 93312.56119296  
 95236.73396458 97122.79297925 98972.91718414 100789.08549932  
 102573.1016185 104326.61499104]  
Real-life Test VaRs: [ 63310.13595813 69448.43193824 78662.64491847 75664.36875177  
 74839.13161177 78831.67394592 75303.3958381 67225.78159912  
 66525.13512969 62565.38277278 67911.62351394 74561.69235285  
 74748.05706864 78552.08093836 86097.49373484 93515.81220078  
 99789.53581918 105501.77404251 100283.23847998 96580.34615069  
 91356.57321114 89081.29075277 87759.84358624 81534.25082731  
 60956.80937951 45283.57660339]

# Convert results to pandas Series with days as index  
days\_range = [5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30]  
hist\_VaRs\_series = pd.Series(hist\_VaRs, index=days\_range, name="Historical VaR")  
mc\_VaRs\_series = pd.Series(mc\_VaRs, index=days\_range, name="Monte Carlo VaR")  
va\_cov\_VaRs\_series = pd.Series(va\_cov\_VaRs, index=days\_range, name="Variance-Covariance VaR")  
test\_VaRs\_series = pd.Series(test\_VaRs, index=days\_range, name="Real-life Test VaR")  
  
# Combine all Series into a DataFrame for easy plotting  
var\_df = pd.DataFrame({  
 "Historical VaR": hist\_VaRs\_series,  
 "Monte Carlo VaR": mc\_VaRs\_series,  
 "Variance-Covariance VaR": va\_cov\_VaRs\_series,  
 "Real-life Test VaR": test\_VaRs\_series  
})  
  
# Plot the VaR results  
plt.figure(figsize=(14, 8))  
plt.plot(var\_df)  
plt.title("Value at Risk (VaR) Comparison Over Different Time Periods")  
plt.xlabel("Days")  
plt.ylabel("VaR (in Naira)")  
plt.legend(var\_df.columns)  
plt.grid(True)  
plt.show()  
var\_df.head()



# Perform t-tests comparing each predicted VaR to the real VaR  
t\_test\_results = {}  
for column in ['Historical VaR', 'Monte Carlo VaR', 'Variance-Covariance VaR']:  
 t\_stat, p\_value = stats.ttest\_rel(var\_df['Real-life Test VaR'], var\_df[column])  
 t\_test\_results[column] = (t\_stat, p\_value)  
  
# Print the t-test results  
print(f'{"Method":<20} {"T-Statistic":<20} {"P-Value":<20}')  
print("-" \* 60)  
for method, (t\_stat, p\_value) in t\_test\_results.items():  
 print(f'{method:<20} {t\_stat:<20.4f} {p\_value:<20.4f}')  
# var\_df.to\_csv('var\_df.csv')

Method T-Statistic P-Value   
------------------------------------------------------------  
Historical VaR -5.8802 0.0000   
Monte Carlo VaR 0.2720 0.7879   
Variance-Covariance VaR 0.2822 0.7801